UNIT 5

GUI PROGRAMMING WITH SWING AND EVENT HADLING

MVC Architecture:

The Model-View-Controller (MVC) framework is an architectural/design pattern that separates an application into three main logical components Model, View, and Controller. Each architectural component is built to handle specific development aspects of an application. It isolates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs). Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. It is also used for designing mobile apps.

MVC was created by Trygve Reenskaug. The main goal of this design pattern was to solve the problem of users controlling a large and complex data set by splitting a large application into specific sections that all have their own purpose.

Features of MVC

It provides a clear separation of business logic, UI logic, and input logic.

It offers full control over your HTML and URLs which makes it easy to design web application architecture.

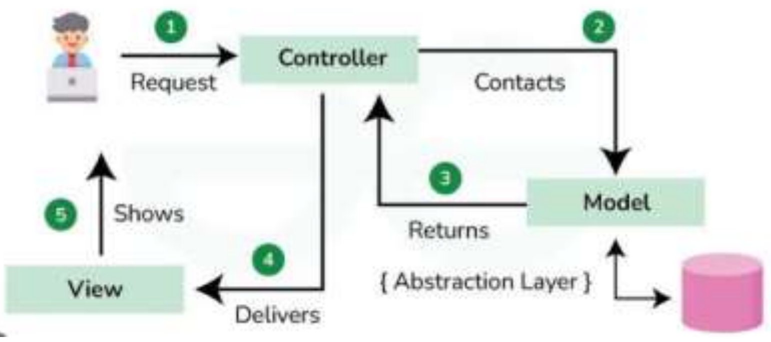
It is a powerful URL-mapping component using which we can build applications that have comprehensible and searchable URLs.

It supports Test Driven Development (TDD).

Components of MVC

The MVC framework includes the following 3 components:

- Controller
- Model
- View



Controller:

The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary. The controller doesn't have to worry about handling data logic, it just tells the model what to do. It processes all the business logic and incoming requests, manipulates data using the Model component, and interact with the View to render the final output.

View:

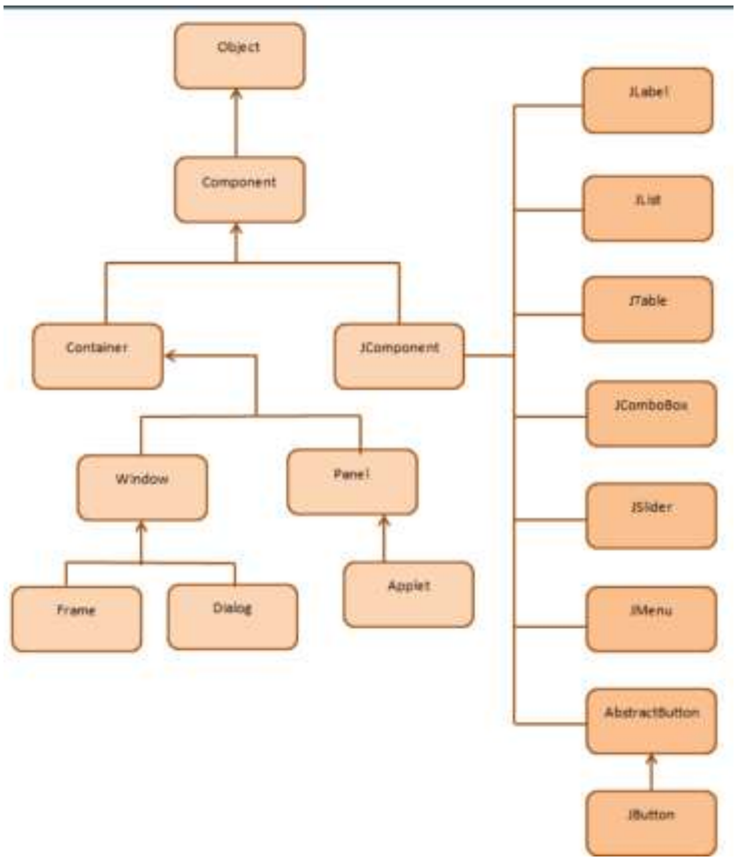
The View component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren't taken directly but through the controller. It only interacts with the controller.

Model:

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. It can add or retrieve data from the database. It responds to the controller's request because the controller can't interact with the database by itself. The model interacts with the database and gives the required data back to the controller.

Component and Container in Java

In the world of Java programming, graphical user interfaces (GUIs) play a crucial role in providing a user-friendly and interactive experience. GUI components are the building blocks of these interfaces, allowing developers to design and create sophisticated applications. Among these components, two fundamental concepts stand out: "Component" and "Container."



Components:

Components, in Java, refer to the basic elements that form the user interface. These elements are responsible for rendering specific functionalities or features on the screen. Common examples of components include buttons, labels, text fields, checkboxes, radio buttons, sliders, and more. Each component represents a specific user interface control with specific behaviors and attributes. Components can be interactive or non-interactive, depending on their nature.

- **JFrame**: JFrame is a top-level container that represents the main window of a GUI application. It provides a title bar, and minimizes, maximizes, and closes buttons.
- **JPanel**: JPanel is a container that can hold other components. It is commonly used to group related components together.
- **JButton**: JButton is a component that represents a clickable button. It is commonly used to trigger actions in a GUI application.
- **JLabel**: JLabel is a component that displays text or an image. It is commonly used to provide information or to label other components.
- **JTextField**: JTextField is a component that allows the user to input text. It is commonly used to get input from the user, such as a name or an address.
- **JCheckBox**: JCheckBox is a component that represents a checkbox. It is commonly used to get a binary input from the user, such as whether or not to enable a feature. JList: JList is a component that represents a list of elements. It is typically used to display a list of options from which the user can select one or more items. JTable: JTable is a component that represents a data table. It is typically used to present data in a tabular fashion, such as a list of products or a list of orders.

- **JScrollPane**: JScrollPane is a component that provides scrolling functionality to other components. It is commonly used to add scrolling to a panel or a table.

Containers

Containers, on the other hand, are components that serve as a holding space for other components. In essence, containers are responsible for organizing and managing the layout of their child components. They provide structure and help create complex UI designs by defining how components are positioned and displayed within them.

- Panel : JPanel is the simplest container. It provides space in which any other component can be placed, including other panels.
- Frame : A JFrame is a top-level window with a title and a border
- Window : A JWindow object is a top-level window with no borders and no menubar.

Containers are of two types:

Top-level containers

These containers are at the top of the containment hierarchy and hold other Swing components. The four top-level containers in Swing are:

- JFrame
- JApplet
- JWindow
- JDialog

Lightweight containers

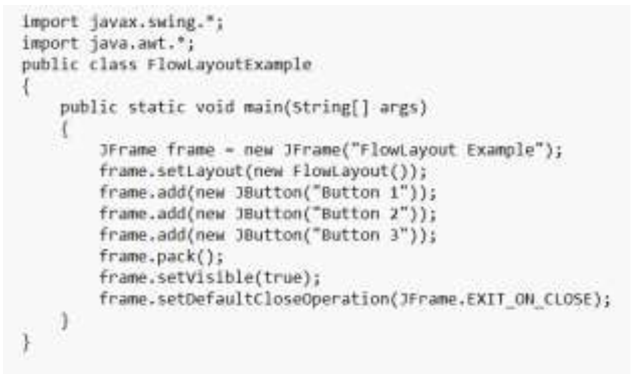
These containers are inherited from Jcomponent and include components like JButton, JPanel, and JMenu.

Layout Managers

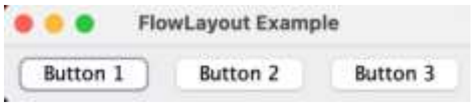
In Java, graphical user interfaces (GUIs) play a vital role in creating interactive applications. To design a visually appealing and organized interface, the choice of layout manager becomes crucial. Layout managers define how components are arranged within a container, such as a JFrame or JPanel. Java provides several layout managers to suit various design needs.

FlowLayout

FlowLayout is a simple layout manager that arranges components in a row, left to right, wrapping to the next line as needed. It is ideal for scenarios where components need to maintain their natural sizes and maintain a flow-like structure.



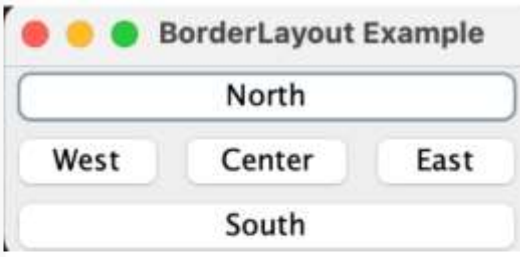
```java
import javax.swing.*;
import java.awt.*;
public class FlowLayoutExample
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("FlowLayout Example");
        frame.setLayout(new FlowLayout());
        frame.add(new JButton("Button 1"));
        frame.add(new JButton("Button 2"));
        frame.add(new JButton("Button 3"));
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



BorderLayout

BorderLayout divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER. Components can be added to these regions, and they will occupy the available space accordingly. This layout manager is suitable for creating interfaces with distinct sections, such as a title bar, content area, and status bar.

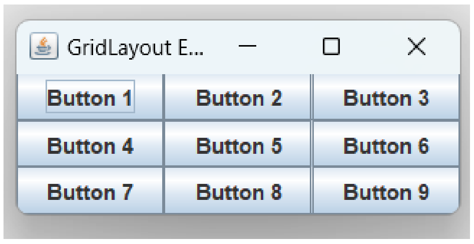
```
import javax.swing.*;
import java.awt.*;
public class BorderLayoutExample
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("BorderLayout Example");
        frame.setLayout(new BorderLayout());
        frame.add(new JButton("North"), BorderLayout.NORTH);
        frame.add(new JButton("South"), BorderLayout.SOUTH);
        frame.add(new JButton("East"), BorderLayout.EAST);
        frame.add(new JButton("West"), BorderLayout.WEST);
        frame.add(new JButton("Center"), BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



GridLayout

GridLayout arranges components in a grid with a specified number of rows and columns. Each cell in the grid can hold a component. This layout manager is ideal for creating a uniform grid of components, such as a calculator or a game board.

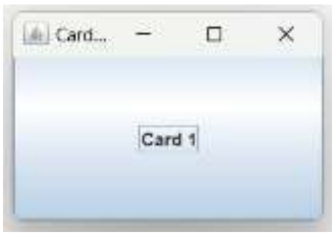
```
import javax.swing.*;
import java.awt.*;
public class GridLayoutExample
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("GridLayout Example");
        frame.setLayout(new GridLayout(3, 3));
        for (int i = 1; i <= 9; i++) {
            frame.add(new JButton("Button " + i));
        }
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



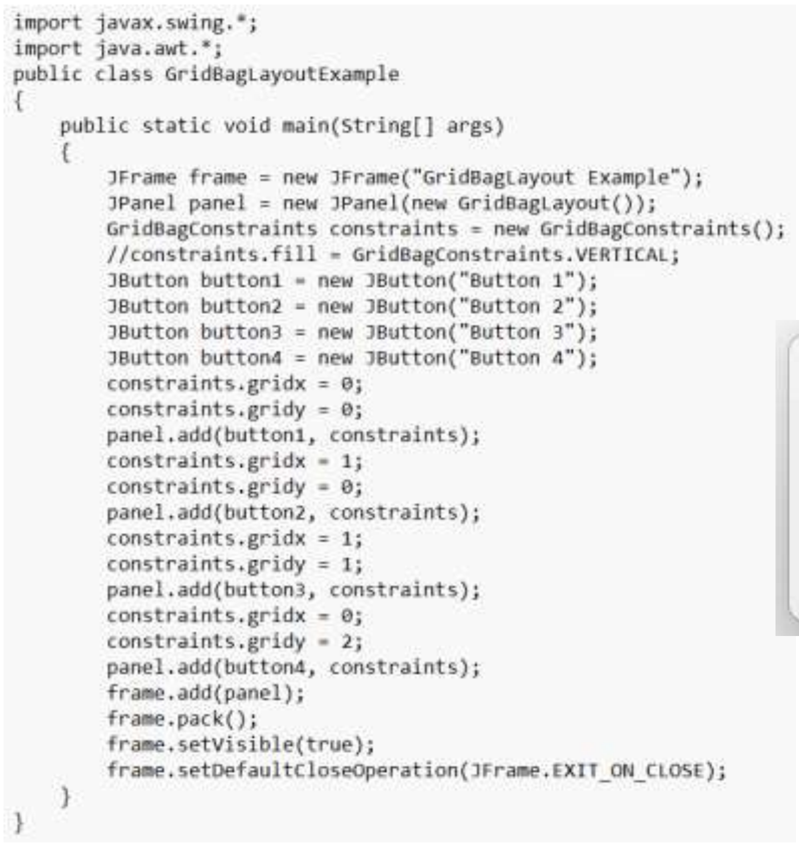
CardLayout

CardLayout allows components to be stacked on top of each other, like a deck of cards. Only one component is visible at a time, and you can switch between components using methods like next() and previous(). This layout is useful for creating wizards or multi-step processes.

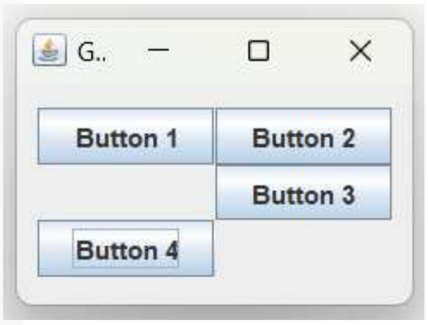
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class CardLayoutExample
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("CardLayout Example");
        CardLayout cardLayout = new CardLayout();
        JPanel cardPanel = new JPanel(cardLayout);
        JButton button1 = new JButton("Card 1");
        JButton button2 = new JButton("Card 2");
        JButton button3 = new JButton("Card 3");
        cardPanel.add(button1, "Card 1");
        cardPanel.add(button2, "Card 2");
        cardPanel.add(button3, "Card 3");
        frame.add(cardPanel);
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        button1.addActionListener(e -> cardLayout.show(cardPanel, "Card 2"));
        button2.addActionListener(e -> cardLayout.show(cardPanel, "Card 3"));
        button3.addActionListener(e -> cardLayout.show(cardPanel, "Card 1"));
    }
}
```



GridBagLayout
GridBagLayout is a powerful layout manager that allows you to create complex layouts by specifying constraints for each component. It arranges components in a grid, but unlike GridLayout, it allows components to span multiple rows and columns and have varying sizes.



```
import javax.swing.*;
import java.awt.*;
public class GridBagLayoutExample
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("GridBagLayout Example");
        JPanel panel = new JPanel(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();
        //constraints.fill = GridBagConstraints.VERTICAL;
        JButton button1 = new JButton("Button 1");
        JButton button2 = new JButton("Button 2");
        JButton button3 = new JButton("Button 3");
        JButton button4 = new JButton("Button 4");
        constraints.gridx = 0;
        constraints.gridy = 0;
        panel.add(button1, constraints);
        constraints.gridx = 1;
        constraints.gridy = 0;
        panel.add(button2, constraints);
        constraints.gridx = 1;
        constraints.gridy = 1;
        panel.add(button3, constraints);
        constraints.gridx = 0;
        constraints.gridy = 2;
        panel.add(button4, constraints);
        frame.add(panel);
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



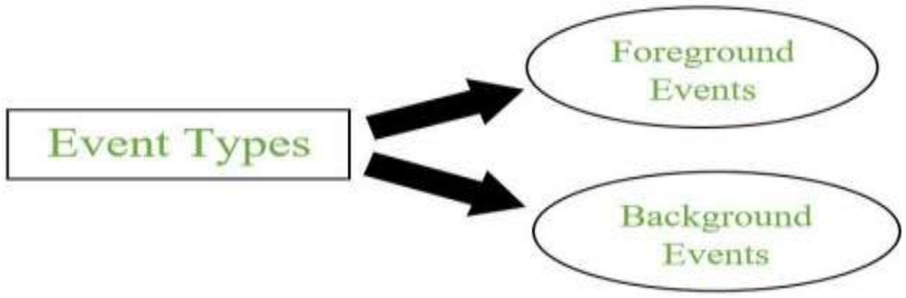
Event Handling in java

Event in Java

- An **event** can be defined as changing the state of an object or behavior by performing actions. Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.
- The java.awt.event package can be used to provide various event classes.

Classification of Events

- Foreground Events
- Background Events



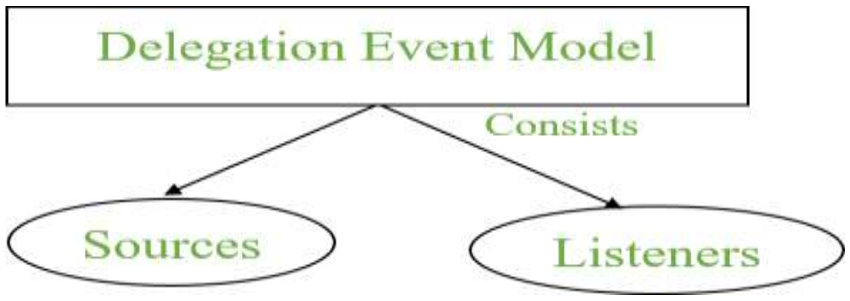
1. Foreground Events
   Foreground events are the events that require user interaction to generate, i.e., foreground events are generated due to interaction by the user on components in Graphic User Interface (**GUI**). Interactions are nothing but clicking on a button, scrolling the scroll bar, cursor moments, etc.
2. Background Events
   Events that don't require interactions of users to generate are known as backgroundevents. Examples of these events are operating system failures/interrupts, operationcompletion, etc.

Event Handling

It is a mechanism to **control the events** and to **decide what should happen afteran event** occur. To handle the events, Java follows the ***Delegation Event model.***

Delegation Event model

- It has Sources and Listeners.



- **Source:** Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components,windows, etc., to generate events.
- **Listeners:** Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible forhandling events.

Event Class

The class that represent events are event class. At the root of the java event class hierarchy is "Event Object" which is defined in java.util. Thus it is a super class ofall events.

Main Event Classes :-

1. ActionEvent :- Normal action are taken by it. (button press)
2. AdjucentEvent :- Scroll bar.
3. ComponentEvent :- To move all the visible events.
4. ContainerEvent :- For adding or removing things.
5. FocusEvent :- To focus on a particular things. Ex. — Using tab shifting cursor toone button to
another.
6. InputEvent :- For taking any input.
7. ItemEvent :- To select or be select a checkbox or listbox or radiobutton.
8. KeyEvent :- For taking input from key board.
9. MouseEvent :- Mouse click/drag/up/down.
10. WindowEvent :- To open or close a frame or a dialog box.

Handling Mouse Events:

•To handle mouse events, we must implement one of the appropriate interfaces asfollows:

1) MouseListener
2) MouseMotionListener
3) MouseWheelListener

Methods available in “MouseListener” interface are:

1) mouseClicked( )
2) mousePressed( )
3) mouseReleased( )
4) mouseEntered( )
5) mouseExited( )

Methods available in “MouseMotionListener” interface are:

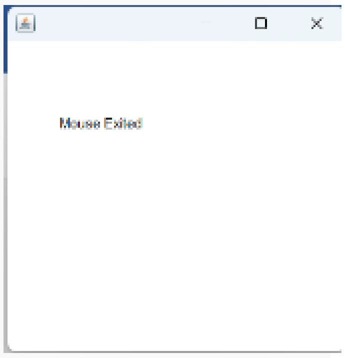
1) mouseMoved( )
2) mouseDragged( )

Methods available in “MouseWheelListener” are:
1) mouseWheelMoved( )

- ✓ MOUSE_CLICKED The user clicked the mouse.
- ✓ MOUSE_DRAGGED The user dragged the mouse.
- ✓ MOUSE_ENTERED The mouse entered a component.
- ✓ MOUSE_EXITED The mouse exited from a component.
- ✓ MOUSE_MOVED The mouse moved.
- ✓ MOUSE_PRESSED The mouse was ressed.
- ✓ MOUSE_RELEASED The mouse was released.
- ✓ MOUSE_WHEEL The mouse wheel was moved

Example:

```java
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
    Label l;
    MouseListenerExample()
    {
        addMouseListener(this);
        l=new Label();
        l.setBounds(50,50,100,100);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e)
    {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e)
    {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e)
    {
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e)
    {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e)
    {
        l.setText("Mouse Released");
    }
    public static void main(String[] args)
    {
        MouseListenerExample m=new MouseListenerExample();
    }
}
```



```
Mouse Exited
```

Handling Keyboard Events:

•To handle keyboard events, we must implement the “KeyListener” interface.

•When the “KeyListener” interface is implemented, we must provide implementations for three methods available in that interface. They are:

1) keyPressed( )

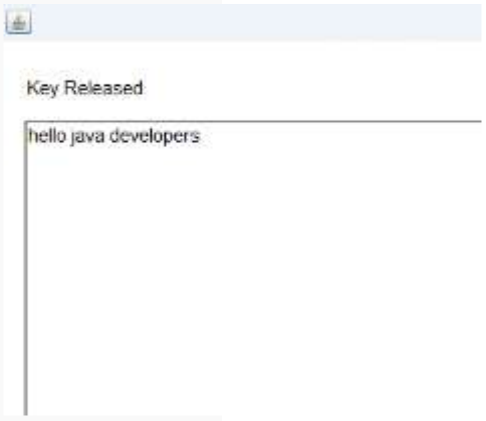
2) keyReleased( )

3) keyTyped( )

•To get the character, when the keyTyped( ) event occurs, we use the method:char getKeyChar( )

Example:

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener
{
    Label l;
    TextArea area;
    KeyListenerExample()
    {
        l = new Label();
        l.setBounds (20, 50, 100, 20);
        area = new TextArea();
        area.setBounds (20, 80, 3000, 300);
        area.addKeyListener(this);
        add(l);
        add(area);
        setSize (400, 400);
        setLayout (null);
        setVisible (true);
    }
    public void keyPressed (KeyEvent e)
    {
        l.setText ("Key Pressed");
    }
    public void keyReleased (KeyEvent e)
    {
        l.setText ("Key Released");
    }
    public void keyTyped (KeyEvent e)
    {
        l.setText ("Key Typed");
    }
    public static void main(String[] args)
    {
        new KeyListenerExample();
    }
}
```



```
Key Released
hello java developers
```

Adapter classes

Adapter Classes In Event Handling.

Adapter Classes are commonly used in GUI (Graphical User Interface)programming where there are many different events that need to be handled, such as button clicks, mouse movements, and key presses.

The Java language provides several adapter classes for common events such asthe MouseAdapter, KeyAdapter, and WindowAdapter classes.

MouseAdapter:

The MouseAdapter class is used to handle mouse events. It contains empty methods for each type of mouse event, such as mouseClicked( ),mousePressed( ), and mouseReleased( ).

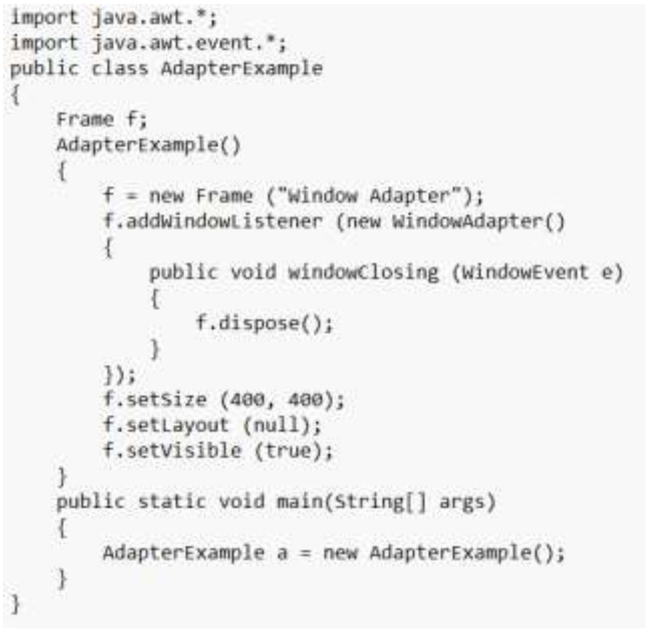
KeyAdapter:

The KeyAdapter class is used to handle key events. It contains empty methods foreach type of key events, such as keyPressed( ), keyReleased( ), and keyTyped( ).

WindowAdapter:

The WindowAdapter class is used to handle window events. It contains empty methods for each type of window event, such as windowOpened( ), windowClosed( ), and windowIconified( ).

programmer only needs to write code for the events that are relevant to the application. This makes the code more readable, easier to maintain, and less proneto errors.



```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample
{
    Frame f;
    AdapterExample()
    {
        f = new Frame ("Window Adapter");
        f.addWindowListener (new WindowAdapter()
        {
            public void windowClosing (WindowEvent e)
            {
                f.dispose();
            }
        });
        f.setSize (400, 400);
        f.setLayout (null);
        f.setVisible (true);
    }
    public static void main(String[] args)
    {
        AdapterExample a = new AdapterExample();
    }
}
```

Explain Inner classes?

Inner classes

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

Syntax of Inner class

```
 class Java_Outer_class
 {
         //code
         class Java_Inner_class
        {
                //code
         }
 }
```

Example:

```
class OuterClass
{
      int x = 10;
      class InnerClass
     {
         int y = 5;
      }
}
public class Main
{
     public static void main(String[ ] args){
      OuterClass myOuter = new OuterClass( ); OuterClass.InnerClass myInner = myOuter.new
      InnerClass( );System.out.println(myInner.y + myOuter.x);}
}
```

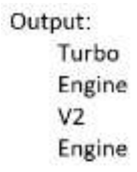
Output

15Explain Anonymous classJava Anonymous ClassAn anonymous class in Java is an **inner class** which is declared without any class name at all. In other words, a nameless inner class in Java is called an anonymous inner class.**Use of Java Anonymous Inner Classes**Anonymous inner classes are used when you want to create a simple class that is needed for one time only for a specific purpose. For example, implementingan interface or extending a class.Anonymous inner class that extends a class

Example:

```
class Car
{
    public void engineType( )
    {
       System.out.println("Turbo Engine");
    }

}
public class Tester
{
        public static void main(String args[ ])
        {
                Car c1 = new Car();
                 c1.engineType();
                 Car c2 = newCar( )
                 {
                    public void engineType( )
                    {
                            System.out.println("V2 Engine");
                    }
                };
                c2.engineType(  );
         }
}
```



```
Output:
    Turbo
    Engine
    V2
    Engine
```