**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.

A computer screen shot of text

Description automatically generated

**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.
  1. **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)**

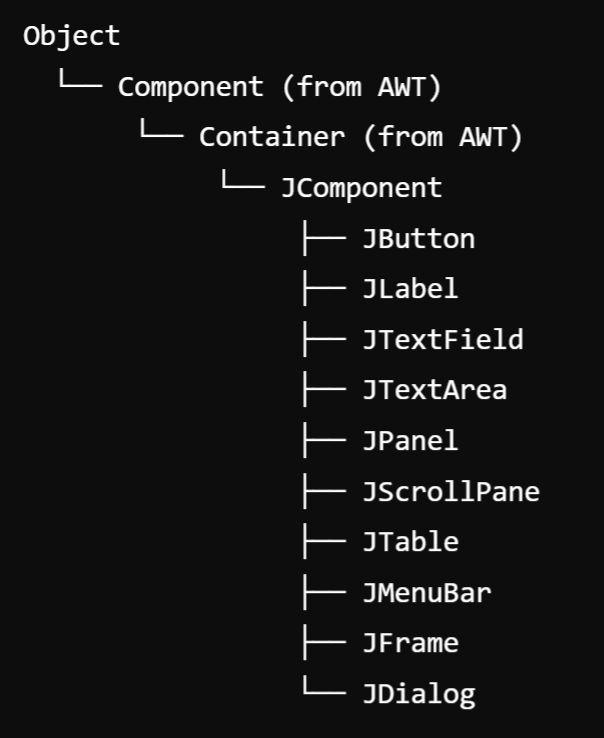
The base class for all graphical components that can be added to a GUI.

**3. Container (from AWT)**

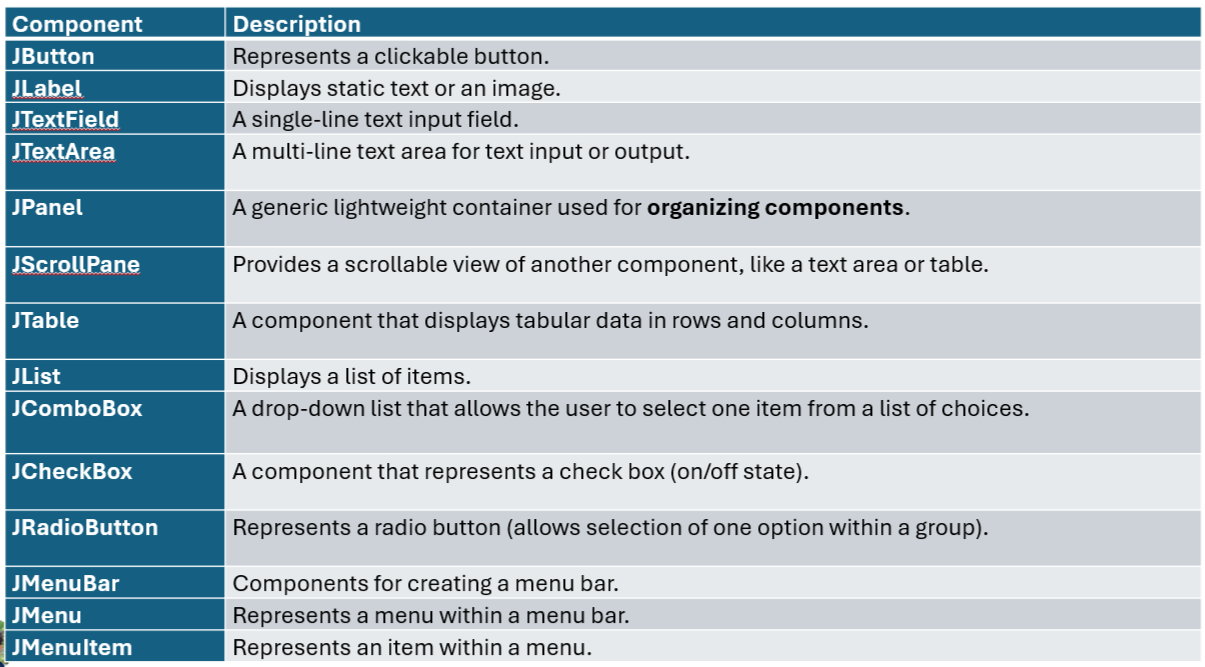
A subclass of Component that can hold other components (like panels or frames).

**4. JComponent (extends Container)**

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



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



**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).

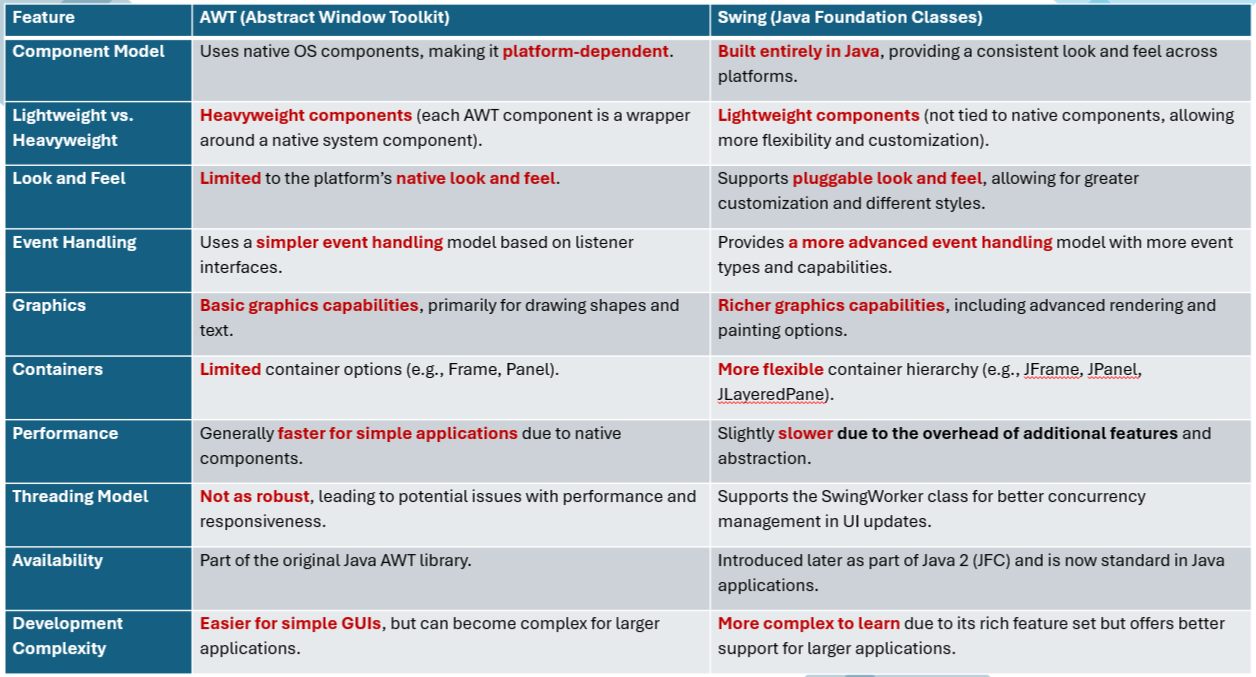
**3. JProgressBar:**

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:**



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:**

****

1. **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.

1. **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**.

1. **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**

A screen shot of a computer program

Description automatically generated

**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**.

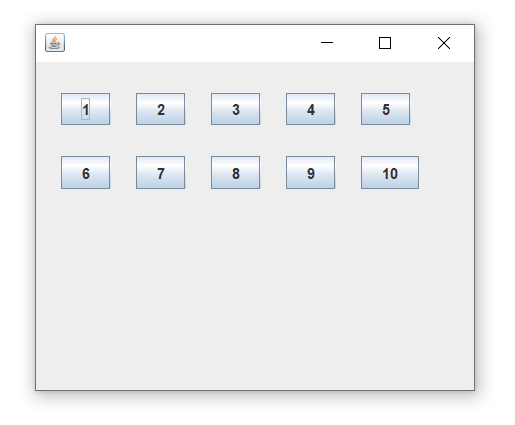
**Layout manager types:**

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
4. **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:**

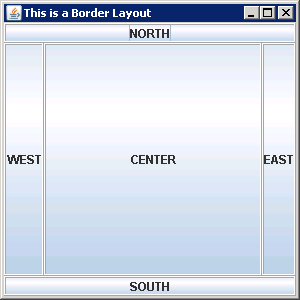
A computer screen with text and images

Description automatically generated

1. **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).



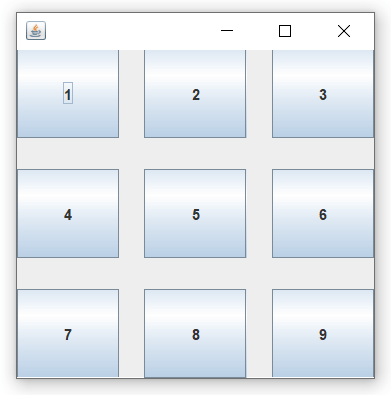
**Example:**

****

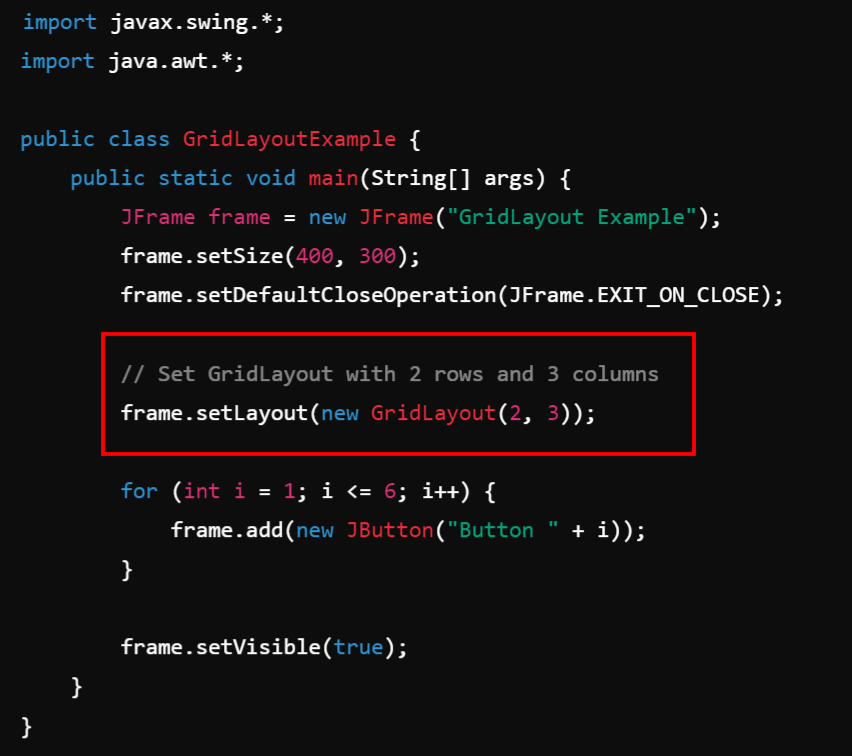
1. **GridLayout**:

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:**

****

**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:**

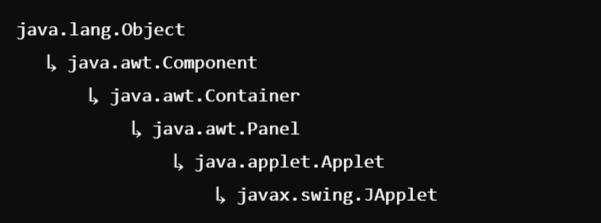
AWT-based applets inherit from the **java.applet.Applet** class, which in turn inherits from other standard Java classes. Here is the detailed **inheritance hierarchy:**



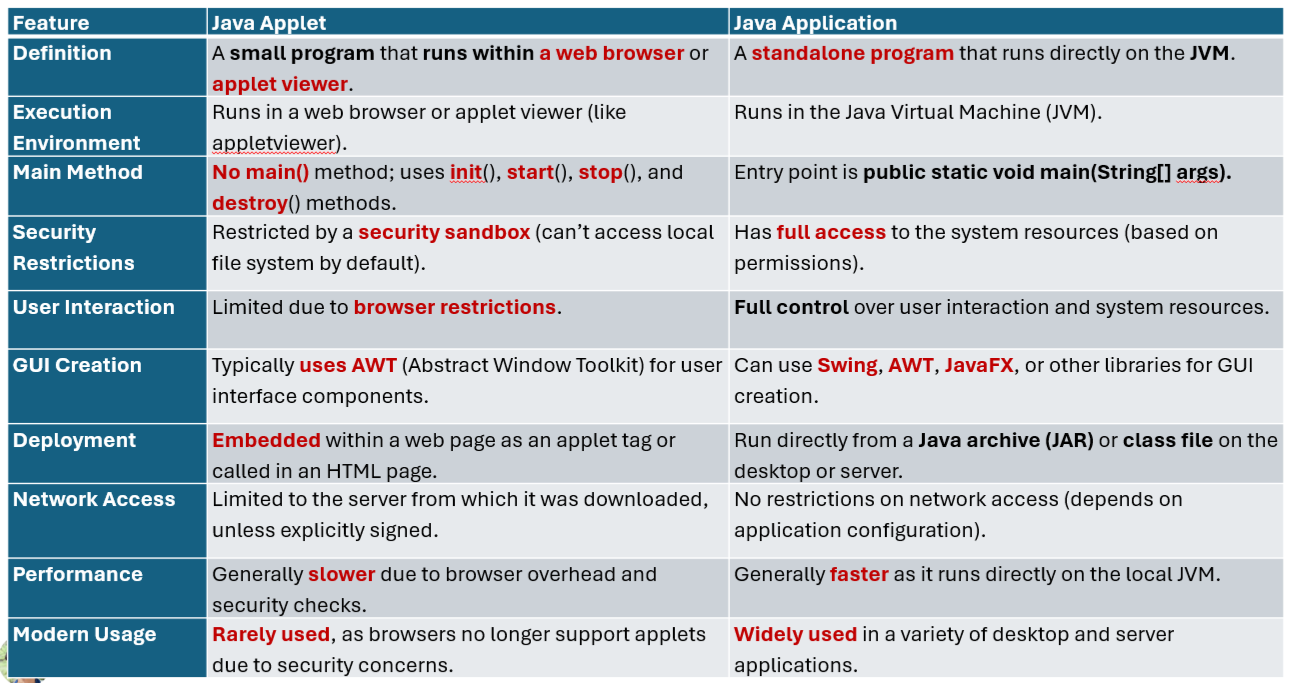
**2. Swing Applet Inheritance Hierarchy:**

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

**Hierarchy:**

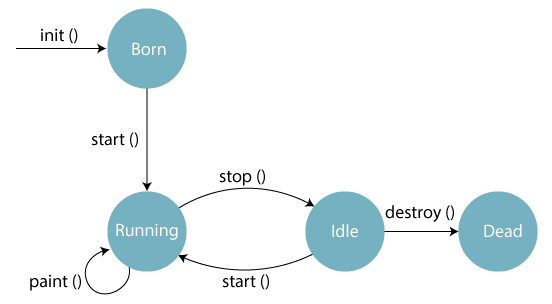


**Differences between Applets and Applications:**



**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**.

A screen shot of a computer program

Description automatically generated

**Embedding the .class into HTML File:**

**A screen shot of a computer program

Description automatically generated**

**Output: AppletViewer Tool**

**A screenshot of a computer

Description automatically generated**

**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:**

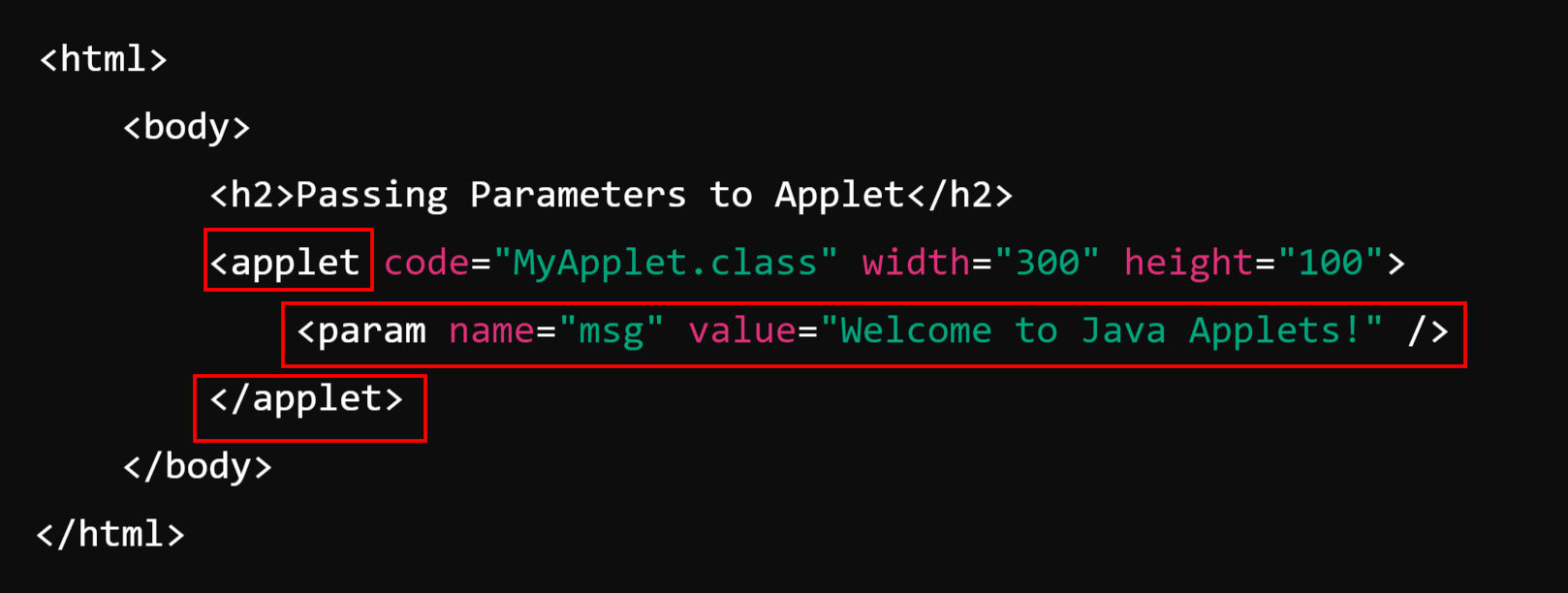
In the applet's code, the **getParameter(String name)** method is used to **retrieve** the parameter values by specifying the **parameter name.**

**3. Key Methods:**

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

**Example:**

1. **HTML File:**



**2. Applet Code: MyApplet.java**

