LIVE

Advance Java Seminar

Date – 10th & 11th Nov 2023

Organized By -:

UR ENGINEERING FRIEND

#1 stop Destination for Diploma & Degree

Day 1

1) Abstract Windowing Toolkit

- What is AWT ?
- How to create Frames ?
- Adding controls to frame
- Applets and its controls
- Layout manager
- Menubars & menus
- Dialog Box

2) Swing

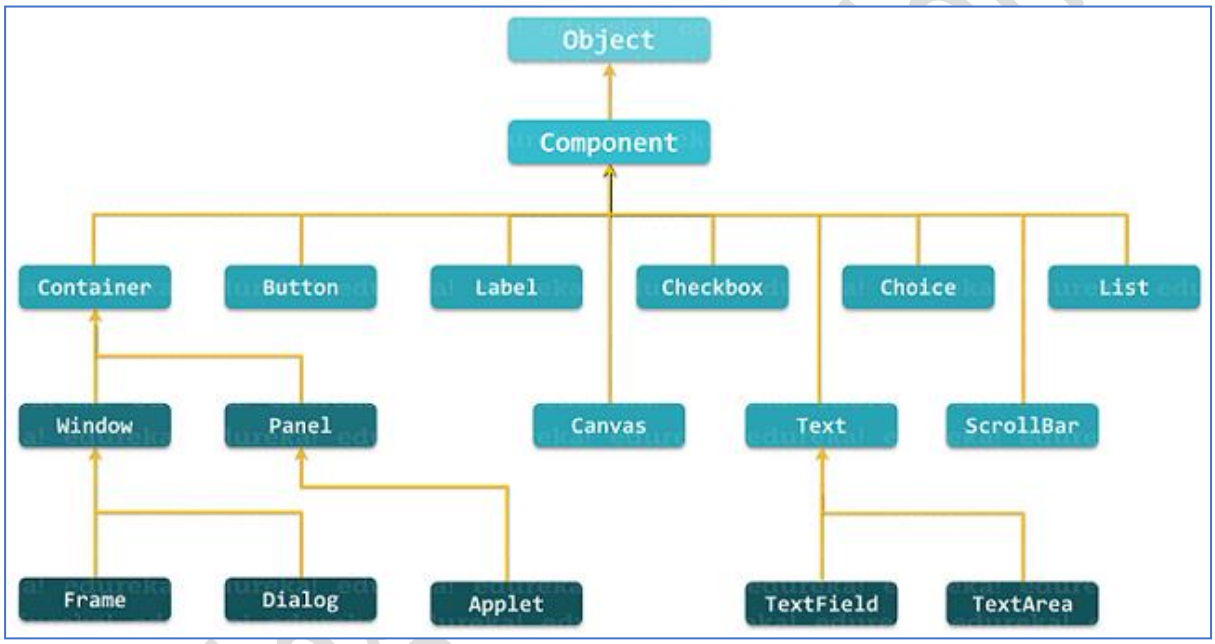
- What is Swing ?
- How to create Frames In swing ?
- Swing Advanced controls
- MVC Architecture

3) Event Handling

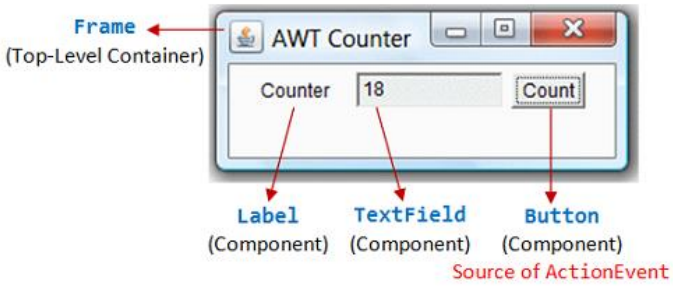
- What is an Event ?
- Event delegation model
- Event sources
- Event Listeners
- Event Classes
- Mouse Event
- Key Events
- Adapter Class

1) Abstract Windowing Toolkit

AWT stands for Abstract Window Toolkit, and it is a set of classes and APIs (Application Programming Interfaces) provided by Java for creating graphical user interfaces (GUIs) in Java applications. AWT was one of the first GUI frameworks in Java and is part of the Java Foundation Classes (JFC).



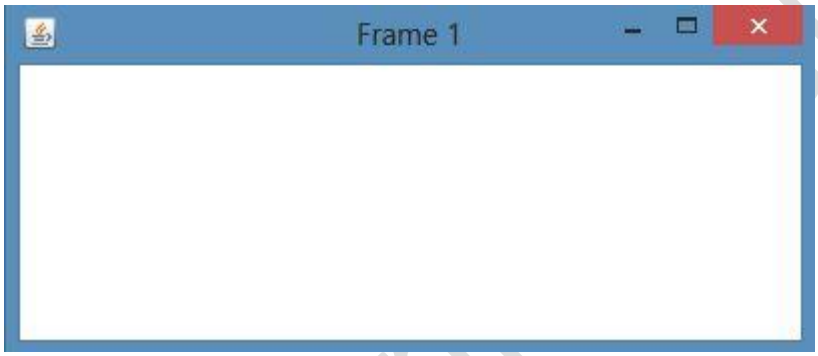
AWT Controls -:



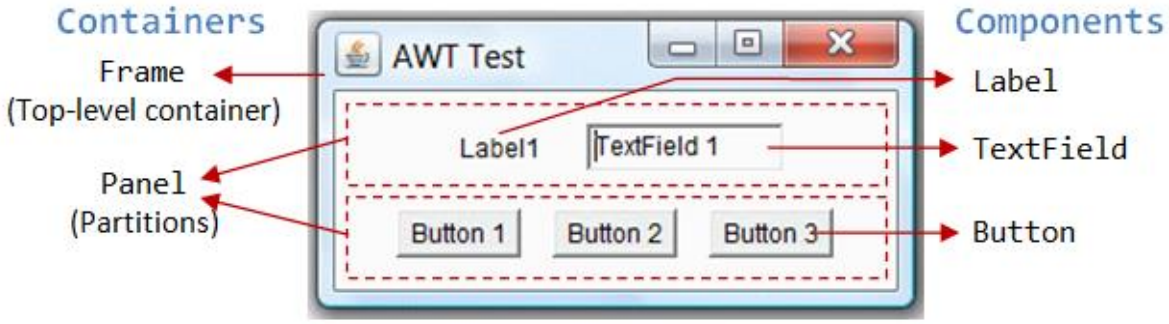
How to Create Container ? (Frame)

By extending the frame class

Using instance of frame class



Adding Controls to Frame



Applets

In Java, an applet is a small Java program that is embedded and run within a web page. Applets were a popular way to add interactive content to web pages in the early days of the internet. They provided a way to create graphical user interfaces (GUIs) and perform various tasks within a web browser.

Key features of Java applets include:

1. Platform Independence: Like other Java programs, applets are platform-independent, meaning they can run on any system that has a Java Virtual Machine (JVM) installed.

2. Embedding in HTML: Applets are typically embedded in HTML (Hypertext Markup Language) pages using the `<applet>` tag. The HTML page acts as a container for the applet.

3. Graphics and User Interface: Applets can create graphical user interfaces using the Abstract Window Toolkit (AWT) or Swing components. They can draw graphics, handle user input, and respond to events.

4. Security Restrictions: Applets run in a restricted environment known as the "sandbox" to prevent potentially harmful actions. They have limited access to the local system to ensure security.

5. Lifecycle Methods: Applets have lifecycle methods such as `init()`, `start()`, `stop()`, and `destroy()`, which are called at different stages of the applet's life. These methods allow developers to initialize resources, start and stop animations, and perform cleanup tasks.

How to create an Applet ?

```java
import java.applet.Applet;
import java.awt.Graphics;

public class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello, this is a simple applet!", 20, 20);
    }
}
```

Adding Controls to Applet

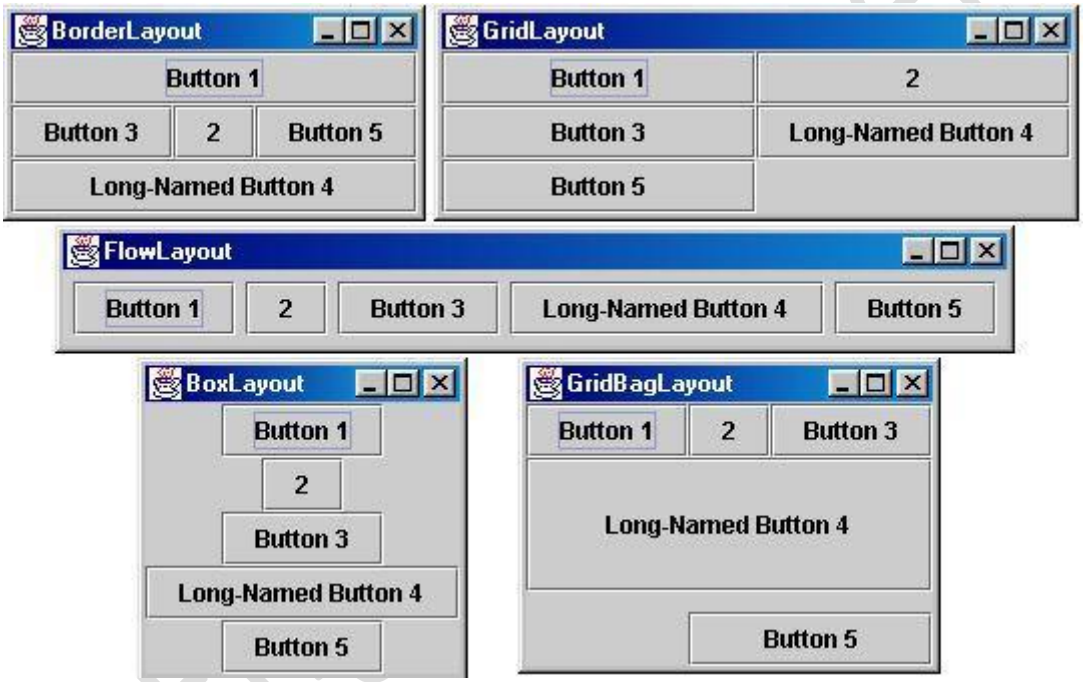
```java
public void init() {
    // Create a Button
    clickButton = new Button("Click me");

    // Add an ActionListener to the Button
    clickButton.addActionListener(this);

    // Add the Button to the applet
    add(clickButton);
}
```

Layout Managers

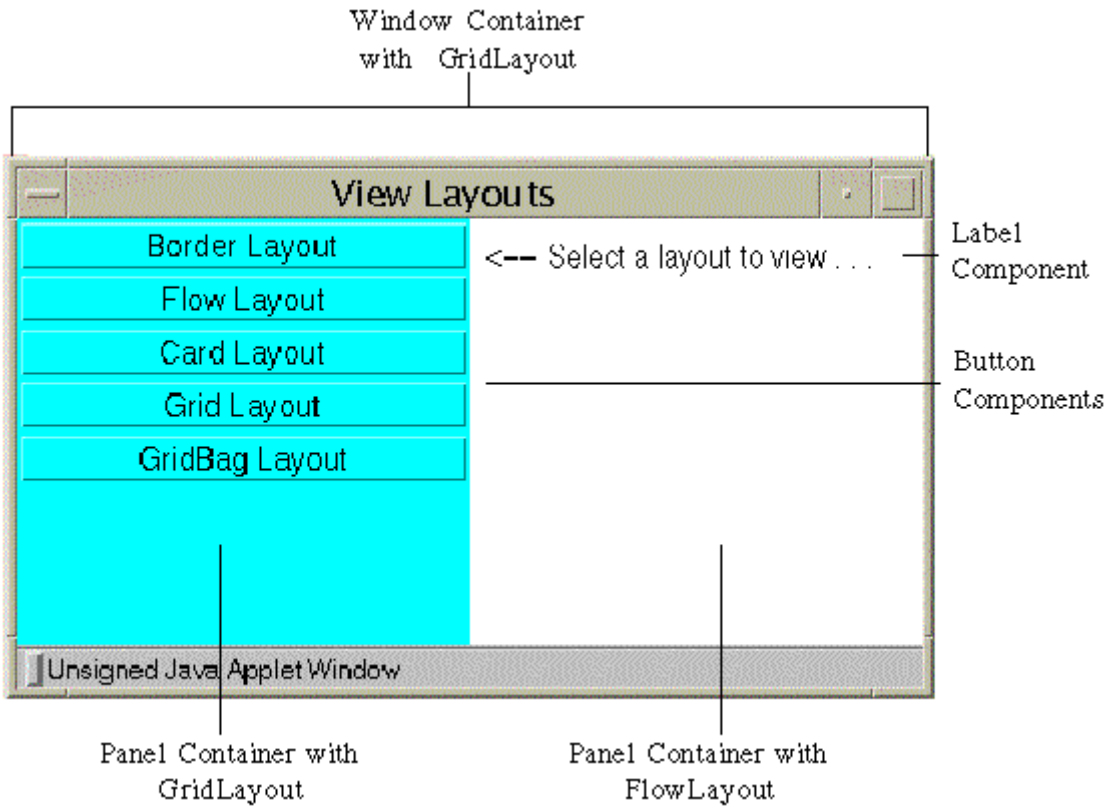
In AWT (Abstract Window Toolkit) in Java, layout managers are objects that control the placement and sizing of components within a container. A container is a component that can hold and organize other components, such as panels, frames, and applets. Layout managers are used to define how components are arranged and resized when the container is resized.



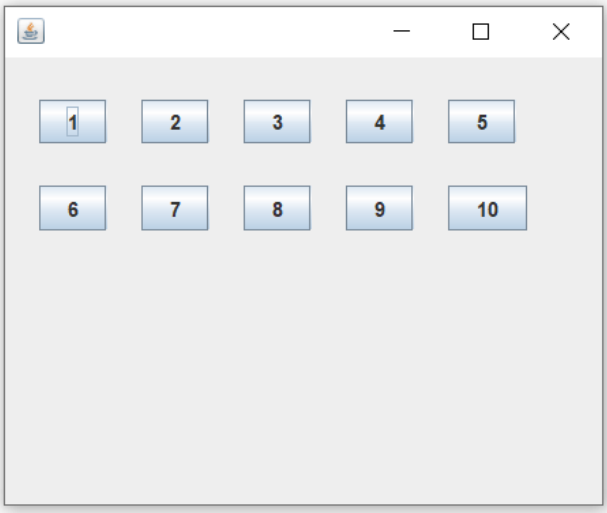
- Layout managers simplify the process of designing user interfaces by handling the details of component placement and resizing. Choosing the appropriate layout manager depends on the desired structure and organization of components within the container.
- Developers can also combine multiple layout managers within nested containers to achieve more complex layouts.

AWT provides several layout managers to assist with the organization of components.

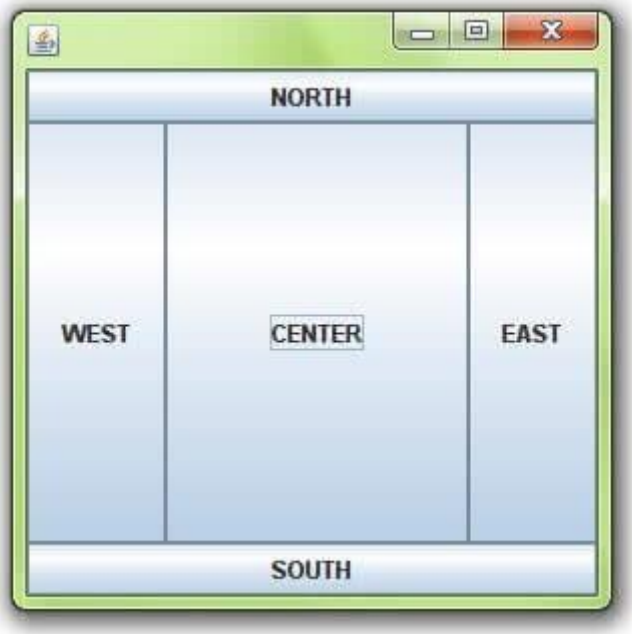
The main AWT layout managers include:



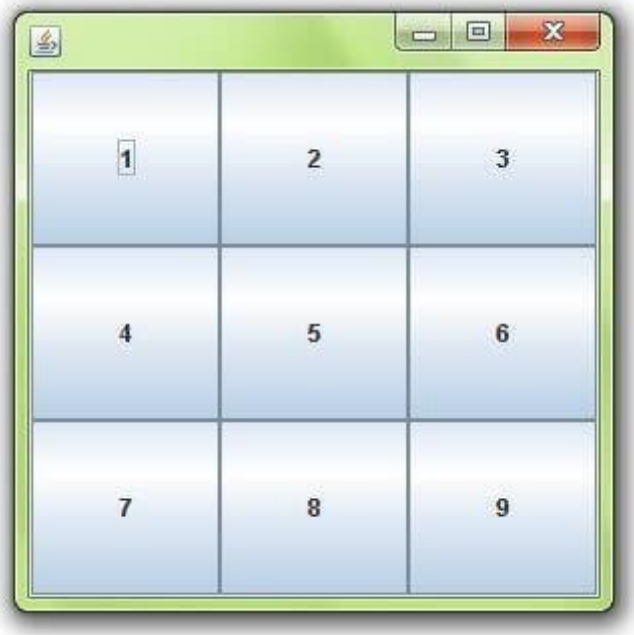
a) Flow Layout



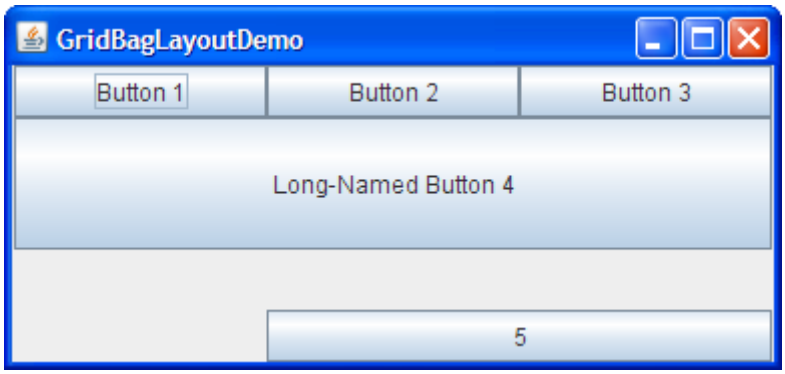
b) Border Layout



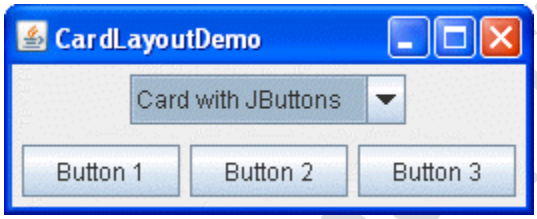
c) Grid Layout



d) Gridbag Layout



e) Card Layout



Menu Bars in AWT

In AWT (Abstract Window Toolkit) in Java, the primary class used to create menu bars is `MenuBar`.

- The `MenuBar` class provides constructors to create instances of menu bars. Additionally, you'll use other classes such as `Menu` and `MenuItem` to construct the actual menus and menu items within the menu bar.
- **Below are the constructors for creating `MenuBar`:**

1. Default Constructor:

- The default constructor creates an empty menu bar.

```
MenuBar menuBar = new MenuBar();
```

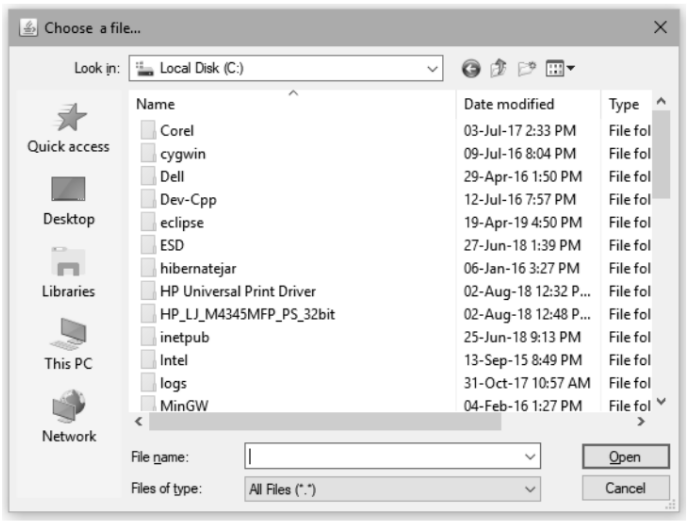
2. None-Default Constructor:

- You can create a `MenuBar` with an initial set of menus using the constructor that takes a `Menu` object as a parameter.

```
MenuBar menuBar = new MenuBar();
Menu fileMenu = new Menu("File");
Menu editMenu = new Menu("Edit");
menuBar.add(fileMenu);
menuBar.add(editMenu);
```

File Dialog Box

Output



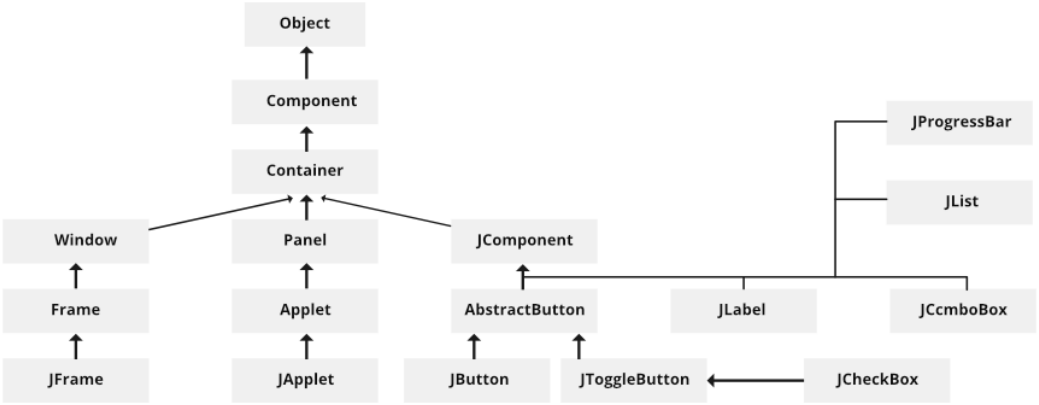
2) Swing

What is Swing ?

Swing is a GUI (Graphical User Interface) toolkit for Java that provides a set of lightweight, platform-independent components for building graphical user interfaces.

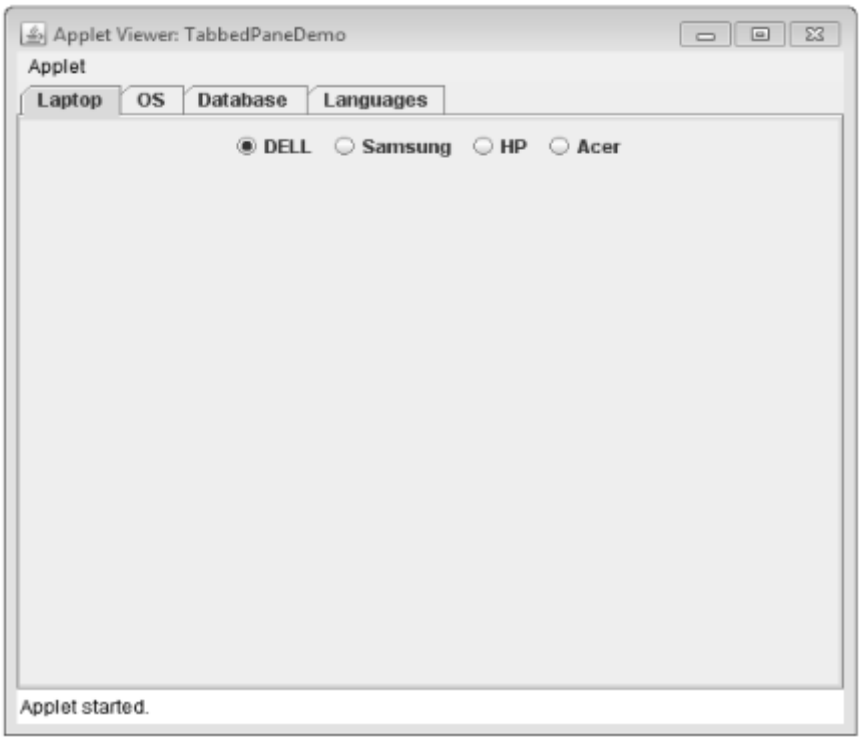
- It is part of the Java Foundation Classes (JFC) and is an extension of the original Abstract Window Toolkit (AWT).
- Unlike AWT, which relies on the native components of the underlying operating system, Swing components are entirely written in Java, making them consistent across different platforms.

Basic controls in Swing

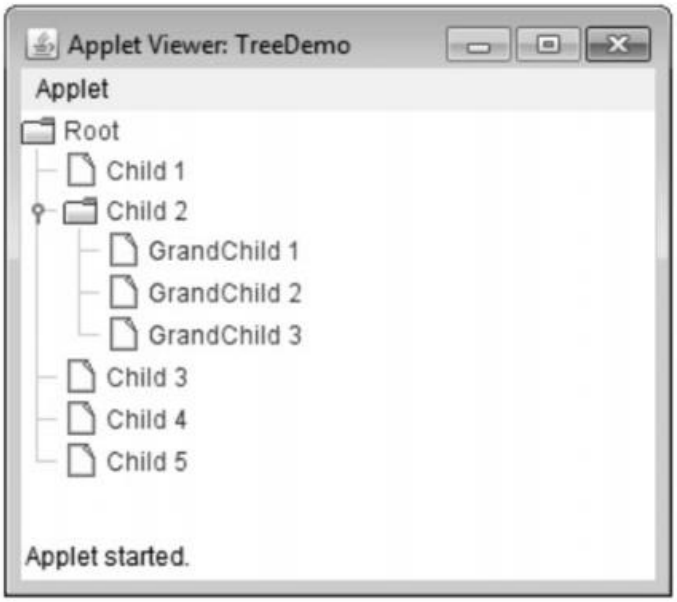


Swing Advanced Components

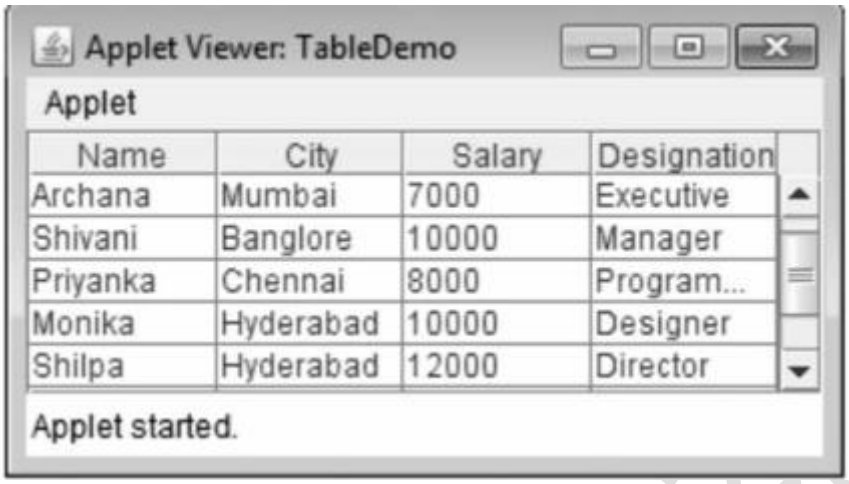
Tabbed Pane



Trees



Table



Progress Bar

MVC Architecture

The Model-View-Controller (MVC) architecture is a design pattern commonly used in software development, including Swing applications in Java. MVC separates an application into three main components: Model, View, and Controller. This separation promotes modularity, code organization, and maintainability.

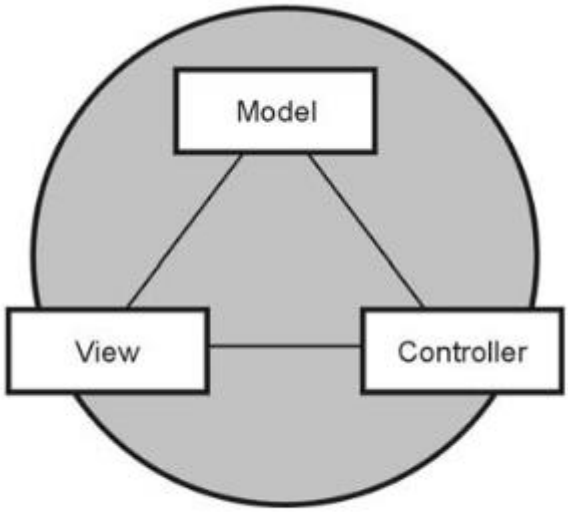


Fig. 2.5.1 MVC design pattern

Here's how MVC is typically applied in Swing:

1. Model:

- The Model represents the application's data and business logic. It is responsible for managing the state of the application and responding to requests for information or updates.
- In a Swing application, the Model is often implemented using plain Java classes or custom classes that encapsulate the application's data and behavior.

- The Model notifies registered observers (typically the View components) about changes in the data through the Observer pattern.

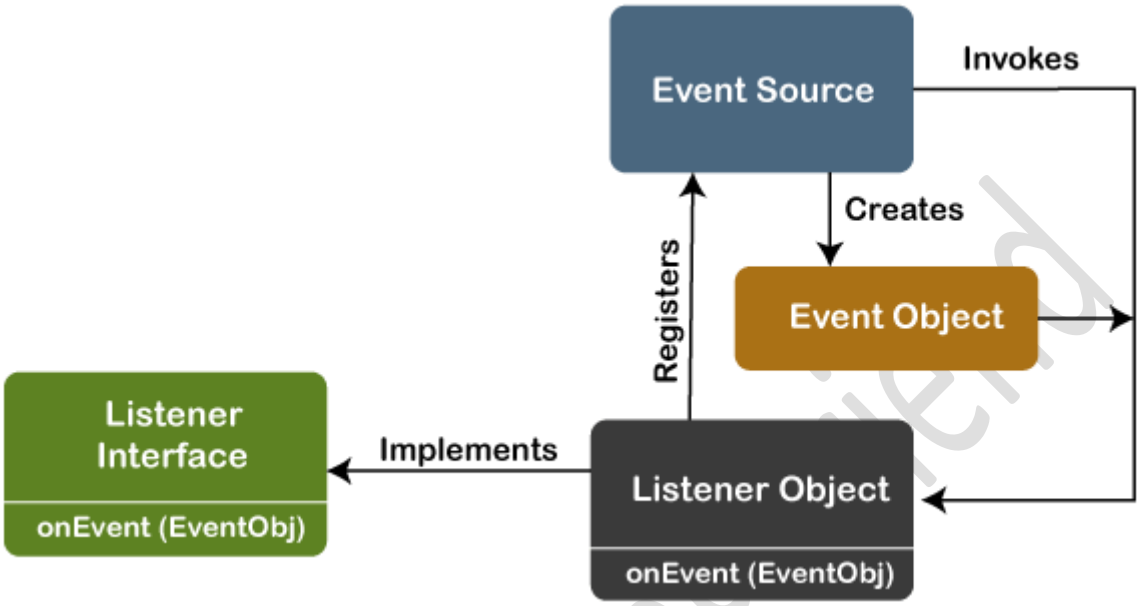
2. View:

- The View represents the presentation and display of the application's user interface. It is responsible for rendering the data to the user and capturing user input.
- In Swing, the View is typically implemented using Swing components such as JFrame, JPanel, JLabel, etc. These components are responsible for displaying the data provided by the Model.
- Views are observers of the Model. They update their display in response to changes in the Model's state.

3. Controller:

- The Controller acts as an intermediary between the Model and the View. It receives user input from the View, processes it, and updates the Model accordingly.
- In Swing, controllers are often implemented as event listeners. They respond to user actions (button clicks, menu selections, etc.) and invoke methods on the Model to update the data.
- The Controller also updates the View based on changes in the Model. It retrieves data from the Model and updates the display components in the View.

3) Event Handling



Event Delegation Model

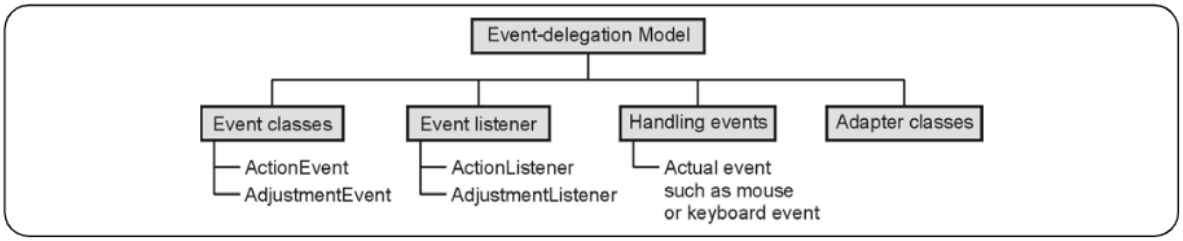


Fig. 3.1.1 Components of event delegation model

Event -:

An event is a notification or signal that something has occurred. It encapsulates information about the occurrence, such as the type of event and any relevant data.

- Examples of events include button clicks, key presses, mouse movements, and so on.

Event Source -:

An object, often a graphical user interface (GUI) component or a part of a software system, that has the capability to generate events.

- Examples of event sources in Java include buttons, text fields, checkboxes, and other user interface components. Non-UI examples could include timers, sensors, or other components that trigger events based on specific conditions.

Event Listener -:

An event listener is an object or a method that is designed to handle and respond to specific types of events. It is registered with an event source, and when the source generates an event, the corresponding event listener is notified.

- In Java, event listeners often implement specific listener interfaces provided by the Java API, such as ActionListener, MouseListener, or KeyListener.

Event Handling -:

Event handling is the process of designing code (event listeners) that responds to events generated by event sources.

- Event handlers are typically defined to perform specific actions or execute specific code when a particular event occurs.

Event Classes -:

In Java, the `java.awt.event` package provides a set of event classes and listener interfaces that are commonly used for handling events in GUI (Graphical User Interface) applications.

Here is a list of various event classes defined in the `java.awt.event` package:

1. ActionEvent:

- Represents an event generated by a user action, such as a button click or a menu item selection.
- Commonly associated with the ActionListener interface.

2. ActionListener:

- An interface for handling ActionEvents. Classes that implement this interface can respond to user actions, such as button clicks.

3. ItemEvent:

- Represents an event generated by an item selection or deselection, for example, in a Checkbox or Choice component.
- Associated with the ItemListener interface.

4. ItemListener:

- An interface for handling ItemEvents. Classes that implement this interface can respond to changes in the state of items, such as checkboxes or choice items.

5. KeyEvent:

- Represents an event generated by a key press or release.
- Commonly used with the KeyListener interface.

6. KeyListener:

- An interface for handling KeyEvent events. Classes that implement this interface can respond to keyboard events.

7. MouseEvent:

- Represents an event generated by a mouse action, such as a mouse click, movement, or drag.
- Commonly used with the MouseListener and MouseMotionListener interfaces.

8. MouseListener:

- An interface for handling mouse events, including clicks, releases, and enters/exits.
- Classes implementing this interface can respond to mouse-related events

9. MouseMotionEvent:

- Represents an event generated by a mouse movement.
- Used with the MouseMotionListener interface.

10. MouseMotionListener:

- An interface for handling mouse motion events. Classes that implement this interface can respond to mouse movement.

Event Classes With Methods (Must Learn)

3.2.5 TextEvent Class

Description	This event indicates that object's text is changed.
Constructor	TextEvent(Object source, int id) It constructs a TextEvent object.
Methods	**String paramString() :** Returns a parameter string identifying this text event.
Fields	• **TEXT_FIRST :** The first number in the range of ids used for text events. • **TEXT_LAST :** The last number in the range of ids used for text events. • **TEXT_VALUE_CHANGED :** This event id indicates that object's text changed.

3.2.4 MouseEvent Class

Description	This event occurs when mouse action occurs in a component. It reacts for both mouse event and mouse motion event. • Mouse Events o A mouse button is pressed o A mouse button is released o A mouse button is clicked (pressed and released) o The mouse cursor enters a component o The mouse cursor exits a component • Mouse Motion Events o The mouse is moved o The mouse is dragged
Methods	**1) int getButton() :** It returns which of the mouse button has changed the state. **2) int getClickCount() :** It returns number of mouse clicks. **3) Point getLocationOnScreen() :** Returns the absolute x, y position at that event. **4) Point getPoint() :** Returns the x,y position of the event relative to the source component. **5) int getX() :** Returns the horizontal x position of the event relative to the source component. **6) int getY() :** Returns the vertical y position of the event relative to the source component.

3.2.3 KeyEvent Class

Description	This event is generated when key on the keyboard is pressed or released.
Constructors	**KeyEvent**(Component *source*, int *type*, long *t*, int *modifiers*, int *code*) The *source* is a reference to the component that generates the event. The *type* specifies the type of event. The *t* denotes the system time at which the event occurs. The *modifiers* represent the modifier keys such as ALT, CNTRL, SHIFT that are pressed when an event occurs.

	The *code* represents the virtual keycode such as VK_UP, VK_ESCAPE and so on.
Methods	• **char getKeyChar() :** It returns the character when a key is pressed.
Constants	• KEY_PRESSED : This event is generated when the key is pressed. • KEY_RELEASED : This event is generated when the key is released. • KEY_TYPED : This event is generated when the key is typed.

3.2.2 ItemEvent Class

Description	This event gets caused when an item is selected or deselected.
Constructors	ItemEvent(ItemSelectable, int, Object, int) Constructs a ItemSelectEvent object with the specified ItemSelectable source, type, item and item select state.
Methods	**1) getItem() :** It returns the item where the event occurred. **2) getItemSelectable() :** Returns the ItemSelectable object where this event originated. **3) getStateChange() :** Returns the state change type which generated the event.
Constants	1) ITEM_FIRST : Marks the first integer id for the range of item 2) ITEM_LAST : Marks the last integer id for the range of item 3) ITEM_STATE_CHANGED : The item state changed event type 4) SELECTED : The item selected state change type 5) DESELECTED : The item de-selected state change type

3.2.1 ActionEvent Class

Description	When an event gets generated due to pressing of button, or by selecting menu item or by selecting an item, this event occurs.
Constructors	**ActionEvent**(Object *source*, int *id*, string *command*, long *when*, int *modifier*) The *source* indicates the object due to which the event is generated. The *id* which is used to identify the type of event. The *command* is a string that specifies the command that is associated with the event. The *when* denotes the time of event. The *modifier* indicates the modifier keys such as ALT, CNTRL, SHIFT that are pressed when an event occurs.
Methods	• **String getActionCommand() :** This method is useful for obtaining the *command* string which is specified during the generation of event. • **int getModifiers() :** This method returns the value which indicates the type of key being pressed at the time of event. • **long getWhen() :** It returns the time at which the event occurs.
Constants	There are four constants that are used to indicate the modifier keys being pressed. These constants are CTRL_MASK, SHIFT_MASK, META_MASK and ALT_MASK.

3.2.6 WindowEvent Class

Description	This is an event that indicates that a window has changed its status.
Constructor	**WindowEvent(Window source, int id)** : Constructs a WindowEvent object. **WindowEvent(Window source, int id, int oldState, int newState)** : Constructs a WindowEvent object with the specified previous and new window states.
Methods	**1) int getNewState() :** It returns the new state of the window. **2) int getOldState() :** It returns the previous state of the window. **3) Window getOppositeWindow() :** Returns the other Window involved in this focus or activation change. **4) Window getWindow() :** Returns the originator of the event.
Fields	• **WINDOW_ACTIVATED :** The window-activated event type. • **WINDOW_CLOSED :** The window closed event. • **WINDOW_DEACTIVATED :** The window-deactivated event type. • **WINDOW_FIRST :** The first number in the range of ids used for window events. • **WINDOW_LAST :** The last number in the range of ids used for window events. • **WINDOW_OPENED :** The window opened event. • **WINDOW_STATE_CHANGED :** The window-state-changed event type.

Adapter Class

In Java event handling, an adapter class is a convenience class that provides empty implementations (default or "no-op" implementations) of all methods defined in a particular listener interface. Adapter classes are useful when you want to create an object that responds to events but doesn't need to provide a concrete implementation for all the methods of the listener interface.

- The adapter classes are part of the `java.awt.event` package and typically have names ending with "Adapter." These classes are designed to be extended, and you can override only the methods you are interested in, instead of implementing all methods in the listener interface.

Here are some commonly used adapter classes in Java event handling:

1. ActionAdapter (for ActionListener):

- Provides empty implementations for the `actionPerformed` method of the ActionListener interface.

2. ItemAdapter (for ItemListener):

- Provides empty implementations for the `itemStateChanged` method of the ItemListener interface.

3. KeyAdapter (for KeyListener):

- Provides empty implementations for the `keyPressed`, `keyReleased`, and `keyTyped` methods of the KeyListener interface.

4. MouseAdapter (for MouseListener):

- Provides empty implementations for the `mouseClicked`, `mouseEntered`, `mouseExited`, `mousePressed`, and `mouseReleased` methods of the MouseListener interface.

5. MouseMotionAdapter (for MouseMotionListener):

- Provides empty implementations for the `mouseDragged` and `mouseMoved` methods of the MouseMotionListener interface.

6. MouseWheelAdapter (for MouseWheelListener):

- Provides an empty implementation for the `mouseWheelMoved` method of the MouseWheelListener interface.

7. FocusAdapter (for FocusListener):

- Provides empty implementations for the `focusGained` and `focusLost` methods of the FocusListener interface.

8. WindowAdapter (for WindowListener):

- Provides empty implementations for all methods of the WindowListener interface.

End of Day 1

UR ENGINEERING FRIEND

#1 stop Destination for Diploma & Degree