Java Programming (Part 2)

Academic Year 2014-2015
Lecturer: Silas Majyambere
Computer Science Department

1. Introduction to GUI Programming

1.1 Applet Basics

Applets are small applications that are accessible on an Internet server, transported over the internet, automatically installed, and run as part of a web document.

A simple applet program

```
// Program for simple applet
import java.awt.*;
import java.applet.*;

public class AppletExample extends Applet
{
  public void paint(Graphics g)
 {
   g.drawString("A Simple Applet", 20, 20);
  }
}
```

In this program:

- The applet begins with two import statements.
- Applets interact with the user through AWT (Abstract Window Tool kit) not by console I/O classes
- Every applet program must be the sub class of Applet package.
- The class must be declared as publics, because it will be accessed by code the code that is outside the program.
- All the applet programs inherited from Applet.
- The paint method is used draw or redraws the output in an applet window.
- Graphics is one of the parameter of paint method.
- The graphics method outputs a string at the specified locations X, Y

To run applet

1. Use any Java compatible web browser
2. Use Applet viewer (a tool which comes along with SDK)

How to run applet programs

There are two ways to run the applet programs

1. We can have an individual applet code and an individual html code which calls the applet class file.
2. We can have a direct code which contains both applet and html code. (The html code will not be considered while the time of java compilation).

By using the first method to run the above program:

1. Compile AppletExample.java
2. Write a HTML file which can call the SimpleApplet.class and specify the size of the applet window you want.

A simple html code which can call the applet

```
<HTML>
<HEAD>
<TITLE>HELLO APPLET</TITLE>
</HEAD>
<BODY>
<APPLET CODE=”SimpleApplet.class” WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

3. Save the html file and open from any java compatible web browser. (or)
4. You can use appletviewer from your command line to execute this html file

By using the direct method to run the above program

The same program can be added with few lines of html code (which should mark as comments).

```
 // Program for simple applet using direct method
import java.awt.*;
import java.applet.*;

public class SimpleApplet extends Applet
{
  public void paint(Graphics g)
 {
    g.drawString("A Simple Applet", 20, 20);
  }
}
```

Output:

The output for these two methods gives you the same result. When you run this program you can see a applet window which prints you string “A Simple Applet” in the location of x=20 and y=20 in that window.

Applet Initialization and Termination

There are some methods provides the basic mechanism by which the browser or applet viewer interfaces to the applet and controls its execution. They are,

1. init()
   The init() method is the first method to be called.
2. start()
   The start() method is called after init().
3. paint()
   The paint() method is called each time your applet's output redrawn.
4. stop()
   The stop() method is called when a web browser leaves the HTML document containing the applet
5. destroy()
   The destroy() method called when your applet needs to be removed completely form memory.

In which the paint() method belongs to the AWT component class. All other four methods defined by Applet.

The following program is used to demonstrate this applet skeleton:

```
// simple applet that sets the foreground and background colors and outputs a string.

import java.awt.*;
import java.applet.*;

public class Sample extends Applet
{
  String msg;

  public void init()
{
    setBackground(Color.cyan);
    setForeground(Color.red);
    msg = "Inside init( ) --";
  }

  public void start()
{
    msg += " Inside start( ) --";
  }

  public void paint(Graphics g)
{
    msg += " Inside paint( ).";
    g.drawString(msg, 10, 30);
  }
}
```

Applets with status window

showstaus() method is used to display the message about what is occurring in the applet. The following program demonstrates showstatus().

```
// Using the Status Window.
import java.awt.*;
import java.applet.*;

public class StatusWindow extends Applet
{
  public void init()
{
    setBackground(Color.cyan);
  }
  public void paint(Graphics g)
{
    g.drawString("This is in the applet window.", 10, 20);
    showStatus("This is shown in the status window.");
  }
}
```

1.2 Graphics in Applet

The graphics class defines a number of drawing functions. Each shape can be drawn edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default.

Drawing lines: using drawLine(int startX, int startY,int endX, int endY)

```
// Draw lines
import java.awt.*;
import java.applet.*;

public class Lines extends Applet
 {
  public void paint(Graphics g)
{
    g.drawLine(0, 0, 100, 100);
    g.drawLine(0, 100, 100, 0);
    g.drawLine(40, 25, 250, 180);
    g.drawLine(75, 90, 400, 400);
    g.drawLine(20, 150, 400, 40);
    g.drawLine(5, 290, 80, 19);
  }
}
```

Drawing Rectangles: using 1. drawRect(int top, int left, int width, int height)
2. fillRect(int top, int left, int width, int height)

```
// Draw rectangles
import java.awt.*;
import java.applet.*;

public class Rectangles extends Applet
{
 public void paint(Graphics g)
{
   g.drawRect(10, 10, 60, 50);
   g.fillRect(100, 10, 60, 50);
   g.drawRoundRect(190, 10, 60, 50, 15, 15);
   g.fillRoundRect(70, 90, 140, 100, 30, 40);
 }
}
```

Drawing Ellipses: using 1. drawOval(int top, int left, int width, int height)
2. fillOval(int top, int left, int width, int height)

```
// Draw Ellipses
import java.awt.*;
import java.applet.*;

public class Ellipses extends Applet
{
 public void paint(Graphics g)
{
   g.drawOval(10, 10, 50, 50);
   g.fillOval(100, 10, 75, 50);
   g.drawOval(190, 10, 90, 30);
   g.fillOval(70, 90, 140, 100);
 }
}
```

Drawing Arcs: using
1. drawArc(int top, int left, int width, int height, int startangle, int sweepangle)
2. fillArc(int top, int left, int width, int height, int startangle, int sweepangle)

```
// Draw Arcs
import java.awt.*;
import java.applet.*;

public class Arcs extends Applet
{
 public void paint(Graphics g)
{
   g.drawArc(10, 40, 70, 70, 0, 75);
   g.fillArc(100, 40, 70, 70, 0, 75);
   g.drawArc(10, 100, 70, 80, 0, 175);
   g.fillArc(100, 100, 70, 90, 0, 270);
   g.drawArc(200, 80, 80, 80, 0, 180);
 }
}
```

Working with Color

Color defines to several constants to specify a number of common colors.

- To set the background color of an applet’s window use **setBackground( )**
- To set the foreground color of an applet’s window use **setForeground( )**

Syntax:

```
setBacground(Color newColor)
setForeground(Color newColor)
```

Here, newColor specifies the new color as shown as below.

Color.black	Color.megenta
Color.blue	Color.orange
Color.cyan	Color.pink
Color.green	Color.red
Color.gray	Color.white
Color.lightGray	Color.yellow
Color.darkGray	

There is one other form for Color constructor is:

Syntax:

```
Color(int red, int green, int blue)
```

This constructor takes three integers that specify the color as a mix of red, green and blue. These values must be between 0 and 255.

```
// Demonstrate color.
import java.awt.*;
import java.applet.*;

public class ColorDemo extends Applet
{
  public void paint(Graphics g)
{
    Color c1 = new Color(255, 100, 100);
    Color c2 = new Color(100, 255, 100);
    Color c3 = new Color(100, 100, 255);

    g.setColor(c1);
    g.drawLine(0, 0, 100, 100);
    g.drawLine(0, 100, 100, 0);

    g.setColor(c2);
    g.drawLine(40, 25, 250, 180);
    g.drawLine(75, 90, 400, 400);

    g.setColor(c3);
    g.drawLine(20, 150, 400, 40);
    g.drawLine(5, 290, 80, 19);

    g.setColor(Color.red);
    g.drawOval(10, 10, 50, 50);
    g.fillOval(70, 90, 140, 100);

    g.setColor(Color.blue);
    g.drawOval(190, 10, 90, 30);
    g.drawRect(10, 10, 60, 50);

    g.setColor(Color.cyan);
    g.fillRect(100, 10, 60, 50);
    g.drawRoundRect(190, 10, 60, 50, 15, 15);
  }
}
```

Working with Fonts

Fonts are encapsulated by the Font class. To obtain the fonts which are available in the system, we can use **getAvailableFoontFamilyNames()** method defined by **GraphicsEnvironment** class.

```
// Display Fonts
import java.applet.*;
import java.awt.*;

public class ShowFonts extends Applet
{
  public void paint(Graphics g)
{
    String msg = "";
    String FontList[];
    GraphicsEnvironment ge =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
    FontList = ge.getAvailableFontFamilyNames();
    for(int i = 0; i < FontList.length; i++)
      msg += FontList[i] + " ";
    g.drawString(msg, 4, 16);
  }
}
```

1.3 Event Handling in Applet

Event

An event is an object that describes a state of change in a source.

Event Source

A source is an object that generates an event.

Event Listeners

A listener is an object that is notified when an event occurs.

Handling Mouse Events

To handle mouse events there are two listeners used.

- MouseListener
- MouseMotionListener

The MouseEvents

There are different mouse events which override with other events. They are

- MouseEntered

- MouseExited
- MouseClicked
- MousePressed
- MouseReleased
- MouseDragged
- MouseMoved

The following program demonstrates the mouse event handlers

```
// Demonstrate the mouse event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class MouseEvents extends Applet implements MouseListener,
MouseMotionListener
 {
  String msg = "";
  int mouseX = 0, mouseY = 0;

  public void init()
{
    addMouseListener(this);
    addMouseMotionListener(this);
  }

  public void mouseClicked(MouseEvent me)
{
   mouseX = 0;
   mouseY = 10;
   msg = "Mouse clicked.";
   repaint();
  }

  public void mouseEntered(MouseEvent me)
{
   mouseX = 0;
   mouseY = 10;
   msg = "Mouse entered.";
   repaint();
  }

  public void mouseExited(MouseEvent me)
{
   mouseX = 0;
   mouseY = 10;
   msg = "Mouse exited.";
   repaint();
  }

  public void mousePressed(MouseEvent me)
{
   mouseX = me.getX();
```

```
    mouseY = me.getY();
    msg = "Down";
    repaint();
  }

  public void mouseReleased(MouseEvent me)
{
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "Up";
    repaint();
  }

  public void mouseDragged(MouseEvent me)
{
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "*";
    showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
    repaint();
  }

  public void mouseMoved(MouseEvent me)
{
    showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
  }

  public void paint(Graphics g)
{
    g.drawString(msg, mouseX, mouseY);
  }
}
```

1.1 Controls in Applet

The AWT supports the following types of controls:

- Labels
- Push Buttons
- Check Boxes
- Choice Lists
- Lists
- Scroll Bars
- Text Editing

Labels

A label is an object type of Label, and it contains a string which it displays. Labels are passive controls that do not support any interaction with the user. The string in the default label is left-justified. The alignment can be modified by using Label. LEFT, Label.RIGHT and Label.CENTER.

The following example creates three labels and adds them to an applet.

```
// Demonstrate Labels
import java.awt.*;
import java.applet.*;

public class LabelDemo extends Applet
{
  public void init()
{
    Label one = new Label("One");
    Label two = new Label("Two");
    Label three = new Label("Three");
    add(one);
    add(two);
    add(three);
  }
}
```

Buttons (Implements ActionListener)

A bush button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button. The label of a button is used to determine which button has been pressed. The label is obtained by calling the getActionCommand() method on the ActionEvent object passed to actionPerformed(). Sample program to demonstrate buttons

```
// Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class ButtonDemo extends Applet implements ActionListener
{
  String msg = "";
  Button yes, no, maybe;
  public void init()
{
    yes = new Button("Yes");
    no = new Button("No");
    maybe = new Button("Undecided");
    add(yes);
    add(no);
    add(maybe);
    yes.addActionListener(this);
    no.addActionListener(this);
    maybe.addActionListener(this);
  }
  public void actionPerformed(ActionEvent ae)
{
    String str = ae.getActionCommand();
    if(str.equals("Yes"))
{
      msg = "You pressed Yes.";
    }
    else if(str.equals("No"))
{
      msg = "You pressed No.";
    }
    else
 {
      msg = "You pressed Undecided.";
    }
    repaint();
  }
  public void paint(Graphics g)
{
     g.drawString(msg, 6, 100);
  }
}
```

Check Boxes (Implements ItemListener)

A check box is an control that is used to turn an option on or off. To retrieve the current state of a box, call getState(). To set this state, call setState(). You can obtain the current checkbox label by calling getLabel() and set the label by calling setLabel().

ItemListener interface defines the itemStateChanged method. An ItemEvent object is supplied as the argument to this method.

Demonstration of check boxes

```
// Demonstrate check boxes.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class CheckboxDemo extends Applet implements ItemListener
{
  String msg = "";
  Checkbox Win98, winNT, solaris, mac;
  public void init()
{
    Win98 = new Checkbox("Windows 98/XP", null, true);
    winNT = new Checkbox("Windows NT/2000");
    solaris = new Checkbox("Solaris");
    mac = new Checkbox("MacOS");
    add(Win98);
    add(winNT);
    add(solaris);
    add(mac);
    Win98.addItemListener(this);
    winNT.addItemListener(this);
    solaris.addItemListener(this);
    mac.addItemListener(this);
  }
```

```
 public void itemStateChanged(ItemEvent ie)
{
   repaint();
 }
  public void paint(Graphics g)
{
   msg = "Current state: ";
   g.drawString(msg, 6, 80);
   msg = "  Windows 98/XP: " + Win98.getState();
   g.drawString(msg, 6, 100);
   msg = "  Windows NT/2000: " + winNT.getState();
   g.drawString(msg, 6, 120);
   msg = "  Solaris: " + solaris.getState();
   g.drawString(msg, 6, 140);
   msg = "  MacOS: " + mac.getState();
   g.drawString(msg, 6, 160);
 }
}
```

Check Box Group (Implements ItemListener)

Check box group allows the check boxes can be selected at any one time. To determine which check box is currently selected by calling getSelectedCheckbox(). To set a check box by calling setSelectedCheckbox().

Here is a program that uses check boxes that are part of group.

```
// Demonstrate check box group.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class CBGroup extends Applet implements ItemListener
{
 String msg = "";
 Checkbox Win98, winNT, solaris, mac;
 CheckboxGroup cbg;
 public void init()
{
   cbg = new CheckboxGroup();
   Win98 = new Checkbox("Windows 98/XP", cbg, true);
   winNT = new Checkbox("Windows NT/2000", cbg, false);
   solaris = new Checkbox("Solaris", cbg, false);
   mac = new Checkbox("MacOS", cbg, false);
   add(Win98);
   add(winNT);
   add(solaris);
   add(mac);
   Win98.addItemListener(this);
   winNT.addItemListener(this);
   solaris.addItemListener(this);
   mac.addItemListener(this);
 }
 public void itemStateChanged(ItemEvent ie)
{
   repaint();
 }
 public void paint(Graphics g)
{
   msg = "Current selection: ";
   msg += cbg.getSelectedCheckbox().getLabel();
   g.drawString(msg, 6, 100);
 }
}
```

Text Field (Implements ActionListener)

The text field implements a single line text entry area, usually called an edit control. To obtain the string currently contained in the text field, call getText() and to set the text, call setText().

Here is an example that creates the classic username and password screen

```
// Demonstrate text field.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class TextFieldDemo extends Applet implements ActionListener
{
  TextField name, pass;
  public void init()
{
    Label namep = new Label("Name: ", Label.RIGHT);
    Label passp = new Label("Password: ", Label.RIGHT);
    name = new TextField(12);
    pass = new TextField(8);
    pass.setEchoChar('?');
    add(namep);
    add(name);
    add(passp);
    add(pass);
    name.addActionListener(this);
    pass.addActionListener(this);
  }
  public void actionPerformed(ActionEvent ae)
{
    repaint();
  }

  public void paint(Graphics g)
{
     g.drawString("Name: " + name.getText(), 6, 60);
     g.drawString("Password: " + pass.getText(), 6, 100);
  }
}
```

2. Swing Programming

2.1 Swing Basics

Java Foundation Classes (JFC) consists of number of class libraries that can be used to construct GUI programs. AWT and swing are the two libraries in JFC.

Swing is a set of classes that provides more powerful and flexible components which can have look and feel concept for platform-independent which again known as light weight elements.

Swing related classes are contained in javax.swing and its sub packages. Fundamental to swing is the JApplet class, which extends Applet. When adding a component to JApplet as like Applet we can't use add(). Instead of add(), JApplet using add() for the content pane.

Swing Classes

There are number of classes in swing packages. Some of them are,

Class	**Description**
ImageIcon	Encapsulates an icon
JApplet	The swing version of Applet
JButton	The swing push button class
JCheckBox	The swing check box class
JComboBox	The Swing combo box
JLabel	The swing version of label
JRadioButton	The swing version of a radio button
JScrollPane	Encapsulates a scrollable window
JTabbedPane	Encapsulates a tabbed window
JTable	Encapsulates a table-based control
JTextField	The wing version of a text field
JTree	Encapsulates a tree-based control

How to run Swing programs

1. Compose and compile the swing program with the extension .java
2. Execute the swing program as like the normal java program execution.

2.2 Controls in Swing

Icons and Labels

In swing icons are encapsulated by the ImageIcon class, which paints an icon from and image. Swing labels are instances of the JLabel class. It can display text and / or an icon.

The following example illustrates how to create and display a label containing both an icon and a string.

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

```
public class JLabelDemo extends JApplet
{
  public void init()
{
    Container contentPane = getContentPane();
    ImageIcon ii = new ImageIcon("france.gif");
    JLabel jl = new JLabel("France", ii, JLabel.CENTER);
    contentPane.add(jl);
  }
}
```

Text Fields

JTextFiled allows you to edit one line of text. It can be specified with the number of columns in the text field. The following example illustrates how to create a text field.

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

public class JTextFieldDemo extends JApplet
{
  JTextField jtf;
  public void init()
{
    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());
    jtf = new JTextField(15);
    contentPane.add(jtf);
  }
}
```

Buttons

The JButton class provides the functionality of a push button. JButton allows an icon a string, or both to be associated with the push button. The following example displays four push buttons and a text field.

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


  public class JButtonDemo extends JApplet implements ActionListener
{
  JTextField jtf;
  public void init()
{
    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());

    ImageIcon france = new ImageIcon("france.gif");
    JButton jb = new JButton(france);
    jb.setActionCommand("France");
    jb.addActionListener(this);
    contentPane.add(jb);
    ImageIcon germany = new ImageIcon("germany.gif");
    jb = new JButton(germany);
    jb.setActionCommand("Germany");
    jb.addActionListener(this);
    contentPane.add(jb);
    ImageIcon italy = new ImageIcon("italy.gif");
    jb = new JButton(italy);
    jb.setActionCommand("Italy");
    jb.addActionListener(this);
    contentPane.add(jb);
    ImageIcon japan = new ImageIcon("japan.gif");
    jb = new JButton(japan);
    jb.setActionCommand("Japan");
    jb.addActionListener(this);
    contentPane.add(jb);
    jtf = new JTextField(15);
    contentPane.add(jtf);
  }
  public void actionPerformed(ActionEvent ae)
{
    jtf.setText(ae.getActionCommand());
  }
}
```

Check Boxes

The JCheckBox class, which provides the functionality of check box. When a check box is selected or deselected, an item event is generated. This is handled by itemStateChanged().

The following example illustrates how to create an applet that displays four check boxes
and a text field.

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

public class JCheckBoxDemo extends JApplet implements ItemListener
{
  JTextField jtf;
  public void init()
{
    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());
    ImageIcon normal = new ImageIcon("normal.gif");
    ImageIcon rollover = new ImageIcon("rollover.gif");
    ImageIcon selected = new ImageIcon("selected.gif");
    JCheckBox cb = new JCheckBox("C", normal);
    cb.setRolloverIcon(rollover);
    cb.setSelectedIcon(selected);
    cb.addItemListener(this);
    contentPane.add(cb);
    cb = new JCheckBox("C++", normal);
    cb.setRolloverIcon(rollover);
    cb.setSelectedIcon(selected);
    cb.addItemListener(this);
    contentPane.add(cb);
    cb = new JCheckBox("Java", normal);
    cb.setRolloverIcon(rollover);
    cb.setSelectedIcon(selected);
    cb.addItemListener(this);
    contentPane.add(cb);
    cb = new JCheckBox("Perl", normal);
    cb.setRolloverIcon(rollover);
    cb.setSelectedIcon(selected);
    cb.addItemListener(this);
    contentPane.add(cb);
    jtf = new JTextField(15);
    contentPane.add(jtf);
  }
  public void itemStateChanged(ItemEvent ie)
{
    JCheckBox cb = (JCheckBox)ie.getItem();
    jtf.setText(cb.getText());
  }
}
```

Radio Buttons

Radio buttons are supported by JRadioButton class. Radio buttons must be configured into a group. The ButtonGroup class is instantiated to create a button group.
Radio button presses generate action events that are handled by actionPerformed(). The getActionCommand() method gets the text associated with a radio button.

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

public class JRadioButtonDemo extends JApplet implements ActionListener
{
  JTextField tf;
  public void init()
{
    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());
    JRadioButton b1 = new JRadioButton("A");
    b1.addActionListener(this);
    contentPane.add(b1);
    JRadioButton b2 = new JRadioButton("B");
    b2.addActionListener(this);
    contentPane.add(b2);
    JRadioButton b3 = new JRadioButton("C");
    b3.addActionListener(this);
    contentPane.add(b3);
    ButtonGroup bg = new ButtonGroup();
    bg.add(b1);
    bg.add(b2);
    bg.add(b3);

    tf = new JTextField(5);
    contentPane.add(tf);
  }
  public void actionPerformed(ActionEvent ae)
{
    tf.setText(ae.getActionCommand());
  }
}
```

Combo Boxes

Swing provides a combo box (a combination of a text field and a drop down list)
through the JComboBox class. A combo box normally displays one entry. It can also display a drop down list that allows a user to select a different entry. Items are added to
the list of choices via the addItem() method.

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

public class JComboBoxDemo extends JApplet implements ItemListener
{
  JLabel jl;
  ImageIcon france, germany, italy, japan;
  public void init()
{
    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());
    JComboBox jc = new JComboBox();
    jc.addItem("France");
    jc.addItem("Germany");
    jc.addItem("Italy");
    jc.addItem("Japan");
    jc.addItemListener(this);
    contentPane.add(jc);
    jl = new JLabel(new ImageIcon("france.gif"));
    contentPane.add(jl);
  }
  public void itemStateChanged(ItemEvent ie)
{
   String s = (String)ie.getItem();
   jl.setIcon(new ImageIcon(s + ".gif"));
  }
}
```

Tabbed Panes

A tabbed pane is a componenet that appears as a group of folders in a file or cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time. Tabbed panes are encapsulated by the JTabbedPane class. To add to the tab to the pane call, addTab().

```
import javax.swing.*;

  public class JTabbedPaneDemo extends JApplet
{
  public void init()
 {
   JTabbedPane jtp = new JTabbedPane();
   jtp.addTab("Cities", new CitiesPanel());
   jtp.addTab("Colors", new ColorsPanel());
   jtp.addTab("Flavors", new FlavorsPanel());
   getContentPane().add(jtp);
  }
}
class CitiesPanel extends JPanel
{
  public CitiesPanel()
 {
   JButton b1 = new JButton("New York");
   add(b1);
   JButton b2 = new JButton("London");
   add(b2);
   JButton b3 = new JButton("Hong Kong");
   add(b3);
   JButton b4 = new JButton("Tokyo");
   add(b4);
  }
}
class ColorsPanel extends JPanel
{
  public ColorsPanel()
 {
   JCheckBox cb1 = new JCheckBox("Red");
   add(cb1);
   JCheckBox cb2 = new JCheckBox("Green");
   add(cb2);
   JCheckBox cb3 = new JCheckBox("Blue");
   add(cb3);
  }
}
class FlavorsPanel extends JPanel
{
  public FlavorsPanel()
 {
   JComboBox jcb = new JComboBox();
    jcb.addItem("Vanilla");
    jcb.addItem("Chocolate");
    jcb.addItem("Strawberry");
    add(jcb);
  }
}
```

Swing Events

Listeners Supported by Swing Components
You can tell what kinds of events a component can fire by looking at the kinds of event listeners you can register on it. For example, the `JComboBox` class defines these listener registration methods:

- `addActionListener`
- `addItemListener`
- `addPopupMenuListener`

Thus, a combo box supports action, item, and popup menu listeners in addition to the listener methods it inherits from `JComponent`.

Listeners supported by Swing components fall into two categories:

- Listeners that All Swing Components Support
- Other Listeners that Swing Components Support

Listeners that All Swing Components Support

Because all Swing components descend from the AWT `Component` class, you can register the following listeners on any Swing component:

component listener
Listens for changes in the component's size, position, or visibility.

focus listener
Listens for whether the component gained or lost the keyboard focus.

key listener
Listens for key presses; key events are fired only by the component that has the current keyboard focus.

mouse listener
Listens for mouse clicks, mouse presses, mouse releases and mouse movement into or out of the component's drawing area.

mouse-motion listener
Listens for changes in the mouse cursor's position over the component.

mouse-wheel listener
Listens for mouse wheel movement over the component.

Hierarchy Listener
Listens for changes to a component's containment hierarchy of changed events.

Hierarchy Bounds Listener

Listens for changes to a component's containment hierarchy of moved and resized events.

All Swing components descend from the AWT `Container` class, but many of them are not used as containers. So, technically speaking, any Swing component can fire container events, which notify listeners that a component has been added to or removed from the container. Realistically speaking, however, only containers (such as panels and frames) and compound components (such as combo boxes) typically fire container events.

`JComponent` provides support for three more listener types. You can register an ancestor listener to be notified when a component's containment ancestors are added to or removed from a container, hidden, made visible, or moved. This listener type is an implementation detail which predated hierarchy listeners.

The other two listener types are part of the Swing components' conformance to the JavaBeans™ specification. Among other things, this means that Swing components support bound and constrained properties and notifies listeners of changes to the properties. Property change listeners listen for changes to bound properties and are used by several Swing components, such as formatted text fields, to track changes on a component's bound properties. Also, property change listeners, as well as vetoable change listeners are used by builder tools to listen for changes on constrained properties. For more information refer to the Properties lesson in the JavaBeans trail.

Other Listeners that Swing Components Support

The following table lists Swing components and the specialized listeners they support, not including listeners supported by all `Component`s, `Container`s, or `JComponent`s. In many cases, the events are fired directly from the component. In other cases, the events are fired from the component's data or selection model. To find out the details for the particular component and listener you are interested in, go first to the component how-to section, and then if necessary to the listener how-to section.

This table lists Swing components with their specialized listeners

Component	**Listener**							
	action	caret	change	document, undoable edit	item	list selection	window	other
button	✓		✓		✓			
check box	✓		✓		✓			
color chooser			✓					
combo box	✓				✓			
dialog							✓	
editor pane		✓		✓				hyperlink
file chooser	✓							
formatted text field	✓	✓		✓				
frame							✓	
internal frame								internal frame
list						✓		list data
menu								menu

menu item	✓		✓		✓			menu key menu drag mouse
option pane								
password field	✓	✓		✓				
popup menu								popup menu
progress bar			✓					
radio button	✓		✓		✓			
slider			✓					
spinner			✓					
tabbed pane			✓					
table						✓		table model table column model cell editor
text area		✓		✓				
text field	✓	✓		✓				
text pane		✓		✓				hyperlink
toggle button	✓		✓		✓			
tree								tree expansion tree will expand tree model tree selection
viewport (used by scrollpane)			✓					

4. Java Networking

4.1 Networking Basics

Networking classes in Java are defined in the **java.net** package. These networking classes encapsulate the socket paradigm pioneered in the Berkley Sockets.

Socket

A network socket can be used to implement reliable connections between hosts and Internet.

Client / Server

A server is anything that has some resource that can be shared. A client is simply any other entity that wants to gain access to a particular sever.

Reserved Sockets

TCP/IP reserves the lower 1024 ports for specific protocols. Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP and 119 is for net news.

Proxy Servers

A proxy server speaks the client side of a protocol to another server. A proxy server has the additional ability to filer certain requests or caches the results of those requests for future use.

Internet Addressing

Every computer on the internet has an address. An internet address is a number that uniquely identifies each computer on the net. Ex: 193.168.211.34

Domain Naming Service (DNS)

Domain naming service is used to refer the four numbers of an IP address described in network hierarchy as domain name of the machine’s name.

4.2 Java and Net

InetAddress

The InetAddress class is used to encapsulate both the numerical IP address and domain name. The **getLocalHost()** method return the InetAddress of the local host. The **getByName()** method returns an InetAddress for a host name passed to it. The **getAllByName()** method returns InetAddresses if it is more than one. If these methods are unable to resolve a hostname, they throw an **UnKnownHostException**.

The following example prints the address and names of the local machine.

```
// Demonstrate InetAddress.
import java.net.*;

class InetAddressTest
{
  public static void main(String args[]) throws UnknownHostException
{
    InetAddress Address = InetAddress.getLocalHost();
    System.out.println(Address);
    Address = InetAddress.getByName("yahoo.com");
    System.out.println(Address);
    InetAddress ia[] = InetAddress.getAllByName("www.nur.ac.rw");
    for (int i=0; i<ia.length; i++)
      System.out.println(ia[i]);
  }
}
```

Whois

Whois port allows to open a connections on the InterNIC server, sends the domain through the socket and prints the data that is returned. InterNIC will try to look up the domain as internet registered domain name, then send back the IP address and contact information for that site.

```
//Demonstrate Sockets.
import java.net.*;
import java.io.*;

class Whois
{
 public static void main(String args[]) throws Exception
{
   int c;
   Socket s = new Socket("internic.net", 43);
   InputStream in = s.getInputStream();
   OutputStream out = s.getOutputStream();
   String str = “nba.com”;
   byte buf[] = str.getBytes();
   out.write(buf);
   while ((c = in.read()) != -1) {
     System.out.print((char) c);
   }
   s.close();
 }
}
```

URL

The Uniform Resource Locator provides intelligible form to uniquely identify or address information on the Internet. The following example we create URL to google’s downloaded page and then examine its properties.

```
// Demonstrate URL.
import java.net.*;
class URLDemo
{
 public static void main(String args[]) throws MalformedURLException
{
   URL hp = new URL("http://www.osborne/downloads");

   System.out.println("Protocol: " + hp.getProtocol());
   System.out.println("Port: " + hp.getPort());
   System.out.println("Host: " + hp.getHost());
   System.out.println("File: " + hp.getFile());
   System.out.println("Ext:" + hp.toExternalForm());
 }
}
```

URLConnection

URLConnection is a general purpose class for accessing the attributes of a remote resource. In the following example, we create a URLConnection using the **openConnection()** method of a URL object and then use it to examine the properties of document or content.

```
// Demonstrate URLConnection.

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

class UCDemo
{
  public static void main(String args[]) throws Exception
{
    int c;
    URL hp = new URL("http://www.internic.net");
    URLConnection hpCon = hp.openConnection();

    long d = hpCon.getDate();
    if(d==0)
      System.out.println("No date information.");
    else
      System.out.println("Date: " + new Date(d));

    System.out.println("Content-Type: " + hpCon.getContentType());

    d = hpCon.getExpiration();
    if(d==0)
      System.out.println("No expiration information.");
    else
      System.out.println("Expires: " + new Date(d));

    d = hpCon.getLastModified();
    if(d==0)
      System.out.println("No last-modified information.");
    else
      System.out.println("Last-Modified: " + new Date(d));

    int len = hpCon.getContentLength();
    if(len == -1)
      System.out.println("Content length unavailable.");
    else
      System.out.println("Content-Length: " + len);

    if(len != 0) {
      System.out.println("=== Content ===");
      InputStream input = hpCon.getInputStream();

      int i = len;
      while (((c = input.read()) != -1)) { // && (--i > 0)) {
        System.out.print((char) c);
      }
      input.close();

    } else {
      System.out.println("No content available.");
    }

  }
}
```

Java Socket Programming

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

1. The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
2. The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
3. After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
4. The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a twoway communication protocol, so data can be sent across both streams at the same time. There are following usefull classes providing complete set of methods to implement sockets.

ServerSocket Class Methods:

The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests

The ServerSocket class has four constructors:

SN	Methods with Description
1	**public ServerSocket(int port) throws IOException** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
2	**public ServerSocket(int port, int backlog) throws IOException** Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
3	**public ServerSocket(int port, int backlog, InetAddress address) throws**

	IOException Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on
4	**public ServerSocket() throws IOException** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

Here are some of the common methods of the ServerSocket class:

SN	Methods with Description
1	**public int getLocalPort()** Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2	**public Socket accept() throws IOException** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely
3	**public void setSoTimeout(int timeout)** Sets the time-out value for how long the server socket waits for a client during the accept().
4	**public void bind(SocketAddress host, int backlog)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and server, and communication can begin.

Socket Class Methods:

The **java.net.Socket** class represents the socket that both the client and server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

The Socket class has five constructors that a client uses to connect to a server:

SN	Methods with Description

1	**public Socket(String host, int port) throws UnknownHostException, IOException.** This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
2	**public Socket(InetAddress host, int port) throws IOException** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** Connects to the specified host and port, creating a socket on the local host at the specified address and port.
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String
5	**public Socket()** Creates an unconnected socket. Use the connect() method to connect this socket to a server.

When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.

Some methods of interest in the Socket class are listed here. Notice that both the client and server have a Socket object, so these methods can be invoked by both the client and server.

SN	Methods with Description
1	**public void connect(SocketAddress host, int timeout) throws IOException** This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.
2	**public InetAddress getInetAddress()** This method returns the address of the other computer that this socket is connected to.
3	**public int getPort()** Returns the port the socket is bound to on the remote machine.
4	**public int getLocalPort()** Returns the port the socket is bound to on the local machine.
5	**public SocketAddress getRemoteSocketAddress()** Returns the address of the remote socket.
6	**public InputStream getInputStream() throws IOException** Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7	**public OutputStream getOutputStream() throws IOException** Returns the output stream of the socket. The output stream is connected to the

	input stream of the remote socket
8	**public void close() throws IOException** Closes the socket, which makes this Socket object no longer capable of connecting again to any server

InetAddress Class Methods:

This class represents an Internet Protocol (IP) address. Here are following usefull methods which you would need while doing socket programming:

SN	Methods with Description
1	**static InetAddress getByAddress(byte[] addr)** Returns an InetAddress object given the raw IP address .
2	**static InetAddress getByAddress(String host, byte[] addr)** Create an InetAddress based on the provided host name and IP address.
3	**static InetAddress getByName(String host)** Determines the IP address of a host, given the host's name.
4	**String getHostAddress()** Returns the IP address string in textual presentation.
5	**String getHostName()** Gets the host name for this IP address.
6	**static InetAddress InetAddress getLocalHost()** Returns the local host.
7	**String toString()** Converts this IP address to a String.

Socket Client Example:

The following GreetingClient is a client program that connects to a server by using a socket and sends a greeting, and then waits for a response.

```
// File Name GreetingClient.java

import java.net.*;
import java.io.*;

public class GreetingClient
{
   public static void main(String [] args)
   {
      String serverName = args[0];
      int port = Integer.parseInt(args[1]);
      try
      {
         System.out.println("Connecting to " + serverName
                             + " on port " + port);
         Socket client = new Socket(serverName, port);
         System.out.println("Just connected to "
                      + client.getRemoteSocketAddress());
         OutputStream outToServer = client.getOutputStream();
         DataOutputStream out =
                       new DataOutputStream(outToServer);

         out.writeUTF("Hello from "
                      + client.getLocalSocketAddress());
         InputStream inFromServer = client.getInputStream();
         DataInputStream in =
                        new DataInputStream(inFromServer);
         System.out.println("Server says " + in.readUTF());
         client.close();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

Socket Server Example:

The following GreetingServer program is an example of a server application that uses the Socket class to listen for clients on a port number specified by a command-line argument:

```
// File Name GreetingServer.java

import java.net.*;
import java.io.*;

public class GreetingServer extends Thread
{
   private ServerSocket serverSocket;

   public GreetingServer(int port) throws IOException
   {
      serverSocket = new ServerSocket(port);
      serverSocket.setSoTimeout(10000);
   }

   public void run()
   {
      while(true)
      {
         try
         {
            System.out.println("Waiting for client on port " +
            serverSocket.getLocalPort() + "...");
            Socket server = serverSocket.accept();
            System.out.println("Just connected to "
                  + server.getRemoteSocketAddress());
            DataInputStream in =
                  new DataInputStream(server.getInputStream());
            System.out.println(in.readUTF());
            DataOutputStream out =
                 new DataOutputStream(server.getOutputStream());
            out.writeUTF("Thank you for connecting to "
              + server.getLocalSocketAddress() + "\nGoodbye!");
            server.close();
         }catch(SocketTimeoutException s)
         {
            System.out.println("Socket timed out!");
            break;
         }catch(IOException e)
         {
            e.printStackTrace();
            break;
         }
      }
   }
   public static void main(String [] args)
   {
      int port = Integer.parseInt(args[0]);
      try
      {
         Thread t = new GreetingServer(port);
         t.start();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

Compile client and server and then start server as follows:

```
$ java GreetingServer 7890
Waiting for client on port 7890...
```

Check client program as follows:

```
$ java GreetingClient localhost 7890
Connecting to localhost on port 7890
Just connected to localhost/127.0.0.1:7890
Server says Thank you for connecting to /127.0.0.1:7890
Goodbye!
```

Java Exceptions Handling

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

To understand how exception handling works in Java, you need to understand the three categories of exceptions:

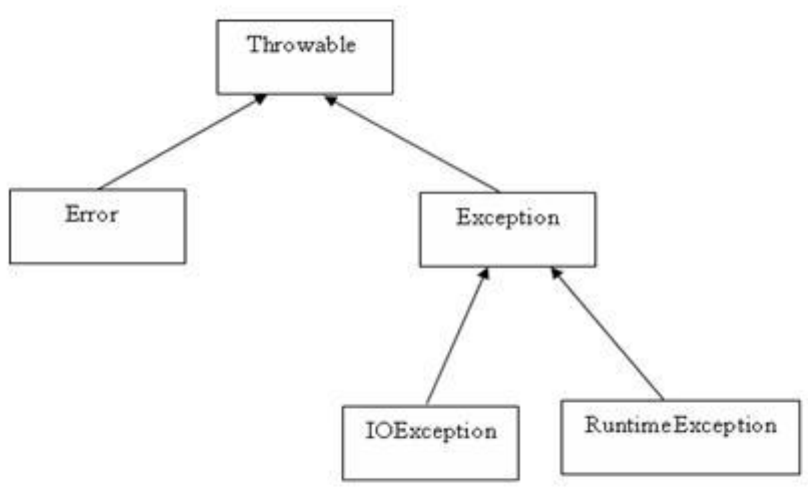
- **Checked exceptions:** Achecked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compliation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Hierarchy:

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses : IOException class and RuntimeException Class.



Here is a list of most common checked and unchecked Java's Built-in Exceptions.

Exceptions Methods:

Following is the list of important medthods available in the Throwable class.

SN	Methods with Description
1	**public String getMessage()** Returns a detailed message about the exception that has occurred. This message

	is initialized in the Throwable constructor.
2	**public Throwable getCause()** Returns the cause of the exception as represented by a Throwable object.
3	**public String toString()** Returns the name of the class concatenated with the result of getMessage()
4	**public void printStackTrace()** Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	**public StackTraceElement [] getStackTrace()** Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	**public Throwable fillInStackTrace()** Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Catching Exceptions:

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
   //Protected code
}catch(ExceptionName e1)
{
   //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example:

The following is an array is declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{

   public static void main(String args[]){
      try{
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

This would produce following result:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Multiple catch Blocks:

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Example:

Here is code segment showing how to use multiple try/catch statements.

```
try
{
   file = new FileInputStream(fileName);
   x = (byte) file.read();
}catch(IOException i)
{
   i.printStackTrace();
   return -1;
}catch(FileNotFoundException f) //Not valid!
{
   f.printStackTrace();
   return -1;
}
```

The throws/throw Keywords:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword. Try to understand the different in throws and throw keywords.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

Amethod can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException:

```
import java.io.*;
public class className
{
   public void withdraw(double amount) throws RemoteException,
                              InsufficientFundsException
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

The finally Keyword

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
```

```
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}finally
{
   //The finally block always executes.
}
```

Example:

```
public class ExcepTest{

   public static void main(String args[]){
      int a[] = new int[2];
      try{
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      finally{
         a[0] = 6;
         System.out.println("First element value: " +a[0]);
         System.out.println("The finally statement is executed");
      }
   }
}
```

This would produce following result:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

Note the followings:

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses when ever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

Declaring you own Exception:

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- All exceptions must be a child of Throwable.

- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below:

```
class MyException extends Exception{
}
```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example:

```
// File Name InsufficientFundsException.java
import java.io.*;

public class InsufficientFundsException extends Exception
{
   private double amount;
   public InsufficientFundsException(double amount)
   {
      this.amount = amount;
   }
   public double getAmount()
   {
      return amount;
   }
}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java
import java.io.*;

public class CheckingAccount
{
   private double balance;
   private int number;
   public CheckingAccount(int number)
   {
      this.number = number;
   }
   public void deposit(double amount)
   {
      balance += amount;
   }
   public void withdraw(double amount) throws
                           InsufficientFundsException
   {
      if(amount <= balance)
      {
         balance -= amount;
      }
      else
      {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }
   public double getBalance()
   {
      return balance;
   }
   public int getNumber()
   {
      return number;
   }
}
```

The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

```
// File Name BankDemo.java
public class BankDemo
{
   public static void main(String [] args)
   {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);
      try
      {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      }catch(InsufficientFundsException e)
      {
         System.out.println("Sorry, but you are short $"
                                  + e.getAmount());
         e.printStackTrace();
      }
    }
}
```

Compile all the above three files and run BankDemo, this would produce following result:

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:25)
        at BankDemo.main(BankDemo.java:13)
```

Common Exceptions:

In java it is possible to define two catergories of Exceptions and Errors.

- **JVM Exceptions:** - These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples : NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException,
- **Programmatic exceptions** . These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException.

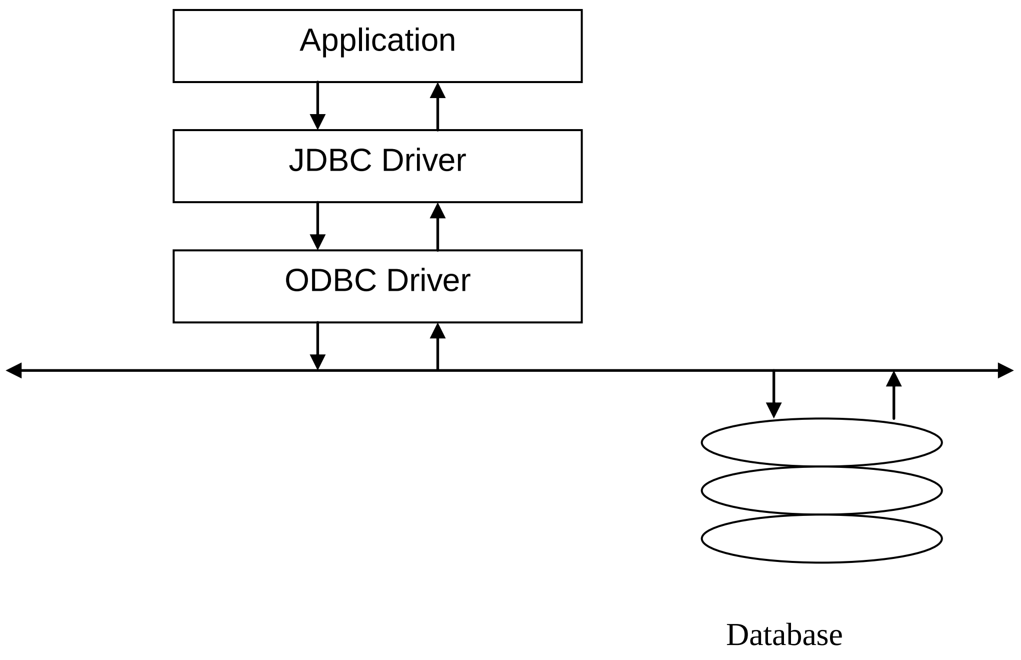
5. Java Database Connectivity

5.1 JDBC Basics

Java Database Connectivity is used to connect the database applications with Java programs. JDBC is designed to be platform-independent. JDBC Application Programming Interface defines how a program written in Java can communicate and interact with the database.

JDBC-ODBC Bridge

The JDBC-ODBC bridge driver is the only driver supplied by JavaSoft. JDBC driver only can talk with ODBC. Once JDBC passes the request off to the ODBC driver, it the responsibility of the ODBC driver to communicate with the database.



Driver Manager

JDBC provides a driver manager to load the driver using the Class.forName() method. The driver manager for Java JDBC-ODBC is **sun.jdbc.odbc.JdbcOdbcDriver.**

Connection

A connection object represents a connection with the database.

Statement

A statement object is used to send SQL statements to a database.

ResultSet

A ResultSet is a object that contains the result of execution of a SQL query.

5.2 JDBC Programming

Steps in JDBC Programming

1. **Find the JDBC Driver**
   Class.forName("sun.jdbc.odbc.JdbcOdbcDr
   ----------+…055
   iver");

2. **Configure the database**
   Create a ODBC for your database application by providing a DataSourceName.

3. **Test the configuration by establishing the connection**
   Connection c=DriverManager.
   getCOnnection(jdbc:odbc:DSN,user,pws);

4. **Generate SQL query and ResultSet.**

5. **Close Connection and close statement.**

Sample program which connects Microsoft Access Database

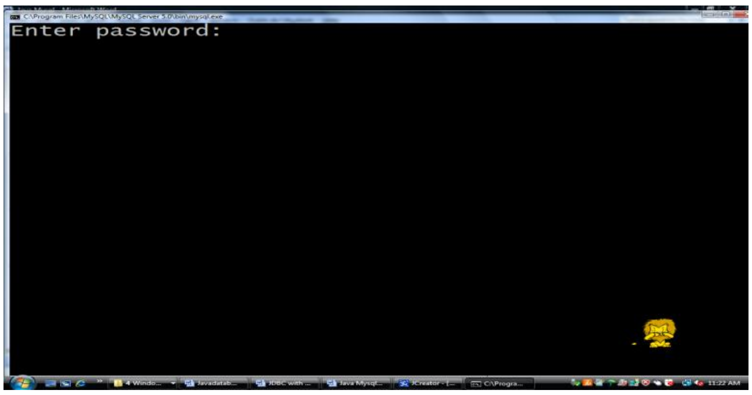
```
import java.sql.*;
class JavaDBExample
{
        public static void main(String args[])

{
        try
        {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

                String dsn1="DSN1";
        String DbUrl="jdbc:odbc:"+dsn1;

Connection  c=DriverManager.getConnection(DbUrl,"","");

Statement st=c.createStatement();

st.execute("create table StudentCS2 (id int,fname text,lname
text,home text,tel int)");
st.execute("insert          into          StudentCS2
values(1,'Paul','Habimana','Tumba',0785567890)");
st.execute("select * from StudentCS2");

ResultSet rs=st.getResultSet();

while(rs.next())
{

     System.out.println(rs.getString("fname"));

     System.out.println(rs.getInt("tel"));

     }
     st.close();

     c.close();
     }

     catch(Exception e)

          {
          System.out.println("error"+e.getMessage());
          }
          }
}
```

Java Database Application with MYSQL Server

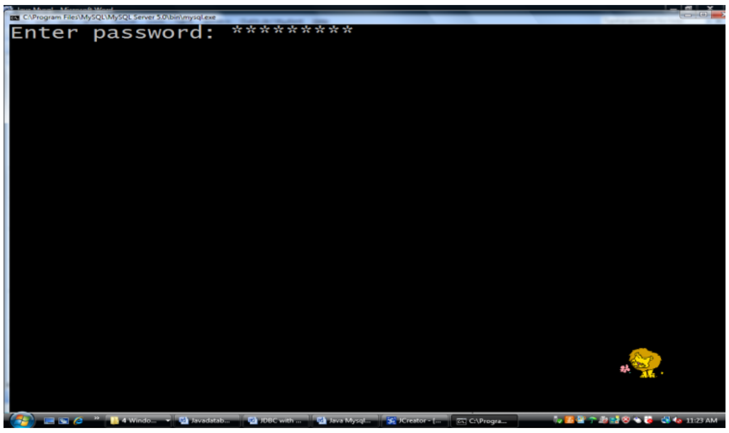
Installing MYQL Server

MYSQL is a popular Open Source used to manage data stored in database,
You can download and install the appropriate release of the MySQL database server software at http://dev.mysql.com/downloads/ and install it as a server machine. There are different version Msql Server, for this course we use
mysql-5.0.45-win32

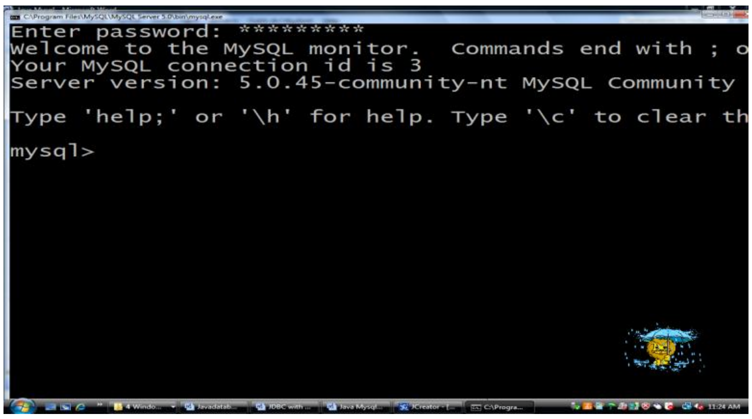
Creating Database in MYSQL Server
First you log on the server using the password you have created during the installation process. The default user name for MYQSL Server is called root.



```
Enter password:
```



```
Enter password: *********
```



```
Enter password: *********
Welcome to the MySQL monitor.  Commands end with ; o
Your MySQL connection id is 3
Server version: 5.0.45-community-nt MySQL Community

Type 'help;' or '\h' for help. Type '\c' to clear th

mysql>
```

In this **msql>** command prompt, type the following **SQL** queries:

Query 1: **msql> create database CSJava2;** to create a database called CSJava2.

Query 2: **msql>use CSJava2;** to select a database to use.

Query 3: **msql>create table student(id int not null primary key auto_increment, Fname text, Lname text, Home text, Tel int(12));** to create a table that will contain the student data.

Query 4: **msql>insert into student values(1,'Peter','Kagabo','NPC 64',0788689876);** to insert the 1st record in student table.

Installing Mysql Connector

In order to connect Java to Mysql you must download the file named **mysql-connector-java-5.1.6** and then do the following to install this connector:

Extract the jar file named **mysql-connector-java-5.1.6-bin.jar** from the zip file and copy it into the folder named **C:\ProgramFiles\Java\jdk1.6.0_06\jre\lib\ext**, which is the installation directory tree for the currently installed version of Java on my computer.

Critical steps in using JDBC

There are five critical steps in using JDBC to manipulate a database:

1. Load and register the JDBC driver classes *(programming interface)* for the database server that you intend to use.
2. Get a **Connection** object that represents a connection to the database server *(analogous to logging onto the server).*
3. Get one or more **Statement** objects for use in manipulating the database.
4. Use the **Statement** objects to manipulate the database.
5. Close the connection to the database.

The following java program example uses these steps to interact with MSQL database:

```
import  java.sql.*;//package containing all sql queries

class mysql1 {
          public static void main(String a[])
         {
             Connection con = null;
     String url = "jdbc:mysql://localhost:3306/";// mysql server runs on the port 3036
             String dbName = "test";// choosing a database to use
             String driver = "com.mysql.jdbc.Driver";
             String userName = "root";
             String password = "assistant";
             try {
                  Class.forName(driver).newInstance();//loading jdbc driver for mysql
//creating a connection to the database.

                 con =DriverManager.getConnection(url+dbName,userName,password);
                  Statement  st  =  con.createStatement();//creating  statement  on  a
connection established

                  ResultSet rs = st.executeQuery("SELECT * from records");//executing
the query

              System.out.println("Results");

//displaying data form database

              while( rs.next() ) {
                            String data = rs.getString("first");
                            String data1 = rs.getString("last");
                            System.out.println( data+" "+data1 );

                            }
```

```
                st.close();
               con.close();
                }
                catch( Exception e ) {
                System.out.println(e.getMessage());
                e.printStackTrace();
                }
        }
}
```

Once you become familiar with jdbc mysql you go further and design a java application that use a graphical user interface using Java Swing to manipulate data stored in mysql database.

6. Remote Method Invocation

Remote Method Invocation (RMI) allows a java object executes on one machine to invoke a method of a java object that executes on another machine.

A Simple Client/Sever Application using RMI

Step 1: Enter and compile source code

a) Second.java

```
import java.rmi.*;
public interface Second extends Remote {
long getMilliSeconds() throws RemoteException;
}
```

b) SecondImpl.java

```
import java.rmi.*;
import java.util.Date;
import java.rmi.server.UnicastRemoteObject;
public class SecondImpl extends UnicastRemoteObject
implements Second {
public SecondImpl() throws RemoteException { };
public long getMilliSeconds() throws RemoteException {
return(new Date().getTime());
//The method getTime returns the time in msecs
};
}
```

c) Server.java

```
import java.rmi.*;

import java.rmi.Naming;
public class Server {
public static void main(String[] args) {
RMISecurityManager sManager = new RMISecurityManager();
//System.setSecurityManager(sManager);
try {
SecondImpl remote = new SecondImpl();
Naming.rebind ("Dater", remote);
System.out.println("Server Started");
}
catch(Exception e) {
System.out.println("error occurred at server"+e.getMessage());
}
}
}
```

d) TimeClient.java

```
import java.rmi.*;
public class TimeClient {
public static void main(String[] args) {
try {
Second sgen = (Second) Naming.lookup("rmi://localhost/Dater");
System.out.println("Milliseconds are"+sgen.getMilliSeconds());
}
catch(Exception e) {
System.out.println("Problem encountered accessing remoteobject "+e);
}
}
}
```

Step 2: Generate Stubs and Skeletons

Stub

A stub is a java object that resides on the client machine. Its function is to present the same interfaces as the remote server.

Skeleton

A skeleton is a java object that resides on a server machine. It works with the other parts of RMI system to receive the requests and invoke appropriate code on the server.

To generate Stubs and Skeletons

To generate stubs and skeletons use the tool RMI compiler, this is invoked from the command line as shown as shown here:

rmic AddServerImpl

This command generates two new files: **AddServerImpl_Skel.class**
AddServerImpl_Stub.class

Step 3: Install files on the client and server machines

Under Client Machine copy the following files:

AddClient.class
AddServerImpl_Stub.class
AddServerIntf.class

Under Server Machine copy the following files:

```
AddServerIntf.class
AddServerImpl.class
AddServerImpl_Skel.class
AddServerImpl_Stub.class
AddServer.class
```

Step 4: Start the RMI Registry on the Server Machine

On the server machine, type the following in the command line

```
start rmiregistry
```

Step 5: Start the Server

The server code is started from the command line, as shown as here:

```
java AddServer
```

Step 6: Start the Client

The AddClient software requires three argunments: the name or IP address of the server and the two numbers that are to be summed together. From the command line use any one of the following command,

```
java AddClient server1 8 9
java AddClient 192.168.0.1 8 9
```

Second RMI Example

a) AddServerIntf.java

```
import java.rmi.*;
public interface AddServerIntf extends Remote
{
  double add(double d1, double d2) throws RemoteException;
}
```

b) AddServerImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class AddServerImpl extends UnicastRemoteObject implements AddServerIntf
{
 public AddServerImpl() throws RemoteException {  }
 public double add(double d1, double d2) throws RemoteException
 {
   return d1 + d2;
```

```
  }
}
```

c) AddServer.java

```
import java.net.*;
import java.rmi.*;
public class AddServer
{
 public static void main(String args[])
 {
   try
   {
     AddServerImpl addServerImpl = new AddServerImpl();
     Naming.rebind("AddServer", addServerImpl);
   }
   catch(Exception e)
   {
     System.out.println("Exception: " + e);
   }
  } }
```

d) AddClient.java

```
import java.rmi.*;
public class AddClient
{
  public static void main(String args[])
 {
   try
   {
     String addServerURL = "rmi://" + args[0] + "/AddServer";
     AddServerIntf addServerIntf =(AddServerIntf)Naming.lookup(addServerURL);
     System.out.println("The first number is: " + args[1]);
     double d1 = Double.valueOf(args[1]).doubleValue();
     System.out.println("The second number is: " + args[2]);

     double d2 = Double.valueOf(args[2]).doubleValue();
     System.out.println("The sum is: " + addServerIntf.add(d1, d2));
   }
   catch(Exception e)
   {
     System.out.println("Exception: " + e);
   }
  }
}
```

8. Servlet

8.1 Servlet Basics

Servlets are modules that extend request/response oriented services. Applets are to browsers and Servlets are to servers. Servlets have no GUI. Servlet supports multiple requests. Servlets can run with web servers.

A Simple Servlet

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

public class HelloServlet extends GenericServlet
{
  public void service(ServletRequest request, ServletResponse response)  throws
                                                                  ServletException, IOException
{
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>Hello!");
    pw.close();
  }
}
```

How to Run Servlets

1. Create and Compile the Servlet source code
2. Start Tomcat (or any web servers that supports Servlets)
3. Start a web browser and request the Servlet.

To start Tomcat (Use Tomcat 4.0 for J2sdk1.4.0.)

Go to C:\Program Files\Apache\Tomcat\bin\

- use startup.bat to start the tomcat
- use shutdown.bat to stop the tomcat

Copy .class file into Tomcat

The Servlet .class file should be copied into tomcat's directory

..Tomcat4.0\webapps\examples\WEB-INF\classes

To start Web Browser

Type the following URL in the browser:

http://localhost:8080/examples/servlet/HelloServlet

8.2 Handling HTTPServlet Requests

The HTTPServlet class provides specializes methods that handle the various types of HTTP requests.

Servlet can handle both HTTP GET and POST requests. POST method is identical to GET method except that the method parameter for the dorm tag explicitly specifies that the POST method should be used.

Using doGet() method

//ColorGetServlet.java using DoGet method

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

public class ColorGetServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    String color = request.getParameter("color");
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>The selected color is:  ");
    pw.println(color);
    pw.close();
  }
}

//HTML file to call the ColorGetServlet under DoGet method
<html>
<body>
<center>
<form name="Form1"
  action="http://localhost:8080/examples/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

Using doPost() method

```
//ColorPostServlet.java using DoPost method
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ColorPostServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    String color = request.getParameter("color");
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>The selected color is:  ");
    pw.println(color);
    pw.close();
  }
}

//HTML file to call the ColorPostServlet under DoPost method
<html>
<body>
<center>
<form name="Form1"
  method="post"
  action="http://localhost:8080/examples/servlet/ColorPostServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

8.3 Sessions & Cookies

Sessions

Session tracking is a mechanism that Servlets use to maintain state about a series of requests form the same user across same period of time.

Cookies

Cookies are the mechanism that a Servlet uses to have clients hold a small amount of state information associated with the user.

Using Sessions

A session can be created via the getSession() method of HTTPServletRequest.

The following Servelt illustrates how to use session state.

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

public class DateServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    // Get the HttpSession object.
    HttpSession hs = request.getSession(true);

    // Get writer.
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.print("<B>");

    // Display date/time of last access.
    Date date = (Date)hs.getAttribute("date");
    if(date != null) {
      pw.print("Last access: " + date + "<br>");
    }

    // Display current date/time.
    date = new Date();
    hs.setAttribute("date", date);
    pw.println("Current date: " + date);
  }
}
```

When you first request this Servlet, the browser displays one line with the current date and time information. On subsequent invocations, two lines are displayed. The first line shows the data and time when the Servlet was last accessed. The second line shows the current data and time.

Using Cookies

There are three files used to illustrate how to use cookies.

AddCookie.htm	Allows a user to specify value for a cookie **MyCookie**
AddCookieServlet.java	Processes the submission of AddCookie.htm
GetCookiesServlet.java	Displays Cookie values

// AddCookie.htm

```
<html>
<body>
<center>
<form name="Form1"
  method="post"
  action="http://localhost:8080/examples/servlet/AddCookieServlet">
<B>Enter a value for MyCookie:</B>
<input type=textbox name="data" size=25 value="">
<input type=submit value="Submit">
</form>
</body>
</html>
```

// AddCookieServlet.java

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

```
public class AddCookieServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    // Get parameter from HTTP request.
    String data = request.getParameter("data");

    // Create cookie.
    Cookie cookie = new Cookie("MyCookie", data);

    // Add cookie to HTTP response.
    response.addCookie(cookie);

    // Write output to browser.
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>MyCookie has been set to");
    pw.println(data);
    pw.close();
  }
}
```

// GetCookiesServlet.java

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

public class GetCookiesServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    // Get cookies from header of HTTP request.
    Cookie[] cookies = request.getCookies();

    // Display these cookies.
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>");
    for(int i = 0; i < cookies.length; i++) {
      String name = cookies[i].getName();
      String value = cookies[i].getValue();
      pw.println("name = " + name +
        "; value = " + value);
    }
    pw.close();
  }
}
```

How to Run

1. Compile Servlet files
2. Start Tomcat
3. Display AddCookie.htm in a browser
4. Enter a value for MyCookie
5. Submit the Web page
6. Request the following URL in the browser
   http:..localhost:8080/examples/Servlet/GetCookiesServlet

Links:

1. **http://java.sun.com/developer/onlineTraining/corba/corba.html**
2. **http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html**