JV – Object-Oriented Programming in Java

Seminar 8 – Advanced GUI

1. Introduction

We have discussed GUI application for a few weeks by now, this week, I want to bring you to the world of Swing application.

Swing is a collection of GUI components that you can use to design a GUI application. It was developed to provide a more sophisticated set of GUI components than the AWT. Swing provides a more platform-specific native look and feel.

There are many Swing components you can use. We will mainly address two issues. One is to choose appropriate Swing components and arrange them properly. Two is to add the logic to the application and make sure that GUI interface interacts with the logic.

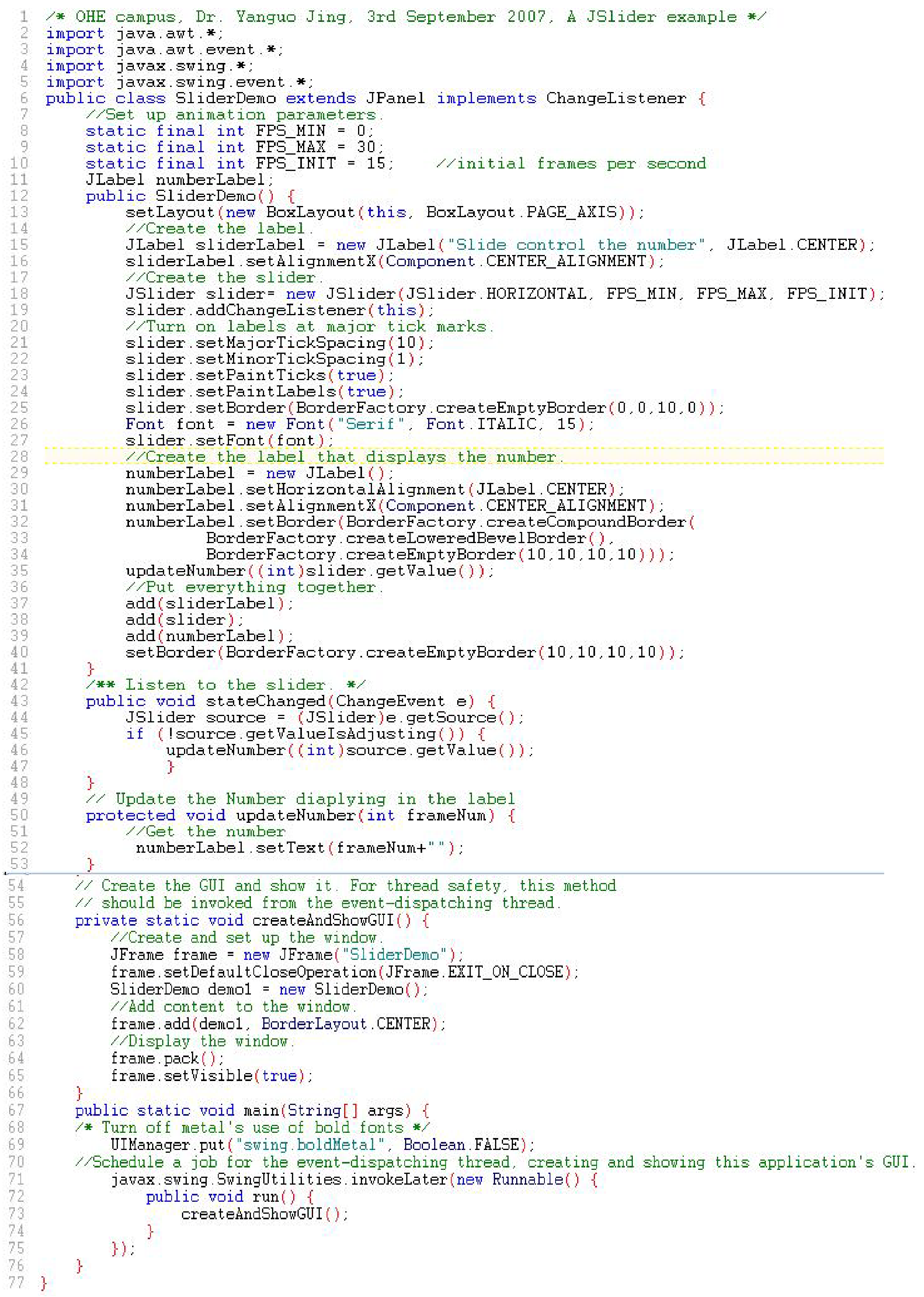
Apart from the examples give in the text there are many good examples you can find in the Java.sun.com official tutorial (http://java.sun.com/docs/books/tutorial/ui/features/components.html accessed on 3 May 2008). There are many Swing components you can choose from, they are JTextField, JPanel, JScrollPane, JFrame, JLabel , JButton, JCheckBox , JComboBox, JList , JMenu, JRadioButton, JSlider, JSpinner, JPasswordField, JColorChooser , JEditorPane, JTextPane, JFileChooser, JTable , JTextArea, JTree , JProgressBar, JSeparator , JToolTip, JApplet, JDialog, JSplitPane, JTabbedPane, JToolBar, JInternalFrame, JLayeredPane, etc.

You should also find it useful to have the official API library at hand, this is the link to it http://java.sun.com/javase/6/docs/api/ (access on 3 May 2008). The address may change for the future JDK release, but you can always find it from http://java.sun.com/

It is difficult to list all of these Swing components, the following five components are chosen to give you an initial idea of how they are used.

2. Swing GUI components:

(2.1) A **JSlider** component is used to let the user easily enter a numeric value bounded by a minimum and maximum value. The following picture shows an application that uses a slider to control the number from 0-30:



```java
/* OHE campus, Dr. Yanguo Jing, 3rd September 2007, A JSlider example */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class SliderDemo extends JPanel implements ChangeListener {
    //Set up animation parameters.
    static final int FPS_MIN = 0;
    static final int FPS_MAX = 30;
    static final int FPS_INIT = 15;    //initial frames per second
    JLabel numberLabel;
    public SliderDemo() {
        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        //Create the label.
        JLabel sliderLabel = new JLabel("Slide control the number", JLabel.CENTER);
        sliderLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        //Create the slider.
        JSlider slider= new JSlider(JSlider.HORIZONTAL, FPS_MIN, FPS_MAX, FPS_INIT);
        slider.addChangeListener(this);
        //Turn on labels at major tick marks.
        slider.setMajorTickSpacing(10);
        slider.setMinorTickSpacing(1);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);
        slider.setBorder(BorderFactory.createEmptyBorder(0,0,10,0));
        Font font = new Font("Serif", Font.ITALIC, 15);
        slider.setFont(font);
        //Create the label that displays the number.
        numberLabel = new JLabel();
        numberLabel.setHorizontalAlignment(JLabel.CENTER);
        numberLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        numberLabel.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLoweredBevelBorder(),
                BorderFactory.createEmptyBorder(10,10,10,10)));
        updateNumber((int)slider.getValue());
        //Put everything together.
        add(sliderLabel);
        add(slider);
        add(numberLabel);
        setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
    }
    /** Listen to the slider. */
    public void stateChanged(ChangeEvent e) {
        JSlider source = (JSlider)e.getSource();
        if (!source.getValueIsAdjusting()) {
            updateNumber((int)source.getValue());
            }
    }
    // Update the Number diaplying in the label
    protected void updateNumber(int frameNum) {
        //Get the number
         numberLabel.setText(frameNum+"");
    }
    // Create the GUI and show it. For thread safety, this method
    // should be invoked from the event-dispatching thread.
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("SliderDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        SliderDemo demo1 = new SliderDemo();
        //Add content to the window.
        frame.add(demo1, BorderLayout.CENTER);
        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
    /* Turn off metal's use of bold fonts */
        UIManager.put("swing.boldMetal", Boolean.FALSE);
    //Schedule a job for the event-dispatching thread, creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Figure 8-1 part 1, a JSlider example source code

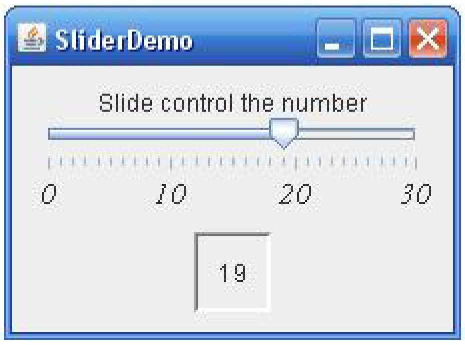


Figure 8-1 part 2, the output of the A JSlider exampleapplication

Line 18: Define a JSlider object – slider.
Line 19: add an action listener to the slider.
Line 21-22: set the max and min slide range by one click.
Line 23: set tick marks to be painted on the slider
Line 24: set labels to be painted on the slider.
Line 27: set the font used for the text of slider.
Line 38: add the slider object in the JPanel object.
Line 43-48: implement the action listener method of JSlider– stateChanged method. To get the current value on the slider and update the number on the numberLabel.

The following methods and attributes of JSlider may be useful. You should try to build up a habit of having the Java API library in hand when Java programming.

Creating the Slider

Constructor	Purpose
JSlider()	Creates a horizontal slider with the range 0 to 100 and an initial value of 50.
JSlider(int min, int max) JSlider(int min, int max, int value)	Creates a horizontal slider with the specified minimum and maximum values. The third int argument, when present, specifies the slider's initial value.
JSlider(int orientation) JSlider(int orientation, int min, int max, int value)	Creates a slider with the specified orientation, which must be either JSlider.HORIZONTAL or JSlider.VERTICAL. The last three int arguments, when present, specify the slider's minimum, maximum, and initial values, respectively.

JSlider(BoundedRangeModel)	Creates a horizontal slider with the specified model, which manages the slider's minimum, maximum, and current values and their relationships.

Fine Tuning the Slider's Appearance

Method	Purpose
void setValue(int) int getValue()	Sets or gets the slider's current value. The set method also positions the slider's knob.
void setOrientation(int) int getOrientation()	Sets or gets the orientation of the slider. Possible values are JSlider.HORIZONTAL or JSlider.VERTICAL.
void setInverted(boolean) boolean getInverted()	Sets or gets whether the maximum is shown at the left of a horizontal slider or at the bottom of a vertical one, thereby inverting the slider's range.
void setMinimum(int) int getMinimum() void setMaximum(int) int getMaximum()	Sets or gets the minimum or maximum values of the slider. Together, these methods set or get the slider's range.
void setMajorTickSpacing(int) int getMajorTickSpacing() void setMinorTickSpacing(int) int getMinorTickSpacing()	Sets or gets the range between major and minor ticks. You must call setPaintTicks(true) for the tick marks to appear.
void setPaintTicks(boolean) boolean getPaintTicks()	Sets or gets whether tick marks are painted on the slider.
void setPaintLabels(boolean) boolean getPaintLabels()	Sets or gets whether labels are painted on the slider. You can provide custom labels with setLabelTable or get automatic labels by setting the major tick spacing to a non-zero value.
void setLabelTable(Dictionary) Dictionary getLabelTable()	Sets or gets the labels for the slider. You must call setPaintLabels(true) for the labels to appear.
Hashtable createStandardLabels(int) Hashtable createStandardLabels(int, int)	Creates a standard set of numeric labels. The first int argument specifies the increment, the second int argument specifies the starting point. When left unspecified, the starting point is set to the slider's minimum number.
setFont(java.awt.Font)	Sets the font for slider labels .

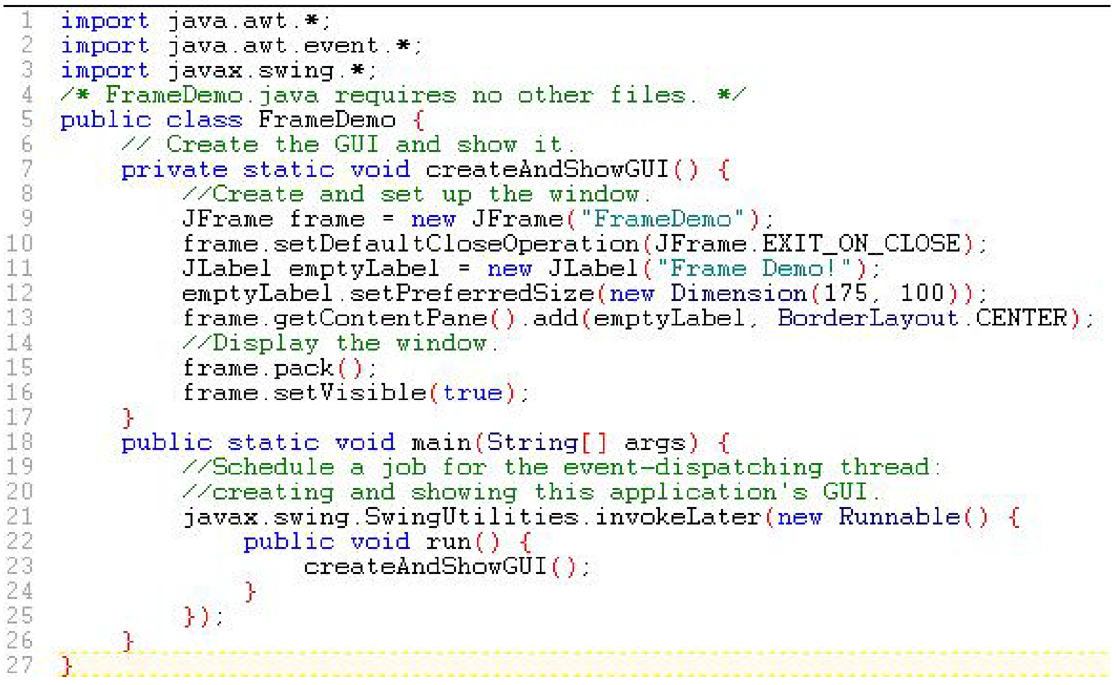
Watching the Slider Operate

Method	Purpose
void addChangeListener(ChangeListener)	Registers a change listener with the slider.
boolean getValueIsAdjusting()	Determines whether the user gesture to

	move the slider's knob is complete.

Figure 8-2 the Slider API

(2.2) If you want to create a window for your application, a **JFrame** component is the one you will be looking for. A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. The getInsets method is used to get the dimensions of the border area. A frame implemented as an instance of the JFrame class. Applications with a GUI usually include at least one frame. Class JFrame supports three operations when the user closes the window. It can be controlled by the method **setDefaultCloseOperation**. See the simple example in Figure 8-2, which demonstrates the basic use of JFrame.



```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/* FrameDemo.java requires no other files. */
public class FrameDemo {
    // Create the GUI and show it.
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("FrameDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel emptyLabel = new JLabel("Frame Demo!");
        emptyLabel.setPreferredSize(new Dimension(175, 100));
        frame.getContentPane().add(emptyLabel, BorderLayout.CENTER);
        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

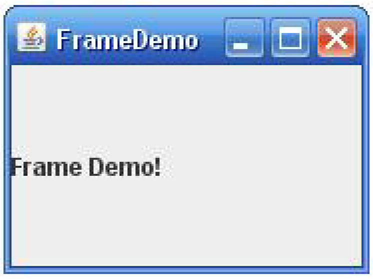
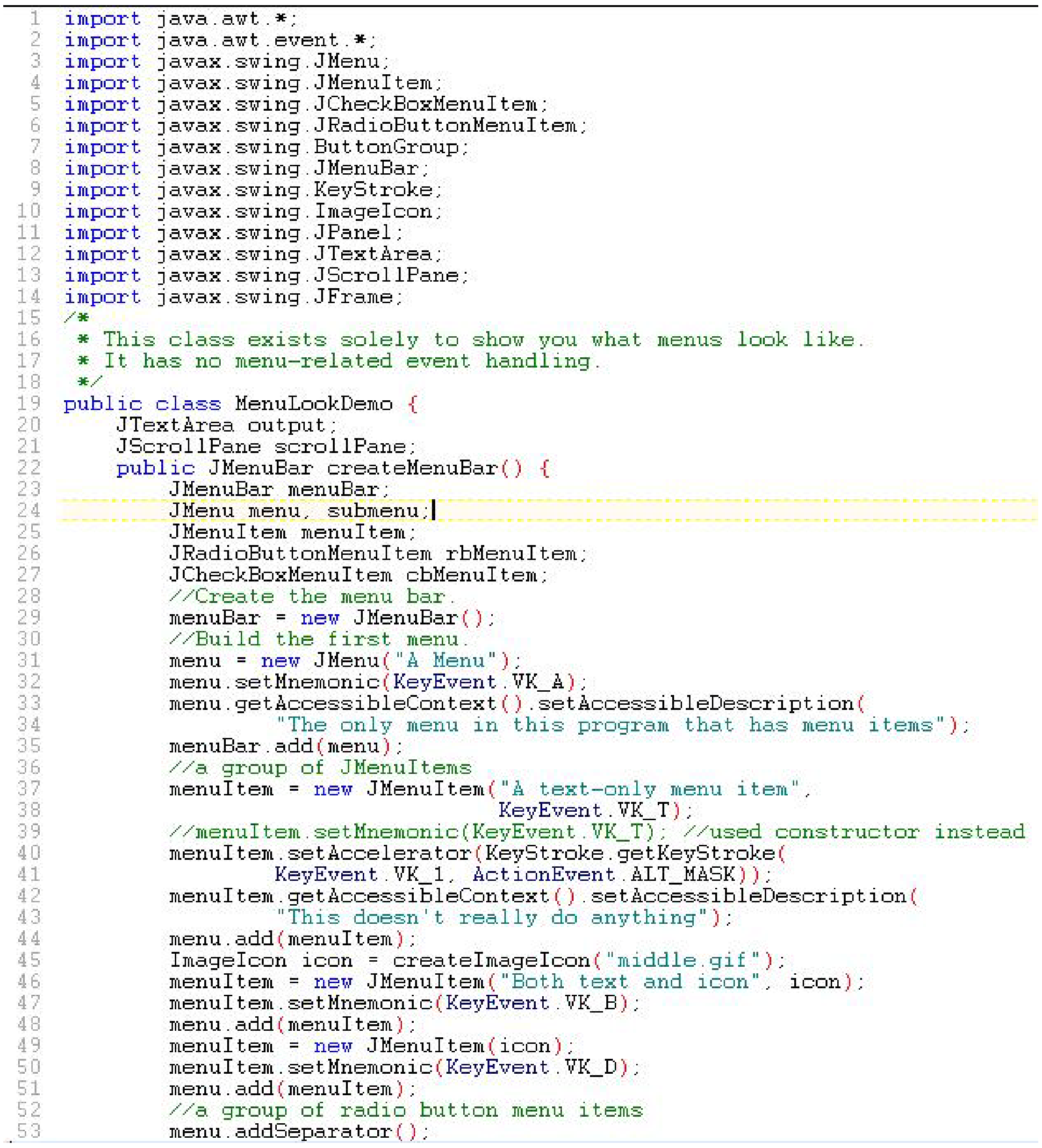


Figure 8-2 JFrame example

(2.3) You may want to let the user choose one of several operation options on Frame, you need **Menus with JFrame**. Menus are unique in space saving, because they appear either in a menu bar or as a popup menu. A menu bar contains one or more menus usually along the top of a window. A popup menu is a menu that is invisible until the user makes a specified mouse action (e. g. pressing the right mouse button, etc) over a popup-enabled component. The popup menu then appears under the cursor. There are many menu-related components including a menu bar, menus, menu items, radio button menu items, check box menu items, and separators. As you can see, a menu item can have either an image or text, or both. You can also specify other properties, such as font and color. The example in Fig 8-3 is an example to demonstrate the use of JMenu components.



```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.ButtonGroup;
import javax.swing.JMenuBar;
import javax.swing.KeyStroke;
import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
/*
 * This class exists solely to show you what menus look like.
 * It has no menu-related event handling.
 */
public class MenuLookDemo {
    JTextArea output;
    JScrollPane scrollPane;
    public JMenuBar createMenuBar() {
        JMenuBar menuBar;
        JMenu menu, submenu;
        JMenuItem menuItem;
        JRadioButtonMenuItem rbMenuItem;
        JCheckBoxMenuItem cbMenuItem;
        //Create the menu bar.
        menuBar = new JMenuBar();
        //Build the first menu.
        menu = new JMenu("A Menu");
        menu.setMnemonic(KeyEvent.VK_A);
        menu.getAccessibleContext().setAccessibleDescription(
                "The only menu in this program that has menu items");
        menuBar.add(menu);
        //a group of JMenuItems
        menuItem = new JMenuItem("A text-only menu item",
                                 KeyEvent.VK_T);
        //menuItem.setMnemonic(KeyEvent.VK_T); //used constructor instead
        menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_1, ActionEvent.ALT_MASK));
        menuItem.getAccessibleContext().setAccessibleDescription(
                "This doesn't really do anything");
        menu.add(menuItem);
        ImageIcon icon = createImageIcon("middle.gif");
        menuItem = new JMenuItem("Both text and icon", icon);
        menuItem.setMnemonic(KeyEvent.VK_B);
        menu.add(menuItem);
        menuItem = new JMenuItem(icon);
        menuItem.setMnemonic(KeyEvent.VK_D);
        menu.add(menuItem);
        //a group of radio button menu items
        menu.addSeparator();
        ButtonGroup group = new ButtonGroup();
        rbMenuItem = new JRadioButtonMenuItem("A radio button menu item");
        rbMenuItem.setSelected(true);
        rbMenuItem.setMnemonic(KeyEvent.VK_R);
        group.add(rbMenuItem);
        menu.add(rbMenuItem);
        rbMenuItem = new JRadioButtonMenuItem("Another one");
        rbMenuItem.setMnemonic(KeyEvent.VK_O);
        group.add(rbMenuItem);
        menu.add(rbMenuItem);
        //a group of check box menu items
        menu.addSeparator();
        cbMenuItem = new JCheckBoxMenuItem("A check box menu item");
        cbMenuItem.setMnemonic(KeyEvent.VK_C);
        menu.add(cbMenuItem);
        cbMenuItem = new JCheckBoxMenuItem("Another one");
        cbMenuItem.setMnemonic(KeyEvent.VK_H);
        menu.add(cbMenuItem);
        //a submenu
        menu.addSeparator();
        submenu = new JMenu("A submenu");
        submenu.setMnemonic(KeyEvent.VK_S);
        menuItem = new JMenuItem("An item in the submenu");
        menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_2, ActionEvent.ALT_MASK));
        submenu.add(menuItem);
        menuItem = new JMenuItem("Another item");
        submenu.add(menuItem);
        menu.add(submenu);
        //Build second menu in the menu bar.
        menu = new JMenu("Another Menu");
        menu.setMnemonic(KeyEvent.VK_N);
        menu.getAccessibleContext().setAccessibleDescription(
                "This menu does nothing");
        menuBar.add(menu);
        return menuBar;
    }
    public Container createContentPane() {
        //Create the content-pane-to-be.
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setOpaque(true);
        //Create a scrolled text area.
        output = new JTextArea(5, 30);
        output.setEditable(false);
        scrollPane = new JScrollPane(output);
        //Add the text area to the content pane.
        contentPane.add(scrollPane, BorderLayout.CENTER);
        return contentPane;
    }
    /** Returns an ImageIcon, or null if the path was invalid. */
    protected static ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = MenuLookDemo.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }
    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("MenuLookDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Create and set up the content pane.
        MenuLookDemo demo = new MenuLookDemo();
        frame.setJMenuBar(demo.createMenuBar());
        frame.setContentPane(demo.createContentPane());
        //Display the window.
        frame.setSize(450, 260);
        frame.setVisible(true);
    }
```

```
130        public static void main(String[] args) {
131            //Schedule a job for the event-dispatching thread:
132            //creating and showing this application's GUI.
133            javax.swing.SwingUtilities.invokeLater(new Runnable() {
134                public void run() {
135                    createAndShowGUI();
136                }
137            });
138        }
139    }
```

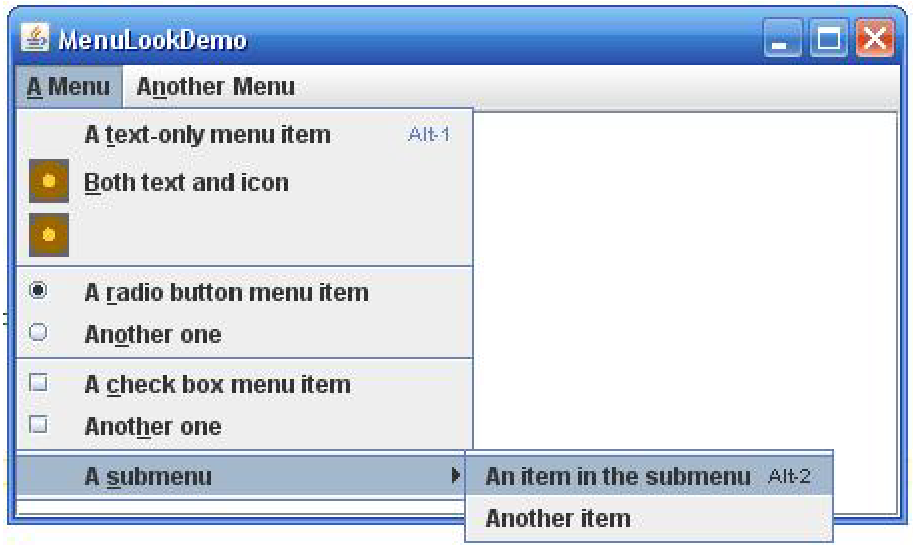


Figure 8-3 a Menu example

In the example in Figure 8-3:

Line 22-27: define all the menu objects that will be needed.
Line 29: initialize the menu bar.
Line 31-35: Define the first top-level Menu and add it to the menu bar.
Line 37-44: initialize a menu item and add it to the menu defined in line 31.

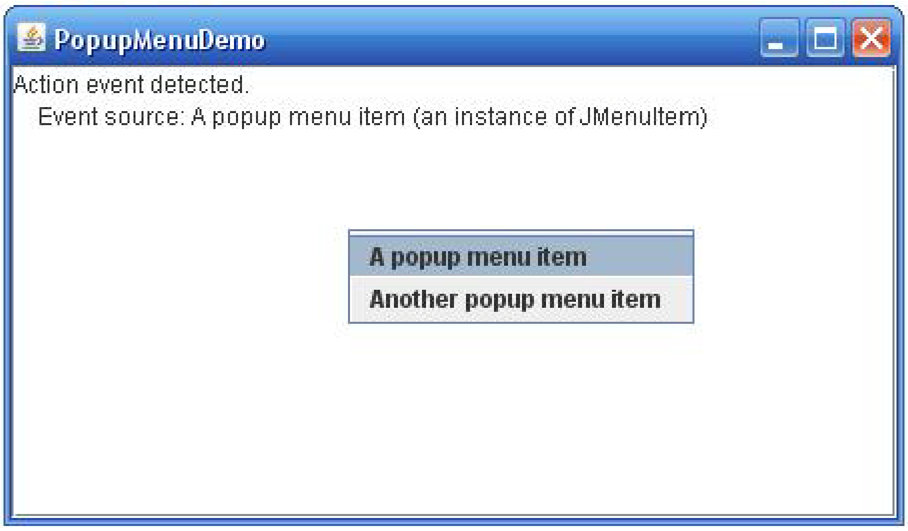
You can associate an action listener to menu item, implement the associated actions accordingly see Fig 22.5 and Fig 22.6 in the textbook.

Enabling Keyboard Operation, You may want to provide keyboard alternatives to use Menus. Mnemonics can be implemented to use the keyboard to navigate the menu hierarchy. Accelerators are shortcuts to bypass the menu hierarchy. You can define a mnemonic to make an already visible menu item to be chosen. For example, in the Fig 8-3 example, the first menu has the mnemonic A, and its second menu item has the mnemonic B.

By pressing the Alt and A keys, the first menu will appear. A menu item generally displays its mnemonic by underlining the first occurrence of the mnemonic character in the menu item's text. **An accelerator** is a key combination that causes a menu item without the need to make it visible. For example, pressing the Alt and 2 keys in the Fig 8-3 example makes the first item in the first menu's submenu chosen, without bringing up any menus at all. Only leaf menu items can have accelerators.

(2.4) We use Popup menu all the time, the most common practice in Microsoft Windows is to click the right button of a mouse to bring up a Popup menu. In Java, you can use JPopupMenu to implement a Popup menu. You need to add and register an mouse listener on each component that you want to bring up popup menus. Let’s have a look at the following example in Fig 8-4.

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.JPopupMenu;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.KeyStroke;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
/*
 * Like MenuDemo, but with popup menus added.
 */
public class PopupMenuDemo implements ActionListener {
    JTextArea output;
    JScrollPane scrollPane;
    String newline = "\n";
    public Container createContentPane() {
        //Create the content-pane-to-be.
        JPanel contentPane = new JPanel(new BorderLayout());
        contentPane.setOpaque(true);
        //Create a scrolled text area.
        output = new JTextArea(5, 30);
        output.setEditable(false);
        scrollPane = new JScrollPane(output);
        //Add the text area to the content pane.
        contentPane.add(scrollPane, BorderLayout.CENTER);
        return contentPane;
    }
    public void createPopupMenu() {
        JMenuItem menuItem;
        //Create the popup menu.
        JPopupMenu popup = new JPopupMenu();
        menuItem = new JMenuItem("A popup menu item");
        menuItem.addActionListener(this);
        popup.add(menuItem);
        menuItem = new JMenuItem("Another popup menu item");
        menuItem.addActionListener(this);
        popup.add(menuItem);
        //Add listener to the text area so the popup menu can come up.
        MouseListener popupListener = new PopupListener(popup);
        output.addMouseListener(popupListener);
    }
    public void actionPerformed(ActionEvent e) {
        JMenuItem source = (JMenuItem)(e.getSource());
        String s = "Action event detected."
                   + newline
                   + "    Event source: " + source.getText()
                   + " (an instance of " + getClassName(source) + ")";
        output.append(s + newline);
        output.setCaretPosition(output.getDocument().getLength());
    }
    // Returns just the class name -- no package info.
    protected String getClassName(Object o) {
        String classString = o.getClass().getName();
        int dotIndex = classString.lastIndexOf(".");
        return classString.substring(dotIndex+1);
    }
    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("PopupMenuDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Create/set menu bar and content pane.
        PopupMenuDemo demo = new PopupMenuDemo();
 71          frame.setContentPane(demo.createContentPane());
 72          //Create and set up the popup menu.
 73          demo.createPopupMenu();
 74          //Display the window.
 75          frame.setSize(450, 260);
 76          frame.setVisible(true);
 77      }
 78      public static void main(String[] args) {
 79          //Schedule a job for the event-dispatching thread:
 80          //creating and showing this application's GUI.
 81          javax.swing.SwingUtilities.invokeLater(new Runnable() {
 82              public void run() {
 83                  createAndShowGUI();
 84              }
 85          });
 86      }
 87      class PopupListener extends MouseAdapter {
 88          JPopupMenu popup;
 89          PopupListener(JPopupMenu popupMenu) {
 90              popup = popupMenu;
 91          }
 92          public void mousePressed(MouseEvent e) {
 93              maybeShowPopup(e);
 94          }
 95          public void mouseReleased(MouseEvent e) {
 96              maybeShowPopup(e);
 97          }
 98          private void maybeShowPopup(MouseEvent e) {
 99              if (e.isPopupTrigger()) {
100                  popup.show(e.getComponent(),
101                             e.getX(), e.getY());
102              }
103          }
104      }
105  }
```



```
Action event detected.
   Event source: A popup menu item (an instance of JMenuItem)
```

Figure: 8-4 A Popup Menu example

Line 19-30: To define a content panel, which will be put inside a Frame (see line71)

Line 42: define a PopupListener object (it is a child class of MouseListener) and attach it to the background JTextArea object – output. This is to listen the mouse actions occur on output.

Line 87-104: Define a PopupListener class extended from MouseListener. This class will bring up the popup menu when mouse actions are observed.

(2.5) A **JTabbedPane** arranges GUI components into layers, of which, only one layer is visible at a time. Users can click on the tab label to visualize the required layer. Have a look at the example in Fig 22.13 and 22.14 in the textbook.

Required Reading

In **Chapter 22** of the text, read the following sections:

> **22.1 - 22.5, 22.8**
> **Skim 22.6-22.8**

Acknowlegement

The examples in Fig 8-1, 8-2, 8-3 and 8-4 for this lecture were drawn from the above web site and according to SUN's directive are copyright protected according to the following copyright message:

Appendix 1: Applets

Well done on making it through to the final seminar! This appendix is included for the benefit of those that take the IN specialization, as java Applets are an important aspect of WWW sites. It is on one of the most interesting features of Java.
An applet is a small Java program designed to be run by a Java-enabled web browser (i.e. it has a Java Virtual Machine embedded within it) or an applet viewer. To be run, the applet must be referenced within a HTML file (web page). When a web page that contains a reference to an applet is loaded, the browser downloads the applet from the web and executes the applet on the client machine. This process is potentially very dangerous to the client machine so there are some restrictions imposed on applets. We will discuss these in a later section of the lecture.

Before we start to write our own applets we must briefly consider how and applet is called from a HTML script file. Therefore we will now deal with the small amount of HTML knowledge that we require.

I. A Brief Note on HTML

Web pages are typically created using a *mark-up language* that is understood by WWW browsers. There are a number of these mark-up languages, but one of the most common at present is HTML (Hyper Text Mark-up Language). HTML, like all other mark-up languages, comprises a set of *tags* which are interpreted by a browser so as to produce a particular look. All HTML tags start with a < (less than) symbol and end with a > (greater than symbol), and are generally used in pairs. For example:

```
<CENTER>Hello World!</CENTER>
```

will cause the text “Hello World!” to be centred on a WWW page. Note that the closing tag has a / (forward slash) after the < symbol. Figure 1 gives the minimum amount of HTML required to embed an applet into a WWW page.

```
<HTML>
```

```
  <HEAD>
    <TITLE>Applet Example 1</TITLE>
  </HEAD>

  <BODY>
    <CENTER>
      <APPLET code=AppletEx1.class width=500
                height=150> </APPLET>
    </CENTER>
  </BODY>

</HTML>
```

Figure 1: *Minimum amount of HTML require to embed an applet into a WWW page*

Things to note about the HTML code presented in **Figure 1** are:

1. The code is written using an editor in the same way that we would create a Java program, except that in this case the file should have the extension .html or .htm.

2. An HTML WWW page always starts and ends with the tags <HTML> and </HTML>, this tells the browser that the page is written in HTML and not some other mark-up language.

3. The page itself is split into a head and body. The head contains some preliminary information used by the browser such as the name of the WWW page, the body contains the material that will actually be displayed on the page. The head is contained between the tags <HEAD> and </HEAD>, the body between the tags <BODY> and </BODY>.

4. We have only included one thing in the head, the title of the page which is contained between the tags <TITLE> and </TITLE>.

5. Some HTML tags can have arguments associated with them. In this case they are simply listed prior to the closing > of the opening tag. The applet tag in **Figure 1** has three arguments:

1. code: The name of the java byte code file in which the applet is contained, i.e. the name of the executable java .class file (AppletEx1.class in the example). ***Remember this – you're calling the .class file, **NOT** the .java file!!!*

2. width: The width of the applet display area in pixels (300 in the example).

3. height: The height of the applet display area in pixel (100 in the example).

6. Just to make the page look nice we have caused the applet to be centred using the <CENTER> and </CENTER> tags, although the applet would work equally well without this centring.

And that is all we need to know about HTML for the moment!

II. Overview of Applets and First Example

When working with applets we can make use of the applet package contained within the Java API. This package contains the Applet class which in turn contains the most commonly required methods to create and manipulate applets. A partial listing of the Applet class is as follows:

```
public class Applet extends Panel{
   //Constructors
   public Applet();

   //Methods
   public void destroy();
   public String getAppletInfo();
   public AudioClip getAudioClip(URL url);
   public AudioClip getAudioClip (URL url, String name);
   public Image getImage(URL url);
   public Image getImage(URL url, String name);
   public String getParameter(String name);
   public void init();
   public void play(URL url);
```

```
        public void resize(int width, int height);
        public void start();
        public void stop();
    }
```

A complete overview can be found within the Java API documentation. Your applet class will extend this Applet class and thus inherit the above methods.

Of the above methods, the four that give you the framework on which to build any serious applet are: init(), start(), stop() and destroy(). The init() method is described below. The start() method is called **automatically** after Java calls the init() method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages. This means that the start() method can be called repeatedly, unlike the init() method. For this reason, put the code that you want executed only once in the init() method, for example user interface components, rather than in the start() method. The start method is where you usually restart a thread for your applet, for example to resume an animation. If your applet does nothing that needs to be suspended when the user leaves the current web page, you do not need to implement this method.

Like the start() method, the stop() method is also called automatically. However, this method is called when the user leaves the page on which the applet sits. Thus it can be called more than once for the same applet if the user keeps returning and leaving the page with the applet. Its function is to halt (pause) the applet from running (and thus prevent wasting system resources) when the user is not paying attention to the applet. You should not call this method explicitly. Finally, the destroy() method is called when the browser shuts down. This also happens automatically and for now does not need to be explicitly implemented. It reclaims any system resources that the applet was using.

The Applet class is a subclass of the Panel, Container and Component classes found in the AWT package which we met in the last seminar. To create an applet you must create a subclass of the Applet class and include within it a method with the signature init() that overrides the init() method of the Applet class.

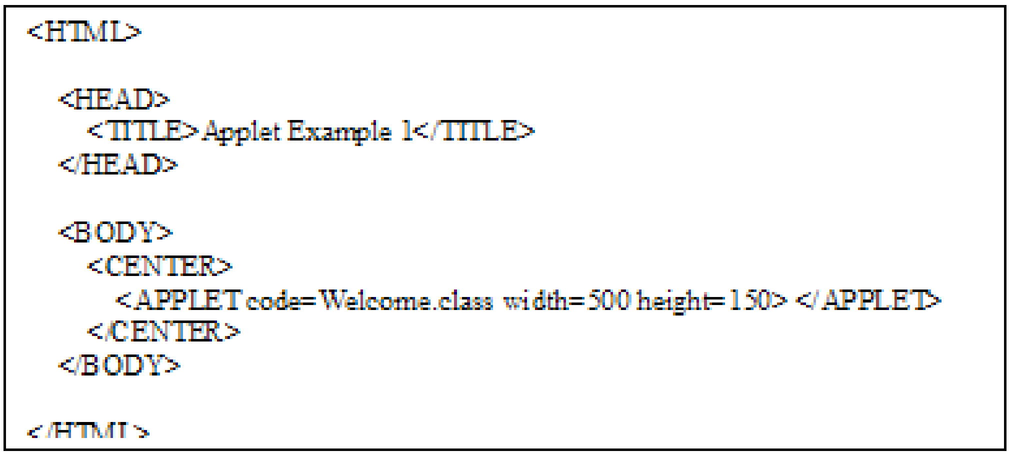
The init() method is implicitly invoked by the browser when the applet is discovered. Therefore, you do not need to explicitly call it. If you like you can think of init() as the applet equivalent of the main() method used in Java application programs. So there is no main() method for applets and there is only one init() method per applet.

A simple example applet is given in Figure 2. Note that a user defined subclass of the Applet class must always be present.

```
//Applet Example
//Kimberly Watson
//Thursday 10 June 2004
//The University of Liverpool, UK

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

public class Welcome extends Applet{

   //override init() to set background color

   public void init(){
      setBackground(Color.yellow);
   }

   //override paint() to display infromation

   public void paint(Graphics g){
      Font f = new Font("Serif", Font.BOLD, 48);
      g.setFont(f);
      g.setColor(Color.blue);
      g.drawString("Welcome to Applets!", 40, 50);
   }
}
```

Figure 2: *First applet example*

For the above applet to work we need an appropriately written WWW page containing the name of the applet and its desired width and height. Figure 3 presents an appropriate HTML file that I called myWebPage.html.



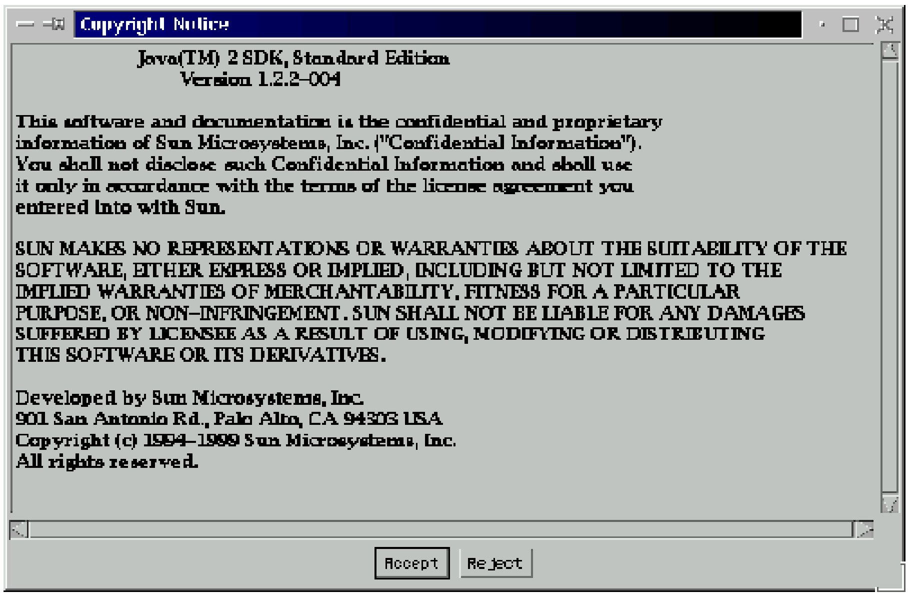
```
<HTML>

   <HEAD>
      <TITLE>Applet Example 1</TITLE>
   </HEAD>

   <BODY>
      <CENTER>
         <APPLET code=Welcome.class width=500 height=150> </APPLET>
      </CENTER>
   </BODY>

</HTML>
```

Figure 3: *HTML page for the Welcome applet*

Note that for the above HTML page, since we have not specified a path to the Welcome class file, both the MyWebPage.html and the Welcome.class files must be within the same directory. Once compiled, to run the applet we can view it through an appropriately enabled WWW browser. Alternatively we can use the appletviewer program that comes with the JDK by typing:

```
appletviewer MyWebPage.html
```

For all the examples given in this seminar, the latter approach is recommended. When you first invoke the appletviewer program, you will probably be asked to agree to some copyright agreement of the form presented in Figure 4. Having done this you will be able to view your applet - the result of the above should be as depicted in Figure 5.



Copyright Notice

Java(TM) 2 SDK, Standard Edition
Version 1.2.2-004

This software and documentation is the confidential and proprietary information of Sun Microsystems, Inc. ("Confidential Information"). You shall not disclose such Confidential Information and shall use it only in accordance with the terms of the license agreement you entered into with Sun.

SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

Developed by Sun Microsystems, Inc.
901 San Antonio Rd., Palo Alto, CA 94303 USA
Copyright (c) 1994-1999 Sun Microsystems, Inc.
All rights reserved.

Figure 4: *Appletviewer copyright agreement*

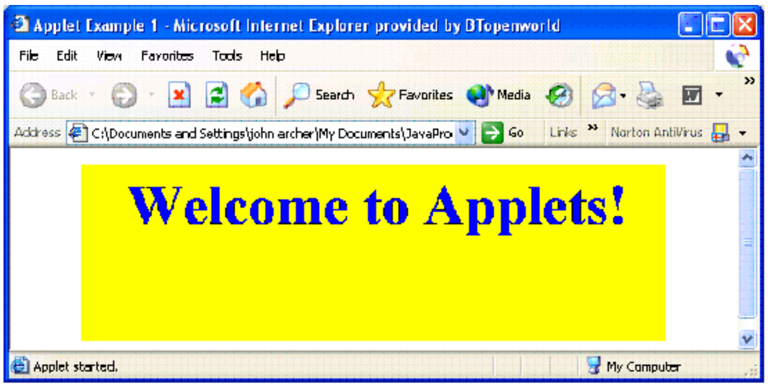

Applet Example 1 - Microsoft Internet Explorer provided by BTopenworld
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Documents and Settings\john archer\My Documents\JavaPro
Go Links Norton AntiVirus
Welcome to Applets!
Applet started.
My Computer

Figure 5: *Result from running Welcome.class using the appletviewer program*

In the above example we have used methods from the Graphics class, yet we did not set up a window to draw the graphics. As well as this, we could close the window by pointing at the X button in the top right-hand corner of the window, yet no window event listeners were declared within the program.

The above is explained by the fact that applets take advantage of certain facilities offered by the host (web browser or applet viewer) software. For example:

1. Applets may run in the browser window.

2. Event-handling already exists for the browser and can be shared by the applet.

3. The Graphics class may be used in the context of the browser's window.

4. The interface of the web browser or applet viewer may also be used to control the applet - for example, stopping the applet from running.

Before we go into a slightly more complicated example of applets, we will now briefly take a look at some of the security issues that you should be aware of in relation to applets (simply because one should be aware of these security issues).

III. Security Issues for Applets

All Java code that is installed locally on a machine is trusted implicitly. On the other hand, all Java code that is downloaded over the network is not trusted and run in a restricted environment that is called the sandbox. This is because if the user's web browser is Java enabled, it will download all the applet code on a web page and execute it immediately - the user does not have control over the applets running. Therefore, security is a major issue. The access control policies of the sandbox are defined by the currently installed *java.lang.SecurityManager object.* The applet security manager restricts what applets can do by throwing a SecurityException every time an applet tries to break a security rule.

So what are applets restricted from doing from within the sandbox? Here is a list:

1. They can never run any local executable program

2. They can not communicate with any host other than the server from which they were downloaded (originating host).

3. Applets cannot read or write to the local computer's file system.

4. Applets cannot find any information about the local computer e.g. users e-mail, users name etc. They can, however, find what Java version is used, the name and version of the operating system and characters used to separate files(/ or \), paths(: or ;) and lines(\n or \r\n).

5. Finally, all windows that an applet pops up in carry a warning message.

So this leads to the question - what can applets do? They can show images and play sounds, get keystrokes and mouse clicks from the user, and they can send user input back to the host from which they were loaded.

The security restrictions are possible due to the fact that applets are interpreted by the JVM before they are executed and thus are not directly executed by the user computer's cpu. The interpreter checks all critical instructions and program areas thus a hostile (or poorly written) applet will not be able to crash the computer, overwrite system memory, or change privileges granted by the operating system.

Often these restrictions are too strong (e.g. on a corporate intranet, applets would be required to access local files). To allow for different levels of security you can use signed applets. A signed applet carries a secure certificate that indicates where the applet comes from. If you trust the signer of the applet you can give the applet more privileges by configuring your browser to trust downloaded code that bears the valid digital signature. Completely trusted applets can have the same privileges as local applications. Here, I will not discuss such issues any further as the aim of this section is simply to make you aware that security issues do exist.

IV. A Second Applet Example

As we saw in the previous example, the init() method of the applet class is automatically called by the browser once the applet is loaded. We also learned that the Graphics class may be used in the context of the browser's window. Thus we can set up simple GUIs from within the init() method. **As well as this, all the OO features of Java programming should still be maintained when writing applets.** The following example in **Figure 6** and **Figure 7** illustrates this. Basically, it creates a very simple interface that allows the user to enter some data, after which some calculations on this data are performed. The results are then output to a text area of the user interface. The GUI is created within the init() method of the StatDesk class. This class then passes the user input data to another class, StatMethods, which performs the calculations and then sends the results back to the StatDesk class to be displayed.

Figure 7 contains the StatDesk class which contains the init() method, which has been used to create a simple interface. There is no exception handling for this class at the moment. An optional exercise is to add such handling.

```
//Second Applet Example
//Kimberly Watson
//Thursday 10 June 2004
//The University of Liverpool, UK

import java.io.*;
import java.util.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class StatDesk extends Applet implements
ActionListener{

   static BufferedReader keyboard = new
       BufferedReader(new
InputStreamReader(System.in));

   private TextField data;
   private TextArea information;
   private Button pushButton;
   private int TrialNo;

   public void init(){
        setLayout(null);
        setBackground(Color.cyan);
        TrialNo = 1;

        pushButton = new Button("Analyze");
        pushButton.setLocation(10,220);
        pushButton.setSize(380,30);
        pushButton.setBackground(Color.red);
        add(pushButton);
        pushButton.addActionListener(this);

        data = new TextField(1000);
        Label l1 = new Label("Enter samples:");
        l1.setLocation(10,190);
        l1.setSize(80,20);
        add(l1);
        data.setLocation(100,190);
        data.setSize(290,20);
        add(data);
        data.addActionListener(this);

        information = new TextArea();
        information.setEditable(true);
        information.setLocation(10, 10);
        information.setSize(380, 170);
        add(information);
    }

    public void actionPerformed(ActionEvent event){

        if(event.getActionCommand().equals("Analyze")){
            String token;
            String sampleData = data.getText();
            StringTokenizer dataStuff = new
                        StringTokenizer(sampleData);
            int NoOfSamples = dataStuff.countTokens();

            double data[ ] = new double[NoOfSamples];

            for(int x=0; x!=NoOfSamples; x++){
                token = dataStuff.nextToken();
                double next = new
Double(token).doubleValue();
```

```
                data[x] = next;
            }

            StatMethods sm = new StatMethods(data);

            OutputDialog("----------------------------------------");
            OutputDialog("Trial No. = " + TrialNo+" ");
            OutputDialog("sumation = " + sm.getSum()+" ");
            OutputDialog("mean = " + sm.getAverage()+" ");
            OutputDialog("variance = "+ sm.getVariance()+"
");
            OutputDialog("standard deviation = " +
                        sm.getStandardDeviation()+" ");
            TrialNo++;

            //reset data array to 0's

            for(int x=0; x!=data.length; x++)
                data[x] = 0;

        }//end if
    }//end method

    // Prints data to TestArea infromation
    private void OutputDialog(String data) {
        information.append(data);
        information.append("\n");
    }
}
```

Figure 6: *StatDesk class containing the GUI for our applet*

Figure 7 contains the StatMethods class which contains all the methods that are used to calculate results.

```
public class StatMethods{
    private double[ ] sampleData;

    //constructor
    public StatMethods(double [ ] sd){
        sampleData = new double[sd.length];
        for(int x=0; x!=sd.length; x++)
            sampleData[x] = new
Double(sd[x]).doubleValue();
    }

    public double getAverage(){
        return getSum()/sampleData.length;
    }

    public double getSum(){
        double sum = 0;
        for(int x=0; x!=sampleData.length; x++)
            sum += sampleData[x];
        return sum;
    }

    public double getStandardDeviation(){
        int sdSum=0;
        for(int x=0; x!=sampleData.length; x++)
            sdSum += Math.pow((sampleData[x]-
                            getAverage()),2);
        return Math.sqrt(sdSum/(sampleData.length-1));
    }

    public double getVariance(){
        int sdSum=0;
        for(int x=0; x!=sampleData.length; x++)
            sdSum += Math.pow((sampleData[x]-
                            getAverage()),2);
        return sdSum/(sampleData.length-1);
    }
}
```

Figure 7: *StatMethods class containing some simple statistical methods that our GUI class will use*

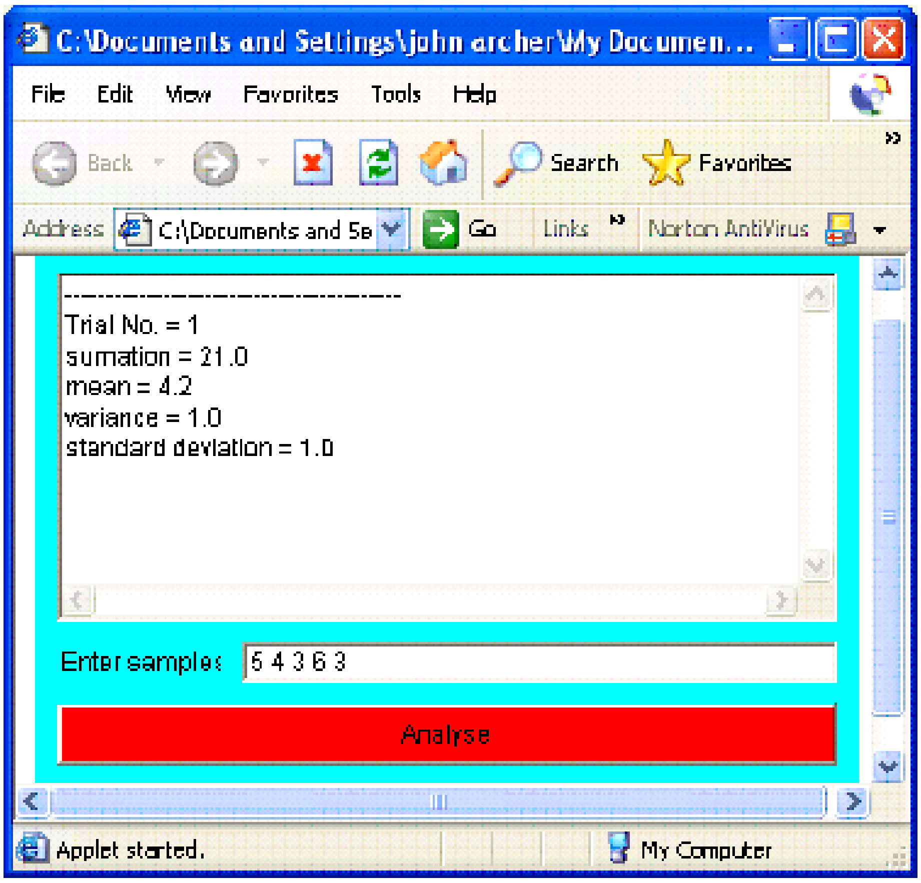
The HTML file required can be seen in **Figure 8**.

```
<HTML>

    <HEAD>
        <TITLE>Applet Example 2</TITLE>
    </HEAD>

    <BODY>
       <CENTER>
          <APPLET code=StatDesk.class width=400
                      height=260> </APPLET>
       </CENTER>
    </BODY>

</HTML>
```

Figure 8: *HTML code for example 2*

Note here that the tag includes the class that contains the init() method.

To perform a simple statistical analysis on a sequence of user input numbers, simply enter the sequence into the appropriate data entry field, leaving a space in between each entry (do not use commas or full stops etc.). Once satisfied that all the samples have been entered, press the analyze button to view the mean, variance, standard deviation and the summation of the entered data. A sample output is found in **Figure 9**.



```
------------------------------------
Trial No. = 1
sumation = 21.0
mean = 4.2
variance = 1.0
standard deviation = 1.0
```

C:\Documents and Settings\john archer\My Documen...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address C:\Documents and Se
Go
Links
Norton AntiVirus
Enter samples 5 4 3 6 3
Analyse
Applet started.
My Computer

Figure 9: *Example output from example 2*

V. Graphics

Something that many students wish to do once they have started to understand GUIs and applets is to include some drawing (graphics). The Component class in the AWT package contains a method paint(), which is automatically invoked whenever a component is drawn or redrawn. Through the mechanism of inheritance, this method is inherited by all sub-classes of the Component class, however, we must override the paint() method.

To draw using paint(), we must first define a drawing area. The awt package contains an appropriate component, Canvas, which describes such an area. The methods required to produce a drawing can be found in the Graphics class. Note that a Canvas is the same as any other component and thus must be placed in a container. When the container is made visible, this will then automatically cause the paint() method to be invoked.

In the code presented in **Figure 10**, we create a class AppletEx3 which extends the class applet in the usual manner. This has a single component MyCanvas. MyCanvas is user defined class which extends the class Canvas and includes our own version of the paint() method contained in the Component class. The paint() method comprises a sequence of methods taken from the Graphics class which will produce some output of the form shown in **Figure 11**. The parameter g refers to an object of the class Graphics which comes with the JDK.

```
//Applet Example 3
//Kimberly Watson
//Thursday 10 June 2004
//The University of Liverpool, UK

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

public class AppletEx3 extends Applet {

  MyCanvas1 pic = new MyCanvas1();

  public void init() {
      setBackground(Color.yellow);
      setLayout(null);
      addCanvas(pic,400,300,50,70);
  }

   private void addCanvas(MyCanvas1 name, int width,
                     int height, int xCoord, int yCoord) {
      name.setBackground(Color.white);
        name.setLocation(xCoord,yCoord);
        name.setSize(width,height);
        add(name);
   }
}

class MyCanvas1 extends Canvas {

   public void paint(Graphics g) {
```

```
// Car body
   g.setColor(Color.red);
   g.fillRect( 70, 40,270,100);
   g.fillRect( 60,140,290,100);
   g.setColor(Color.black);
   g.drawRect( 70, 40,270,100);
   g.drawRect( 60,140,290,100);

   // Winscreen
   g.setColor(Color.cyan);
   g.fillRect( 80, 50,250, 85);

   // Wheels
   g.setColor(Color.black);
   g.fillRect( 80,240, 50, 50);
   g.fillRect(280,240, 50, 50);

   // Headlights
   g.setColor(Color.yellow);
   g.fillOval( 80,155, 50, 50);
   g.fillOval(280,155, 50, 50);
   g.setColor(Color.black);
   g.drawOval( 80,155, 50, 50);
   g.drawOval(280,155, 50, 50);

   // Radiator
   int xStart = 140;
   int yStart = 155;
   for(int index=0;index<60;index=index+10) {
      g.drawLine(xStart,yStart+index,xStart+130,
                 yStart+index);
      }

   // Bumper
   g.setColor(Color.lightGray);
   g.fillOval( 40,210, 40, 40);
   g.fillOval(330,210, 40, 40);
   g.fillRect( 60,210,290, 40);

   // Road
   g.drawLine( 10,290,390,290);
```

```
        // String

        g.setColor(Color.black);
        Font mono = new
                  Font("Monospaced",Font.BOLD,20);
        g.setFont(mono);
        FontMetrics newFM = g.getFontMetrics(mono);
        String s1 = "Java Jeep";
        int xLength = newFM.stringWidth(s1);
        g.drawString(s1,(205-(xLength/2)),235);
    }
}
```

Figure 10: *Graphics applet*

The graphics methods used in **Figure 10** are as follows:

- setColor(Color c): Sets the colour for the graphic to be drawn, c must be taken from the Color Class.

- fillRect(x,y,width,height): Draw a solid rectangle with its top left hand corner located at coordinates x-y with respect to the top left hand corner of the drawing area.

- drawRect(x,y,width,height): Draw an unfilled rectangle (in this case outlining the previous rectangle).

- fillOval(x,y,width,height): Draw a solid oval with the given dimensions (note that to draw a circle the width and height arguments must be equal).

- drawOval(x,y,width,height) : Draw an unfilled oval/circle.

- drawLine(x1,y1,x2,y2): Draw a line starting at coordinates x1-y1 and ending at coordinates x2-y2.

- drawString(s1,x,y): Draw a string (s1) at the specified location.

It is possible to draw quite sophisticated graphics using the limited number of methods presented above. Note that in the code given in **Figure 10**, the radiator is drawn using a for loop construct.

Something else I have slipped into the code in **Figure 10** is a font specification, the general form for which is:

Font name = new Font(String s1, Font.style, int size);

There are five fonts that come with the JDK as standard: Serif (TimesRoman), SansSerif (Helvetica), Monospaced (Courier), Dialog and DialogInput. The available Font styles are: PLAIN, BOLD, ITALIC. Thus we might declare a font, tr, as follows:

Font tr = new Font("Serif",Font.PLAIN,12);

The other thing I have done in the example (and this is advanced stuff) is to create an instance of the class FontMetrics, which then allows us the access the instance method stringWidth() which returns the pixel length of its argument which must be string. I then use this value to centre the text on the jeep's bumper!

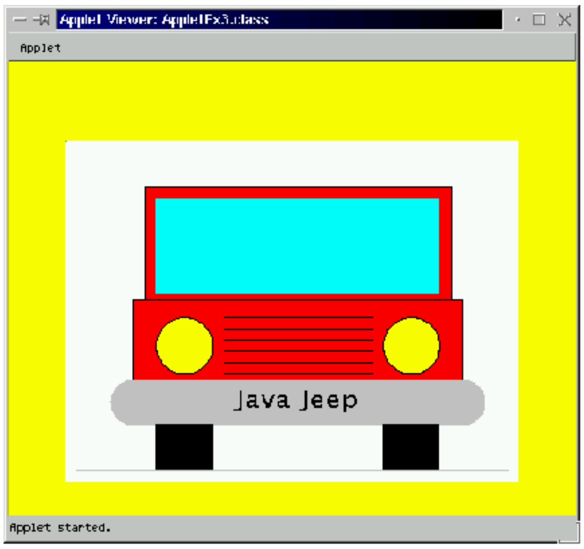


Figure 11: *Applet example 3*

VI. Graphics Together With Other Components

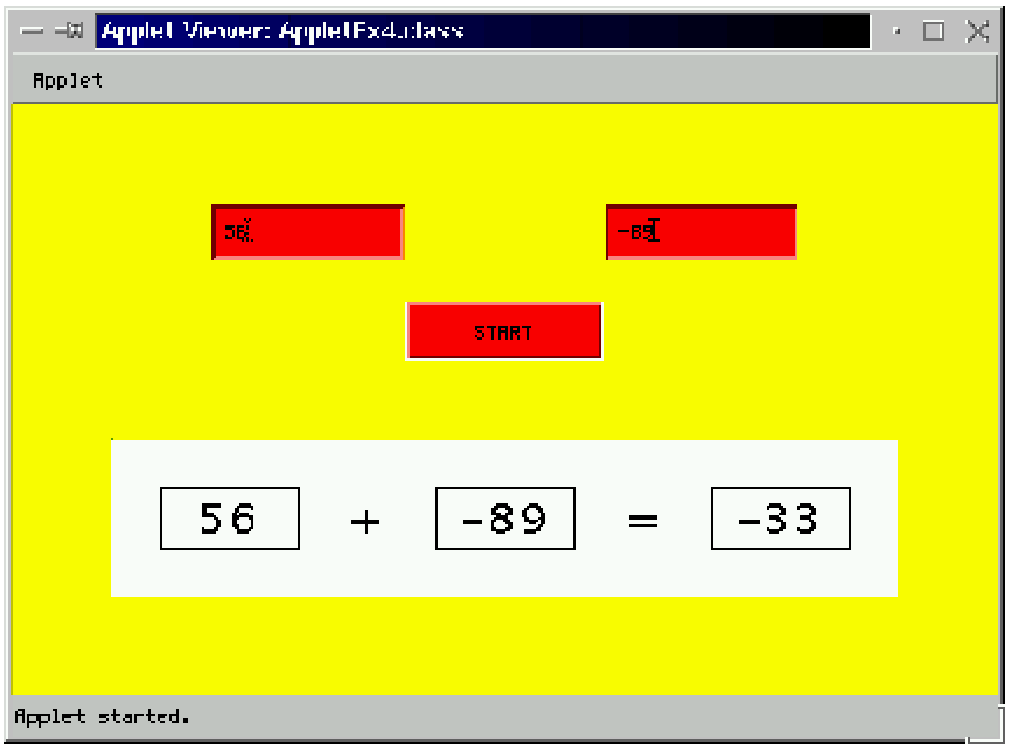
The last example in this seminar simply puts a GUI applet together with a graphics applet. There is nothing new covered in this example, it serves only as one last example which might prove useful when tackling the practical at the end of this seminar. The output that results from this code is presented in **Figure 13**.

```
//Applet Example 4
//Kimberly Watson
//Thursday 10 June 2004
//The University of Liverpool, UK

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

public class AppletEx4 extends Applet implements
ActionListener {

   MyCanvas2 pic = new MyCanvas2();

   TextField input1 = new TextField(10);
   TextField input2 = new TextField(10);
   TextField output = new TextField(10);

   Button pushButton = new Button("START");

   private final int WIDTH  = 100;
   private final int HEIGHT = 30;
   public static int dataItem1    = 0;
   public static int dataItem2    = 0;

   public void init() {
      setBackground(Color.yellow);
      setLayout(null);

      addTextField(input1,100,50);
      addTextField(input2,300,50);
      output.setEditable(false);

      // Add push button

      addPushButton(pushButton,200,100);
   // Add push Canvas

   addCanvas(pic,400,80,50,170);
   }

private void addTextField(TextField name, int xCoord,
                                    int yCoord) {
   name.setBackground(Color.red);
      name.setForeground(Color.black);
      name.setLocation(xCoord,yCoord);
      name.setSize(WIDTH,HEIGHT);
      add(name);
      name.addActionListener(this);
   }

/* Add push button */

private void addPushButton(Button name, int xCoord,
                                    int yCoord) {
   name.setBackground(Color.red);
      name.setForeground(Color.black);
      name.setLocation(xCoord,yCoord);
      name.setSize(WIDTH,HEIGHT);
      add(name);
      name.addActionListener(this);
   }

/* Add canvas */

private void addCanvas(MyCanvas2 name, int width,
                  int height, int xCoord, int yCoord) {
   name.setBackground(Color.white);
      name.setLocation(xCoord,yCoord);
      name.setSize(width,height);
      add(name);
      }

/* Action Performed */

public void actionPerformed(ActionEvent event) {

   // Get Action
```

```
      String data = event.getActionCommand();

      // Test for push Button or text field

      if (event.getSource() == pushButton)
                        addCanvas(pic,400,80,50,170);
         else if (event.getSource() == input1)
               dataItem1 = (new Integer(data)).intValue();
      else dataItem2 = (new Integer(data)).intValue();
      }
   }


public class MyCanvas2 extends Canvas {


 /* Fonts */

   //Graphics g;
   Font mono = new Font("monospaced",Font.BOLD,20);

   public void paint(Graphics g) {
      String s1;

      // Rectangles

      g.setColor(Color.black);
      g.setFont(mono);
      g.drawRect( 25, 25, 70, 30);
      g.drawRect(165, 25, 70, 30);
      g.drawRect(305, 25, 70, 30);

      // Output

outputString(g,Integer.toString(AppletEx4.dataItem1),
                                          60,48);
      outputString(g,"+",130,48);

outputString(g,Integer.toString(AppletEx4.dataItem2),
                                          200,48);
      outputString(g,"=",270,48);
```

```
        int total = AppletEx4.dataItem1 +
AppletEx4.dataItem2;
        outputString(g,Integer.toString(total),340,48);
        }

    private void outputString(Graphics g,String s1,int
         xCoord, int yCoord) {

        // Font metric

        FontMetrics newFM = g.getFontMetrics(mono);

        // Output

        int xLength = newFM.stringWidth(s1);
        g.drawString(s1,(xCoord-(xLength/2)),yCoord);
          }
    }
```

Figure 12: *Applet example 4*



```
56  +  -89  =  -33
```

```
Applet started.
```

Figure 13: *Applet example 4 output*