# CS390-JAVA
# Advanced Java

Gustavo Rodriguez-Rivera

# General Information

- Web Page: http://www.cs.purdue.edu/homes/cs390lang/java
- Office: LWSN1185
- E-mail: grr@cs.purdue.edu
- Textbook:
  - Core Java Volume II, 8th edition, Cay S. Horstmann, Gary Cornell
- They are great books.

# Mailing List

- E-mail questions to
  cs390-ta@cs.purdue.edu
- TAs office hours will be posted in the web page.

# Grading

- Grade allocation
  - Final: 50%
  - Projects: 50%
- Exams also include questions about the projects.
- There will be 1 single project for the 5 weeks with intermediate milestones.

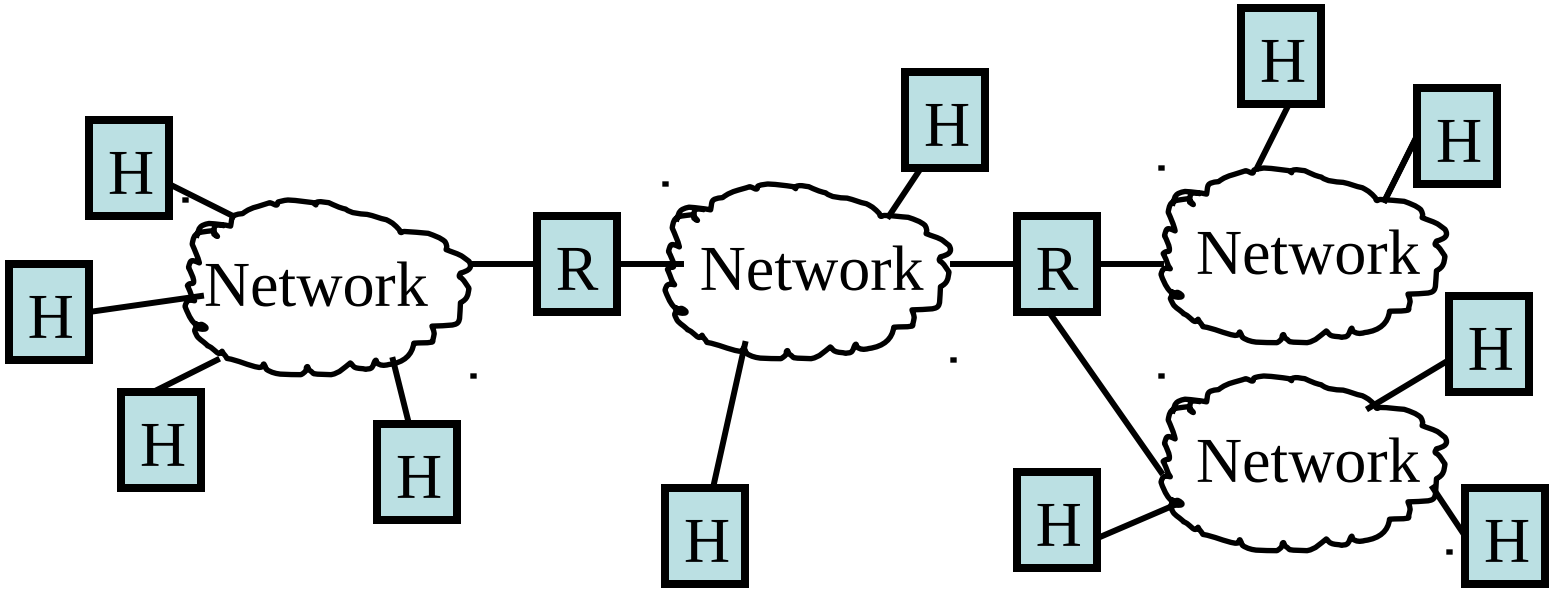
# Course Organization

- Networking
- JDBC
- J2EE
- Swing
- Reflection

# Java Networking

# Internet Architecture

- The Internet is a collection of
  - Networks
  - Routers interconnecting networks
  - Hosts connected to networks.



# IP Addresses

- Every computer in the network has an IP address.
- AN IP address identifies a network interface in a computer.
- The IP address is made of 4 bytes of the form d.d.d.d like 128.10.3.4
- DNS (Domain Name Service) servers are used to translate names such as www.google.com to IP addresses such as 128.10.4.5

# Java Networking

- To communicate with each other computers use a protocol TCP/IP.
- TCP/IP handles the retransmission of packets when data is lost.
- Java makes it easy to implement network applications.
- It simplifies the communication using a Socket class.

# Using telnet

- telnet is a very useful debugging program for network applications.
- It connects to a server and sends what it is typed. It then prints on the screen what it receives.
- Try:

  lore 128 $ telnet time-A.timefreq.bldrdoc.gov 13

  Trying 132.163.4.102...

  Connected to time-A.timefreq.bldrdoc.gov.

  Escape character is '^]'.

  55604 11-02-12 23:46:16 00 0 0 251.7 UTC(NIST) *

# Using telnet to connect to a HTTP server

```
lore 129 $ telnet www.cs.purdue.edu 80
Trying 128.10.19.20...
Connected to lucan.cs.purdue.edu.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 12 Feb 2011 23:50:18 GMT
Server: Apache/2.2.11 (Unix)
  mod_ssl/2.2.11 OpenSSL/0.9.8q
Vary: Accept,Accept-Charset
…
```

# The Socket Class

- Socket(String host, int port);
  - Creates a Socket to connect to this host and port
- InputStream getInputStream()
  - Gets an input stream to receive data to the server.
- OutputStream getOutputStream()
  - Gets an output stream to send data to the server.

# A Socket Test

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * This program makes a socket connection to the atomic clock in Boulder, Colorado, and prints the
 * time that the server sends.
 * @version 1.20 2004-08-03
 * @author Cay Horstmann
 */
public class SocketTest
{
   public static void main(String[] args)
   {
      try
      {
         Socket s = new Socket("time-A.timefreq.bldrdoc.gov", 13);
         try
         {
            InputStream inStream = s.getInputStream();
            Scanner in = new Scanner(inStream);
```

# A Socket Test

```
            while (in.hasNextLine())
            {
               String line = in.nextLine();
               System.out.println(line);
            }
         }
         finally
         {
            s.close();
         }
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

# Socket Timeouts

- Sometimes a call to Socket(server, port) may block indefinitely.
- You may set a timeout to the connection:

```
Socket s = new Socket();
s.connect(new
  InetSocketAddress(host, port),
  timeout);
```

# Implementing a Server

- A server is a program that runs forever listening for incoming connections in a well known port.
- To implement servers in Java we use the ServerSocket class.
- ServerSocket(int port)
  - Creates a new server socket waiting for connections in a port.
- Socket accept()
  - Waits until a connection arrives. Then it returns a Socket object.
- void close()
  - Close connection.

# EchoServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * This program implements a simple server that listens to port 8189 and
   echoes back all client
 * input.
 * @version 1.20 2004-08-03
 * @author Cay Horstmann
 */
public class EchoServer
{
   public static void main(String[] args)
   {
      try
      {
         // establish server socket
         ServerSocket s = new ServerSocket(8189);

          while (true) {
             // wait for client connection
             Socket incoming = s.accept();
```

# EchoServer.java

```
try
{
   InputStream inStream = incoming.getInputStream();
   OutputStream outStream = incoming.getOutputStream();

   Scanner in = new Scanner(inStream);
   PrintWriter out =
      new PrintWriter(outStream, true /* autoFlush */);

   out.println("Hello! Enter BYE to exit.");
```

# EchoServer.java

```
            // echo client input
            boolean done = false;
            while (!done && in.hasNextLine())
            {
               String line = in.nextLine();
               out.println("Echo: " + line);
               if (line.trim().equals("BYE")) done = true;
            }
         }
         finally
         {
            incoming.close();
         }
   }// while
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

# Serving Multiple Clients Simultaneously

- The problem with a single-threaded server is that only one request has to be served at a time.
- If two requests arrive one of the may need to wait until the other one is served.
- If the first request is using a slow internet line, the second request may need to wait a long time.
- To solve this problem, we create a new thread for every request received.

# Multithreaded Echo Server

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
   This program implements a multithreaded server that listens to port 8189 and echoes back
   all client input.
   @author Cay Horstmann
   @version 1.20 2004-08-03
*/
public class ThreadedEchoServer
{
   public static void main(String[] args )
   {
      try
      {
         int i = 1;
         ServerSocket s = new ServerSocket(8189);
```

# Multithreaded Echo Server

```
    while (true)
         {
            Socket incoming = s.accept();
            System.out.println("Spawning " + i);
            Runnable r = new ThreadedEchoHandler(incoming);
            Thread t = new Thread(r);
            t.start();
            i++;
         }
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

# Multithreaded Echo Server

```
/**
   This class handles the client input for one server
   socket connection.
*/
class ThreadedEchoHandler implements Runnable
{
   /**
      Constructs a handler.
      @param i the incoming socket
      @param c the counter for the handlers (used in
   prompts)
   */
   public ThreadedEchoHandler(Socket i)
   {
      incoming = i;
   }
```

# Multithreaded Echo Server

```
public void run() {
      try
      {
         try
         {
            InputStream inStream = incoming.getInputStream();
            OutputStream outStream = incoming.getOutputStream();

            Scanner in = new Scanner(inStream);
            PrintWriter out = new PrintWriter(outStream,
                                       true /* autoFlush */);

            out.println( "Hello! Enter BYE to exit." );
```

# Multithreaded Echo Server

```
            // echo client input
            boolean done = false;
            while (!done && in.hasNextLine())
            {
               String line = in.nextLine();
               out.println("Echo: " + line);
               if (line.trim().equals("BYE"))
                  done = true;
            }
         }
         finally
         {
            incoming.close();
         }
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }

   private Socket incoming;
}
```

# Making URL Connections

- Java provides a URL and URLConnection classes that allow connecting to a HTTP server.
- URL url = new URL(urlName)
  - Creates a new URL
- URLConnection conn = url.openConnection()
  - Creates a new URLConnection object connected to this URL
- connect()
  - Connects to the remote server and gets the header information.

# URLConnectionTest

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * This program connects to an URL and displays the response header data and the first
    10 lines of
 * the requested data.
 *
 * Supply the URL and an optional username and password (for HTTP basic authentication)
    on the
 * command line.
 * @version 1.11 2007-06-26
 * @author Cay Horstmann
 */
public class URLConnectionTest
{
   public static void main(String[] args)
   {
      try
      {
         String urlName;
         if (args.length > 0) urlName = args[0];
         else urlName = "http://java.sun.com";
```

# URLConnectionTest

```
URL url = new URL(urlName);
URLConnection connection = url.openConnection();

connection.connect();

// print header fields
Map<String, List<String>> headers =
           connection.getHeaderFields();
for (Map.Entry<String, List<String>> entry :
                                headers.entrySet())
{
   String key = entry.getKey();
   for (String value : entry.getValue())
      System.out.println(key + ": " + value);
}
```

# URLConnectionTest

```
// print convenience functions

System.out.println("----------");
System.out.println("getContentType: " +
                   connection.getContentType());
System.out.println("getContentLength: " +
                   connection.getContentLength());
System.out.println("getContentEncoding: " +
                   connection.getContentEncoding());
System.out.println("getDate: " + connection.getDate());
System.out.println("getExpiration: " +
                   connection.getExpiration());
System.out.println("getLastModifed: " +
                   connection.getLastModified());
System.out.println("----------");
```

# URLConnectionTest

```
            Scanner in =
                new Scanner(connection.getInputStream());

            // print first ten lines of contents

            for (int n = 1; in.hasNextLine() && n <= 10; n++)
               System.out.println(in.nextLine());
            if (in.hasNextLine())
                 System.out.println(". . .");
         }
         catch (IOException e)
         {
            e.printStackTrace();
         }
      }
}
```

# Using Pattern Matching

- Java has a powerful Pattern matching class to check if a string matches some pattern or not.
- A pattern is described with a regular expression. Examples:

  ab*c - A string that starts with “a” followed by 0 or more “b”s and ending with c. abbbc, abc, ac match.

  ab+c – Same as before but b has to occur 1 or more times. abbbc matches but ac will not match.

# Using Pattern Matching

- Also we have predefined sets like
  - \s - A whitespace character: [ \t\n\x0B\f\r]
  - \S - A non-whitespace character: [^\s]
- See more details in:

  http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html

# Using Pattern Matching

- Define a regular expression:

String **`patternString`** = “ab+c”;

- Compile pattern:

```
Pattern pattern =
      Pattern.compile(patternString,
      Pattern.CASE_INSENSITIVE);
```

- Get matcher object from string to match.

```
Matcher matcher = pattern.matcher(str);
```

- Iterate over string to match multiple times.

```
while (matcher.find()) {
  int start = matcher.start();
  int end = matcher.end();
  String match = input.substring(start, end);
  System.out.println(match);
}
```

# A pattern for <a> tags in HTML

- An <a> tag or anchor tag in HTML contains links to other URLs. Example:

  <a href="http://www.cs.purdue.edu">
- A simple regular expression to match a <a> tag would be:

  **"<a\\s+href=\"[^\"]*\"|[^>]*>"**

  where:
  - \\s+ – Matches 1 or more space characters
  - \" – Matches a " character.
  - [^\"] - Matches everything that is no " character.

# Example: Printing the href tags

```
import java.io.*;
import java.net.*;
import java.util.regex.*;

/**
 * This program displays all URLs in a web page by
   matching a regular expression that describes the
 * <a href=...> HTML tag. Start the program as <br>
 * java HrefMatch URL
 * @version 1.01 2004-06-04
 * @author Cay Horstmann
 */
public class HrefMatch
{
   public static void main(String[] args)
   {
```

# Example: Printing the href tags

```
try {
  // get URL string from command line or
  // use default
  String urlString;
  if (args.length > 0) urlString = args[0];
  else urlString = "http://java.sun.com";

  // open reader for URL
  InputStreamReader in =
     new InputStreamReader(
        new URL(urlString).openStream());

  // read contents into string builder
  StringBuilder input = new StringBuilder();
  int ch;
```

# Example: Printing the href tags

```
while ((ch = in.read()) != -1)
    input.append((char) ch);

// search for all occurrences of pattern
String patternString =
"<a\\s+href\\s*=\\s*(\"[^\"]*\"|[^\\s>]*)\\s*>";
Pattern pattern =
   Pattern.compile(patternString,
   Pattern.CASE_INSENSITIVE);
Matcher matcher = pattern.matcher(input);
```

# Example: Printing the href tags

```
while (matcher.find()) {
    int start = matcher.start();
    int end = matcher.end();
    String match = input.substring(start, end);
    System.out.println(match);
}
```

# Example: Printing the href tags

```
    }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        catch (PatternSyntaxException e)
        {
            e.printStackTrace();
        }
    }
}
```

# Groups in Pattern Matching

- Sometimes you want to use pattern matching to extract group of characters from inside the matched string.
- For that you use “groups”
- A group is represented by a regular sub-expression inside parenthesis.
- Example:

  a(b*c*)d

  aa(b*)c(de*)f

# Groups in Pattern Matching

- The groups are identified by an index.
- Group 0 represents the whole string matched by the regular expression.
- Group 1 represents the characters matched by the regular sub-expression in the first set of parenthesis and so on.

# Printing only the URLs

- The previous example prints the whole <a> tag.
- If we want to print only the URLs in
  <a href="http://www.cs.purdue.edu">
  then we need to print the group in the href=(….)
- We do this by modifying the while matcher loop with:

```
while (matcher.find()) {
    int start = matcher.start();
    int end = matcher.end();
    //String match = input.substring(start, end);
    System.out.println(matcher.group(1) );
 }
```

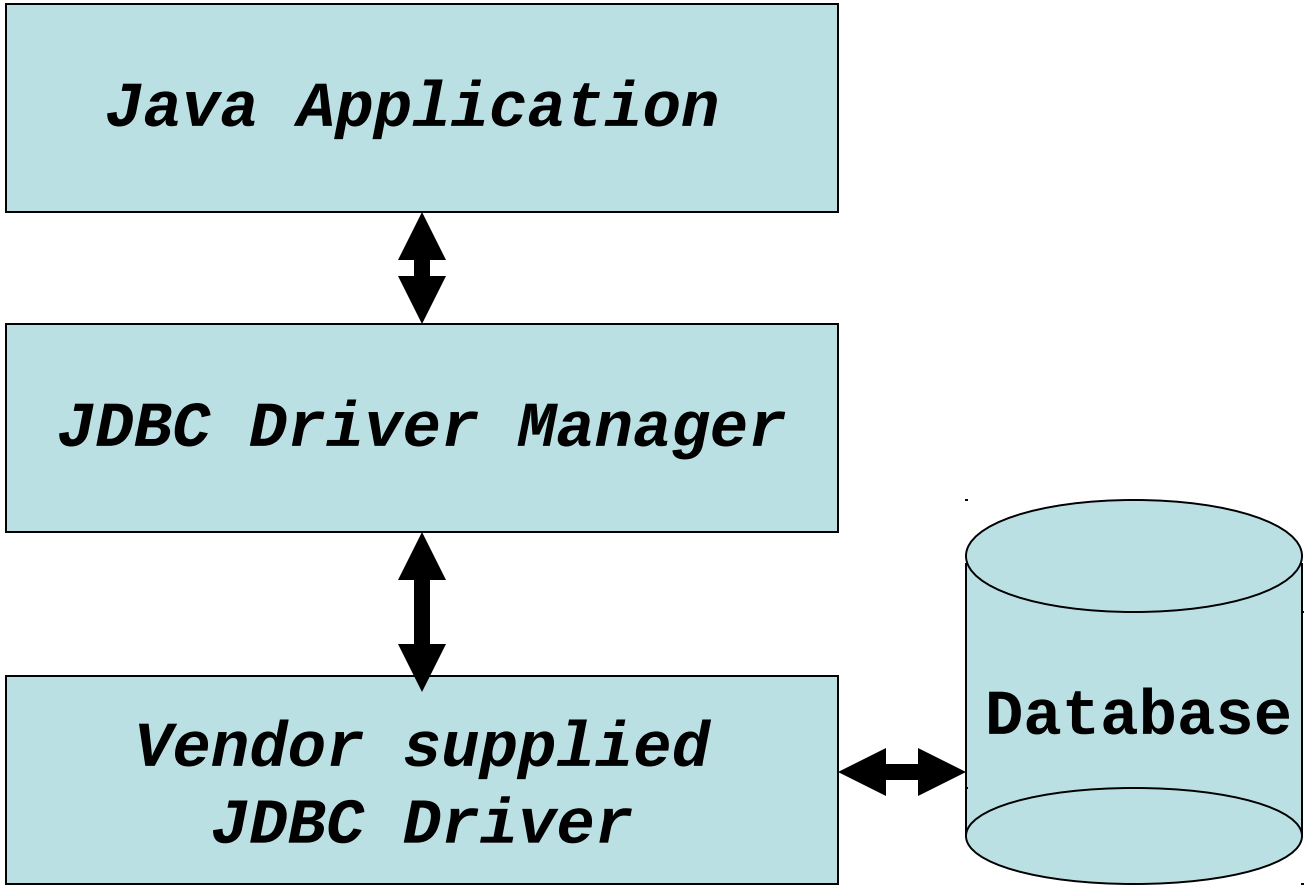
- Match.group(1) represents the string matched by the regular sub-expression between the parenthesis in patternString.

# Database Programming

# JDBC

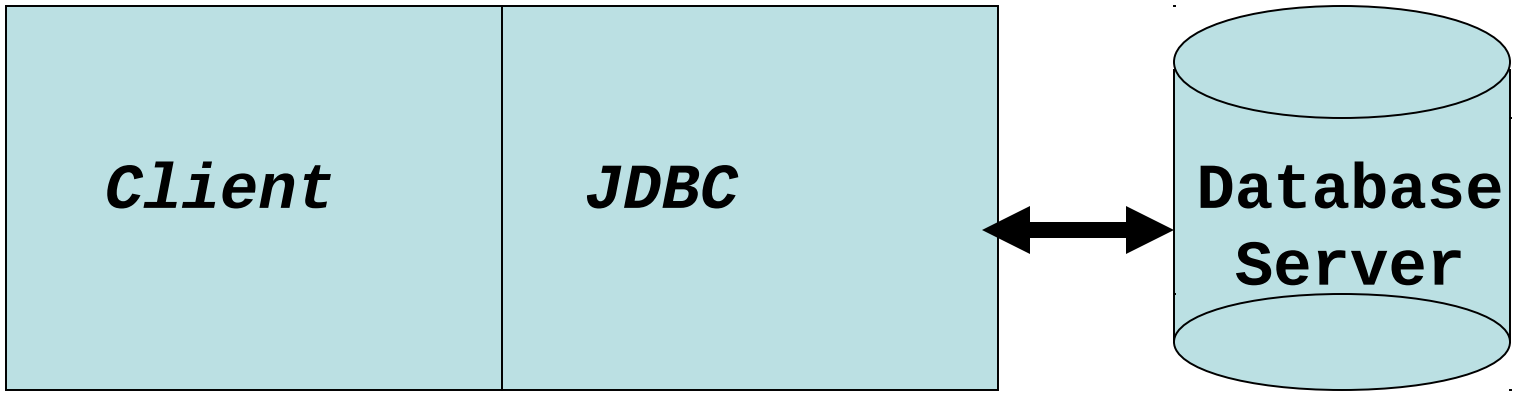
- JDBC is a Java API that allows a Java program to connect to a database to query and modify it.
- JDBC is divided in two parts
  - A JDBC common API that is written in Java and supplied by Sun
  - A JDBC Driver that is supplied by the database vendor and is registered with the driver manager.
- There is a JDBC driver for every database such as MySQL, Oracle, Microsoft SQL, etc.

# JDBC Communication Path



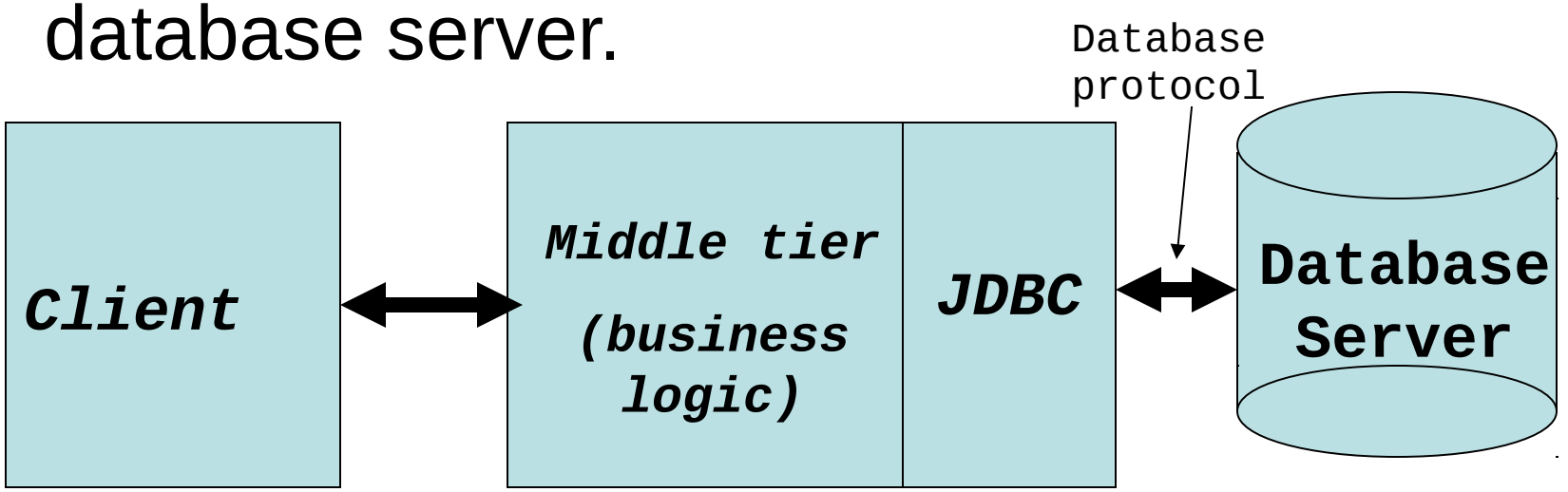
# Client/Server Application

- In a client/Server application, the client talks directly to the database server.



# Three-Tier Application

- In a three-tier application the client talks to a middle tier or program that talks to the database server.




- Three-tier is more common than the simple client/server model.

# Available SQL databases

- MySQL
  - http://www.mysql.com
  - Very popular free database
- Oracle
  - http://www.oracle.com
  - Very popular database for enterprise use
- Microsoft SQL
  - http://www.microsoft.com/sqlserver
  - Also very popular.

# Structured Query Language (SQL)

- JDBC lets you communicate with a database using SQL.
- Databases have a GUI program that lets you manipulate the database directly and can be used for database administration.
- You can think of a database as a group of named tables with rows and columns.
- Each column has a name and each row contains a set of related data.

# SQL by Example (from textbook)

***Authors Table***

| Autor_ID | Name | Fname |
|---|---|---|
| ALEX | Alexander | Christopher |
| BROO | Brooks | Frederick P. |

***Books Table***

| Title | ISBN | Publisher_ID | Price |
|---|---|---|---|
| A Guide to the SQL Standard | 0-201-96426-0 | 0201 | 47.95 |
| A Pattern Language: Towns, Buildings, Construction | 0-19-501919-9 | 0407 | 65.00 |

# SQL by Example (from textbook)

***BookAuthors Table***

| ISBN | Author_ID | Seq_No |
|---|---|---|
| 0-201-96426-0 | DATE | 1 |
| 0-201-96426-0 | DARW | 2 |
| 0-19-501919-9 | ALEX | 1 |

***Publishers Table***

| Publisher_ID | Name | URL |
|---|---|---|
| 0201 | Addison-Wesley | www.aw-bc.com |
| 0407 | John Wiley & Sons | www.wiley.com |

# SQL by Example

- By convention SQL keywords are written in uppercase.
- SELECT * FROM Books
  - This query returns all rows in the Books table.
  - SQL statements always require FROM
- SELECT ISBN, Price, Title

  FROM Books
  - This query returns a table with only the ISBN, price and title columns from the Books table.

# SQL by Example

- SELECT ISBN, Price, Title

  FROM Books

  WHERE Price <=29.95
  - This query returns a table with the ISBN, Price and Title from the Books table but only for the books where the price is less or equal to 29.95.

# SQL by Example

- SELECT ISBN, Price, Title

  FROM Books

  WHERE Title NOT LIKE “%n_x%”
  - Returns a table with Price and Title as columns excluding the boos that contain Linux or UNIX in the title.
  - The “%” character means any zero or more characters. “_” means  any single character.

# SQL by Example

- SELECT Title, Name, URL

  FROM Books, Publishers

  WHERE Books.Publisher_ID=Publishers.Publisher_ID

It returns a table with the Title, Name of pusblisher, and URL from Books and Publishers.

| Title | Name | URL |
|---|---|---|
| A Guide to the SQL Standard | Addison-Wesley | www.aw-bc.com |
| A Pattern Language: Towns, Buildings, Construction | John Wiley & Sons | www.wiley.com |

# SQL by Example

- You can also use SQL to change data inside a database.
- UPDATE Books

  SET Price = Price – 5.00

  WHERE Title Like “%C++%”

  This reduces the price by $5.00 for all books that have C++ in the title.

# SQL by Example (from textbook)

- You can also delete rows with SQL
- DELETE FROM Books

  WHERE Title Like “%C++%”
  - This deletes all books that have C++ in the title.

# SQL by Example

- Use INSERT to insert a new row in the table.
- INSERT INTO Books

  VALUES (‘A Guide to the SQL Standard’, ‘0-201-96426-0’, ‘0201’, 47.95)
  - This inserts a new book in the Books table.

# SQL by Example

- You can also create a new table using SQL
- CREATE TABLE Books

```
(
    TITLE CHAR(60),
    ISBN CHAR(13),
    Publisher_ID CHAR(6),
    Price DECIMAL(10,2)
)
```

# Speeding Up Searches with Indexes

- You can speed up the search in a table by using Indexes.
- An index can be created in a table to find data faster.
- Updating a table with indexes is slower since the index need also be updated.
- The index is often implemented using B-trees.
- The users will not see the indexes. Indexes will just speed up the search.

# Speeding Up Searches with Indexes

- To create an index in a table use this syntax.

```
CREATE INDEX index_name
    ON table_name (column_name)
```

- Duplicate values are allowed.

- If you want to create a unique index that does not allow duplicates, create a Unique Index

```
CREATE UNIQUE INDEX index_name
    ON table_name (column_name)
```

# SQL Tutorials

- For more information about SQL, see the SQL tutorial in
  - http://www.w3schools.com/sql/default.asp
- You can also run some SQL examples there.

# Using JDBC from Java

- Install a database like MySQL in your computer.
  - http://dev.mysql.com/get/Downloads/MySQL-5.5/mysql-5.5.9-win32.msi/from/http://mirror.services.wisc.edu/mysql/
- Save the login and password you used during the installation. You will need it later.
- Also download the mysql java driver
  - http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.15.zip/from/http://mirror.services.wisc.edu/mysql/
- You will use the Connection and Statement classes in JDBC to access the database.

# TestDB.java

```
import java.sql.*;
import java.io.*;
import java.util.*;

/**
 * This program tests that the database and the JDBC driver are
   correctly configured.
 * @version 1.01 2004-09-24
 * @author Cay Horstmann
 */
class TestDB
{
   public static void main(String args[])
   {
      try
      {
         runTest();
      }
```

# TestDB.java

```
   catch (SQLException ex)
   {
      for (Throwable t : ex)
         t.printStackTrace();
   }
   catch (IOException ex)
   {
      ex.printStackTrace();
   }
}
```

# TestDB.java

```
/**
  * Runs a test by creating a table, adding a value, showing the table contents, and removing the
  * table.
  */
public static void runTest() throws SQLException, IOException
{
   Connection conn = getConnection();
   try
   {
      Statement stat = conn.createStatement();

      stat.executeUpdate(
        "CREATE TABLE Greetings (Message CHAR(20))");
      stat.executeUpdate(
        "INSERT INTO Greetings VALUES ('Hello, World!')");
```

# TestDB.java

```
         ResultSet result =
            stat.executeQuery(
               "SELECT * FROM Greetings");
         while(result.next())
            System.out.println(result.getString(1));
         result.close();
         stat.executeUpdate("DROP TABLE Greetings");
      }
      finally
      {
         conn.close();
      }
}
```

# TestDB.java

```
/**
 * Gets a connection from the
 * properties specified in the file
 * database.properties
 * @return the database connection
 */
public static Connection getConnection()
       throws SQLException, IOException
{
   Properties props = new Properties();
   FileInputStream in = new FileInputStream(
                  "database.properties");
   props.load(in);
   in.close();
```

# TestDB.java

```
        String drivers =
            props.getProperty("jdbc.drivers");
        if (drivers != null)
          System.setProperty("jdbc.drivers", drivers);
        String url = props.getProperty("jdbc.url");
        String username =
            props.getProperty("jdbc.username");
        String password =
            props.getProperty("jdbc.password");

        return DriverManager.getConnection(
                        url, username, password);
    }
}
```

# database.properties

```
jdbc.url=jdbc:mysql://localhost:3306/COREJAVA
jdbc.username=root
jdbc.password=database
```

# Running TestDB

- Create the COREJAVA database
  - Run All Programs->MySQL->MySQL Server 5.1->MySQL Command Line Client
  - Type

  mysql> CREATE DATABASE COREJAVA;

  mysql> quit
- Start a Command Prompt window
- Set the path to point to the jdk bin dir

  ```
  > set path = "C:\Program Files (x86)\Java\jdk1.6.0_18\bin";%PATH%
  ```
- Compile TestDB.java

  ```
  javac TestDB.java
  ```

# Running TestDB

- Copy the MySQL connector to the same directory TestDB.class is in

```
$ copy C:\mysql-connector-java-5.1.15\mysql-connector-java-5.1.15-bin.jar .
```

- Run TestDB and include the MySQL connector in the class path

```
$ java.exe -cp ".;./mysql-connector-java-5.1.15-bin.jar" TestDB
 Hello, World!
```

# Executing SQL Statements in Java

- Create first a Statement object

```
Statement st =
  conn.createStatement();
```

- Put the statement in a string.

```
String cmd = “UPDATE Books”
+ “SET Price = Price – 5.00”
+ “WHERE Title NOT LIKE ‘%C++%’”;
```

- Execute statement

```
st.executeUpdate(cmd);
```

# Executing SQL Statements in Java

- executeUpdate(cmd) can execute commands like INSERT, UPDATE, DELETE, CREATE TABLE, and DROP TABLE.
- executeQuery(cmd) can execute commands like SELECT

```
ResultSet rs = stat.executeQuery(“SELECT * FROM Books”);
```

- To iterate over the results use:

```
while (rs.next()) {
  // Use the row in the result
  String isbn = rs.getString(1);
  double price = rs.getDouble(“Price”);
}
```

# Constructing SQL Commands Safely

- Instead of building the commands in a string with the “+” operator, use PrepareStatement.

```
PreparedStatement pstmt = connection.prepareStatement("SELECT * FROM books WHERE Price = ?");
pstmt.setFloat(1, price);
ResultSet rs = pstmt.executeQuery()
```

setFloat will add the adequate quotes to the statement.

# Constructing SQL Commands Safely

- If you build the SQL commands as a string with “+”  you have the risk of somebody injecting commands to your statement.

- See:

http://docs.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html

# Other Commands in SQL

Create a database on the sql server.

```
mysql> create database [databasename];
```

List all databases on the sql server.

```
mysql> show databases;
```

Switch to a database.

```
mysql> use [db name];
```

To see all the tables in the db.

```
mysql> show tables;
```

To see database's field formats.

```
mysql> describe [table name];
```

(* From http://www.pantz.org/software/mysql/mysqlcommands.html)

# Other Commands in SQL

To delete a db.

mysql> drop database [database name];

To delete a table.

mysql> drop table [table name];

Show all data in a table.

mysql> SELECT * FROM [table name];

Returns the columns and column information pertaining to the designated table.

mysql> show columns from [table name];

Show certain selected rows with the value "whatever".

mysql> SELECT * FROM [table name] WHERE [field name] = "whatever";

Show all records containing the name "Bob" AND the phone number '3444444'.

mysql> SELECT * FROM [table name] WHERE name = "Bob" AND phone_number = '3444444';

# Java 2 Platform, Enterprise Edition (J2EE)

# Java 2 Platform, Enterprise Edition (J2EE)

- Java Standard to build multitier enterprise applications.
- J2EE includes:
  - Java Beans – Distributed Java Components that communicate with each other.
  - Servlets – Java Server modules that process http requests.
  - JSP – (Java Server Pages) HTML like language with Java code embedded.
  - XML – XML parsing technology
- We will cover servlets and JSP
- The corejava books do not include J2EE.
- See http://pdf.coreservlets.com/ for more information.

# Apache Tomcat

- It is a web server written completely in Java
- It is also called a Java Application Server because it can run Java Servlets and JSP.
- There are other commercial and open source Java Application Servers:
  - JBoss
  - WebLogic
  - Websphere
  - Glassfish

# Installing Apache Tomcat

- Download Tomcat from
  - http://tomcat.apache.org/download-70.cgi
  - Select Windows-32 or your corresponding platform.
- Unzip it into C:\. Now tomcat should be in

```
C:\apache-tomcat-7.0.8
(Adjust the version component in the directory if
   needed).
```

- Also, put your JDK in C:\ . The JDK has to be in a path without spaces. Assume that the JDK is in:

```
C:\jdk1.6.0_18
```

**Important: Place your JDK in a directory where the PATH has no spaces. Otherwise, Java will get confused and have some bizarre problems.**

# Installing Apache Tomcat

- Open a  Command Prompt window
- Set the variables:

  set JAVA_HOME=C:\jdk1.6.0_18

  set CATALINA_HOME=C:\apache-tomcat-7.0.8
- Start tomcat

  C:\apache-tomcat-7.0.8\bin\startup.bat
- Open a browser and connect to

  http://localhost:8080

  You should see the tomcat web page.

# Tomcat Directories

- bin  - Startup scripts and other executables.
- conf - Server configuration files (including server.xml)
- logs - Log and output files
- webapps - Automatically loaded web applications
- work - Temporary working directories for web applications
- temp - Directory used by the JVM for temporary files (java.io.tmpdir)

# conf/server.xml

- This is the most important configuration file.
- The following entry defines the port and timeout.

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

- The following entry defines the root directory and where the web applications (servlets) are installed

```
<Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
```

# Statically served files

- In webapps/ROOT you can install HTML files, drectories and other statically served files..
- For example, create a HTML file in a webapps/ROOT/hello.html with

  <h1>Hello how are you?</h1>

  and access it using

  ```
  http://localhost:8080/hello.html
  ```
- Make sure your tomcat server is running.

# Servlets

- See the examples in:

  http://localhost:8080/examples
- Servlets are Java classes that can process HTTP requests to generate dynamic content.
- Servlets are installed in the webapps directory.
- For example, the example servlets are installed in the webapps/example directory.

# Servlet Class

- Servlets need to extend the `HttpServlet` class.
- This class has the methods:

```
public void init(          // Initializes servlet
 ServletConfig config)
 throws ServletException{}


public void doGet(         // Handles GET requests
 HttpServletRequest request,
 HttpServletResponse response)
 throws IOException, ServletException {}

public void doPost(        // Handles POST requests
 HttpServletRequest request,
 HttpServletResponse response)
 throws IOException, ServletException {}
```

# HTTPServlet Request/Response

- HttpServletRequest request
  - It is the class that represents the request.
  - You need to get a BufferReader object from it.

```
BufferedReader reader =
  request.getReader();
```

- HttpServletResponse response
  - It is the class that represents the response.
  - You need to get a PrintWriter object to use it.

```
PrintWriter out = response.getWriter();
```

# The HelloWorld Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
   response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

# Installing a Servlet

- A servlet needs to be installed in a subdirectory in webapps.

  **`webapps/APP`**

  For example:

  **`webapps/examples`**

- Also, this directory needs to contain a WEB-INF subdirectory with the Java classes and configuration.

  **`webapps/APP/WEB-INF`**

  For example:

  **`webapps/examples/WEB-INF`**

# Installing a Servlet

- **`webapps/APP/WEB-INF/classes`**
  - Contains the classes that make the web application.
  - For example:

**`webapps/examples/WEB-INF/classes/HelloWorldExample.class`**

# APP/WEB-INF/web.xml

- **`webapps/APP/WEB-INF/web.xml`**
  - Contains the description of the servlets int the APP directory
  - **web.xml** is the “glue” that tells tomcat what servlet classes will process what requests.
  - For example:

    **`webapps/examples/WEB-INF/web.xml`**
  - The <servlet> section in **`web.xml`** lists the servlets available in this directory.

```
<servlet>
 <servlet-name>
   HelloWorldExample
 </servlet-name>
 <servlet-class>
   HelloWorldExample2
 </servlet-class>
 </servlet>
```

# APP/WEB-INF/web.xml

- The <servlet-mapping> sections contains the mappings from the URL to servlet.

```
<servlet-mapping>
    <servlet-name>
        HelloWorldExample
    </servlet-name>
    <url-pattern>
        /servlets/servlet/HelloWorldExample
     </url-pattern>
</servlet-mapping>
```

# Another example:myapp.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import util.HTMLFilter;

public class myapp extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse
   response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Request Parameters Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>Request Parameters Example</h3>");
        out.println("Parameters in this request:<br>");
```

# Another example: myapp.java

```
String firstName = request.getParameter("firstname");
String lastName = request.getParameter("lastname");

if (firstName != null || lastName != null) {
    out.println("First Name:");
    out.println(" = " + HTMLFilter.filter(firstName) +
                "<br>");
    out.println("Last Name:");
    out.println(" = " + HTMLFilter.filter(lastName));
} else {
    out.println("No Parameters, Please enter some");
}
```

# Another example: myapp.java

```
    out.println("<P>");
         out.print("<form action=\"");
         out.print("myapp\" ");
         out.println("method=POST>");
         out.println("First Name:");
         out.println("<input type=text size=20 name=firstname>");
         out.println("<br>");
         out.println("Last Name:");
         out.println("<input type=text size=20 name=lastname>");
         out.println("<br>");
         out.println("<input type=submit>");
         out.println("</form>");
         out.println("</body>");
         out.println("</html>");
     }

     public void doPost(HttpServletRequest request, HttpServletResponse res)
     throws IOException, ServletException
     {
         doGet(request, res);
     }

}
```

# webapps/myapp/WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
      app_2_5.xsd"
   version="2.5">

    <description>
       myapp
    </description>

    <display-name>myapp</display-name>

    <servlet>
      <servlet-name>myapp</servlet-name>
      <servlet-class>myapp</servlet-class>
    </servlet>

    <servlet-mapping>
      <servlet-name>myapp</servlet-name>
      <url-pattern>/myapp</url-pattern>
    </servlet-mapping>

</web-app>
```

# Installing myapp.java

- Create
  - webapps/myapp/WEB-INF/web.xml
  - webapps/myapp/WEB-INF/classes/myapp.java
  - mkdir webapps/myapp/WEB-INF/classes/util
- Copy

  copy examples\WEB-INF\classes\util myapp\WEB-INF\classes\util
- Run

```
cd webapps/myapp/WEB-INF/classes
set CLASSPATH=C:\apache-tomcat-7.0.8\lib\servlet-api.jar;.
javac myapp.java
```

# Running myapp

- Run your web browser
- Connect to
  - http://localhost:8080/myapp/myapp

# Collections and Generics

# Collections

- A Collection is a class that is used to store a sequence of objects of the same kind.
- A collection has two fundamental methods:

```
public interface Collection<E>
{
    boolean add(E element);
   Iterator<E> iterator();
   ...
}
```

- The ***add*** method is used to add elements to the collection.
- The ***iterator*** method returns a helper object that can be used to iterate over the elements in the collection.

# Iterators

- The iterator interface has three methods:

```
public interface Iterator<E>
{
   E next();
   boolean hasNext();
   void remove();
}
```

  - By repeatedly calling the next method you can iterate over all the elements in the collection.

```
Collection<String> coll = …;
Iterator<String> it = coll.iterator();
While (it.hasNext())
{
  String element = it.next();
  // Do something with element
}
```

# Iterators

- As of Java 5.0 there is a more concise construction to iterate over a collection:

```
Collection<String> coll = …;
for (String element: coll) {
   // Do something with element
}
```

# Additional Collection Methods

- Also Collections define the following additional methods:

```
int size();
boolean isEmpty();
boolean contains(Object obj);
boolean remove(obj);
void clear();
T [] toArray()
```

# Types of Collections

- Java includes the following standard collection classes:
  - ArrayList – A sequence that can be indexed. It grows dynamically.
  - LinkedList - An ordered sequence that has efficient insertion and removal at any location.
  - HashMap – A data structre that stores key, value pairs.
  - and also ArrayDeque, HashSet, TreeSet EnumSet, LinkedHashSet, PriorityQueue, HashMap, TreeMap, EnumMap, LinkedHashMap, WeakHashMap etc.

# Using a LinkedList

- Creating a LinkedList of type String

  LinkedList<String> students = new LinkedList<String>;
- Adding elements to the list.

  students.add(“Peter”);

  students.add(“Mary”);

  students.add(“John”);
- Iterating over the LinkedList

```
for (String student: students) {
  System.out.println(“Student: “+ student);
}
```

# Using a HashMap

- HashMaps store (key, value) pairs.

- Creating a HashMap that stores (name, student) pairs where name is a String and student is a reference to a Student object.

```
HashMap<String, Student> map = new HashMap<String, Student>();
```

- Storing information in the map:

```
Student s = new Student(
    “Peter Parker”, “Hawkings 456”, “CS390”);
map.put(“Peter Parker”, s );
```

- Getting information from the map:

```
Student s = map.get(“Peter Parker”);
```

- Iterating over the elements:

```
for (HashMap.Entry<String, Student> entry: students.entrySet()) {
  String name = entry.getKey();
  Student value = entry.getValue();
  // Do something with key, value
}
```

# Advanced Swing

# Swing History

- When Java 1.0 was introduced, Sun introduced a basic GUI package called Abstract Window Toolkit (AWT).
- This package delegated the window creation to the native GUI toolkit in each platform.
- It worked well for simple applications but not for complex GUIs.
- In 1996 Netscape and Sun joined forces to create Swing.
- Swing has few dependencies from the underlying GUI.
- It was included in Java 1.2 as part of the Java Foundation Class (JFC).

# JFrame

- The simplest top level window is called a JFrame.
- A JFrame contains a title, minimize, maximize box and content.
- A window is a class that extends JFrame

# SimpleFrameTest.java

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

/**
 * @version 1.32 2007-06-12
 * @author Cay Horstmann
 */
public class SimpleFrameTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               SimpleFrame frame = new SimpleFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# SimpleFrameTest.java

```
class SimpleFrame extends JFrame
{
   public SimpleFrame()
   {
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;
}
```

# SimpleFrameTest.java

- All Swing components should be configured in the “event dispatch thread”, that is the thread that passes events such as mouse clicks and keystrokes.

```
EventQueue.invokeLater(
    new Runnable()
    {
       public void run()
       {
          SimpleFrame frame = new SimpleFrame();
          frame.setDefaultCloseOperation(
                JFrame.EXIT_ON_CLOSE);
          frame.setVisible(true);
       }
    });
```

# SimpleFrameTest.java

- By default the close box does nothing.
- We need to set the default behavior for the close box to close the window.

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

# Drawing 2D Shapes

- To draw 2D shapes we need to overwrite the paintComponent method of a JFrame.
- The paintComponent is called when the window has to be repainted.
- The paintComponent receives a graphic context that we used to draw.

```
Graphics2D g2 = (Graphics2D) g;
Rectangle2D rect =
    new Rectangle2D.Double(leftX, topY,
                width, height);
g2.draw(rect);
```

# DrawTest.java

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

/**
 * @version 1.32 2007-04-14
 * @author Cay Horstmann
 */
public class DrawTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               DrawFrame frame = new DrawFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# DrawTest.java

```
/**
 * A frame that contains a panel with drawings
 */
class DrawFrame extends JFrame
{
   public DrawFrame()
   {
      setTitle("DrawTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add panel to frame

      DrawComponent component = new DrawComponent();
      add(component);
   }

   public static final int DEFAULT_WIDTH = 400;
   public static final int DEFAULT_HEIGHT = 400;
}
```

# DrawTest.java

```
/**
 * A component that displays rectangles and ellipses.
 */
class DrawComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;

      // draw a rectangle

      double leftX = 100;
      double topY = 100;
      double width = 200;
      double height = 150;

      Rectangle2D rect =
        new Rectangle2D.Double(leftX, topY, width, height);
      g2.draw(rect);
```

# DrawTest.java

```
      // draw the enclosed ellipse

      Ellipse2D ellipse = new Ellipse2D.Double();
      ellipse.setFrame(rect);
      g2.draw(ellipse);

      // draw a diagonal line

      g2.draw(new Line2D.Double(leftX, topY,
                           leftX + width, topY + height));

      // draw a circle with the same center

      double centerX = rect.getCenterX();
      double centerY = rect.getCenterY();
      double radius = 150;

      Ellipse2D circle = new Ellipse2D.Double();
      circle.setFrameFromCenter(centerX, centerY,
                     centerX + radius, centerY + radius);
      g2.draw(circle);
   }
}
```

# Drawing Text

- To draw text we also rewrite the paintComponent method.
- We first set the font

  Font f = new Font(“Serif”, Font.BOLD, 36);

  g2.setFont(f)
- The we use the method

  g2.drawString( str, x, y);

# FontTest.java

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

/**
 * @version 1.33 2007-04-14
 * @author Cay Horstmann
 */
public class FontTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               FontFrame frame = new FontFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# FontTest.java

```
/**
 * A frame with a text message component
 */
class FontFrame extends JFrame
{
   public FontFrame()
   {
      setTitle("FontTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add component to frame

      FontComponent component = new FontComponent();
      add(component);
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;
}
```

# FontTest.java

```
/**
 * A component that shows a centered message in a box.
 */
class FontComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;

      String message = "Hello, World!";

      Font f = new Font("Serif", Font.BOLD, 36);
      g2.setFont(f);

      // measure the size of the message

      FontRenderContext context = g2.getFontRenderContext();
      Rectangle2D bounds = f.getStringBounds(message, context);
```

# FontTest.java

```
    // set (x,y) = top left corner of text

    double x = (getWidth() - bounds.getWidth()) / 2;
    double y = (getHeight() - bounds.getHeight()) / 2;

    // add ascent to y to reach the baseline

    double ascent = -bounds.getY();
    double baseY = y + ascent;

    // draw the message

    g2.drawString(message, (int) x, (int) baseY);

    g2.setPaint(Color.LIGHT_GRAY);

    // draw the baseline

    g2.draw(new Line2D.Double(x, baseY, x + bounds.getWidth(), baseY));

    // draw the enclosing rectangle

    Rectangle2D rect = new Rectangle2D.Double(x, y, bounds.getWidth(), bounds.getHeight());
    g2.draw(rect);
  }
}
```

# ActionListener objects

- An ActionListener is an object that defines a method called actionPerformed() that is called when an event is produced.
- In the case of buttons, the ActionListener object is added to the button by calling addActionLister().

```
ActionListener listener = "…";
JButton button = new JButton("Ok");
Button.addActionListener(listener);
```

- The Program ButtonTest shows Buttons and ActionListeners.

# ButtonTest.java

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

/**
 * @version 1.33 2007-06-12
 * @author Cay Horstmann
 */
public class ButtonTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               ButtonFrame frame = new ButtonFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# ButtonTest.java

```
/**
 * A frame with a button panel
 */
class ButtonFrame extends JFrame
{
   public ButtonFrame()
   {
      setTitle("ButtonTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // create buttons
      JButton yellowButton = new JButton("Yellow");
      JButton blueButton = new JButton("Blue");
      JButton redButton = new JButton("Red");

      buttonPanel = new JPanel();

      // add buttons to panel
      buttonPanel.add(yellowButton);
      buttonPanel.add(blueButton);
      buttonPanel.add(redButton);
```

# ButtonTest.java

```
   // add panel to frame
   add(buttonPanel);

   // create button actions
   ColorAction yellowAction = new ColorAction(Color.YELLOW);
   ColorAction blueAction = new ColorAction(Color.BLUE);
   ColorAction redAction = new ColorAction(Color.RED);

   // associate actions with buttons
   yellowButton.addActionListener(yellowAction);
   blueButton.addActionListener(blueAction);
   redButton.addActionListener(redAction);
}
```

# ButtonTest.java

```
/**
    * An action listener that sets the panel's background color.
    */
   private class ColorAction implements ActionListener
   {
      public ColorAction(Color c)
      {
         backgroundColor = c;
      }

      public void actionPerformed(ActionEvent event)
      {
         buttonPanel.setBackground(backgroundColor);
      }

      private Color backgroundColor;
   }

   private JPanel buttonPanel;

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;
}
```

# Changing Look and Feel

- The appearance of the Swing components can be changed by changing the Look and Feel.
- For example, Swing may look like a windows component in Windows, or like a Macintosh window in MacOS etc.

# Changing Look and Feel

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

public static void setNativeLookAndFeel() {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
      System.out.println("Error setting native LAF: " + e);
    }
  }

  public static void setJavaLookAndFeel() {
    try {
      UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    } catch(Exception e) {
      System.out.println("Error setting Java LAF: " + e);
    }
  }
```

# Adapters

- An Adapter object is like an ActionListener object but instead it lists more than one actionPerformed  method for different events.
- For instance, the MouseAdapter lists the functions:
  - mousePressed(MouseEvent e)
  - mouseClicked(MouseEvent e)

# Handling Mouse Events

- To handle mouse events, we need to subclass the MouseAdapter and the MouseMotionListener.
- In these classes we overwrite the methods:
- MouseAdapter:
  - mousePressed, mouseClicked
- MouseMotionListener:
  - mouseMoved
  - mouseDragged

# MouseTest.java

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

/**
 * @version 1.32 2007-06-12
 * @author Cay Horstmann
 */
public class MouseTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               MouseFrame frame = new MouseFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# MouseTest.java

```
/**
 * A frame containing a panel for testing mouse operations
 */
class MouseFrame extends JFrame
{
   public MouseFrame()
   {
      setTitle("MouseTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add component to frame

      MouseComponent component = new MouseComponent();
      add(component);
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;
}
```

# MouseTest.java

```
/**
 * A component with mouse operations for adding and removing squares.
 */
class MouseComponent extends JComponent
{
   public MouseComponent()
   {
      squares = new ArrayList<Rectangle2D>();
      current = null;

      addMouseListener(new MouseHandler());
      addMouseMotionListener(new MouseMotionHandler());
   }

   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;

      // draw all squares
      for (Rectangle2D r : squares)
         g2.draw(r);
   }
```

# MouseTest.java

```
/**
   * Finds the first square containing a point.
   * @param p a point
   * @return the first square that contains p
   */
  public Rectangle2D find(Point2D p)
  {
     for (Rectangle2D r : squares)
     {
        if (r.contains(p)) return r;
     }
     return null;
  }
```

# MouseTest.java

```
/**
 * Adds a square to the collection.
 * @param p the center of the square
 */
public void add(Point2D p)
{
   double x = p.getX();
   double y = p.getY();

   current = new Rectangle2D.Double(x - SIDELENGTH / 2,
         y - SIDELENGTH / 2, SIDELENGTH,
         SIDELENGTH);
   squares.add(current);
   repaint();
}
```

# MouseTest.java

```
/**
 * Removes a square from the collection.
 * @param s the square to remove
 */
public void remove(Rectangle2D s)
{
   if (s == null) return;
   if (s == current) current = null;
   squares.remove(s);
   repaint();
}

private static final int SIDELENGTH = 10;
private ArrayList<Rectangle2D> squares;
private Rectangle2D current;

// the square containing the mouse cursor
```

# MouseTest.java

```
private class MouseHandler extends MouseAdapter
   {
      public void mousePressed(MouseEvent event)
      {
         // add a new square if the cursor isn't inside a square
         current = find(event.getPoint());
         if (current == null) add(event.getPoint());
      }

      public void mouseClicked(MouseEvent event)
      {
         // remove the current square if double clicked
         current = find(event.getPoint());
         if (current != null && event.getClickCount() >= 2) remove(current);
      }
   }
```

# MouseTest.java

```
private class MouseMotionHandler implements MouseMotionListener
  {
    public void mouseMoved(MouseEvent event)
    {
      // set the mouse cursor to cross hairs if it is inside
      // a rectangle

      if (find(event.getPoint()) == null) setCursor(Cursor.getDefaultCursor());
      else setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
    }

    public void mouseDragged(MouseEvent event)
    {
      if (current != null)
      {
        int x = event.getX();
        int y = event.getY();

        // drag the current rectangle to center it at (x, y)
        current.setFrame(x - SIDELENGTH / 2, y - SIDELENGTH / 2, SIDELENGTH, SIDELENGTH);
        repaint();
      }
    }
  }
}
```

# Layout Managers

- A Layout Manager is an object that helps you control where to place GUI components.
- There are three basic Layout managers
  - FlowLayoutManager – Places components one after the other. This is the default of the JPanel.
  - BorderLayoutManager – Allows placing objects in the North, West, South, East and Center of the frame.
  - Grid Layout – Divides the frame in rows and columns. It uses the grid to place the elements.

# Calculator.java

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

/**
 * @version 1.33 2007-06-12
 * @author Cay Horstmann
 */
public class Calculator
{
   public static void main(String[] args)
   {
         EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               CalculatorFrame frame = new CalculatorFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# Calculator.java

```
/**
 * A frame with a calculator panel.
 */
class CalculatorFrame extends JFrame
{
   public CalculatorFrame()
   {
      setTitle("Calculator");
      CalculatorPanel panel = new CalculatorPanel();
      add(panel);
      pack();
   }
}

/**
 * A panel with calculator buttons and a result display.
 */
class CalculatorPanel extends JPanel
{
   public CalculatorPanel()
   {
      setLayout(new BorderLayout());
```

# Calculator.java

```
result = 0;
lastCommand = "=";
start = true;

// add the display

display = new JButton("0");
display.setEnabled(false);
add(display, BorderLayout.NORTH);

ActionListener insert = new InsertAction();
ActionListener command = new CommandAction();

// add the buttons in a 4 x 4 grid

panel = new JPanel();
panel.setLayout(new GridLayout(4, 4));
```

# Calculator.java

```
    addButton("7", insert);
    addButton("8", insert);
    addButton("9", insert);
    addButton("/", command);

    addButton("4", insert);
    addButton("5", insert);
    addButton("6", insert);
    addButton("*", command);

    addButton("1", insert);
    addButton("2", insert);
    addButton("3", insert);
    addButton("-", command);

    addButton("0", insert);
    addButton(".", insert);
    addButton("=", command);
    addButton("+", command);

    add(panel, BorderLayout.CENTER);
}
```

# Calculator.java

```
/**
   * Adds a button to the center panel.
   * @param label the button label
   * @param listener the button listener
   */
  private void addButton(String label, ActionListener listener)
  {
     JButton button = new JButton(label);
     button.addActionListener(listener);
     panel.add(button);
  }

  /**
   * This action inserts the button action string to the end of the display text.
   */
  private class InsertAction implements ActionListener
  {
     public void actionPerformed(ActionEvent event)
     {
        String input = event.getActionCommand();
        if (start)
        {
           display.setText("");
           start = false;
        }
        display.setText(display.getText() + input);
     }
  }
```

# Calculator.java

```
/**
    * This action executes the command that the button action string denotes.
    */
   private class CommandAction implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         String command = event.getActionCommand();

         if (start)
         {
            if (command.equals("-"))
            {
               display.setText(command);
               start = false;
            }
            else lastCommand = command;
         }
         else
         {
            calculate(Double.parseDouble(display.getText()));
            lastCommand = command;
            start = true;
         }
      }
   }
```

# Calculator.java

```
   /**
    * Carries out the pending calculation.
    * @param x the value to be accumulated with the prior result.
    */
   public void calculate(double x)
   {
      if (lastCommand.equals("+")) result += x;
      else if (lastCommand.equals("-")) result -= x;
      else if (lastCommand.equals("*")) result *= x;
      else if (lastCommand.equals("/")) result /= x;
      else if (lastCommand.equals("=")) result = x;
      display.setText("" + result);
   }

   private JButton display;
   private JPanel panel;
   private double result;
   private String lastCommand;
   private boolean start;
}
```

# Text Input

- To input text you can use a:
  - JTextField – Input only one line
  - JTextArea – Input multiple lines

```
JPanel panel = new JPanel();
JTextField textField = new JTextField(“default value”, 20);
panel.add(textField)
```

- To get the value use textField.getText().

# Labels

- A JLabel is a component that show text.
- They usually appear at the left of a TextField.

```
JLabel label =
   new JLabel(“User name: “,JLabel.RIGHT);
panel.add(label);
```

# Text Areas

- A JTextArea allow the input multiple lines.

```
// Create a text area with 8 rows of 30 columns.
JTextArea textArea = new JTextArea(8, 40);
panel.add(textArea);
```

- By default a text area does not have scroll bars. You have to add them.

```
textArea = new JTextArea(8,40)
JScrollPane scrollPane = new JScrollPAne(textArea);
panel.add(scrollPane);
```

# TextComponentTest.java

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

/**
 * @version 1.40 2007-04-27
 * @author Cay Horstmann
 */
public class TextComponentTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               TextComponentFrame frame = new TextComponentFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# TextComponentTest.java

```
/**
 * A frame with sample text components.
 */
class TextComponentFrame extends JFrame
{
   public TextComponentFrame()
   {
      setTitle("TextComponentTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      final JTextField textField = new JTextField();
      final JPasswordField passwordField = new JPasswordField();

      JPanel northPanel = new JPanel();
      northPanel.setLayout(new GridLayout(2, 2));
      northPanel.add(new JLabel("User name: ", SwingConstants.RIGHT));
      northPanel.add(textField);
      northPanel.add(new JLabel("Password: ", SwingConstants.RIGHT));
      northPanel.add(passwordField);
```

# TextComponentTest.java

```
add(northPanel, BorderLayout.NORTH);

 final JTextArea textArea = new JTextArea(8, 40);
 JScrollPane scrollPane = new JScrollPane(textArea);

 add(scrollPane, BorderLayout.CENTER);

 // add button to append text into the text area

 JPanel southPanel = new JPanel();

 JButton insertButton = new JButton("Insert");
 southPanel.add(insertButton);
 insertButton.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent event)
      {
         textArea.append("User name: " + textField.getText() + " Password: "
               + new String(passwordField.getPassword()) + "\n");
      }
   });
```

# TextComponentTest.java

```
        add(southPanel, BorderLayout.SOUTH);
    }

    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 300;
}
```

# Checkboxes

- Used to turn on/off different options.

  JCheckBox(“label”);
- To initialize

  boldCheckBox = new JCheckBox(“Bold”);

  boldCheckBox.setSelected(true);
- Then add Listener

  ActionListener listener = ….

  boldCheckBox.addActionListener(listener);

  add(boldCheckBox);

# CheckBoxTest

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

/**
 * @version 1.33 2007-06-12
 * @author Cay Horstmann
 */
public class CheckBoxTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               CheckBoxFrame frame = new CheckBoxFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# CheckBoxTest

```
/**
 * A frame with a sample text label and check boxes for selecting font attributes.
 */
class CheckBoxFrame extends JFrame
{
   public CheckBoxFrame()
   {
      setTitle("CheckBoxTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add the sample text label

      label = new JLabel("The quick brown fox jumps over the lazy dog.");
      label.setFont(new Font("Serif", Font.PLAIN, FONTSIZE));
      add(label, BorderLayout.CENTER);

      // this listener sets the font attribute of
      // the label to the check box state

      ActionListener listener = new ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               int mode = 0;
               if (bold.isSelected()) mode += Font.BOLD;
               if (italic.isSelected()) mode += Font.ITALIC;
               label.setFont(new Font("Serif", mode, FONTSIZE));
            }
         };
```

# CheckBoxTest

```
// add the check boxes

      JPanel buttonPanel = new JPanel();

      bold = new JCheckBox("Bold");
      bold.addActionListener(listener);
      buttonPanel.add(bold);

      italic = new JCheckBox("Italic");
      italic.addActionListener(listener);
      buttonPanel.add(italic);

      add(buttonPanel, BorderLayout.SOUTH);
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;

   private JLabel label;
   private JCheckBox bold;
   private JCheckBox italic;

   private static final int FONTSIZE = 12;
}
```

# RadioButtons

- Used to choose one option among many.
- You need first to define one button group and then add radioButtons to the group.
- Only one of the radio buttons in the group can be chosen.

```
ButtonGroup group = new ButtonGroup();
JRadioButton small = new
            JRadioButton(“Small”, false);
group.add(small);
JRadioButton big = new
            JRadioButton(“Big”, false);
group.add(big);
```

# RadioButtonTest

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

/**
 * @version 1.33 2007-06-12
 * @author Cay Horstmann
 */
public class RadioButtonTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               RadioButtonFrame frame = new RadioButtonFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# RadioButtonTest

```
/**
 * A frame with a sample text label and radio buttons for selecting font sizes.
 */
class RadioButtonFrame extends JFrame
{
   public RadioButtonFrame()
   {
      setTitle("RadioButtonTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add the sample text label

      label = new JLabel("The quick brown fox jumps over the lazy dog.");
      label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));
      add(label, BorderLayout.CENTER);

      // add the radio buttons

      buttonPanel = new JPanel();
      group = new ButtonGroup();

      addRadioButton("Small", 8);
      addRadioButton("Medium", 12);
      addRadioButton("Large", 18);
      addRadioButton("Extra large", 36);

      add(buttonPanel, BorderLayout.SOUTH);
   }
```

# RadioButtonTest

```
   /**
    * Adds a radio button that sets the font size of the sample text.
    * @param name the string to appear on the button
    * @param size the font size that this button sets
    */
   public void addRadioButton(String name, final int size)
   {
      boolean selected = size == DEFAULT_SIZE;
      JRadioButton button = new JRadioButton(name, selected);
      group.add(button);
      buttonPanel.add(button);

      // this listener sets the label font size

      ActionListener listener = new ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               // size refers to the final parameter of the addRadioButton
               // method
               label.setFont(new Font("Serif", Font.PLAIN, size));
            }
         };

      button.addActionListener(listener);
   }

   public static final int DEFAULT_WIDTH = 400;
   public static final int DEFAULT_HEIGHT = 200;

   private JPanel buttonPanel;
   private ButtonGroup group;
   private JLabel label;

   private static final int DEFAULT_SIZE = 12;
}
```

# Menus

- Menus are used on top of the windows to start an operation such as File->Open etc.
- First you need to create a JMenuBar and attach it to the frame.

```
JMenuBar menuBar = new JMenuBar();
frame.setMenuBar(menuBar);
```

- Then create a JMenu and add it to the menuBar.

```
JMenu editMenu = new JMenu(“Edit”);
 menuBar.add(editMenu);
```

- Then add the multiple items to the JMenu.

```
JMenuItem pasteItem = new JMenuItem(“Paste”);
editMenu.add(pasteItem);
```

# MenuTest

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

/**
 * @version 1.23 2007-05-30
 * @author Cay Horstmann
 */
public class MenuTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {
               MenuFrame frame = new MenuFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}
```

# MenuTest

```
/**
 * A frame with a sample menu bar.
 */
class MenuFrame extends JFrame
{
   public MenuFrame()
   {
      setTitle("MenuTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      JMenu fileMenu = new JMenu("File");
      fileMenu.add(new TestAction("New"));

      // demonstrate accelerators

      JMenuItem openItem = fileMenu.add(new TestAction("Open"));
      openItem.setAccelerator(KeyStroke.getKeyStroke("ctrl O"));

      fileMenu.addSeparator();

      saveAction = new TestAction("Save");
      JMenuItem saveItem = fileMenu.add(saveAction);
      saveItem.setAccelerator(KeyStroke.getKeyStroke("ctrl S"));
```

# MenuTest

```
saveAsAction = new TestAction("Save As");
fileMenu.add(saveAsAction);
fileMenu.addSeparator();

fileMenu.add(new AbstractAction("Exit")
   {
      public void actionPerformed(ActionEvent event)
      {
         System.exit(0);
      }
   });

// demonstrate check box and radio button menus

readonlyItem = new JCheckBoxMenuItem("Read-only");
readonlyItem.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent event)
      {
         boolean saveOk = !readonlyItem.isSelected();
         saveAction.setEnabled(saveOk);
         saveAsAction.setEnabled(saveOk);
      }
   });
```

# MenuTest

```
ButtonGroup group = new ButtonGroup();

JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("Insert");
insertItem.setSelected(true);
JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("Overtype");

group.add(insertItem);
group.add(overtypeItem);

// demonstrate icons

Action cutAction = new TestAction("Cut");
cutAction.putValue(Action.SMALL_ICON, new ImageIcon("cut.gif"));
Action copyAction = new TestAction("Copy");
copyAction.putValue(Action.SMALL_ICON, new ImageIcon("copy.gif"));
Action pasteAction = new TestAction("Paste");
pasteAction.putValue(Action.SMALL_ICON, new ImageIcon("paste.gif"));
```

# MenuTest

```
JMenu editMenu = new JMenu("Edit");
editMenu.add(cutAction);
editMenu.add(copyAction);
editMenu.add(pasteAction);

// demonstrate nested menus

JMenu optionMenu = new JMenu("Options");

optionMenu.add(readonlyItem);
optionMenu.addSeparator();
optionMenu.add(insertItem);
optionMenu.add(overtypeItem);
```

# MenuTest

```
editMenu.addSeparator();
 editMenu.add(optionMenu);

 // demonstrate mnemonics

 JMenu helpMenu = new JMenu("Help");
 helpMenu.setMnemonic('H');

 JMenuItem indexItem = new JMenuItem("Index");
 indexItem.setMnemonic('I');
 helpMenu.add(indexItem);

 // you can also add the mnemonic key to an action
 Action aboutAction = new TestAction("About");
 aboutAction.putValue(Action.MNEMONIC_KEY, new Integer('A'));
 helpMenu.add(aboutAction);

 // add all top-level menus to menu bar

 JMenuBar menuBar = new JMenuBar();
 setJMenuBar(menuBar);

 menuBar.add(fileMenu);
 menuBar.add(editMenu);
 menuBar.add(helpMenu);
```

# MenuTest

```
    // demonstrate pop-ups

      popup = new JPopupMenu();
      popup.add(cutAction);
      popup.add(copyAction);
      popup.add(pasteAction);

      JPanel panel = new JPanel();
      panel.setComponentPopupMenu(popup);
      add(panel);

      // the following line is a workaround for bug 4966109
      panel.addMouseListener(new MouseAdapter()
         {
         });
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;

   private Action saveAction;
   private Action saveAsAction;
   private JCheckBoxMenuItem readonlyItem;
   private JPopupMenu popup;
}
```

# MenuTest

```
/**
 * A sample action that prints the action name to System.out
 */
class TestAction extends AbstractAction
{
   public TestAction(String name)
   {
      super(name);
   }

   public void actionPerformed(ActionEvent event)
   {
      System.out.println(getValue(Action.NAME) + " selected.");
   }
}
```

# Advanced Components

- JList
  - It shows a number of items in a single box. You may select one or more than one of them at a time.
- JTable
  - It shows a table similar to a spreadsheet. The cells may contain pictures.
- JTree
  - It is used to show hierarchical  data. It is similar to the Windows Explorer.
- JEditorPane
  - It can display text in HTML or RTF (Rich Text Format).

# Advanced Components

- JSplitPane
  - It allows to split a frame into two and allow to resize it.
- JTabbedPane
  - It allows multiple frames use the same area using tabs.
- JDesktopPane
  - It allows multiple internal frames. It can be used to keep multiple windows open inside the frame.

# Advanced Components

- You can see a demo of all the Swing components by running:
  > cd C:\jdk1.6.0_18\demo\jfc\SwingSet2\src
  > javac *.java
  > java SwingSet2
- Also try the Java2D demos to see the Java extensions for drawing 2D shapes and characters all written in Java
  > cd C:\jdk1.6.0_18\demo\jfc\Java2D
  > java -jar Java2Demo.jar
- You will see that many of these demos use a lot of CPU or run slow in slow machines due that they are written in Java.
- However, as machines run faster this may no longer by an issue, it might be worth to try them in your next application.

# Java GC and Finalizers

- In Java Objects that are not reachable by the program are collected by the Garbage Collector.
- Two phases:
  - Mark – Determine objects that are reachable (marked)
  - Sweep – Reclaim the objects that ar eunreachable.
- If a class has a finalize() method, the unreachable object will be added to the “finalizer queue” in the reclaim phase.
- The finalizer method of the objects in the finalizer queue are called at the end of the GC.

# Finalizers Disadvantages

- Advantages:
  - Finalizers can be used to automatically release resources that the object is using  before it is collected.
- Disadvantages:
  - Finalizers may create new strong references that can prevent the object from being collected.
  - Finalizers run in nondeterministic order.
  - Finalizers are not guaranteed to be called.
  - Not recommended to manage scarce resources like file handles.

# Types of Object References

- Strong References
  - A reference to an object like “Object obj;”.
- SoftReference
  - A soft reference does not prevent the collection of an object.
  - The object is retained in memory until the memory is low.
  - You can use a SoftReference to an object to cache results or objects that we can keep in memory to accelerate access until memory is needed for other items.

# Types of Object References

- WeakReference
  - A weak reference does not prevent the collection of an object.
  - The object can be obtained from the weak reference if it has not been collected.
  - If the object the weakreference points to has been collected, the weakreference will return NULL.

# Types of Object References

- PhantomReference
  - It is used to manage resources associated to an object.
  - The resources associated to the object are cleaned up when the object is collected without the need of a finalizer.

# GC Algorithms

- Stop the World
  - The GC stops the program called “mutator” while the GC takes place.
  - The pauses may be very long.
- Incremental
  - The GC and the program alternate the use of the CPU to do part of the work.
  - The GC may run as part of the allocation calls.
- Concurrent
  - The GC and the program run in parallel.
  - Only a samall pause is needed at the end of the collection.
- Generational Collection
  - A type of GC that concentrates on the most recently allocated objects.
  - The most recently allocated objects are the most likely to be garbage.

# Generational Collection

- Generational Collection is a type of GC that concentrates on the most recently allocated objects.
- The most recently allocated objects are the most likely to be garbage.
- Several “quick” generational garbage collections can run before one “slow” full collection.
  - Minor collection – Takes place when a partition of the heap called young space is close to be full.
  - Full collection - Runs when the heap is close to be full.

# GC Options in the java command

- -verbose:gc
  - Output information about the gc running
- -XX:+PrintGCDetails
  - Prints more details of the gc
- -Xloggc:gc.log
  - Directs the output to a file gc.log instead of stdout
- -Xmx<mem-in-MB>m
  - Increases the maximum size of the heap to “mem-in-MB.
  - Example: -Xmx500m increases max heap to 500MB.
  - A 32bit JVM can use 1GB max heap in windows or 2GB in Linux.
  - If you want a larger heap, use a 64 bit JVM

# Final Review

- Java Networking
  - Internet Architecture
  - IP Addresses
  - Socket Class
  - Implementing a Client (SocketTest.java)
  - Implementing a Server (EchoServer.java)
  - Multi-threaded server (ThreadedEchoServer.java)
  - URL Connections (URLConnectionTest.java)
  - Pattern Matching (HrefMatch.java)

# Final Review

- Database Programming
  - JDBC Architecture
  - Structured Query Language (SQL)
    - SELECT/FROM/WHERE statement
    - UPDATE/SET/WHERE statement
    - DELETE FROM/WHERE statement
    - INSERT INTO statement
    - CREATE TABLE statement

# Final Review

- Database Programming (cont.)
  - Using JDBC in JAVA
    - MySQL installation
    - Mysql Java driver
  - TestDB.java

# Final Review

- Java 2 Platform, Enterprise Edition (J2EE)
  - J2EE Architecture
  - Apache Tomcat
    - Tomcat Directories
    - conf/server.xml
  - Servlets
    - Servlet Class
    - doGet/doPost
    - HelloWord.java
    - Installing a Servlet
    - The WEB-INF directory
    - myapp.java
    - myapp installation

# Final Review

- Collections and Generics
  - Collection Interface
  - Iterator Interface
  - Using an Iterator
  - Types of Collections
  - LinkedList
  - HashMap

# Final Review

- Advanced Swing
  - Swing History
  - JFrame (SimpleFrameTest.java)
  - Drawing 2D Shapes
    - DrawTest.java
    - FontTest.java
  - ActionListener objects
  - JButton (ButtonTest.java)
  - Handling Mouse Events (MouseTest.java)
  - Layout Managers (Calculator.java)

# Final Review

- JTextField, JLabel, JTextArea (TextComponentTest.java)
- CheckBoxes (CheckBoxTest.java)
- RadioButtons (RadioButtonTest.java)
- Menus (MenuTest.java)
- Advanced Components JList, JTable, JTree, JEditorPane, JSplitPane, JTabbedPane, JDesktopPane
- Running the SwigSet2 demo

# Final Review

- To Study
  - Slides and example programs
  - Final Review Homework due the day of the exam.
  - Projects
  - Core Java Books