Chapter 5 Event, Event handling & AWT/Swing : (20)

5.1 Basics of Event handling, AWT Event hierarchy
5.2 Semantics & low level Event handling in AWT Event handling
5.3 Individual Events, Seperating GUI & Application codes
5.4 Multicasting, Advance Event handling tech
5.5 An Introduction of Layout Management, input choice, scroll bar.
5.6 Complex Layout management, Menus, Dialog Box.

5.1 Basics of event handling

↳ Event handling is the concept to trap different action and behaviour performed by user, such as clicking a button, moving mouse, closing window.
• Through the help of event handling, we are able to trap those actions and based on it, different actions can be performed.

AWT Event Hierarchy

Java AWT (Abstract window Toolkit) is an API to develop GUI or window based applications in Java. Java AWT itself is platform independent but AWT component is platform dependent as it uses the resources of native O.S.

⇒ That means component are displayed according to the view of operating system. AWT is considered heavyweight coz, it uses the resources of O.S.

Some of the key features of Java AWT

(a) Platform independent
↳ AWT allows developers to built application that runs on multiple platforms without any modification.

(b) Event handling : AWT provides event classes and their interface for handling users event like mouse click, drag, hover, keyboard input.

(c) Layout manager : Layout Manager is a part of AWT that helps to position the item, element with containers

(d) Graphics drawing

AWT Event hierarchy is:

Semantic and low level event

① Semantic Event
↳ Action event
Adjustment Event
Item event
Text event
These classes are used for high level semantic events, to represent user interactions with a GUI components. Eg: clicking a button, selecting a menu item, selecting a checkbox, etc are semantic events.

② Low level event
↳ Component event
Container event
Focus event
key event
Mouse event
Paint event
Low level event corresponds to more fundamental interactions, such as mouse movement, cursor pointer, hover, etc. They are the building block for semantic event.

③ Window event
↳ These classes are used to represent low level input or window operations.

③ Window Event : Window event are the subset of low level event that deals with window related operation.
For eg: click a button located at particular position is the combination of several low level event like : mouse movement, positioning the cursor & clicking the button.

④ Multicasting Event
↳ Multicasting Event are quite handy & useful when we have to work with many windows in our applications and we just want to perform the same action or group of actions for all the windows at the same time.

For eg: if i have opened multiple windows of chrome browser and i wanna close all of them at one click, then there comes the use of multicasting event.

5.4 Layout Management

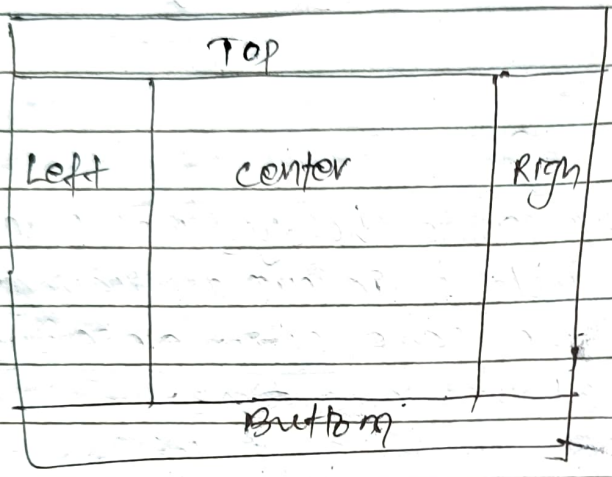
↳ Layout Determines the size and position of component within a container. So the proper management of layout of any GUI application is known to be Layout Management.

Layout management is done by Layout Manager which is an interface which is implemented by all the classes of layout managers.

There are multiple types of Layout:

(a) Border Layout : (23)

⇒ Border Layout places components in five different positions ; i.e. Top, button, Left, right, center. Border Layout is the default Layout of frame. All the extra space which is obtained from top, button, Left, right is covered by center position.

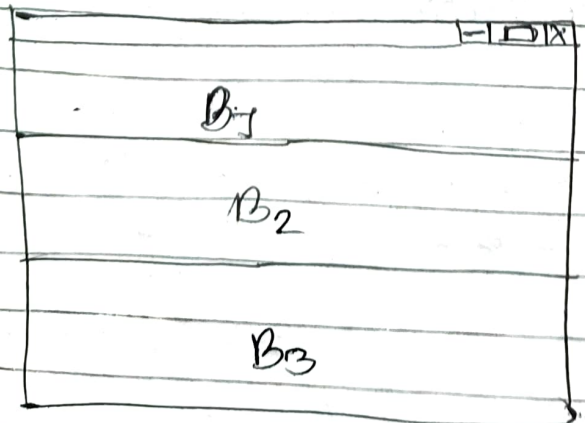


(b) Box Layout

↳ Box Layout places the component either Row wise or column wise. For this purpose, Box Layout provides four different constants as:

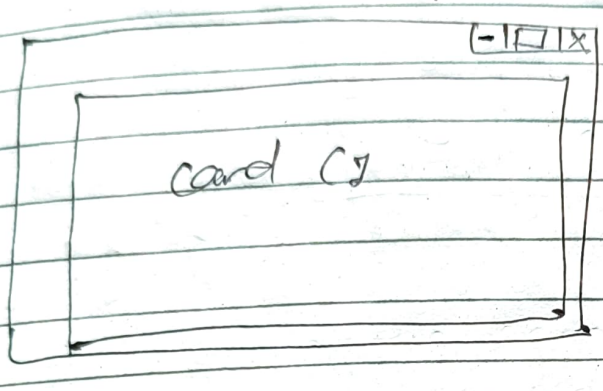
X-AXIS, Y-AXIS, LINE-AXIS, PAGE-AXIS

. BoxLayout class is found in javax.swing package



© CardLayout

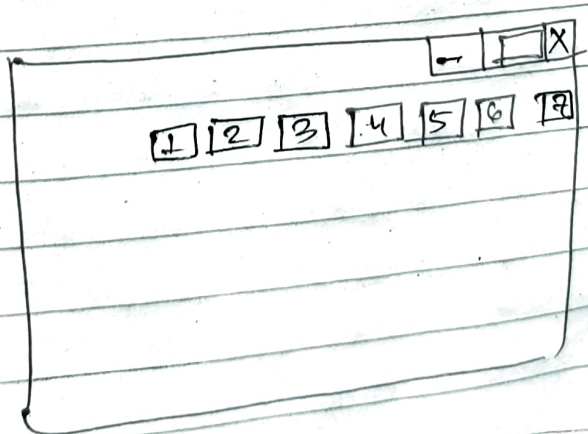
↳ CardLayout manages the component in such a way that only one component is visible at a time, out of many other components. It treates every component as a card. So It is known to be card Layout.



ⓓ Flow Layout : Flow Layout is the default Layout for every JPanel and it places the component in a row, maintaining a flow and starts new row if the space is not wide enough to fit the components. It is also the default Layout of panel.

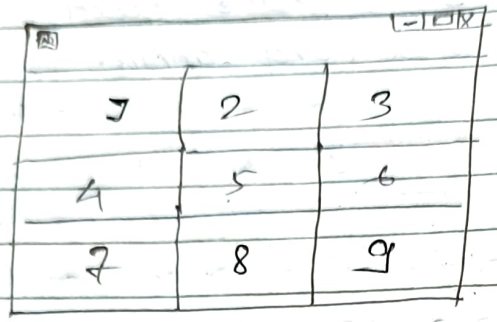
⇒ Flow Layout also provide 5 constants:
ⓐ Left, ⓑ Right ⓒ Center ⓓ Leading, ⓔ Trailing

ⓔ ~~GridBagLayout~~



⑥ Grid Layout :

↳ Grid Layout helps to make the component equal in size and displays them according to the required number of rows & columns.



⑦ Grid Bag Layout

↳ Grid Bag Layout is the sophisticated, and advance Layout which is an updated version of Grid Layout. It places components like in the grid, and also allows component to expand more than one cell.

⑧ Group Layout :

⑨ Spring Layout

Q Bank chapter 5

28 fall 36 How to close AWT window in Java. Explain all methods with example.

↳ There are multiple ways of closing AWT window in Java. They are described below:

(a) By using a window Listener to the frame

↳ Here we can close the AWT window using window Listener interface of which there will be window Adaptor which further contains windowClosing() methods to handle the operations.

Frame
- setSize(w, h)
- setLayout
- setVisibility
- setDefaultCloseOperation()

addWindowListener

```
Eg: import java.awt.*;
    import java.awt.event.WindowEvent;
    import java.awt.event.WindowAdapter;

public class FrameDemo {
    public FrameDemo()
    {
        Frame frame = new Frame("Demo frame closing");
        frame.setSize(400, 200); // width & height
        frame.setVisible(true);

// adding windowListener
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing
    (WindowEvent e)
    {
        frame.dispose();
    }
});
```

```
}
public static void main(String[] args)
{
    FrameDemo fd= new FrameDemo1();
}
}
```

(5) closing a dialog

↳ Again we can close the dialog, which is a smaller sub window inside of the window frame using dispose method as:

```
import java.awt.*;
import java.awt.event.*;
class
public^ DialogClosing{
public DialogClosing()
{
Frame frame= new Frame("Dialog closing");
frame.setSize(400,200);
frame.setVisible(true);

Dialog dialog= new Dialog(frame,"Demo dialog
true);
Button button= new Button("closedialog");
dialog.add(button);
dialog.pack(); // provide size.
button.addActionListener(new ActionerListener()
{
public void actionPerformed(ActionEvent e)
{
```

chalog
}
});

(c) us
the

22 fall

ya
be sh
"_

⇒ Import
Import

public

```
dialog.dispose();
}
});
```

(c) using System.exit(0) close the entire apps. It terminates the JVM and exit the program.

22 fall 4a create a popup menu attached to a frame. If you click within the area of frame, popup menu will be shown. Similarly clicked on menu item shows message "___ items is selected" on label.

⇒
```
import java.awt.*;
import java.awt.event.*;

public class FrameWithPopupMenu {
    public FrameWithPopupMenu()
    {
        Frame frame = new Frame("DemoFrame");
        frame.setSize(400, 600);
        frame.setVisible(true);

        Dialog PopupDialog = new Dialog(frame, "Demo Dialog");
        PopupDialog.setSize(200, 100);
        PopupDialog.setVisible(false);

        Button button = new Button("Show dialog");
        button.setSize(50, 40);
```

(3M)

```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
            PopupDialog.setVisible(true);
        }
    });

Button mbutton = new Button("Menu Item");
mbutton.setSize(50, 20);
PopupDialog.add(mbutton);
mbutton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            Label label = new Label("4 items are selected");
            label.setSize(100, 50);
            PopupDialog.add(label);
        }
    });

public static void main(String[] args)
{
    new FramewithPopupMenu();
}
}
```

2nd spring 24

How event is handled in java? Write a program to draw pie chart using AWT

⇒ Event handling is the concept of mapping different action, behaviour performed by user and taking some steps like performing any kind of operation, computation.

Java uses the delegation event model to handle the event. This model defines the standard mechanism to generate & handle the event. The main working concept of DEM is quite simple:

Step 1: A src generate an event and send it to one or more listeners. until & unless event is received, listener simply waits.

Step 2: once an event is received, listener processes the event, & then does further operation. and then returns.

The advantages of DEM is that it seperates the application logic that process the event from the UI logic.

// code to draw piechart using AWT.