

www.vuCTn.com

CS506 MERGED

PPT Slides 1 TO 45

Regards : Zarva Chaudhary

Merged File date 22/12/2022

Admin:

* Zarva Chaudhary *

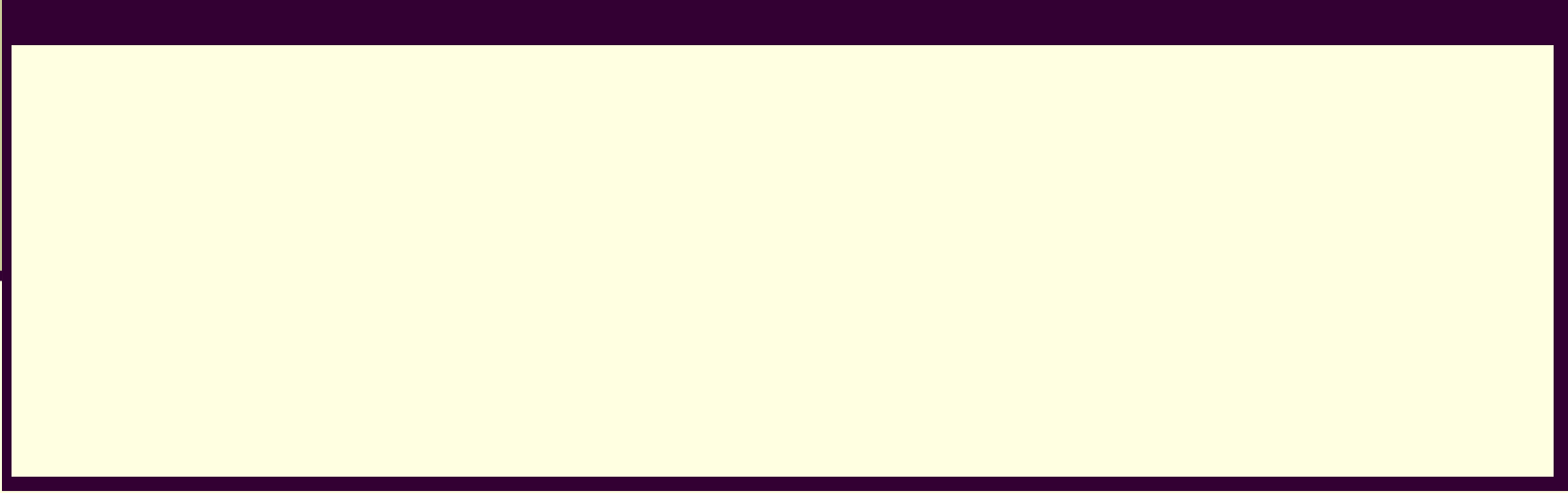
* Chandhary Moazzam *

* Laiba Maki *



Web Design & Development

Lecture 1



Course Introduction

- Modules

- Introduction & Basic Programming in Java

- Basic Syntax
- OOP
- Exception Handling
- IO Streams
- GUIs & Event Handling
- Graphics (time dependent)
- Data Base Connectivity

- Network programming

- Socket Based Client Server Programming
- Remote Method Invocation

Course Introduction

- Modules

- Web Application Development

- Introduction to basic web technologies
 - JSPs
 - Servlet
 - JSF
 - Java Beans
 - Other J2EE technologies

- Web / Enterprise Application Design

- Application Architecture Considerations
 - Design Strategies
 - Programming in Layers

Course Introduction

- Modules

- Others (Time dependent)

- Web Services
- XML and related stuff
- Design Patterns
- Comparison with .NET Architecture

Objectives & Outcome of the course

- Students should be able to design and develop a Web based Enterprise Application
- Understand and appreciate various Web Architectures, technologies and solutions that can be used
- Learn Java and other Allied Technologies as a by product of the course

Course Material

- Reference Books
 - Core Web Programming by Marty Hall
 - Java2 Complete Reference by Helbert
 - Advanced Java How To Program by Deitel
 - (Any good book on Java & J2EE will do)
- A lot of material is available online. There are useful sites such as
 - www.w3schools.com
 - <http://java.sun.com>
- Handouts
- Lecture CDs and possible demo AVIs

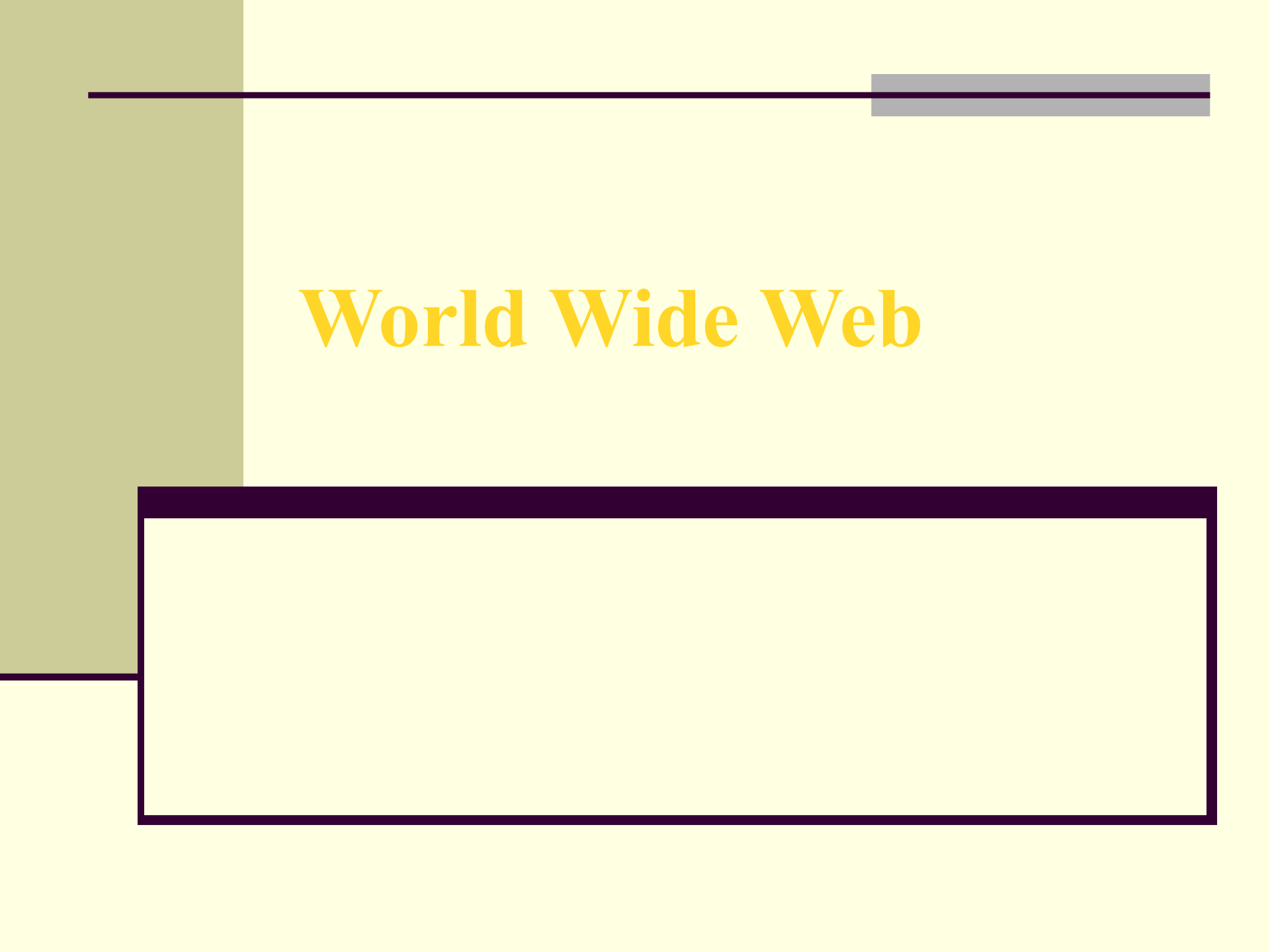
Learning Methodology

**I hear and I forget,
I see and I remember,
I do and I understand**

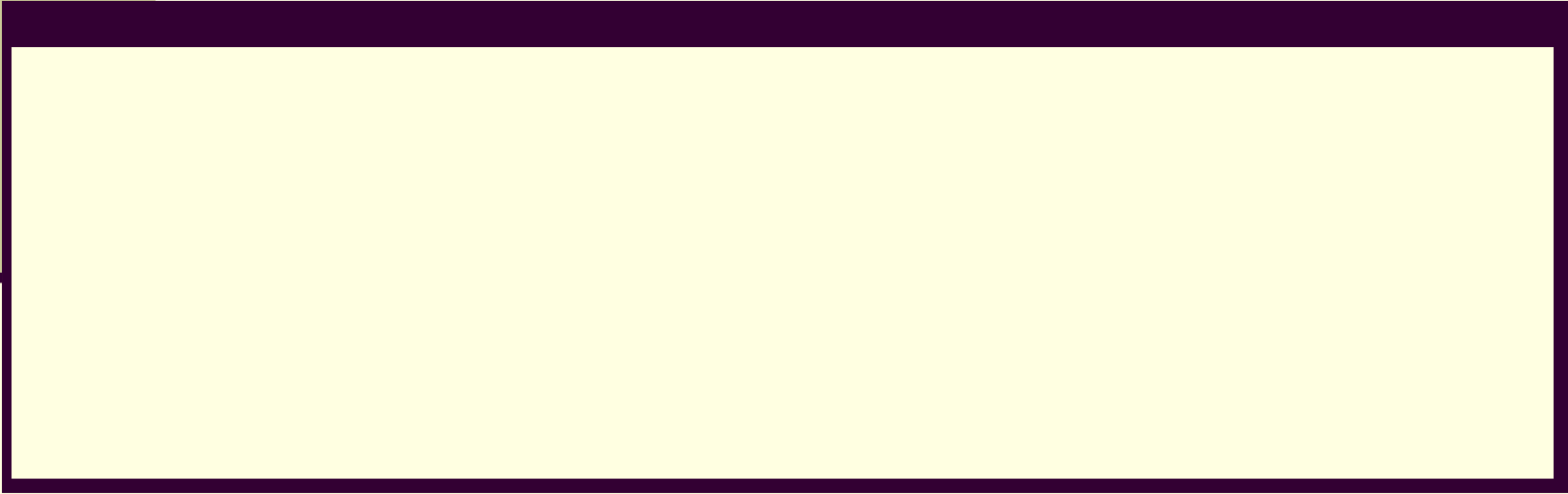
Chinese Proverb

Assignments

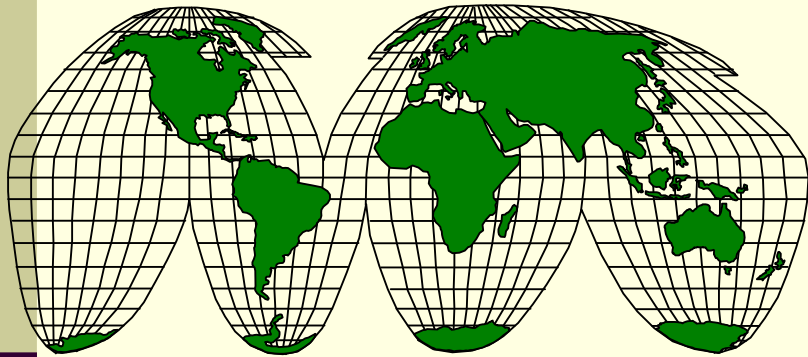
- Assignments On
 - General Programming
 - GUI based
 - Graphics (small game)
 - Network Programming (chat / file sharing server / multi user network game)
 - 2-3 Assignments on simple WEB Applications to make you comfortable with different technologies
 - One relatively large Project (Web based Application)
- Talk about Plagiarism



World Wide Web



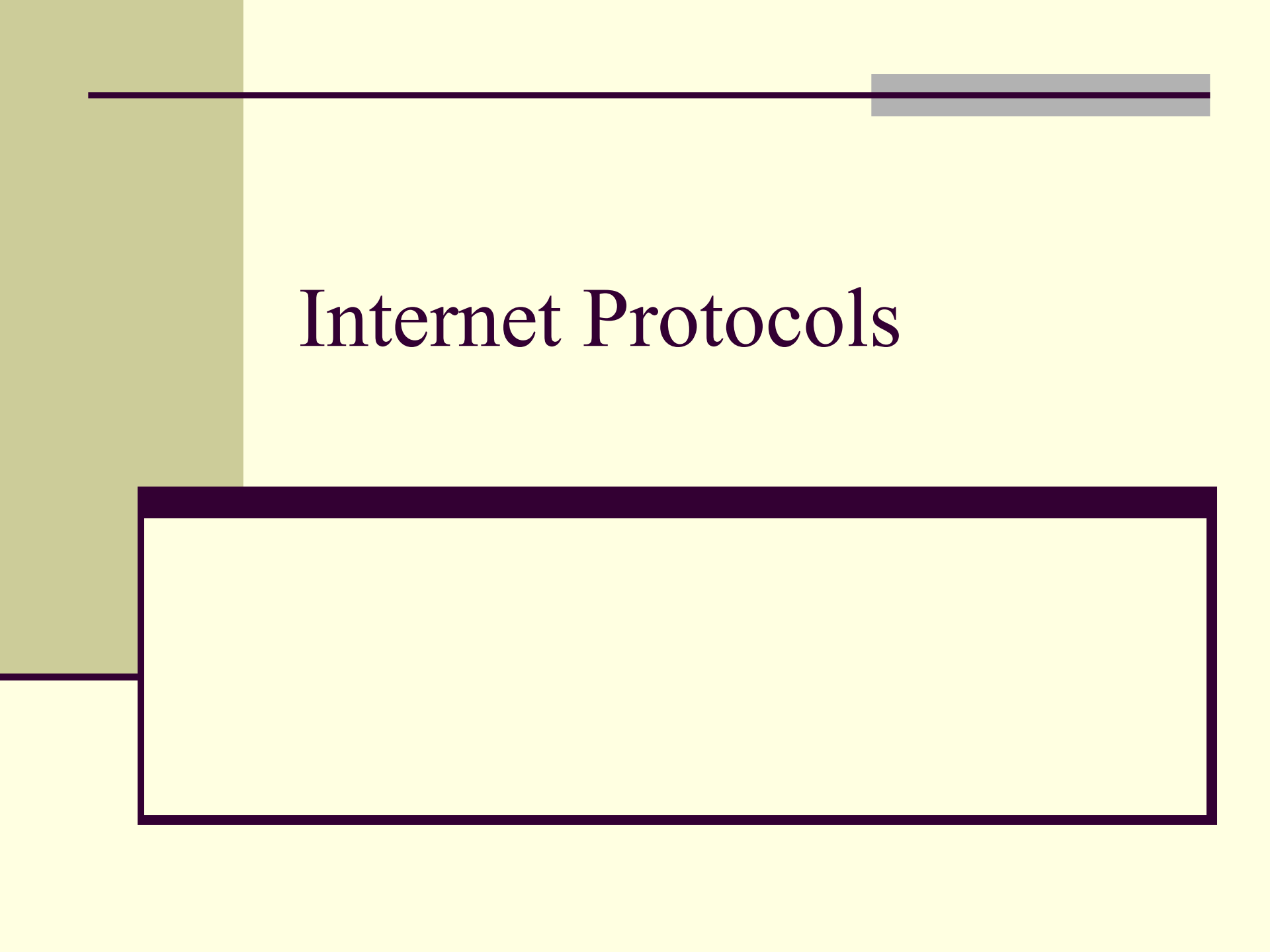
What Is the World Wide Web?



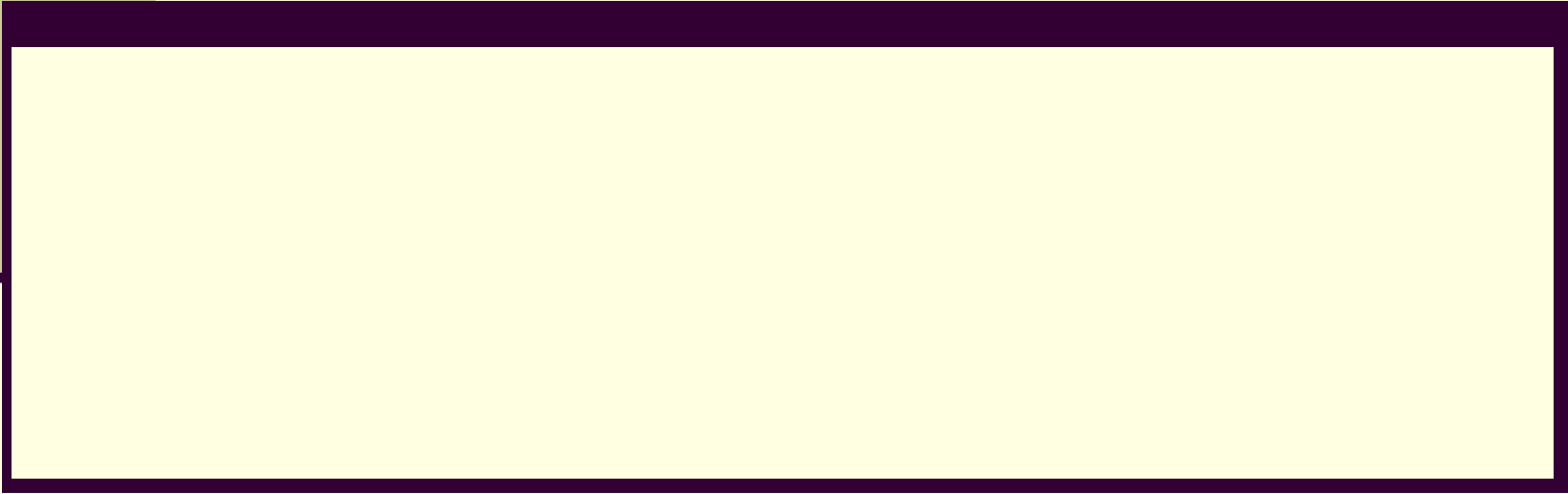
- The internet is a network of computer networks worldwide
- The **web** is a tool used to retrieve information published on the internet
- To navigate the web we use a **browser** i.E. Netscape or internet explorer

Brief History of the www

- 1968 - DARPA (Defense Advanced Research Projects Agency) contracts with BBN (Bolt, Beranek & Newman) to create ARPAnet
- 1970 - First five nodes:
 - UCLA
 - Stanford
 - UC Santa Barbara
 - U of Utah, and
 - BBN
- 1974 - TCP specification by Vint Cerf
- 1984 – On January 1, the Internet with its 1000 hosts converts en masse to using TCP/IP for its messaging



Internet Protocols



HTTP

- Original web communication protocol
- Request-Response type
 - Client (browser) will open a connection to a server and then send a request using a very specific format
 - Server will respond and close the connection
- Stateless
 - Does not maintain any connection information between transaction information
- Feature
 - Negotiation of data representation, allowing systems to build **independently** of the data being transferred

TCP/IP

- Transmission Control Protocol over Internet Protocol
- IP ensures that data packets reach the destination from the source of communication
- Not necessarily data packets follow the same path
- TCP keeps track of these packets, and manages the assembling of these packets to form the original message
- Thus, both TCP and IP work together to ensure that effective transmission of data over the internet



WebSite (Web Brouchers)

VS

Web Application



Web Brochures vs. Web Applications

	WebSite (Web Brochures)	Web Applications
Goal	Information: Text, graphics, tables	Web interface to any application Interaction: Data entry, data selection, checking, ordering
Emphasis	Professional look, graphic design	Optimal support for the work flow
Implementation	Few interaction steps, mostly static pages (e.g. data entries) Navigation (browsing) - many integrated hyperlinks with jumps to other pages	Usually interacts with data source Incorporates business logic May contain simple or complex navigation Dynamic behaviour

Web Brochures vs. Web Applications

	Web Brochures	Web Applications
Deployment	WWW	May present on companies LAN or WWW
Examples Apps	University Web Sites, Personal Home Page etc.	Amazon.com Purchasing System Address Book etc



QUICK SEARCH

PREHISTORIC ART +
 ANTIQUITY +
 WESTERN EUROPEAN ART +
 THE ARSENAL +
 ORIENTAL ART +
 RUSSIAN CULTURE +
 NUMISMATICS +
 THE GOLDEN ROOMS +
 THE PALACE OF PETER I +
 THE MENSHIKOV PALACE +

ORIENTAL ART

+ THE MIDDLE EAST
 + ANCIENT EGYPT
 + BYZANTIUM
 + SASSANIAN IRAN
 + ISLAMIC ART OF THE COUNTRIES OF THE NEAR EAST
 + INDIA
 + MONGOLIA AND TIBET
 + CHINA
 + JAPAN
 + THAILAND
 + CENTRAL ASIA
 + THE CAUCASUS
 + THE GOLDEN HORDE
 + ORIENTAL JEWELLERY



Over 180,000 items including paintings, sculptures and examples of applied art ([jewellery](#), domestic and cult objects), give an idea of a remarkable cultural heritage of the Orient from the time of the ancient civilizations emergence to the present. The exhibitions occupying 50 museum rooms contain the collections of items from Ancient Egypt, Mesopotamia, Central Asia, Byzantium, countries of the Near and Far East.

Shortcuts

- Create Contact
- Enter Business Card
- Create Account
- Create Lead
- Create Opportunity
- Create Quote
- Create Case
- Report Bug
- Schedule Meeting
- Schedule Call
- Create Task

New Contact

First Name:

Last Name: *

Phone:

Email:

My Upcoming Appointments

through today (2005-07-28)

Close	Subject	Date	Accept?
☒	Review needs	2005-06-16 10:47	☒
☒	Initial discussion	2005-06-18 10:46	☒
☒	Introduce all players	2005-06-20 10:47	☒
☒	Introduce all players	2005-07-03 10:46	☒
☒	Introduce all players	2005-07-28 10:47	☒

My Top Open Opportunities

Opportunity	Account Name	Amount	Close
WEST ARKANSAS 111580 - 1000 units	WEST ARKANSAS 111580	\$ 75000	2005-10-17
Rhyme & Reason Inc 195479 - 1000 units	Rhyme & Reason Inc 195479	\$ 75000	2005-12-14
Chandler Logistics Inc 302035 - 1000 units	Chandler Logistics Inc 302035	\$ 75000	2005-07-18
Trait Institute Inc 511227 - 1000 units	Trait Institute Inc 511227	\$ 75000	2006-01-01
A B Drivers Limited 347200 - 1000 units	A B Drivers Limited 347200	\$ 50000	2005-12-22

My Open Cases

Start Previous (0 - 0 of 0) Next End				
Close	Num.	Subject	Account Name	Priority

My Leads

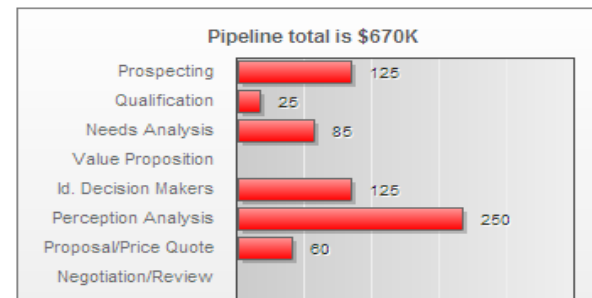
Start Previous (1 - 20 of 103) Next End				

July 2005

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						
Previous Month				Next Month		

Team Notices

My Pipeline



Advantages of Using Web Applications



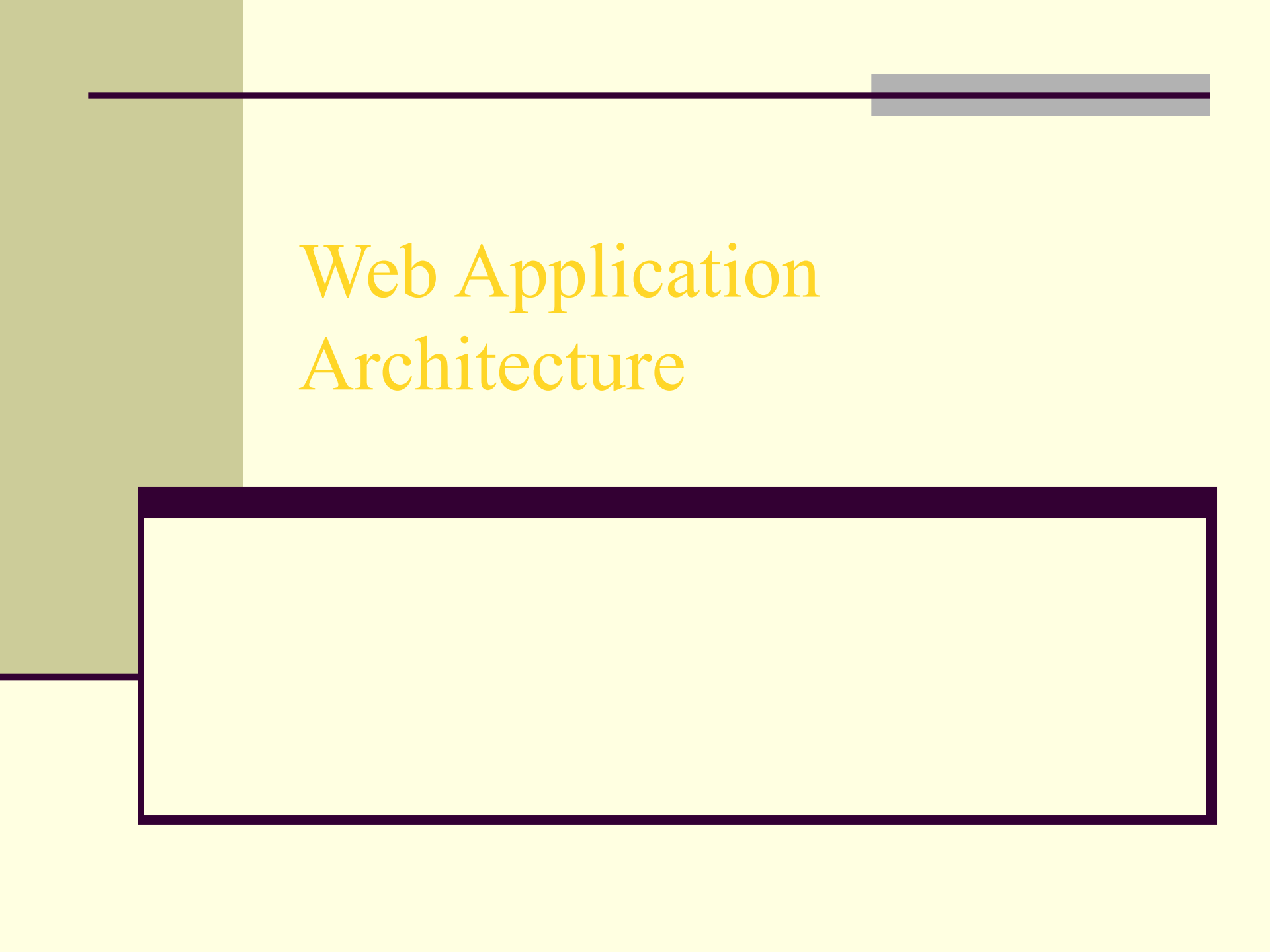
- Allows you to browse a wide variety of internet sources

- Instantaneous connections to internet sites world wide

Disadvantages of Using Web Applications

- Connections can be slow or busy
- No standard methods of organization
- Out-of-date materials may not be removed
- Contents can be (maliciously) altered
- Sites can simply be moved/removed





Web Application Architecture



Layers

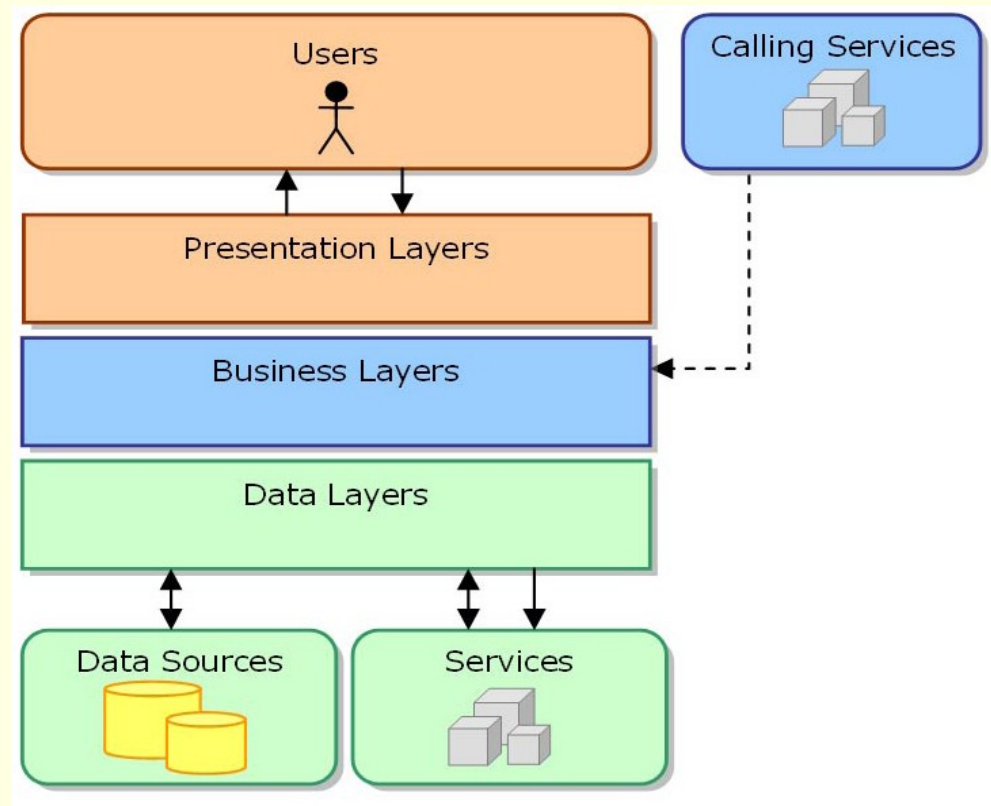


Figure shows a simplified view of one application and its layers.

Layers

- It is important to note that the layers are merely logical groupings of the software components that make up the application or service.
- They help to differentiate between the different kinds of tasks performed by the components.
- They make it easier to design reusability into the solution.
- Each logical layer contains a number of discrete component types grouped into sublayers, with each sublayer performing a specific kind of task.
- By identifying the generic *kinds* of components that exist in most solutions, you can construct a meaningful map of an application or service, and then use this map as a blueprint for your design.

Evolution of Enterprise Application Framework for C/S Arc

- Single tier
- Two tier
- Three tier
 - RPC based
 - Remote object based
 - Three tier (HTML browser and Web server)
 - Proprietary application server
 - Standard application server
- N-Tier

Client-server architectures

1-tier Architecture

- Entire application exists on single node
 - Installed on individual machines

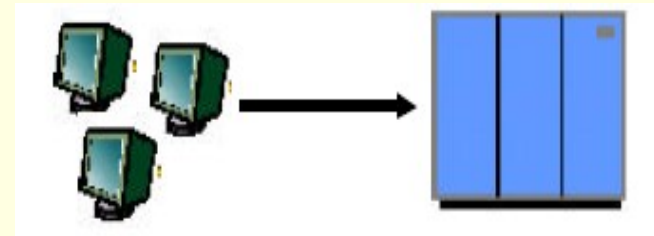
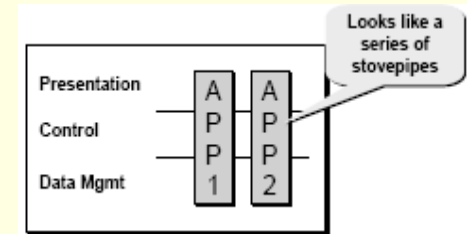
Dumb terminals are directly connected to mainframe

Centralized model (as opposed distributed model)

- Presentation, business logic, and data access are intertwined in one monolithic mainframe application

Types of systems

- Standalone executable
- Mainframe applications



Client-server architectures

1-tier Evaluation

- Advantages

- Simple to build
- Natural
- No client side management is required
- Data consistency is easy to achieve

- Disadvantages

- Facilitates very little reuse
- Maintenance can be expensive
- Single point of failure
- Scaling systems requires buying bigger, costly hardware

Client-server architectures

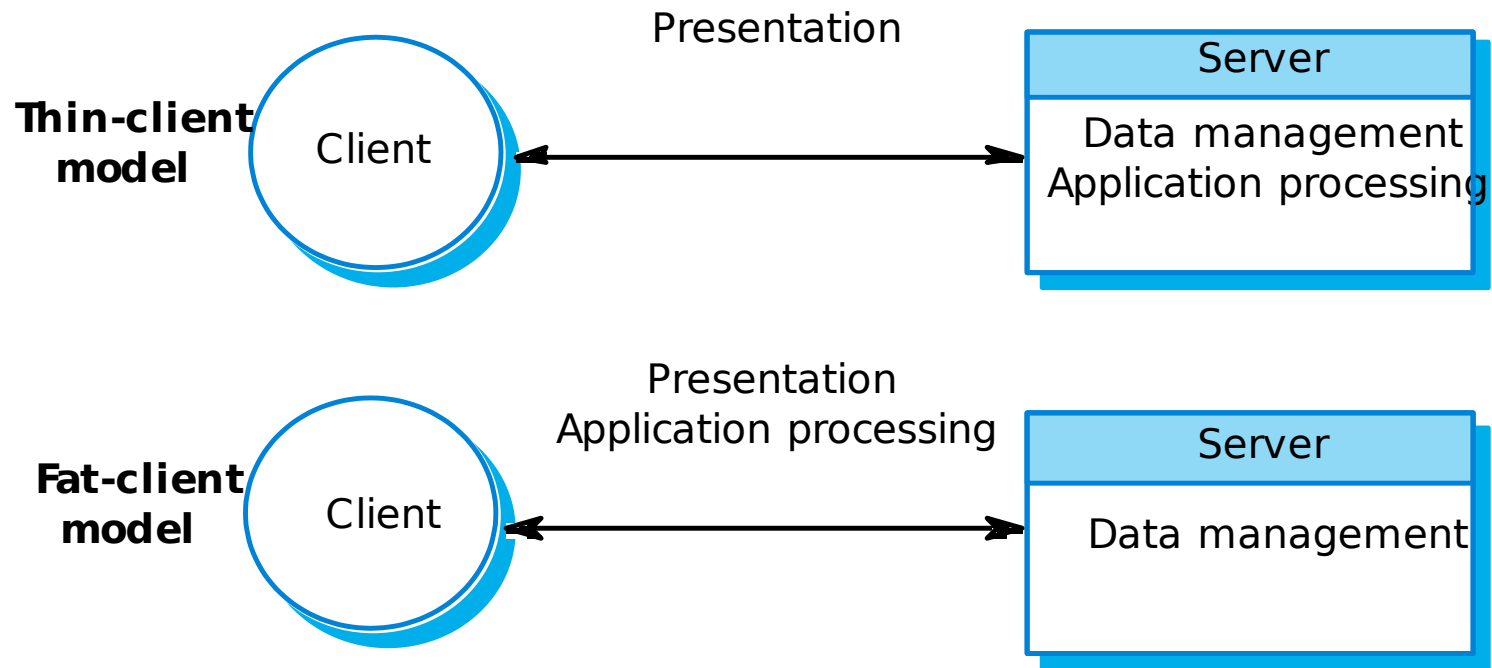
2-tier Architecture

■ **Client Server Applications**

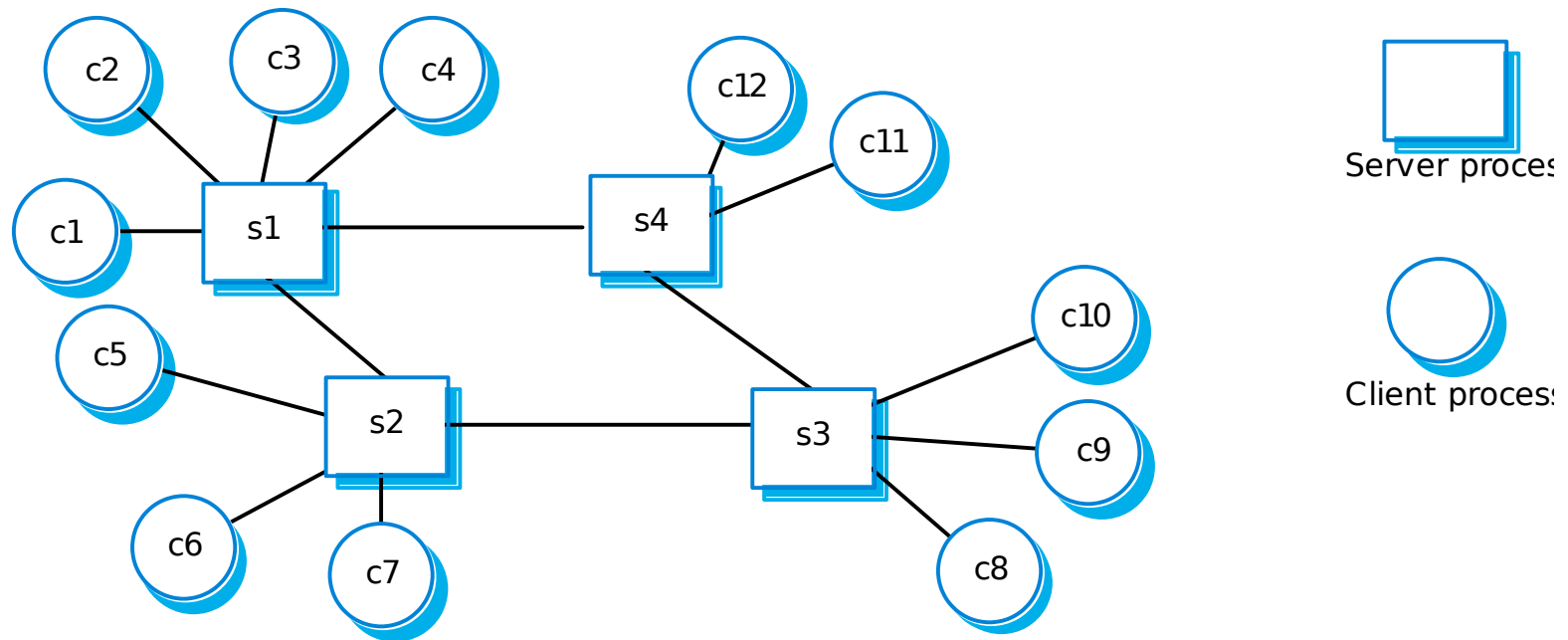
- The client and server can be heterogeneous
 - Different implementation languages
 - Different operating systems
- The roles can be transient
- Fat client – server only manages data
 - talk to back end database
 - SQL queries sent, raw data returned
 - Some Windows GUI based application
- Thin Client – server manages data and business logic
 - Browser – server manages presentation too

Client-server architectures

2-Tier Thin and fat clients



A client-server system



Client-server architectures

2-tier Evaluation

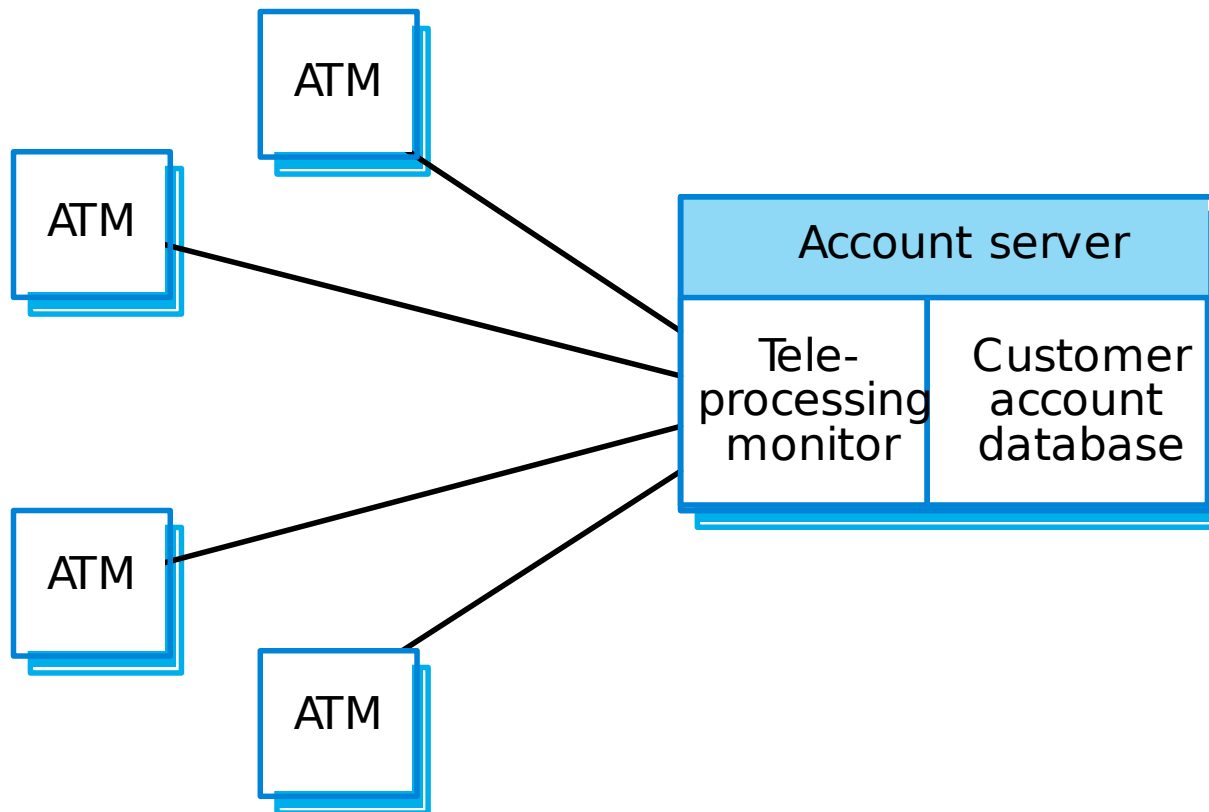
Advantages

- Modifications on server propagated to clients
- Can distribute processing load
- Better scalability by adding server nodes and clients
- Database type independence

Disadvantages

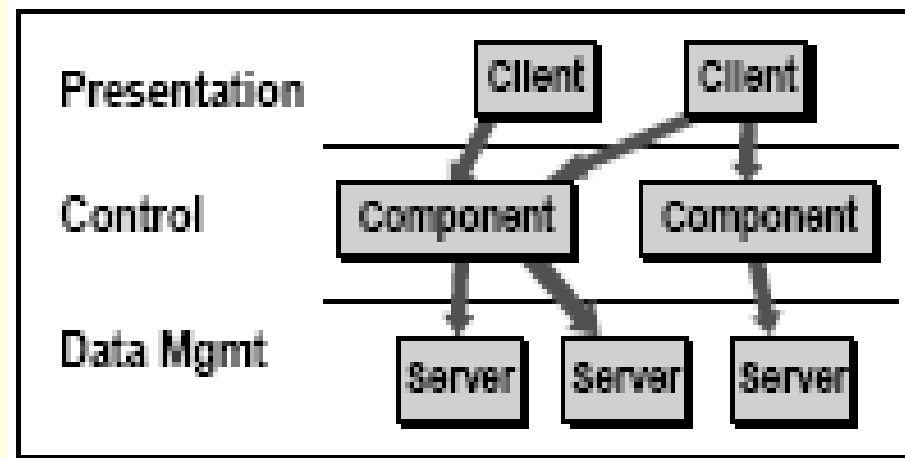
- Client nodes require more computing power
- Development and maintenance more complex
- Presentation, data model, business logic are intertwined (at client side), difficult for updates and maintenance
- Data Model is “tightly coupled” to every client: If DB Schema changes, all clients break
- Updates have to be deployed to all clients making System maintenance nightmare
- DB connection for every client, thus difficult to scale
- Raw data transferred to client for processing causes high network traffic

A client-server ATM system



3-tier Architecture (General)

- Applications are generally partitioned as Client, Control (business) and Data Components
- In 3-Tier
 - Each logical partition maps to a layer in the system
 - Modeling layers and software layers match 1-to-1
 - Each layer implemented with appropriate technologies
 - Layers have their own internal architectures



3-tier Evaluation

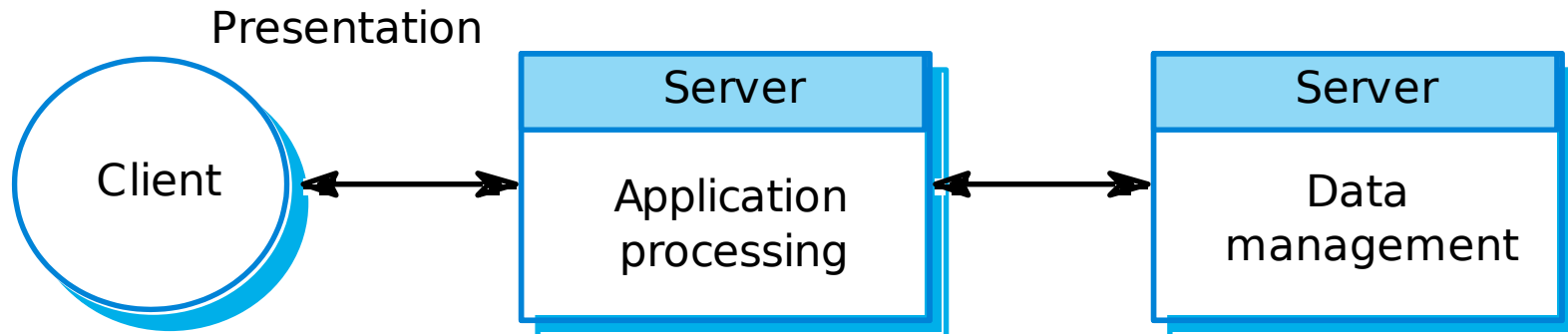
■ Advantages

- Complete separation of concerns
- Control logic can be reused by client applications
- Caches results in the controller layer
- Maximum flexibility for enterprise-wide applications

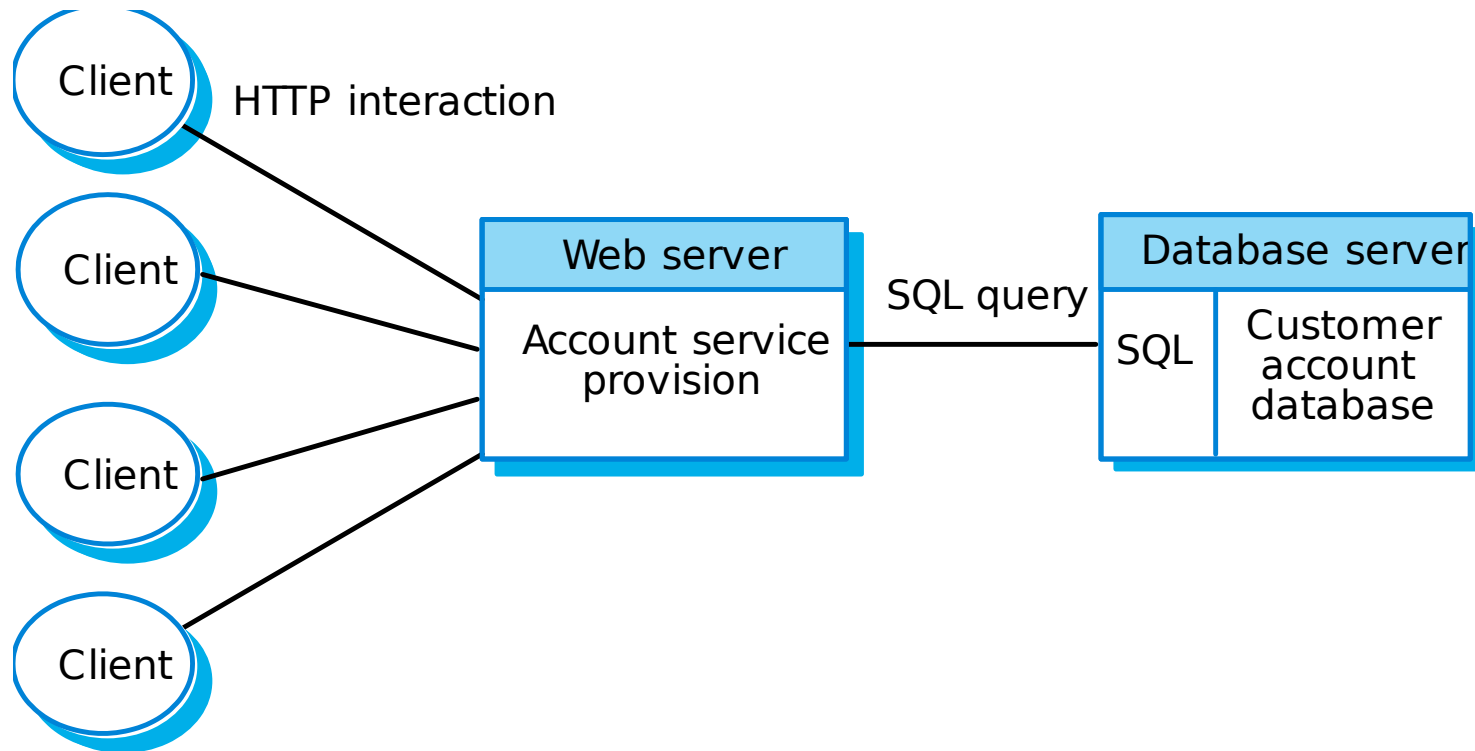
■ Disadvantages

- Difficult to integrate legacy stovepipes which represent large investment
- Complexity
- Speed decreases with levels of indirection and latency
- Increases cost and development time
- Lack of knowledgeable developers and managers

A 3-tier C/S architecture



An internet banking system



N-tier (multi-tier & multi-layered) Architecture

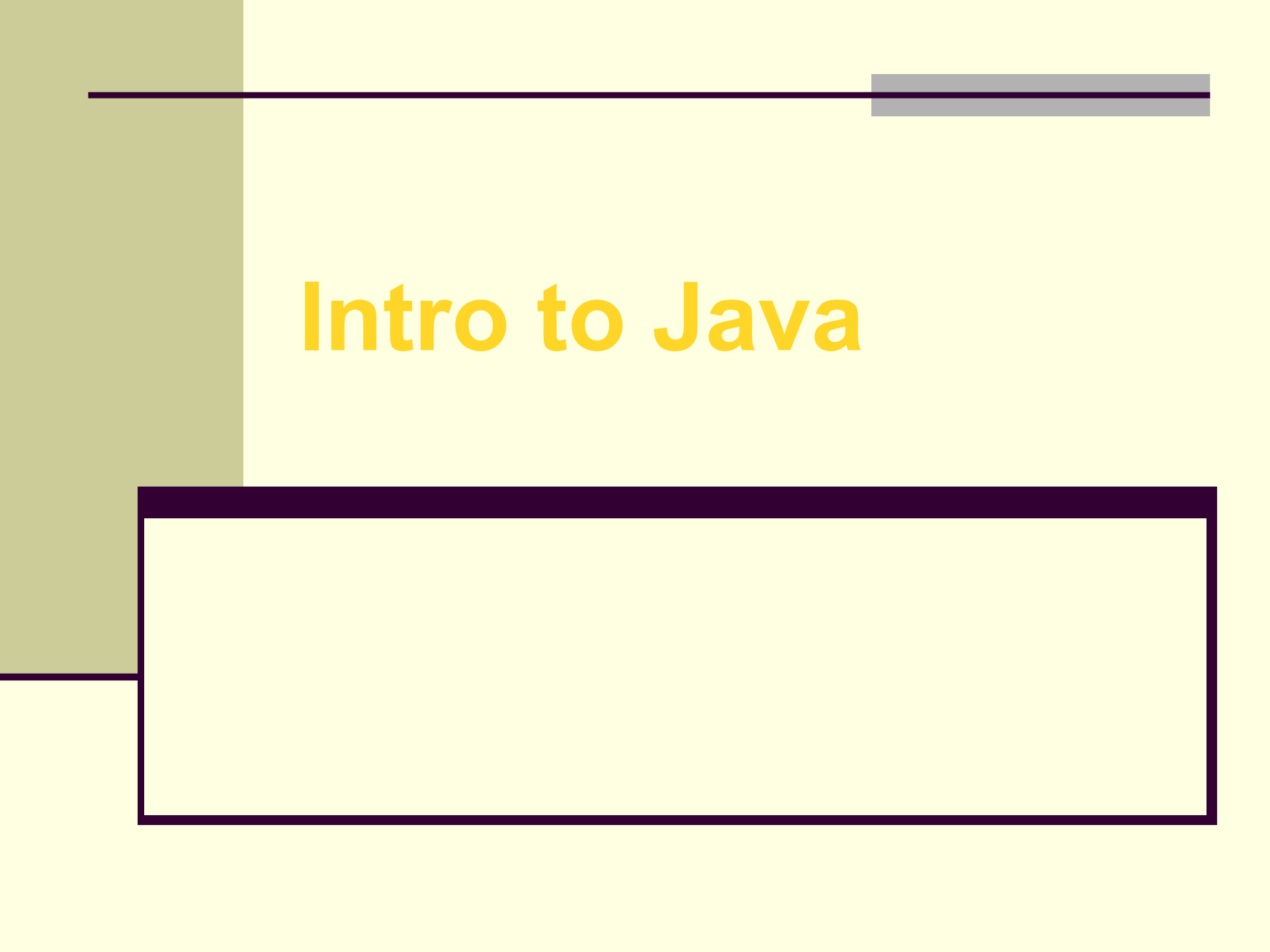
- Layers added for better separation of concerns
 - Every layer does a specified task, which improves cohesion and lowers coupling
 - Application can be divided among developers with well defined roles.
 - Layers and Tiers need not have 1-1 mapping. 3-tier is popular as its maps to typical IT problems
 - For example a fortune five company uses
 - JSP-Servlet-Handler-Translator-EJB-DAO layers
 - These layers are deployed on three physical tiers which are
 - Thin client → Web + Application server → Database Tiers



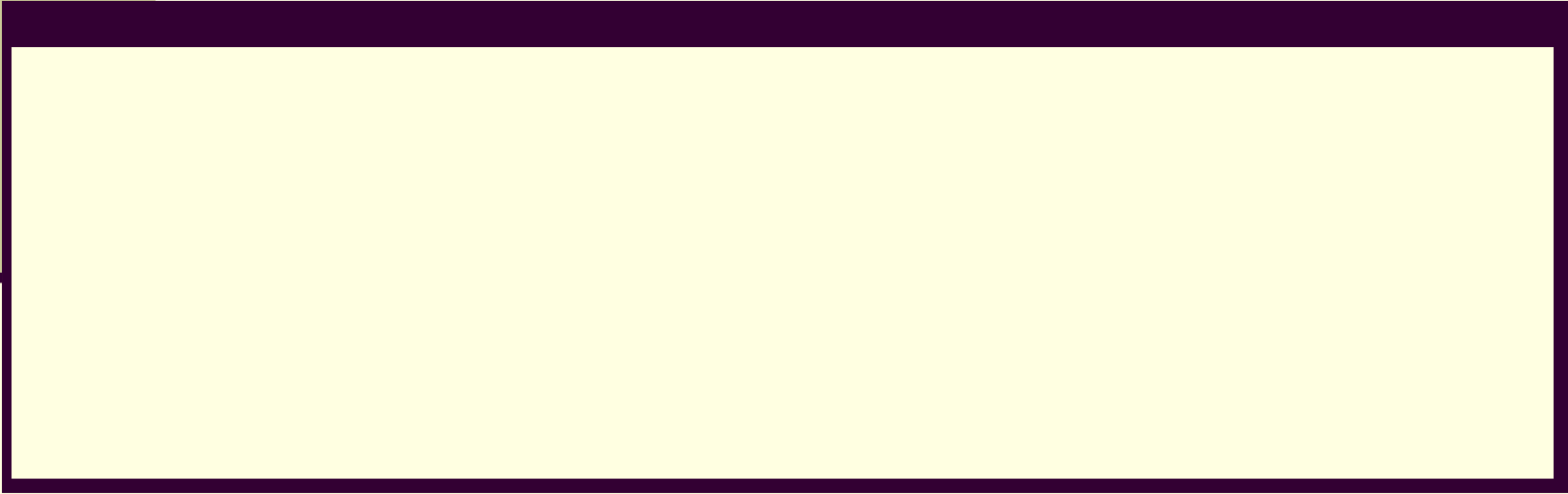
Web Design & Development

Lecture 2





Intro to Java



Topics We Will Cover Today

- History of Java
- Why Java ?
- Some Sample Java Based Applications
- Writing a Basic Java Program
- Java Program Development and Execution Steps

History

■ Java

- Based on C and C++
- Developed in 1991 for intelligent consumer electronic devices
- Green Project (According to Gosling, "the goal was ... to build a system that would let us do a large, distributed, heterogeneous network of consumer electronic devices all talking to each other.")
- James Gosling Modified C++
- Named Oak then Java

History....

- Platform independent
- Interpreted Language
- Intermediate Code (Byte Code)
- Reliable
- Multiple inheritance and Operator overloading removed
- No Pointers because of security reasons
- Built in network support

History...

- Internet exploded in 1993, saved project
 - Reliability
 - Security
 - Platform Independence
- Java formally announced in 1995
- Now used to create interactive web applications, distributed enterprise application for consumer devices (pagers, cell phones) and much more..



Why Java???

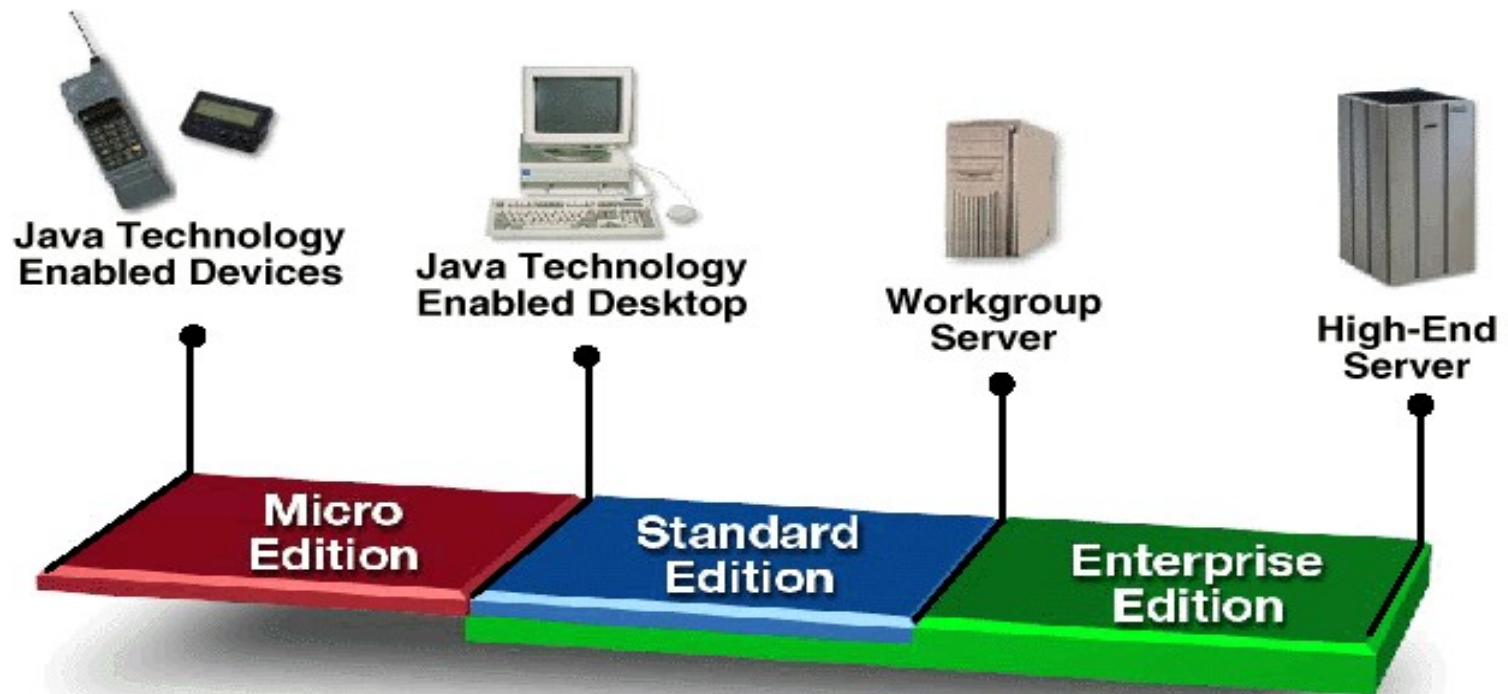
Motivation ☺

- Portable WORA!!!!!!
- Simple
- “Pure” Object Oriented Language
- Support for Distributed and Web Applications
- Rich Libraries
 - Multithreading , Swing , RMI , NET , SQL
- Automatic Garbage Collection
- More Robust

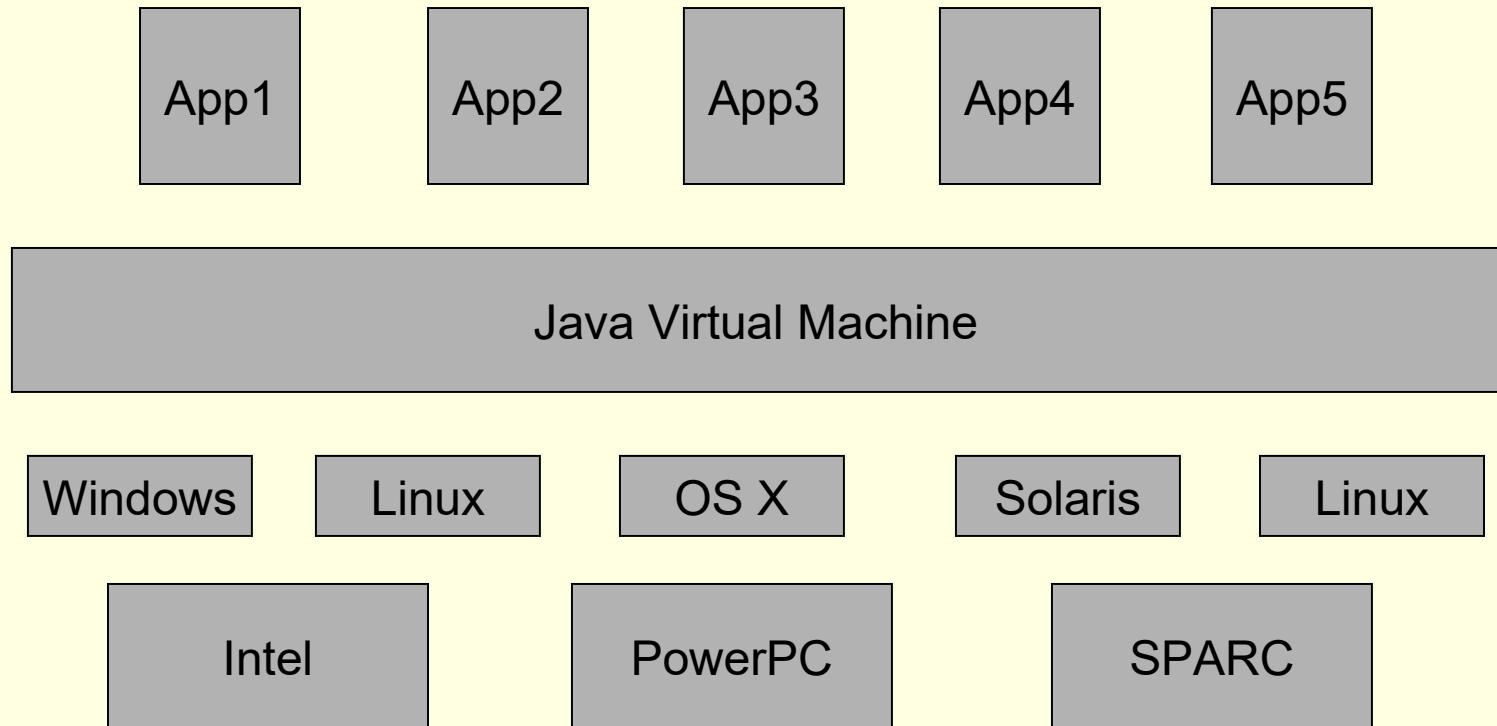
Portable

- “Write-Once Run-Anywhere”
- The Java Virtual Machine becomes the common denominator
 - Bytecodes are common across all platforms
 - JVM hides the complexity of working on a particular platform
 - Difficult to implement a JVM
 - But simpler for the application developer
- Java does this well

Java Platform



The Java Platform



Simple

- Similar to C/C++ in syntax
- In-fact Java is C++ minus
 - operator overloading
 - direct pointer manipulation or pointer arithmetic
 - multiple inheritance
 - Destructors (Garbage Collector– handles memory automatically)
 - No Templates
 - Header/make files
- Lots of other things which make Java more attractive.

Object-Oriented

- Fundamentally based on OOP
- All functions belong to classes or objects. No global variables or functions exist
- All classes by default inherit from a common ancestor known as “Object”
- “Almost” all data types are objects
- OOP will be covered in a little more detail later.

Distributed / Network Oriented

- Java grew up in the days of the Internet
 - Inherently network friendly
 - Original release of Java came with Networking libraries
 - Newer releases contain even more for handling distributed applications
 - RMI, Transactions

Support for Web and Enterprise Applications

- Given below are some of the Java technologies that can be used for web and enterprise application development
 - Servlets
 - JSP
 - Applets
 - JDBC
 - RMI
 - EJBs
 - JSF
- And many more...

Robust / Secure / Safe

- Designed with the intention of being secure
 - No pointer arithmetic or memory management!
 - The JVM “verifier”
 - Checks integrity of byte-codes
 - Dynamic runtime checking for pointer and array access
 - No buffer overflow bugs!
 - SecurityManager to check which operations a piece of code is allowed to do
 - “Sandbox” operation for applets and other untrusted code
 - Limited set of operations or resources made available
 - Contrast to ActiveX

Rich Set of Libraries

- Multithreading
- Swing
- Regular Expression
- NET
- SQL
- Util
- Serialization

Java Programmer Efficiency

- Faster Development
 - More programmer friendly
 - Less error prone
- OOP
 - Easier to manage large development projects
- Robust memory system
 - No pointer arithmetic and manual memory management. Garbage collector!
- Libraries
 - Re-use of code

Disadvantages ☹️

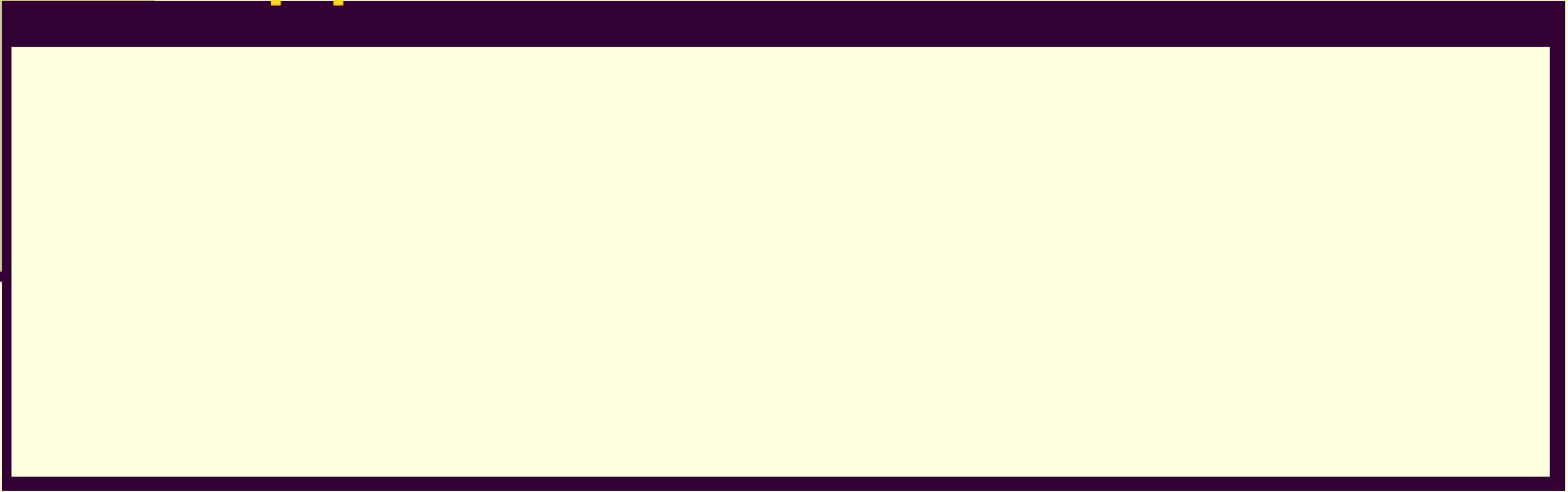
- Java performance IS slower than C/C++
 - Tradeoff between development time vs. run time
 - Additional checks in Java which make it secure and robust and network aware etc, all have a small cost.
- BUT
 - JIT compilation and HotSpot
 - Dynamic compilation of bytecode to native code at runtime to improve performance
 - HotSpot optimizes code on the fly based on dynamic execution patterns
 - Can sometimes be even faster than compiled C code!
- Increasing processing speeds helps in overcoming this short fall

Microsoft vs. Java

- Java is platform independent
 - Was considered a threat to Microsoft's dominance
 - Sun vs. Microsoft Law Suit
- Microsoft's latest response to Java
 - C#
 - Very similar to Java in structure and style
 - Some improvements over past releases of Java (which have now emerged in Java 1.5)



Some Sample Java Based Applications



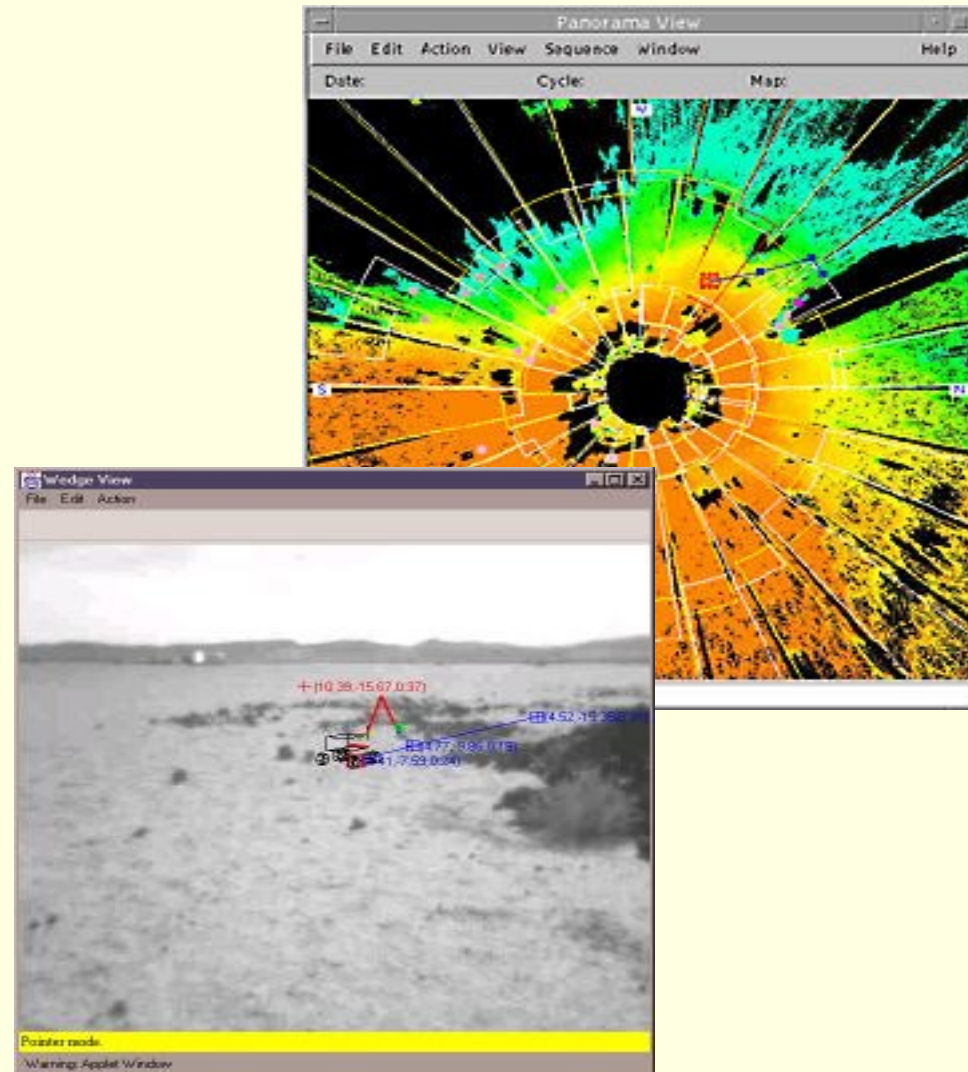
Hubble Space Telescope Monitoring

- NASA Goddard's most successful project ever
- Launched in 1990.
- Has sensitive light detectors and cameras
- Provided view of galaxies up to 10 billion light years away



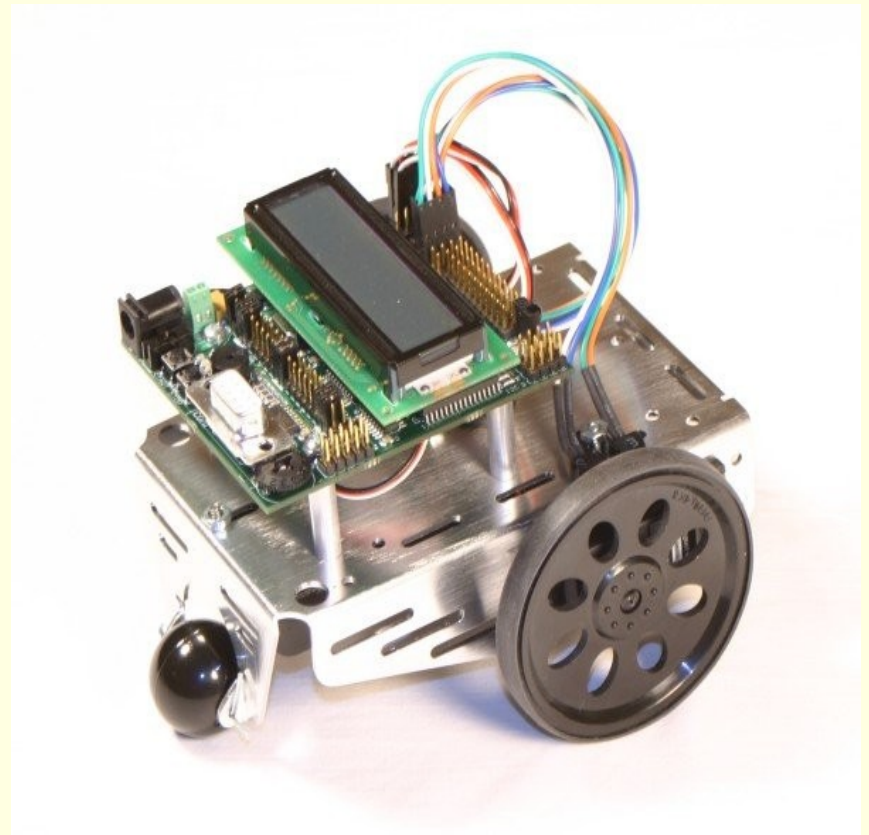
Mars Pathfinder Mission Simulator

- Used for world-wide data viewing
- Winner of the 1997 NASA software of the year
- The current rover location is displayed, along with visual indications of “targets”
- Provides close-ups of the wedge photograph



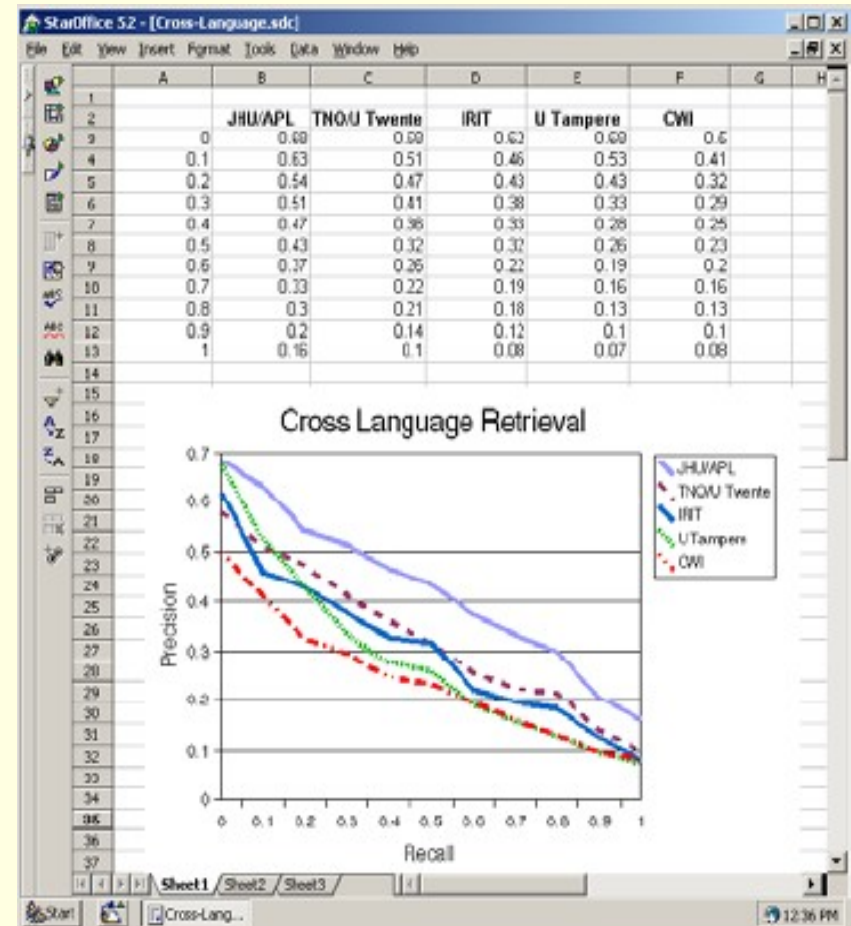
IntelliBrain™ -Bot

- Java Programmable
- RoboJDE™ java enabled robotics software development environment
- Makes developing, debugging robotics program a snap



Star Office 5.2

- Cross platform Office suite completely written in java



Web Based Course Registration System


**GOVERNMENT COLLEGE
UNIVERSITY FAISALABAD**

[Search](#)
[Sitemap](#)
[Contact](#)

[GCUF](#)
[News & Events](#)
[Prospective Students](#)
[Students](#)
[Alumni](#)
[Executive Education](#)
[Faculty & Research](#)



PCO Options:

- Manage Quarter
- Manage Programs
- Manage Fields
- Customized Query
- Manage Waiver
- Manage Instructors
- Search Details
- FeedBack
- Evaluation Options
- Manage Streams
- Manage Courses
- Manage Venues

Fields	Please select a field to view its detail			
Field ID	Field Name	Field Code		
Details	Computer Science	CS	Edit	Delete
Details	Business Administration	BA	Edit	Delete
Details	Law and Politics	LLB	Edit	Delete
Details	Computer Engineering	CE	Edit	Delete

4 Record(s) exist

details	details of the currently selected field						
Program Type	Major Units	Minor Units	Min GPA	Min Units	Policy URL		
Masters	9	3	2.5	45	policy.doc	Edit	Delete
Bachelor	6	3	2	133	policyB.doc	Edit	Delete

2 Record(s) found corresponding to selected field

Web Based Performance Review Management System

PERFORMANCE MANAGEMENT SYSTEM

Manage Employees

Career
[Competency Scale](#)
[Competency Set](#)
[Reviews To Do](#)
[Initiate Review](#)

Organization
[Organization Employee](#)
[Position Job](#)

First Name

Last Name

Employee Category

Position

Gender

Year of Joining

	Emp Code	First Name	Last Name	Position Name	Category	Date Of Joining	Email
☐	LE005	Madiha	Shahid	Senior Software Engineer, VIPER	permanent	Feb 5, 2004	
☐	LE006	Qasim	Bilal	Software Engineer, VIPER	Permanent	Jun 5, 2005	qasim@lmkr.org
☐	LE007	Umer	Sohail	Software Engineer, VIPER	temporary	Mar 1, 2005	
☐	LE008	Zohaib	Hassan	Programmer, VIPER	temporary	Jun 5, 2004	

Web Based School Management System

**Educators School System**
The Best Education For You

AcademicsGradingStudentsTeachersUsersLogOff

Search Student

Back

Enter Student Search Criteria	
Roll No	
Name	
Class	Class 1
Section	Section45
Batch	
Rank	
Enrollment Year	

SearchAddCancelReset

| Search || Academics || Profile || Fee || Results || Back |

Copyright © 2005. All Rights Reserved.

Writing Basic Java Program

Syntax for C++ programmers

Software Requirements

- For the start following software will do the job
 - You need to have the latest version of JDK. (J2SE 5.0) You can download it for free from
 - <http://java.sun.com/j2se/>
 - A little older versions such JDK 1.4 or 1.3 will also work
 - Notepad
 - And you should set your path variable.

Canonical Example

HelloWorld Application in Java

```
/* The HelloWorldApp class implements an application  
   that simply displays "Hello World!" to the standard  
   output. */
```

```
public class HelloWorldApp {
```

```
    public static void main(String[] args) {
```

```
        //Display the string. No global main
```

```
        System.out.println("Hello World!");    }
```

```
    }
```

Compiling and Running the program

- Save this file in some directory and compile it using
 - `javac HelloWorldApp.java`
- Run the compiled file by using the command
 - `java HelloWorldApp`

Java Program Development and Execution Steps

■ Java Systems

- Consist of environment, language, Java Applications Programming Interface (API)
- Java programs have five phases

1. Edit

- **.java** extension

2. Compile

- **javac** command: **javac MyProgram.java**
- Creates **.class** file containing bytecodes with similar name

Java Program Development and Execution Steps...

3. Loading

- Class loader transfers `.class` file into memory
- Classes loaded and executed by interpreter with `java` command
- To load,

`appletviewer Welcome.html`

OR

`java MyProgram`

Java Program Development and Execution Steps...

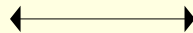
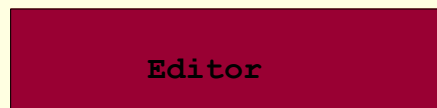
4. Verify

- Bytecode verifier makes sure bytecodes are valid and do not violate security

5. Execute

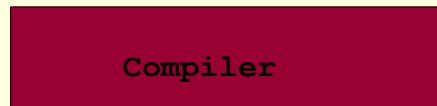
- Computer interprets program one bytecode at a time
- Performs actions specified in program

Phase 1



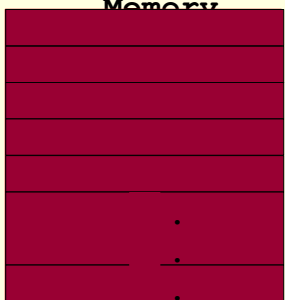
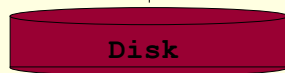
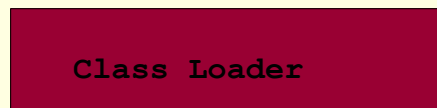
Program is created in the editor and stored on disk.

Phase 2



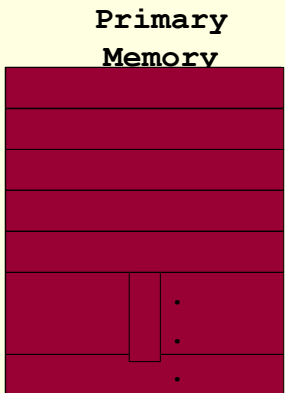
Compiler creates bytecodes and stores them on disk.

Phase 3



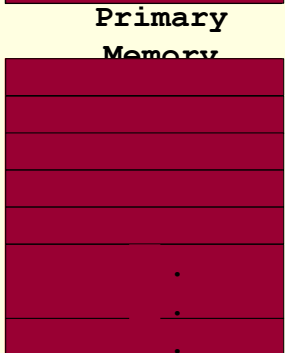
Class loader puts bytecodes in memory.

Phase 4



Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.

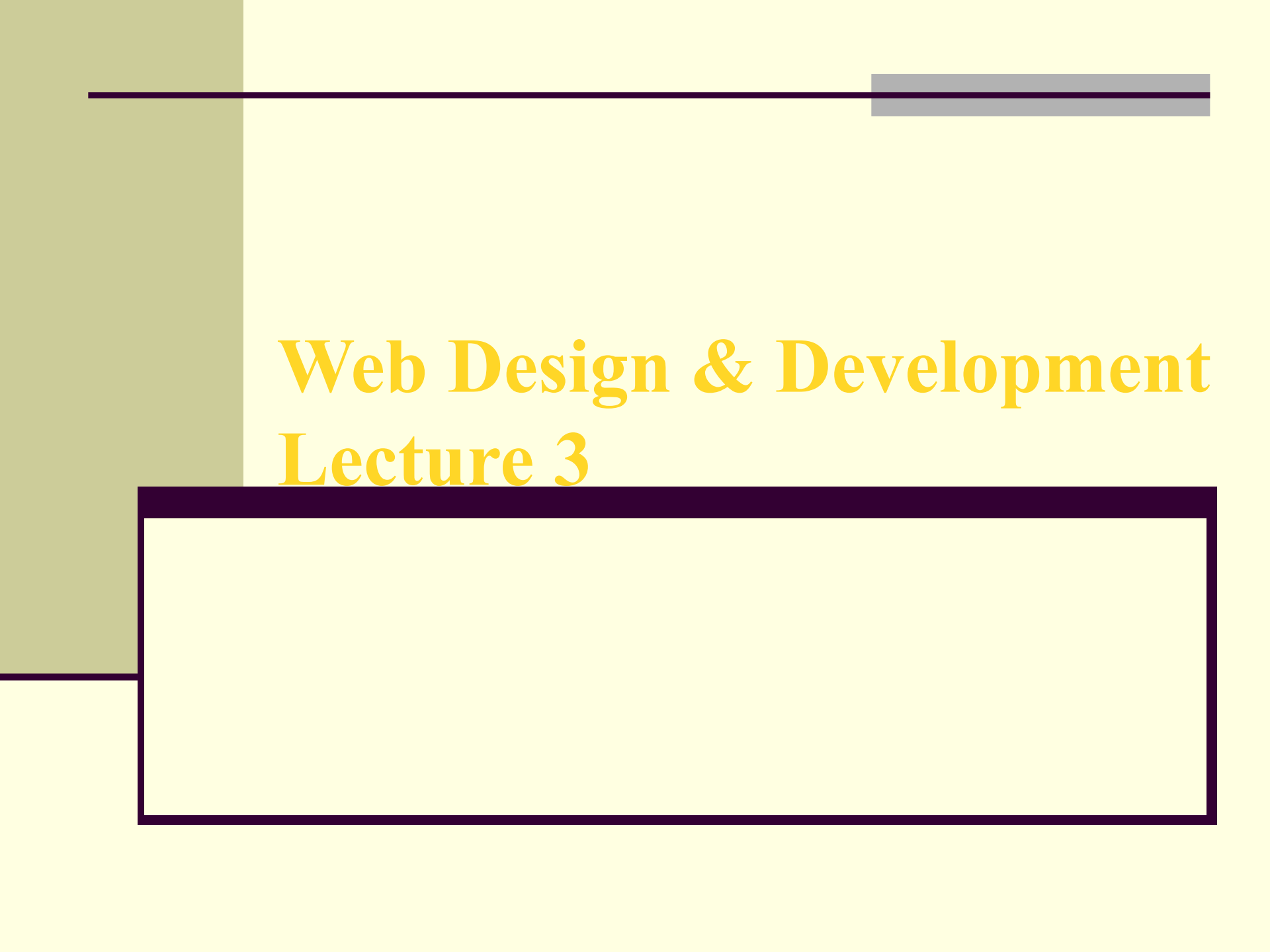
Phase 5



Interpreter reads bytecodes and translates them into a language that the computer can understand, possibly storing data values as the program executes.

Understanding Basics

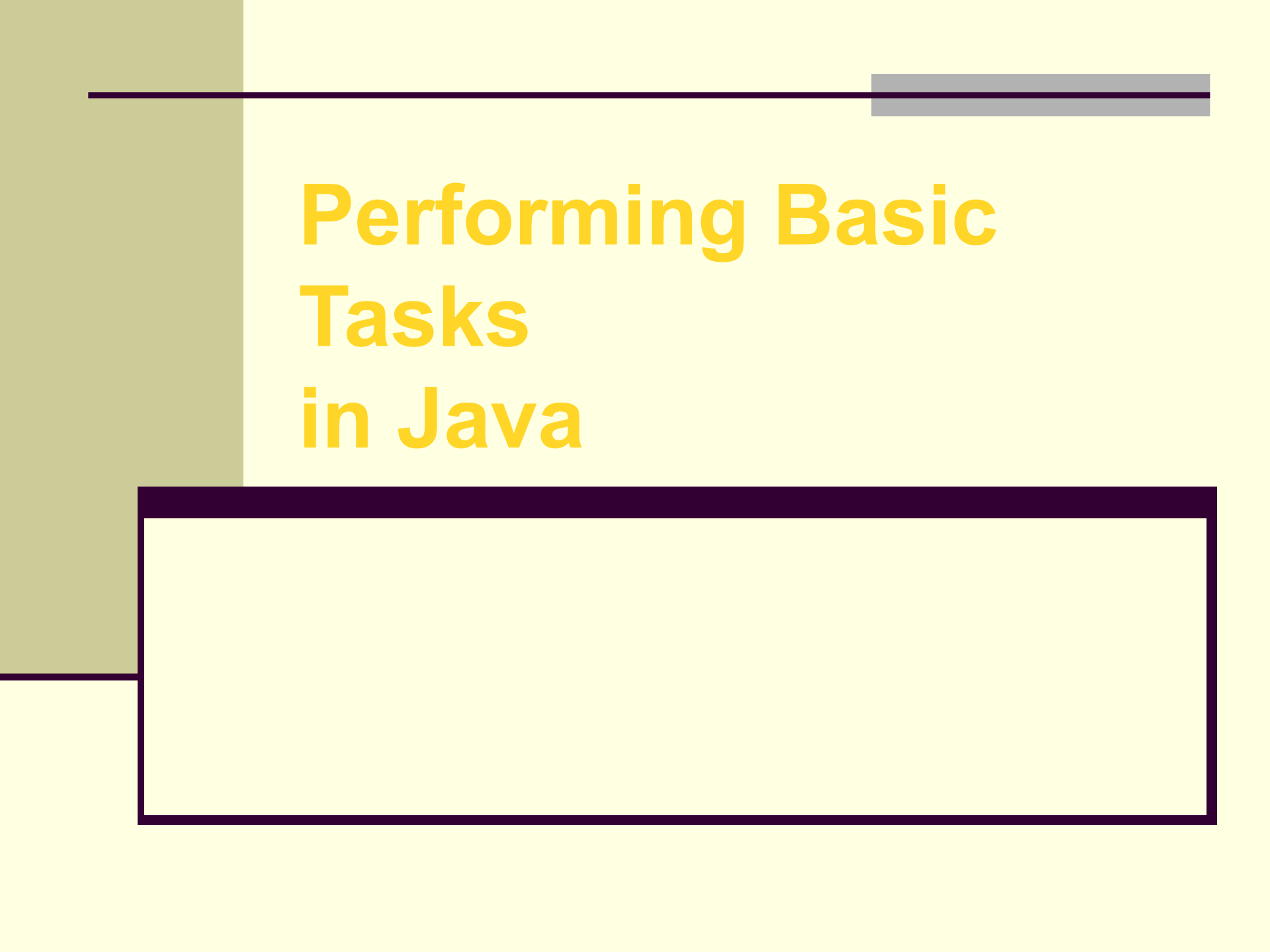
- Naming Conventions
 - MyClass
 - myMethod()
 - myVariable
 - MY_CONSTANT



Web Design & Development

Lecture 3





Performing Basic Tasks in Java



Topics We Will Cover Today

- Things to Remember
- Taking in command line arguments
- Primitives vs. Objects
- Wrapper classes and Conversions
- Taking Input and Output using Swing
- Selection and Control Structures
- OOP in java (Defining and using class)

Last Lecture Example

■ File: HelloWorldApp.java

```
public class HelloWorldApp{  
  
    public static void main(String[] args) {  
  
        System.out.println("Hello world");  
  
    }  
}
```

Compile and Execute

C:\WINDOWS\system32\cmd.exe

D:\vu>set path=C:\Program Files\Java\jdk1.5.0\bin

D:\vu>dir

Volume in drive D is NEW VOLUME
Volume Serial Number is B4BC-8496

Directory of D:\vu

07/09/2005	03:47 AM	<DIR>	.
07/09/2005	03:47 AM	<DIR>	..
07/09/2005	03:57 AM		141 HelloWorldApp.java
	1 File(s)		141 bytes
	2 Dir(s)		13,144,793,088 bytes free

D:\vu>javac HelloWorldApp.java

D:\vu>dir

Volume in drive D is NEW VOLUME
Volume Serial Number is B4BC-8496

Directory of D:\vu

07/09/2005	03:47 AM	<DIR>	.
07/09/2005	03:47 AM	<DIR>	..
07/09/2005	03:57 AM		141 HelloWorldApp.java
07/09/2005	03:59 AM		429 HelloWorldApp.class
	2 File(s)		570 bytes
	2 Dir(s)		13,144,784,896 bytes free

D:\vu>java HelloWorldApp

Hello World

D:\vu>

Things to remember

- Name of file must match name of class
 - It is case sensitive
- Processing starts in main
 - `public static void main(String[] args)`
- Printing is done with `System.out`
 - `System.out.println`, `System.out.print`
- Compile with “javac”
 - Open DOS/command prompt window; work from there
 - Supply full case-sensitive file name (with file extension)
- Execute with “java”
 - Supply base class name (no file extension)

An idiom explained

- You will see the following line of code often:
 - `public static void main(String args[]) { ...}`
- About main()
 - “main” is the function from which your program starts
 - Why public?
 - So that run time can call it from outside
 - Why static ?
 - it is made static so that we can call it without creating an object
 - What is String args[] ?
 - Way of specifying input at startup of application

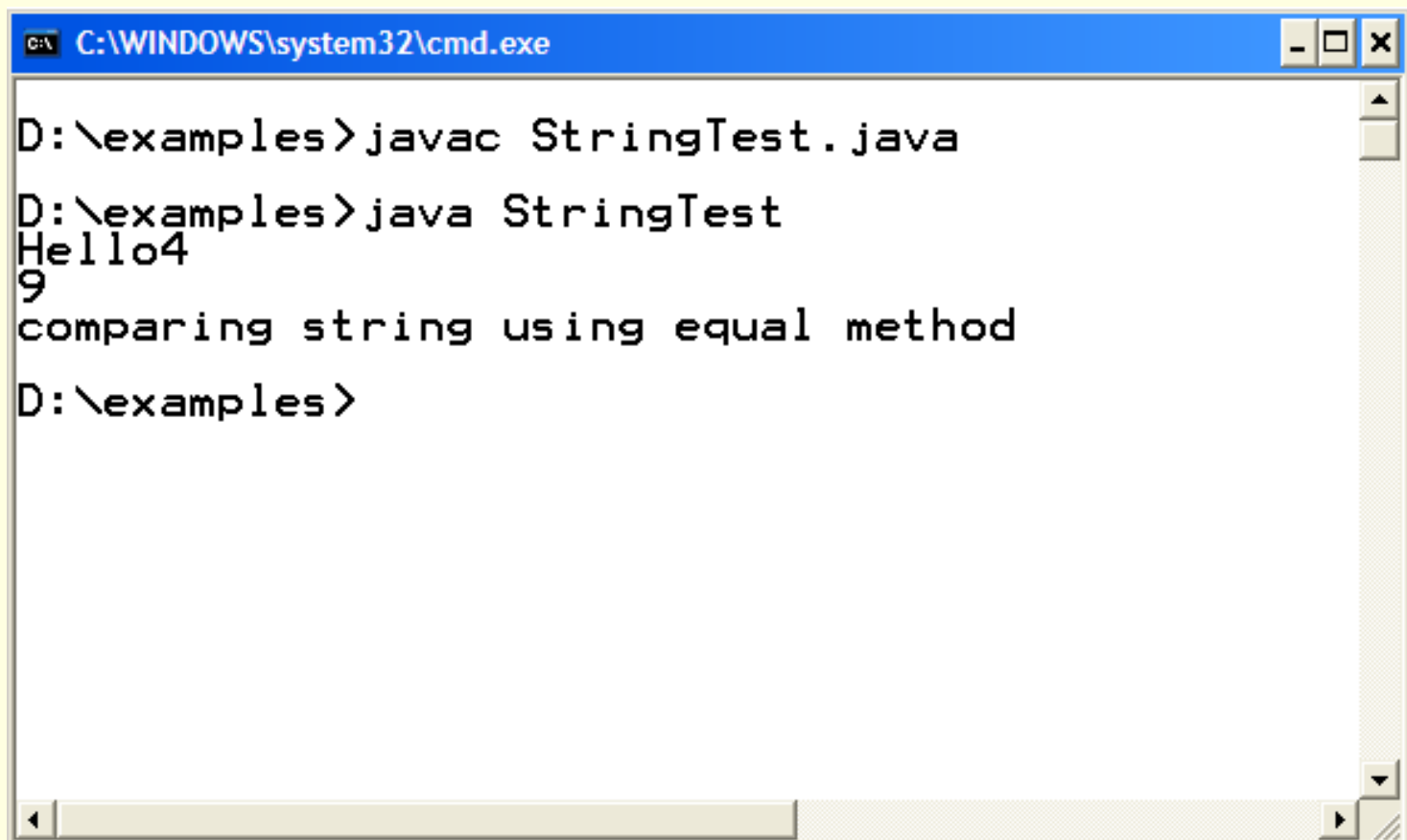
Things to Remember

- “+” operator when used with Strings concatenates them
 - `System.out.println(“Hello” + “World”)` will produce Hello World on console
- String concatenated with any other data type such as int will also convert that datatype to String and the result will be a concatenated String displayed on console
 - For Example
 - `int i = 4`
 - `int j = 5 ;`
 - `System.out.println(“Hello” + i)` // will print Hello 4 on screen
 - However
 - `System.out.println( i+j) ;` // will print 9 on the console
- For comparing Strings never use `==` operator, use equals method.
 - `==` compares addresses (shallow comparison) while equals compares values (deep comparison)
 - E.g `string1.equals(string2)`

String Concatenation

```
public class StringTest {  
  
    public static void main(String[] args) {  
        int i = 4;  
        int j = 5;  
  
        System.out.println("Hello" + i);  
        System.out.println(i + j);  
  
        String s1 = new String ("pakistan");  
        String s2 = "pakistan";  
  
        if (s1 == s2) {  
            System.out.println("comparing string using == operator");  
        }  
  
        if (s1.equals( s2) ) {  
            System.out.println("comparing string using equal method");  
        }  
    }  
}
```

Compile and Execute

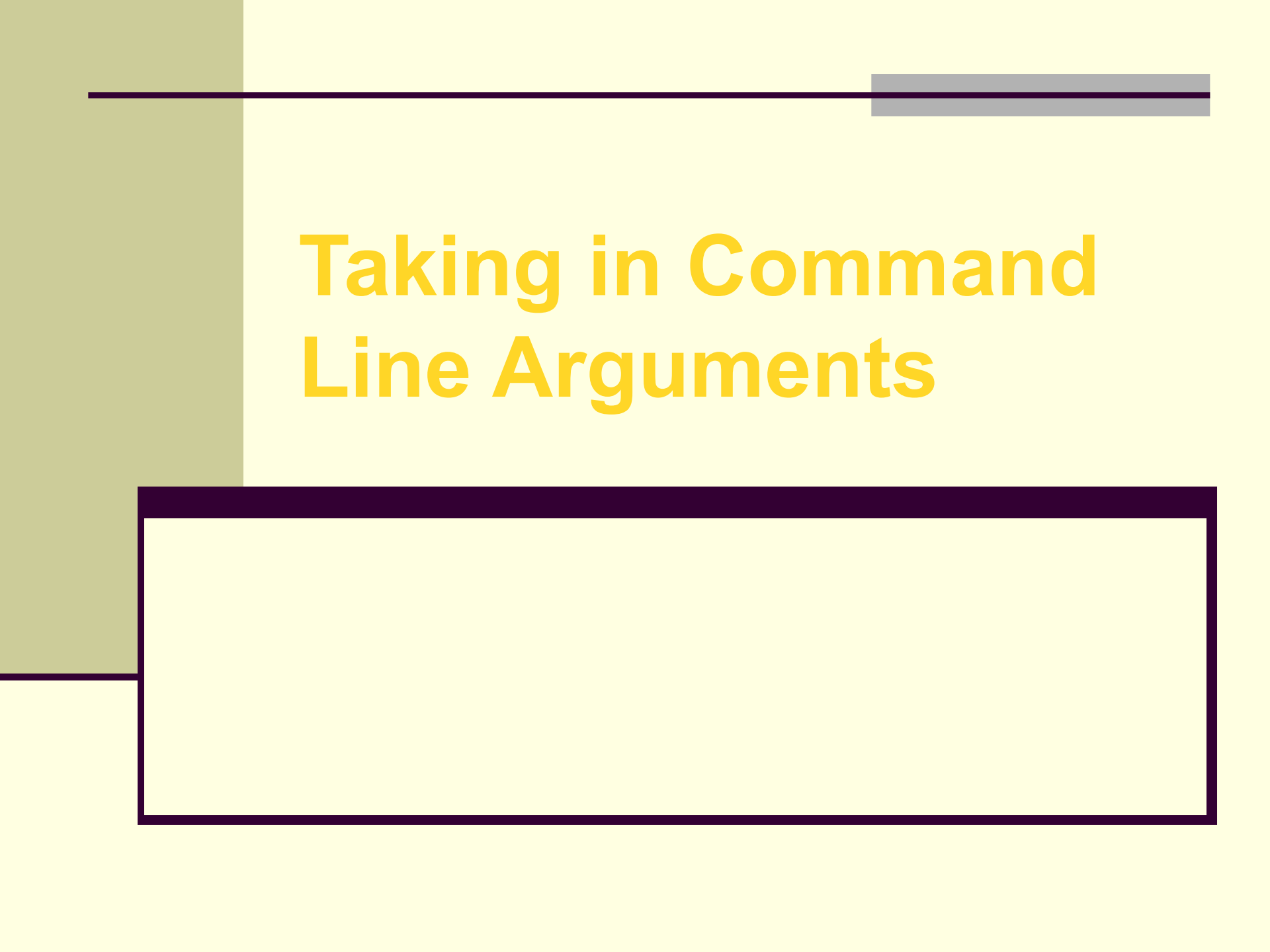


```
C:\WINDOWS\system32\cmd.exe

D:\examples>javac StringTest.java
D:\examples>java StringTest
Hello4
9
comparing string using equal method
D:\examples>
```

The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window contains the following text:

```
D:\examples>javac StringTest.java
D:\examples>java StringTest
Hello4
9
comparing string using equal method
D:\examples>
```



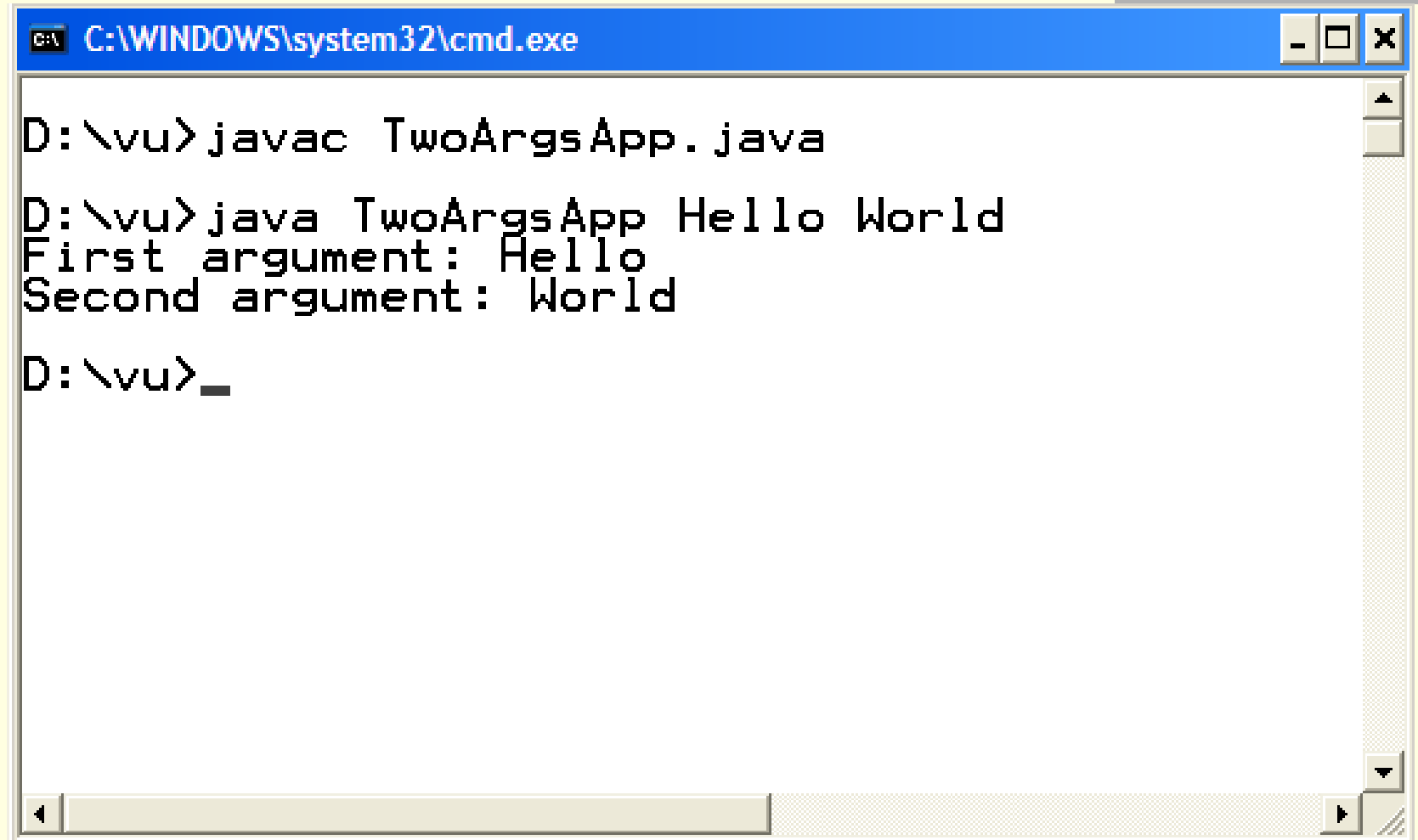
Taking in Command Line Arguments

Taking in Command Line Arguments

/* This program will take two arguments *Hello World* from the command prompt and prints them to standard console. If you specify less than two arguments an exception will be thrown */

```
public class TwoArgsApp {  
  
    public static void main(String[] args) {  
  
        //Displays the first argument on console  
        System.out.println("First argument " + args[0]);  
  
        //Displays the second argument on console  
        System.out.println("Second argument " + args[1]);  
    }  
}
```

Compile and Execute



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" along with standard window control buttons (minimize, maximize, close). The command prompt shows the following sequence of commands and output:

```
D:\vu>javac TwoArgsApp.java
D:\vu>java TwoArgsApp Hello World
First argument: Hello
Second argument: World
D:\vu>_
```

The output shows that the Java program "TwoArgsApp" was successfully compiled and executed, displaying the two command-line arguments "Hello" and "World".

Passing any Number of Arguments

```
/* This program is able to receive any number of arguments and prints them to console using for loop. In  
   java, arrays knows about their size by using length property  
*/
```

```
public class AnyArgsApp {
```

```
    public static void main(String[] args) {
```

```
        for (int i=0; i<args.length; i++)  
        {
```

```
            // The "+" operator here works similar to "<<" operator in C++. This line is  
            // equivalent to cout<<"Arguments:"<<i<<"value"<<args[i];  
            // where cout is replaced by System.out.println, and "<<" is replaced by + for  
            // concatenation
```

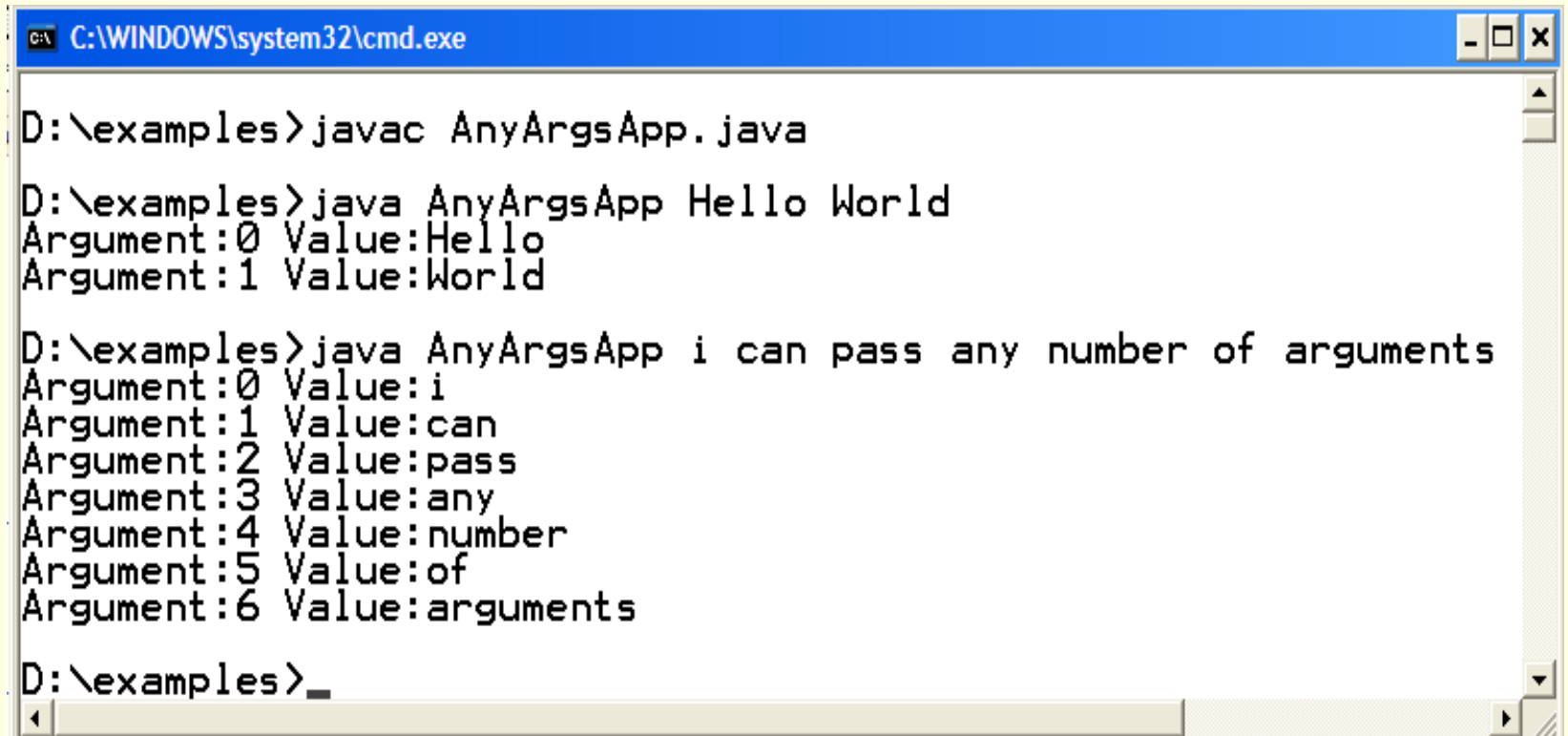
```
            System.out.println("Argument:" + i + "value: " + args[i] );
```

```
        }
```

```
    }
```

```
}
```

Compile and Execute



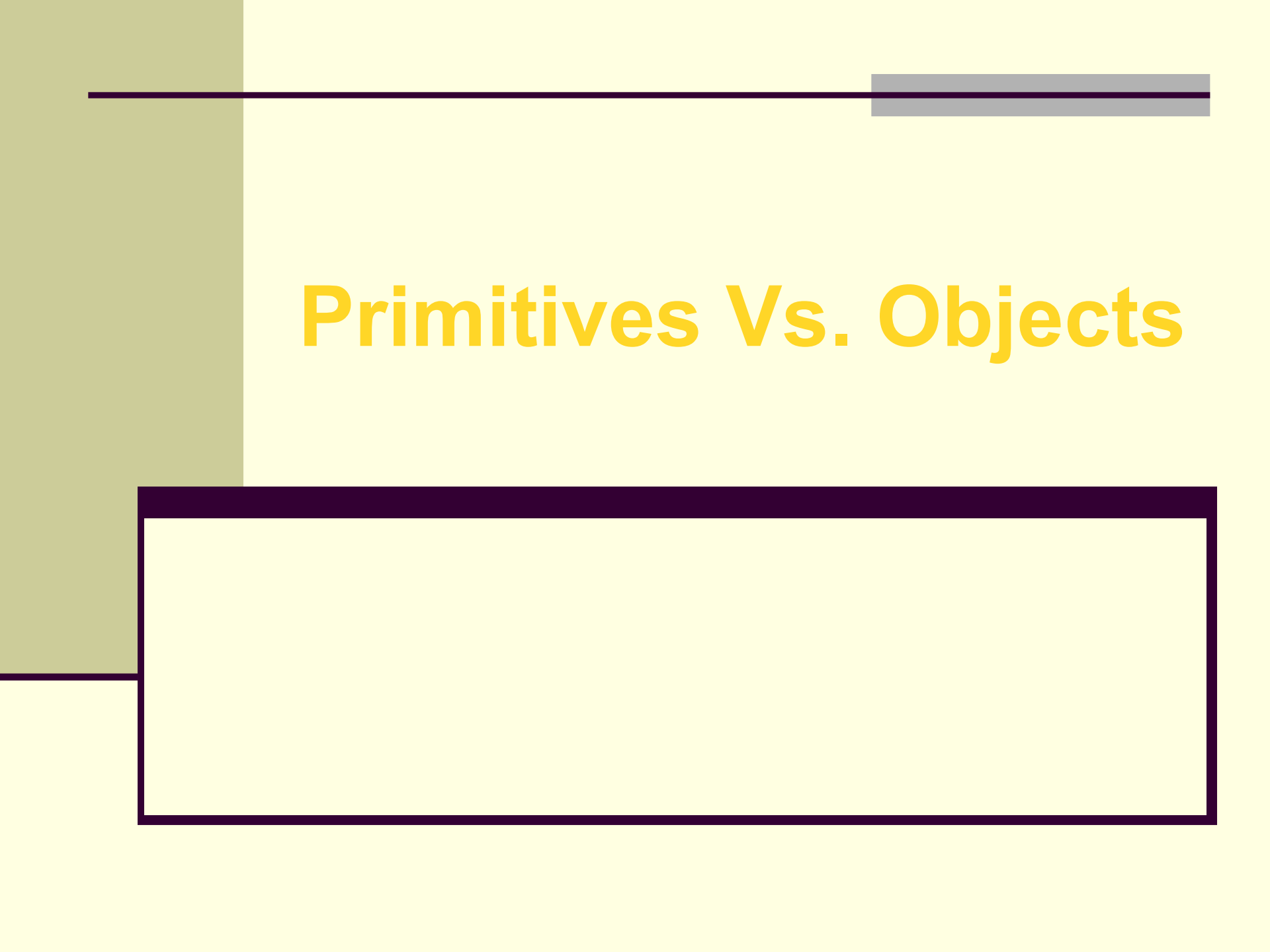
```
C:\WINDOWS\system32\cmd.exe

D:\examples>javac AnyArgsApp.java

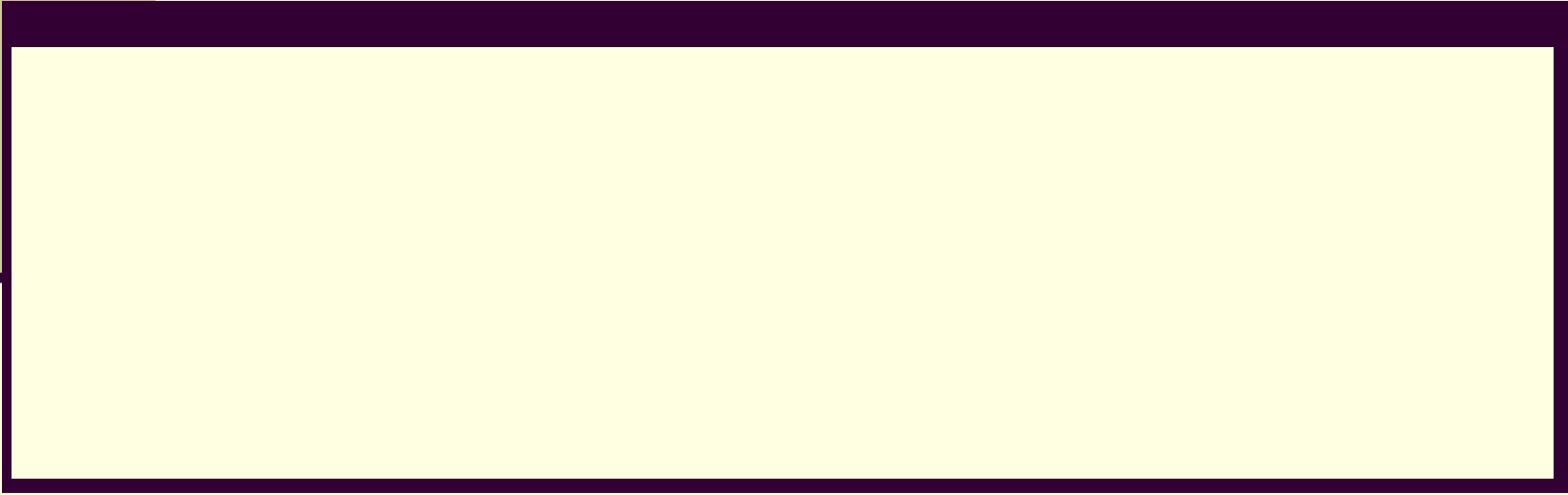
D:\examples>java AnyArgsApp Hello World
Argument:0 Value:Hello
Argument:1 Value:World

D:\examples>java AnyArgsApp i can pass any number of arguments
Argument:0 Value:i
Argument:1 Value:can
Argument:2 Value:pass
Argument:3 Value:any
Argument:4 Value:number
Argument:5 Value:of
Argument:6 Value:arguments

D:\examples>_
```

Primitives Vs. Objects



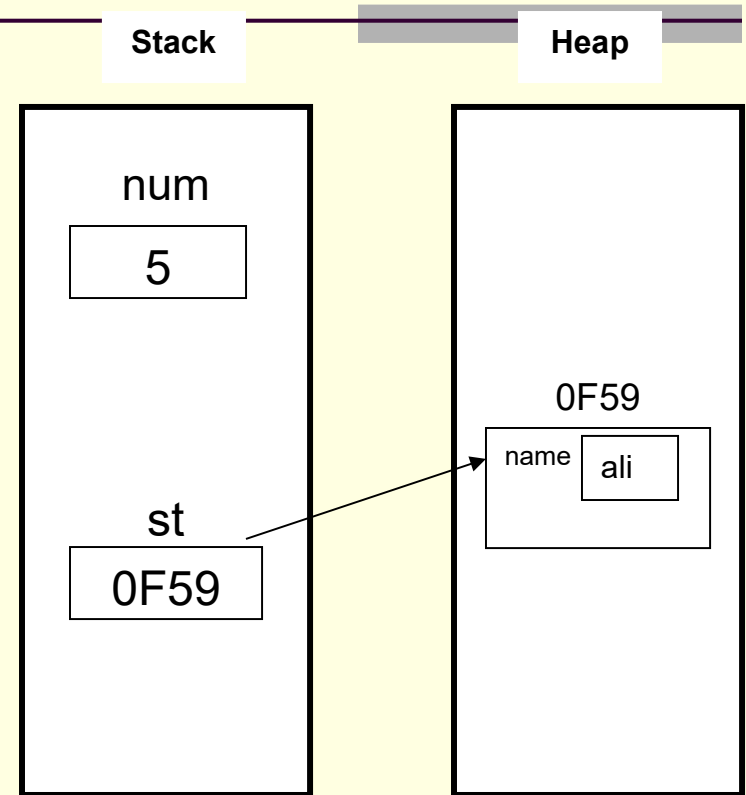
Primitives Vs. Objects

- Everything in Java is an “Object”, as every class by default inherits from class “Object” , except a few primitive data types, which are there for efficiency reasons.
- Primitive Data Types
 - 8 Primitive Data types of java
 - boolean, byte → 1 byte
 - char, short → 2 bytes
 - int, float → 4 bytes
 - long, double → 8 bytes
- Primitive data types are generally used for local variables, parameters and instance variables (properties of an object)
- Primitive datatypes are located on the stack and we can only access their value, while objects are located on heap and we have a reference to these objects
- Also primitive data types are always passed by value while objects are always passed by reference in java. There is no C++ like methods
 - `void someMethod(int &a, int &b )` // not available in java

Stack vs. Heap

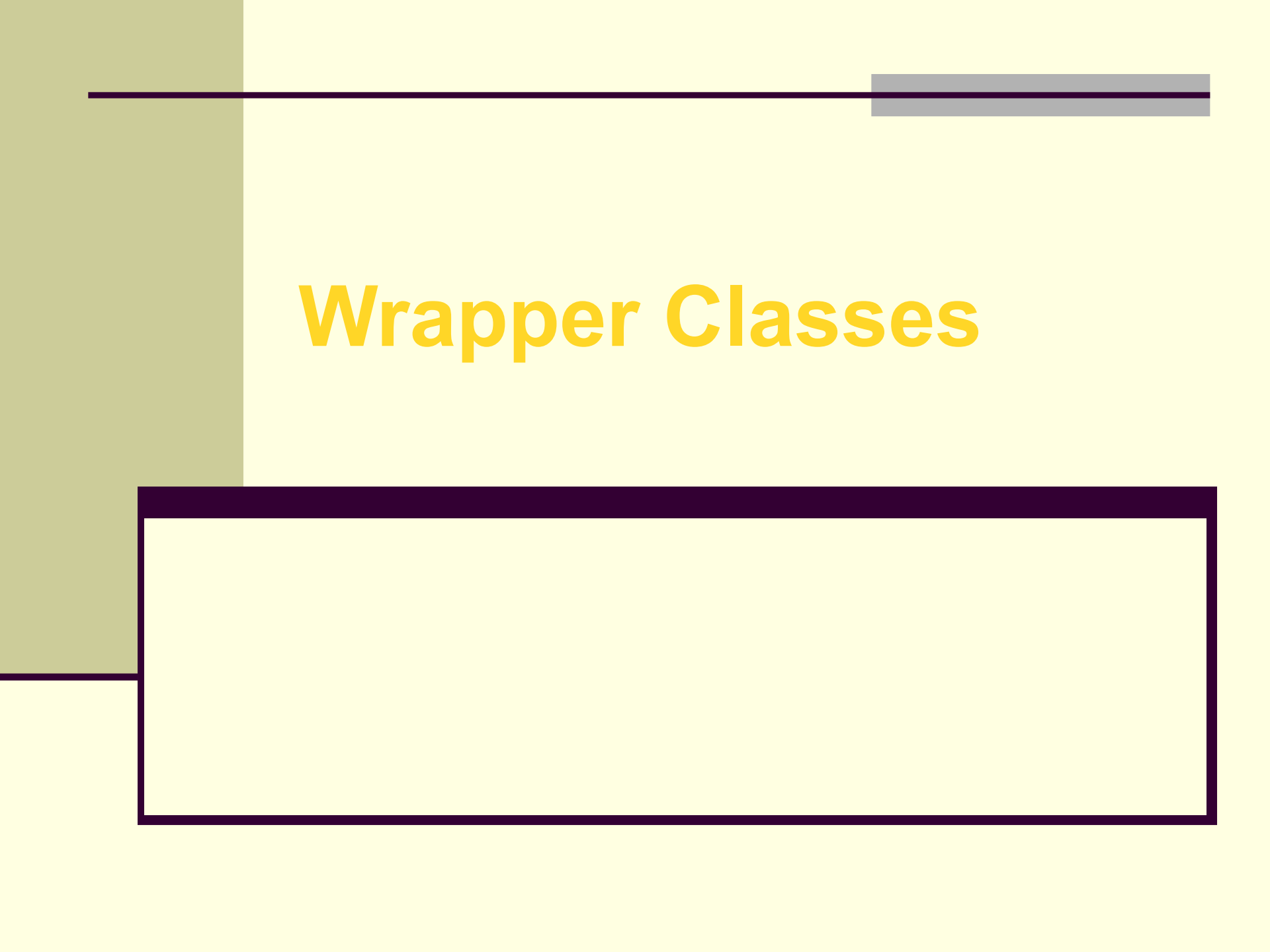
```
public static void main(String args[])
{
    int num= 5;

    Student st = new Student();
}
```

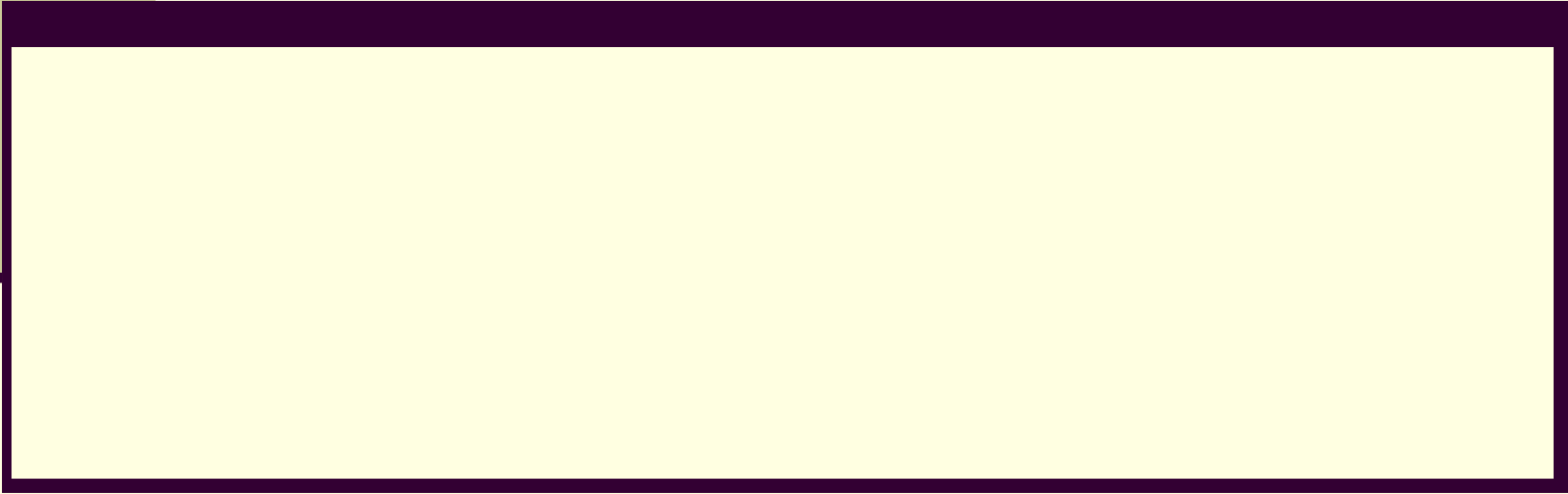


Primitives (cont)

- For all built-in primitive data types java uses lowercase. E.g int , float etc
- Primitives can be stored in arrays
- You cannot get a reference to a primitive
 - To do that you need an Object or a Wrapper class



Wrapper Classes



Wrapper Classes

- Each primitive data type has a corresponding object (wrapper class)
- These Wrapper classes provides additional functionality (conversion, size checking etc), which a primitive data type can not provide

Primitive Data Type	Corresponding Object Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

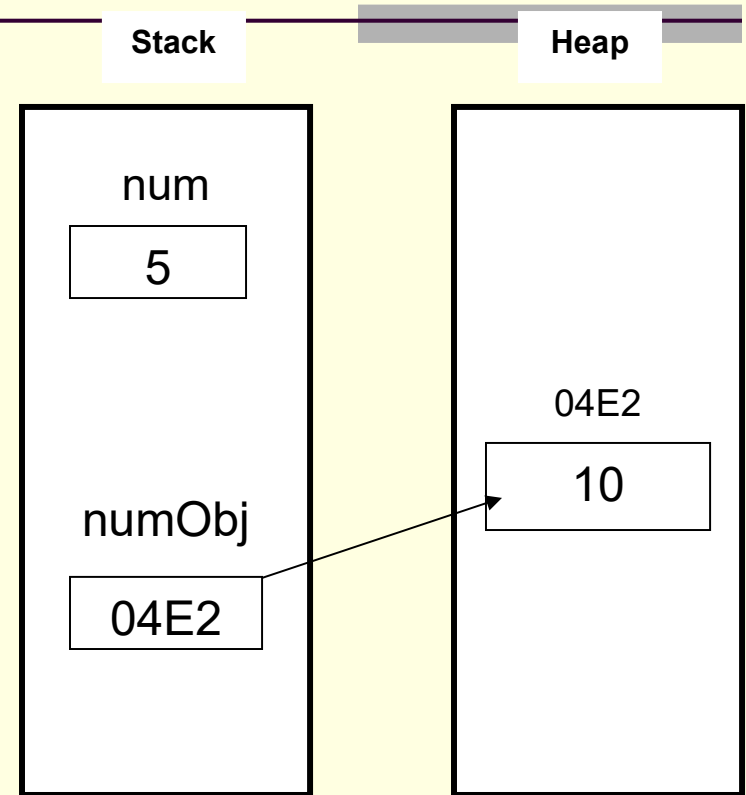
Wrapper Use

- You can create an object of Wrapper class using a String or a primitive data type
 - `Integer num = new Integer(4);` or
 - `Integer num = new Integer("4");`
 - Num is an object over here not a primitive data type
- You can get a primitive data type from a Wrapper using the corresponding value function
 - `int primNum = num.intValue();`

Stack vs. Heap

```
public static void main(String args[])
{
    int num= 5;

    Integer numObj = new Integer (10);
}
```



Wrapper Uses

- Defines useful constants for each data type
 - For example,

`Integer.MAX_VALUE`

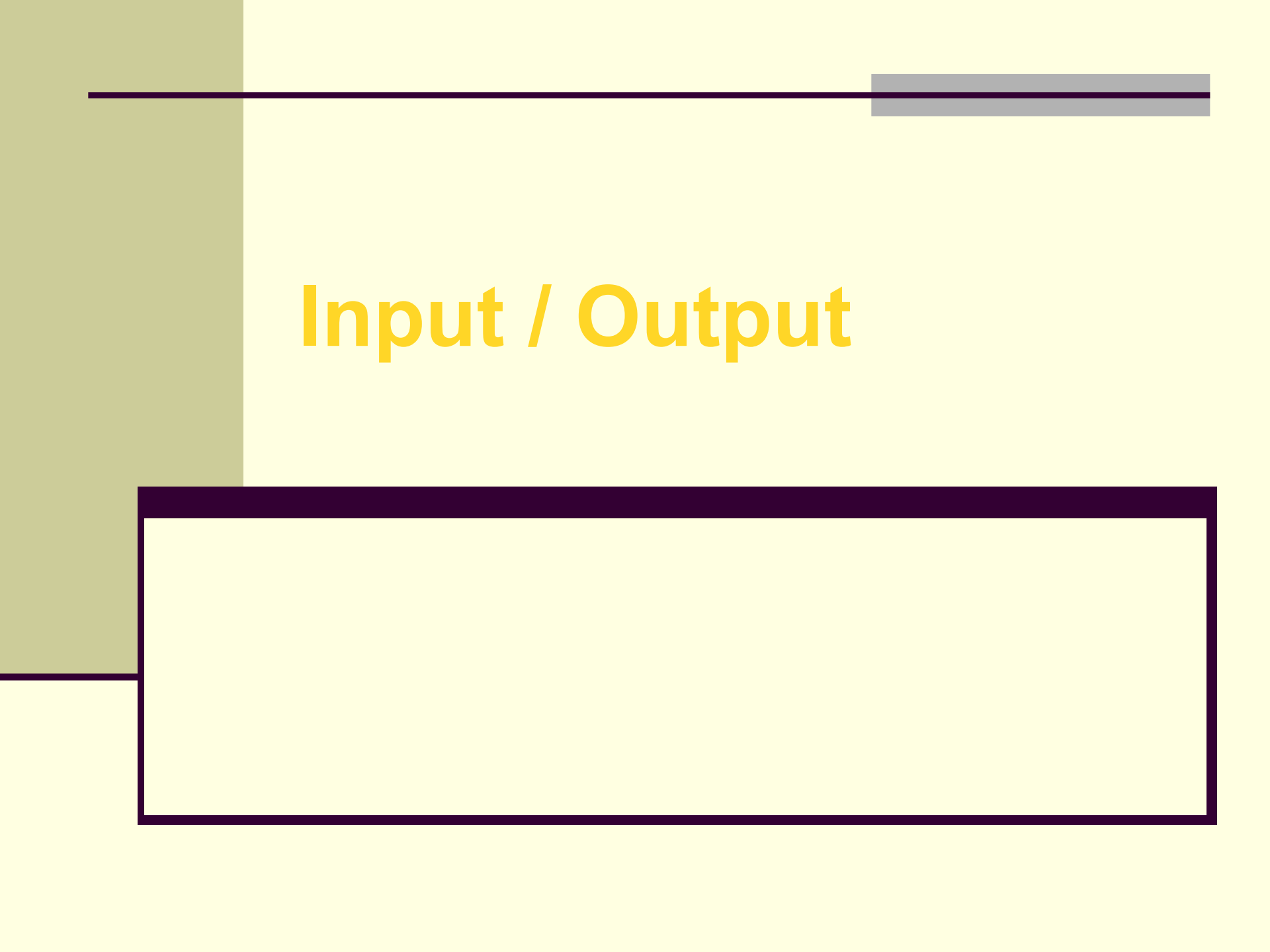
- Convert between data types
 - Use `parseXxx` method to convert a String to the corresponding primitive data type
 - **String value = "532";**
int d = Integer.parseInt(value);
 - **String value = "3.14e6";**
double d = Double.parseDouble(value);

Wrappers: Converting Strings

Data Type	Convert String using either ...
byte	<code>Byte.parseByte(<i>string</i>)</code> <code>new Byte(<i>string</i>).byteValue()</code>
short	<code>Short.parseShort(<i>string</i>)</code> <code>new Short(<i>string</i>).shortValue()</code>
int	<code>Integer.parseInt(<i>string</i>)</code> <code>new Integer(<i>string</i>).intValue()</code>
long	<code>Long.parseLong(<i>string</i>)</code> <code>new Long(<i>string</i>).longValue()</code>
float	<code>Float.parseFloat(<i>string</i>)</code> <code>new Float(<i>string</i>).floatValue()</code>
double	<code>Double.parseDouble(<i>string</i>)</code> <code>new Double(<i>string</i>).doubleValue()</code>

Wrapper Uses

- When a method does not except an int primitive but still you need to pass an int value, you can use the corresponding Wrapper.
 - `someVector.add(new Integer(4) ); // this was required prior to jdk5.0 the`
|
- Boxing/Unboxing Conversions
 - New feature added in j2se 5.0
 - Boxing
 - `Integer iWrapper = 10;`
 - Prior to J2SE 5.0, we use
 - `Integer a = new Integer(10);`
 - Unboxing
 - `int iPrimitive = iWrapper;`
 - Prior to J2SE 5.0, we use
 - `int b = iWrapper.intValue();`



Input / Output



Console based Output

System.out

- System class
 - Out represents the screen
- System.out.println()
 - Prints the string followed by an end of line
 - Forces a flush
- System.out.print()
 - Does not print the end of line
 - Does not force a flush
- System.out.flush()
 - Force a flush

Input / Output

```
/* This program will takes the input (number) through GUI and prints its square on the console as well as on  
the GUI. */
```

```
import javax.swing.*;
```

```
public class InputOutputTest {
```

```
    public static void main(String[] args) {
```

```
        //takes input through GUI
```

```
        String input = JOptionPane.showInputDialog("Enter the number");
```

```
        int number = Integer.parseInt(input);
```

```
        int square = number * number;
```

```
        //Display square on console
```

```
        System.out.println("square:" + square);
```

```
        //Display square on GUI
```

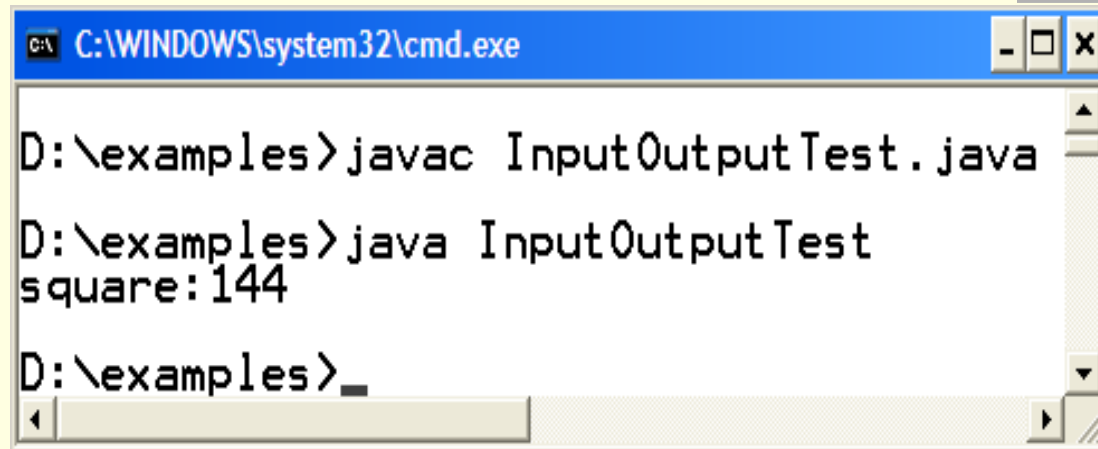
```
        JOptionPane.showMessageDialog(null, "square:" + square);
```

```
        System.exit(0); //Don't forget to write when using JOptionPane. Don't need it in  
        //J2SE 5.0
```

```
    }
```

```
}
```

Compile and Execute

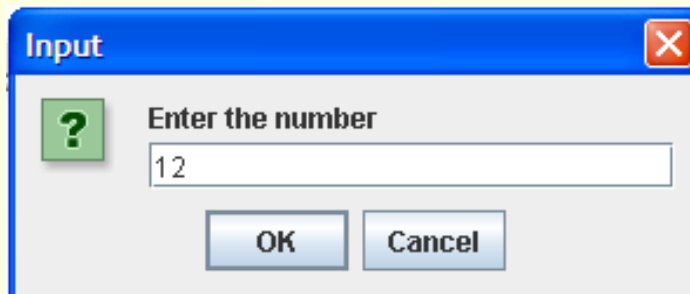


```
C:\WINDOWS\system32\cmd.exe

D:\examples>javac InputOutputTest.java

D:\examples>java InputOutputTest
square:144

D:\examples>_
```

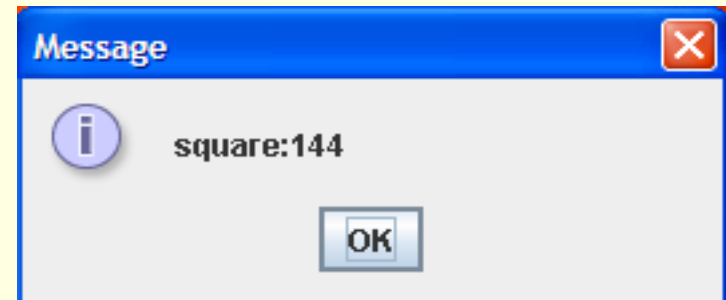


Input

Enter the number

12

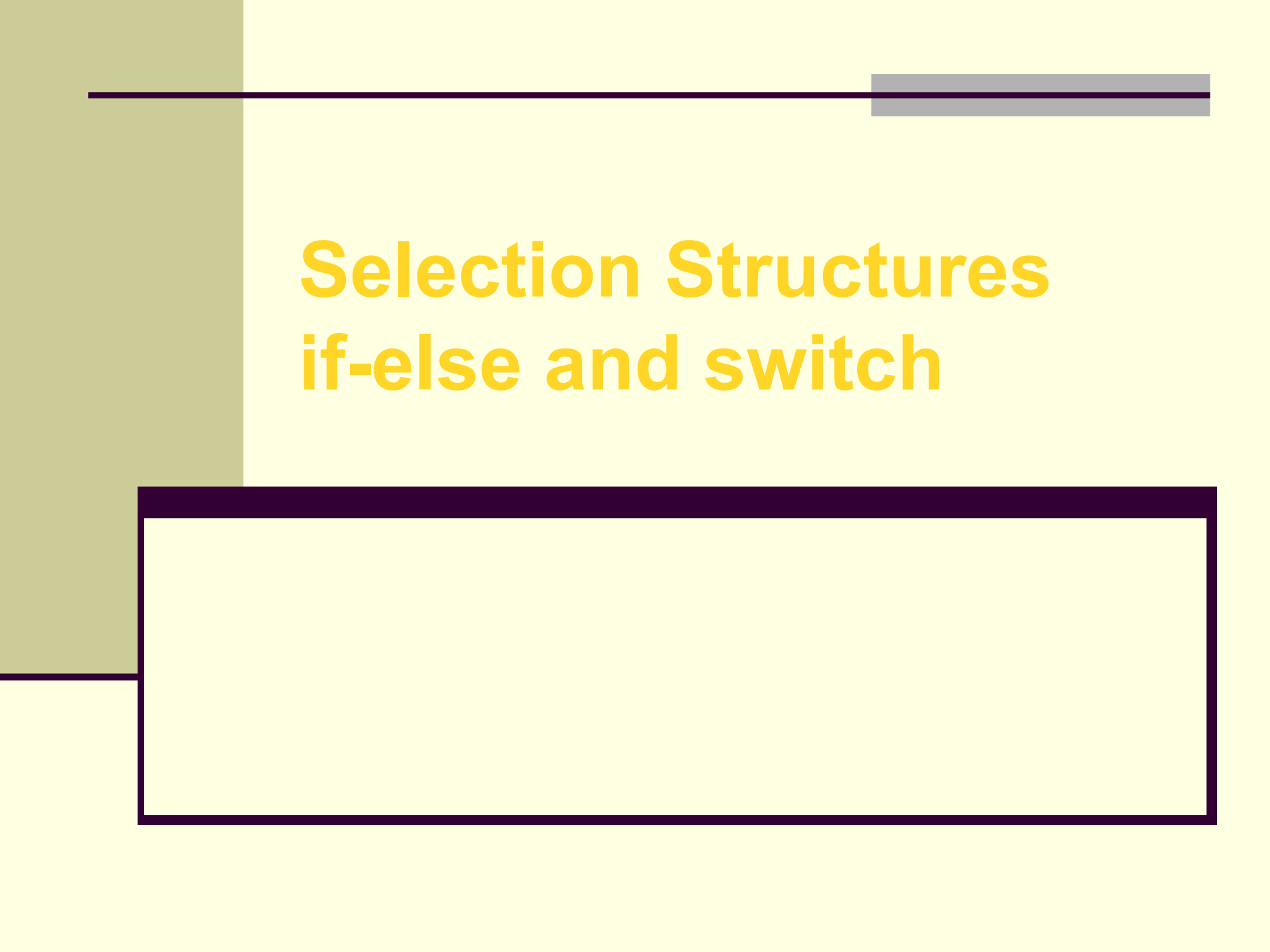
OK Cancel



Message

square:144

OK



Selection Structures

if-else and switch



if–else Selection Structure

/* This program will demonstrates the use of if-else selection structure. Note that its syntax is very similar to C++

*/

```
public class IfElseTest {
```

```
    public static void main(String[] args) {
```

```
        int firstNumber    = 10;
```

```
        int secondNumber = 20;
```

```
        //comparing first number with second number
```

```
        if (firstNumber > secondNumber) {
```

```
            System.out.println("first number is greater than second");
```

```
        }
```

```
        else if (firstNumber == secondNumber) {
```

```
            System.out.println("first number is equals to second number");
```

```
        }
```

```
        else {
```

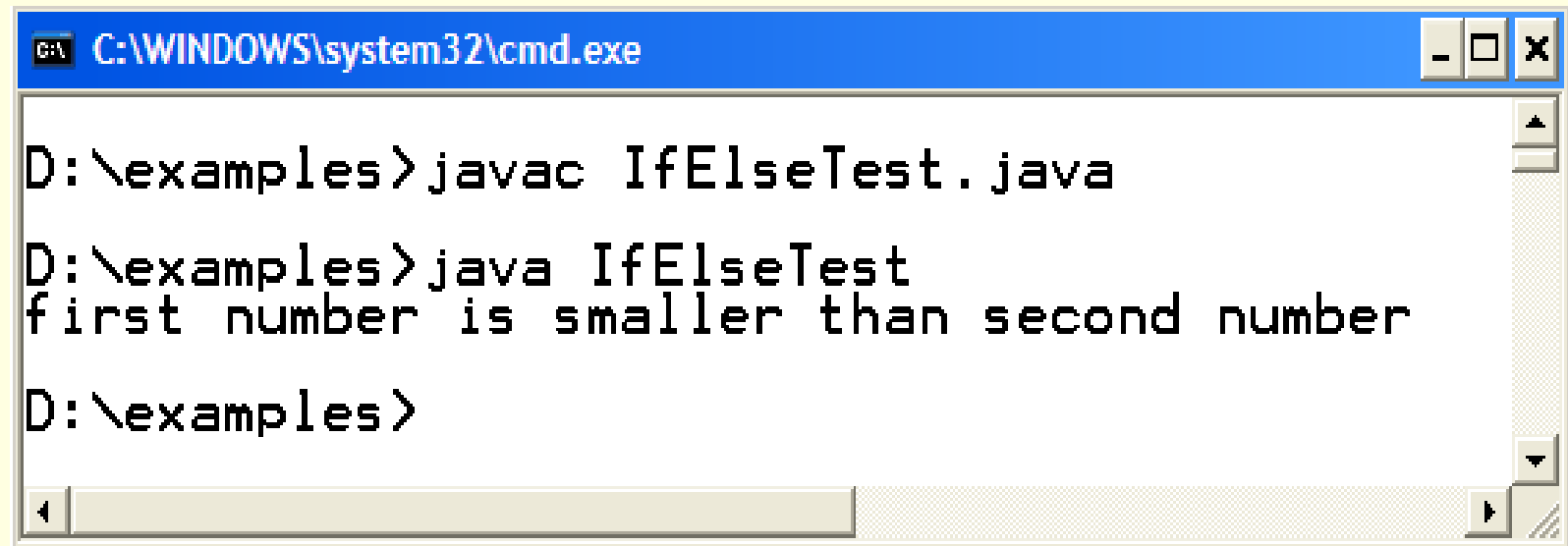
```
            System.out.println("first number is smaller than second number");
```

```
        }
```

```
    }
```

```
}
```

Compile and Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples>javac IfElseTest.java

D:\examples>java IfElseTest
first number is smaller than second number

D:\examples>
```

The screenshot shows a Windows command prompt window with a blue title bar that reads "C:\WINDOWS\system32\cmd.exe". The window contains three lines of text: a command prompt "D:\examples>", the compilation command "javac IfElseTest.java", another command prompt "D:\examples>", the execution command "java IfElseTest", and the output "first number is smaller than second number". A third command prompt "D:\examples>" is visible at the bottom. The window has standard Windows controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

Boolean Operators

- `==, !=`
 - Equality, inequality. In addition to comparing primitive types, `==` tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects.
- `<, <=, >, >=`
 - Numeric less than, less than or equal to, greater than, greater than or equal to.
- `&&, ||`
 - Logical AND, OR. Both use short-circuit evaluation to more efficiently compute the results of complicated expressions.
- `!`
 - Logical negation.

switch Selection Structure

```
import javax.swing.*;
public class SwitchTest {

    public static void main(String[] args) {
        int operand1 = 10;
        int operand2 = 20;

        String choice = JOptionPane.showInputDialog("Enter 1 for sum,
            2 for product");
        int ch = Integer.parseInt(choice);

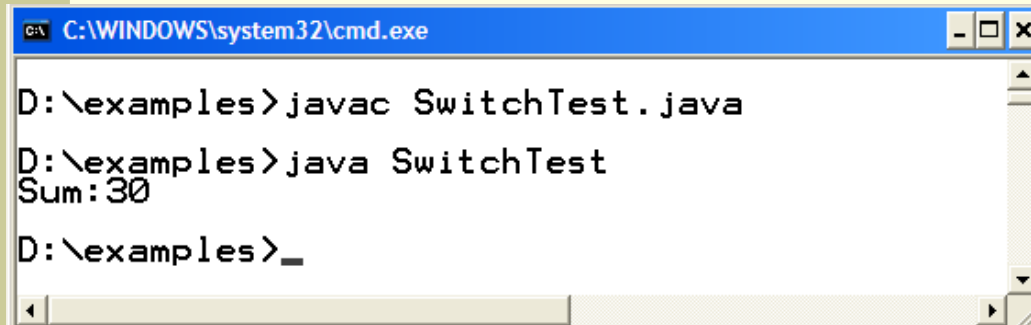
        // continue....
```

switch Selection Structure...

```
switch(ch)
{
    case 1:
        int sum = operand1 + operand2;
        System.out.println("sum: " + sum );
        break;
    case 2:
        int product = operand1 * operand2;
        System.out.println("product: " + product );
        break;
    default:
        System.out.println("wrong choice!");
}
```

```
System.exit(0);
}
}
```

Compile and Execute

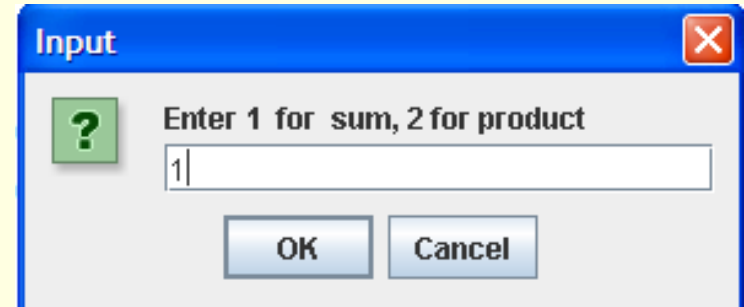


```
C:\WINDOWS\system32\cmd.exe

D:\examples>javac SwitchTest.java

D:\examples>java SwitchTest
Sum: 30

D:\examples>_
```

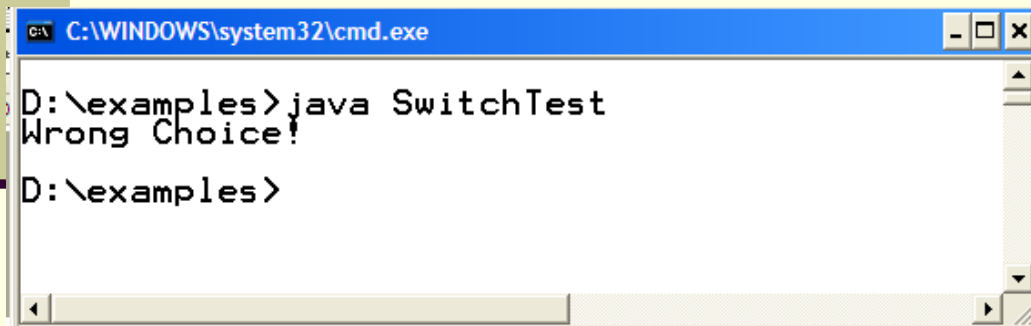


Input

Enter 1 for sum, 2 for product

1

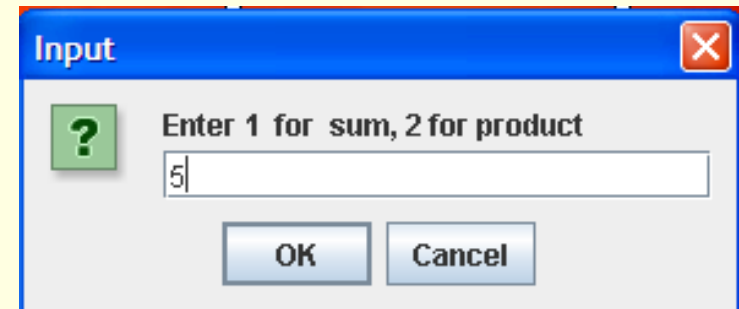
OK Cancel



```
C:\WINDOWS\system32\cmd.exe

D:\examples>java SwitchTest
Wrong Choice!

D:\examples>
```

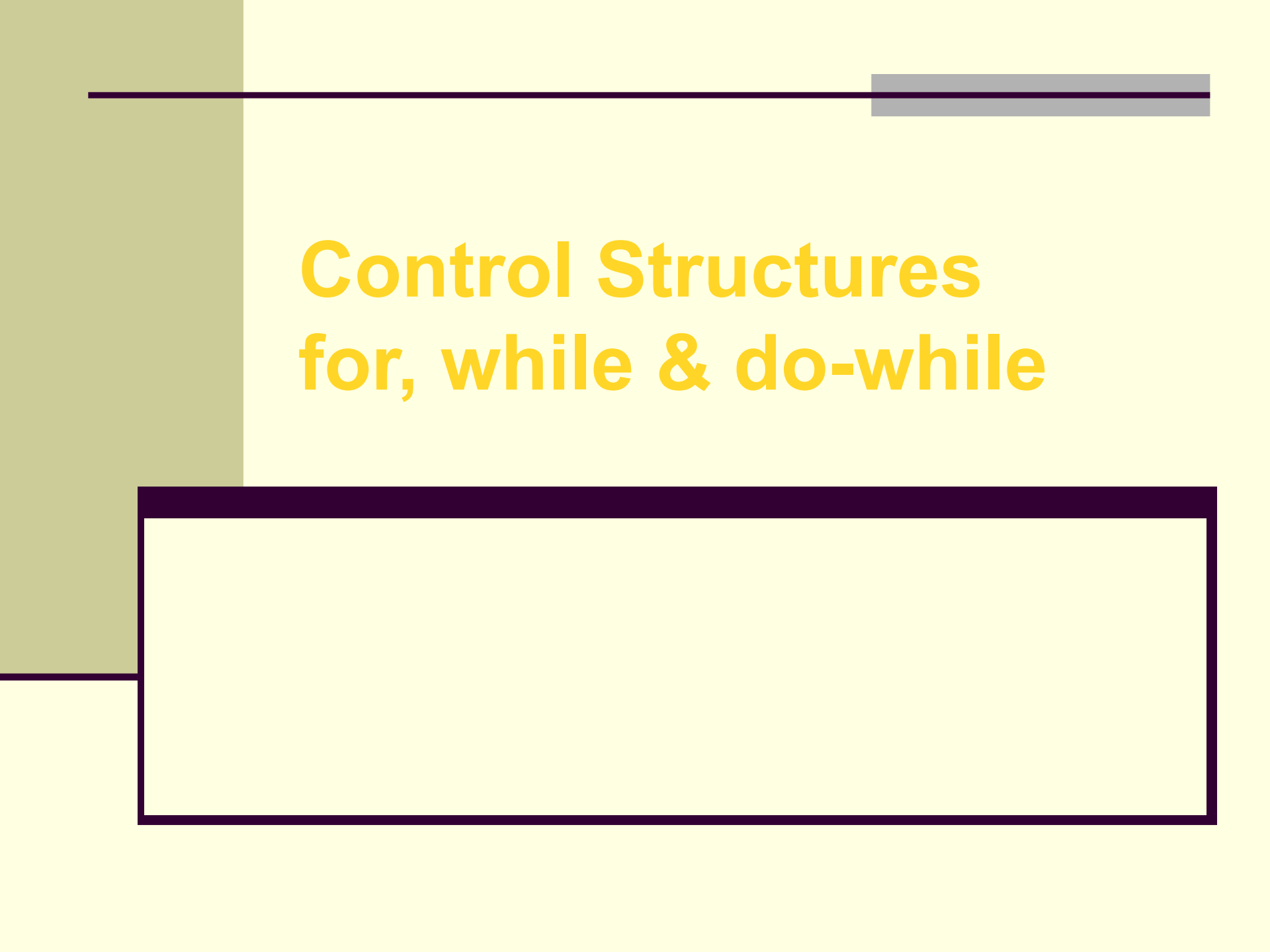


Input

Enter 1 for sum, 2 for product

5

OK Cancel



Control Structures for, while & do-while



Looping Constructs

- while

```
while (continueTest) {  
    body;  
}
```

- do

```
do {  
    body;  
} while (continueTest);  
// ^ don't forget semicolon
```

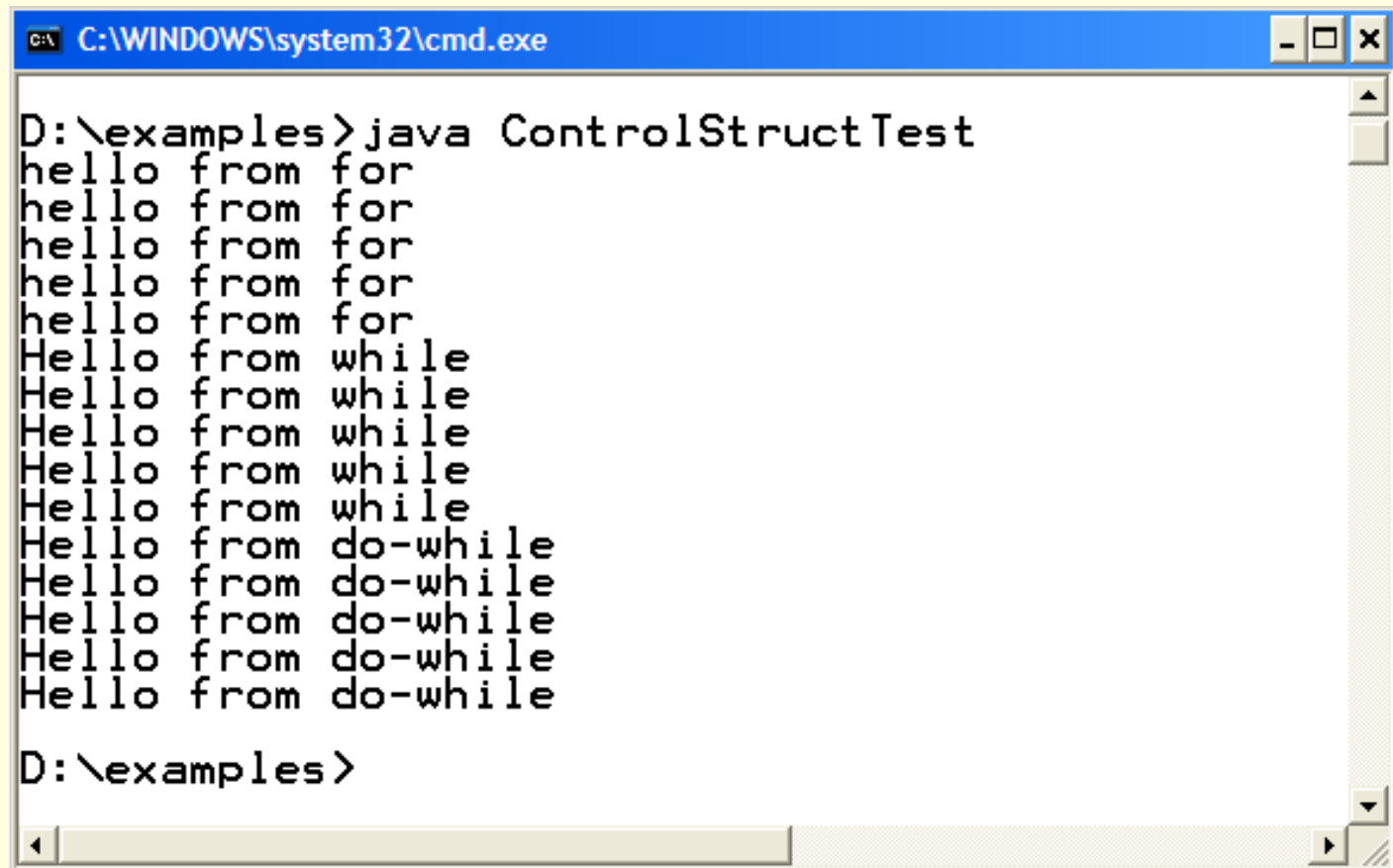
- for

```
for (init; continueTest; updateOp) {  
    body;  
}
```


Control Structures

```
public class ControlStructTest {  
    public static void main(String[] args) {  
        // for loop  
        for (int i=1; i<= 5; i++) {  
            System.out.println("hello from for");  
        }  
  
        // while loop  
        int j = 1;  
        while (j <= 5) {  
            System.out.println("Hello from while");  
            j++;  
        }  
  
        //do while loop  
        int k =1;  
        do{  
            System.out.println("Hello from do-while");  
            k++;  
        }while(k <= 5);  
    }  
}
```

Compile and Execute



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" and standard window control buttons (minimize, maximize, close). The command prompt shows the following text:

```
D:\examples>java ControlStructTest
hello from for
hello from for
hello from for
hello from for
hello from for
Hello from while
Hello from while
Hello from while
Hello from while
Hello from while
Hello from do-while
Hello from do-while
Hello from do-while
Hello from do-while
Hello from do-while
D:\examples>
```

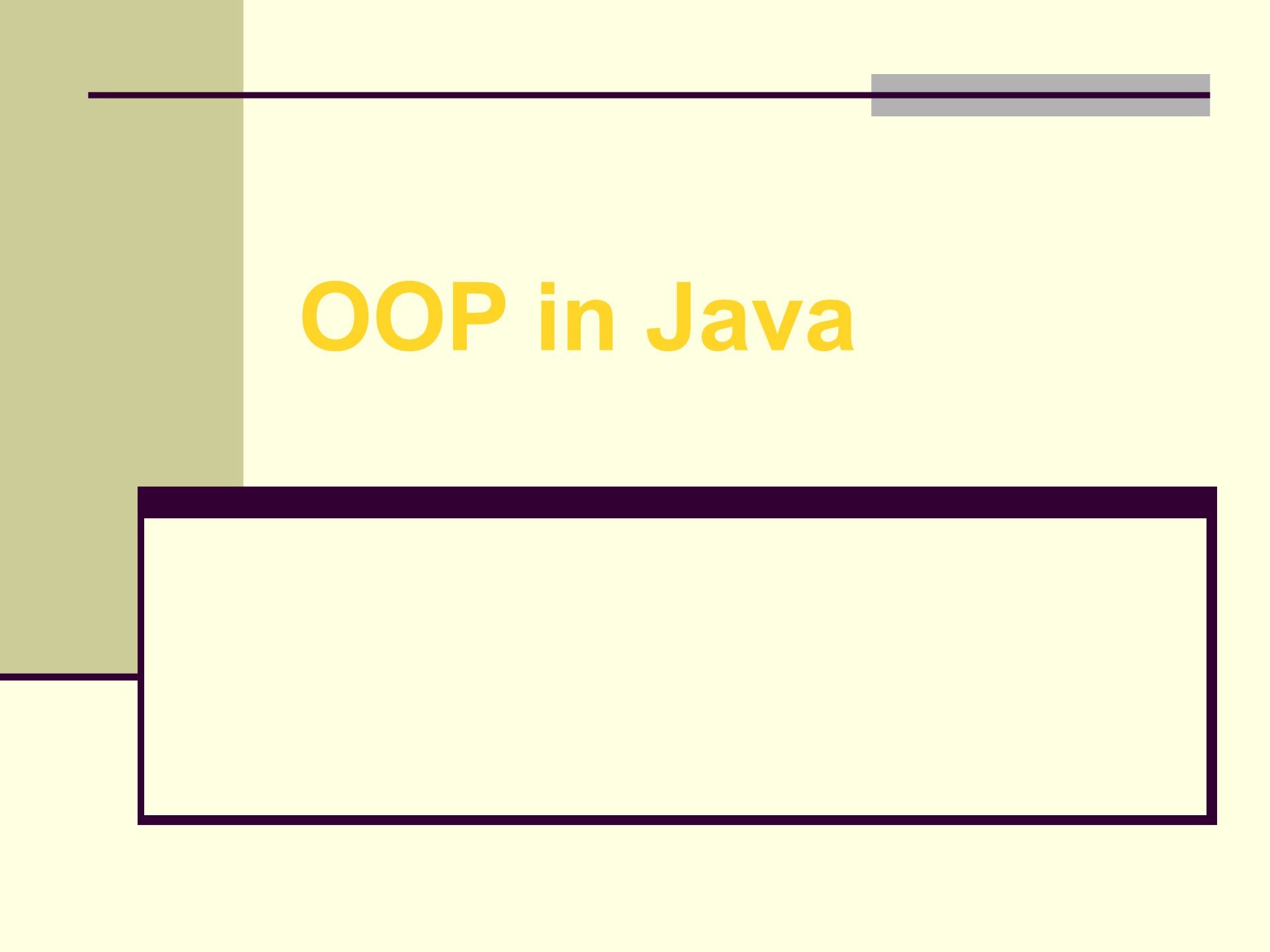
The output consists of 15 lines. The first five lines are "hello from for", the next five lines are "Hello from while", and the last five lines are "Hello from do-while". The command prompt has a scroll bar on the right and a status bar at the bottom.



Web Design & Development

Lecture 4





OOP in Java

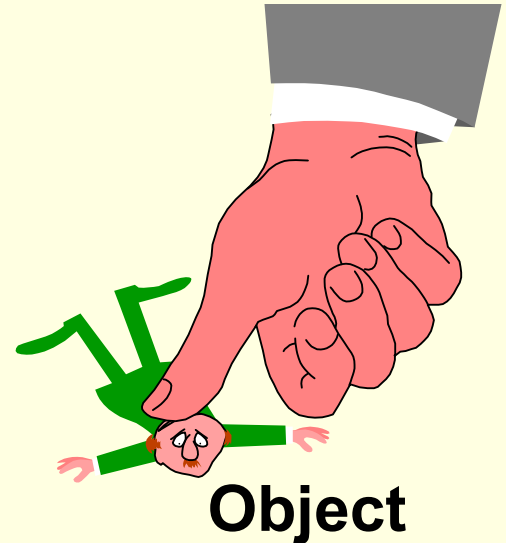


OOP in Java

- Java is fundamentally Object-Oriented
 - Every line of code you write in Java must be inside a Class (not counting import directives)
- Clear use of
 - Variables
 - Methods
- Re-use through “packages”
- Modularity, Encapsulation, Inheritance, Polymorphism etc

OOP Vocabulary Review

- **Classes**
 - Definition or a blueprint of a userdefined datatype
 - Prototypes for objects
- **Objects**
 - Nouns, things in the world
- **Constructor**
 - Given a Class, the way to create an Object (that is, an Instance of the Class) and initialize it
- **Attributes**
 - Properties an object has
- **Methods**
 - Actions that an object can do



Anything we can put a thumb on



Defining Classes



The Structure of Classes

class *name* {

declarations

constructor definitions

method definitions

}

← instance variables
and symbolic constants

← how to create and
initialize objects

← how to manipulate those
objects (may or may not
include its own “driver”,
i.e., `main( )`)

These parts of a class can
actually be in any order

Defining a Class

Comparison with C++

- Java gives you the ability to write classes or user-defined data types similar to the way C++ does, with a few differences
- Points to consider when defining a class
 - There are no global variables or functions. Everything resides inside a class. Remember we wrote our main method inside a class
 - Specify access modifiers (public, private or protected) for each member method or data members at every line.
 - No semicolon (;) at the end of class
 - All methods (functions) are written inline. There are no separate header and implementation files.

The Point Class

```
class Point {
```

```
    private int x;
```

```
    private int y;
```

```
    public Point (.....) {.....}
```

```
    public void Display (.....) {
```

```
        .....
```

```
    }
```

```
}
```

← instance variables
and symbolic constants

← how to create and
initialize objects

← how to manipulate those
objects (may or may not
include its own “driver”,
i.e., `main( )`)

Defining a Class

Comparison with C++ (cont)

- Points to consider when defining a class (cont)
 - Automatic initialization of class level data members if you do not initialize them
 - Primitive types
 - Numeric (int, float etc) with zero
 - Char with null
 - Boolean with false
 - Object References
 - With null
 - Remember, the same rule is not applied to local variables. Using a local variable without initialization is a compile time error.

```
public void someMethod () {  
    int x;    //local variable  
    System.out.println(x); //compile time error  
}
```

Defining a Class

Comparison with C++ (cont)

- Points to consider when defining a class (cont)
 - Access Modifiers
 - public : Accessible anywhere by anyone
 - Private : Only accessible within this class
 - Protected : Accessible only to the class itself and to its subclasses or other classes in the same “package”
 - Package : **Default access** if no access modifier is provided.
Accessible to all classes in the same package
 - Constructor
 - Same name as class name
 - Does not have a return type
 - No initialization list
 - JVM provides a **zero-argument** constructor only *if a class doesn't define its own constructor*
 - Destructor
 - Destructors are not required in a java class

Example

Task - Defining a Class

- Create a class for Student
 - should be able to store the following characteristics of student
 - Roll No
 - Name
 - Provide default, parameterized and copy constructors
 - Provide standard getters/setters for instance variables
 - Make sure, roll no has never assigned a negative value i.e. ensuring the correct state of the object
 - Provide print method capable of printing student object on console

Student

Attributes:

Roll NO

Name

Methods:

constructors

getters/setters

print

Student Implementation Code

```
// Student.java
```

```
/*
```

Demonstrates the most basic features of a class. A student is defined by their name and rollNo. There are standard get/set accessors for name and rollNo.

NOTE A well documented class should include an introductory comment like this. Don't get into all the details – just introduce the landscape.

```
*/
```

```
public class Student {  
    private String name;  
    private int rollNo;
```

Student Implementation Code cont.

// Standard Setters

```
public void setName (String name) {  
    this.name = name;  
}
```

// Note the masking of class level variable rollNo

```
public void setRollNo (int rollNo) {  
    if (rollNo > 0) {  
        this.rollNo = rollNo;  
    }else {  
        this.rollNo = 100;  
    }  
}
```


Student Implementation Code cont.

// Standard Getters

```
public String getName ( ) {  
    return name;  
}
```

```
public int getRollNo ( ) {  
    return rollNo;  
}
```

Student Implementation Code cont.

// Constructor that uses a default value instead of taking an argument.

```
public Student() {  
    name = "not set";  
    rollNo = 100;  
}
```

// parameterized Constructor for a new student

```
public Student(String name, int rollNo) {  
    setName(name);           //call to setter of name  
    setRollNo(rollNo);       //call to setter of rollNo  
}
```

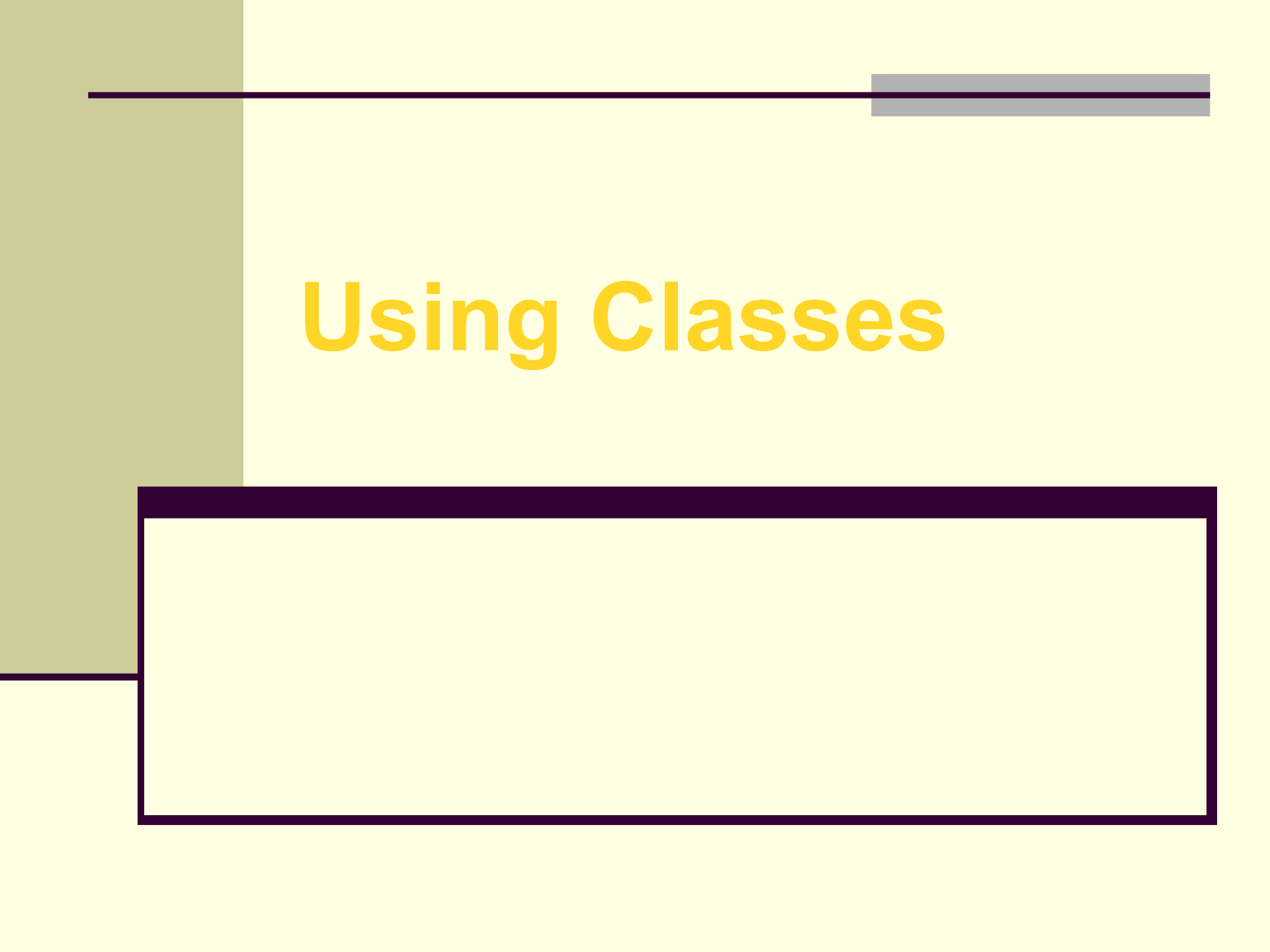
// Copy Constructor for a new student

```
public Student(Student s) {  
    name = s.name;  
    rollNo = s.rollNo;  
}
```

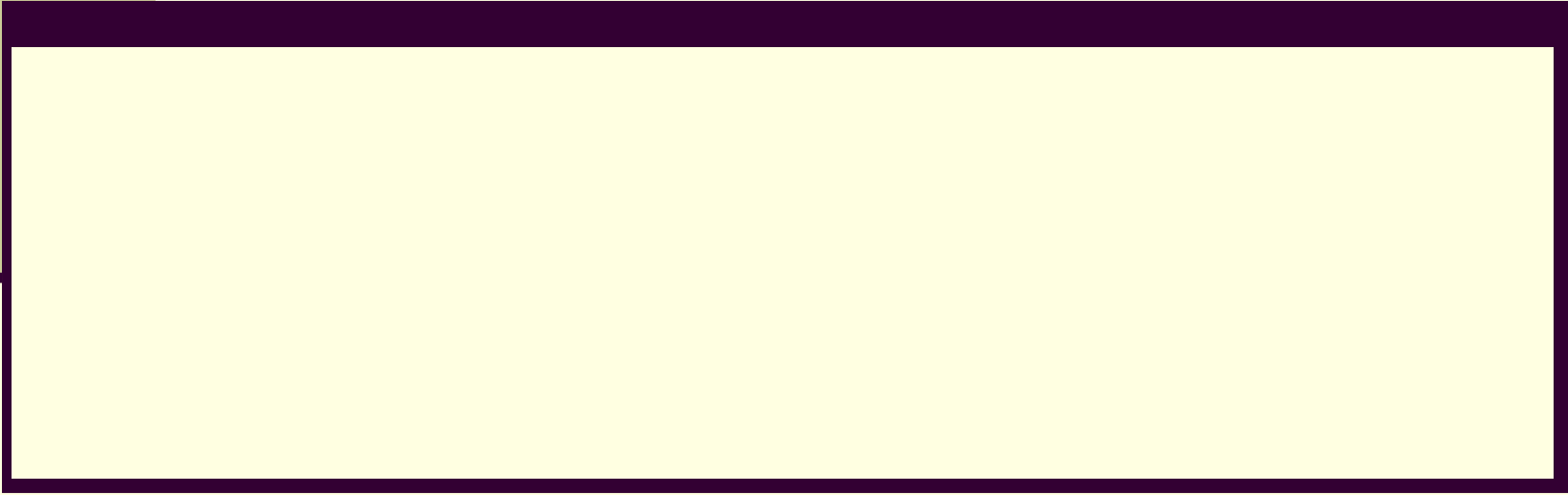
Student Implementation Code cont.

// method used to display method on console

```
public void print () {  
    System.out.println("Student name:" +name+ ", roll no:" +rollNo);  
}  
  
} // end of class
```



Using Classes



Using a Class

Comparison with C++

- Objects of a class are always created on heap using the “new” operator followed by constructor
 - `Student s = new Student ()` // no pointer operator “*” between
// Student and s
 - Only String constant is an exception
 - `String greet = “Hello” ;` // No new operator
 - However you can use
 - `String greet2 = new String(“Hello”);`
- Members of a class (member variables and methods also known as instance variables/methods) are accessed using “.” operator. There is no “→” operator in java
 - `s.setName(“Ali”);`
 - `S→setName(“Ali”)` is incorrect and will not compile in java

Using a class

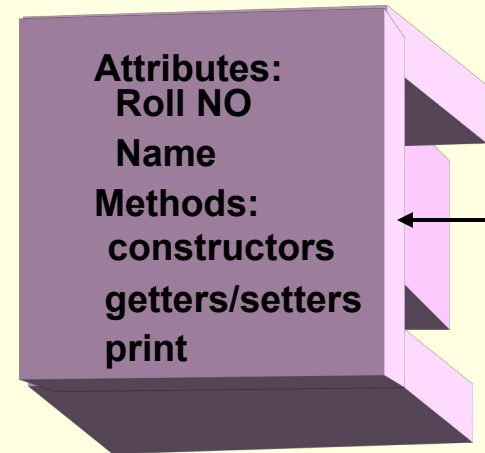
Comparison with C++

- Differences from C++ (cont)
 - Objects are always passed by reference whereas primitive data types are passed by value.
 - All methods use the run-time, not compile-time, types (i.e. all Java methods are like C++ virtual functions)
 - The types of all objects are known at run-time
 - All objects are allocated on the heap (always safe to return objects from methods)

Task - Using Student Class

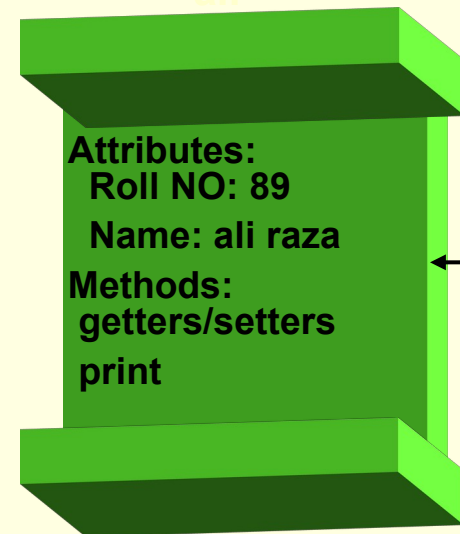
- Create objects of Student class by calling default, parameterized and copy constructors.
- Call Students class various methods on objects

Student



class

ali



object

Student Client Code

```
public class Test{

    public static void main (String args[]){
        // Make two students
        Student s1 = new Student("ali", 15);
        Student s2 = new Student();           //call to default constructor
        s1.print();
        s2.print();
        s2.setName("usman");
        s2.setRollNo(20);

        System.out.print("Student name:" + s2.getName());
        System.out.println(" rollNo:" + s2.getRollNo());

        //continue....
    }
}
```


Student Client Code

```
System.out.println("calling copy constructor");  
Student s3 = new Student(s2); //call to copy constructor
```

```
s2.print();  
s3.print();
```

```
s3.setRollNo(-10); //Roll No would be set to 100
```

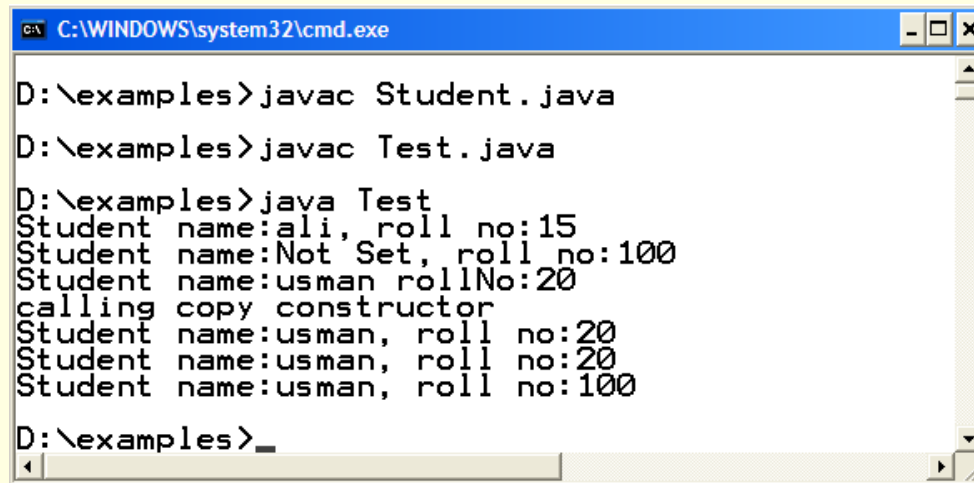
```
s3.print();
```

/*NOTE: public vs. private

A statement like "b.rollNo = 10;" will not compile in a client of the Student class when rollNo is declared protected or private */

```
} //end of main  
} //end of class
```

Compile and Execute

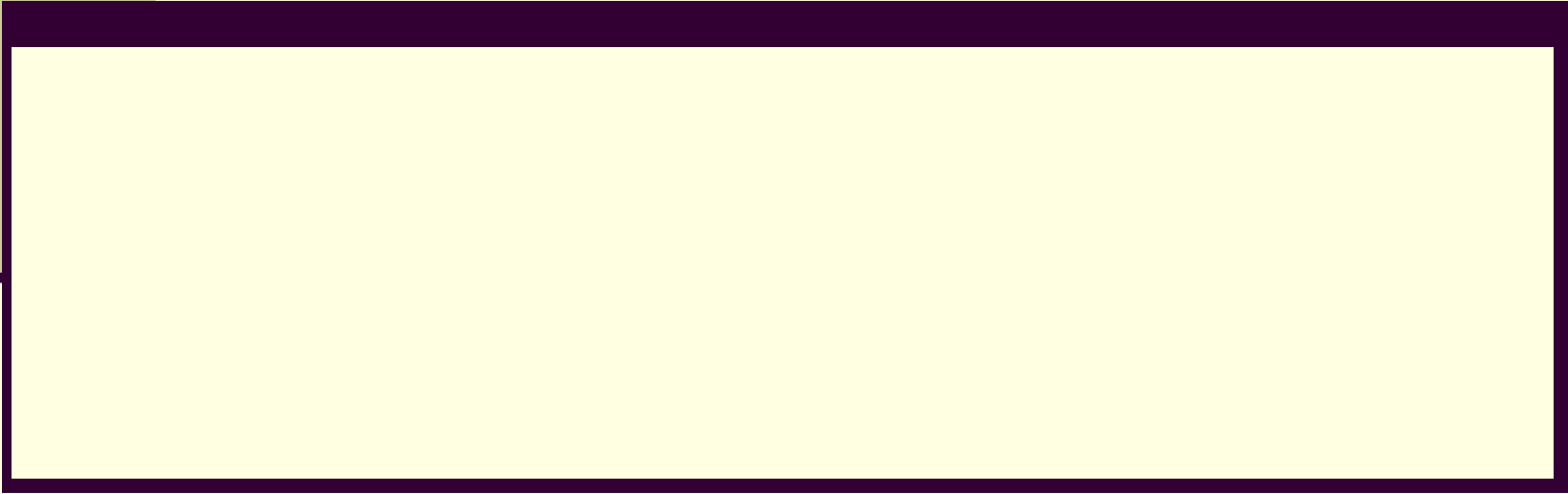


```
C:\WINDOWS\system32\cmd.exe

D:\examples>javac Student.java
D:\examples>javac Test.java
D:\examples>java Test
Student name:ali, roll no:15
Student name:Not Set, roll no:100
Student name:usman rollNo:20
calling copy constructor
Student name:usman, roll no:20
Student name:usman, roll no:20
Student name:usman, roll no:100
D:\examples>_
```



More on Classes



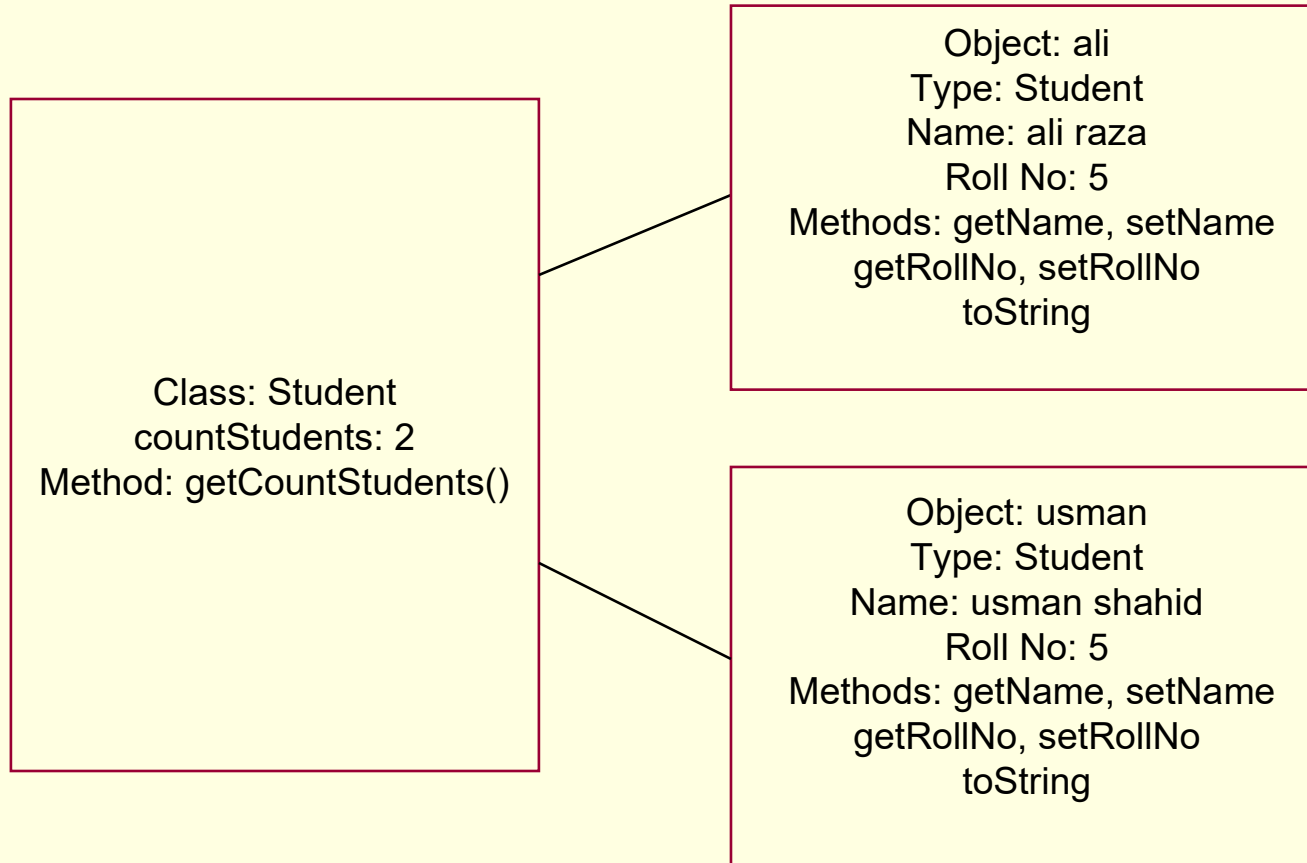
Static

- A class can have *static*
 - Variables
 - Methods
- Static variables and methods
 - Are associated with the class itself!!
 - Not associated with the object
- Therefore Statics can be accessed without instantiating an object!
- Generally accessed by class name
- Cannot refer to a non-static instance variable in a static method
 - No this reference

Static Variable & Methods

- Occurs as a **single copy** in the class
- For example;
 - `System.out` is a static variable
 - `JOptionPane.showInputDialog(String)`

Static Fun





Garbage Collection



Garbage collection and finalize

- Java performs garbage collection and eliminates the need to free objects explicitly.
- When an object has no references to it anywhere, except in other objects that are also unreferenced, its space can be reclaimed.
- Before the object is destroyed, it might be necessary for the object to perform some actions.
 - For example closing an open file. In such a case define a `finalize()` method with the actions to be performed before the object is destroyed.

finalize

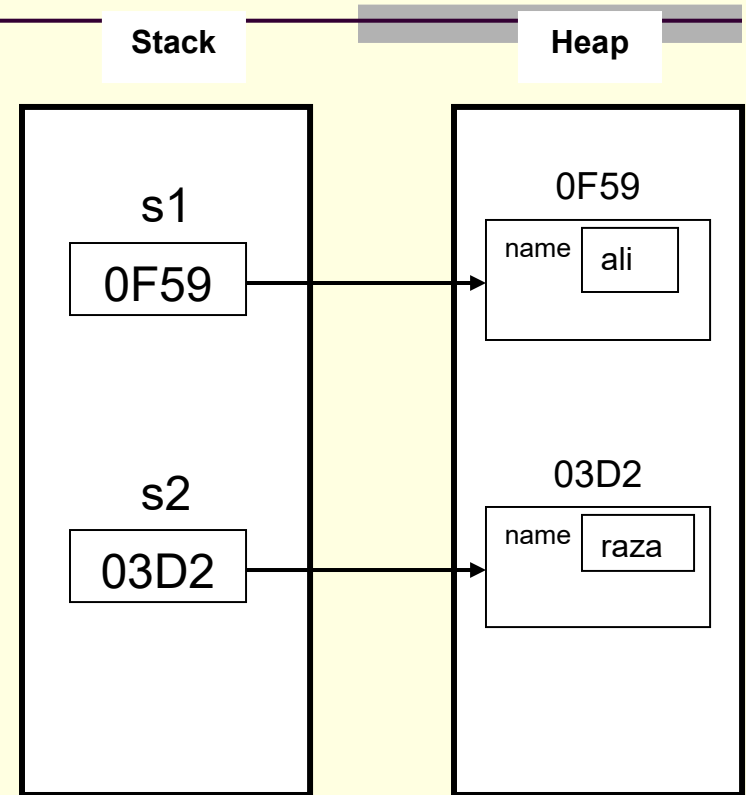
- When a finalize method is defined in a class, Java run time calls `finalize()` whenever it is about to recycle an object of that class.

```
protected void finalize()  
{  
    // code  
}
```

- A garbage collector reclaims objects in any order or never reclaim them.
- `System.gc()`
 - Request the JVM to run the garbage collector
 - Not necessary it will run

Memory Mangement

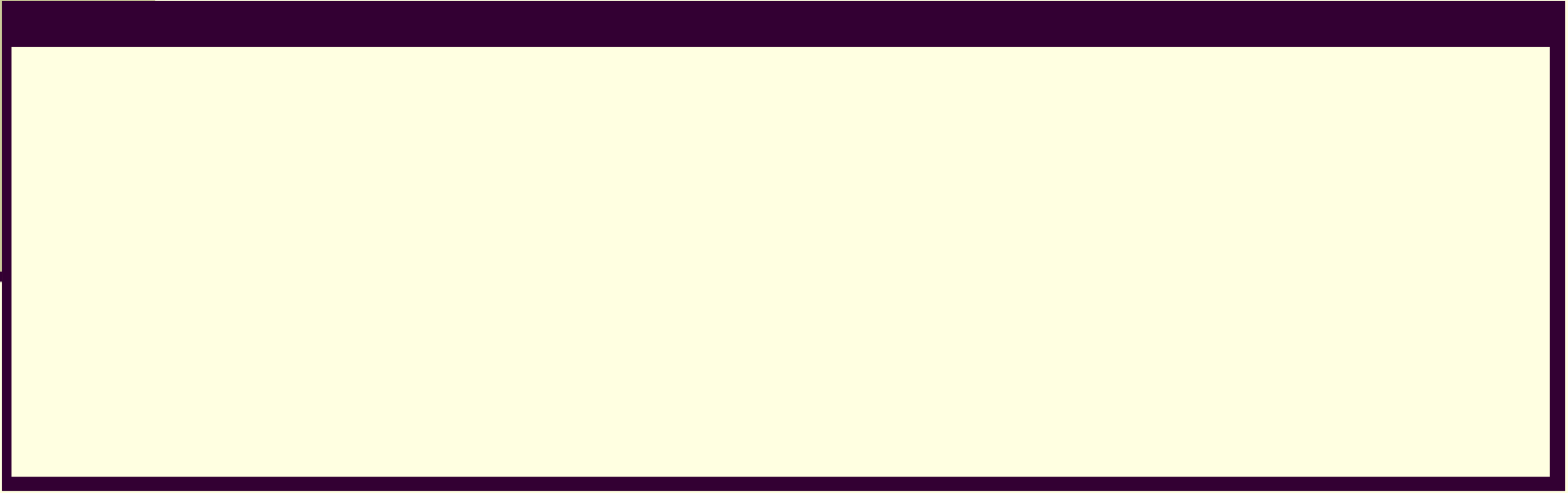
```
public class Test{  
  
    public static void main|(String args[]){  
  
        Student s1 = new Student("ali");  
        Student s2 = new Student("raza");  
  
        s1= s2;  
  
    }  
  
}
```



No Memory leakage in Java, Automatic Garbage Collection will take care of such scenarios



Example



Modify Student Class

```
public class Student {  
    ....  
    private static int countStudents = 0;  
  
    public static int getCountStudents() {  
        return countStudents;  
    }  
  
    .....  
}
```

Modify Student Class

// Constructor that uses a default value instead of taking an argument.

```
public Student() {  
    name = "not set";  
    rollNo = 100;  
    countStudents += 1;  
}
```

// parameterized Constructor for a new student

```
public Student(String name, int rollNo) {  
    setName(name);           //call to setter of name  
    setRollNo(rollNo);       //call to setter of rollNo  
    countStudents += 1;  
}
```

// Copy Constructor for a new student

```
public Student(Student s) {  
    name = s.name;  
    rollNo = s.rollNo;  
    countStudents += 1;  
}
```

Modify Student Class

// Overridden methods

// Overriding toString method of class java.lang.Object

//Overriding finalize method of Object class

} // end of class

Student Client Code

```
public class Test{

    public static void main (String args[]){

        int numObjs;

        numObjs = Student.getCountStudents();
        System.out.println("Students Objects:"+numObjs);

        Student s1 = new Student("ali", 15);
        System.out.println("Student:" + s1.toString() );

        numObjs = Student.getCountStudents();
        System.out.println("Students Objects:"+numObjs);
```

Student Client Code

```
Student s2 = new Student("usman", 49);
System.out.println("Student:" +s2);    //implicit call to toString()

numObjs = Student.getCountStudents();
System.out.println("Students Objects:"+numObjs);

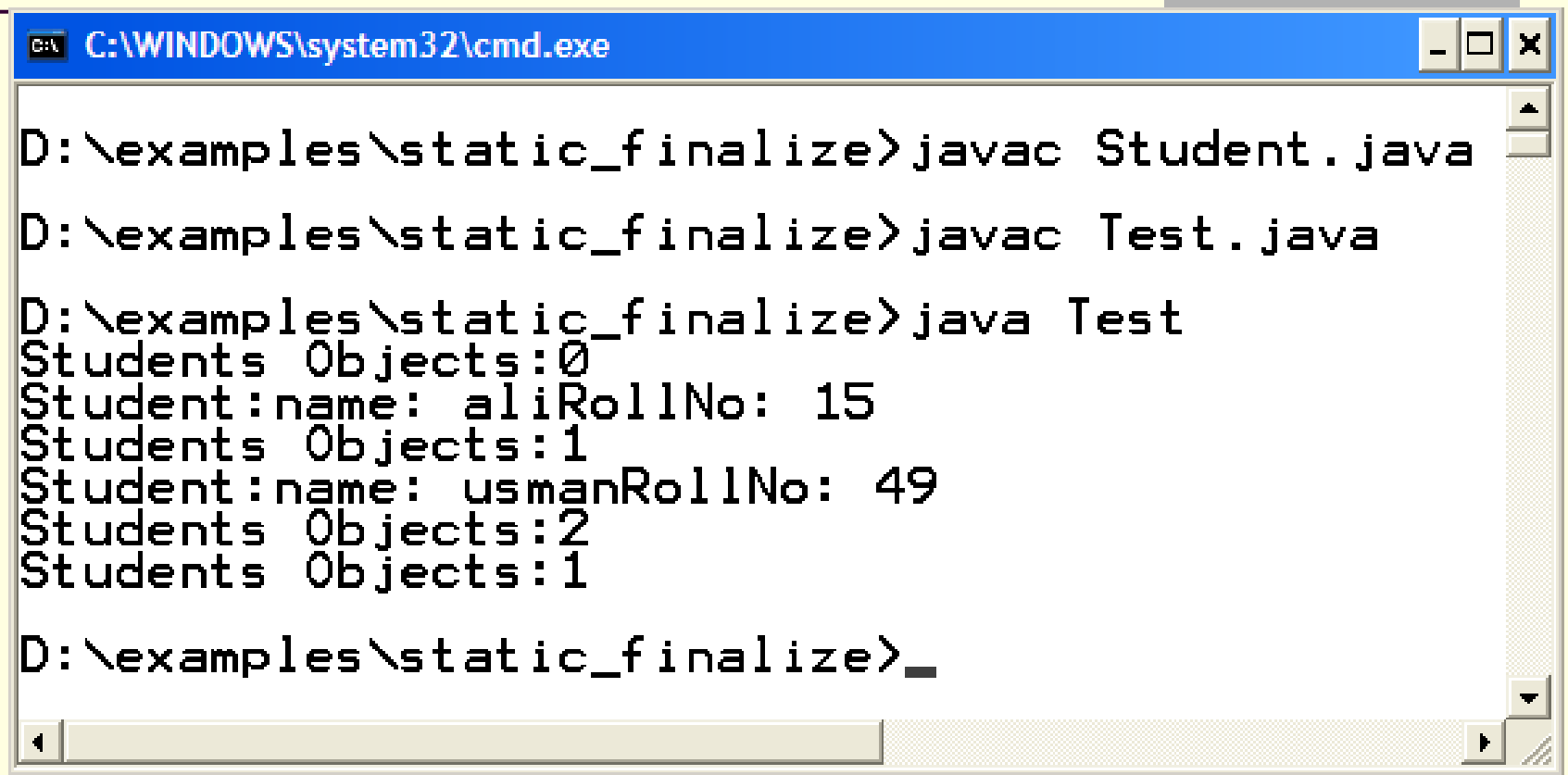
s1 = null;

System.gc(); // request the JVM to run the garbage collector But
              // there is no gaurantee that garbage collector will run

numObjs = Student.getCountStudents();
System.out.println("Students Objects:"+numObjs);

} //end of main
} //end of class
```


Compile and Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\static_finalize>javac Student.java
D:\examples\static_finalize>javac Test.java
D:\examples\static_finalize>java Test
Students Objects:0
Student:name: aliRollNo: 15
Students Objects:1
Student:name: usmanRollNo: 49
Students Objects:2
Students Objects:1

D:\examples\static_finalize>_
```

The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The window contains the following text:

```
D:\examples\static_finalize>javac Student.java
D:\examples\static_finalize>javac Test.java
D:\examples\static_finalize>java Test
Students Objects:0
Student:name: aliRollNo: 15
Students Objects:1
Student:name: usmanRollNo: 49
Students Objects:2
Students Objects:1

D:\examples\static_finalize>_
```

The output of the Java program shows the number of objects created and the details of the students created. The first student is "ali" with roll number 15, and the second student is "usman" with roll number 49. The number of objects increases from 0 to 2, and then decreases to 1, indicating that the first student object was garbage collected.



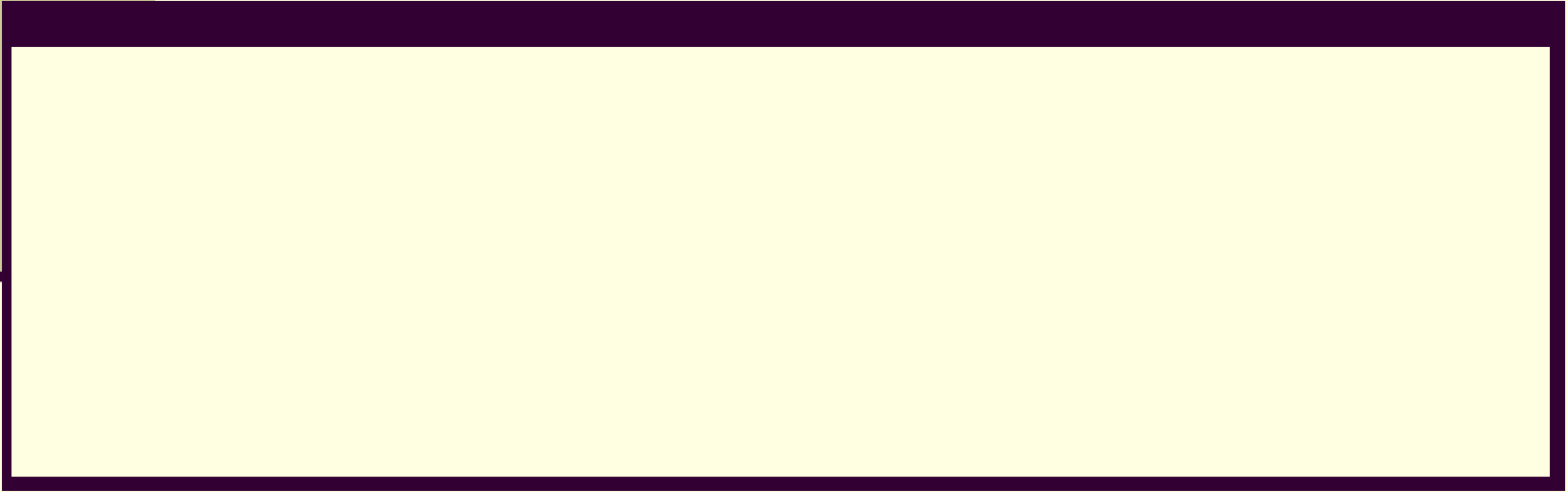
Web Design & Development

Lecture 5





Common Errors

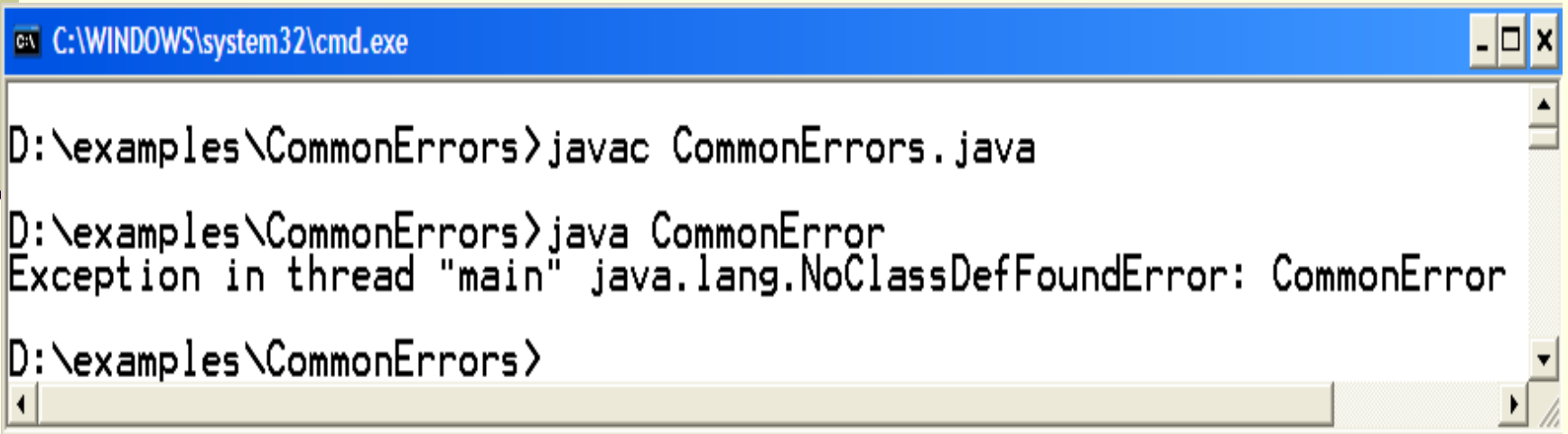


Common Errors

- `NoClassDefFoundError`
- `NoSuchMethodError: main`
- Cannot resolve symbol
- Local variables must be initialized
- `ArrayIndexOutOfBoundsException` & `NumberFormatException` exception

Error: Exception in thread “main” NoClassDefFoundError

```
public class CommonErrors{  
    public static void main (String args[]) {  
        System.out.println("Hello World");  
    }  
}
```

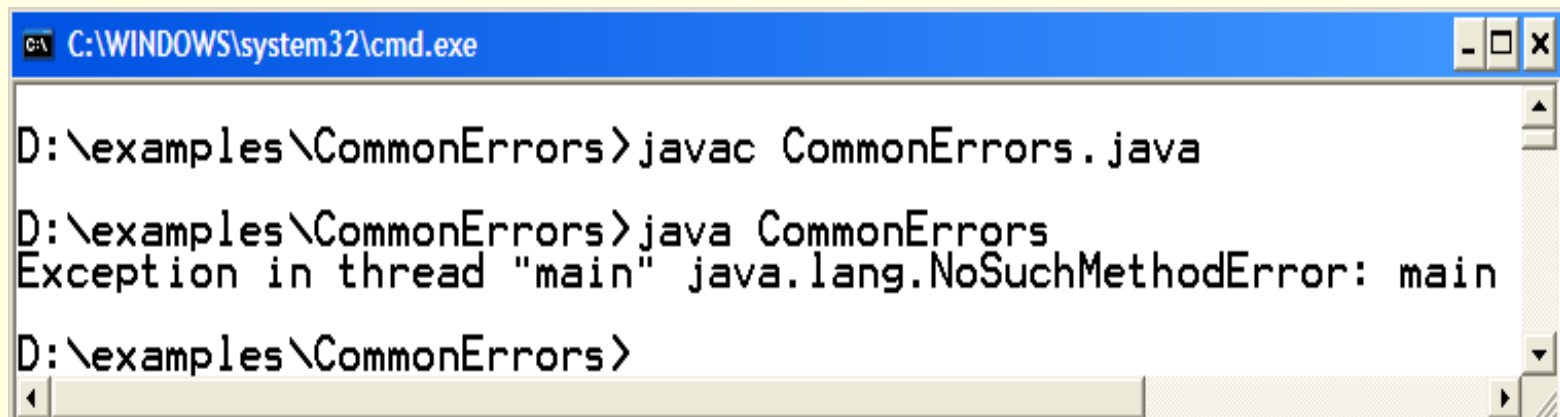


A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" and standard window control buttons. The command prompt shows the following sequence of commands and output:

```
D:\examples\CommonErrors>javac CommonErrors.java  
D:\examples\CommonErrors>java CommonError  
Exception in thread "main" java.lang.NoClassDefFoundError: CommonError  
D:\examples\CommonErrors>
```

Error: Exception in thread “main” NoSuchMethodError: main

```
public class CommonErrors{  
    public static void display () {  
        System.out.println("hello world");  
    }  
  
    public static void main (String args[]) {  
        CommonErrors.display();  
    }  
}
```



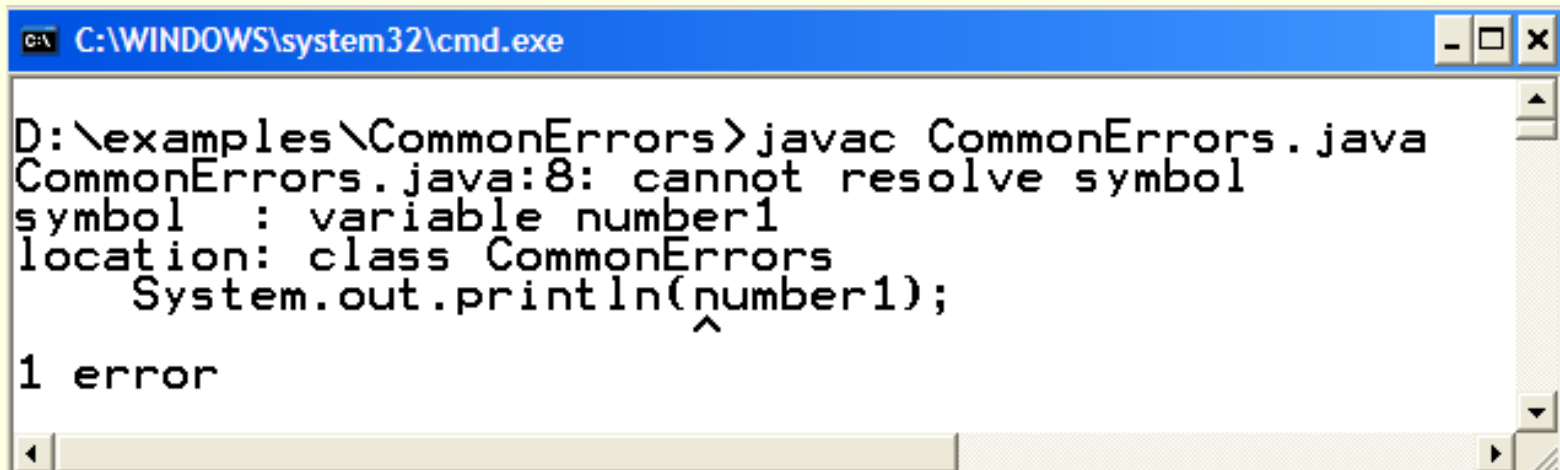
The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The command history is as follows:

```
D:\examples\CommonErrors>javac CommonErrors.java  
D:\examples\CommonErrors>java CommonErrors  
Exception in thread "main" java.lang.NoSuchMethodError: main  
D:\examples\CommonErrors>
```

The error message "Exception in thread 'main' java.lang.NoSuchMethodError: main" is displayed, indicating that the JVM cannot find the main method in the class.

Error: cannot resolve symbol

```
public class CommonErrors{  
    public static void main (String args[]) {  
        int number;  
        System.out.println( number);  
    }  
}
```



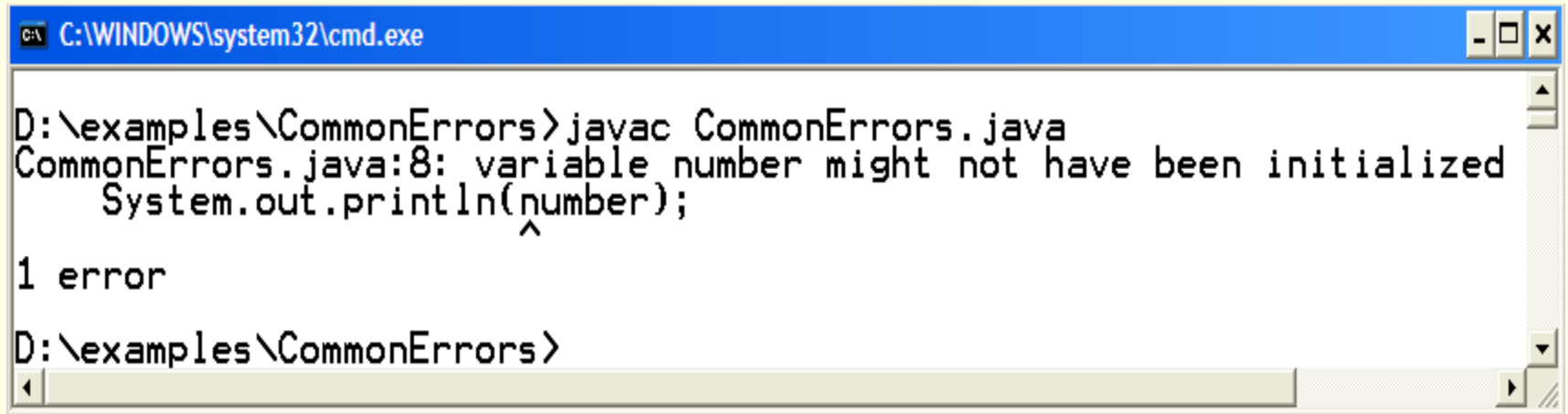
A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" along with standard window control buttons. The command prompt shows the following text:

```
D:\examples\CommonErrors>javac CommonErrors.java  
CommonErrors.java:8: cannot resolve symbol  
symbol  : variable number1  
location: class CommonErrors  
    System.out.println(number1);  
                        ^  
1 error
```

The error message indicates that the compiler cannot find the variable `number1` in the class `CommonErrors`. The variable `number` is declared in the code above, but `number1` is used in the `println` statement.

Error: Local variable not initialized

```
public class CommonErrors{  
    public static void main (String args[]) {  
        int number = 10 ;  
        System.out.println(number);  
    }  
}
```

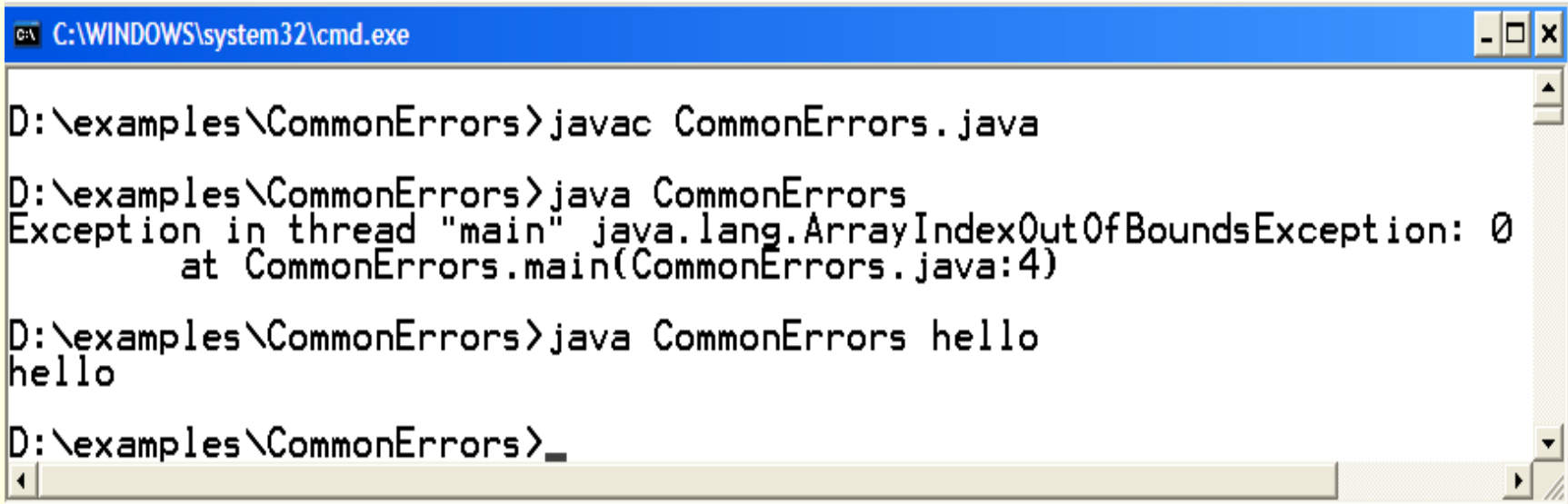


A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The window shows the output of a Java compilation command. The prompt is "D:\examples\CommonErrors>". The command entered is "javac CommonErrors.java". The output is "CommonErrors.java:8: variable number might not have been initialized" followed by a line of code "System.out.println(number);". An arrow points to the variable "number" in the code. Below the error message, it says "1 error". The prompt is "D:\examples\CommonErrors>".

```
C:\WINDOWS\system32\cmd.exe  
D:\examples\CommonErrors>javac CommonErrors.java  
CommonErrors.java:8: variable number might not have been initialized  
    System.out.println(number);  
                        ^  
1 error  
D:\examples\CommonErrors>
```


Error: ArrayIndexOutOfBoundsException

```
public class CommonErrors{  
    public static void main (String args[]) {  
        System.out.println(args[0]);  
    }  
}
```

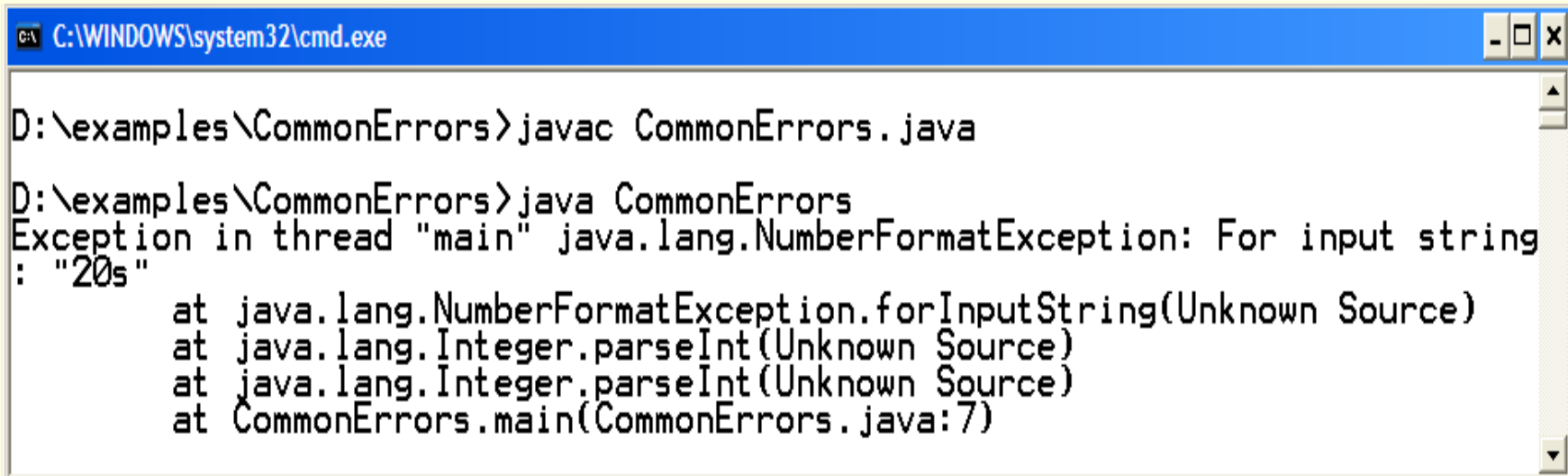


The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt is at "D:\examples\CommonErrors>". The user enters "javac CommonErrors.java", which compiles successfully. Then, the user enters "java CommonErrors", which results in an "Exception in thread 'main' java.lang.ArrayIndexOutOfBoundsException: 0 at CommonErrors.main(CommonErrors.java:4)". Finally, the user enters "java CommonErrors hello", which outputs "hello". The prompt is currently at "D:\examples\CommonErrors>_".

```
C:\WINDOWS\system32\cmd.exe  
D:\examples\CommonErrors>javac CommonErrors.java  
D:\examples\CommonErrors>java CommonErrors  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at CommonErrors.main(CommonErrors.java:4)  
D:\examples\CommonErrors>java CommonErrors hello  
hello  
D:\examples\CommonErrors>_
```

Error: NumberFormatException

```
public class CommonErrors{  
    public static void main (String args[]) {  
        String s = "20" ;  
        int number = Integer.parseInt(s);  
    }  
}
```

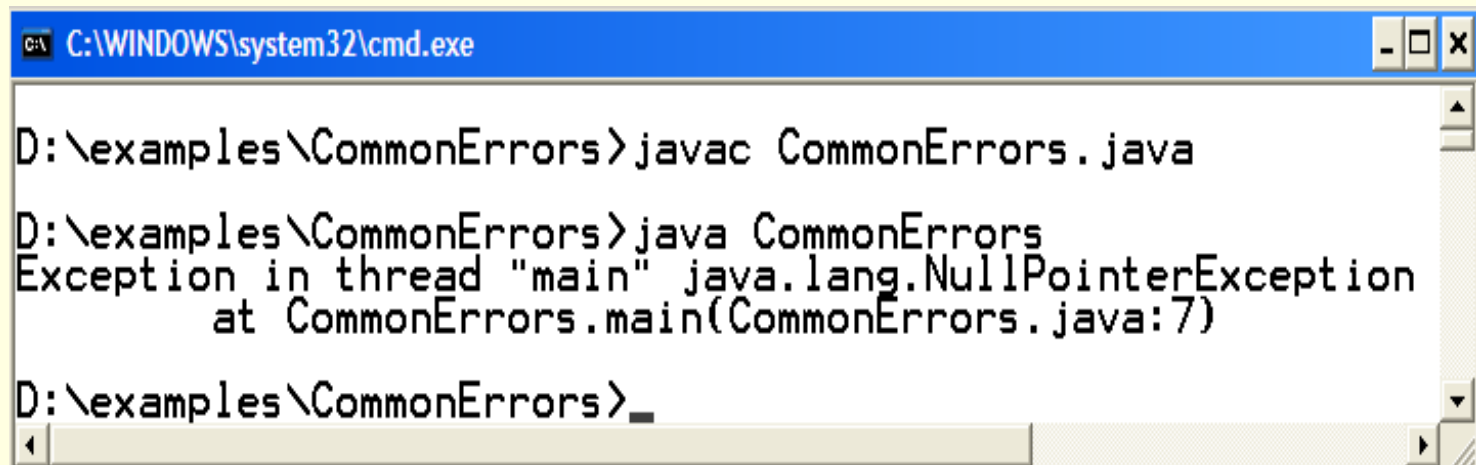


The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The command prompt displays the following text:

```
D:\examples\CommonErrors>javac CommonErrors.java  
D:\examples\CommonErrors>java CommonErrors  
Exception in thread "main" java.lang.NumberFormatException: For input string  
: "20s"  
    at java.lang.NumberFormatException.forInputString(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at CommonErrors.main(CommonErrors.java:7)
```

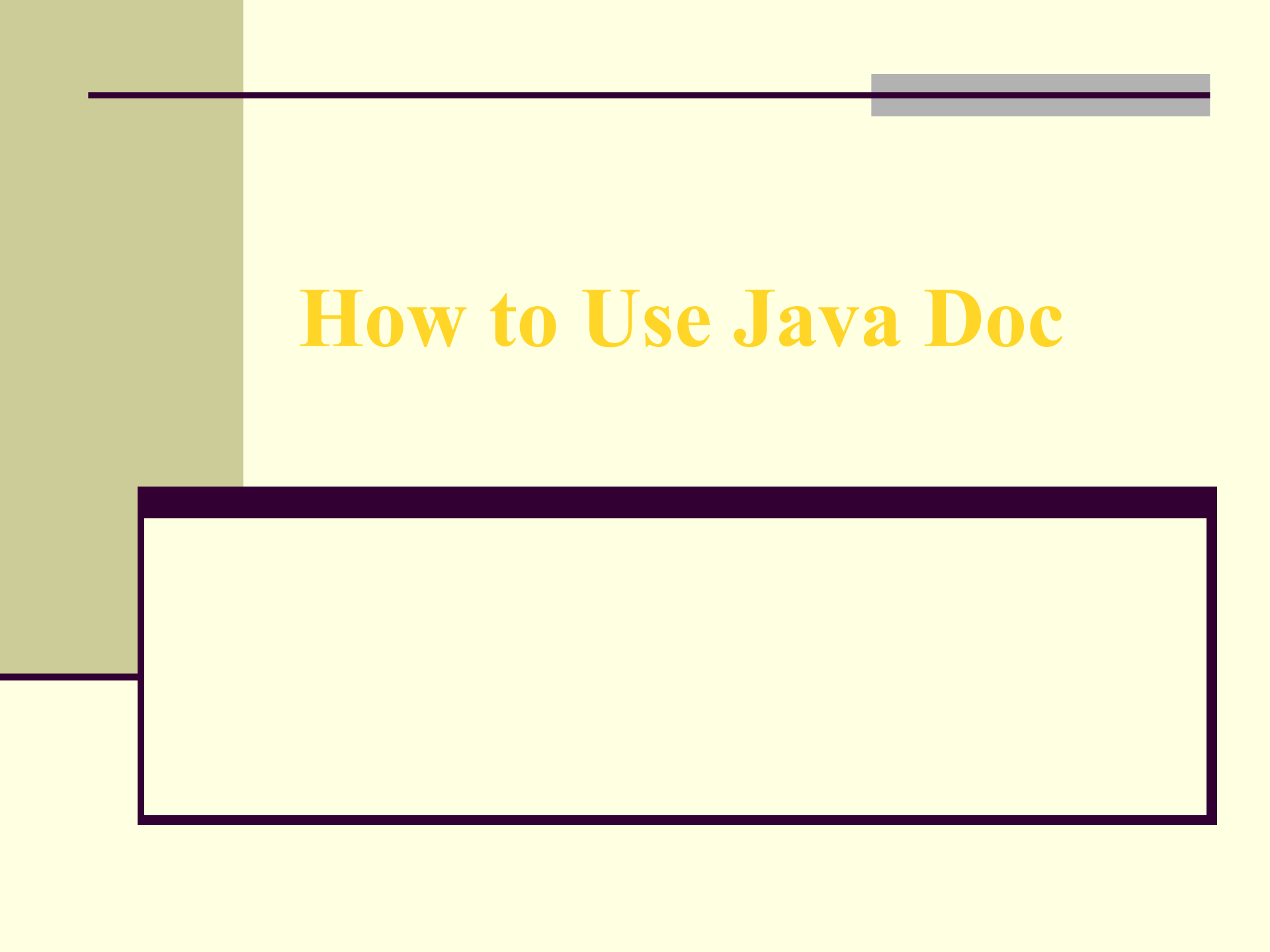
Error: NullPointerException

```
public class CommonErrors{  
    public static void main (String args[ ]) {  
        Student s = null;  
        s.print( );  
    }  
}
```



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" and standard window control buttons. The command prompt shows the following sequence of commands and output:

```
D:\examples\CommonErrors>javac CommonErrors.java  
D:\examples\CommonErrors>java CommonErrors  
Exception in thread "main" java.lang.NullPointerException  
    at CommonErrors.main(CommonErrors.java:7)  
D:\examples\CommonErrors>_
```

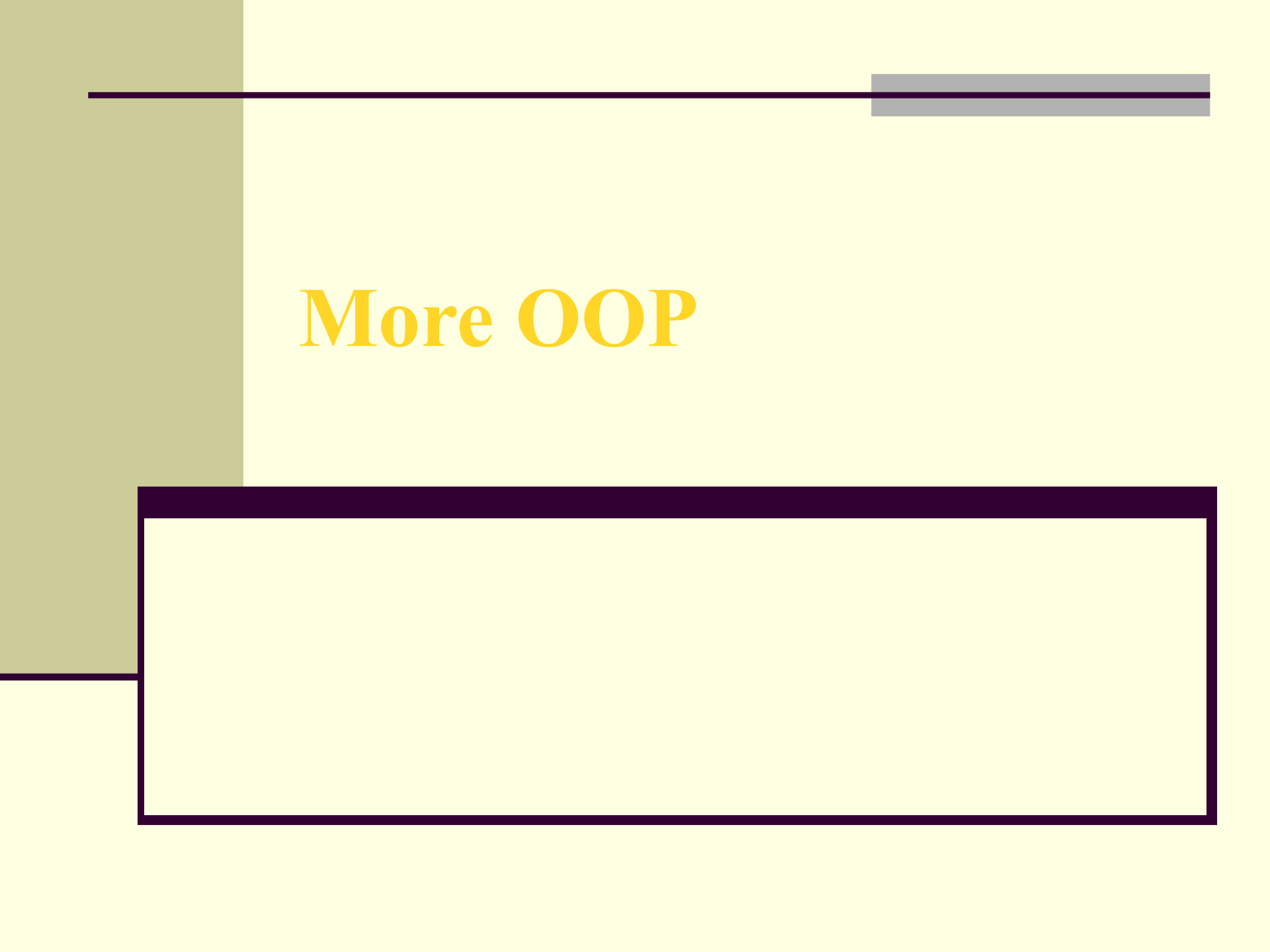


How to Use Java Doc

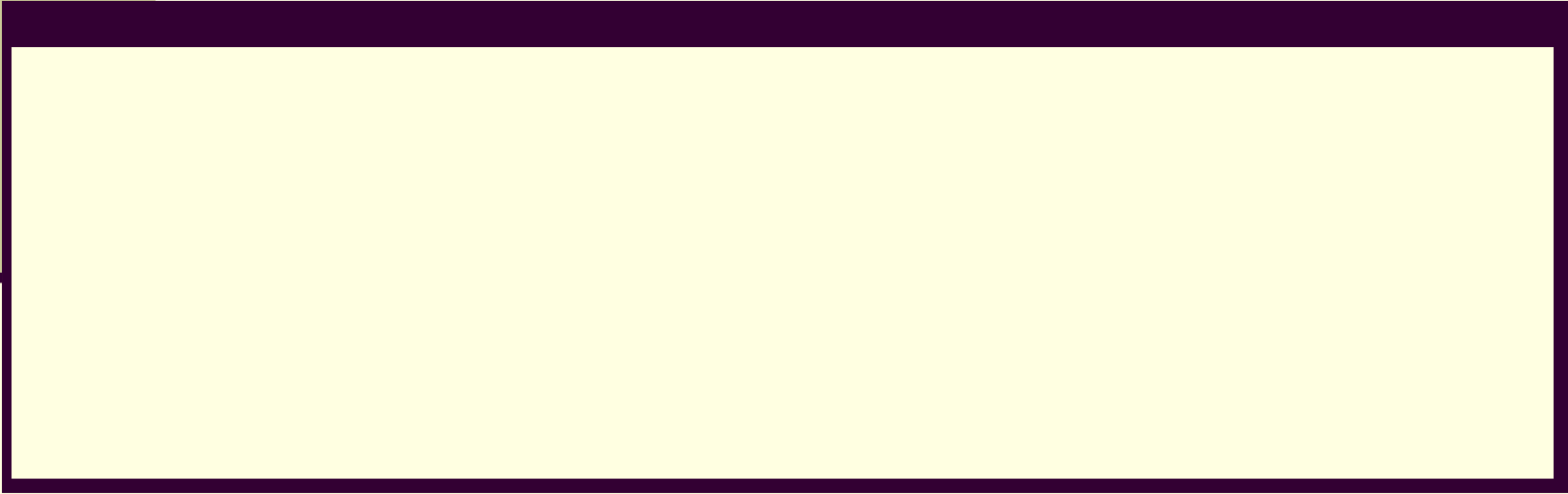


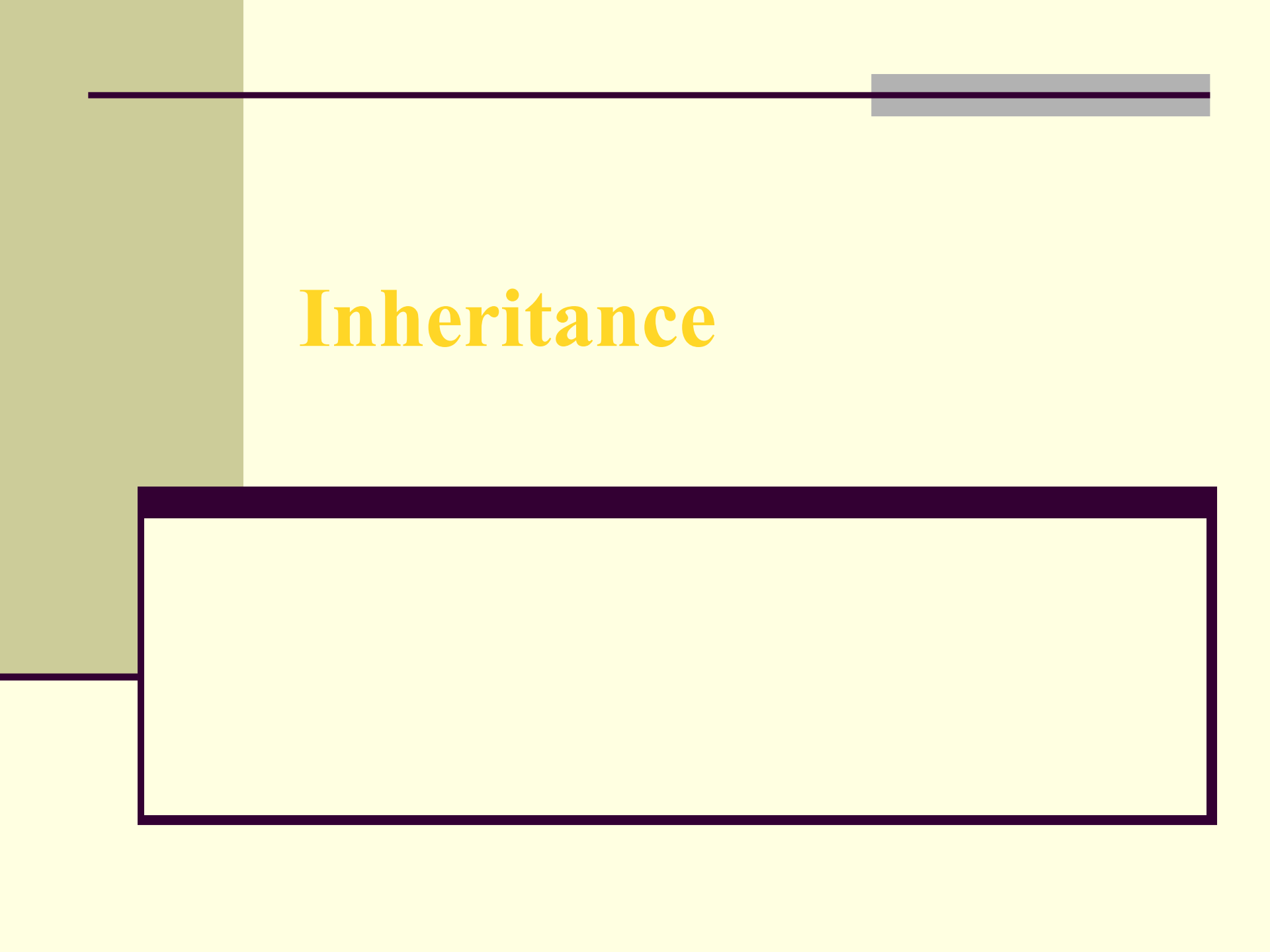
How to use Java Doc

```
public class DocDemo{  
    public static void main (String args[]) {  
        String s = "Hello World";  
  
        // I want to convert this string into Upper case?  
        String upper = s.toUpperCase(); //upper contains "HELLO WORLD"  
        System.out.println(upper); //will print HELLO WORLD  
    }  
}
```

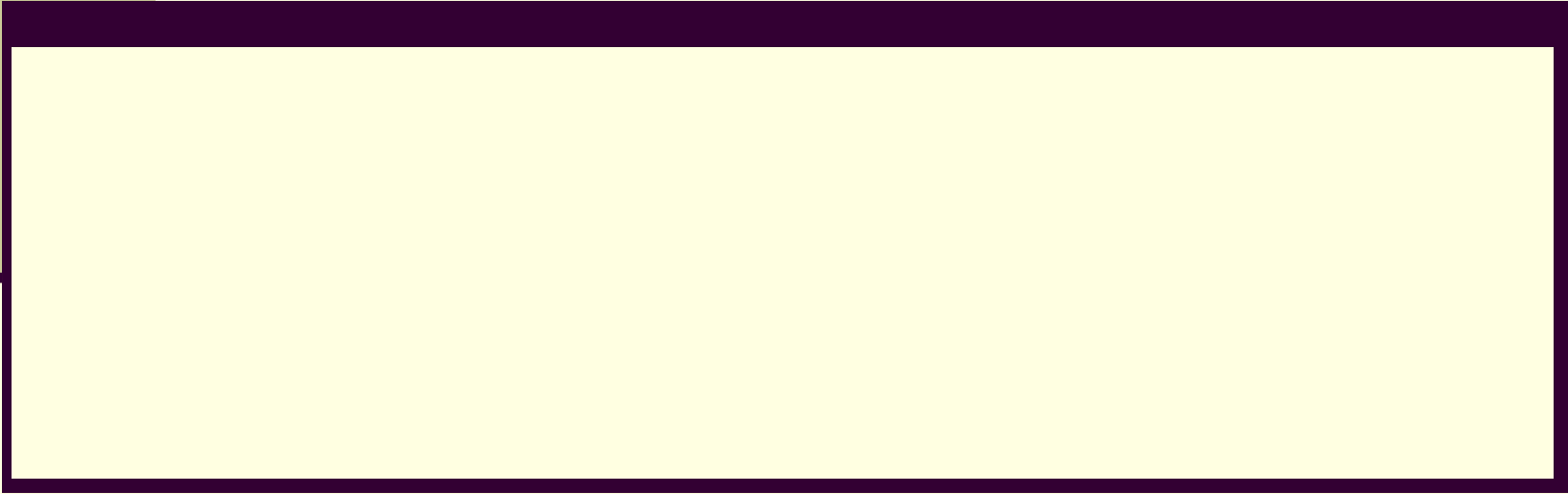


More OOP

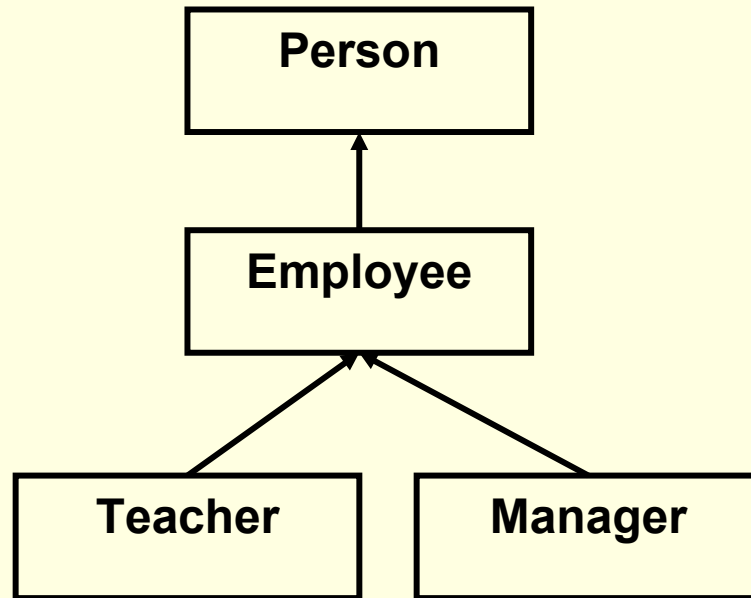




Inheritance



Employee class hierarchy



Employee. java (super class)

```
public class Employee{
    protected int id;
    protected String name;

    //parameterized constructor
    public Employee(int id, String name){

        this.id = id;
        this.name = name;
    }

    //default constructor
    public Employee(){
        this (10, "not set");
    }

    //continue
```

Employee.java (super class)

```
public void setId (int id) {  
    this.id = id;  
}  
  
public int getId () {  
    return id;  
}  
  
public void setName (String name) {  
    this.name = name;  
}  
  
public String getName () {  
    return name;  
}  
  
//continue ....
```

Employee. java (super class)

```
public void display(){
    System.out.println("in employee display method");
    System.out.println("Employee id:" + id + " name:" + name);
}

//overriding object class toString method
public String toString() {
    System.out.println("in employee toString method");
    return "id:" + id + "name:" + name;
}
}
```

Teacher. java (sub class)

```
public class Teacher extends Employee{

    private String qual;

    //default constructor
    public Teacher () {
        //implicit call to superclass default construct
        qual = "";
    }

    //parameterized constructor
    public Teacher(int i, String n, String q){

        super(i,n); //call to superclass const must be first line
        qual = q;
    }
    //continue
```

Teacher. java (sub class)

```
//accessors for qual
```

```
public void setQual (String qual){  
    this.qual = qual;  
}
```

```
public String getQual(){  
    return qual;  
}
```

```
//continue
```

Teacher. java (sub class)

```
//overriding display method of Employee class
public void display(){
    System.out.println("in teacher's display method");
    super.display(); //call to superclass display method
    System.out.println("Teacher qualification:" + qual);
}
```

```
//overriding toString method of Employee class
public String toString() {
    System.out.println("in teacher's toString method");
    String emp = super.toString();
    return emp + " qualification:" + qual;
}
```

```
// end of class
```

Test. java (Driver class)

```
public class Test{
    public static void main (String args[]){

        System.out.println("making object of employee");
        Employee e = new Employee(89, "khurram ahmad");

        System.out.println("making object of teacher");
        Teacher t = new Teacher (91, "ali raza", "phd");

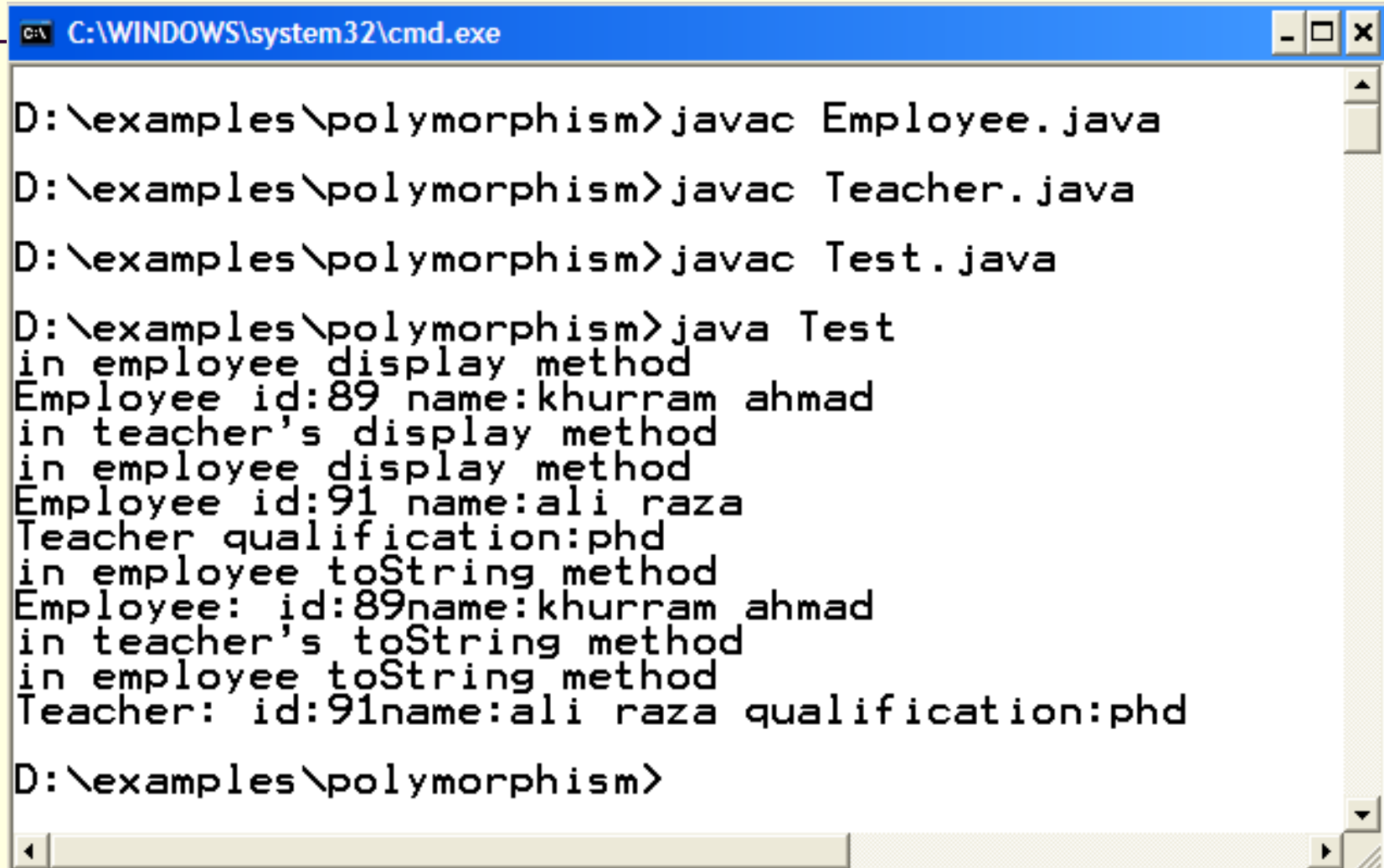
        e.display(); //call to Employee class display method
        t.display(); //call to Teacher class display method

        System.out.println("Employee: " +e.toString());

        System.out.println("Teacher: " + t);

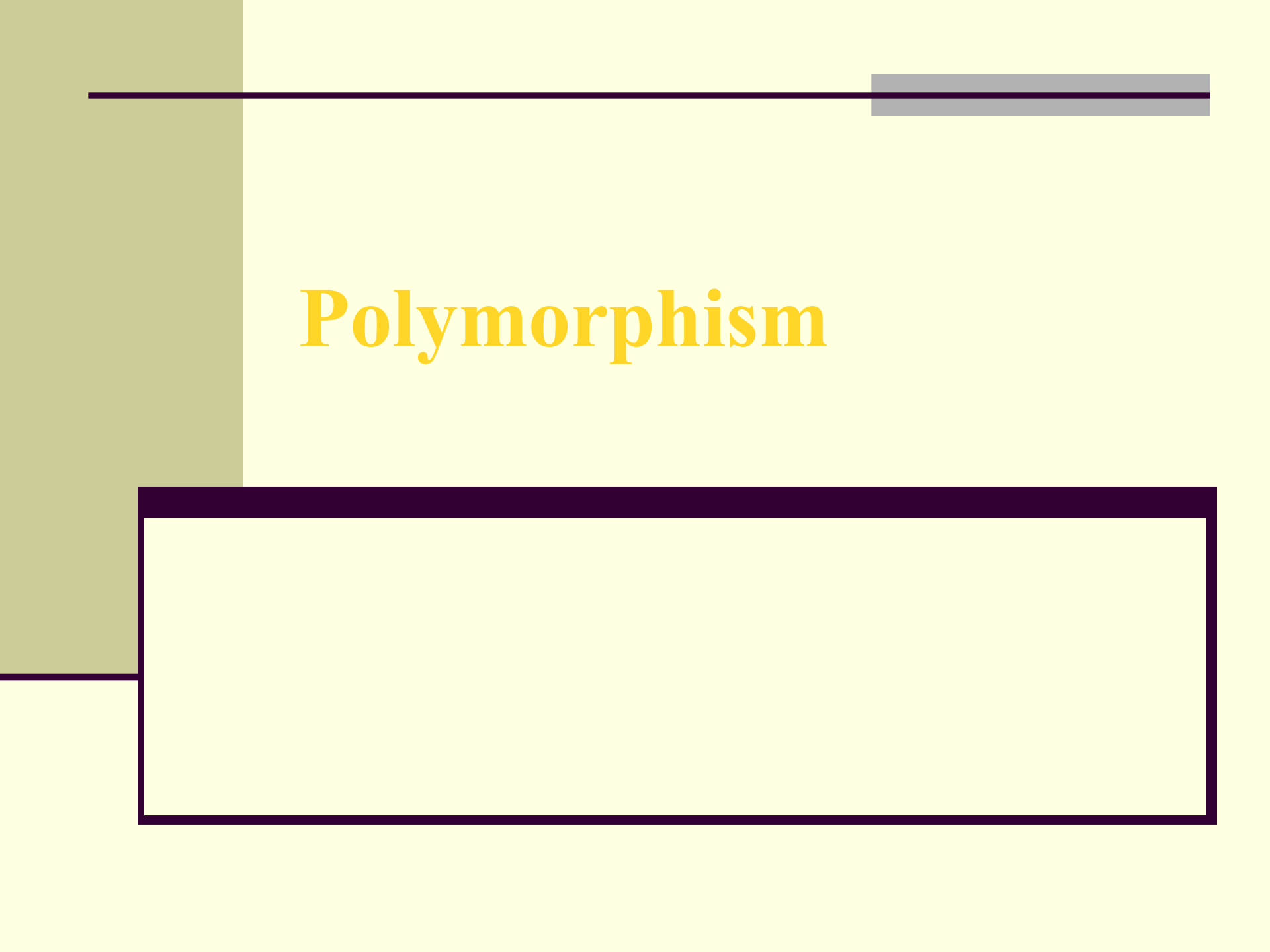
    } //end of main
}
```

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\polymorphism>javac Employee.java
D:\examples\polymorphism>javac Teacher.java
D:\examples\polymorphism>javac Test.java
D:\examples\polymorphism>java Test
in employee display method
Employee id:89 name:khurram ahmad
in teacher's display method
in employee display method
Employee id:91 name:ali raza
Teacher qualification:phd
in employee toString method
Employee: id:89name:khurram ahmad
in teacher's toString method
in employee toString method
Teacher: id:91name:ali raza qualification:phd
D:\examples\polymorphism>
```

Polymorphism

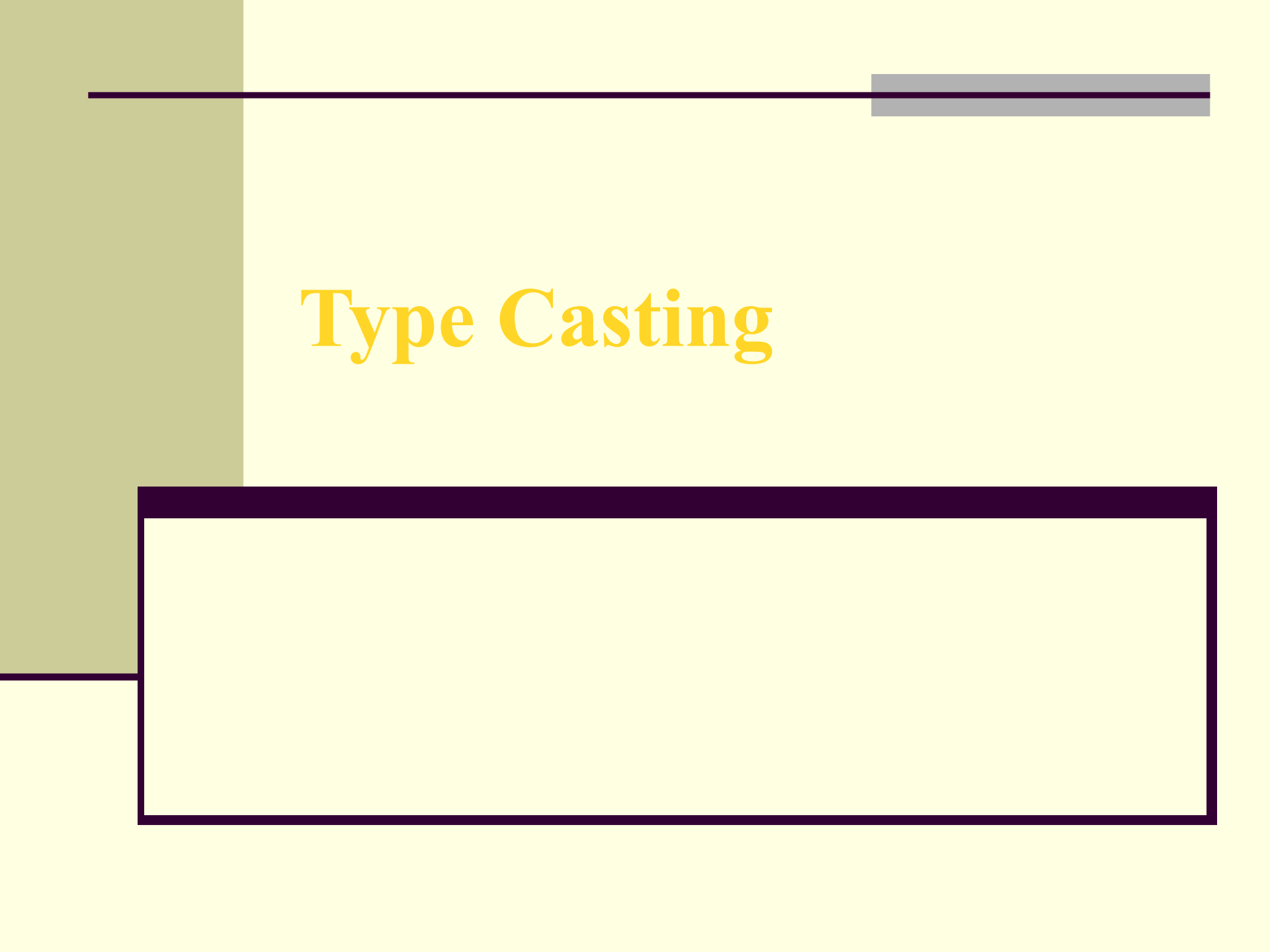
Polymorphism

- “Polymorphic” literally means “of multiple shapes” and in the context of OOP, polymorphic means “having multiple behavior”
- A parent class reference can point to the subclass objects. For example Employee reference
 - Can point to the Employee Object
 - Can point to the Teacher Object
 - Can point to the Manager Object
- A polymorphic method results in different actions depending on the object being referenced
 - Also known as *late binding* or *run-time binding*

Modify Test. java

(of Inheritance Example)

```
public class Test {  
    public static void main (String args[]){  
        // Make employee references  
        Employee ref1, ref2;  
  
        ref1 = new Employee(89, "khurram ahmad");  
  
        //upcasting, is-a relationship  
        ref2 = new Teacher (91, "ali raza", "phd");  
  
        ref1.display(); //call to Employee class display method  
        ref2.display(); //call to Teacher class display method  
  
        System.out.println("Employee: " +ref1.toString());  
  
        System.out.println("Teacher: " + ref2.toString());  
    } //end of main  
}
```



Type Casting

Type Casting

- Java is strongly typed language

- Up-casting

- Implicit
- No loss of information
- Example of

- Primitives

```
int a = 10;  
double b = a;
```

- Classes

```
Employee e = new Teacher();
```

Type Casting

- Downcasting

- Explicit

- Possible loss of information

- Example of

- Primitives

```
double a = 7.65;  
int b = (int) a;
```

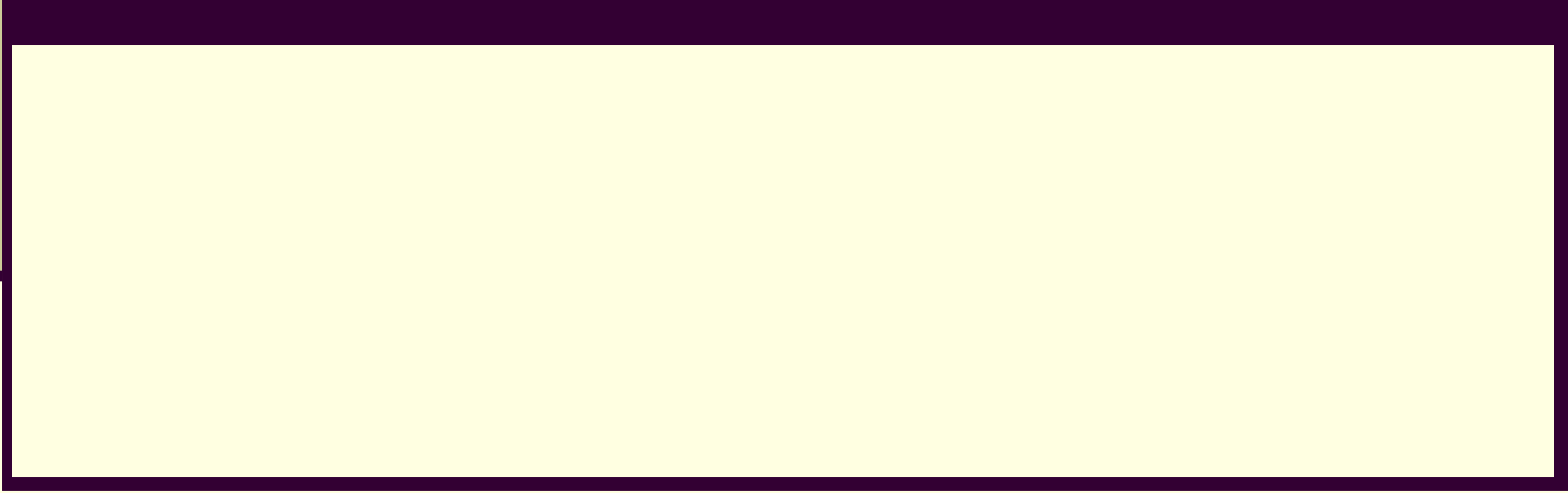
- Classes

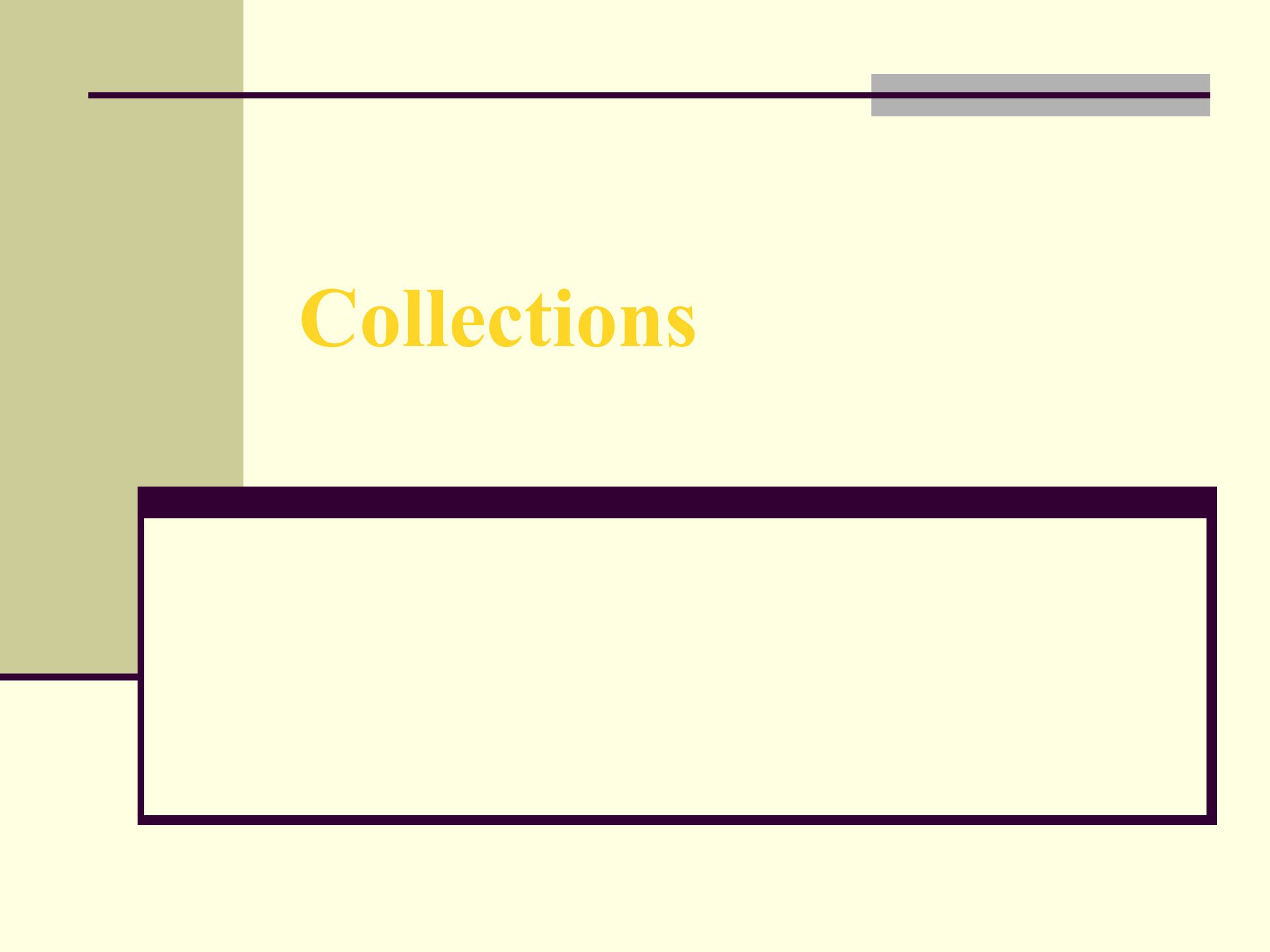
```
Employee e = new Teacher();  
//upcasting  
Teacher t = (Teacher) e;  
//downcasting
```



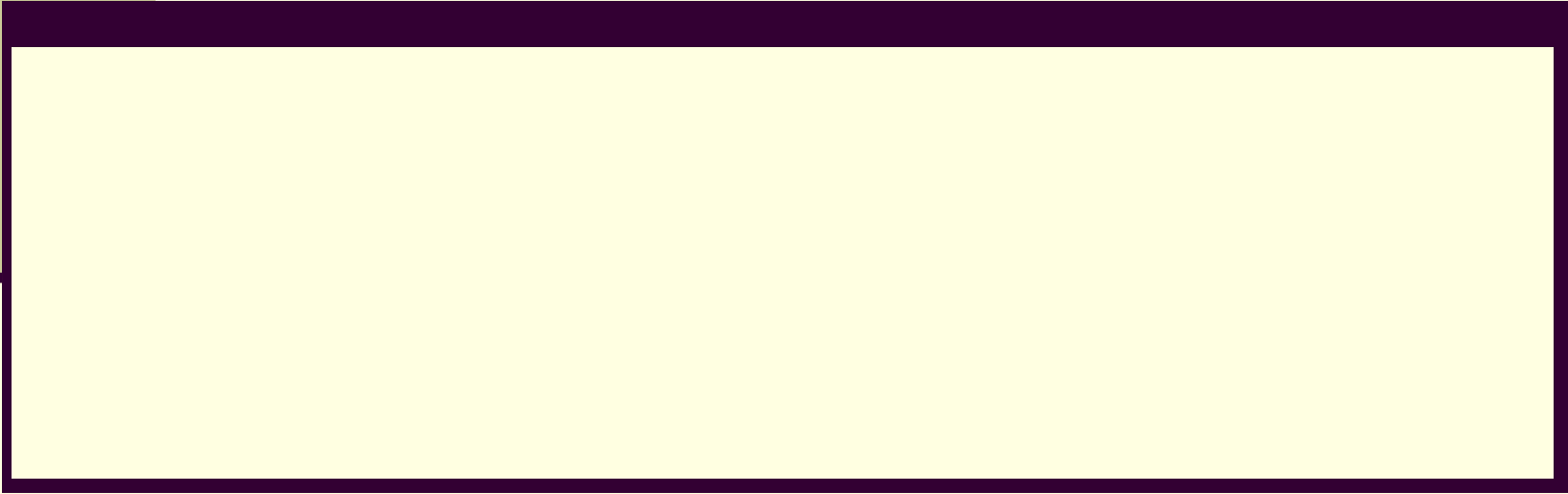
Web Design & Development

Lecture 6





Collections



Collections

- Built-in support for collections
 - Similar to STL in C++
- Collection type
 - Sequence/Set
 - Example ArrayList
- Map type
 - Hashtable/dictionary
 - Example HashMap
- Collections store references to objects
- Use inheritance and interfaces
- Read
 - <http://java.sun.com/docs/books/tutorial/collections>

Collection Design

- All classes provides almost same methods
 - `get()`, `size()`, `isEmpty()`...
 - Easy learning curve for using Collections
- Implemented as reference to Object
 - Similar to using a `void *` in C
 - Require a cast back (down casting) to the actual type
 - Example
 - `String element = (String)arraylist.get(i)`
- Remember, Java checks all casts at run-time

Collection Messages

- Basic messages

- constructor()

- Creates a collection with no elements

- int size()

- Number of elements in the collection

- boolean add()

- Add a new reference/element at the end of the collection
 - Returns true if the collection is modified.

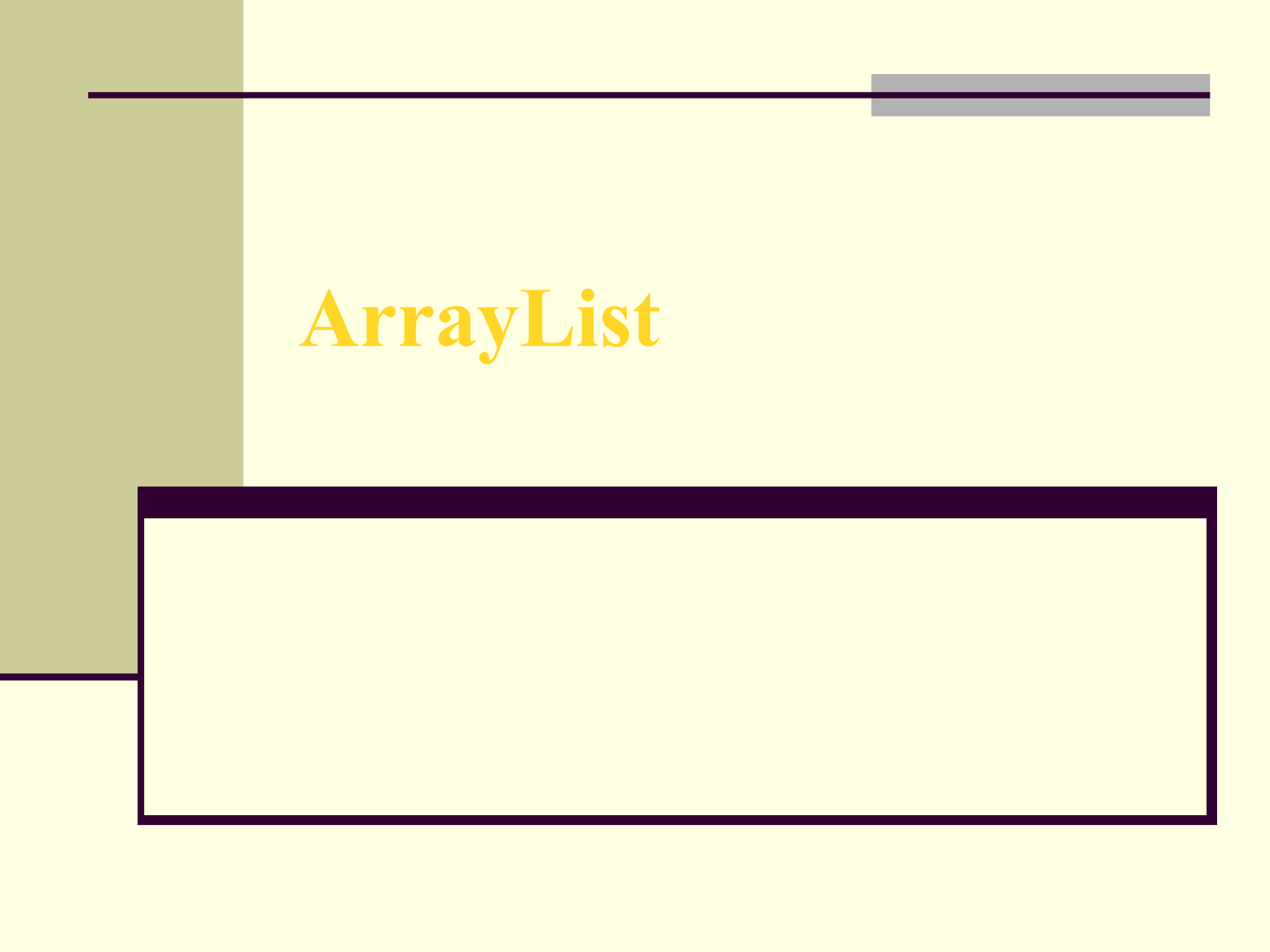
- iterator()

- Returns an Iterator Object

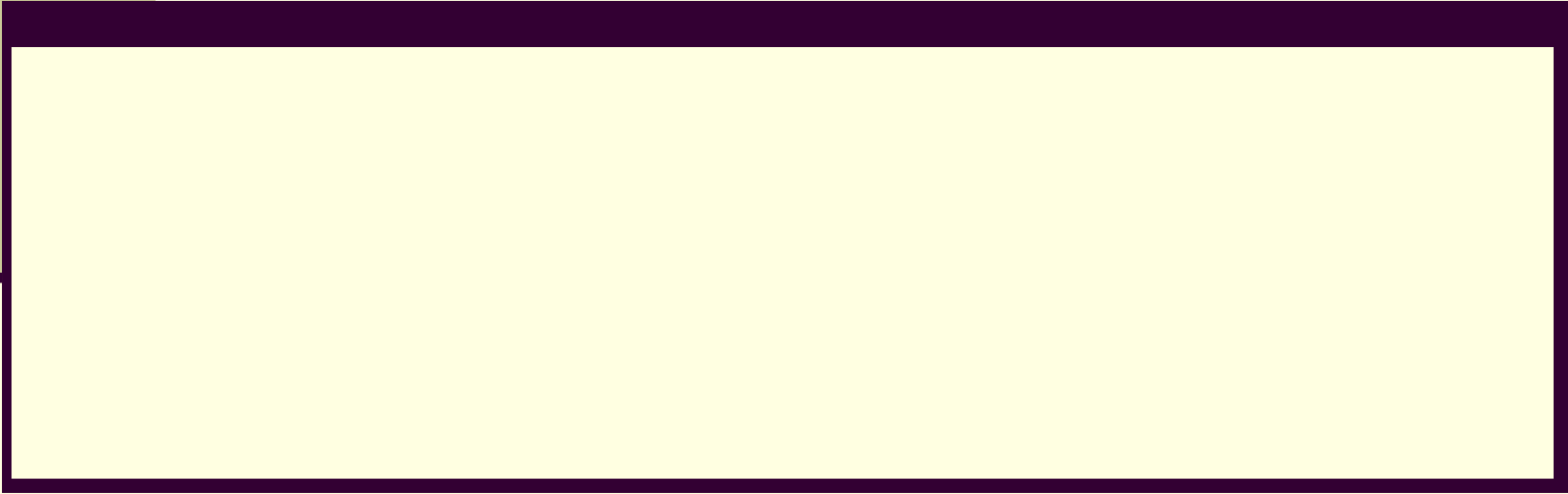
Additional Collection Messages

■ Utilities

- Additional useful methods
- `boolean isEmpty()`
- `boolean contains(Object o)`
 - Iterative search, uses `equals()`
- `boolean remove(Object o)`
 - Iterative remove(), uses `equals()`
- `Boolean addAll(Collection c)`



ArrayList



ArrayList

- Replaces the “Vector”
- Can grow over time
- Methods
 - `add(Object)`
 - Can add all kinds of objects
 - implicit upcasting
 - `int size()`
 - `Object get(int index)`
 - Index is from 0 to `size() - 1`
 - Must cast to appropriate type when used.
 - `remove(index)`
 - Removes object reference at the specified index
 - `iterator()`
 - We'll see an example!

Example TestArrayList. java

```
/* ArrayList in this example, is used to store Student objects.
   We are using the same Student class which we build in our
   previous lectures */
import java.util.*;
public class TestArrayList {
    public static void main (String args[ ]){
        //Create ArrayList object
        ArrayList al = new ArrayList();

        Student s1 = new Student("ali", 1);
        Student s2 = new Student("saad", 2);
        Student s3 = new Student("raza", 3);

        al.add( s1 );
        al.add( s2 );
        al.add( s3 );

        //continue..
    }
}
```

Example TestArrayList.java

```
//checking whether arraylist contains student objects or not
```

```
boolean b = al.isEmpty();
```

```
if (b == true){
```

```
    System.out.println("arraylist is empty");
```

```
}
```

```
else {
```

```
    int size = al.size();
```

```
    System.out.println("arraylist size: "+ size);
```

```
}
```

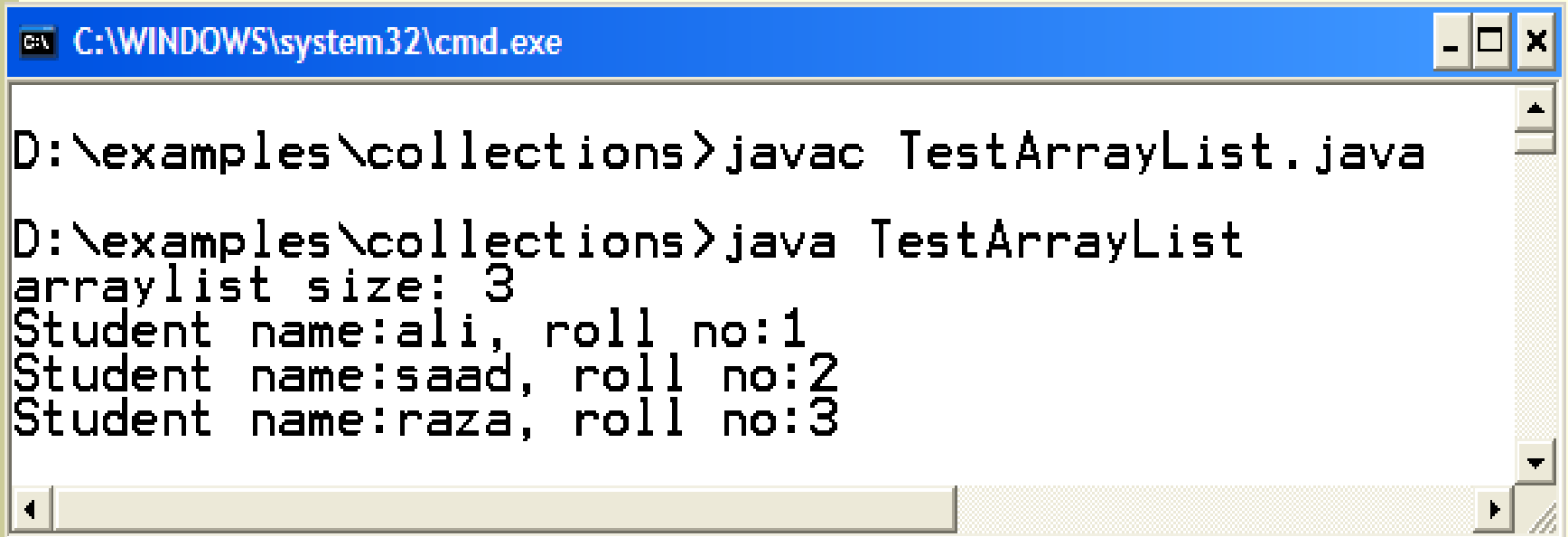
```
// continue..
```


Example TestArrayList.java

```
//using loop to iterate
for (int i=0; i< al.size(); i++) {
    Student s = (Student) al.get(i);
    s.print();
}

} //end of main
}
```

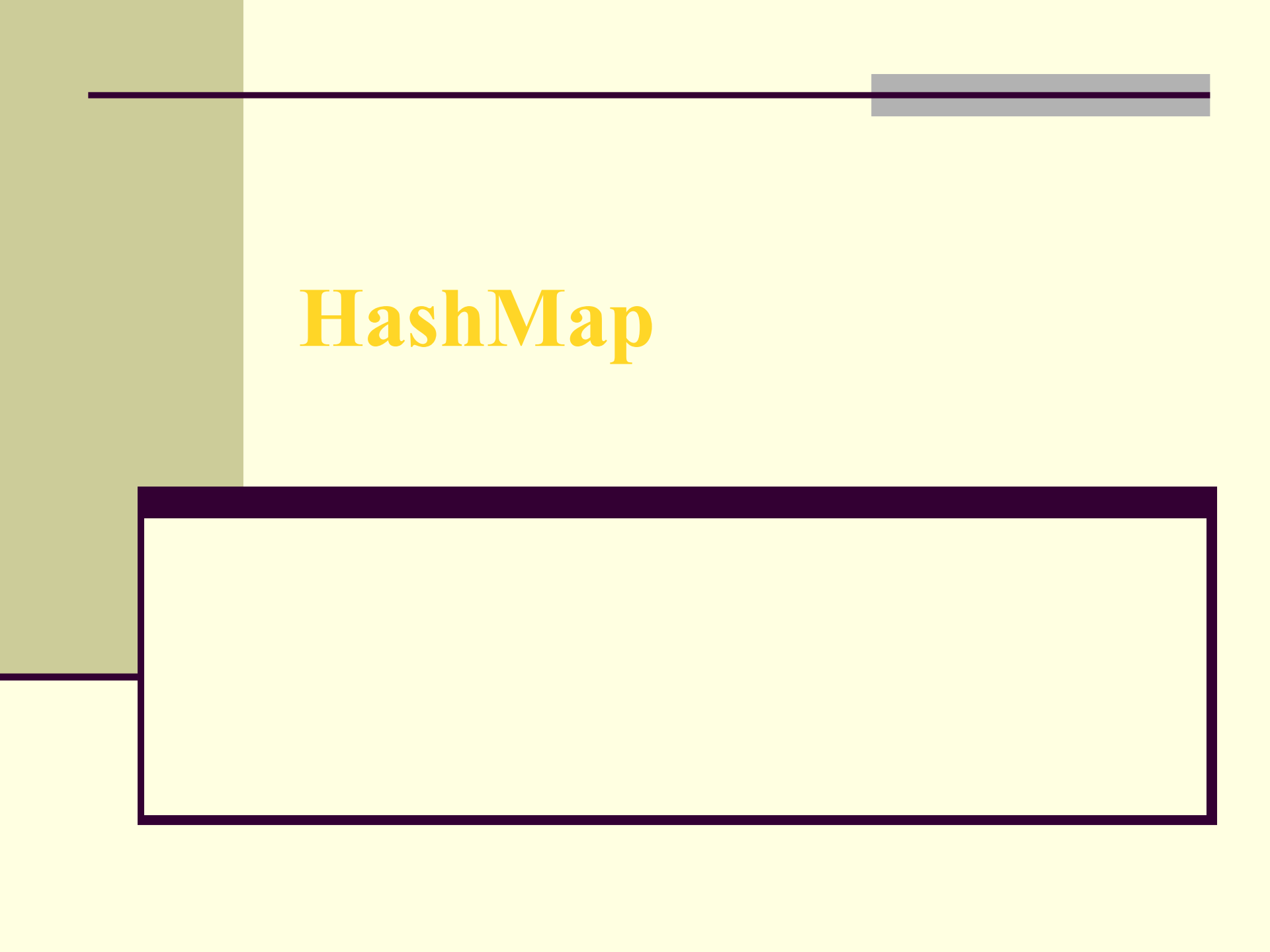
Compile & Execute



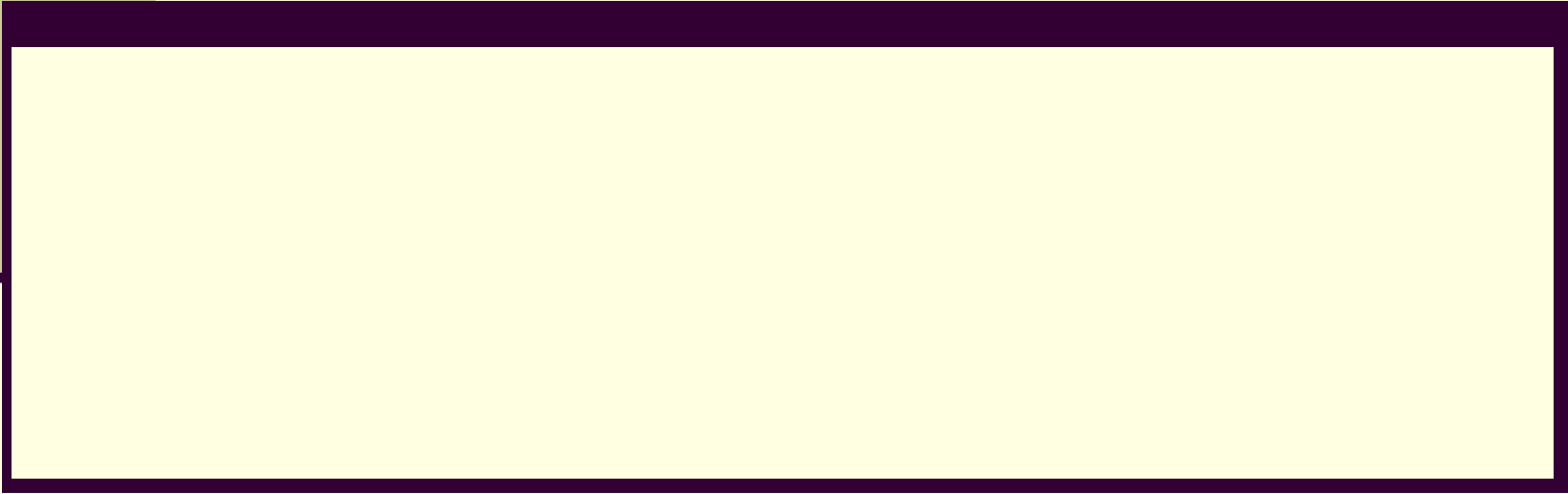
A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" along with standard window control buttons (minimize, maximize, close). The command prompt shows the following sequence of commands and output:

```
D:\examples\collections>javac TestArrayList.java
D:\examples\collections>java TestArrayList
arraylist size: 3
Student name:ali, roll no:1
Student name:saad, roll no:2
Student name:raza, roll no:3
```

The output displays the size of the arraylist as 3 and lists three students with their names and roll numbers.



HashMap



HashMap

- Stores key & value in pair form
- Cannot contain duplicate keys
- Implements the Map interface
- Keys and Values are stored as Objects
 - `put(Key, Value)`
 - implicit upcasting
 - `Object get(Key)`
 - Must cast to appropriate type when used.
 - `int size()`

Example TestHashMap. java

```
/* HashMap is used to store Student objects as value and  
rollnos as keys.  
We are using the same Student class which we build in our  
previous lectures */  
  
import java.util.*;  
  
public class TestHashMap {  
    public static void main (String args[]){  
        HashMap h = new HashMap();  
  
        Student s1 = new Student("ali", 1);  
        Student s2 = new Student("saad", 2);  
        Student s3 = new Student("raza", 6);  
  
        h.put("one", s1 );  
        h.put("two", s2 );  
        h.put("six", s6 );  
  
        //continue..
```

Example TestHashMap.java

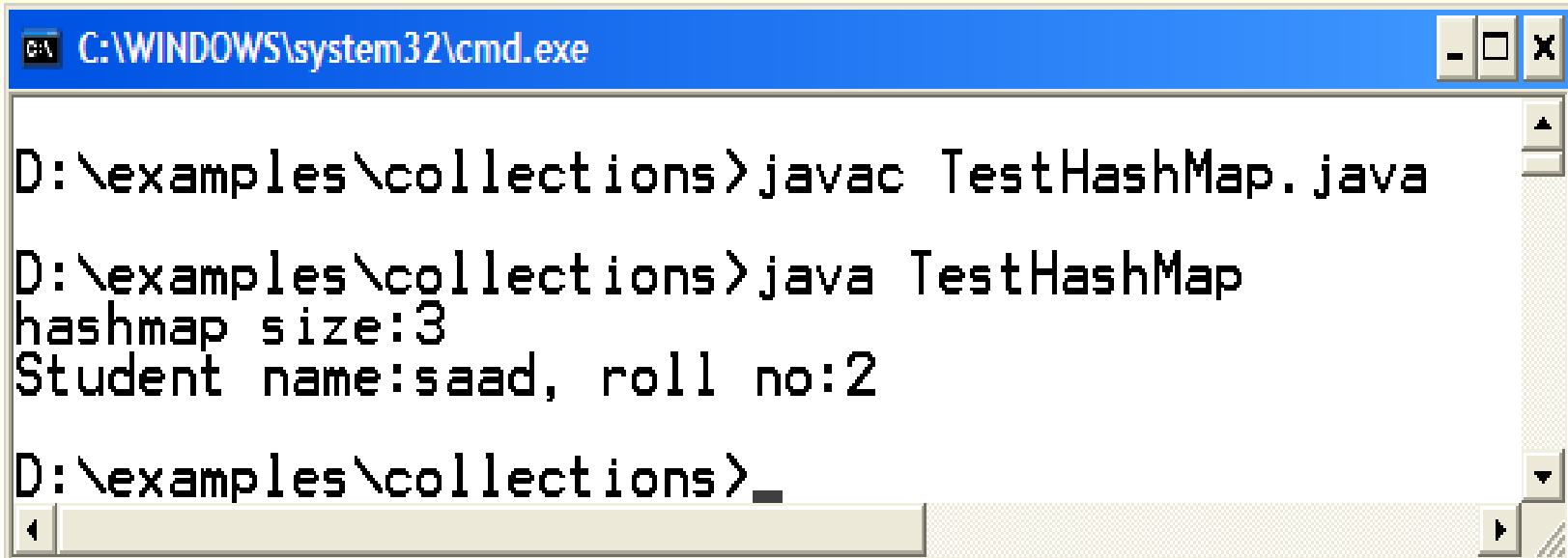
```
boolean b = h.isEmpty();

if (b == true){
    System.out.println("hashmap is empty");
}
else {
    int size = h.size();
    System.out.println("hashmap size:"+ size);
}

Student s = (Student) h.get("two");
s.print();

} //end of main
}
```

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\collections>javac TestHashMap.java

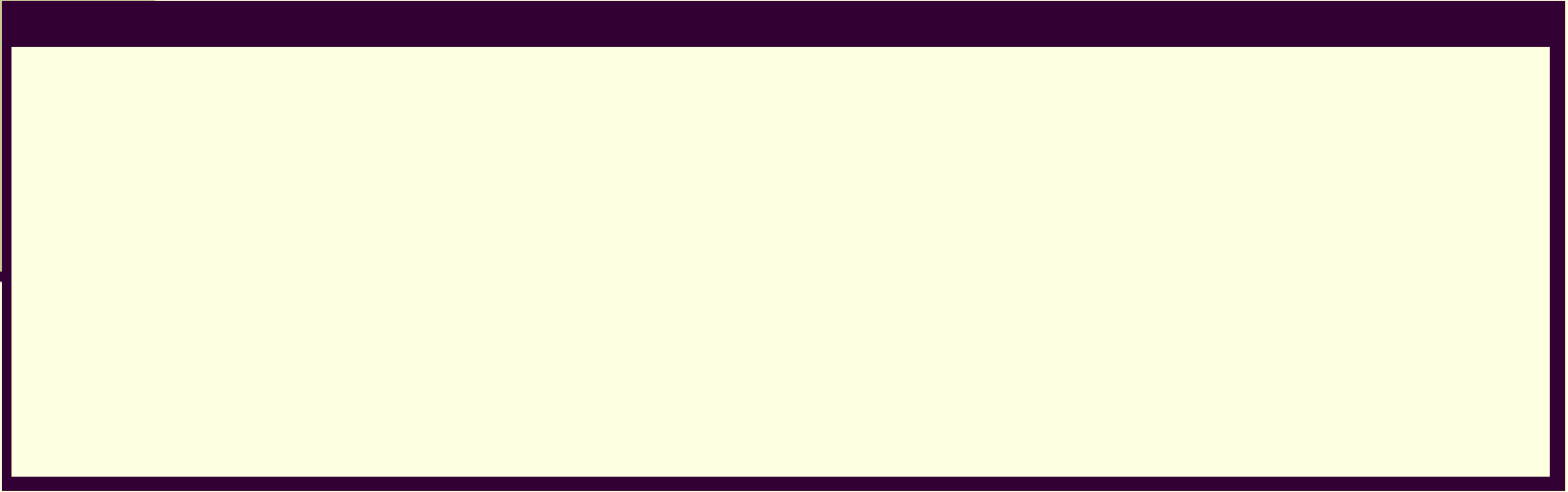
D:\examples\collections>java TestHashMap
hashmap size:3
Student name:saad, roll no:2

D:\examples\collections>
```

The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window contains the following text:
D:\examples\collections>javac TestHashMap.java
D:\examples\collections>java TestHashMap
hashmap size:3
Student name:saad, roll no:2
D:\examples\collections>_
The text is displayed in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.



Putting All Together



We have learned So far

- How to Perform IO
- Selection and Control Structures
- OOP
- Collections
- So, lets do a small problem

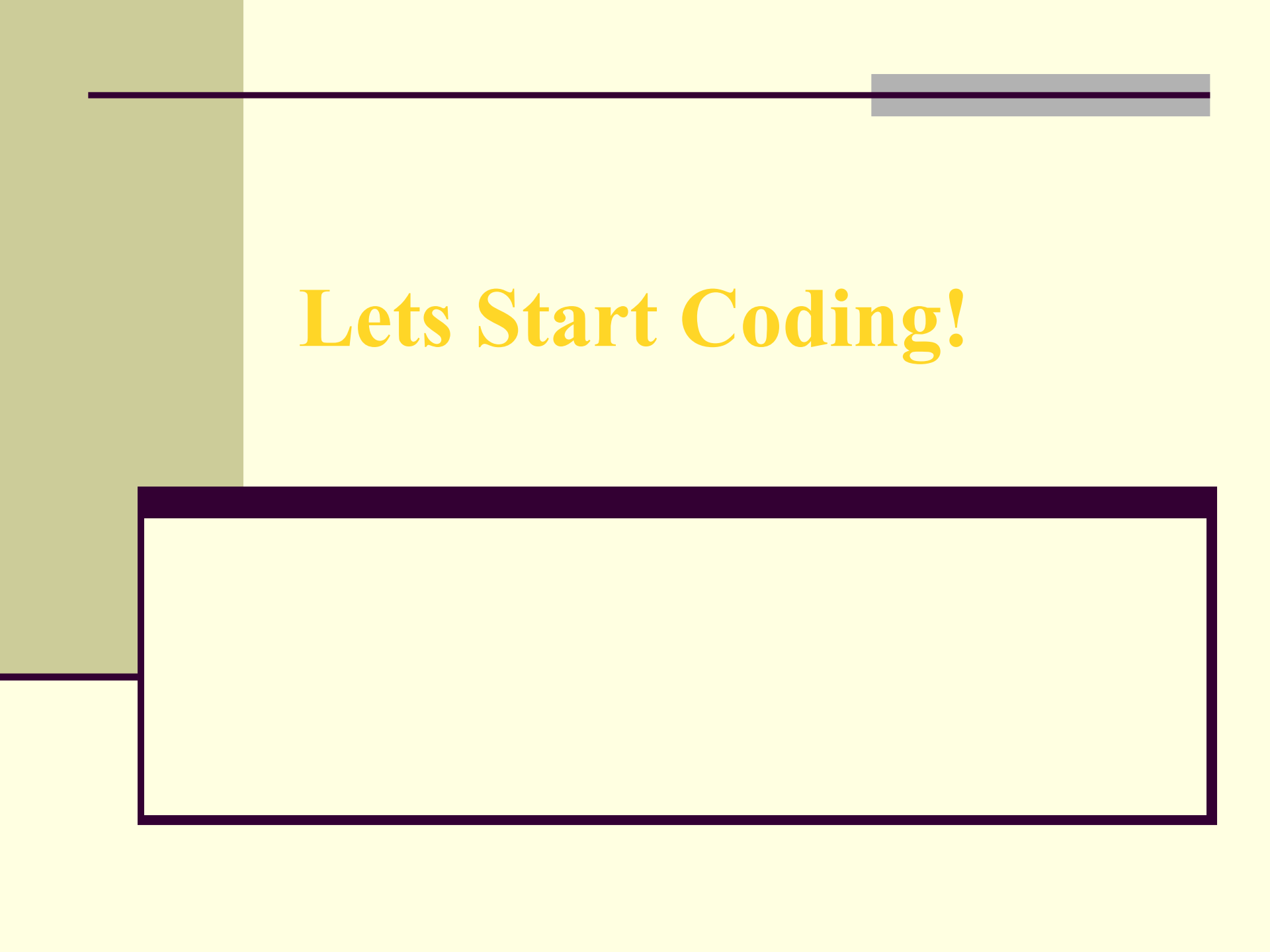
Problem

Address Book

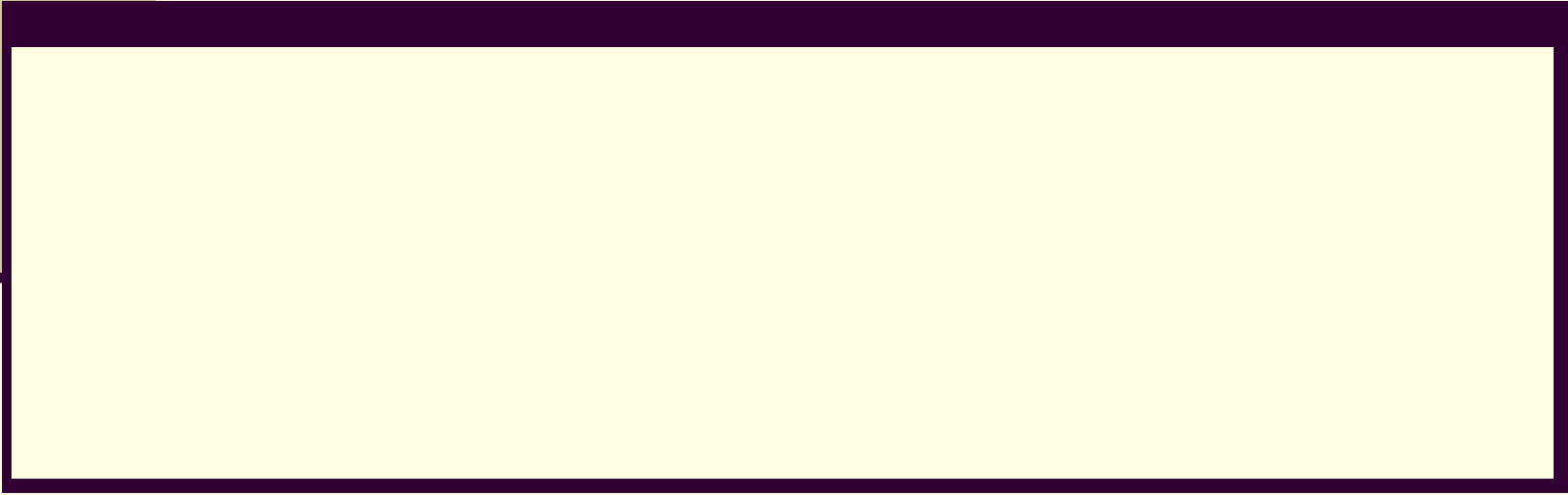
- Wants to store name, address, phone no of a Person
- Features
 - Add
 - Delete
 - Search (on name)
 - Exit (from application)
- The above listed features must be available to user in the form of JOptionPane based MENU.

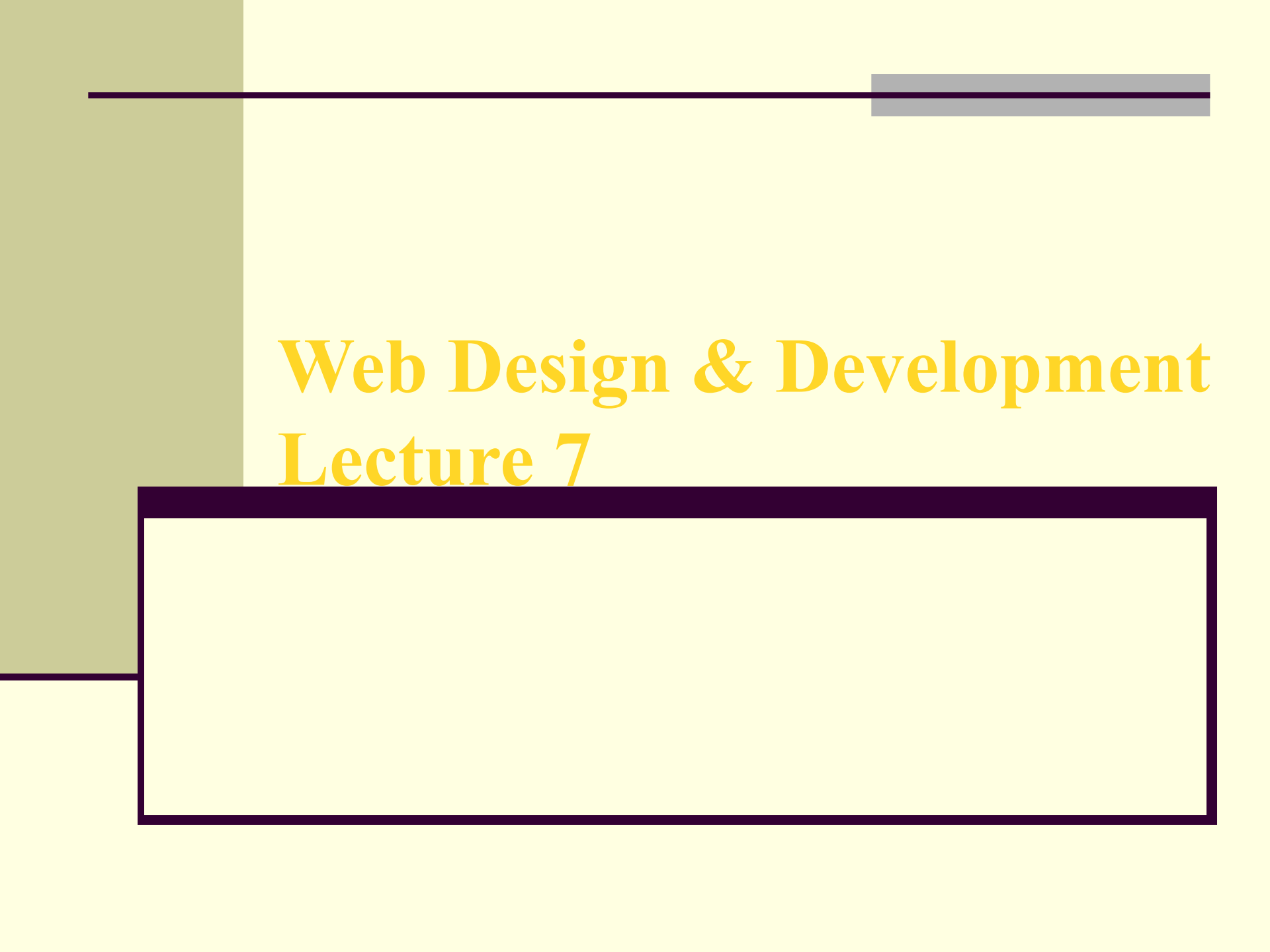
Approach for Solving Problem

- Step 1
 - Make a Person class with name, address, phone no attributed
- Step 2
 - Make a AddressBook class
 - Use ArrayList to store Person's Obejcts
 - Write methods add, delete and Search
 - Use JOptionPane methods for IO
- Step 3
 - Make a Test class (Driver Program)
 - Build Menu using switch selection structure
 - Call appropriate methods of AddressBook



Lets Start Coding!





Web Design & Development

Lecture 7





Intro to Exceptions



Types of Programming Errors

- Three types of error:
 - **Syntax Errors** – arise because the rules of the language are not followed.
 - **Runtime Errors** – arise because the program tries to perform an operation that is impossible to carry out.
 - **Logic Errors** – arise because the program does perform the way it was intended to.
- **Syntax errors** are caught by the compiler, and fixed before the program is run.
- **Logic Errors** are detected by testing, and are fixed through debugging.
- **Runtime Errors** cause *Exceptions* and may be handled at runtime.

Exceptions

- An *exception* is an event that describes an unusual or erroneous situation at runtime.
- Exceptions are wrapped up as objects
- A program can deal with an exception in one of three ways:
 - ignore it
 - handle it where it occurs
 - handle it an another place in the program
- An *error* is also represented as an object in Java, but usually represents an unrecoverable situation and should not be caught

Why Use Exceptions?

- Uses of exception handling
 - Process exceptions from program components
 - Handle exceptions in a uniform manner in large projects
 - Remove error-handling code from “main line” of execution
- What if Exception is not handled?
 - Might terminate program execution

Exceptions Types

- Two Types

- Unchecked

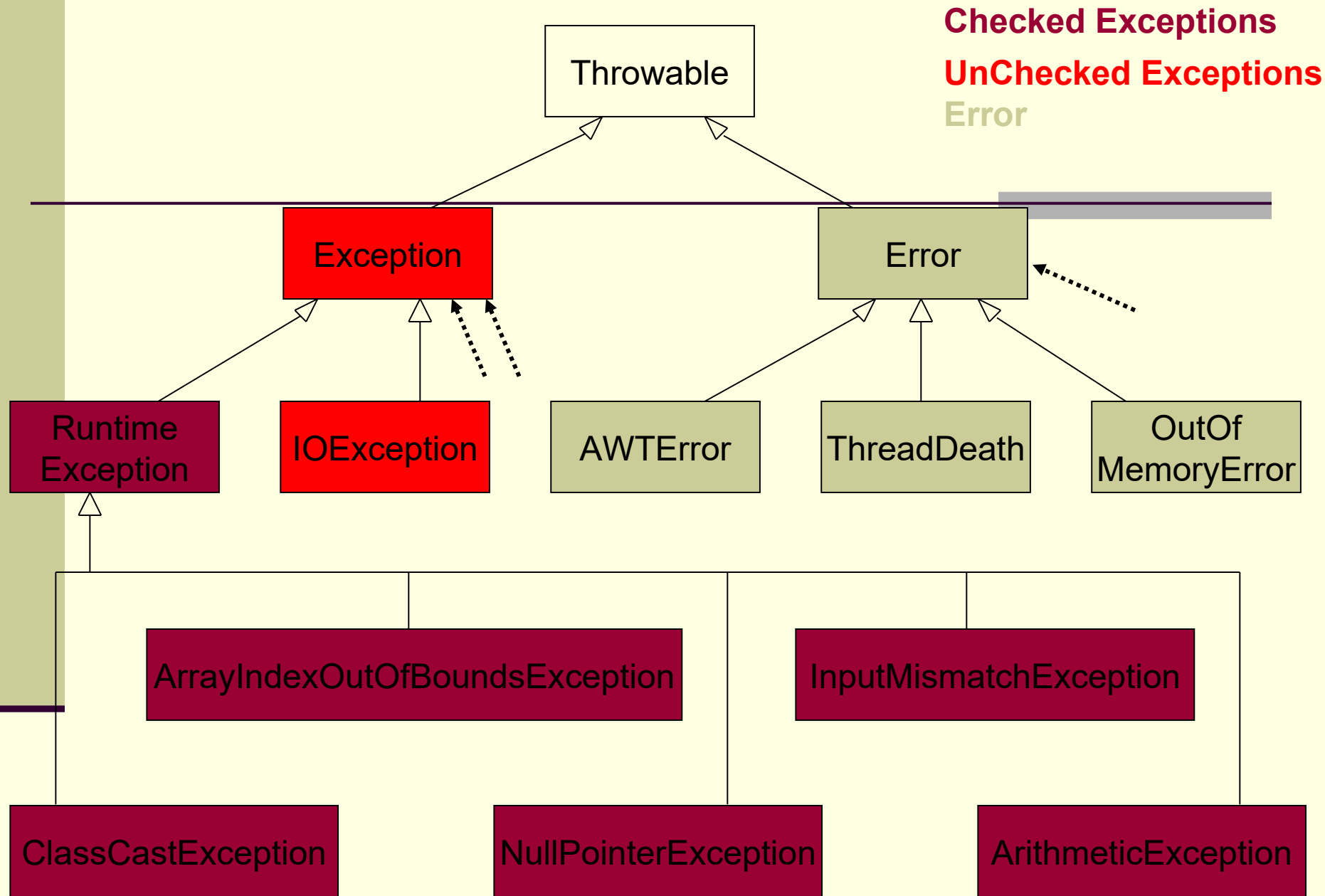
- Subclasses of RuntimeException and Error.
 - Does not require explicit handling
 - Run-time errors are internal to your program, so you can get rid of them by debugging your code
 - For example, null pointer exception; index out of bounds exception; division by zero exception; ...

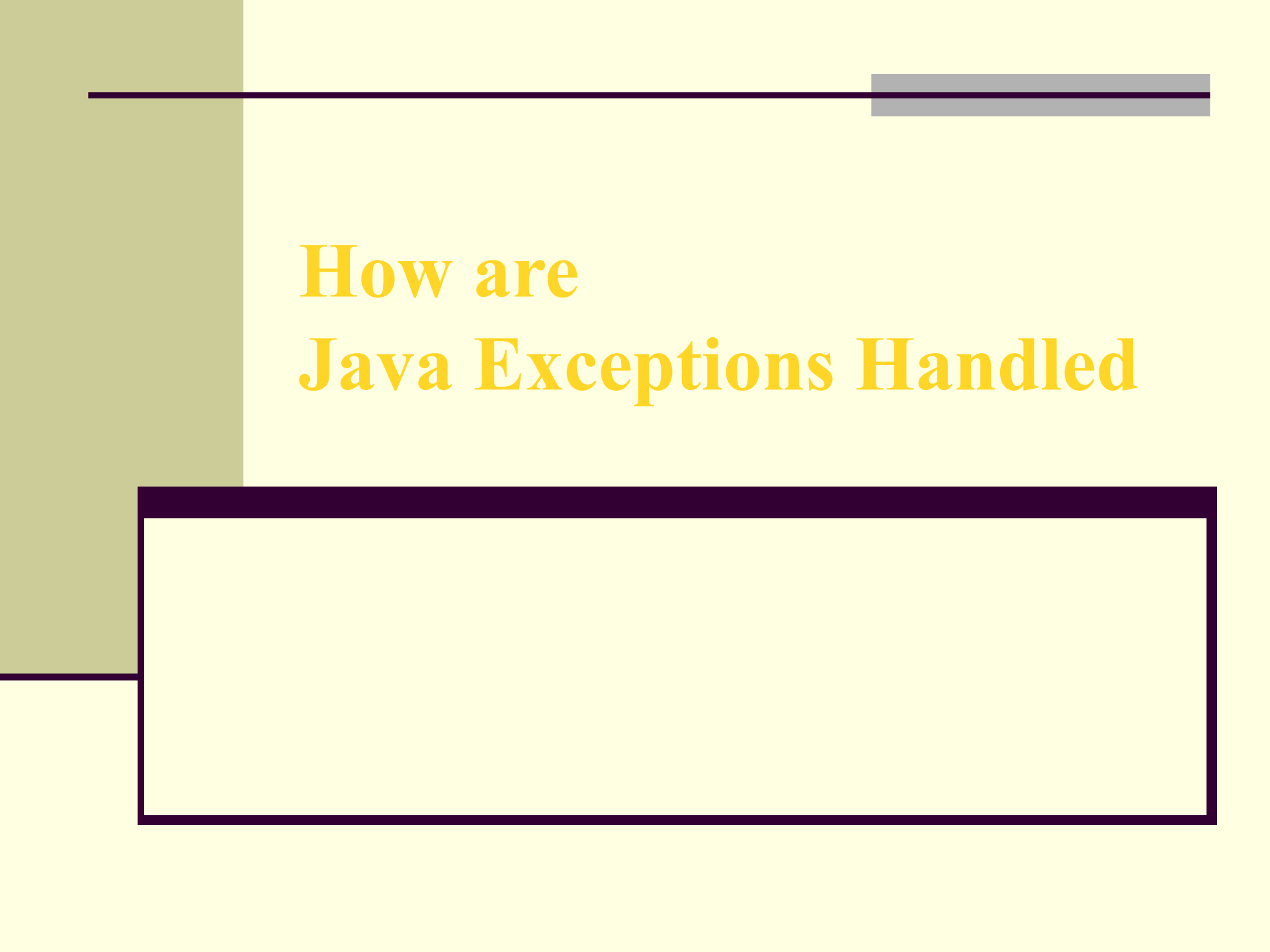
Exceptions Types

- Two Types

- Checked

- Must be caught or declared in a throws clause
 - Compile will issue an error if not handled appropriately
 - Subclasses of Exception other than subclasses of RuntimeException.
 - Other arrive from external factors, and cannot be solved by debugging
 - Communication from an external resource – e.g. a file server or database





How are Java Exceptions Handled

How are Java exceptions handled

- Basic Java exception handling is managed via keywords: **try**, **catch**, **finally**, **throw**, **throws**.
- try block
 - Code that could generate errors put in try blocks
- catch block
 - Code for error handling enclosed in a catch clause
- finally block
 - The finally clause always executes
 - Resources that are opened may need to be closed during exception handling
 - Appears after the last catch block
 - It will not execute if System.exit(0) occurs first

How are Java exceptions handled

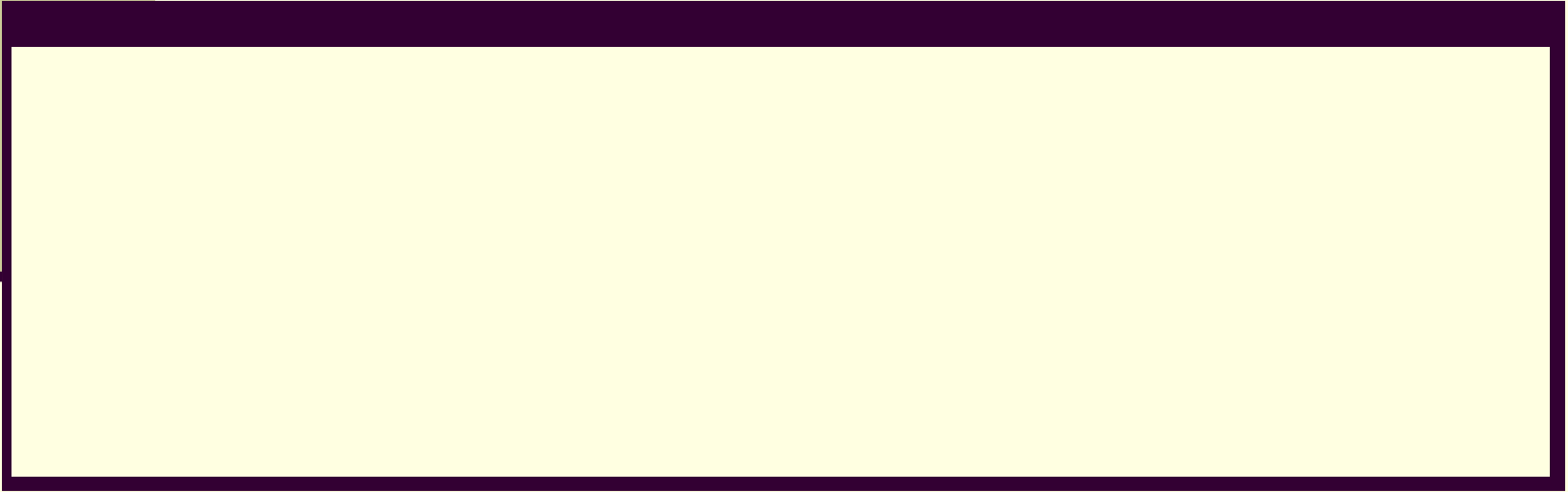
- throw
 - To manually throw an exception, use the keyword **throw**.
- throws
 - **throws** exception out of the method, requiring it to be **caught** and handled by an appropriate exception handler
 - Any exception that is thrown out of a method must be specified as such by a **throws** clause.

Exception-Handling Struct

```
try                                     //try block
{
    // write code that could generate exceptions
} catch (<exception to be caught>)      //catch block
{
    //write code for exception handling
}
.....
catch (<exception to be caught>)      //catch block
{
    //code for exception handling
} finally                               //finally block
{
    //any clean-up code, release the acquired resources
}
```



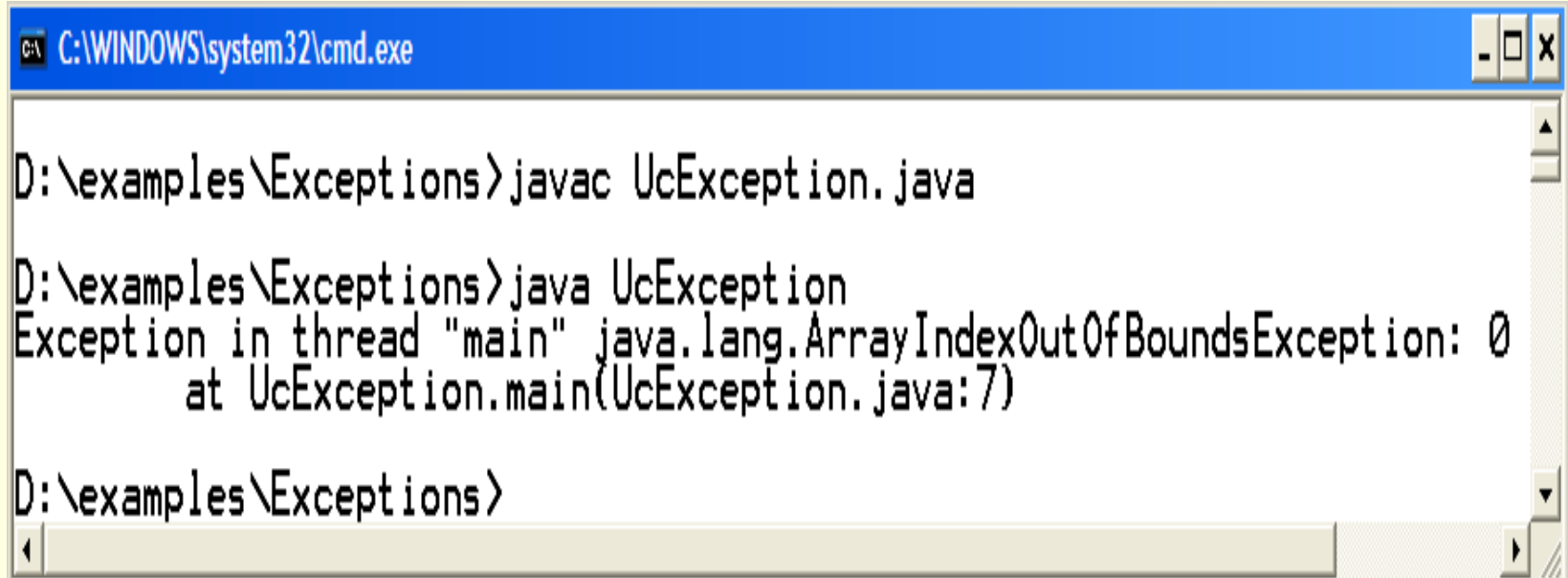

Examples



Example: Unchecked Exceptions

```
public class UcException {  
    public static void main(String args[ ]) {  
        System.out.println(args[0]);  
    }  
}
```

Example: Unchecked Exceptions compile & execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\Exceptions>javac UcException.java

D:\examples\Exceptions>java UcException
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at UcException.main(UcException.java:7)

D:\examples\Exceptions>
```

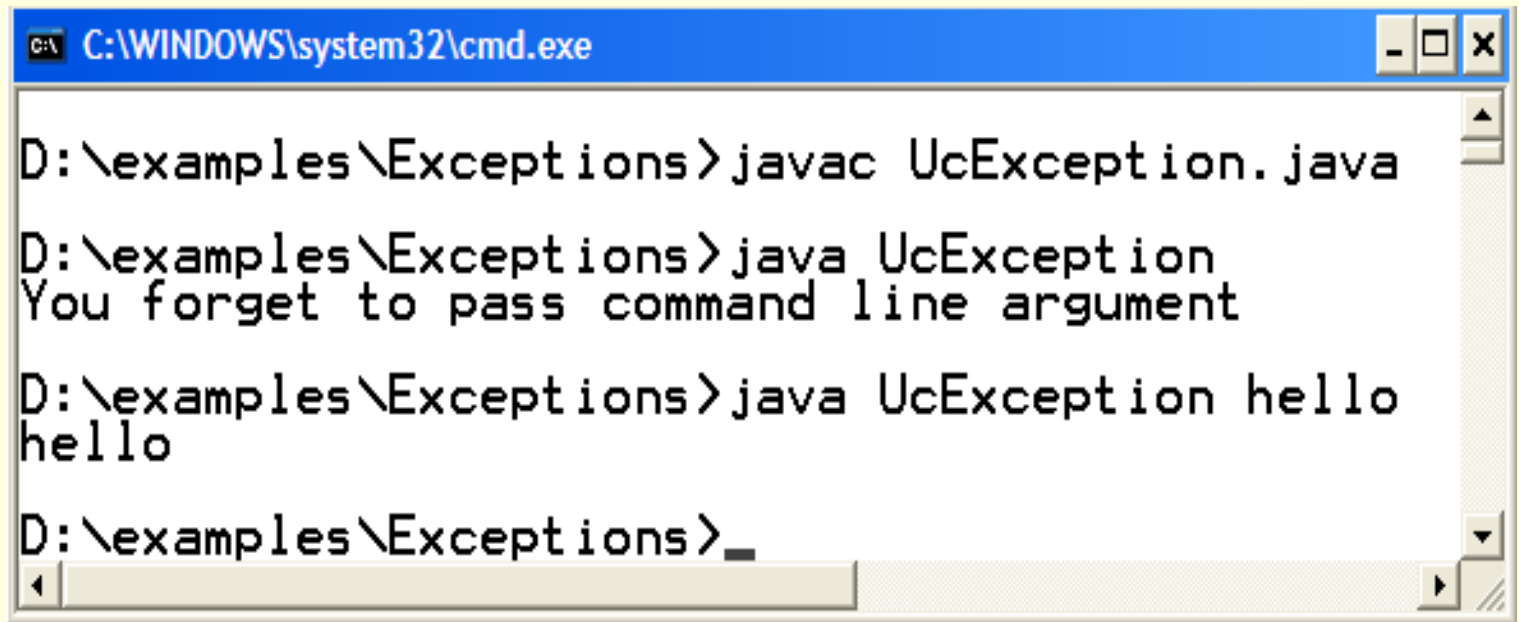
Example: Unchecked Exceptions

```
public class UcException {  
    public static void main(String args[ ]) {  
        System.out.println(args[0]);  
    }  
}
```

Example: Unchecked Exceptions Modification

```
public class UcException {  
    public static void main(String args[ ]) {  
        try {  
            System.out.println(args[0]);  
        } catch (IndexOutOfBoundsException ex) {  
            System.out.println("You forget to pass command line argument");  
        }  
    }  
}
```

Example: Unchecked Exceptions compile & execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\Exceptions>javac UcException.java

D:\examples\Exceptions>java UcException
You forget to pass command line argument

D:\examples\Exceptions>java UcException hello
hello

D:\examples\Exceptions>_
```

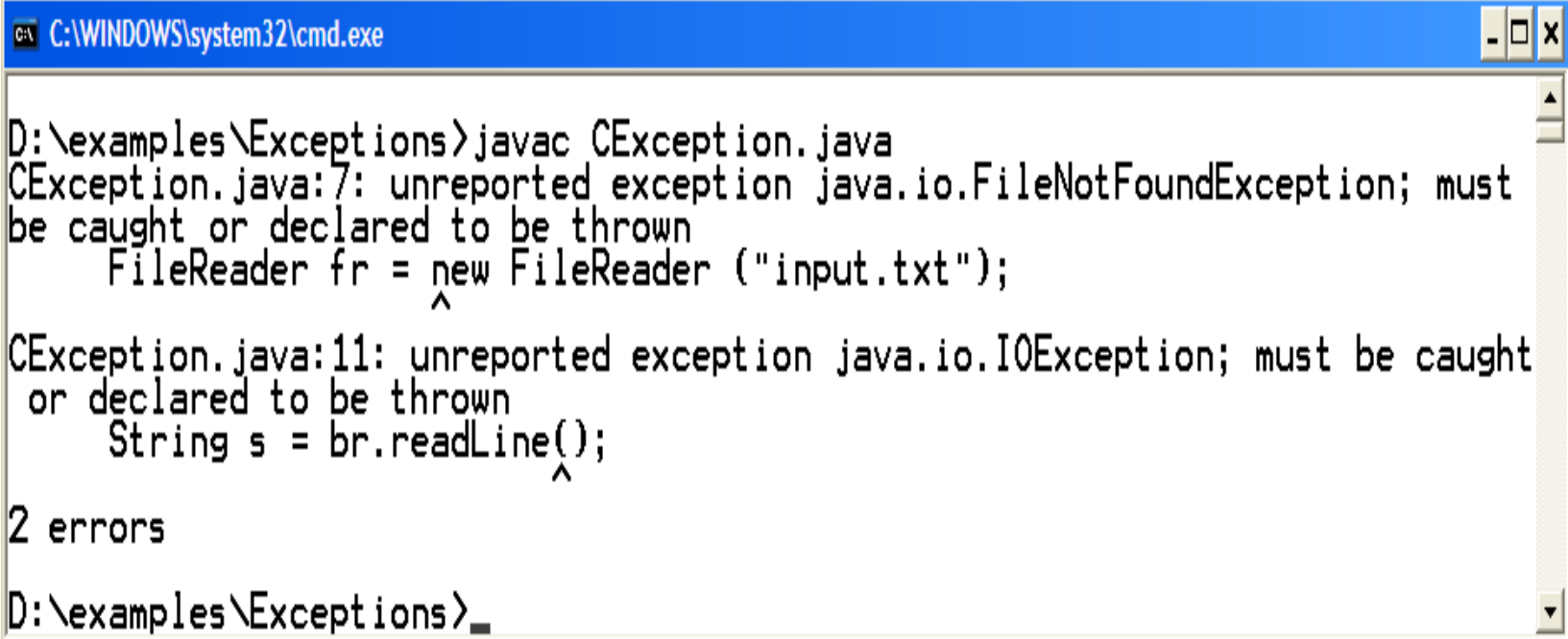
Example: Checked Exceptions

```
import java.io.*;

public class CException {
    public static void main(String args[ ]) {
        FileReader fr = new FileReader ("input.txt");
        BufferedReader br = new BufferedReader(fr);

        //read the line
        String s = br.readLine();
        System.out.println(s);
    }
}
```

Example: Checked Exceptions compile & execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\Exceptions>javac CException.java
CException.java:7: unreported exception java.io.FileNotFoundException; must
be caught or declared to be thrown
    FileReader fr = new FileReader ("input.txt");
                      ^
CException.java:11: unreported exception java.io.IOException; must be caught
or declared to be thrown
    String s = br.readLine();
                  ^
2 errors

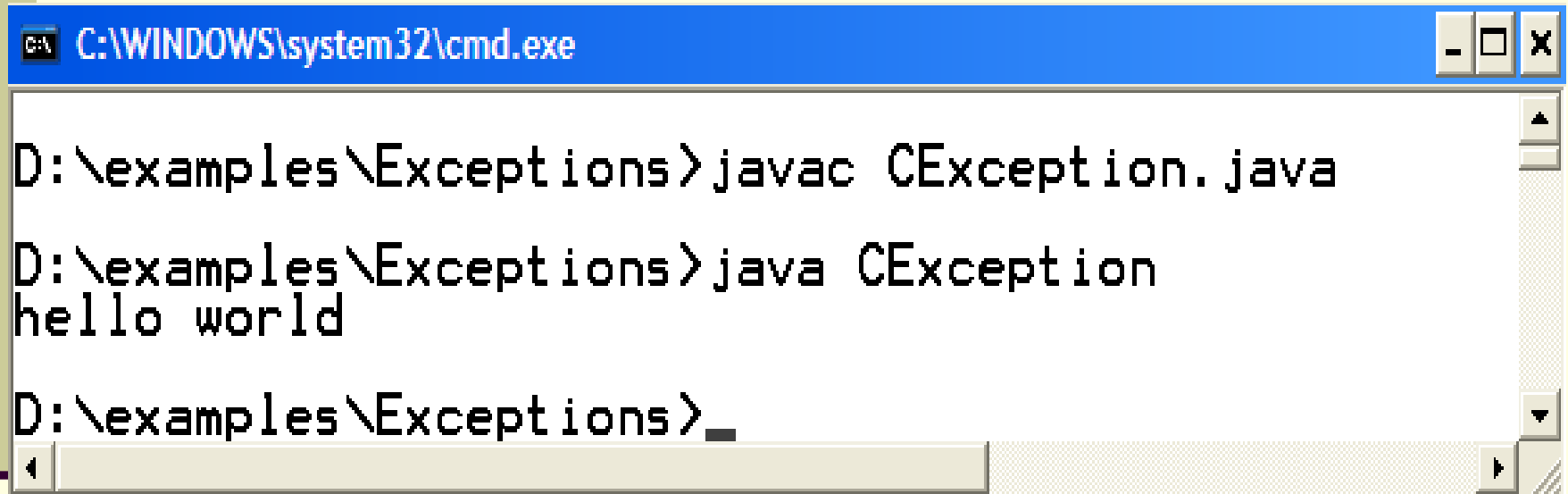
D:\examples\Exceptions>_
```


Example: Checked Exceptions Modification

```
import java.io.*;
public class CException {
    public static void main(String args[ ]) {
        try {
            FileReader fr = new FileReader ("input.txt");
            BufferedReader br = new BufferedReader(fr);

            //read the line
            String s = br.readLine();
            System.out.println(s);
        } catch (IOException ex ) {
            System.out.println(ex);
        }
    }
}
```

Example: Checked Exceptions compile & execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\Exceptions>javac CException.java

D:\examples\Exceptions>java CException
hello world

D:\examples\Exceptions>_
```

The screenshot shows a Windows command prompt window with a blue title bar containing the text "C:\WINDOWS\system32\cmd.exe". The window has standard minimize, maximize, and close buttons. The command prompt shows the following sequence of commands and output:

- Current directory: `D:\examples\Exceptions`
- Command: `javac CException.java`
- Command: `java CException`
- Output: `hello world`
- Current directory: `D:\examples\Exceptions`

The window includes a vertical scrollbar on the right and a horizontal scrollbar at the bottom.

Example: finally block

```
import java.io.*;

public class FBlockDemo {
    public static void main(String args[ ]) {
        try {
            FileReader fr = new FileReader ("numbers.txt");
            BufferedReader br = new BufferedReader(fr);

            String s = br.readLine();
            System.out.println(s);

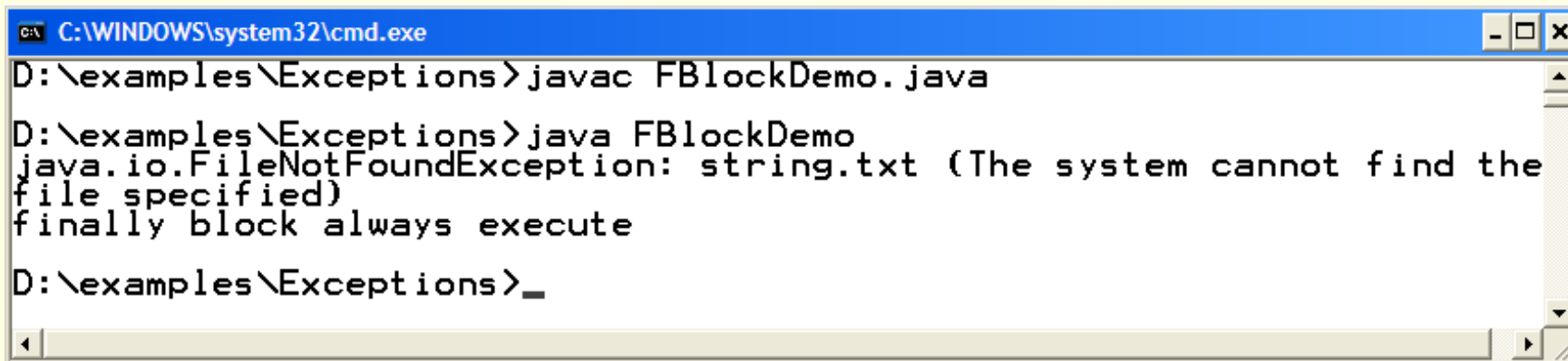
        } catch (IOException ioEx) {
            System.out.println(ioEx);
        } finally {
            System.out.println("finally block always execute");
            //write any clean up code if required
        }

        //end of main
    } //end of class
}
```

Compile & Execute

If “string.txt” isn’t there, it will throw FileNotFoundException

Note that finally block executes



```
C:\WINDOWS\system32\cmd.exe
D:\examples\Exceptions>javac FBlockDemo.java
D:\examples\Exceptions>java FBlockDemo
java.io.FileNotFoundException: string.txt (The system cannot find the
file specified)
finally block always execute
D:\examples\Exceptions>_
```

If “string.txt” exist, hopefully no exception would be thrown

Note that finally block still executes



```
C:\WINDOWS\system32\cmd.exe
D:\examples\Exceptions>javac FBlockDemo.java
D:\examples\Exceptions>java FBlockDemo
hello world
finally block always execute
D:\examples\Exceptions>_
```

Multiple Catch Blocks

- Possible to have multiple catch clauses for a single try statement
 - Essentially checking for different types of exceptions that may happen
- Evaluated in the order of the code
 - *Bear in mind the Exception hierarchy when writing multiple catch clauses!*
 - If you catch Exception first and then IOException, the IOException will never be caught!

Example: Multiple catch blocks

```
/* numbers.txt contains numbers. After reading number
   from file, prints its square on console */

import java.io.*;

public class MCatchDemo {
    public static void main(String args[ ]) {
        try {
            //may throw FileNotFoundException & IOException
            FileReader fr = new FileReader ("numbers.txt");
            BufferedReader br = new BufferedReader(fr);

            //read the line
            String s = br.readLine();

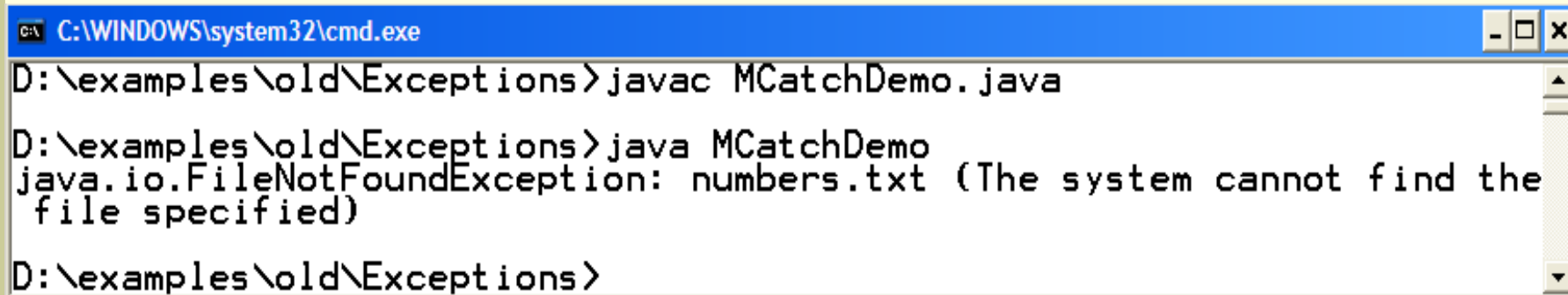
            //may throw NumberFormatException, if s is not no.
            int number = Integer.parseInt(s);
            System.out.println(number * number);
        }
    }
}
```

Example: Multiple catch blocks

```
} catch (NumberFormatException nfEx ) {  
    System.out.println(nfEx);  
} catch (FileNotFoundException fnfEx) {  
    System.out.println(fnfEx);  
} catch (IOException ioEx) {  
    System.out.println(ioEx);  
}  
}  
}
```

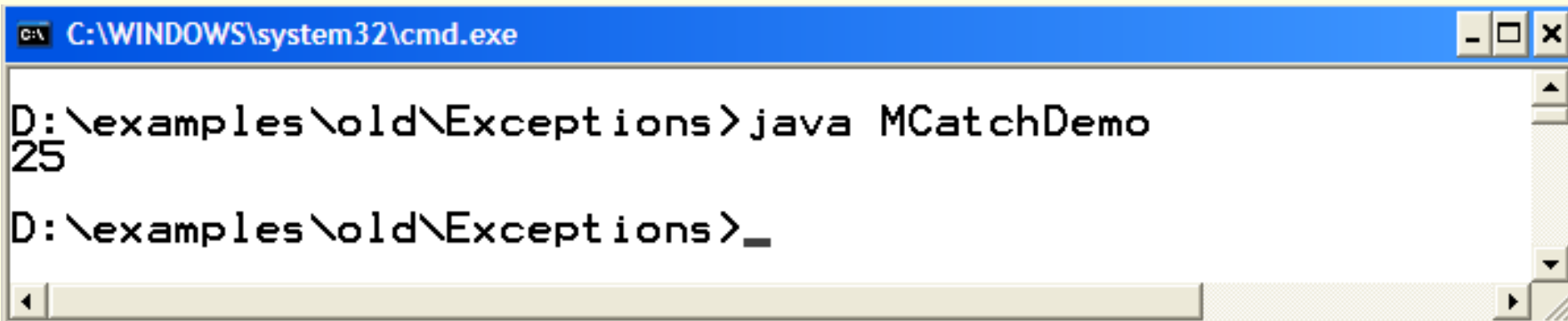
Compile & Execute

If “numbers.txt” isn’t there, it will throw FileNotFoundException



```
C:\WINDOWS\system32\cmd.exe
D:\examples\old\Exceptions>javac MCatchDemo.java
D:\examples\old\Exceptions>java MCatchDemo
java.io.FileNotFoundException: numbers.txt (The system cannot find the
file specified)
D:\examples\old\Exceptions>
```

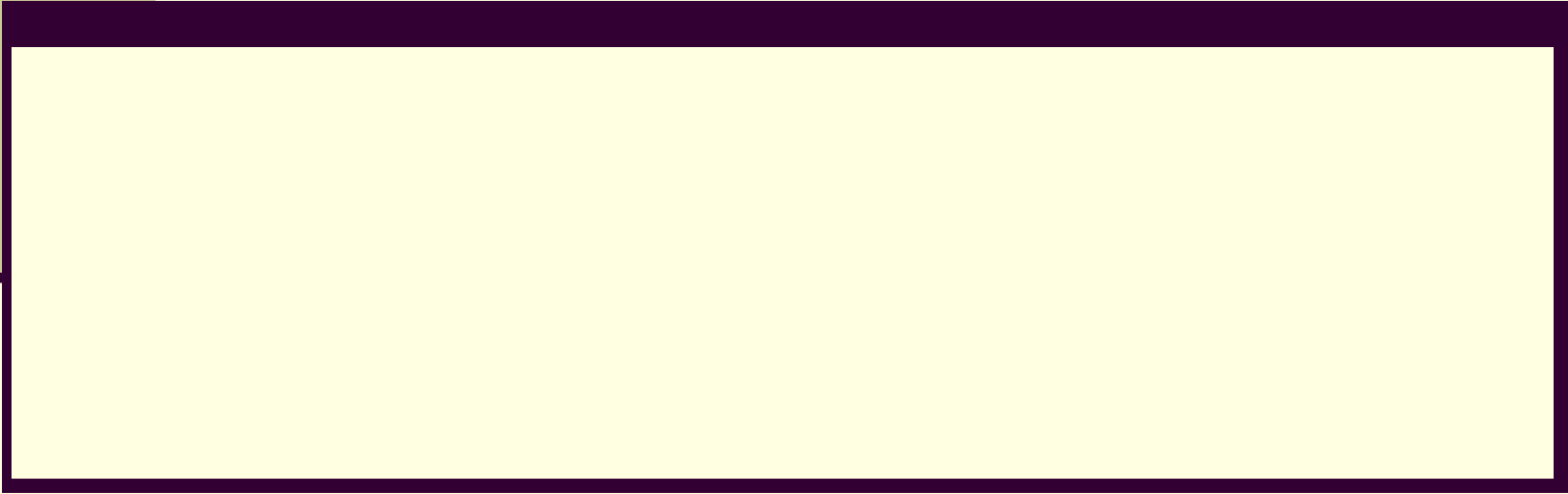
If “numbers.txt” exist and contains a number, Hopefully no exception would be occurred



```
C:\WINDOWS\system32\cmd.exe
D:\examples\old\Exceptions>java MCatchDemo
25
D:\examples\old\Exceptions>_
```




The throws clause



The throws clause

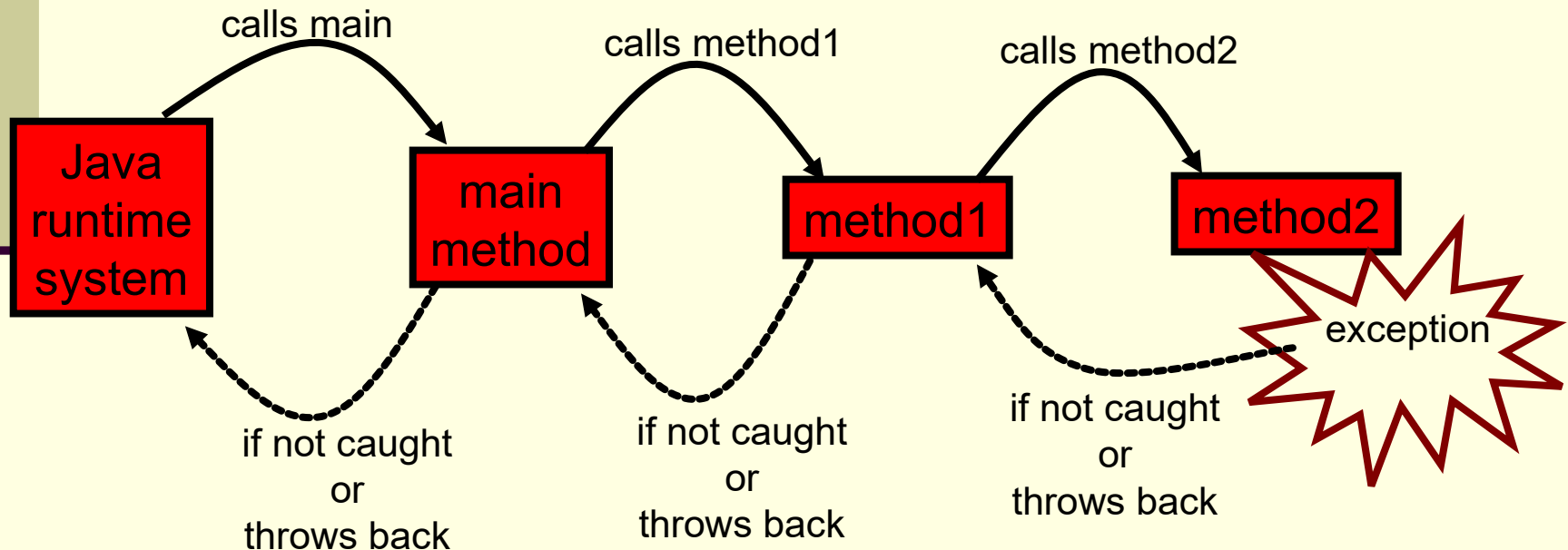
- Method doesn't want to handle exception itself
- it **throws** the exception, the caller should handle this exception or throws the exception itself
- A method should specify the exceptions it throws by placing a throws clause after the parameter list

printStackTrace() is your friend!

- When dealing with exceptions
- Especially when debugging
- printStackTrace() will:
 - Show you the full calling history
 - With line numbers
- Note:
 - *Bad* idea to eat an exception silently!
 - Either printStackTrace() or pass it along to be handled at a different level

What if

- Method throws back the exception
- No catch blocks matches
- Exception is not handled



Example: throws clause

```
// this example shows the use of throws clause and printStackTrace() method
import java.io.*;

public class ThrowsDemo {
    //method used to read line from file
    public static void method1 () {
        try {
            FileReader fr = new FileReader("string.txt");
            BufferedReader br = new BufferedReader(fr);

            String s = br.readLine();
            System.out.println(s);
        } catch (IOException ioEx) {
            ioEx.printStackTrace();
        }
    } //end of method
}
```

Example: throws clause

```
// used to call method1  
public static void method1 () {  
    method2();  
}
```

```
public static void main(String args[]){  
    ThrowsDemo.method1();  
}
```

```
}//end of class
```

Example: throws clause

- Method2 doesn't want to handle exception itself, so it throws the exception to the caller
- So method2 modified as

```
.....  
    public static void method2 () throws IOException {  
  
        FileReader fr = new FileReader();  
        BufferedReader br = new BufferedReader(fr);  
  
        String s = br.readLine();  
        System.out.println(s);  
  
    } //end of method  
.....
```

Example: throws clause

- As method2 is throwing the exception and method1 is invoking method
- So method1 either can handles the coming exception or rethrows it
- If method1 is handling the exception than method1 would be modified as

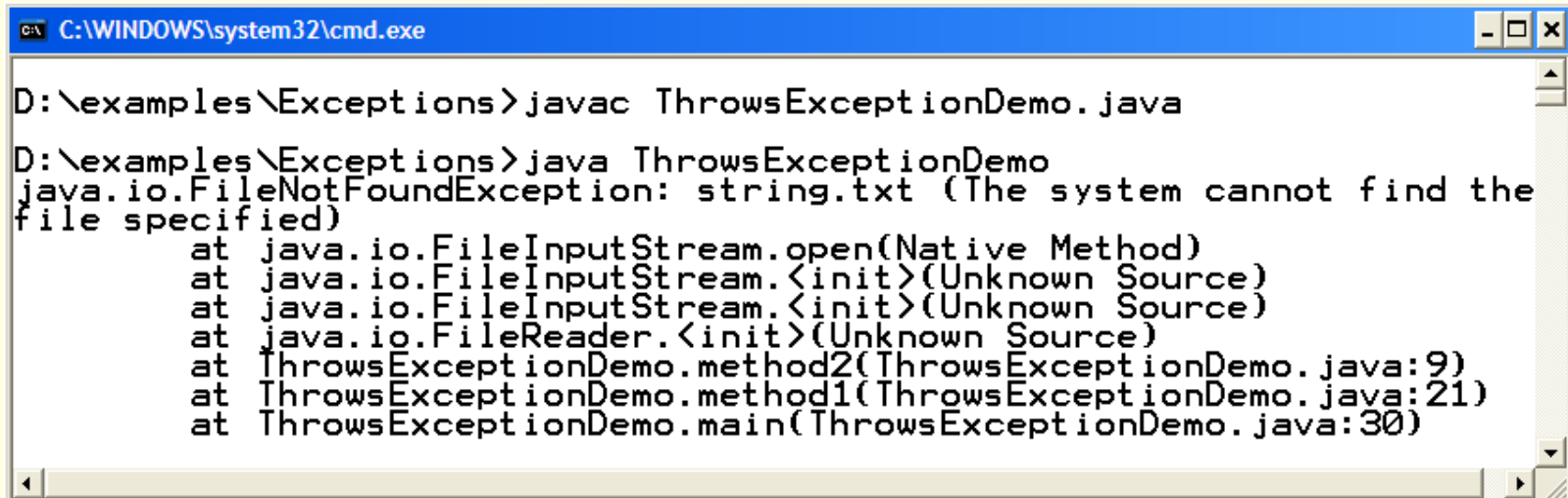
```
// used to call method1
public static void method1 () {
    try {
        method2();
    } catch (IOException ioEx) {
        ioEx.printStackTrace();
    }
}

public static void main(String args[]){
    ThrowsDemo.method1();
}
```

```
}//end of class
```


Compile & Execute

If “string.txt” isn’t there, it will throw FileNotFoundException



```
C:\WINDOWS\system32\cmd.exe

D:\examples\Exceptions>javac ThrowsExceptionDemo.java

D:\examples\Exceptions>java ThrowsExceptionDemo
java.io.FileNotFoundException: string.txt (The system cannot find the
file specified)
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.<init>(Unknown Source)
    at java.io.FileInputStream.<init>(Unknown Source)
    at java.io.FileReader.<init>(Unknown Source)
    at ThrowsExceptionDemo.method2(ThrowsExceptionDemo.java:9)
    at ThrowsExceptionDemo.method1(ThrowsExceptionDemo.java:21)
    at ThrowsExceptionDemo.main(ThrowsExceptionDemo.java:30)
```

If “string.txt” exist there and contains string



```
C:\WINDOWS\system32\cmd.exe

D:\examples\Exceptions>java ThrowsExceptionDemo
hello world

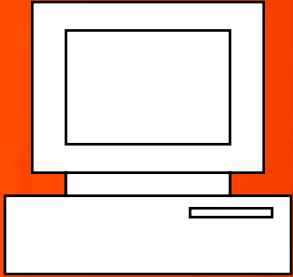
D:\examples\Exceptions>
```

Web Design & Development

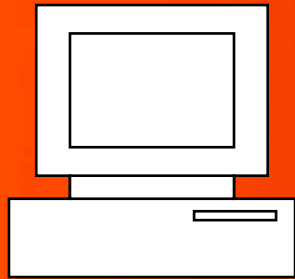
Lecture 8

Streams

Why Streams?



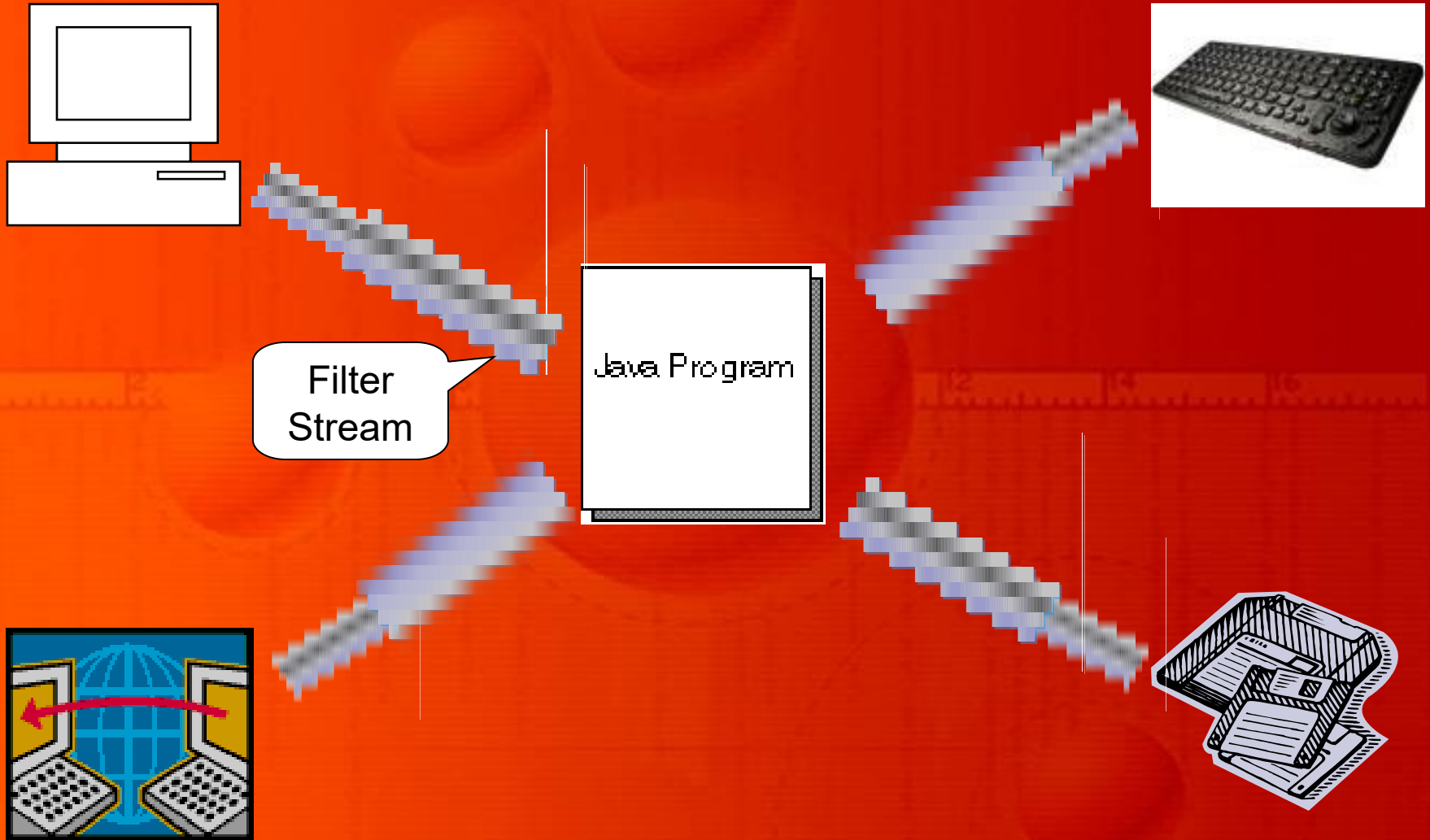
Node Streams



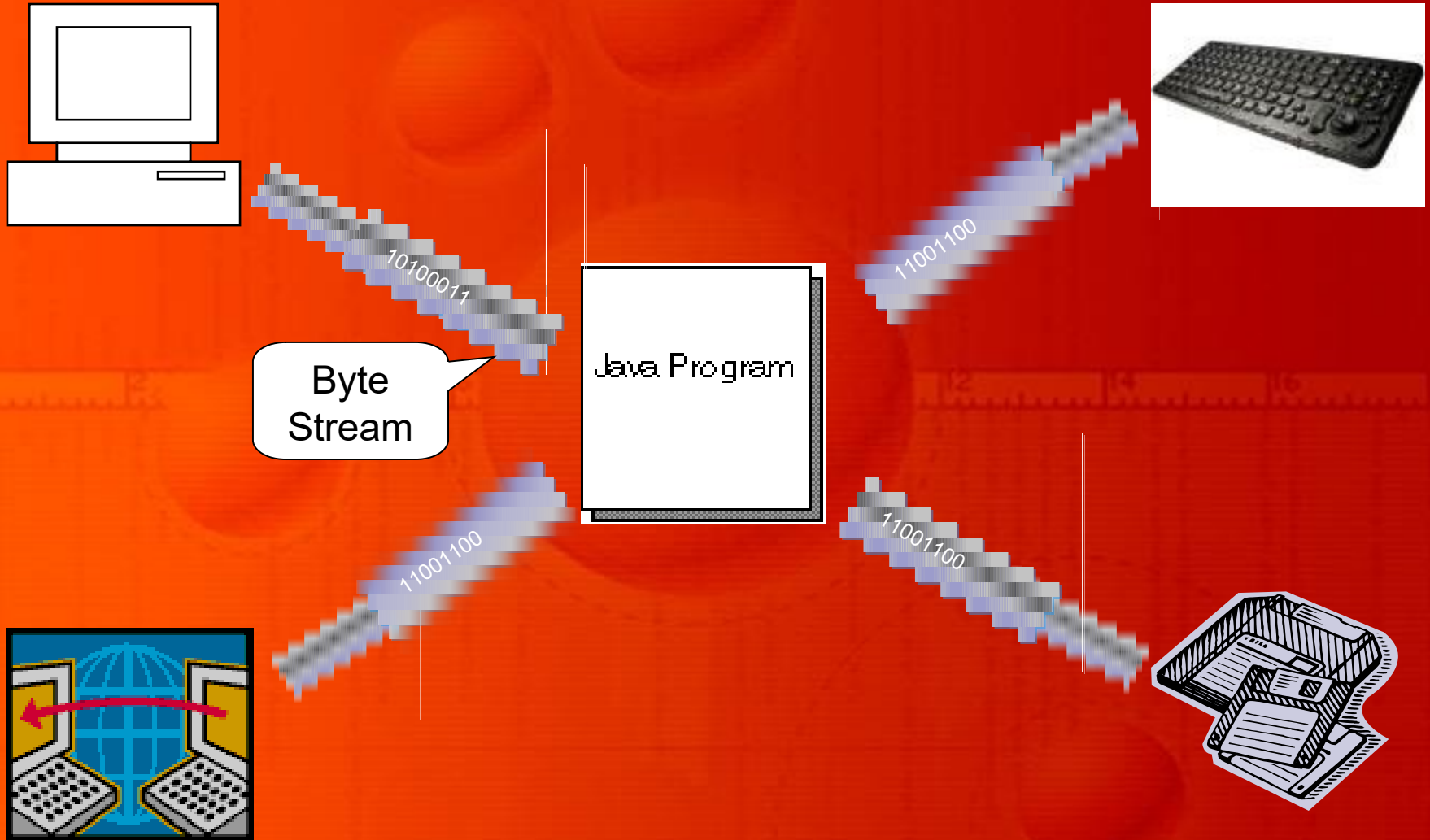
Node
Stream



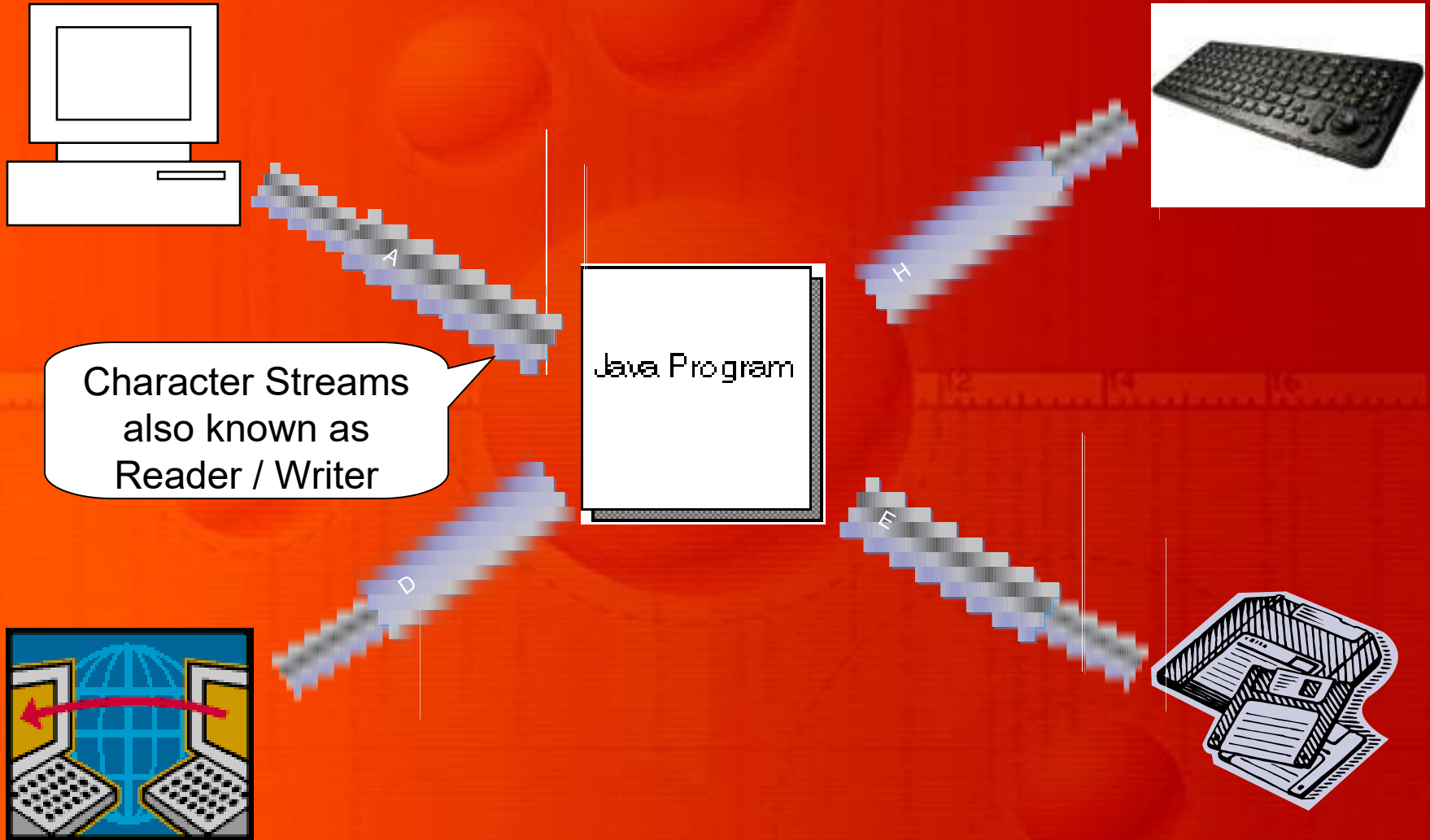
Filter Streams



Filter Streams



Filter Streams



Character vs. Byte Streams

- A *character stream* manages 16-bit Unicode characters
- A *byte stream* manages 8-bit bytes of raw binary data
 - A program must determine how to interpret and use the bytes in a byte stream
 - Typically they are used to read and write sounds and images
- The `InputStream` and `OutputStream` classes (and their descendants) represent byte streams
- The `Reader` and `Writer` classes (and their descendants) represent character streams

Types of Streams

- InputStream / OutputStream
 - Base class streams with few features
 - read() and write()
- FileInputStream / FileOutputStream
 - Specifically for connecting to files
- ByteArrayInputStream / ByteArrayOutputStream
 - Use an in-memory array of bytes for storage!
- BufferedInputStream / BufferedOutputStream
 - Improve performance by adding buffers
 - Should almost always use buffers
- BufferedReader / BufferedWriter
 - Convert bytes to unicode Char and String data
 - Probably most useful for what we need

Reading from File

Reading From File

Hello World

Pakistan is our homeland

Web Design and Development

input.txt

ReadFile. java

```
/* This program reads file line by line and  
   prints them on to the console */  
import java.io.*;  
public class ReadFile {  
    public static void main (String args[ ]) {  
        FileReader fr    = null;  
        BufferedReader br = null;  
        try {  
            fr = new FileReader("input.txt");  
            br = new BufferedReader(fr);  
  
            //continue
```

ReadFile. java

```
String s = br.readLine();
```

```
while (s != null) {
```

```
    System.out.println(s);
```

```
    s = br.readLine();
```

```
}
```

```
br.close();
```

```
fr.close();
```

```
} catch (IOException ex) {
```

```
    System.out.println(ex);
```

```
}
```

```
}// end of main
```

```
//end of class
```

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\io>javac ReadFile.java

D:\examples\io>java ReadFile
Hello World
Pakistan is our homeland
Web Design and Development
```

The image shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window contains the following text:

```
D:\examples\io>javac ReadFile.java

D:\examples\io>java ReadFile
Hello World
Pakistan is our homeland
Web Design and Development
```

The output of the Java program is displayed on four lines: "Hello World", "Pakistan is our homeland", and "Web Design and Development". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. A scrollbar is visible on the right side of the command prompt area.

Writing into File

WriteFile. java

```
// This program writes the strings into the file "output.txt"
import java.io.*;
public class WriteFile {
    public static void main (String args[ ]) {

        FileWriter fw    = null;
        PrintWriter pw = null;
        try {
            // if file does not exist, it automatically create for you
            fw = new FileWriter("output.txt");

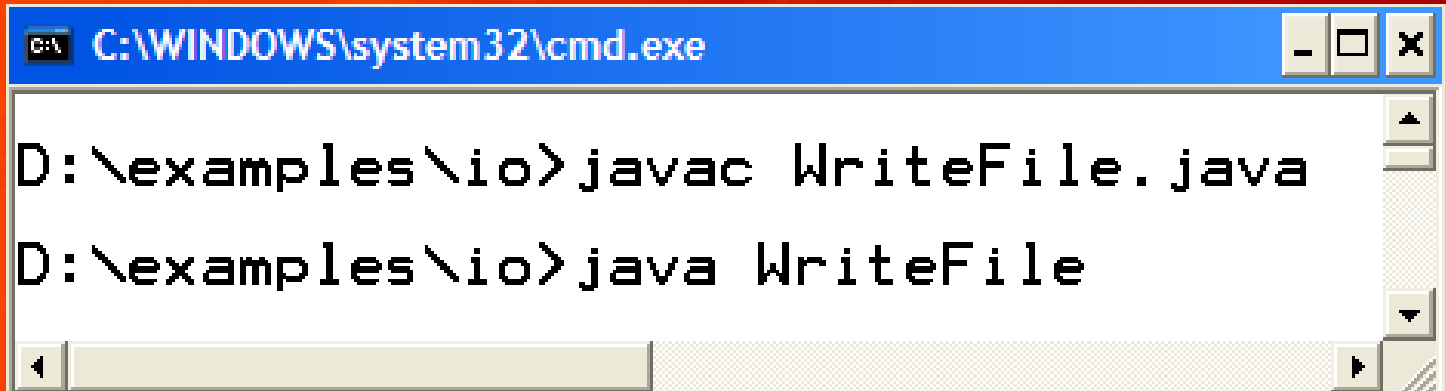
            pw = new PrintWriter(fw);

            //continue
```

WriteFile. java

```
String s1 = "Hello World";  
String s2 = "Web Design and Development";  
  
pw.println(s1);  
pw.println(s2);  
  
pw.flush();  
pw.close();  
fw.close();  
  
} catch (IOException ex) {  
    System.out.println(ex);  
}  
  
} //end of main  
} //end of class
```

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\io>javac WriteFile.java
D:\examples\io>java WriteFile
```

The image shows a Windows command prompt window with a blue title bar that reads "C:\WINDOWS\system32\cmd.exe". The window contains two lines of text: "D:\examples\io>javac WriteFile.java" and "D:\examples\io>java WriteFile". The text is in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

Hello World

Web Design and Development

output.txt

Why Streams

Reading Data from

- **File**

```
FileReader fr = new FileReader("input.txt");  
BufferedReader br = new BufferedReader (fr);
```

- **Console**

```
FileReader fr = new FileReader(FileDescriptor.in);  
BufferedReader br = new BufferedReader (fr);
```

- **Network**

```
InputStream is = s.getInputStream() ;  
BufferedInputStream br = new BufferedInputStream (is) ;
```

//where "s" is the socket

Writing Data to

- **File**

```
FileWriter fw = new FileWriter("output.txt");
```

```
PrintWriter pw = new PrintWriter (pw);
```

- **Console**

```
FileWriter fw = new FileWriter(FileDescriptor.out);
```

```
PrintWriter pw = new PrintWriter (fw);
```

- **Network**

```
OutputStream os = s.getOutputStream() ;
```

```
PrintWriter pw = new PrintWrtier (os) ;
```

//where "s" is the socket

Problem

- Take input from console
- Write that input to text file with line numbers
- Program should only exit when user press enters without typing anything

Web Design & Development

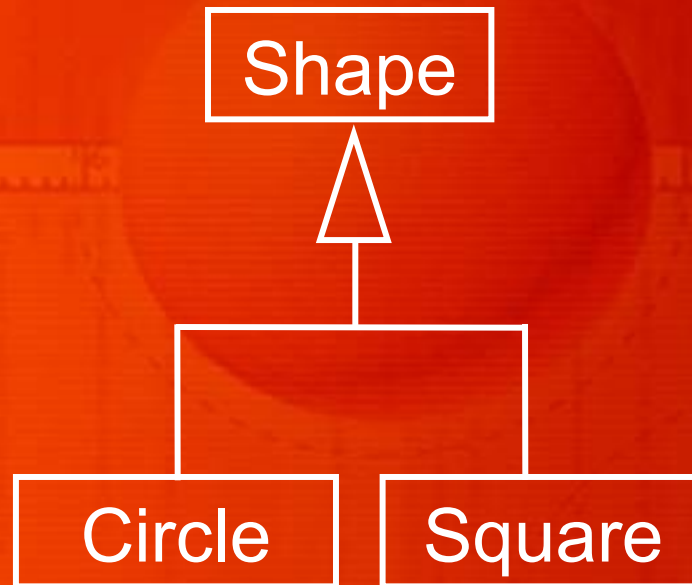
Lecture 9

Abstract Classes a and Interfaces

Abstract Classes

Class Shape Hierarchy

- Consider the following class hierarchy



Problem AND Requirements

- Suppose that in order to exploit polymorphism, we specify that 2-D objects must be able to compute their area.
 - All 2-D classes must respond to area() message.
- How do we ensure that?
 - Define area method in class Shape
 - Force the subclasses of Shape to respond area() message
- Java's Solutions
 - Abstract Classes
 - Interfaces

Abstract Classes

- Idea
 - To define only part of an implementation
 - Can contain instance variables & methods that are fully implemented
 - Leaving the subclasses to provide the details
- Any class with an abstract method must be declared abstract
 - However you can declare a class abstract that has no abstract method.
 - An abstract method has no implementation (known in C++ as a pure virtual function)

Abstract Classes

- If subclass overrides all abstract methods of the superclass, than it becomes a **concrete** class otherwise we have to declare it as abstract or we can not compile it
- Any subclass can override a concrete method inherited from the superclass and declare them abstract
- An abstract class cannot be instantiated
- However references to an abstract class can be declared
 - Can point to the objects of concrete subclasses

Example of abstract class

Shape.java

```
/* This is an example of abstract class. Note that  
this class contains an abstract method with no  
definition.
```

```
*/
```

```
public abstract class Shape {  
    public abstract void calculateArea();  
}
```

Circle.java

```
/* This class extends from abstract Shape class. Therefore to  
become concrete class it must provides the definition of  
calculateArea method.
```

```
*/
```

```
public class Circle extends Shape {
```

```
    private int x, y;
```

```
    private int radius;
```

```
    public Circle() {
```

```
        x = 5;
```

```
        y = 5;
```

```
        radius = 10;
```

```
    }
```

```
// continue
```

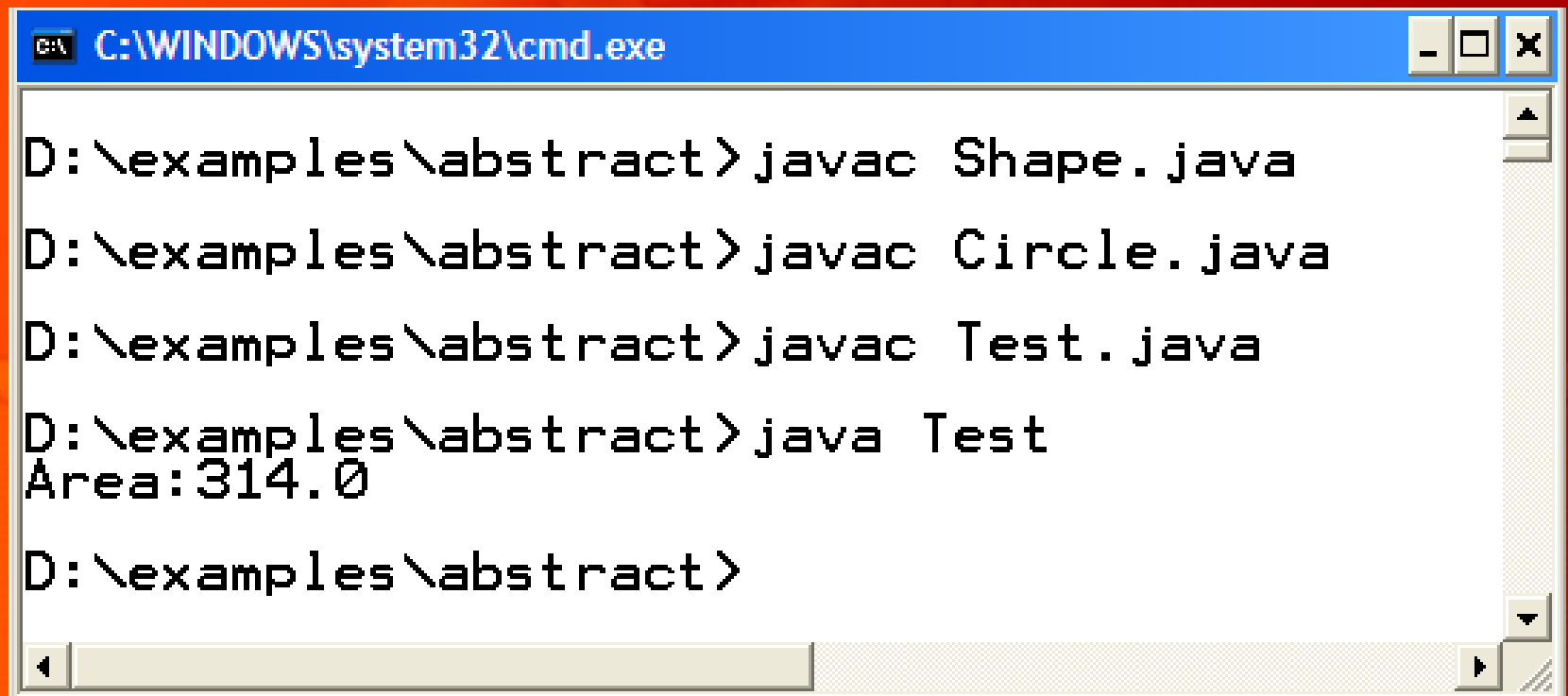

Circle.java

```
// providing definition of abstract method  
public void calculateArea () {  
    double area = 3.14 * (radius * radius);  
    System.out.println("Area: " + area);  
}  
  
} //end of class
```

Test.java (Driver class)

```
public class Test {  
    public static void main (String args[ ]){  
        //can only create references of abstract class  
        Shape s = null;  
        // Shape s1 = new Shape(); //cannot instantiate abstract class  
  
        //can point to the concrete subclass  
        s = new Circle();  
        s.calculateArea();  
    }  
}
```

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe
D:\examples\abstract>javac Shape.java
D:\examples\abstract>javac Circle.java
D:\examples\abstract>javac Test.java
D:\examples\abstract>java Test
Area:314.0
D:\examples\abstract>
```

Interfaces

Interfaces

- A special java type which
 - Defines a set of method prototypes, but does not provide the implementation for the prototypes
 - Essentially all the methods inside an interface are Abstract Methods or we can say that an interface is like a pure abstract class (Zero Implementation)
 - Can also define static final constants

Interfaces Definition Example

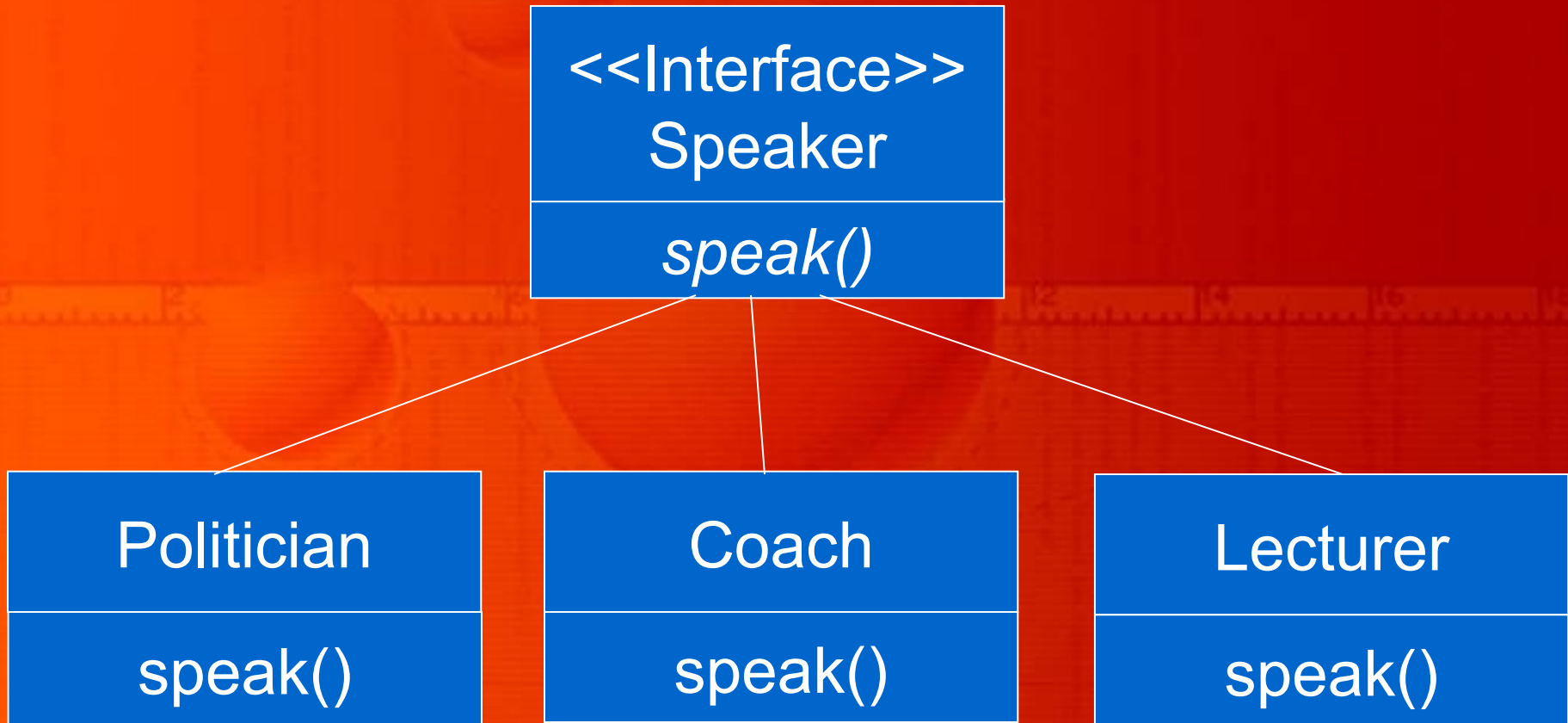
- Syntax (appears like abstract class):
- All methods are abstract and public by default
- All constants are static and final by default

```
public interface Speaker {  
  
    public void speak( );  
  
}
```

Implementing (Using) Interfaces

- Classes Implement interfaces
 - Implementing an interface is like signing a contract.
 - A class that implements an interface will have to provide the definition of all the methods that are present inside an interface“
 - If the class does not provide definitions of all methods, the class would not compile. We have to declare it as an abstract class in order to get it compiled.
- “Responds to” relationship
 - Relationship between a class and interface

Interface - Example



Implementing Interfaces Example

```
class Politician implements Speaker {  
    public void speak(){  
        System.out.println("Talk politics");  
    }  
}
```

```
class Coach implements Speaker {  
    public void speak(){  
        System.out.println("Sports Talks");  
    }  
}
```

```
class Lecturer implements Speaker {  
    public void speak(){  
        System.out.println("Web Desing and Development Talks");  
    }  
}
```

Example Code

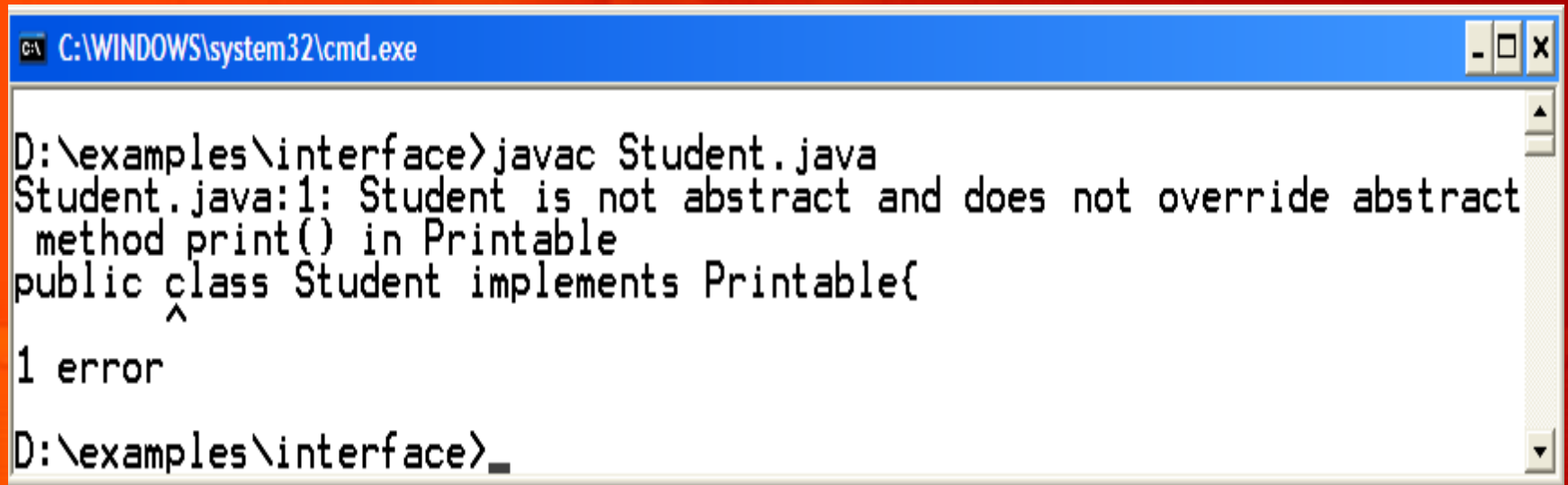
Defining Interface

```
public interface Printable {  
    public void print();  
}
```

Implementing Interface

```
public class Student implements Printable{  
    private String name;  
    private String address;  
  
    public String toString () {  
        return "name:"+name +" address:"+address;  
    }  
    // NOT providing implementation of print method  
}
```

Compile



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" and standard window control buttons (minimize, maximize, close). The main area is white and contains the following text: "D:\examples\interface>javac Student.java", "Student.java:1: Student is not abstract and does not override abstract", "method print() in Printable", "public class Student implements Printable{", "1 error", and "D:\examples\interface>_". A small caret symbol (^) is positioned under the 'c' in "class" on the line "public class Student implements Printable{".

```
C:\WINDOWS\system32\cmd.exe

D:\examples\interface>javac Student.java
Student.java:1: Student is not abstract and does not override abstract
method print() in Printable
public class Student implements Printable{
      ^
1 error

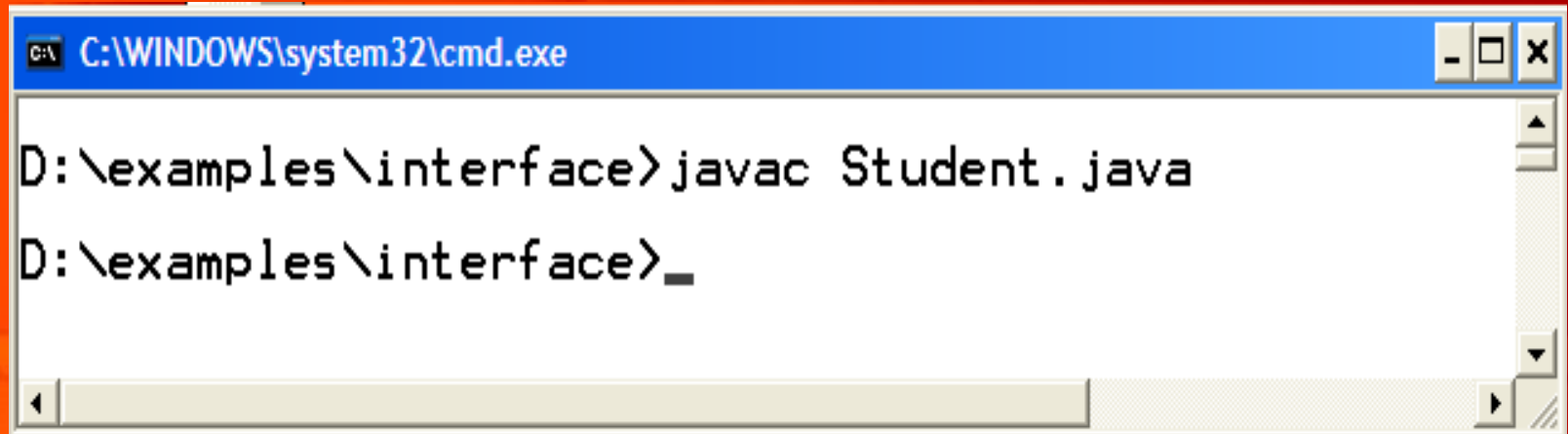
D:\examples\interface>_
```

Example Code (cont.)

Implementing Interface (Modification)

```
public class Student implements Printable{  
    private String name;  
    private String address;  
  
    public String toString () {  
        return "name:"+name +" address:"+address;  
    }  
  
    public void print() {  
        System.out.println("Name:" +name+" address"+address);  
    }  
}
```

Compile



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" and standard window control buttons (minimize, maximize, close). The main area is white and contains the text "D:\examples\interface>javac Student.java" on the first line and "D:\examples\interface>_" on the second line. A horizontal scrollbar is visible at the bottom of the window.

```
C:\WINDOWS\system32\cmd.exe
D:\examples\interface>javac Student.java
D:\examples\interface>_
```

More on Interfaces

- Interface imposes a **design structure** on any class that uses the interface
- Leaves the implementation details to the implementing class and hides that implementation from the client.
- A class can implement more than one interfaces. Java's way of multiple inheritance

```
class Circle implements Drawable, Printable {  
    //additional constants and abstract methods  
}
```

More on Interfaces (cont.)

- Classes inherit from classes (Single), interfaces inherit from interfaces (Can be multiple) and classes implement interfaces (Can be multiple)

```
public interface Displayable extends Drawable, Printable
{
    //additional constants and abstract methods
}
```

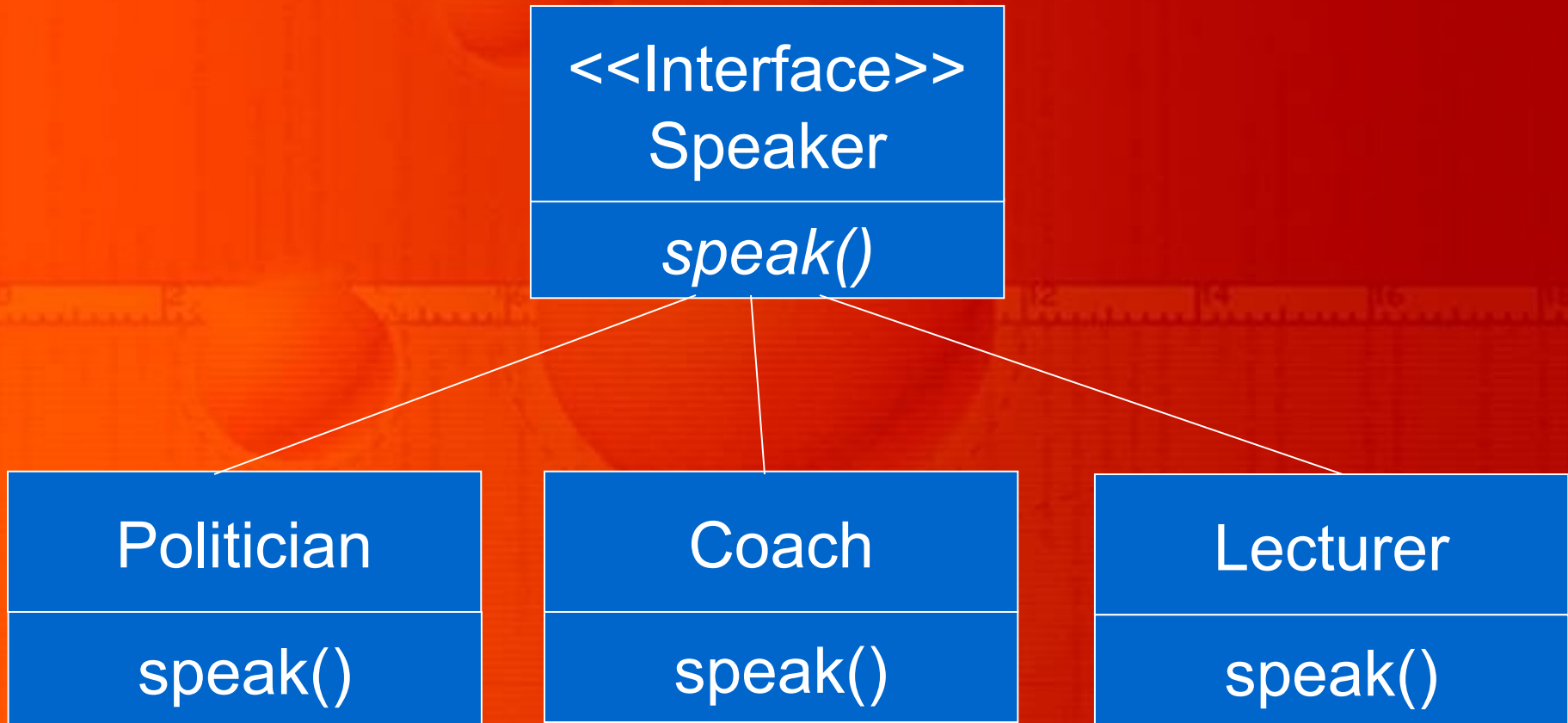
- Objects of interfaces cannot be instantiated.

```
Speaker sp = new Speaker();    // not comaile
```

- However a reference of interface can be created to point to any of its implementation class (Interface based polymorphism).

Interface based Polymorphism

Review again – Interface Example



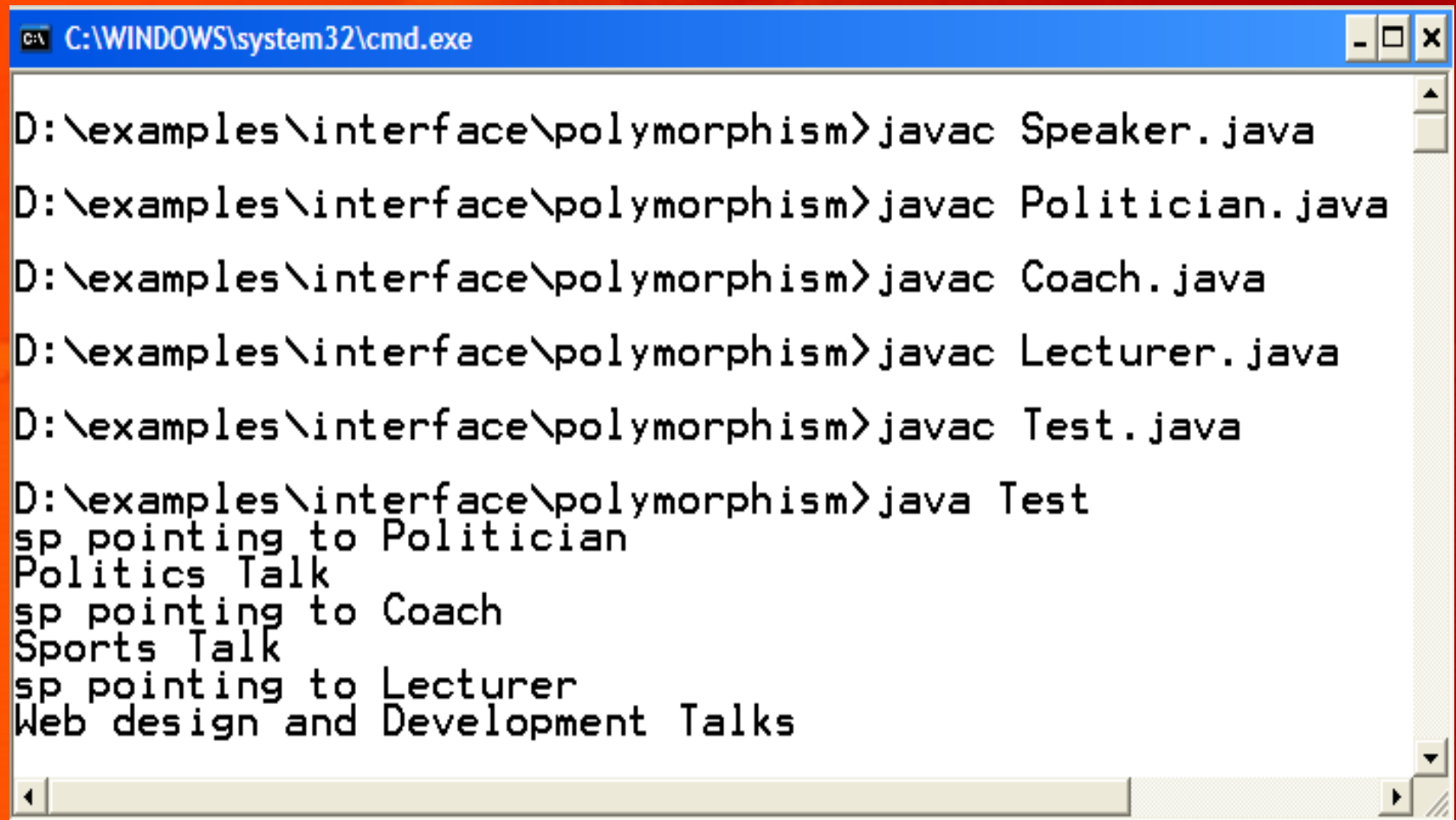
Example: Interface based Polymorphism

```
/* Speaker interface is implemented by the Politician, Coach and  
Lecturer class.  */
```

```
public class Test{  
    public static void main (String args[ ]) {  
  
        Speaker sp = null;  
  
        System.out.println("sp pointing to Politician");  
        sp = new Politician();  
        sp.speak();  
  
        System.out.println("sp pointing to Coach");  
        sp = new Coach();  
        sp.speak();  
  
        System.out.println("sp pointing to Lecturer");  
        sp = new Lecturer();  
        sp.speak();  
    }  
}
```

Interface based Polymorphism

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe

D:\examples\interface\polymorphism>javac Speaker.java
D:\examples\interface\polymorphism>javac Politician.java
D:\examples\interface\polymorphism>javac Coach.java
D:\examples\interface\polymorphism>javac Lecturer.java
D:\examples\interface\polymorphism>javac Test.java
D:\examples\interface\polymorphism>java Test
sp pointing to Politician
Politics Talk
sp pointing to Coach
Sports Talk
sp pointing to Lecturer
Web design and Development Talks
```

Interfaces vs. Abstract classes

- Fairly similar uses
 - designed to group behavior, allow upcasting, exploit polymorphism
- Rules of thumb
 - Choose abstract class if we have shared code and logical “is a” relationship
 - Choose interface if only want to ensure design structure (method signatures) and/or it is not **logical** to use “is a” relationship.

Web Design & Development

Lecture 10

Basics of GUI

GUI ?

File Look & Feel Themes

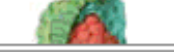
Table Demo Source Code

☒ Reordering allowed
☒ Horiz. Lines
☒ Vert. Lines
Inter-cell spacing:

☐ Column selection
☒ Row selection
Row height:

Selection mode
Multiple ranges

Autoresize mode
Subsequent columns

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Demon	3	
Brian	Beck	Black	The Blues Brothers	2.718	
Lara	Bunni	Red	Airplane (the whole series)	15	
Roger	Brinkley	Blue	The Man Who Knew Too Much	13	
Brent	Christian	Black	Blade Runner (Director's Cut)	23	
Mark	Davidson	Dark Green	Brazil	27	
Jeff	Dinkins	Blue	The Lady Vanishes	8	
Ewan	Dinkins	Yellow	A Bug's Life	2	
Amy	Fowler	Violet	Reservoir Dogs	3	

Support for GUI in Java

Support for GUI

- Abstract Windowing Toolkit (AWT) & Swing packages
 - Provides rich set of user interface components
 - java.awt & javax.swing
 - Old (AWT) VS. New(Swing)
- Components in awt & swing (start with J)
 - Frame, JFrame
 - Menu, JMenu
 - Button, JButton
 - TextField, JTextFiled
 - Label, JLabel
 - and many more....
- Use Java API Documentation well, its your FRIEND.

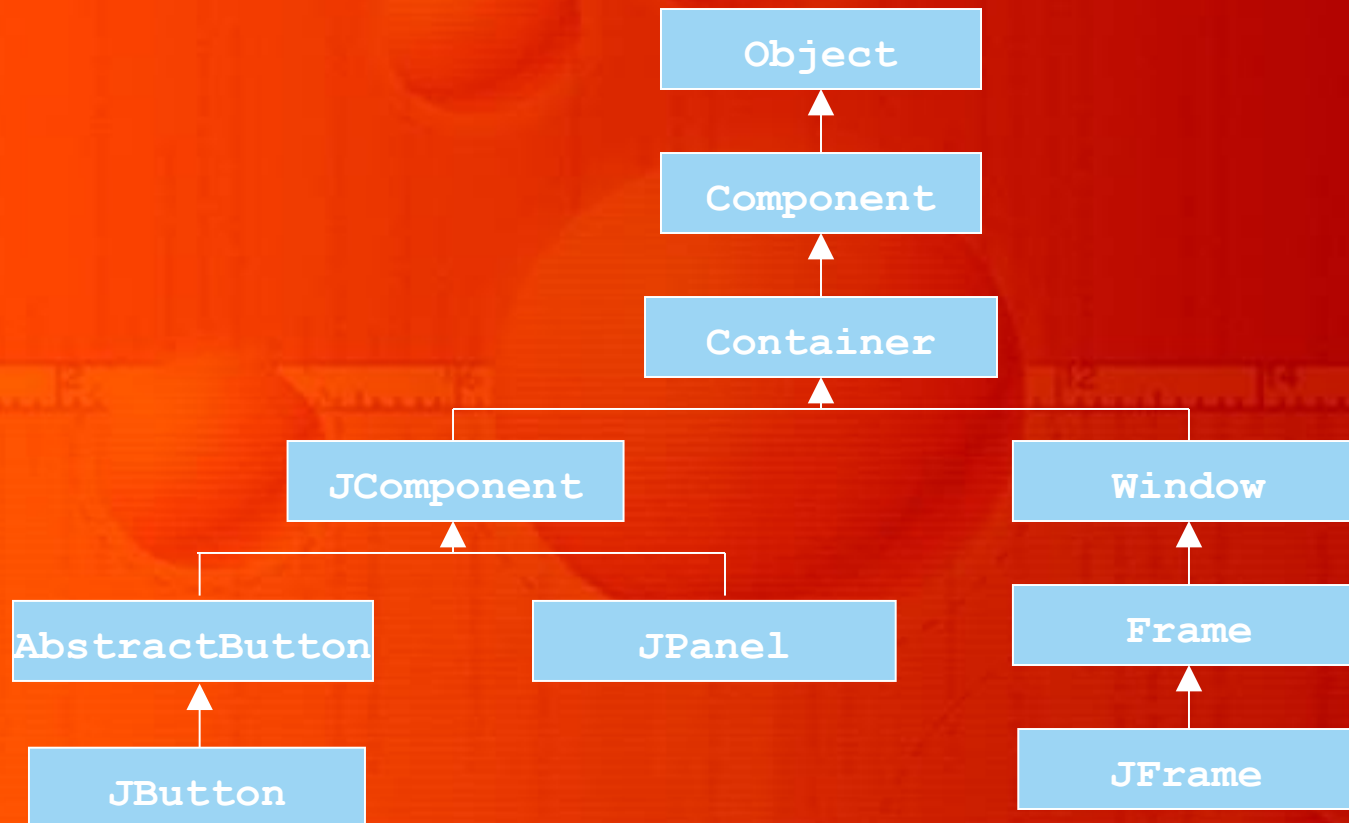
Abstract Windowing Toolkit

- AWT
 - The original GUI components
 - Referred as “Heavy Weight Components (HWC)”
 - Tied directly to the local platform’s GUI capabilities
 - Provides
 - robust event-handling model
 - Layout Managers

Swing

- Swing
 - Newest GUI components, Names start with J can be identified
 - “replacement” to the AWT
 - Referred as “Light Weight Components (LWC)”
 - Swing components are written, manipulated and displayed completely in java
 - So they are not “weighed down” by the GUI capabilities of local platform
 - Several Swing components are still HWC like JFrame etc.
 - Allows uniform “look & feel” across all platforms

A Part of the Framework



GUI Creation Steps

GUI Creation Steps

1. import required package

- e.g. swing, awt

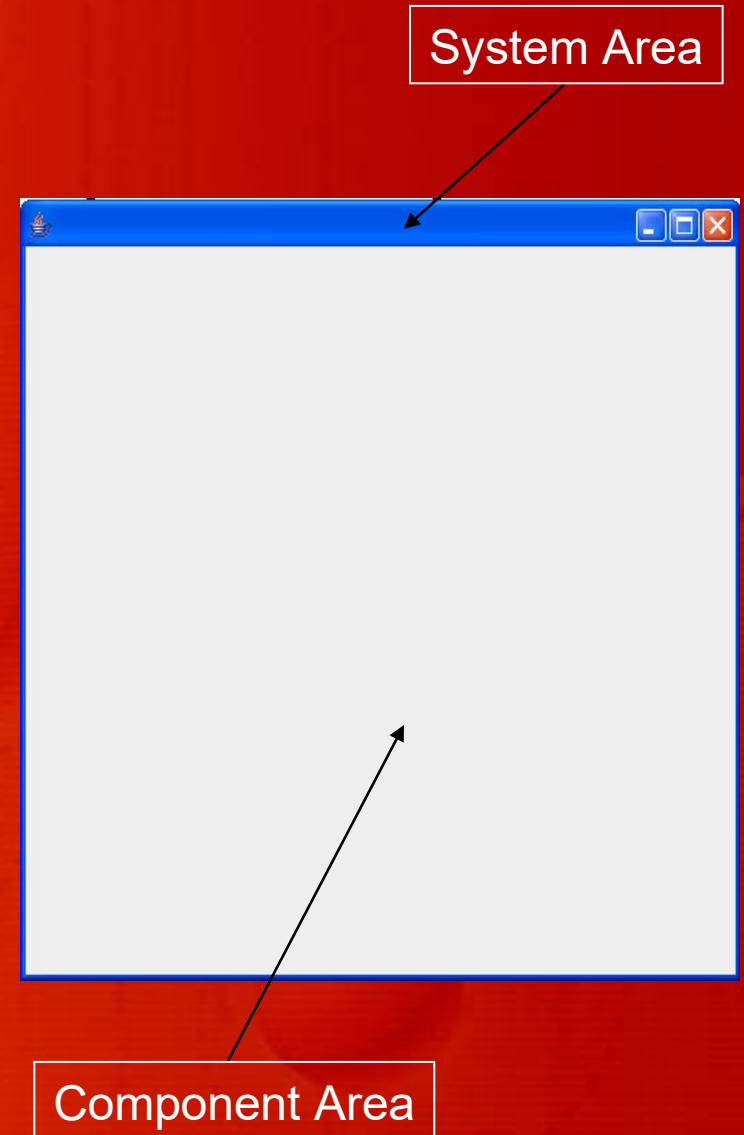
2. Setup the top level container

- e.g. `JFrame myframe = new JFrame();`

GUI Creation Steps (cont.)

3. Get the component Area of the top level Container

```
Container c = myFrame.getContentPane();
```



GUI Creation Steps (cont.)

4. Apply layout to that Area

- `c.setLayout(new FlowLayout());`

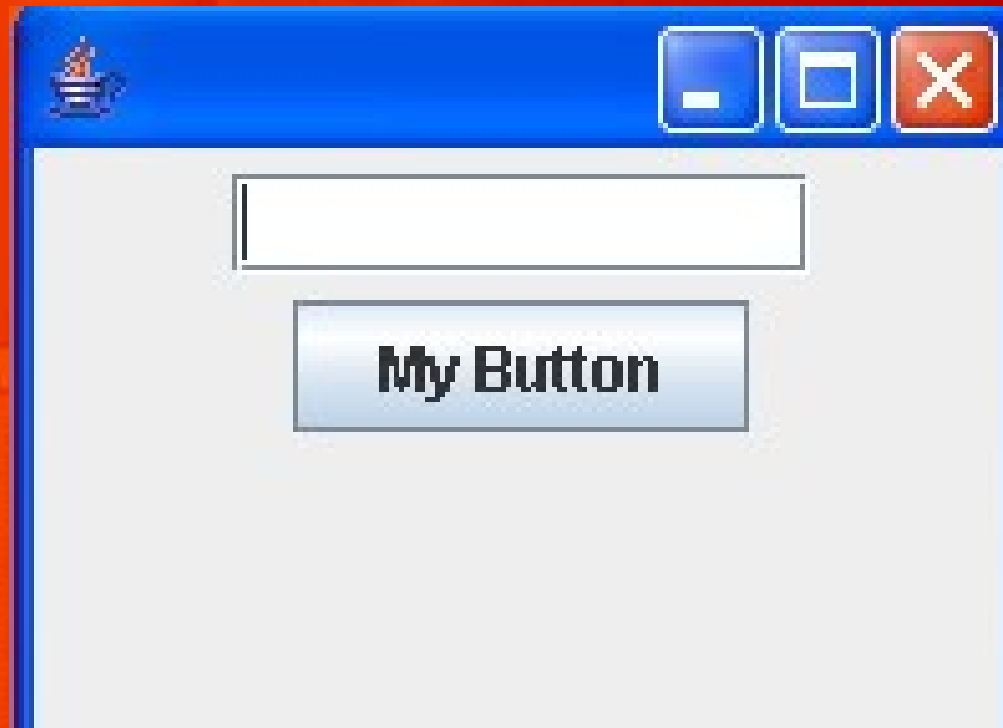
5. Create & add components

- `JButton b1 = new JButton("Hello");`
- `c.add(b1);`

6. Set size of Frame and make it Visible

- `myFrame.setSize(200,200);`
- `myFrame.setVisible(true);`

Example GUI



GUI: Example Code

//Step 1: import packages

```
import java.awt.*;  
import javax.swing.*;
```

```
public class GUITest {  
    JFrame myFrame ;  
    JTextField tf;  
    JButton b1;
```

```
public void initGUI ( ) { //method used for setting layout of GUI
```

//Step 2: setup the top level container

```
    myFrame = new JFrame();
```

//Step 3: Get the component area of top-level container

```
    Container c = myFrame.getContentPane();
```

//Step 4: Apply layouts

```
    c.setLayout( new FlowLayout( ) );
```

```
    ....
```

GUI: Example Code (cont.)

```
//Step 5: create & add components
```

```
JTextField tf = new JTextField(10);
```

```
JButton b1 = new JButton("My Button");
```

```
c.add(tf);
```

```
c.add(b1);
```

```
//Step 6: set size of frame and make it visible
```

```
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
myFrame.setSize(200,150);
```

```
myFrame.setVisible(true);
```

```
} //end init method
```

```
public GUITest () { // constructor
```

```
    initGUI ();
```

```
}
```

```
.....
```

GUI: Example Code (cont.)

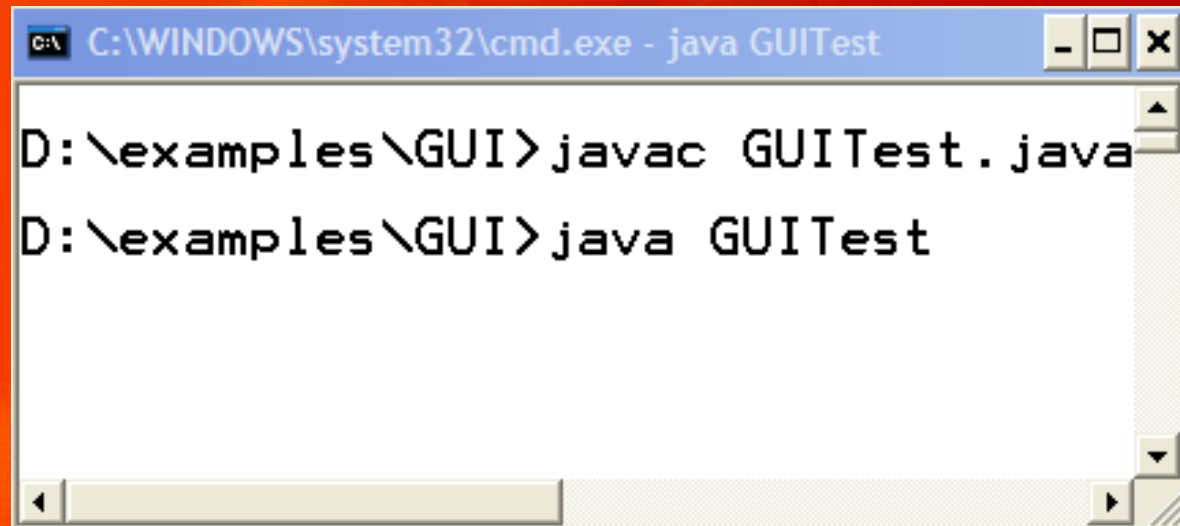
```
public static void main (String args[ ]) {
```

```
    GUITest gui = new GUITest();
```

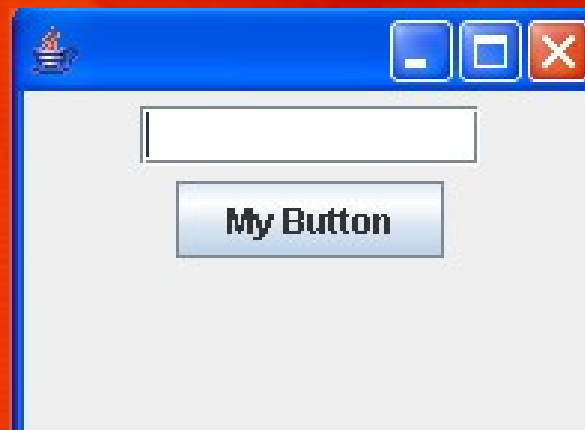
```
}
```

```
} // end of class
```

Compile & Execute



```
C:\WINDOWS\system32\cmd.exe - java GUITest  
D:\examples\GUI>javac GUITest.java  
D:\examples\GUI>java GUITest
```



GUI Creation Approaches

Composition

```
class GUITest{  
    JFrame frame;  
    Container c;  
    public GUITest ( ) {  
        frame = new JFrame ( );  
        c = frame.getContentPane();  
        .....  
        frame.setVisible(true);  
    }  
    .....  
}
```

Inheritance

```
class GUITest extends JFrame{  
    Container c;  
    public GUITest ( ) {  
        c = getContentPane();  
        .....  
        setVisible(true);  
    }  
    .....  
}
```

Layout Managers

Layout Managers

- The layout of components in a container is usually governed by layout managers
- Similar to HTML – policy, not position
 - Do not set explicit pixel sizes or positions of things
 - Layout Managers know the intent (policy)
 - Layout Managers apply the intent to figure out the correct size on the fly

Layout Managers

- Layout Managers
 - Java supplies many layout managers. Five commonly used are:
 - `FlowLayout`
 - `GridLayout`
 - `BorderLayout`
 - `BoxLayout`
 - `GridBagLayout`

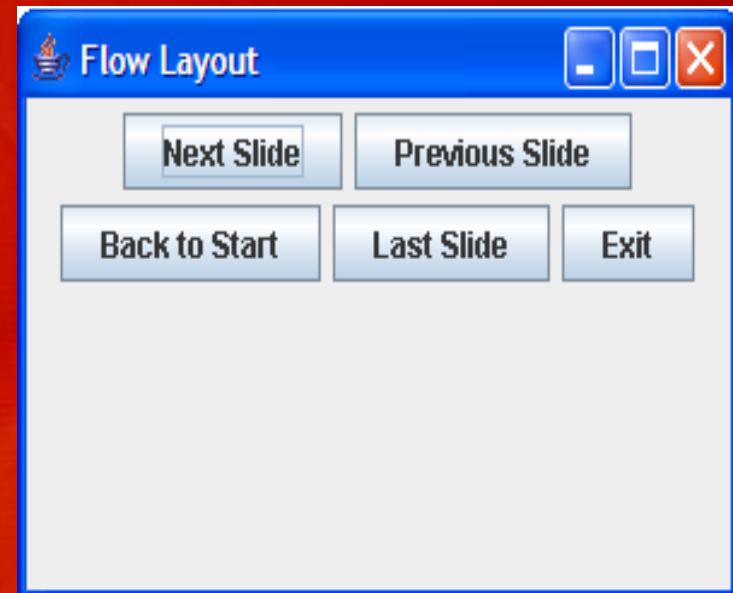
Layout Managers

- Layout Managers
 - FlowLayout
 - Places components in a line as long as they fit, then starts the next line.
 - Uses “best judgement” in spacing components. Centers by default.
 - Lets each component assume its natural (preferred) size.
 - Often used for placing buttons on panels.

GUI: Example Code

FlowLayout

```
....  
c.setLayout (new FlowLayout() );  
  
JButton b1 = new JButton("Next Slide");  
JButton b2 = new JButton("Previous Slide");  
JButton b3 = new JButton("Back to Start");  
JButton b4 = new JButton("Last Slide");  
JButton b5 = new JButton("Exit");  
  
c.add(b1);  
c.add(b2);  
c.add(b3);  
c.add(b4);  
c.add(b5);  
} //end of main  
}
```



Layout Managers

- Layout Managers
 - GridLayout
 - Splits the panel into a grid with given number of rows and columns.
 - Places components into the grid cells.
 - Forces the size of each component to occupy the whole cell.
 - Allows additional spacing between cells.

GUI: Example Code

GridLayout

```
....  
c.setLayout (new GridLayout(3 , 2) );  
  
JButton b1 = new JButton("Next Slide");  
JButton b2 = new JButton("Previous Slide");  
JButton b3 = new JButton("Back to Start");  
JButton b4 = new JButton("Last Slide");  
JButton b5 = new JButton("Exit");  
  
c.add(b1);  
c.add(b2);  
c.add(b3);  
c.add(b4);  
c.add(b5);  
.....  
}
```



GUI: Example Code

GridLayout

```
....  
c.setLayout (new GridLayout(3 , 2, 10, 20 ) );
```

```
JButton b1 = new JButton("Next Slide");  
JButton b2 = new JButton("Previous Slide");  
JButton b3 = new JButton("Back to Start");  
JButton b4 = new JButton("Last Slide");  
JButton b5 = new JButton("Exit");
```

```
c.add(b1);  
c.add(b2);  
c.add(b3);  
c.add(b4);  
c.add(b5);  
  
size(200, 200)
```

```
//end of main
```

```
}
```

Extra space
between the cells



Layout Managers

- Layout Managers
 - BorderLayout
 - Divides the area into five regions
 - Adds a component to the specified region
 - Forces the size of each component to occupy the whole region.



GUI: Example Code

BorderLayout

```
....  
c.setLayout (new BorderLayout( ) );  
  
JButton b1 = new JButton("Next Slide");  
JButton b2 = new JButton("Previous Slide");  
JButton b3 = new JButton("Back to Start");  
JButton b4 = new JButton("Last Slide");  
JButton b5 = new JButton("Exit");  
  
c.add(b1, BorderLayout.NORTH);  
c.add(b2, BorderLayout.SOUTH);  
c.add(b3, BorderLayout.EAST);  
c.add(b4, BorderLayout.WEST);  
c.add(b5, BorderLayout.CENTER);  
}  
//end of main  
}
```

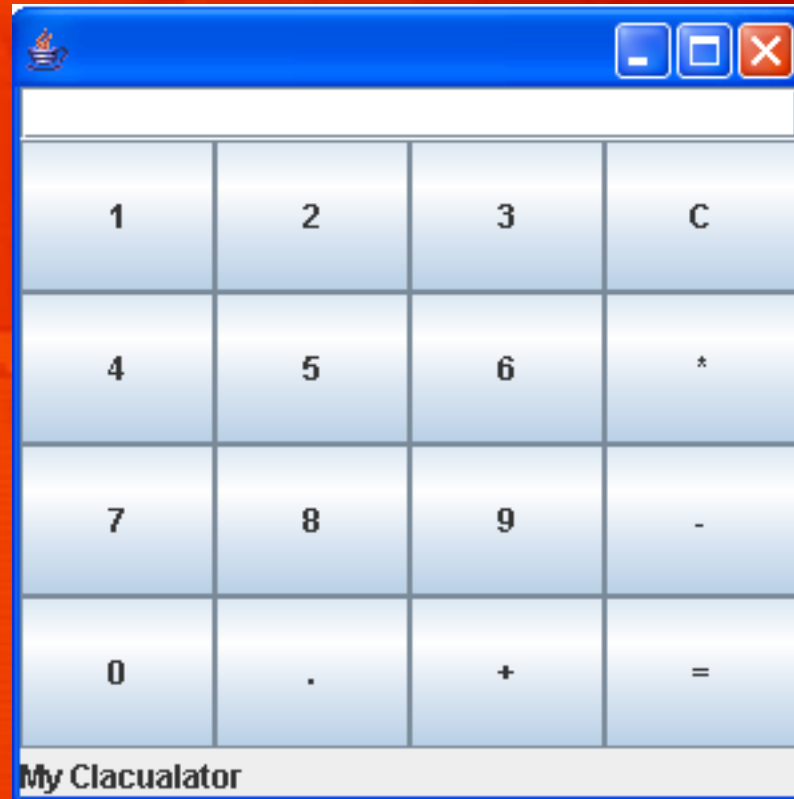


Layout Managers

- Layout Managers
 - Default Layouts
 - Each container has a default layout manager, which remains in effect until the component's `setLayout` method is called.
 - Some of the defaults are:
 - Content pane → BorderLayout
 - JPanel → FlowLayouts

Making Your own Calculator

Calculator GUI



Code: CalculatorGUI

```
import java.awt.*;  
import javax.swing.*;  
  
public class CalculatorGUI {  
  
    JFrame fCalc;  
  
    JButton b1, b2, b3, b4, b5, b6, b7, b8, b9, b0;  
    JButton bPlus, bMinus, bMul, bPoint, bEqual, bClear;  
  
    JPanel pButtons;  
  
    JTextField tfAnswer;  
  
    JLabel lMyCalc;
```

Code: CalculatorGUI (cont.)

```
public void initGUI ( ) { //used for setting layout of calculator
```

```
    fCalc = new JFrame();
```

```
    b0 = new JButton("0");
```

```
    b1 = new JButton("1");
```

```
    b2 = new JButton("2");
```

```
    b3 = new JButton("3");
```

```
    b4 = new JButton("4");
```

```
    b5 = new JButton("5");
```

```
    b6 = new JButton("6");
```

```
    b7 = new JButton("7");
```

```
    b8 = new JButton("8");
```

```
    b9 = new JButton("9");
```

```
    bPlus    = new JButton("+");
```

```
    bMinus   = new JButton("-");
```

```
    bMul      = new JButton("*");
```

```
    bPoint    = new JButton(".");
```

```
    bEqual    = new JButton("=");
```

```
    bClear    = new JButton("C");
```

```
    // continue....
```

Code: CalculatorGUI (cont.)

```
tfAnswer = new JTextField();
```

```
lMyCalc = new JLabel("My Clacualator");
```

```
//creating panel object and setting its layout  
pButtons = new JPanel (new GridLayout(4,4));
```

```
//adding components (buttons) to panel
```

```
pButtons.add(b1);  
pButtons.add(b2);  
pButtons.add(b3);  
pButtons.add(bClear);
```

```
pButtons.add(b4);  
pButtons.add(b5);  
pButtons.add(b6);  
pButtons.add(bMul);
```

```
//continue
```

Code: CalculatorGUI (cont.)

```
pButtons.add(b7);  
pButtons.add(b8);  
pButtons.add(b9);  
pButtons.add(bMinus);
```

```
pButtons.add(b0);  
pButtons.add(bPoint);  
pButtons.add(bPlus);  
pButtons.add(bEqual);
```

```
Container con = fCalc.getContentPane();  
con.setLayout(new BorderLayout());
```

```
//adding components to container  
con.add(tfAnswer, BorderLayout.NORTH);  
con.add(IMyCalc, BorderLayout.SOUTH);  
con.add(pButtons, BorderLayout.CENTER);
```

```
fCalc.setSize(300, 300);  
fCalc.setVisible(true);
```

```
} // end of initGUI method
```

```
.....
```

Code: CalculatorGUI (cont.)

```
//default constructor
```

```
public CalculatorGUI ( ) {  
    initGUI();  
}
```

```
//main method
```

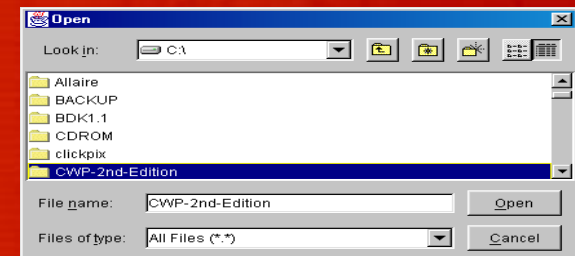
```
public static void main(String args[ ]) {
```

```
    CalculatorGUI calGUI = new CalculatorGUI();  
}
```

```
} //end of class
```

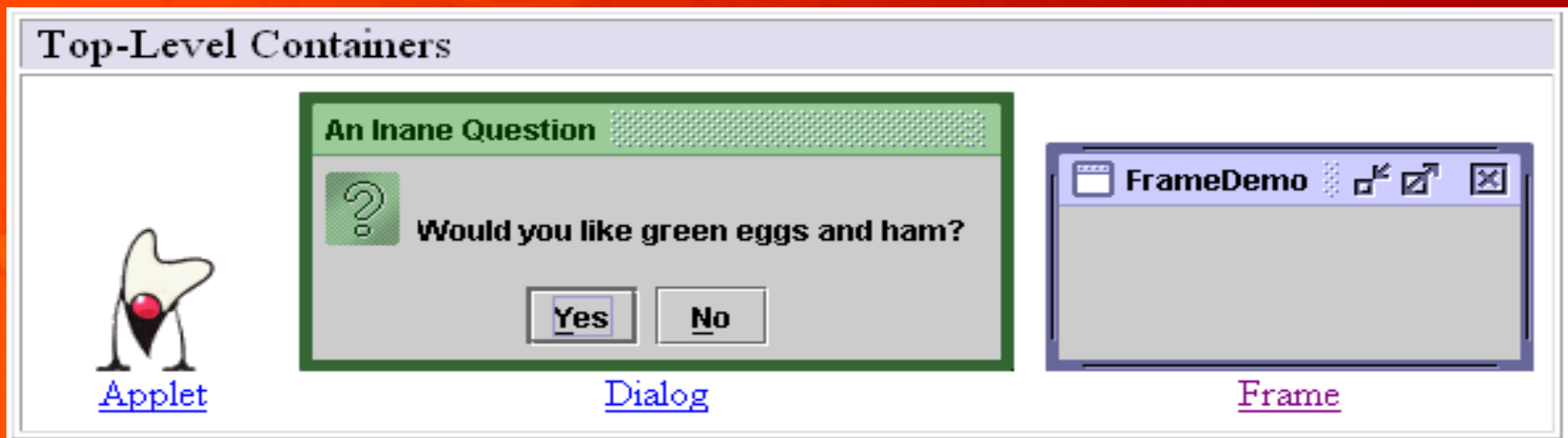

More Swing Components

- JCheckBox
 - Note uppercase B (vs. Checkbox in AWT)
- JRadioButton
 - Use a ButtonGroup to link radio buttons
- JTextArea
 - Place in JScrollPane if you want scrolling
- JFileChooser



Swing Components

Top Level Containers

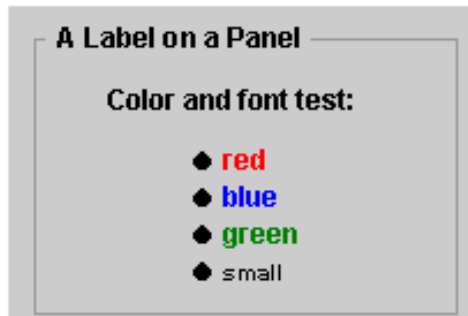


Your application usually extends one of these classes !

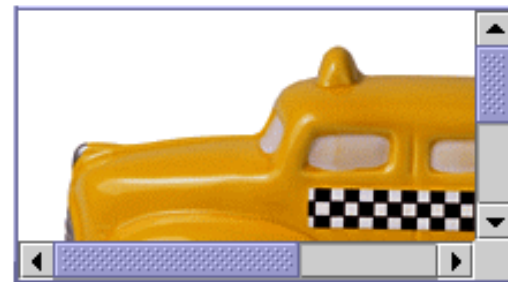
Swing Components

- General Purpose Containers

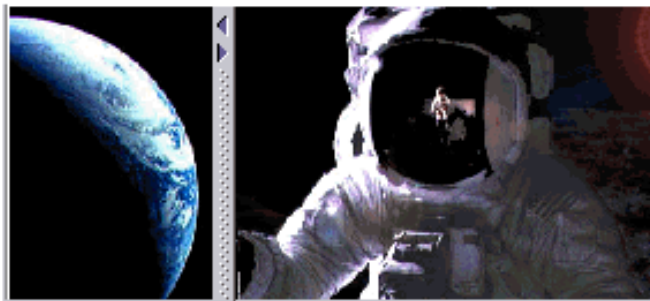
General-Purpose Containers



Panel



Scroll pane



Split pane



Tabbed pane



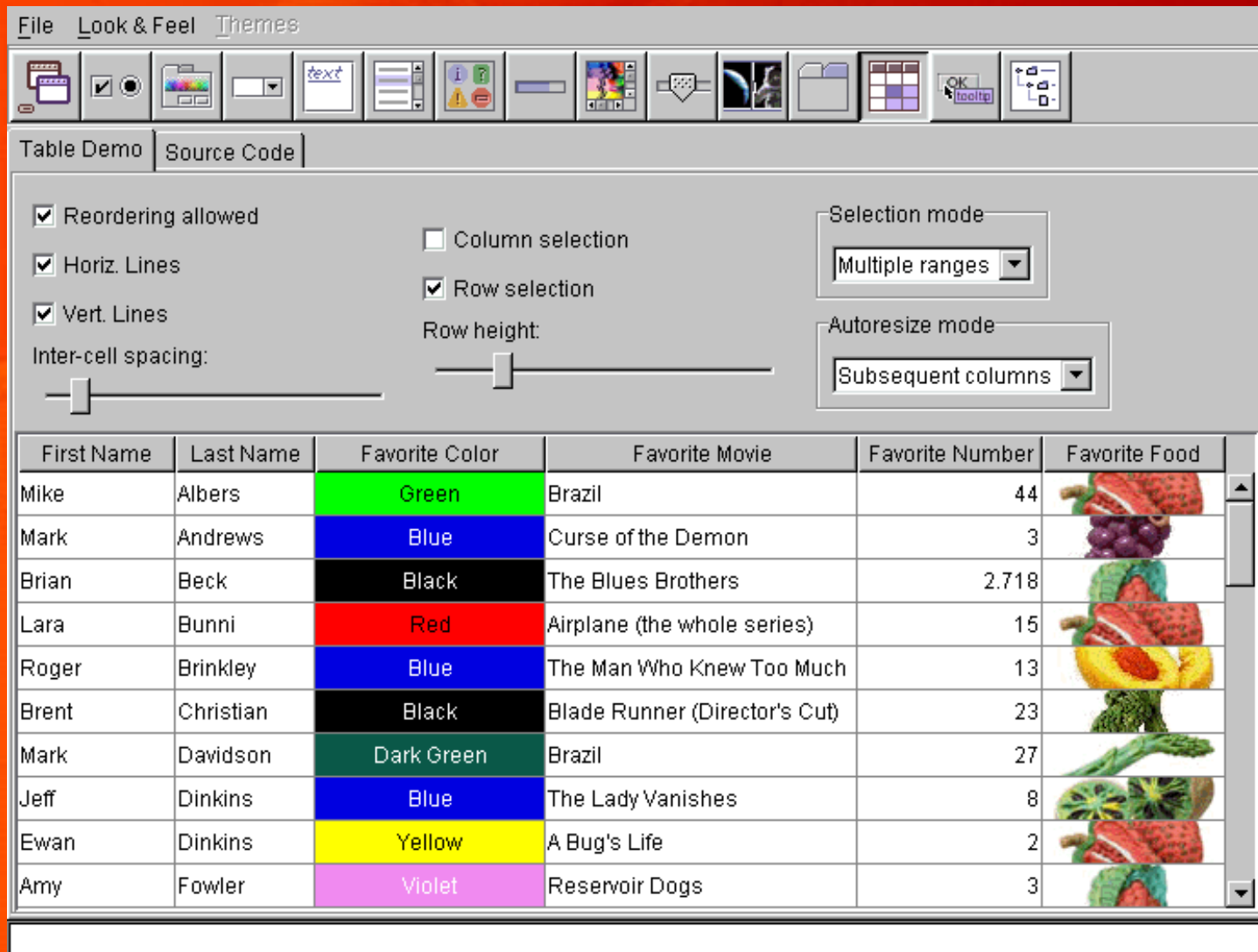
Tool bar

Web Design & Development

Lecture 11

Event Handling

GUI are Event Driven



Events

- GUIs generate events when the user interacts with GUI
- For example,
 - Clicking a button
 - Moving the mouse
 - Closing Window etc
- In java, events are represented by Objects
 - These objects tells us about event and its source. Examples are
 - `ActionEvent` (Clicking a button)
 - `WindowEvent` (Doing something with window e.g. closing , minimizing)
- Both AWT and swing components (not all) generate events
 - `java.awt.event.*`;
 - `javax.swing.event.*`;

```

graph LR
    P[P] --> R[R]
    R --> P
    P --> PR1[P & R]
    R --> PR1
    PR1 --> PR2[P & R]
    PR2 --> PR1
  
```


Event Handling Model

- Common for both AWT and Swing components
- Event Delegation Model
 - Processing of an event is delegated to a particular object (handlers) in the program
 - Publish-Subscribe
 - Separate UI code from program logic

Event Handling Steps

- For a programmer the event Handling is a three step process in terms of code
- Step 1
 - Create components which can generate events
- Step 2
 - Build component (objects) that can handle events (Event Handlers)
- Step 3
 - Register handlers with generators

Event Handling Process [1]

Event Generators

- You have already seen a lot of event generators
 - Buttons
 - Mouse
 - Key
 - Window
 - Etc
- `JButton b1 = new JButton("Hello");`
- Now `b1` can generate events

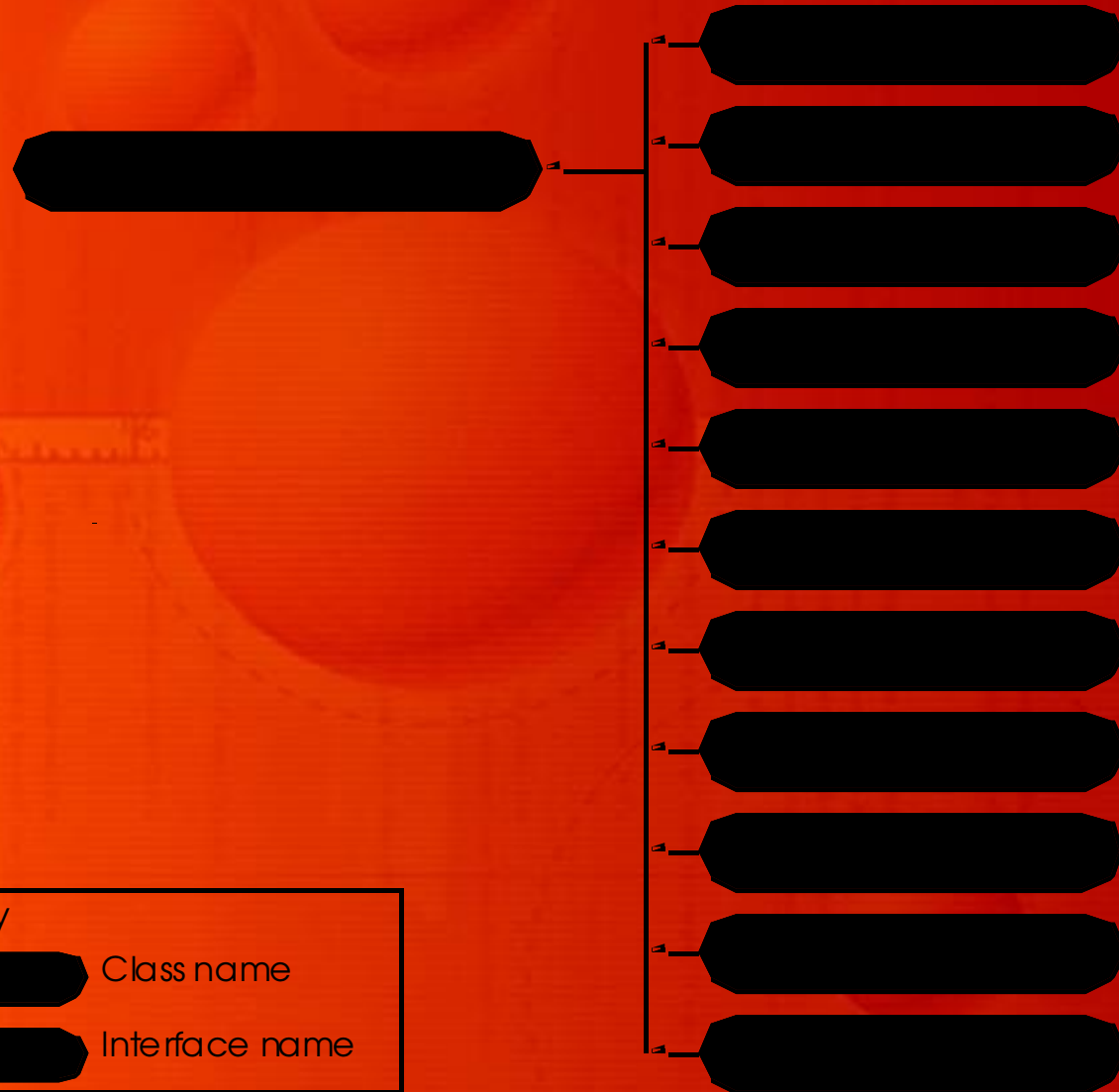
Event Handling Process [2]

Event Handlers/ Event Listener

- First Technique - By Implementing Listener Interfaces
 - Java defines interfaces for every event type
 - If a class needs to handle an event. It needs to implement the corresponding listener interface
 - To handle “ActionEvent” a class needs to implement “ActionListener”
 - To handle “KeyEvent” a class needs to implement “KeyListener”
 - To handle “MouseEvent” a class needs to implement “MouseListener”

And so on

Event Listener interfaces of package java.awt.event



Example Listeners

```
public interface ActionListener {  
    public void actionPerformed(ActionEvent e);  
}
```

```
public interface ItemListener {  
    public void itemStateChanged(ItemEvent e);  
}
```

```
public interface ComponentListener {  
    public void componentHidden(ComponentEvent e);  
    public void componentMoved(ComponentEvent e);  
    public void componentResized(ComponentEvent e);  
    public void componentShown(ComponentEvent e);  
}
```

Event Handling Process [3]

Event Handlers

- By implementing an interface the class agrees to implement all the methods that are present in that interface.
- Implementing an interface is like signing a contract
- Inside the method the class can do what ever it wants to do with that event
- Event Generator and Event Handler can be the same or different classes

Event Handling Process [4]

Event Handlers

- To handle events generated by Button. A class needs to implement ActionListener interface.

- public class Test implements ActionListener{

```
    public void actionPerformed(ActionEvent ae){
```

```
        // do something
```

```
    }
```

```
}
```


Event Handling Process [4]

Registering Handler with Generator

- The event generator is told about the object which can handle its events
- Event Generators have a method
 - `add_____Listener(_____)`
- `b1.addActionListener(objectOfTestClass)`

Event Handling

Simple Example

Event Handling: Simple Example Scenario



When Hello button is pressed, the Dialog box would be displayed

Event Handling: Simple Example

Step 1(cont.)

```
/* This program demonstrates the handling of Action Event.  
Whenever "Hello" button is presses, a dialog box would  
be displayed in response containing some informative  
message
```

```
*/
```

```
import java.awt.*;
```

```
import javax.swing.*;
```

```
import java.awt.event.*;
```

```
public class ActionEventTest {
```

```
    JFrame frame;
```

```
    JButton bHello;
```

Event Handling: Simple Example

Step 1 (cont.)

```
public void initGUI () {  
    frame = new JFrame();  
    // Event Generator  
    bHello = new JButton("Hello");  
  
    Container con = frame.getContentPane();  
    con.add(bHello);  
    frame.setSize(200,200);  
    frame.setVisible(true);  
  
} //end initGUI
```

Event Handling: Simple Example (cont.)

Step 2

```
// import your packages
```

```
public class ActionEventTest implements ActionListener {
```

```
.....
```

```
public void initGUI() ....
```

```
public void actionPerformed (ActionEvent ae ){
```

```
    JOptionPane.showMessageDialog("Hello is pressed");
```

```
}
```

```
}
```

Event Handling: Simple Example

Step 3 (cont.)

```
public void initGUI () {  
    // Event Generator  
    bHello = new JButton("Hello");  
  
    Container con = frame.getContentPane();  
    con.add(bHello);  
  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setSize(200,200);  
    frame.setVisible(true);  
  
    // Event Registration  
    bHello.addActionListener(this);  
  
} //end initGUI
```

Event Handling: Simple Example (cont.)

```
public ActionEventTest( ) {  
    initGUI ();  
}
```

```
public static void main (String args [ ]) {  
  
    ActionEventTest aeTest = new ActionEventTest();  
}
```

```
}//end ActionEvent class
```


Event Handling: Simple Example

Complete Code

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ActionEventTest implements ActionListener{

    JFrame frame;
    JButton bHello;

    public void initGUI () {
        frame = new JFrame();
        // Event Generator
        bHello = new JButton("Hello");

        Container con = frame.getContentPane();
        con.add(bHello);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200,200);
        frame.setVisible(true);

        // Event Registration
        bHello.addActionListener(this);
    }

    public void actionPerformed (ActionEvent ae ){
        JOptionPane.showMessageDialog("Hello is pressed");
    }

    public ActionEventTest() {
        initGUI ();
    }

    public static void main (String args []) {

        ActionEventTest aeTest = new ActionEventTest();
    }

}

//end ActionEvent class
```

Behind the Scenes

Event Handling Participants

1. Event Generator / Source

- Swing and awt components
- For example, JButton, JTextField, JFrame etc
- Generates an event object
- Registers listeners with itself

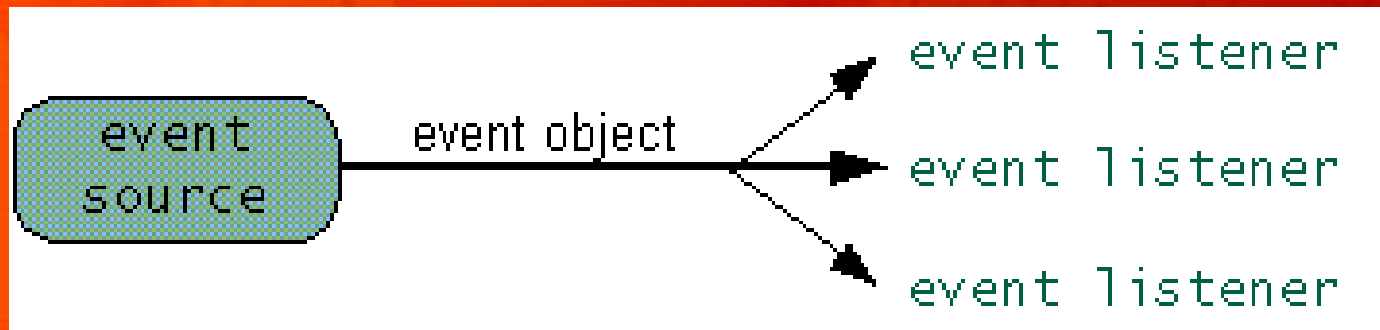
2. Event Object

- Encapsulate information about event that occurred and the source of that event
- For example, if you click a button, `ActionEvent` object is created

Event Handling Participants (cont.)

3. Event Listener/handler

- Receives event objects when notified, then responds
- Each event source can have multiple listeners registered on it
- Conversely, a single listener can register with multiple event sources

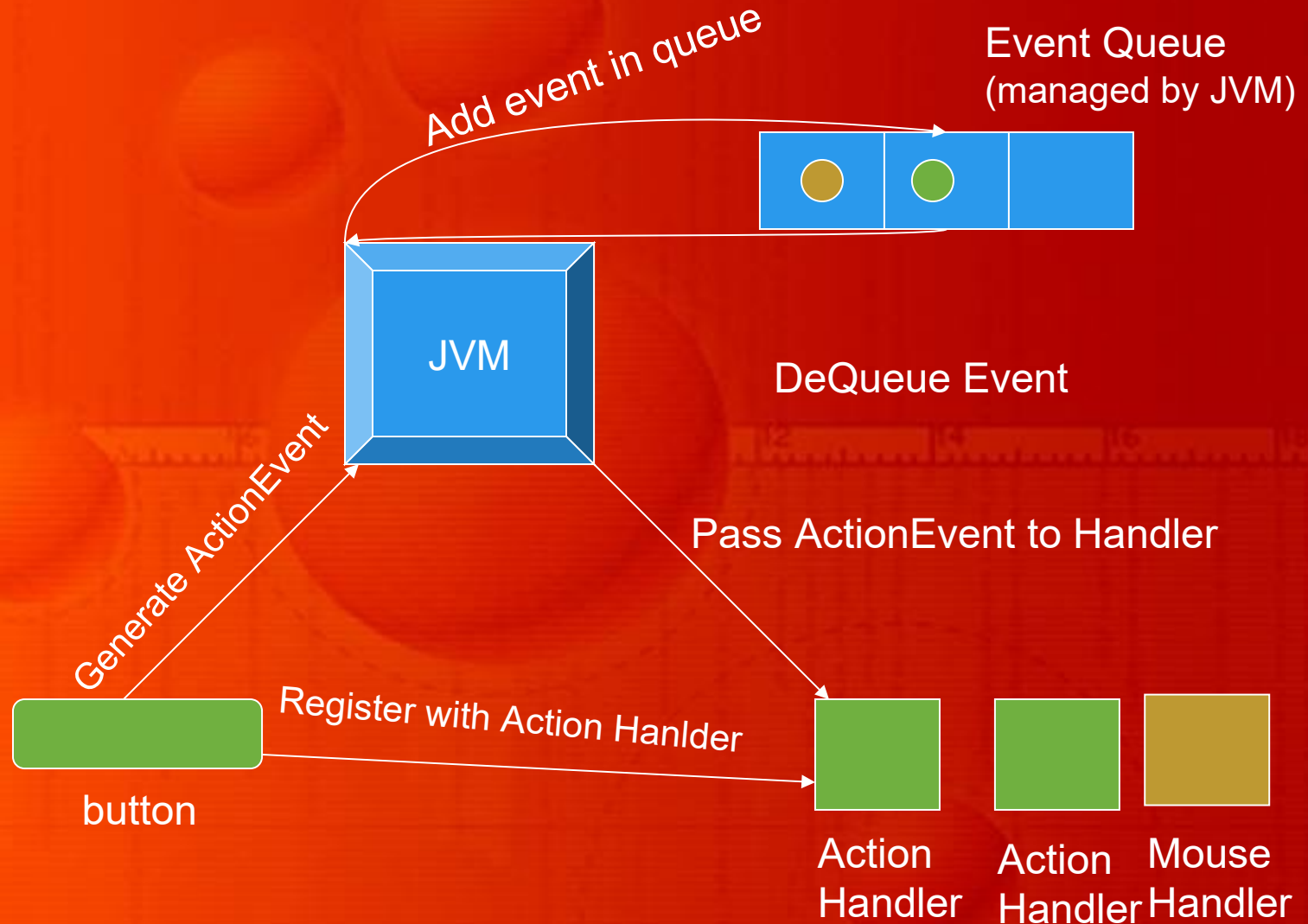


Event Handling Participants (cont.)

4. JVM

- Receives an event whenever one is generated
- Looks for the listener/handler of that event
- If exist, delegate it for processing
- If not, discard it (event).

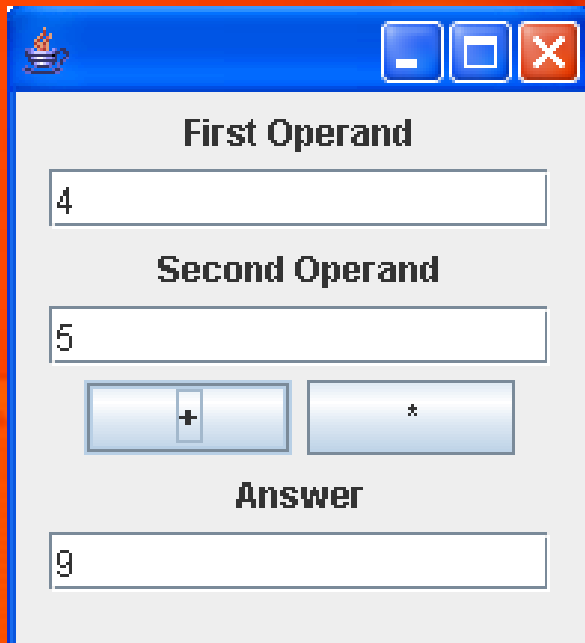
Event Handling Diagram



Making Small Calculator

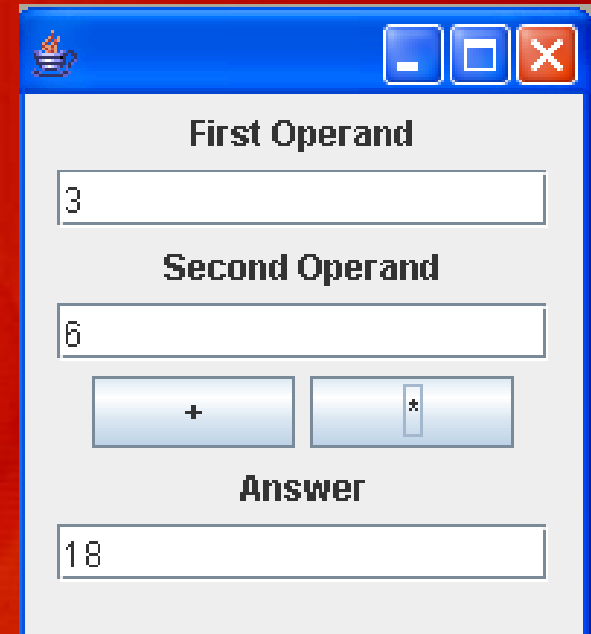
Example Code

Event Handling: Small Calculator Scenario



A small calculator window with a blue title bar and standard Windows window controls. It contains three text input fields and two buttons. The first field is labeled "First Operand" and contains the number "4". The second field is labeled "Second Operand" and contains the number "5". Below these fields are two buttons: a "+" button and a "*" button. At the bottom is an "Answer" field containing the number "9".

Field	Value
First Operand	4
Second Operand	5
Answer	9



A small calculator window with a blue title bar and standard Windows window controls. It contains three text input fields and two buttons. The first field is labeled "First Operand" and contains the number "3". The second field is labeled "Second Operand" and contains the number "6". Below these fields are two buttons: a "+" button and a "*" button. At the bottom is an "Answer" field containing the number "18".

Field	Value
First Operand	3
Second Operand	6
Answer	18

- User enters numbers in the provided fields
- On pressing “+” button, sum would be displayed in the answer field
- On pressing “*” button, product would be displayed in the answer field

Code: Small Calculator

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class SmallCalcApp implements ActionListener {
    JFrame frame;
    JLabel firstOperand, secondOperand, answer;
    JTextField op1, op2, ans;
    JButton plus, mul;

    public void initGUI () {
        .....
        plus = new JButton("+");
        mul = new JButton("*");
        .....
        plus.addActionListener(this);
        mul.addActionListener(this);
        .....
    } //end initGUI
}
```

Code: Small Calculator (cont.)

```
// providing definition of interface ActionListener's methods
```

```
public void actionPerformed (ActionEvent event ) {
```

```
    String oper, result;  
    int num1, num2, res;
```

```
    if (event.getSource() == plus) {
```

```
        oper = op1.getText();  
        num1 = Integer.parseInt(oper);
```

```
        oper = op2.getText();  
        num2 = Integer.parseInt (oper);
```

```
        res = num1+num2;  
        result = res+"";
```

```
        ans.setText(result);
```

```
    } // end if
```

```
    //continue
```

Code: Small Calculator (cont.)

```
else if (event.getSource() == mul) {  
  
    oper = op1.getText();  
    num1 = Integer.parseInt(oper);  
  
    oper = op2.getText();  
    num2 = Integer.parseInt (oper);  
  
    res = num1*num2;  
    result = res+"";  
  
    ans.setText(result);  
  
}  
  
} // end actionPerformed method
```

.....

Code: Small Calculator (cont.)

```
..... //write default constructor and call initGUI
```

```
public static void main (String args[ ]) {  
    SmallCalcApp scApp = new SmallCalcApp();  
}
```

```
} // end SmallCalcApp
```

Web Design & Development

Lecture 12

Event Handling

part 2

Important Points Revisited

Important Points Revisited

- To handle a particular kind of event, we have to implement a corresponding interface
- For example
 - Button generates action event, we have to implement **ActionListener** interface to handle action events
 - Window generates window event, we have to implement **WindowListener** interface to handle window events

Important Points Revisited

- Event Handling Steps
 - Step 1
 - Create components which can generate events
 - Step 2
 - Build component (objects) that can handle events (Event Handlers)
 - Step 3
 - Register handlers with generators

More Examples

Handling Mouse Events

Handling Mouse Events

- Mouse events can be trapped for any GUI component that inherit from `Component` class. For example, `JPanel`, `JFrame` & `JButton` etc.
- To handle Mouse events, two types of listener interfaces are available
 - `MouseMotionListener`
 - `MouseListener`

Handling Mouse Events

- MouseMotionListener
 - For processing mouse motion events
 - Mouse motion event is generated when mouse is moved or dragged

```
public interface MouseMotionListener {  
  
    public void mouseDragged (MouseEvent me);  
  
    public void mouseMoved (MouseEvent me);  
  
}
```

Handling Mouse Events

- **MouseListener**
 - For processing “interesting” mouse events
 - Mouse event is generated when mouse is
 - Pressed
 - Released
 - clicked (pressed & released without moving the cursor)
 - Enter (mouse cursor enters the bounds of component)
 - Exit (mouse cursor leaves the bounds of component)

Handling Mouse Events

Interface MouseListener

```
public interface MouseListener {  
  
    public void mousePressed (MouseEvent me);  
  
    public void mouseClicked (MouseEvent me);  
  
    public void mouseReleased (MouseEvent me);  
  
    public void mouseEntered (MouseEvent me);  
  
    public void mouseExited (MouseEvent me);  
  
}
```

Example Code

Handling Mouse Motion Events

Steps for Handling Mouse Motion Events

1. ? Create mouse motion events generating components

- `JFrame myFrame = new JFrame();`

2. ? Create Event Handler

- Implement `MouseMotionListener` Interface
- Provide implementation for the required method

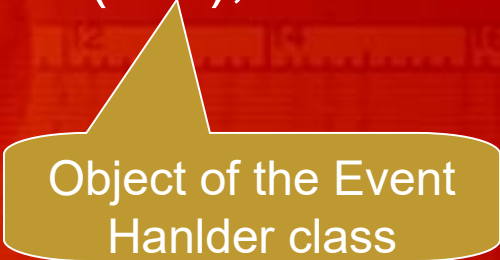
```
public void mouseMoved (MouseEvent e) {  
    // write your code here  
}
```

- provide empty bodies for remaining methods in the interface.

Steps for Handling Mouse Motion Events

3. Register event generator with event handler

```
myFrame.addMouseMotionListener (this);
```



Object of the Event
Handler class

Example Code (cont.)

Handling Mouse Motion Events

// File EventsEx.java, Code listed in Handout section **12.1**

..... // import required packages

public class EventsEx implements MouseMotionListener {

 JFrame f;

 JLabel coord;

 public void initGUI (){

 // set layout

 // add label to container

 // registration

f.addMouseListener(this)

 } // end of initGui

Example Code (cont.)

Handling Mouse Motion Events

```
// MouseMotionListener event handlers
public void mouseDragged (MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    coord.setText("Dragged at [" + x + "," + y + "]" );
}
```

```
public void mouseMoved (MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    coord.setText("Moved at [" + x + "," + y + "]" );
}
```

Example Code (cont.)

Handling Mouse Motion Events

```
public static void main (String args[ ]){  
    EventsEx ex = new EventsEx();  
}  
} // end of EventsEx class
```

Handling Mouse Motion Events

Live Output

Another Example

Handling Window Events

Handling Window Events

- Want to handle Window Exit event only
 - Why ?
 - When window is closed, control should return back to command prompt
 - But we have already achieved this functionality through following line of code

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```
 - But, what if we want to display some message (Good Bye) before exiting

Handling Window Events

– How ?

- To handle window events, we need to implement “WindowListner” interface.
- “WindowListner” interface contains 7 methods
- We require only one i.e. **windowClosing**
- We have to provide definitions of all methods to make concrete class

Interface WindowListener

```
public interface WindowListener {  
  
    public void windowActivated (WindowEvent we);  
  
    public void windowClosed (WindowEvent we);  
  
    public void windowClosing (WindowEvent we);  
  
    public void windowDeactivated (WindowEvent we);  
  
    public void windowDeiconified (WindowEvent we);  
  
    public void windowIconified (WindowEvent we);  
  
    public void windowOpened (WindowEvent we);  
  
}
```

Example Code: Window Exit Handler

Modification of EventsTest.java

```
// File EventsEx.java, Code listed in Handout section 12.2
.....
public class EventsEx implements MouseMotionListener , WindowListener, {
    JFrame f;
    JLabel coord;

    public void initGUI () {
        ..... // set layouts

        window.addMouseMotionListener(this);

        f.addWindowListener(this);
        .....
    } //end initGUI

    // Event Handlers for MouseMotionListener
    public void mouseDragged(MouseEvent me) {.....}
    public void mouseMoved(MouseEvent) {....}
```

Example Code: Window Exit Handler

Modification of EventsTest.java

```
// Event Handlers for WindowListener

public void windowActivated (WindowEvent we) {    }

public void windowClosed (WindowEvent we) {    }

public void windowClosing (WindowEvent we) {
    JOptionPane.showMessageDialog(null, "Good Bye");
    System.exit(0)
}

public void windowDeactivated (WindowEvent we) {    }

public void windowDeiconified (WindowEvent we) {    }

public void windowIconified (WindowEvent we) {    }

public void windowOpened (WindowEvent we) {    }
```

Example Code: Window Exit Handler

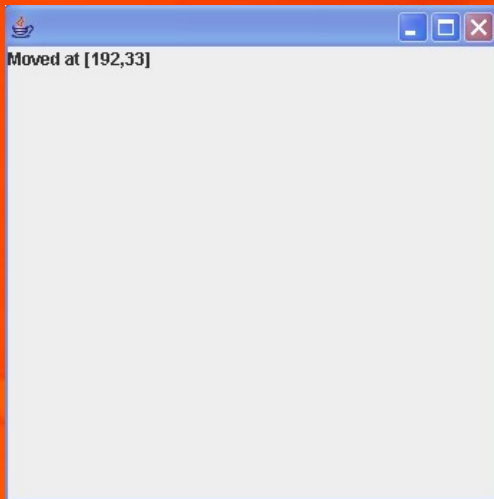
Modification of EventsTest.java

```
public static void main (String args[ ]){  
    EventsEx ex = new EventsEx();  
}  
} // end of EventsEx class
```

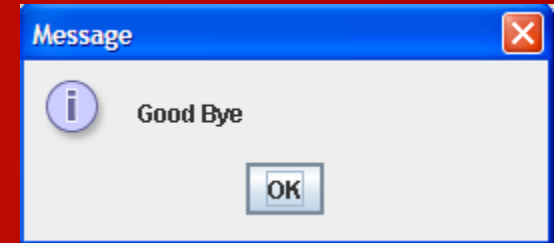
Example Code: Window Exit Handler

Modification of EventsTest.java

Output



When user closes the window,
Message would be displayed



After pressing Ok button
Program will exit

Web Design & Development

Lecture 13

Event Handling

part 3

Handling Window Events

Example Revisited

Window Exit Handler Example Reveisted

```
// Event Handlers for WindowListener

public void windowActivated (WindowEvent we) {   }

public void windowClosed (WindowEvent we) {   }

public void windowClosing (WindowEvent we) {
    JOptionPane.showMessageDialog(null, "Good Bye");
    System.exit(0)
}

public void windowDeactivated (WindowEvent we) {   }

public void windowDeiconified (WindowEvent we) {   }

public void windowIconified (WindowEvent we) {   }

public void windowOpened (WindowEvent we) {   }
```

Last Code Example

- Problem
 - Interested in windowClosing method only
 - But have to provide definitions of all the methods, Why ?
 - Because a class implementing an interface has to provide definitions of all methods present in that interface.
- Solution
 - Use Adapter classes

Adapter Classes

Adapter Classes

- For listener interfaces containing more than one event handling methods, jdk defines adapter classes. Examples are
 - For WindowListener → WindowAdapter
 - For MouseMotionListener → MouseMotionAdapter
 - and many more
- Adapter classes provides definitions for all the methods (empty bodies) of their corresponding Listener interface
- It means that WindowAdapter class implements WindowListener interface.

Adapter Classes

```
public interface MouseMotionListener {  
  
    public void mouseDragged (MouseEvent me);  
  
    public void mouseMoved (MouseEvent me);  
  
}
```

```
public class MouseMotionAdapter implements  
                                MouseMotionListener {  
  
    public void mouseDragged (MouseEvent me) { }  
  
    public void mouseMoved (MouseEvent me) { }  
  
}
```

How to use Adapter Classes

- Previously handler class need to implement interface

```
public class EventsEx implements  
    MouseMotionListener { .... }
```

- Therefore it has to provide definitions of all the methods inside that interface

How to use Adapter Classes

- Now our handler class will inherit from adapter class
 - `public class EventsEx extends MouseMotionAdapter`
`{ .... }`
 - Due to inheritance, all the methods of the adapter class will be available inside our handler class
 - Since adapter classes has already provided definitions with empty bodies.
 - We do not have to provide implementations of all the methods again
 - We only need to override our method of interest.

Adapter Classes

Listener	Adapter Class (If Any)	Registration Method
ActionListener	ComponentAdapter ContainerAdapter FocusAdapter	addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener		addComponentListener
ContainerListener		addContainerListener
FocusListener		addFocusListener
ItemListener	KeyAdapter MouseAdapter MouseMotionAdapter	addItemListener
KeyListener		addKeyListener
MouseListener		addMouseListener
MouseMotionListener	WindowAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener		addWindowListener

Example Code: Adapter Classes

Modification of EventsEx

```
// File EventsEx.java, Code listed in Handout section 13.1
```

```
.....
```

```
public class EventsEx extends WindowAdapter {
```

```
    JFrame f;
```

```
    JLabel coord;
```

```
    public void initGUI () {
```

```
        ..... // set layouts
```

```
        f.addWindowListener(this);
```

```
        .....
```

```
    } //end initGUI
```

Example Code: Adpater Classes

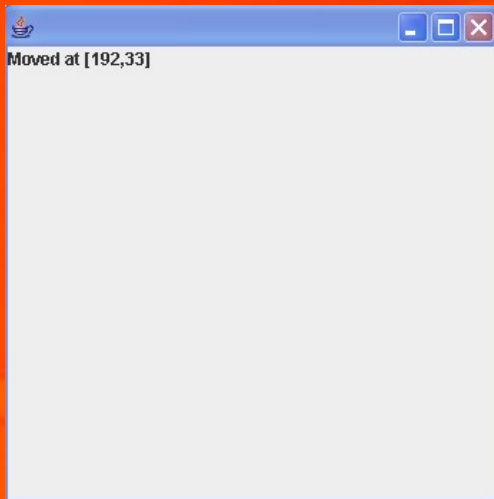
Modification of EventsEx.java

```
// Event Handler for WindowListener  
public void windowClosing (WindowEvent we) {  
    JOptionPane.showMessageDialog(null, "Good Bye");  
    System.exit(0)  
}  
  
public static void main (String args[ ]){  
    EventsEx ex = new EventsEx();  
}  
  
} // end class
```

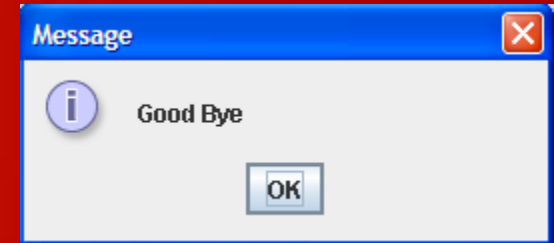
Example Code: Window Exit Handler

Modification of EventsEx.java

Output



When user closes the window,
Message would be displayed



After pressing Ok button
Program will exit

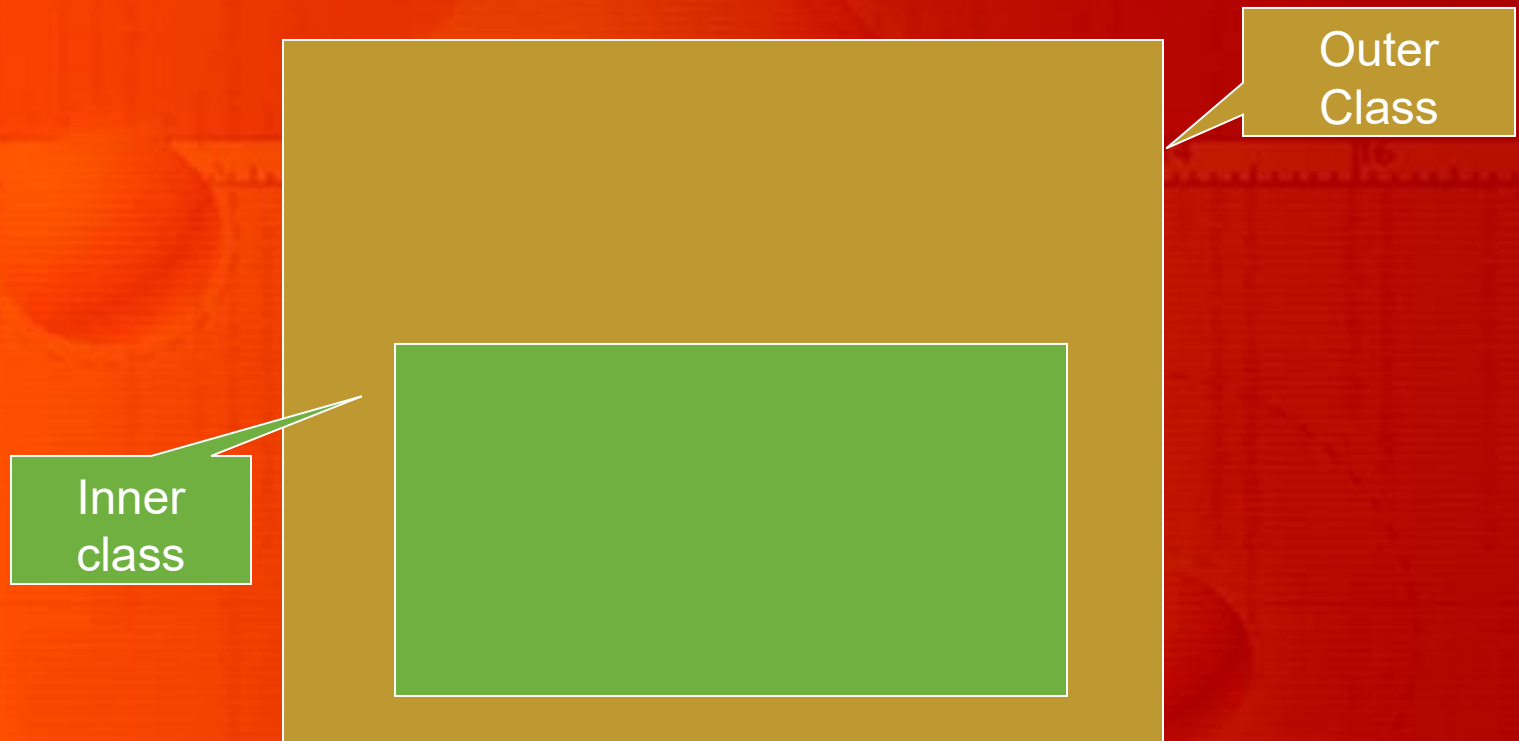
Last Code Example

- We have inherited from WindowAdapter
- What if we want to use MouseMotionAdpater as well ? or what if our class already inherits form some other class ?
- **Problem**
 - But, Java allows single inheritance
- **Solution**
 - Use Inner Classes

Inner Classes

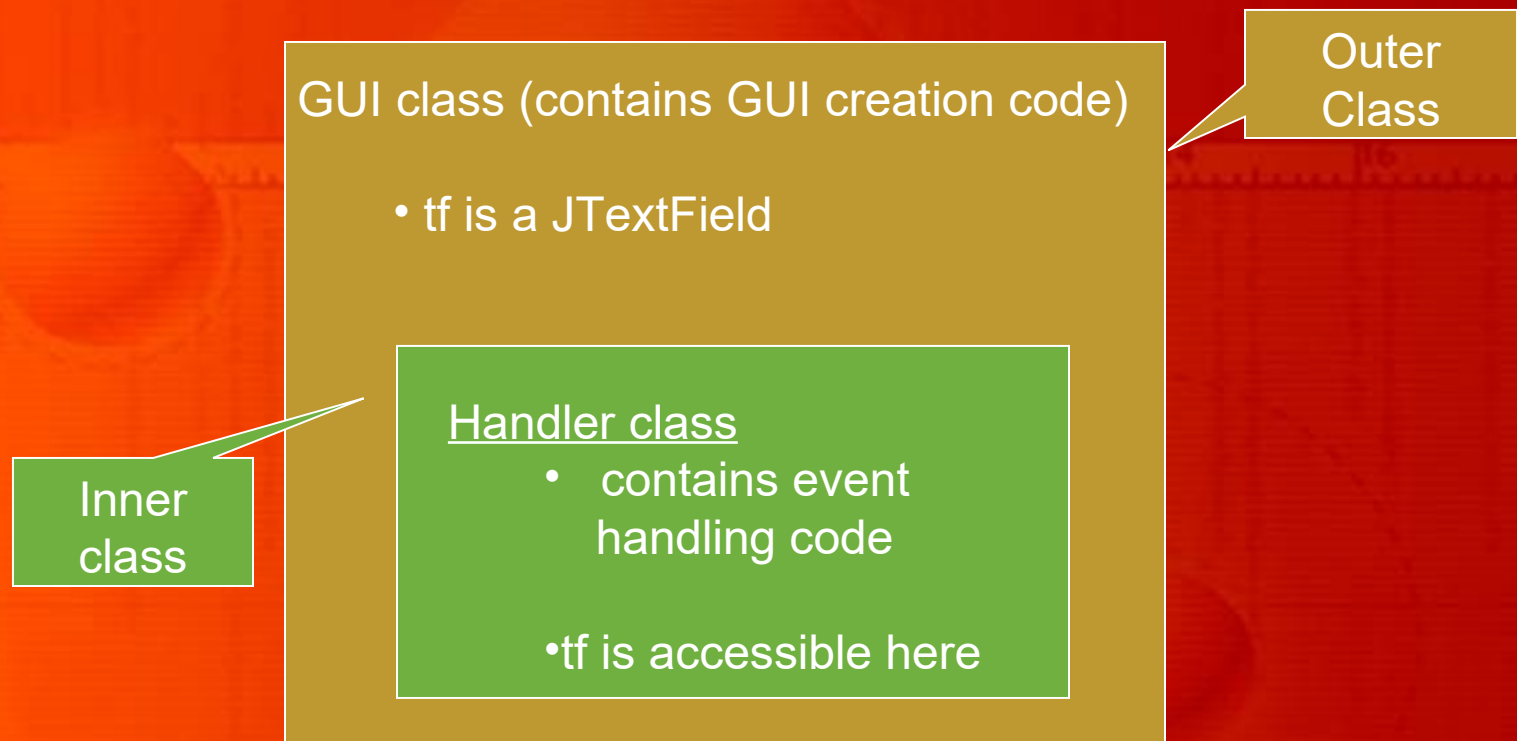
Inner Classes

- A class defined inside another class



Inner Classes

- Inner class can access the instance variables and members of outer class



Inner Classes

- It can have constructors, instance variables and methods, just like a regular class
- Generally used as a private utility class which does not need to be seen by others classes

Inner Classes

Handling Window Events

Example Code: Inner Classes Handling Window Events

```
// File EventsEx.java, Code listed in Handout section 13.2
.....
public class EventsEx {
    JFrame f;
    JLabel coord;

    //inner class
    private class WindowHandler extends WindowAdapter {

        // Event Handler for WindowListener
        public void windowClosing (WindowEvent we) {
            JOptionPane.showMessageDialog(null, "Good Bye");
            System.exit(0)
        }
    } // end of WindowHandler
```

Example Code: Inner Classes Handling Window Events

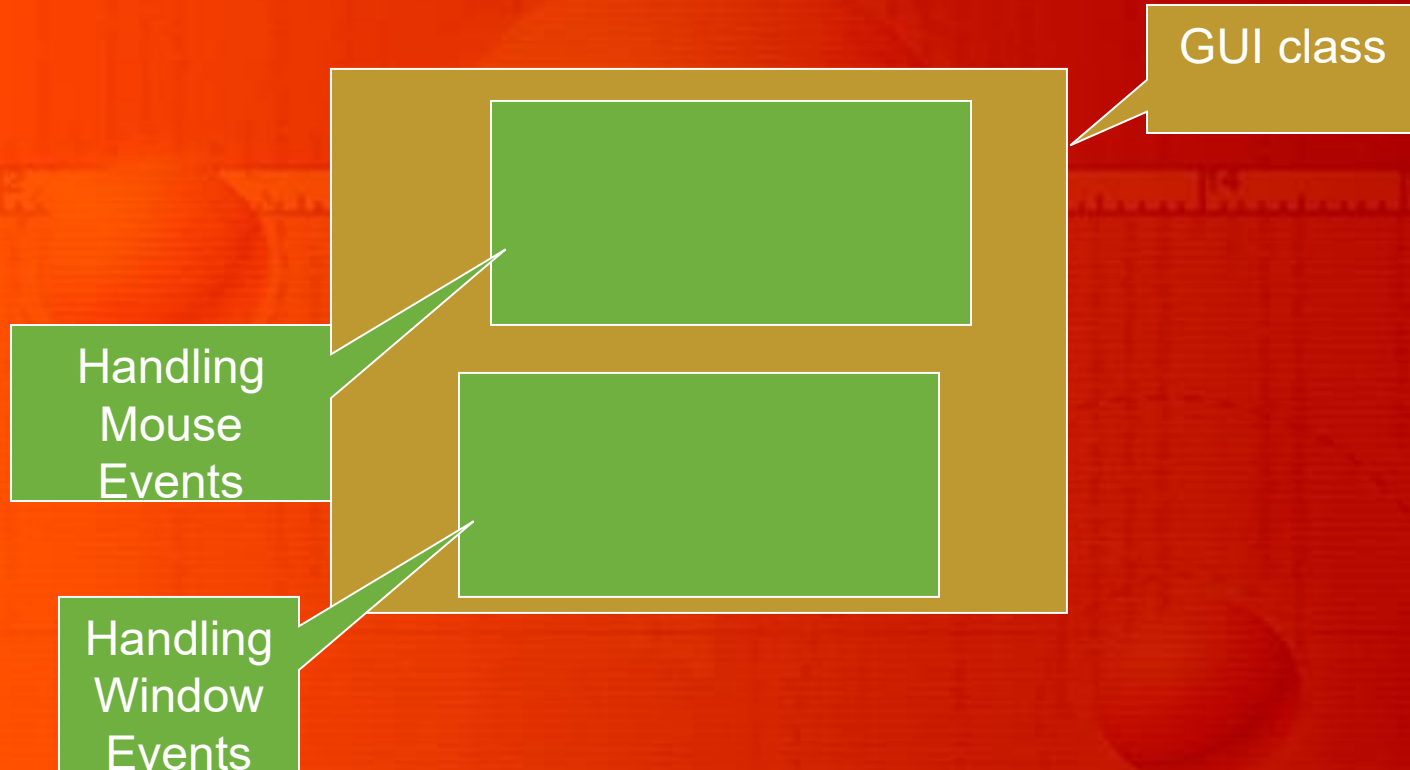
```
public void initGUI () {  
    ..... // set layouts  
  
    WindowHandler handler = new Window Handler ();  
    f.addWindowListener(handler);  
    .....  
} //end initGUI  
  
// main method  
  
} // end of EventsEx class
```

Inner Classes

Handling Window & Mouse
Motion Events

Inner Classes

- Handling Windows & Mouse motion events



Example Code: Inner Classes

Handling Window & MouseMotion Events

// File EventsEx.java, Code listed in Handout section 13.3

.....

```
public class EventsEx {
```

```
    JFrame f;
```

```
    JLabel coord;
```

```
    //inner class
```

```
    private class WindowHandler extends WindowAdapter {
```

```
        // Event Handler for WindowListener
```

```
        public void windowClosing (WindowEvent we) {
```

```
            JOptionPane.showMessageDialog(null, "Good Bye");
```

```
            System.exit(0)
```

```
        }
```

```
    } // end of WindowHandler
```

Example Code: Inner Classes Handling Window Events

```
//inner class
private class MouseHandler extends MouseMotionAdapter {

    // Event Handler for mouse motion events
    public void mouseMoved (MouseEvent me) {
        int x = me.getX();
        int y = me.getY();

        coord.setText("Moved at [" + x + "," + y + "]" );
    }
} // end of MouseHandler
```


Example Code: Inner Classes Handling Window Events

```
public void initGUI () {  
    ..... // set layouts  
  
    WindowHandler wHandler = new WindowHandler ();  
    f.addWindowListener(wHandler);  
  
    MouseHandler mHandler = new MouseHandler();  
    f.addMouseMotionListener(mHandler);  
  
    .....  
} //end initGUI  
  
//main method  
  
} // end EventsExclass
```

Why Not Outer Class?



Event Generator class

```
.....  
.....  
  
JFrame frame;  
JTextField tf;  
  
.....  
.....
```

Action Event Handler

```
.....  
String s = tf.getText();  
  
.....
```

Window Event Handler



Anonymous Inner Classes

Anonymous Inner Classes

- Has no name
- Same as inner class in capabilities
- much shorter
- difficult to understand

Named vs. Anonymous Objects

- Named
 - `String s = "hello";`
`System.out.println(s);`
 - “hello” has a named reference `s`.
- Anonymous
 - `System.out.println("hello");`

Example Code: Anonymous Classes

```
// File EventsEx.java, Code listed in Handout section 13.4
```

```
.....
```

```
public class EventsEx {
```

```
    JFrame f;
```

```
    JLabel coord;
```

Example Code: Anonymous Classes

```
public void initGUI () {  
    ..... // set layouts  
  
    //anonymous inner class  
    f.addWindowListner (new WindowAdapter ( ) {  
  
        public void windowClosing (WindowEvent we) {  
            JOptionPane.showMessageDialog(null, "Good Bye");  
            System.exit(0)  
        }  
    }  
);  
    .....  
} //end initGUI
```

Summary of Handling Events Approaches

Approaches for Handling Events

1. By implementing Interfaces
2. By extending from Adapter classes

To implement the above two techniques we can use

- Same class
 - (putting event handler & generator in one class)
- Separate class
 - Outer class
 - Putting event handlers & generator in two different classes
 - Inner classes
 - Anonymous Inner classes



Web

Design & Development

Lec - 14



Web

Design & Development

Java Database Connectivity (JDBC)

Agenda

- **Overview of JDBC technology**
- **JDBC drivers**
- **Seven basic steps in using JDBC**
- **Example of Retrieving data from ResultSet**
- **Example of Executing DML statements**
- **Using prepared Statement**

JDBC Introduction

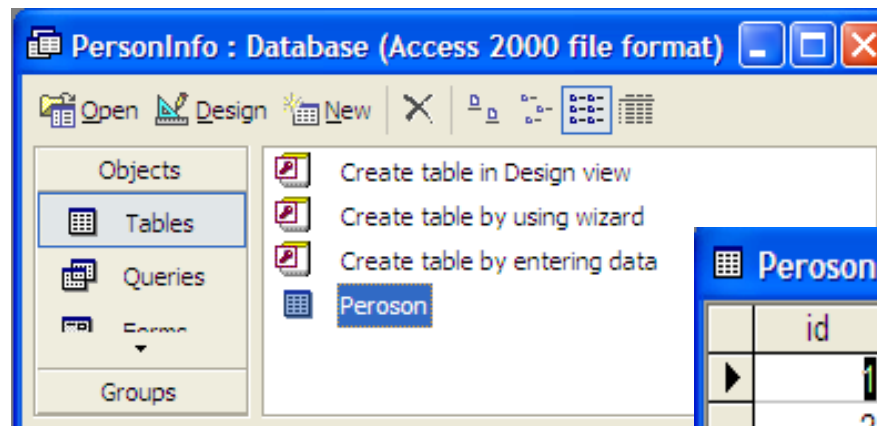
- **JDBC provides a standard library for accessing relational databases**
 - API standardizes
 - Way to establish connection to database
 - Approach to **initiate** queries
 - Method to create stored (parameterized) queries
 - The data structure of query result (table)
 - Determining the number of columns
 - Looking up metadata, etc.

JDBC Introduction

- API does *not* standardize SQL syntax
 - JDBC is not embedded SQL
- JDBC classes located in `java.sql` package

Connecting Microsoft Access Example

- Create **PersonInfo** database
- Create **Person** table

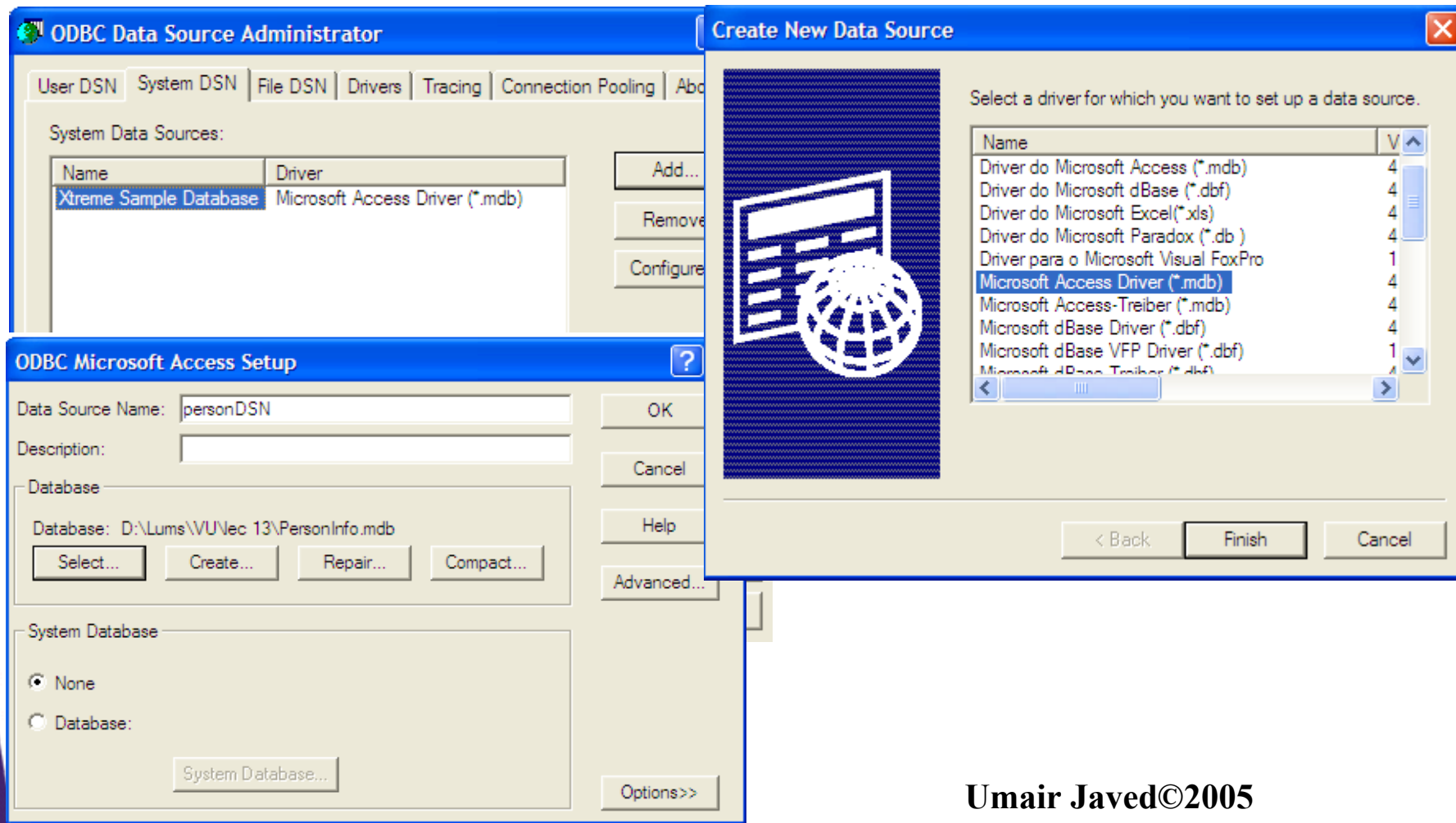


	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

Connecting MS Access Example: Setup System DSN

- settings → Control Panel → Administrative Tools → data sources(ODBC)





Web

Design & Development

Basic Steps in Using JDBC

Basic Steps in Using JDBC

- 1. Import required packages**
- 2. Load driver**
- 3. Define Connection URL**
- 4. Establish Connection**
- 5. Create a Statement object**

Basic Steps in Using JDBC (cont.)

- 6. Execute query / DML**
- 7. Process results**
- 8. Close connection**



Web

Design & Development

JDBC

Details of Process

JDBC: Details of Process

1. Import package

- Import `java.sql` package
 - `import java.sql.*;`

JDBC: Details of Process

2. Loading driver

- Need to load suitable driver for underlying database
- Different drivers for different databases are available

- For MS Access

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- For Oracle

```
Class.forName("oracle.jdbc.driver.OracleDriver ");
```

JDBC: Details of Process

3. Define Connection URL

- To get a connection, we need to specify URL of database.
- If you are using a JDBC-ODBC driver you need to create a DSN. DSN is the name of your DataSource
- If the name of your DSN is “personDSN” then the url of the database will be
 - **String conURL = “jdbc:odbc:personDSN”**

JDBC: Details of Process, cont.

4. Establish Connection

- Connection con = null;
- Use driver manager to get the connection object

```
con = DriverManager.getConnection(conURL);
```

- If the Db requires username and password you can use overloaded version

- String usr = "umair";
- String pswd = "vu";
- Connection con = null;

```
con = DriverManager.getConnection(conURL, usr, pswd);
```

Umair Javed © 2005

JDBC: Details of Process, cont.

5. Create Statement

- A statement is obtained from a Connection object.

```
Statement statement = con.createStatement();
```

- Once you have a statement, you can use it for various kind of SQL queries

JDBC: Details of Process, cont.

6(a) Execute Query / DML

- executeQuery(sql) method
 - Used for SQL SELECT queries
 - Returns the *ResultSet* object which is used to access the rows of the query results

```
String sql = "SELECT * FROM sometable";  
ResultSet rs = statement.executeQuery(sql);
```

JDBC: Details of Process, cont.

6(b) Execute Query / DML

- `executeUpdate(sql)` method
 - Used for an update statement (INSERT, UPDATE or DELETE)
 - Returns an integer value representing the number of rows updated.

```
String sql = "INSERT INTO tableName " +  
             "(columnNames) Values (values)";
```

```
int count = statement.executeUpdate(sql) ;
```

JDBC: Details of Process, cont.

7. Process Results

- ResultSet provides various getXxx methods that take a column index or name and returns the data
- First column has index 1, not 0

```
while(resultSet.next()) {  
    //by using column name  
    String name = rs.getString("columnName");  
  
    //or by using index  
    String name = rs.getString(1);  
}
```

JDBC: Details of Process, cont.

8. Close Connection

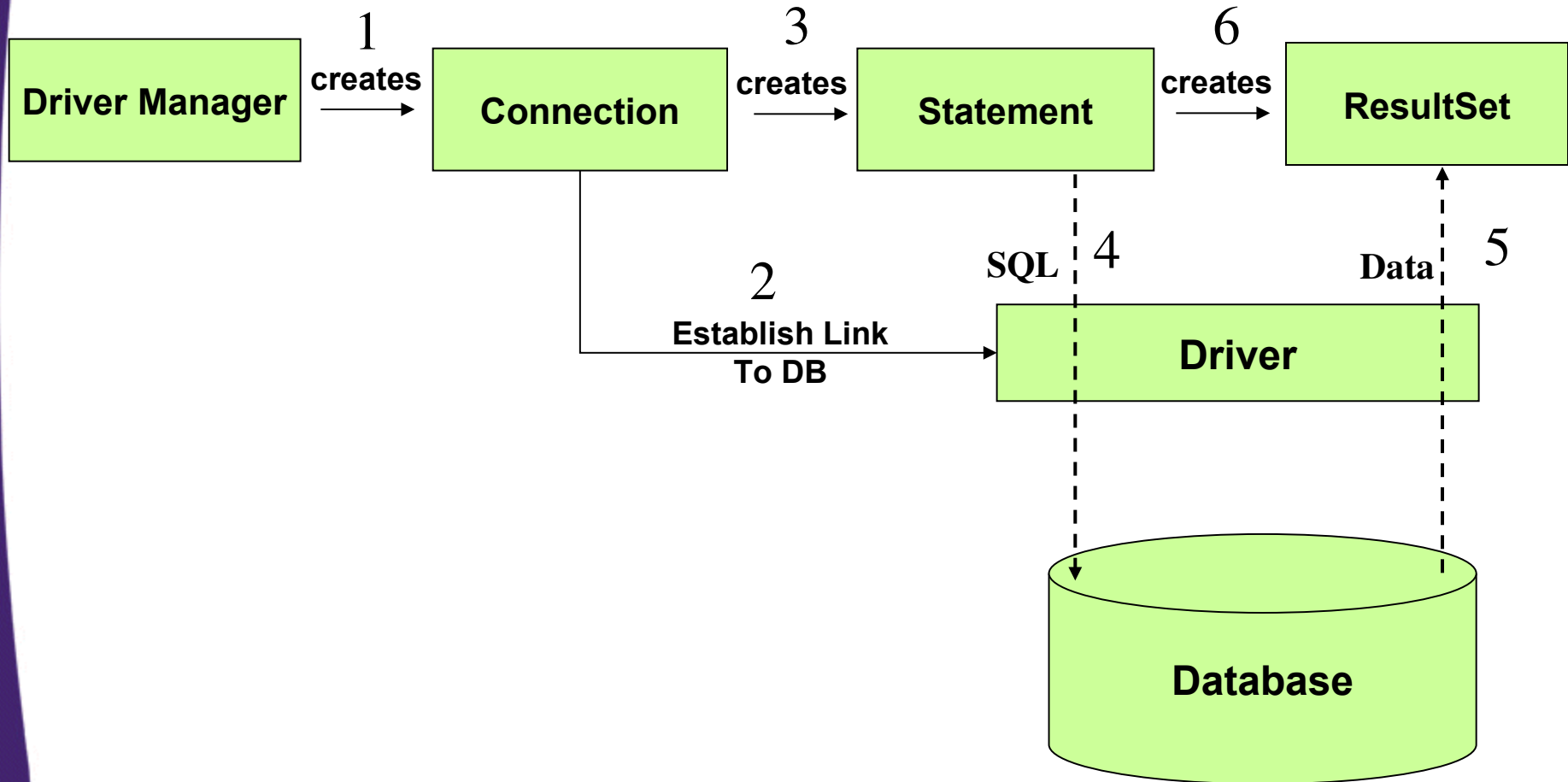
```
connection.close();
```

- As opening a connection is expensive, postpone this step if additional database operations are expected

In a nut shell

- **Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);**
- **Connection con = null;**
con = DriverManager.getConnection(url, usr, pwd);
- **Statement st = con.createStatement();**
- **ResultSet rs = st.exectuteQuery(“Select * from Person”);**

JDBC Architecture





Web

Design & Development

Example Code

Retrieving Data from ResultSet

Example Code 14.1

Retrieving Data from ResultSet

```
//Step 1: import package  
import java.sql.*;
```

```
public class JdbcEx {
```

```
    public static void main (String args [ ]){
```

```
        try {
```

```
            //Step 2: Load driver
```

```
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
            //Step 3: Define the connection URL
```

```
            String url = "jdbc:odbc:personDSN";
```

```
            //Step 4: Establish the connection
```

```
            Connection con = null;
```

```
            con = DriverManager.getConnection (url , "" , "");
```

Example Code 14.1

Retrieving Data from ResultSet (cont.)

```
//Step 5: create the statement  
Statement st = con.createStatement();
```

```
//Step 6: Execute the query  
String sql = "SELECT * FROM Person";  
  
ResultSet rs = st.executeQuery(sql);
```

```
//Step 7: Process the results  
while ( rs.next() ) {
```

```
    String name = rs.getString("name");  
    String add  = rs.getString("address");  
    String pNum = rs.getString("phoneNum");
```

```
    System.out.println(name + " " +add +" "+pNum);
```

```
} // end while
```

Example Code 14.1

Retrieving Data from ResultSet (cont.)

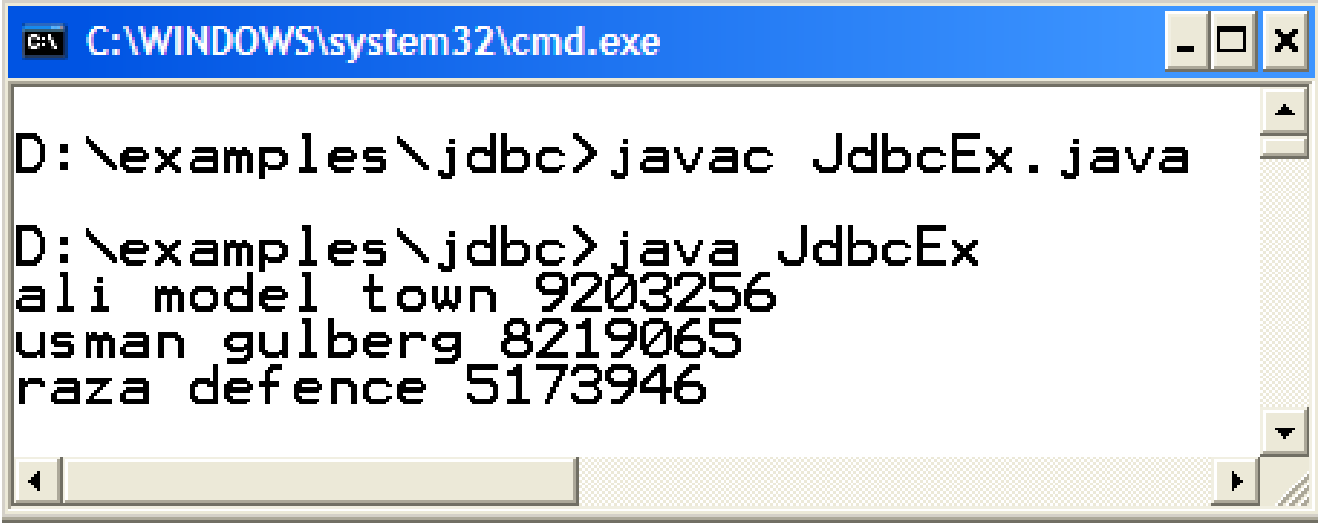
```
//Step 8: close the connection  
con.close();
```

```
}catch (Exception sqlEx) {  
  
    System.out.println(sqlEx);  
  
}
```

```
} //end main
```

```
}//end class
```

Compile & Execute



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" along with standard window control buttons (minimize, maximize, close). The command prompt shows the following sequence of commands and output:

```
D:\examples\jdbc>javac JdbcEx.java  
D:\examples\jdbc>java JdbcEx  
ali model town 9203256  
usman gulberg 8219065  
raza defence 5173946
```

The output displays three lines of data, each consisting of a name, a location, and a number. The command prompt has a scroll bar on the right and a horizontal scroll bar at the bottom.



Web

Design & Development

Lec - 15



Web

Design & Development

More on JDBC



Web

Design & Development

Example Code

Executing SQL DML statements

Useful Statement Methods (N-I)

- **executeQuery**

- Executes the SQL query and returns the data in a table (ResultSet)
- The resulting table may be empty but never null

```
ResultSet rs = stmt.executeQuery("select * from table");
```


Example Code 15.1

Executing SQL DML Statements

/* This program will take two command line arguments that are used to update records in the database */

import java.sql.*; //step 1

public class JdbcDmlEx {

public static void main (String args []){

try {

//steps 2 to 5

Class.forName("driver name");

Connection con=null;

con = DriverManager.getConnection(url, usr, pwd);

Statement st = con.createStatement();

Example Code 15.1

Executing SQL DML Statements (cont.)

//Step 6: Execute the Query / DML

```
String addVar    = args[0];  
String nameVar = args[1];
```

```
String sql = "UPDATE Person " +  
             " SET address = ' " + addVar + " ' " +  
             " WHERE name = ' " + nameVar + " ' " ;
```

```
int num = st.executeUpdate(sql);
```

//Step 7: Process the results of the query

```
System.out.println(num + " records updated");
```

Example Code 15.1

Executing SQL DML Statements (cont.)

```
//Step 8: close the connection  
con.close();
```

```
}catch (Exception sqlEx) {  
  
    System.out.println(sqlEx);  
  
}
```

```
} //end main
```

```
}//end class
```

Compile & Execute

Person : Table

	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

Before execution

```
C:\WINDOWS\system32\cmd.exe

D:\examples\jdbc>javac JdbcDmlEx.java
D:\examples\jdbc>java JdbcDmlEx defence ali
1 record updated
```

Person : Table

	id	name	address	phoneNum
▶	1	ali	defence	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

After execution

Useful Statement Methods (Continued)

- **getMaxRows() / setMaxRows(int)**
 - Determines the number of rows a `ResultSet` may contain
 - Unless explicitly set, the number of rows are unlimited (return value of 0)
- **getQueryTimeout() / setQueryTimeout(int)**
 - Specifies the amount of a time (seconds) a driver will wait for a `STATEMENT` to complete before throwing a `SQLException`

Different Types of Statements

- **Overview**

- Through the Statement object, SQL statements are sent to the database.
- Three types of statement objects are available:

- 1. Statement**

- for executing a **simple SQL** statements

- 2. PreparedStatement**

- for executing a **precompiled SQL statement** passing in parameters

- 3. CallableStatement**

- for executing a **database stored procedure**



Web

Design & Development

Prepared Statements

Prepared Statements (Precompiled Queries)

- **Idea**

- If you are going to execute **similar SQL statements** multiple times, using “**prepared**” (parameterized) **statements** can be more efficient
- Create a statement in standard form that is sent to the database for compilation before actually being used
- Each time you use it, you simply replace some of the marked parameters (?) using some **set** methods

Prepared Statement, Example

```
PreparedStatement pstmt =  
    con.prepareStatement("UPDATE tableName " +  
        "SET columnName = ? " +  
        "WHERE columnName = ?");
```

- First marked parameter(?) has index 1.

```
pstmt.setString(1, stringValue);  
pstmt.setInt    (2, intValue);
```

```
pstmt.executeUpdate();
```



Web

Design & Development

Example Code

Using Prepared Statements

Example Code: Modify JdbcDmlEx.java 15.1

Executing Prepared Statements

**`/* Modification to the last example code 15.1,
to show the usage of prepared statements */`**

`import java.sql.*; // step1`

`public class JdbcDmlEx {`

`public static void main (String args [ ]){`

`try {`

`//steps 2 to 4`

`Class.forName("driver name");`

`Connection con=null;`

`con = DriverManager.getConnection(url, usr, pwd);`

Example Code: Modify JdbcDmlEx.java 15.1

Executing Prepared Statements

//Step 5: Create the statement

PreparedStatement pStmt = null;

String sql = "UPDATE Person SET address = ? WHERE name = ? " ;

pStmt = con.prepareStatement(sql);

//Step 6: Execute the Query

String addVar = args[0];

String nameVar = args[1];

pStmt.setString(1 , addVar);

pStmt.setString(2, nameVar);

// sql = "UPDATE Person SET address = "defence" WHERE name = "ali" "

int num = pStmt.executeUpdate();

Example Code: Modify JdbcDmlEx.java 15.1

Executing Prepared Statements

//Step 7: Process the results of the query
System.out.println(num + “ records updated”);

//Step 8: close the connection

```
}catch (Exception sqlEx) {  
    .....  
}  
  
} //end main  
} //end class
```

Compile & Execute

Person : Table

	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

Before execution

```
C:\WINDOWS\system32\cmd.exe

D:\examples\jdbc>javac JdbcDmlEx.java
D:\examples\jdbc>java JdbcDmlEx defence ali
1 record updated
```

Person : Table

	id	name	address	phoneNum
▶	1	ali	defence	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

After execution



Web

Design & Development

Result Set

ResultSet

The diagram illustrates a **ResultSet** object containing a table of data. A green speech bubble labeled "Row numbers" points to the first column of the table, which contains the indices 0, 1, 2, and 3. A green arrow points from the "Row numbers" label to the first column. Another green speech bubble labeled "ResultSet" points to the entire table structure.

0	id	Name	Address	phoneNum
1	1	ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	raza	defence	5173946



Web

Design & Development

Lec - 16



Web

Design & Development

More on JDBC



Web

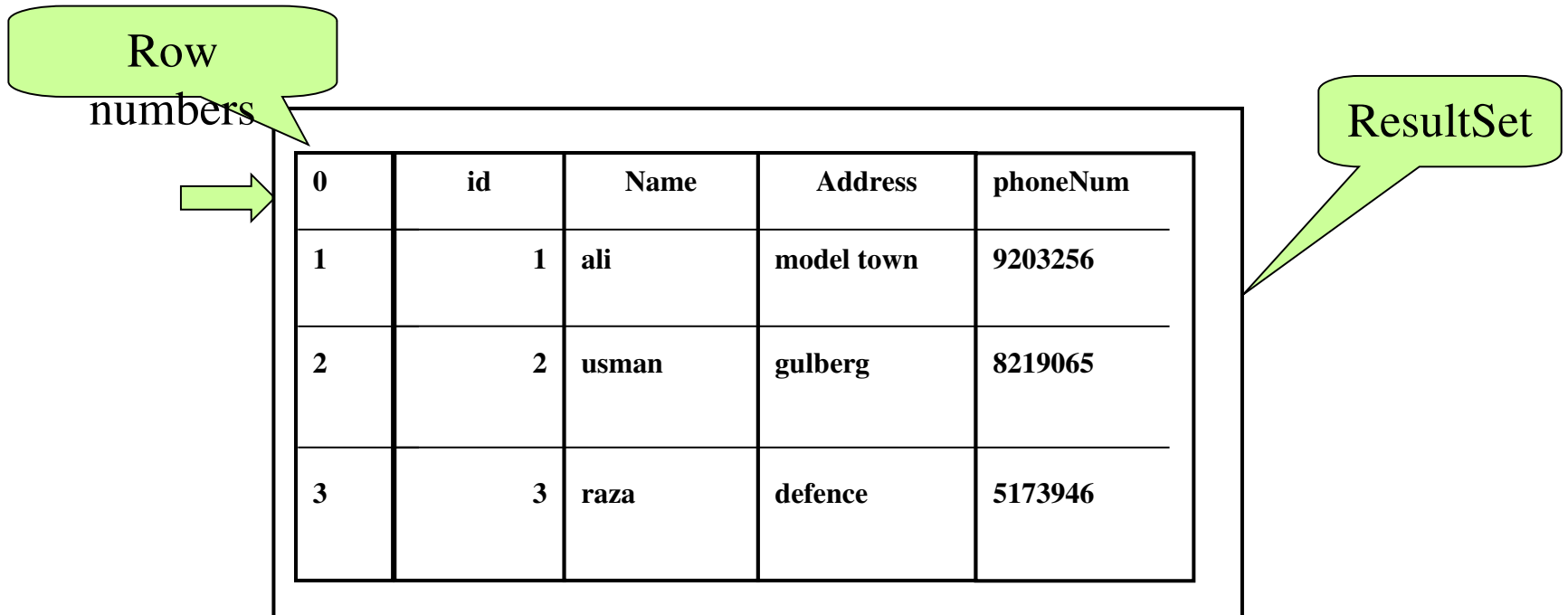
Design & Development

Result Set

ResultSet

- **Overview**
 - A `ResultSet` contains the results of the SQL query
 - Represented by a table with rows and columns
 - Maintains a cursor pointing to its current row of data.
 - Initially the cursor positioned before the row (0).
 - First row has index 1

ResultSet (cont.)



The diagram illustrates a **ResultSet** object containing a table of data. A green speech bubble labeled "Row numbers" points to the first column of the table, which contains the indices 0, 1, 2, and 3. A green arrow points from the "Row numbers" label to the first column. Another green speech bubble labeled "ResultSet" points to the entire table structure. The table has five columns: "id", "Name", "Address", and "phoneNum". The first column is labeled "0" for the first row, "1" for the second, "2" for the third, and "3" for the fourth. The data rows are as follows:

0	id	Name	Address	phoneNum
1	1	ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	raza	defence	5173946

ResultSet (cont.)

- A **default** `ResultSet` object is not updateable and has a cursor that moves forward only
 - You can iterate through it only once and only from the **first row to last row**.

```
String sql = "SELECT * FROM Person";
```

```
PreparedStatement pStmt = con.prepareStatement(sql);
```

```
ResultSet rs = pStmt.executeQuery( );
```

ResultSet (cont.)

- **Useful Methods**

- **next()**

- Attempts to move to the **next row** in the `ResultSet`
 - If successful `true` is returned; otherwise, `false`
 - The first call to `next`, moves the cursor to the first row

- **close()**

- Releases the JDBC and database resources
 - The result set is **automatically closed** when the associated Statement object **executes a new query** or **closed by method call**

ResultSet (cont.)

- **Useful Methods**

- **getters**

- Returns the value from the column specified by the **column name** or **index**

- String name = rs.getString("name");
 - String add = rs.getString(3);
 - double sal = rs.getDouble("Salary")

- Returns the value in a specified format

double	byte	int	Date	String
float	short	long	Time	Object

ResultSet (cont.)

- It is possible to produce `ResultSet` objects that are scrollable and/or updatable (since JDK 1.2).

```
String sql = "SELECT * FROM Person";
```

```
PreparedStatement pStmt = con.prepareStatement( sql,
```

```
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE );
```

```
ResultSet rs = pStmt.executeQuery( );
```

ResultSet (cont.)

- **Useful Methods**

- **previous()**

- Moves the cursor to the previous row in the `ResultSet` object.
 - Returns `true` if cursor is on a valid row, `false` if it is off the result set.
 - Throws exception if result type is `TYPE_FORWARD_ONLY`.

Example Code: ResultSetEx

previous, next & getters methods

```
import java.sql.*;

public class ResultSetEx {

    public static void main ( String args[ ]) {

        try {
            // load driver & make connection

            String sql = "SELECT * FROM Person";

            PreparedStatement pStmt = con.prepareStatement( sql,
                ResultSet.TYPE_SCROLL_INSENSITIVE ,
                ResultSet.CONCUR_UPDATABLE );

            ResultSet rs = pStmt.executeQuery( );
```

Example Code: ResultSetEx

previous, next & getters methods

rs.next();

```
System.out.println("moving cursor forward");  
String name = rs.getString("name");  
System.out.println(name);
```

rs.next();

rs.previous();

```
System.out.println("moving cursor backward");  
name = rs.getString("name");  
System.out.println(name);
```

Example Code: ResultSetEx

previous, next & getters methods

```
        con.close();

    } catch (Exception ex) {
        System.out.println(ex);
    }

}

} // end main

} // end class
```

Compile & Execute

Person : Table				
	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

```
C:\WINDOWS\system32\cmd.exe

D:\examples\jdbc>javac ResultSetEx.java

D:\examples\jdbc>java ResultSetEx
moving cursor forward
ali
moving cursor backward
ali
```

ResultSet (cont.)

- **Useful Methods**

- **absolute(int)**

- Move the cursor to the given row number in the `ResultSet` object
 - If the row number is positive, moves the cursor forward with respect to beginning of the result set.
 - If the given row number is **negative**, the cursor moves to the absolute row position with respect to the end of the result set.
 - For example, calling `absolute(-1)` positions the cursor on the last row; calling `absolute(-2)` moves the cursor to next-to-last row, and so on.
 - Throws exception if result type is `TYPE_FORWARD_ONLY`.

ResultSet (cont.)

- **Useful Methods**

- **updaters** (for primitives, Object & String)

- Used to update column values in the current row or the *insert row*.

- Do not update the underlying database

- Each update method is overloaded.

- For example of String

- `updateString(String columnName, String value)`

- `updateString(int columnIndex, String value)`

Umais Javed©2005

ResultSet (cont.)

- **Useful Methods**
 - **updateRow()**
 - Updates the underlying database with new contents of the current row of this `ResultSet` object.

Modify Example : ResultSetEx

updating existing rows

```
Import java.sql.*;
public class ResultSetEx {
    ..... // main method
    ..... Load driver, make connection
    ..... Make updatable resultset

    //move cursor to 2nd row of rs
    rs.absolute(2);

    //update address column of 2nd row in rs
    rs.updateString("address", "model town");

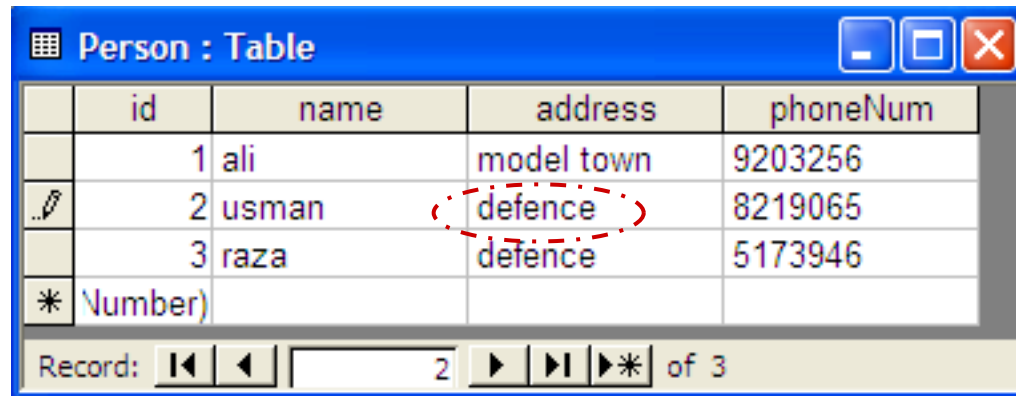
    //update the row in database
    rs.updateRow();

    ..... // close connection etc

    .... //end main
} // end class
```

Compile & Execute

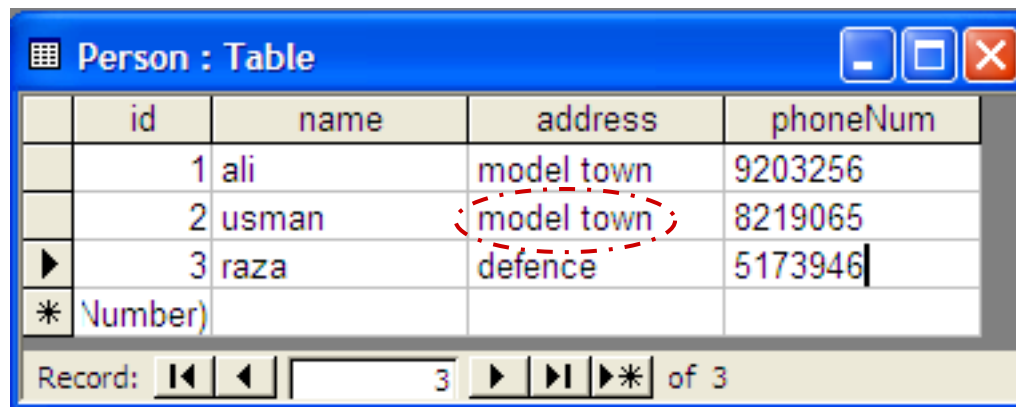
Before execution



	id	name	address	phoneNum
	1	ali	model town	9203256
	2	usman	defence	8219065
	3	raza	defence	5173946

Record: 2 of 3

After execution



	id	name	address	phoneNum
	1	ali	model town	9203256
	2	usman	model town	8219065
	3	raza	defence	5173946

Record: 3 of 3

ResultSet (cont.)

- **Useful Methods**

- **moveToInsertRow()**

- An **Updatable** `ResultSet` object has a special row associated with it i.e. insert row.
 - Insert row – a buffer, where a new row may be constructed by calling the updater methods
 - **Doesn't** insert row into a result set or into a database

ResultSet (cont.)

- **Useful Methods**

- **insertRow()**

- Inserts the contents of the insert row into this `ResultSet` object and into the database.
 - The cursor must be on the insert row when this method is called

ResultSet (cont.)

`moveToInsertRow( )`

Row
numbers



0	id	Name	Address	phoneNum
1	1	ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	raza	defence	5173946

Updatable
ResultSet

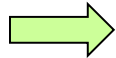
	imitiaz	cantt	9201211

Insert Row

ResultSet (cont.)

insertRow()

Row
numbers



0	id	Name	Address	phoneNum
1	1	ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	raza	defence	5173946
4	4	imitiaz	cantt	9201211

Updatable
ResultSet

Insert Row

Modify Example : ResultSetEx

Inserting new row

```
Import java.sql.*;
public class ResultSetEx {
    .... // main method
        .... Load driver, make connection
        .... Make updatable resultset

        //move cursor to insert row
        rs.moveToInsertRow();

        // updating values into insert row
        rs.updateString("name", "imitaz");
        rs.updateString("address", "cantt");
        rs.updateString("phoneNum", "9201211");

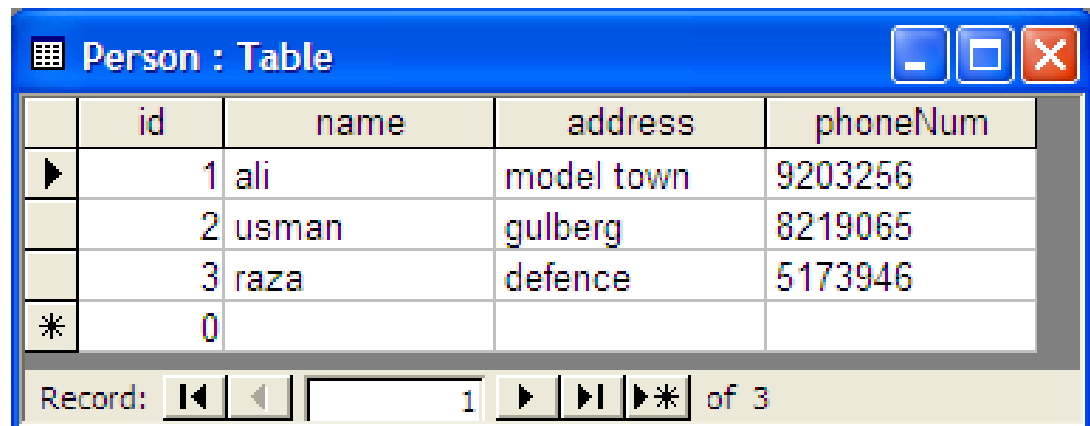
        //insert row into rs & db
        rs.insertRow();

        .....

    .... //end main
} // end class
```


Compile & Execute

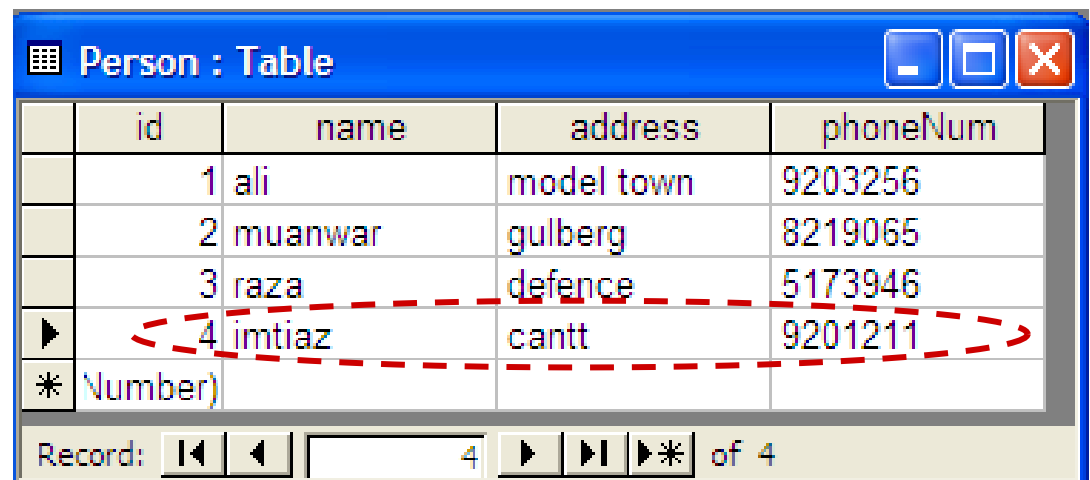
Before execution



	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

After execution



	id	name	address	phoneNum
	1	ali	model town	9203256
	2	muanwar	gulberg	8219065
	3	raza	defence	5173946
▶	4	imtiaz	cantt	9201211
*	Number)			

Record: 4 of 4

ResultSet (cont.)

- **Useful Methods**

- **last() & first()**

- Moves the cursor to the last & first row of the `ResultSet` object respectively.
 - Throws exception if the result set is `TYPE_FORWARD_ONLY`

- **getRow()**

- Returns the current row number
 - The first row number is 1, second row number is 2 and so on

ResultSet (cont.)

- **Useful Methods**

- **deleteRow()**

- Deletes the current row from this `ResultSet` object and from the underlying database.
 - Throws exception when the cursor is on the insert row

Modify Example : ResultSetEx

deleting existing row

```
Import java.sql.*;
public class ResultSetEx {
    ..... // main method
    ..... Load driver, make connection
    ..... Make updatable resultset

    //moves to last row
    rs.last( );

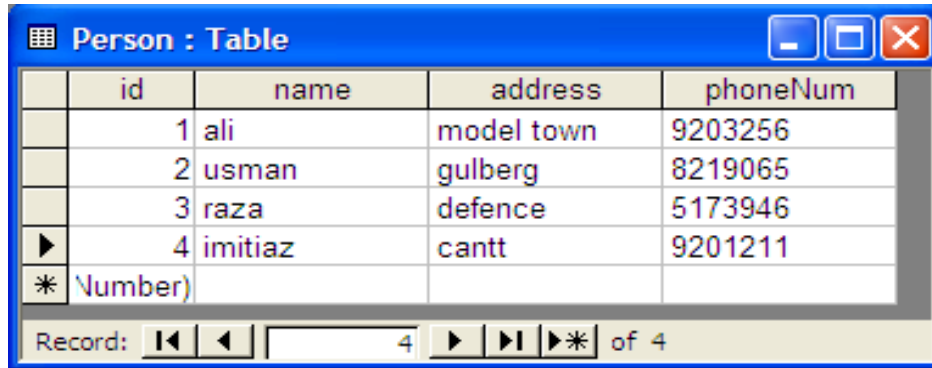
    int rNo = rs.getRow();
    System.out.println("curr row no: "+ rNo );

    //delete current row (4) from rs & db
    rs.deleteRow();

    .....

    .... //end main
} // end class
```

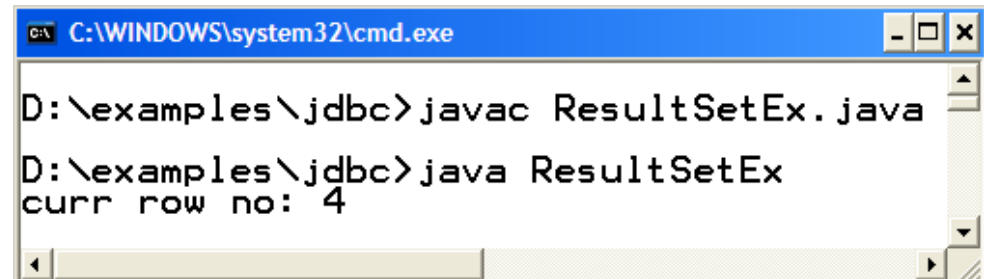
Compile & Execute



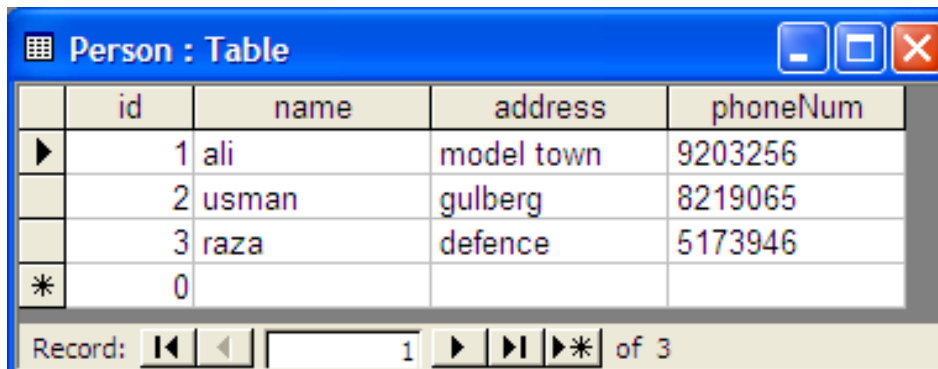
	id	name	address	phoneNum
	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
▶	4	imitiaz	cantt	9201211
*	Number)			

Record: 4 of 4

Before execution



```
C:\WINDOWS\system32\cmd.exe
D:\examples\jdbc>javac ResultSetEx.java
D:\examples\jdbc>java ResultSetEx
curr row no: 4
```



	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

After execution



Web

Design & Development

Lec - 17



Web

Design & Development

More on JDBC



Web

Design & Development

Meta Data

Meta Data

- **What if you want to know:**
 - How many columns are in the result set?
 - What is the name of a given column?
 - Are the column names case sensitive?
 - What is the data type of a specific column?
 - What is the maximum character size of a column?
 - Can you search on a given column?

Using ResultSetMetaData

- **Idea**
 - From a `ResultSet` (the return type of `executeQuery`), derive a `ResultSetMetaData` object
 - Use that object to look up the number, names, and types of columns

Useful ResultSetMetaData Methods

- **getColumnCount ()**
 - Returns the number of columns in the result set
- **getColumnDisplaySize (int)**
 - Returns the maximum width of the specified column in characters
- **getColumnName(int) / getColumnLabel (int)**
 - The `getColumnName` method returns the database name of the column
 - The `getColumnLabel` method returns the suggested column label for printouts
- **getColumnType (int)**
 - Returns the SQL type for the column to compare against types in `java.sql.Types`

Example Code: MetaDataEx using ResultSetMetaData

```
import java.sql.*;

public class MetaDataEx {

    public static void main ( String args[ ]) {
        try {

            Class.forName("Driver name");

            Connection con = DriverManager.getConnection(url, usr, pwd);

            String sql = "SELECT * FROM Person";
            PreparedStatement pStmt = con.prepareStatement(sql);

            ResultSet rs = pStmt.executeQuery( );
```

Example Code: MetadataEx (cont.) using ResultSetMetaData

```
ResultSetMetaData rsmd = rs.getMetaData();
```

```
int numColumns = rsmd.getColumnCount();
```

```
System.out.println("Number of Columns:" + numColumns);
```

```
String cName;
```

```
for (int i=1; i <= numColumns; i++)
```

```
{
```

```
    cName = rsmd洗getColumnName(i);
```

```
    System.out.print(cName);
```

```
    System.out.print("\t");
```

```
}
```

```
// changing line
```

```
System.out.println("");
```

Example Code: MetaDataEx (cont.) using ResultSetMetaData

```
String id, name, add, ph;  
while (rs.next())  
{  
    id    = rs.getString(1);  
    name  = rs.getString(2);  
    add   = rs.getString(3);  
    ph    = rs.getString(4);  
  
    System.out.print(id);  
  
    System.out.print("\t");  
    System.out.print(name);  
  
    System.out.print("\t");  
    System.out.print(add);  
  
    System.out.print("\t");  
    System.out.print(ph);  
  
    System.out.println("");  
}
```

Example Code: MetaDataEx (cont.)

using ResultSetMetaData

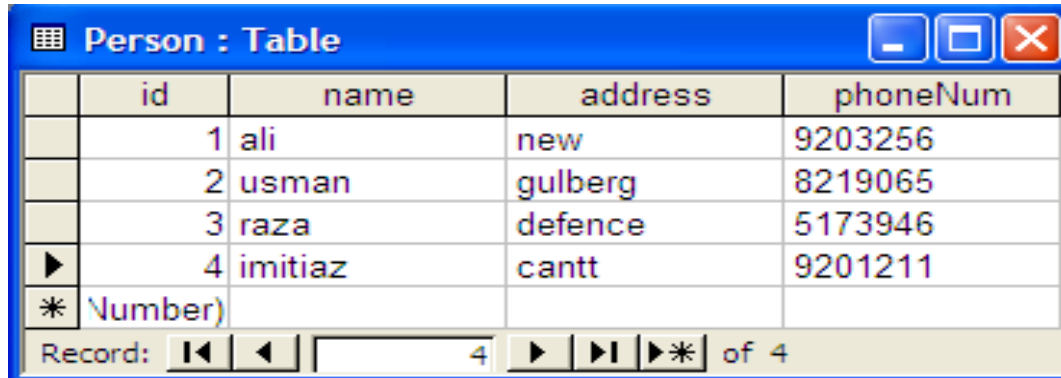
```
con.close();

} catch (Exception ex) {
    System.out.println(ex);
}

} // end main
} // end class
```

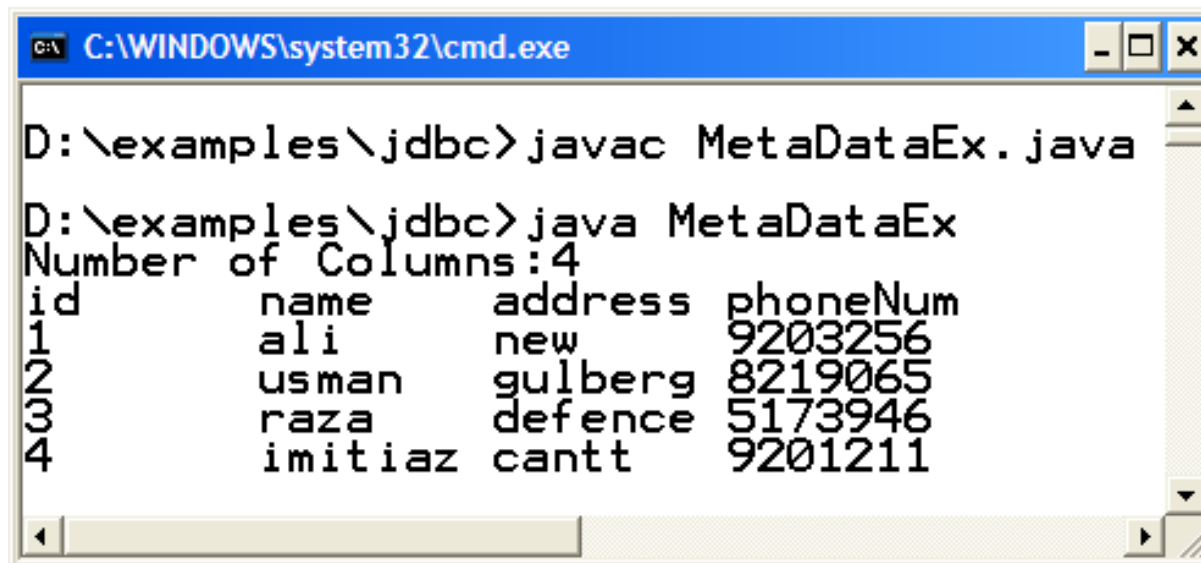
Example Code: MetaDataEx

Compile & Execute



	id	name	address	phoneNum
	1	ali	new	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
▶	4	imitiaz	cantt	9201211
* Number)				

Record: 4 of 4



```
C:\WINDOWS\system32\cmd.exe

D:\examples\jdbc>javac MetaDataEx.java

D:\examples\jdbc>java MetaDataEx
Number of Columns:4
id      name      address  phoneNum
1       ali       new      9203256
2       usman     gulberg  8219065
3       raza      defence  5173946
4       imitiaz   cantt    9201211
```


Database MetaData

- **What if we want to know**
 - What SQL types are supported by DBMS to create table?
 - What is the name of a database product?
 - What is the version number of this database product?
 - What is the name of the JDBC driver that is used?
 - Is the database in a read-only mode?

Using DatabaseMetaData

- **Idea**
 - From a `Connection`, derive a `DatabaseMetaData` object
 - Contains the comprehensive information about the database as a whole

Using DatabaseMetaData

- **Idea**
 - From a `Connection`, derive a `DatabaseMetaData` object
 - Contains the comprehensive information about the database as a whole

Useful DataBaseMetaData Methods

- **getDatabaseProductName ()**
 - Returns the name of the database product name
- **getDatabaseProductVersion ()**
 - Returns the version number of this database product
- **getDriverName()**
 - Returns the name of the JDBC driver used to established the connection
- **isReadOnly ()**
 - Retrieves whether this database is in read-only mode.
 - Returns `true` if so, `false` otherwise.

Example Code: Modify MetadataEx using DataBaseMetaData

```
import java.sql.*;

public class MetadataEx {

    public static void main ( String args[ ]) {
        try {

            Class.forName("Driver name");

            Connection con = DriverManager.getConnection(url, usr, pwd);

            DatabaseMetaData dbMetadata = con.getMetaData();

            .....
```

Example Code: Modify MetadataEx using DataBaseMetaData

```
String pName = dbMetaData.getDatabaseProductName();  
System.out.println("Database: " + pName);
```

```
String pVer = dbMetaData.getDatabaseProductVersion();  
System.out.println("Version: " + pVer);
```

```
String dName = dbMetaData.getDriverName();  
System.out.println("Driver: " + dName);
```

```
boolean rOnly = dbMetaData.isReadOnly();  
System.out.println("Read-Only: " + rOnly);
```

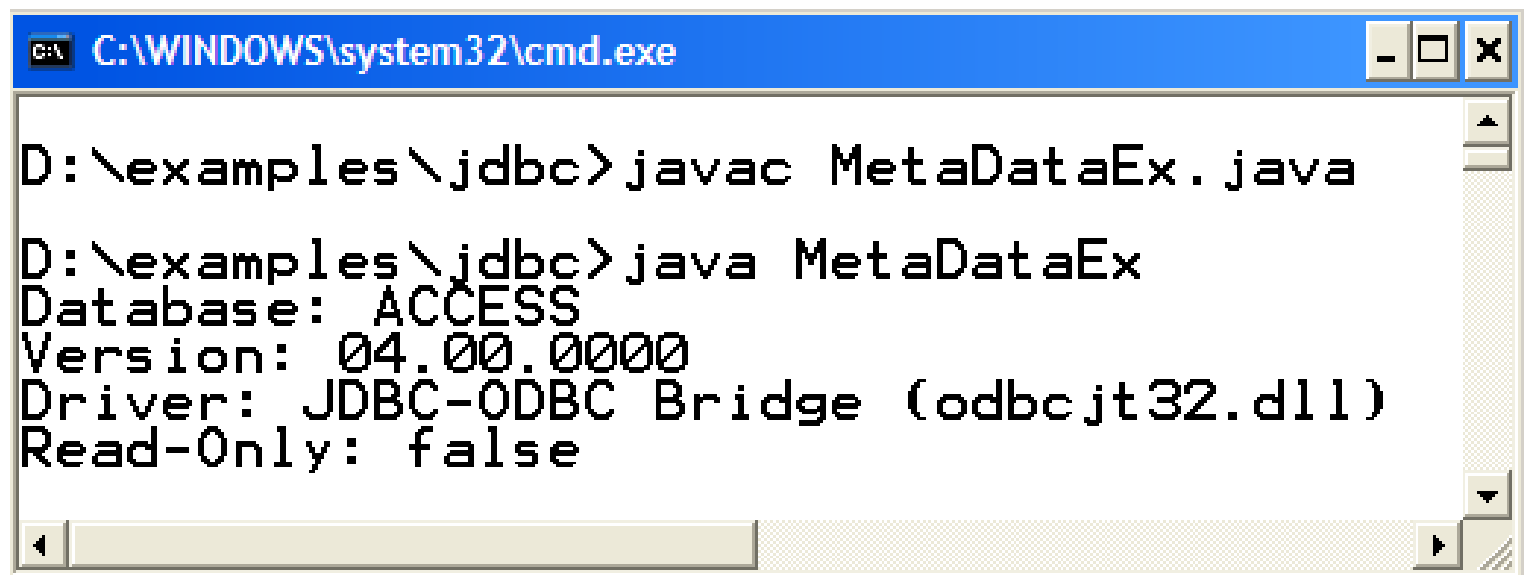
.....

Example Code: Modify MetaDataEx using DataBaseMetaData

```
// create Statement & execute query  
  
// process results  
  
con.close();  
  
}catch ( Exception ex) {  
    System.out.println(ex);  
}  
} // end main  
} // end class
```

Example Code: Modify MetaDataEx

Compile & Execute



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" along with standard window control buttons (minimize, maximize, close). The command prompt shows the following sequence of commands and output:

```
D:\examples\jdbc>javac MetaDataEx.java  
D:\examples\jdbc>java MetaDataEx  
Database: ACCESS  
Version: 04.00.0000  
Driver: JDBC-ODBC Bridge (odbcjt32.dll)  
Read-Only: false
```

The output displays the database name, version, driver, and read-only status. The window has a scroll bar on the right side.



Web

Design & Development

RowSet

RowSet

- A JDBC RowSet object holds tabular data in a way that makes it more flexible and easier to use than a result set.
- Interface RowSet configures the database connection and prepares query statements automatically.
- It is part of package **javax.sql**.
- It is part of J2SE, but it is normally used in the context of J2EE.

RowSet (cont.)

There are two kinds of RowSet objects:

- **Connected**
 - Makes the connection to the database and stays connected until the application ends
- **Disconnected**
 - Connects, queries the database, then closes.
 - Connection can be reestablished for updates.

RowSet (cont.)

JDBC provides the five versions of the RowSets. Two of them are:

1. JdbcRowSet

- **Connected** RowSet that wraps a ResultSet object, allowing scrolling and updating.
- It is most similar to a ResultSet object.

RowSet (cont.)

2. CachedRowSet

- **Disconnected** RowSet that is scrollable and updateable.
- It caches the data of a ResultSet in memory.
- Manipulate data and make changes to data while it is disconnected.
- Reconnect to the data source to write changes back to it.
- It is also **serializable**, so it can be sent across a network.



Web

Design & Development

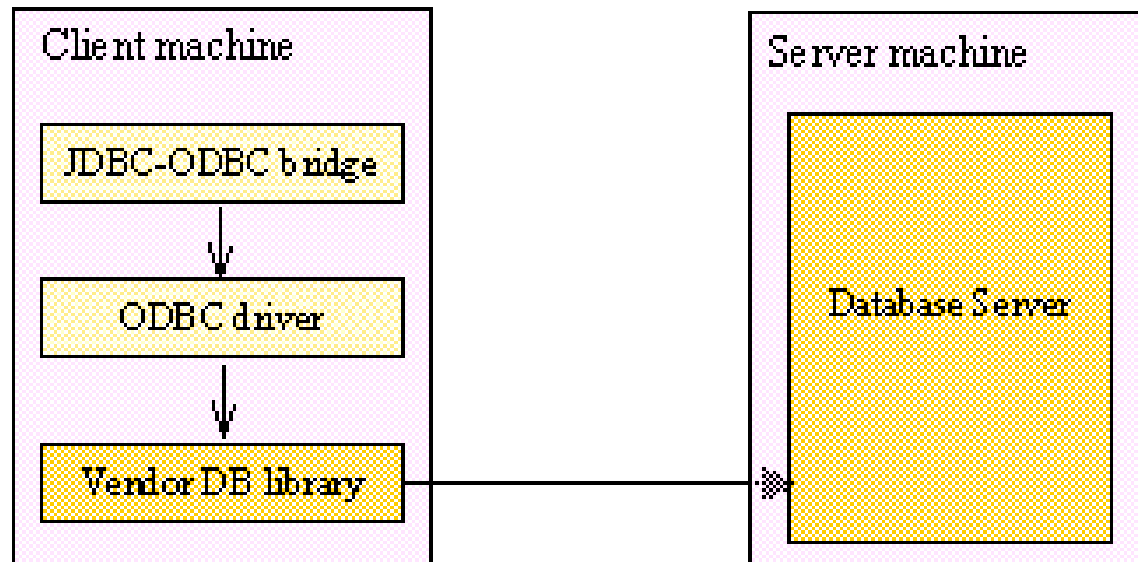
JDBC Drivers Types

JDBC Driver Types

- **JDBC drivers are divided into four types or levels.**
- **Each type defines a JDBC driver implementation with increasingly higher level of platform independence, performance, deployment and administration.**
- **The four types are:**
 1. Type 1: JDBC – ODBC Bridge
 2. Type 2: Native – API/partly Java driver
 3. Type 3: Net – protocol/all–Java driver
 4. Type 4: Native – protocol/all–Java driver

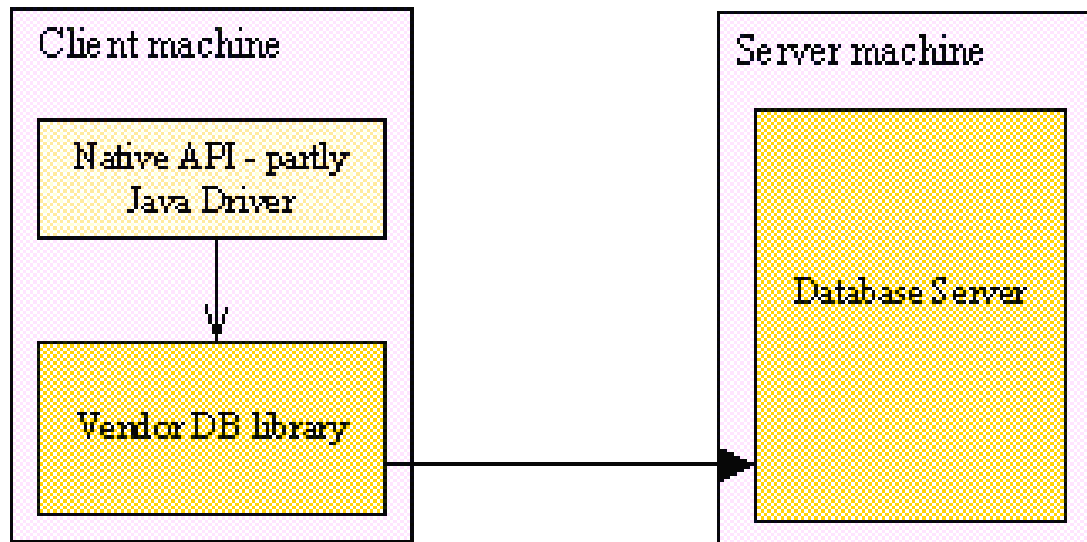
JDBC Driver Types (cont.)

1. Type 1: JDBC – ODBC Bridge



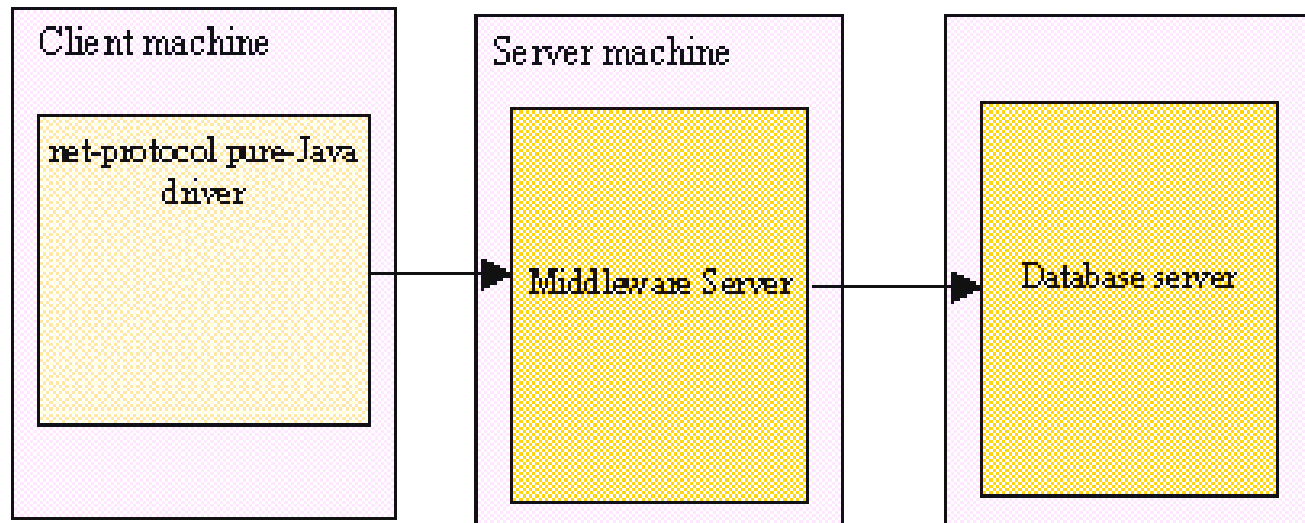
JDBC Driver Types (cont.)

2. Type 2: Native – API/partly Java driver



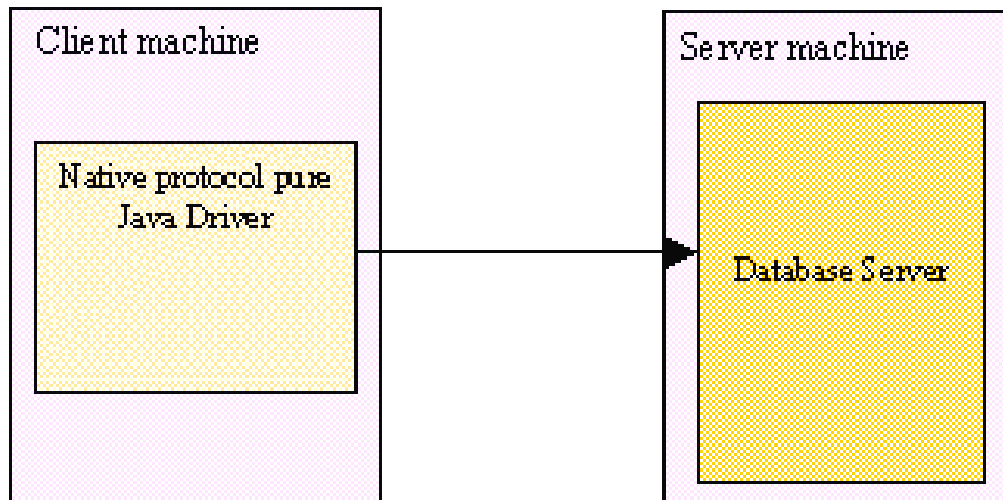
JDBC Driver Types (cont.)

3. Type 3: Net – protocol/all–Java driver



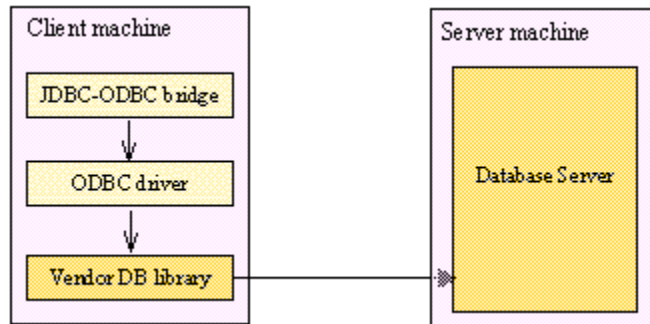
JDBC Driver Types (cont.)

4. Type 4: Native – protocol/all–Java driver

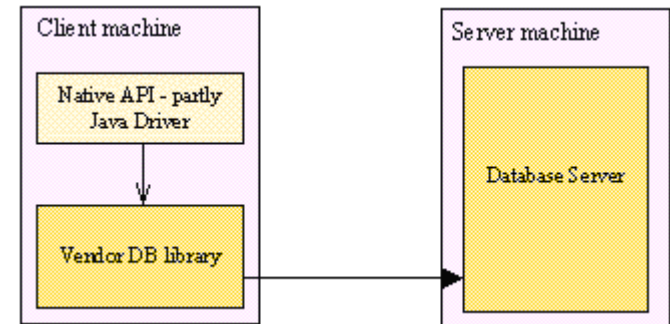


Summary of Driver Types

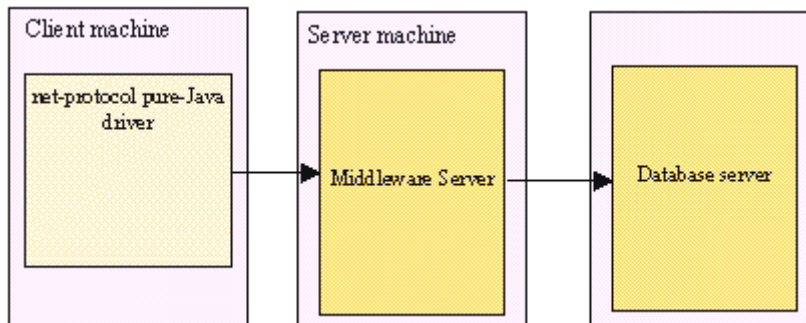
Type 1



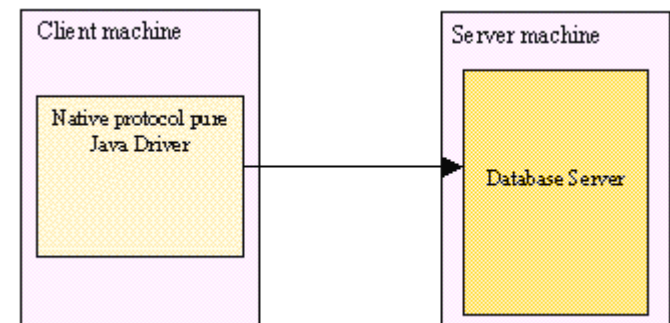
Type 2



Type 3



Type 4



General Design Guideline

```
class Employee {  
  
    String name;  
    String sal;  
  
    // constructor  
    // getter / setters  
  
    void update( ){  
        // connect database  
        // execute query  
        // process results  
    }  
  
    void insertEmp ( ){ ..... }  
  
    void calculateTax ( ) { .... }  
  
}
```

**Database connectivity & business logic
all in one class**

General Design Guideline

Business Logic

```
class Employee {  
  
    String name;  
    String sal;  
  
    // constructor  
    // getter / setters  
  
    void update( ){  
  
        DAO dao = new DAO();  
        dao.update(name, sal);  
    }  
  
    void insertEmp ( ){ ..... }  
  
    void calculateTax ( ) { .... }  
  
}
```

Database Connectivity

```
//step 1  
public class DAO {  
    public DAO ( ) {  
        //step 2 - 5  
        .....  
        .....  
    }  
  
    public void update (String n, String s)  
    {  
        //step 6 -7  
        .....  
        .....  
    }  
  
    public void insert(....){.....}  
  
    protected void finalize() {  
        //step 8 - close connection  
    }  
}
```

On-line Resources

- **Sun's JDBC Site**
 - <http://java.sun.com/products/jdbc/>
- **JDBC Tutorial**
 - <http://java.sun.com/docs/books/tutorial/jdbc/>
- **List of Available JDBC Drivers**
 - <http://industry.java.sun.com/products/jdbc/drivers/>
- **API for java.sql**
 - <http://java.sun.com/j2se/1.3/docs/api/java/sql/package-summary.html>

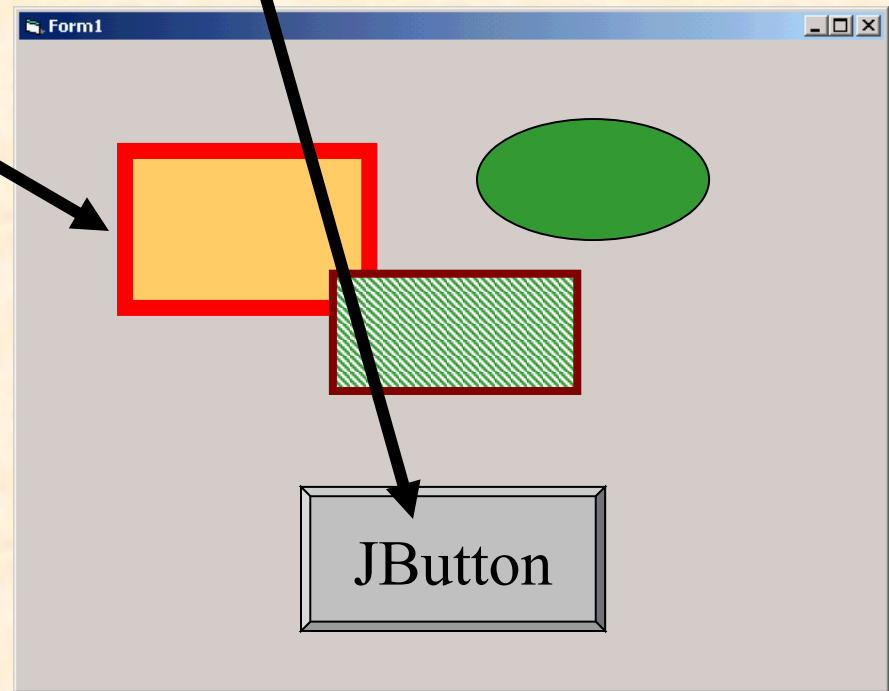
Web Design & Development

Lecture 18

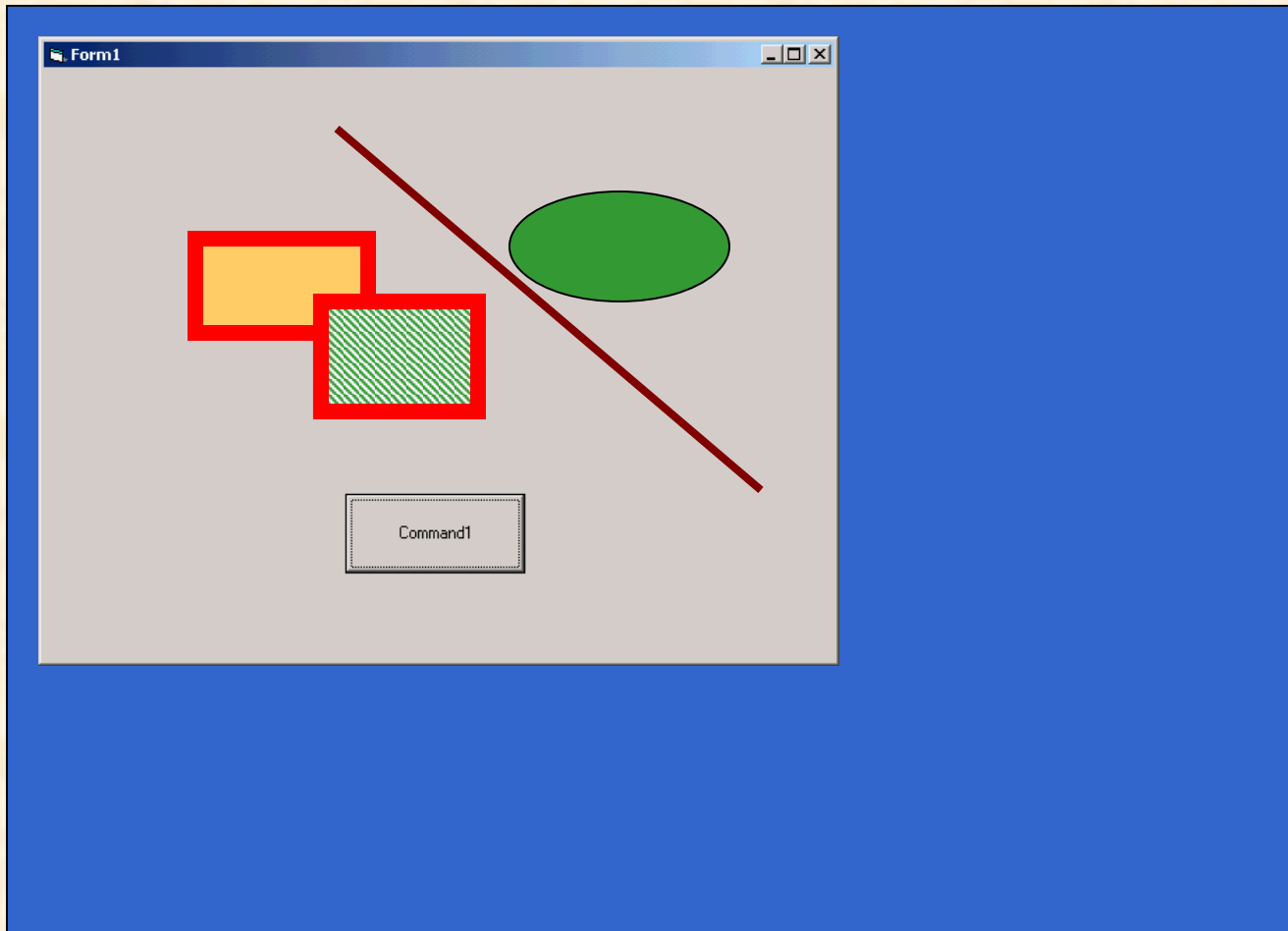
Java Graphics

Graphics

- Window is like a painter's canvas
- App must paint its window contents
 - Java components paint themselves
 - Anything else: Programmer
- When to paint?
- How to paint?

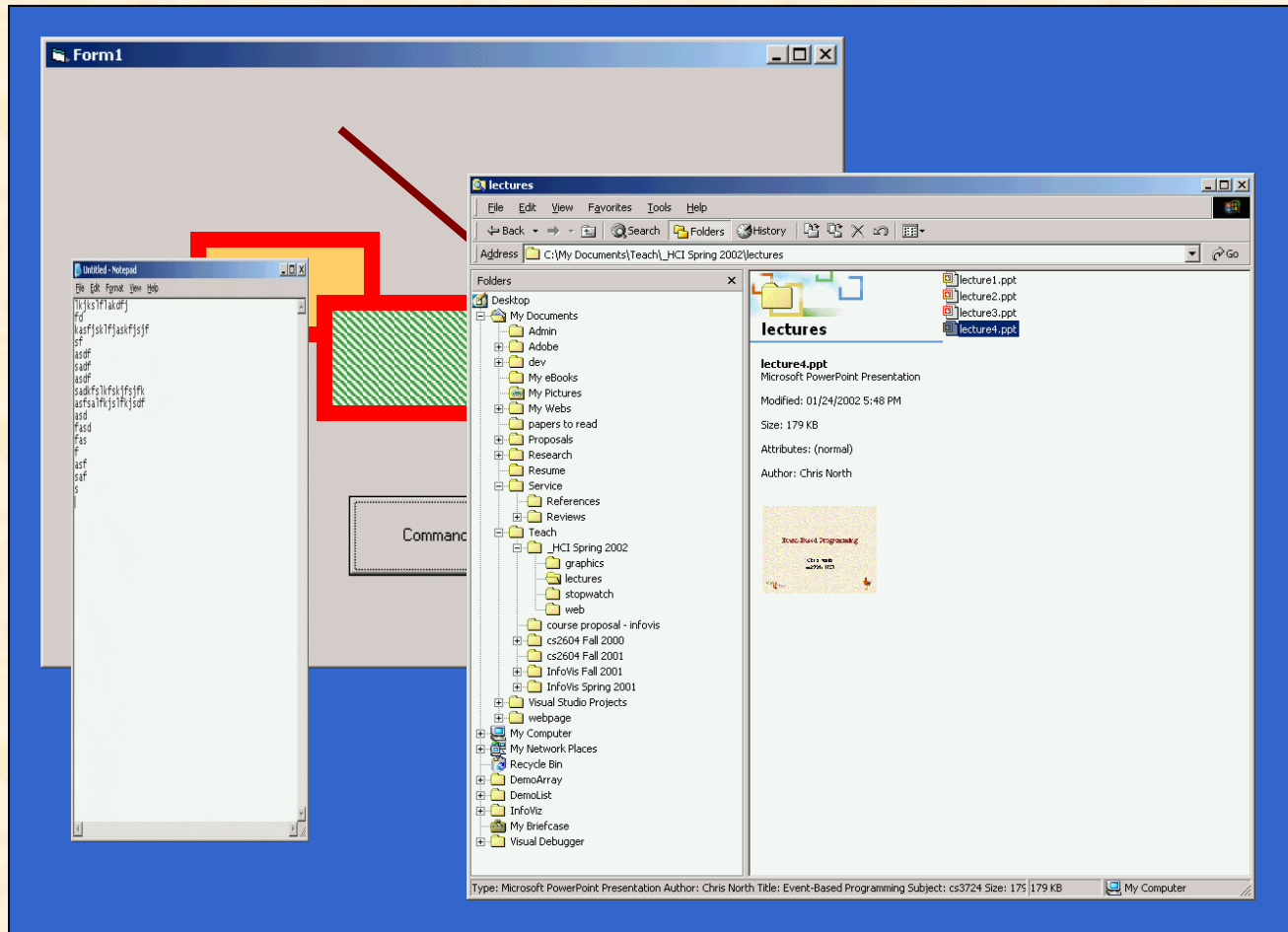


MyApp



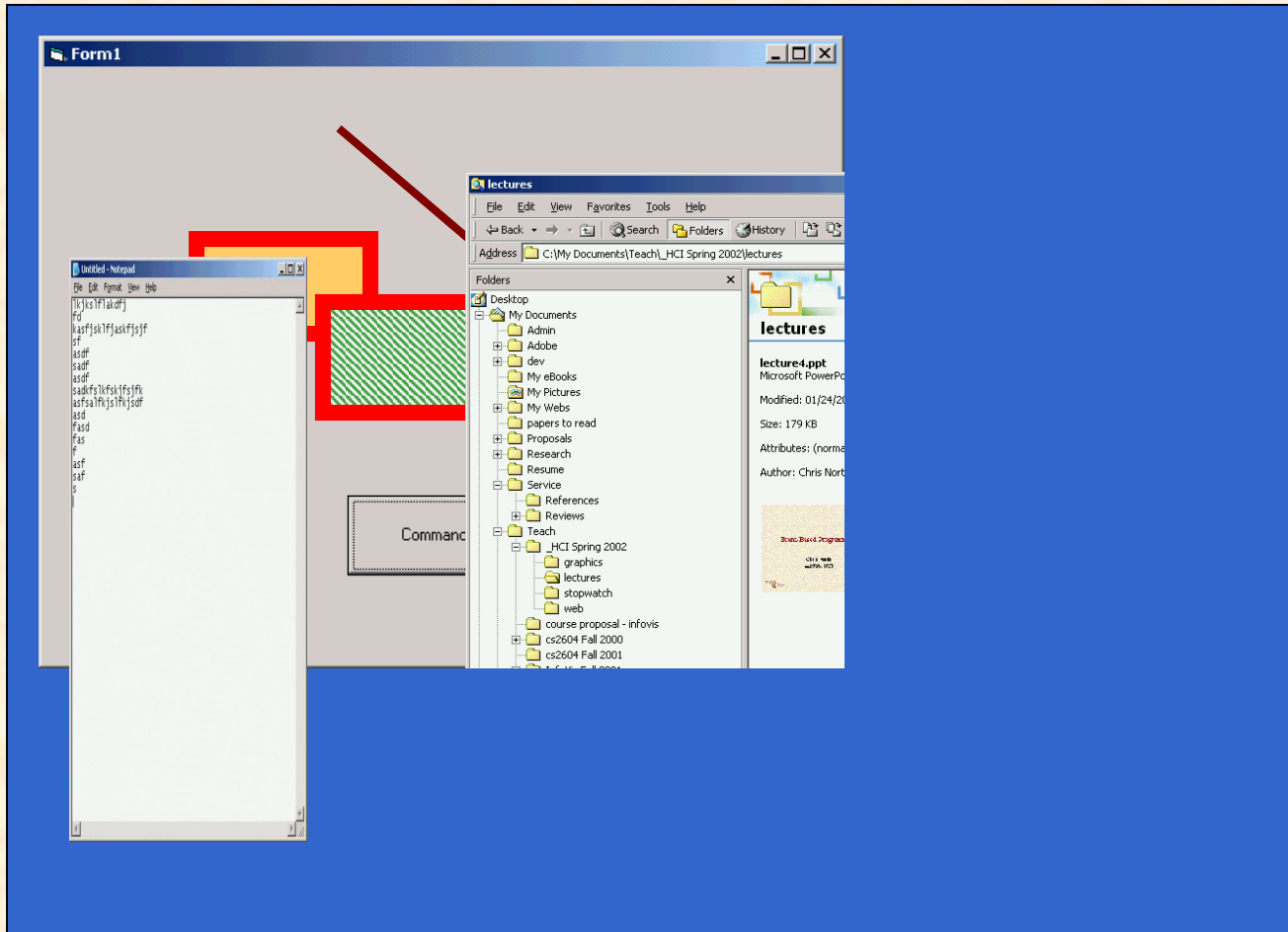
The screenshot shows a Windows XP desktop with three windows. The 'Form1' window is in the background, partially obscured. The 'Untitled - Notepad' window is in the foreground, showing a text editor with a green hatched area. The 'lectures' window is open, displaying a file explorer view of the 'C:\My Documents\Teach\HCI Spring 2002\lectures' directory. The file list shows four PowerPoint files: 'lecture1.ppt', 'lecture2.ppt', 'lecture3.ppt', and 'lecture4.ppt'. The details pane on the right shows information for 'lecture4.ppt', including its size (179 KB) and author (Chris North). A red arrow points from the 'Form1' window to the 'lectures' window. A red box highlights a green hatched area in the 'Form1' window. A red box highlights the 'lectures' window's title bar and menu bar. A red box highlights the 'lectures' window's address bar and search bar. A red box highlights the 'lectures' window's file list. A red box highlights the 'lectures' window's details pane.

Close WinExplorer

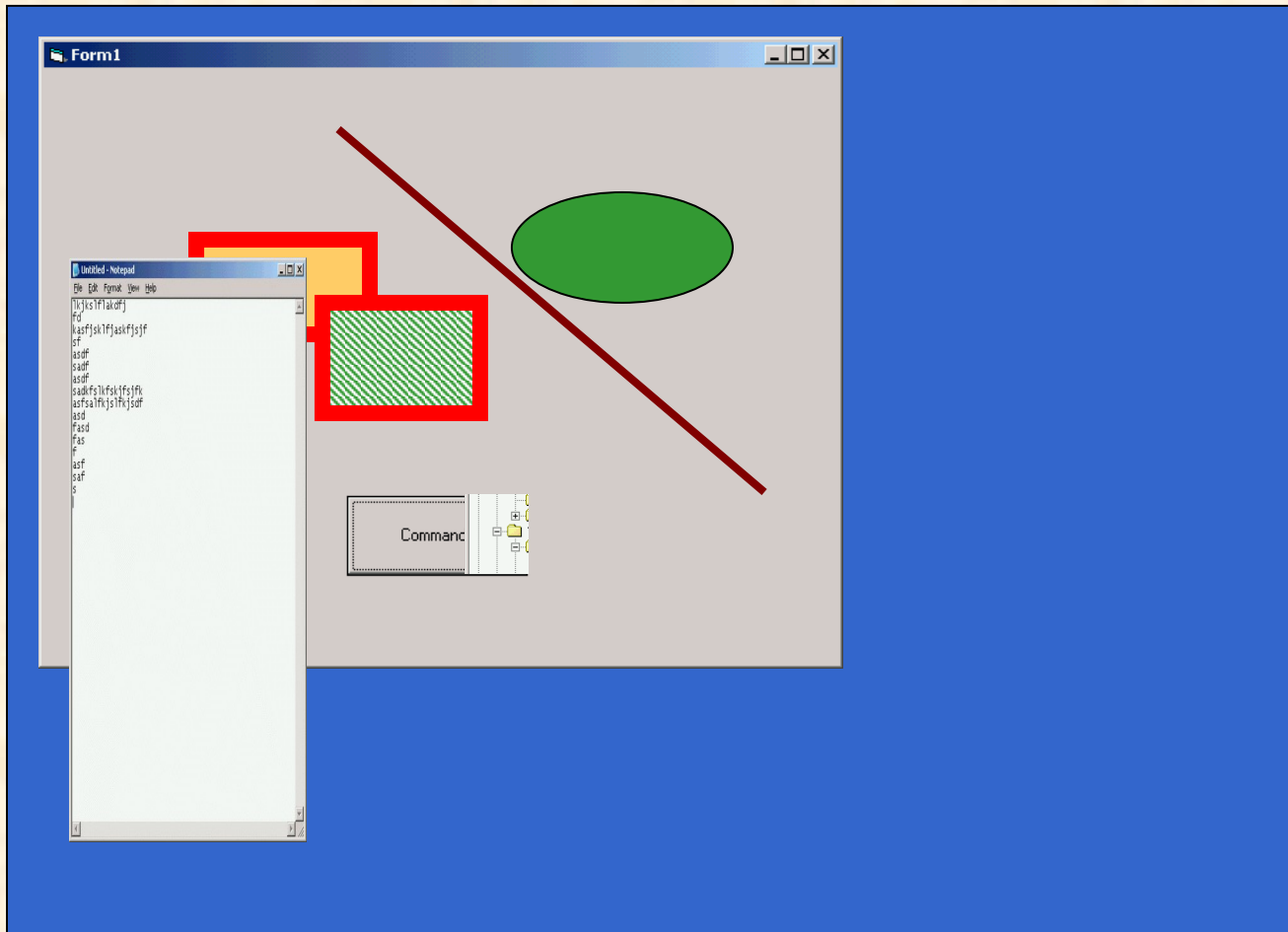


Repaint event sent to: MyApp, Desktop

Desktop gets repaint event

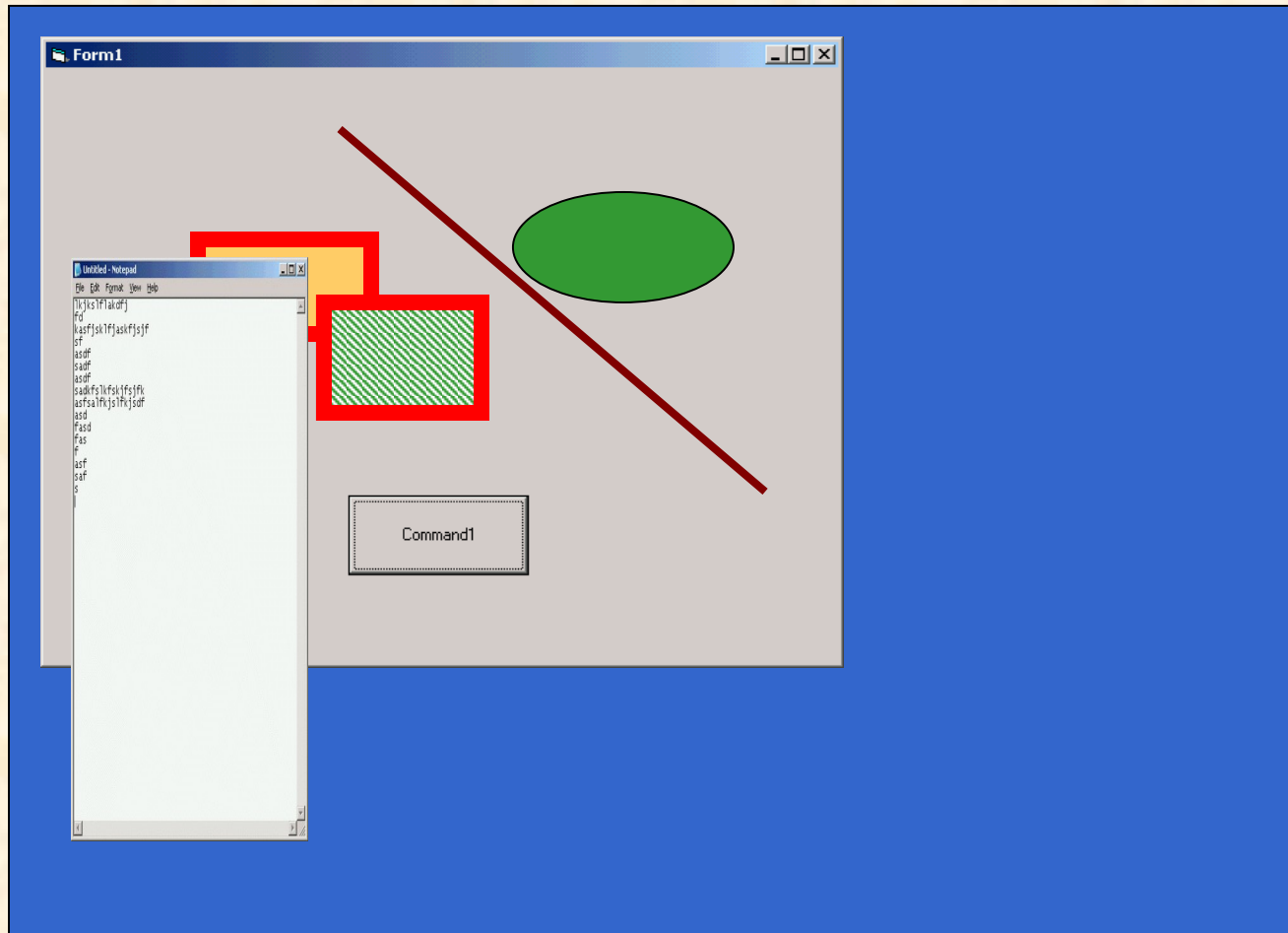


MyApp gets repaint event



MyApp JPanel gets repaint event

MyApp gets repaint event



MyApp JPanel forwards repaint event to JButton

Painting a Swing Component

- Three methods are at the heart of painting a swing component
 - `paintComponent()`
 - `paintBorder()`
 - `paintChildren()`

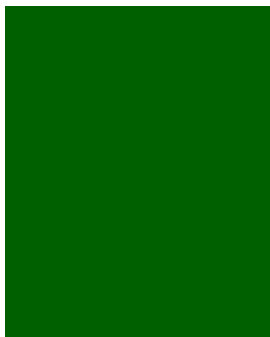
The Swing Painting Methods

- It's one of three methods, that objects of type JComponent use to paint themselves. The three methods are invoked in this order:
 - paintComponent — The main method for painting. By default, it first paints the background. Then it performs any custom painting.
 - paintBorder — Tells the component's border (if any) to paint. *Do not invoke or override this method.*
 - paintChildren — Tells any components contained by this component to paint themselves. *Do not invoke or override this method*

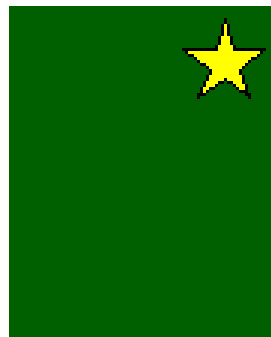
The following figure illustrates the order in which each component that inherits from JComponent paints itself.

Steps 1 and 2 — painting the background and performing custom painting — are performed by the `paintComponent` method. Step 3 is performed by `paintBorder`, and step 4 is performed by `paintChildren`

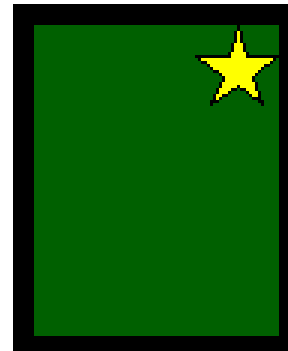
1. background
(if opaque)



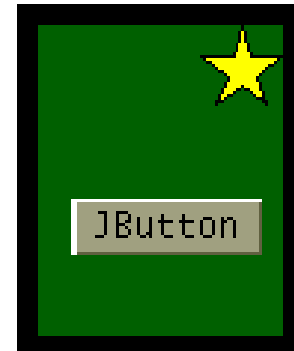
2. custom painting
(if any)



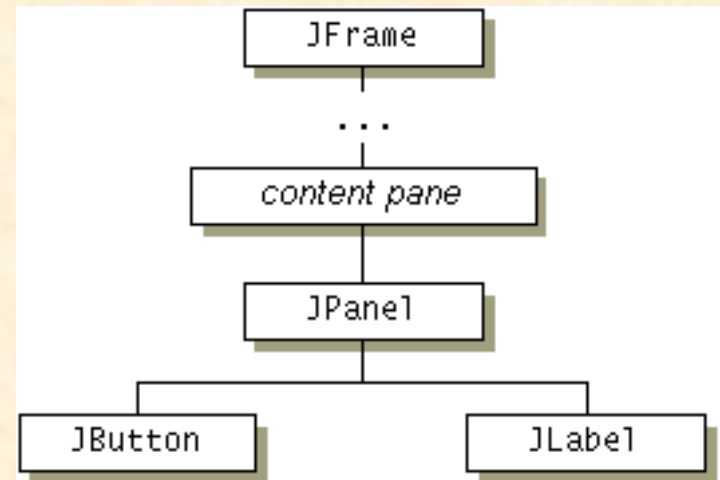
3. border
(if any)



4. children
(if any)



- **An Example of Painting**
- To illustrate painting, we'll use the swingApplication program. Here is its GUI:



Your Painting Strategy

- Steps
 - Subclass JPanel
 - class MyPanel extends JPanel
 - Override the paintComponent(Graphics g) method
 - Inside method using graphics object Do whatever drawing you want to do
 - Install that JPanel inside a JFrame
 - When frame becomes visible through the paintChildren() method your panel become visible
 - To become visible your panel will call paintComponent() method which will do your custom drawing

Example Code: MyPan1.java

```
import javax.swing.*;
import java.awt.*;

public class MyPan1 extends JPanel {
    public void paintComponent(Graphics g){

        super.paintComponent(g); // erasing behaviour

        Graphics2D g2 = (Graphics2D)g;

        g2.drawRect(20,20,20,20);
        g2.setColor(Color.blue);

        g2.fillOval(50,50,20,20);

        g2.drawString("Hello World", 120, 50);

    }
}
```

Example Code: Test1.java

```
import javax.swing.*;
import java.awt.*;

public class Test1{

    JFrame f;
    MyPan1 p;

    public Test1(){
        f = new JFrame();
        Container c = f.getContentPane();

        c.setLayout(new BorderLayout());

        p = new MyPan1();
        c.add(p);

        f.setSize(400,400);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } // end constructor
```


Example Code: Test1.java

```
public static void main(String args[ ]){
```

```
    Test1 t = new Test1();
```

```
}
```

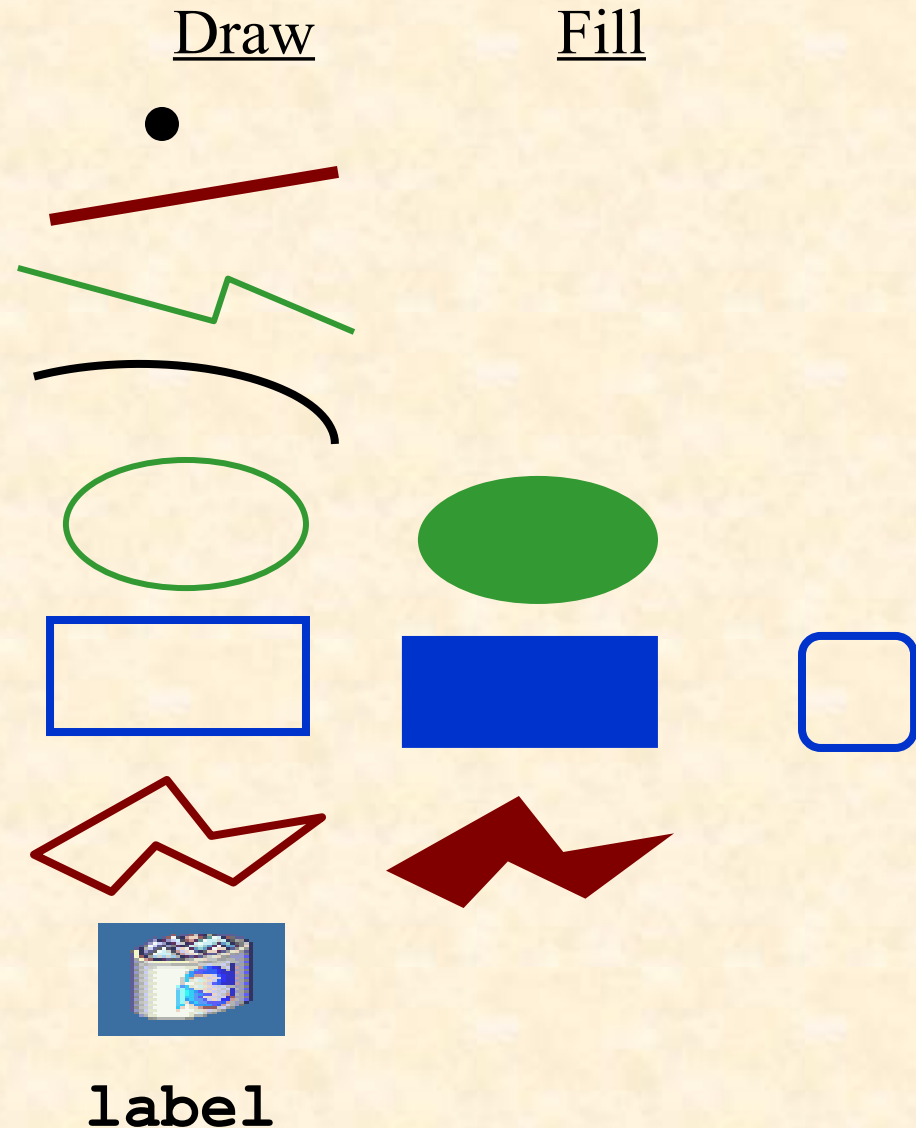
```
} // end class
```


Output



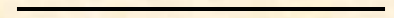
Graphics Primitives

- Point (x,y)
- Line (pt1,pt2)
- PolyLine (pt list)
- Arc
- Oval (pt, w,h)
- Rectangle (pt, w,h)
 - RoundedRectangle
- Polygon (pt list)
- Image (file, x,y)
- Text (string, x,y)



Graphics Attributes

- Color
- Font
- Stroke attributes:
 - Line width, dash, end caps, joins, miter
- Paint attributes:
 - Color, gradient, texture
- Composite:
 - Blending
- Transforms:
 - Translate, rotate, flip, shear, scale



Color

- Combinations of Red, Green, Blue
- Each $[0, 255]$
- Java: *new Color(255, 150, 0)*

Hokie Orange

Web Design & Development

Lecture 19

Java Graphics 2

When to Paint?

Understanding Repaint Cycle

Your Painting Strategy

- Steps
 - Subclass JPanel
 - `class MyPanel extends JPanel`
 - Override the `paintComponent(Graphics g)` method
 - Inside method do what ever you want to do by using graphics object.
 - Install that JPanel inside a JFrame
 - When frame becomes visible and its `paintChildren()` gets called your panel will become visible.
 - To become visible your panel calls `paintComponent()` method on itself which will do your custom drawing

Last Example Code: MyPanel.java

```
import javax.swing.*;
import java.awt.*;

public class MyPanel extends JPanel {
    public void paintComponent(Graphics g){

        super.paintComponent(g); // erasing behaviour

        Graphics2D g2 = (Graphics2D)g;

        g2.drawRect(20,20,20,20);
        g2.setColor(Color.blue);

        g2.fillOval(50,50,20,20);

        g2.drawString("Hello World", 120, 50);

    }
}
```

Last Example Code: Test.java

```
import javax.swing.*;
import java.awt.*;

public class Test{

    JFrame f;
    MyPanel p;

    public Test1(){
        f = new JFrame();
        Container c = f.getContentPane();

        c.setLayout(new BorderLayout());

        p = new MyPanel();
        c.add(p);

        f.setSize(400,400);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } // end constructor
```

Last Example Code: Test.java

```
public static void main(String args[ ]){  
    Test1 = new Test();  
}  
  
} // end class
```

How to Animate

- Constantly need to call `paintComponent()` and draw the shape at new place (new co-ordinates)
- Painting is managed by system and calling `paintComponent()` directly is not recommended at all
- Use `Repaint()` mechanism

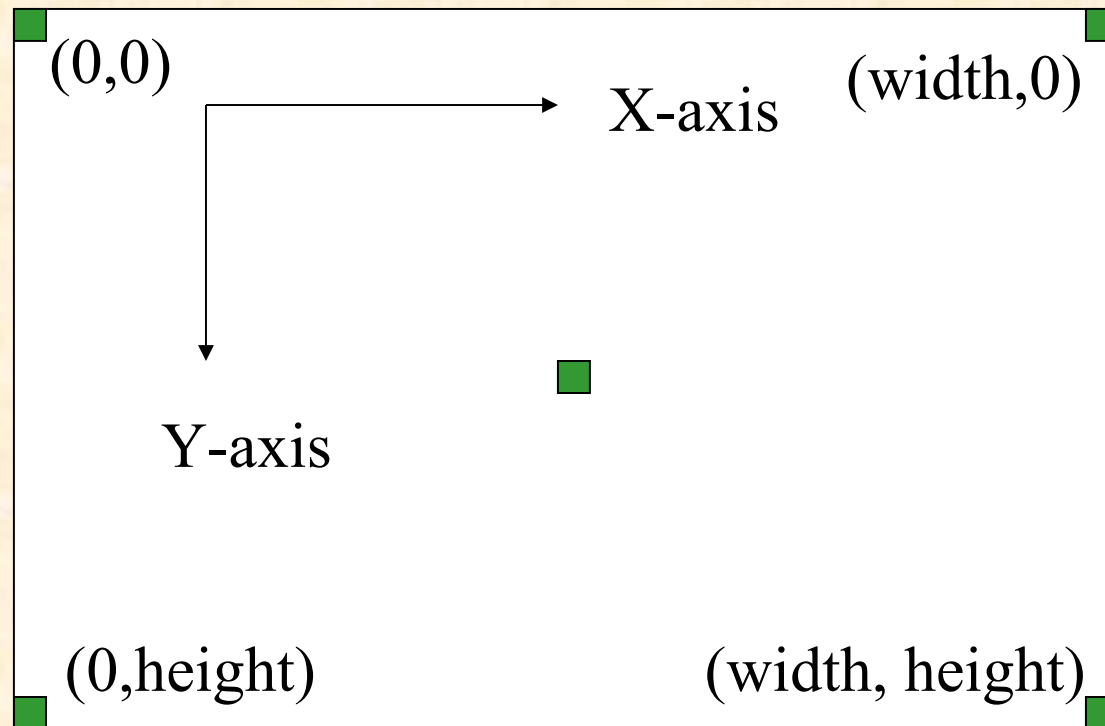
Problem & Solution

- What to do to move the shapes present in previous code when a mouse is dragged
- First time painting is what we already have done
- When a mouse is clicked find the co-ordinates of that place and paint Rectangle at that place by requesting, using `repaint()` call
- Here instead of Hardcoding the position of co-ordinates use some variables e.g. `mx` , `my`
 - `g.drawRect(20,20,20,20)` (small anim here)
 - `g.drawRect(mx,my,20,20)` (small anim here)

- Yesterday we have shown you a tennis game.
- Problem, How to code the paddle movement ?
- We will do it using mouse.
- Try it using Keys

Coordinate System

- Upside-down Cartesian



- $Y_{\text{window}} = \text{height} - Y_{\text{cartesian}}$

Live Output

Example Code: MyPanel.java

```
import javax.swing.*;
import java.awt.*;

public class MyPanel extends JPanel {
    int mX = 20;
    int mY = 20;

    public void paintComponent(Graphics g){
        super.paintComponent(g); // erasing behaviour

        Graphics2D g2 = (Graphics2D)g;

        g2.fillRect(mX, mY, 10, 20);
    }
}
```

Example Code: Test.java

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
  
public class Test{  
  
    JFrame f;  
    MyPanel p;  
  
    public Test(){  
        f = new JFrame();  
        Container c = f.getContentPane();  
        c.setLayout(new BorderLayout());  
  
        p = new MyPanel();  
        c.add(p);  
    }  
}
```

Example Code: Test.java

```
f.setSize(400,400);  
f.setVisible(true);  
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
Handler h = new Handler ();  
p.addMouseMotionListener(h);
```

```
} // end constructor
```

```
public class Handler extends MouseMotionAdapter{
```

```
    public void mouseDragged(MouseEvent me){
```

```
        p.mX = me.getX() ;
```

```
        p.mY = me.getY() ;
```

```
        p.repaint() ;
```

```
    }
```

```
}
```

Example Code: Test.java

```
public static void main(String args[ ]){  
    Test t = new Test( );  
}  
  
} // end class
```

Example Code: MyPanel.java

```
import javax.swing.*;
import java.awt.*;

public class MyPanel extends JPanel {

    int mX = 200;
    int mY = 0;

    public void paintComponent(Graphics g){

        super.paintComponent(g); // erasing behaviour

        Graphics2D g2 = (Graphics2D)g;
        g2.setColor(Color.blue);
        g2.fillOval(mX, mY, 20, 20);

    }
} // end class
```

Example Code: AnimTest.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class AnimTest implements ActionListener {

    JFrame f;
    MyPanel p;
    int x, y;

    public Test(){
        f = new JFrame();
        Container c = f.getContentPane();
        c.setLayout(new BorderLayout());

        x = 5;
        y = 3;
```

Example Code: AnimTest.java

```
p = new MyPanel();  
c.add(p);
```

```
f.setSize(400,400);  
f.setVisible(true);  
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
Timer t = new Timer (5, this);  
t.start();
```

```
} // end constructor
```

```
.....
```

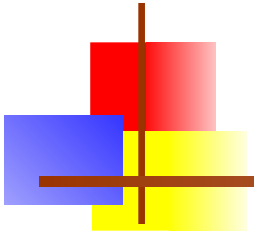
Example Code: AnimTest.java

```
public void actionPerformed(ActionEvent ae){  
    if (f.getWidth()-40 == p.mX)  
        x= -5;  
  
    if (f.getHeight()-40 == p.mY)  
        y= -3;  
  
    if (p.mX == 0 )  
        x = 5;  
  
    if (p.mY == 0 )  
        y=3;  
  
    p.mX+=x;  
    p.mY+=y;  
  
    p.repaint() ;  
}
```


Example Code: AnimTest.java

```
public static void main(String args[ ]){  
    AnimTest at = new AnimTest( );  
}  
  
} // end class
```

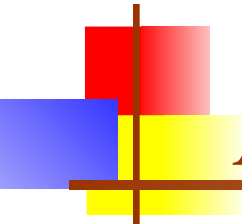
Live Output



Applets

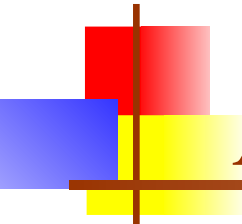
Lec - 20





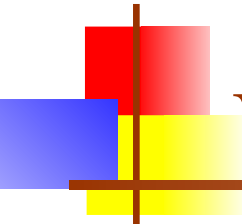
Applets

- An **applet** is a **Panel** that allows interaction with a Java program
- A applet is typically embedded in a Web page and can be run from a browser
- You need special HTML in the Web page to tell the browser about the applet
- For security reasons, applets run in a **sandbox**: they have no access to the client's file system



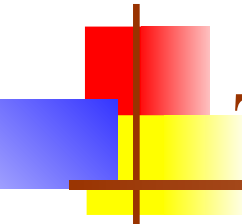
Applet Support

- Most modern browsers support Java 1.4 if they have the appropriate **plugin**
- Sun provides an application **appletviewer** to view applets without using browser.
- In general you should try to write applets that can be run with any browser



What an applet is

- You write an applet by extending the class **Applet** or **JApplet**
- **Applet** is just a class like any other; you can even use it in applications if you want
- When you write an applet, you are only writing *part* of a program
- The browser supplies the **main** method



The genealogy of Applet

java.lang.Object

|

+-----java.awt.Component

|

+-----java.awt.Container

|

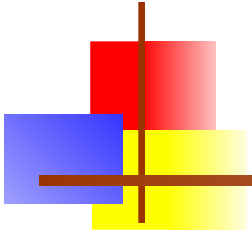
+-----java.awt.Panel

|

+-----java.applet.Applet

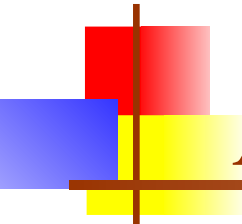
|

+-----javax.swing.JApplet



Code Example



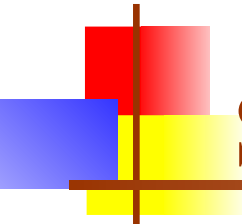


A simple applet

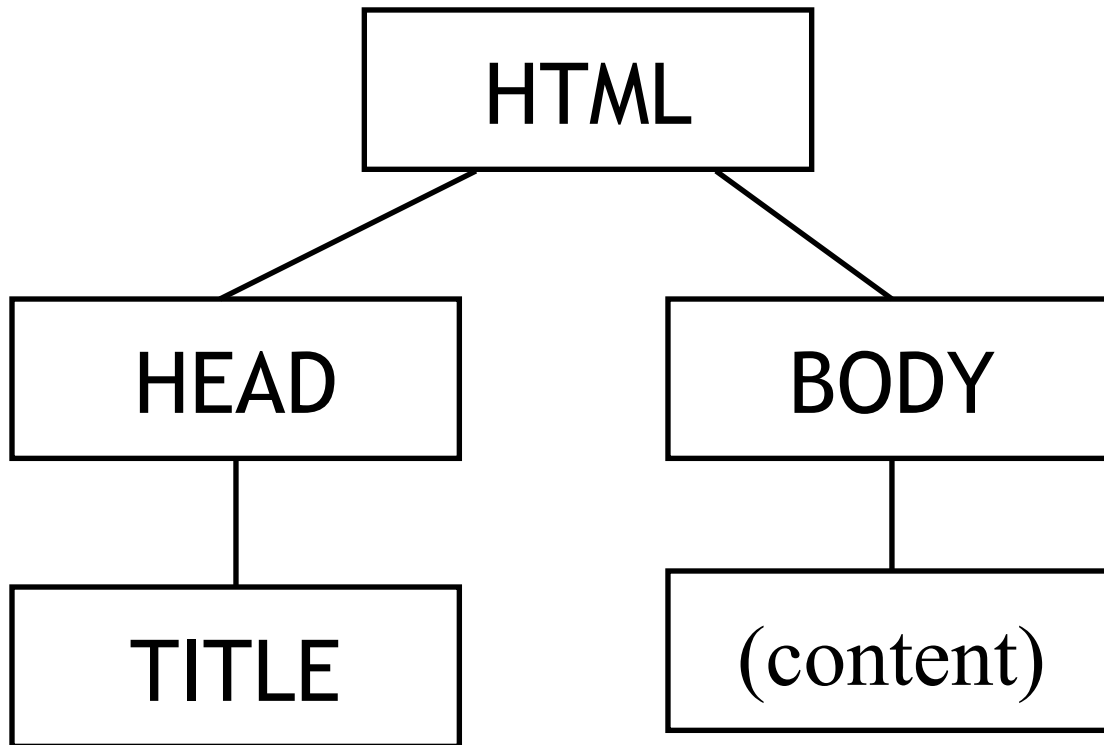
HelloApplet.java

```
import java.awt.*;
import javax.swing.JApplet;

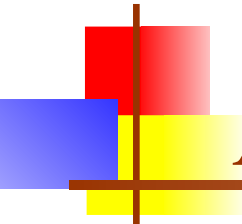
public class HelloApplet extends JApplet {
    public void paint( Graphics g ) {
        g.drawString( "Hello World!", 30, 30 );
    }
}
```



Structure of an HTML page



- Most HTML tags are containers.
- A container is `<tag>` to `</tag>`



A simple applet

Test.html

```
<html>
```

```
<head>
```

```
<title> Simple Applet </title>
```

```
</head>
```

```
<body>
```

```
<applet code="HelloApplet.class"
```

```
width=150 height=100>
```

```
</applet>
```

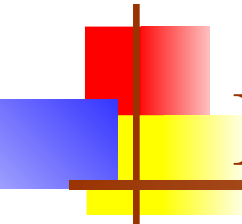
```
</body>
```

```
</html>
```

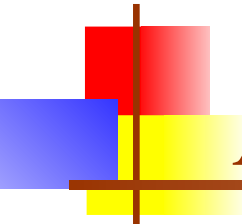
A simple applet



Command prompt output



Browser output of applet



Applet methods

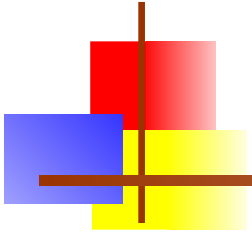
public void init ()

public void start ()

public void stop ()

public void destroy ()

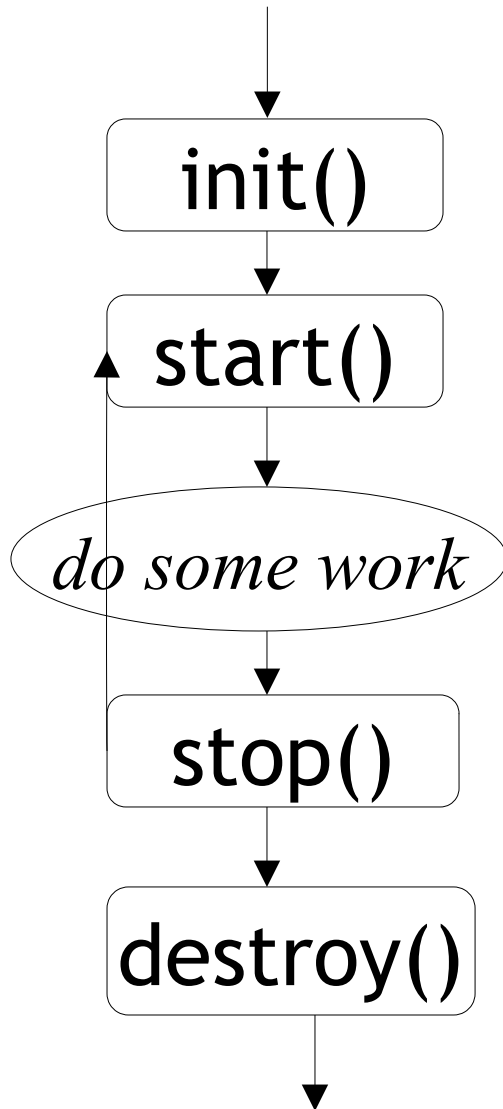
public void paint (Graphics)



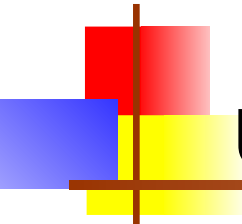
The Applet Life Cycle



Methods are called in this order



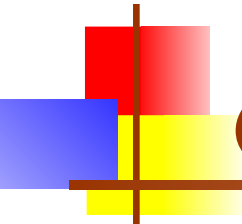
- **init** and **destroy** are only called once each
- **start** is called whenever applet is maximized
- **stop** is called whenever applet is minimized
- **Paint (Graphics g)**
- *do some work* is code called by your listeners
- **paint** is called for the first time when the applet becomes visible and whenever applet needs to be repainted



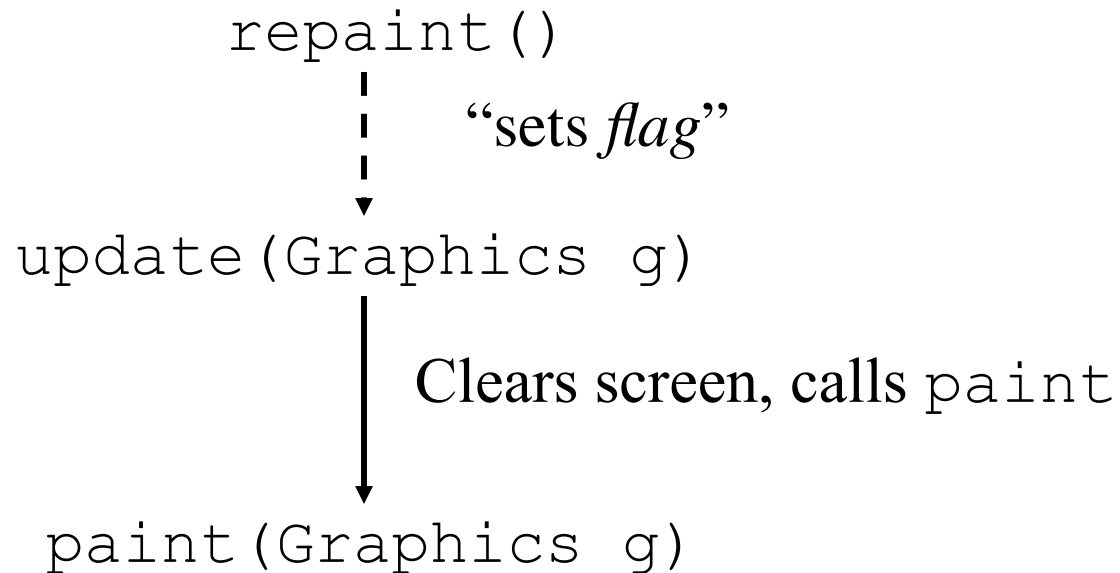
update()

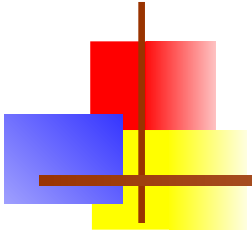
- When you call `repaint( )`, Java schedules a call to `update(Graphics g)`
- Here's what `update` does:

```
public void update(Graphics g) {  
    // Fills applet with background color, then  
    paint(g);  
}
```



Graphics Behavior

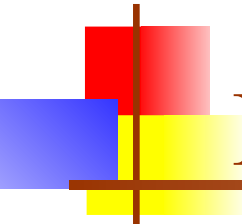




Code Example

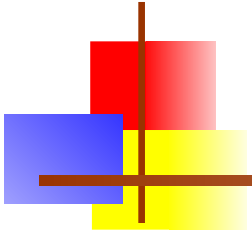
Understanding Applet Life Cycle





Painting at the right time is hard

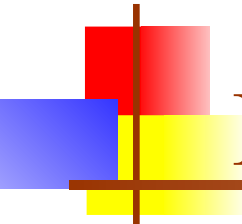
- When you modify common components (**Buttons**, **Labels**, **TextFields**, etc.), *Java* keeps the screen display up to date
- When you paint on a **Graphics** object, *you* have to make your changes appear on the screen
- To help ensure your changes appear on screen, follow these rules:
 - **Rule #1:** Never call **paint(Graphics g)**, call **repaint()**
 - **Rule #2:** Do *all* your painting in **paint**, or in a method that is called from **paint**
 - **Rule #3:** If you paint on any **Graphics** other than the Applet's, call its **update** method from the Applet's **paint** method
 - **Rule #4.** Do your painting in a separate **Thread**
- These rules aren't perfect, but they should help
- If you follow these rules and the screen still doesn't change, I probably won't be able to find the problem, either :-)



Code Example

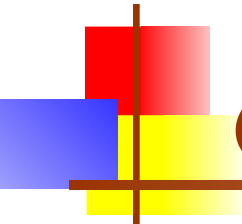
Animated “Java” Word





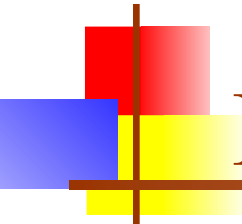
Design Process

- The Program in a single call of paint method
 - Draws string “java” on 40 random locations
 - For every drawing, it selects random font out of 4 different fonts
 - For every drawing, it selects random color out of $256 * 256 * 256$ RGB colors
- Repaint is called after every 1000 ms.
- After 10 calls to repaint, screen is cleared



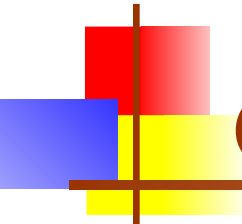
Generating Random Numbers

- Use static method `random` of `Math` class
 - `Math.random()` ;
- Returns positive double value greater than or equal to 0.0 or less than 1.0.
- Multiply the number with appropriate scaling factor to increase the range and type cast it, if needed.
 - `int i = (int)( Math.random() * 5 );` will generate random numbers between 0 & 4.



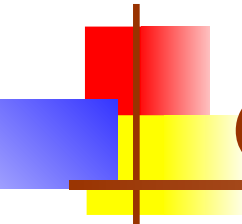
Draw String “java”

- `public void drawJava(Graphics2D g2) {`
- `int x = (int) (Math.random() * 1000);`
- `int y = (int) (Math.random() * 700);`
- `g2.drawString("java", x, y);`
- `}`



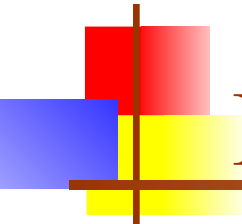
Choosing a random color

```
public Color chooseColor() {  
    int r = (int) (Math.random() * 255);  
    int gr = (int) (Math.random() * 255);  
    int b = (int) (Math.random() * 255);  
  
    Color c = new Color(r, gr, b);  
    return c;  
  
}
```



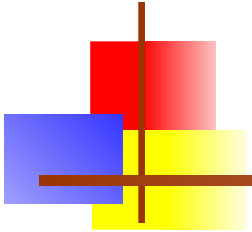
Choosing a random font

```
public Font chooseFont() {  
    int fontChoice = (int) (Math.random() * 4) + 1;  
  
    Font f = null;  
  
    switch (fontChoice) {  
  
        case 1:  
            f = new Font("Serif", Font.BOLD + Font.ITALIC, 20);  
            break;  
  
        case 2:  
            f = new Font("SansSerif", Font.PLAIN, 17);  
            break;  
  
        case 3:  
            f = new Font("Monospaced", Font.ITALIC, 23);  
            break;  
  
        case 4:  
            f = new Font("Dialog", Font.ITALIC, 30);  
            break;  
  
    }  
  
    return f;  
}
```



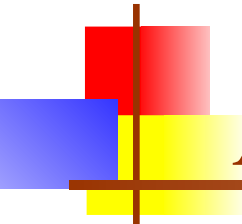
Paint method

```
■ // write any painting code here
■ public void paint(Graphics g) {
■
■     clearCounter++;
■
■     for (int i = 1; i <= 40; i++) {
■
■         Color c = chooseColor();
■         g2.setColor(c);
■
■         Font f = chooseFont();
■         g2.setFont(f);
■
■         drawJava(g2);
■     }
■
■     Graphics2D g2 = (Graphics2D) g;
■
■     if (clearCounter == 10) {
■         g2.clearRect(0, 0, 1000, 700);
■         clearCounter = 0;
■     }
■ }
```



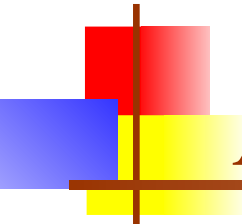
Security Restrictions





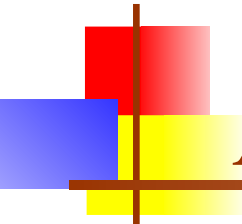
Applets Cannot...

- Read from the local (client) disk
- Write to the local (client) disk
- Open network connections other than to the server from which the applet was loaded
- Link to client-side C code or call programs installed on the browser machine
- Discover private information about the user



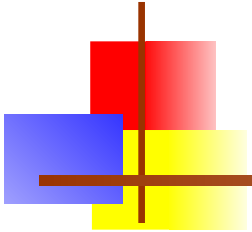
Applets Cannot...

- Read from the local (client) disk
 - Applets cannot read arbitrary files
 - They can, however, instruct the browser to display pages that are generally accessible on the Web, which might include some local files
- Write to the local (client) disk
 - The browser may choose to cache certain files, including some loaded by applets, but this choice is not under direct control of the applet
- Open network connections other than to the server from which the applet was loaded
 - This restriction prevents applets from browsing behind network firewalls



Applets Cannot...

- Link to client-side C code or call programs installed on the browser machine
 - Ordinary Java applications can invoke locally installed programs (with the `exec` method of the `Runtime` class) as well as link to local C/C++ modules (“native” methods)
 - These actions are prohibited in applets because there is no way to determine whether the operations these local programs perform are safe
- Discover private information about the user
 - Applets should not be able to discover the username of the person running them or specific system information such as current users, directory names or listings, system software, and so forth
 - However, applets *can* determine the name of the host they are on; this information is already reported to the HTTP server that delivered the applet



The End





Web

Design & Development

Lec - 21

Umair Javed



Web

Design & Development

Socket Programming

Request – Response Model

- **animation**

Socket

- A socket is one endpoint of a two-way communication link between two programs running on the network.
- A socket is a bi-directional communication channel between hosts (a computer on a network can be termed as a host).
- A socket is bound to a port number

Socket's Dynamics

- A Socket is an **abstraction of the network** similar to the way a file is an abstraction of your hard drive.
- You store and retrieve data through files from hard drive without knowing the actual dynamics of the hard drive.
- Similarly you send and receive data to and from network through socket without actually going into underlying mechanics.

Sending/Receiving Messages using Socket

- You read from or write data to a file, using streams.
- Similarly to read from or write data to a socket, you use streams.

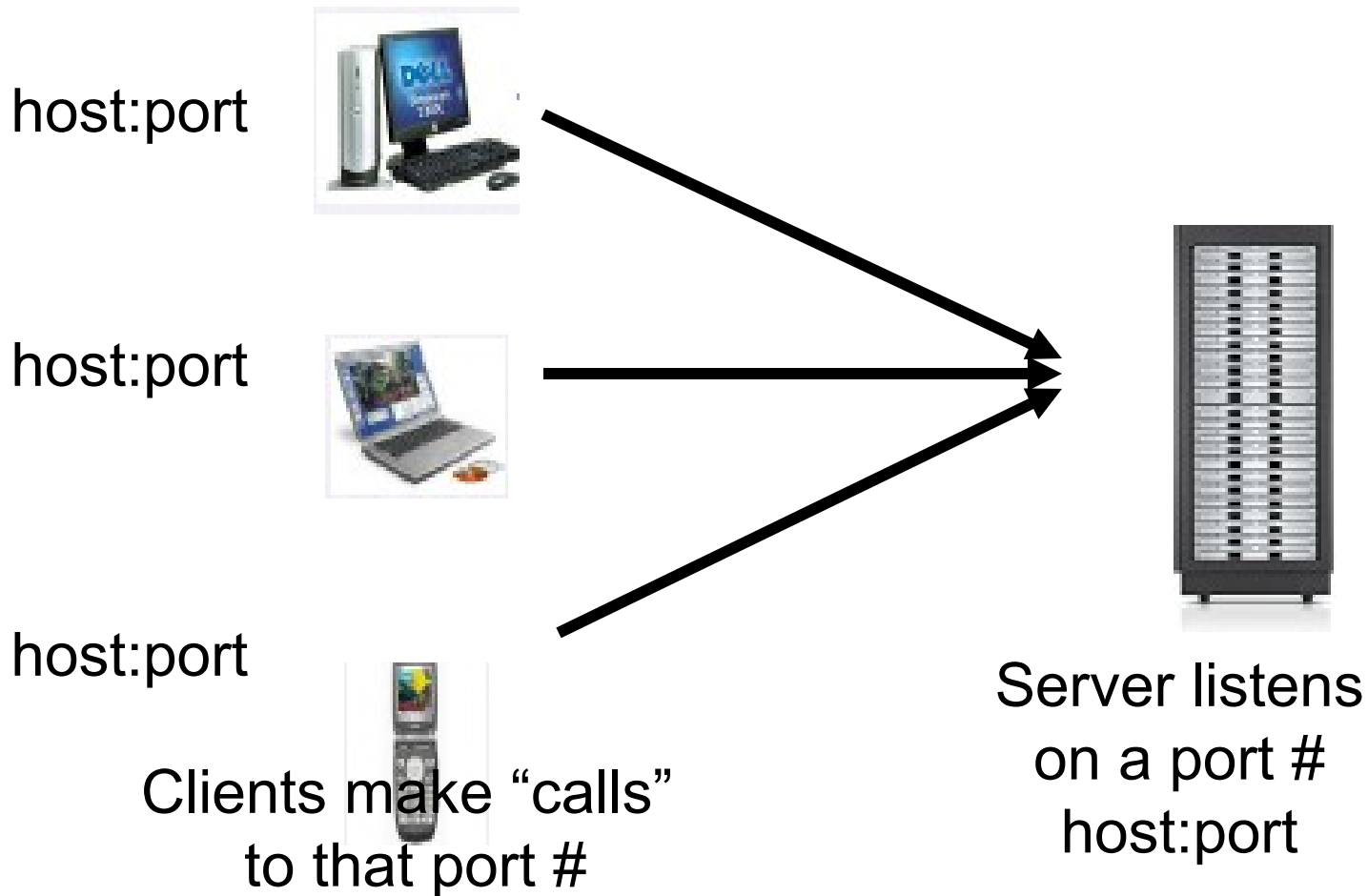
What is a Port?

- **Transport address to which processes can listen for connection requests**
- **Local to host: 64K TCP & 64K UDP ports**
- **Well-known ports**
 - Below 1024
 - Standard services
 - Only supervisor privileged enough to access

Examples

- FTP: 21
- HTTP: 80
- TELNET: 23

Request – Response Model



How Client & Server communicate

- **animation**

Steps – To Make a Simple Client

- 1. Import required Package**
- 2. Connect / Open a Socket with Server**
- 3. Get I/O Streams of Socket**
- 4. Send / Receive Message**
- 5. Close Socket**

Steps – To Make a Simple Client

1. Import required Package

```
import java.net.*;  
import java.io.*;
```

2. Connect / Open a Socket with Server

```
Socket s = new Socket("sAdd", sPort);
```

Steps – To Make a Simple Client

Show code here

3. Get I/O Streams of Socket

```
FileReader fr = new  
BufferedReader br :
```

```
InputStream is = s.getInputStream();  
InputStreamReader isr= new InputStreamReader(is);  
BufferedReader br = new BufferedReader(isr);
```

```
OutputStream os = s.getOutputStream();  
PrintWriter pw = new PrintWriter(os, true);
```

animation

Steps – To Make a Simple Client

4. Send / Receive Message

```
pw.println("hello world");  
String recMsg = br.readLine();
```

5. Close Socket

```
s.close();
```

Steps – To Make a Simple Server

- 1. Import required Package**
- 2. Create a Server Socket**
- 3. Wait for Incoming Connections**
- 4. Get I/O Streams of communication Socket**
- 5. Send / Receive Message**
- 6. Close Socket**

Steps – To Make a Simple Server

1. Import required Package

```
import java.net.*;  
import java.io.*;
```

2. Create a Server Socket

```
ServerSocket ss = new  
ServerSocket(portNo);
```


Steps – To Make a Simple Server

3. Wait for Incoming Connection

- When connection established, communication socket is returned

```
Socket s = ss.accept();
```

Steps – To Make a Simple Server

4. Get I/O Streams of Socket

```
InputStream is = s.getInputStream();  
InputStreamReader isr= new InputStreamReader(is);  
BufferedReader br = new BufferedReader(isr);
```

```
OutputStream os = s.getOutputStream();  
PrintWriter pw = new PrintWriter(os, true);
```

Steps – To Make a Simple Server

5. Send / Receive Message

```
pw.println("hello world");  
String recMsg = br.readLine();
```

6. Close Socket

```
s.close();
```

animation



Web

Design & Development

Example Code

Echo Server

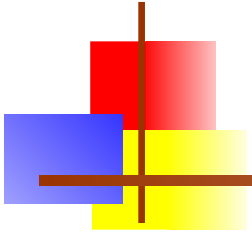


Web

Design & Development

Example Code

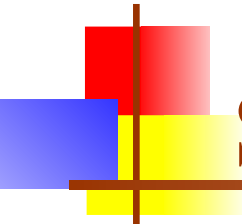
Echo Client



Object Serialization Explanation + Example of file + network

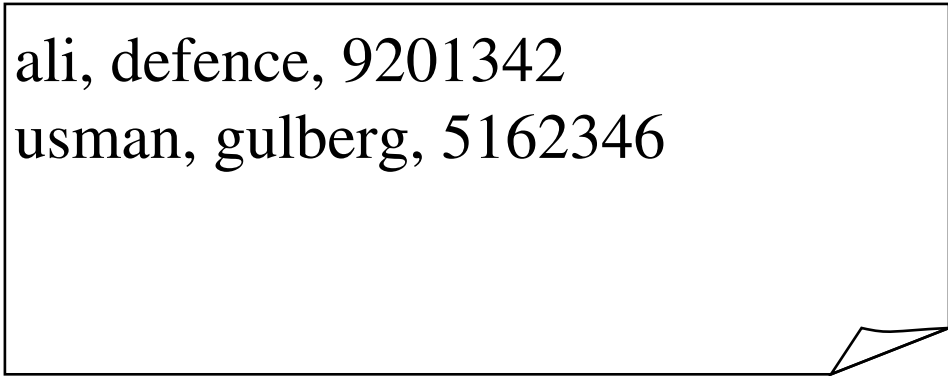
Lec 22





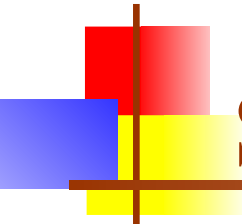
Serialization

- You want to send an object to a stream
- Motivation
 - A lot of code involves boring conversion from a file to memory
 - AddressBook program reads data from file and then parses it
 - This is a common problem!



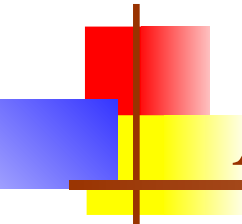
```
ali, defence, 9201342  
usman, gulberg, 5162346
```

address.txt



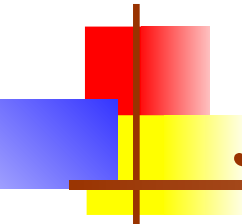
Serialization

- Java's answer:
 - Serialization
 - Object know how to read/write themselves to streams
- Problem - Objects have state in memory
- Serialization is also called
 - Flattening, Streaming, Dehydrate (rehydrate = read), Archiving



Animation

- Object diagram with stack & heap
- After serialized, passed through a stream (pipe) and reconstructed on the other side



Java: Automatic Serialization

■ **Serializable** Interface

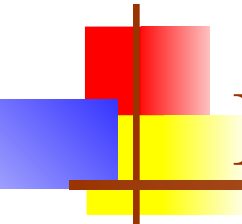
- By implementing this interface a class declares that it is willing to be read/written by automatic serialization machinery
- Found in **java.io** package
- **Tagging** interface – has no methods and serves only to identify the semantics of being serializable

■ Automatic Writing

- System knows how to recursively write out the state of an object to stream
- Recursively follows references and writes out those objects too!

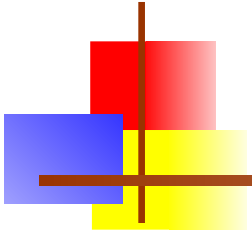
■ Automatic Reading

- System knows how to read the data from Stream and re-create object in memory
- **Downcasting is required**



How it works?

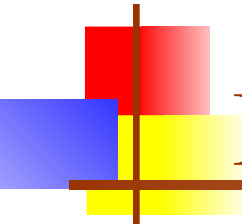
- To write out an object of `PersonInfo`
 - `PersonInfo p = new PersonInfo();`
 - `ObjectOutputStream out;`
 - `out.writeObject(p)`
- To read that object back in
 - `ObjectInputStream in;`
 - `PersonInfo obj = (PersonInfo) in.readObject();`
- Must be of the same type
 - class and version issue



Example Code: Serialization

Reading/Writing PersonInfo objects





Example Code: Serialization

```
import javax.swing.*;
```

```
import java.io.*;
```

```
public class PersonInfo implements Serializable {
```

```
    String name;
```

```
    String address;
```

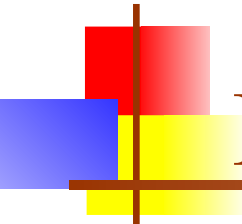
```
    String phoneNum;
```

```
    public void printPersonInfo() {
```

```
        JOptionPane.showMessageDialog( "name: " + name + "address:" + address +  
                                         "phoneNum: " + phoneNum);
```

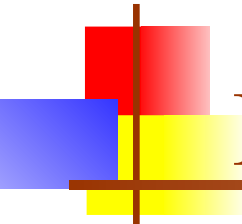
```
    }
```

```
}
```



Example Code: Serialization (cont.)

```
import java.io*;  
  
public class WriteEx{  
    public static void main(String args[ ]){  
  
        PersonInfo pWrite = new PersonInfo("ali", "defence", "9201211");  
  
        try {  
  
            FileOutputStream fos  = new FileOutputStream("ali.dat");  
            ObjectOutputStream out = new ObjectOutputStream(fos);  
  
            //serialization  
            out.writeObject(pWrite);  
  
            out.close();  
            fos.close();  
  
        } catch (Exception ex){  
            System.out.println(ex)  
        }  
    }  
}
```



Example Code: Serialization (cont.)

```
import java.io*;

public class ReadEx {
    public static void main(String args[ ]){

        try {

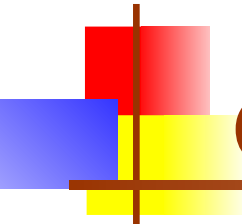
            FileInputStream fis = new FileInputStream("ali.dat");
            ObjectInputStream in = new ObjectInputStream(fis);

            //de - serialization
            PersonInfo pRead = (PersonInfo) in.readObject( );

            pRead.printPersonInfo();

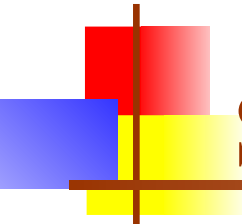
            in.close();
            fis.close();

        } catch (Exception ex){
            System.out.println(ex)
        }
    }
}
```

Object Serialization and Network

- You can read/write objects to network using sockets
- The class version should be same on both sides (client & Server) of the network

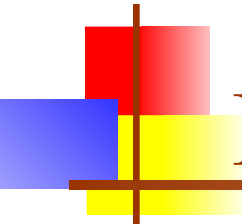


Sending Objects over Network

.....

- `PersonInfo p = new PersonInfo (“ali”, “defence”, “9201211”);`
- `Socket s = new Socket(“localhost”, 4444);`
- `OutputStream os = s.getOutputStream();`
- `ObjectOutputStream oos = new ObjectOutputStream(os);`
- `oos.write(p);`

.....



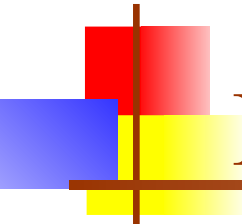
Reading Objects from Network

.....

- `Socket s = ss.accept();`
- `InputStream in = s.getInputStream();`
- `ObjectInputStream ois = new ObjectInputStream(is);`

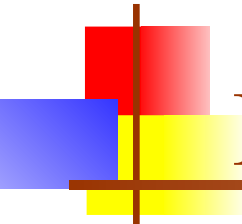
- `PersonInfo p = (PersonInfo) ois.read( );`

.....



Preventing Serailization

- **transient** keyword is used to mark a field that should not be serialized
- Often there is no need to serialize sockets, streams & DB connections etc (they do not represent the state of object, rather connections to external resources)
- So, we can mark them as
 - public **transient** Socket s;
 - public **transient** OutputStream os;
 - public **transient** Connection con;
- Transient fields are returned as **null** on reading



Example Code: transient

```
import javax.swing.*;
```

```
import java.io.*;
```

```
public class PersonInfo implements Serializable {
```

```
    String name;
```

```
    String address;
```

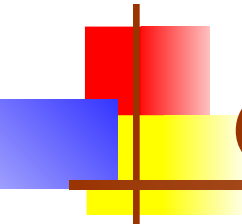
```
    transient String phoneNum;
```

```
    public void printPersonInfo() {
```

```
        JOptionPane.showMessageDialog( "name: " + name + "address:" + address +  
                                         "phoneNum: " + phoneNum);
```

```
    }
```

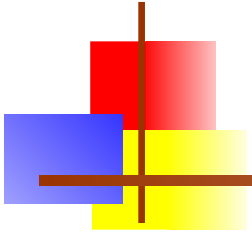
```
}
```



Circularity: not an issue

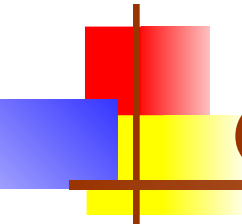
- Serialization machinery will take circular references into account and do the right thing!

Animation



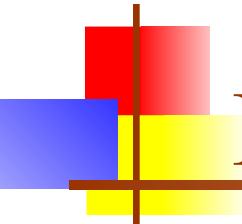
Multithreading





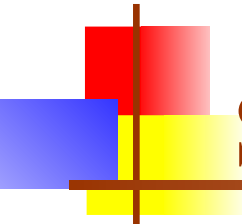
Concurrency Hardware VS Software

- Hardware Concurrency Headings
- Software Concurrency
 - Processes VS Threads
- Three loop code in a single class



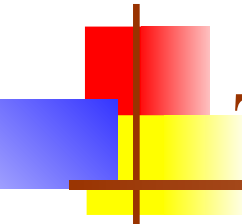
Hardware Concurrency trends

- Multiple CPU's
- Multiple Cores on a Single Chip
- Simultaneous Multi-Threading (SMT)



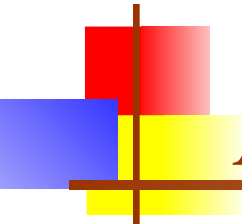
Software concurrency

- Processes
 - Unix-style concurrency
 - The ability to run multiple applications at once
 - Example: word, powerpoint , game
 - Separate address space
 - Cooperate using read/write streams (pipes)
 - Synchronization is easy
 - Since there is no shared address space



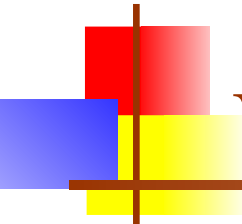
Threads

- The ability to do multiple things at once within the same application
 - Finer granularity of concurrency
- Lightweight
 - Easy to create and destroy
- Shared address-space
 - Can share memory variables directly
 - May require more complex synchronization logic because of shared address space



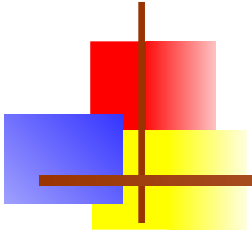
Advantages of threads...

- Use multiple processors
 - Code is partitioned in order to be able to use n processors at once
 - This is not easy to do! But Moore's Law may force us in this direction
- Hide network/disk latency
 - While one thread is waiting for something, run the others
 - Dramatic improvements even with a single CPU
 - Need to efficiently block the connections that are waiting, while doing useful work with the data that has arrived
 - Writing good network codes relies on concurrency!
- Keeping the GUI responsive
 - Separate worker threads from GUI thread



Why Concurrency is a Hard Problem

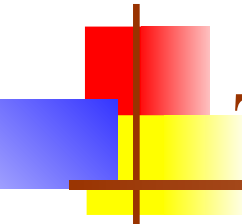
- No language construct to alleviate the problem
 - Memory management can be solved by a garbage collector, no analog for concurrency
- Counter-intuitive
 - Concurrency bugs are hard to spot in the code
 - Difficult to get into the concurrency mindset
- No fixed programmer recipe either
- Client may need to know the internal model to use it correctly
 - Hard to pass the Clueless-Client test
- Concurrency bugs are random
 - Show up rarely, often not deterministic/reproducible easily
 - Rule of thumb: if something bizarre happens try and note the current state as well as possible



Multithreading

Lec 23



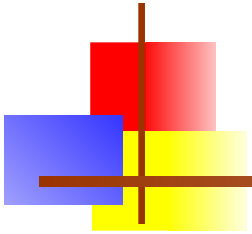


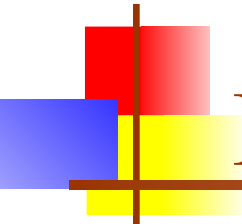
Threads

- The ability to do multiple things at once within the same application
 - Finer granularity of concurrency
- Lightweight
 - Easy to create and destroy
- Shared address space
 - Can share memory variables directly
 - May require more complex synchronization logic because of shared address space

Three Loops

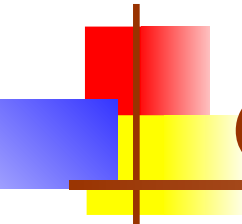
Sequential Execution





Example Code: ThreeLoopTest

```
public class ThreeLoopTest {  
    public static void main (String args[ ]){  
  
        // first loop  
        for (int i=1; i<= 5; i++)  
            System.out.println("first " +i);  
  
        // second loop  
        for (int j=1; j<= 5; j++)  
            System.out.println("second " + j);  
  
        // third loop  
        for (int k=1; k<= 5; k++)  
            System.out.println("third " + k);  
  
    }  
}
```



Compile & Execute: ThreeLoopTest



```
C:\WINDOWS\system32\cmd.exe

D:\examples\multithreading>javac ThreeLoopTest.java

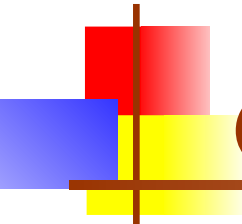
D:\examples\multithreading>java ThreeLoopTest
first job 1
first job 2
first job 3
first job 4
first job 5
second job 1
second job 2
second job 3
second job 4
second job 5
third job 1
third job 2
third job 3
third job 4
third job 5
```

Multi-Threaded Output



```
C:\WINDOWS\system32\cmd.exe

D:\examples\multithreading>java ThreadTest
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
second job = 1
third job = 1
second job = 2
third job = 2
second job = 3
third job = 3
second job = 4
third job = 4
second job = 5
third job = 5
second job = 6
third job = 6
first job = 9
second job = 7
first job = 10
second job = 8
second job = 9
second job = 10
third job = 7
third job = 8
third job = 9
third job = 10
```



Creating Threads in Java

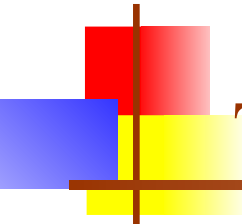
- Two approaches

- Using Interface

- Implement the runnable interface in a class
 - Provide an implementation for the run() method
 - Instantiate Thread object by passing runnable object in constructor
 - Start thread

- Using Inheritance

- Subclass java.lang.Thread
 - Override the run() method
 - Instantiate Subclass Thread Object
 - Start thread



Thread Creation Steps : using Interface

- **Step 1 - Implement the Runnable Interface**

```
class Worker implements Runnable
```

- **Step 2 - Provide an Implementation of run method**

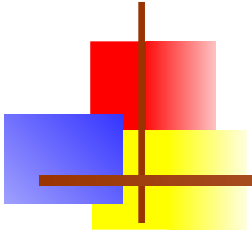
```
public void run ( ){  
    // write thread behavior  
    // code that will execute by thread  
}
```

- **Step 3 - Instantiate Thread object by passing runnable object in constructor**

```
Worker w = new Worker("first");  
Thread t = new Thread (w);
```

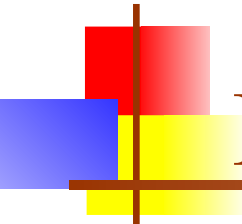
- **Step 4 – Start thread**

```
t.start()
```



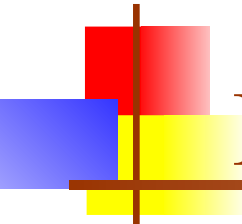
Three Loops Multi-Threaded Execution





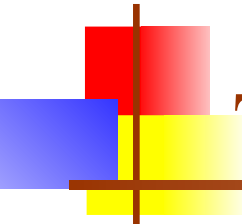
Example Code: using Interface

```
public class Worker implements Runnable {  
    private String job ;  
  
    public Worker (String j ){  
        job = j;  
    }  
  
    public void run ( ) {  
        for(int i=1; i<= 10; i++)  
            System.out.println(job + " = " + i);  
    }  
}
```



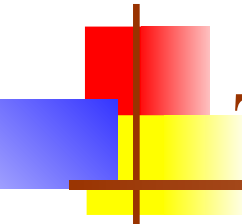
Example Code: using Interface

```
public class ThreadTest{  
    public static void main (String args[ ]){  
  
        Worker first    = new Worker ("first job");  
        Worker second = new Worker ("second job");  
        Worker third   = new Worker ("third job")  
  
        Thread t1 = new Thread (first );  
        Thread t2 = new Thread (second);  
        Thread t3 = new Thread (third);  
  
        t1.start();  
        t2.start();  
        t3.start();  
    }  
}
```

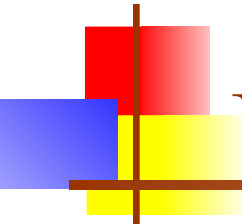
Thread Priorities

- Every Thread has a priority
- Threads with higher priority are executed in preference to threads with lower priority
- A thread's default priority is same as that of the creating thread



Thread Priorities

- You can change thread priority by using any of the 3 predefined constants
 - `Thread.MAX_PRIORITY` (typically 10)
 - `Thread.NORM_PRIORITY` (typically 5)
 - `Thread.MIN_PRIORITY` (typically 1)
- OR any integer value from 1 to 10 can be used as thread priority.

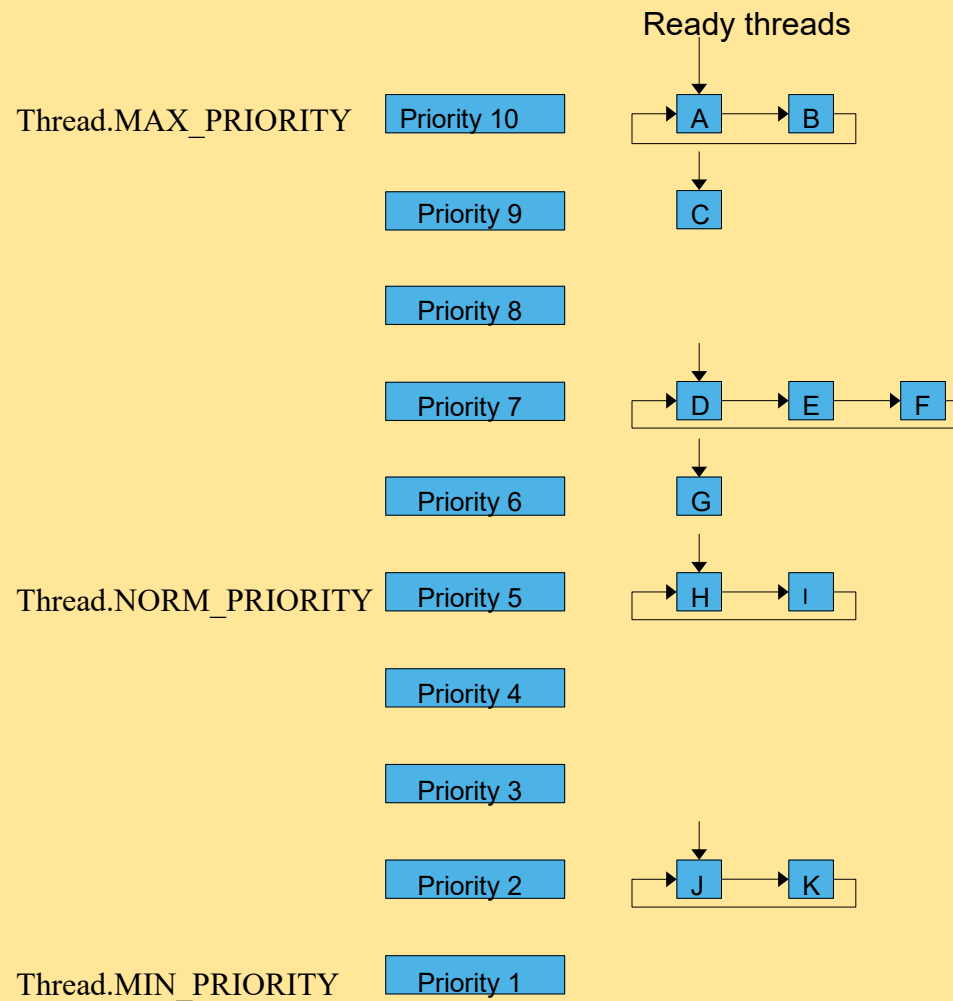


Useful Thread Methods

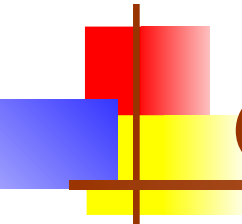
- `setPriority(int priority)`
 - Changes the priority of this thread
 - Throws **IllegalArgumentException** if the priority is not in the range `MIN_PRIORITY` to `MAX_PRIORITY`
 - For Example,

```
Thread t = new Thread(RunnableObject);  
t.SetPriority(Thread.MAX_PRIORITY);  
t.setPriority(7);
```

Thread priority scheduling example

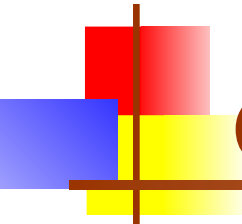


Credit: Advanced Java, Dietel & Dietel

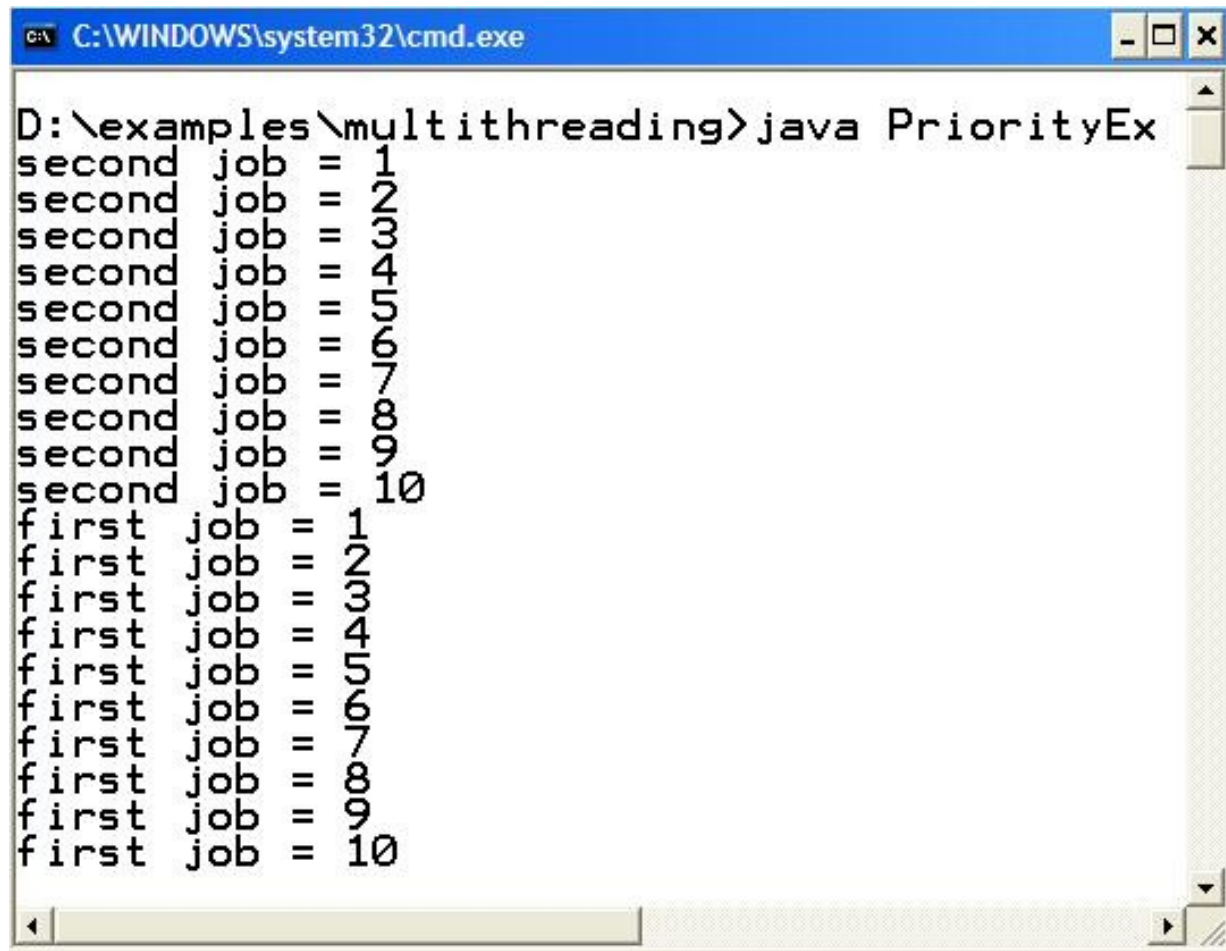


Code Example: PriorityEx.java

```
public class PriorityEx {  
    public static void main (String args[ ]){  
        Worker first    = new Worker ("first job");  
        Worker second = new Worker ("second job");  
  
        Thread t1 = new Thread (first );  
        Thread t2 = new Thread (second);  
  
        t1.setPriority(Thread.MIN_PRIORITY);  
        t2.setPriority(Thread. MAX_PRIORITY);  
  
        t1.start();  
        t2.start();  
    }  
}
```

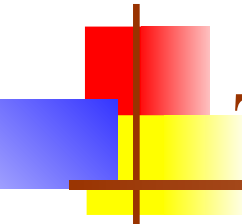


Output: PriorityEx.java



```
C:\WINDOWS\system32\cmd.exe

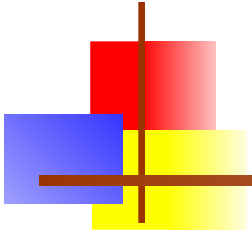
D:\examples\multithreading>java PriorityEx
second job = 1
second job = 2
second job = 3
second job = 4
second job = 5
second job = 6
second job = 7
second job = 8
second job = 9
second job = 10
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
first job = 9
first job = 10
```



Thread Priorities

■ Problems

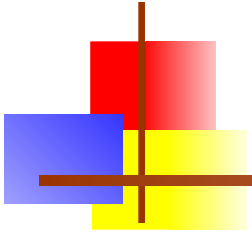
- A Java thread **priority may map differently** to the thread priorities of the underlying OS
 - Solaris has $2^{32}-1$ priority levels;
 - Windows NT has only 7 user priority levels
- **Starvation can occur** for lower-priority threads if the higher-priority threads never terminate, sleep, or wait for I/O



Multithreading 2

Lec 24

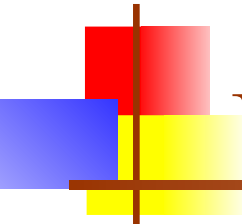




Reading Two Files Simultaneously

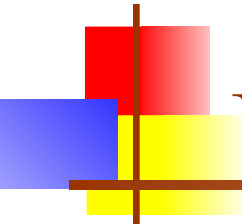
Example Code





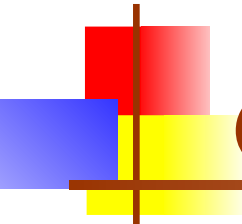
Useful Thread Methods

- sleep (int time)
 - Causes the currently executing thread to wait for the time (milliseconds) specified
 - Waiting is efficient (non-busy)
 - Threads come out of the sleep when the specified time interval **expires** or when **interrupted** by some other thread
 - Thread coming out of sleep may go to the **running** or **ready** state depending upon the availability of the processor.



Useful Thread Methods (cont.)

- sleep (int time)
 - High priority threads should execute sleep method after some time to give low priority threads a chance to run otherwise **starvation** may occur
 - Sleep can be used for delay purpose



Code Example: Modify Worker.java

```
public class Worker implements Runnable {
```

```
.....
```

```
public void run ( ) {
```

```
    for(int i=1; i<= 10; i++) {
```

```
        try {
```

```
            Thread.sleep(100);
```

```
        } catch (Exception ex) {
```

```
            System.out.println(ex);
```

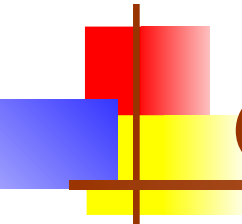
```
        }
```

```
        System.out.println(job + " = " + i);
```

```
    } // end for
```

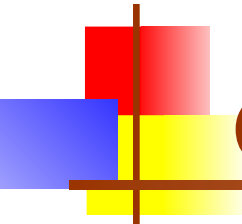
```
} // end run
```

```
}// end Worker
```

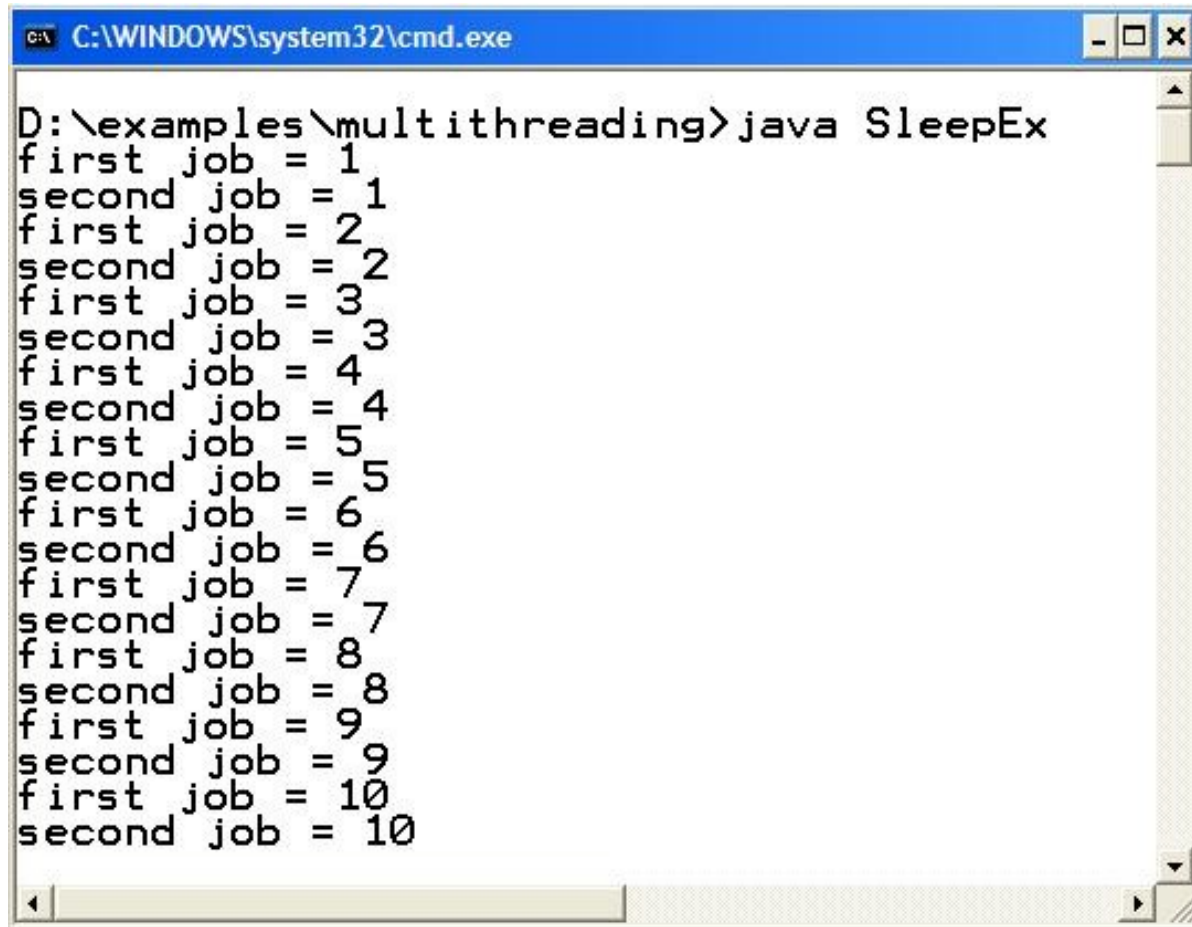


Code Example: SleepEx.java

```
public class SleepEx {  
    public static void main (String args[ ]){  
  
        Worker first    = new Worker ("first job");  
        Worker second = new Worker ("second job");  
  
        Thread t1 = new Thread (first );  
        Thread t2 = new Thread (second);  
  
        t1.start();  
        t2.start();  
    }  
}
```

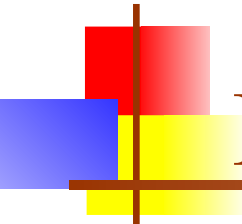


Output: SleepEx.java



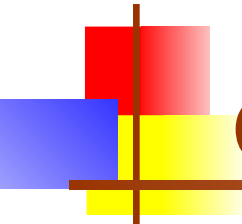
```
C:\WINDOWS\system32\cmd.exe

D:\examples\multithreading>java SleepEx
first job = 1
second job = 1
first job = 2
second job = 2
first job = 3
second job = 3
first job = 4
second job = 4
first job = 5
second job = 5
first job = 6
second job = 6
first job = 7
second job = 7
first job = 8
second job = 8
first job = 9
second job = 9
first job = 10
second job = 10
```

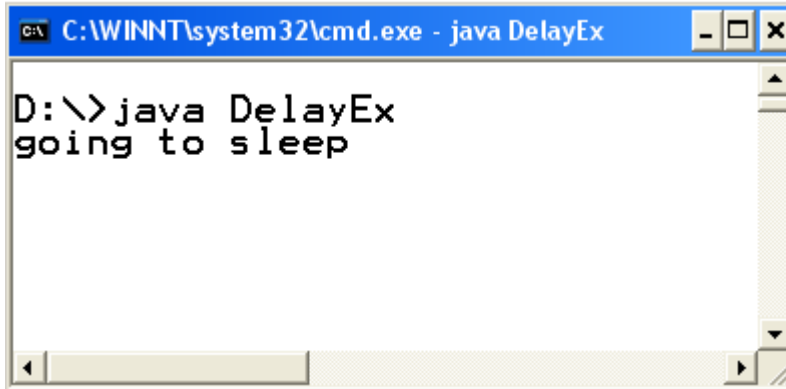


Example code: DelayEx.java

```
public class DelayEx {  
    public static void main (String args[ ]){  
        System.out.println("going to sleep");  
        try {  
            Thread.sleep(1000);  
        } catch (Exception ex){  
            System.out.println(ex);  
        }  
        System.out.println("waking up");  
    }  
}
```

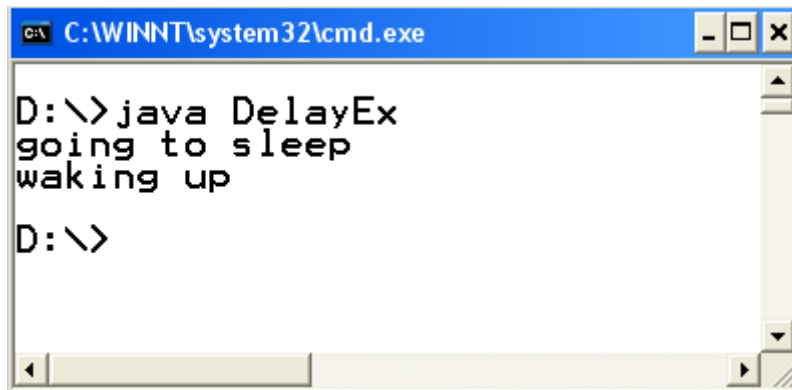


Output: DelayEx.java



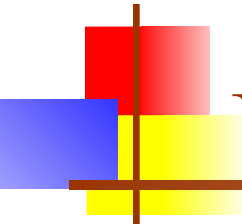
```
C:\WINNT\system32\cmd.exe - java DelayEx

D:\>java DelayEx
going to sleep
```



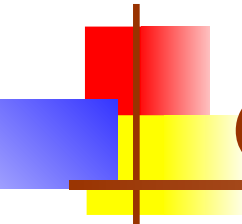
```
C:\WINNT\system32\cmd.exe

D:\>java DelayEx
going to sleep
waking up
D:\>
```

Useful Thread Methods (cont.)

- `yield ( )`
 - Allows any other threads of the **same priority** to execute (moves itself to the end of the priority queue)
 - If all waiting threads have a lower priority, then the yielding thread resumes execution on the CPU
 - Generally used in **cooperative scheduling** schemes



Code Example: Modify Worker.java

```
public class Worker implements Runnable {
```

```
.....
```

```
public void run ( ) {
```

```
    for(int i=1; i<= 10; i++) {
```

```
        Thread.yield( );
```

```
        System.out.println(job + " = " + i);
```

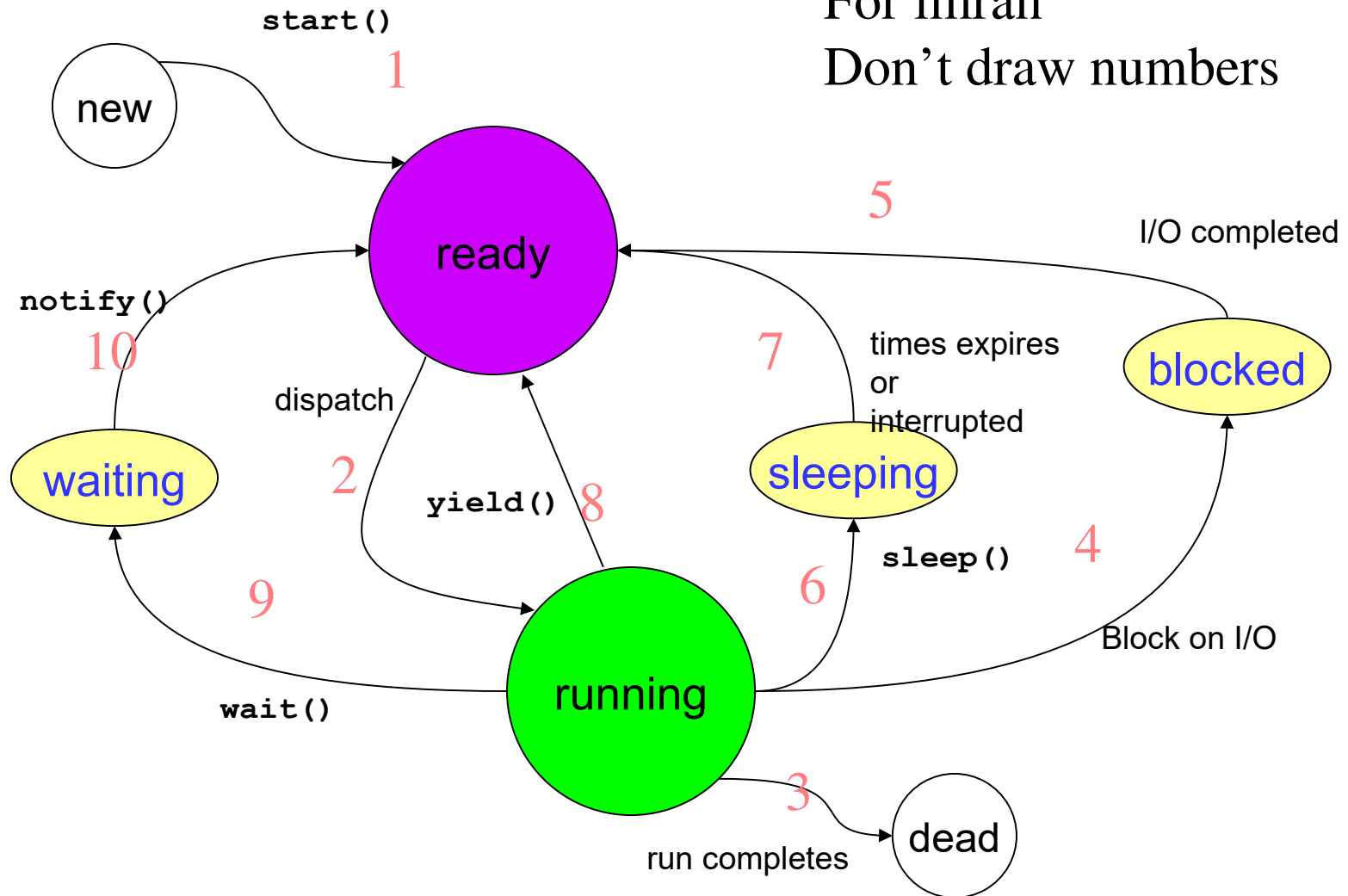
```
    } // end for
```

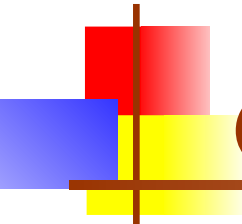
```
} // end run
```

```
}// end Worker
```

Thread Lifecycle

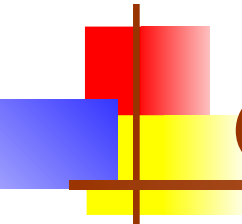
For imran
Don't draw numbers





Code Example: JoinEx.java

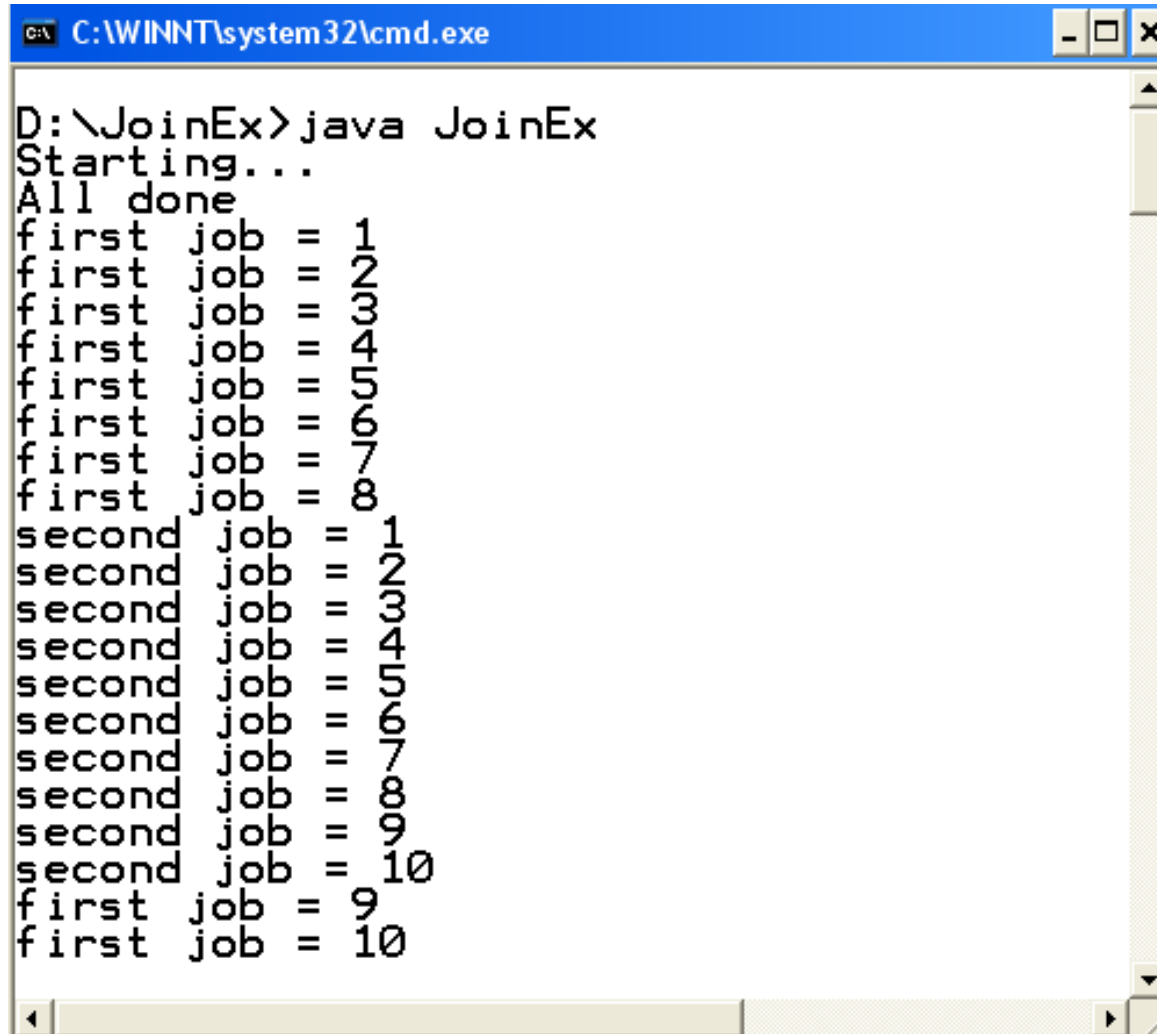
```
public class JoinEx {  
    public static void main (String args[ ]){  
  
        Worker first    = new Worker ("first job");  
        Worker second = new Worker ("second job");  
  
        Thread t1 = new Thread (first );  
        Thread t2 = new Thread (second);  
  
        System.out.println("Starting...");  
  
        t1.start();  
        t2.start();  
  
        .....  
    }  
}
```



Code Example: JoinEx.java (cont.)

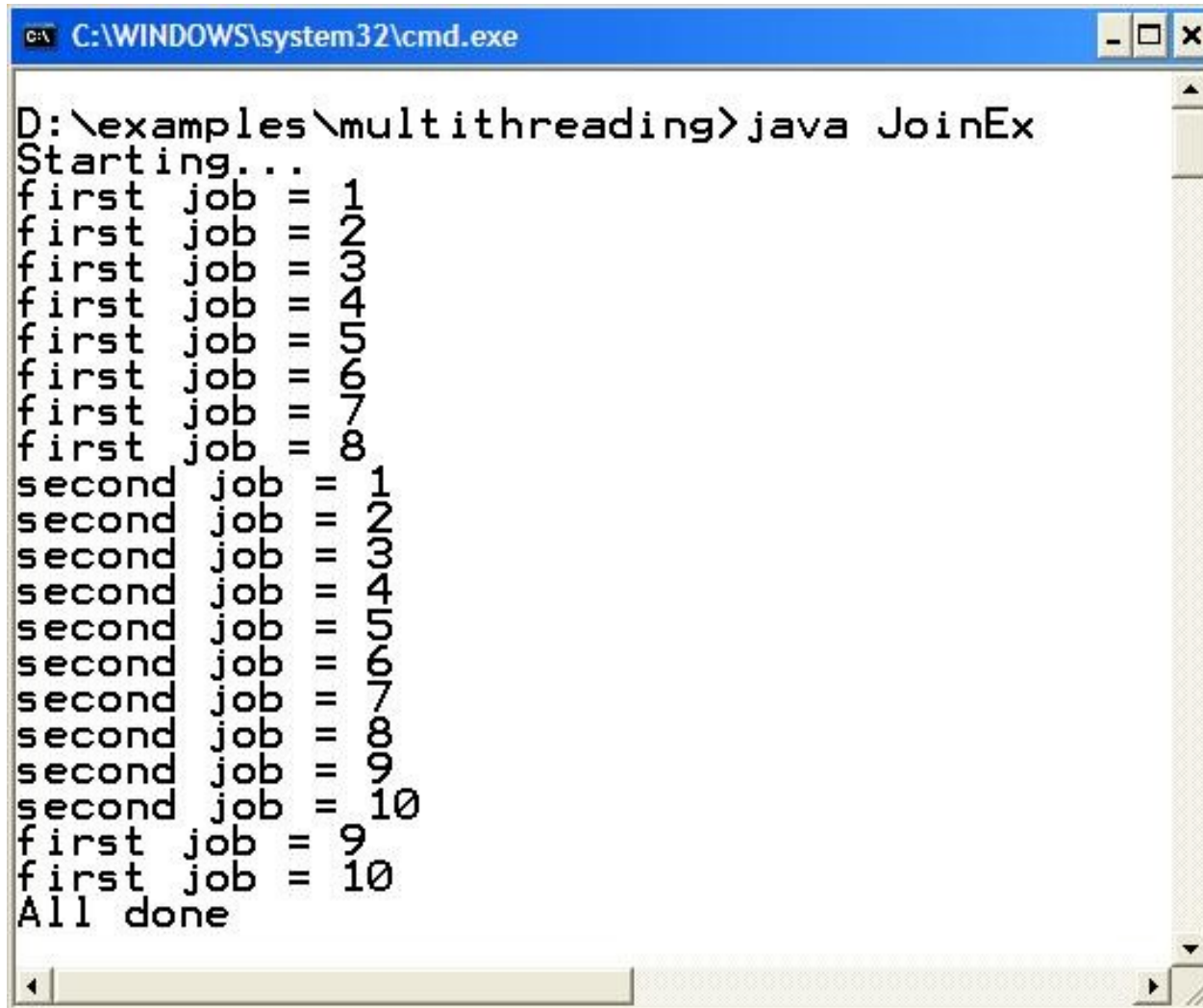
```
try {  
    t1.join();  
    t2.join();  
}  
catch (Exception ex) {  
    System.out.println(ex);  
}  
  
System.out.println("All done ");  
  
}  
}
```

Output: without join statements



```
C:\WINNT\system32\cmd.exe
D:\JoinEx>java JoinEx
Starting...
All done
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
second job = 1
second job = 2
second job = 3
second job = 4
second job = 5
second job = 6
second job = 7
second job = 8
second job = 9
second job = 10
first job = 9
first job = 10
```

Output: with join statements



```
C:\WINDOWS\system32\cmd.exe
D:\examples\multithreading>java JoinEx
Starting...
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
second job = 1
second job = 2
second job = 3
second job = 4
second job = 5
second job = 6
second job = 7
second job = 8
second job = 9
second job = 10
first job = 9
first job = 10
All done
```

Web Application Development

Lec 25

Web Application Development

- we used J2SE until now
- For web application development, we need J2EE (Servlets API, JSP API, JSF API etc)

Web Based Performance Review Management System

PERFORMANCE MANAGEMENT SYSTEM

► Manage Employees

Career

[Competency Scale](#)
[Competency Set](#)
[Reviews To Do](#)
[Initiate Review](#)

Organization

[Organization](#)
[Employee Position](#)
[Job](#)

First Name Last Name
Employee Category Position
Gender Year of Joining

	Emp Code	First Name	Last Name	Position Name	Category	Date Of Joining	Email
☐	LE005	Madiha	Shahid	Senior Software Engineer, VIPER	permanent	Feb 5, 2004	
☐	LE006	Qasim	Bilal	Software Engineer, VIPER	Permanent	Jun 5, 2005	qasim@lmkr.org
☐	LE007	Umer	Sohail	Software Engineer, VIPER	temporary	Mar 1, 2005	
☐	LE008	Zohaib	Hassan	Programmer, VIPER	temporary	Jun 5, 2004	

⏮ ⏪ ⏩ ⏭

Add

View

Update

Delete

Back

Web Based School Management System

**Educators School System**
The Best Education For You

AcademicsGradingStudentsTeachersUsersLogOff

Search Student

Back

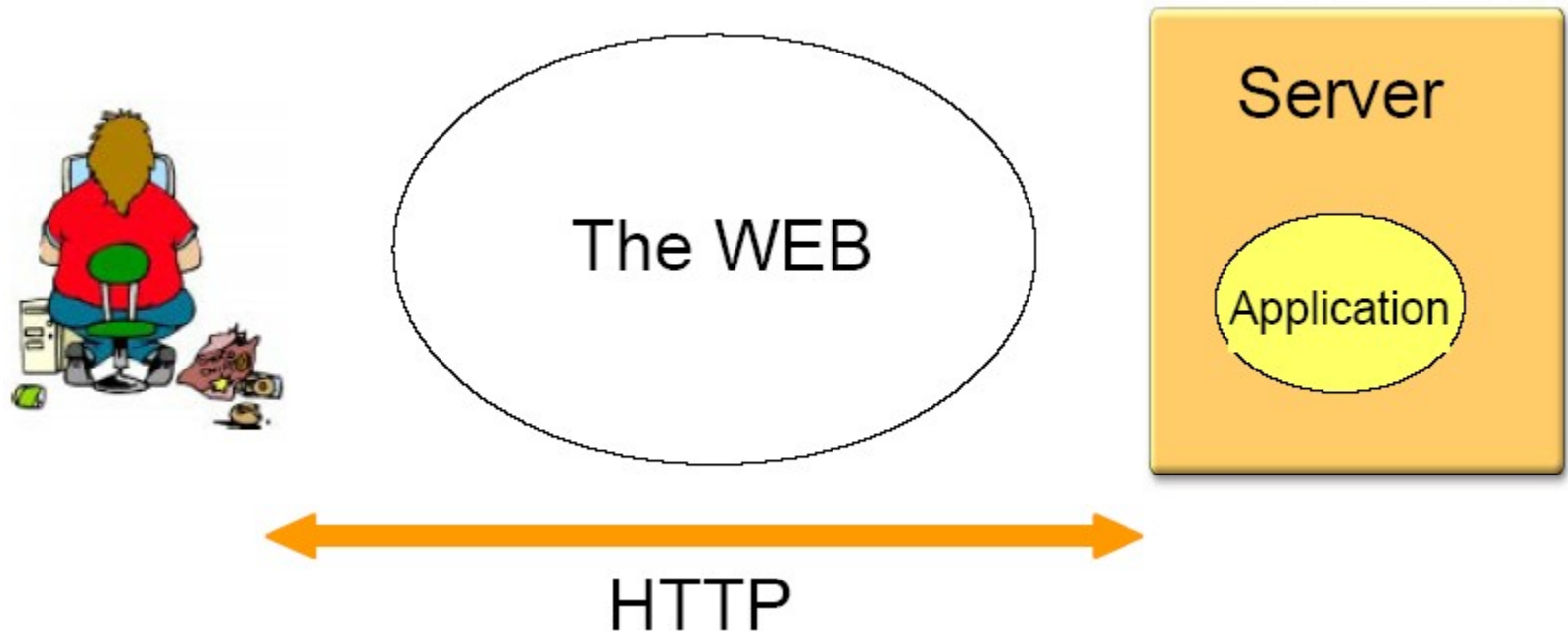
Enter Student Search Criteria

Roll No	
Name	
Class	Class 1
Section	Section45
Batch	
Rank	
Enrollment Year	

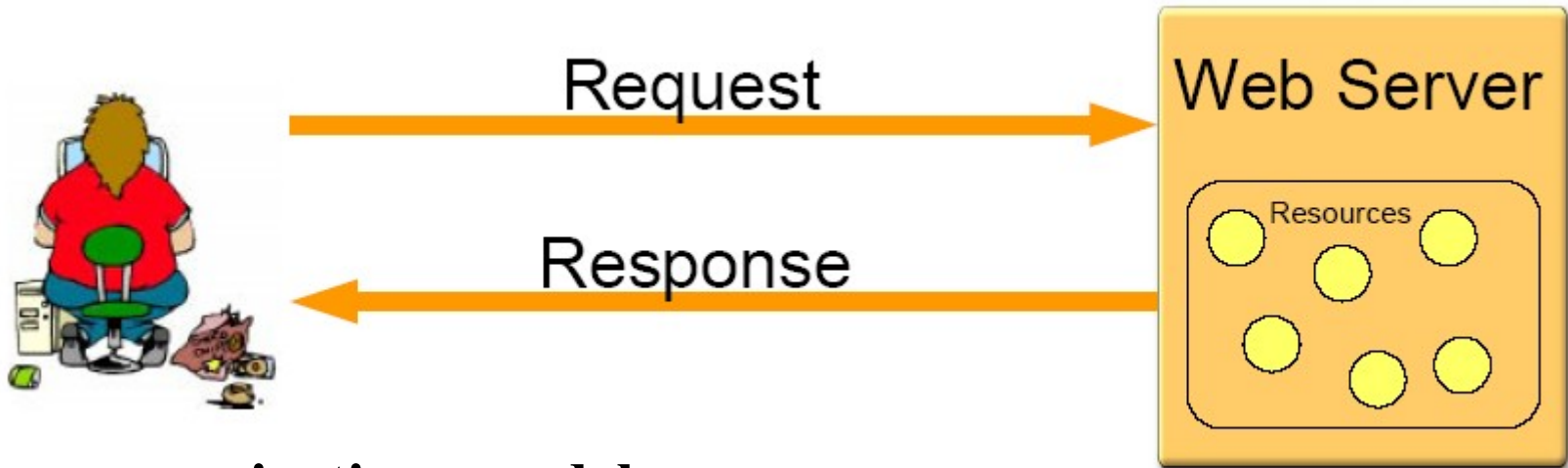
SearchAddCancelReset

| Search || Academics || Profile || Fee || Results || Back |

What is a Web Application



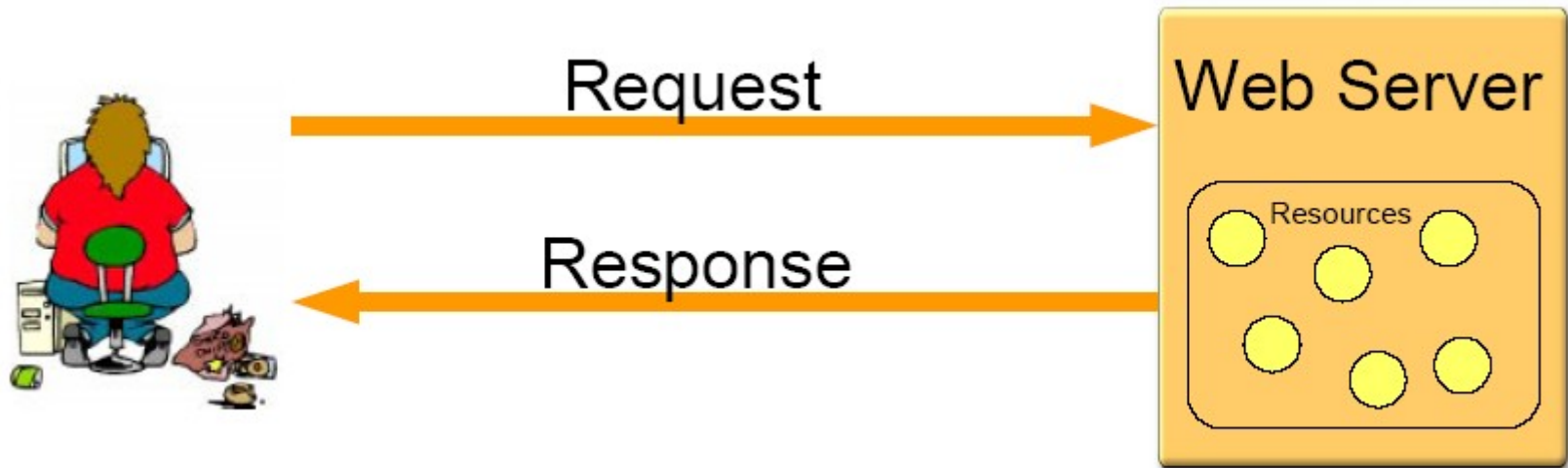
HTTP Communication Model



A communications model

- Client, often *but not always* a web browser, sends a request for a resource to a server.
- The server returns a response or an error message.
- Stateless
- No immediate feedback
- No details on how request is made

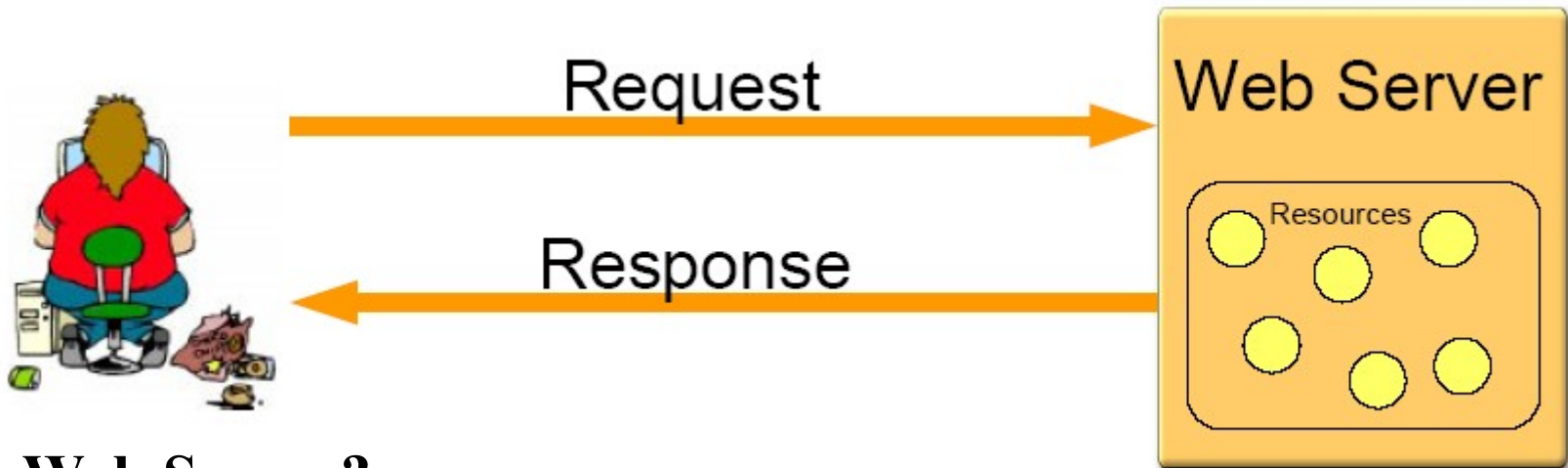
HTTP Communication Model



Different clients like

- Web browsers
- FTP clients (e.g., interarchy on MacOS X)
- Software registration programs
- telnet

HTTP Communication Model



Web Server?

Software that provides services to access the Internet, or an intranet.

A web server hosts web sites Provides support for HTTP and other protocols Executes server-side programs (such as CGI scripts or servlets) that perform certain functions.



HTTP



HTTP Basics

- HTTP is a request-response oriented protocol.
 - Parts of an HTTP request
 - Request Method
 - URI
 - Header Fields
 - Body
 - HTTP response contains
 - Result Code
 - Header Fields
 - Body
-

HTTP Request Example

```
GET /index.html HTTP/1.1
```

request line

```
Host: java.sun.com
```

request headers

```
User-Agent: Mozilla/4.5 [en]
```

```
Accept: image/gif, image/jpeg, image/pjpeg, */*
```

```
Accept-Language: en
```

```
Accept-Charset: iso-8859-1,*,utf-8
```

Request parameters etc

optional request body

Request Parameters

- Request can also contain additional info in the form of parameters
 - In the URL itself as a query string

`http://www.gmail.com/register?name=ali&state=punjab`
 - As part of the request body
 - Shown earlier in the diagram
-

HTTP Response Example

```
HTTP/1.1 200 OK
```

status line

```
Last-Modified: Mon, Aug 4 2003 22:10:40 GMT
```

```
Date: Wed, Aug 8 2003 14:23:35 GMT
```

```
Status: 200
```

```
Content-Type: text/html
```

```
Content-Length: 59
```

response headers

```
<html>
```

optional response body

```
  <body>
```

```
    <h1>Hello World!</h1>
```

```
  </body>
```

```
</html>
```

HTTP Response Codes

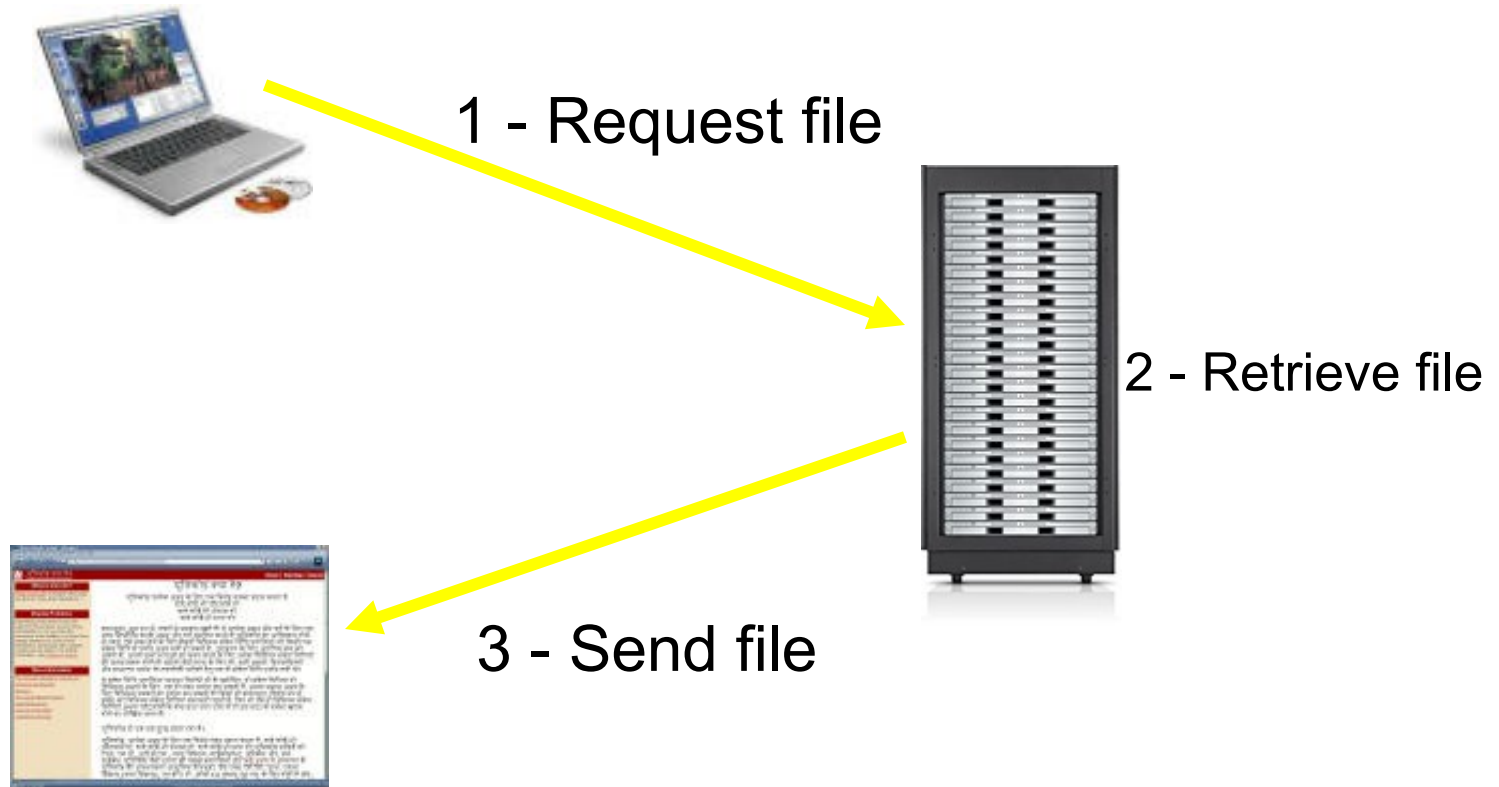
- Codes fall into five general categories
 - **100-199**
 - Codes in the 100s are informational, indicating that the client should respond with some other action.
 - **200-299**
 - Values in the 200s signify that the request was successful.
 - **300-399**
 - Values in the 300s are used for files that have moved and usually include a Location header indicating the new address.
 - **400-499**
 - Values in the 400s indicate an error by the client.
 - **500-599**
 - Codes in the 500s signify an error by the server.

Common HTTP Response Codes

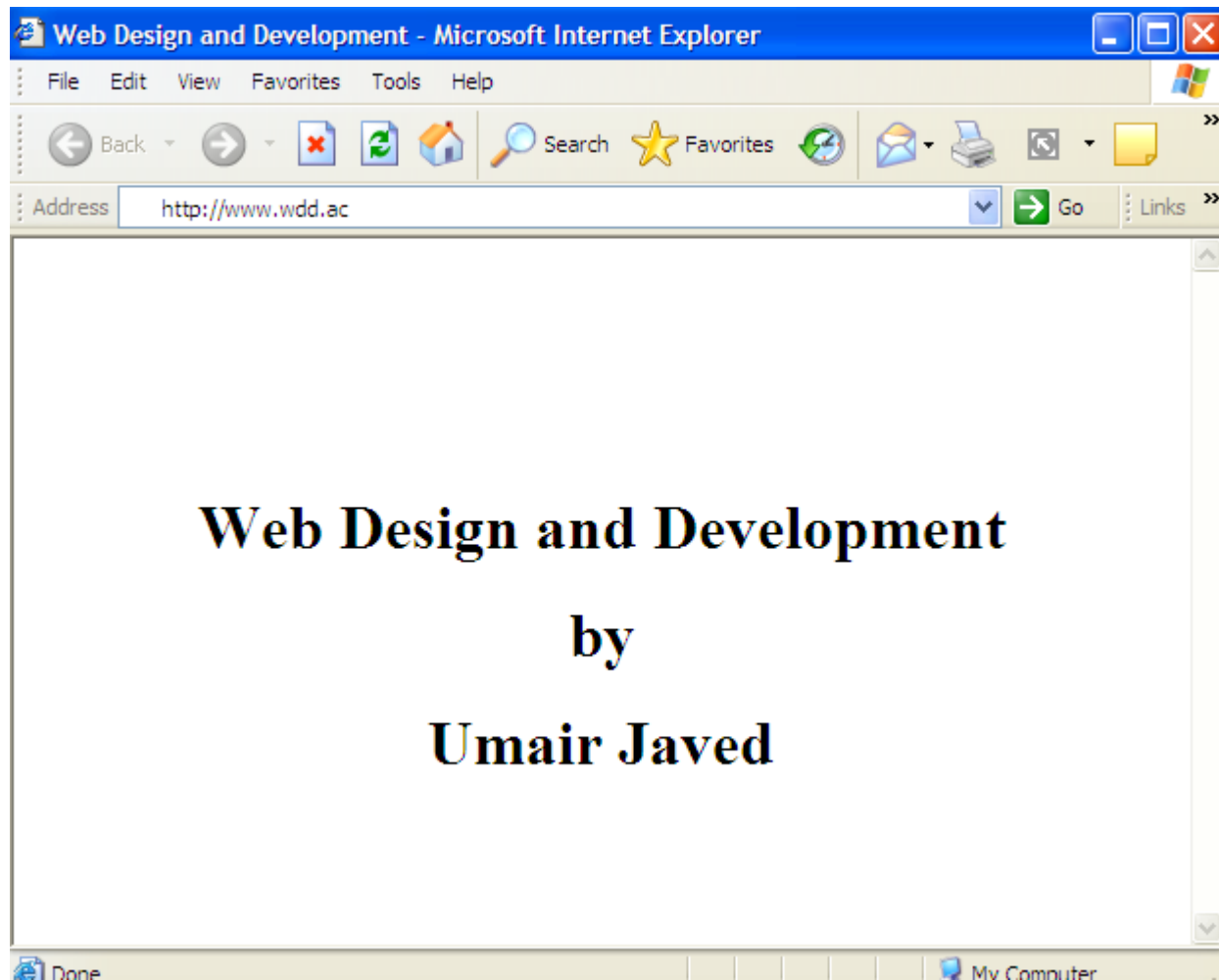
- **200:** means every thing is fine.
 - **404:** Indicates that the requested resource is not available
 - **401:** Indicates that the request requires HTTP authentication
 - **403:** Indicates that access to the requested resource has been denied
 - **503:** Indicates that the HTTP server is temporarily overloaded and unable to handle the request
-

Server Side Programming

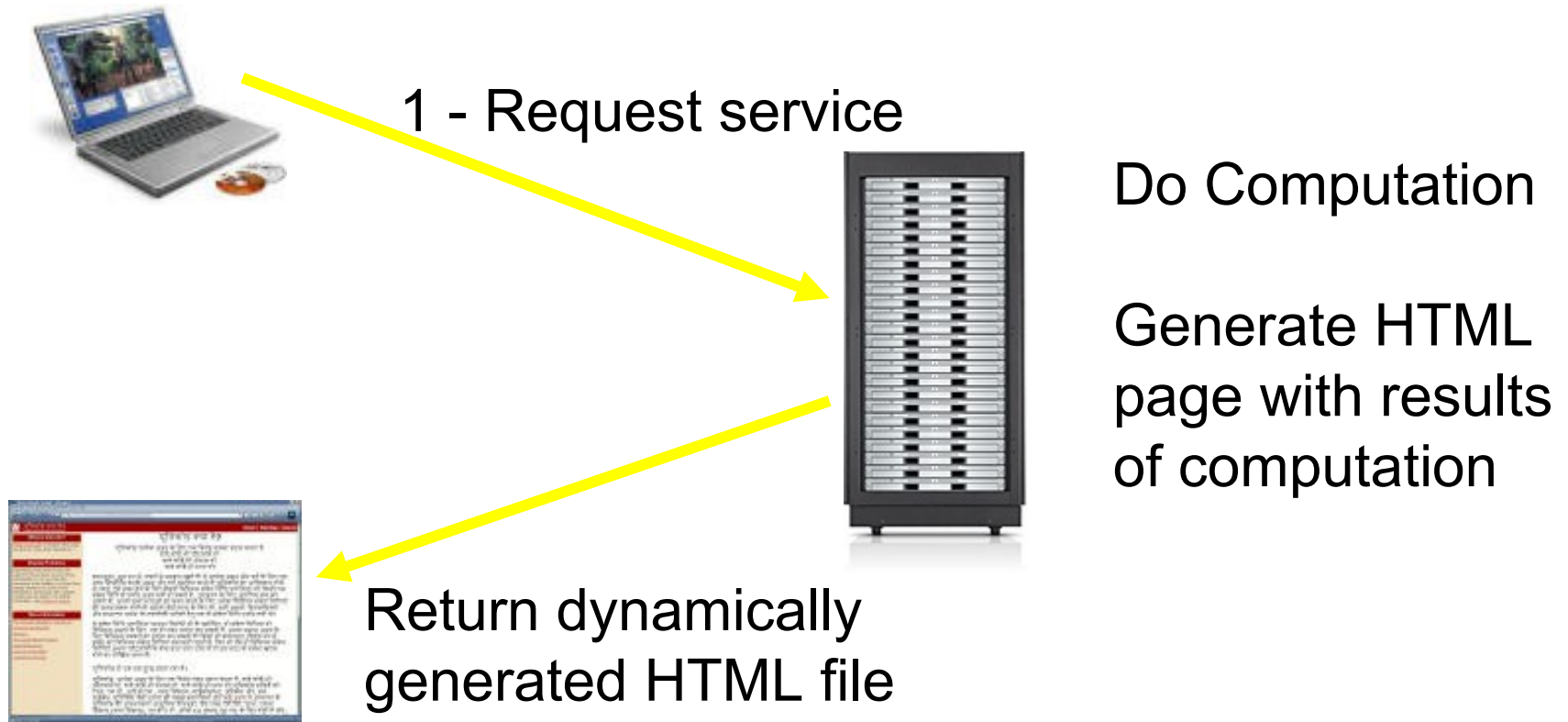
Static Pages



Example: Static Page



Dynamic Pages



Why build Pages Dynamically?

- The Web page is based on data submitted by the user
 - E.g. results page from search engines and order confirmation pages at on line stores
 - The Web page is derived from data that changes frequently
 - E.g. a weather report or news headlines page
 - The Web page uses information from databases or other server-side resources
 - E.g. an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale\
 - Course Registration System
-



[Web](#) [Images](#) [Groups](#) [News](#) [Froogle](#) [Local](#) [more »](#)

Google Search

I'm Feeling Lucky

[Advanced Search](#)

[Preferences](#)

[Language Tools](#)

[Advertising Programs](#) - [Business Solutions](#) - [About Google](#) - [Go to Google Deutschland](#)

©2005 Google - Searching 8,168,684,336 web pages

Course Registration System

**GOVERNMENT COLLEGE
UNIVERSITY FAISALABAD**

Search Sitemap Contact

Quicksearch...

[GCUF](#) [News & Events](#) [Prospective Students](#) [Students](#) [Alumni](#) [Executive Education](#) [Faculty & Research](#)



PCO Options:

- Manage Quarter
- Manage Programs
- Manage Fields
- Customized Query
- Manage Waiver
- Manage Instructors
- Search Details
- FeedBack
- Evaluation Options
- Manage Streams
- Manage Courses
- Manage Venues

Fields Please select a field to view its detail

Field ID	Field Name	Field Code		
Details	Computer Science	CS	Edit	Delete
Details	Business Administration	BA	Edit	Delete
Details	Law and Politics	LLB	Edit	Delete
Details	Computer Engineering	CE	Edit	Delete

4 Record(s) exist

details details of the currently selected field

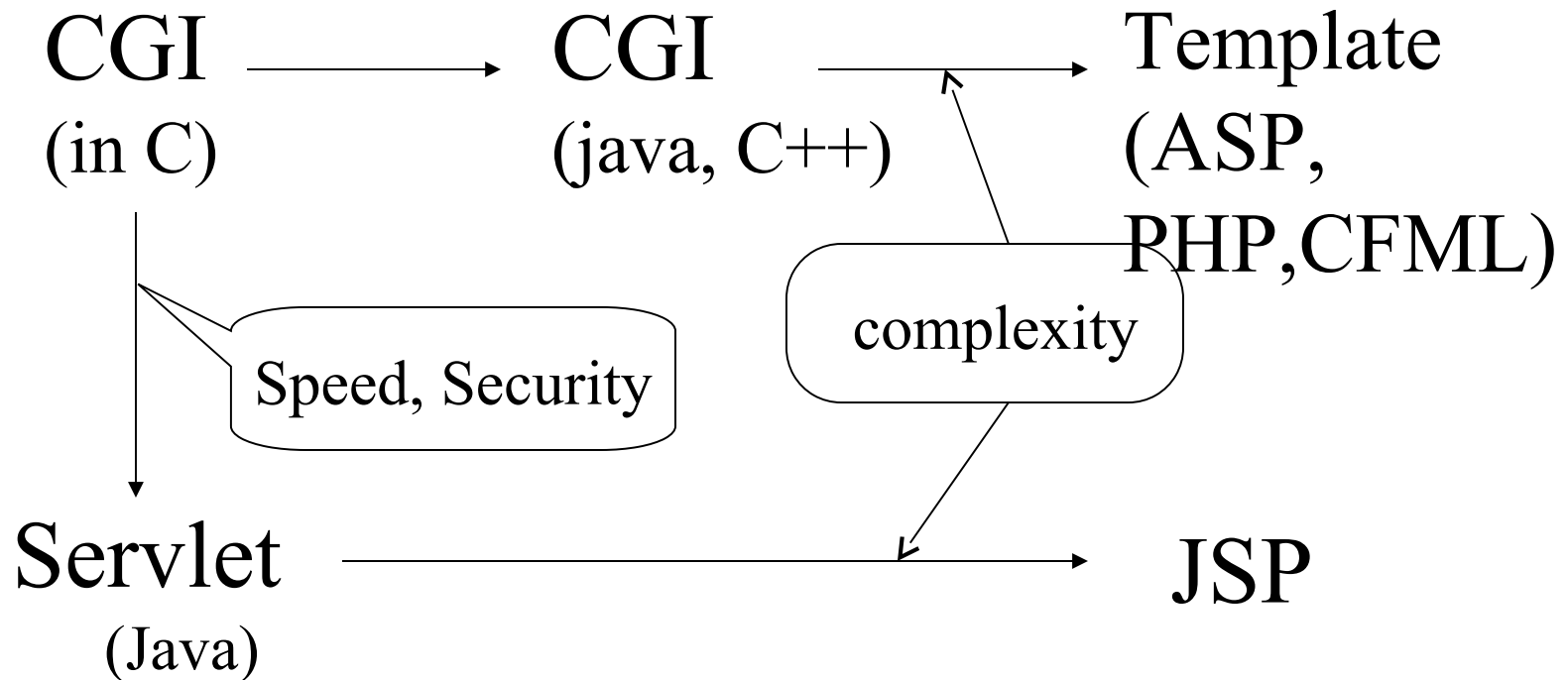
Program Type	Major Units	Minor Units	Min GPA	Min Units	Policy URL		
Masters	9	3	2.5	45	policy.doc	Edit	Delete
Bachelor	6	3	2	133	policyB.doc	Edit	Delete

2 Record(s) found corresponding to selected field

What is Server-Side Programming (SSP)?

- Technologies for developing web pages that include *dynamic* content.
- Can produce web pages that contain information that is connection- or time-dependent.

Dynamic Web Content Technologies Evolution (only figure)



Web-Based Enterprise Applications in Java

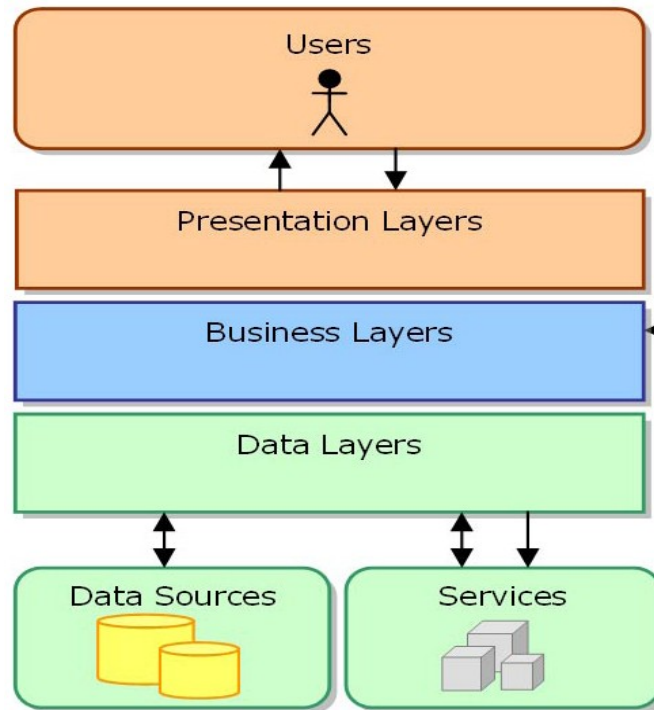
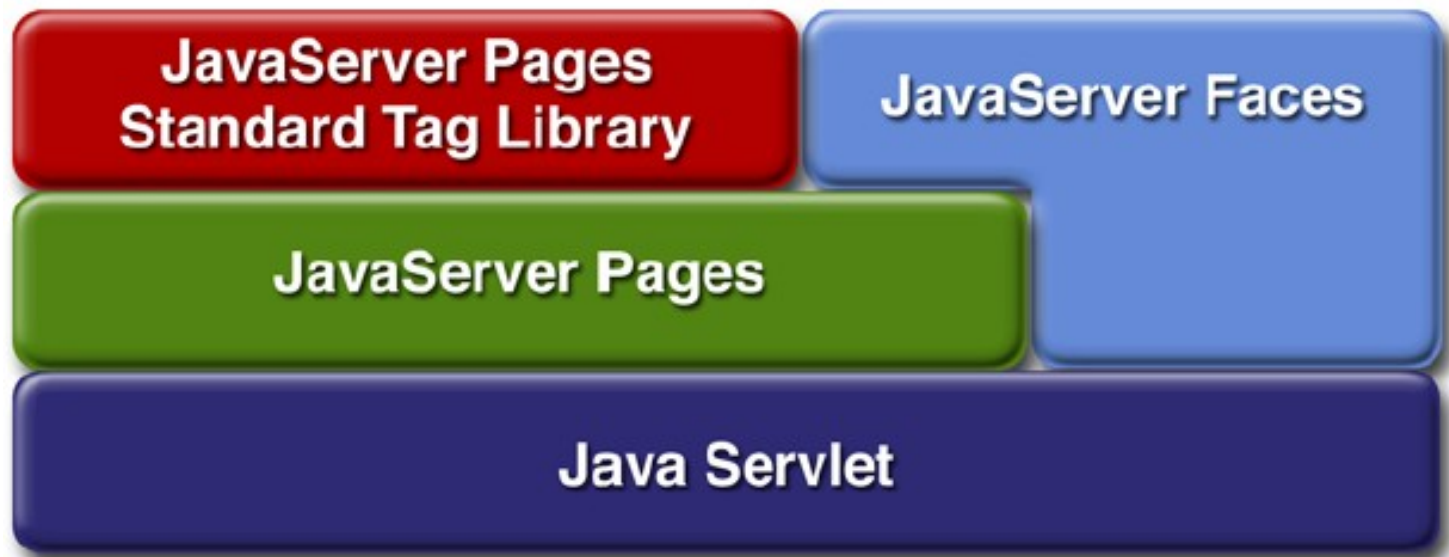


Figure shows a simplified view of one application and its layers.

Java Web Application Technologies (Presentation/Web Tier)



Intro to Servlets

Lec 26

Web-Based Enterprise Applications in Java

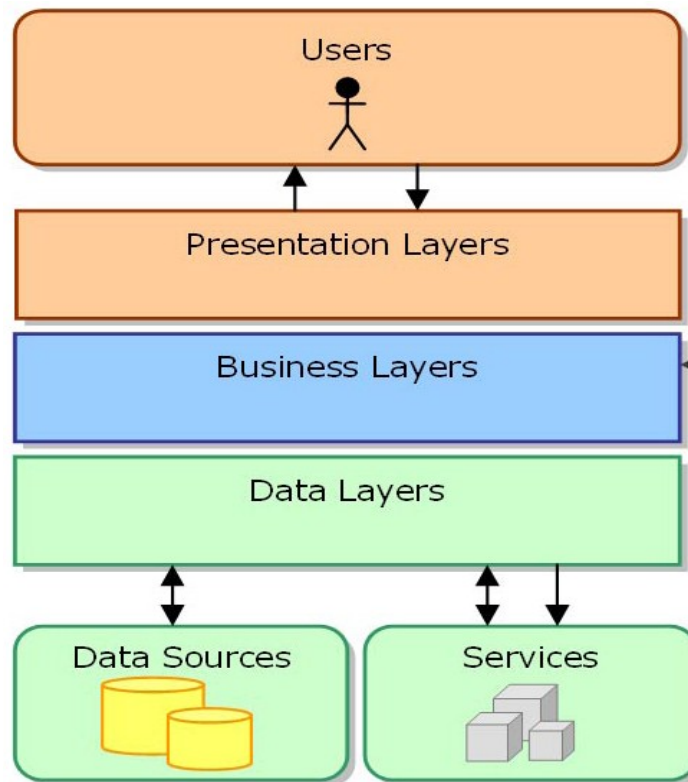
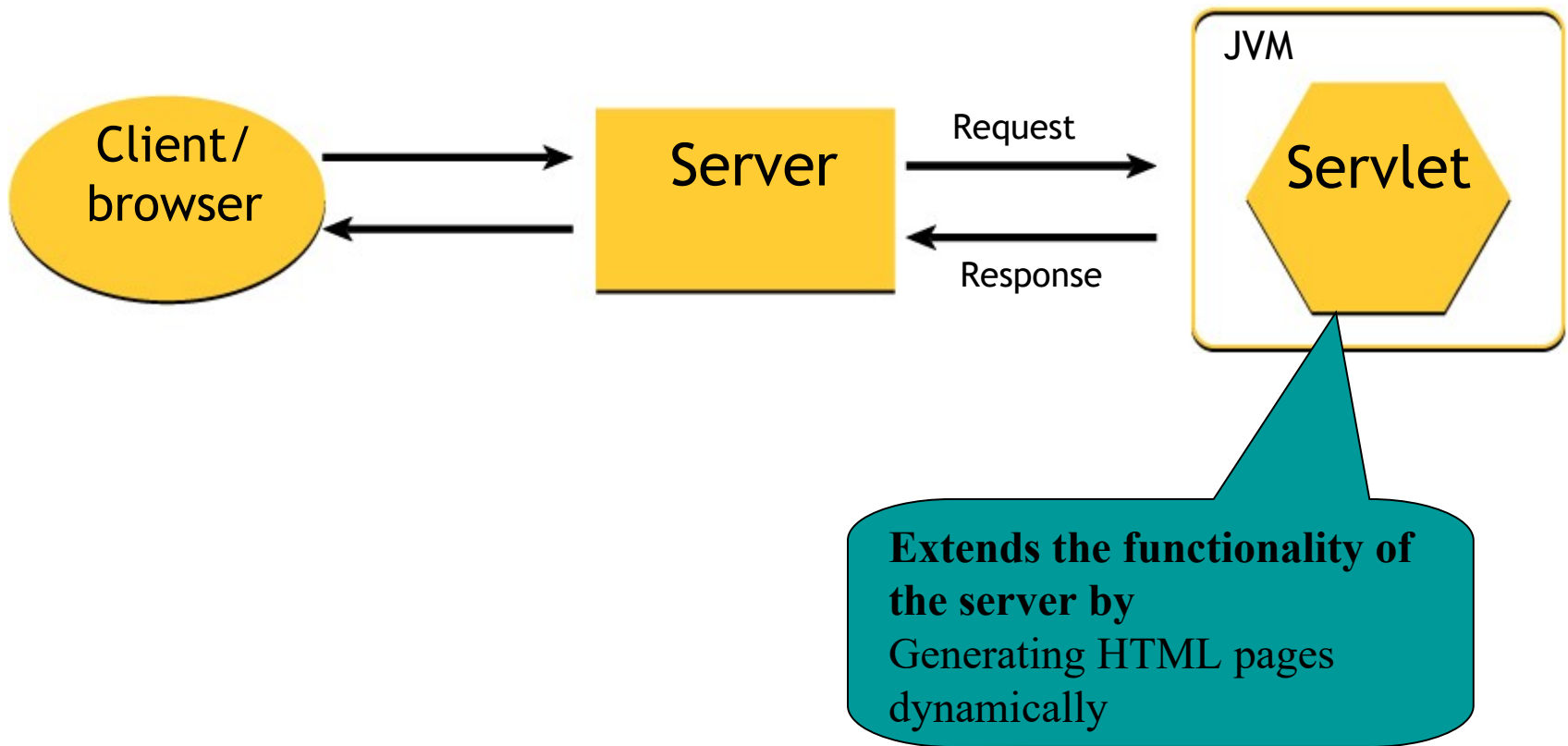


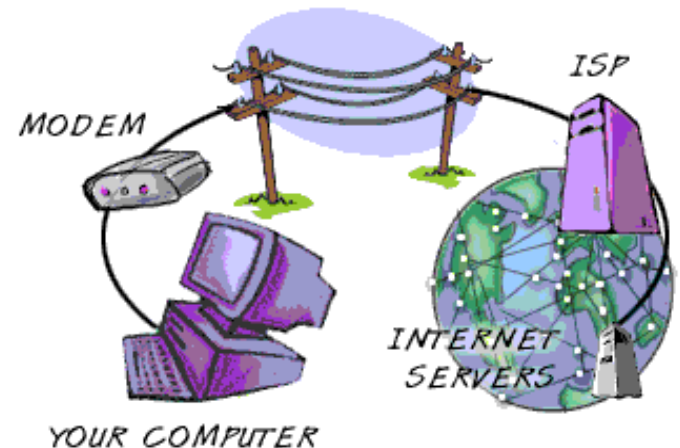
Figure shows a simplified view of one application and its layers.

Java Servlets



What Do They Do?

- Anything that java can do (e.g. connect to database services)
- Handle data/requests sent by users (clients)
- Create and format results
- Send results back to user



Java Servlets

- Advantages over other SSP technologies:
 - ❑ Can use the Java APIs e.g. JDBC.
 - ❑ Persistent between invocations, avoiding process instantiations.
 - ❑ Portable across operating systems and servers.
 - ❑ Good security.
 - ❑ Is integrated closely with the J2EE environment.
-

Software Requirements

Software Requirements

- To use Java servlets, following softwares will be needed
 - J2SE
 - Additional J2EE based libraries for servlets such as `servlet-api.jar`, `jsp-api.jar`. You can download these APIs separately but they are also available with the web server you'll be using.
 - A capable servlet web engine (web server)
-

Jakarta Servlet Engine (Tomcat)



- Jakarta is an Apache project
 - Tomcat is one of its subprojects
- Tomcat is the official reference implementation for the Java servlet and JavaServer pages technologies (according to Sun's specifications)
- Tomcat is developed in an open and participatory environment and released under the Apache software license

Environment Setup

Environment Setup

Steps

1. Download the Apache Tomcat Server
 2. Install Tomcat
 3. Set the JAVA_HOME variable
 4. Set the CATALINA_HOME variable
 5. Set the CLASSPATH variable
 6. Test the Server
-

1. Download the Apache Tomcat Server

- From the <http://jakarta.apache.org>
 - Download the zip file for the current release of Jakarta Tomcat
 - For windows, there is also a **.exe** installer

2. Install Tomcat

■ Installing .zip file

- Unzip the file into a location (e.g. `C:\`)
- The zip file had embedded subdirectories (e.g., `jakarta-tomcat-5.5.9`).
- `C:\jakarta-tomcat-5.5.9` generally referred as `install_dir`

■ Installing .exe file

- Check the handouts
-

3. Set the JAVA_HOME variable

- Set the JAVA_HOME environment variable to tell Tomcat where to find Java
- This variable should list the base JDK installation directory, not the `bin` subdirectory

```
JAVA_HOME = c:\j2sdk1.5.0
```

4. Set the CATALINA_HOME variable

- Set the CATALINA_HOME environment variable to refer to the top-level directory of the Apache Tomcat installation, not the `bin` subdirectory

```
CATALINA_HOME = c:\jakarta-tomcat-5.5.9
```

5. Set the CLASSPATH variable

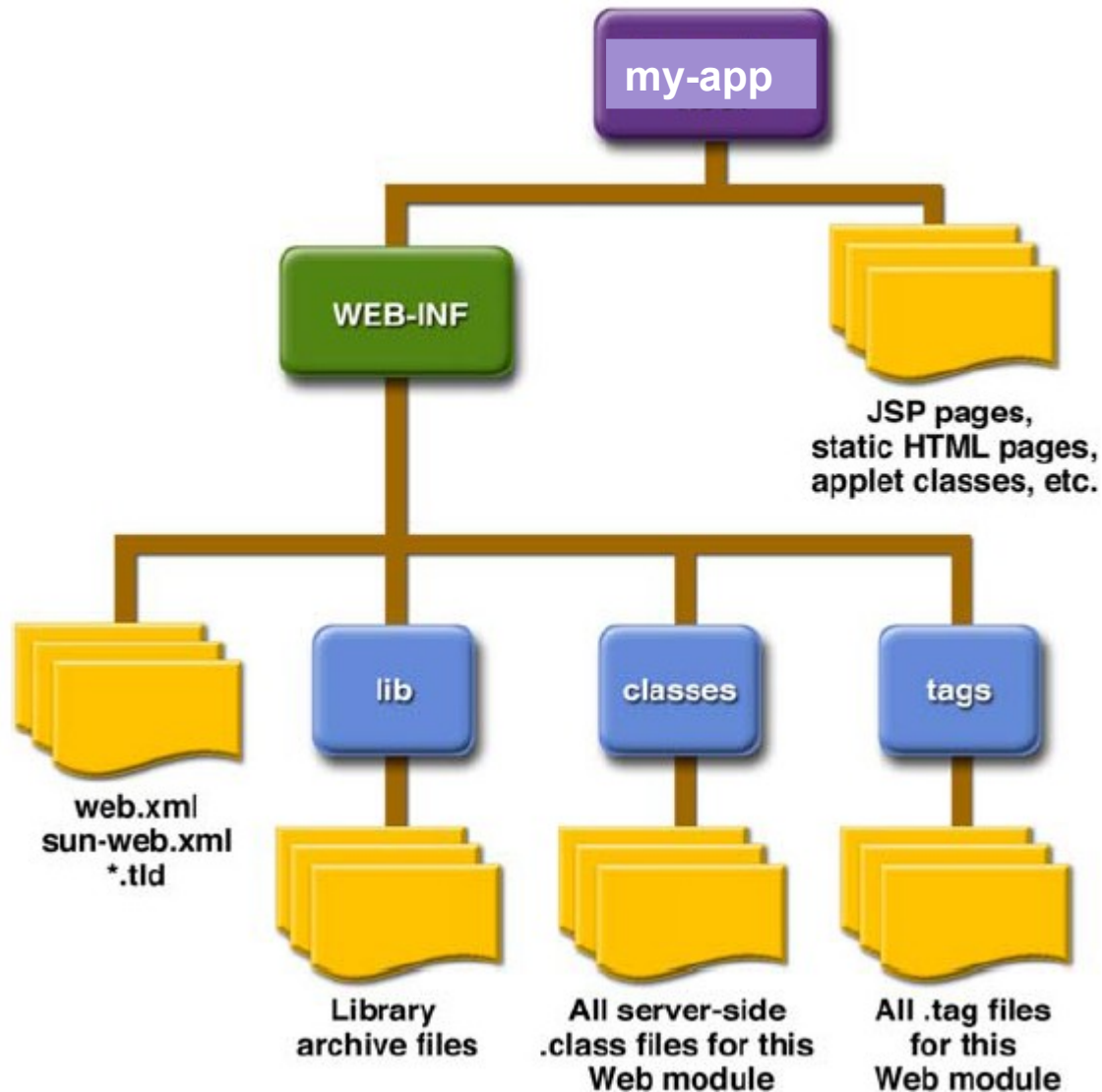
- Since `servlets` and `JSP` are not part of the J2SE, `javac` doesn't know about the `servlet` and `jsp` classes
- The standard Tomcat locations for these files are
 - ▣ `install_dir/common/lib/servlet-api.jar`
 - ▣ `install_dir/common/lib/jsp-api.jar`
- You need to include both files in your CLASSPATH

6. Test the Server

- Before making your own Servlets and JSP, verify that the server can start
 - Run `install_dir/bin/startup.bat`
 - Enter the URL `http://localhost:8080` in your browser and make sure you will get the Tomcat homepage
-

Lets do it LIVE

Tomcat Directory Structure

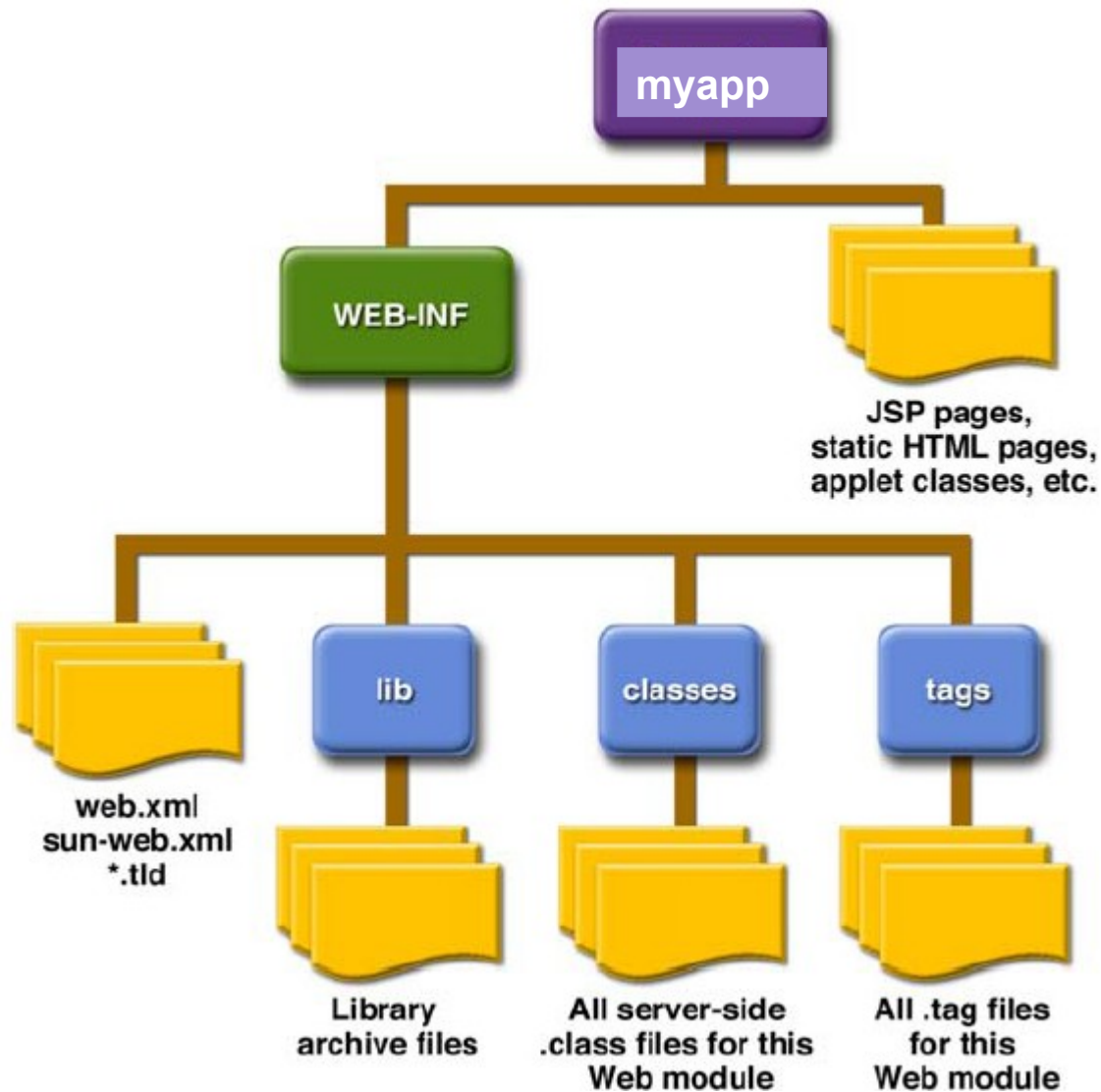


Java Servlets

Lec 27

Creating a Simple Web Application in Tomcat

Tomcat Directory Structure



Tomcat Setup

[-] jakarta-tomcat-5.0.27

[-] bin

[+] common

[+] conf

logs

[+] server

[+] shared

temp

[-] webapps

[+] admin

[+] balancer

[+] jsp-examples

[+] manager

[-] my-app

images

[-] WEB-INF

classes

lib

tags

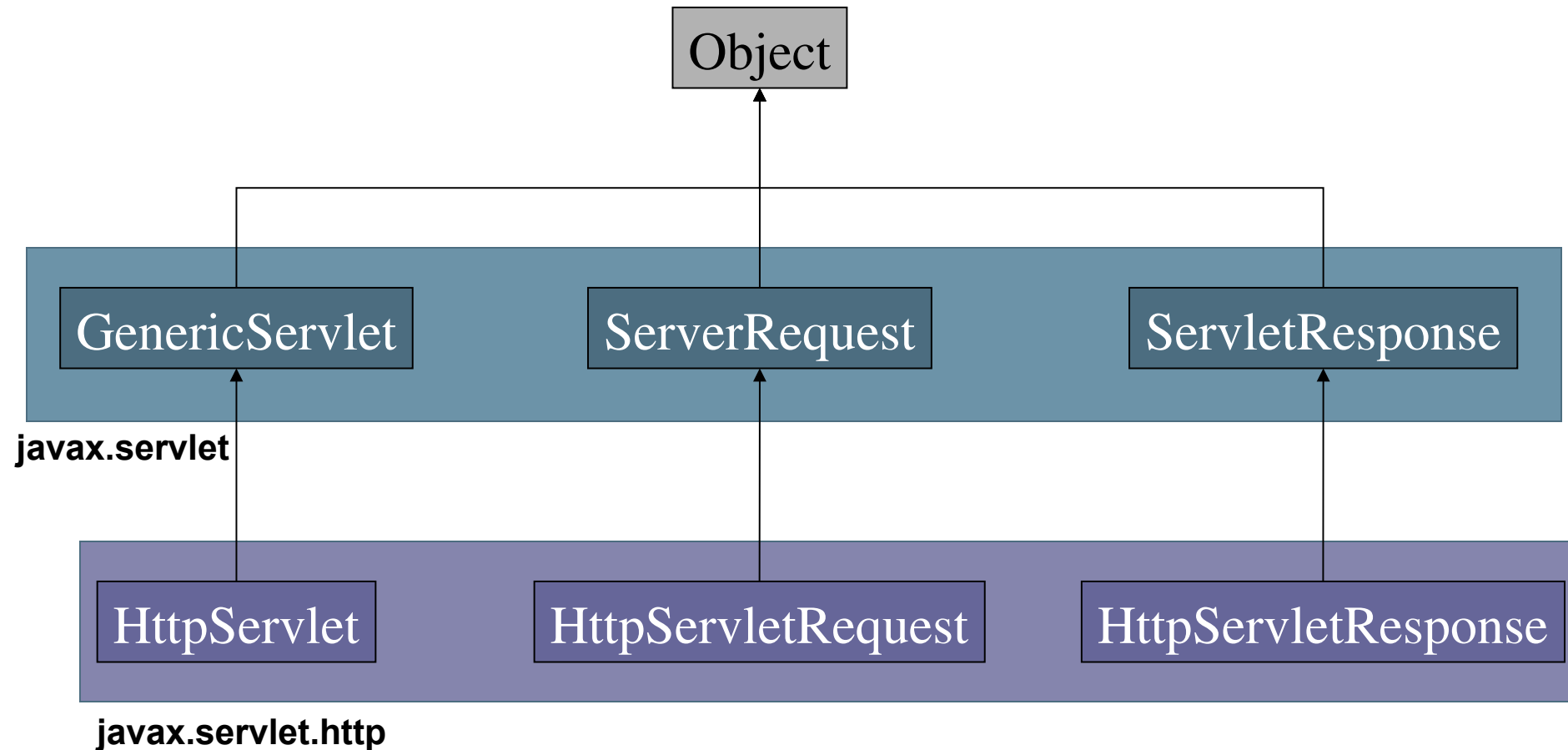
**Folder for
HTML and
JSP pages**

**Folder for
servlet and other
java classes**

Servlet Types

- Servlets are based on two main packages
 - `javax.servlet`
 - `javax.servlet.http`
- **GenericServlet**
 - For writing protocol independent servlets
- **HttpServlet**
 - Extends from `GenericServlet` class
 - Adds functionality for writing HTTP specific servlets

Servlet class hierarchy



Writing Servlet

- Every servlet must implement the `javax.servlet.Servlet` interface
 - Contains the servlet's life cycle methods etc.
- These are implemented by `GenericServlet` and `HttpServlet` classes
- **Extend** your servlet from one of these classes and add your own functionality

```
public class MyServlet extends GenericServlet
```

```
public class HelloServlet extends HttpServlet
```

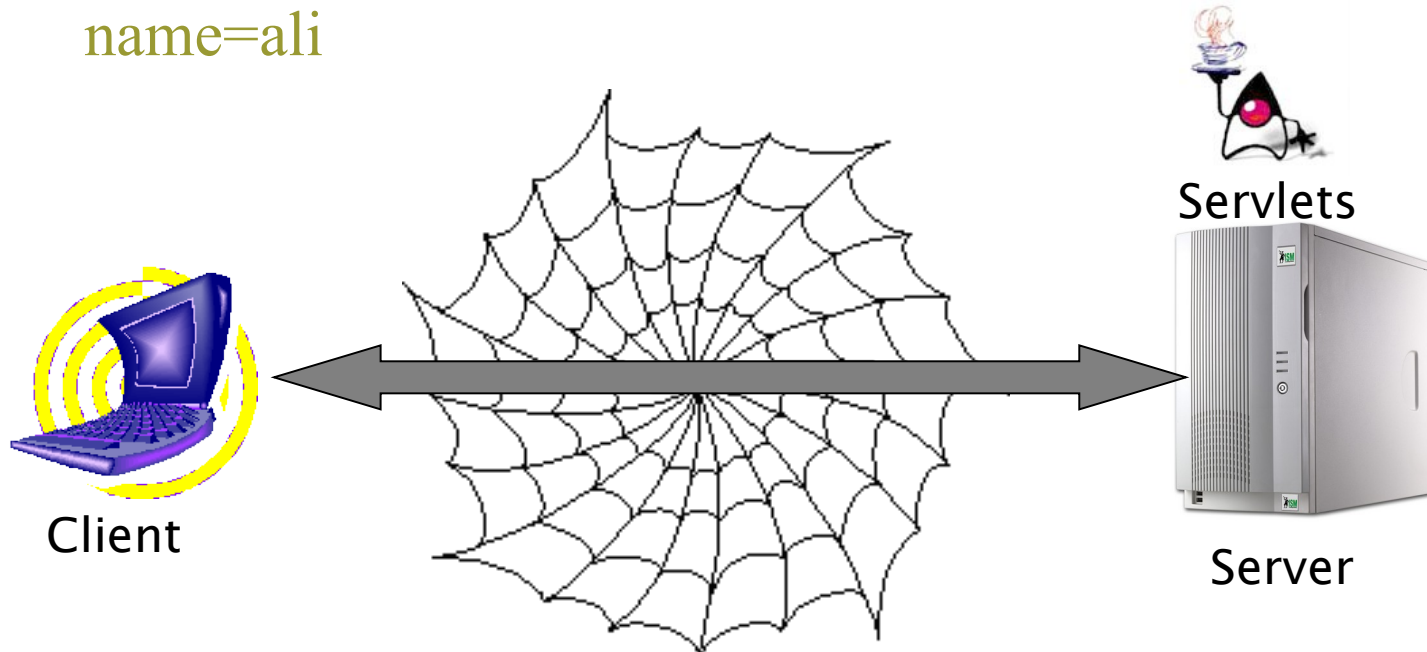
Types of HTTP Requests

- ❑ Get
 - ❑ Post
 - ❑ Delete
 - ❑ Options
 - ❑ Put
 - ❑ Trace
-

How HTTP Sends Request

Some HTTP request types

- **Get** -- Attr/Val pairs attached after ? of URL
E.g. `http://www.gmail.com/register?`
`name=ali`



- **Post** -- Attr/Val pairs attached with the **request body**

HTTP Request Example

```
GET /index.html HTTP/1.1
```

request line

```
Host: java.sun.com
```

request headers

```
User-Agent: Mozilla/4.5 [en]
```

```
Accept: image/gif, image/jpeg, image/pjpeg, */*
```

```
Accept-Language: en
```

```
Accept-Charset: iso-8859-1,*,utf-8
```

Request parameters etc

optional request body

Writing Servlet

Steps for making a HelloWorldServlet

1. Create a directory structure for your application (myapp). This is a one time process for any application
 2. Create a `HelloWorldServlet` source file by extending this class from `HttpServlet` and override your desired method
 3. Compile it (If compiler complains of not having required packages, check your class paths)
-

Writing Servlet

Steps for making a HelloWorldServlet

1. Place the class file in the *classes* folder of your webapplication. (myapp) If you are using packages (recommended) then create a complete directory structure under classes folder
 2. Create a deployment descriptor (`web.xml`) and put it inside WEB-INF folder
 3. Restart your server, if already running
 4. Access your application using Web browser
-

HelloServlet

Example Code

HelloWorldServlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("Hello World!");
    }
}
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<web-app>
```

```
  <servlet>
```

```
    <servlet-name>HelloWorldServlet</servlet-name>
```

```
    <servlet-class>HelloWorldServlet</servlet-class>
```

```
  </servlet>
```

```
  <servlet-mapping>
```

```
    <servlet-name>HelloWorldServlet</servlet-name>
```

```
    <url-pattern>/myfirstservlet</url-pattern>
```

```
  </servlet-mapping>
```

```
</web-app>
```

Compiling and Invoking Servlets

- Compile `HelloWorldServlet.java` using `javac` command
- Put `HelloWorldServlet` class in

`install_dir/webapps/myapp/WEB-INF/classes`

- Put `web.xml` file in

`install_dir/webapps/myapp/WEB-INF`

- Invoke your servlet by writing following command in the web browser

`http://localhost:8080/myapp/myfirstservlet`

Lets do it LIVE

Free Servlet and JSP Engines

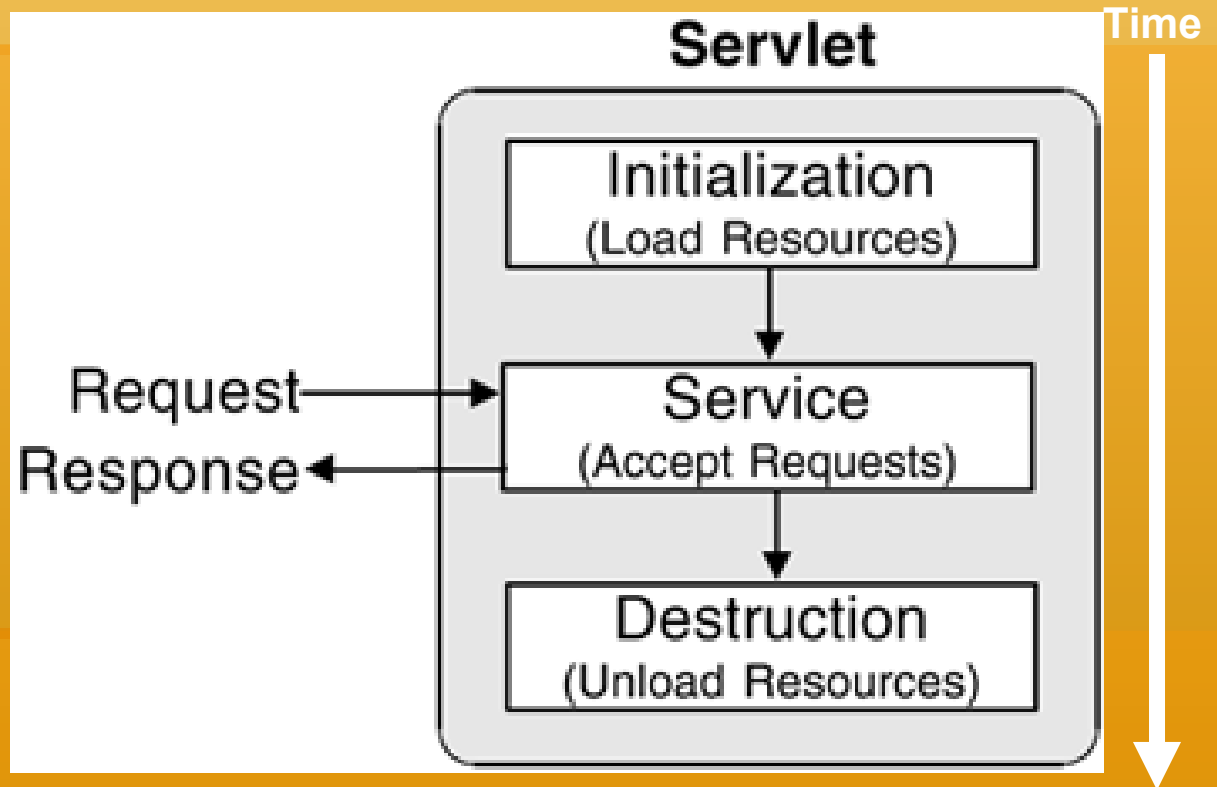
- Apache Tomcat
 - <http://jakarta.apache.org/tomcat/>
 - Allaire/Macromedia JRun
 - <http://www.allaire.com/products/jrun>
 - New Atlanta ServletExec
 - <http://www.servletexec.com>
 - Causho's Resin
 - <http://www.caucho.com>
-

Servlet Lifecycle

Lec 28

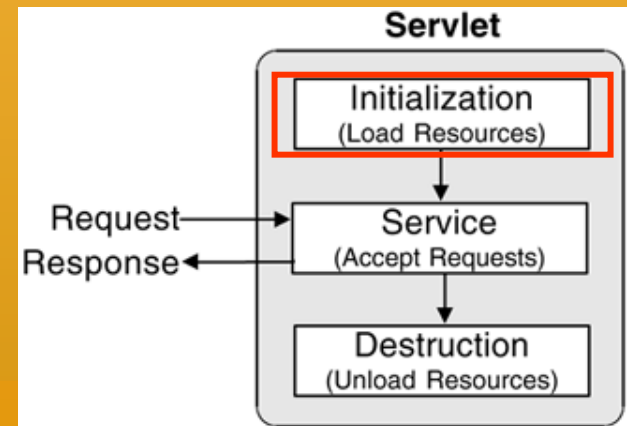
Servlet Life Cycle

- Initialize
- Service
- Destroy



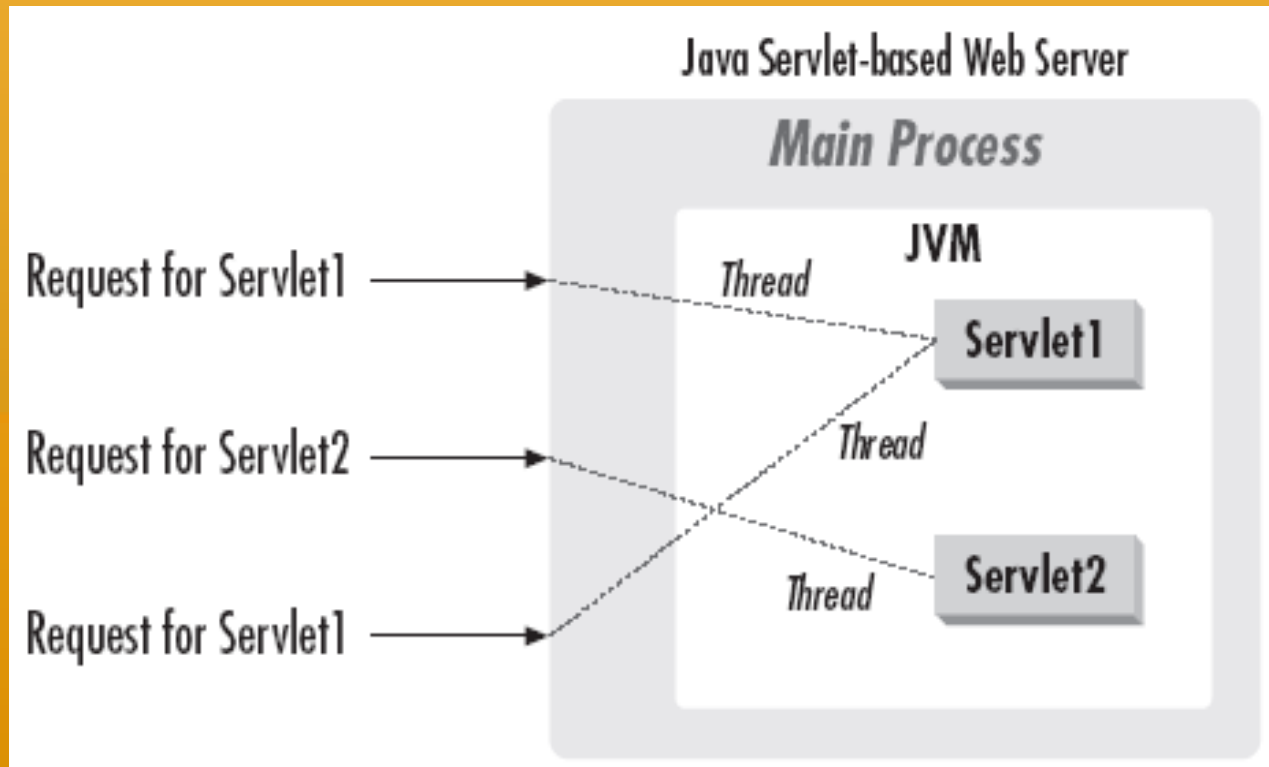
Life Cycle: Initialize

- Executed once, when the servlet gets loaded for the first time
- Not called for each client request
- Put your initialization code here. (No constructor available)



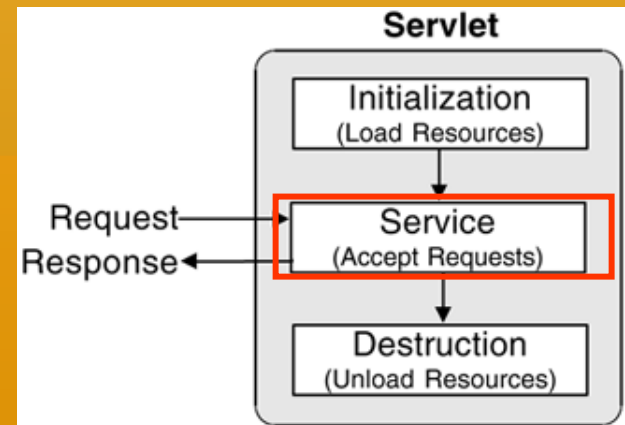
Life Cycle: Service

- On each request, the server spawns a new thread and calls service()

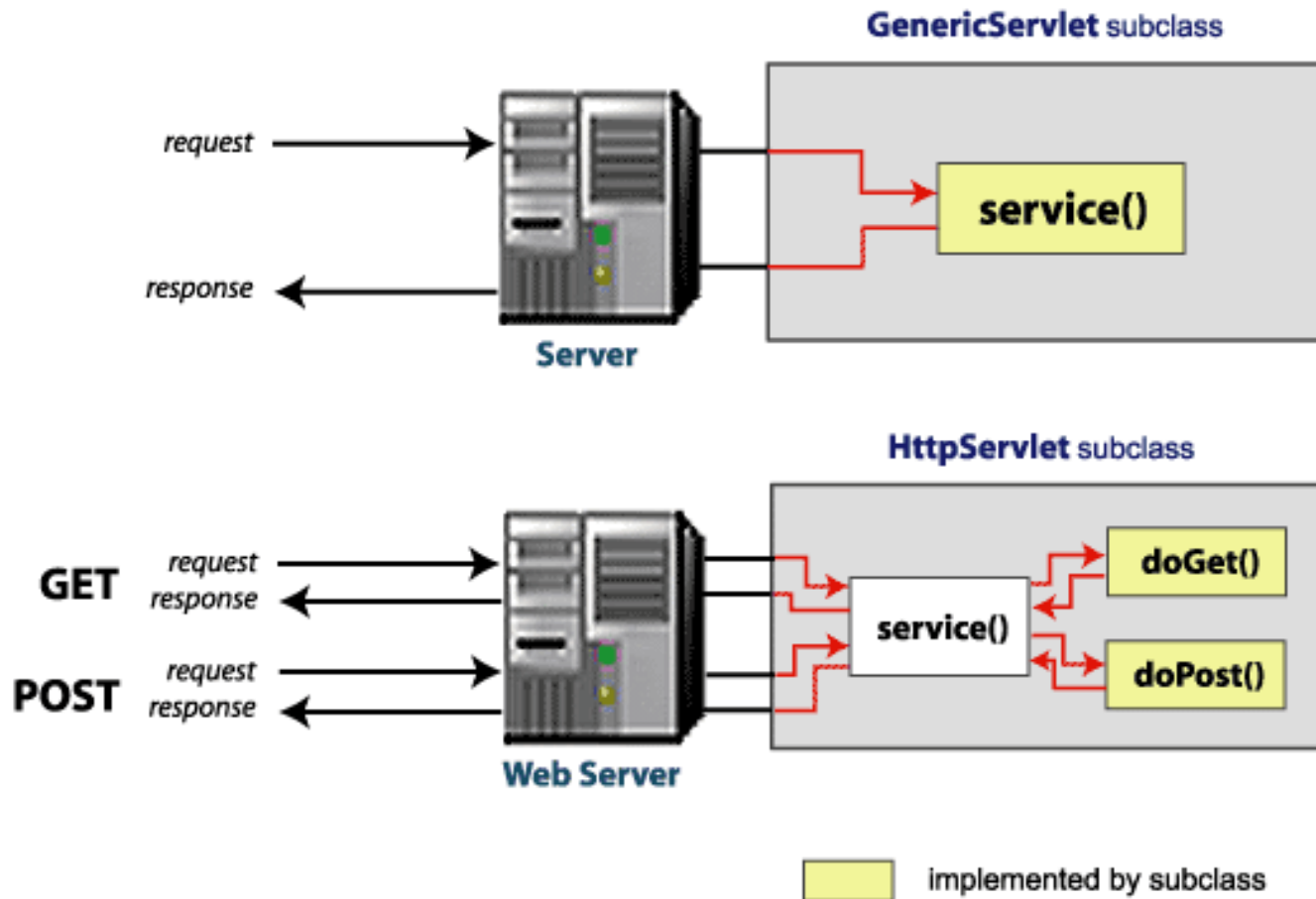


Life Cycle: Service

- `service()` checks HTTP request type and calls `doGet()` or `doPost()`
- Override `doGet()` or `doPost()` to provide desired behaviour.

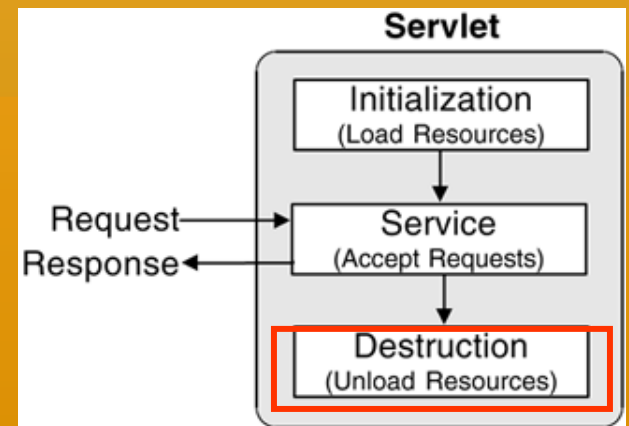


Life Cycle: Service

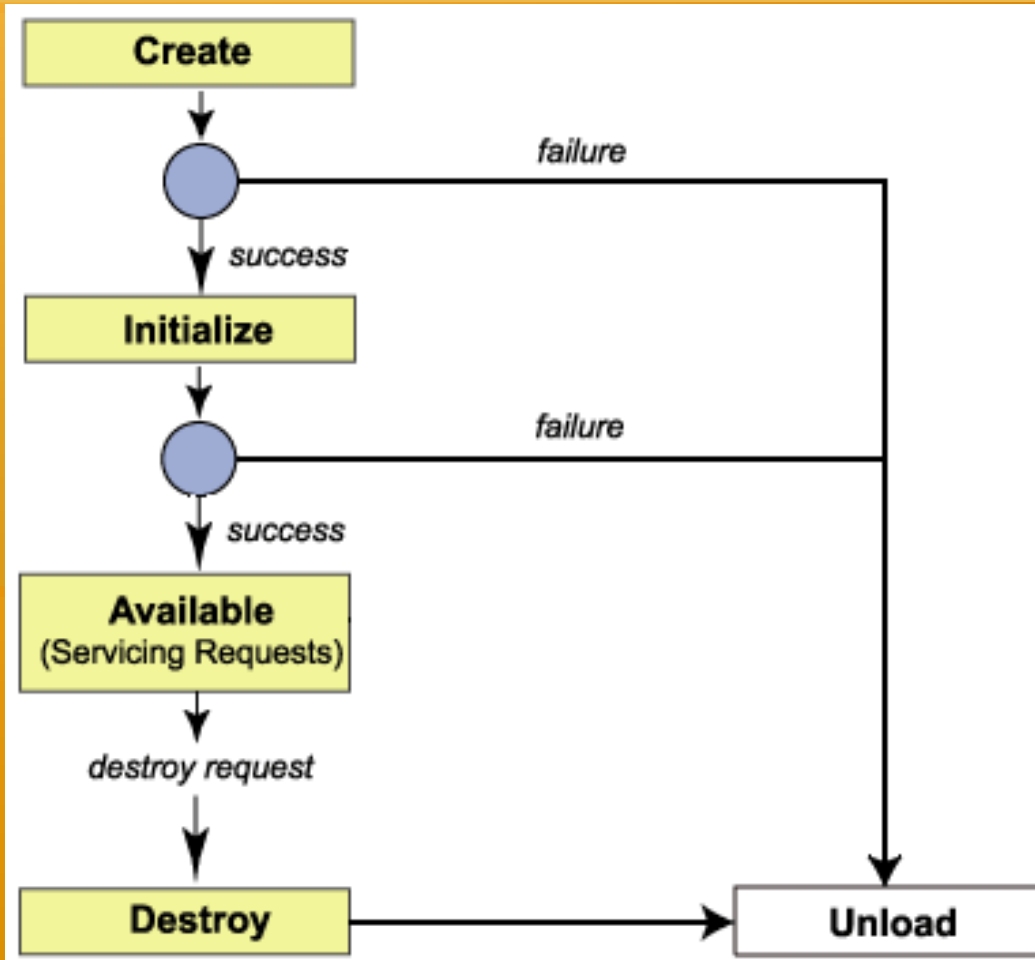


Life Cycle: Destroy

- `destroy()` method is called only once
- Call takes place when
 - Application is stopped
 - Servlet container shuts down
- Allows resources to be freed



Servlet Life Cycle Summary



Java Servlets

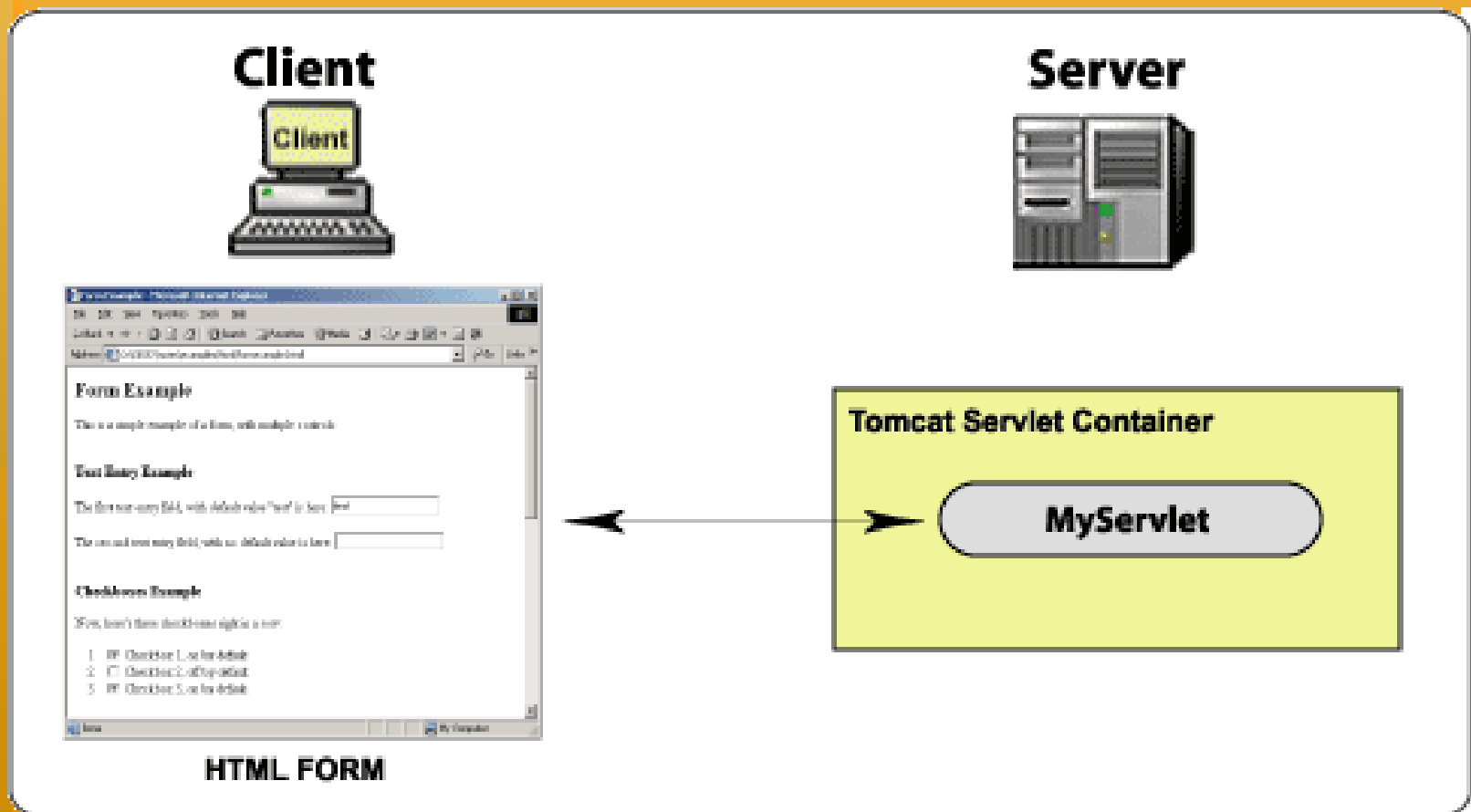
Form Data

How client sends data?

- When a user submits a browser request to a web server, it sends two categories of data:
 - **Form Data:** Data that the user explicitly typed into an HTML form.
 - For example: registration information.
 - **HTTP Request Header Data:** Data that is automatically appended to the HTTP Request from the client.
 - For example: cookies, browser type, browser IP address.

Html Form & Servlet

- HTML is Graphical User Interface (GUI) for a servlet

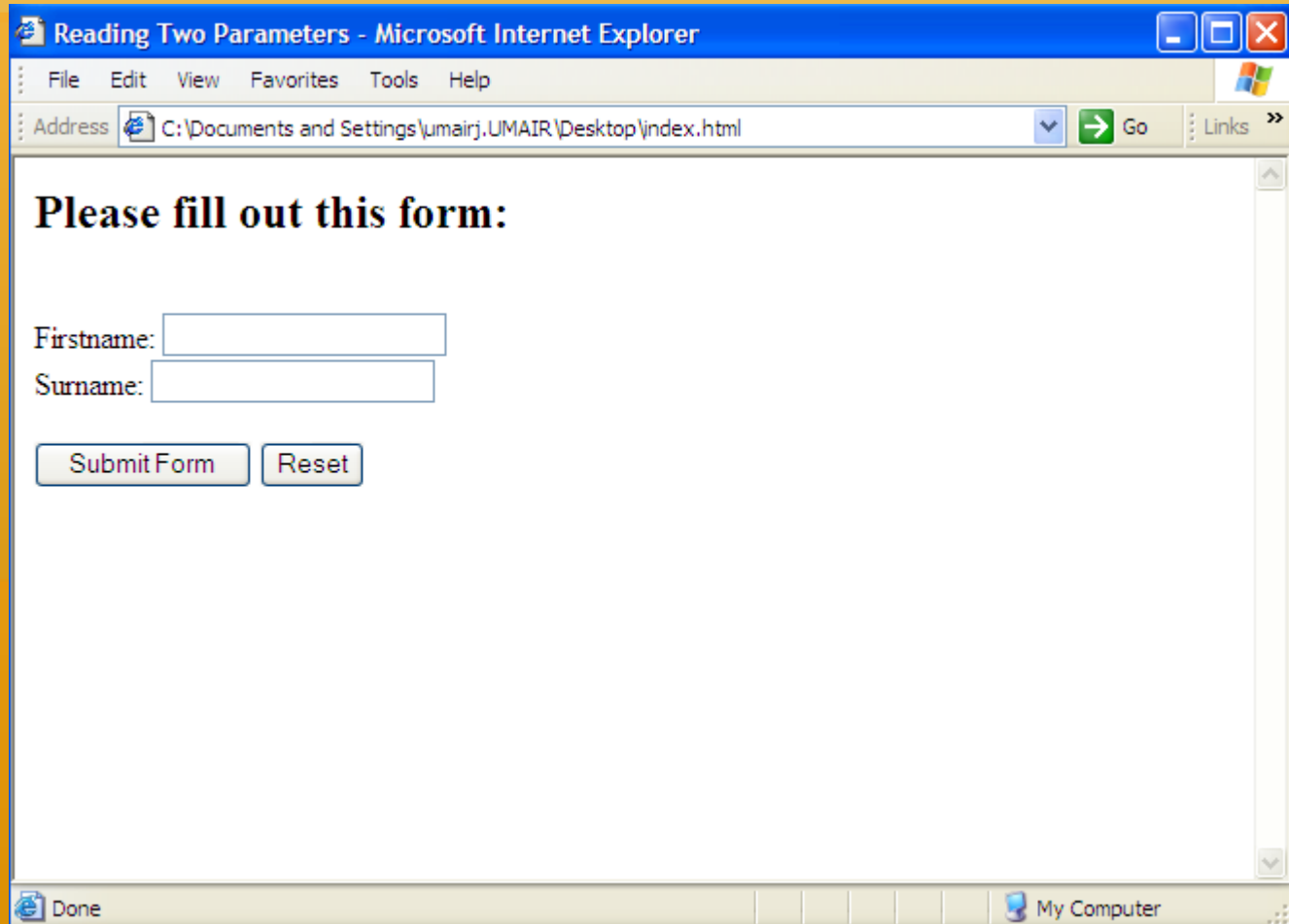


**Example : Reading
two explicit
parameters**

Form Example

- Our first example consists of one HTML page (index.html), and one servlet (FormServlet.java).
- The HTML page contains two form parameters: **firstname**, **surname**
- The Servlet extracts these specific parameters and echoes them back to the browser after appending Hello.

index.html



The screenshot shows a Microsoft Internet Explorer window titled "Reading Two Parameters - Microsoft Internet Explorer". The address bar displays the local file path "C:\Documents and Settings\umairj.UMAIR\Desktop\index.html". The main content area contains the text "Please fill out this form:" followed by two input fields labeled "Firstname:" and "Surname:". Below these fields are two buttons: "Submit Form" and "Reset". The status bar at the bottom shows "Done" and "My Computer".

Reading Two Parameters - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address  C:\Documents and Settings\umairj.UMAIR\Desktop\index.html  Go  Links >>

Please fill out this form:

Firstname:

Surname:

Done  My Computer

Form Example (using GET)

```
<html>
<head>
    <title> Reading Two Parameters </title>
</head>
<body>

    <H2>Please fill out this form:</H2>

    <FORM METHOD="GET"
        ACTION="http://localhost:8080/paramapp/formservlet"
        NAME="myform">

    <BR> Firstname: <INPUT TYPE = "text" NAME="firstname">
    <BR> Surname: <INPUT TYPE = "text" NAME="surname">
    .....
```

Form Example cont. (using GET)

```
<BR>
```

```
<INPUT TYPE="submit" value="Submit Form">
```

```
<INPUT TYPE="reset" value="Reset">
```

```
</FORM>
```

```
</body>
```

```
</html>
```

FormServlet.java (using GET)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        <input TYPE="text" NAME="firstname">

        String fName = req.getParameter("firstname");
        String sName = req.getParameter("surname");

        PrintWriter out = res.getWriter();

        out.println("Hello: " + fName + " " + sName);
        out.close();
    }
}
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<web-app>
```

```
  <servlet>
```

```
    <servlet-name>FormServlet</servlet-name>
```

```
    <servlet-class>MyServlet</servlet-class>
```

```
  </servlet>
```

```
  <servlet-mapping>
```

```
    <servlet-name>FormServlet</servlet-name>
```

```
    <url-pattern>/formservlet</url-pattern>
```

```
  </servlet-mapping>
```

```
</web-app>
```

ACTION="http://localhost:8080/paramapp/formservlet"

Using NetBeans4.1



More on Servlets

Lec 29

Web Design and Development



Looking Back...

- Subclass from `HttpServlet`
- Override `doGet(...)` or `doPost(...)` method
- `HttpServletRequest`
 - `getParameter("paramName")`
- `HttpServletResponse`
 - `get PrintWriter`
 - `send text to client via PrintWriter`

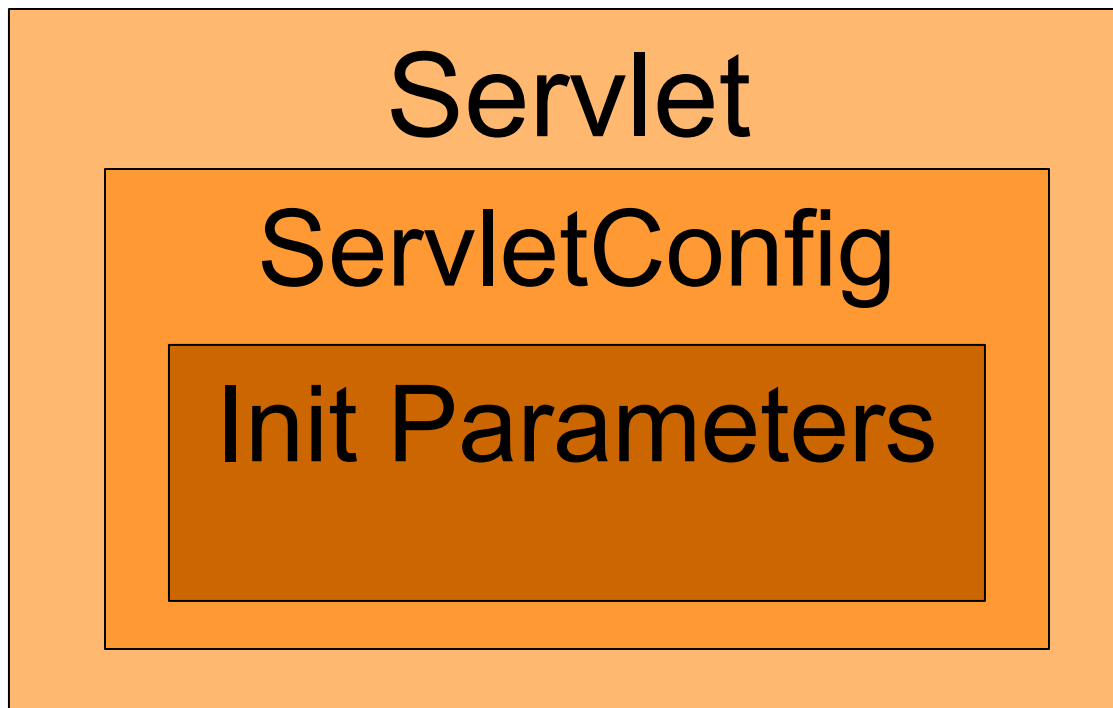
Init. Parameters

- Specifies Initialization Parameters for this servlet
 - Similar to command line parameter for a standard app
- Init params are defined using name value pairs in web.xml

<param-name>

<param-value>

ServletConfig



Reading init parameters

- Override init (life-cycle) methods
- Use `getInitParameter()` method of `ServletConfig` to read init parameters. For example,

```
public void init (ServletConfig config) throws ServletException
{
    String name = config.getInitParameter("paramName");
}
```

Reading init parameters

- For methods other than `init()`
 - Call `getServletConfig()` to obtain the `ServletConfig` object.
 - Use `getInitParameter()` of `ServletConfig` to read initialization parameters.

`String name ;`

`ServletConfig config = getServletConfig();`

`name = config.getInitParameter("paramName");`



Example Code

Reading init parameters



Response Redirection

- We can redirect the response of the servlet to another application resource (another servlet, an HTML page or a JSP)
- The resource (URL) must be available to the calling servlet, in the same servlet context (discussed later)



Response Redirection (cont.)

- Two forms of response redirection
 - Sending a standard redirect
 - Sending a redirect to an error page

Response Redirection (cont.)

- Sending a standard redirect
 - Using `response.sendRedirect("myHtml.html")` method, a new request is generated which redirects the user to the specified URL.
 - If the URL is of another Servlet, that second Servlet will not have access to the original request object.



Response Redirection (cont.)

- Sending a standard redirect
 - To have access to the original request object, you must use the request dispatching technique (discussed later) instead of redirect.

Response Redirection (cont.)

- Sending a redirect to an error page
 - An error code is sent as a parameter
`response.sendError(int, msg)` method.
 - The error page is displayed with the msg passed to method

Response Redirection (cont.)

- Sending a redirect to an error page
 - The error numbers are predefined constants of the response class.

For example:

- SC_NOT_FOUND (404)
- SC_NO_CONTENT (204)
- SC_REQUEST_TIMEOUT (408)



Example Code

Response Redirection



ServletContext

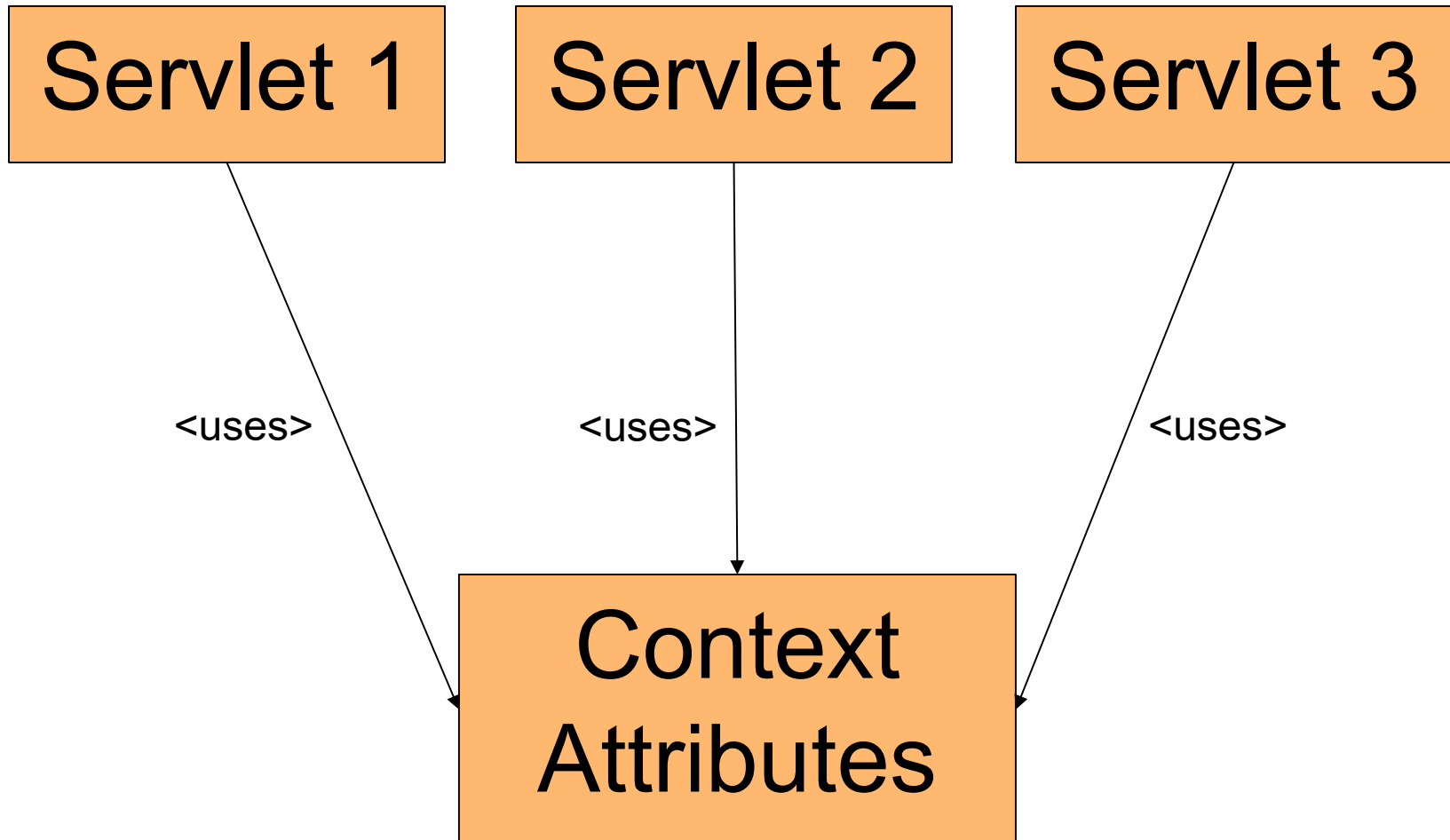
- For sharing resources among servlets in the same web application, we use ServletContext
- Can store web application initialization parameters also called Context Attributes (similar to ServletConfig init parameters)



ServletContext (cont.)

- Can store attributes (exist for the lifetime of application)
- Can get ServletContext using `getServletContext()`. This is a method of Servlet interface.

Servlet Context





Note about ServletContext

- There is a single ServletContext per web application
- Different Sevlets will get the same ServletContext object, when calling `getServletContext()` during different sessions.



Request Dispatcher

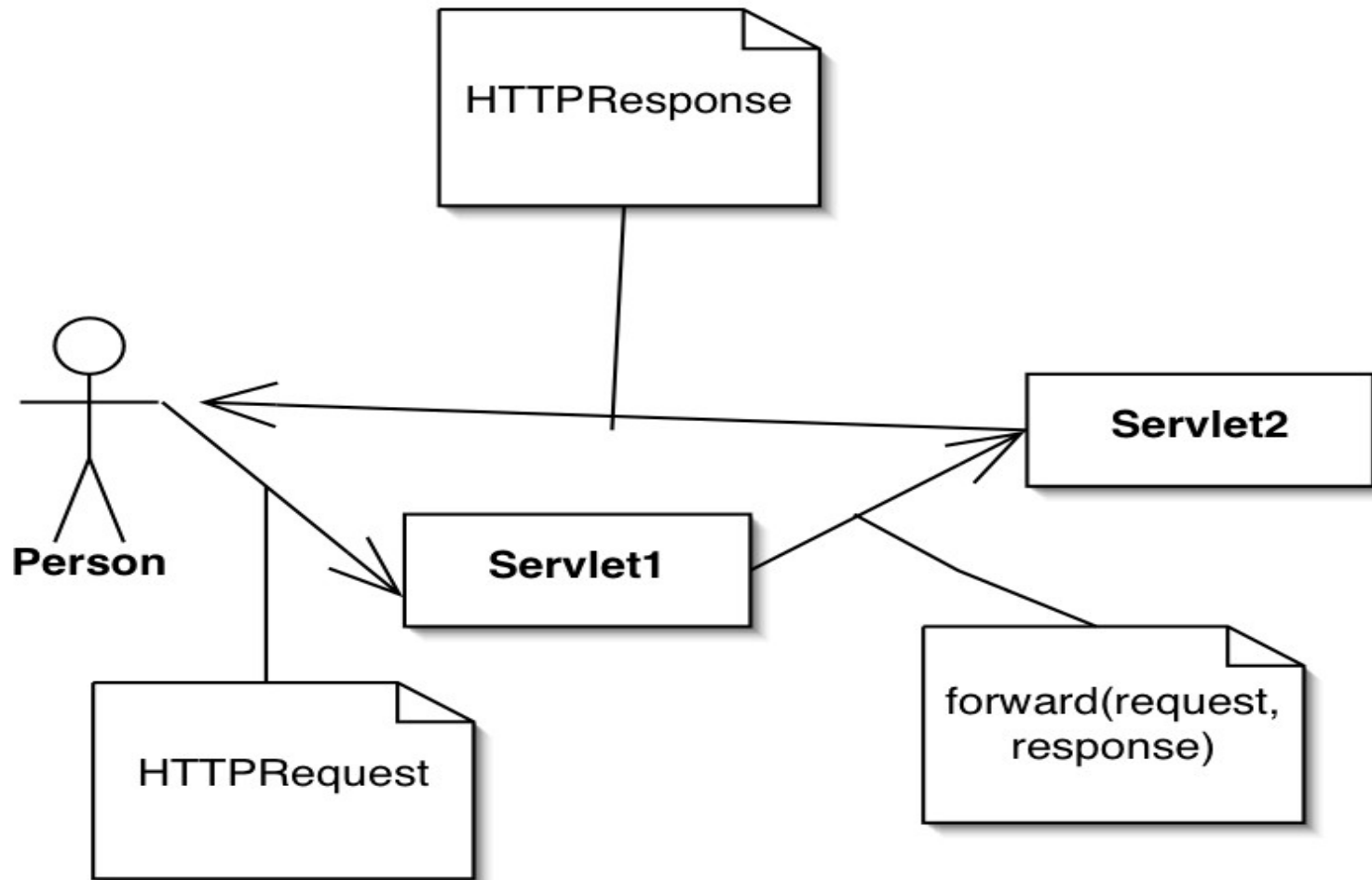
- `getRequestDispatcher(String path)` method of `ServletContext` which returns a `RequestDispatcher` associated with the resource at the given path.



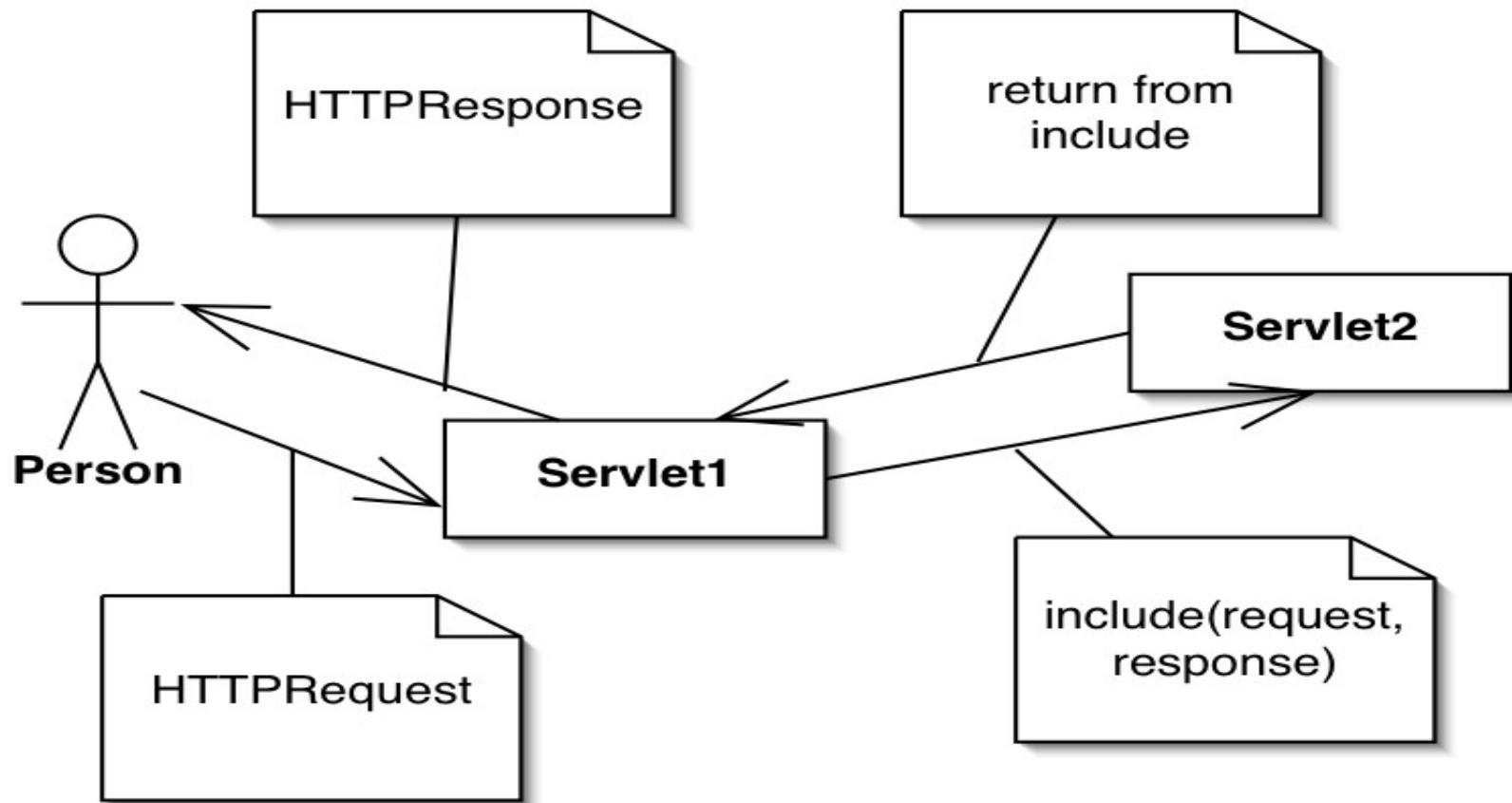
RequestDispatcher: forward

- Allows a Servlet to forward the request to another resource (Servlet, JSP or HTML file) in the same Servlet context.
- Forwarding remains transparent to the client unlike `res.sendRedirect(String location)`.
- Request Object is available to the called resource. In other words, it remains in scope.
- Can also set headers or status codes, but can't add output content.

Forward Diagram



Include Diagram





Servlets 4

Lec 30

Web Design and Development



Looking Back...

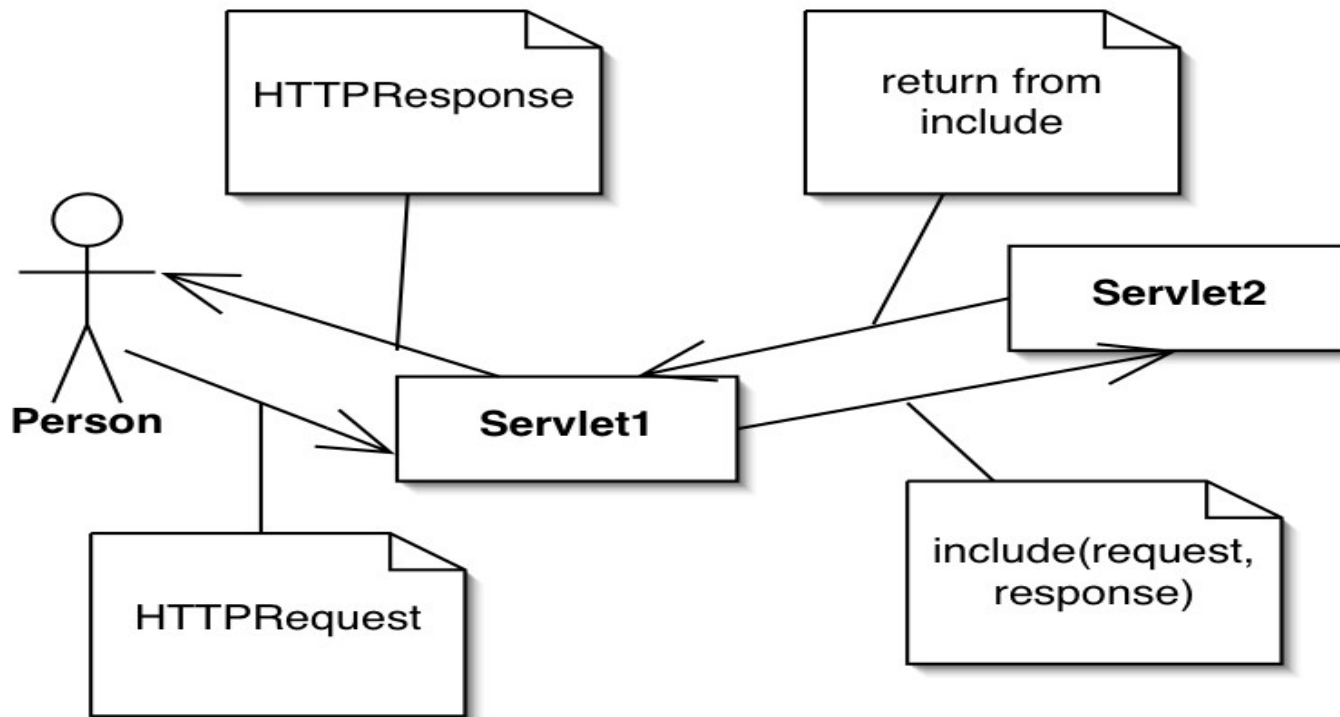
■ Response Redirection

- ☐ Sending a standard redirect
- ☐ Sending a redirect to an error page

■ Request Dispatching

- ☐ Forward
- ☐ Include

Include Diagram

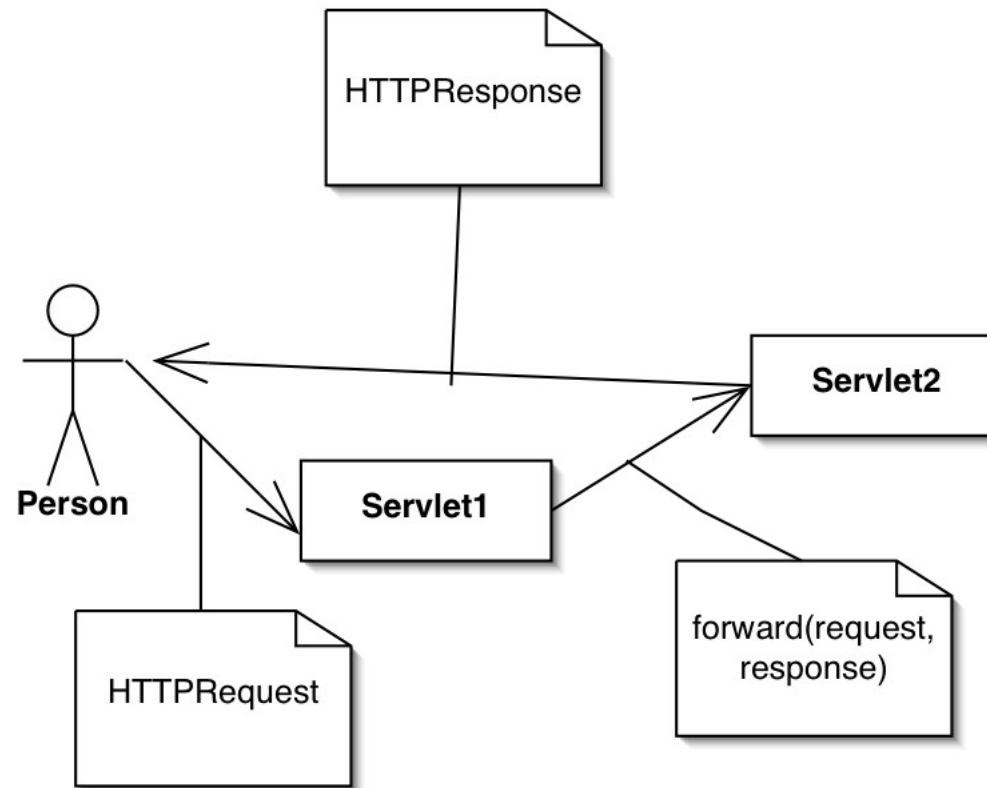




Example Code

Request Dispatching: include

Forward Diagram





Example Code

Request Dispatching: forward



HttpServletRequest methods

More HttpServletRequest methods

- **setAttribute (String, Object) /
getAttribute(String)**
 - Stores an attribute in this request as **key-value** pair form
 - Attributes are set/reset between requests
 - Often used in conjunction with RequestDispatcher

More HttpServletRequest methods cont.

■ **getMethod()**

- Returns the name of HTTP method (get & post etc)

■ **getRequestURL()**

- Returns the part of this request's URL
- Query string is not included

■ **getProtocol()**

- Returns the name & version of the protocol used

More HttpServletRequest methods cont.

■ **getHeaderNames()**

- Returns an **enumeration** of all the header names this request contains

■ **getHeader(String headername)**

- Returns the value of the specified request header as String
- Returns **null**, if no header with the specified name



HTTPServletResponse methods

More HttpServletResponse methods

- **setContentType(String)**
 - Servlets almost always use this header
 - MIME types
- **setContentLength(int)**
 - Sets the Content-Length header.
- **addCookie(Cookie)**
 - Adds a value to the Set-Cookie header.
- **sendRedirect (String)**
 - Redirect client to the specified URL
 - Can accept relatives URL

A decorative graphic on the left side of the slide consisting of a grid of orange squares of varying shades, arranged in a pattern that suggests a staircase or a series of steps.

Session Tracking



Session Tracking

- HTTP is a **stateless** protocol
 - Every request is considered independent of every other request
- Many web applications need to maintain a **conversational state** with the client
 - A shopping cart is a classic example

Continuity Problem (User's Point of View)



Server State



Page 1



Added book to
cart

Page 2



Added book to
cart

Page 3



CC# = XXX
Billing address

Page 4



Order submitted
& logged

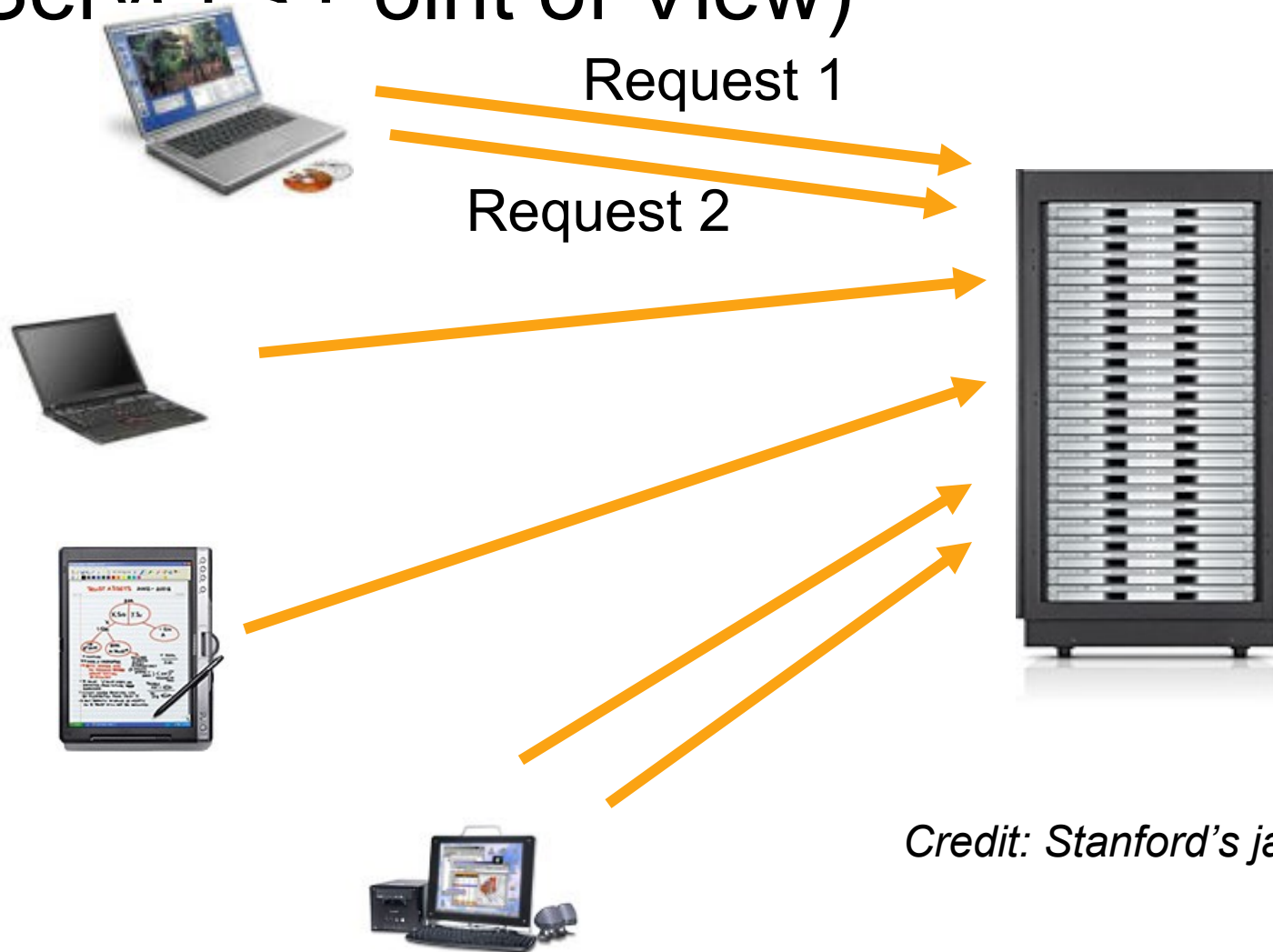
Credit: Stanford's java course



The illusion of Continuity

- User thinks that choices made on page 1 are remembered on page 3
- However
 - HTTP is Stateless
 - Requests from same user do not necessarily come in adjacent requests

Continuity Problem (Server's Point of View)



Credit: Stanford's java course

Session Tracking

Lec 31

Revisiting Session Tracking

- HTTP is a stateless protocol
 - Every request is considered independent of every other request
- Many web applications need to maintain a **conversational state** with the client
 - A shopping cart is a classic example

Store State Somewhere

- Server Side?
 - Makes Server Really Complicated
 - State per client!
- Client Side?

“Post-Notes”

- Server puts little notes on the client side
- When client submits the next form, it also (unknowingly) submits these little notes
- Server reads the notes, remembers who the client is

Three Typical Solutions

- Cookies
- URL Rewriting
- Hidden Fields

Handling Cookies

Potential of Cookies

- Idea

- ❑ Web server sends a simple **name-value** pair to client (web browser etc.)
- ❑ Saved by the client
- ❑ Later, Client returns same name and value when it connects to same site (or same domain, depending on cookie settings)

Potential of Cookies

■ Typical Uses of Cookies

- ❑ Identifying a user during an e-commerce session
 - Servlets have a higher-level API for this task
- ❑ Avoiding username and password
- ❑ Customizing a site
- ❑ Focused advertising

Sending Cookies to Browser

- **Create a Cookie object**

```
Cookie c = new Cookie("name", "value");
```

- **Set the Maximum age etc**

- Cookie persists on disk

```
c.setMaxAge(seconds);  
// Set other attributes.
```

- **Place the Cookie into HTTP response**

- If you forget this step, no cookie will be sent to the browser

```
response.addCookie(c);
```


Reading Cookies from Browser

- To read incoming cookies, get them from `request` object

```
Cookie[] cookies = request.getCookies();
```

- Once you have an array of cookies, you can iterate over it
 - Use `getName` and `getValue` to retrieve cookie name & value respectively

```
for(int i=0; i<cookies.length; i++) {  
    Cookie c = cookies[i];  
    if (c.getName().equals("someName")) {  
        // doSomethingWith cookie  
        break;  
    }  
}
```

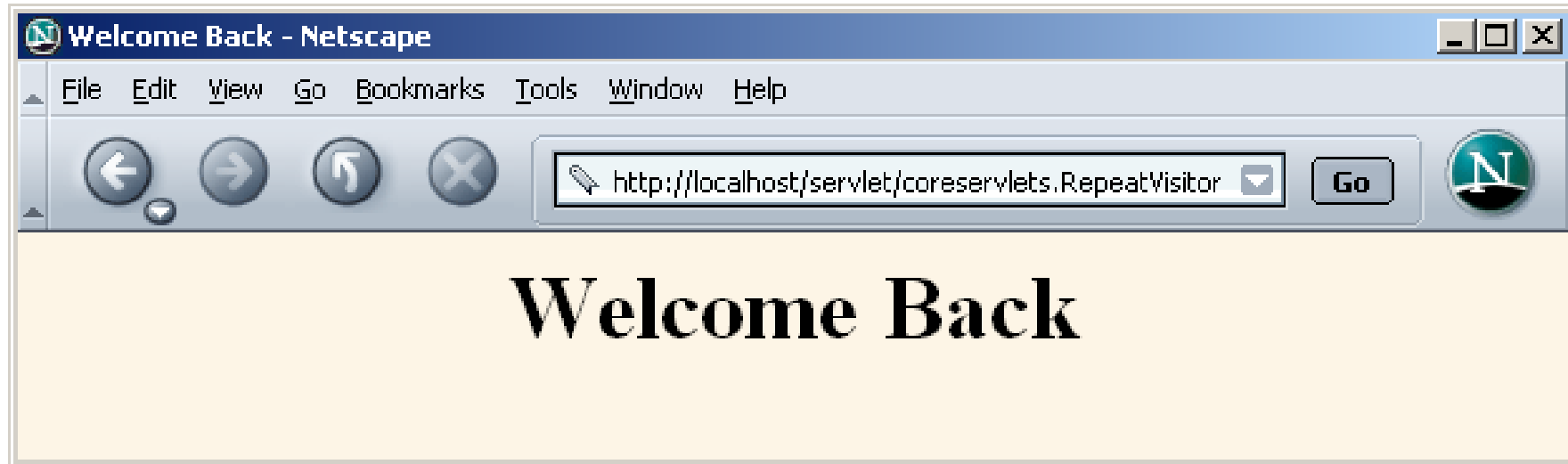
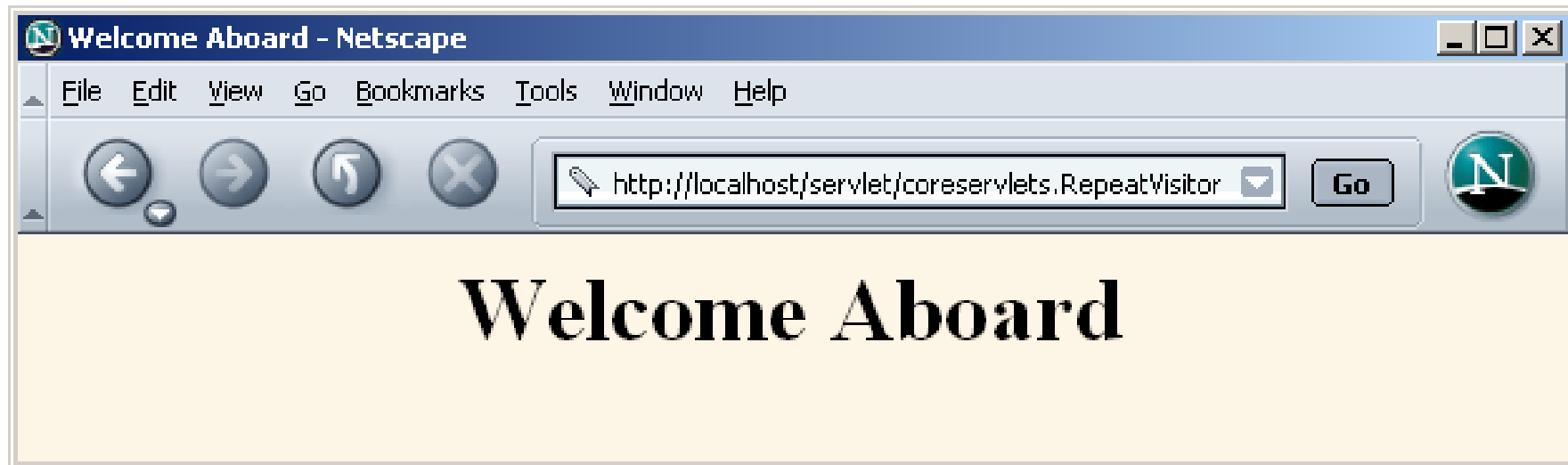
Example 1: RepeatVisitor

- This servlet checks for a unique cookie, named “repeat”.
 - If the cookie is present, servlet says “Welcome Back”
 - Otherwise, servlet says “Welcome Aboard”.

Example Code

Repeat Visitor

Using Cookies to Detect First-Time Visitors (Results)



HTTP Cookies



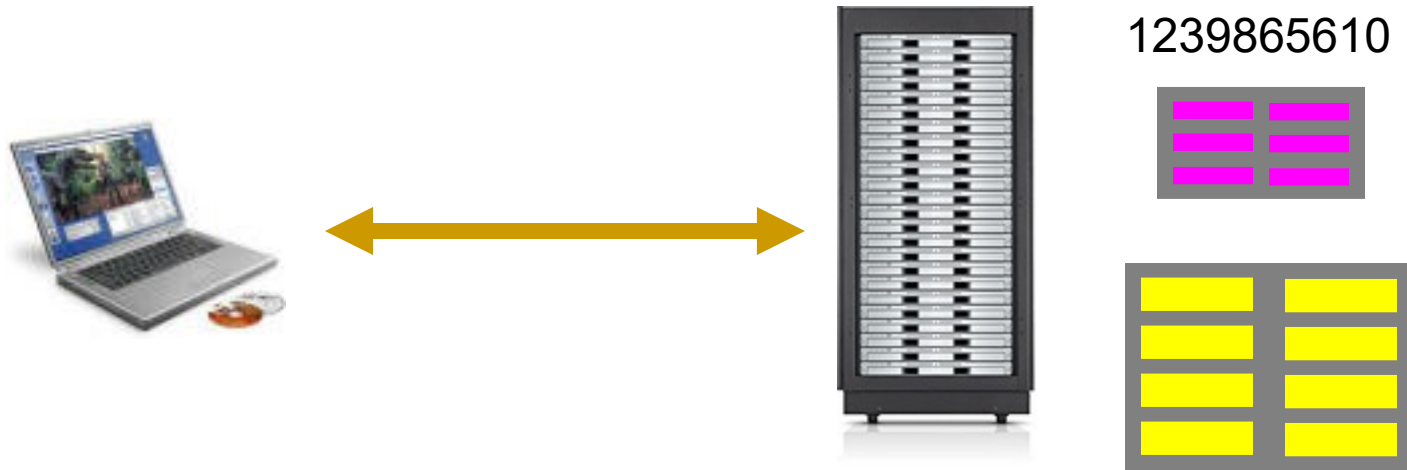
```
String sID = makeUniqueString();  
Hashtable sessionInfo = new Hashtable();  
Hashtable globalTable = findTableStoringSessions();  
globalTable.put(sID, sessionInfo);  
Cookie sessionCookie = new Cookie("JSESSIONID", sID);  
response.addCookie(sessionCookie);
```

HTTP Cookies



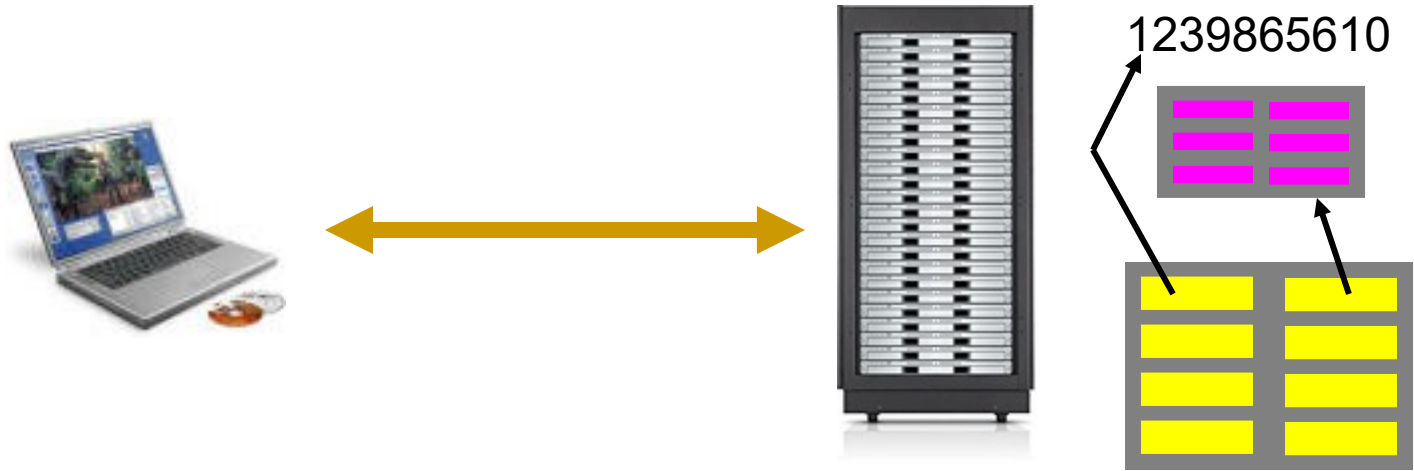
```
String sID = makeUniqueString();  
Hashtable sessionInfo = new Hashtable();  
Hashtable globalTable = findTableStoringSessions();  
globalTable.put(sID, sessionInfo);  
Cookie sessionCookie = new Cookie("JSESSIONID", sID);  
response.addCookie(sessionCookie);
```

HTTP Cookies



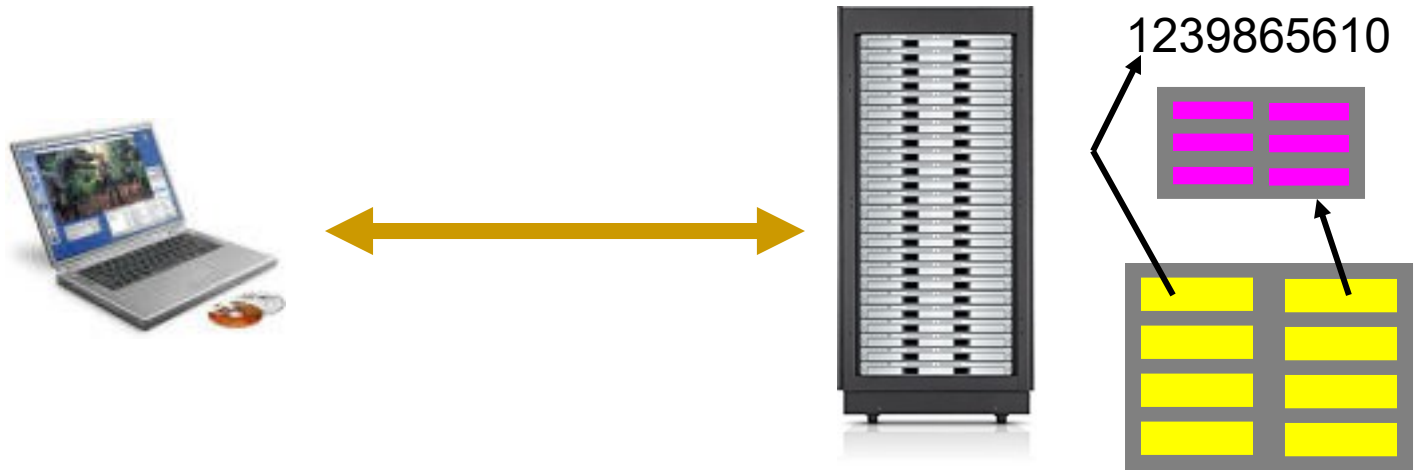
```
String sID = makeUniqueString();  
Hashtable sessionInfo = new Hashtable();  
Hashtable globalTable = findTableStoringSessions();  
globalTable.put(sID, sessionInfo);  
Cookie sessionCookie = new Cookie("JSESSIONID", sID);  
response.addCookie(sessionCookie);
```

HTTP Cookies



```
String sID = makeUniqueString();  
Hashtable sessionInfo = new Hashtable();  
Hashtable globalTable = findTableStoringSessions();  
globalTable.put(sID, sessionInfo);  
Cookie sessionCookie = new Cookie("JSESSIONID", sID);  
response.addCookie(sessionCookie);
```

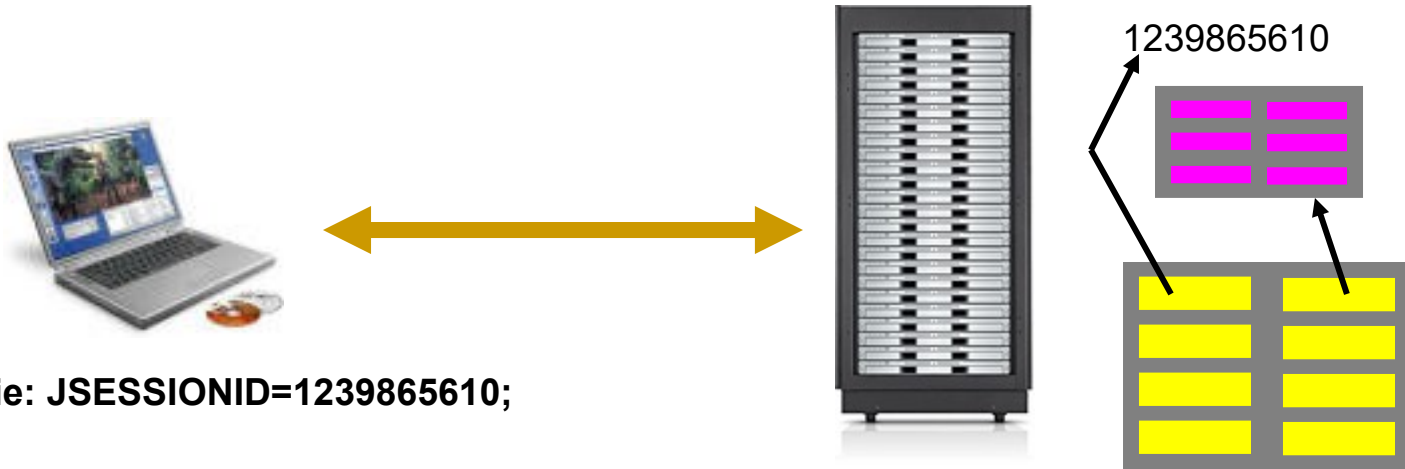

HTTP Cookies



JSESSIONID → 1239865610

```
String sID = makeUniqueString();  
Hashtable sessionInfo = new Hashtable();  
Hashtable globalTable = findTableStoringSessions();  
globalTable.put(sID, sessionInfo);  
Cookie sessionCookie = new Cookie("JSESSIONID", sID);  
response.addCookie(sessionCookie);
```

HTTP Cookies

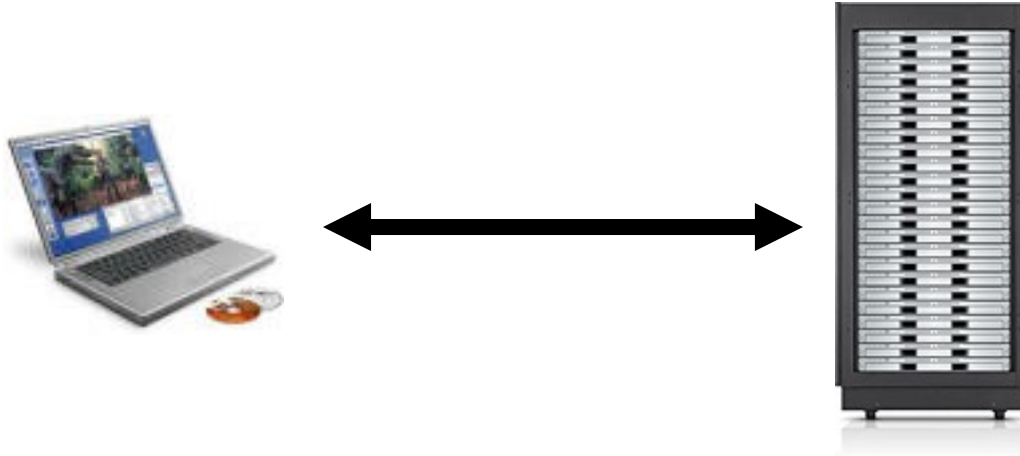


Set-Cookie: JSESSIONID=1239865610;

```
String sID = makeUniqueString();  
Hashtable sessionInfo = new Hashtable();  
Hashtable globalTable = findTableStoringSessions();  
globalTable.put(sID, sessionInfo);  
Cookie sessionCookie = new Cookie("JSESSIONID", sID);  
response.addCookie(sessionCookie);
```

Credit: cs193i at Stanford

HTTP Cookies



Cookie: JSESSIONID=1239865610;

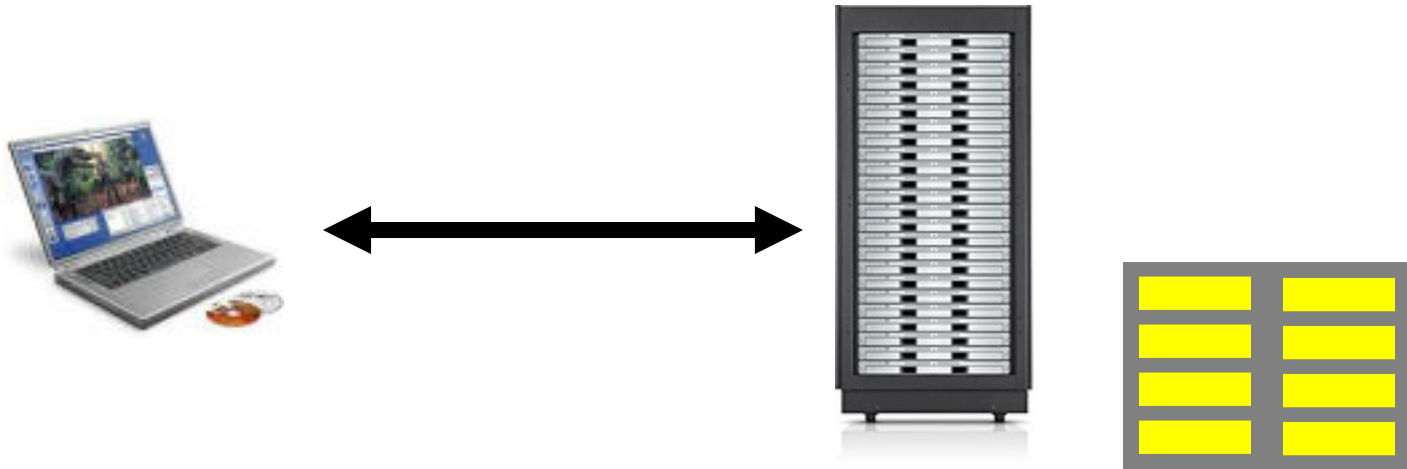
// On request

```
String sID = getCookieValue("JSESSIONID");  
Hashtable globalTable = findTableStoringSessions();  
Hashtable sInfo = (Hashtable) globalTable.get(sID);
```

// sInfo contains the data related to user

Credit: cs193i at Stanford

HTTP Cookies



Cookie: JSESSIONID=1239865610;

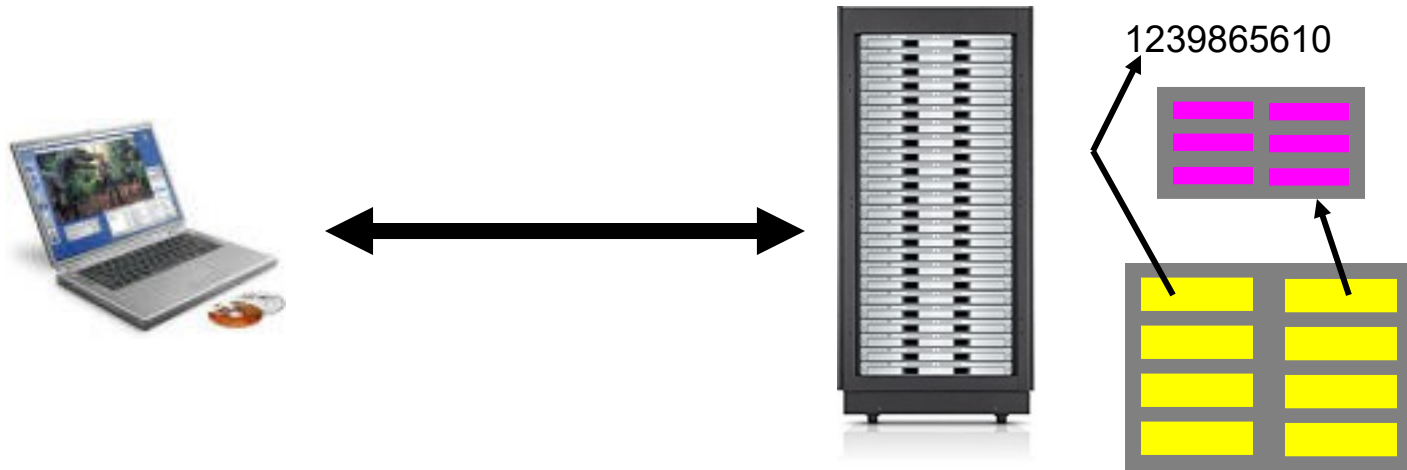
// On request

```
String sID = getCookieValue("JSESSIONID");  
Hashtable globalTable = findTableStoringSessions();  
Hashtable sInfo = (Hashtable) globalTable.get(sID);
```

// sInfo contains the data related to user

Credit: cs193i at Stanford

HTTP Cookies



Cookie: JSESSIONID=1239865610;

// On request

```
String sID = getCookieValue("JSESSIONID");  
Hashtable globalTable = findTableStoringSessions();  
Hashtable sInfo = (Hashtable) globalTable.get(sID);
```

// sInfo contains the data related to user

Credit: cs193i at Stanford

Example : Online Book Store

using cookies

netBeans project -CookieSessionEx

Session Tracking - 2

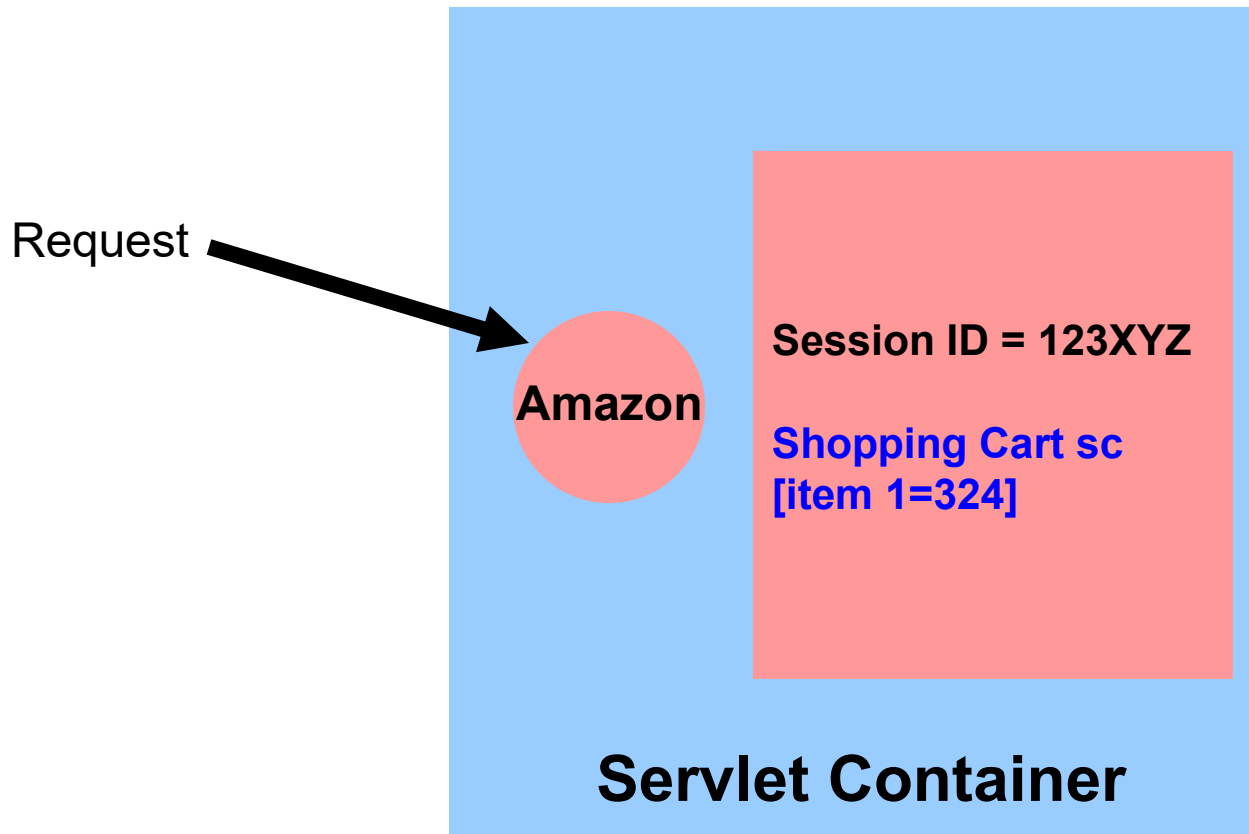


Lec 32

Last Lecture Review

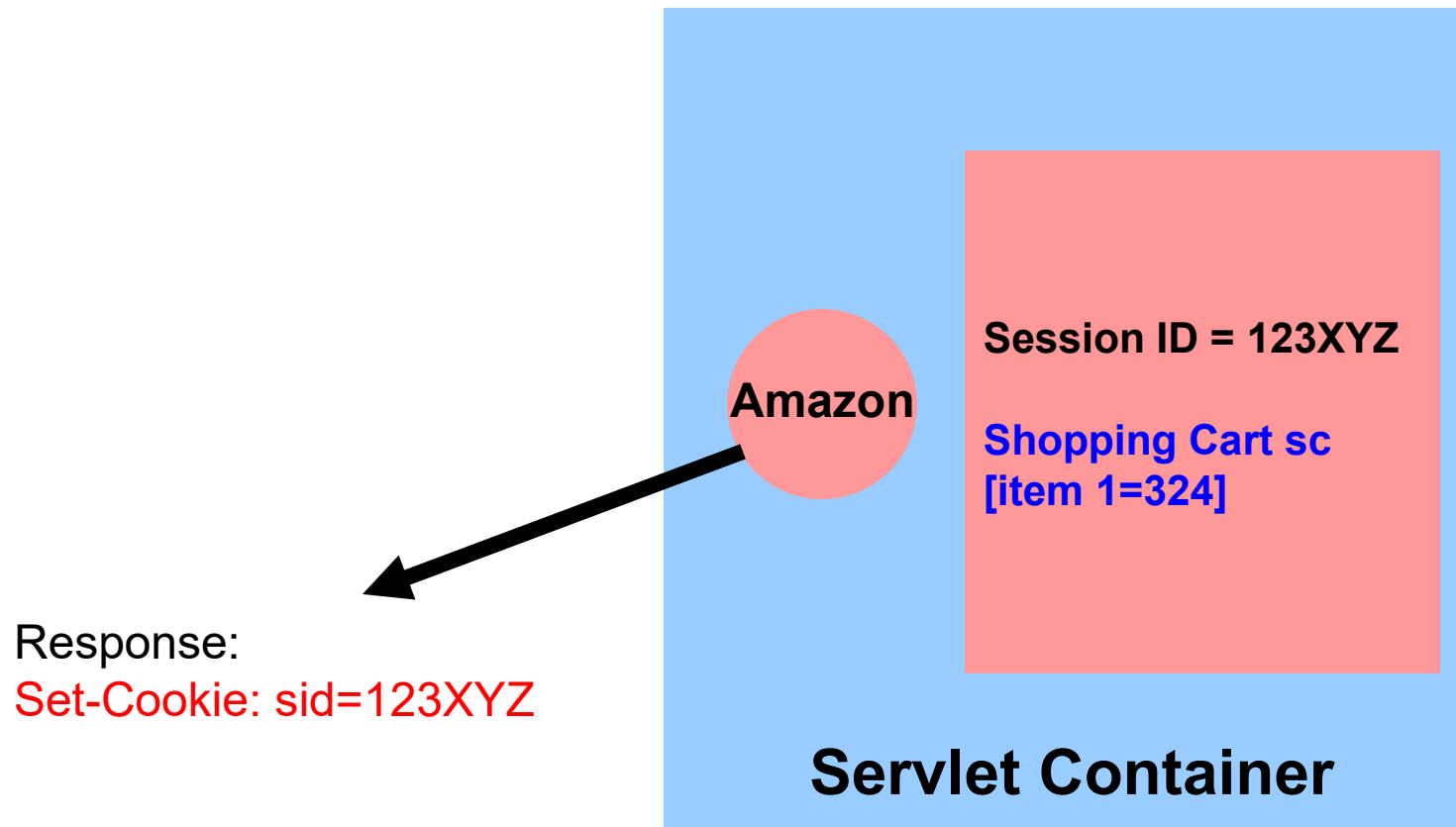
- Session Tracking – why?
- Need to store state – typical solutions
 - Cookies – already learned
 - URL Rewriting
 - Hidden Form Fields

Session Tracking



Credit: cs193i at Stanford

Session Tracking



Credit: cs193i at Stanford

Session Tracking

Request:

Set-Cookie: sid=123XYZ



Amazon

Session ID = 123XYZ

Shopping Cart sc
[item 1=324]

Servlet Container

Session Tracking

Request:

Set-Cookie: sid=123XYZ



Amazon

Session ID = 123XYZ

Shopping Cart sc
[item 1=324
item 2=115]

Servlet Container

URL Rewriting



URL Rewriting

- We can pass **extra information** to client by rewriting URLs. (appending info with URL)
- The extra information can be in the form of
 - Extra path information,
 - Added parameters, or
 - Some custom, server-specific URL change
- Due to limited space available in rewriting a URL, the extra information is usually limited to a **unique session ID**

URL Rewriting: Examples

- For example, the following URLs have been rewritten to pass the session id **123**
 - Original
<http://server:port/servlet/rewrite>
 - Extra path information
<http://server:port/servlet/rewrite/123>
 - Added parameter
<http://server:port/servlet/rewrite?id=123>
 - Custom change
[http://server:port/servlet/rewrite;\\$id\\$123](http://server:port/servlet/rewrite;id123)

URL Rewriting: Disadvantages

- ❑ What if the user bookmarks the page?
- ❑ Every URL on a page which needs the session information must be rewritten each time page is served
 - Computationally expensive
 - Can increase communication overhead
- ❑ State stored in URLs is not persistent
- ❑ Limits the client's interaction with the server to HTTP GET request

Hidden Form Fields



Hidden Form Fields

- `<input type="hidden" name="sessionid" value="123">`

```
<form method="POST" action="/exec/obidos/handle-buy-box=/  
ref=bp_add/103-4591077-2490203">
```

```
<input type="hidden" name="colid" value="">  
<input type="hidden" name="template-name" value="">  
<input type="hidden" name="store-name" value="gateway">  
<input type="hidden" name="maw" value="1">  
<input type="hidden" name="coliid" value="">
```

```
<input type="hidden" name="dropdown-selection"  
value="default-address">
```

```
<table border="0" width="100%" cellpadding="0"  
cellpadding="6">
```

Java's Solution for Session Tracking



HttpSession API

Using HttpSession

1. To get the user's session object

- Call `getSession( )` method of `HttpServletRequest` class
- pass `false` to the `getSession()` method
`HttpSession ses = request.getSession(false);`
- If no current session exists:
 - ▣ You will get a `null object`

Using HttpSession cont.

1. To get the user's session object (cont.)

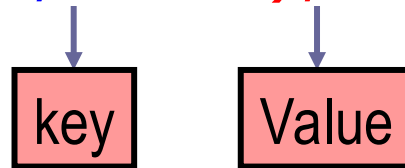
- If **true** is passed to the **getSession()** method then
- If **user** already has a **session**
 - the **existing session** is returned
- For example:
HttpSession ses = request.getSession(true);
- If **no session** exists
 - a **new one** is created and returned

Using HttpSession cont.

2. Storing information in a session

- **Session objects** works like a **HashMap**
 - **HashMap** is able to **store any type of java object**
- **You can therefore store any number of keys and their values**
- **For example**

```
ses.setAttribute("id", "123");
```



Using HttpSession cont.

3. Looking up information associated with a session

```
String sID =  
    (String) ses.getAttribute("id");
```



returns an **Object** type, so you will need to perform a **type cast**

Using HttpSession cont.

4. Terminating session

- Automatic
 - After the amount of time session gets terminated automatically(`getMaxInactiveInterval( )`)
- Manual

`ses.invalidate( ) ;`

Example Code



Showing Session Information

Encoding URLs Sent to Client

- HttpServletResponse provides two methods to perform encoding
 1. String encodeURL(String URL)
 2. String encodeRedirectURL(String URL)
- If Cookies disabled
 - Both methods encodes (rewrites) the specified URL to include the session ID and returns the new URL
- If Cookies enabled
 - Returns the URL unchanged

Encoding URLs Sent to Client

cont.

1. String encodeURL(String URL)

- For example

```
String URL = "/servlet/sessiontracker";
```

```
String eURL = response.encodeURL(URL);
```

```
out. println("<A HREF=\"\" + eURL + \"\">...</A>");
```

Encoding URLs Sent to Client cont.

2. String encodeRedirectURL(String URL)

- For example

```
String URL = "/servlet/sessiontracker";
```

```
String eURL = response.encodeRedirectURL(URL);  
response.sendRedirect(eURL);
```

Example Code



Online Book Store

AddressBook using Servlets and Intro to JSP



Lec - 33

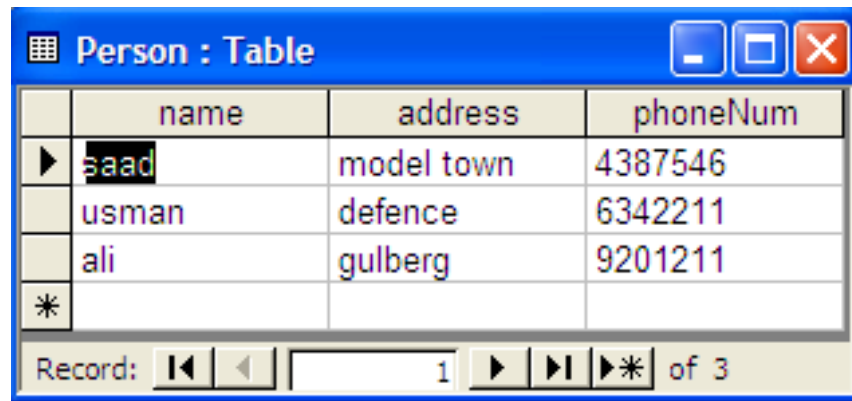
Servlets Example



A part of
Address Book

Design Process

- Step 1 – Create a database (AddressBook)
 - Make a table named Person



A screenshot of a database application window titled "Person : Table". The window contains a table with three columns: "name", "address", and "phoneNum". The table has three data rows and a footer row marked with an asterisk (*). The first row contains "saad", "model town", and "4387546". The second row contains "usman", "defence", and "6342211". The third row contains "ali", "gulberg", and "9201211". The footer row is empty. Below the table, there is a record navigation bar showing "Record: 1 of 3" with navigation buttons.

	name	address	phoneNum
▶	saad	model town	4387546
	usman	defence	6342211
	ali	gulberg	9201211
*			

Record: 1 of 3

Design Process

- Step 2 – Make a PersonInfo class
 - Three attributes: name, address, ph. No.
 - Has Parameterized constructor
 - Override toString() method

- Step 3 – Make a DAO class which contains
 - an establishConnection() method
 - a searchPerson(string name) method that returns PersonInfo object

Design Process

- Step 4 – Write SearchPerson servlet
 - Will take input for name to search in address book
 - Submits the request to ShowPerson servlet

- Step 5 – Write ShowPerson servlet
 - Recieves request from SearchPerson servlet
 - Instantiate objects of PersonInfo and DAO class
 - Call searchPerson method
 - Show results

package

- What is package?
- Examples:
 - java.lang
 - Javax.swing
- How to use package
 - import java.lang.*;
 - import javax.swing.*;

JSP Example



A part of
Address Book

JavaServer Pages



Overview

Agenda

- Introducing JavaServer Pages™ (JSP™)
- JSP scripting elements
- The JSP page Directive
- Including Files in JSP Documents
- Using JavaBeans™ components with JSP
- Creating custom JSP tag libraries
- Integrating servlets and JSP with the MVC architecture

The Need for JSP

- With servlets, it is easy to
 - Read form data
 - Read HTTP request headers
 - Set HTTP status codes and response headers
 - Use cookies and session tracking
 - Share data among servlets
 - Remember data between requests
 - Get fun, high-paying jobs
- But, it sure is a pain to
 - Use those println statements to generate HTML
 - Maintain that HTML

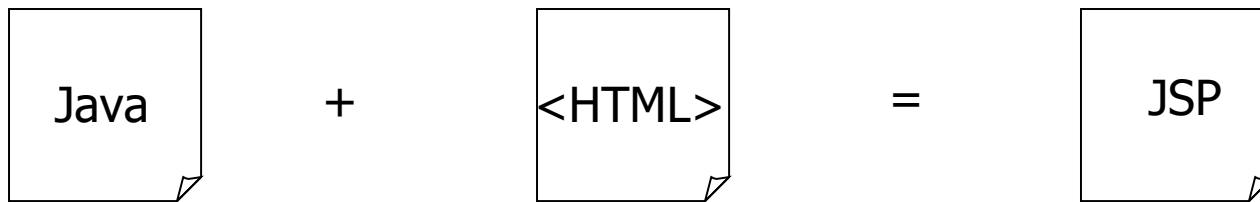
The JSP Framework

□ Idea:

- Use regular HTML for most of page
- Mark servlet code with special tags
- Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)

What are Java Server Pages?

- The Java Server Pages technology combine Java code and HTML tags in the same document to produce a JSP file.

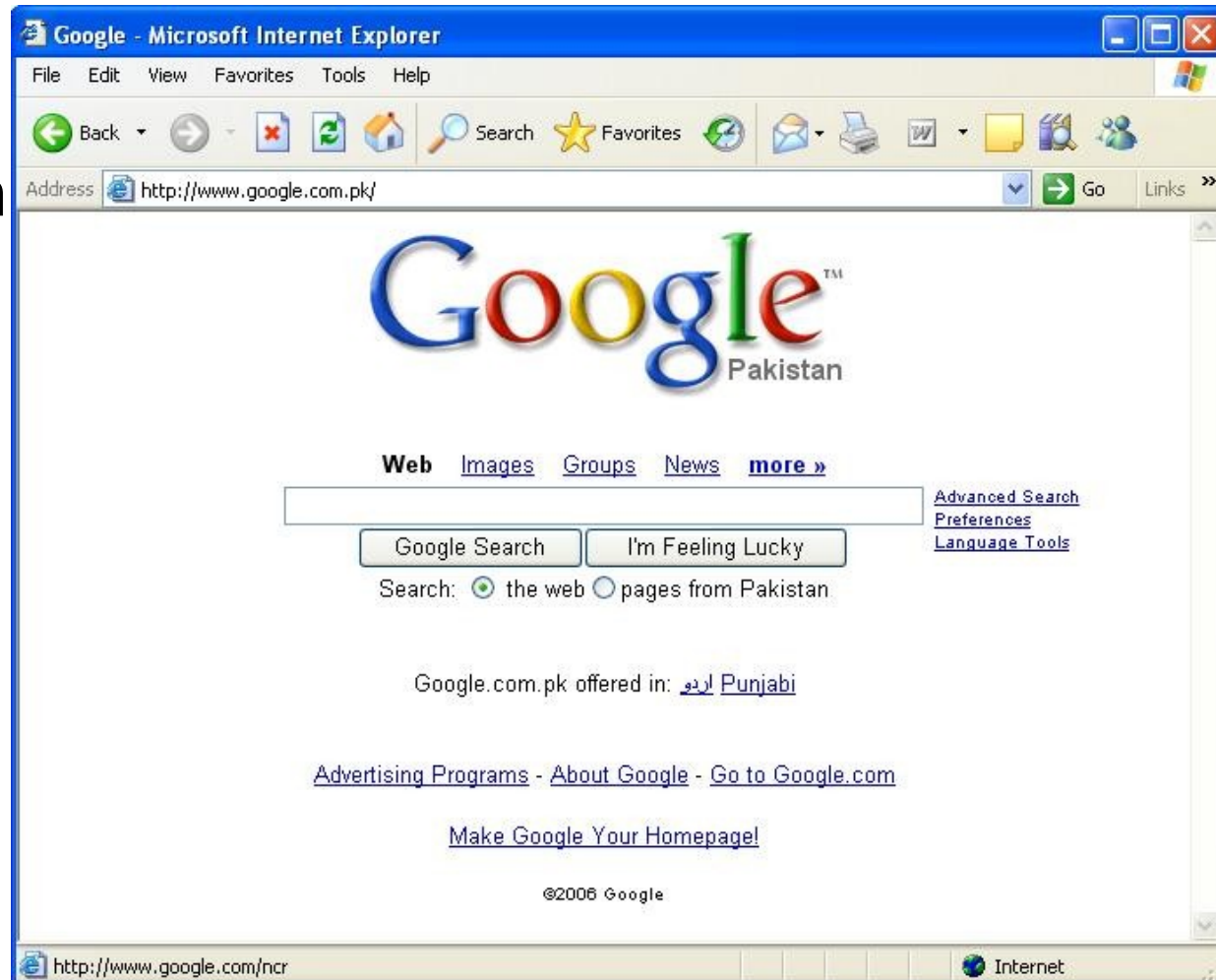


Advantages of JSP Over Competing Technologies

- Versus ASP or ColdFusion
 - Better language for dynamic part
 - Portable to multiple servers and operating systems
- Versus PHP
 - Better language for dynamic part
 - Better tool support
- Versus WebMacro or Velocity
 - Standard
- Versus pure servlets
 - More convenient to create HTML
 - Can use standard tools (e.g., UltraDev)
 - Divide and conquer
 - JSP developers still need to know servlet programming

JSP/Servlets in the Real World

- Google:
the world's
biggest search
portal



JSP/Servlets in the Real World

- ofoto.com:
print and
manage
digital and
conventional
photos.



JSP/Servlets in the Real World

First USA Bank:

largest credit card issuer in the world;
most on-line banking customers



JSP/Servlets in the Real World

Delta Airlines:
entire Web site,
including real-time
schedule info

The screenshot shows the Delta Airlines website as it appeared in the year 2000, viewed through a Netscape browser. The browser's title bar reads "Delta Air Lines - Welcome to Delta - Netscape". The address bar shows the URL "http://www.delta.com/home/index.jsp". The website features a blue header with the Delta logo and a navigation menu with links to HOME, TRAVEL, SKYMILES®, PROGRAMS & SERVICES, INSIDE DELTA, and CUSTOMER CARE. The main content area is divided into several sections: a SkyMiles login form on the left, a "ROUND-TRIP RESERVATIONS" section in the center, and a "DELTA FOR YOU" section on the right. The "ROUND-TRIP RESERVATIONS" section includes fields for "Leaving from", "Going to", departure and return dates and times, and "Passengers and preferred cabin". The "DELTA FOR YOU" section displays the date "Wednesday, September 6, 2000", "Specials" such as "Up to 10% off", and "News" items. A "GO" button is present in the reservation section. The browser's status bar at the bottom shows the same URL.

Delta Air Lines - Welcome to Delta - Netscape

File Edit View Go Communicator Help

Bookmarks Netsite: http://www.delta.com/home/index.jsp

Delta
delta.com

HOME TRAVEL SKYMILES® PROGRAMS & SERVICES INSIDE DELTA CUSTOMER CARE

SkyMiles log in
SkyMiles number and PIN

Start in:
Home
GO

[SkyMiles Enrollment & PIN](#)

Links
[Reservations](#)
[View Itineraries](#)
[Upgrades](#)
[Flight Schedules](#)
[Special Offers](#)
[Investor Relations](#)
[E-mail Programs](#)

ROUND-TRIP RESERVATIONS
[One-way & multi-city reservations](#) | [Calendar](#) | [City codes](#)
Purchase online and earn up to 1,000 [Bonus SkyMiles!](#)
Leaving from Select departure date and time
Sep 06 10 AM
Going to Select return date and time
Sep 06 10 AM
Passengers and preferred cabin
1 Coach (Restricted)
GO

DELTA FOR YOU
Wednesday, September 6, 2000
Specials
• Up to **10% off** - [Offer Ends 9/8/00](#)
News
• Introducing the [767-400](#), our newest aircraft
• Celebrate SkyTeam's take-off with our [Million Miles Sweepstakes](#)
• More breaking [news](#)

 Stay connected with flight info on your Palm VII™ or cell phone. [click here](#)

ARRIVAL / DEPARTURE INFO
Get instant arrival and departure information.
Flight number Yesterday Today Tomorrow
GO

http://www.delta.com/home/index.jsp

JSP/Servlets in the Real World

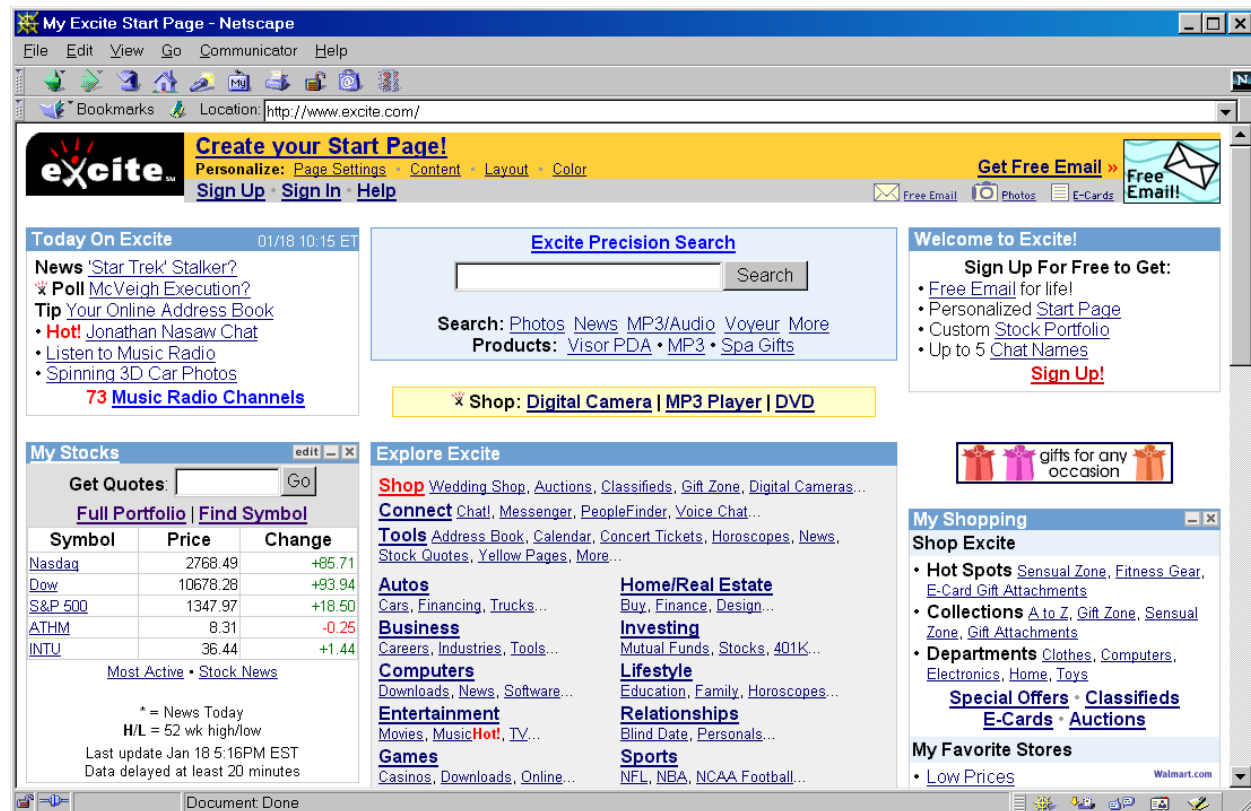
American Century Investments:

more than 70 mutual
funds, \$90 billion under
management, two million
investors



JSP/Servlets in the Real World

- Excite: one of the top five Internet portals; one of the ten busiest sites on the Web



JSP – The Page becomes a Class

- Page Compiler
- Chunks
- Page compiled into a Servlet
- First invocation is slow – has to compile
- Subsequent invocations more rapid



JavaServer Pages

Lec - 34



What is JSP page?

- A **text based document** capable of returning both static & dynamic content to a client browser
- Static content & dynamic content can be intermixed
- Static content
 - HTML, XML, Text
- Dynamic content
 - Java code
 - Displaying properties of JavaBeans
 - Invoking business logic defined in Custom tags



Web-Based Enterprise Applications in Java

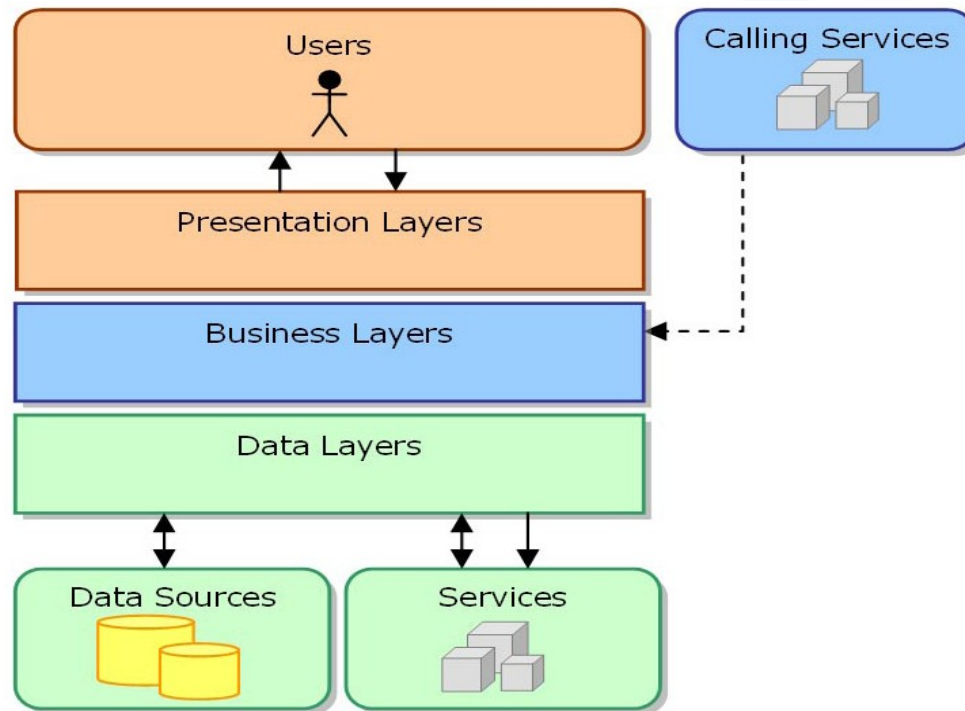
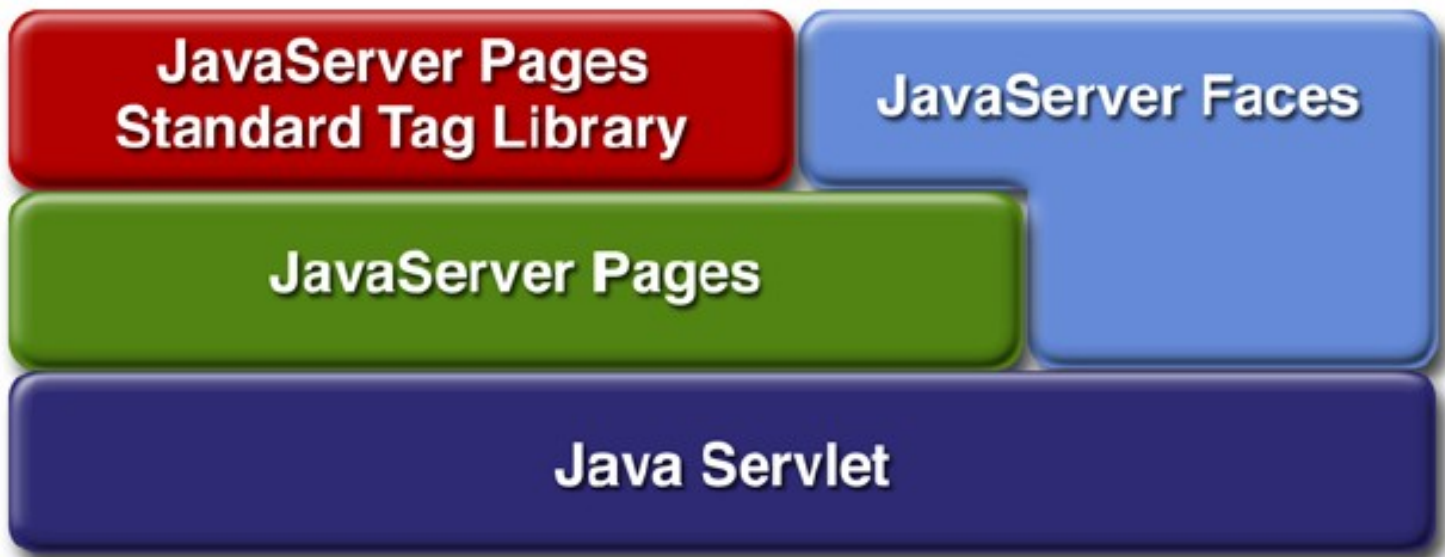


Figure shows a simplified view of one application and its layers.

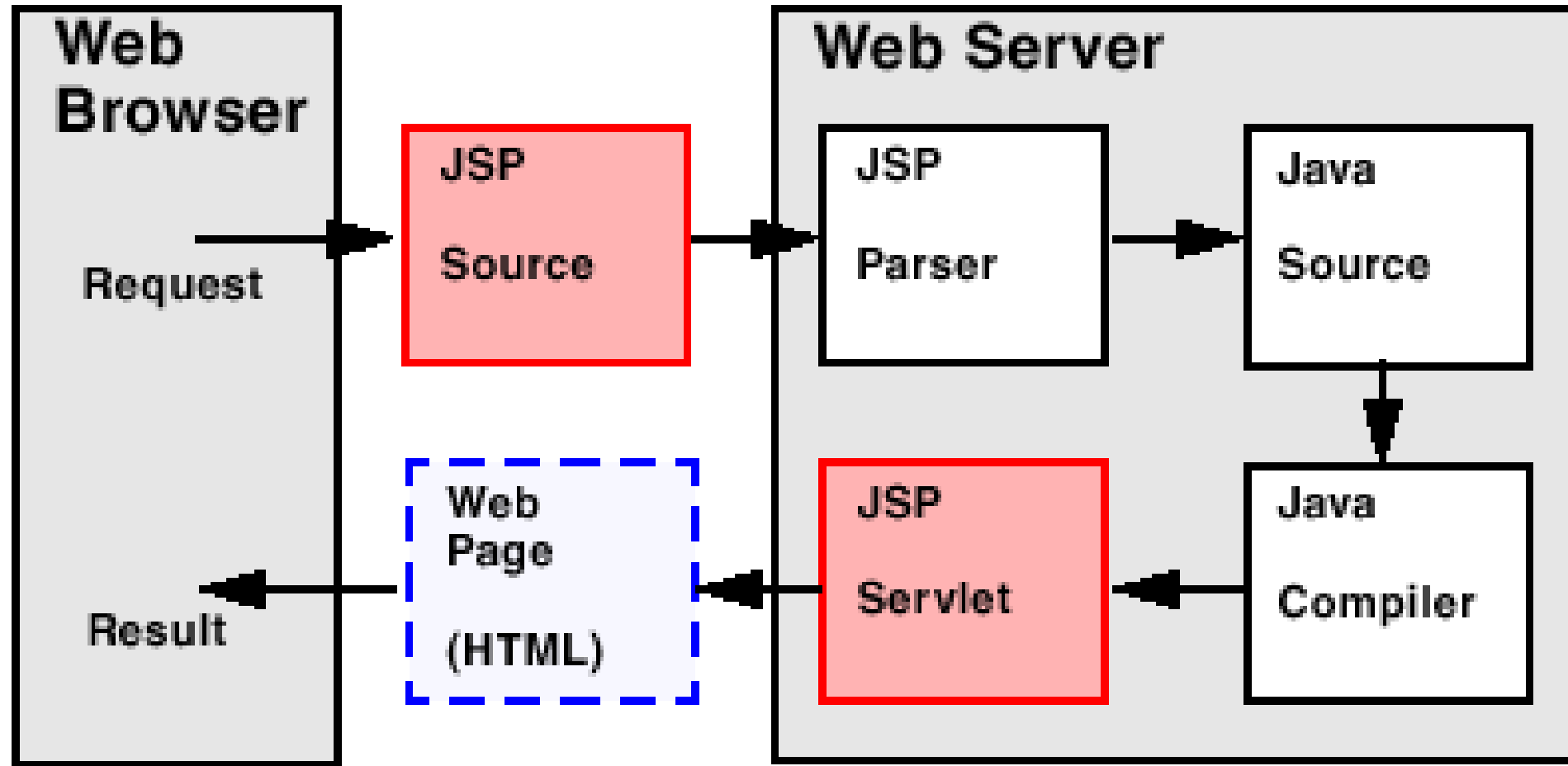


Java Web Application Technologies (Presentation/Web Tier)





First run of a JSP





Benefits of JSP

- **Convenient:**
 - We already know Java and HTML!
 - Provides an extensive infrastructure for:
 - Tracking sessions.
 - Managing cookies.
 - Reading and sending HTML headers.
 - Parsing and decoding HTML form data.
- **Efficient:**
 - Every request for a JSP is handled by a simple Java thread. Hence, the time to execute a JSP document is not dominated by starting a process.



Benefits of JSP cont.

- **Portable**
 - JSP follow a well standardized API.
 - The Java VM which is used to execute a JSP file is supported on many architectures and operating systems.
- **Inexpensive**
 - There are a number of free or inexpensive Web Servers that are good for commercial-quality websites.



JSP vs Servlet

Writing Simple Program









JSP Ingredients

- Besides HTML, a JSP may contain following elements
 - **directive elements**
 - Global control of JSP
 - **scripting elements**
 - JSP comments.....
 - declarations
 - Instance variables and methods
 - expressions
 - Java code fragment which returns String
 - scriptlets.....
 - Blocks of java code
 - **action elements**
 - special JSP tags

`<%@` `%>`

`<!--` `-->`

`<#!` `%>`

`<%=` `%>`

`<%` `%>`

`<jsp:...` `/>`



JSP Scripting Elements



Scripting Elements: **Comments**

- Two types of comments used in JSP

1. **HTML comment:** comment is in browser

```
<!-- comment text -->
```

2. **JSP comment (Fully secret comment):** Comment is not in browser

```
<%-- comment text --%>
```



Scripting Elements: Expressions

- **Format**

- `<%= Java Expression %>`

- **Result**

- Expression evaluated, **converted to String**, and placed into HTML page at the place it occurred in JSP page

- **Examples**

- Time: `<%= new java.util.Date() %>`
- Welcome: `<%= request.getParameter("name") %>`



Scripting Elements: Scriptlets

- **Format**

- `<% Java Code %>`

- **Result**

- Variables defined inside scriptlet are local

- **Example**

`<%`

```
String n = request.getParameter("name");  
out.println("Welcome" + n);
```

`%>`



Scripting Elements: Declarations

- **Format**
 - `<%! Java Code %>`
- **Result**
 - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- **Examples**
 - `<%! private int someField = 5; %>`
 - `<%! private void someMethod(...) {...} %>`



Example Code

Sum of two numbers

Using JSP scripting elements



JSP Scripting Elements

XML style

- **Comment:** No equivalent
- **Declaration:** `<jsp:declaration>` `</jsp:declaration>`
- **Expression:** `<jsp:expression>` `</jsp:expression>`
- **Scriptlet:** `<jsp:scriptlet>` `</jsp:scriptlet>`



JSP implicit objects & directive elements

Lec - 35



JSP Journey

- **directive elements**
- **scripting elements**
 - JSP comments.....
 - declarations
 - expressions
 - scriptlets.....
- **action elements**
 - special JSP tags

implicit
objects



<code><%@</code>	<code>%></code>
<code><!--</code>	<code>--%></code>
<code><%!</code>	<code>%></code>
<code><%=</code>	<code>%></code>
<code><%</code>	<code>%></code>
<code><jsp:...</code>	<code>/></code>



Implicit Objects



Implicit Objects

- Objects that can be used in scriptlets & expressions without defining them before.
 - **request**
 - **response**
 - **out**
 - **Session**
 - **application**
 - **config**
 - **pageContext**
 - **exception**
 - **page**

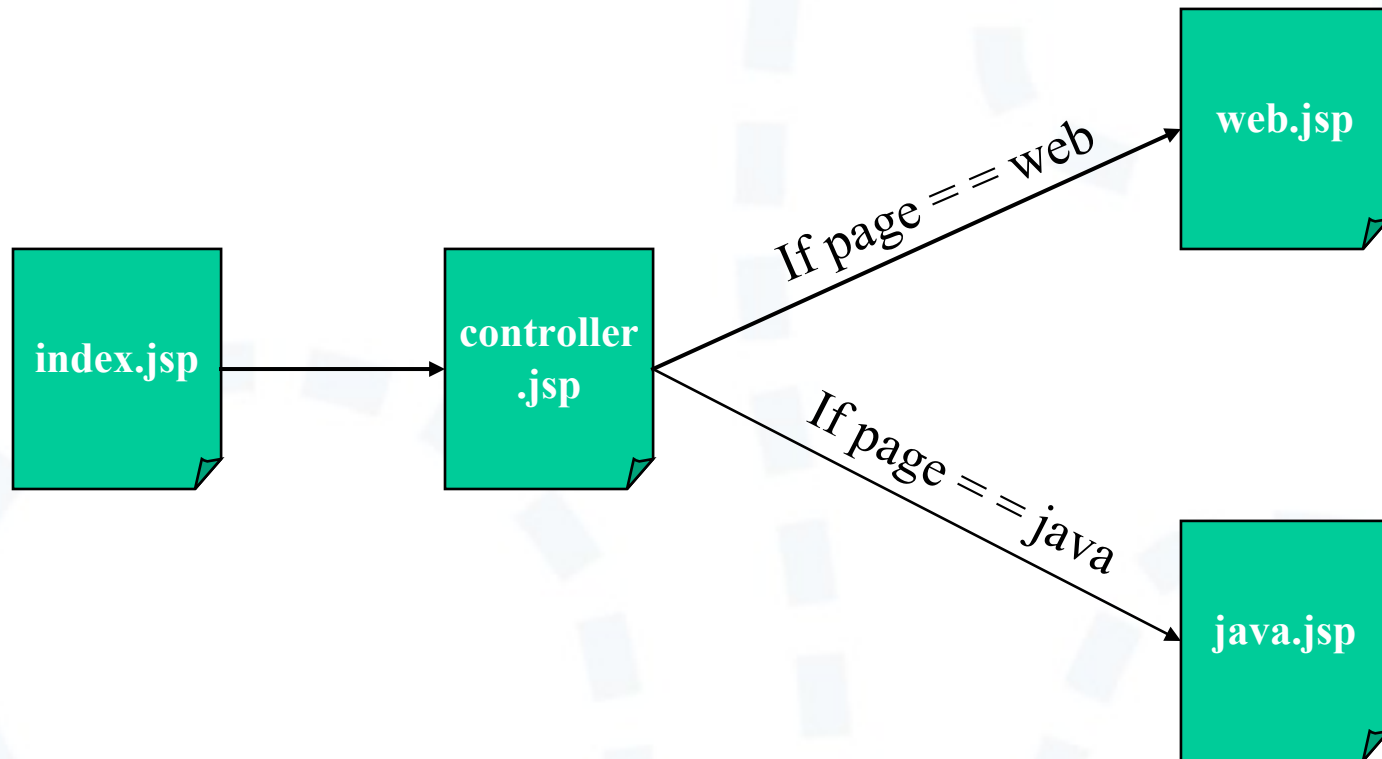


Implicit Objects cont.

- **request:** `javax.servlet.HttpServletRequest`
 - request object that is the reason for the servlet to run
- **response:** `javax.servlet.HttpServletResponse`
 - Response for the request
- **out:** `javax.servlet.jsp.JspWriter`
 - Output stream writer



Example of Implicit Objects





Example Code

Use of implicit objects



JSP Journey

- directive elements
- scripting elements
 - JSP comments.....
 - declarations
 - implicit
objects {
 - expressions
 - scriptlets.....
- action elements
 - special JSP tags

`<%@            %>`

`<!--            --%>`

`<%!            %>`

`<%=            %>`

`<%            %>`

`<jsp:...    />`



JSP Directives



JSP Directives

- Used to convey special processing information about the page to JSP container
- Enable programmer to:
 - Specify page settings
 - Include content from other resources
 - Specify custom-tag libraries



JSP Directives cont.

- **Format**

- `<%@ directive {attribute="val"}* %>`

- **JSP directives**

- page: `<%@ page {attribute="val"}* %>`

- include: `<%@ include {attribute="val"}* %>`

- taglib: `<%@ taglib {attribute="val"}* %>`



JSP page Directive



JSP **page** Directive

- Give high level information about servlet that will result from JSP page
- Can be used anywhere in the document
- Can control
 - Which classes are imported
 - What class the servlet extends
 - What MIME type is generated
 - How multithreading is handled
 - If the servlet participates in session
 - Which page handles unexpected errors



JSP page Directive cont.

- Defines attributes of the page
 - `<%@ page {attribute="val"}* %>`
 - Some Attributes
 - `language` = "java"
 - `extends` = "package.class"
 - `import` = "package.*,package.class,..."
 - `session` = "true | false"
 - `Info` = "text"
 - `contentType` = "mimeType"
 - `isThreadSafe` = "true | false"
 - `errorPage` = "relativeURL"
 - `isErrorPage` = "true | false"



JSP page Directive cont.

- Some example uses are:
 - To import the package like java.util
 - `<%@ page import = "java.util.*" info = "using util package" %>`
 - To declare this page as an error page
 - `<%@ page isErrorPage = "true" %>`
 - To generate the excel spread sheet
 - `<%@ page contentType = "application/vnd.ms-excel" %>`



JSP **include** Directive



JSP **include** Directive

- Lets you include (reuse) navigation bars, tables and other elements in JSP page
- You can include files at
 - Translation Time (by using include directives)
 - Request Time (by using Action elements)



Including Pages at Translation Time using **include** directive

- Format

```
<%@include file="relativeURL"%>
```

- Purpose

- To include a file in JSP document at the time document is translated into a servlet.
- May contain JSP code that affect the main page such as response header settings etc.



Example Code

use of include directive



Modify Last Example to

Generate Excel Spread-sheets by setting content type



XML Syntax for Directives

- `<jsp:directive.directiveType attribute="value" />`
- For example, the XML equivalent of

`<%@ page import = "java.util.*" %>`

Is

`<jsp:directive.page import="java.util.*" />`



JSP Journey

- **directive elements**
- **scripting elements**
 - JSP comments.....
 - declarations
 - expressions
 - scriptlets.....
- **action elements**
 - special JSP tags

implicit
objects



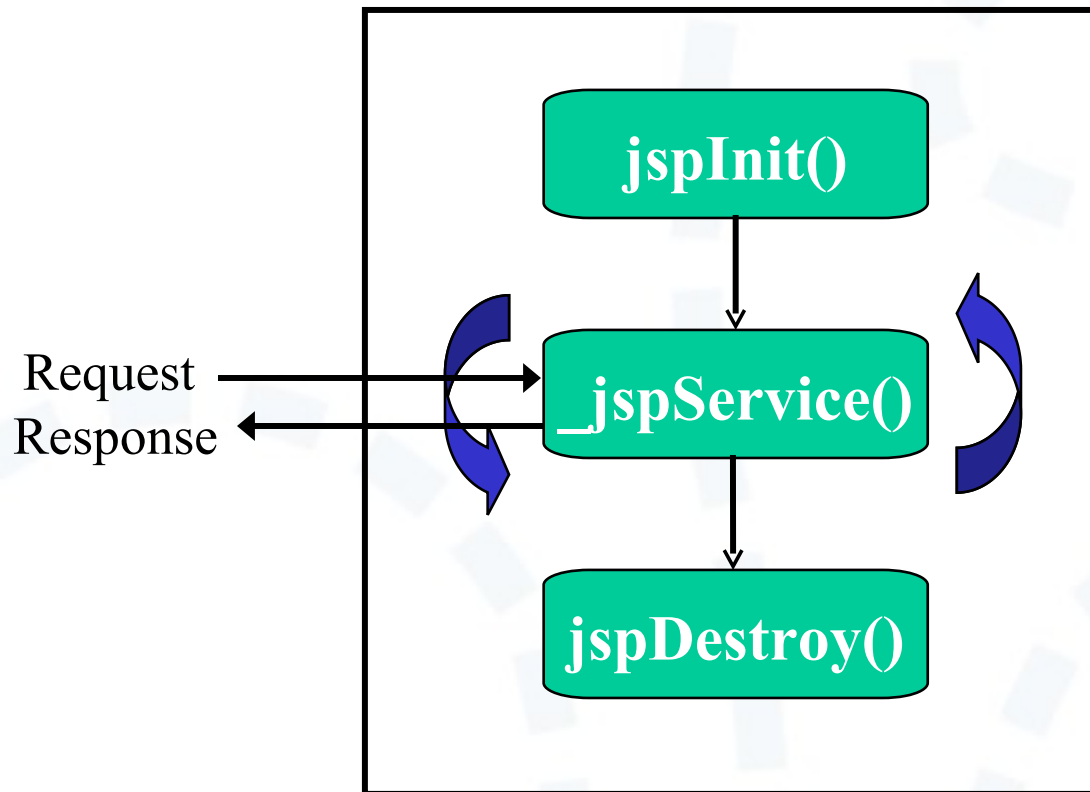
<code><%@</code>	<code>%></code>
<code><%--</code>	<code>--%></code>
<code><%!</code>	<code>%></code>
<code><%=</code>	<code>%></code>
<code><%</code>	<code>%></code>
<code><jsp:...</code>	<code>/></code>



JSP Life Cycle Methods



JSP Life Cycle Methods





CourseOutline Example & JavaBeans

Lec - 36



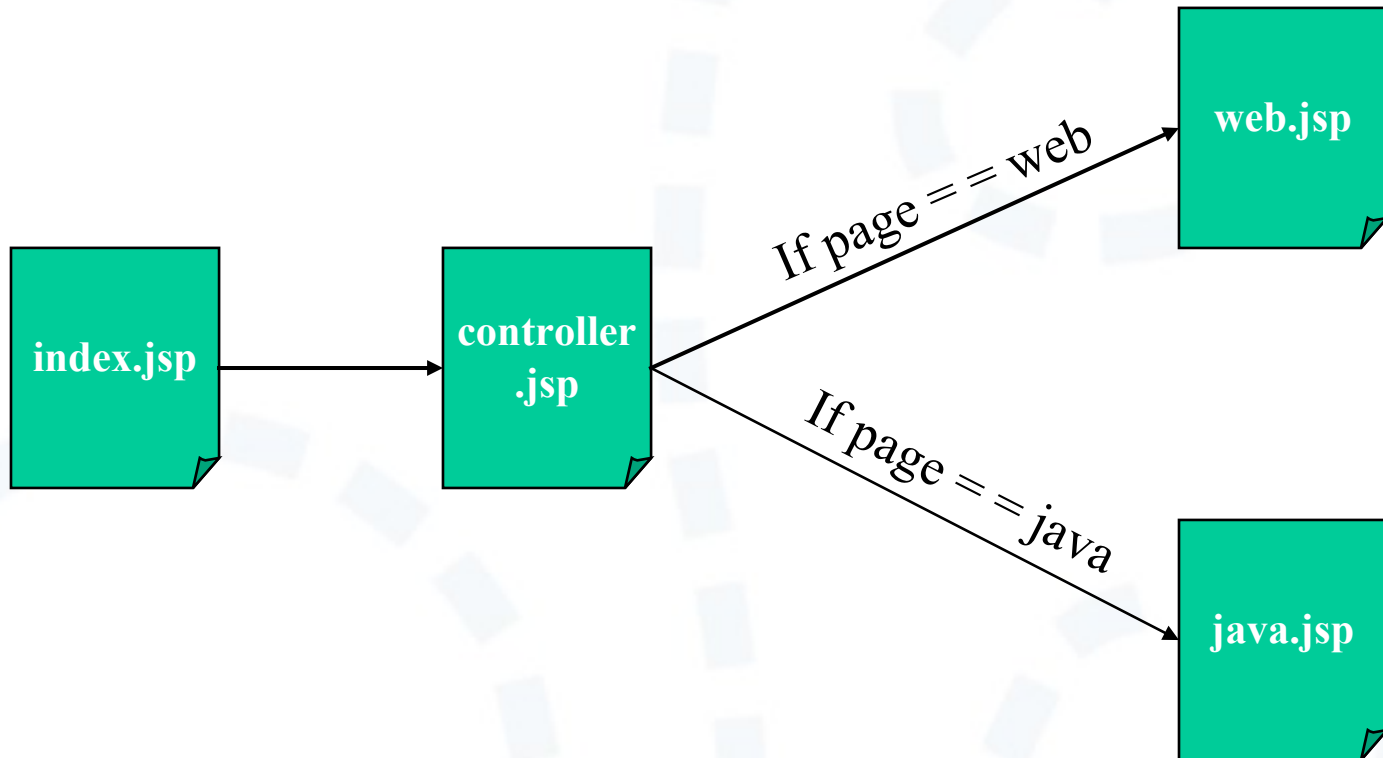
Start with Example

Displaying Course Outlines

- User will select either course “*web design & development*” or “*java*”.
- On submitting request, course outline would be displayed of the selected course from database



Program Flow





Issues with Last Example

- Too much cluttered code in JSP
 - Difficult to understand
- A single page is doing everything – bad approach
 - Displaying contents (Presentation logic)
 - Connecting with database (DB connectivity logic)
 - Results Processing (Business Logic)
- Solution:
 - Use **JavaBeans**



JavaBeans



What are JavaBeans?

- Java class that can be easily reused and composed together in an application
- Any java class that follows **certain design conventions** can be a JavaBeans component



JavaBeans Design Conventions

- A bean class must have a zero argument constructor
- A bean class should not have any public instance variables (fields)
- Private values should be accessed through setters/getters
 - For boolean data types, use **boolean isXXX()** & **setXXX(boolean)**
- A bean class must be serializable



A Sample JavaBean

```
public class MyBean implements Serializable {  
  
    private String name;  
  
    // zero argument constructor  
    public MyBean( ){  
        name = "";  
    }  
  
    // standard setter  
    public void setName(String n) {  
        name = n;  
    }  
}
```



A Sample JavaBean cont.

```
// standard getter
public String getName( ) {
    return name;
}

// any other method
public void print( ) {
    System.out.println("Name is: " + name);
}

} // end Bean class
```



Incorporating Bean in Last Example

- Create a *CourseOutlineBean*
 - Has following attributes
 - sessionNo
 - topic
 - assignment
- Create a *CourseDAO (bean)*
 - Encapsualte database connectivity logic
- *web.jsp & java.jsp will use these beans*



Example Code

Incorporating JavaBeans in
CourseOutline Example



JSP Action Elements

Lec - 37



JSP Journey

- **directive elements**
- **scripting elements**
 - JSP comments.....
 - declarations
 - implicit objects** {
 - expressions
 - scriptlets.....
- **action elements**
 - special JSP tags

`<%@` `%>`

`<!--` `-->`

`<%!` `%>`

`<%=` `%>`

`<%` `%>`

`<jsp:...` `/>`



JSP Action Elements

JSP Action Elements

- **Format**
 - Expressed using XML syntax
 - Opening tag `<jsp:actionElement attribute="value" ...`
`>`
 - Body `body`
 - Closing tag `</jsp:actionElement >`
 - Empty tags without body
`<jsp:actionElement attribute="value" ... />`



JSP Action Elements cont.

- Purpose
 - To work with **JavaBeans**
 - To include pages at request time
 - To forward request to another resource etc.



Some JSP Action Elements

- To work with JavaBeans
 - `<jsp:useBean />`
 - `<jsp:setProperty />`
 - `<jsp:getProperty />`
- To include resources at request time
 - `<jsp:include />`
- To forward request to another JSP or Servlet
 - `<jsp:forward />`
- To work with applets
 - `<jsp:plugin />`
- And many more....



Working with JavaBeans using JSP Action Elements



Working with JavaBeans using JSP action elements

- JSP action elements also helps to work with JavaBeans
 - `<jsp:useBean />`
 - Makes a JavaBeans object available in a page
 - `<jsp:setProperty />`
 - Sets a JavaBeans property value
 - `<jsp:getProperty />`
 - Gets a JavaBeans property value



JSP **useBean** Action Element

- `jsp:useBean` is being equivalent to building an object in scriptlet

```
<%  
    MyBean m = new MyBean();  
%>
```

OR

```
<jsp:useBean id="m" scope="page" class="vu.MyBean" />
```




JSP **setProperty** Action Element

- `jsp:setProperty` is being equivalent to following code of scriptlet

```
<%  
    m.setName("ali");  
%>
```

OR

```
<jsp:setProperty id="m" property="name"  
                value="ali" />
```



JSP getProperty Action Element

- `jsp:getProperty` is being equivalent to following code of scriptlet

```
<%  
    String name = m.getName();  
    out.println(name);  
%>
```

OR

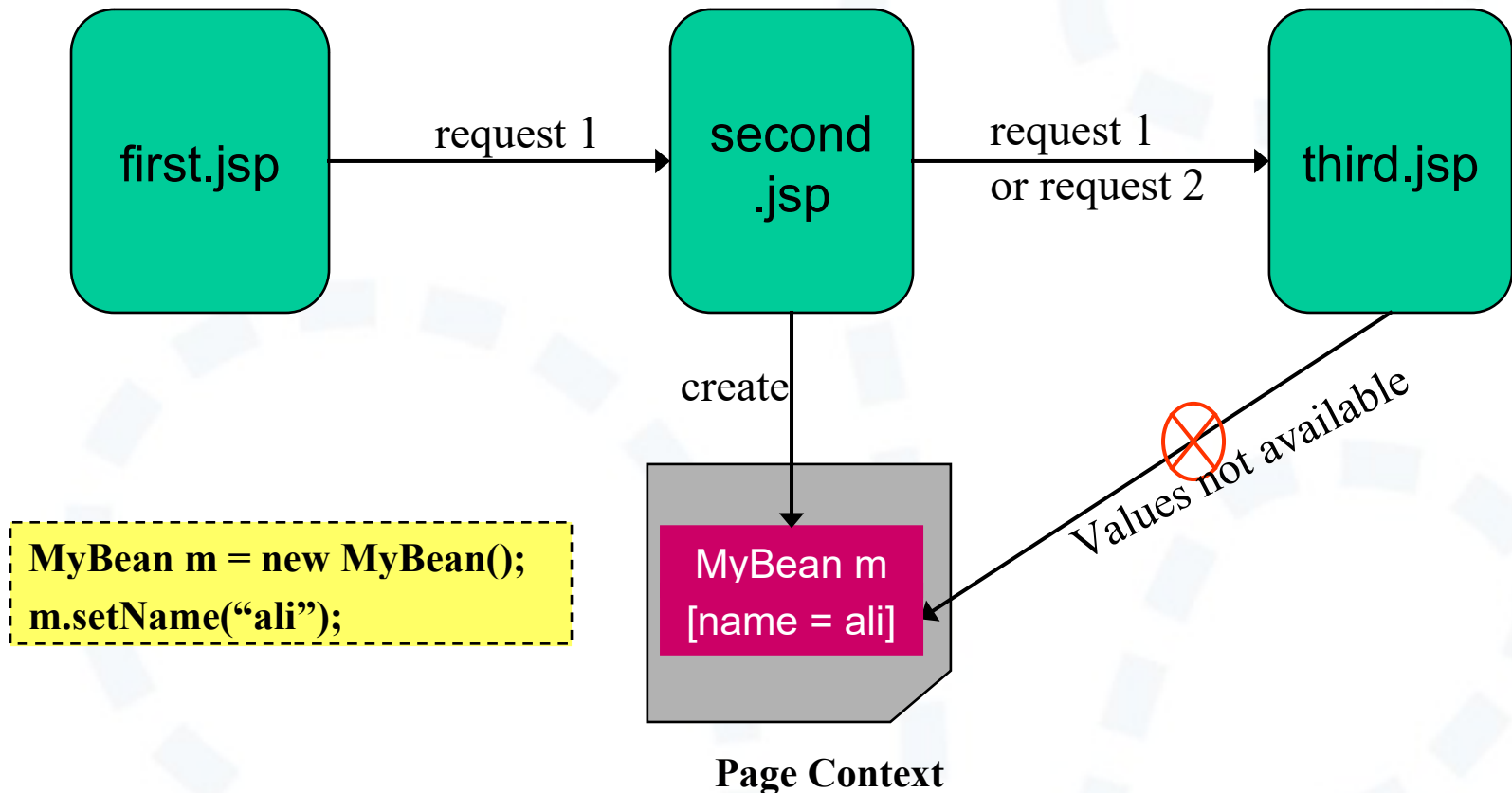
```
<jsp:getProperty id="m" property="name" />
```



Sharing Beans & Object Scopes

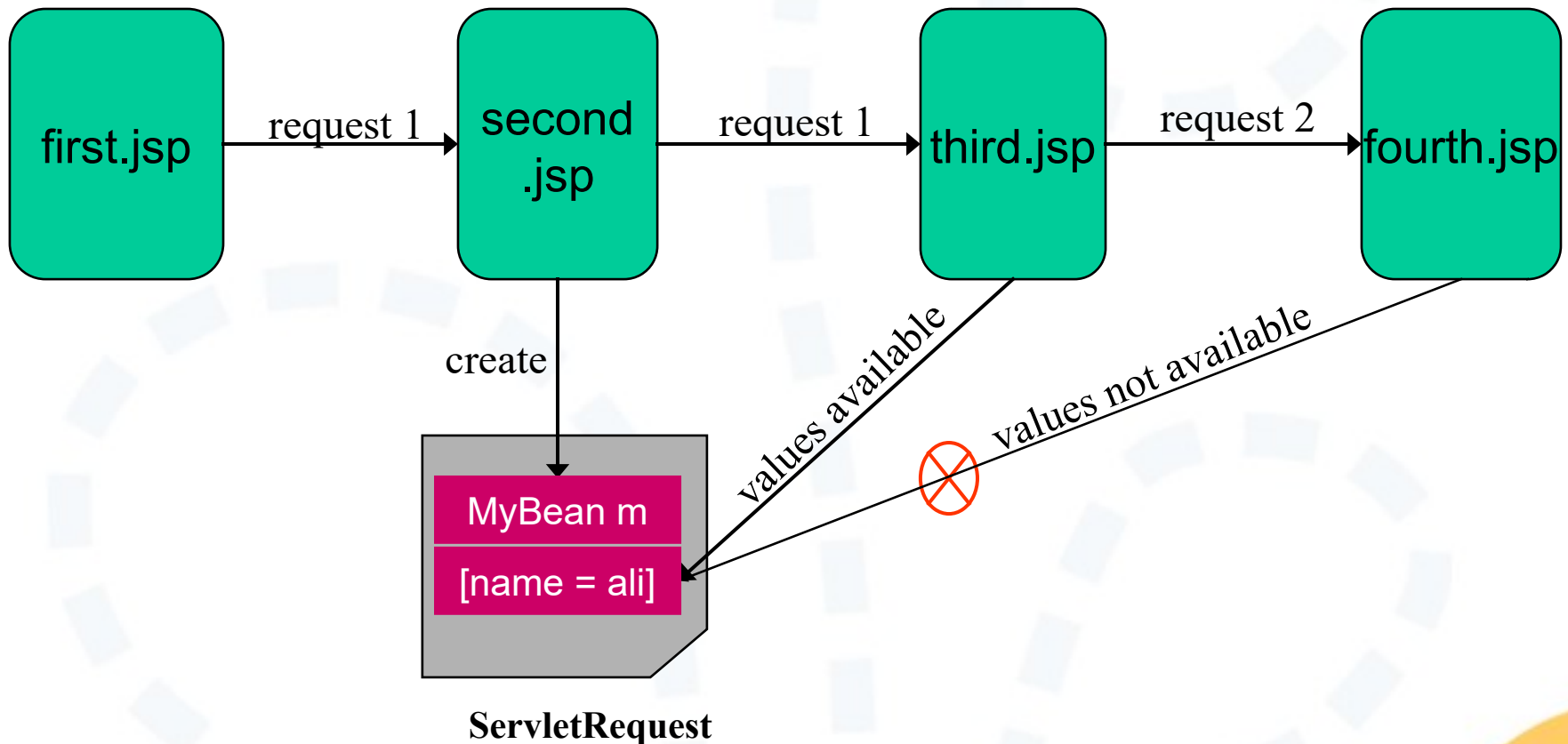


Understanding page Scope



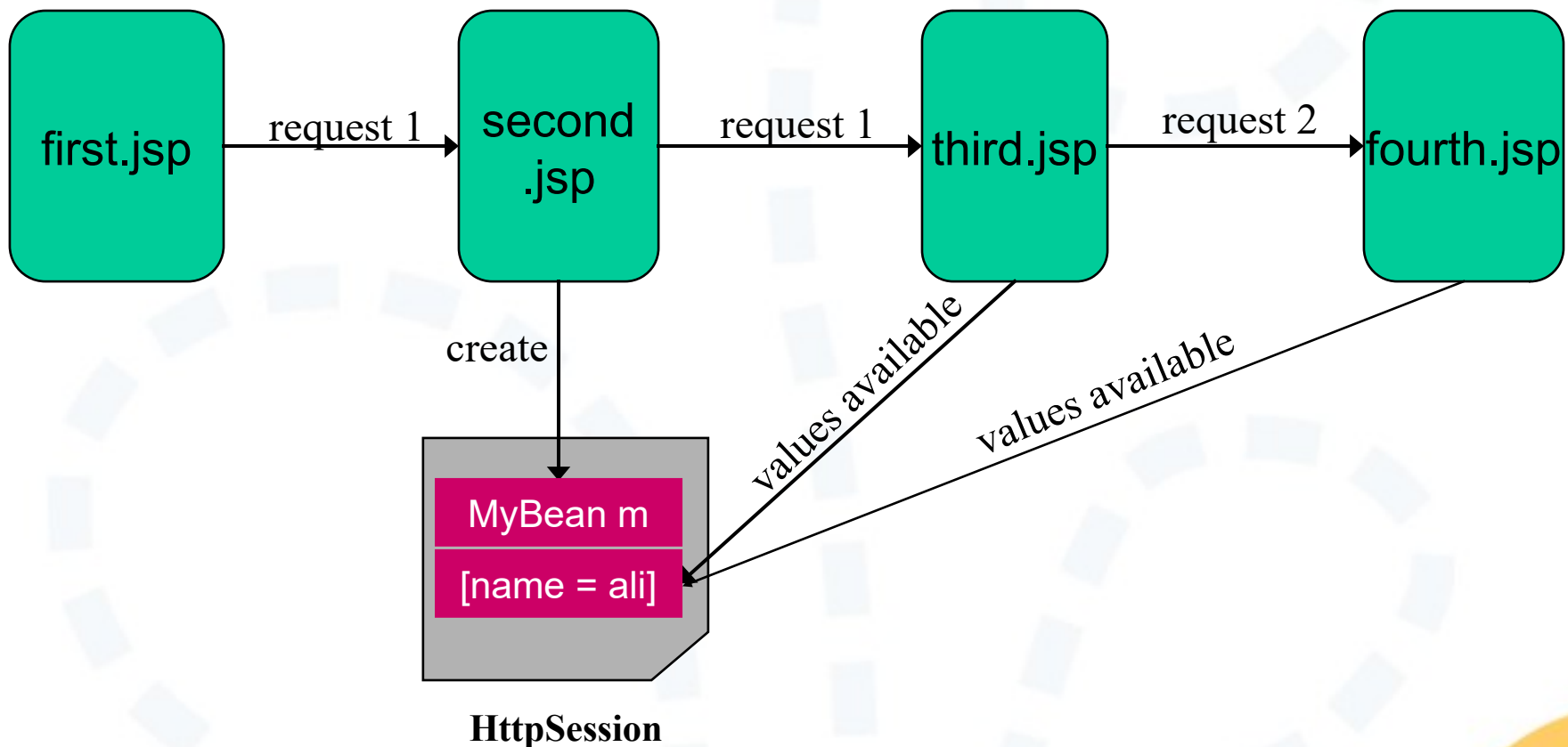


Understanding request Scope



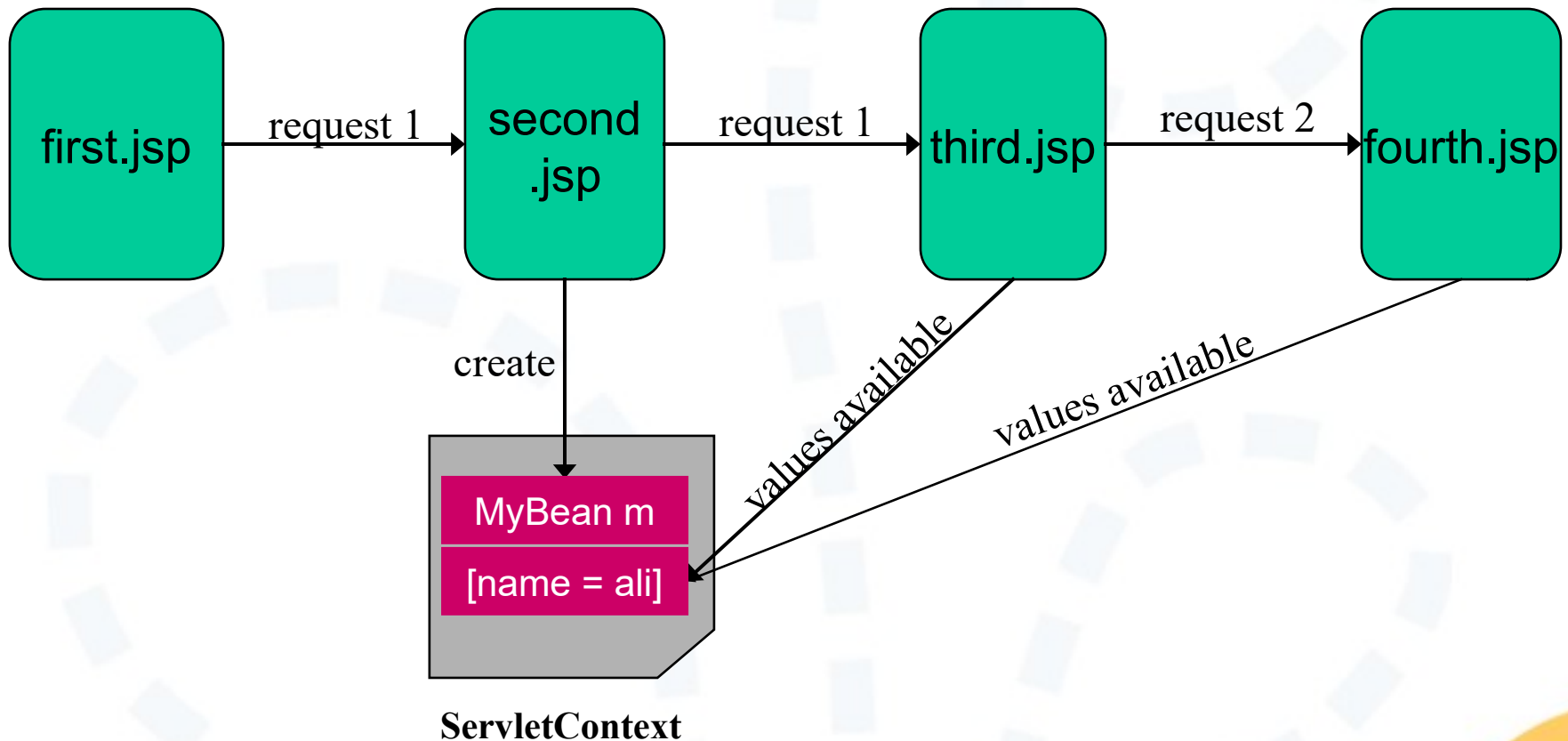


Understanding session Scope





Understanding application Scope





Different Object Scopes

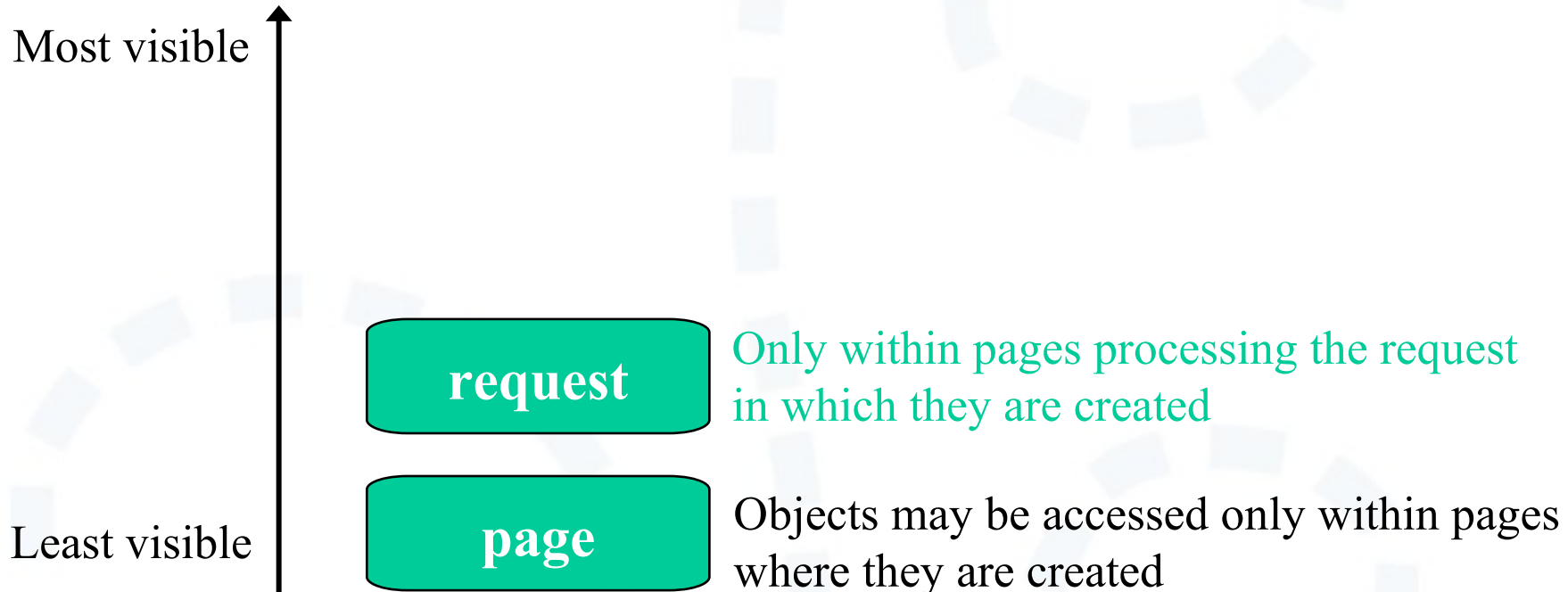
Most visible ↑
Least visible ↓

page

Objects may be accessed only within pages where they are created

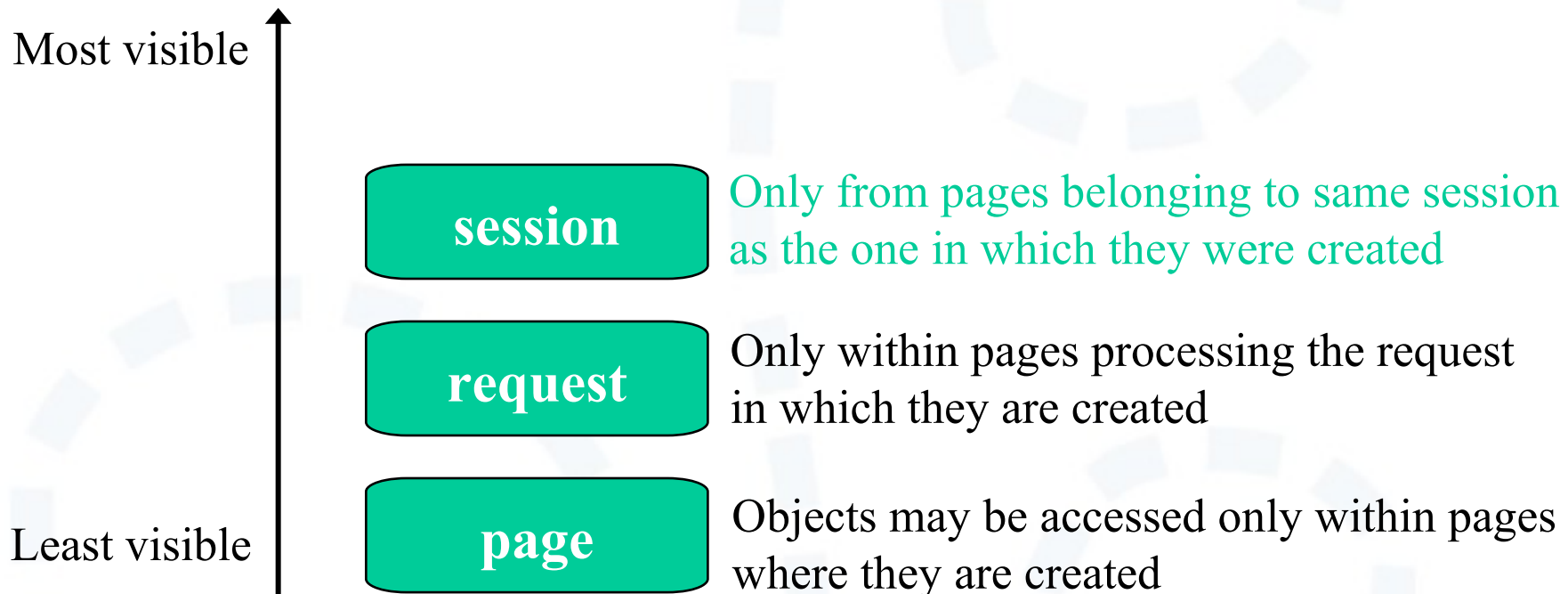


Different Object Scopes



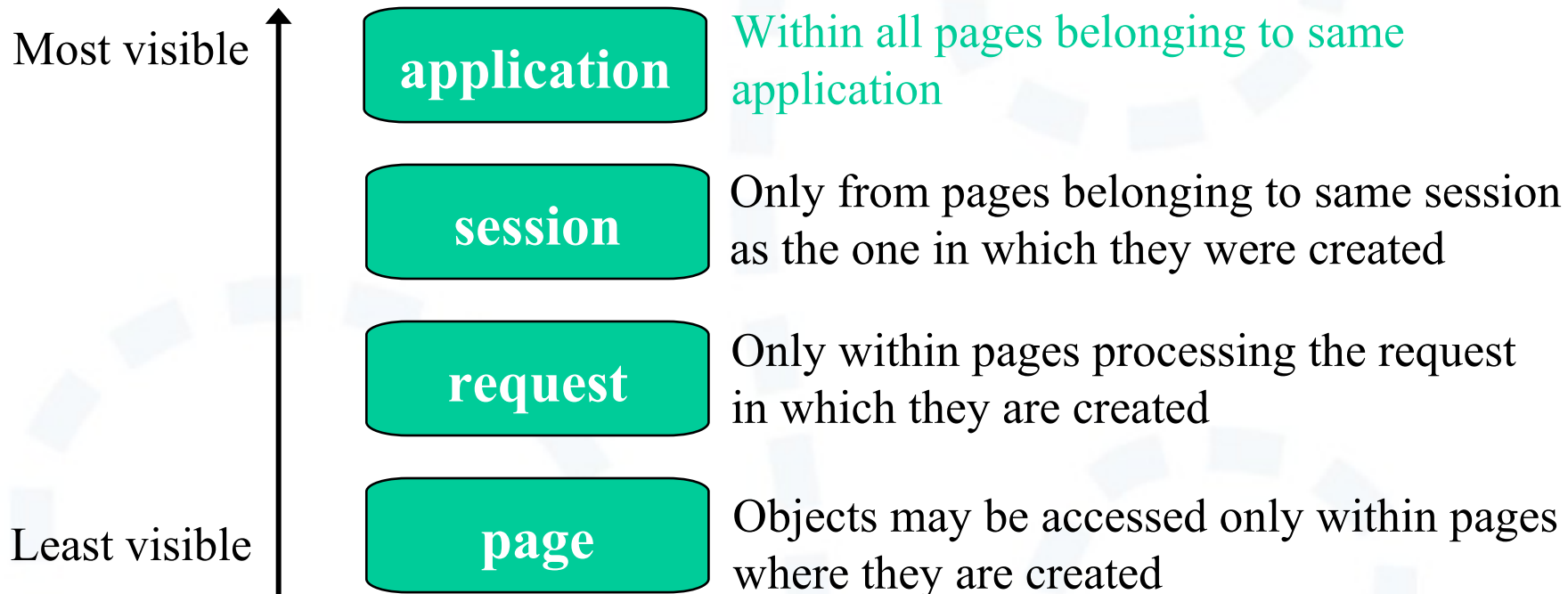


Different Object Scopes





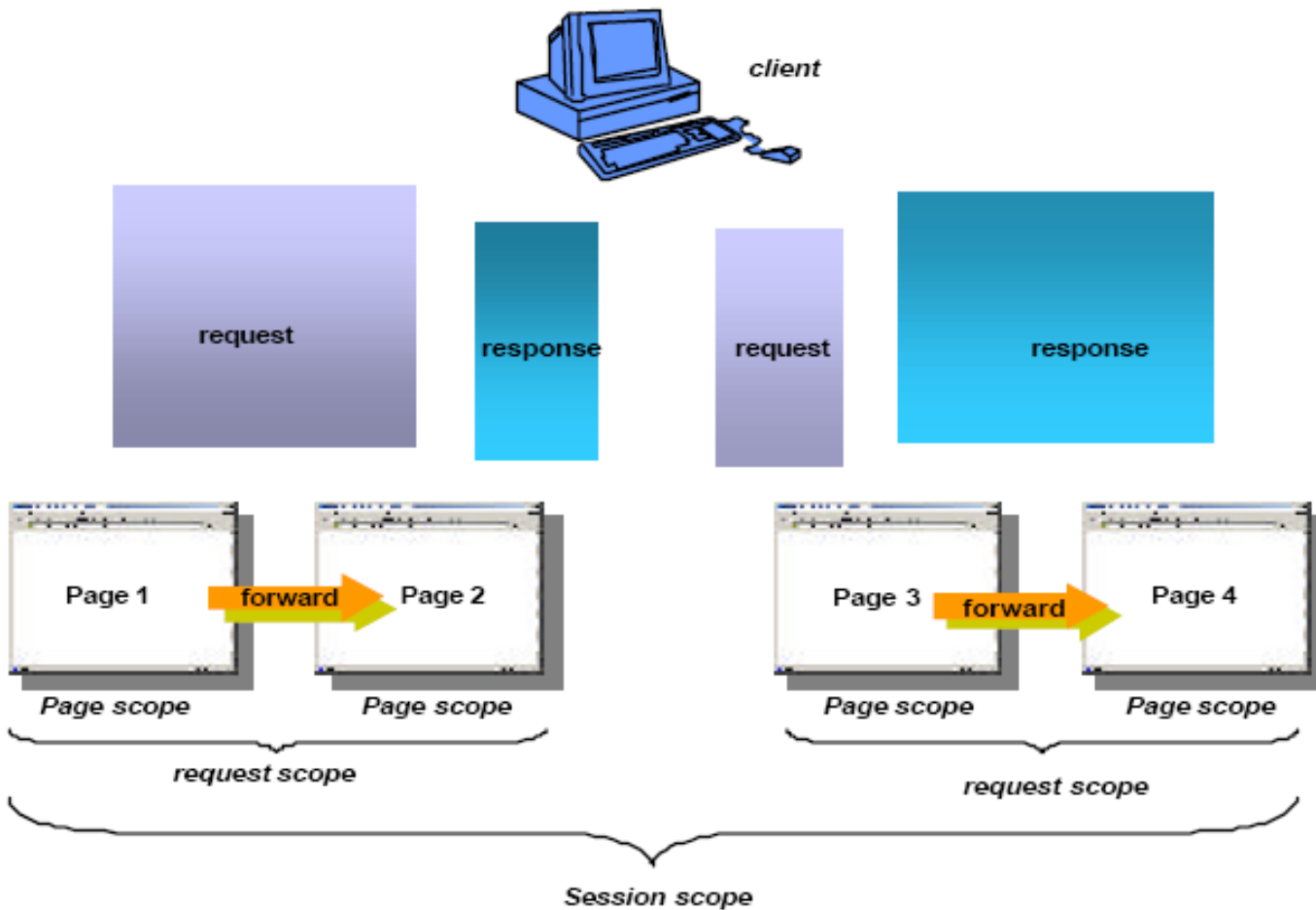
Different Object Scopes





Object Scopes

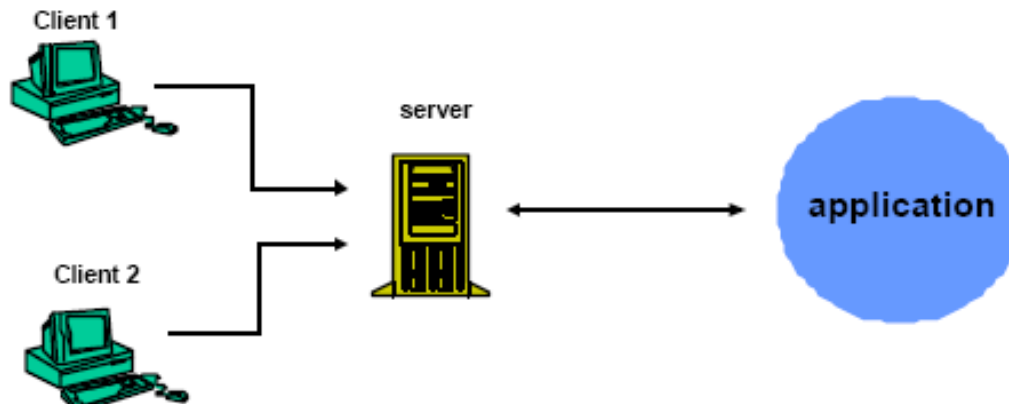
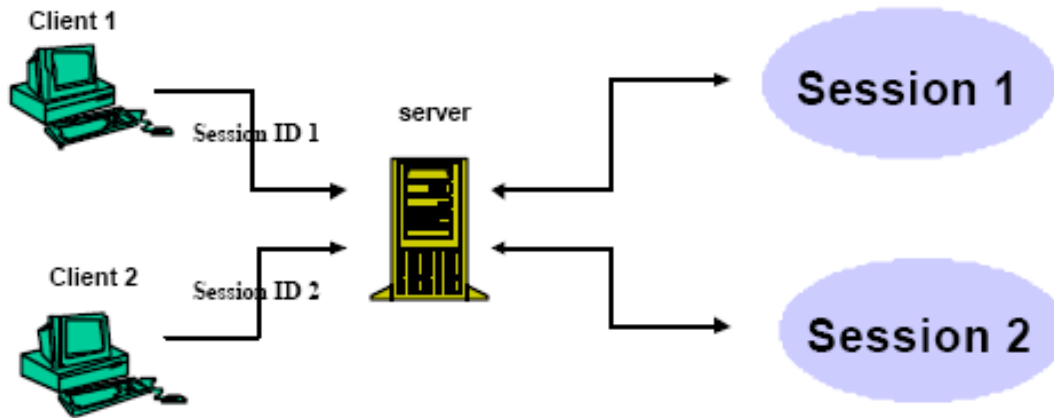
session, request & page





Object Scopes

session vs. application





More JSP Action Elements



JSP include Action Element cont.

- `jsp:include` is being equivalent to following code of scriptlet

```
<%  
    RequestDispatcher rd = request.getRequestDispatcher("one.jsp");  
    rd.include(request, response);  
%>
```

OR

```
<jsp:include page="one.jsp" flush="true" />
```



JSP forward Action Element

- To forward request to another resource
- Format

```
<jsp:forward page="one.jsp" />
```

- Works similar to following code of scriptlet

```
<%
```

```
    RequestDispatcher rd = request.getRequestDispatcher("one.jsp");  
    rd.forward(request, response);
```

```
%>
```


JSP Tag Libraries

Lec - 38

Last Lecture Example

- We incorporated JavaBeans in “Course Outline” Example
- But still have to write java code inside `java.jsp` & `web.jsp`
- As discussed, JSPs are built for presentation purpose

Last Lecture Example

- What if we replace all code of scriptlet with one single line (custom tag)

```
<%  
    CourseDAO courseDAO = new CourseDAO();  
    .....  
    for ( ..... ) {  
        .....  
    }  
    .....  
%>
```

- `<mytag:displaycourse pageName="java" />`

Contents

- What is a Custom Tag ?
- Why build Custom Tag?
- Types of Tags
- Developing Custom Tags
- Using Custom Tags
- Examples

What is Custom Tag ?

- A user defined component to perform certain action.
- Provides mechanism for encapsulating complex functionality for use in JSPs.
- We have already seen & used many built-in tags
 - `<jsp:useBean/>`
 - `<jsp:include/>`
 - `<jsp:forward/>` etc

Advantages

- More cleaner separation of processing logic and presentation, than Java Beans
- Have access to all the JSP page objects
- Can be customized using attributes

Types of Tags

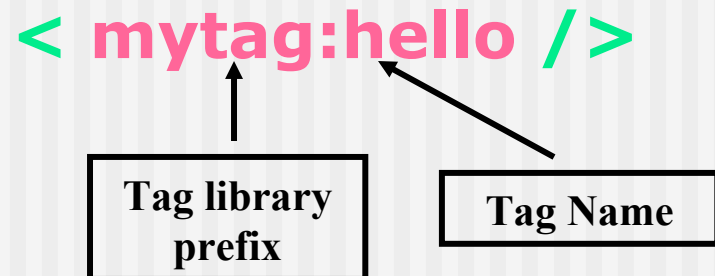
Types of Tags

1. Simple Tag
2. Tag with Attributes
3. Tag with Body

Types of tags

1. Simple Tags

- Start and End of tag
- No body within tag
- No Attributes
- For Example:



Types of tags

2. Tags with attributes

- Start and End of tag
- Attributes within tag
- No body enclosed
- For Example :

`< mytag:hello attribute="value" />`

Types of tags

3. Tags with body

- Start and End of tag
- Body enclosed within tag
- Attributes (optional)
- For Example :

```
<mytag:hello optional_attributes .....>  
    some body  
</mytag:hello>
```

Building Custom Tags

Building Custom Tags

- We can build custom tag using either of the following specs
 - JSP 1.2
 - JSP 2.0
- We will use JSP 2.0

Building Custom Tags - Steps

1. Develop the Tag Handler class
2. Write the Tag Library Descriptor (tld) file
3. Deployment

1 - Tag Handler class

- Tag Handler is the java class
- Implicitly called when the associated tag is encountered in the JSP
- Must implement **SimpleTag** interface
- Usually extend **SimpleTagSupport** class. For example

```
public class MyTagHandler
    extends SimpleTagSupport {
    .....
}
```

1 - Tag Handler class cont.

- `doTag()` method
 - By default does nothing
 - Need to implement / override to code tag's functionality
 - Invoked when the `end element` of the tag encountered

1 - Tag Handler class cont.

- **Implicit objects** are available to tag handler class through **pageContext** object
- **pageContext** object can be obtained using **getJspContext()** method.
- For example, to retrieve out object

```
PageContext pc = (PageContext) getJspContext();  
JspWriter out = pc.getOut();
```

2 - Tag Library Descriptor (tld)

- XML based document
- Specifies information required by the JSP container such as:
 - Tag library version
 - JSP version
 - Tag name
 - Tag handler class name
 - Attribute names etc.

3 - Deployment

- Place Tag Handler class in `myapp/WEB-INF/classes` folder of web application
- Place TLD file in `myapp/WEB-INF/tlds` folder

Using Custom Tags

Using Custom Tags

- Use “taglib” directive in JSP to refer to the tag library
`<%@ taglib uri = “TLDFileName” prefix = “mytag” %>`
- Call the tag by its name as defined in TLD
 - If tag name defined in TLD is *hello* then we write
`< mytag:hello />`
- Behind the Scenes
 - Container calls the appropriate Tag Handler
 - Tag Handler will write the appropriate response back to the page

Building Simple Tag

Building Simple Tag

- A simple tag that displays "*Hello World*"
- Approach
 - Extend Tag Handler class from `SimpleTagSupport` class
 - Override `doTag()` method
 - Build TLD file
 - Deploy

Example Code

Building Simple Tag
(displays "*Hello World*")

Building Tag with Attributes

Building Tags with Attributes

- E.g. `<mytag:hello attribute="value" />`
- To handle attributes, need to add
 - Instance variables and
 - Corresponding setter methods
- Container call these setter methods
 - Passed the custom tag attribute's value as arguments

Building Tag with Attribute (cont.)

■ **Example:**

- We will modify our Course Outline Example to incorporate tags
- Based on attribute value, tag will display the respective course outline

Example Code

Building Tag with Attribute
(Modifying CourseOutline Example)



Case Study + MVC

Lec - 39



Error Pages



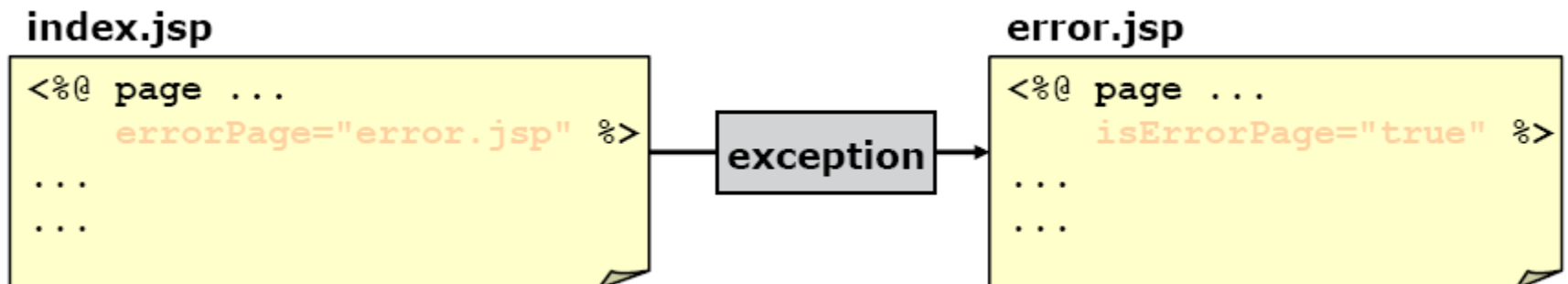
Error Pages

- By means of the page directive, a JSP can be given the responsibility of an **Error page**
- An Error JSP will be called by the Web Server when an **uncaught exception** gets generated
- The exception is passed as an instance of **java.lang.Throwable** to the Error JSP (accessible via implicit **exception** object)



Defining & Using Error Pages

- **isErrorPage** attribute of **page** directive is used to declare a JSP as an error page.
- JSP pages are informed about the error page by setting **errorPage** attribute of **page** directive.





Example Case Study Address Book

using Model-1 architecture



Address Book ingredients

- Database

PersonDB : Database (Access 2000 file format)

Open Design New

Objects

- Tables
- Queries
- Forms
- Reports
- Pages
- Macros

Person : Table

	name	address	phoneNumber
	ali	model town	9231256
	raza	clifton	4329871
▶	suffian	defence	6753311
*			0

Record: 1 2 3 4 5 of 3



Address Book ingredients

- **JavaBeans**

- **PersonInfo**

- Has attributes
 - name
 - address
 - phoneNum

- **PersonDAO**

- Encapsulate Database logic
 - Used to retrieve & save PersonInfo data



Address Book ingredients

- **JavaServer Pages**

- **addperson.jsp**

- Used to collect new person info

- **saveperson.jsp**

- Receives person info from addperson.jsp
 - Saves it into database

- **searchperson.jsp**

- Used to provide search criteria to search Person's info, based on name

- **showperson.jsp**

- Receives person name, to be searched in database
 - Retrieves & display person records against that person name

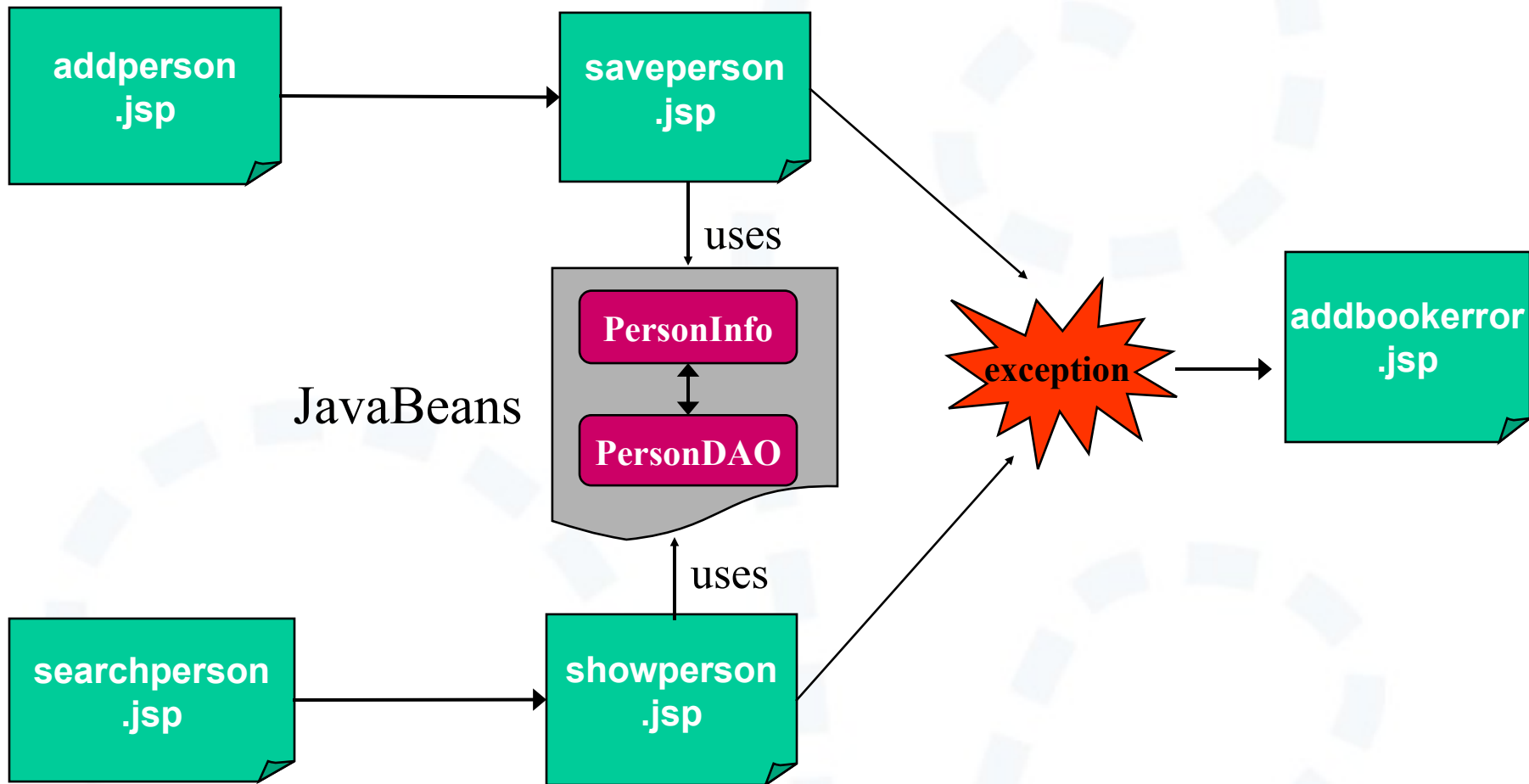


Address Book ingredients

- **JavaServer Pages (cont.)**
 - **Error Pages**
 - **addbookerror.jsp**
 - Identifies the type of exception
 - Displays the message associated with the received exception



Program Flow





Lets put it all together

Netbeans project
addressbookusingmodel1



Model View Controller Design Pattern



Model-View-Controller Design Pattern

- MVC separates
 - Application data (contained in the model) from
 - Graphical presentation components (the view) &
 - Input/Event processing logic (the controller)



MVC in general



controller



view

model



MVC in general: **model**



controller

- Represents the state of the component (ie its data and the methods required to manipulate it)



view

model

- Independent of how the component is viewed or rendered



MVC in general: **view**



controller



view

model

- Appearance of the component
- Can be multiple views for the same model within a single applications
- Model may have different views in different applications or operating systems



MVC in general: **controller**



controller

- Dictates how a component interacts with events



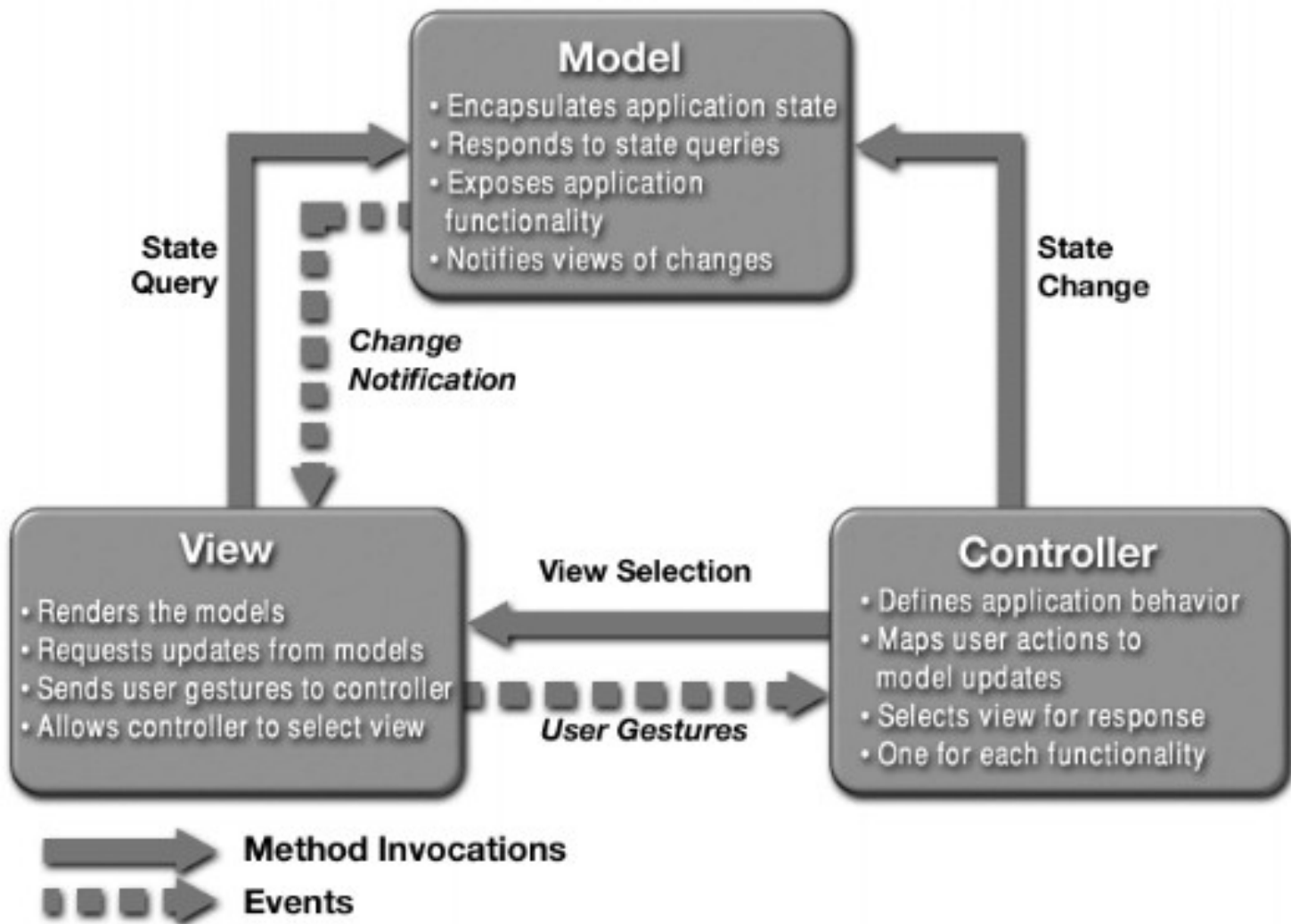
view

model

- Sends events of interest to the model so that it can update itself



MVC in summary





Evolution of MVC Architecture

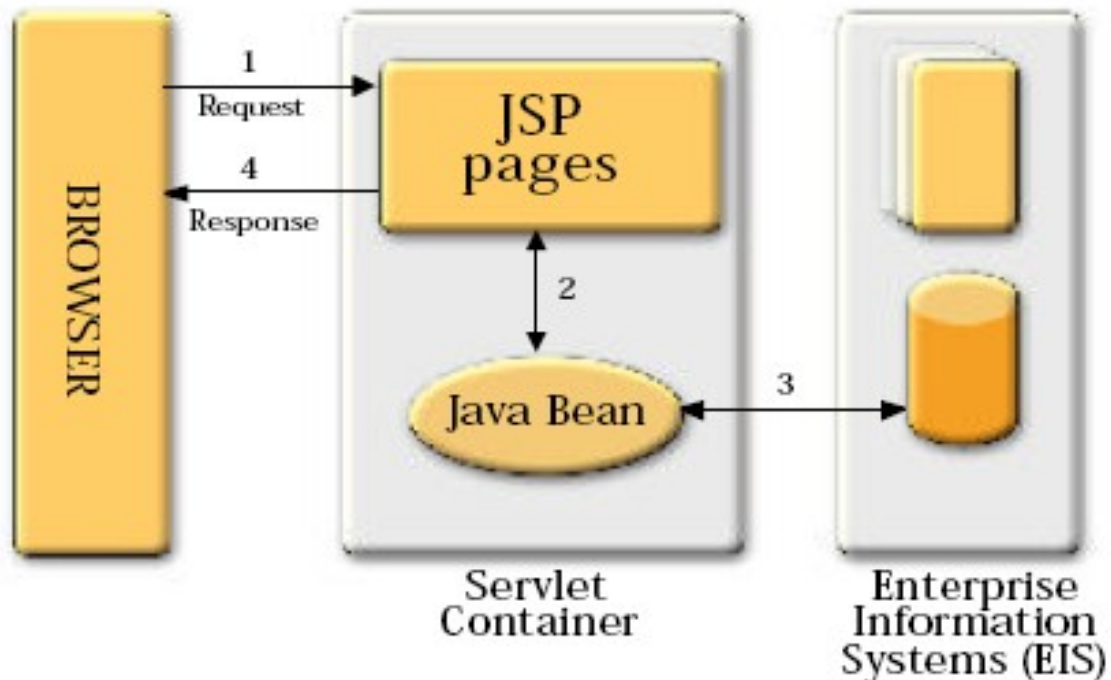
- No MVC
- 2. MVC Model 1
- 3. MVC Model 2
- 4. Web application framework (based on MVC model2)
 - Struts, Sun ONE application framework
- 5. Web application framework
 - JavaServer Faces (JSF)



MVC Model 1 Architecture

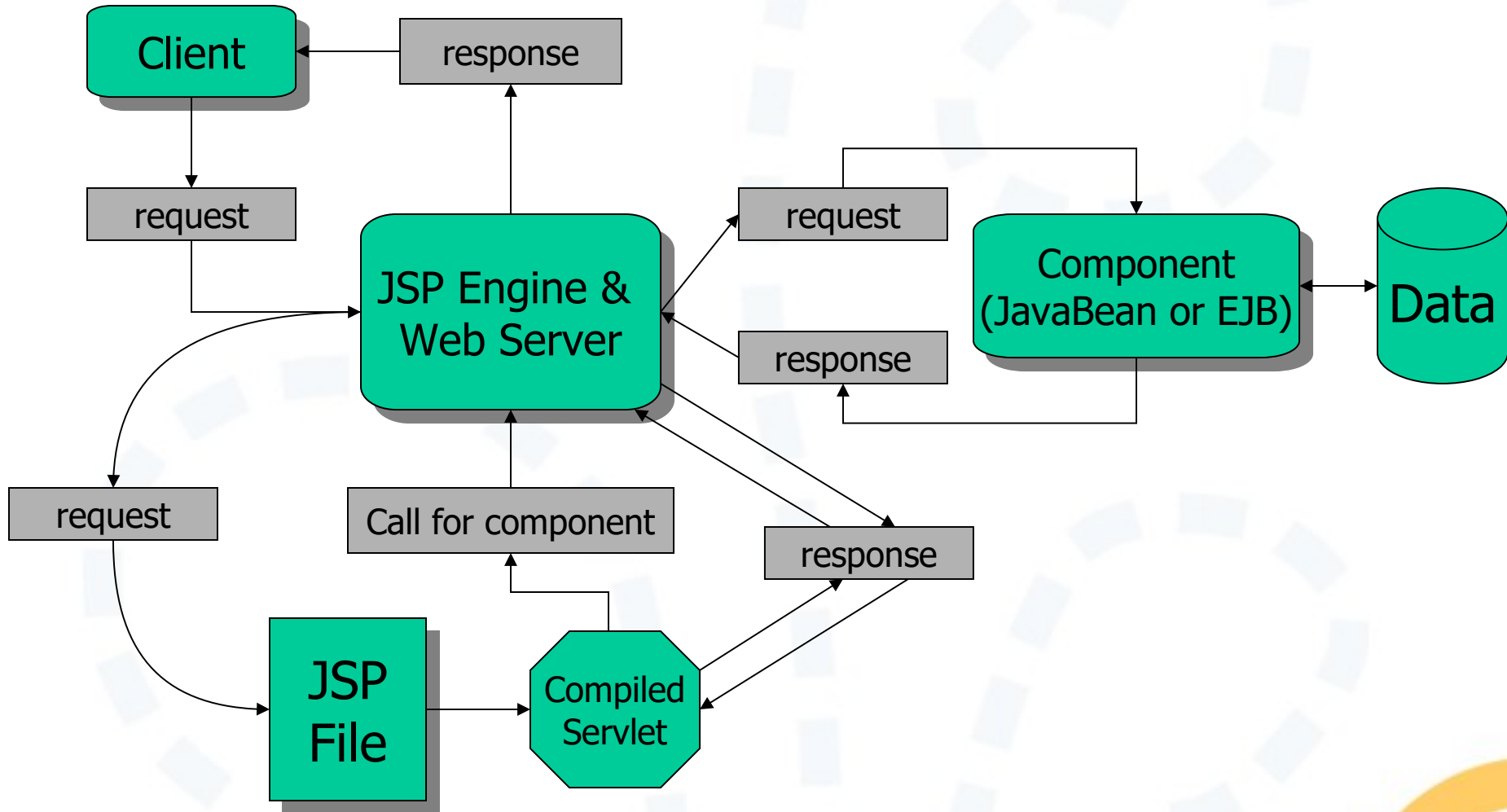


MVC Model 1



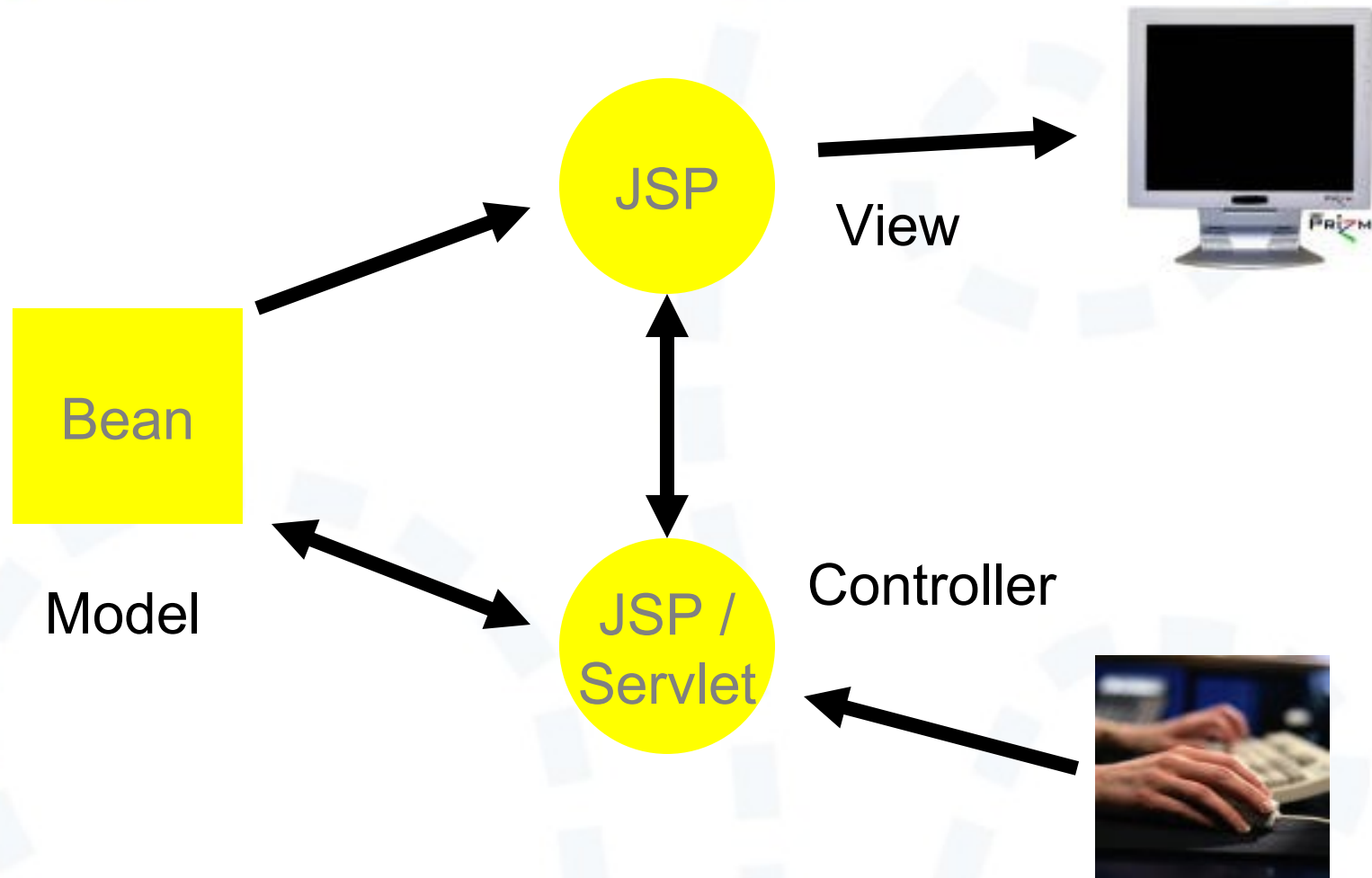


MVC Model 1 – A Detailed Story





MVC Model 2





MVC Model 2 Architecture & AddressBook case study

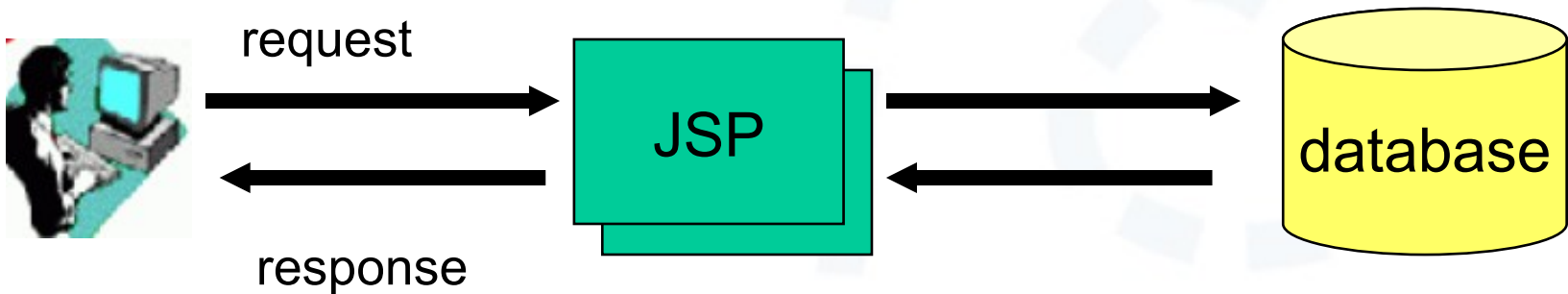
Lec - 40



Last Lecture Review



Page-Centric Approach

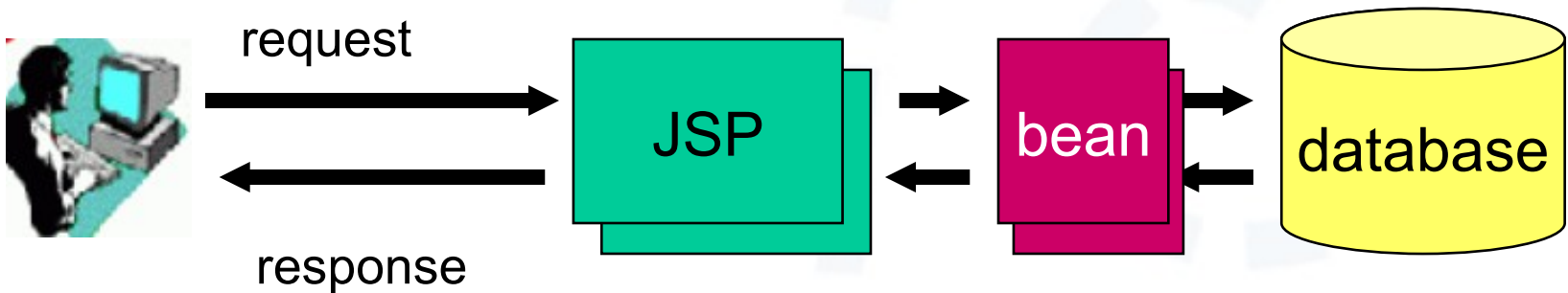


Web-app = collection of JSPs

- + easy to get started (and carried away)
- code is a mixture of presentation, business & data access logic: a maintenance nightmare
- won't scale
- lots of duplication



Page-with-Bean Approach (MVC Model 1)



Web-app = collection of JSPs + beans

- + factors some of the business logic into beans
- insufficient separation of logics



MVC Model 2 Architecture

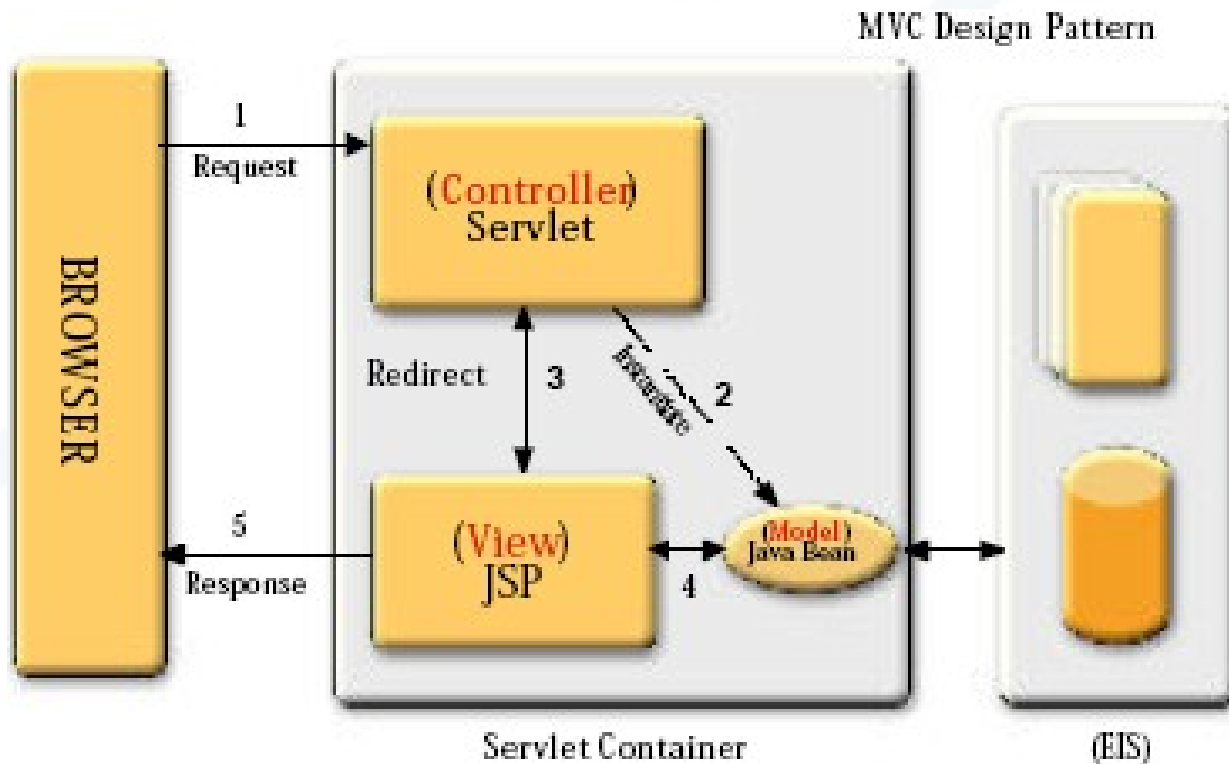


MVC Model 2 Architecture

- **Introduces a Controller**
 - Centralizes logic for Dispatching Requests to the next view based on:
 - The Request URL
 - Input Parameters
 - Application state
 - Single Point of Control
 - For security & logging
 - Encapsulates Incoming Data
 - Into a form usable by the back-end MVC model.

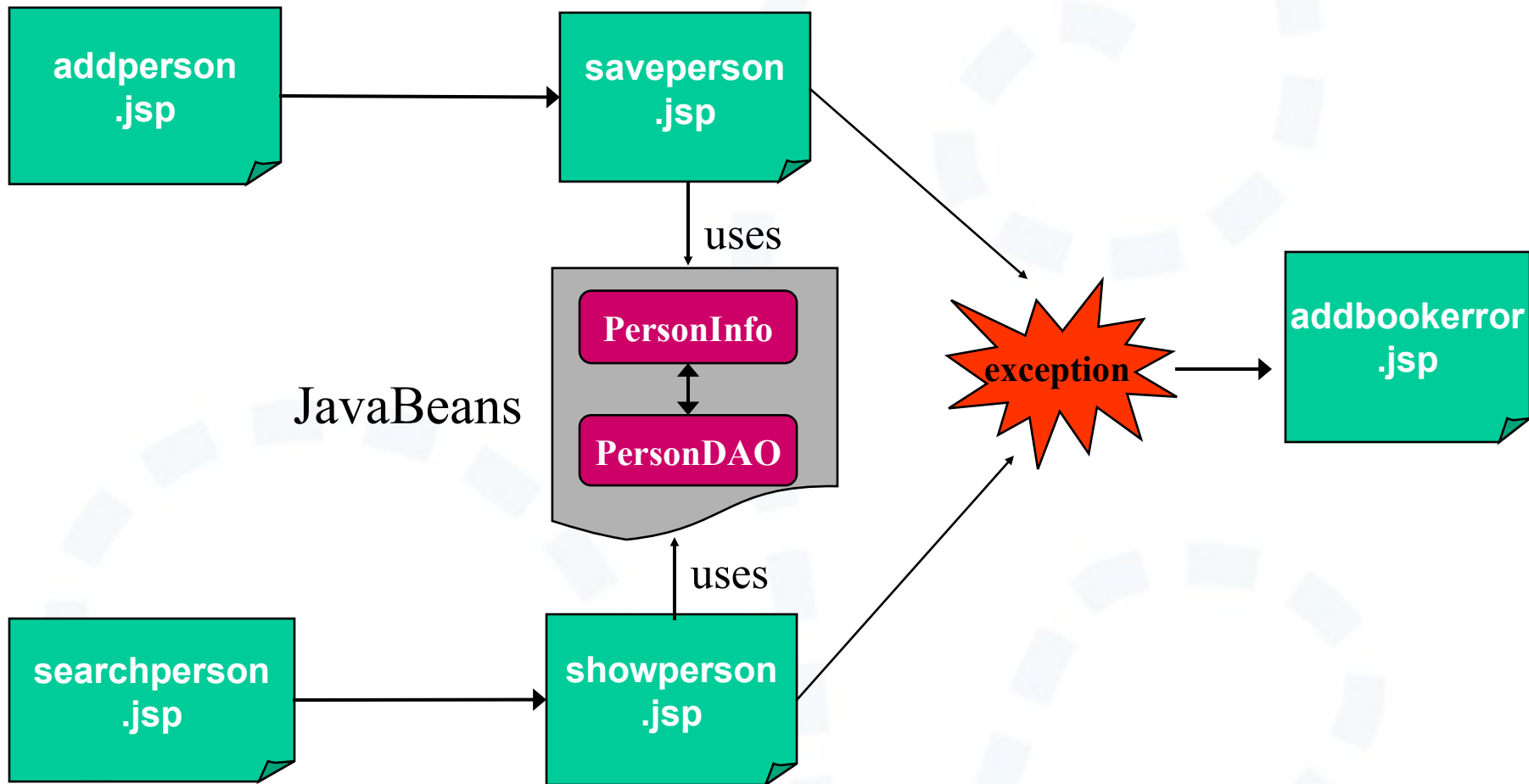


MVC Model 2



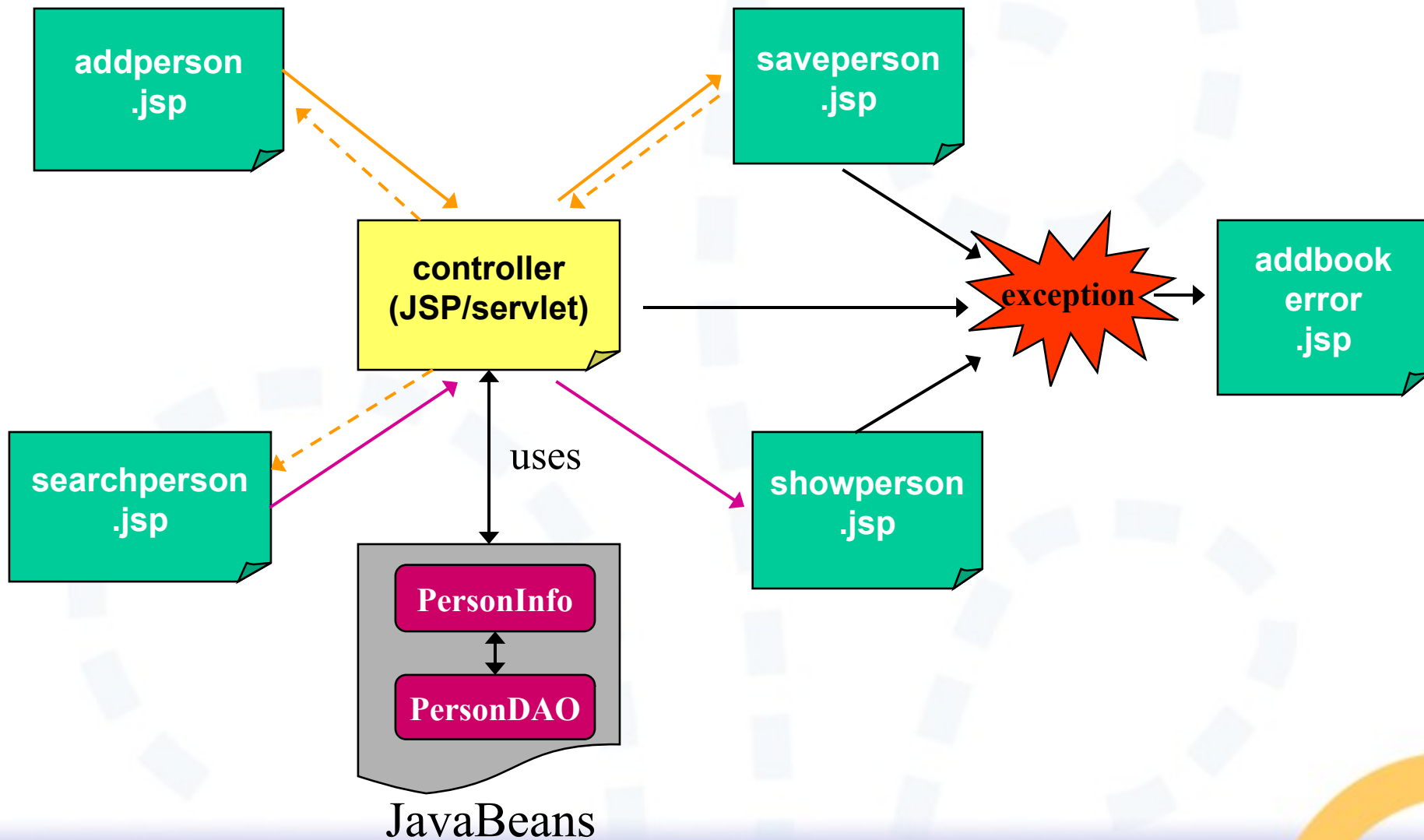


Program Flow (Page with-Bean approach)





Program Flow (Page with-MVC approach)





Case Study: Address Book using MVC Model 2

(JSP as controller)

Addressbookmodel2ex1



Address Book using MVC Model 2 (ni)

- Add another JSP (**controller.jsp**) that
 - Acts as a **controller**
 - **addperson.jsp** & **searchperson.jsp** will submit requests to it.
 - Identifies the page which initiates the request
 - Uses JavaBeans to save or search persons to/from database
 - Forwards or redirects the request to appropriate (**saveperson.jsp** or **showperson.jsp**) page



Give code of both example in
handout (controller as servlet and
JSP)



Introducing Servlet as Controller ? (ni)

- JSP as Controller
 - Doing only processing – no view available
 - Includes logic for selecting pages
 - JSP is really not a good place for such logic
 - JSPs are built for presentation (view) only.
- Solution:
 - Use Servlet as Controller



Pure Servlet

```
public class OrderServlet ... {  
    public void doGet(...) {  
        if(isOrderValid(req)) {  
            saveOrder(req);  
        }  
        ...  
        out.println("<html><body>");  
        ...  
    }  
    private void isOrderValid(...) {  
        ...  
    }  
    private void saveOrder(...) {  
        ...  
    }  
}
```

Servlet

```
Public class OrderServlet ... {  
    public void doGet(...) {  
        ...  
        if(bean.isOrderValid(...)) {  
            bean.saveOrder(req);  
        }  
        ...  
        forward("conf.jsp");  
    }  
}
```

JSP

```
<html>  
  <body>  
    <c:forEach items="${order}">  
      ...  
    </c:forEach>  
  </body>  
</html>
```

Java Bean

```
isOrderValid()  
saveOrder()  
  
-----  
private state
```



Implementing JSP Model 2 (ni)

- As Servlet acts as a Controller
 - Communication from servlet to JSP is necessary
 - **RequestDispatcher** interface can be used to *forward* the request to another Servlet or JSP

```
RequestDispatcher rd =  
    request.getRequestDispatcher("URL");  
rd.forward(request, response);
```



Implementing JSP Model 2 (ni)

- Calling Error Pages from Servlet
 - Servlet can use existing error pages
 - Set the request attribute to `javax.servlet.jsp.JspException` with the exception you want to pass
 - After that, forward the request to the error page



Implementing JSP Model 2 (ni)

- Calling Error Pages from Servlet (cont.)

```
try{  
    ....  
}catch (SQLException sqllex){  
  
    request.setAttribute("javax.servlet.jsp.JspException" , sqllex);  
  
    RequestDispatcher rd = request.getRequestDispatcher("error.jsp");  
    rd.forward(request, response);  
}
```



Case Study: Address Book using MVC Model 2

(Servlet as controller)

netbeans Project: addressbookusingmodel2ex2



Choosing between Model 1 & Model 2



MVC Model 1 (ni)

- Page Centric
- May encourage spaghetti JSP pages
 - Business logic may get lost in display pages
 - Use JavaBeans that captures business logic (instead of scriptlet)
 - Page selection is done by each page
- JSPs are harder to debug than straight java code



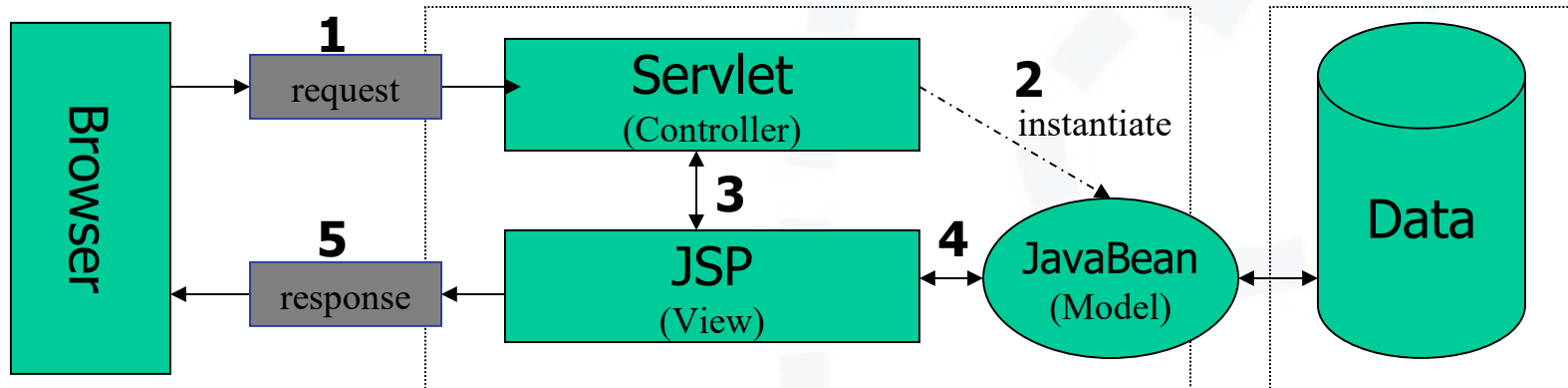
MVC Model 2 (ni)

- Servlet Centric
- Loosens the coupling between the pages and improves the abstraction between presentation and business logic
 - Use JSPs for pure data display and input collection activities
- Most of the business logic can be debugged through the servlet before passed to JavaBeans and JSP



JSP Model 2 Architecture (ni)

Applying MVC Design Pattern



Model 2 binds Servlets and JSP pages to go hand-in-hand using MVC pattern.

Servlet acts as the *Controller* and is in charge of the request processing and creation of any beans or objects (*Models*) used by the JSP.

JSP is working as *View* and there is not much processing logic within the JSP page itself, it is simply responsible for retrieving objects and/or beans, created by the Servlet, extracting dynamic content from them and put them into the static templates.

Layers & Tiers

Lec - 41

Layers & Tiers

- ❑ Layers are merely logical groupings of the software components that make up the application or service.
 - ❑ Whereas tiers refer to the physical residence of those layers (hardware)
 - ❑ In general,
 - **Layers** – represents logical view of application
 - **Tiers** – represents physical view of application
 - ❑ However, both terms are used interchangeably.
-

Layers

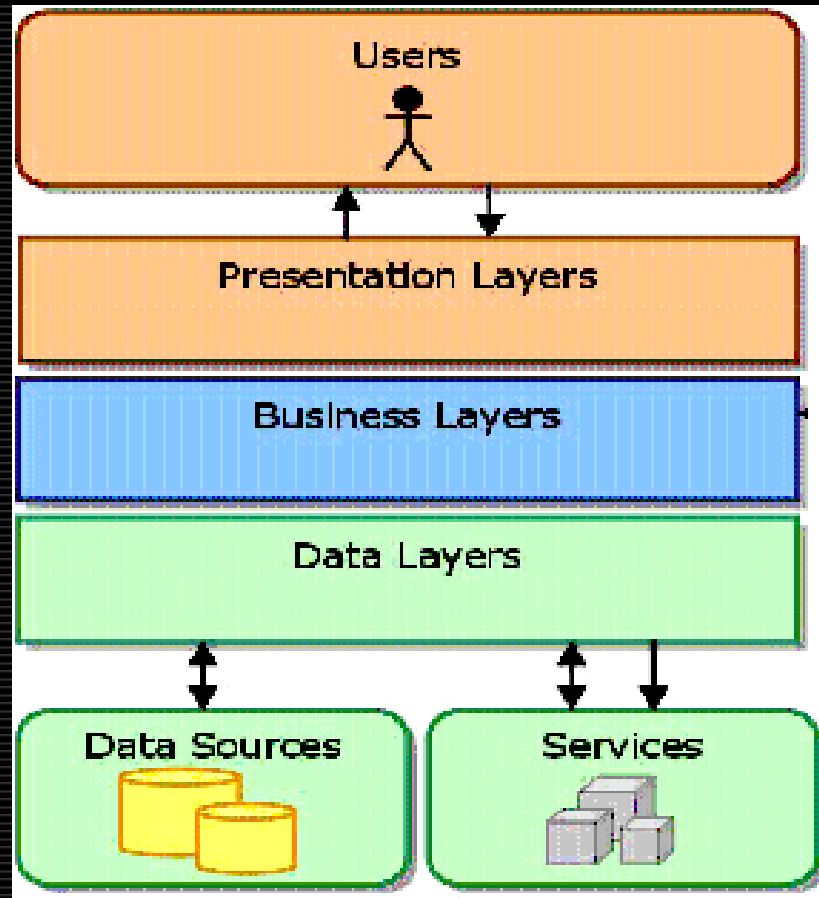
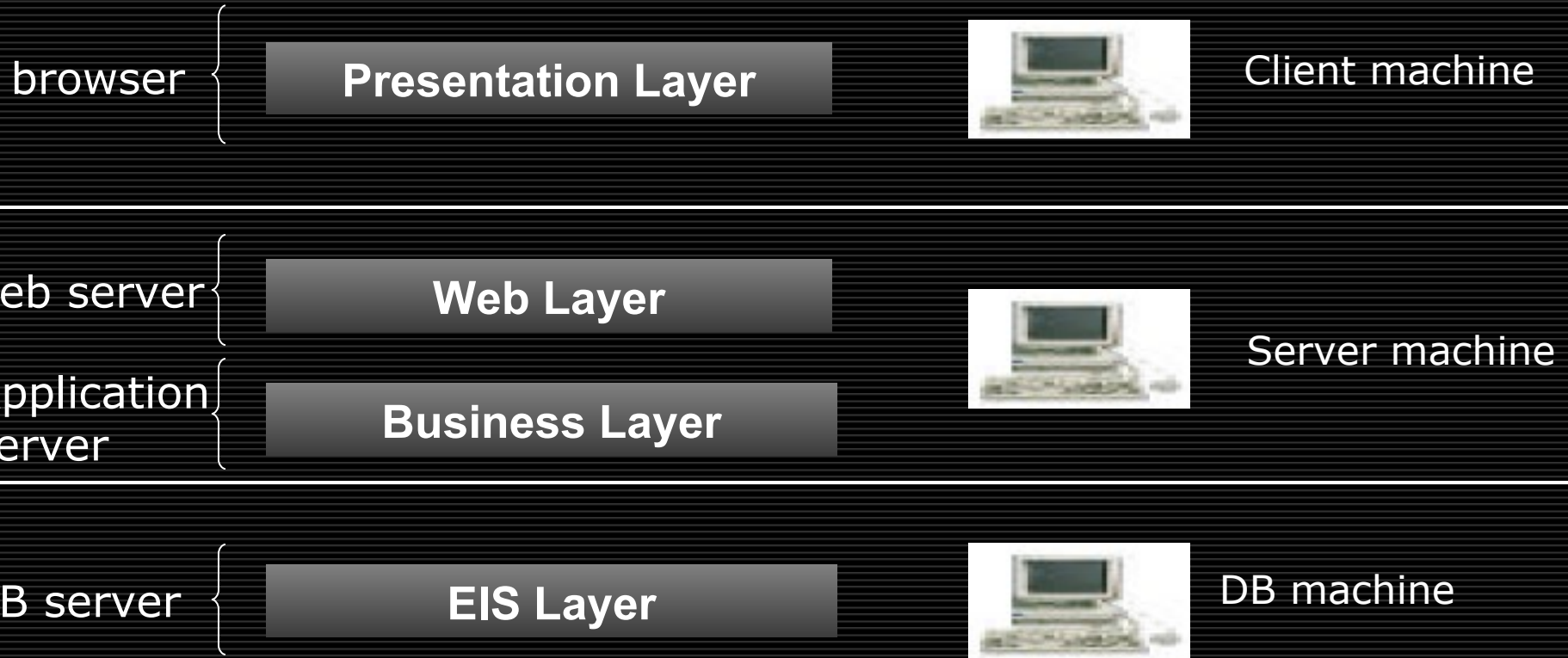


Figure shows a simplified view of one application and its layers.

Tiers



Layers [1]

□ **Presentation layer**

- Collects user inputs
 - Validates user inputs
 - Presents the results of a computation
 - Controls the screen flow
-

Layers [2]

□ **Business Layer**

- Concerned with the application specific functionality
 - Used to implement business rules and to perform business tasks
 - For example, in a banking system, banking functions such as opening an account, calculation of Tax, etc.
-

Layers [3]

□ Data Layer

- Concerned with the management of the data & data sources of the system.
 - Data sources can be database, XML , web services, flat file etc.
 - Encapsulates data retrieval & storage logic
 - For example, the address book application needs to retrieve all person records from a database to display them to the user
-

Layers Support in Java

Layers

Client Presentation

Server Presentation

Business

Data

Java/J2EE Technology

HTML/Applets

**JSP / Servlets
Frameworks(Struts,JSF etc)**

JavaBeans / EJB

DAO / Connectors

Java 2 Enterprise Edition (J2EE)

What is J2EE?

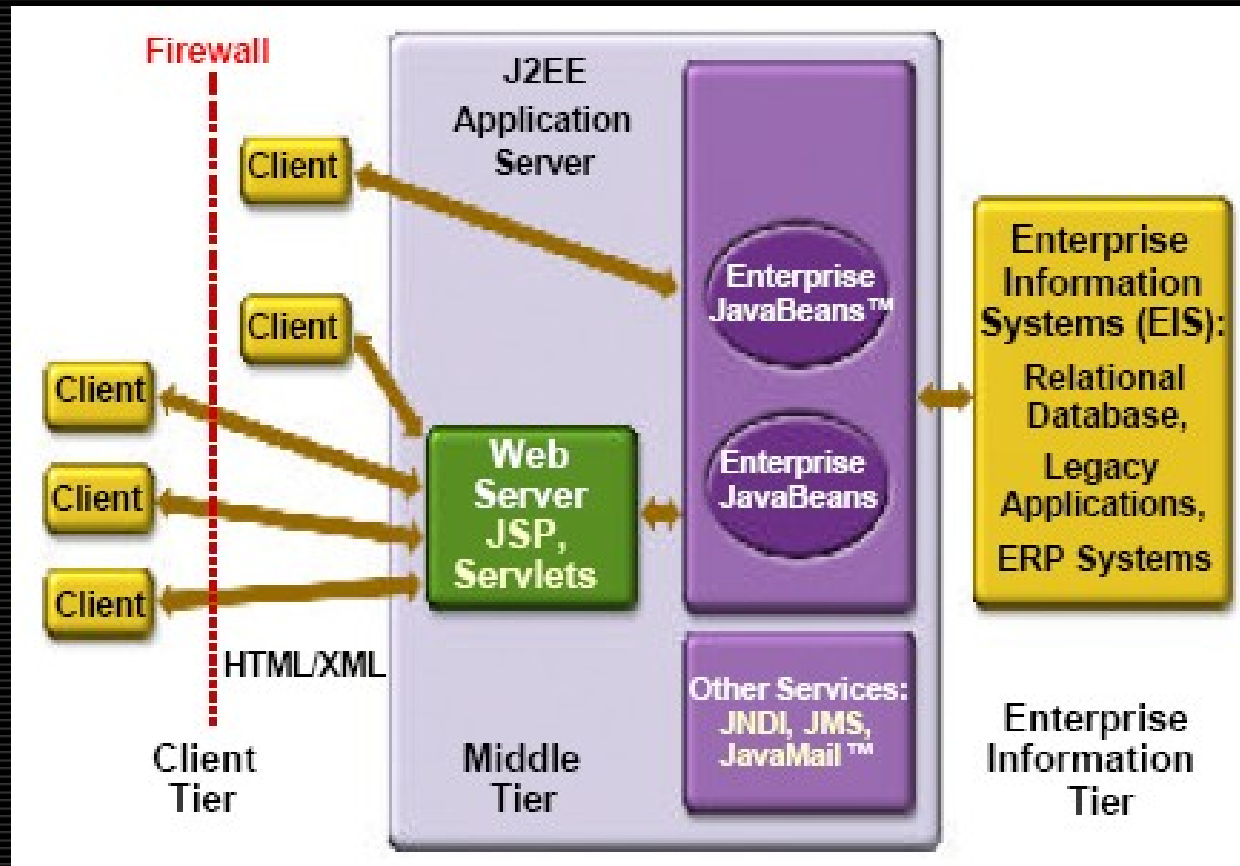
- ❑ Open and standard based platform for:
 - Development & deployment of n-tier web based and component based enterprise applications
 - Reusability, enhance-ability, scalability & interoperability
-

J2EE Suite

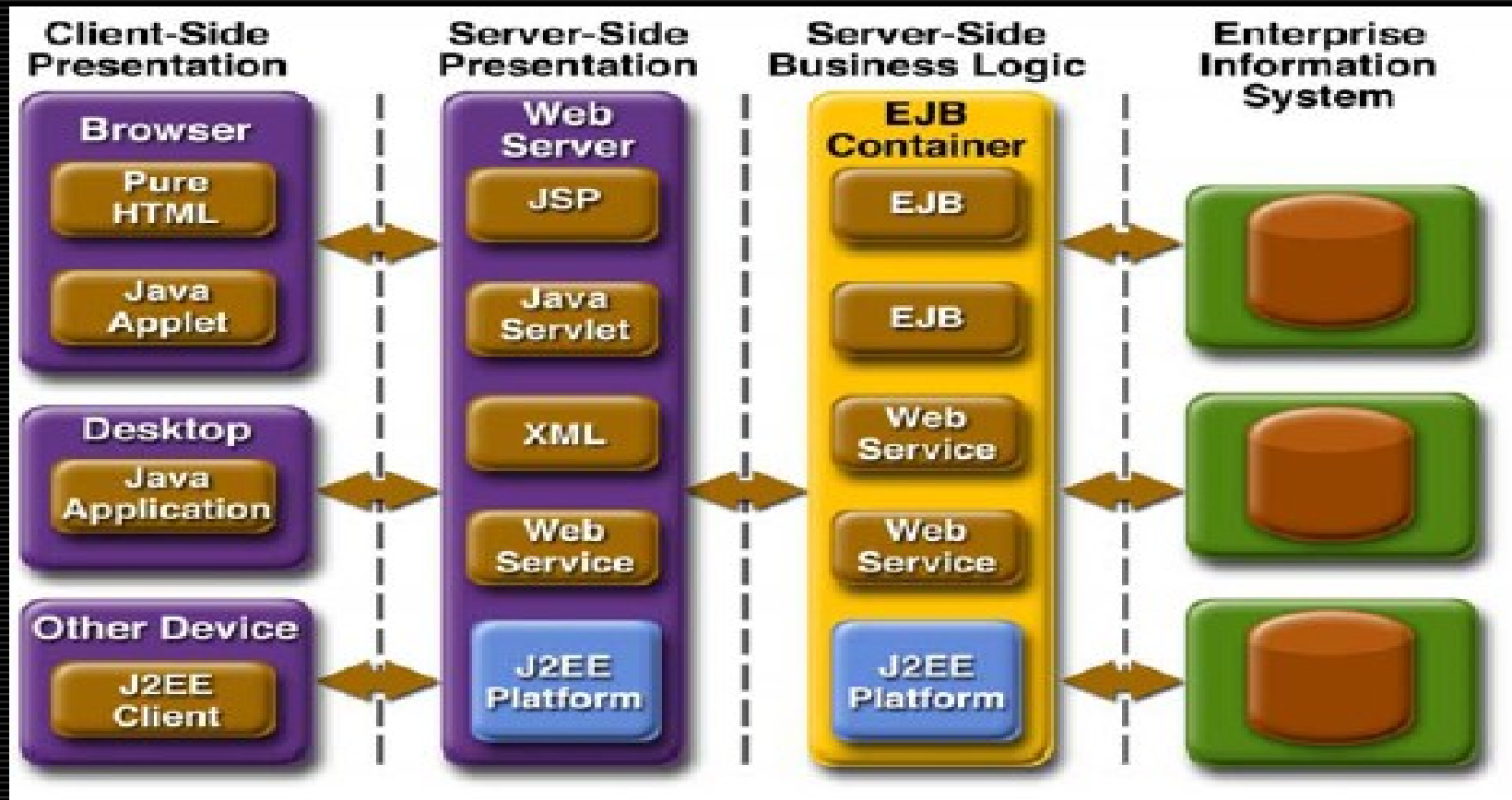
- ❑ **Core technology:**
 - Container infrastructure, language and environment support
 - ❑ **Web Technology**
 - Java Servlets
 - JavaServer Pages (JSP)
 - JavaServer Pages Standard Tag Library (JSTL)
 - JavaServer Faces (JSF)
 - ❑ **Platform services**
 - Security
 - Transactions
 - Resources
 - Connectors
 - Java Message Service
 - ❑ **And extensive support for**
 - XML technology
 - Enterprise Java Bean (EJB) technology etc.
-

J2EE Distributed Multi-tiered Applications

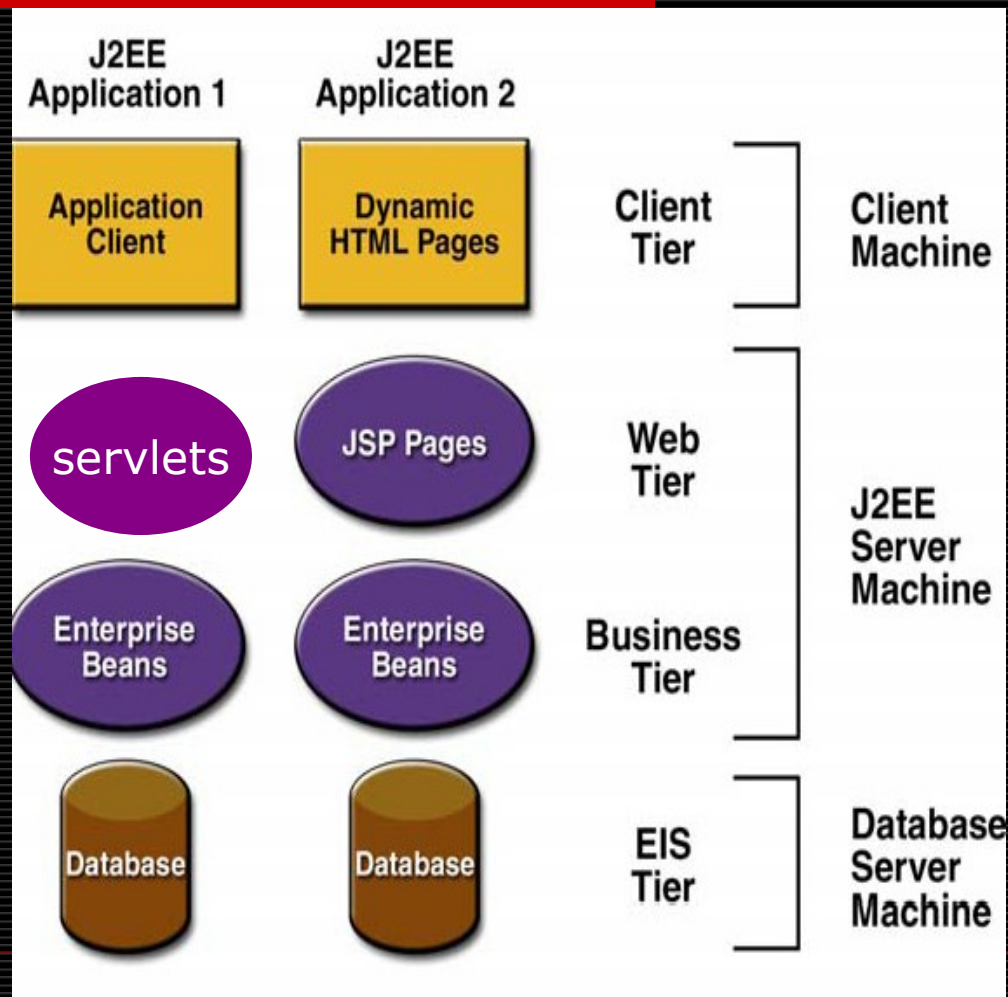
J2EE : Multi-tiered Applications



J2EE : Multi-tiered Applications

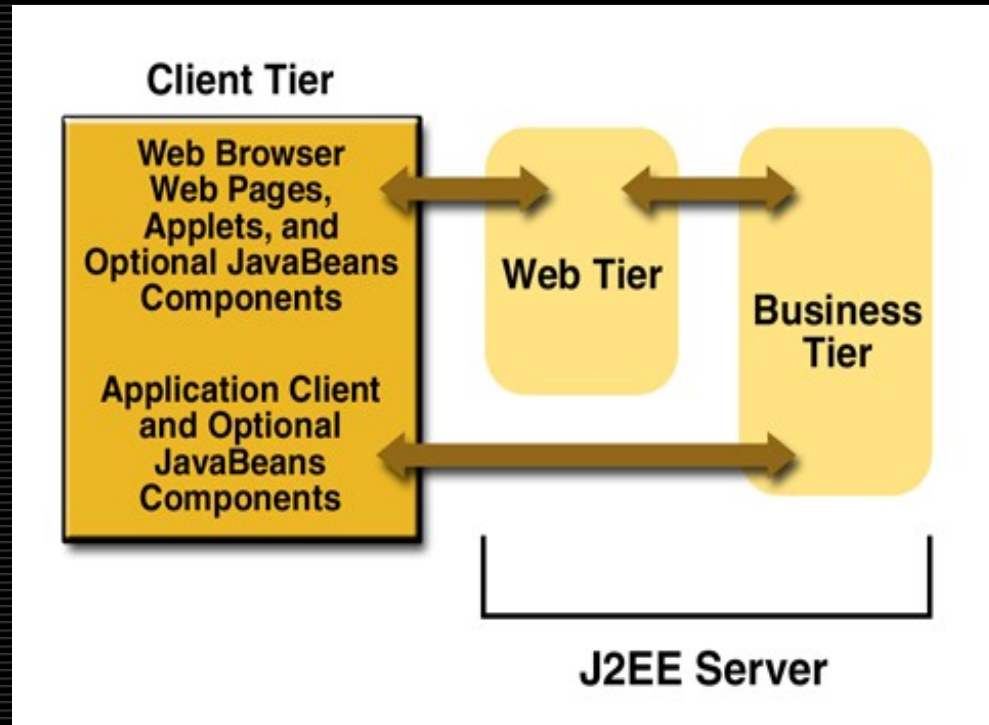


J2EE : Multi-tiered Applications

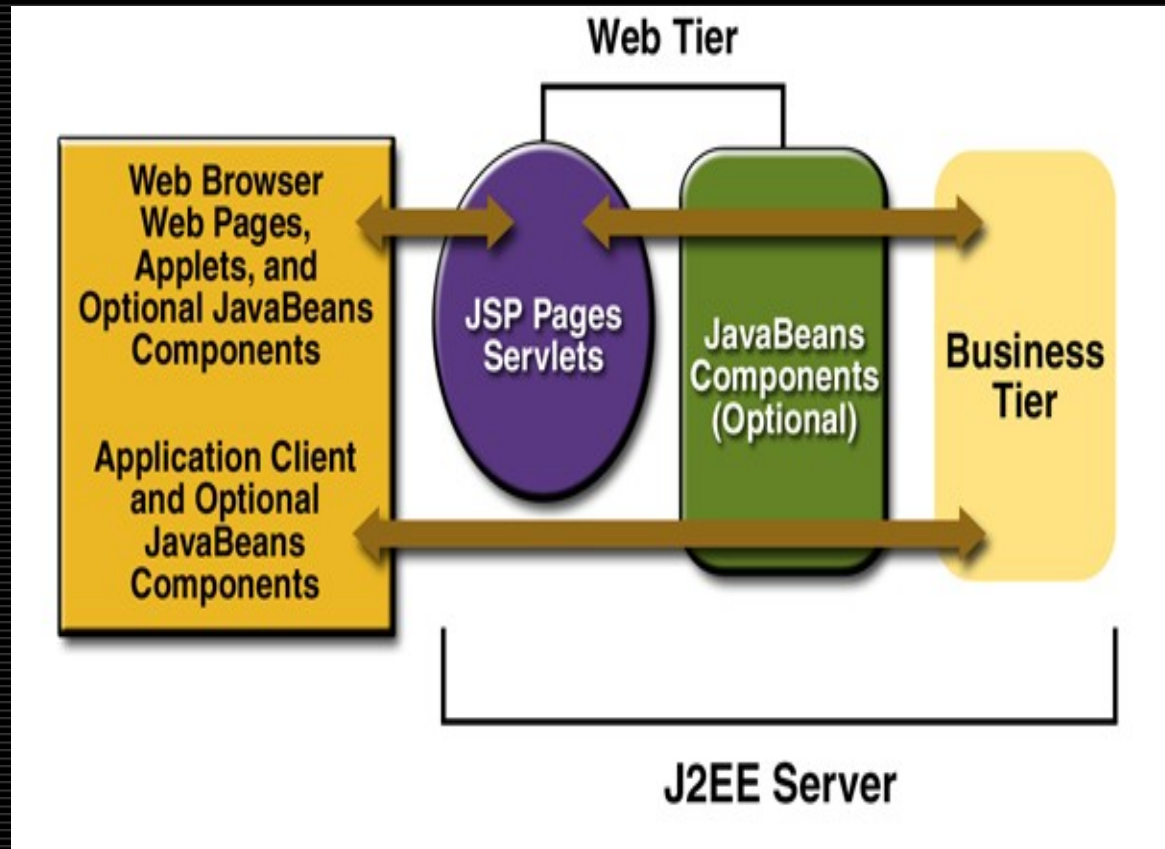


Client Tier

Communication Options



Communication with Business Tier



Case Study: Matrix Multiplication

Using Layers

Case Study: Matrix Multiplication

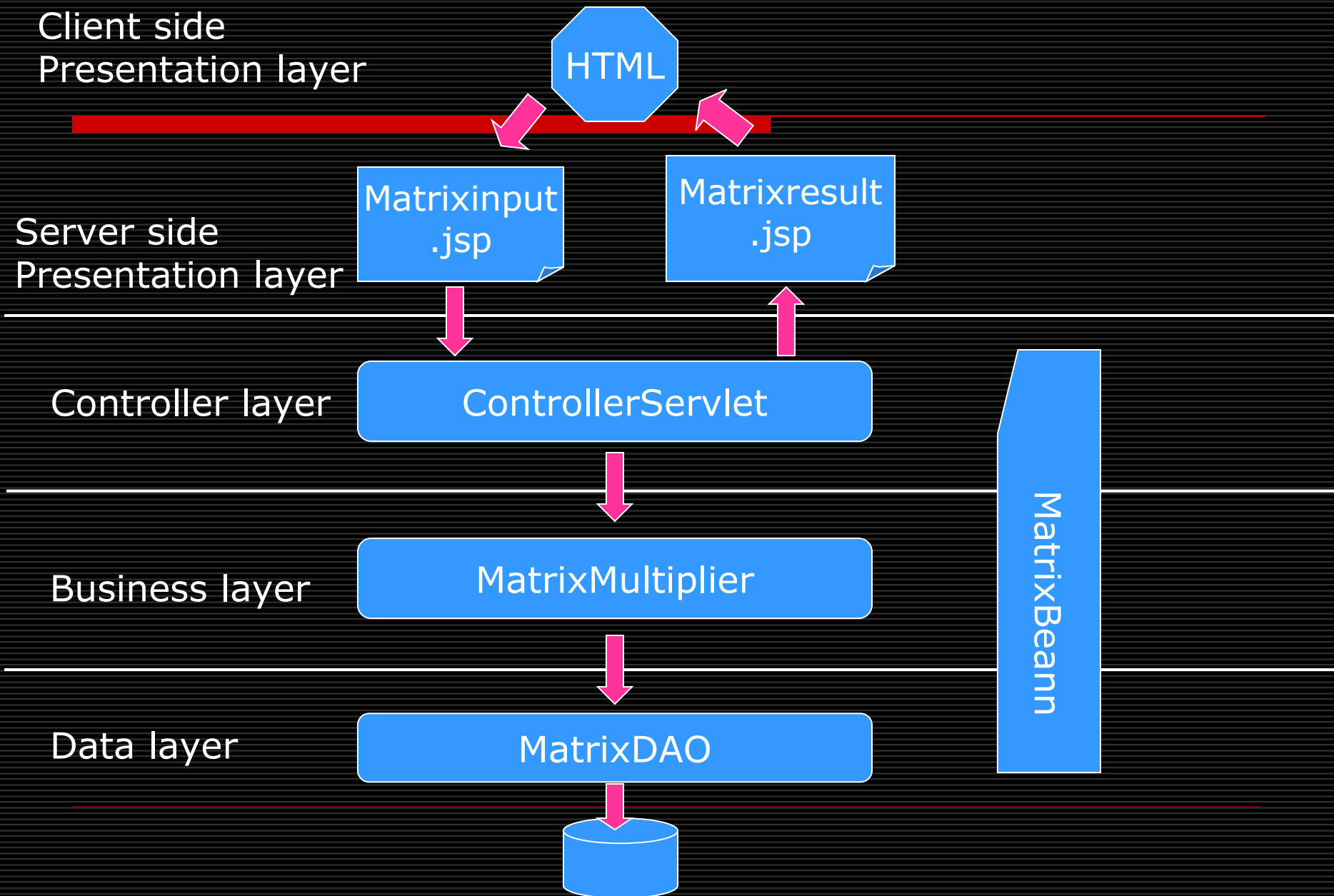
□ Problem Statement

- Calculate product of two matrices of order 2×2
 - Result of multiplication should be stored in DB as well as shown to the user.
-

Case Study: Matrix Multiplication

- Format
 - Input format
 - input will be in 4,2,6,5 format separated by commas where 4,2 represents entries of the first row
 - Display format
 - Displays the matrix as a square
 - Storage format
 - Matrix will be stored as a string
-

Matrix Multiplication: Layer by Layer View



Case Study: Matrix Multiplication

Example Code

netBeans4.1 Project



Expression Language

Lec - 42

Generating Dynamic Contents

- Technologies available
 - Servlets
 - JSP
 - JavaBeans
 - Custom Tags
 - Expression Language (EL)
 - JSTL
 - JavaServer Faces (JSF)



Mike

- Hard code developer
- Handles all business logic and backend matters
- Expert in Java, Database, XML etc.



Ernie

- Jack of all trades
- Not an expert in anything, but will eventually get the job done....



Philippe

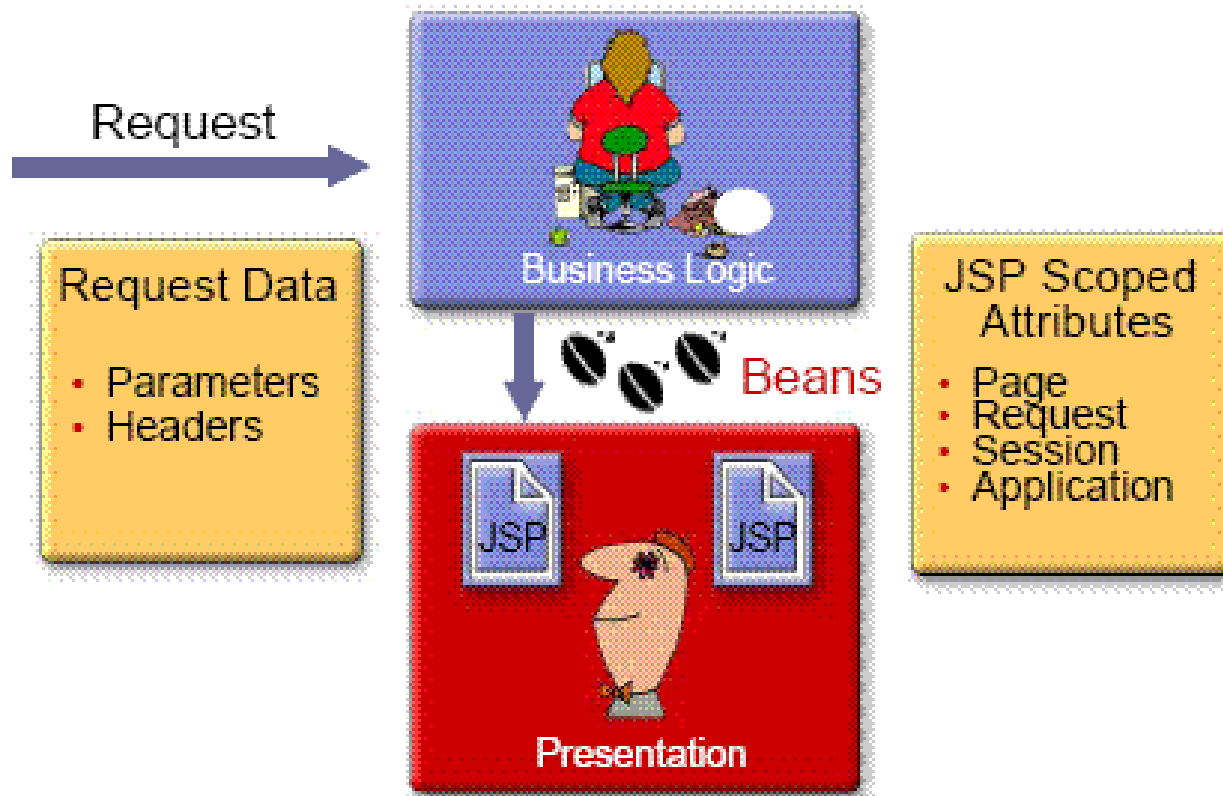
- Web site designer
- Knows how to make a Website look really cool !
- HTML / JavaScript expert

Credit: Pierre Delisle (spec lead)

EL Purpose (ni)

- Java as the scripting language in JSP
scares many people (e.g. Philippe)
- Can we simplify ?
 - Expression Language (EL)
 - A language adapted for the Web Developer

EL Benefits



Credit: Pierre Delisle (spec lead)

EL overview (ni)

- Not a programming or scripting language
- Major goal: **Simplicity** (and really is)
- Inspiration from
 - JavaScript &
 - XML path language (XPath)
- Geared towards looking up objects & their properties and performing simple operation on them

JSP Before EL ...

1. Must Declare

<%

Person p = (Person) request.getAttribute("person")

%>

.....

Person Name: <%= p.getName() %>

.....

<% if (p.getAddress().equals("defence")) { %>

.....

<% } %>

2. Must Know Type

3. Awkward Syntax

4. Knowledge of Scripting Language required even for simple manipulations

JSP After EL ...

1. Direct access

Person Name: `${ p.name }`

2. Easier syntax

...

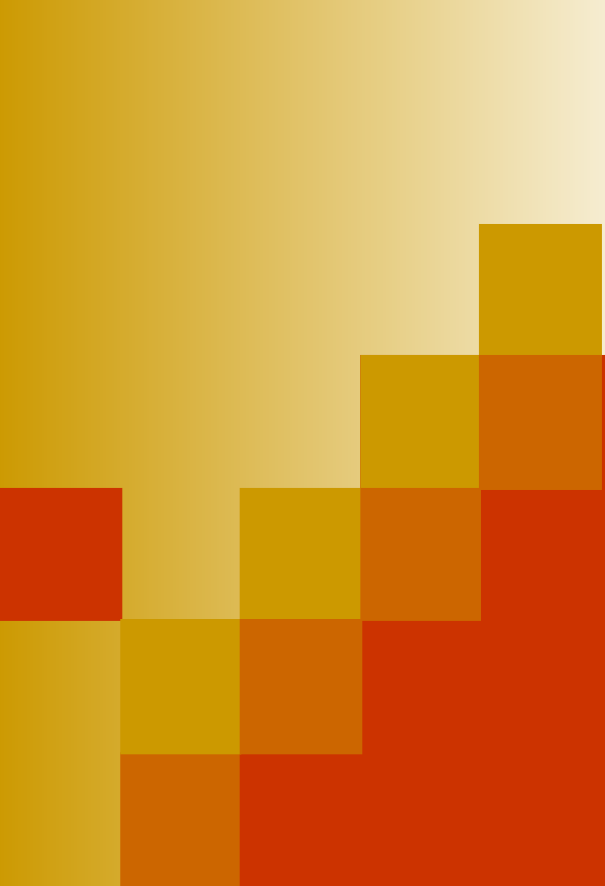
`<c:if test = "${ p.address == param.add }" >`

`${ p.name }`

`</c:if>`

3. All app data easily accessible

4. Better adapted expression language



Expression Language nuggets



EL nuggets

- Expressions & identifiers
- Arithmetic, logical & relational operators
- Automatic type conversion
- Access to beans, arrays, lists & maps
- Access to set of implicit objects & servlet properties

EL Syntax

- Format

$\$ \{ \text{validExpression} \}$

- Valid Expressions

- ☐ Literals
- ☐ Operators
- ☐ Variables (object references)
- ☐ Implicit call to function using property name

EL literals

Literals	Literal Values
Boolean	true or false
Integer	Similar to Java e.g. 243, -9642
Floating Point	Similar to Java e.g. 54.67, 1.83
String	Any string delimited by single or double quote e.g. "hello" , 'hello'
Null	null

EL literals (cont.)

■ Examples

□ `${ false } <%-- evaluate to false --%>`

□ `${ 8*3 } <%-- evaluate to 24 --%>`

EL Operators

Type	Operator
Arithmetic	+ - * / (div) % (mod)
Grouping	()
Logical	&& (and) (or) ! (not)
Relational	== (eq) != (ne) < (lt) > (gt) <= (le) >= (ge)
Empty	prefix operation to determine value is null or empty, returns boolean value
Conditional	? :

EL Operators (cont.)

■ Examples

□ `${ (6*5) + 5 }` `<%-- evaluate to 35 --%>`

□ `${ (x >= min) && (x <= max) }`

□ `${ empty name }`

■ Returns **true** if name is

- Empty string (""),
- Null etc.

EL Identifiers

■ Identifiers

- Represents the name of the object
- Objects stored in JSP scopes (page, request, session, application) referred as **scoped variables**
- EL has **11 reserved** identifiers, corresponding to 11 **implicit objects**
- All other identifiers assumed to refer to scoped variables

EL Identifiers (cont.)

Implicit Objects [1]

Category	Implicit Object	Description
JSP	pageContext	used to access JSP implicit objects
Scopes	pageScope	A <i>Map</i> associating names & values of page scoped attributes
	requestScope	A <i>Map</i> associating names & values of request scoped attributes
	sessionScope	A <i>Map</i> associating names & values of session scoped attributes
	applicationScope	A <i>Map</i> associating names & values of page scoped attributes

EL Identifiers (cont.)

Implicit Objects [2]

Category	Operator	Description
Request Parameters	Param	Maps a request parameter name to a single String parameter value
	paramValues	Maps a request parameter name to an array of values
Request Headers	header	Maps a request header name to a single header value
	headerValues	Maps a request header name to an array of values

EL Identifiers (cont.)

Implicit Objects [3]

Category	Operator	Description
Cookies	cookie	A <i>Map</i> storing the cookies accompanying the request by name
Initialization parameters	initParam	A <i>Map</i> storing the context initialization parameters of the web application by name

EL Identifiers (cont.)

Implicit Objects

■ Examples

□ `${ pageContext.response }`

- evaluates to response implicit object of JSP

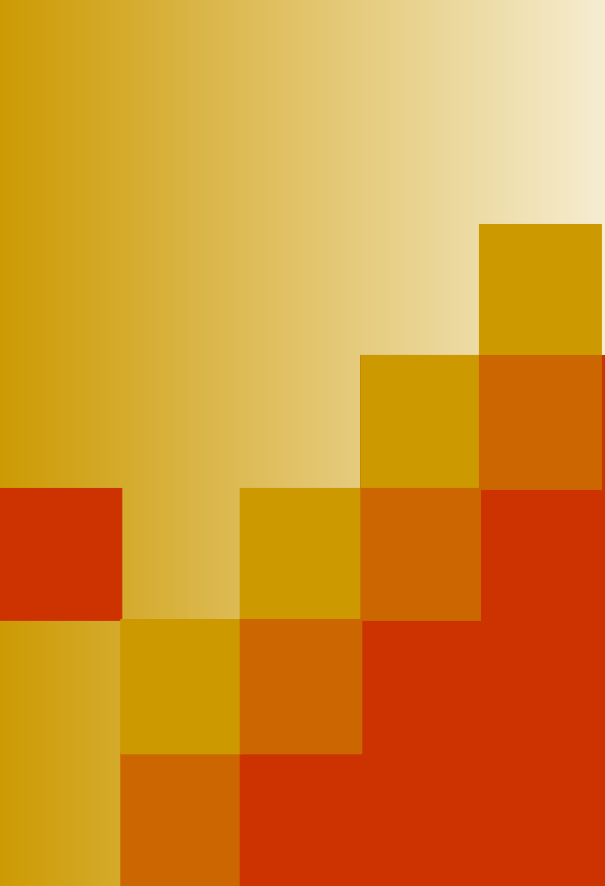
□ `${ param.name }`

- Equivalent to **`request.getParameter("name")`**

□ `${ cookie.name.value }`

- Returns the value of the first cookie with the given name
- Equivalent to

```
if (cookie.getName().equals("name") {  
    String val = cookie.getValue();  
}
```



Example Code:

Summation of two numbers
using EL

netBeans project - elexample

EL Identifiers (cont.)

Scoped Variables

■ Storing Scoped Variables

- `HttpSession ses = request.getSession(true);`
- `Person p = new Person();`
- `P.setName("ali");`
`ses.setAttribute("person" , p);`

```
Person p = new Person();  
P.setName("ali");  
request.setAttribute("person" , p);
```

EL Identifiers (cont.)

Scoped Variables

■ Storing Scoped Variables

□ `<jsp:useBean id="" class="" scope=""`

`<jsp:setProperty name="p"
property="name" value="ali" />`

EL Identifiers (cont.)

(ni)

Scoped Variables

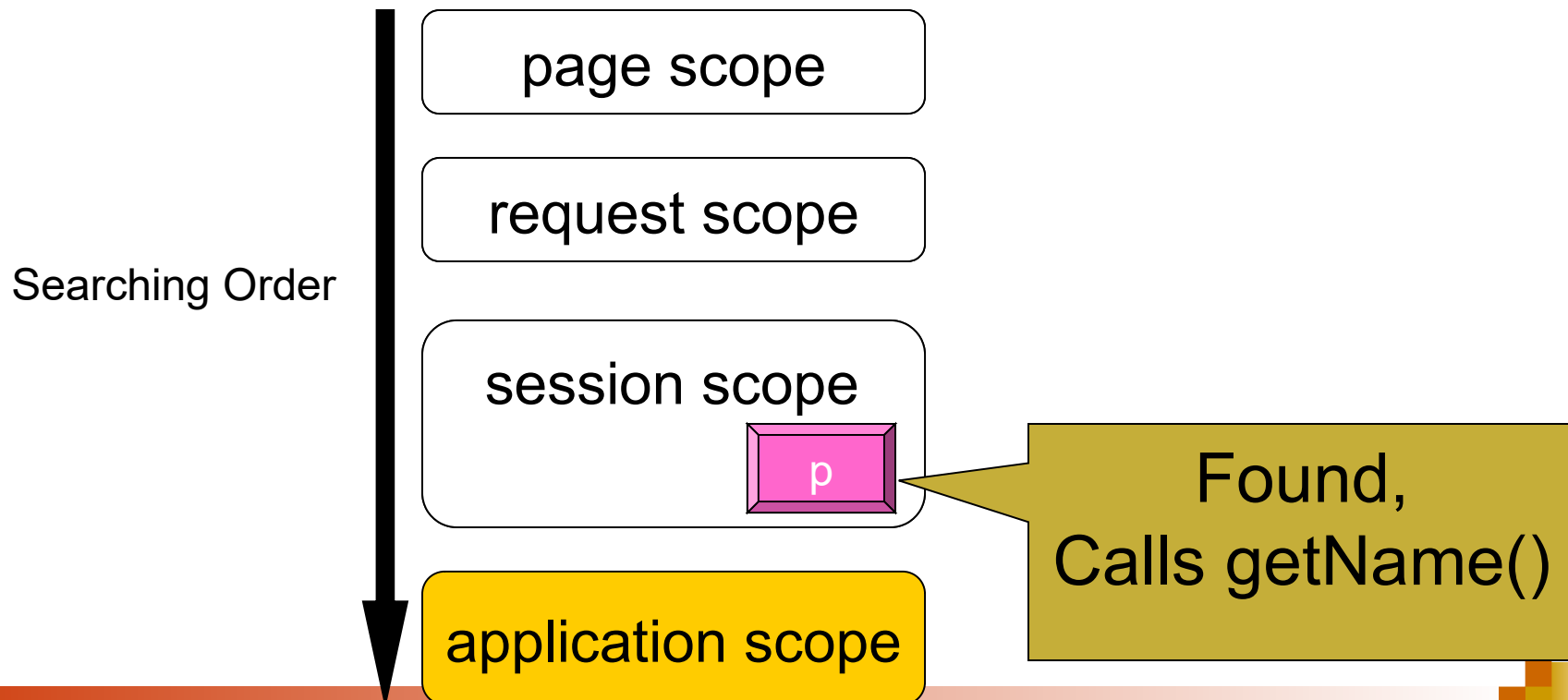
■ Retrieving Scoped Variables

- Any identifier (**not an implicit object**) is first checked against
 - page scope,
 - then request scope,
 - then session scope and
 - finally application scope
- If no such variable is located in four scopes, **null** is returned

EL Identifiers (cont.)

Scoped Variables

- Search for a scoped variable, E.g. `${ p.name }`





No slide insertion after

EL Accessors

- The `.` & `[]` operator let you access identifiers and their properties
- Dot (`.`) operator
 - Typically used for accessing the **properties of an object**
- Bracket (`[]`) operator
 - Typically used to retrieve **elements of arrays & collections**

EL Accessors (cont.)

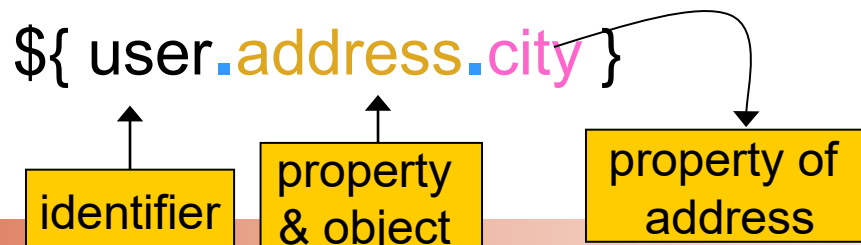
■ Dot (.) operator

- Consider the expression `${ person.name }`



- The EL accesses object properties using the **JavaBeans conventions** (e.g. `getName()` must be defined)

- If property being accessed itself an **object**, the dot operator can be applied recursively. E.g.



EL Accessors (cont.)

■ Bracket ([]) operator

- For arrays & collections implementing **List** interface e.g. ArrayList etc.
 - **Index** of the element appears inside brackets
 - For example, `$ { personList[2] }` returns the 3rd element stored in it
- For collections implementing **Map** interface e.g. HashMap etc.
 - **key** is specified inside brackets
 - For example, `$ { myMap["id"] }` returns the value associated with the id (key)

EL – Robust Features

- Multiple Expression can be combined & mixed with static text

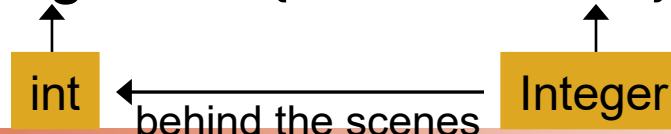
- For example,

`$ { “Hello” ${user.firstName} ${user.lastName} }`

- Automatic type conversion

- EL can automatically wrap and unwrap primitives in their corresponding Java classes. E.g.

`begin = “${ student.marks }”`



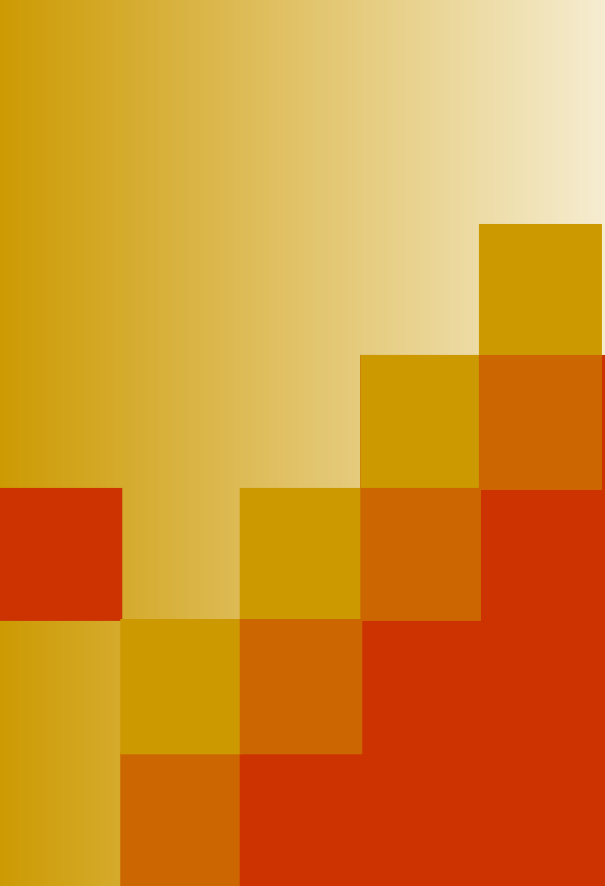
EL – Robust Features (cont.)

■ No NullPointerException 😊

- If the object/identifier is **null**

 - For example `${person.name}`, e.g. if person is **null**

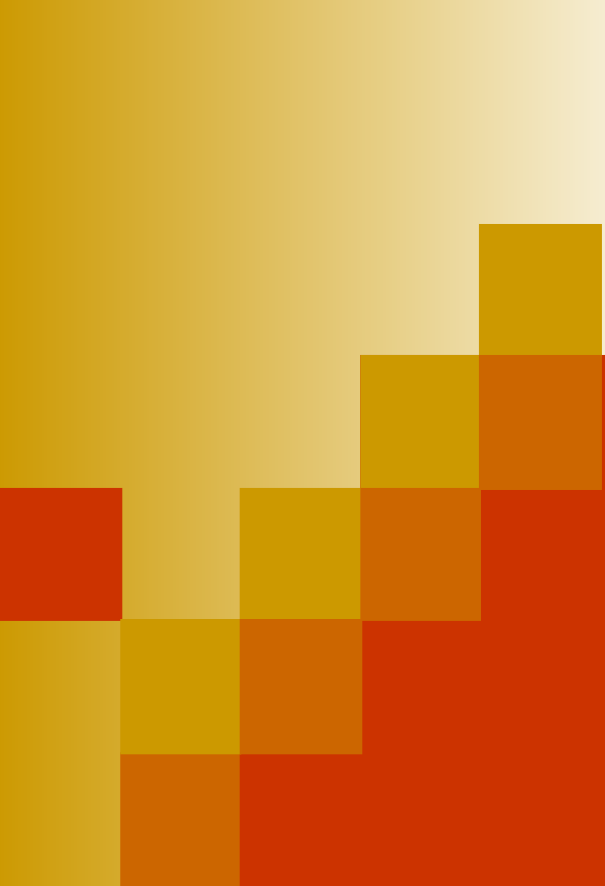
- No exception would be thrown – the **result would also be null**



Using Expression Language

Using EL Expressions

- Can be used in following situations
 - As attribute values in standard & custom actions. E.g.
`<jsp:setProperty id = "person" value = ${....} />`
 - In template text – the value of the expression is inserted into the current output. E.g.
`<h3> $ { .... } </h3>`
 - With JSTL (discussed shortly)



Example Code: Address book using EL

netBeans project -
ExpressionLanguageExample



JSTL

Lec - 43

JSTL (ni)

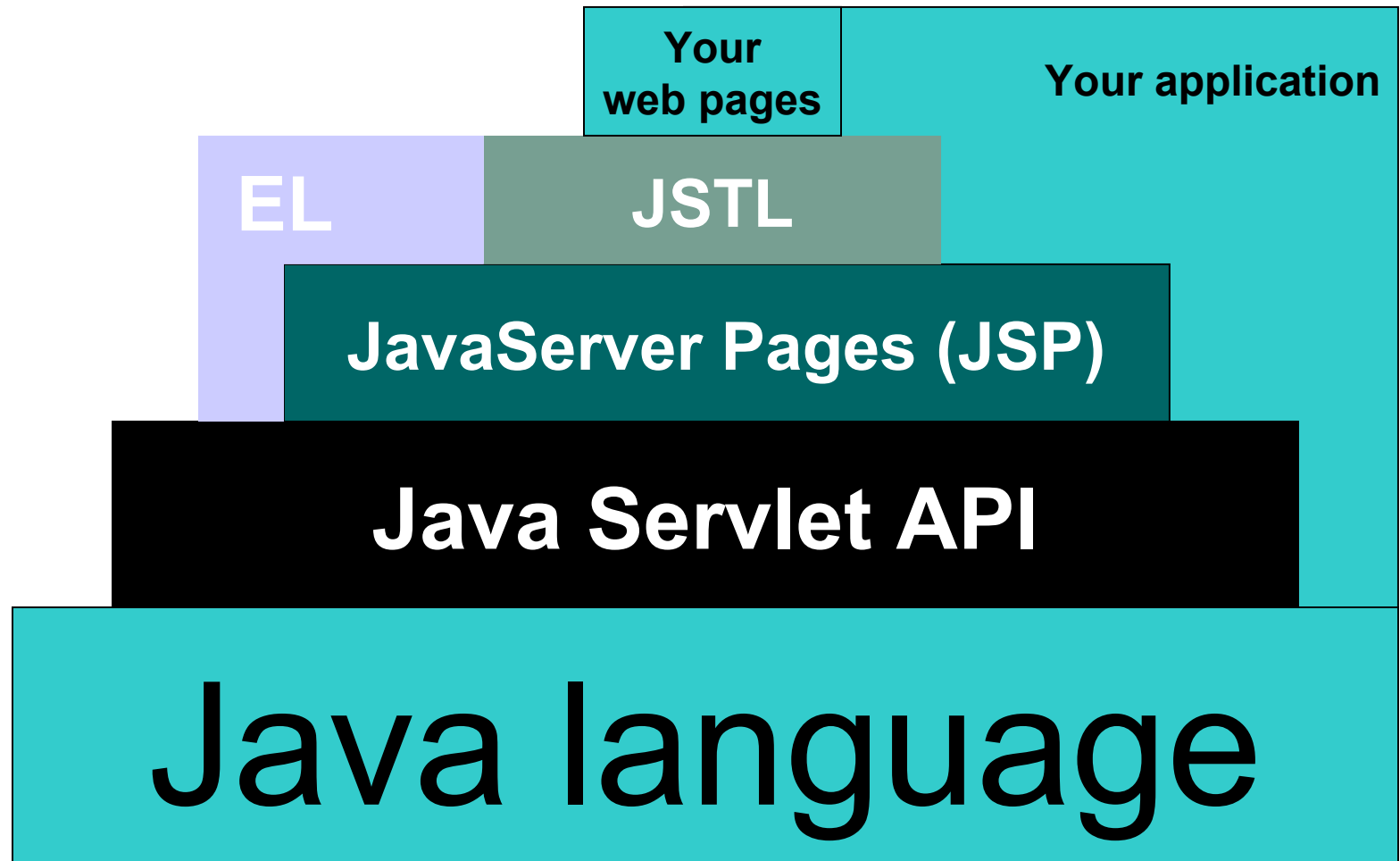
- Acronym of
 - JavaServer Pages Standard Tag Library
- JSTL (like JSP) is a specification, not an implementation
- Official reference implementation
 - <http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html>
- Development theme
 - No Scriptlets

JSTL Overview (ni)

- JSTL tags are valid XML
- JSTL contains action (tags) for common tasks
 - Iteration
 - Session Tracking
 - Redirect
 - XML
 - SQL etc.

Remember – the
development theme
is “no scriptlets”

Organization of the platform



Why JSTL? (ni)

- JSTL tags provide a standard implementation for typical application functionality
 - Reusability
 - Avoid **reinventing the wheel**
- Another mechanism for avoiding the use of JSP scripting elements
 - EL considerably simple than Java

JSTL & EL (ni)

- JSTL includes support for EL
 - To replace the JSP expressions
- EL is also standard part of JSP 2.0
- EL can only be used in attributes of JSTL tags prior to JSP 2.0
 - With JSP 2.0 and onwards, it can be used anywhere in the document

JSTL Tag libraries

- Composed of 4 standard tag libraries
 - **Core**
 - Conditions, Control flow & variables access etc.
 - **Internationalization / format**
 - Locale messages
 - Text, numbers & date formation
 - **XML**
 - XML parsing / processing
 - **SQL**
 - Tags for accessing an SQL database

Twin Tag Libraries

- JSTL comes in two flavors
 - Request Time (RT) version
 - Dynamic attribute values are specified using **JSP expression (i.e. <%= **expression** %>)**
 - Expression Language (EL) version
 - Dynamic attribute values are specified using **JSTL expression language (i.e. \${ **expression** })**
- Why ?

JSTL Tag libraries (cont.)

- EL based

Library	URI	Prefix
Core	http://java.sun.com/jsp/jstl/core	c
SQL	http://java.sun.com/jsp/jstl/sql	sql
Internationalization / Format	http://java.sun.com/jsp/jstl/fmt	fmt
XML	http://java.sun.com/jsp/jstl/xml	x

JSTL Tag libraries (cont.)

- RT based

Library	URI	Prefix
Core	http://java.sun.com/jsp/jstl/core_rt	c_rt
SQL	http://java.sun.com/jsp/jstl/sql_rt	sql_rt
Internationalization / Format	http://java.sun.com/jsp/jstl/fmt_rt	fmt_rt
XML	http://java.sun.com/jsp/jstl/xml_rt	x_rt

Using JSTL

- Use **taglib** directive, as we used for custom tag library
- For example
 - To use **EL based core** tag library
`<%@taglib prefix = "c" uri = http://java.sun.com/jsp/jstl/core %>`
 - To use **RT based core** tag library
`<%@taglib prefix = "c_rt" uri = http://java.sun.com/jsp/jstl/core_rt %>`



Working with Core Actions (tags)



Core Tag Library

- Support actions
 - ☐ Manipulation of scoped variables
 - ☐ Output
 - ☐ Conditional logic
 - ☐ loops
 - ☐ URL manipulation
 - ☐ and Handling errors.

Core Tag Library

■ **c:set** [1]

- Provides a tag based mechanism for creating and setting scope based variables

- **Syntax:**

`<c:set var="name" scope = "scope" value = "expression" />`

- **var** → Specifies the name of the scoped variables
- **scope** → page | request | session | application
→ Optional and defaults to **page**
- **value** → Specifies the value to be bound to the variable
→ If evaluates to **NULL**, attribute/var will be **removed** if exist.

Core Tag Library

■ **c:set** [2]

□ Examples

```
<c:set var="timezone" value="Asia / Karachi" />
```

```
<c:set var="email" scope="request" value="me@gmail.com" />
```

```
<input type="text" name="email" />
```

```
<c:set var="email" scope="page" value="${param.email}" />
```


Core Tag Library

■ Using c:set for JavaBeans & Map [1]

□ Syntax:

```
<c:set target="beanOrMap" property="propertyOrKey" value="value" />
```

Target must not
be NULL

If target is a bean, sets the value
of the property, equivalent to
<jsp:setProperty()>

If target is a Map, sets the value
of the key

Core Tag Library

■ Using c:set for JavaBeans [2]

□ Example

```
<jsp:useBean id="person" class="vu.PersonInfo"  
scope="request" />
```

```
<c:set target="person" property="name" value="ali" />
```

Core Tag Library

■ **c:out** [1]

- Equivalent to JSP expression i.e. `<%= expression %>`

- **Syntax:**

`<c:out value="expression" default = "expression" />`

- **value** → Evaluates the value attribute and outputs the result as **string**

- **default** → Prints it if the **value** attribute evaluates to **null** or **empty string**

→ Optional

Core Tag Library

■ **c:out** [2]

□ Examples:

```
<c:out value = "Hello" />
```

```
<c:out value = "${param.num}" default = "0" />
```

Is equivalent to:

```
<%
```

```
String no = request.getParameter("num");
```

```
if (no == null )
```

```
    System.out.println(no);
```

```
%>
```

```
<c:out value= "${person.name}" default = "Not Set" />
```

Core Tag Library

■ **c:remove**

- Used to delete a scoped variable

- **Syntax:**

```
<c:remove var= "name" scope = "scope" />
```

- **Examples:**

```
<c:remove var= "square" />
```

```
<c:remove var= "email" scope = "request" />
```

Core Tag Library

■ **c:forEach** [1]

- Used for iteration purpose
- Supports two different styles of iteration
 - **Iteration over an integer range**
 - **Iteration over a collection**

Core Tag Library

- **c:forEach** [2] (ni)

- **Iteration over an Integer Range** [1]

- Like java language's **for statement**

- **Syntax:**

- ```
<c:forEach var="name" begin="expression"
end="expression" step="expression" />
```

**Body Content**

**</c:forEach>**

**If omitted, by default 1**

# Core Tag Library

- **c:forEach** [3] (ni)

- **Iteration over an Integer Range** [2]

- **Example:** To generate squares corresponding to range of integer values

```
<c:forEach var="x" begin="0" end="10" step="2" />
```

```
<c:out value="{x * x}" />
```

```
</c:forEach>
```



# Core Tag Library

## ■ c:forEach [4]

### □ Iteration over a Collection [1]

- Can loop down on arrays, strings, list & map etc.
- Syntax:

```
<c:forEach var="name" items="expression" />
```

Body Content

```
</c:forEach>
```

- ArrayList
- HashMap
- Arrays etc.

# Core Tag Library

## ■ **c:forEach** [5]

### □ **Iteration over a Collection** [2]

- **Example:** To iterate over a String array

```
<%
for(int i=0; i<messages.length; i++)
<c:forEach var="message" items="${messages}" >
 <c:out value="${message}" />
</c:forEach>
=>
```

```
<%= message %>
<%
 } // end for
%>
```

JSP after JSTL

JSP

JSTL

# Core Tag Library

## ■ c:forEach [6]

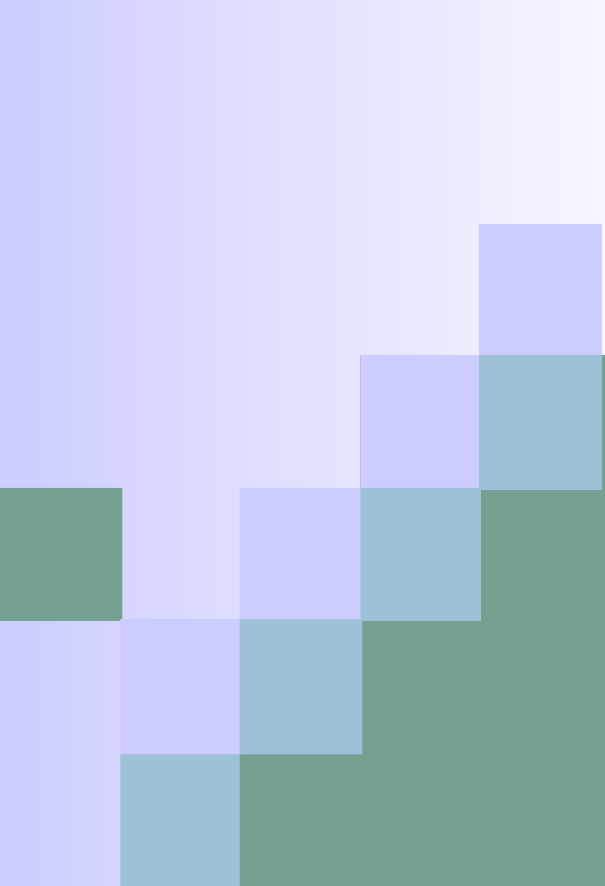
### □ Iteration over a Collection [3]

- **Example:** To iterate over a **persons** ArrayList, contains **PersonInfo** objects

```
<%
 ArrayList persons = (ArrayList)request.getAttribute("pList")
 for(int i=0; i<persons.size(); i++) {
 <c:forEach var="p" items="{persons}" >
 <c:out value="{p.name}" />
 </c:forEach>
 } // end for
%>
```

**Automatic Type Conversion**  
*to appropriate type*

**JSP after JSTL**



# Example code

Addressbook(MVC) using  
core tags

netBeans project – jstl\_ex2

# Core Tag Library (ni)

## ■ **c:if**

- Used to conditionally process the body content

- **Syntax:**

```
<c:if test= "expression" />
```

Body content

```
</c:if>
```

- **Example:**

```
<c:if test= "${a==b}" />
```

a equals b

```
</c:if>
```

# Core Tag Library (ni)

## ■ **c:choose** [1]

- Enables mutually exclusive conditionals

- **Syntax:**

**<c:choose>**

**<c:when test= "expression" >**

Body content

**</c:when>**

.....

**<c:otherwise >**

Body content

**</c:otherwise>**

**</c:choose>**

-- Must appear at-least once

-- Only one **<c:when>** is processed  
whose test evaluates to true

-- Can appear at-most once

-- Only execute if all **<c:when>**  
tests evaluate to false

# Core Tag Library (ni)

## ■ **c:choose** [2]

□ Example:

**<c:choose>**

**<c:when test= “\${a == b}” />**  
a equal b  
**</c:when>**

**<c:when test= “\${a == c}” />**  
a equal c  
**</c:when>**

**<c:otherwise />**  
Don't know what 'a' equal  
**</c:otherwise>**

**</c:choose>**



# Working with SQL Actions (tags)





# SQL Tag Library

- Support actions
  - ☐ To interact with relational databases
  - ☐ Issuing queries & updates
  - ☐ transactions etc.

# SQL Tag Library

## ■ **sql:setDataSource** [1]

- Used for specifying data source

- Syntax:

```
<sql:setDataSource
 driver="driver_name"
 url = "url"
 user = "user"
 password = "pwd"
/>
```

# SQL Tag Library

## ■ **sql:setDataSource** [2]

- **Example:** To connect with Microsoft Access database

```
<sql:setDataSource
```

```
 driver = "sun.jdbc.odbc.JdbcOdbcDriver"
```

```
 url = "jdbc:odbc:PersonDSN"
```

```
 user = ""
```

```
 password = ""
```

```
/>
```

# SQL Tag Library

## ■ **sql:query**

- Used to execute queries

- Syntax:

**<sql:query**

**sql = “expression”**

**var = “name”**      -- Results of query stored

**scope = “scope”**      -- Optional attribute to specify  
where to store **var**

**/>**

# SQL Tag Library

## ■ sql:query [2]

- **Example:** executing a simple select query & processing results

```
<sql:query sql = "SELECT * FROM PersonInfo"
 var = "res"
/>
```

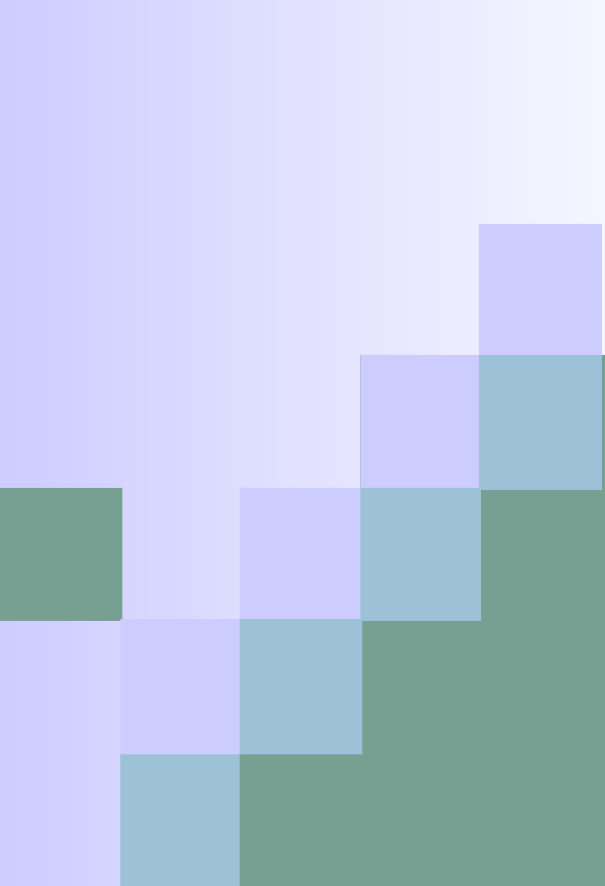
```
<c:forEach var = "row" items="${res.rows}" >
```

```
 <c:out value = "${row.name}" />
```

```
 <c:out value = "${row.address}" />
```

```
</forEach>
```

Contains results of query



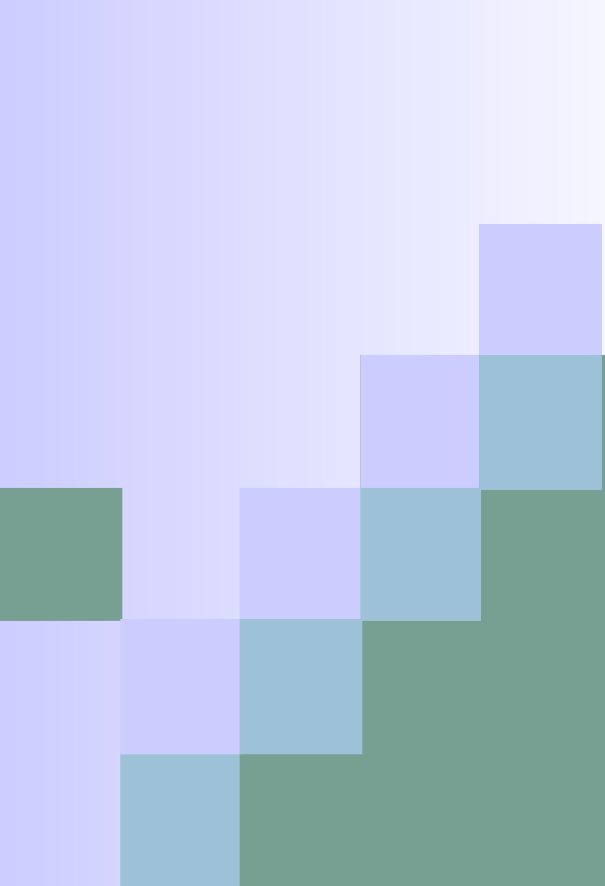
# Example code

Addressbook(MVC) using  
core & sql tags  
netBeans project – jstl\_ex

# SQL Tag Library (ni)

## ■ Problems

- Heinous violation of MVC design pattern
  - DB code (i.e. raw SQL) doesn't belong in the presentation layer

A decorative graphic on the left side of the slide, consisting of a grid of squares in shades of purple, blue, and green, arranged in a stepped pattern.

Insert slide that  
contains only jsp  
pages – addressbook  
using only isp pages



# Bringing it all together (lec 45 slide)

- Java provides these mechanisms for internet programming
  - Applets
  - Servlets
  - Java Server pages
    - Scripting elements
    - JSP Standard Tag library
    - JSTL Expression language
    - Java Beans
    - (Enterprise Java Beans)
- Ultimately leads to easier, faster, and more powerful web application development?



## JSF road map (NI)

- Clientside validation
- Show problems
  - State example (search page don't show back results)
  - Event Model (swing example , now tags data need to be parsed)
  - Validation model
  - We are doing a lot of low work ourselves (e.g. request.getParameter etc). Need a framework



# Client Side validation Example



(NI)

- Discuss problems here
- Problems with servlets and JSPs
- Servlet and JSP
  - Provide no direct GUI component support
  - No mechanism to manipulate stateful objects at the server
  - No way to auto-connect client events to server methods
  - Requires programming skill
    - Low level details of HTTP and session
    - Undefined programming model – lots of tedious code
  - State example (search page don't show back results)
  - Event Model (swing example , now tags data need to be parsed)
  - Validation model
  - We are doing a lot of low work ourselves (e.g. request.getParameter etc).  
Need a framework



# Intro to Framework

- Framework vs. API
- Different existing frameworks
  - Struts
    - Helps define a structured programming model (MVC), a validation framework and reduces tedious coding But...
      - Adds complexity and doesn't provide UI tags
      - Very Java programmer centric
  - Tapestry
  - JSF



# JavaServer Faces

A close-up, blue-tinted image of interlocking mechanical gears, serving as a background for the lower half of the slide.

A new face on application  
development in Java



# JSF Architecture & Technology (NI)

## Architecture

**UI Component Model**

**Model Object Integration**

**Rendering Model**

**Server Side UI Event Handling**

**Type Conversion & Validation**

**Navigation**

**Internationalization**

## Technology

**A Set Of UI Components**

**APIs And Programming  
Model**

**A JSF Custom Tags**



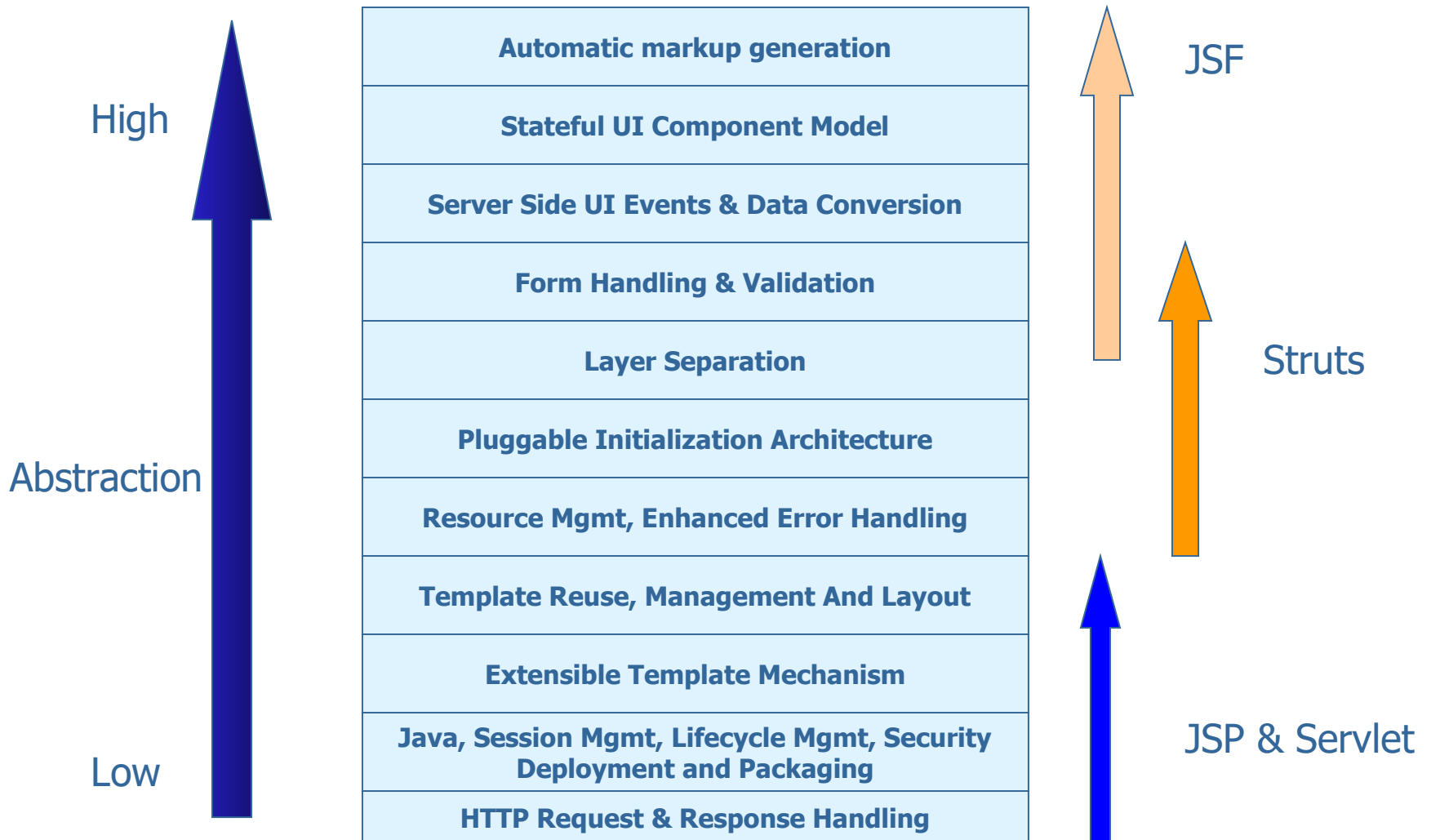
# Java Server Faces – Major Features (NI)

- Components
  - Allows creation of user interfaces from a set of standard, reusable server-side components
    - Provides JSP tags to access those components
  - Allows component rendering to support multiple markups and device types
  - Provides a framework for implementing custom components
- Easier Programming Model
  - Transparently saves state information and repopulates forms when they redisplay
  - **Provides a mechanism for tying client side events to server side logic / processing**
    - Components available to scripts on server
  - Contains mechanisms for validation and conversion
  - Separates presentation from logic
- Enables more functional “RAD” Tooling





# JSF – Web Application Infrastructure (NI)





# What is JSF?

- A framework which provides solutions for
  - representing UI components
  - managing their state
  - handling events
  - input validation
  - Data binding →
  - Automatic conversion →
  - defining page navigation
  - supporting internationalization and accessibility.



## Enhanced Productivity for UI design (NI)

- Page level RAD
- Allows building web pages in a manner very similar to Visual Basic, PowerBuilder, or Domino Designer
  - Provides a component model
  - Allows users to think about components, **events** and scripting instead of the details of HTTP requests / responses
- Competes directly with MS .Net WebForms



# UI Components (Standard)

Some of the standard JavaServer Faces Components

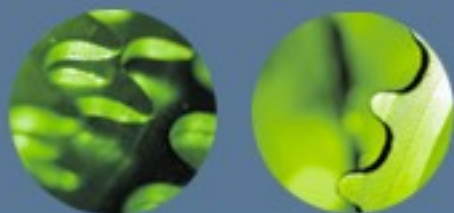
| Application Field Group                                                         |                      |
|---------------------------------------------------------------------------------|----------------------|
| <input checked="" type="radio"/> New Group <input type="radio"/> Existing Group |                      |
| New Group                                                                       | <input type="text"/> |

| Application Field Type                                                                    |                                                                                 |
|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <input type="radio"/> New Field Type <input checked="" type="radio"/> Existing Field Type |                                                                                 |
| Name                                                                                      | <input type="text" value="Cell Phone Number"/> <input type="button" value="v"/> |

| Application Field Display                                                       |                                                                                                                                                                                                                                                                        |      |  |     |                                     |     |                          |       |                          |
|---------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|--|-----|-------------------------------------|-----|--------------------------|-------|--------------------------|
| <input type="text" value="Select Checkboxes"/> <input type="button" value="v"/> | <table border="1"><thead><tr><th>Name</th><th></th></tr></thead><tbody><tr><td>One</td><td><input checked="" type="checkbox"/></td></tr><tr><td>Two</td><td><input type="checkbox"/></td></tr><tr><td>Three</td><td><input type="checkbox"/></td></tr></tbody></table> | Name |  | One | <input checked="" type="checkbox"/> | Two | <input type="checkbox"/> | Three | <input type="checkbox"/> |
| Name                                                                            |                                                                                                                                                                                                                                                                        |      |  |     |                                     |     |                          |       |                          |
| One                                                                             | <input checked="" type="checkbox"/>                                                                                                                                                                                                                                    |      |  |     |                                     |     |                          |       |                          |
| Two                                                                             | <input type="checkbox"/>                                                                                                                                                                                                                                               |      |  |     |                                     |     |                          |       |                          |
| Three                                                                           | <input type="checkbox"/>                                                                                                                                                                                                                                               |      |  |     |                                     |     |                          |       |                          |



# UI Components (Custom)

## Some custom JavaServer Faces Components

| < January > 2005 |    |    |    |    |    |    |
|------------------|----|----|----|----|----|----|
| Su               | Mo | Tu | We | Th | Fr | Sa |
| 26               | 27 | 28 | 29 | 30 | 31 | 1  |
| 2                | 3  | 4  | 5  | 6  | 7  | 8  |
| 9                | 10 | 11 | 12 | 13 | 14 | 15 |
| 16               | 17 | 18 | 19 | 20 | 21 | 22 |
| 23               | 24 | 25 | 26 | 27 | 28 | 29 |
| 30               | 31 | 1  | 2  | 3  | 4  | 5  |

### JSF Messages:

#### Application Map:

|                                              |                                                                                                                                                                                                                                                                                           |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| weblogic.servlet.WebAppComponentMBean        | [Caching Stub]Proxy for mydomain:ApplicationConf                                                                                                                                                                                                                                          |
| com.sun.faces.HTML_BASIC                     | com.sun.faces.renderkit.RenderKitImpl@1911e61                                                                                                                                                                                                                                             |
| weblogic.servlet.WebAppComponentRuntimeMBean | weblogic.servlet.internal.WebAppRuntimeMBeanIn<br>mydomain:ApplicationRuntime=myserver__apps<br>ui_dir,Location=myserver,Name=myserver_myse<br>ui,ServerRuntime=myserver,Type=WebAppComp<br>mydomain:Location=myserver,Name=myserver__<br>ui_dir,ServerRuntime=myserver,Type=ApplicationR |
| com.sun.faces.ApplicationAssociate           | com.sun.faces.application.ApplicationAssociate@                                                                                                                                                                                                                                           |
| javax.servlet.context.tempdir                | C:\bea\user_projects\domains\mydomain\myserve<br>ui_dir_accessweb-ui\public                                                                                                                                                                                                               |
| com.sun.faces.OneTimeInitialization          | com.sun.faces.OneTimeInitialization                                                                                                                                                                                                                                                       |

#### Session Map:

|                             |                                                  |
|-----------------------------|--------------------------------------------------|
| dsa_notify                  | com.ihc.issa.accessweb.ui.dsa.NotificationMgrBes |
| javax.faces.request.charset | UTF-8                                            |
| /AdminApplication.jsp       | javax.faces.component.UIViewRoot@119e003         |
| admin_template              | com.ihc.issa.accessweb.ui.admin.TemplateBean     |

ApplicationMap: ApplicationMap

SessionMap: SessionMap



# UI Components (Open Source)

Some open source JavaServer Faces Components

4 - 6 of 7



| ID | First Name | Family Name |
|----|------------|-------------|
| 4  | Hans       | Mueller     |
| 5  | Amadeus    | Mozart      |
| 6  | Harry      | Potter      |



|     |      |               |      |     |     |     |   |
|-----|------|---------------|------|-----|-----|-----|---|
| <   |      | February 2004 |      |     |     |     | > |
| Mon | Tues | Wed           | Thur | Fri | Sat | Sun |   |
| 26  | 27   | 28            | 29   | 30  | 31  | 1   |   |
| 2   | 3    | 4             | 5    | 6   | 7   | 8   |   |
| 9   | 10   | 11            | 12   | 13  | 14  | 15  |   |
| 16  | 17   | 18            | 19   | 20  | 21  | 22  |   |
| 23  | 24   | 25            | 26   | 27  | 28  | 29  |   |

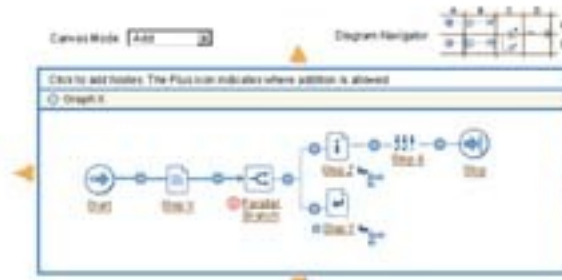


# UI Components (Third Party)

Some third-party JavaServer Faces Components

Show All Details | Hide All Details | Previous | 1-3 of 6 | Next 3

| Details              | Name     | Age    |
|----------------------|----------|--------|
| ▼ Hide               | Sally    | 24     |
| Detailed Information |          |        |
|                      | Name     | Sally  |
|                      | Age      | 24     |
|                      | Title    | Doctor |
|                      | Patients | 30     |
|                      | Hospital | UCSF   |
| ▶ Show               | Willy    | 25     |
| ▶ Show               | Kelly    | 20     |



136-255-136

Root > Computer > Hardware > CDROM

Product Name that spans 2 lines

Home Shopping Cart Cart Items

Processing: Baking the cake

Using processingLayout element.

In this demo we use the processingLayout element. It is intended to make it easier to create the BLAF Processing Template.

Cancel

Cancel

Cancel

▶ UX Components

▼ UX Controller

- Architecture
- Release

▶ Architecture

▶ Release



Home Help

Finance

Go

Switch Applications





# JSF Events (NI)

- Event notification and listener based on JavaBean 1.0.1
- Events are fired by each UI component
- Event handlers are registered with each component
- Three standard events
  - Value Change Event – generates by UIInput component
  - Action Event – generates by UICommand component
  - Phase Event – fire by JSF life cycle
- Custom events can easily be created and integrated into JSF





# JSF Events – Action Event

## Action Listener:

```
<h:commandButton value="Login"
 actionListener="#{customer.loginActionListener}"
 action="#{customer.login}" />
```

```
public void loginActionListener(ActionEvent e) {

}
```

```
public String login() {
 return "OK";
 // return "FAI";
}
```



- Lets do it
- Hello user example



## **JSF Events – Listener vs Action (NI personal)**

- Listener Handlers
  - Implement UI logic
  - Have access to event source
  - Do not participate in navigation handling
- Action Handlers
  - Implement business logic
  - Don't have access to action source
  - Returned outcome affects the navigation handling



# JSF – Multiple Event Handlers (NI)

```
<h:selectOneMenu value="#{customer.country}"
 <f:valueChangeListener type="com.comp.CntrListener"
 <f:valueChangeListener type="com.comp.CCListener"
</h:selectionOneMenu>
```

```
<h:commandButton action="#{search.doSearch()}">
 <f:actionListener type="com.comp.AAciontListener" />
 <f:actionListener type="com.comp.BActionListener" />
</h:commandButton>
```



- Example Hello User



## **JSF Validators (NI)**

- For validating user input. (You can use client side validation as well we are telling server side process – delete it)
- 0 or more validators can be registered with an UIInput component
- Standard validators and custom validator



# JSF Validators

- **DoubleRangeValidator**
  - Any numeric type, between specified maximum and minimum values
- **LongRangeValidator**
  - Any numeric type convertible to long, between specified maximum and minimum values
- **LengthValidator**
  - String type, between specified maximum and minimum values



# JSF Validators (NI)

Required Validation Example:

```
<h:inputText value="#{user.id}" required="true" />
```

Length Validation Example:

```
<h:inputText value="#{user.password}" >
 <f:validateLength minimum="6" />
</h:inputText>
```





## Validation (NI)

- If validation or conversion fails nothing happens
  - Action method bindings do not execute
  - Page just comes back
  - Most common JSF forum post
  - Use `h:message` or `h:messages`



- Example Hello User with required + six character validation



# What is JSF?

- A framework which provides solutions for
  - representing UI components
  - managing their state
  - handling events
  - input validation
  - Data binding →
  - Automatic conversion →
  - defining page navigation
  - supporting internationalization and accessibility.



## **JSF – Managed Bean-Intro**

- Use to separate presentation from business logic
- Based on JavaBeans
- Use the declarative model
- Entry point into the model and event handlers
- Can have beans with various states



## JSF – Value Binding (NI)

- Bind component value and attribute to model objects

### Literal:

```
<h:outputText rendered="true" value="$1000.00"/>
```

### Value Binding:

```
<h:outputText rendered="{user.manager}"
value="{employee.salary}"/>
```



# JSF – Value Binding

- Value binding expression
  - Bean properties
  - List
  - Array
  - Map
  - Predefine objects- header, header values, request parameters, cookie, request/session/application scope attributes, initial parameters



# JSF – Method Binding

- Binding an event handler to a method

```
<h:commandMethod action="#{user.login}" />
```

- Four component attributes:

- Action
- Action listener
- Value change listener
- Validator



# JSF Converters

- Type conversion between server-side objects and their representation in markup language
- Standard converter implementations
  - DateTime
  - Number
- Custom convert – implements Converter interface
  - Object `getAsObject(...)`
  - String `getAsString(...)`





# JSF Converters

Number converter example:

```
<h:inputText value="#{rent.amt}" converter="Number">
 <f:attribute name="numberStyle" value="currency" />
</h:inputText>
```

Date convert example:

```
<h:inputText value="#{rent.dueDate}" converter="DateFormat">
 <f:attribute name="formatPattern" value="MM/DD" />
</h:inputText>
```



# JSF Navigation

- JSF provides a default navigational handler
- Behavior is configured in configuration file (faces-config.xml)
- You can do it visually in most tools



# JSF Navigation - Example

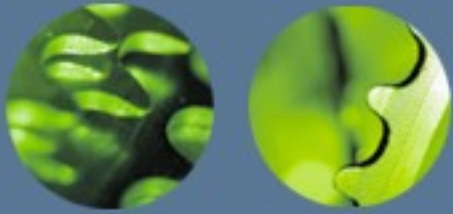
```
<navigation-rule>
 <description>LOGIN PAGE NAVIGATION HANDLING</description>
 <from-view-id> /login.jsp </from-view-id>

 <navigation-case>
 <description>Handle case where login succeeded.</description>
 <display-name>Successful Login</display-name>
 <from-action>#{userBean.login}</from-action>
 <from-outcome>success</from-outcome>
 <to-view-id>/home.jsp</to-view-id>
 </navigation-case>

 <navigation-case>
 <description>User registration for a new user succeeded.</description>
 <display-name>Successful New User Registration</display-name>
 <from-action>#{userBean.register}</from-action>
 <from-outcome>success</from-outcome>
 <to-view-id>/welcome.jsp</to-view-id>
 </navigation-case>
</navigation-rule>
```



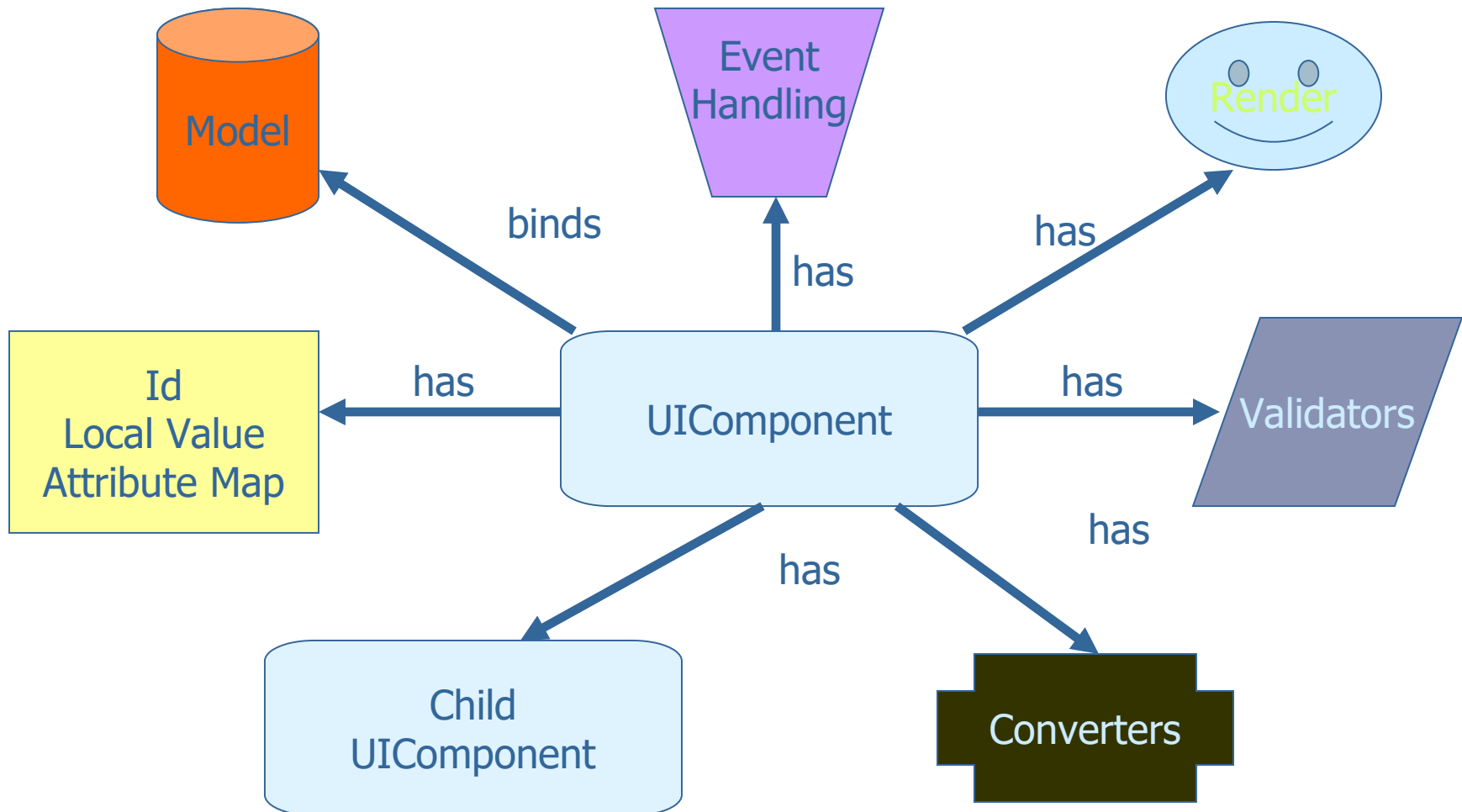
- Example Here of two numbers



- EXAMPLE HERE
- After that “to pta yai chala”

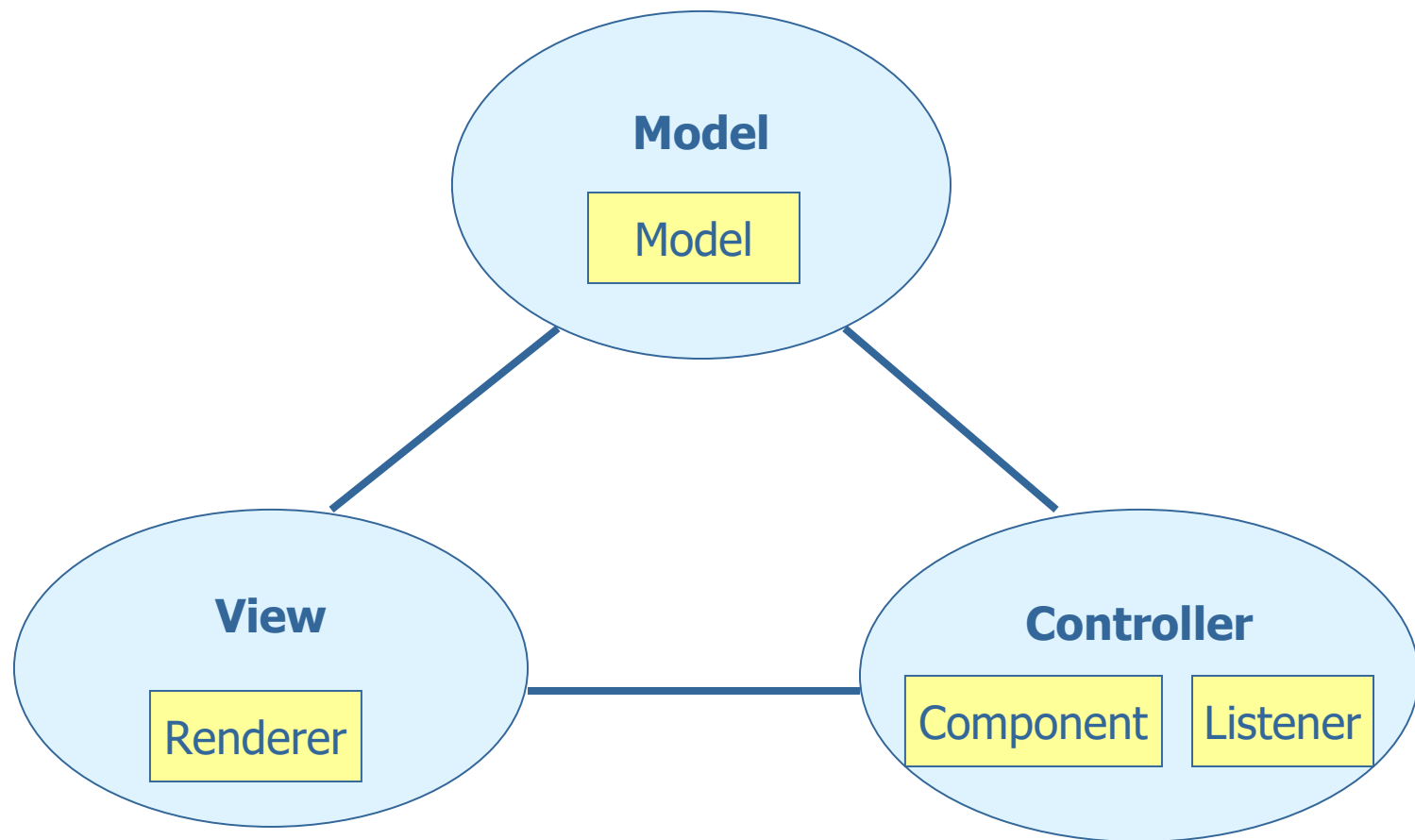


# JSF Component



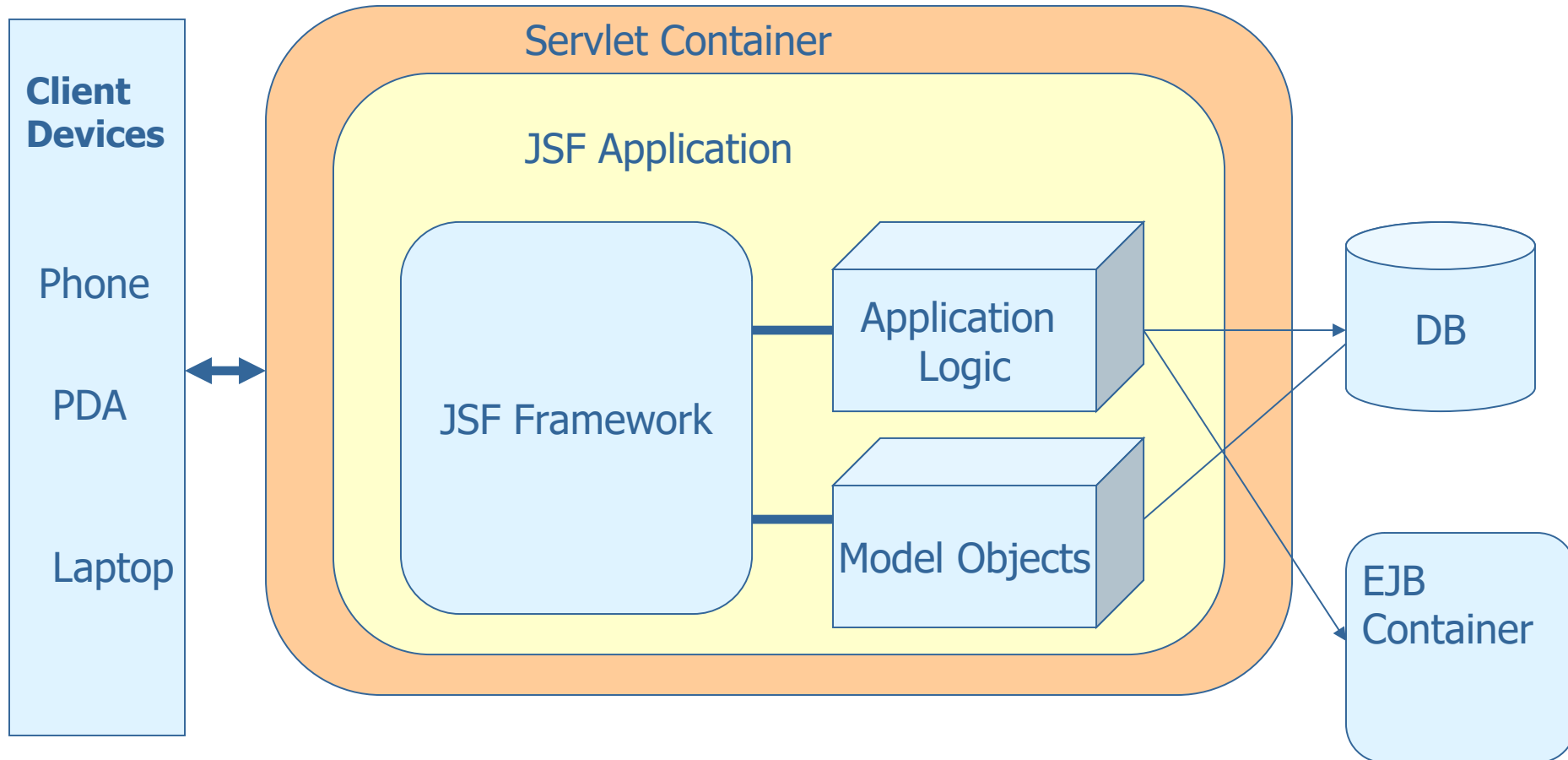


# JSF MVC Model





# JSF Application







# JSF - HTML & CSS Integration

- HTML Integration
  - Pass-through attributes  
`<h:inputText size="5" onblur="checkValue();" />`
- HTML within JSF tags does not work without `f:verbatim`  
`<h:panelGroup>`  
    `<f:verbatim>html</f:verbatim>`  
`</h:panelGroup>`
- Stylesheets Integration
  - Most HTML tags have one or more attributes (`style`, `styleClass`) for passing style information  
`<h:outputText styleClass="header" value="#{bundle.welcome}" />`
  - For data table  
`<h:dataTable rowClasses="odd, even", columnClasses="columnOne, columnTwo" ..`



# JSF - Summary

- Powerful framework based on reusable UI components for building web-based applications in Java
- Make it easy to develop web-based application using WYSIWYG tool
- No file upload support, client side validation for standard components
- Will find out in the next 12 months



# Support





# Support

- Technical
  - <http://www.google.com>
  - <http://forum.java.sun.com/forum.jspa?forumID=427>
- Political (Sun, IBM, Oracle)
- Most IDEs have limited JSF support

|                       |                             |                         |           |
|-----------------------|-----------------------------|-------------------------|-----------|
| Eclipse*              | IntelliJ*                   | JBuilder<br>(WYSIWYG)   | Netbeans* |
| IBM WSAD<br>(WYSIWYG) | Studio Creator<br>(WYSIWYG) | JDeveloper<br>(WYSIWYG) | Notepad** |

\* Requires a free plugin



# Resources

- Become a member of SDN (sun developer network)
- <http://java.sun.com/j2ee/javaxserverfaces>
- <http://forum.java.sun.com/forum.jsp?forum=427>
- <http://www.jsfcentral.com>
- <http://www.corejsf.com>
- <http://www.theserverside.com>
- <http://www.javaworld.com>



# Recommended Reading

(the end)



## Books (NI)

### Books I can recommend:

- **Core JavaServer Faces**  
by David Geary, Cay Horstmann
- **JavaServer Faces in Action**  
by Kito D. Mann

### Other books

- **JavaServer Faces**  
by Hans Bergsten
- **Mastering JavaServer Faces**  
by Bill Dudney, Jonathan Lehr, Bill Willis, LeRoy Mattingly
- **JavaServer Faces Programming**  
by Budi Kurniawan
- **Javasever Faces Kick Start (Kick Start)**  
by James Turner, Craig McClanahan, Kunal Mittal



## **Topics to be covered (ni)**

- Client side validation
- JSF (free tools as well)
- Webservices
- Tell Resources e.g. sites
- Packaging and deploying web applications
- Big Enterprise diagram re-visit
- What to do next (EJBs)
- Wrap up
- Give acknowledgment





# Packaging and Deploying Web Application (ni)

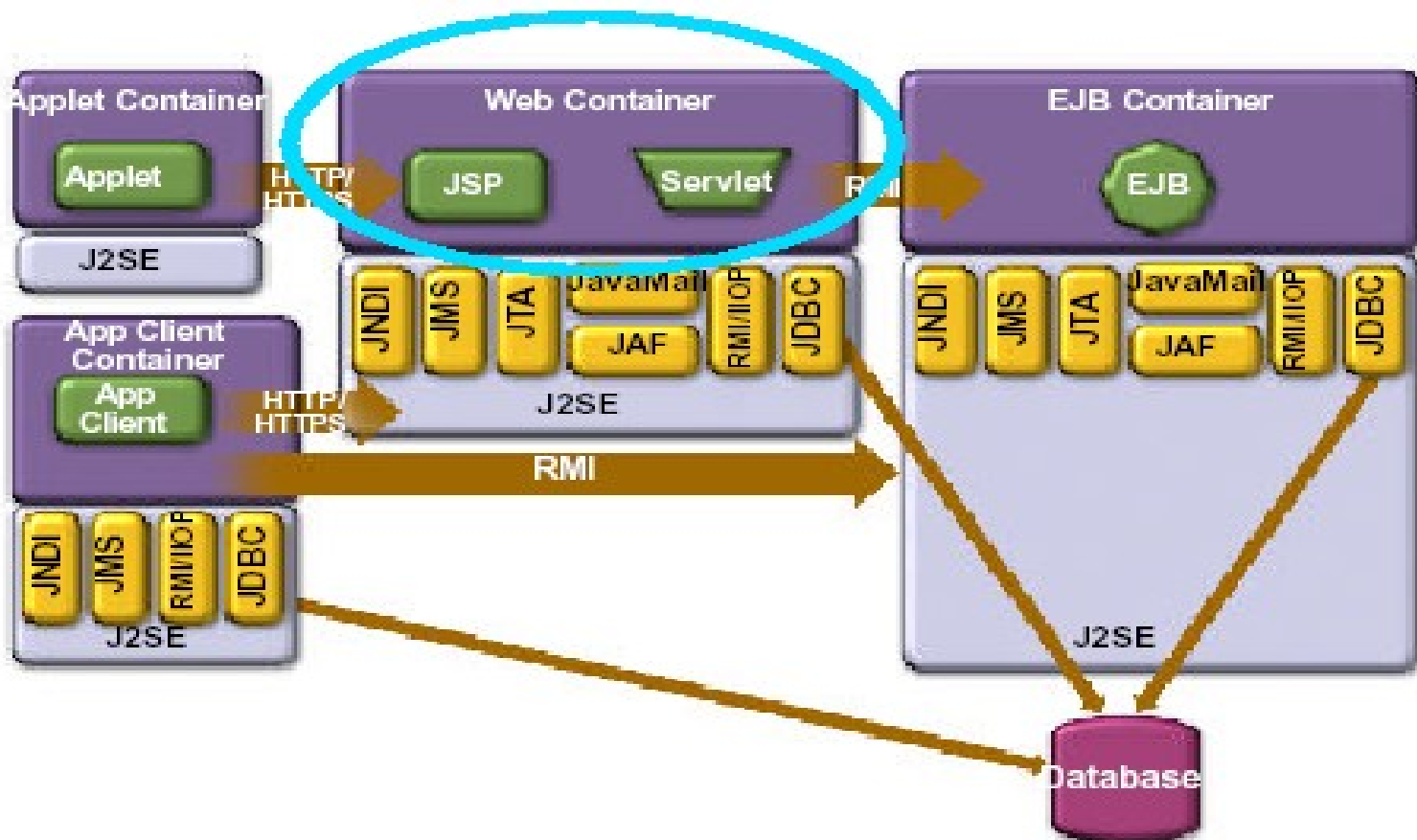
- **Agenda**

- Web application, components and Web container
- Web application development and deployment steps
- Web Application Archive (\*.WAR file)
  - WAR directory structure
  - WEB-INF subdirectory
- Configuring Web application
  - Web application deployment descriptor (web.xml file)
  - Web application life-cycle management
- Installing & Deploying Web application
- Listing Web applications



**Put a better pic here  
(NI)**

## Web Components & Container





## Web Components & Container (NI)

- Web components are in the form of either Servlet or JSP (along with JavaBean's, JSTL and custom tags)
- Web components run in a Web container e.g. Tomcat
- Web container provides system services to Web components
  - Request dispatching, security, and life cycle management



**NI**

- **Web Application & Components**
  - Web Application is a deployable package
    - Web components (Servlets and JSP's)
    - Static resource files such as images
    - Helper classes. Libraries
    - Deployment descriptor (web.xml file)
  - Web Application can be represented as
    - A hierarchy of directories and files (unpacked form) or
    - WAR file reflecting the same hierarchy (packed form)
    - Development mode vs Deployment mode



**NI**

- **Web Application Development and Deployment Steps**
  1. Write (and compile) the Web component code (Servlet or JSP) and helper classes referenced by the web component code
  2. Create any static resources (for example, images or HTML pages)
  3. Create deployment descriptor (web.xml)
  4. Build the Web application (\*.war file or deployment-ready directory)
  5. Install or deploy the web application into a Web container Web clients are now ready to access it via URL



**NI**

- **4. Build the Web application**
  - Either \*.WAR file or unpacked form of \*.WAR file
- **5. Install or Deploy Web application**
  - Deploy the application over deployment platform such as Sun Java System App Server or Tomcat
  - Ways to deploy
  - Use some console based command
    - `asadmin deploy --port 4848 --host localhost --passwordfile "c:\j2eetutorial14\examples\common\adminpassword."`
  - Use some graphical interface of the server.
    - App server admin console
  - NetBeans



- **What is \*.WAR file?**
  - Ready to deploy'able package over web container
  - Similar to \*.jar file
  - Contains things to be deployed
    - Web services components
    - Server-side utility classes
  - Static Web presentation content (HTML, image, etc)
  - Client-side classes (applets and utility classes)
  - Reflects contents in build directory



# NI

- **Document Root & Context**

- Document Root of the Web application is the Top-level directory of WAR
- Contains JSP pages, client-side classes and archives, static Web resources
- Also contains WEB-INF directory

- A context is a name that gets mapped to the document root of a Web application

- /hello1 is context for hello1 example
- A way to distinguish Web applications in a single Web container
- Has to be specified as part of client URN (request path)
- Name of document root



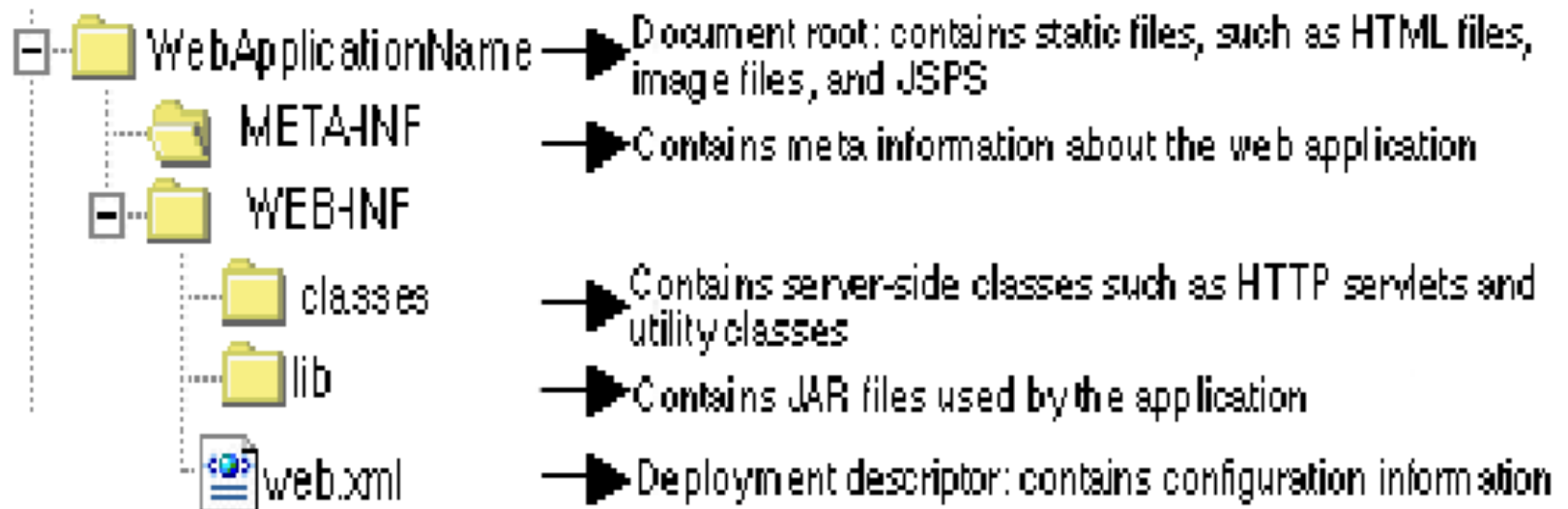


# NI

- **HTTP request URL & Webcomponent URN (alias) & Context**
- Request URL: User specified access point of a web resource
  - `http://[host]:[port]/[request path]?[query string]`
  - [request path] is made of context and web component's URN
  - <http://localhost:8080/hello1/greeting?username=Monica>
- Context: Name of the root document of a web application
  - Identifies a particular application on that server
  - `/hello1` is context



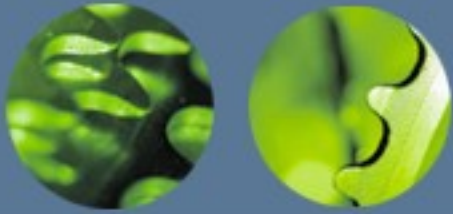
# Directory Structure of \*.WAR file



This picture shows the directory structure of \*.war file. Please note that the “WebApplicationName” directory reflects the Document root, thus the context.



- **How to Create \*.WAR file?**
- 3 different ways
  - Use IDE (NetBeans)
  - Use ant tool after putting proper build instruction in build.xml file
  - Use “jar cvf <filename>.war .” command under build directory



- Show demo here

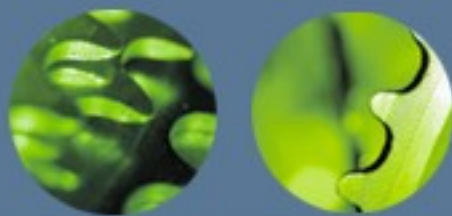


# Evolutionary Path (NI)

- Client Server Model
- Web-based Computing
- Web Services Model-SOA

(Remove below after describing)

- In the beginning, things were built and deployed typically in the form of client and server model in which clients talk to a single server, for example, doing remote procedure calls or sockets
- The second phase can be called web-based computing in which many clients talk to many servers through the net. In this phase, communicating partners still have to go through some pre-arrangement in terms of what common object model they have to use or what common communication protocol they have to agree upon. Largely end user interaction with site
- Finally, the web services model. As we will see in the following slides, under web services model, service users and service providers can be dynamically connected. And the pretty much every computing device and application participate as both service user and service provider. Example of service (NADRA-PASSPORT-PIA-TRAVEL AGENT)



# Distributed Computing Evolution





## WEB Services (NI)

- Web services are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients.
- Java Platform, Enterprise Edition (Java EE)  
provides the APIs and tools you need to create and deploy interoperable Web services and clients



# What is Web Services? (NI)

- **Web Services Definition by W3C**
  - A Web service is a software application
  - identified by a URI,
  - whose interfaces and binding are capable of being defined, described and discovered by XML artifacts
  - and supports direct interactions with other software applications
  - using XML based messages via internet-based protocols





# NI

## Traditional C/S

- Within enterprise
- Tied to a set of programming languages
- Procedural
- Usually bound to a particular transport
- Tightly-coupled
- Efficient processing (space/time)

## Web Service

- Between enterprises
- Program language independent
- Message-driven
- Easily bound to different transports
- Loosely-coupled
- Relatively not efficient processing



NI

# Web Application vs. Web Services

## Web Application

- 7 User-to-program interaction
- 7 Static integration of components
- 7 Monolithic service

## Web Service

- 7 Program-to-program interaction
- 7 Possibility of dynamic integration of components (in the future)
- 7 Possibility of service aggregation (in the future)



# Comparison to the Server/Browser Model

- Web page

- Has a UI

- Interacts with users

- Works with web browser client

## Web Service

No UI

Interacts with applications

Works with any type client



(NI)

- Web application in its current form is typically focused on between an end user and program
- while under web services model, program to program communication would be much more common form of communication.
- Also under web application model, things are pretty much in static mode. That is, in order to integrate various applications, you have to statically configure them to talk each other.
- Under web services model, things could be a lot more dynamic, that is, service users will find service provider and use the services of those providers more dynamically.
- Also under Web services model, services can be aggregated again in a adhoc and dynamic fashion.

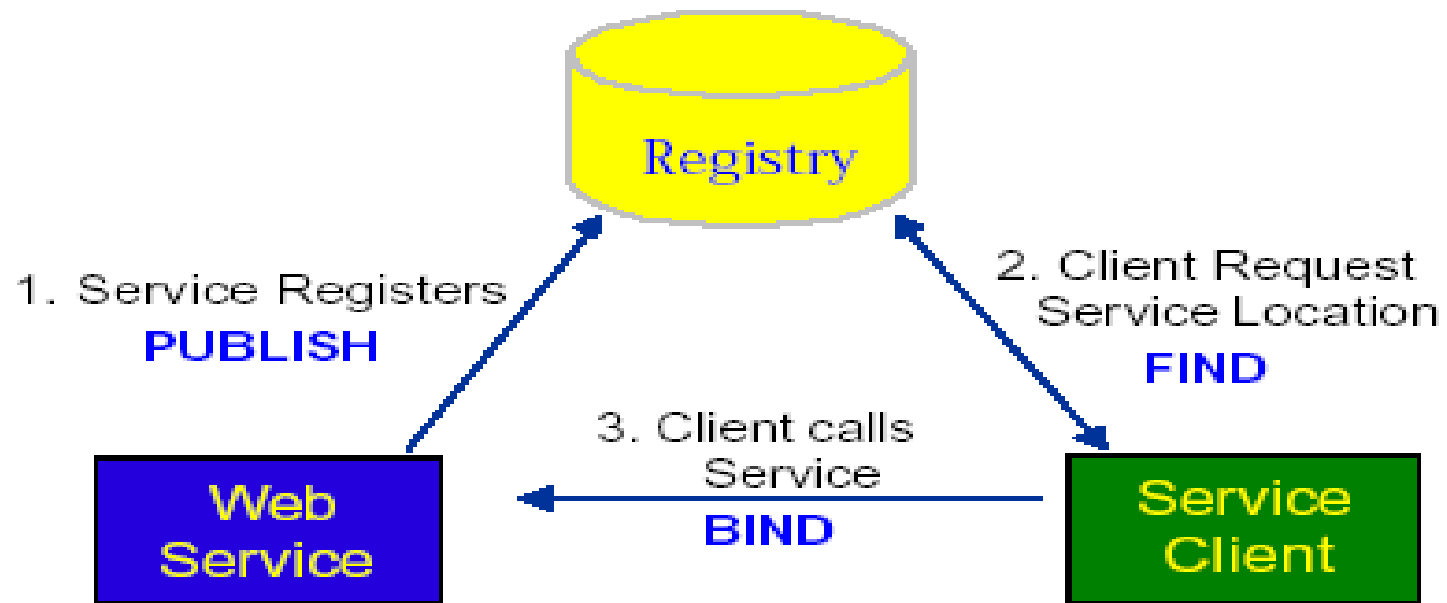


# Types of Web Services

- Data providers, e.g., a service providing stock quotes
- Business-to-business process integration, e.g., purchase orders
- Enterprise application integration
  - Different applications work together simply by adding a webservice wrapper.



# (Simplified) Web Service Architecture





# Web Service Architecture (correct it accd. To VHS)

- Publishing the service—how do clients find it?
  - UDDI (Universal Description, Discovery & Integration).
  - A way to publish and find web services. A repository of web services on the internet where a machine or a human can find different web services
  - [www.uddi.org](http://www.uddi.org)
- Describing the service—how do clients know how it works (which functions, parameters etc)?
  - WSDL (Web service description language) pronounced as viz-dal
  - An XML format for describing web services in terms of methods , properties , data types and protocols.
  - Makes the web service self describing



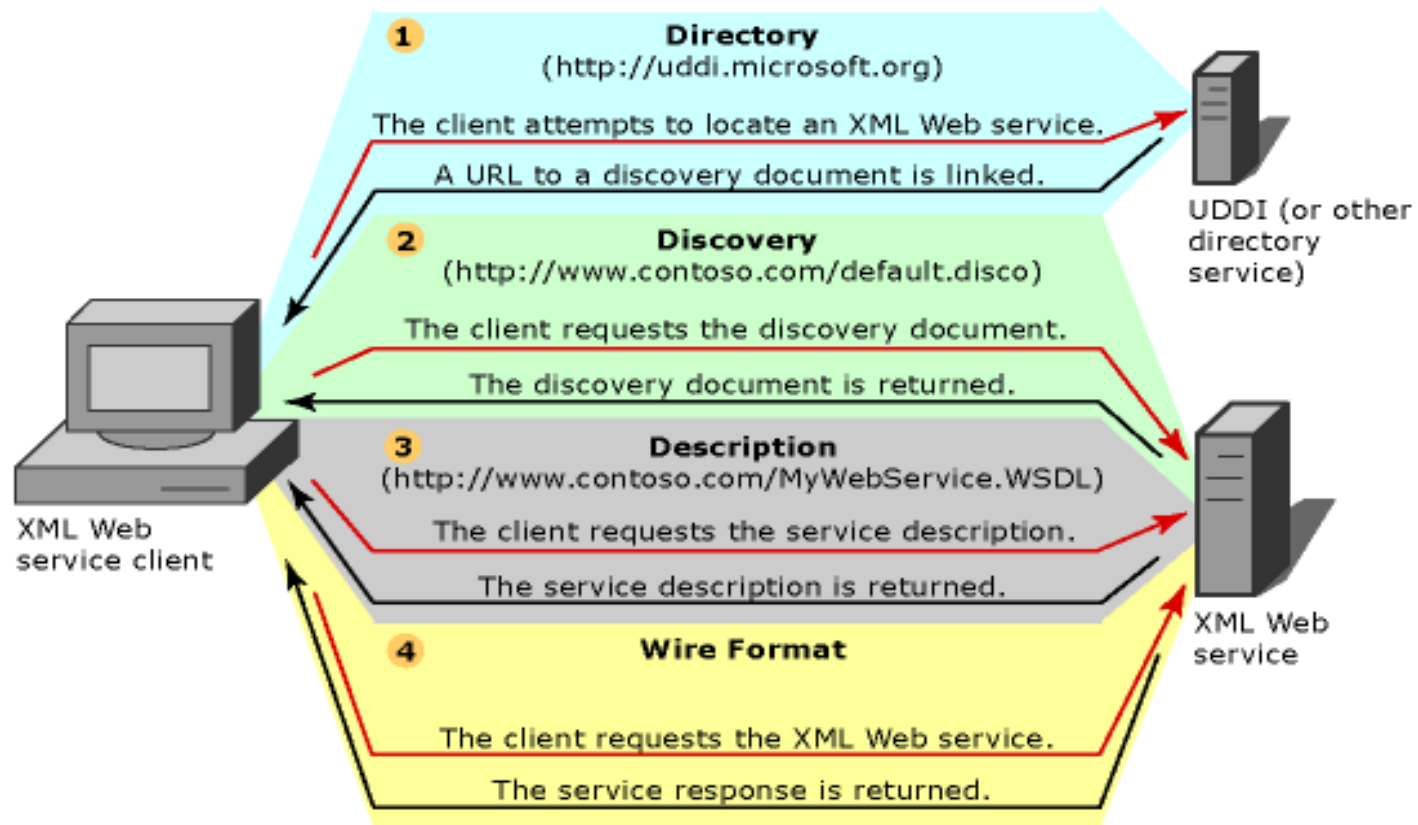
# Some Web Service Issues & Solutions... (NI)

- Network—how can we avoid firewall issues (firewalls generally block binary data)?
  - Easy, we use HTTP and send XML over it (SOAP)
- Development tools—how can we build services in any language?
  - Very easy , use Visual studio .NET, JBuilder , Studio Creator ☺
- Plumbing—how can we simplify making all of this work?
  - Again very easy , use Visual studio .NET, JBuilder , Studio Creator ☺



# XML Web Service Infrastructure

## Make Your own





# Summing up WebServices (NI)

- **Characteristics of Web Services**
  - XML based everywhere
  - Message-based
  - Programming language independent
  - Could be dynamically located
  - Could be dynamically assembled or aggregated
  - Accessed over the internet
  - Loosely coupled
  - Based on industry standards



# NI

## Why Web Services?



### Web Services:

- ⌘ Are platform neutral
- ⌘ Are accessible in a standard way
- ⌘ Are accessible in an **interoperable** way
- ⌘ Use simple and ubiquitous plumbing
- ⌘ Are relatively cheap
- ⌘ Simplify **enterprise integration**



# Resources

- <http://www.apl.jhu.edu/~hall/java/>

Java Programming Resources -- Java, Java, and more Java - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites

Address <http://www.apl.jhu.edu/~hall/java/> Go Links

## Java Programming Resources

**J2EE Training Courses:** Short courses on [JSP](#), [servlets](#), [Struts](#), [JSF](#), and [Java programming](#) are available from a well-known author and award-winning instructor. Public or on-site at *your* company.

**Servlet/JSP Books:** Check out these bestselling books from Sun Press:  
- [Core Servlets & JSP](#)  
- [More Servlets & JSP](#)

|                                                   |                                                                          |                                                                       |
|---------------------------------------------------|--------------------------------------------------------------------------|-----------------------------------------------------------------------|
| <a href="#">Java FAQs and Tutorials</a>           | <a href="#">Downloading Compilers &amp; Browsers</a>                     | <a href="#">Java Programming Courses</a>                              |
| <a href="#">Official Java Documentation</a>       | <a href="#">Integrated Development Environments (IDEs) &amp; Editors</a> | <a href="#">Core Servlets &amp; JSP Code Archive</a>                  |
| <a href="#">Java Books</a>                        | <a href="#">Java Resources at Sun</a>                                    | <a href="#">Core Web Programming Code Archive</a>                     |
| <a href="#">Java Forums and Discussion Groups</a> | <a href="#">HTML &amp; JavaScript</a>                                    | <a href="#">Beginners' Corner</a>                                     |
| <a href="#">Java Jobs</a>                         | <a href="#">coreservlets.com</a>                                         | <a href="#">JSP, Servlet, Struts, &amp; JSF Programming Resources</a> |

**Free Java GUI Framework** Write any Swing GUI using only XML. A totally free MVC java framework.

**J2EE Jobs** Join GetAFreelancer.com and bid on J2EE projects. Join Free.

**Java Discussion Group** Collaborate with your peers! Free e-mail-based support.

**Java Reporting made easy** Exceptional Java Reporting Tool. PDF, HTML, Excel, Swing and more.

Ads by Goooooogle Advertise on this site

Java programming resources collected by [Marty Hall](#) for courses in the Johns Hopkins [Part-Time MS Program in Computer Science](#) and for various industry seminars and [short courses](#). Many are extracted from [Core Servlets and JavaServer Pages](#) and [Core Web Programming](#) from Sun Microsystems Press and Prentice Hall.

start

Internet 1:38 AM



# Acknowledgment

- Sang Shin, [sang.shin@sun.com](mailto:sang.shin@sun.com)
- <http://java.sun.com>
- <http://www.jsfcentral.com>
- <http://www.corejsf.com>
- <http://www.theserverside.com>
- <http://www.javaworld.com>



[www.vuCtn.com](http://www.vuCtn.com)

Good Luck

Regards : Zarva Chaudhary

Admin:

\* Zarva Chaudhary \*

\* Chaudhary Moazzam \*

\* Laiba Maki \*