Unit – II : AWT (Abstract Windows Toolkit)

Contents

- AWT Concept
- AWT components
- Containers
- Frames & Panels
- Event Delegation Model
- Event Source & Handler
- Event categories
- Listeners & Interfaces
- RMI concept
- RMI Architecture
- Stub & Skeleton
- RMI classes & interfaces
- Writing simple RMI application

⇨ *Question Bank*

- ***Concept of AWT (Abstract Windows Toolkit):***

AWT stands for **Abstract Windows Toolkit**, which is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of Operating System.

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List, etc.

In other words, **Abstract Windows Toolkit (AWT)** is Java's original platform-dependent windowing, graphics and user interface toolkit, preceding with Swing. The AWT is part of the Java Foundation Class(JFC) with standard API for providing a Graphical User Interface(GUI) for a Java program. AWT is also the GU toolkit for a number of Java Micro Edition profiles.

AWT Components:

A component is an object with a graphical representation that can be displaye

On the screen and that can interact with the user. The Component class is tl abstractof the non menu-related AWT components.

Following are commonly used AWT components in Java;

1. Button
2. Labels
3. Checkboxes
4. Choice Buttons
5. Text Fields
6. Radio Buttons

1.Button (java.awt.Button)

To create a Button object, simply create an instance of the Button class by calling one of the constructors. The most commonly used constructor of the Button class takes a String argument, that gives the Button object a text title. The two constructors are:

Button() // Constructs a Button with no label.

Button(String label) // Constructs a Button with the specified label.

2. Labels (java.awt.Label)

Allow us to add a text description to a point on the applet or application.

3.Checkboxes (java.awt.Checkbox)

Checkboxes have two states, on and off. The state of the button is returned as the Object argument, when a Checkbox event occurs. To find out the state of a checkbox object we can use getState() that returns a true or false value. We can also get the labe of the checkbox using getLabel() that returns a String object.

4.Choice Buttons (java.awt.Choice)

Like a radio button, where we make a selection, however it requires less space and allows us to add items to the menu dynamically using the addItem() method.

5.TextFields (java.awt.TextField):

The areas where the user can enter text. They are useful for displaying and eceiving text messages. We can make this textfield read-only or editable. We can he setEditable(false) to set a textfield read-only. There are numerous ways that w onstruct a Textfield object.

6.Radio Buttons (java.awt.CheckboxGroup):

Is a group of checkboxes, where only one of the items in the group can be elected only at one time.

AWT Containers:

Containers are integral part of AWT GUI components. A container provides a sp where a component can be located. A Container in AWT is a component itself an adds the capability to add component to itself.

Following are some basic points to be considered with Containers:

- It uses its Sub classes such as Panel, Frame and Window.
- Container can add only Component to itself.
- A default layout is present in each container which can be overridden using setLayout( ) method.

Simply, Container is a generic container object which can contain other AWT components.Following is the list of commonly used containers while designed GU using AWT.

S.No.	Container	Description
1	Panel	Panel is the simplest container. It provides space in which any of componentcan be placed, including other panels.
2	Frame	A Frame is a top-level window with a title and a border.
3	Window	A Window object is a top-level window with no borders and no

Declaration :

java.awt.Container class has the following declaration:
public class Container extends Component

Some of the commonly used Container class methods are:

add() : Component is included in the container by overloading this method.

invalidate() : The components that are setup in the container can be invalidated.

validated(): The components that are invalidated by using the above method invalidate() can be validated again using validate().

Frame:

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 dimensions of the border area may be obtained using the getInsets( ) method. Since the border area is included in the overall size of the frame, the border effectively obscures a portion of the frame, constraining the area available for rendering and/or displaying subcomponents to the rectangle. A Frame, implemented as an instance of the_JFrame class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with a GUI usually include at least one frame. Applets sometimes use Frames. Simply, a Frame is a window that has nice borders, various buttons along the top border and other features. Java uses class Frame is a to level window with border and title. It uses BorderLayout as default layout manager.

Frame() constructor is used to create Frame window.

Frame Class Constructors are,

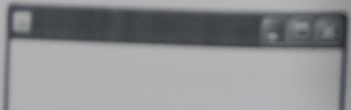
Frame() - Constructs new instance of Frame.

Frame(String title) - Constructs the initial Frame with specified title.

Following program demonstrate the concept of Frame in Java.

/** Program of AWT where we are inheriting Frame class and showing Button component on the Frame*/

```
import java.awt.*;
class First extends Frame
{
First( )
{
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
}
class FrameDemo
{
public static void main(String args[])
{
First f=new First();
}
}
```



```
D:\jdk1.8.0\bin>javac FrameDemo.java
D:\jdk1.8.0\bin>java FrameDemo
```

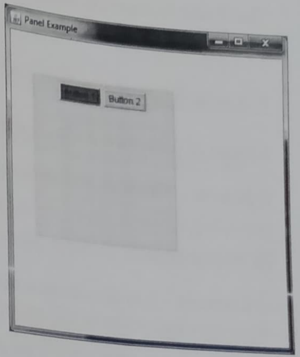
Like Frame, we can also develop Java application for Panel. Following program demonstrate the concept of AWT Panel.

```
/** Program for AWT Panel */

import java.awt.*;

public class PanelExample

{

    PanelExample()

     {
      Frame f= new Frame("Panel Example");

      Panel panel=new Panel();

      panel.setBounds(40,80,200,200);

      panel.setBackground(Color.cyan);

      Button b1=new Button("Button 1");

      b1.setBounds(50,100,80,30);

      b1.setBackground(Color.red);

      Button b2=new Button("Button 2");

      b2.setBounds(100,100,80,30);

      b2.setBackground(Color.green);

      panel.add(b1);

      panel.add(b2);

      f.add(panel);

      f.setSize(400,400);

      f.setLayout(null);

      f.setVisible(true);
```

```
      }

      public static void main(String args[ ])

      {

      new PanelExample();

      }

}
```

***** OUTPUT *****

```
D:\Java\jdk1.8.0_121\bin>javac PanelExample.java

D:\Java\jdk1.8.0_121\bin>java PanelExample
```



Event Delegation Model:

In the event delegation model, a class designated as an event source generates an event and sends it to one or more listeners. This means that a particular event is processed only by a specific listener.

The general diagrammatic representation is shown below;

A user wants the push button to perform some action, when he clicks button.
Clicking a button is called event

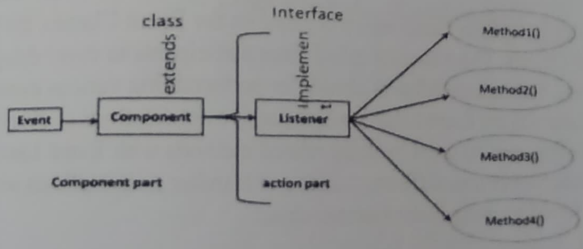


Figure: Event Delegation Model

The Event Delegation Model has divided into two main parts;

1.Component Part

2. Action Part

Instead of these two it also contains following key participants namely:

i) **Source** - The source is an object on which event occurs. Source is responsible fo
providing information of the occurred event to it's handler. Java provide as with cl
for source object.

ii) **Listener** - It is also known as event handler. Listener is responsible for generati
response to an event. From java implementation point of view the listener is also a
object. Listener waits until it receives an event. Once the event is received , the liste
process the event an then returns.

The event model is based on the Event Source and Event Listeners. Event
Listener is an object that receives the messages or events. The Event Source is any
object which creates the message or event.

We can also say that, Event Delegation model is based on the Event Classes, the
Event Listeners & Event Objects. These are also called three participants in event
delegation model in Java where, **Event Source** is the class which broadcasts the
events, **Event Listeners** is the classes which receive notifications of events &
Event Object is the class object which describes the event.

Event Source & Handler:

As we seen in Event Delegation model which is based on the Event Classes, the E
Listeners & Event Objects. These are also called three participants in event delega
model. where, Event Source uses different classes for performining various events
While the Event Listeners are Event handler which receive notifications of events a
performs or handle that specific event by using related methods with Event Listene
object Simply, Event Source uses different classes and Handler uses methods whic
are accessed or refereed by using Event Listener object.

ent Categories:

re are some of the most common types of events in Java:

ionEvent: Represents a graphical element is clicked, such as a button or item in a
st. Related listener: *ActionListener*.

ContainerEvent: Represents an event that occurs to the GUI's container itself, for
ample, if a user adds or removes an object from the interface. Related
stener: *ContainerListener*.

KeyEvent: Represents an event in which the user presses, types or releases a
ey. Related listener: *KeyListener*.

WindowEvent: Represents an event relating to a window, for example, when a window
s closed, activated or deactivated. Related listener: *WindowListener*.

MouseEvent: Represents any event related to a mouse, such as when a mouse is
clicked or pressed. Related listener: *MouseListener*.

Event Listeners:

The Event listener represent the interfaces responsible to handle events.Java provides
s various Event listener classes. Every method of an event listener method has a
single argument as an object which is subclass of Event Object class.

For example, mouse event listener methods will accept instance of Mouse Event,
where Mouse Event derives from Event Object.Every listener interface has to extend.
This class is defined in java.util package. Following is the declaration for
java.util.EventListener interface:

public interface EventListener

AWT Event Listener Interfaces:

Following is the list of commonly used event listeners.

Control	**Description**
ActionListener	This interface is used for receiving the action events.
ComponentListener	This interface is used for receiving the component even
ItemListener	This interface is used for receiving the item events.
KeyListener	This interface is used for receiving the key events.
MouseListener	This interface is used for receiving the mouse events.
TextListener	This interface is used for receiving the text events.
WindowListener	This interface is used for receiving the window events.
AdjustmentListener	This interface is used for receiving the adjustment even
ContainerListener	This interface is used for receiving the container events
MouseMotionListener	This interface is used for receiving the mouse motion e