Unit 1 : GUI Programming

AWT and Swing :

- AWT and Swing are used to develop window-based applications in Java.
- Awt is an abstract window toolkit that provides various component classes like Label, Button, TextField, etc., to show window components on the screen. All these classes are part of the Java.awt package.

- On the other hand, Swing is the part of JFC (Java Foundation Classes) built on the top of AWT and written entirely in Java. The javax.swing API provides all the component classes like JButton, JTextField, JCheckbox, JMenu, etc.

AWT:

- AWT stands for Abstract Window Toolkit.
- It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java.
- It was developed by heavily Sun Microsystems In 1995.
- It is heavy-weight in use because it is generated by the system's host operating system.
- It contains a large number of classes and methods, which are used for creating and managing GUI.

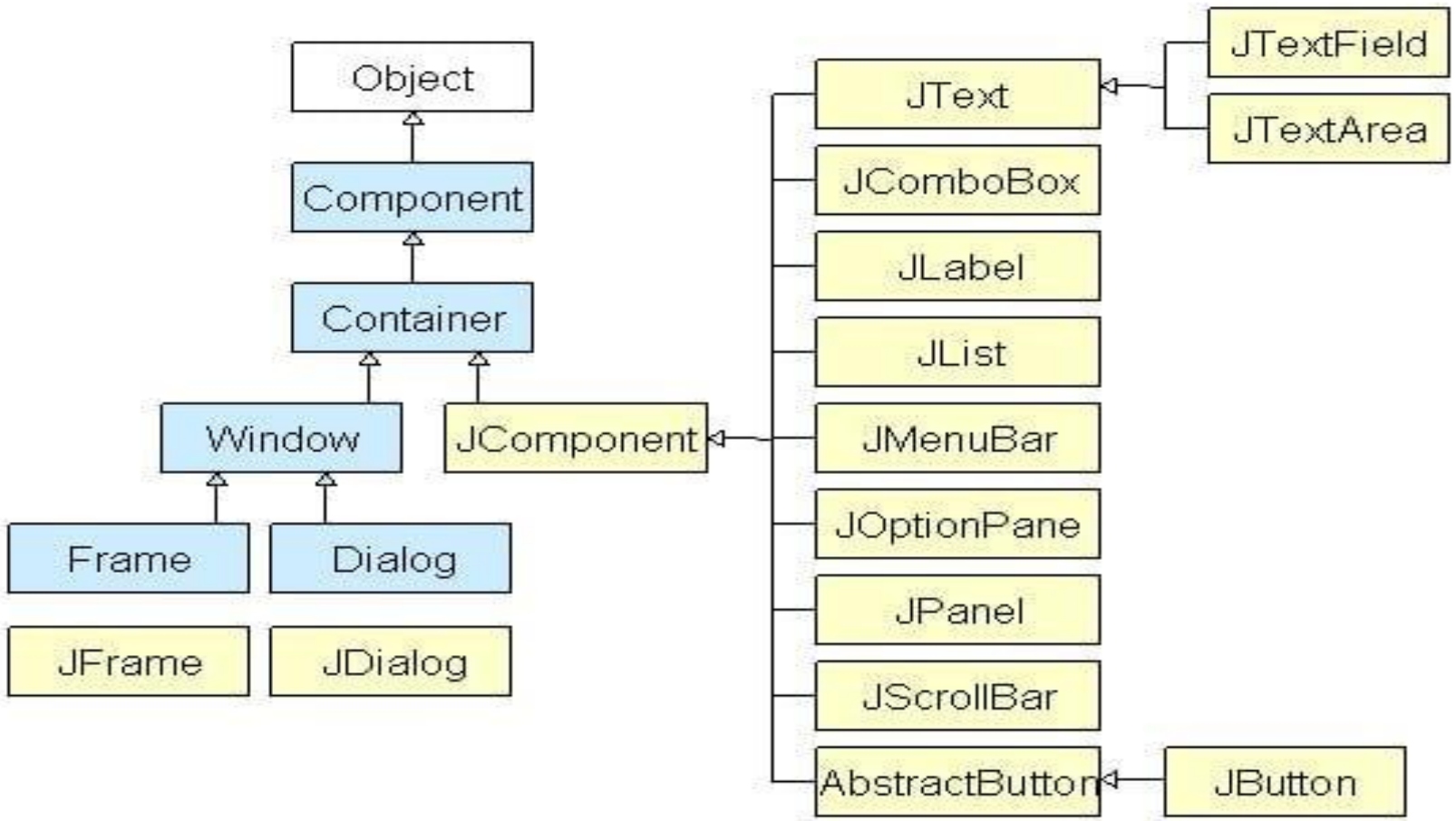
Swing:

- Swing is a lightweight Java graphical user interface (GUI) that is used to create various applications.
- Swing has platform-independent components.
- It enables the user to create buttons and scroll bars.
- Swing includes packages for creating desktop applications in Java.
- Swing components are written in Java language.
- It is a part of Java Foundation Classes(JFC).

Difference between AWT and Swing :

AWT	Swing
Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
AWT provides comparatively less powerful components.	Swing provides more powerful components.
AWT components require java.awt package	Swing components requires javax.swing package
AWT is a thin layer of code on top of the operating system.	Swing is much larger swing also has very much richer functionality.
AWT stands for Abstract windows toolkit .	Swing is also called as JFC(java Foundation classes). It is part of oracle's JFC.
Using AWT , you have to implement a lot of things yourself .	Swing has them built in.

Java Swing Class Hierarchy Diagram :



Swing Components/Control and Containers :

- A component is an independent visual control and Java Swing Framework contains a large set of these components which provide rich functionalities and allow high level of customization. They all are derived from JComponent class.
- All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

- A container holds a group of components. It provides a space where a component can be managed and displayed.

Containers are of two types:

1.Top level Containers

- It inherits Component and Container of AWT.
- It cannot be contained within other containers.
- Heavyweight.
- Example: JFrame, JDialog, JApplet

2.Lightweight Containers

1. It inherits JComponent class.
2. It is a general purpose container.
3. It can be used to organize related components together.
4. Example: JPanel

1. JFrame :

- The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class.
- JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.
- It can be created in two ways:

1. **By creating object of the frame classs**

   Eg. JFrame F = new JFrame();

2. **By extending the frame class**

```
class ClassName extends JFrame{
}
```


Constructors:

- **JFrame() :**

  It constructs a new frame that is initially invisible.
- **JFrame(GraphicsConfiguration gc) :**

  It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
- **JFrame(String title) :**

  It creates a new, initially invisible Frame with the specified title.

- **JFrame(String title, GraphicsConfiguration gc) :**

  It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Methods of the JFrame class :

1. setVisible():

- This method is used to control whether a component will be displayed on the screen or not.
- It takes a Boolean value as an argument.

2. setDefaultCloseOperation()

- This method specifies many options for the close Button. We can choose any one of the given constants to determine the close Button's nature when someone clicks on it.

- Constants are given below :

i. **JFrame.DO_NOTHING_ON_CLOSE –** "As the name suggests, it neglects the click, and nothing happens when someone clicks on the Close Button. "

ii. **JFrame.HIDE_ON_CLOSE –**" As the name suggests, it only hides the frame when someone clicks on the Close Button, but the application still remains on the running state. "

iii. **JFrame.DISPOSE_ON_CLOSE –**"As the name suggests, it disposes the frame when someone clicks on the Close Button, but the application still remains on the running state. "

iv. **JFrame.EXIT_ON_CLOSE –**"As the name suggests, it exits the application when someone clicks on the Close Button and removes the program from memory permanently. "

3. setSize():

- As the name suggests this method is used to set the size of the window displayed on the screen .
- By default the size is passed to the **setSize()** method is ” 0 pixel x 0 pixel “.
- it takes two integer type arguments. first argument specifies the width of the window and second argument specifies the height of the window .
- example – **setSize(100,200) .**

4. setLocation()

- This method is used to set the window’s location on the screen.
- The default location of the window is the top-left corner of the screen.
- It takes two integer type arguments. The first argument specifies the window’s location on the x-axis, and the second argument specifies the window’s location on the y-axis.

5. setBounds()

- This method is used to set the size and position(Location) of the window simultaneously.
- It takes four integer type arguments in which the first two arguments specify the window’s position on the x-axis and y-axis, and the last two arguments specify the size of the window in width and height.
- example –**setBounds(200,200,100,200)**.

6. setTitle()

- This method is used to set the title of the window.
- It takes the string/name of the window to be displayed on the screen as its argument.

7. setBackground()

Steps to Add Color :

- Import the **java.awt.*** package. This package makes the **setBackground()** method available.
- Since the Background Color can only be added to the container, we have to make a container object for the window jframe.
- This is done by calling the **getContentPane()** method through jframe.
- Now we have to call the **setBackground()** method using the container's object, which takes various static pre-defined fields of Color class.

8. setResizable()

- This method takes a boolean value as its parameter.

i. **setResizable(true) –** "allows the user to resize the window. "
ii. **setResizable(false)-** "restricts the user to resize the window. "

2. JLable :

- ❖ JLable is a component which displays a readable text or image in the swing container UI.
- ❖ The application user cannot edit the text, but text can be changed by application.
- ❖ The JLabel Contains 4 constructors. They are as following:

1. JLabel()
2. JLabel(String s)
3. JLabel(Icon i)
4. JLabel(String s, Icon i, int horizontalAlignment)

3. JTextField

- ❖ JTextField object is a text component that allows for the editing of a single line of text.
- ❖ The JTextField Contains 4 constructors. They are as following:

1. **JTextField**() : Creates a new TextField
2. **JTextField**(int cols) : Creates a new empty TextField with the specified number of columns.
3. **JTextField**(String str, int cols) : Creates a new TextField initialized with the specified text and columns.
4. **JTextField**(String str) : Creates a new TextField initialized with the specified text.

4. JPasswordField

- ❖ JPasswordField object is a text component specialized for password entry.
- ❖ Following are the constructors:

1. JPasswordField()
2. JPasswordField(int columns)
3. JPasswordField(String text)
4. JPasswordField(String text, int columns)

5. JTextArea :

- JTextArea object is a text component that allows editing of a multiple lines of text.
- Following are the constructors:

1. JTextArea()
2. JTextArea(String s)
3. JTextArea(int row, int column)
4. JTextArea(String s, int row, int column)

6. JButton :

- JButton class provides functionality of a button. It is used to create button component.
- Following are the constructors:

1. JButton() : It creates a button with no text and icon.
2. JButton(String s) : It creates a button with the specified text.
3. JButton(Icon i) : It creates a button with the specified icon object.

7. JCheckBox :

- The JCheckBox class is used to create a checkbox.
- It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on "
- Following are the constructors:

1. JCheckBox() : Creates an initially unselected check box button with no text, no icon.
2. JChechBox(String s) : Creates an initially unselected check box with text.
3. JCheckBox(String text, boolean selected) : Creates a check box with text and specifies whether or not it is initially selected.

8. JRadioButton :

- The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options.
- It should be added in ButtonGroup to select one radio button only.
- Following are the constructors:

1. JRadioButton()
2. JRadioButton(String s)
3. JRadioButton(String s, boolean selected)

9. JComboBox :

- JComboBox class is used to show popup menu of choices.
- Choice selected by user is shown on the top of a menu.

- Following are the constructors:

1. JComboBox() : Creates a JComboBox with a default data model.
2. JComboBox(Object[] items) : Creates a JComboBox that contains the elements in the specified array.

10. JTable:

- The JTable class is used to display data in tabular form.
- It is composed of rows and columns.
- Following are the constructors:

1. JTable() : Creates a table with empty cells.
2. JTable(Object[][] rows, Object[] columns): Creates a table with the specified data.

10. JList:

- The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items.

- Following are the constructors:

1. JList() : Creates a JList with an empty, read-only, model.
2. JList(ary[] listData): Creates a JList that displays the elements in the specified array.
3. JList(ListModel<ary> dataModel) : Creates a JList that displays elements from the specified, non-null, model.

11. JOptionPane

- The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.
- These dialog boxes are used to display information or get input from the user.
- Following are the constructors:

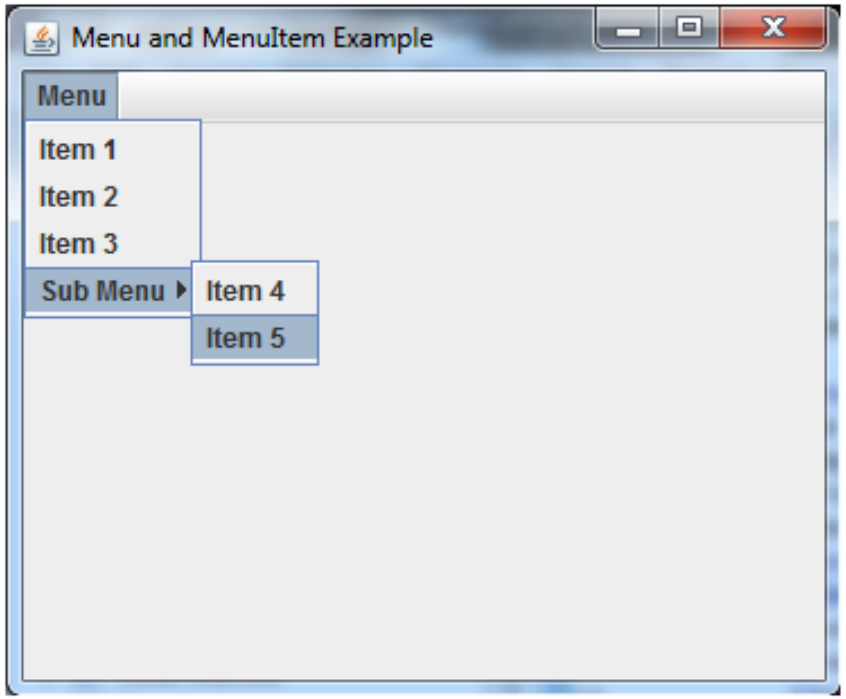
1. JOptionPane() **:** It is used to create a JOptionPane with a test message.
2. JOptionPane(Object message) : It is used to create an instance of JOptionPane to display a message.

12. JMenuBar, JMenu, JMenuItem:

- JMenuBar class used to display menubar on the window or frame.
- Jmenu beasically represents menu. It contains several JMenuIteam object.

Following are the constructors:

- JMenuBar()
- JMenu(String name)
- JMenu()



Event Handling:

Event:

- Change in the state of an object is known as **Event**, i.e., event describes the change in the state of the source.
- Events are generated as a result of user interaction with the graphical user interface components.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

Event Handling :

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism has a code which is known as an event handler, that is executed when an event occurs.
- Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

1. **Source** − The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide us with classes for the source object.

2. **Listener** − It is also known as event handler. The listener is responsible for generating a response to an event. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

Java Event classes and listener interfaces :

- Event classes represent the event.
- Event listeners represent the interfaces responsible to handle events.

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
WindowEvent	WindowListener

ActionListener :

- The Java ActionListener is notified whenever you click on the button or menu item.
- It is notified against ActionEvent.
- The ActionListener interface is found in java.awt.event package.
- It has only one method: actionPerformed().

- The signature of method found in MouseListener interface are given below:
- **public abstract void** actionPerformed(ActionEvent e);

MouseListener :

- The Java MouseListener is notified whenever you change the state of mouse.
- It is notified against MouseEvent.
- The MouseListener interface is found in java.awt.event package.
- It has five methods.
- The signature of 5 methods found in MouseListener interface are given below:

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

KeyListener :

- The **Java KeyListener is notified whenever you change the state of key.**
- It is notified against KeyEvent.
- The KeyListener interface is found in java.awt.event package.
- it has three methods.
- The signature of 3 methods found in KeyListener interface are given below:

1. **public abstract void** keyPressed (KeyEvent e);
2. **public abstract void** keyReleased (KeyEvent e);
3. **public abstract void** keyTyped (KeyEvent e);

Layout Management

- A layout manager determines the location and size of components placed into a container.
- Different layout manager classes use different algorithms for determining size and location.
- Even if you do not use the layout manager, the components are still positioned by the default layout manager.
- LayoutManager is an interface that is implemented by all the classes of layout managers.

There are the following classes that represent the layout managers:

1. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout
5. GridBagLayout
6. SpringLayout
7. GroupLayout
8. BoxLayout

BorderLayout

- The BorderLayout is used to arrange the components in **five regions**: **north, south, east, west, and center.** Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

There are two useful constructors.

a. **new BorderLayout()**
b. **new BorderLayout(int hgap, int vgap)**

When components are added to a container that uses a BorderLayout, they should be added with the following kind of statement.

container.add(component, *whr*);

The parameter *whr* should be one of the following.

BorderLayout.NORTH
BorderLayout.EAST
BorderLayout.SOUTH
BorderLayout.WEST
BorderLayout.CENTER

FlowLayout

- The FlowLayout arranges the components in a directional flow, either from left to right or from right to left. Normally all components are set to one row, according to the order of different components. If all components can not be fit into one row, it will start a new row and fit the rest in.

Constructors :

1. FlowLayout(): construct a new FlowLayout object with center alignment and horizontal and vertical gap to be default size of 5 pixels.
2. FlowLayout(int align): construct similar object with different settings on alignment.
3. FlowLayout(int align, int hgap, int vgap): construct similar object with different settings on alignment and gaps between components.

(For the constructor with the alignment settings, the possible values could be: LEFT, RIGHT, CENTER, LEADING and TRAILING.)

GridLayout

- The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors :

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

CardLayout

- The **Java CardLayout** class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors :

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

1. **public void next(Container parent):** is used to flip to the next card of the given container.
2. **public void previous(Container parent):** is used to flip to the previous card of the given container.
3. **public void first(Container parent):** is used to flip to the first card of the given container.
4. **public void last(Container parent):** is used to flip to the last card of the given container.
5. **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

GridBagLayout

- **GridBagLayout** is a more flexible layout manager, which allows the components to be vertical, horizontal, without specifying the components to be the same size.
- GridBagLayout extends the capabilities of the GridLayout.
- GridBagLayout also allows the component to span to multiple columns or rows.
- The **GridBagConstraints** decides where the component to be displayed and how the component should be positioned.

Constructors : GridBagLayout()

Here are properties of ***GridBagConstraints*** and their descriptions:

1. gridx,gridy :
   Specify the row and column from top to bottom and from left to right starting from zero. For example gridx = 0, gridy = 0 is the top left cell of the grid.

2. gridwidth, gridheight :
   Specify number of rows (gridheight) and columns (gridwidth) to which a component can span. The default value of *gridwidth* and *gridheight* is 1.
3. fill :

This property is used to resolve whether and how to resize the component when the component's display region is larger than the component's requested size. The values for fill property are: **NONE**, **VERTICAL**, **HORIZONTAL** and **BOTH**

GridBagLayout

4. ipadx, ipady :

The ipadx, ipady property are used to set the internal padding of the component.

5. weightx, weighty

These properties are used to specify how to distribute space between rows(weighty) and columns(weightx).

SpringLayout

- A **SpringLayout** *arranges the children of its associated container according to a set of constraints*. Constraints are nothing **but horizontal and vertical distance between two-component edges**. Every constraint is represented by a SpringLayout.Constraint object.

- **SpringLayout** can be visualized as a set of objects that are connected by a set of springs on their edges.

Constructors : SpringLayout()

SpringLayout Fields :

SpringLayout.NORTH : specifies the top edge of a component's bounding rectangle.
SpringLayout.SOUTH : specifies the bottom edge of a component's bounding rectangle.
SpringLayout.EAST : specifies the right edge of a component's bounding rectangle.
SpringLayout.WEST : specifies the left edge of a component's bounding rectangle.
SpringLayout.BASELINE : specifies the baseline of a component.
SpringLayout.HORIZONTAL_CENTER : specifies the horizontal center of a component's bounding rectangle.
SpringLayout.VERTICAL_CENTER : specifies the vertical center of a component's bounding rectangle.

GroupLayout

- **GroupLayout** groups its components and place them hierarchically in a container. The grouping is done by an instance of the Group class.
- Group is an abstract class and two concrete classes that implement this class are **SequentialGroup** and **ParallelGroup**.
- **SequentialGroup** positions its elements sequentially one after the other while **ParallelGroup** aligns its elements on top of each other.
- The **GroupLayout** class provides methods such as **createParallelGroup**() and **createSequentialGroup**() to create groups.
- GroupLayout treats each axis independently. That is, there is a group representing the horizontal axis, and a group representing the vertical axis. Each component must exist in both a horizontal and vertical group.

Constructors : GroupLayout(Container host)

BoxLayout

- The BoxLayout class is used to arrange the components either vertically (along Y-axis) or horizontally (along X-axis). In BoxLayout class, the components are put either in a single row or a single column.

Constructors :

BoxLayout(Container c, int axis): Creates a BoxLayout class that arranges the components with the X-axis or Y-axis.

Swing MVC Desing Pattern

- Swing uses the *model-view-controller architecture* (MVC) as the fundamental design behind each of its components. Essentially, MVC breaks GUI components into three elements. Each of these elements plays a crucial role in how the component behaves.

Model

The model encompasses the state data for each component. There are different models for different types of components. For example, the model of a scrollbar component might contain information about the current position of its adjustable "thumb," its minimum and maximum values, and the thumb's width (relative to the range of values). A menu, on the other hand, may simply contain a list of the menu items the user can select from. Note that this information remains the same no matter how the component is painted on the screen; model data always exists independent of the component's visual representation.

View

The view refers to how you see the component on the screen. For a good example of how views can differ, look at an application window on two different GUI platforms. Almost all window frames will have a titlebar spanning the top of the window. However, the titlebar may have a close box on the left side (like the older MacOS platform), or it may have the close box on the right side (as in the Windows 95 platform). These are examples of different types of views for the same window object.

Controller Swing MVC Desing Pattern

The controller is the portion of the user interface that dictates how the component interacts with events. Events come in many forms — a mouse click, gaining or losing focus, a keyboard event that triggers a specific menu command, or even a directive to repaint part of the screen.

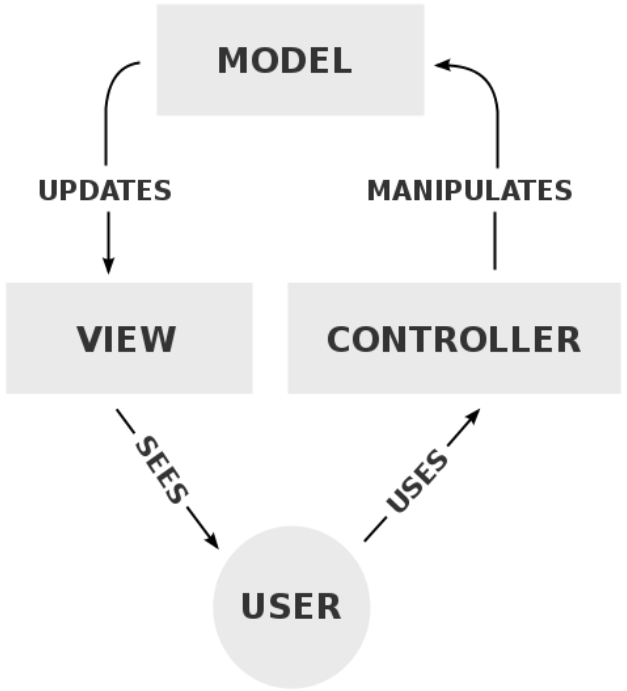


Fig: Internal representation of interaction between model, view and controller.

The Model receives commands and data from the Controller. It stores these data and updates the View. The View lets to present data provided by the Model to the user.

The Controller accepts inputs from the user and converts it to commands for the Model or the View.

Java Swing Font :

- An object of the java.awt.Font class represents a font in a Java program.
- To install a font to a component, use the setFont(Font f) method of the component.
- Java defines five logical font family names and maps them to physical font family names for different system.

The five logical font family names are as follows:

1. Serif
2. SansSerif
3. Dialog
4. DialogInput
5. Monospace

E.g.

```
Font f1 = new Font(Font.SERIF, Font.PLAIN, 10);
```

Create a custom Font object:

```
Font font = new Font("Arial", Font.BOLD, 24);
```

Display Image with Swing GUI :

ImageIcon :

- ImageIcon is a class in Java Swing that is used to represent an image icon or image in an application.
- It is a type of Icon that can be used with Swing components, such as JLabel, JButton, and JMenuItem, to display an image.

Java Swing Colors:

- An object of the java.awt.Color class represents a color.
- We can create a Color object using its RGB (Red, Green, and Blue) components.
- RGB values can be specified as float or int values. As a float value, each component in RGB ranges from 0.0 to 1.0. As an int value, each component in RGB ranges from 0 to 255.
- **Eg:**

Color red = new Color(255, 0, 0);

- A Color object is used with the setBackground(Color c) and setForeground(Color c) methods of the swing components.

- The background color is the color with which a component is painted, where as the foreground color is usually the color of the text displayed in the component.

- If a component is transparent, it does not paint pixels in its bounds. In order to see the background color we must make the component opaque true.

Colors	R	G	B
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
White	255	255	255
Gray	128	128	128
Black	0	0	0

Working with 2D Shapes :

- paintComponent is a method in the JComponent class (which is a superclass of many Swing components such as JPanel) that is called automatically by Swing to paint the component on the screen. When a component needs to be displayed, Swing invokes paintComponent to render the component's appearance on the screen.

- Graphics2D is a subclass of the Graphics class in Java that provides more advanced 2D rendering capabilities than the basic Graphics class.

- It allows you to draw lines, shapes, and images with various effects, such as gradients, alpha transparency, and antialiasing**.**