Sonawane Khushbu K.

UNIT-5:
GUI Programming using Java Applet

Introduction to applet

- **Let’s understand first how many Package does GUI support:**
- AWT(Abstract Window Toolkit)
- Swing

Throwback of making GUI application:

- Java was launched on 23-Jan-1996(JDK 1.0) and at that time it only supported CUI(Character User Interface) application. But in 1996 VB(Visual Basic) of Microsoft was preferred for GUI programming. So the Java developers in hurry(i.e within 7 days) have given the support for GUI from Operating System(OS). Now, the components like button, etc. were platform-dependent(i.e in each platform there will be different size, shape button). But they did the intersection of such components from all platforms and gave a small library which contains these intersections and it is available in AWT(Abstract Window Toolkit) technology but it doesn’t have advanced features like dialogue box, etc.

- Now to run Applet, java needs a browser and at that time only “Internet Explorer” was there of Microsoft but Microsoft believes in monopoly. So “SUN Micro-System”(the company which developed Java) contracted with other company known as “Netscape”(which developed Java Script) and now the “Netscape” company is also known as “Mozilla Firefox” which we all know is a browser. Now, these two companies have developed a technology called “SWING” and the benefit is that the SWING components are produced by Java itself. Therefore now it is platform-independent as well as some additional features have also been added which were not in AWT technology. So we can say that SWING is much more advanced as compared to AWT technology.

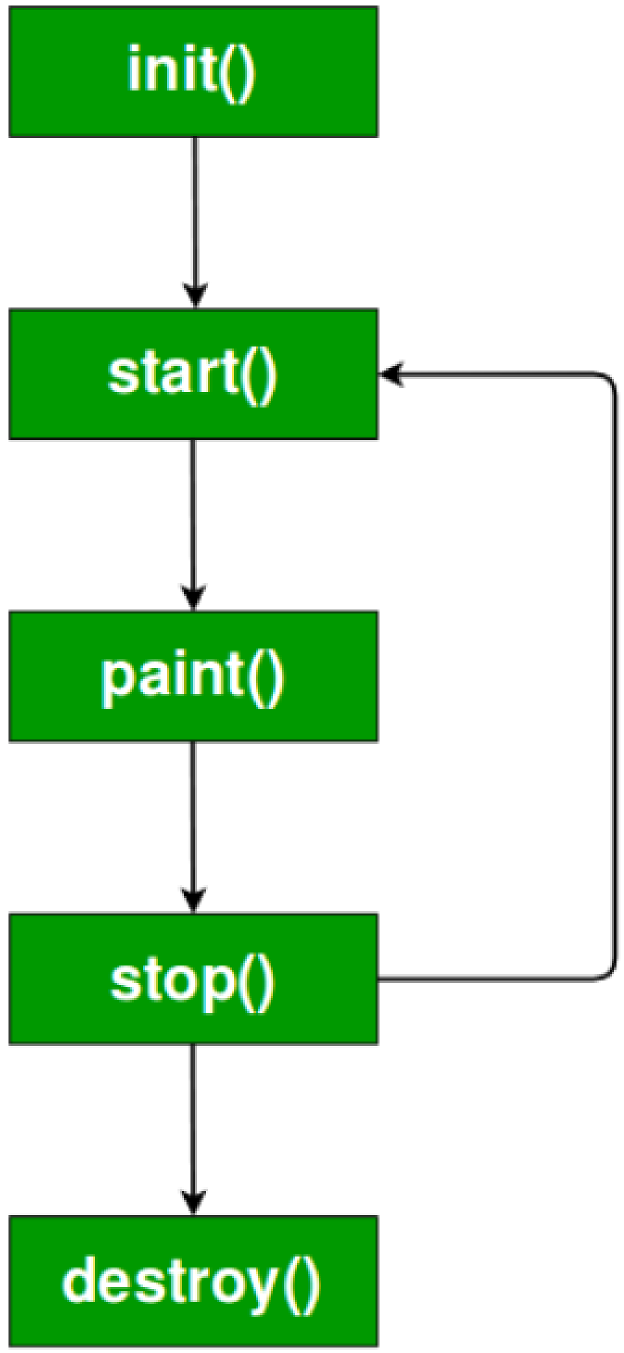
What is Applet?

- An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server. Applets are used to make the website more dynamic and entertaining.
- **Important points :**
- All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.
- Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
- In general, execution of an applet does not begin at main() method.
- Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

- **Life cycle of an applet :**

It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence:
init( )
start( )
paint( )

When an applet is terminated, the following sequence of method calls takes place:
stop( )
destroy( )


$$\begin{tikzcd}[row sep=large] \text{init()} \arrow[d] \\ \text{start()} \arrow[d] \\ \text{paint()} \arrow[d] \\ \text{stop()} \arrow[d] \arrow[uu, bend right=60] \\ \text{destroy()} \end{tikzcd}$$

- Let’s look more closely at these methods.

- **1. init( ) :** The **init( )** method is the first method to be called. This is where you should initialize variables. This method is called **only once** during the run time of your applet.

- **2. start( ) :** The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped. Note that **init( )** is called once i.e. when the first time an applet is loaded whereas **start( )** is called each time an applet’s HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

- **3. paint( ) :** The **paint( )** method is called each time an AWT-based applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored.
  paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint( )** is called.
  The **paint( )** method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.
  Note: This is the only method among all the method mention above, which is parameterized. It's prototype is
  public void paint(Graphics g)
  where g is an object reference of class Graphic.

- Now the **Question Arises:**

- **Q.** In the prototype of paint() method, we have created an object reference without creating its object. But how is it possible to create object reference without creating its object?

- **Ans.** Whenever we pass object reference in arguments then the object will be provided by its caller itself. In this case the caller of paint() method is browser, so it will provide an object. The same thing happens when we create a very basic program in normal Java programs.

- **For Example:**
- public static void main(String []args){}
- Here we have created an object reference without creating its object but it still runs because it’s caller,i.e JVM will provide it with an object.

- **4. stop( ) :**
- The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads that don’t need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

- **5. destroy( ) :**
- The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop( )** method is always called before **destroy( )**.

Difference between Applet and Application

- **1. Applet:**

- **Definition**: An applet is a small Java program that runs in a web browser or an applet viewer. Applets are embedded in HTML pages and executed in a client environment.

- **Execution Environment**: Applets run inside a web browser or in an applet viewer, requiring a Java plugin or JVM (Java Virtual Machine) installed in the browser.

- **Security Restrictions**: Due to security concerns, applets are run in a sandbox environment, meaning they have limited access to system resources (such as local files or network).

- **User Interface**: Applets typically provide a graphical user interface (GUI) and are embedded in web pages.

- **Lifecycle**: Applets have a defined lifecycle managed by the browser:
  - init(): Initializes the applet.
  - start(): Starts or resumes the execution of the applet.
  - stop(): Stops the execution of the applet.
  - destroy(): Terminates the applet.

- **Obsolescence**: Applets have become largely obsolete due to browser security concerns and the rise of modern web technologies (e.g., JavaScript frameworks). Most modern browsers no longer support applets.

- **2. Application:**

- **Definition**: A Java application is a standalone program that runs directly on a Java Virtual Machine (JVM), independent of a web browser.

- **Execution Environment**: Applications are executed directly by the JVM on the user's machine, typically from the command line or through an integrated development environment (IDE).

- **Security**: Unlike applets, applications have full access to system resources (files, network, etc.), although they can be restricted by security policies if needed.

- **User Interface**: Java applications may have no GUI (console-based applications) or can include GUIs using frameworks like AWT (Abstract Window Toolkit), Swing, or JavaFX.

- **Lifecycle**: Java applications have a simpler lifecycle than applets. The program execution starts with the main() method and ends when the main() method finishes executing or when the program is explicitly terminated.

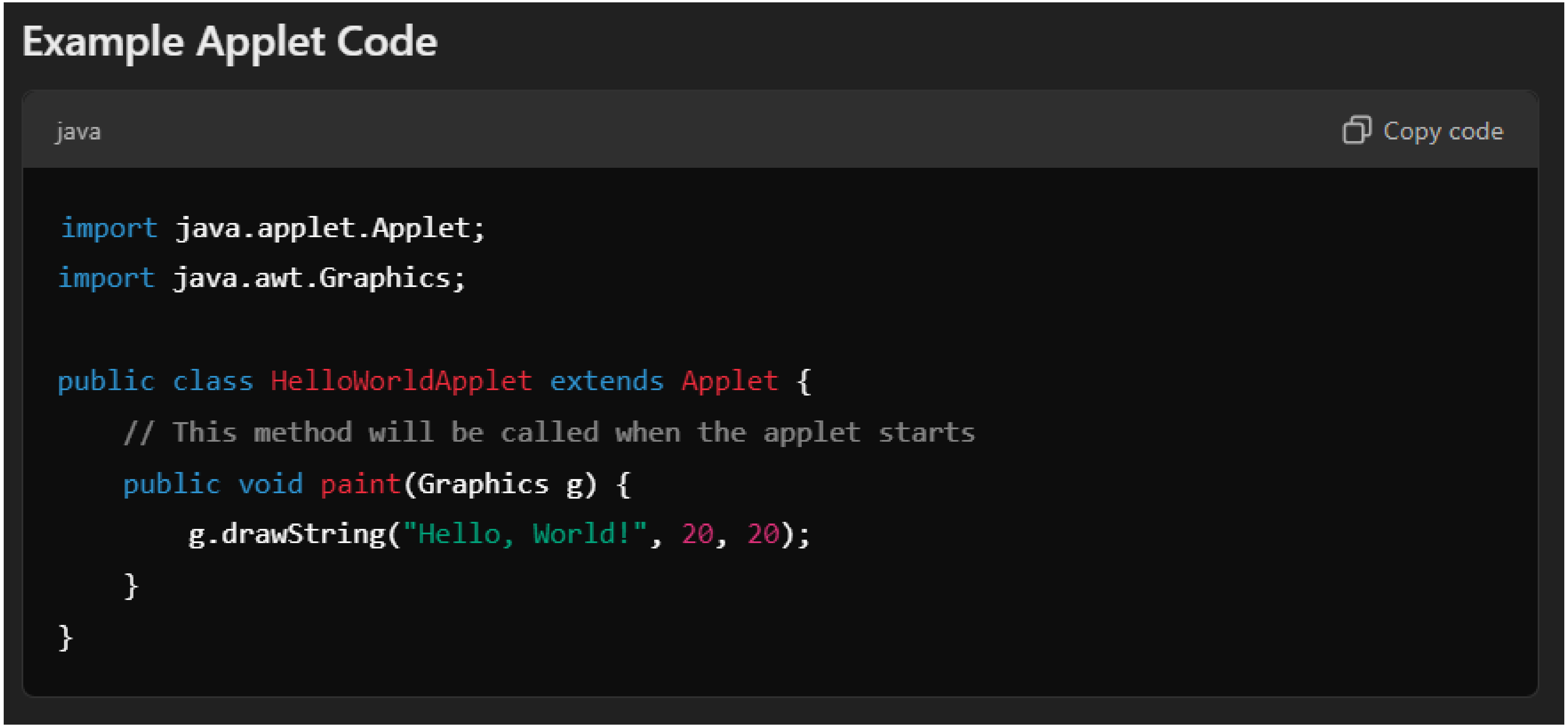
- **Common Uses**: Applications are used for a wide range of purposes, including desktop applications, server-side applications, or command-line utilities.

Invoking Applet in java

- To invoke an **applet** in Java, there are two main ways to do it:

- Embedding the applet in an HTML file and running it through a web browser (with applet support, though modern browsers no longer support applets).

- Using the **Applet Viewer**, which is a command-line utility that comes with the Java Development Kit (JDK).



Example Applet Code

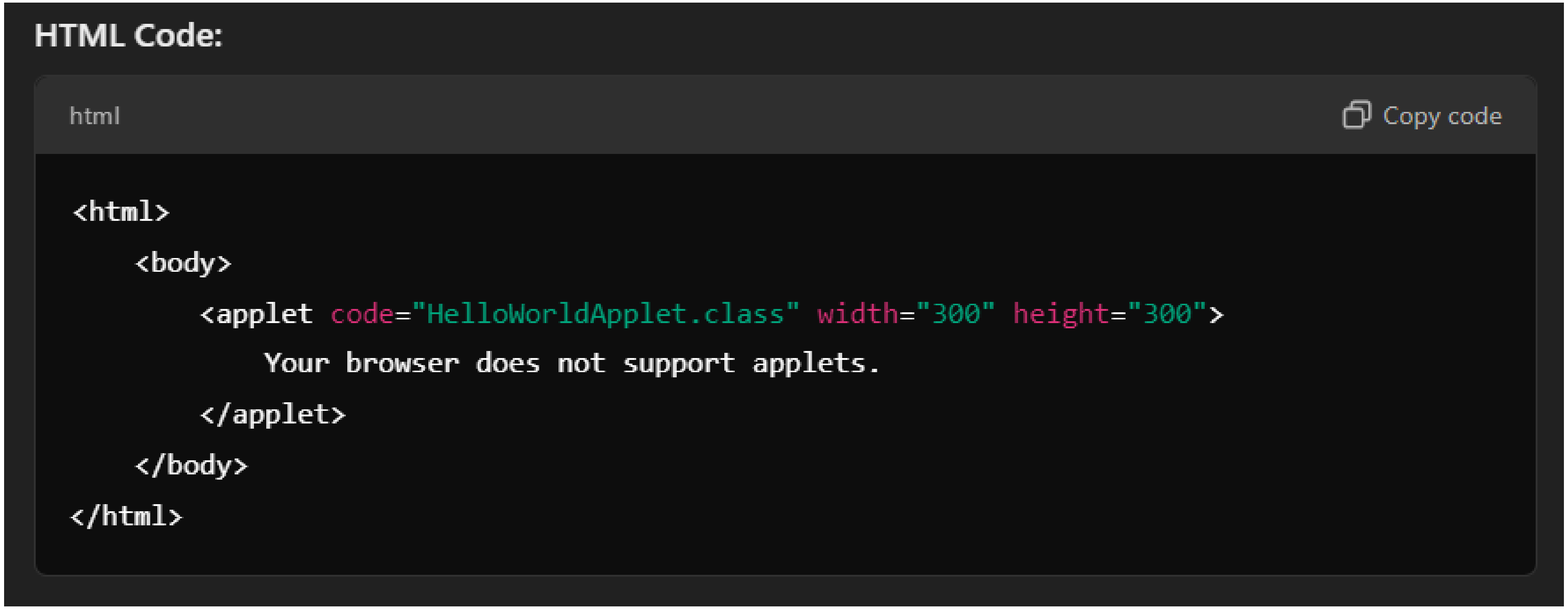
java

Copy code

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

public class HelloWorldApplet extends Applet {
    // This method will be called when the applet starts
    public void paint(Graphics g) {
        g.drawString("Hello, World!", 20, 20);
    }
}
```

- **1. Invoking Applet via HTML (Deprecated in Modern Browsers)**

- In the past, applets were embedded in HTML using the <applet> tag. Though modern browsers no longer support applets, this is how it was done:


HTML Code:

```html
<html>
    <body>
        <applet code="HelloWorldApplet.class" width="300" height="300">
            Your browser does not support applets.
        </applet>
    </body>
</html>
```

- **Steps:**
- Save the applet Java code (HelloWorldApplet.java).

- **Compile the Java code:**
- **javac HelloWorldApplet.java**

- This will generate a HelloWorldApplet.class file.

- Create an HTML file (e.g., HelloWorld.html) with the code above.

- Open the HTML file in a browser with support for applets (not possible in modern browsers, though some older browsers may still work).

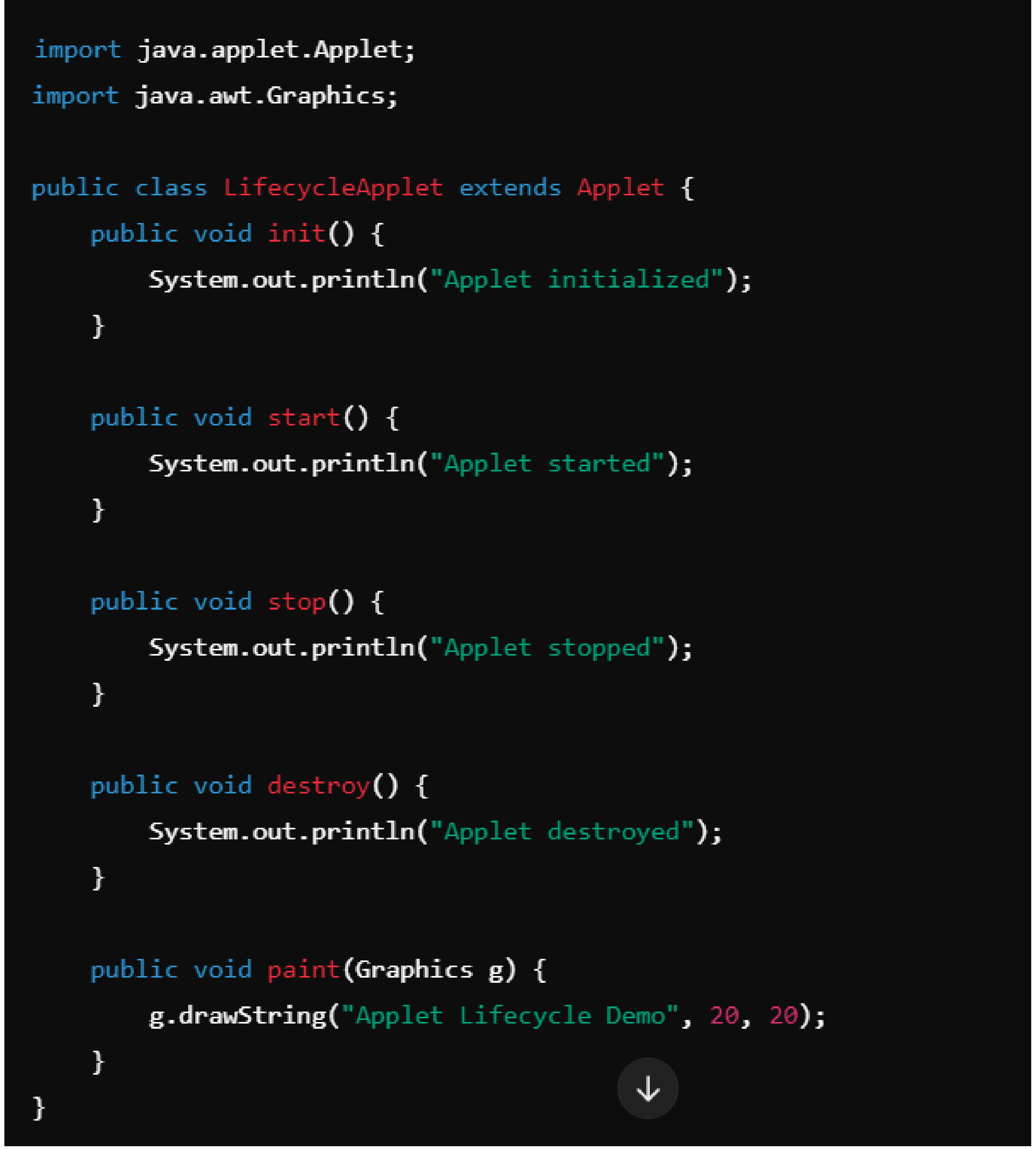
- **2. Invoking Applet Using Applet Viewer (Recommended):**

- You can use the **Applet Viewer** to run an applet without a web browser. This is useful for development and testing.
- **Steps:**
- Write and compile the applet code (HelloWorldApplet.java):
- **javac HelloWorldApplet.java**
- Create an HTML file (e.g., HelloWorld.html) as shown earlier.
- Use the **Applet Viewer** to run the applet from the command line:
- **appletviewer HelloWorld.html**
- This will open a window displaying the applet.

- **Applet Lifecycle Methods**

- When an applet is invoked, the following lifecycle methods are commonly used:

- init(): Initializes the applet (called once).
- start(): Called each time the applet is started or resumed.
- stop(): Called when the applet is stopped.
- destroy(): Called when the applet is being destroyed.



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

public class LifecycleApplet extends Applet {
    public void init() {
        System.out.println("Applet initialized");
    }

    public void start() {
        System.out.println("Applet started");
    }

    public void stop() {
        System.out.println("Applet stopped");
    }

    public void destroy() {
        System.out.println("Applet destroyed");
    }

    public void paint(Graphics g) {
        g.drawString("Applet Lifecycle Demo", 20, 20);
    }
}
```

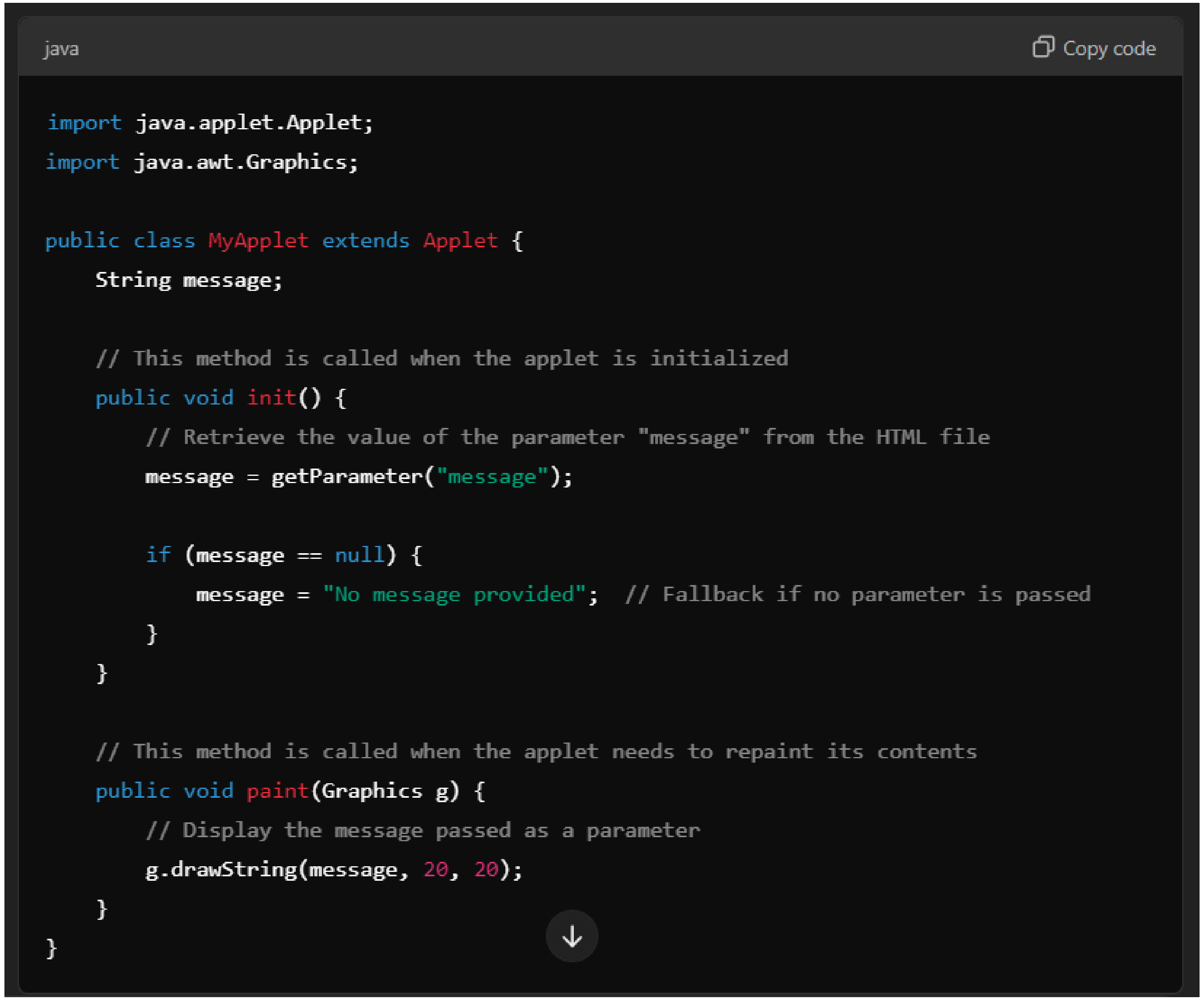
Passing parameters to Applet

- In Java, you can pass parameters to an applet using the HTML <param> tag. The applet can then read these parameters using the getParameter() method. These parameters are useful when you want to provide dynamic values to an applet at runtime, such as configurations or user preferences.

- **1. Steps to Pass Parameters to an Applet**

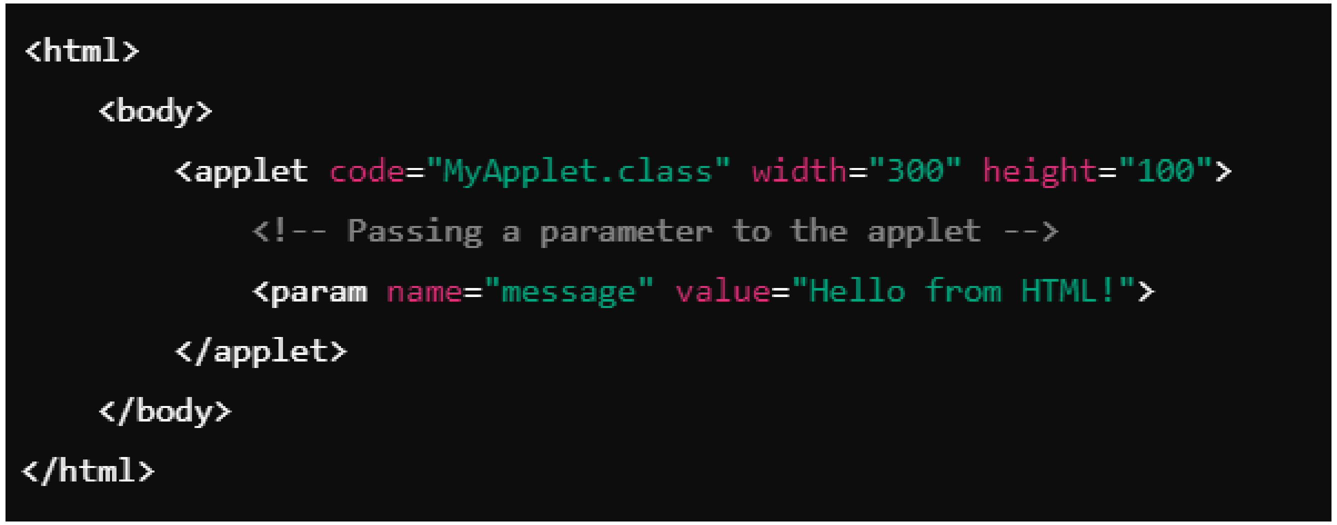
- **Declare the parameters in the HTML file** using the <param> tag within the <applet> tag.
- **Retrieve the parameters in the applet code** using the getParameter(String name) method.

- 2. **Example Code: Passing and Retrieving Parameters in Applet**



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

public class MyApplet extends Applet {
    String message;

    // This method is called when the applet is initialized
    public void init() {
        // Retrieve the value of the parameter "message" from the HTML file
        message = getParameter("message");

        if (message == null) {
            message = "No message provided";  // Fallback if no parameter is passed
        }
    }

    // This method is called when the applet needs to repaint its contents
    public void paint(Graphics g) {
        // Display the message passed as a parameter
        g.drawString(message, 20, 20);
    }
}
```

- **HTML Code: MyApplet.html**
- html



```html
<html>
    <body>
        <applet code="MyApplet.class" width="300" height="100">
            <!-- Passing a parameter to the applet -->
            <param name="message" value="Hello from HTML!">
        </applet>
    </body>
</html>
```

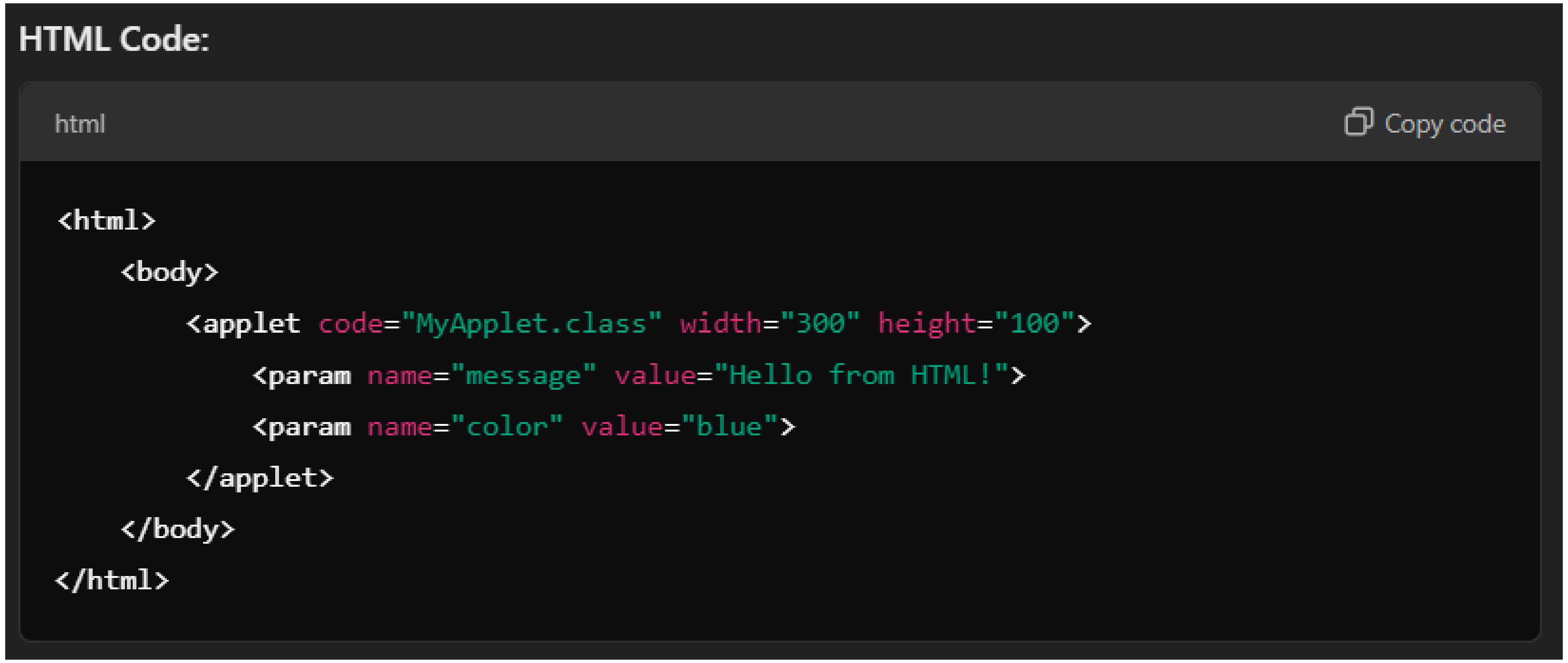
- **3. Explanation of the Example**
- **HTML File:**
  - The <param> tag inside the <applet> tag passes a parameter named "message" with the value "Hello from HTML!".
  - This value is made available to the applet via the getParameter() method.

- **Java Applet Code:**
  - The init() method is called when the applet is initialized. Here, the applet uses getParameter("message") to retrieve the value associated with the "message" parameter.
  - If no parameter is provided, a fallback message ("No message provided") is displayed.
  - The paint(Graphics g) method draws the message onto the applet's canvas.

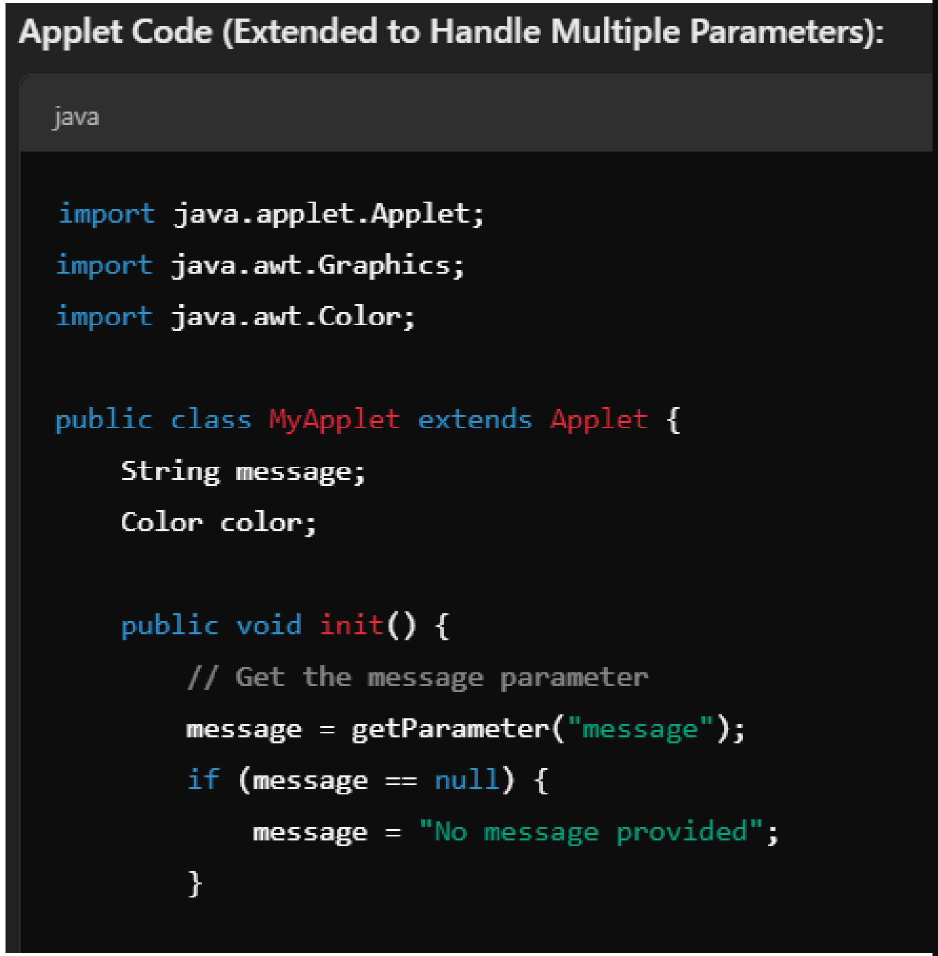
- **4. Running the Applet with Parameters**

- **Compile the Applet:**
  - Use javac to compile the Java applet file
  - **javac MyApplet.java**
- **Create the HTML file** with the <applet> and <param> tags as shown above.
- **Run the Applet:**
- You can use the **Applet Viewer** to test the applet locally:
- appletviewer MyApplet.html
- The Applet Viewer will launch the applet and display the message passed from the HTML file.

- **5. Handling Multiple Parameters**
- You can pass multiple parameters by using multiple <param> tags. The applet can retrieve each parameter individually using the getParameter() method.


HTML Code:

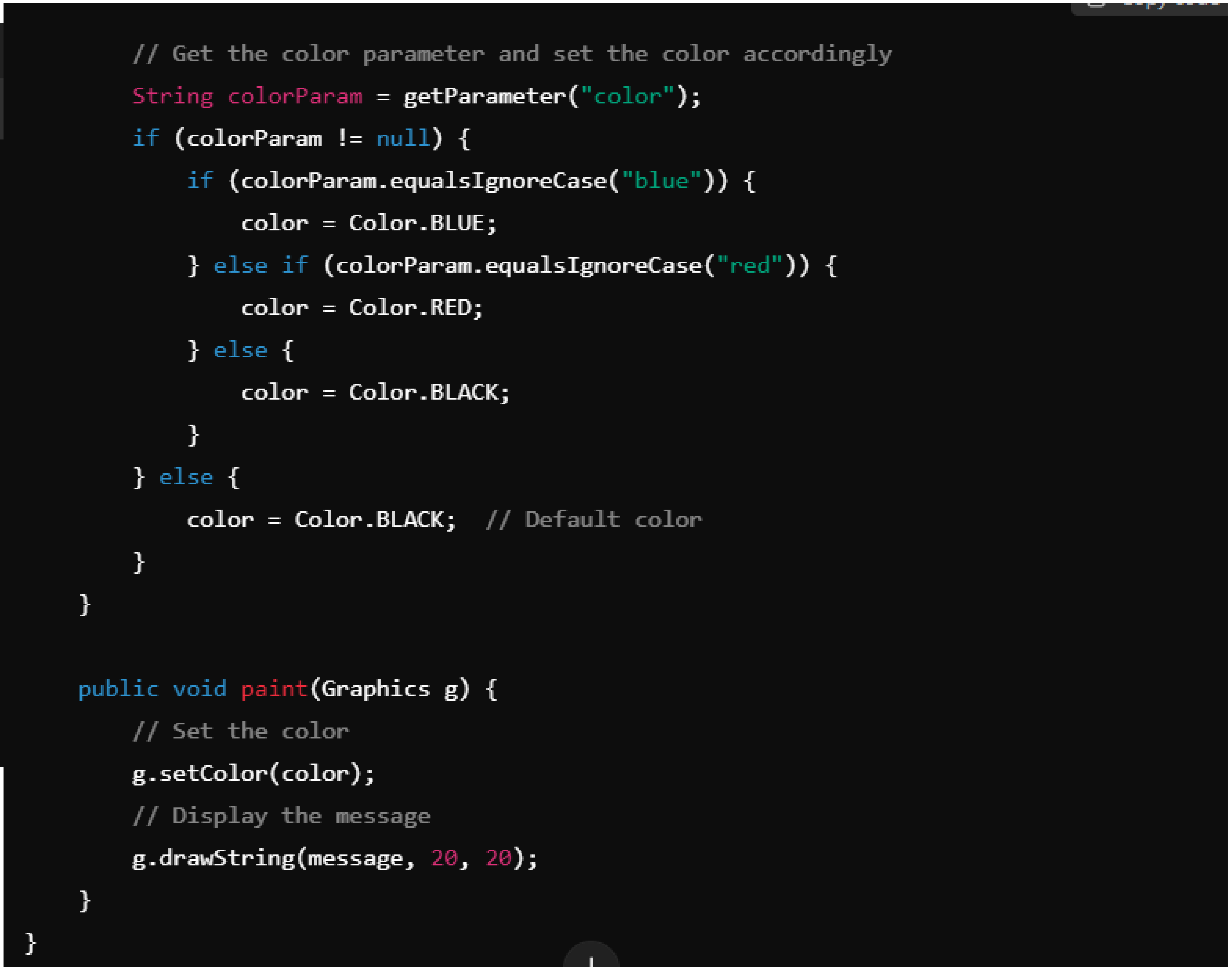
```html
<html>
    <body>
        <applet code="MyApplet.class" width="300" height="100">
            <param name="message" value="Hello from HTML!">
            <param name="color" value="blue">
        </applet>
    </body>
</html>
```



Applet Code (Extended to Handle Multiple Parameters):

java

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

public class MyApplet extends Applet {
    String message;
    Color color;

    public void init() {
        // Get the message parameter
        message = getParameter("message");
        if (message == null) {
            message = "No message provided";
        }
```



```java
        // Get the color parameter and set the color accordingly
        String colorParam = getParameter("color");
        if (colorParam != null) {
            if (colorParam.equalsIgnoreCase("blue")) {
                color = Color.BLUE;
            } else if (colorParam.equalsIgnoreCase("red")) {
                color = Color.RED;
            } else {
                color = Color.BLACK;
            }
        } else {
            color = Color.BLACK;  // Default color
        }
    }

    public void paint(Graphics g) {
        // Set the color
        g.setColor(color);
        // Display the message
        g.drawString(message, 20, 20);
    }
}
```

- **6. Applet Parameters Summary**

- **HTML <param> tag**: Used to pass key-value pairs to the applet.
  - Syntax: <param name="paramName" value="paramValue">

- **getParameter(String name)**: Retrieves the parameter value inside the applet. If the parameter is not found, it returns null.
- Multiple parameters can be passed and retrieved by calling getParameter() for each one.

- **7. Important Notes**

- **Security**: Applets run in a restricted sandbox, which limits their access to system resources for security reasons. This means applets cannot easily read/write files or interact with system resources unless specifically granted permissions.

- **Obsolescence**: Applets are deprecated starting from Java 9 and are completely removed in Java 11. Applet support has been discontinued in most modern browsers due to security concerns. Testing applets is mainly done using the **Applet Viewer** tool.

Abstract Window Toolkit (AWT)- Component Class: Container, Panel, LayoutManager

- Java AWT or Abstract Window Toolkit is an API used for developing GUI(Graphic User Interfaces) or Window-Based Applications in Java. Java AWT is part of the Java Foundation Classes (JFC) that provides a way to build platform-independent graphical applications.

- In this AWT tutorial, you will learn the basics of the AWT, including how to create windows, buttons, labels, and text fields. We will also learn how to add event listeners to components so that they can respond to user input.

- **Java AWT Basics**

- Java AWT (Abstract Window Toolkit) is an API used to create **Graphical User Interface (GUI**) or Windows-based Java programs and Java AWT components are platform-dependent, which means they are shown in accordance with the operating system's view. AWT is heavyweight, which means that its components consume resources from the underlying operating system (OS). The java.awt package contains AWT API classes such as *TextField*, *Label*, *TextArea*, *RadioButton*, *CheckBox*, *Choice*, *Lis*t, and so on.

- ***Points about Java AWT components***
- *i. Components of AWT are heavy and platform dependent*
  ii. AWT has less control as the result can differ because of components are platform dependent.

- **Why AWT is Platform Independent?**

- The Java AWT utilizes the native platform subroutine to create API components such as TextField, CheckBox, and buttons. This results in a different visual format for these components on different platforms such as Windows, MAC OS, and Unix. The reason for this is that each platform has a distinct view of its native components. AWT directly calls this native subroutine to create the components, resulting in an AWT application resembling a Windows application on Windows OS, and a Mac application on the MAC OS. In simpler terms, the AWT application's appearance adapts to the platform it is running on.

- Understanding how to use BeanFactory is vital for optimizing your application's performance. To delve deeper into Java programming, the **Java Backend** course offers a comprehensive look at various Java concepts and their applications.

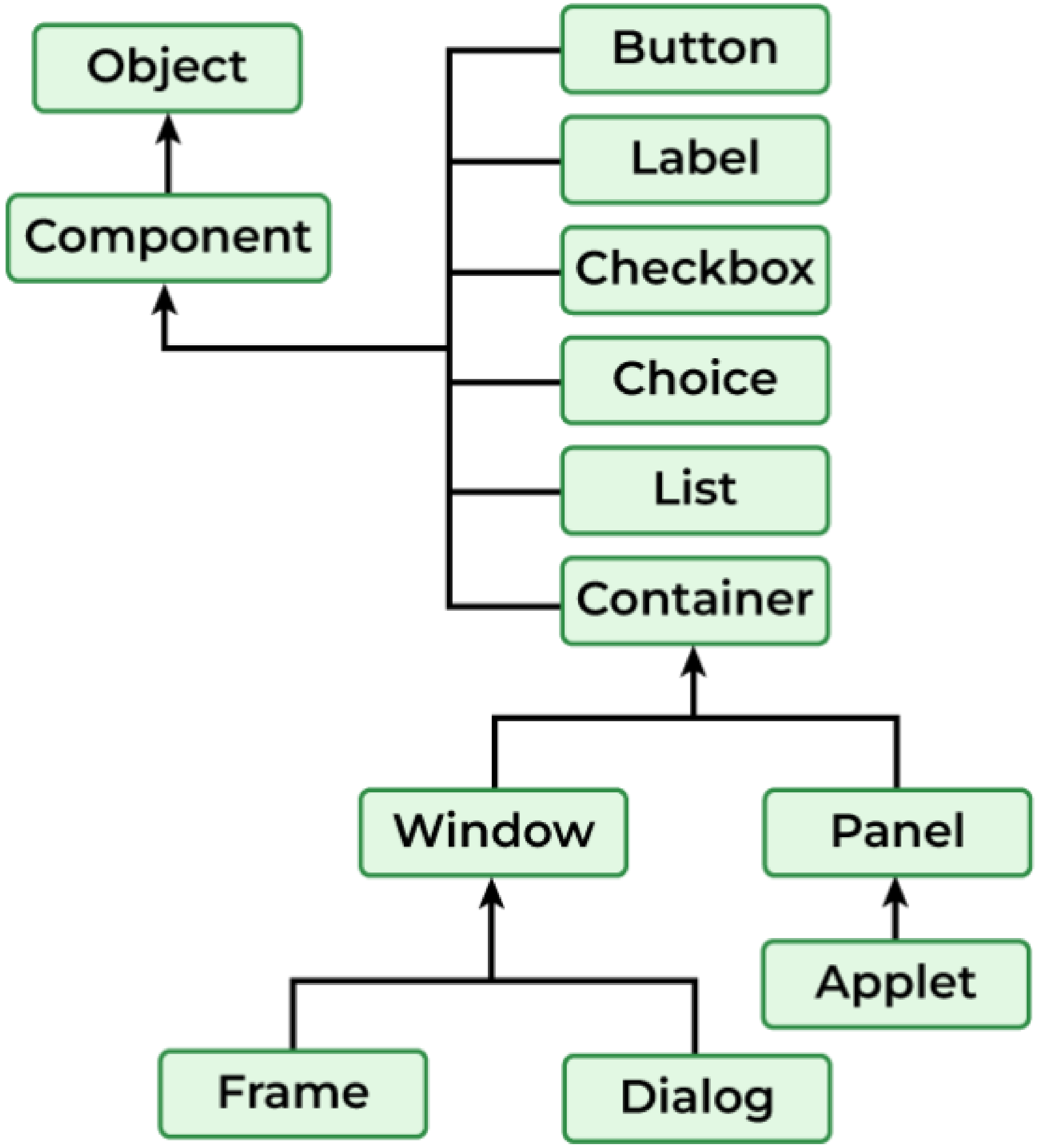
- AWT is platform independent even after the AWT components are platform dependent because of the points mentioned below:

- **1. JVM (Java Virtual Machine):**
- As Java Virtual Machine is platform dependent

- **2. Abstract APIs:**
- AWT provides an abstract layer for GUI. Java applications interact with AWT through Abstract API which are platform independent. Abstract API allows Java to isolate platform-specific details, making code portable across different systems.

- **3. Platform-Independent Libraries:**
- The Libraries of AWT are written in Java which they are totally platform-independent. Because of this, it ensures that AWT functionality remains consistent across different environments.

- **Java AWT Hierarchy**



- **Components:** AWT provides various components such as buttons, labels, text fields, checkboxes, etc used for creating GUI elements for Java Applications.

- **Containers:** AWT provides containers like panels, frames, and dialogues to organize and group components in the Application.

- **Layout Managers:** Layout Managers are responsible for arranging data in the containers some of the layout managers are BorderLayout, FlowLayout, etc.

- **Event Handling:** AWT allows the user to handle the events like mouse clicks, key presses, etc. using event listeners and adapters.

- **Graphics and Drawing:** It is the feature of AWT that helps to draw shapes, insert images and write text in the components of a Java Application.

- **Types of Containers in Java AWT**
- There are four types of containers in Java AWT:

- **Window:** Window is a top-level container that represents a graphical window or dialog box. The Window class extends the Container class, which means it can contain other components, such as buttons, labels, and text fields.

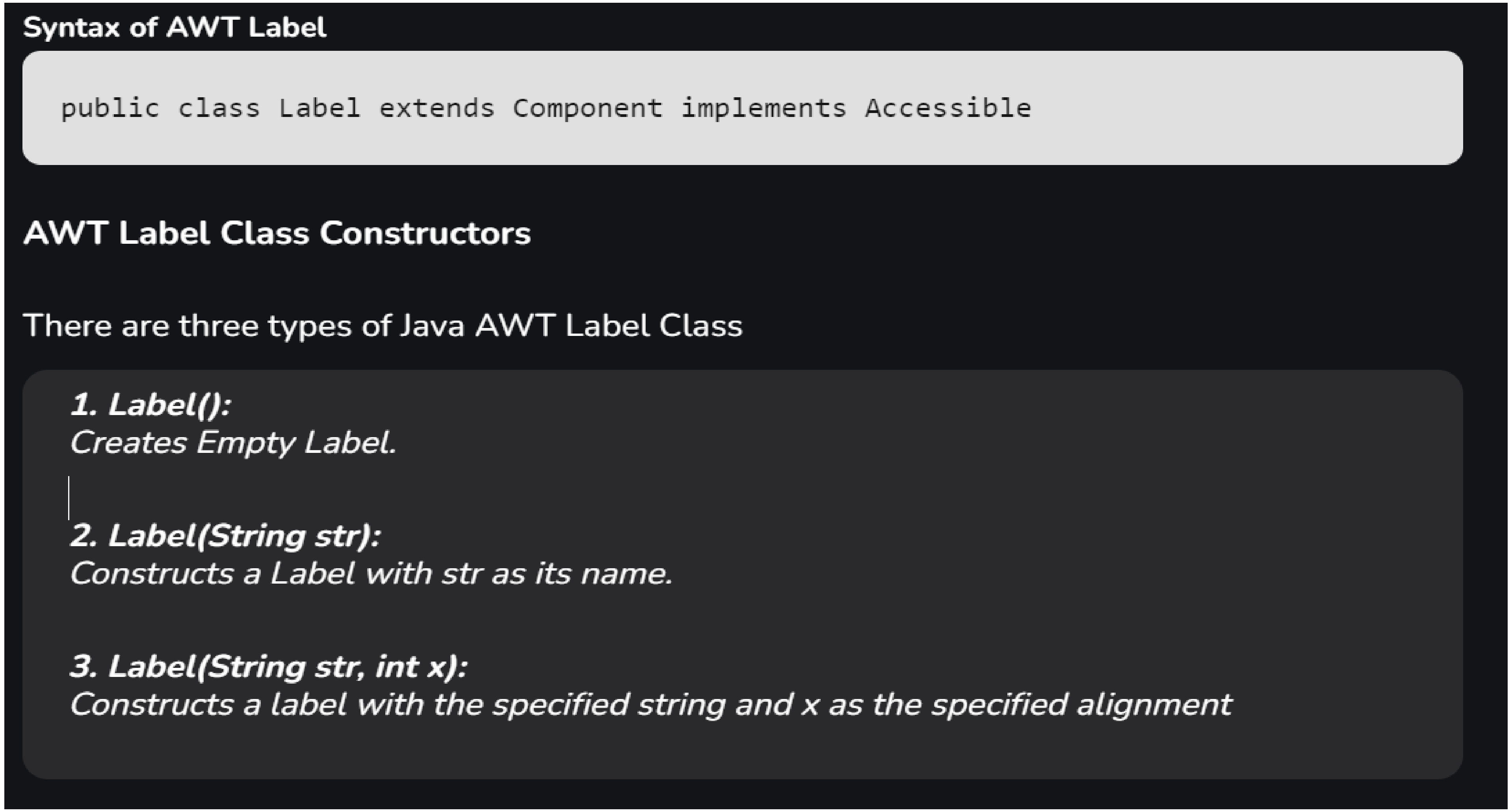
- **Panel:** Panel is a container class in Java. It is a lightweight container that can be used for grouping other components together within a window or a frame.

- **Frame**: The Frame is the container that contains the title bar and border and can have menu bars.

- **Dialog:** A dialog box is a temporary window an application creates to retrieve user input.

UI Controls:- Lables, TextFields, CheckBoxes, RadioButtons, ChoiceList, ChoiceMenu, List

- **1. Java AWT Label**



Syntax of AWT Label

```
public class Label extends Component implements Accessible
```

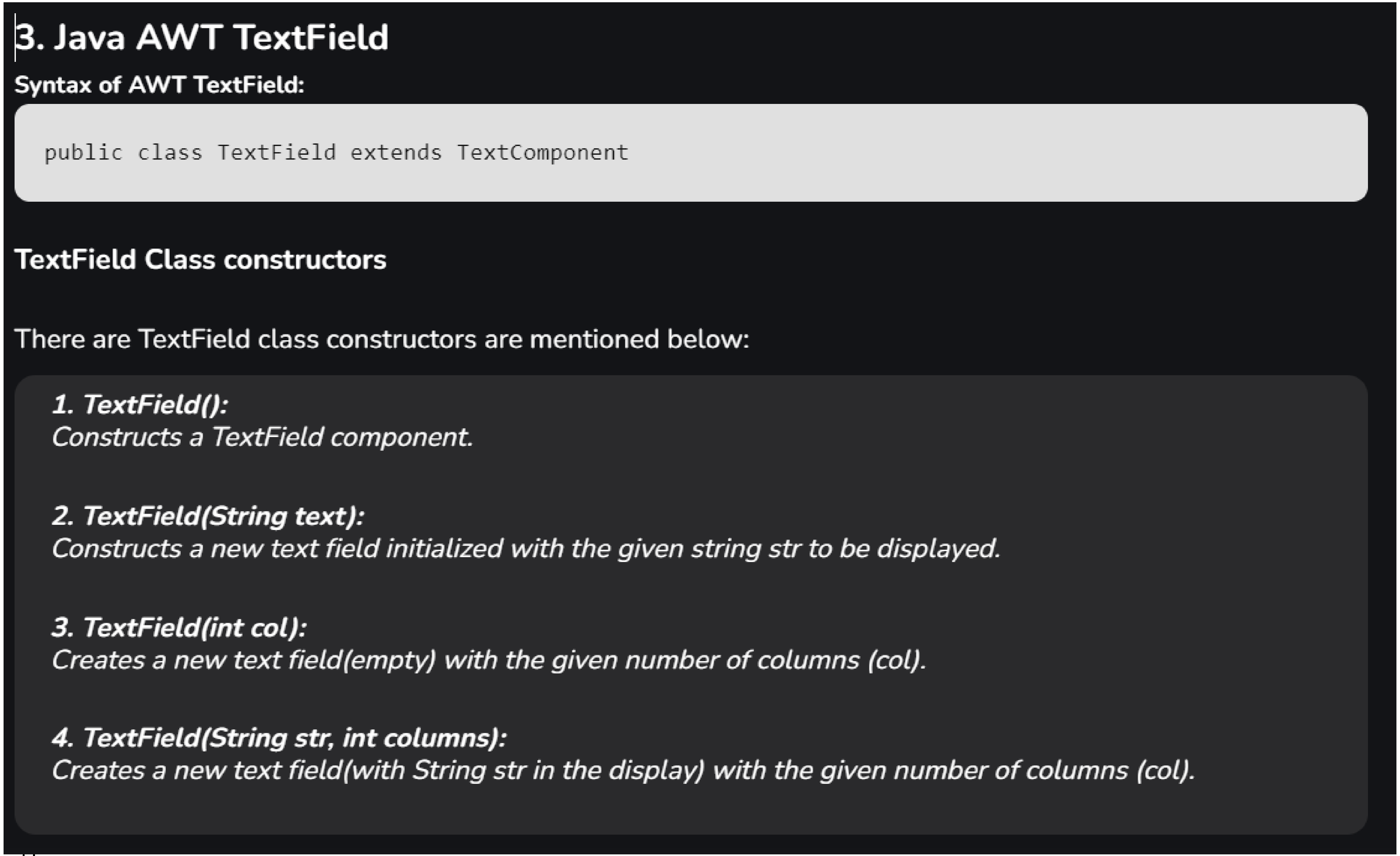
AWT Label Class Constructors

There are three types of Java AWT Label Class

1. Label():
Creates Empty Label.

2. Label(String str):
Constructs a Label with str as its name.

3. Label(String str, int x):
Constructs a label with the specified string and x as the specified alignment



3. Java AWT TextField

Syntax of AWT TextField:

```
public class TextField extends TextComponent
```

TextField Class constructors

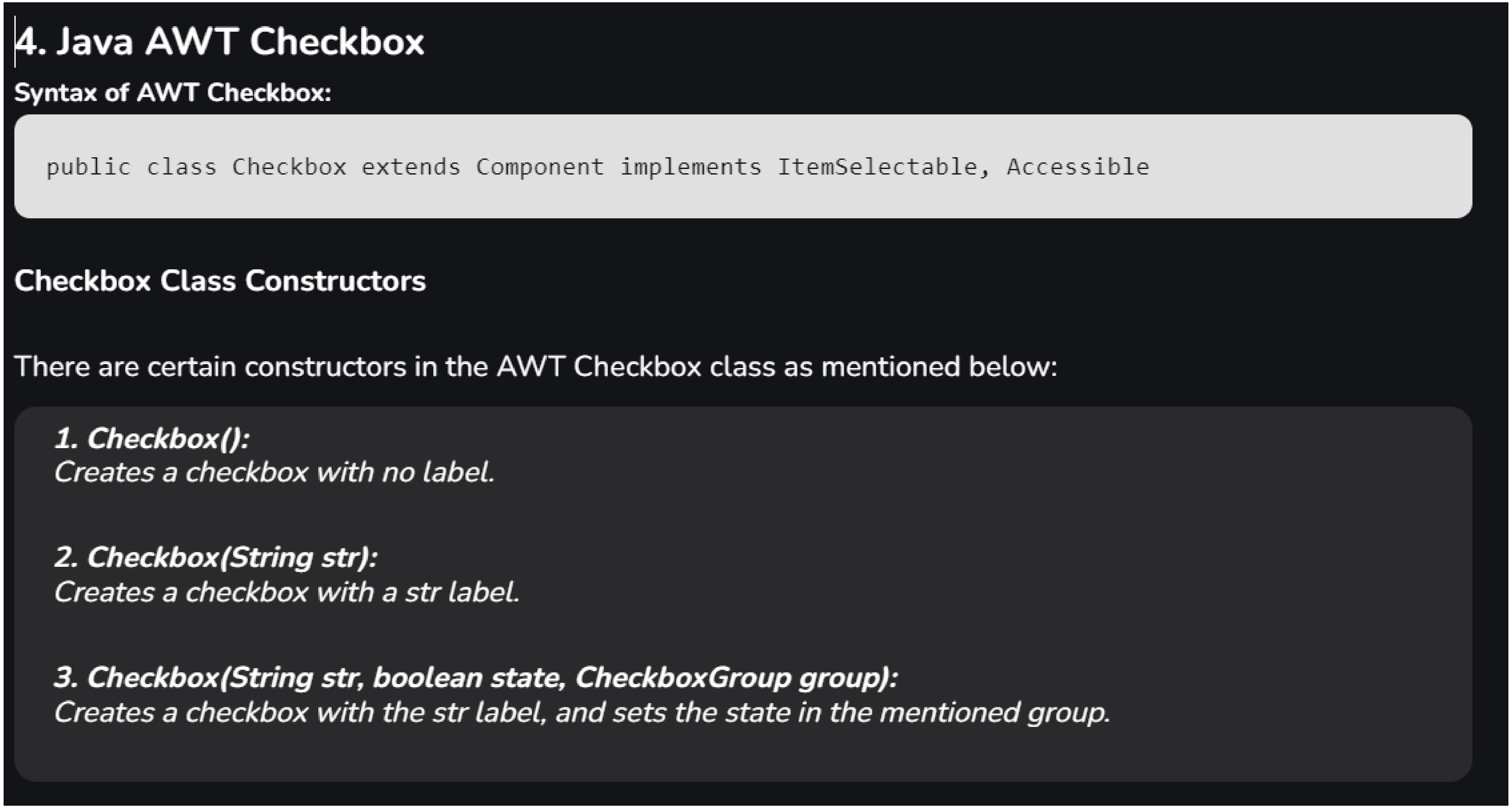
There are TextField class constructors are mentioned below:

1. TextField():
Constructs a TextField component.

2. TextField(String text):
Constructs a new text field initialized with the given string str to be displayed.

3. TextField(int col):
Creates a new text field(empty) with the given number of columns (col).

4. TextField(String str, int columns):
Creates a new text field(with String str in the display) with the given number of columns (col).



4. Java AWT Checkbox

Syntax of AWT Checkbox:

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

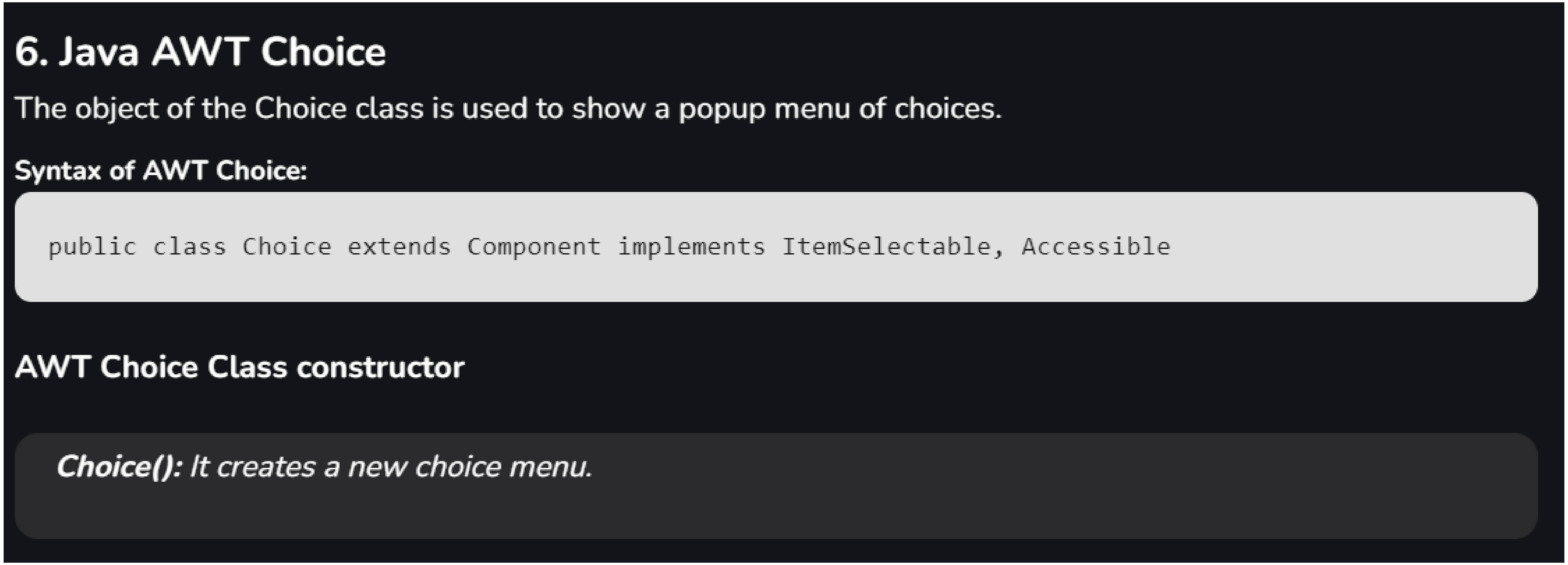
Checkbox Class Constructors

There are certain constructors in the AWT Checkbox class as mentioned below:

1. Checkbox():
Creates a checkbox with no label.

2. Checkbox(String str):
Creates a checkbox with a str label.

3. Checkbox(String str, boolean state, CheckboxGroup group):
Creates a checkbox with the str label, and sets the state in the mentioned group.



6. Java AWT Choice

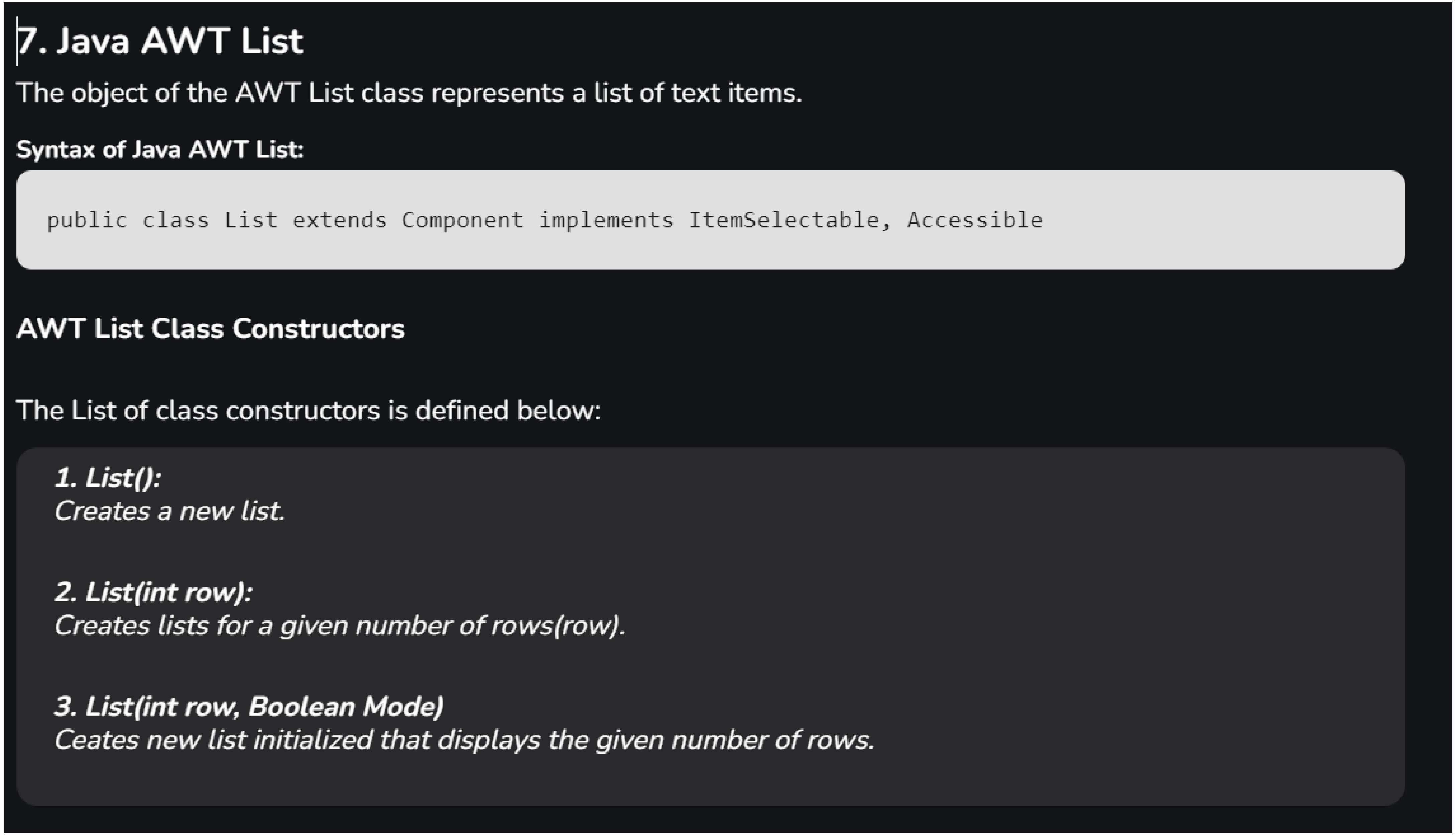
The object of the Choice class is used to show a popup menu of choices.

Syntax of AWT Choice:

```
public class Choice extends Component implements ItemSelectable, Accessible
```

AWT Choice Class constructor

Choice(): *It creates a new choice menu.*



7. Java AWT List

The object of the AWT List class represents a list of text items.

Syntax of Java AWT List:

```
public class List extends Component implements ItemSelectable, Accessible
```

AWT List Class Constructors

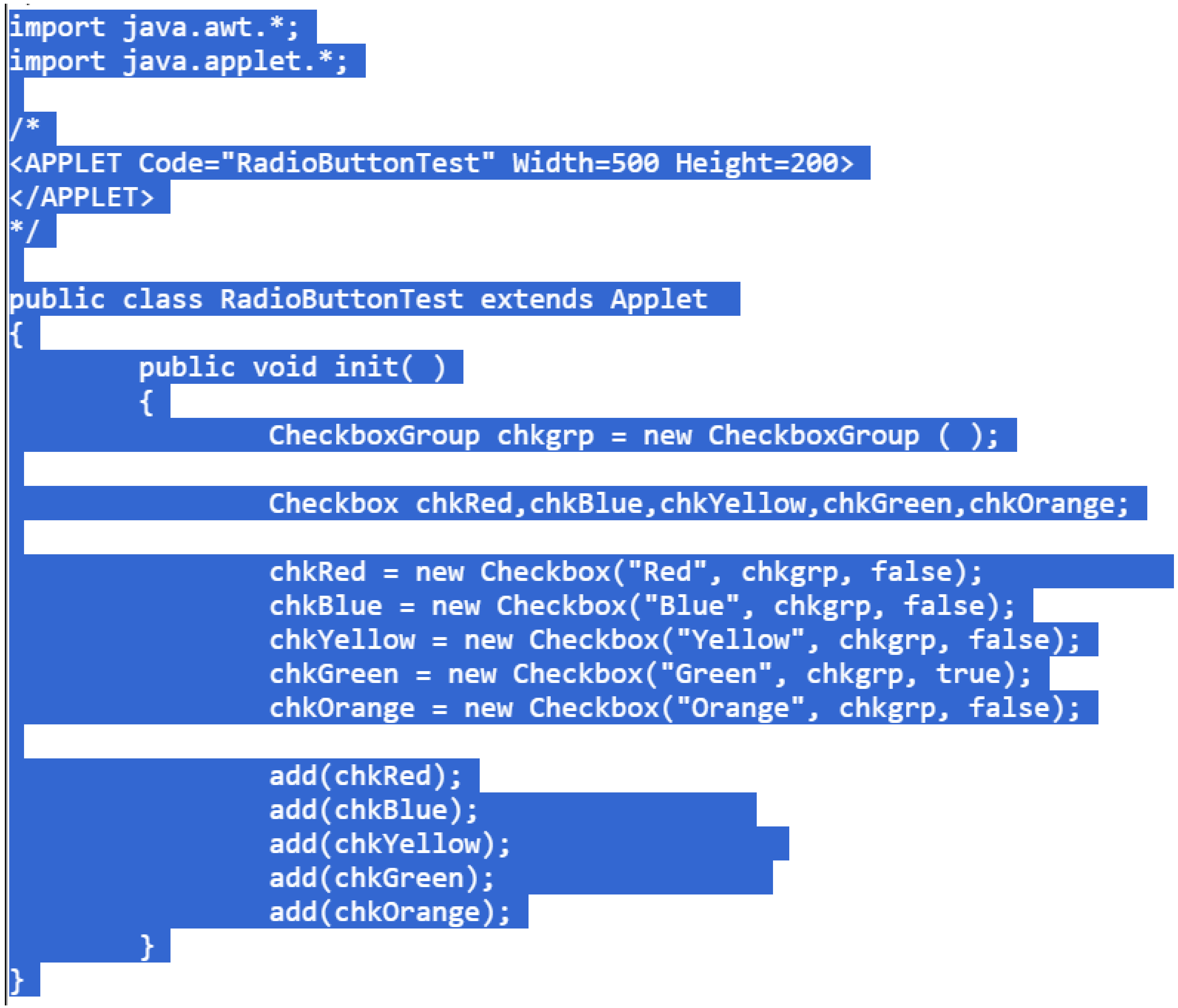
The List of class constructors is defined below:

1. List():
Creates a new list.

2. List(int row):
Creates lists for a given number of rows(row).

3. List(int row, Boolean Mode)
Ceates new list initialized that displays the given number of rows.

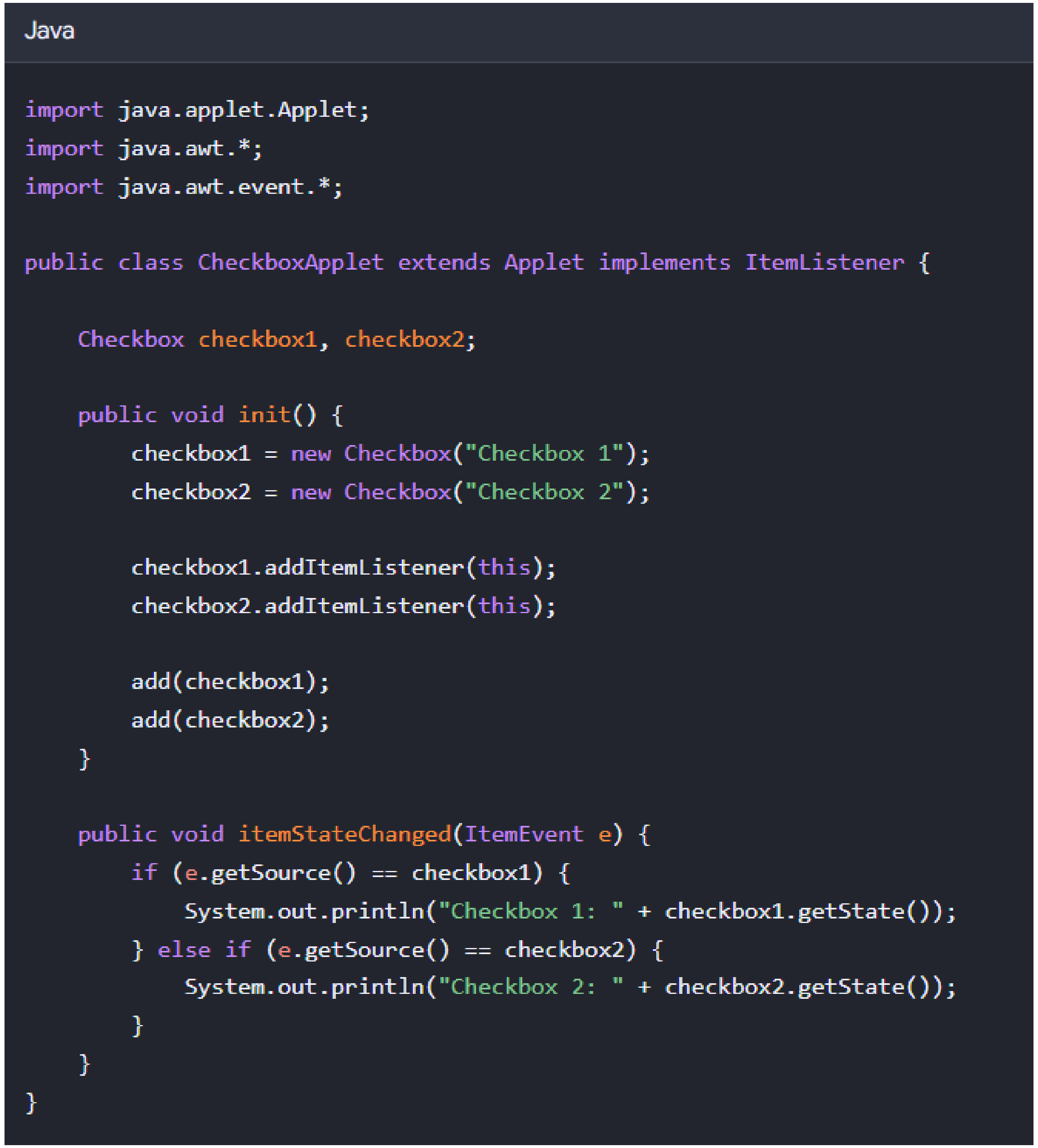
- 8.Radio Button:



```
import java.awt.*;
import java.applet.*;

/*
<APPLET Code="RadioButtonTest" Width=500 Height=200>
</APPLET>
*/

public class RadioButtonTest extends Applet
{
        public void init( )
        {
                CheckboxGroup chkgrp = new CheckboxGroup ( );

                Checkbox chkRed,chkBlue,chkYellow,chkGreen,chkOrange;

                chkRed = new Checkbox("Red", chkgrp, false);
                chkBlue = new Checkbox("Blue", chkgrp, false);
                chkYellow = new Checkbox("Yellow", chkgrp, false);
                chkGreen = new Checkbox("Green", chkgrp, true);
                chkOrange = new Checkbox("Orange", chkgrp, false);

                add(chkRed);
                add(chkBlue);
                add(chkYellow);
                add(chkGreen);
                add(chkOrange);
        }
}
```

CheckBoxes

- While applets are outdated and no longer supported in modern browsers, here's an example of how you could create checkboxes using an applet in Java (for historical reference):


Java

```java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CheckboxApplet extends Applet implements ItemListener {

    Checkbox checkbox1, checkbox2;

    public void init() {
        checkbox1 = new Checkbox("Checkbox 1");
        checkbox2 = new Checkbox("Checkbox 2");

        checkbox1.addItemListener(this);
        checkbox2.addItemListener(this);

        add(checkbox1);
        add(checkbox2);
    }

    public void itemStateChanged(ItemEvent e) {
        if (e.getSource() == checkbox1) {
            System.out.println("Checkbox 1: " + checkbox1.getState());
        } else if (e.getSource() == checkbox2) {
            System.out.println("Checkbox 2: " + checkbox2.getState());
        }
    }
}
```

- **Explanation:**
- Import necessary classes: Import the Applet, Checkbox, ItemListener, ItemEvent, and awt classes.
- Extend the Applet class: Create a class that extends the Applet class.
- Create Checkbox objects: Create Checkbox objects with labels.
- Implement ItemListener: Implement the ItemListener interface to handle checkbox state changes.
- Add Checkboxes to Applet: Use the add() method to add the checkboxes to the applet.
- Handle ItemStateChanged: In the itemStateChanged() method, check which checkbox triggered the event and perform actions based on its state.

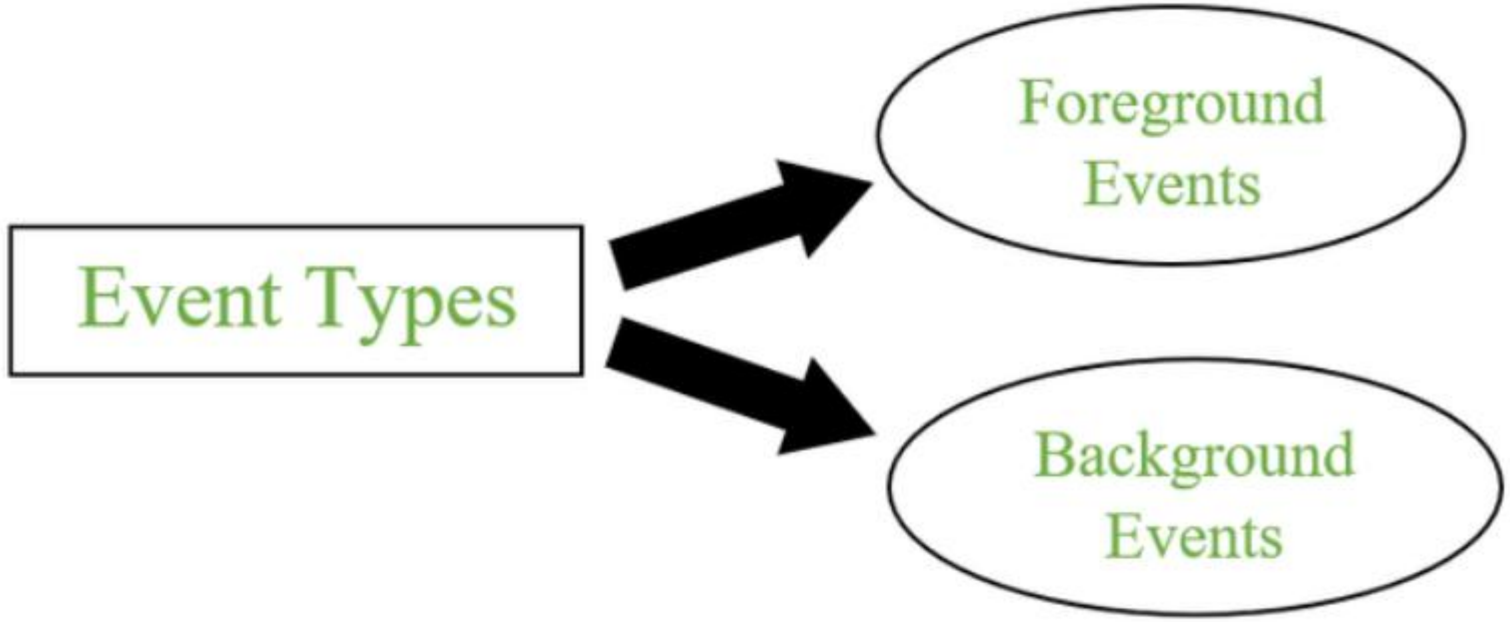
- **Java AWT | Choice Class**
- Choice class is part of Java Abstract Window Toolkit(AWT). The Choice class presents a pop- up menu for the user, the user may select an item from the popup menu. The selected item appears on the top. The Choice class inherits the Component.
- **Constructor for the Choice class**
- **Choice()** : creates an new empty choice menu .

```
import java.awt.*;
import java.applet.*;
/*
<APPLET Code="ChoiceTest" Width=500 Height=200>
</APPLET>
*/
public class ChoiceTest extends Applet
{
        public void init( )
        {
                Choice ch = new Choice();
                ch.add("apples");
                ch.add("oranges");
                ch.add("strawberies");
                ch.add("bananas");
                add(ch );
        }
}
```

Event Handling 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



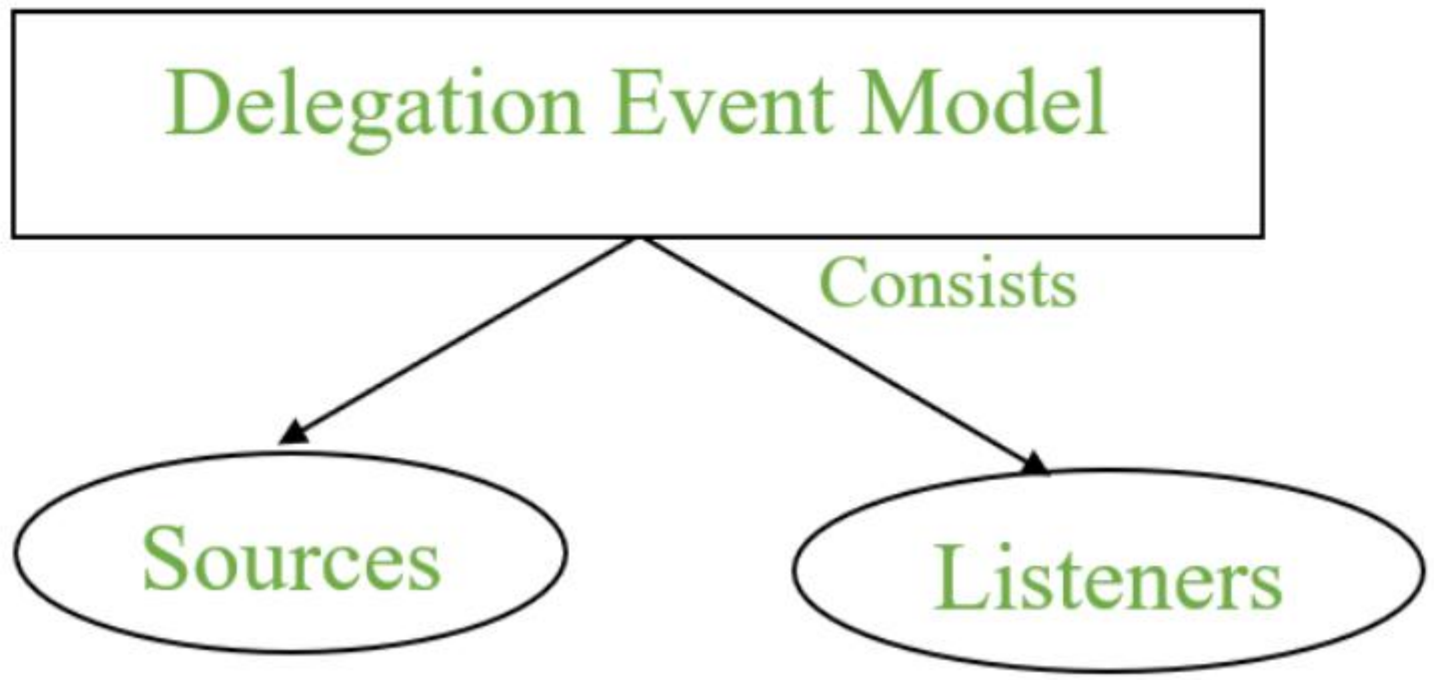
- **1. Foreground Events**
- Foreground events are the events that require user interaction to generate, i.e., foreground events are generated due to interaction by the user on components in Graphic User Interface (**GUI**). Interactions are nothing but clicking on a button, scrolling the scroll bar, cursor moments, etc.

- **2. Background Events**
- Events that don't require interactions of users to generate are known as background events. Examples of these events are operating system failures/interrupts, operation completion, etc.

- **Event Handling**
- It is a mechanism to **control the events** and to **decide what should happen after an 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 for handling events.

- To perform Event Handling, we need to register the source with the listener

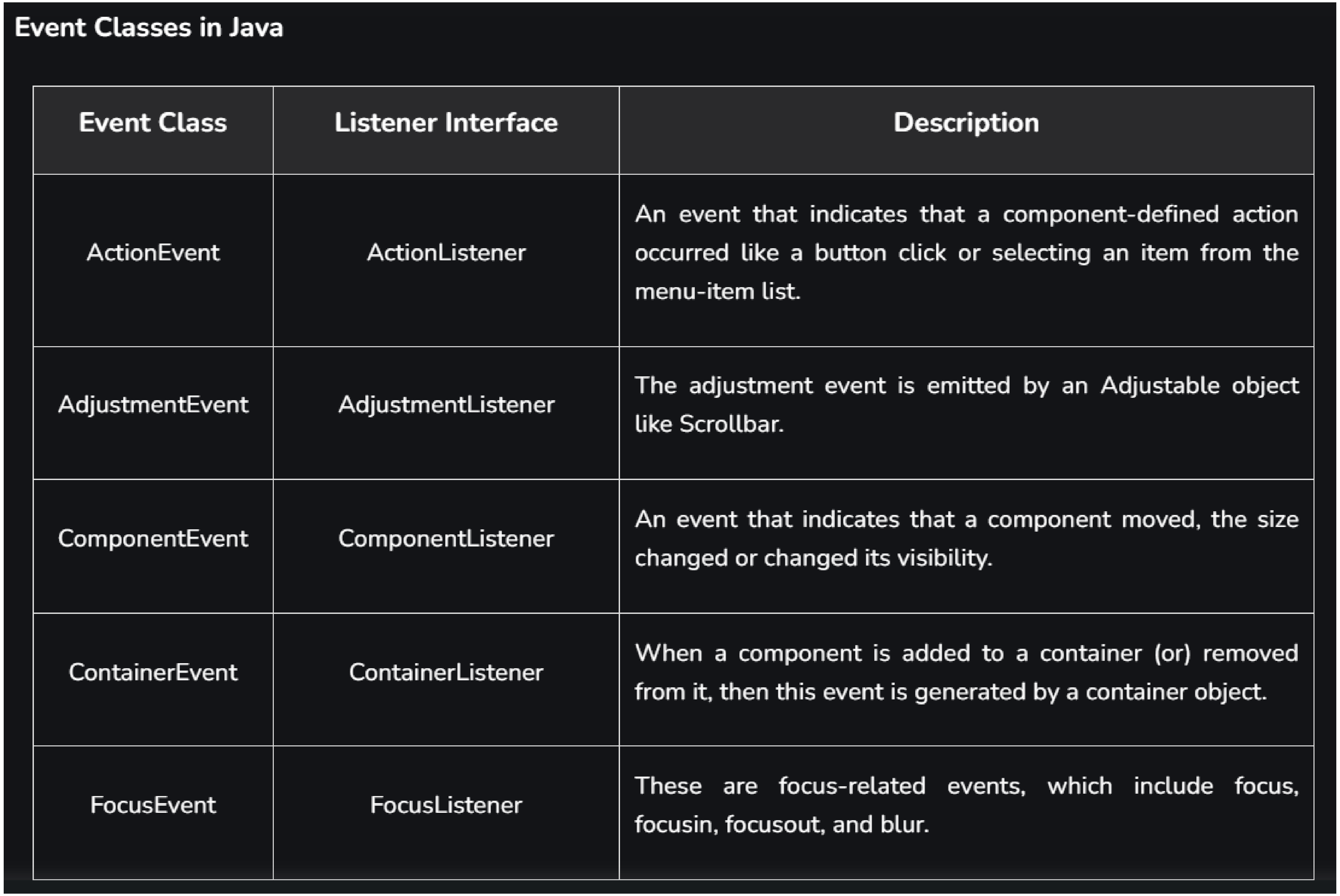
- **Registering the Source With Listener**
- Different Classes provide different registration methods.

- **Syntax:**
- addTypeListener()

- where Type represents the type of event.

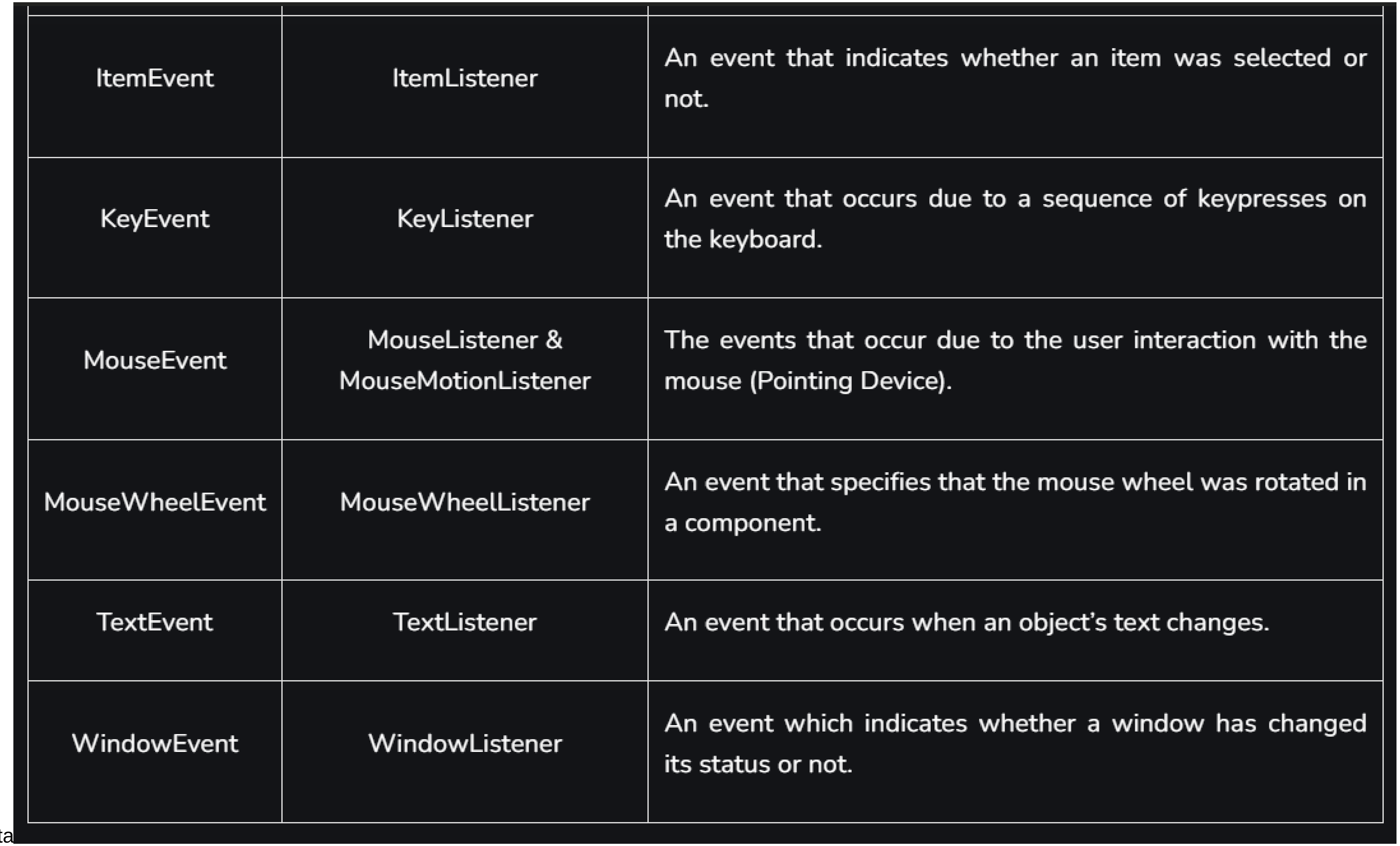
- **Example 1:** For **KeyEvent** we use *addKeyListener()* to register.

- **Example 2:**that For **ActionEvent** we use *addActionListener()* to register.

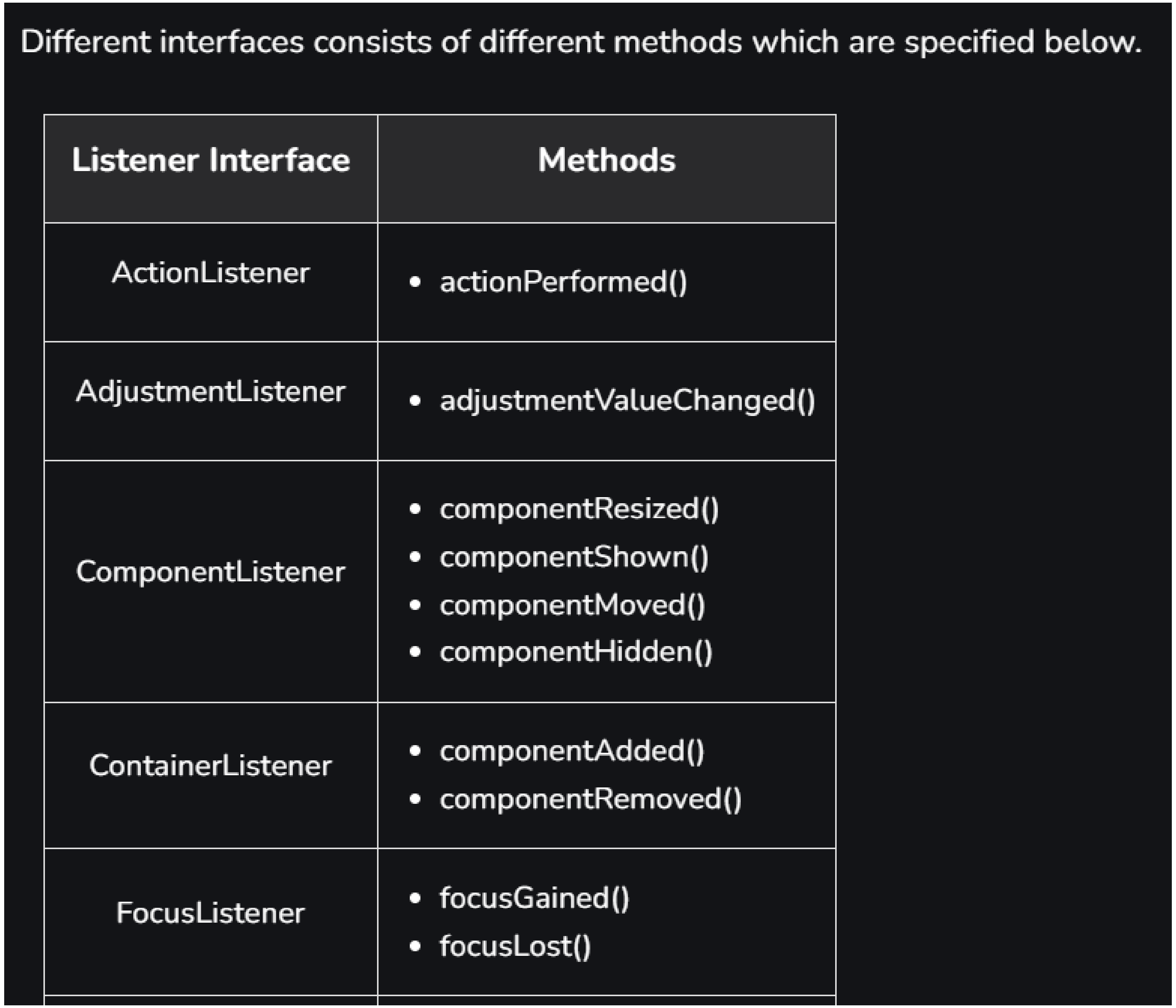


Event Classes in Java

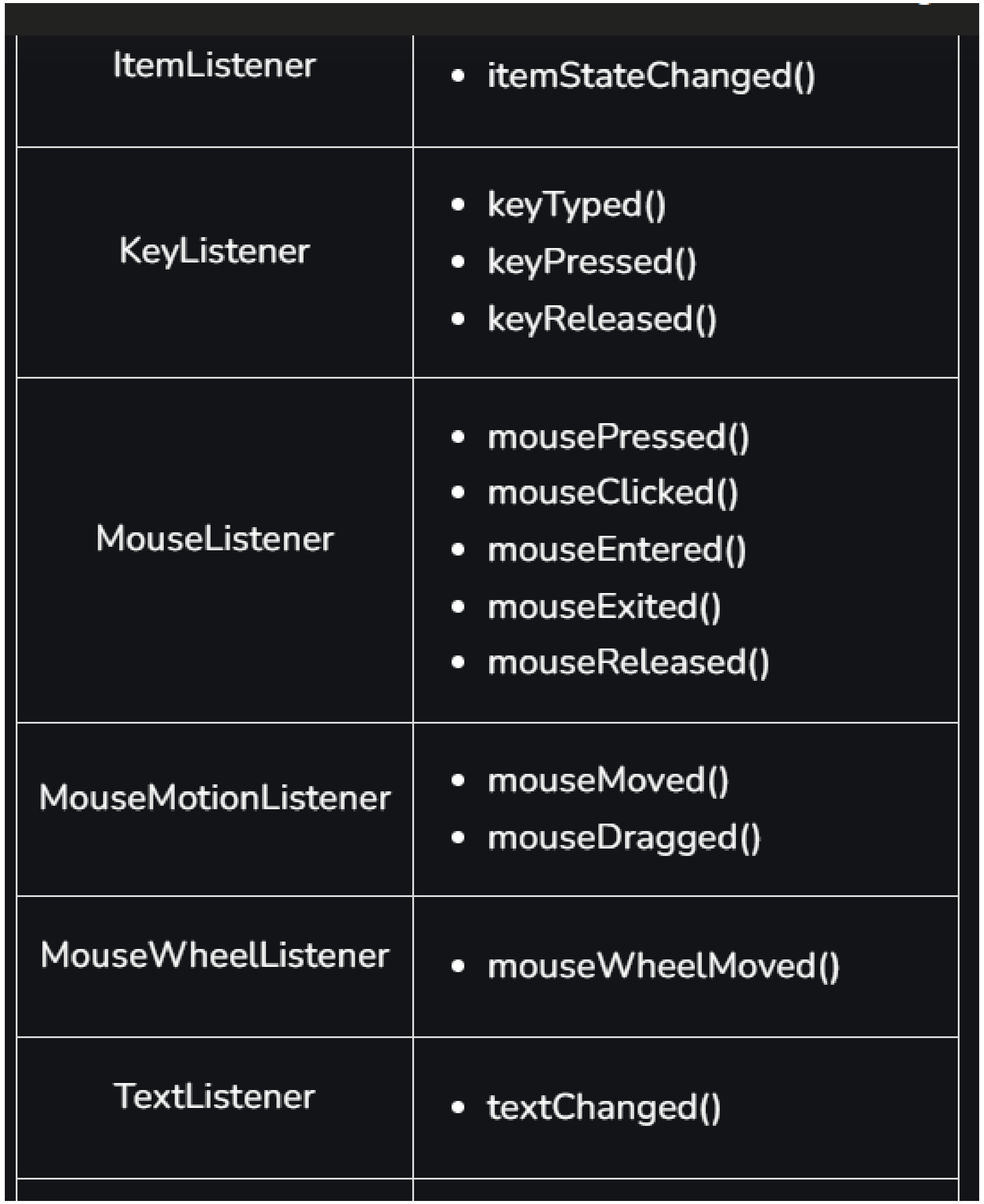
Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.



ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.


Different interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	• actionPerformed()
AdjustmentListener	• adjustmentValueChanged()
ComponentListener	• componentResized() • componentShown() • componentMoved() • componentHidden()
ContainerListener	• componentAdded() • componentRemoved()
FocusListener	• focusGained() • focusLost()

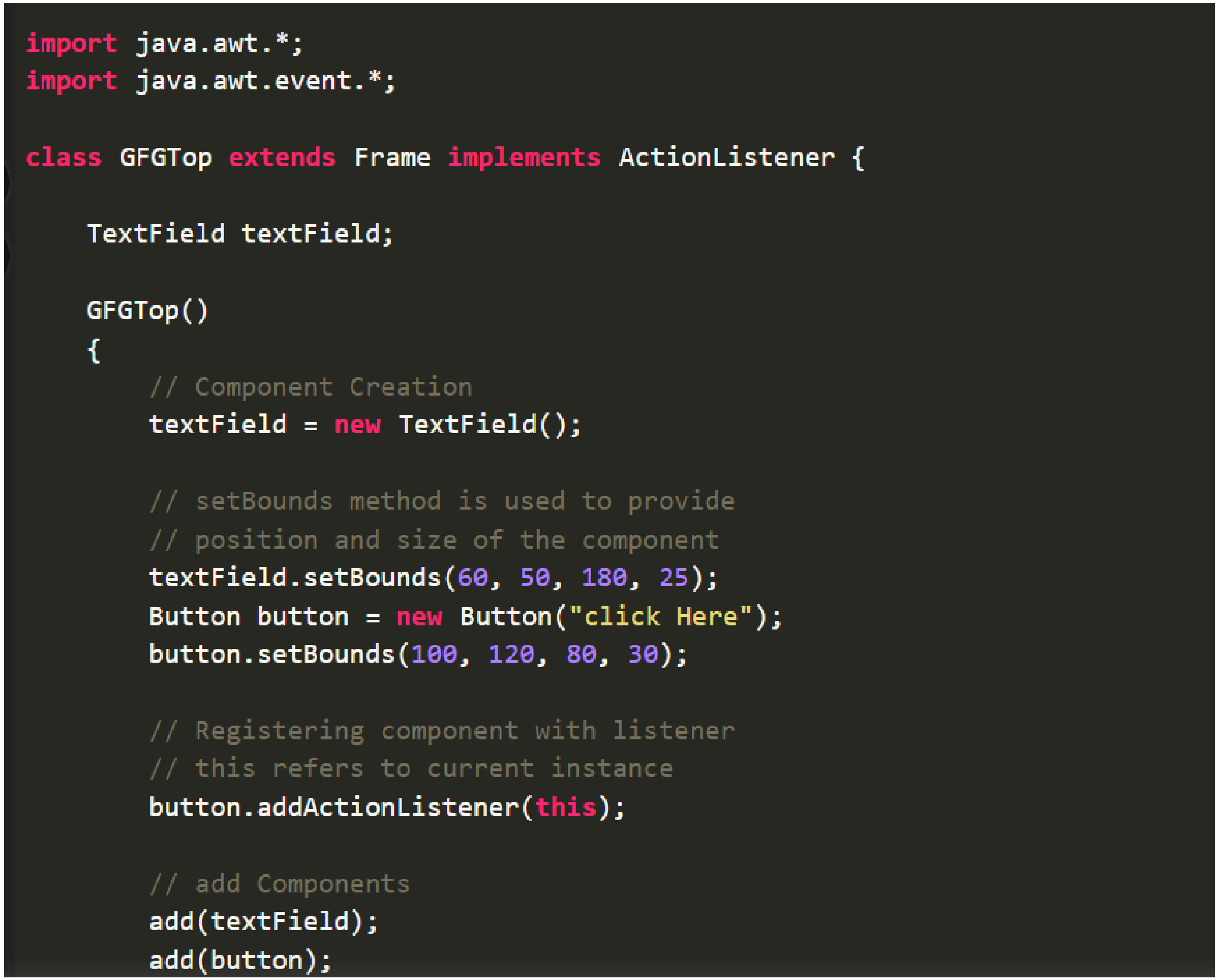


ItemListener	• itemStateChanged()
KeyListener	• keyTyped() • keyPressed() • keyReleased()
MouseListener	• mousePressed() • mouseClicked() • mouseEntered() • mouseExited() • mouseReleased()
MouseMotionListener	• mouseMoved() • mouseDragged()
MouseWheelListener	• mouseWheelMoved()
TextListener	• textChanged()

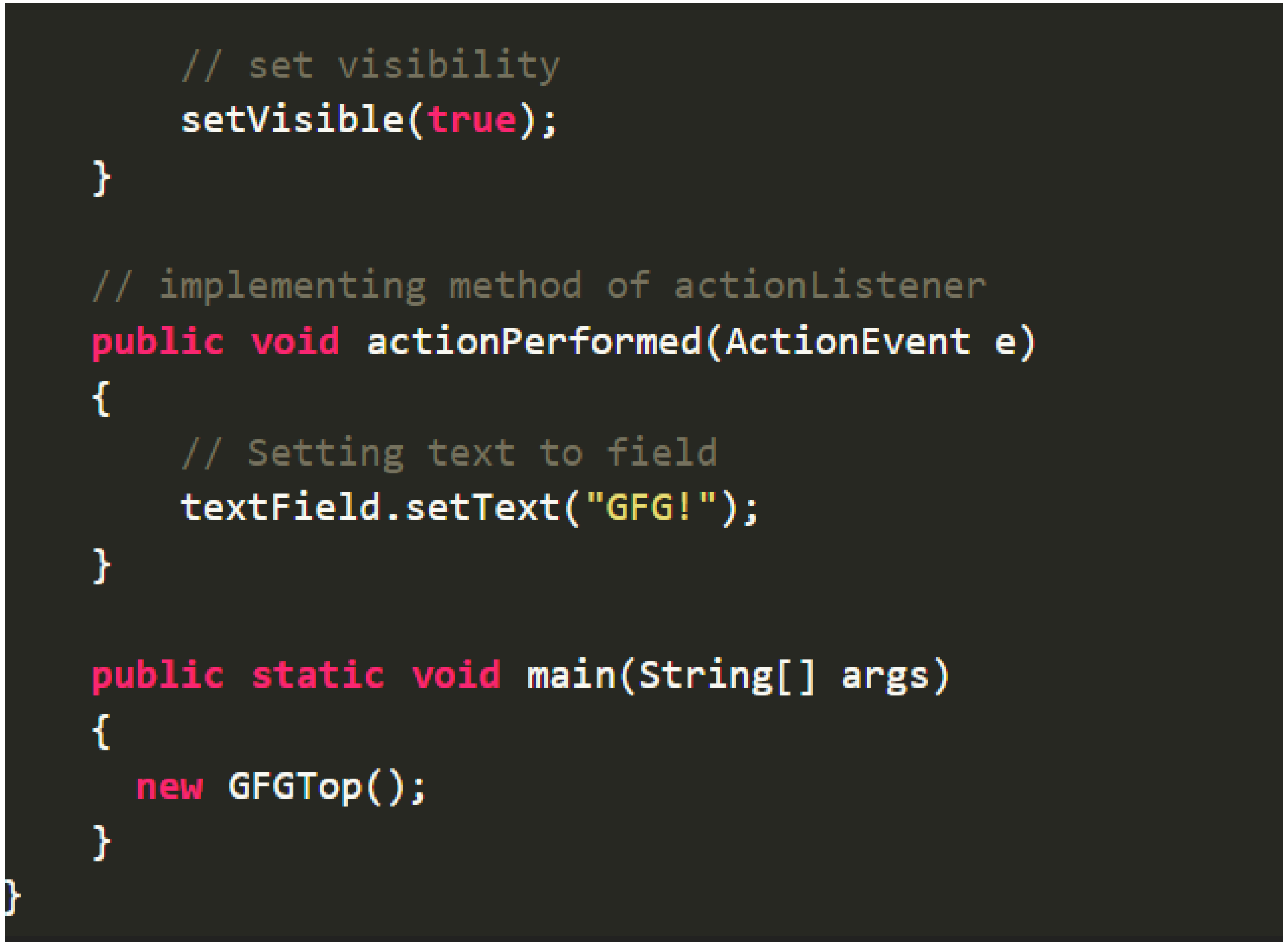
- **Flow of Event Handling**

- User Interaction with a component is required to generate an event.
- The object of the respective event class is created automatically after event generation, and it holds all information of the event source.
- The newly created object is passed to the methods of the registered listener.
- The method executes and returns the result.

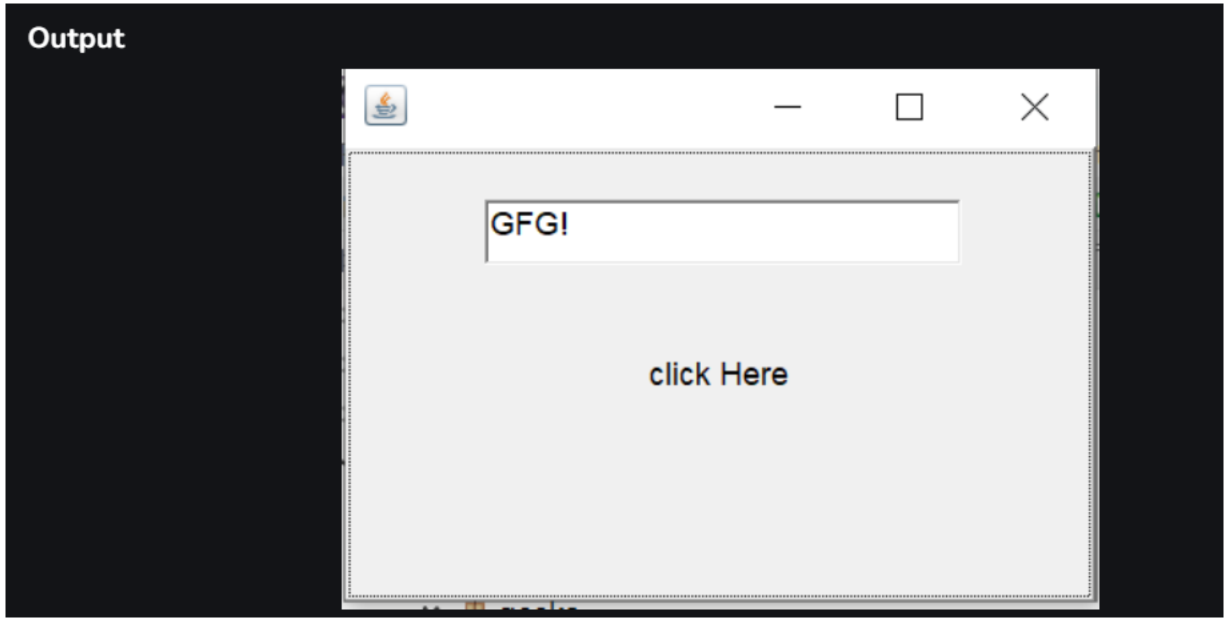
- **Code-Approaches**
- The three approaches for performing event handling are by placing the event handling code in one of the below-specified places.
- Within Class
- Other Class
- Anonymous Class



```java
import java.awt.*;
import java.awt.event.*;

class GFGTop extends Frame implements ActionListener {

    TextField textField;

    GFGTop()
    {
        // Component Creation
        textField = new TextField();

        // setBounds method is used to provide
        // position and size of the component
        textField.setBounds(60, 50, 180, 25);
        Button button = new Button("click Here");
        button.setBounds(100, 120, 80, 30);

        // Registering component with listener
        // this refers to current instance
        button.addActionListener(this);

        // add Components
        add(textField);
        add(button);
```



```
        // set visibility
        setVisible(true);
    }

    // implementing method of actionListener
    public void actionPerformed(ActionEvent e)
    {
        // Setting text to field
        textField.setText("GFG!");
    }

    public static void main(String[] args)
    {
      new GFGTop();
    }
}
```



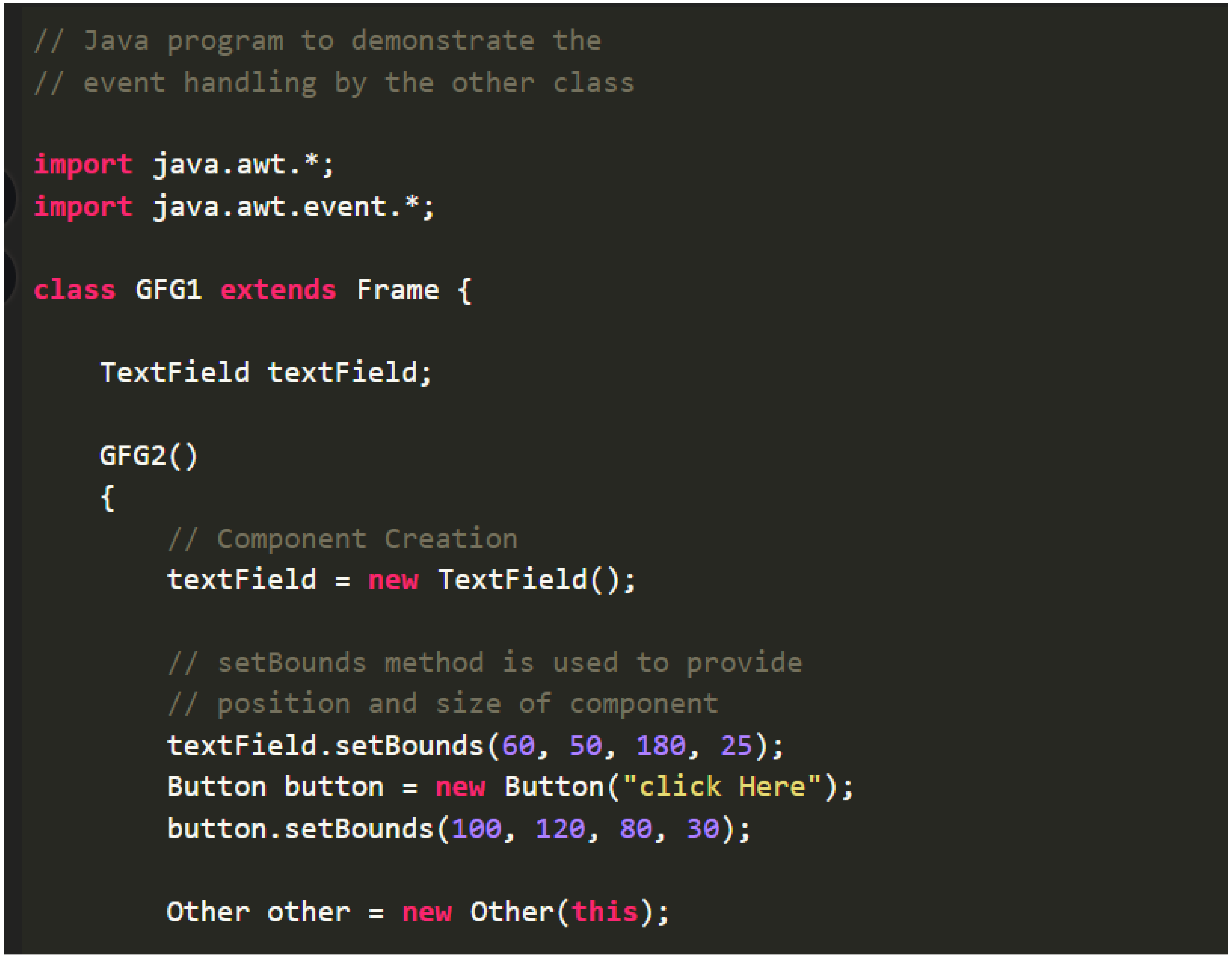
Explanation

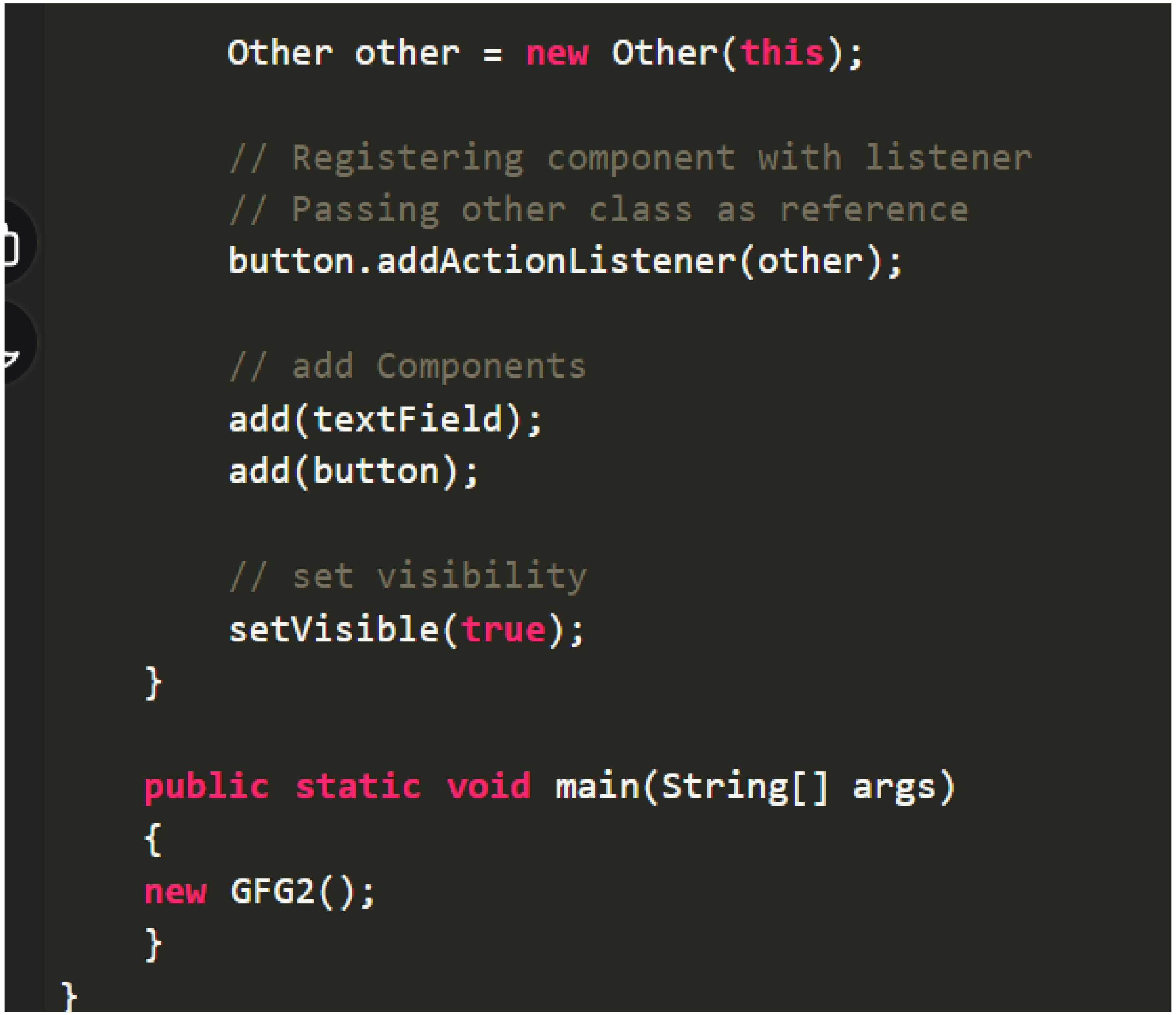
Firstly extend the class with the applet and implement the respective listener.

Create Text-Field and Button components.

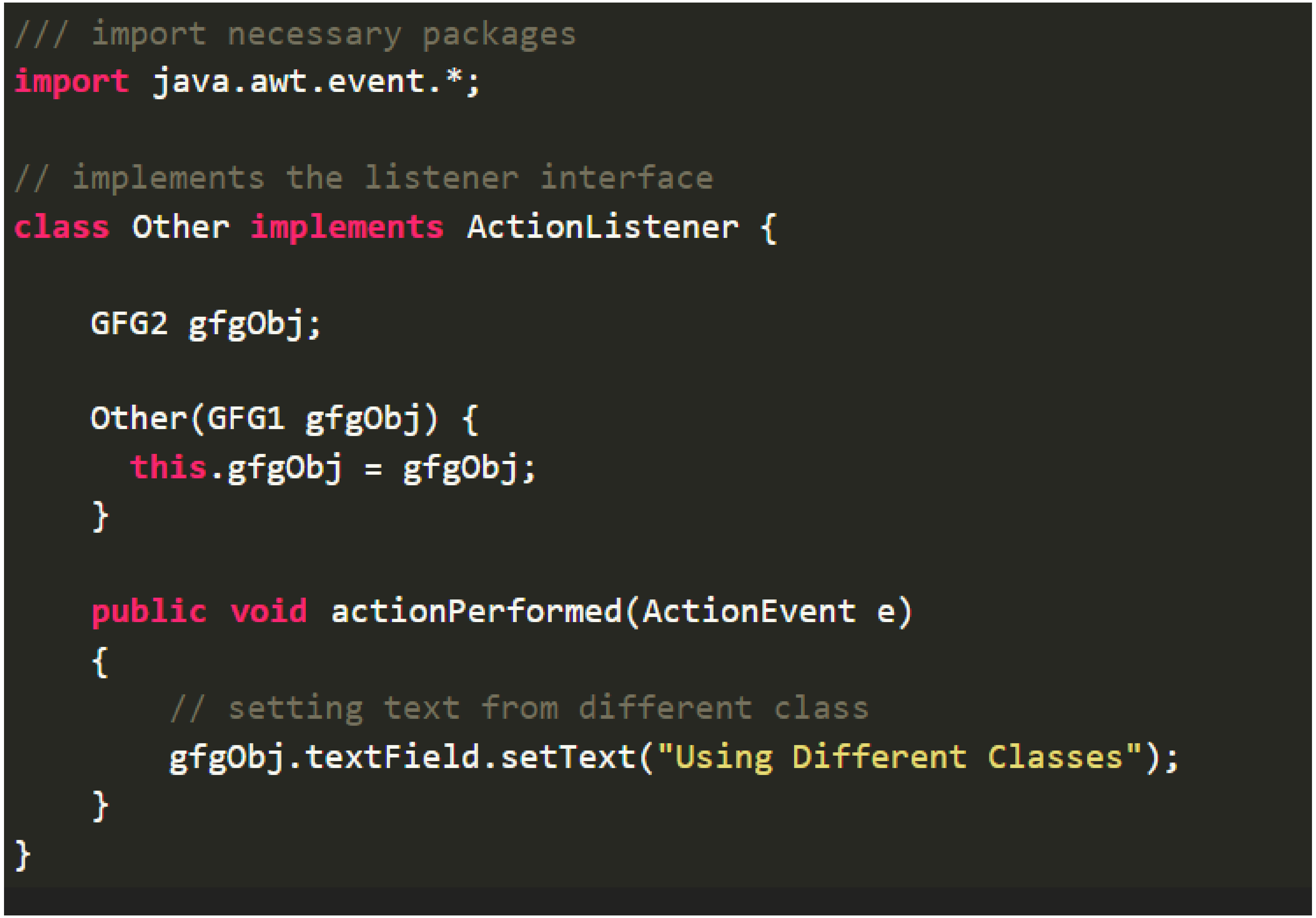
Registered the button component with respective event. i.e. ActionEvent by addActionListener().

In the end, implement the abstract method.

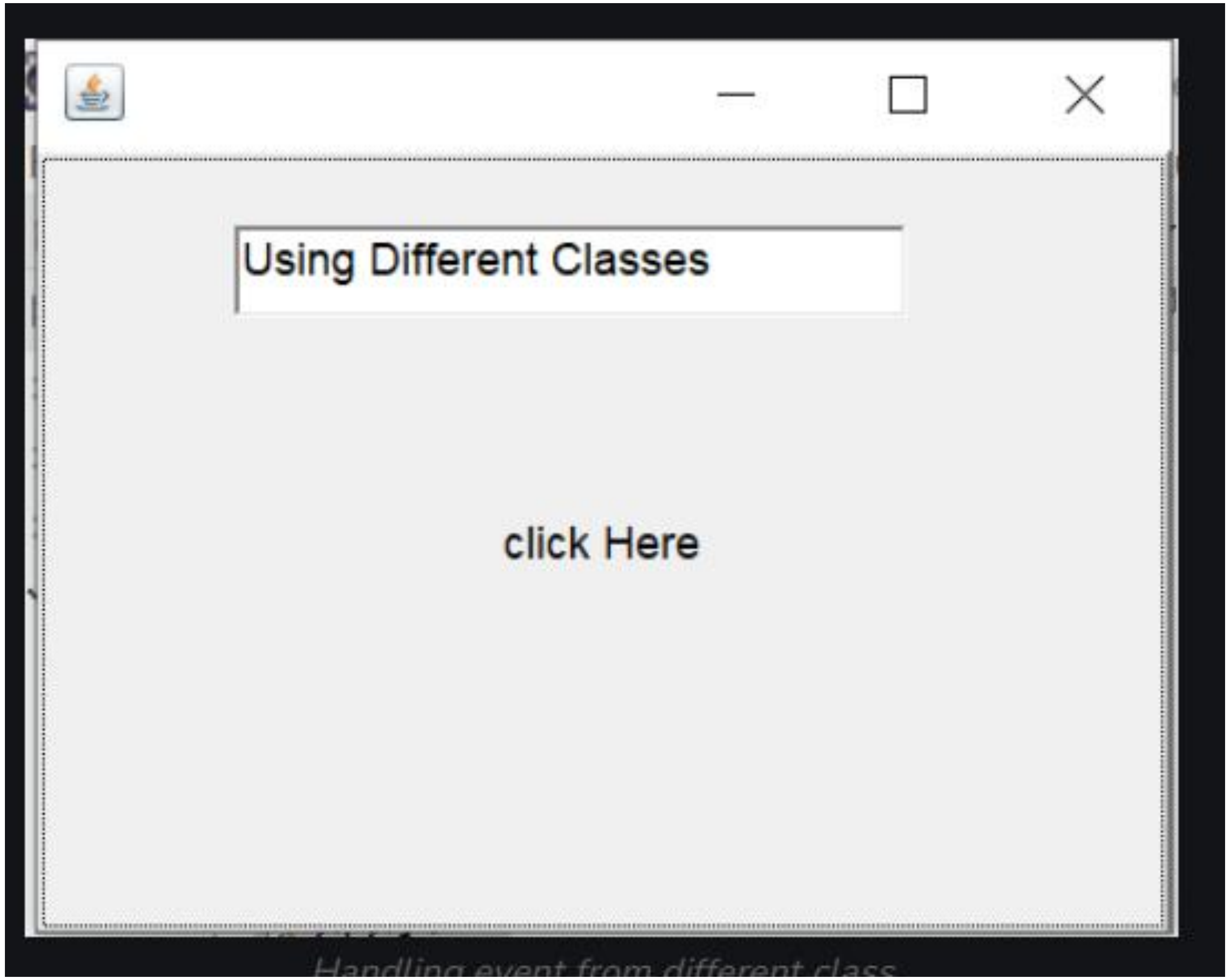

```
// Java program to demonstrate the
// event handling by the other class

import java.awt.*;
import java.awt.event.*;

class GFG1 extends Frame {

    TextField textField;

    GFG2()
    {
        // Component Creation
        textField = new TextField();

        // setBounds method is used to provide
        // position and size of component
        textField.setBounds(60, 50, 180, 25);
        Button button = new Button("click Here");
        button.setBounds(100, 120, 80, 30);

        Other other = new Other(this);
```



```
        Other other = new Other(this);

        // Registering component with listener
        // Passing other class as reference
        button.addActionListener(other);

        // add Components
        add(textField);
        add(button);

        // set visibility
        setVisible(true);
    }

    public static void main(String[] args)
    {
    new GFG2();
    }
}
```



```java
/// import necessary packages
import java.awt.event.*;

// implements the listener interface
class Other implements ActionListener {

    GFG2 gfgObj;

    Other(GFG1 gfgObj) {
      this.gfgObj = gfgObj;
    }

    public void actionPerformed(ActionEvent e)
    {
        // setting text from different class
        gfgObj.textField.setText("Using Different Classes");
    }
}
```


Using Different Classes
click Here
Handling event from different class

- Handling events for Combobox, List, TextField, and TextArea components in a Java applet involves implementing the appropriate listener interfaces and defining the event handling logic within the corresponding methods. Here's an example demonstrating how to handle events for these components

- Explanation:
- **ItemListener:**
- Used for handling item selection events in Choice (Combobox) and List components. The itemStateChanged() method is called when an item is selected or deselected.

- **ActionListener:**
- Used for handling action events in TextField components. The actionPerformed() method is called when an action occurs, such as pressing the Enter key in the text field.

- **TextListener:**
- Used for handling text change events in TextArea components. The textValueChanged() method is called when the text in the text area is modified.

- Remember to:
- **Import necessary classes:**
- Import the required classes from java.awt and java.awt.event packages.
- **Implement listener interfaces:**
- Implement the appropriate listener interfaces for the components you want to handle events for.
- **Register listeners:**
- Register the listeners with the components using methods like addItemListener(), addActionListener(), and addTextListener().
- **Define event handling logic:**
- Implement the methods of the listener interfaces to define the specific actions to be performed when the events occur.

Thank you