Module 5

Applets and Swings

Applets

- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- **Applet** is a predefined class in **java.applet** package used to design distributed application. It is a client side technology. Applets are run on web browser.
- **Two Types of Applets**

  1 .Local applet - operate in single machine browser which is not connected in network,

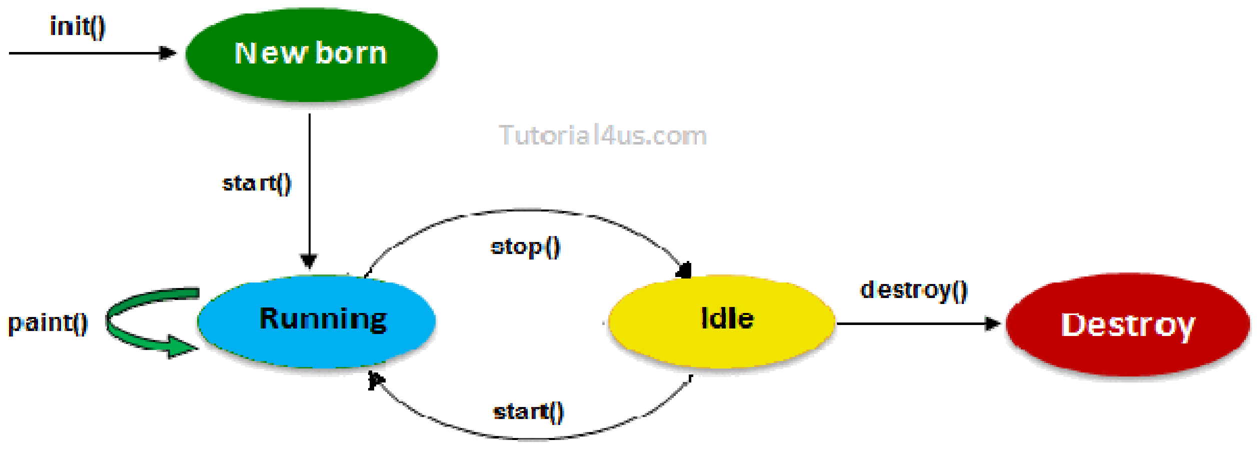
  2.Remote applet - remote applet operate over internet via network.

Advantage of Applet

- Applets are supported by most web browsers.
- Applets works at client side so less response time.
- **Secured:** No access to the local machine and can only access the server it came from.
- Easy to develop applet, just extends applet class.
- To run applets, it requires the Java plug-in at client side.

Life cycle of applet

- init()
- start()
- stop()
- destroy()



init(): Which will be executed whenever an applet program start loading, it contains the logic to initiate the applet properties.

start(): It will be executed whenever applet program starts running.

stop(): Which will be executed whenever applet window or browser is minimized.

destroy(): It will be executed whenever applet window or browser is going to be closed (at the time of destroying the applet program permanently).

Paint():paint method is used to display the content on frame.paint() method work as println() method.

<u>Difference between Applet and Application</u>

	Java Applet	Java Application
User graphics	Inherently graphical	Optional
Memory requirements	Java application requirements plus Web browser requirements	Java application requirements
Distribution	Linked via HTML and transported via HTTP	Loaded from the file system or by a custom class loading process
Environmental input	parameters embedded in the host HTML document	Command-line parameter

Method expected by the virtual machine	init()
	start()
	stop()/pause()
		main()

	destroy()
	paint()	

Typical applications	Public-access. Order-entry systems for the web, online multimedia presentations web page animation	Network server, multimedia kiosks, developer tools appliance and consumer electronics, control and navigation

General structure of applet/Skaleton of applet

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

class  class_name extends Applet
{
        public void init()
        {
        }
        public void start()
        {
        }
        public void paint(Graphics obj)
        {
                Obj.drawString(string,int,int);
        }
        public void stop()
        {
        }
        public void destroy()
        {
        }
}
```

Simple applet program:

```
import java.awt.*;

import java.applet.*;

class A extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("wel come",100,100)
        }
}
```

HTML File

```
<HTML>
        <HEAD>
                <TITLE> My Second Applet </TITLE>

        </HEAD>

        <BODY>
                <applet code =" A.class" width=400 height=400>
                </applet>

        </BODY>
</HTML>
```

Requesting repainting:

- Repaint is a method which is used to display the banner message.
- Repaint method can be used in either paint() method or in update() method
- We can call repaint() method when you want the applet area to be re drawn.
- The default action of update() method is to clear the applet area and call the paint().
- void repaint() :- causes entire window to be repainted .
- void repaint(int left, int top, int width, int height) :- Specifies region that will be repainted.Left and top are co-ordinates of upper left corner, width and height of the region.
- Void repaint(long maxDelay) :-specifies maximum number of milliseconds that can elapse before update is called.

```
import java.awt.*;
import java.applet.*;
class RepaintDemo extends Applet
{
public void paint(Graphics g)
{
repaint(100); // with max delay
repaint(100,left,top,100,100); // specific position will repaint
g.drawString("wel come",100,100)
}
}
```

HTML File

```
<HTML>
        <HEAD>
                <TITLE> My Second Applet </TITLE>

        </HEAD>

        <BODY>
                <applet code =" RepaintDemo.class" width=400 height=400>
                </applet>

        </BODY>
</HTML>
```

getDocumentBase() and getCodeBase():

getDocumentBase():

- is a method which gives the path of html file located which is under execution.
- Is used to return the URL of the document in which applet is embedded.
- **Syntex: URL obj=getDocumentBase();**

getCodeBase():

- is a method which gives the location of class file which is under execution.
- Is used to return the base URL.

- Syntex:

  URL obj=getCodeBase();

Example:

```
import java.awt.*;
import java.applet.*;
import java.net.*;

class A extends Applet
{
public void paint(Graphics g)
{
URL u1=getDocumentBase();
URL u2=getCodeBase();
g.drawString("Document base="+String.valueOf(u1),100,100);
g.drawString("Code base="+String.valueOf(u2),200,100);
}
}
```

HTML File

```
<HTML>
        <HEAD>
                <TITLE> My Second Applet </TITLE>

        </HEAD>

        <BODY>
                <applet code ="A.class" width=400 height=400>
                </applet>

        </BODY>
</HTML>
```

HTML applet tag:

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer. An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers like Netscape Navigator, Internet Explorer, and HotJava will allow many applets on a single page.

The syntax for the standard APPLET tag is shown here. Bracketed items are optional.

< APPLET
[CODEBASE = *codebaseURL*]
CODE = *appletFile*
[ALT = *alternateText*]
[NAME = *appletInstanceName*]
WIDTH = *pixels* HEIGHT = *pixels*
[ALIGN = *alignment*]
[VSPACE = *pixels*] [HSPACE = *pixels*]
>
[< PARAM NAME = *AttributeName* VALUE = *AttributeValue*>]
[< PARAM NAME = *AttributeName2* VALUE = *AttributeValue*>]
. . .
[*HTML Displayed in the absence of Java*]
</APPLET>

- CODEBASE CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag). The HTML document's URL directory is used as the CODEBASE if this attribute is not specified. The CODEBASE does not have to be on the host from which the HTML document was read.
- CODE CODE is a required attribute that gives the name of the file containing your applet's compiled **.class** file. This file is relative to the code base URL of the applet,

which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

- ALT The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser understands the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.
- NAME NAME is an optional attribute used to specify a name for the applet instance. Applets must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name, use **getApplet( )**, which is defined by the **AppletContext** interface.
- WIDTH AND HEIGHT WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.
- ALIGN ALIGN is an optional attribute that specifies the alignment of the applet. This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.
- VSPACE AND HSPACE These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.
- PARAM NAME AND VALUE The PARAM tag allows you to specify applet specific arguments in an HTML page. Applets access their attributes with the **getParameter( )** method.

Passing Parameters to Applets

- Parameters are passed to applets in NAME=VALUE pairs in <PARAM> tags between the opening and closing APPLET tags.
- Inside the applet, you read the values passed through the PARAM tags with the getParameter() method of the java.applet.Applet class

example: **program to set background and foreground color and using passing parameter for fontname,fontsize,typecasting(IMP vtu question)**.

```
<applet code="A.class" width=300 height=80>
<param name=fontName value=Courier>
<param name=fontSize value=14>
<param name=typecasting value=inttofloat>
</applet>

import java.awt.*;
import java.applet.*;
public class A extends Applet
{
String fontName;
Sting fontSize;
Sting typecasting;
public void paint(Graphics g)
{
fontName=getParameter("fontname");
fontSize=getParameter("fontsize");
typecasting=getParameter("typecasting");
setBackGround(Color.cyan);
setForeGround(Color.blue);

g.drawString("Font name: " + fontName, 0, 10);
g.drawString("Font size: " + fontSize, 0, 26);
g.drawString("Typecasting: " + typecasting, 0, 42);
g.drawString("Account Active: " + active, 0, 58);
}
}
```

HTML File for parameter reading

```
<HTML>
<HEAD>
<TITLE> My Second Applet </TITLE>

</HEAD>

<BODY>
<applet code ="Applet1.class" width=400 height=400>
<PARAM name="name" value="Hello Applet">
<PARAM name=fontName value=Courier>
<PARAM name=fontSize value=14>
<PARAM name=typecasting value=inttofloat>

</applet>

</BODY>
</HTML>
```

AudioClip Interface

Below example for loading and playing audio clip from applet programming

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

public class LoadSound extends Applet
{
        AudioClip audioClip;
        public void init()
        {
                audioClip=getAudioClip(getDocumentBase(),"SSS.mp3");
                setBackground(Color.red);

        }

        public void paint(Graphics g)
        {
                audioClip.play();

        }
}
```

HTML File

```
<HTML>
        <HEAD>
                <TITLE> My Second Applet </TITLE>

        </HEAD>

        <BODY>
                <applet code ="LoadSound.class" width=400 height=400>
                </applet>

        </BODY>
</HTML>
```

Swings

Swing is built on top of AWT and is entirely written in Java, using AWT's lightweight component support. In particular, unlike AWT, t he architecture of Swing components makes it easy to customize both their appearance and behavior. Components from AWT and Swing can be mixed, allowing you to add Swing support to existing AWT-based programs. For example, swing components such as JSlider, JButton and JCheckbox could be used in the same program with standard AWT labels, textfields and scrollbars.

Swing is a set of classes that provides more powerful and flexible GUI components than does the AWT. Simply put, Swing provides the look and feel of the modern Java GUI.

The Origins of Swing

Problems of native peers are:

- ✓ Because of variations between operating systems, a component might look,or even act differently on different platform.
- ✓ The look and feel of each component was fixed(because it is defined by the platform)and could not be changed.
- ✓ The use of heavyweight components caused some frustrating restrictions ie., they are rectangular & opaque.

These problems are solved by:

- ✓ Swing components are lightweight, ie., they are written entirely in java and do not map directly to problem-specific peers.
- ✓ These components are using graphics primitives, they can be transparent, which enables nonrectangular shapes.
- ✓ Because lightweight components do not translate into native peers and they are not underlying
- ✓ Operating system ie., components work in a consistent manner across all platforms.

Two Key Swing Features

- **Swing Components Are Lightweight:** Swing components are *lightweight*. This means that they are written entirely in Java and do not map directly to platform-specific peers. Because lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes. Thus, lightweight components are more efficient and more flexible. Furthermore, because lightweight components do not translate into native peers, the look and feel of each component is determined by Swing, not by the underlying operating system. This means that each component will work in a consistent manner across all platforms.

- **Swing Supports a Pluggable Look and Feel:** Swing supports a *pluggable look and feel* (PLAF).

  Because each Swing component is rendered by Java code rather than by native peers, the look and feel of a component is under the control of Swing. This fact means that it is possible to separate the look and feel of a component from the logic of the component, and this is what Swing does. PLAF advantages are as follows:

  (i) It is possible to define a look and feel that is consistent across all platforms. (ii)it is possible to create a look and feel that acts like a specific platform.

  (iii)it is possible to design accustom look and feel

  (iv)The look and feel can be changed dynamically at run time.

Swing Components and Containers

A Swing GUI consists of two key items: *components* and *containers*. a *component* is an independent visual control, such as a push button or slider. A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components. Furthermore, in order for a component to be displayed, it must be held within a container. Thus, all Swing GUIs will have at least one container. Because containers are components, a container can also hold other containers. This enables Swing to define what is called a *containment hierarchy*, at the top of which must be a *top-level container*.

Components

- ✓ In general, Swing components are derived from the JComponent class. JComponent provides
- ✓ the functionality that is common to all components. For example, JComponent supports the pluggable look and feel.
- ✓ JComponent inherits the AWT classes Container and Component.
- ✓ A Swing component is built on and compatible with an AWT component.
- ✓ All of Swing's components are represented by classes defined within the package javax.swing.
- ✓ All component classes begin with the letter J.
- ✓ For example, the class for a label is JLabel; the class for a push button is JButton; and the class for a scroll bar is JScrollBar.

Containers

1.
    - ✓ The first are top-level containers: JFrame, JApplet, JWindow, and JDialog. These containers do not inherit JComponent. They inherit the AWT classes Component and Container.
    - ✓ The top-level containers are heavyweight. This makes the top-level containers a special case in the Swing component library.
    - ✓ As the name implies, a top-level container must be at the top of a containment hierarchy. A top-level container is not contained within any other container. Every containment hierarchy must begin with a top-level container.
    - ✓ The one most commonly used for applications is JFrame. The one used for applets is JApplet.
2.
    - ✓ The second type of containers supported by Swing are lightweight containers. Lightweight containers *do* inherit JComponent.
    - ✓ An example of a lightweight container is JPanel, which is a general-purpose container. Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container.
    - ✓ We can use lightweight containers such as JPanel to create subgroups of related controls that are contained within an outer container.

The Swing Packages

Swing is a very large subsystem and makes use of many packages. These are the packages used by Swing that are defined by Java SE 6. The main package is **javax.swing**. This package must be imported into any program that uses Swing. It contains the classes that implement the basic Swing components, such as push buttons, labels, and check boxes.

A simple Swing Application

Swing programs differ from both the console-based programs and the AWT-based programs. Swing programs also have special requirements that relate to threading. There are two types of Java programs in which Swing is typically used. The first is a desktop application. The second is the applet.

To create a Swing application: In the process, it demonstrates several key features of Swing. It uses two Swing components: JFrame and JLabel. JFrame is the top-level container that is commonly used for Swing applications. JLabel is the Swing component that creates a label, which is a component that displays information. The label is Swing's simplest component because it is passive. That is, a label does not respond to user input. It just displays output. The program uses a JFrame container to hold an instance of a JLabel. The label displays a short text message.

Simple Swing Application

```
import javax.swing.*;
class Employee
{
        Employee()
        {
                JFrame jf = new JFrame("Swing Application");
                jf.setSize(400,200);
                jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                JLabel jl = new JLabel("Welcome");
                jf.add(jl);

                JButton b = new JButton("Click");
                b.setBound(130, 100, 200, 100);
                jf.add(b);
        }
        public static void main(String args[])
                {
                        new Employee();
                }
}
```

Create a simple swing applet program

- ✓ A Swing applet extends **JApplet** rather than **Applet**.
- ✓ **JApplet** is derived from **Applet**. Thus, **JApplet** includes all of the functionality found in **Applet** and adds support for Swing.
- ✓ **JApplet** is a top-level Swing container, which means that it is *not* derived from **JComponent**.
- ✓ **JApplet** is a top-level container, it includes the various panes. All components are added to **JApplet**'s content pane in the same way that components are added to **JFrame**'s content pane.
- ✓ Swing applets use the same four lifecycle methods: **init( )**, **start( )**, **stop( )**, and **destroy( )**. Painting is accomplished differently in Swing than it is in the AWT,
- ✓ Swing applet will not normally override the **paint( )** method. All interaction with components in a Swing applet must take place on the event dispatching thread.

Example:

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

public class Example extends JApplet {
JButton button1;
JButton button2;
JLabel label;

// Initialize the applet.
 public void init()
{   render(); // initialize the GUI
}

private void render()
{   // Set the applet to use flow layout.
setLayout(new FlowLayout());
```

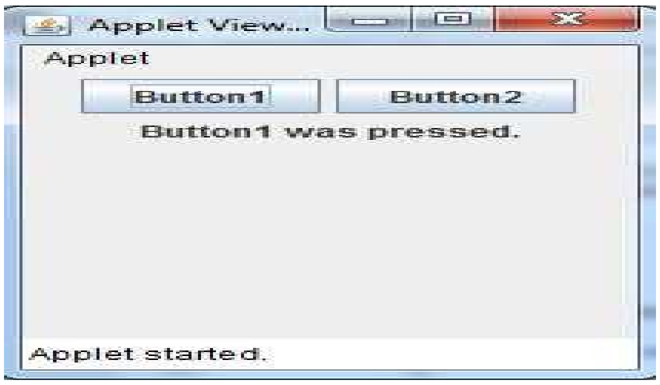
```
 // Make two buttons.
button1 = new JButton("Button1");
button2 = new JButton("Button2");

 // Add action listener for Alpha.
button1.addActionListener(new
ActionListener() {
 public void actionPerformed(ActionEvent le)
{     label.setText("Button1 was pressed.");    }
});

// Add action listener for Beta.
button2.addActionListener(new
ActionListener() {
public void actionPerformed(ActionEvent le)
{     label.setText("Button2 was pressed.");    }
});

// Add the buttons to the content pane.
add(button1);
add(button2);

// Create a text-based label.
label = new JLabel("Press a
button.");   // Add the label to
the content pane.
add(label);
}
}
```



Button1 was pressed.

Applet started.

Jlabel

Syntax : public class **JLabel** extends JComponent implements SwingConstants, Accessible

It is a display area for a short text string or an image, or both.

- A label does not react to input events. As a result, it cannot get the keyboard focus.
- A label can display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.
- A JLabel object can display either text, an image, or both.
- By default, labels are vertically centered in their display area.
- Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.
- Can use the setIconTextGap method to specify how many pixels should appear between the text and the image. The default is 4 pixels.

JTextField

- JTextField is a lightweight component that allows the editing of a single line of text.
- JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class.
- JTextField has a method to establish the string used as the command string for the action event that gets fired.
- The java.awt.TextField used the text of the field as the command string for the ActionEvent.
- JTextField will use the command string set with the setActionCommand method if not null, otherwise it will use the text of the field as a compatibility with java.awt.TextField.

Swing Buttons

Swing defines four types of buttons: JButton, JToggleButton, JCheckBox, and JRadioButton. All are subclasses of the AbstractButton class Which extends Jcomponent. AbstractButton contains many methods that allow you to control the behavior of buttons The model used by all buttons is defined by the ButtonModel interface.

JButton class provides the functionality of a push button.Jbutton allows an icon,a string, or both to be associated with the push button:

- ✓ JButton(Icon icon)
- ✓ JButton(String str)
- ✓ JButton(String str, Icon icon)

Here, str and icon are the string and icon used for the button.

ToggleButton : A useful variation on the push button is called ***a toggle button***. A toggle button looks just like a push button, but it acts differently because it has two states: pushed and released. One of the constructor is JToggleButton(String str)

This creates a toggle button that contains the text passed in str. By default the button is in the off position.

CheckBox class provides a functionality of a check box. Its immediate superclass is JToggleButton, which provides support for two-state buttons, as just described. JCheckBox defines several constructors Eg., JCheckBox(String str).

JRadioButton: Radio buttons are mutually exclusive buttons, in which only one button can be selected at any one time. They are supported by the JRadioButton class, which extends JToggleButton One of the Constructor is

JRadioButton(
String str);

Here,str is the label for
the button

AJRadioButton generates action events, item events, and change events each time the Architec button selection changes

JTabbedPane

Syntax : public class JTabbedPane extends JComponent implements Serializable, Accessible, SwingConstants

- A component that lets the user switch between a group of components by clickingon a tab with a given title and/or icon.
- Tabs/components are added to a TabbedPane object by using the addTab and insertTab methods.
- A tab is represented by an index corresponding to the position it was added in, where the first tab has an index equal to 0 and the last tab has an index equal to the tab count minus
- The TabbedPane uses a Single SelectionModel to represent the set of tab indices and the currently selected index. If the tab count is greater than 0, then there will always be a selected index, which by default will be initialized to the first tab. If the tab count is 0, then the selected index will be -1. **JScrollPane**

Syntax : public class JScrollPane extends JComponent implements ScrollPaneConstants,

Accessible

- Provides a scrollable view of a lightweight component.
- A JScrollPane manages a viewport, optional vertical and horizontal scroll bars, and optional row and column heading viewports.
- The JViewport provides a window, or "viewport" onto a data source – for example, a text file.

  That data source is the "scrollable client" (aka data model) displayed by the JViewport view.
- A JScrollPane basically consists of JScrollBars, a JViewport, and the wiring between them, as shown in the diagram at right.

JList

Syntax : public class JList extends JComponent implements Scrollable, Accessible

A component that allows the user to select one or more objects from a list. A separate model, ListModel, represents the contents of the list.

// Create a JList that displays the strings in data[]

String[] data = {"one", "two", "three", "four"}; JList dataList = new JList(data);

JComboBox

Syntax : public class JComboBox extends JComponent implements ItemSelectable, ListDataListener, ActionListener, Accessible

- A component that combines a button or editable field and a drop-down list.
- The user can select a value from the drop-down list, which appears at the user's request.
- If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

JTable

Syntax : public class JTable extends JComponent implements TableModelListener, Scrollable, TableColumnModelListener, ListSelectionListener, CellEditorListener, Accessible

• The JTable is used to display and edit regular two-dimensional tables of cells.

• The JTable has many facilities that make it possible to customize its rendering and editing but

provides defaults for these features so that simple tables can be set up easily.