

## Module-5: GUI PROGRAMMING AND APPLETS

### AWT class:

The **AWT (Abstract Window Toolkit)** is a key part of Java's **original graphical user interface (GUI) framework**. It provides a set of classes for building user interfaces (UI) like **windows, buttons, text fields, and menus**.

Java **AWT (Abstract Window Toolkit)** is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

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

### AWT Class Hierarchy:

#### 1. Object

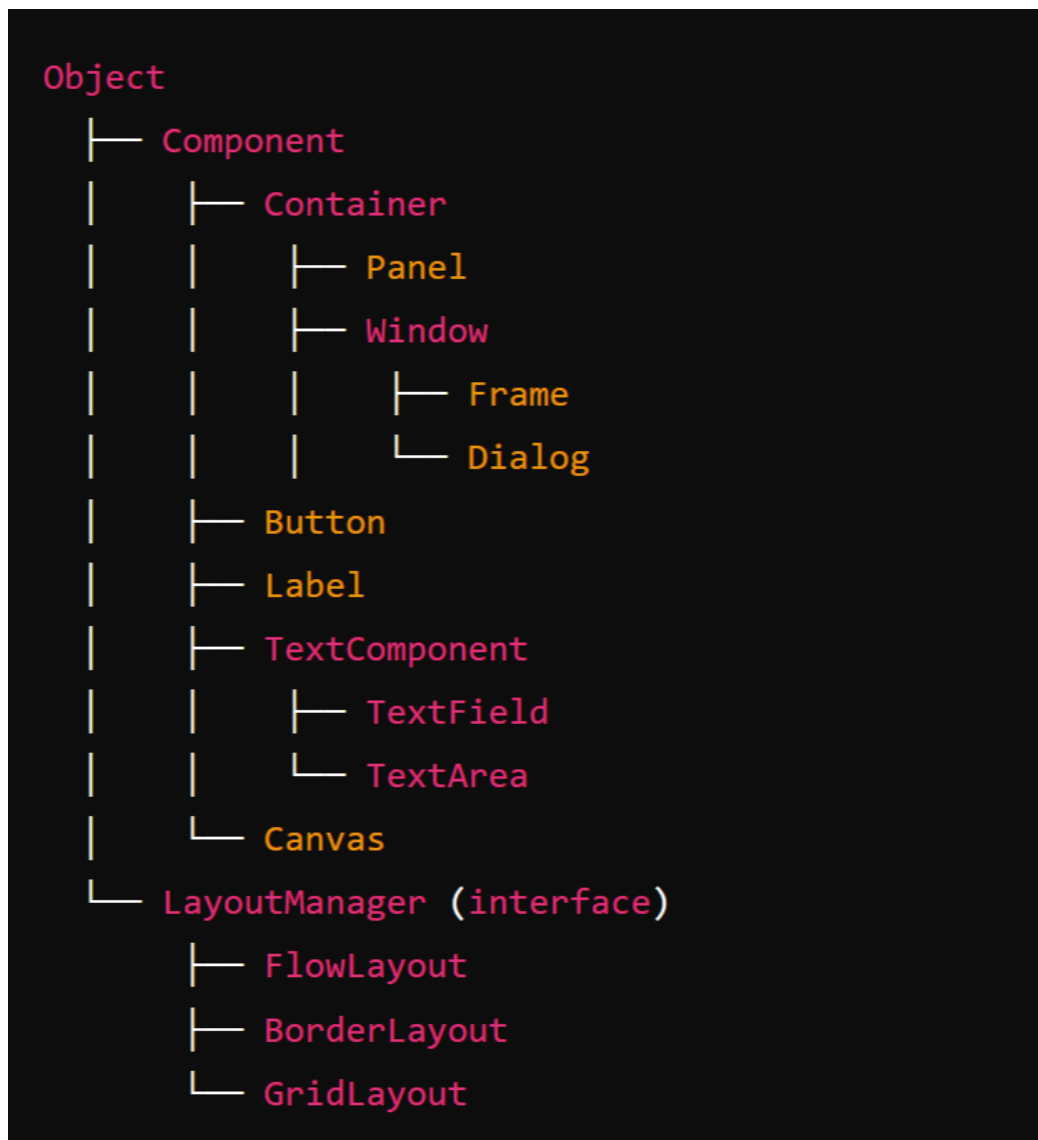
The root of the class hierarchy in Java. All classes inherit from Object, including those in AWT.

#### 2. Component (extends Object)

The **base class** for all AWT components. It represents the common attributes and behaviors of UI elements, such as buttons, labels, text fields, etc.

### Key subclasses of Component:

- **Container:** Can contain other components.
- **Button:** Represents a clickable button.
- **Label:** Displays a text string.
- **TextComponent:** Base class for TextField and TextArea.
- **Scrollbar:** Adds a scrollbar to a component.
- **Canvas:** Provides a blank area on which custom drawing can be done.



### 3. Container (extends Component):

A specialized component that can **hold other components** (like panels, frames).

#### Subclasses of Container:

1. **Window:** Represents a **top-level window** with **no borders** or **menus** (typically used for creating dialogs or windows).

#### Subclasses of Windows:

- **Dialog:** A pop-up window often used to interact with the user.

- **Frame:** A fully functional window with **title**, **borders**, and **buttons** like minimize, maximize, and close.

## 2. **Panel (extends Container):**

A generic container used for **grouping other components**.

A container for organizing components in a specific layout.  
Frequently used for grouping elements.

## 4. **TextComponent (extends Component):**

The base class for text-related components.

### **Subclasses:**

- **TextField:** A single-line text input field.
- **TextArea:** A multi-line text input area.

## 5. **LayoutManager**

An interface used to define how components are arranged in a container.

Key implementations:

- **FlowLayout:** Lays out components in a row.
- **BorderLayout:** Arranges components in five areas (North, South, East, West, and Center).
- **GridLayout:** Arranges components in a grid of rows and columns.

**Swing:**

**Swing** is a part of Java's **JFC (Java Foundation Classes)** and is a more **advanced** and **flexible** GUI toolkit compared to AWT (Abstract Window Toolkit).

While AWT provides **basic components** and is **platform-dependent** (uses native GUI components), Swing is built on top of AWT and is **platform-independent** because it renders its components **using Java code** rather than relying on the **underlying operating system's** native components. This allows for a more consistent look and feel across different platforms.

### **Hierarchy for Swing components:**

The Swing component hierarchy in Java follows a **well-defined structure based on the AWT hierarchy** but adds its **own** set of components for creating **rich user interfaces**.

#### **1. Object**

The root class of all Java classes.

#### **2. Component (from AWT)**

The base class for all graphical components that can be added to a GUI.

#### **3. Container (from AWT)**

A subclass of Component that can hold other components (like panels or frames).

#### **4. JComponent (extends Container)**

The **base class for all Swing components**. It provides additional functionality such as double buffering, **borders**, and **tooltips**.

Object

└─ Component (from AWT)

└─ Container (from AWT)

└─ JComponent

└─ JButton

└─ JLabel

└─ JTextField

└─ JTextArea

└─ JPanel

└─ JScrollPane

└─ JTable

└─ JMenuBar

└─ JFrame

└─ JDialog

**Core Swing Components (Subclasses of JComponent):**

Component	Description
<b>JButton</b>	Represents a clickable button.
<b>JLabel</b>	Displays static text or an image.
<b>TextField</b>	A single-line text input field.
<b>TextArea</b>	A multi-line text area for text input or output.
<b>JPanel</b>	A generic lightweight container used for <b>organizing components</b> .
<b>JScrollPane</b>	Provides a scrollable view of another component, like a text area or table.
<b>JTable</b>	A component that displays tabular data in rows and columns.
<b>JList</b>	Displays a list of items.
<b>JComboBox</b>	A drop-down list that allows the user to select one item from a list of choices.
<b>JCheckBox</b>	A component that represents a check box (on/off state).
<b>JRadioButton</b>	Represents a radio button (allows selection of one option within a group).
<b>JMenuBar</b>	Components for creating a menu bar.
<b>JMenu</b>	Represents a menu within a menu bar.
<b>JMenuItem</b>	Represents an item within a menu.

## Top-Level Containers:

### 1. JFrame (extends Frame):

A top-level window that **contains** the main application window.

### 2. JDialog (extends Dialog):

A pop-up window for user interaction (e.g., confirmation dialogs).

### 3. JWindow (extends Window)

A window **without any borders** or **title bar**.

### 4. JApplet (extends Applet):

A container for applets, used in embedding GUI components in web browsers.

### 5. JToolBar:

Provides a set of actions or controls, often used for creating toolbars.

## Specialized Components:

### 1. JTabbedPane

Manages **multiple components with tabs**, allowing the user to **switch between them**.

### 2. JSpinner:

Allows the user to select a value from a sequence of values (like a number spinner).

### 3. JProgressBar:

Displays the progress of a task.

### 4. JTree

Displays a hierarchical tree of data (e.g., a file directory structure).

## Difference between Swing and AWT Classes:

Feature	AWT (Abstract Window Toolkit)	Swing (Java Foundation Classes)
Component Model	Uses native OS components, making it <b>platform-dependent</b> .	<b>Built entirely in Java</b> , providing a consistent look and feel across platforms.
Lightweight vs. Heavyweight	<b>Heavyweight components</b> (each AWT component is a wrapper around a native system component).	<b>Lightweight components</b> (not tied to native components, allowing more flexibility and customization).
Look and Feel	<b>Limited</b> to the platform's <b>native look and feel</b> .	Supports <b>pluggable look and feel</b> , allowing for greater customization and different styles.
Event Handling	Uses a <b>simpler event handling</b> model based on listener interfaces.	Provides a <b>more advanced event handling</b> model with more event types and capabilities.
Graphics	<b>Basic graphics capabilities</b> , primarily for drawing shapes and text.	<b>Richer graphics capabilities</b> , including advanced rendering and painting options.
Containers	<b>Limited</b> container options (e.g., Frame, Panel).	<b>More flexible</b> container hierarchy (e.g., <a href="#">JFrame</a> , <a href="#">JPanel</a> , <a href="#">JLayeredPane</a> ).
Performance	Generally <b>faster for simple applications</b> due to native components.	Slightly <b>slower due to the overhead of additional features</b> and abstraction.
Threading Model	<b>Not as robust</b> , leading to potential issues with performance and responsiveness.	Supports the <code>SwingWorker</code> class for better concurrency management in UI updates.
Availability	Part of the original Java AWT library.	Introduced later as part of Java 2 (JFC) and is now standard in Java applications.
Development Complexity	<b>Easier for simple GUIs</b> , but can become complex for larger applications.	<b>More complex to learn</b> due to its rich feature set but offers better support for larger applications.

Some Important Classes or Components:

## 1. JFrame:

A JFrame is one of the most **important** classes in the Java Swing library. It is a **top-level container** that represents a **window** in a graphical user interface (GUI) application.

JFrame provides a window with all the standard window features, such as a **title bar**, **minimize/maximize buttons**, and a **close button**.

### Example:

```
import javax.swing.*;

public class JFrameExample {
    public static void main(String[] args) {
        // Create a JFrame instance
        JFrame frame = new JFrame("My JFrame Example");

        // Set the size of the frame
        frame.setSize(400, 300);

        // Specify what happens when the close button is clicked
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Add a button to the frame
        JButton button = new JButton("Click Me");
        frame.add(button);

        // Make the frame visible
        frame.setVisible(true);
    }
}
```

## 2. JApplet :

JApplet is a class in the Java Swing library that **extends** the **Applet class** and provides a framework for **building applets** with a Swing-based graphical user interface (GUI).



Applets are small Java programs that are typically **embedded within a web page** and run in a web browser. JApplet was designed to use the rich, lightweight components from Swing, as opposed to the heavyweight AWT components used in Applet.

### 3. JDialog:

JDialog is a part of the Java Swing library and is used to create **dialog windows**, which are smaller windows that appear on top of the main application window (JFrame).

Unlike **JFrame**, which represents a full-fledged window, JDialog is typically used for **temporary, pop-up windows** that require user interaction, such as **alerts, confirmations, or input forms**.

### 4. Jpanel:

JPanel is one of the most commonly used components in the Java Swing library. It is a **lightweight container** that can hold and organize a group of components, such as **buttons, labels, text fields**, or other panels. It serves as a flexible, invisible container to help structure the layout of GUI.

### Example:

1. **JDialog**
2. **Jpanel**
3. **Jlabel**
4. **JTextField**
5. **JButton**

```

import javax.swing.*;

public class JDialogDemo {
    public static void main(String[] args) {
        // Create the main application window (JFrame)
        JFrame frame = new JFrame("Main Window");
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create a button that opens the dialog
        JButton openDialogButton = new JButton("Enter Name");
        frame.add(openDialogButton); // Add the button to the JFrame

        // Create a dialog (JDialog)
        JDialog dialog = new JDialog(frame, "Name Input", true); // Modal dialog
        dialog.setSize(300, 150);

        // Add components to the dialog
        JLabel nameLabel = new JLabel("Enter your name:");
        JTextField nameField = new JTextField(15);
        JButton okButton = new JButton("OK");

        JPanel panel = new JPanel();
        panel.add(nameLabel); // Add label to the panel
        panel.add(nameField); // Add text field to the panel
        panel.add(okButton); // Add OK button to the panel
        dialog.add(panel); // Add the panel to the dialog

        // Action listener for the "Enter Name" button
        openDialogButton.addActionListener(e -> dialog.setVisible(true));

        // Action listener for the "OK" button in the dialog
        okButton.addActionListener(e -> {
            String name = nameField.getText(); // Get the input from the text field
            if (!name.isEmpty()) {
                JOptionPane.showMessageDialog(frame, "Hello, " + name + "!");
                dialog.setVisible(false); // Close the dialog after OK is clicked
            } else {
                JOptionPane.showMessageDialog(dialog, "Please enter your name.");
            }
        });

        // Show the main window
        frame.setVisible(true);
    }
}

```

## Layout management:

Layout Manager controls the **positioning** and **sizing** of **components within a container**. Different layout managers provide different ways

to arrange components, making GUI design more **flexible** and **dynamic**.

### Layout manager types:

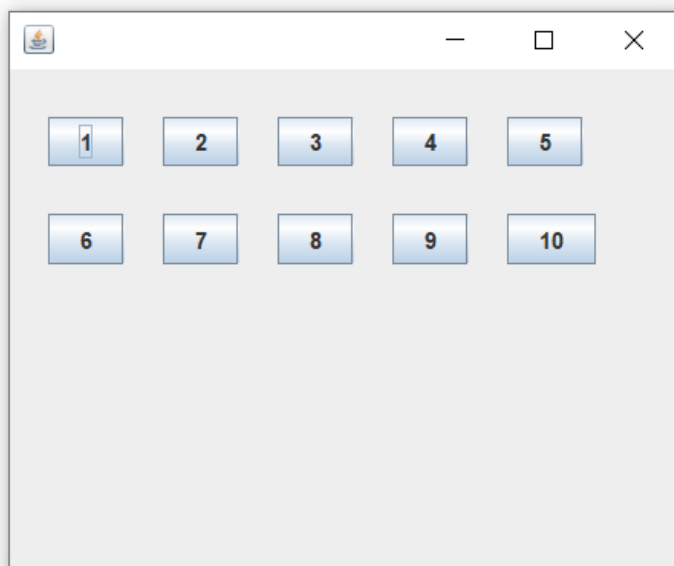
Java Swing provides several layout managers that allow you to organize components in different ways within containers like JFrame, JPanel, and JDialog.

1. FlowLayout
2. BorderLayout
3. GridLayout

#### 1. FlowLayout :

FlowLayout arranges components **in a line** (left to right) and **wraps to the next line** if there isn't enough space. Components are aligned to the **left, center, or right** (default is centered).

Use Case: Common for toolbar-like layouts or for smaller components that don't require precise.



#### Example:

```

import javax.swing.*;
import java.awt.*;

public class FlowLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("FlowLayout Example");
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Set FlowLayout with centered alignment
        frame.setLayout(new FlowLayout(FlowLayout.CENTER));

        for (int i = 1; i <= 5; i++) {
            frame.add(new JButton("Button " + i));
        }

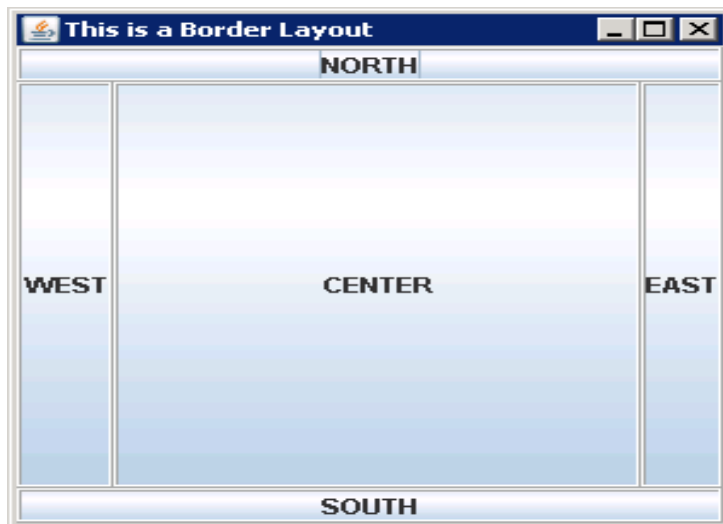
        frame.setVisible(true);
    }
}

```

## 2. BorderLayout:

BorderLayout **divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER.** Each region can hold only one component. The CENTER region expands to fill any remaining space.

**Use Case:** Suitable for applications with a main central area and surrounding sections (like a header, footer, or sidebars).



Example:

```
import javax.swing.*;
import java.awt.*;

public class BorderLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("BorderLayout Example");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Set BorderLayout as the layout manager
        frame.setLayout(new BorderLayout());

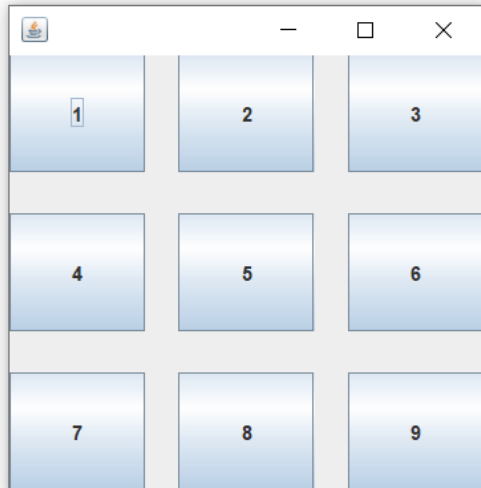
        frame.add(new JButton("North"), BorderLayout.NORTH);
        frame.add(new JButton("South"), BorderLayout.SOUTH);
        frame.add(new JButton("East"), BorderLayout.EAST);
        frame.add(new JButton("West"), BorderLayout.WEST);
        frame.add(new JButton("Center"), BorderLayout.CENTER);

        frame.setVisible(true);
    }
}
```

3. GridLayout:

GridLayout arranges components in a **rectangular grid** of **rows** and **columns**. Each cell in the grid is of **equal size**, and all components are resized to fit within their assigned cell.

**Use Case:** Useful for creating uniform, grid-like layouts, such as calculator buttons or a game board.



**Example:**

```
import javax.swing.*;
import java.awt.*;

public class GridLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("GridLayout Example");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Set GridLayout with 2 rows and 3 columns
        frame.setLayout(new GridLayout(2, 3));

        for (int i = 1; i <= 6; i++) {
            frame.add(new JButton("Button " + i));
        }

        frame.setVisible(true);
    }
}
```

# Applet:

An **applet** in Java is a **small application that can run** in a **web browser** or **applet viewer**. To create a Interactive or dynamic Web Page.

Applets were primarily designed to provide **interactive features** in **web pages**, but over time **their usage has decreased**, especially with the shift towards modern web technologies like **HTML5, JavaScript,** and **CSS**.

## What is an Applet?

An applet is a **subclass** of **java.applet.Applet** or **javax.swing.JApplet**.

It is a Java program **embedded into web pages** and can be executed using a **Java-enabled browser** or an **applet viewer tool**.

Applets are different from standalone applications because they **don't have a main() method**. Instead, they rely on **lifecycle methods** for execution.

## Types of Applets:

- **AWT Applet (java.applet.Applet):** Uses Abstract Window Toolkit (AWT) components like Button, Label, TextField.
- **Swing Applet (javax.swing.JApplet):** Uses Swing components like JButton, JLabel, JTextField. It offers more advanced GUI components compared to AWT.

## Inheritance hierarchy for applets:

In Java, applets are part of the class hierarchy that ultimately inherits from the base class **java.lang.Object**. Applets can be written using either the Abstract Window Toolkit (AWT) or Swing.

### 1. AWT Applet Inheritance Hierarchy:

AWT-based applets inherit from the **java.applet.Applet** class, which in turn inherits from other standard Java classes. Here is the detailed **inheritance hierarchy**:

```
java.lang.Object
└ java.awt.Component
  └ java.awt.Container
    └ java.awt.Panel
      └ java.applet.Applet
```

## 2. Swing Applet Inheritance Hierarchy:

Swing-based applets inherit from **javax.swing.JApplet**, which adds more advanced GUI capabilities by utilizing the Swing toolkit.

**Hierarchy:**

```
java.lang.Object
└ java.awt.Component
  └ java.awt.Container
    └ java.awt.Panel
      └ java.applet.Applet
        └ javax.swing.JApplet
```

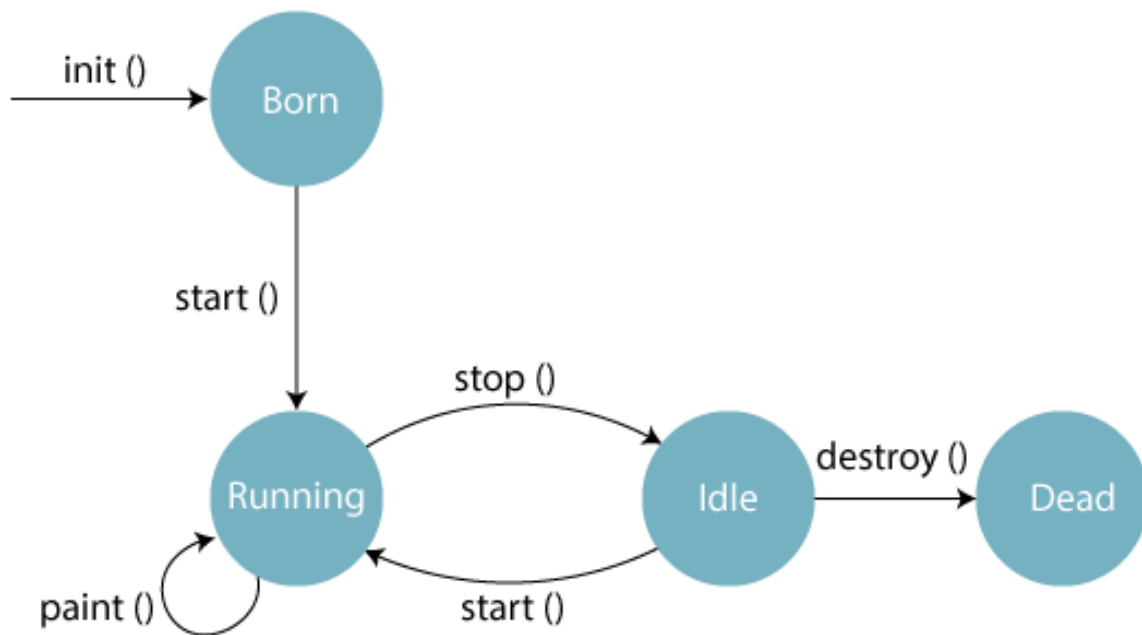


## Differences between Applets and Applications:

Feature	Java Applet	Java Application
Definition	A <b>small program</b> that runs within a <b>web browser</b> or <b>applet viewer</b> .	A <b>standalone program</b> that runs directly on the <b>JVM</b> .
Execution Environment	Runs in a web browser or applet viewer (like <code>appletviewer</code> ).	Runs in the Java Virtual Machine (JVM).
Main Method	<b>No <code>main()</code></b> method; uses <code>init()</code> , <code>start()</code> , <code>stop()</code> , and <code>destroy()</code> methods.	Entry point is <b><code>public static void main(String[] args)</code></b> .
Security Restrictions	Restricted by a <b>security sandbox</b> (can't access local file system by default).	Has <b>full access</b> to the system resources (based on permissions).
User Interaction	Limited due to <b>browser restrictions</b> .	<b>Full control</b> over user interaction and system resources.
GUI Creation	Typically <b>uses AWT</b> (Abstract Window Toolkit) for user interface components.	Can use <b>Swing, AWT, JavaFX</b> , or other libraries for GUI creation.
Deployment	<b>Embedded</b> within a web page as an applet tag or called in an HTML page.	Run directly from a <b>Java archive (JAR)</b> or <b>class file</b> on the desktop or server.
Network Access	Limited to the server from which it was downloaded, unless explicitly signed.	No restrictions on network access (depends on application configuration).
Performance	Generally <b>slower</b> due to browser overhead and security checks.	Generally <b>faster</b> as it runs directly on the local JVM.
Modern Usage	<b>Rarely used</b> , as browsers no longer support applets due to security concerns.	<b>Widely used</b> in a variety of desktop and server applications.

## Life Cycle of an Applet:

- In Java, an applet is a special type of program **embedded** in the web page to generate **dynamic content**. Applet is a class in Java.
- The applet life cycle can be defined as the process of how the **object is created, started, stopped, and destroyed** during the entire execution of its application. It basically has **five core methods** namely `init()`, `start()`, `stop()`, `paint()` and `destroy()`. These methods are invoked by the browser to execute.
- In Java, the life cycle of an applet consists of a sequence of methods that manage the different stages of an applet's existence. These methods ensure that the applet is **initialized, starts running, stops** when needed, and finally, is **destroyed** when no longer required.



### 1. Initialization (init()):

- Called when the applet is first loaded.
- This method is used to **initialize the applet**, such as **setting up the initial state, loading resources, or setting up the user interface**.
- **Executed only once** when the **applet starts**.

### 2. Starting (start()):

- **Called each time** the applet becomes active (e.g., when the user visits or revisits the page containing the applet).
- Typically used **to start animations** or any activity that needs to continue as long as the applet is active.
- **Can be called multiple times** if the applet is paused and resumed.

### 3. Painting (paint(Graphics g)):

- Called **whenever the applet's display needs to be redrawn**.
- Used to **render graphics** and text on the applet's display area.

- **Automatically called after init() and start(),** and can be invoked by calling **repaint()**.

#### **4. Stopping (stop() ):**

- Called when the **applet is no longer active**, like when the user navigates away from the page.
- Used to **pause animations** or **other ongoing activities**.
- Can be called multiple times when the applet is paused and resumed.

#### **5. Destruction (destroy()):**

- Called when the **applet is being removed from memory** (usually when the **browser is closed**).
- Used to **release resources** and **perform cleanup tasks**.
- **Executed only once** in the applet's lifecycle, **after stop()**.

#### **Example:**

#### **2 Ways to Run this Applet Program:**

1. Using **AppletViewer Tool** (Without HTML File)
2. By **Embedding** the .class file in **HTML**.

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

public class LifeCycleApplet extends Applet {

    // Called once when the applet is first loaded
    @Override
    public void init() {
        System.out.println("init() method called: Applet initialized.");
    }

    // Called each time the applet is started or restarted
    @Override
    public void start() {
        System.out.println("start() method called: Applet started.");
    }

    // Called whenever the applet needs to repaint its contents
    @Override
    public void paint(Graphics g) {
        System.out.println("paint() method called: Applet repainting.");
        g.drawString("Hello, Applet Lifecycle!", 20, 20);
    }

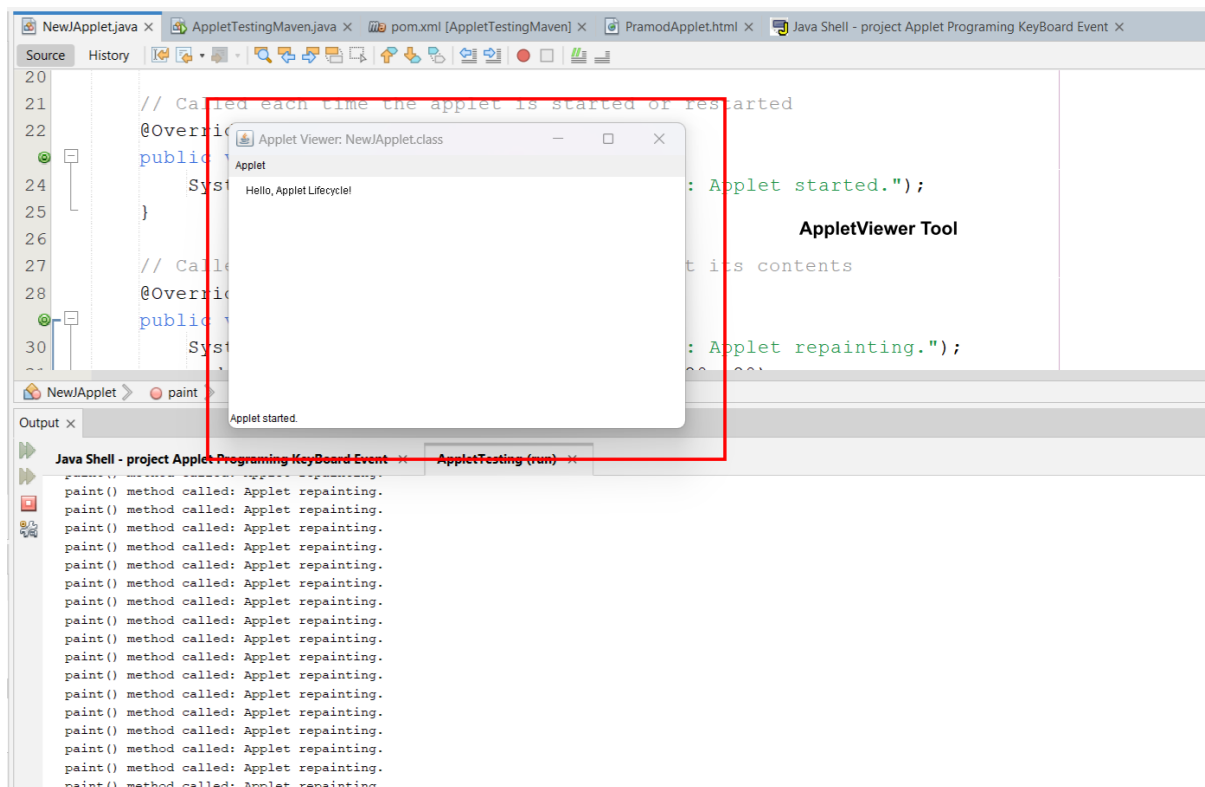
    // Called each time the applet is stopped
    @Override
    public void stop() {
        System.out.println("stop() method called: Applet stopped.");
    }

    // Called once when the applet is being destroyed
    @Override
    public void destroy() {
        System.out.println("destroy() method called: Applet destroyed.");
    }
}
```

## Embedding the .class into HTML File:

```
<!DOCTYPE html>
<html>
<head>
  <title>LifeCycleApplet Example</title>
</head>
<body>
  <h1>Applet Lifecycle Example</h1>
  <applet code="LifeCycleApplet.class" width="300" height="200">
    Your browser does not support Java Applets.
  </applet>
</body>
</html>
```

## Output: AppletViewer Tool



## Passing Parameters to Applets:

- In Java, an applet can **receive parameters from** an **HTML** file using the **<param> tag**.
- These parameters are passed to the applet via the **getParameter()** method, which retrieves the value of a parameter specified in the HTML code.
- The parameters can then be used inside the applet to **control behavior, display messages**, etc.

### Steps to Pass the Parameter from HTML to Applet file:

#### 1. HTML Code:

Parameters are passed from an HTML page where the **applet is embedded**. The **<param> tag** inside the **<applet> tag** specifies the parameter **names** and their **values**.

#### 2. Applet Code:

In the applet's code, the **getParameter(String name)** method is used to **retrieve** the parameter values by specifying the **parameter name**.

#### 3. Key Methods:

**getParameter(String paramName):** Used to get the value of a parameter passed to the applet.

### Example:

#### 1. HTML File:

```
<html>
  <body>
    <h2>Passing Parameters to Applet</h2>
    <applet code="MyApplet.class" width="300" height="100">
      <param name="msg" value="Welcome to Java Applets!" />
    </applet>
  </body>
</html>
```

## 2. Applet Code: MyApplet.java

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

public class MyApplet extends Applet {
    String message;

    // Initialization of the applet
    public void init() {
        // Retrieve the parameter named 'msg' from the HTML
        message = getParameter("msg");

        // If no message is passed, use a default message
        if (message == null) {
            message = "Hello, World!";
        }
    }

    // Paint method to display the message
    public void paint(Graphics g) {
        g.drawString(message, 20, 20);
    }
}
```