AWT & Its Controls PART-1

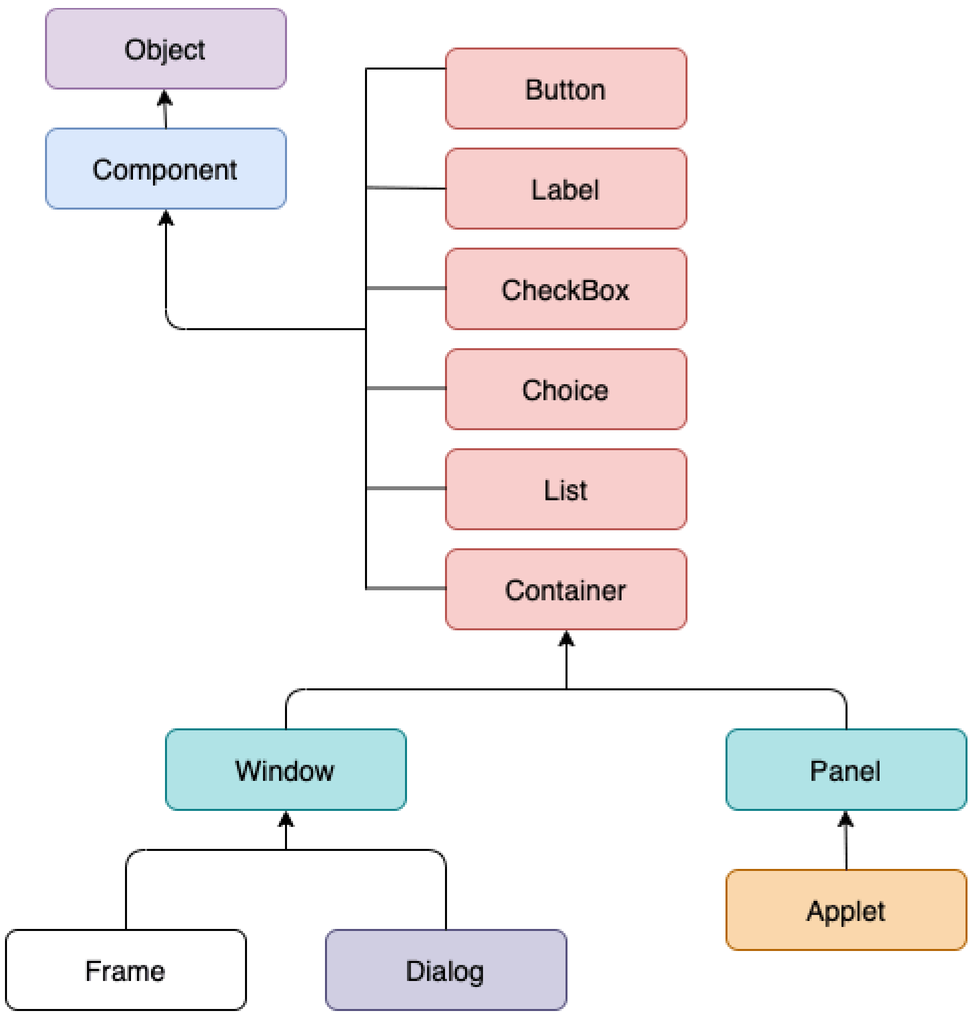
AWT: *Class hierarchy, Component, Container, Panel, Window, Frame, Graphics.*
AWT Controls: *Labels, Button, Scrollbar, Text Components, Checkbox, CheckboxGroup, Choice, List, Panes – ScrollPane, Dialog and Menu Bar.*

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

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

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

AWT Class Hierarchy



Component

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

- Component is the **superclass** for all GUI elements in AWT, such as Button, Label, TextField, etc.
- It provides basic **behavior and properties** for UI elements, including size, position, color, font, and event handling.

Methods of Component

Method	Description
setSize(int width, int height)	Sets the size of the component.
setBounds(int x, int y, int width, int height)	Sets position and size of the component.
setBackground(Color c)	Sets background color.
setForeground(Color c)	Sets text color.
setFont(Font f)	Sets font style.
setVisible(boolean b)	Shows or hides the component.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog,** and **Panel.**

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

A **Container** is a **subclass of Component** that can **hold multiple components** and arrange them using a **layout manager**.

Hierarchy of Container

```
java.lang.Object
   └── java.awt.Component
      └── java.awt.Container
```

Methods of Container

Method	Description
add(Component c)	Adds a component to the container.
remove(Component c)	Removes a component from the container.
setLayout(LayoutManager mgr)	Sets the layout manager for arranging components.
removeAll()	Removes all components from the container.

Types of Containers

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

1. Window

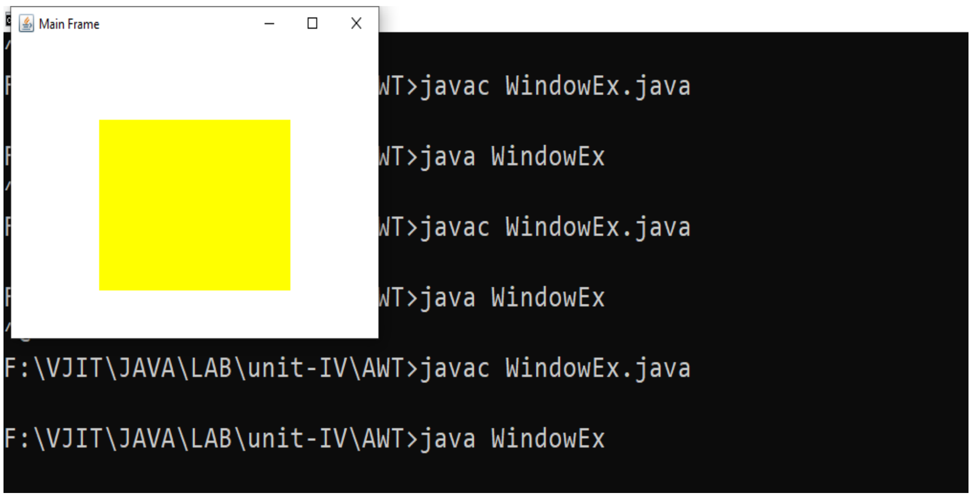
The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

A **Window** in Java AWT is a **top-level container** that **does not have a title bar, menu bar, or borders**. It is a **subclass of Container**, and other classes like Frame and Dialog inherit from it.

Key Features of Window

- Cannot be created directly, must use subclasses (Frame, Dialog).
- Used for **popup windows** or custom **borderless** windows.

```
import java.awt.*;
public class WindowEx
{
   public static void main(String[] args)
  {
      Frame frame = new Frame("Main Frame");
      Window window = new Window(frame);

      window.setSize(200, 150);
      window.setBackground(Color.YELLOW);
      window.setLocation(100, 100);
      window.setVisible(true);

      frame.setSize(400, 300);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```



```
WT>javac WindowEx.java

WT>java WindowEx

WT>javac WindowEx.java

WT>java WindowEx

F:\VJIT\JAVA\LAB\unit-IV\AWT>javac WindowEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT>java WindowEx
```

2. Panel

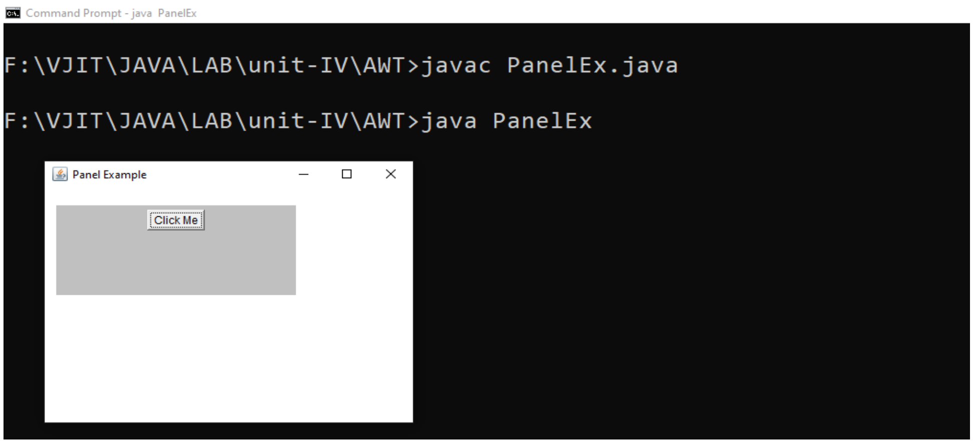
The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

A **Panel** in Java AWT is a **container** that can hold other components, such as buttons, labels, text fields, etc. Unlike Frame, a Panel does not have a **title bar, minimize, maximize, or close buttons**.

Key Features of Panel

- It is a **lightweight container** that can be used inside a Frame or Applets.
- Panels are useful for organizing components in a structured way.
- A Panel can have a **layout manager** to arrange its components.

```
import java.awt.*;
public class PanelEx
{
   public static void main(String[] args)
  {
      Frame frame = new Frame("Panel Example");
      // Creating a panel
      Panel panel = new Panel();
      panel.setBounds(20, 50, 250, 100);
      panel.setBackground(Color.LIGHT_GRAY);

      // Adding a button to the panel
      Button button = new Button("Click Me");
      panel.add(button);
      // Adding panel to the frame
      frame.add(panel);
      frame.setSize(400, 300);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT>javac PanelEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT>java PanelEx
```

3. Frame

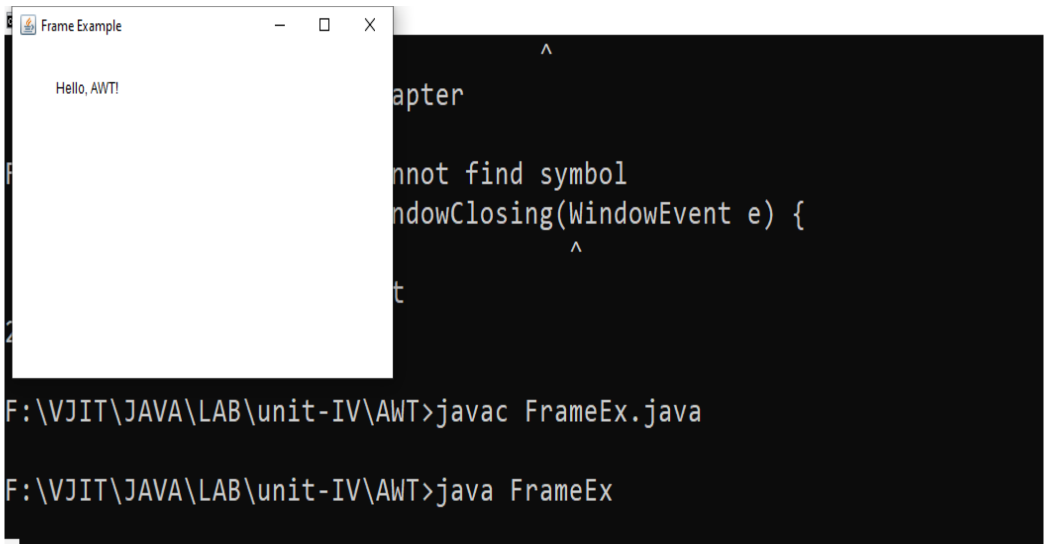
The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

A **Frame** is a **resizable and closable** window with a title bar, minimize, maximize, and close buttons.

Key Features of Frame

- It is a **subclass of Window** and provides **full window functionalities**.
- It is used as the **main application window**.
- The default layout of a Frame is BorderLayout.

```
import java.awt.*;
public class FrameEx
{
    public static void main(String[] args)
   {
        // Creating a frame
        Frame frame = new Frame("Frame Example");
        // Creating a label
        Label label = new Label("Hello, AWT!");
        label.setBounds(50, 50, 100, 30);
        // Adding components to the frame
        frame.add(label);
        frame.setSize(400, 300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
                                       ^
                                      apter

                                      nnot find symbol
                                      ndowClosing(WindowEvent e) {
                                                ^
                                      t

F:\VJIT\JAVA\LAB\unit-IV\AWT>javac FrameEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT>java FrameEx
```

4. Dialog

A **Dialog** in **Java AWT (Abstract Window Toolkit)** is a **popup window** that appears on top of a parent window.

It is primarily used for **messages, warnings, or user input dialogs**.

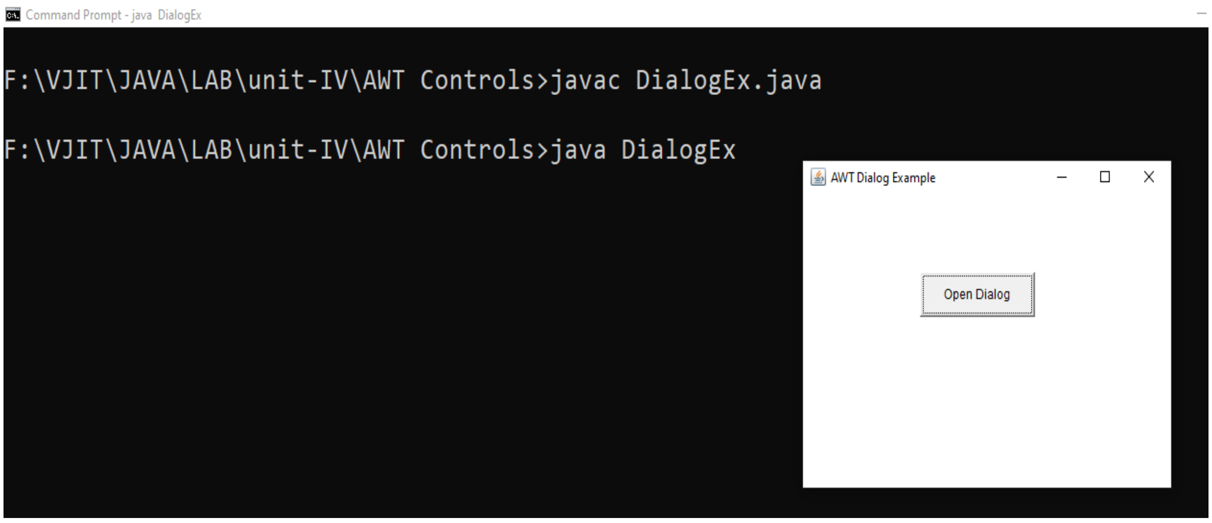
- Requires a **parent Frame** to be created.
- Can be **modal** (blocks interaction with the parent) or **non-modal** (allows interaction).
- Supports adding **buttons, text fields, and other controls**.

Dialog Class Hierarchy

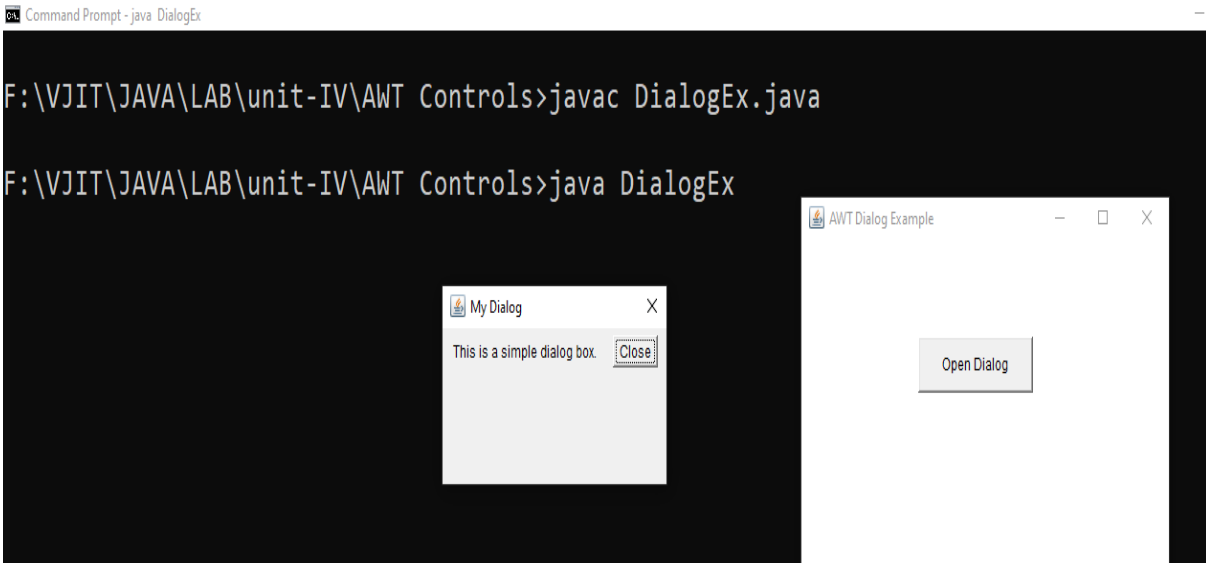
```
java.lang.Object
  └── java.awt.Component
    └── java.awt.Container
      └── java.awt.Window
        └── java.awt.Dialog
```

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

public class DialogEx
{
    public static void main(String[] args)
   {
        // Creating a Frame (Parent Window)
        Frame frame = new Frame("AWT Dialog Example");
        frame.setSize(400, 300);
        frame.setLayout(null);
        // Creating a Button to open the Dialog
        Button button = new Button("Open Dialog");
        button.setBounds(130, 100, 120, 40);
        // Creating a Dialog (Child Window)
        Dialog dialog = new Dialog(frame, "My Dialog", true); // Modal Dialog
        dialog.setSize(250, 150);
        dialog.setLayout(new FlowLayout());

        // Adding Label to Dialog
        Label label = new Label("This is a simple dialog box.");
        dialog.add(label);

        // Adding Close Button to Dialog
        Button closeButton = new Button("Close");
        dialog.add(closeButton);

        // Closing Dialog on Button Click
        closeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                dialog.setVisible(false);
            }
        });
        // Showing Dialog on Button Click
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
           {
                dialog.setVisible(true);
            }
        });
```

```
            // Adding Components to Frame
            frame.add(button);

            // Handling Frame Closing
            frame.addWindowListener(new WindowAdapter() {
               public void windowClosing(WindowEvent e)
              {
                   frame.dispose();
                }
            });
            frame.setVisible(true);
        }
    }
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac DialogEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java DialogEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac DialogEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java DialogEx
```

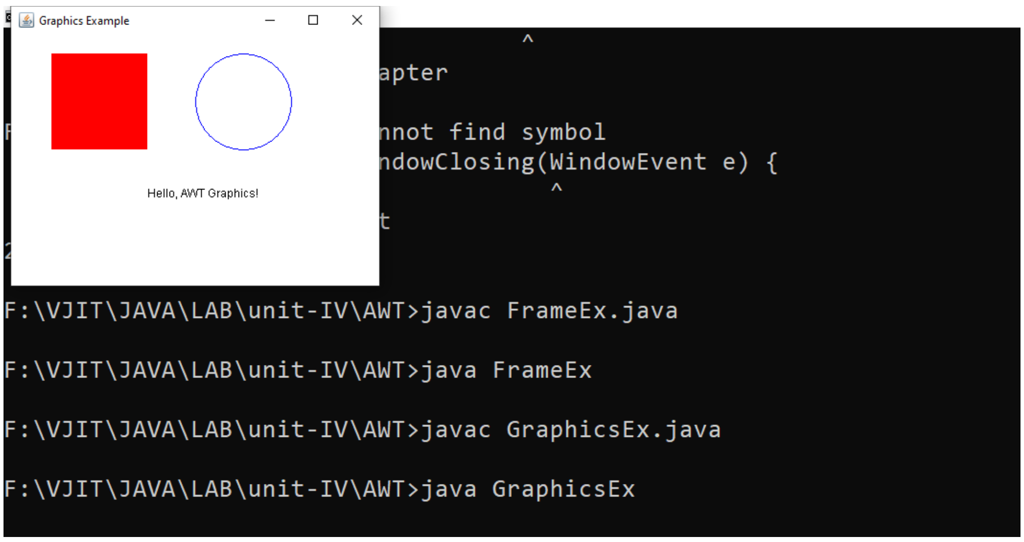
5. Graphics

The **Graphics** class in Java AWT is used for **drawing shapes, text, and images** on GUI components.

Key Features of Graphics

- Provides methods like drawRect(), drawOval(), drawString(), etc.
- Used for **custom drawing** on components.

```
import java.awt.*;
public class GraphicsEx extends Frame
{
   public GraphicsEx()
   {
      setTitle("Graphics Example");
      setSize(400, 300);
      setVisible(true);
   }

   public void paint(Graphics g)
  {
      // Set color and draw a rectangle
      g.setColor(Color.RED);
      g.fillRect(50, 50, 100, 100); // x, y, width, height

      // Set color and draw an oval
      g.setColor(Color.BLUE);
      g.drawOval(200, 50, 100, 100); // x, y, width, height

      // Draw text
      g.setColor(Color.BLACK);
      g.drawString("Hello, AWT Graphics!", 150, 200);
   }

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



```
                                                ^
                                  apter

                                  nnot find symbol
                                  ndowClosing(WindowEvent e) {
                                                  ^
                                  t

F:\VJIT\JAVA\LAB\unit-IV\AWT>javac FrameEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT>java FrameEx

F:\VJIT\JAVA\LAB\unit-IV\AWT>javac GraphicsEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT>java GraphicsEx
```

AWT Controls

AWT (Abstract Window Toolkit) provides a set of In-built UI components (controls) for building Graphical User Interfaces (GUIs) in Java. These components include labels, buttons, text fields, lists, checkboxes, menus, and more.

Types of AWT Controls

AWT controls can be categorized as follows:

1. **Basic Controls**
   - Label
   - Button
   - Checkbox
   - CheckboxGroup
   - Choice
   - List
2. **Text Components**
   - TextField
   - TextArea

3. **Scrollable Components**
    - Scrollbar
    - ScrollPane
4. **Popup Components**
    - Dialog
5. **Menu Components**
    - MenuBar
    - Menu
    - MenuItem

1. Label

A **Label** in AWT (java.awt.Label) is a **non-editable text component** used to display information, instructions, or status messages in a GUI application. It is commonly used in forms, dialogs, and dashboards to provide meaningful descriptions to users.

Key Features of Label

- Displays **static text** (non-editable).
- Used for **instructions, titles, and messages**.
- Supports **alignment** (LEFT, CENTER, RIGHT).
- Can be updated dynamically using setText().

Constructor of Label

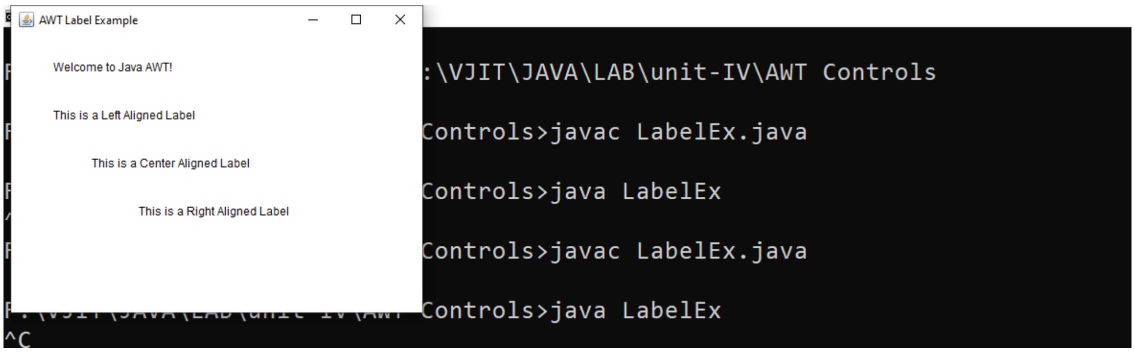
The Label class provides three constructors:

Constructor	Description
Label()	Creates an empty label
Label(String text)	Creates a label with specified text
Label(String text, int alignment)	Creates a label with text and alignment (LEFT, CENTER, RIGHT)

Common Methods of Label

Method	Description
setText(String text)	Sets new text for the label
getText()	Retrieves the text of the label
setAlignment(int alignment)	Sets alignment (Label.LEFT, Label.CENTER, Label.RIGHT)
getAlignment()	Gets current alignment of label

```
import java.awt.*;
public class LabelEx
{
    public static void main(String[] args)
   {
        // Creating a Frame
        Frame frame = new Frame("AWT Label Example");

        // Creating Labels
        Label label1 = new Label("Welcome to Java AWT!"); // Default alignment
       label1.setBounds(50, 50, 200, 30);

        Label label2 = new Label("This is a Left Aligned Label", Label.LEFT);
        label2.setBounds(50, 100, 250, 30);

        Label label3 = new Label("This is a Center Aligned Label", Label.CENTER);
        label3.setBounds(50, 150, 250, 30);

        Label label4 = new Label("This is a Right Aligned Label", Label.RIGHT);
        label4.setBounds(50, 200, 250, 30);

        // Adding Labels to the Frame
        frame.add(label1);
        frame.add(label2);
        frame.add(label3);
        frame.add(label4);
        // Setting Frame properties
        frame.setSize(400, 300);
        frame.setLayout(null);  // Using absolute positioning
        frame.setVisible(true);
    }
}
```



AWT Label Example

```
Welcome to Java AWT!

This is a Left Aligned Label

    This is a Center Aligned Label

            This is a Right Aligned Label
```

```
:\VJIT\JAVA\LAB\unit-IV\AWT Controls

Controls>javac LabelEx.java

Controls>java LabelEx

Controls>javac LabelEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java LabelEx
^C
```

2. Button

A **Button** in Java AWT (java.awt.Button) is a component that triggers an event when clicked. It is commonly used for submitting forms, triggering actions, or navigating between screens.

Key Features of AWT Button

- **Triggers an action when clicked** (e.g., submit, exit, etc.).
- Can **display text or an image** as a label.
- Supports **event handling using ActionListener**.
- Can be **enabled/disabled dynamically**.

Constructors of AWT Button

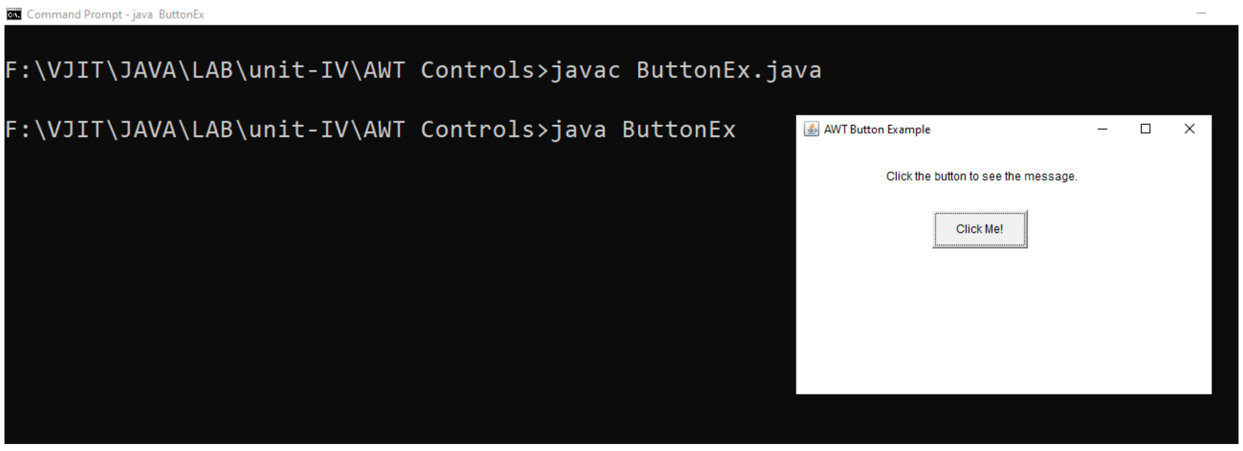
The Button class provides the following constructors:

Constructor	Description
Button()	Creates an empty button (without text)
Button(String text)	Creates a button with specified text

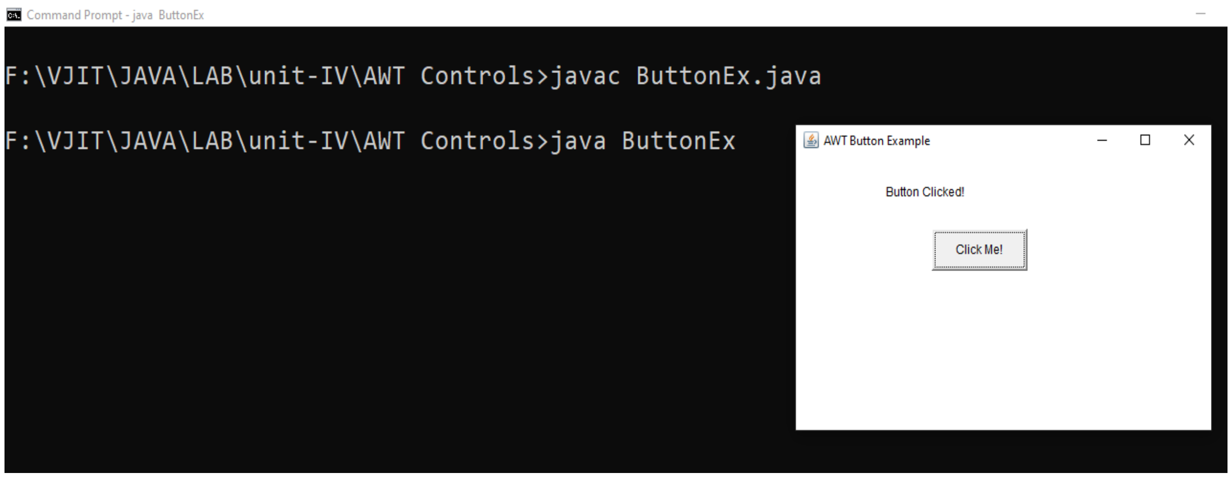
Common Methods of AWT Button

Method	Description
setLabel(String text)	Changes the button text
getLabel()	Retrieves the button text
setEnabled(boolean status)	Enables/disables the button (true/false)
isEnabled()	Checks if the button is enabled
addActionListener(ActionListener l)	Registers an event listener for button clicks

```
import java.awt.*;
import java.awt.event.*;
public class ButtonEx
{
    public static void main(String[] args)
  {
        // Creating a Frame
        Frame frame = new Frame("AWT Button Example");
        // Creating a Button
        Button button = new Button("Click Me!");
        button.setBounds(150, 100, 100, 40); // Centering the button more
        // Creating a Label
        Label label = new Label("Click the button to see the message.");
        label.setBounds(100, 50, 250, 30);
        // Adding an ActionListener to the button
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
           {
                label.setText("Button Clicked!"); // Update the label
            }
        });
        // Adding WindowListener to Close the Frame Properly
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
           {
                frame.dispose(); // Close the frame properly
            }
        });

        // Adding Components to the Frame
        frame.add(button);
        frame.add(label);

        // Setting Frame properties
        frame.setSize(450, 300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
Command Prompt - java ButtonEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ButtonEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ButtonEx
```


```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ButtonEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ButtonEx
```

3. Checkbox

- Allows **single or multiple selections**.
- Can be **checked (selected) or unchecked (deselected)**.
- Can have **labels** to describe the choice.
- Can be **enabled or disabled dynamically**.
- Supports **event handling** for selection changes.

Constructors of AWT Checkbox

The Checkbox class provides several constructors:

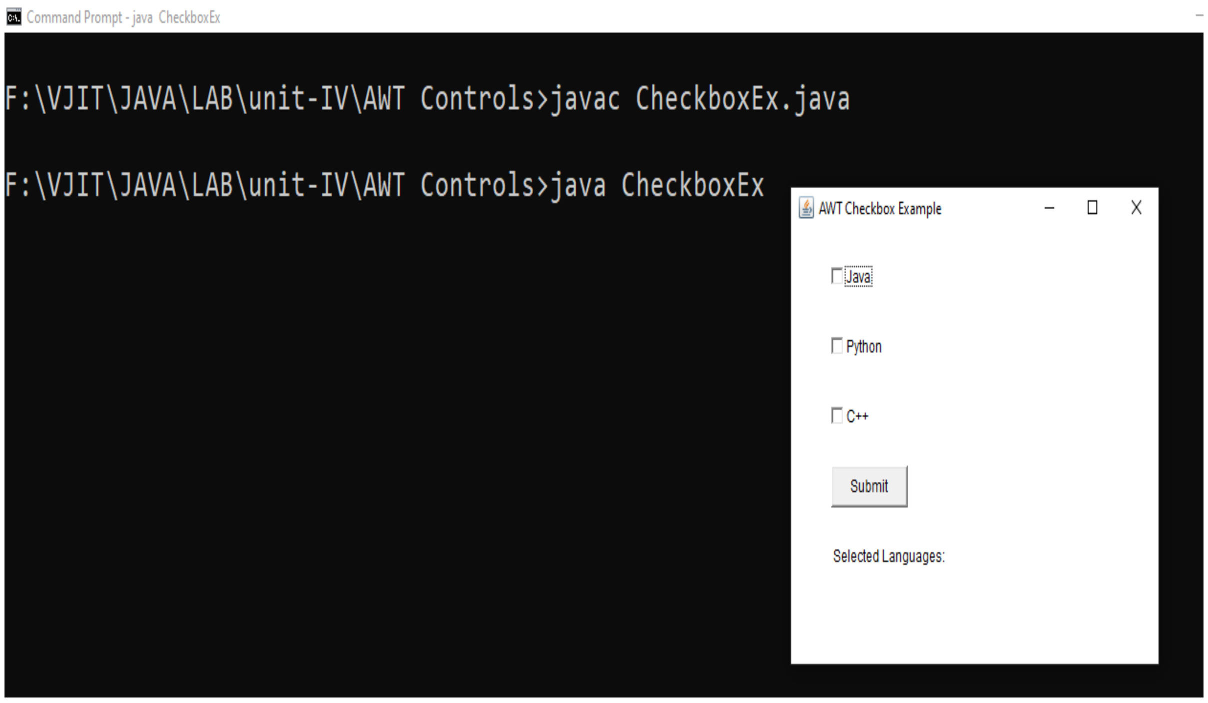
Constructor	Description
Checkbox()	Creates an empty checkbox
Checkbox(String label)	Creates a checkbox with specified label
Checkbox(String label, boolean state)	Creates a checkbox with label and sets initial state (true = checked, false = unchecked)
Checkbox(String label, boolean state, CheckboxGroup group)	Creates a checkbox with label, state, and associates it with a CheckboxGroup (for radio-button behavior)

Common Methods of AWT Checkbox

Method	**Description**
setLabel(String text)	Changes the checkbox label
getLabel()	Retrieves the label text
setState(boolean state)	Checks (true) or unchecks (false) the checkbox
getState()	Returns the checkbox's state (true = checked, false = unchecked)
setEnabled(boolean status)	Enables or disables the checkbox (true = enabled, false = disabled)
addItemListener(ItemListener l)	Registers an event listener to detect checkbox selection changes

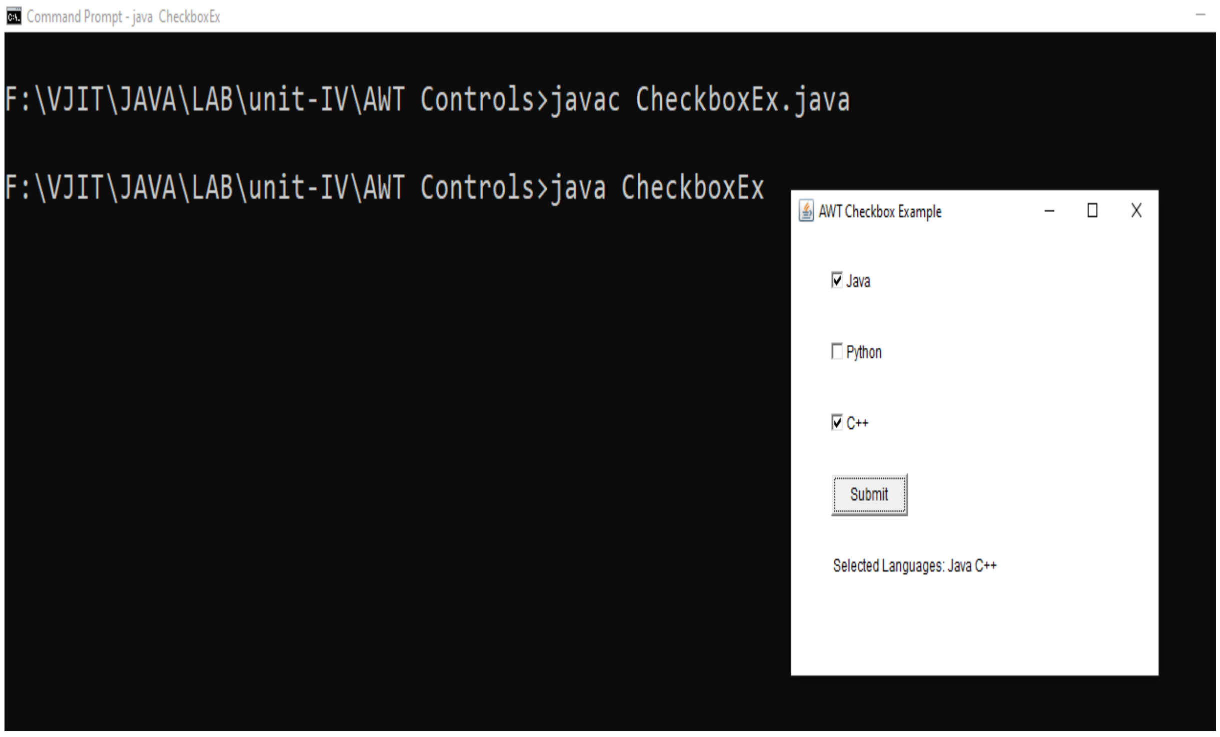
```
import java.awt.*;
import java.awt.event.*;
public class CheckboxEx
{
   public static void main(String[] args)
  {
      // Creating a Frame
      Frame frame = new Frame("AWT Checkbox Example");

      // Creating Checkboxes
      Checkbox checkbox1 = new Checkbox("Java");
      checkbox1.setBounds(50, 50, 100, 30);

      Checkbox checkbox2 = new Checkbox("Python");
      checkbox2.setBounds(50, 100, 100, 30);

      Checkbox checkbox3 = new Checkbox("C++");
      checkbox3.setBounds(50, 150, 100, 30);

      // Creating a Label to display the result
      Label resultLabel = new Label("Selected Languages: ");
      resultLabel.setBounds(50, 250, 300, 30);

      // Creating a Submit Button
      Button submitButton = new Button("Submit");
      submitButton.setBounds(50, 200, 80, 30);
```

```
        // Adding ActionListener to the Button
        submitButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e)
         {
              String selectedLanguages = "Selected Languages: ";

              if (checkbox1.getState()) selectedLanguages += "Java ";
              if (checkbox2.getState()) selectedLanguages += "Python ";
              if (checkbox3.getState()) selectedLanguages += "C++ ";

              resultLabel.setText(selectedLanguages);
          }
        });
        // Adding WindowListener to Close the Frame
        frame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e)
         {
              frame.dispose(); // Close the frame
          }
        });

        // Adding Components to the Frame
        frame.add(checkbox1);
        frame.add(checkbox2);
        frame.add(checkbox3);
        frame.add(submitButton);
        frame.add(resultLabel);

        // Setting Frame properties
        frame.setSize(400, 350);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
Command Prompt - java  CheckboxEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac CheckboxEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java CheckboxEx
```

```
Selected Languages:
```



```
Command Prompt - java  CheckboxEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac CheckboxEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java CheckboxEx
```

```
Selected Languages: Java C++
```

4. CheckboxGroup

CheckboxGroup is used when you want to create a **set of mutually exclusive checkboxes** (similar to radio buttons). Unlike normal checkboxes (Checkbox), where multiple options can be selected, **CheckboxGroup ensures that only one checkbox is selected at a time**.

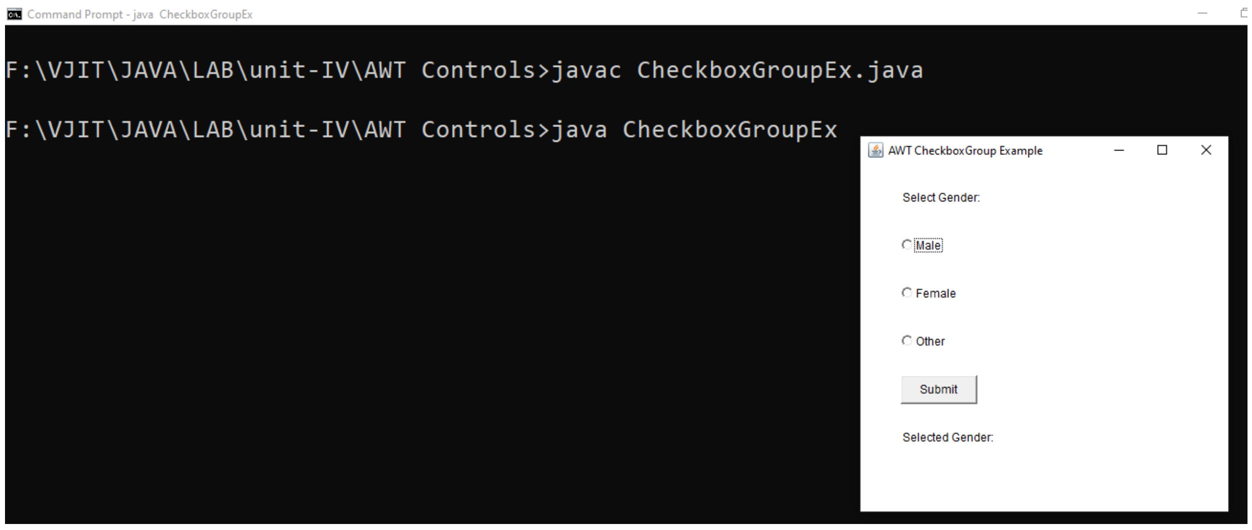
Key Features of CheckboxGroup

- It allows you to create **radio button-like behavior** in AWT.
- Only **one checkbox can be selected at a time**.
- If one checkbox is selected, the previously selected one **automatically gets deselected**.
- Used for **single-choice selections**, such as **gender selection, payment options, and shipping methods**.

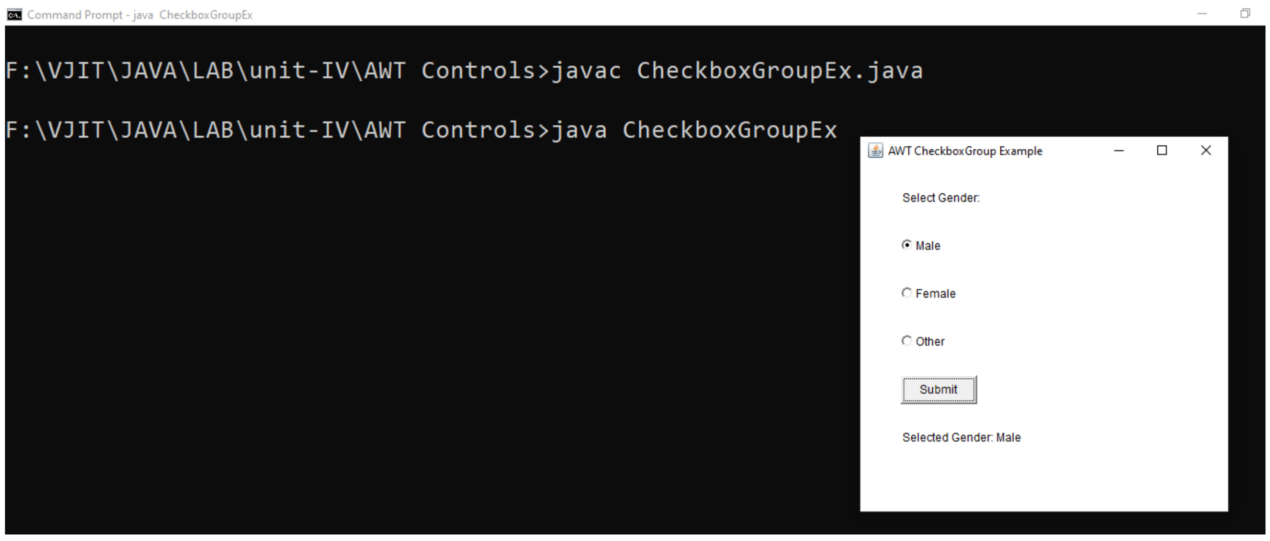
```
import java.awt.*;
import java.awt.event.*;

public class CheckboxGroupEx
{
   public static void main(String[] args)
  {
      // Creating a Frame
      Frame frame = new Frame("AWT CheckboxGroup Example");

      // Creating a Label
      Label label = new Label("Select Gender:");
      label.setBounds(50, 50, 100, 30);

      // Creating CheckboxGroup
      CheckboxGroup genderGroup = new CheckboxGroup();

      // Creating Checkboxes and assigning them to CheckboxGroup
```

```
Checkbox maleCheckbox = new Checkbox("Male", genderGroup, false);
maleCheckbox.setBounds(50, 100, 100, 30);

Checkbox femaleCheckbox = new Checkbox("Female", genderGroup, false);
femaleCheckbox.setBounds(50, 150, 100, 30);

Checkbox otherCheckbox = new Checkbox("Other", genderGroup, false);
otherCheckbox.setBounds(50, 200, 100, 30);

// Creating a Submit Button
Button submitButton = new Button("Submit");
submitButton.setBounds(50, 250, 80, 30);

// Creating a Label to display the selected option
Label resultLabel = new Label("Selected Gender: ");
resultLabel.setBounds(50, 300, 300, 30);

// Adding ActionListener to the button
submitButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e)
  {
      // Getting the selected checkbox
      Checkbox selectedCheckbox = genderGroup.getSelectedCheckbox();
      if (selectedCheckbox != null)
      {
         resultLabel.setText("Selected Gender: " + selectedCheckbox.getLabel());
      }
   }
});

// Adding WindowListener to Close the Frame Properly
frame.addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent e)
  {
      frame.dispose(); // Close the frame properly
   }
});
```

```
        // Adding Components to Frame
        frame.add(label);
        frame.add(maleCheckbox);
        frame.add(femaleCheckbox);
        frame.add(otherCheckbox);
        frame.add(submitButton);
        frame.add(resultLabel);

        // Setting Frame properties
        frame.setSize(400, 400);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
Command Prompt - java  CheckboxGroupEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac CheckboxGroupEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java CheckboxGroupEx
```



```
Command Prompt - java  CheckboxGroupEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac CheckboxGroupEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java CheckboxGroupEx
```

5. Choice

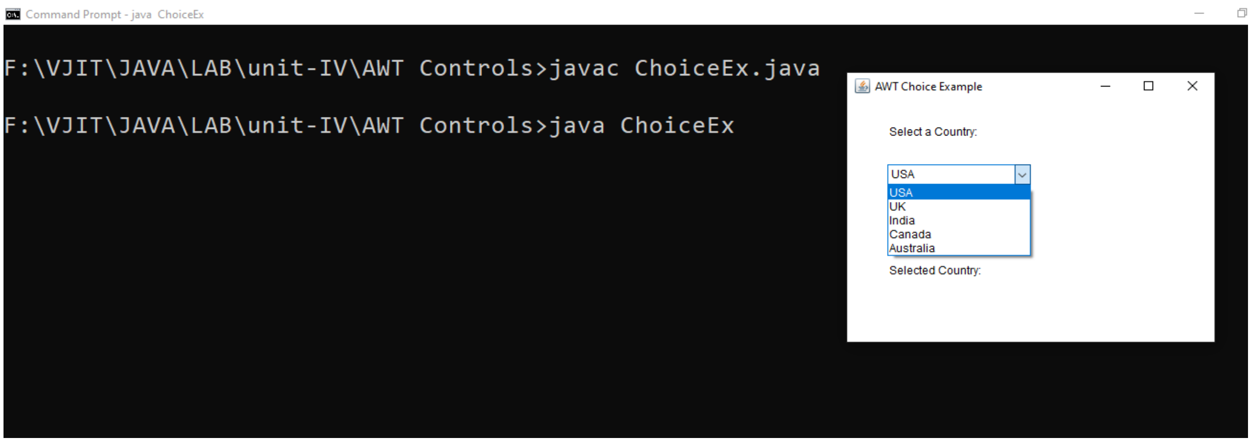
The **Choice** class in **Java AWT (Abstract Window Toolkit)** is a **drop-down list** that allows users to **select one option from multiple choices**. It is similar to a **ComboBox** in Swing or an HTML <select> element.

Key Features of AWT Choice

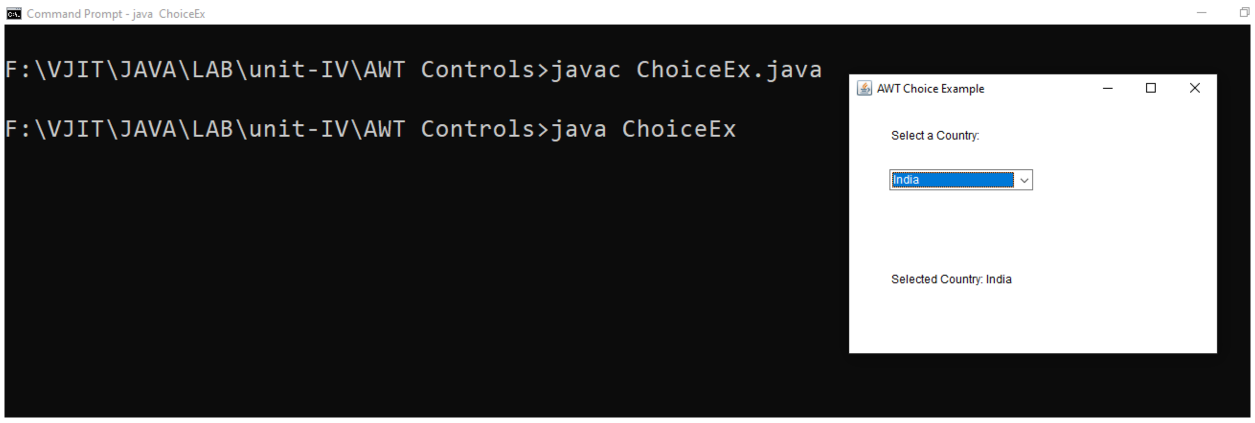
- **Drop-down menu format** – Displays multiple items in a compact UI. **Single selection** – The user can only select **one item at a time**.
- **Efficient for limited options** – Best used when the number of choices is fixed and small.
- **Event handling** – Detects selection changes using ItemListener.

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

public class ChoiceEx
{
   public static void main(String[] args)
  {
      // Creating a Frame
      Frame frame = new Frame("AWT Choice Example");

      // Creating a Label
      Label label = new Label("Select a Country:");
      label.setBounds(50, 50, 150, 30);

      // Creating a Choice (Dropdown List)
      Choice countryChoice = new Choice();
      countryChoice.setBounds(50, 100, 150, 30);

      // Adding Items to Choice
      countryChoice.add("USA");
      countryChoice.add("UK");
      countryChoice.add("India");
      countryChoice.add("Canada");
      countryChoice.add("Australia");
```

```
        // Creating a Label to Display Selected Item
        Label resultLabel = new Label("Selected Country: ");
        resultLabel.setBounds(50, 200, 200, 30);

        // Adding ItemListener to Detect Selection Change
        countryChoice.addItemListener(new ItemListener() {
           public void itemStateChanged(ItemEvent e)
           {
              resultLabel.setText("Selected Country: " +countryChoice.getSelectedItem());
           }
        });

    // Adding WindowListener to Close the Frame Properly
        frame.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e)
           {
              frame.dispose(); // Close the frame properly
           }
        });
        // Adding Components to the Frame
        frame.add(label);
        frame.add(countryChoice);
        frame.add(resultLabel);
        // Setting Frame properties
        frame.setSize(400, 300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ChoiceEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ChoiceEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ChoiceEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ChoiceEx
```

Select a Country:

India

Selected Country: India

6. List

List class is used to create a **scrollable list** of items where users can **select one or multiple items**. It is similar to an HTML <select> element with the multiple attribute enabled.

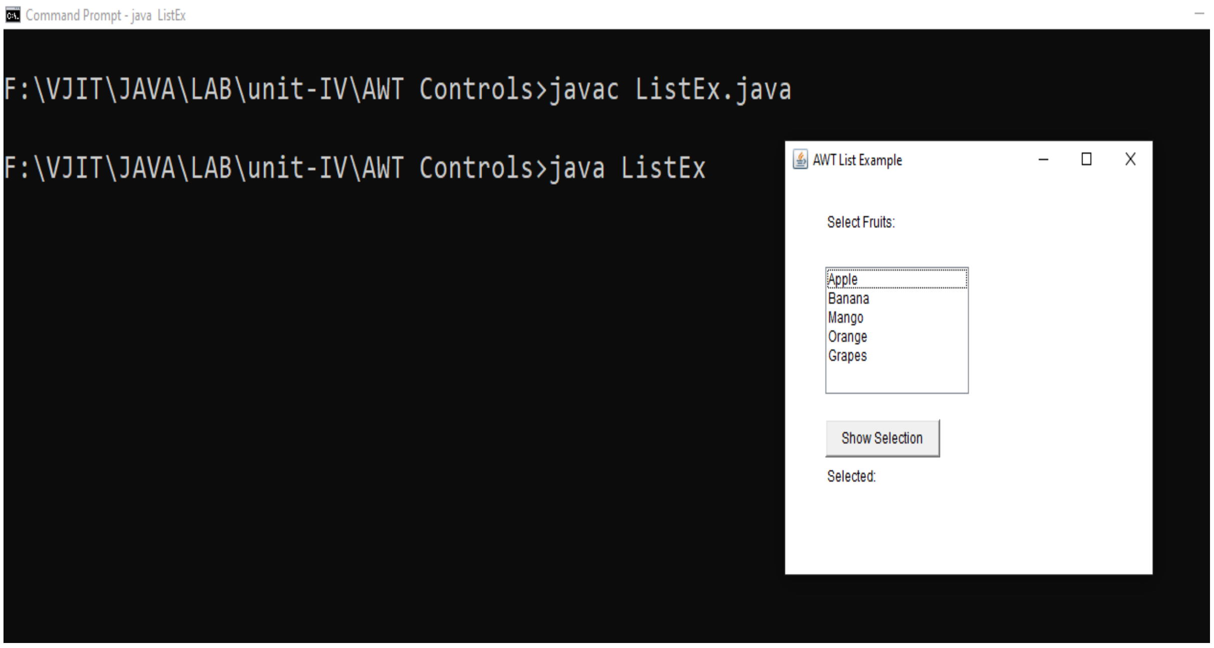
Key Features of AWT List

- **Multi-selection support** – Allows users to select **one or multiple** items.
- **Scrollable list** – If the number of items exceeds the visible area, a **scrollbar** is automatically added.
- **Event handling** – Detects selection changes using ItemListener.
- **Useful for large datasets** – More user-friendly than dropdowns (Choice) for multiple selections.

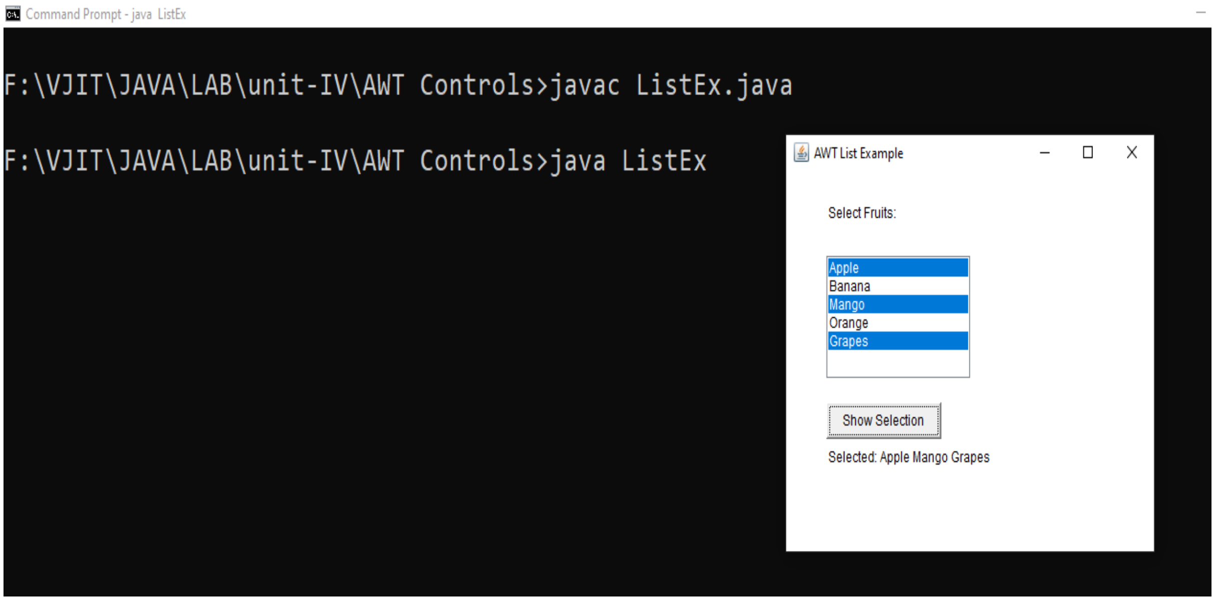
```
import java.awt.*;
import java.awt.event.*;
public class ListEx
{
   public static void main(String[] args)
    {
      // Creating a Frame
      Frame frame = new Frame("AWT List Example");
      // Creating a Label
      Label label = new Label("Select Fruits:");
      label.setBounds(50, 50, 100, 30);

      // Creating a List (with 4 visible items, allowing multiple selection)
      List fruitList = new List(4, true);
      fruitList.setBounds(50, 100, 150, 100);
```

```
// Adding Items to List
fruitList.add("Apple");
fruitList.add("Banana");
fruitList.add("Mango");
fruitList.add("Orange");
fruitList.add("Grapes");
// Creating a Label to Display Selected Items
Label resultLabel = new Label("Selected: ");
resultLabel.setBounds(50, 250, 250, 30);

// Creating a Button to Confirm Selection
Button selectButton = new Button("Show Selection");
selectButton.setBounds(50, 220, 120, 30);

// Adding ActionListener to Button
selectButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e)
   {
      // Get selected items from the list
      String selectedItems = "";
      for (String item : fruitList.getSelectedItems())
      {
         selectedItems += item + " ";
      }
      resultLabel.setText("Selected: " + selectedItems);
   }
});

// Adding WindowListener to Close the Frame Properly
frame.addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent e)
   {
      frame.dispose(); // Close the frame properly
   }
});

// Adding Components to Frame
frame.add(label);
frame.add(fruitList);
frame.add(selectButton);
frame.add(resultLabel);
```

```
            // Setting Frame properties
            frame.setSize(400, 350);
            frame.setLayout(null);
            frame.setVisible(true);
        }
    }
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ListEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ListEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ListEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ListEx
```

7. TextField and TextArea

TextField and **TextArea** components allow users to enter and edit text.

- **TextField** → A single-line text input field (like an HTML <input type="text">).
- **TextArea** → A multi-line text input field (like an HTML <textarea>).

1. TextField – Single Line Input

Features of TextField

- Allows **single-line** text input.
- Supports **copy, cut, paste** operations.
- Can have **password mode** using setEchoChar().
- Supports **event handling** (ActionListener, TextListener).

Syntax for Creating a TextField

```
TextField textField = new TextField("Enter Name"); // Default text

textField.setBounds(50, 50, 200, 30);
```

- **First argument** (optional) – Sets initial text.
- **Bounds** – Defines the position and size (x, y, width, height).

2. TextArea – Multi-line Input

Features of TextArea

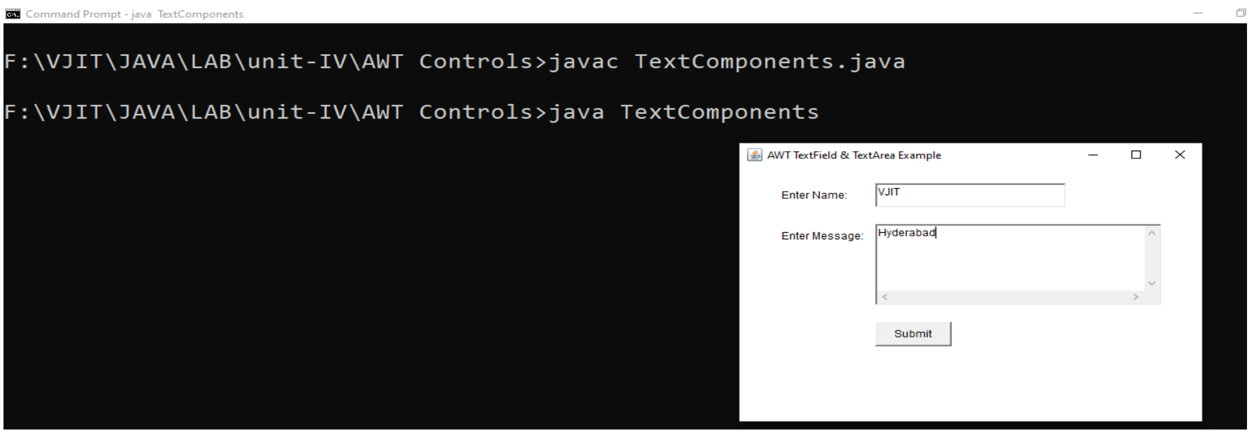
- Allows **multi-line** text input.
- Supports **word wrap** (setLineWrap(true)).
- Can be **editable** or **read-only** (setEditable(false)).
- Detects **text changes** using TextListener.

Syntax for Creating a TextArea

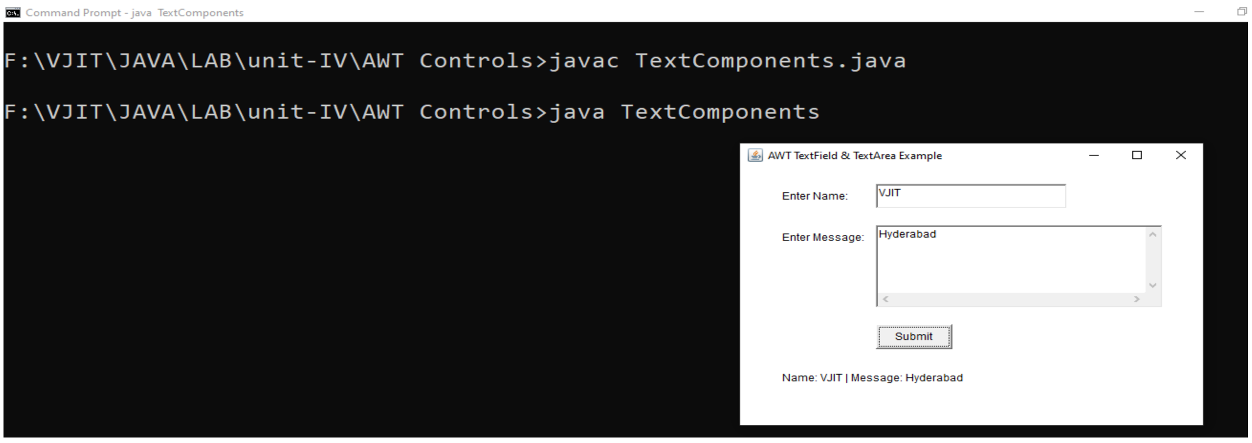
```
TextArea textArea = new TextArea("Write your message here", 5, 30);

textArea.setBounds(50, 100, 300, 100);
```

```
import java.awt.*;
import java.awt.event.*;
public class TextComponents
{
   public static void main(String[] args)
   {
      // Creating a Frame
      Frame frame = new Frame("AWT TextField & TextArea Example");

      // Creating a Label for Name
      Label nameLabel = new Label("Enter Name:");
      nameLabel.setBounds(50, 50, 100, 30);
      // Creating a TextField
      TextField nameField = new TextField();
      nameField.setBounds(150, 50, 200, 30);

      // Creating a Label for Message
      Label messageLabel = new Label("Enter Message:");
      messageLabel.setBounds(50, 100, 100, 30);

      // Creating a TextArea
      TextArea messageArea = new TextArea();
      messageArea.setBounds(150, 100, 300, 100);

      // Creating a Button
      Button submitButton = new Button("Submit");
      submitButton.setBounds(150, 220, 80, 30);

      // Creating a Label to Display Output
      Label outputLabel = new Label();
      outputLabel.setBounds(50, 270, 400, 30);

      // Adding ActionListener to Button
      submitButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e)
         {
            String name = nameField.getText();  // Get text from TextField
            String message = messageArea.getText(); // Get text from TextArea
            outputLabel.setText("Name: " + name + " | Message: " + message);
         }
      });
```

```
        // Adding WindowListener to Close the Frame Properly
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                frame.dispose();
            }
        });
        // Adding Components to Frame
        frame.add(nameLabel);
        frame.add(nameField);
        frame.add(messageLabel);
        frame.add(messageArea);
        frame.add(submitButton);
        frame.add(outputLabel);

        // Setting Frame properties
        frame.setSize(500, 350);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
Command Prompt - java  TextComponents

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac TextComponents.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java TextComponents
```



```
Command Prompt - java  TextComponents

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac TextComponents.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java TextComponents
```

Name: VJIT | Message: Hyderabad

8. ScrollBar

In **Java AWT (Abstract Window Toolkit)**, scrolling components allow users to navigate through content that **exceeds the visible area**.

Java provides two key components for scrolling:

1. **ScrollBar** – A **standalone** vertical or horizontal scrollbar.
2. **ScrollPane** – A **container** that automatically provides scrolling for its child components.

---

1. ScrollBar (Standalone Scroll Control)

Features of ScrollBar

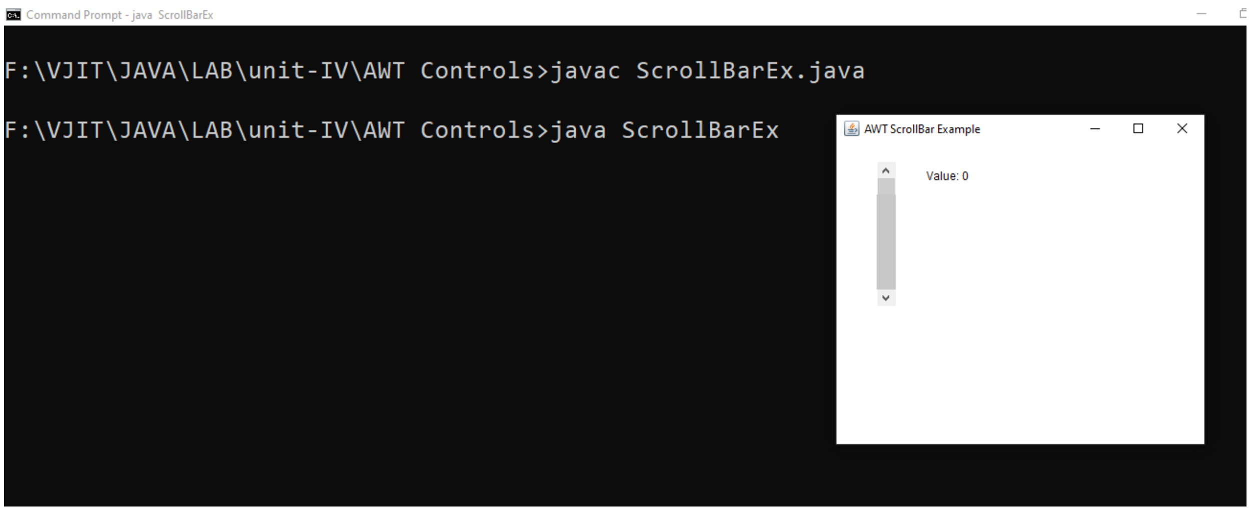
- Supports **horizontal and vertical** scrolling.
- Can be used **independently** or inside containers.
- Allows adjusting **scroll range, page increment, and block increment**.
- Supports event handling (AdjustmentListener) to detect changes.

Syntax for Creating a ScrollBar

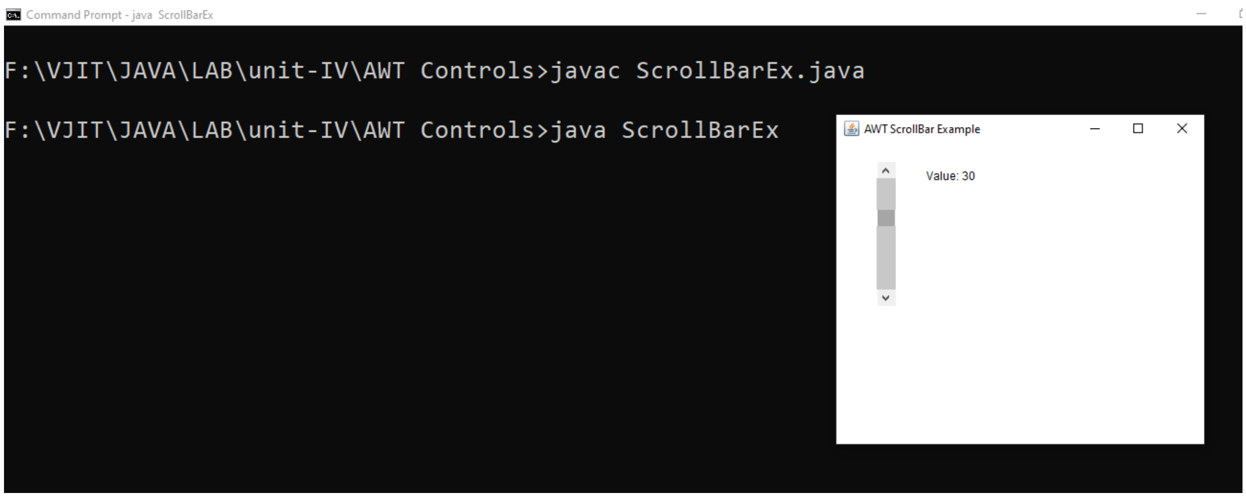
```
ScrollBar sb = new ScrollBar(Scrollbar.VERTICAL, 0, 10, 0, 100);

sb.setBounds(50, 50, 20, 150);
```

- **First Parameter:** Scrollbar.VERTICAL or Scrollbar.HORIZONTAL.
- **Second Parameter:** Initial scrollbar value.
- **Third Parameter:** Thumb size (size of the moving part).
- **Fourth & Fifth Parameters:** Minimum and maximum values.

```
import java.awt.*;
import java.awt.event.*;
public class ScrollBarEx
{
   public static void main(String[] args)
   {
      // Creating a Frame
      Frame frame = new Frame("AWT ScrollBar Example");

      // Creating a Label to Display ScrollBar Value
      Label label = new Label("Value: 0");
      label.setBounds(100, 50, 150, 30);

      // Creating a Vertical ScrollBar
      Scrollbar scrollbar = new Scrollbar(Scrollbar.VERTICAL, 0, 10, 0, 100);
      scrollbar.setBounds(50, 50, 20, 150);

      // Adding AdjustmentListener to detect ScrollBar changes
      scrollbar.addAdjustmentListener(new AdjustmentListener() {
         public void adjustmentValueChanged(AdjustmentEvent e)
        {
            label.setText("Value: " + scrollbar.getValue());
         }
      });
      // Adding Components to Frame
      frame.add(scrollbar);
      frame.add(label);
      // Setting Frame properties
      frame.setSize(400, 350);
      frame.setLayout(null);
      frame.setVisible(true);

      // Closing the Frame Properly
      frame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e)
        {
            frame.dispose();
         }
      });
   }
}
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ScrollBarEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ScrollBarEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ScrollBarEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ScrollBarEx
```

9. ScrollPane (Scrollable Container)

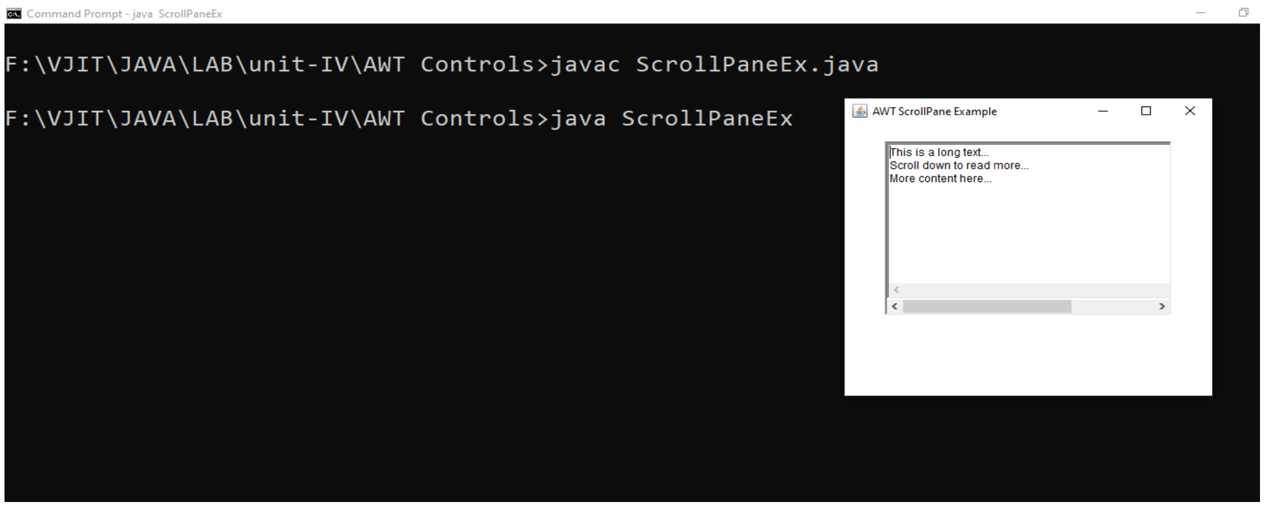
Features of ScrollPane

- Automatically provides **scrollbars** for child components.
- Scrollbars appear **only when necessary** (AUTO mode).
- Useful for **large images, text areas, or tables**.
- Scrolls **horizontally and vertically**.

Syntax for Creating a ScrollPane

```
ScrollPane sp = new ScrollPane();
sp.add(new TextArea("Write here...")); // Adding component
sp.setBounds(50, 50, 300, 200);
```

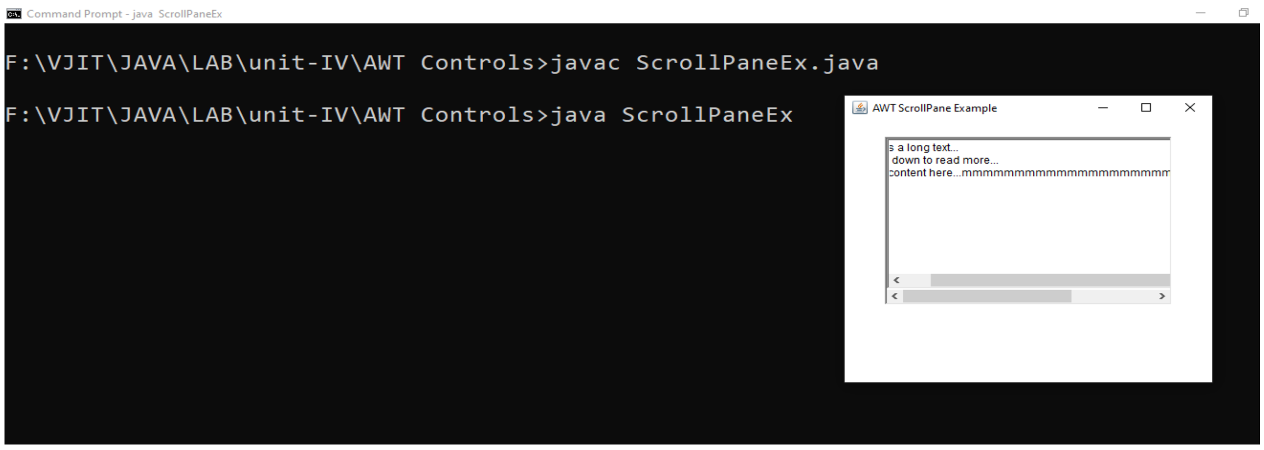
```
import java.awt.*;
import java.awt.event.*;

public class ScrollPaneEx
{
    public static void main(String[] args)
    {
        // Creating a Frame
        Frame frame = new Frame("AWT ScrollPane Example");

        // Creating a ScrollPane (Auto Scrollbars)
        ScrollPane scrollPane = new ScrollPane();
        scrollPane.setBounds(50, 50, 300, 200);

        // Creating a Large TextArea (Content for ScrollPane)
        TextArea textArea = new TextArea("This is a long text...\nScroll down to read
more...\nMore content here...");
        textArea.setBounds(0, 0, 500, 500); // Larger than ScrollPane

        // Adding TextArea to ScrollPane
        scrollPane.add(textArea);

        // Adding ScrollPane to Frame
        frame.add(scrollPane);
        // Setting Frame properties
        frame.setSize(400, 350);
        frame.setLayout(null);
        frame.setVisible(true);

        // Closing the Frame Properly
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
           {
                frame.dispose();
            }
        });
    }
}
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ScrollPaneEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ScrollPaneEx
```

AWT ScrollPane Example

```
This is a long text...
Scroll down to read more...
More content here...
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ScrollPaneEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ScrollPaneEx
```

AWT ScrollPane Example

```
s a long text...
 down to read more...
content here...mmmmmmmmmmmmmmmmmmmmmm
```

10. Dialog

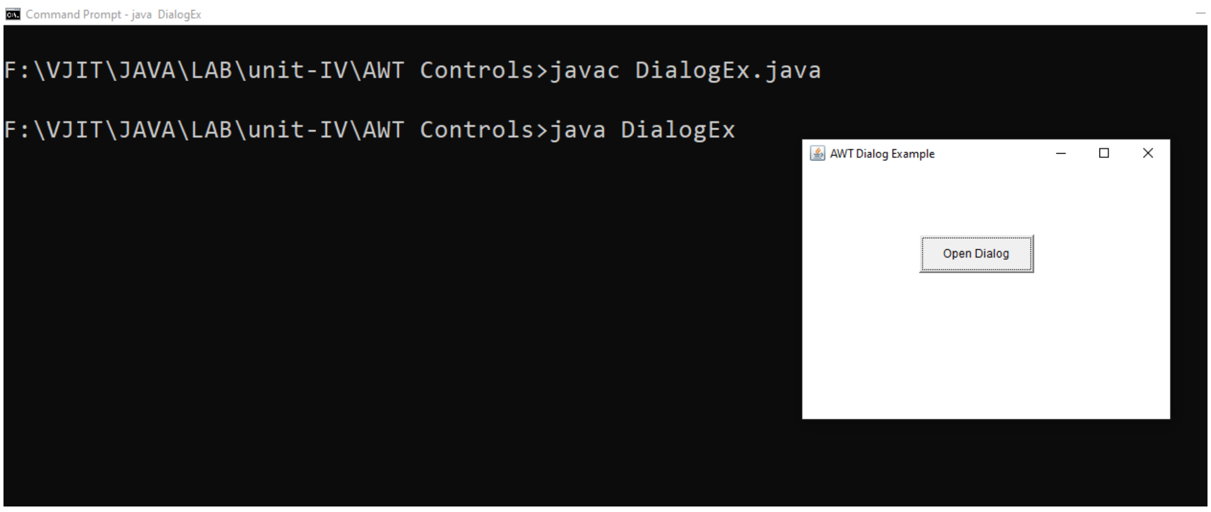
A **Dialog** in **Java AWT (Abstract Window Toolkit)** is a **popup window** that appears on top of a parent window.
It is primarily used for **messages, warnings, or user input dialogs**.

- Requires a **parent Frame** to be created.
- Can be **modal** (blocks interaction with the parent) or **non-modal** (allows interaction).
- Supports adding **buttons, text fields, and other controls**.

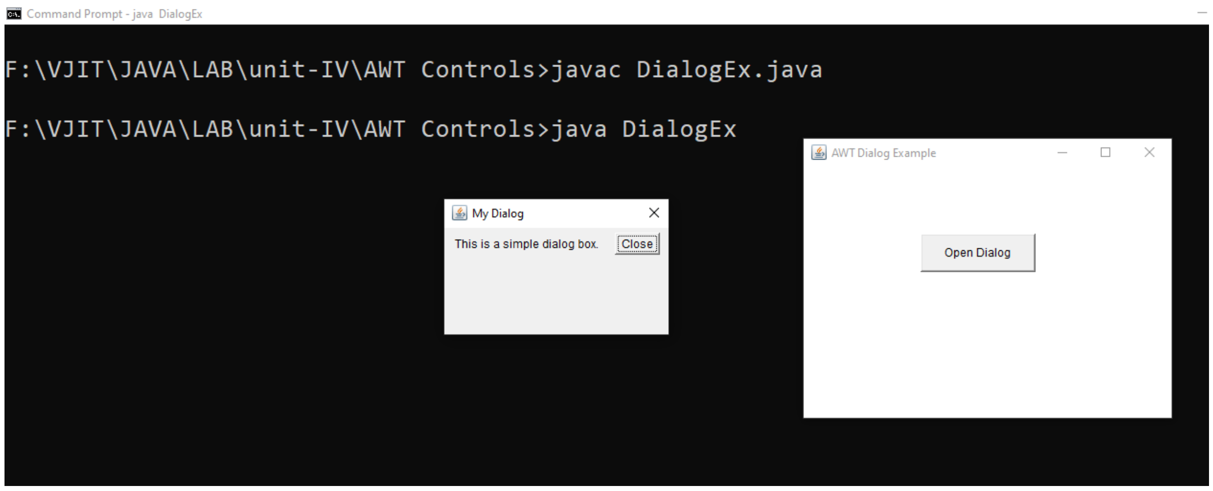
Dialog Class Hierarchy

```
java.lang.Object
  └── java.awt.Component
        └── java.awt.Container
              └── java.awt.Window
                    └── java.awt.Dialog
```

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

public class DialogEx
{
    public static void main(String[] args)
    {
        // Creating a Frame (Parent Window)
        Frame frame = new Frame("AWT Dialog Example");
        frame.setSize(400, 300);
        frame.setLayout(null);

        // Creating a Button to open the Dialog
        Button button = new Button("Open Dialog");
        button.setBounds(130, 100, 120, 40);

        // Creating a Dialog (Child Window)
        Dialog dialog = new Dialog(frame, "My Dialog", true); // Modal Dialog
        dialog.setSize(250, 150);
        dialog.setLayout(new FlowLayout());

        // Adding Label to Dialog
        Label label = new Label("This is a simple dialog box.");
        dialog.add(label);

        // Adding Close Button to Dialog
        Button closeButton = new Button("Close");
        dialog.add(closeButton);

        // Closing Dialog on Button Click
        closeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                dialog.setVisible(false);
            }
        });
        // Showing Dialog on Button Click
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                dialog.setVisible(true);
            }
        });

        // Adding Components to Frame
        frame.add(button);

        // Handling Frame Closing
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                frame.dispose();
            }
        });

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



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac DialogEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java DialogEx
```


```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac DialogEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java DialogEx
```

11. MenuBar

In **Java AWT (Abstract Window Toolkit)**, a **Menu** is used to create **drop-down menu bars**, which contain various menu items (options) to enhance user interaction.

Main Components of AWT Menu

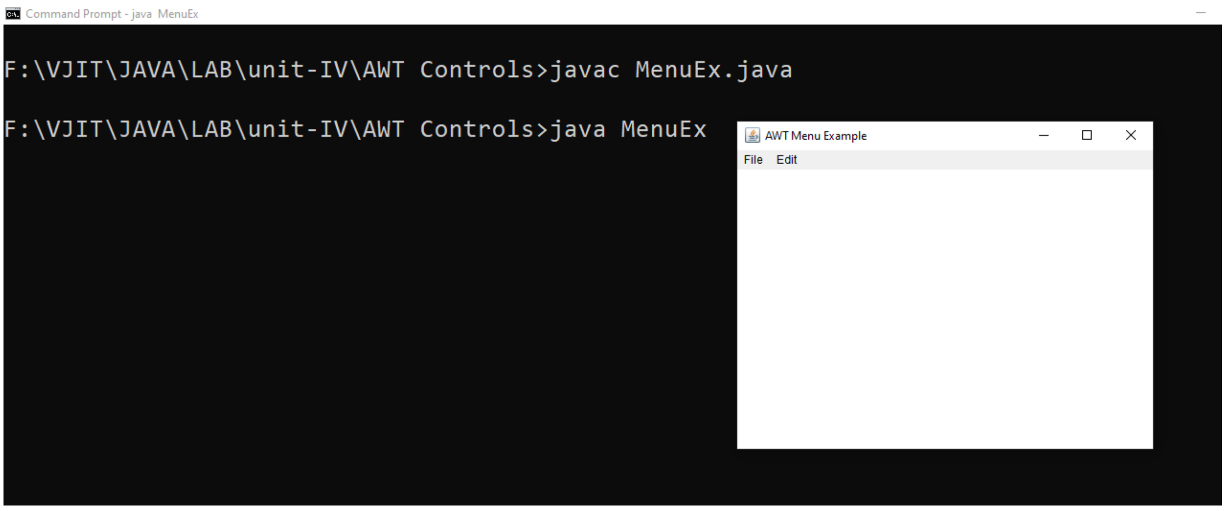
Component	Description
MenuBar	Represents the **top-level menu bar** in a window.
Menu	A **drop-down menu** inside the MenuBar.
MenuItem	Individual **clickable items** inside a Menu.
CheckboxMenuItem	A **checkbox-style** menu item (e.g., "Show Toolbar").

AWT Menu Class Hierarchy

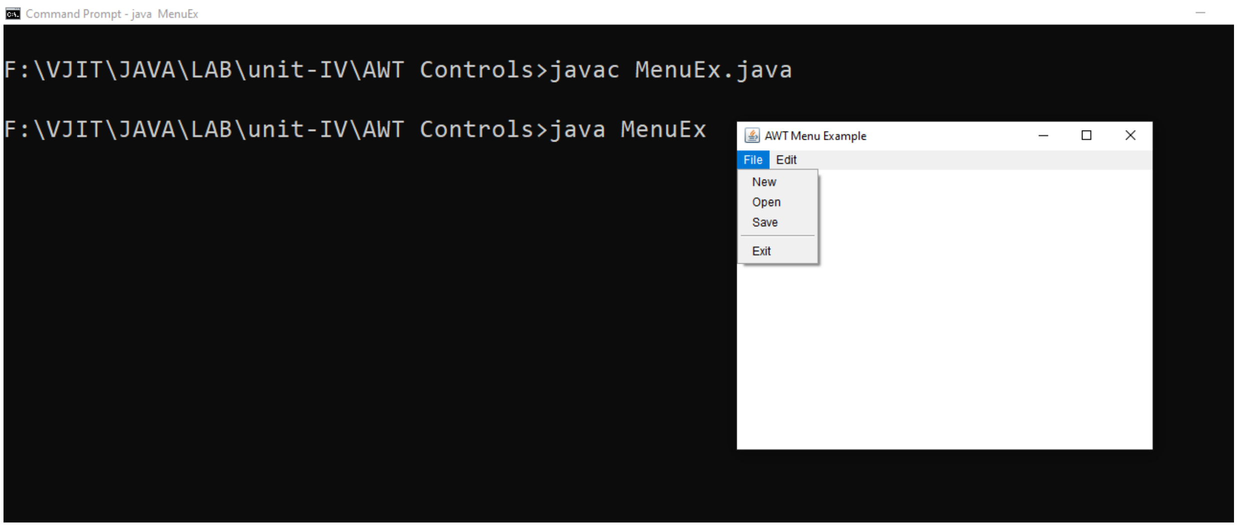
```
java.lang.Object
  └── java.awt.Component
      └── java.awt.MenuComponent
          ├── java.awt.MenuBar
          ├── java.awt.Menu
          ├── java.awt.MenuItem
          ├── java.awt.CheckboxMenuItem
```

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

public class MenuEx
{
   public static void main(String[] args)
    {
      // Creating a Frame
      Frame frame = new Frame("AWT Menu Example");
      frame.setSize(450, 350);

      // Creating a MenuBar
      MenuBar menuBar = new MenuBar();

      // Creating Menus
      Menu fileMenu = new Menu("File");
      Menu editMenu = new Menu("Edit");

      // Creating Menu Items for File Menu
      MenuItem newItem = new MenuItem("New");
      MenuItem openItem = new MenuItem("Open");
      MenuItem saveItem = new MenuItem("Save");
      MenuItem exitItem = new MenuItem("Exit");

      // Adding ActionListener to Exit Item
      exitItem.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e)
        {
            System.exit(0); // Closes the application
         }
      });
      // Adding Menu Items to File Menu
      fileMenu.add(newItem);
      fileMenu.add(openItem);
      fileMenu.add(saveItem);
      fileMenu.addSeparator(); // Adds a separator line
      fileMenu.add(exitItem);
```

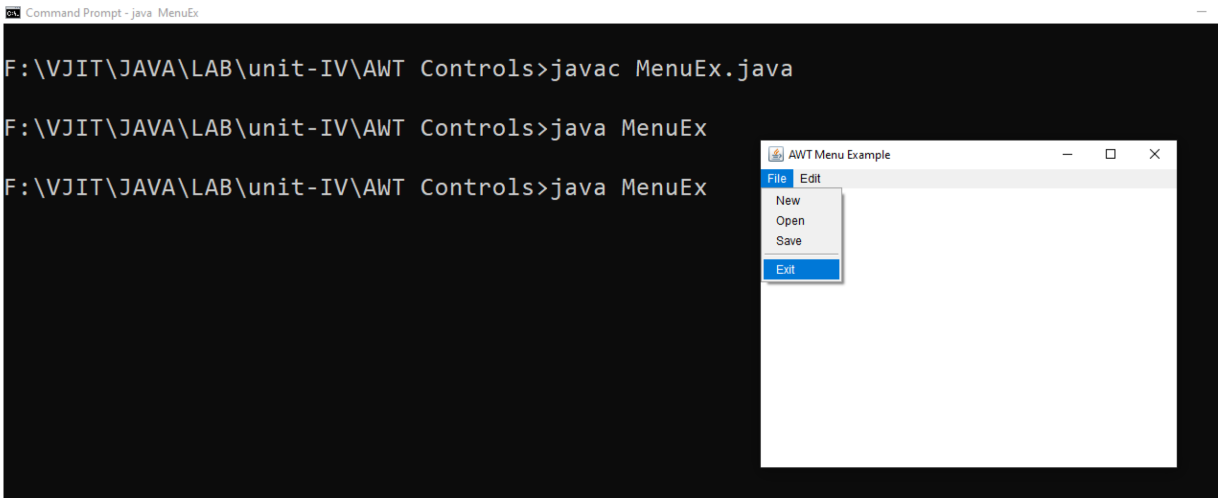
```
        // Creating Menu Items for Edit Menu
        MenuItem cutItem = new MenuItem("Cut");
        MenuItem copyItem = new MenuItem("Copy");
        MenuItem pasteItem = new MenuItem("Paste");

        // Adding Menu Items to Edit Menu
        editMenu.add(cutItem);
        editMenu.add(copyItem);
        editMenu.add(pasteItem);

        // Adding Menus to MenuBar
        menuBar.add(fileMenu);
        menuBar.add(editMenu);

        // Setting MenuBar to Frame
        frame.setMenuBar(menuBar);

        // Handling Window Closing
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
           {
                frame.dispose();
            }
        });

        // Making Frame Visible
        frame.setVisible(true);
    }
}
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac MenuEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java MenuEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac MenuEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java MenuEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac MenuEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java MenuEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java MenuEx
```

Event Handling PART-2

Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes.

Event Handling

Java provides a powerful **event-handling mechanism** to create interactive GUI applications. The **Delegation Event Model** in Java AWT (Abstract Window Toolkit) is used to manage events efficiently.

Event: Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components.

An **event** is an action performed by the user, such as:

✅Clicking a button
✅Moving the mouse
✅Pressing a key
✅Closing a window

When an event occurs, the **Java runtime system** notifies the application, and the appropriate response is executed.

Types of Event

The events can be broadly classified into **two** categories:

1. **Foreground Events:** Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

2. **Background Events:** Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

Event Handling

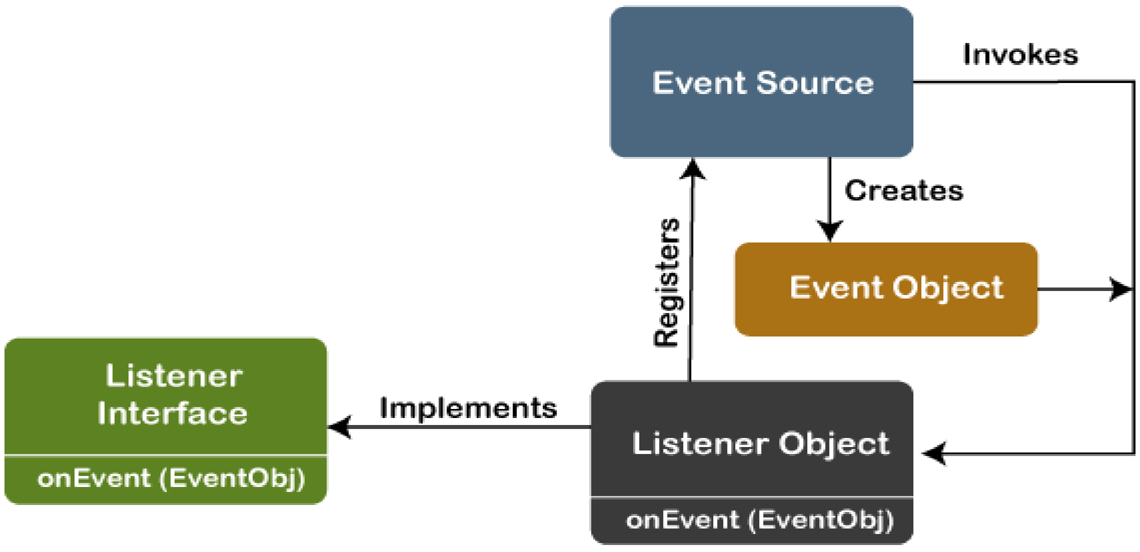
Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. Let's have a brief introduction to this model.

Delegation Event Model

Java follows the **Delegation Event Model**, which has two key components:

1. **Event Source** → Generates an event
2. **Event Listener** → Handles the event

Instead of components handling their own events, **a separate listener class is assigned** to listen and respond to events.



Steps in Delegation Event Model

1. A **component (source)** generates an **event**.
2. The event is passed to an **event listener**.
3. The event listener executes the **callback method**.

Example:

```
button.addActionListener(new MyListener()); // Registers the listener
```

1. **Source -** The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.

   Event Sources

   An **Event Source** is a GUI component that triggers events. Examples:

   - **Button** (click event)
   - **TextField** (text input event)
   - **Window** (window closing event)
   - **Checkbox** (checked/unchecked event)

   Each **Event Source** registers a listener using addXXXListener().

   Example:

   ```
   button.addActionListener(new MyActionListener());
   ```

   This registers MyActionListener to listen for button clicks.

2. Event Classes (Event Objects)

Java provides predefined **event classes** that store information about an event.

Event Class	Description	Generated By
ActionEvent	Button Click, Menu Item Click	Button, MenuItem
MouseEvent	Mouse Click, Movement	Mouse, Panel
KeyEvent	Key Press, Release	Keyboard, TextField
ItemEvent	Checkbox, Choice Selection	Checkbox, Choice
TextEvent	Text Change Event	TextField, TextArea
WindowEvent	Window Open, Close, Minimize	Frame, Dialog

3. **Listener Interface -** It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener process the event and then returns.

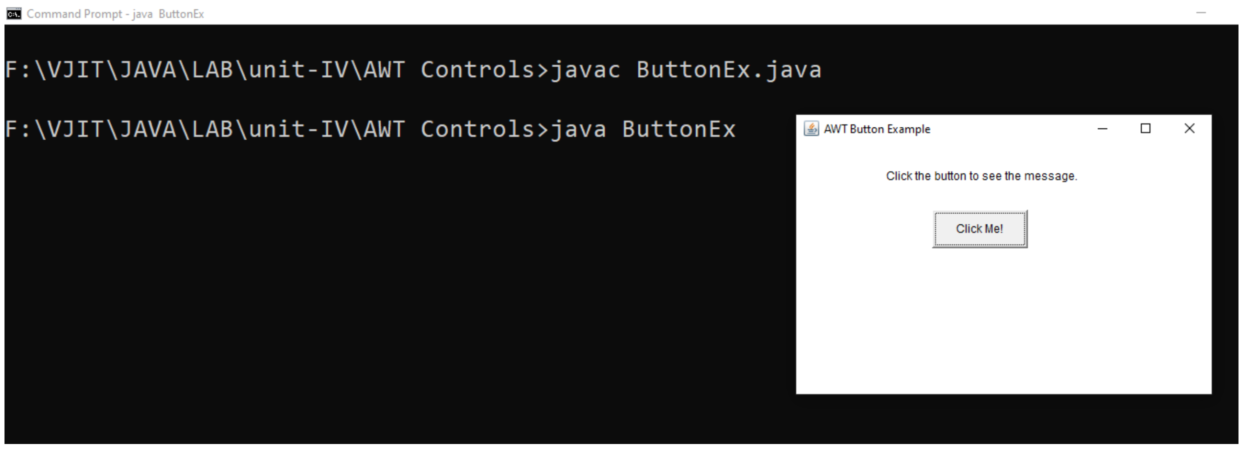
An **Event Listener** is an interface with a method that handles events.

Listener Interface	**Method to Implement**	**Used For**
ActionListener	actionPerformed(ActionEvent e)	Button Click, Menu Click
MouseListener	mouseClicked, mousePressed, mouseReleased, mouseEntered, mouseExited	Mouse Events
KeyListener	keyPressed, keyReleased, keyTyped	Keyboard Input
WindowListener	windowClosing, windowOpened, windowClosed, etc.	Window Events
ItemListener	itemStateChanged(ItemEvent e)	Checkbox, Choice Selection
TextListener	textValueChanged(TextEvent e)	Text Changes

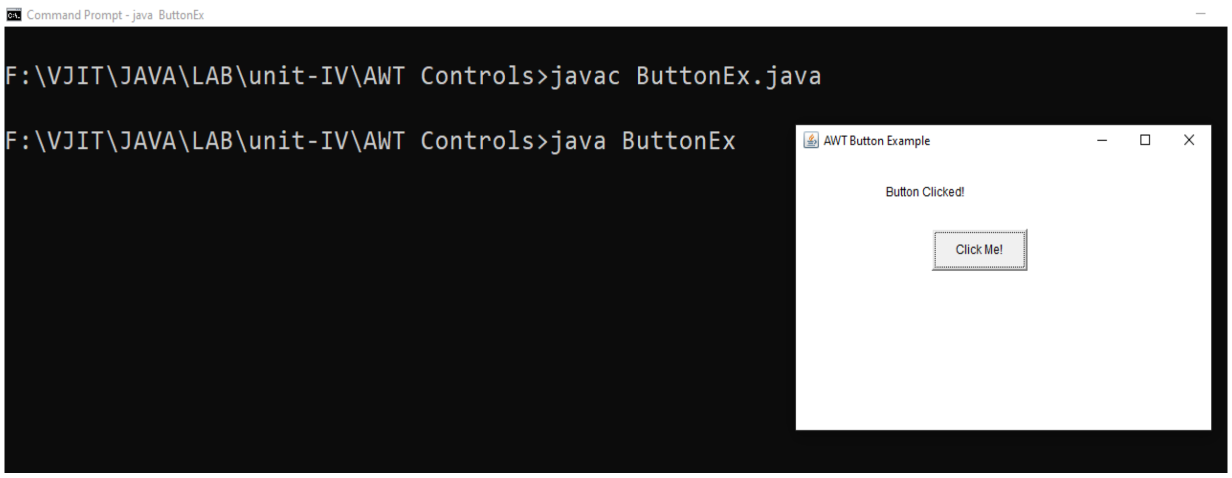
Each **listener** is registered using **addXXXListener()** methods.

```
import java.awt.*;
import java.awt.event.*;
public class ButtonEx
{
   public static void main(String[] args)
  {
      // Creating a Frame
      Frame frame = new Frame("AWT Button Example");
      // Creating a Button
      Button button = new Button("Click Me!");
      button.setBounds(150, 100, 100, 40); // Centering the button
```

```
        // Creating a Label
        Label label = new Label("Click the button to see the message.");
        label.setBounds(100, 50, 250, 30);
        // Adding an ActionListener to the button
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
           {
                label.setText("Button Clicked!"); // Update the label
            }
        });
        // Adding WindowListener to Close the Frame Properly
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
           {
                frame.dispose(); // Close the frame properly
            }
        });
        // Adding Components to the Frame
        frame.add(button);
        frame.add(label);
        // Setting Frame properties
        frame.setSize(450, 300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



```
Command Prompt - java  ButtonEx

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ButtonEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ButtonEx
```



```
F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>javac ButtonEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Controls>java ButtonEx
```

Handling Keyboard Events

Keyboard events are triggered when a user interacts with the keyboard, such as

A **keyboard event** occurs when:

- A key is **pressed** (keyPressed)
- A key is **released** (keyReleased)
- A key is **typed** (keyTyped)

These events can be handled using the **KeyListener** interface.

Common Methods in KeyEvent Class

Method	Description
getKeyChar()	Returns the character of the key that was typed.
getKeyCode()	Returns the keycode of the key that was pressed or released.
getKeyText(int keyCode)	Returns the text description of the key.
isActionKey()	Checks if the key is a special action key (e.g., arrow keys).

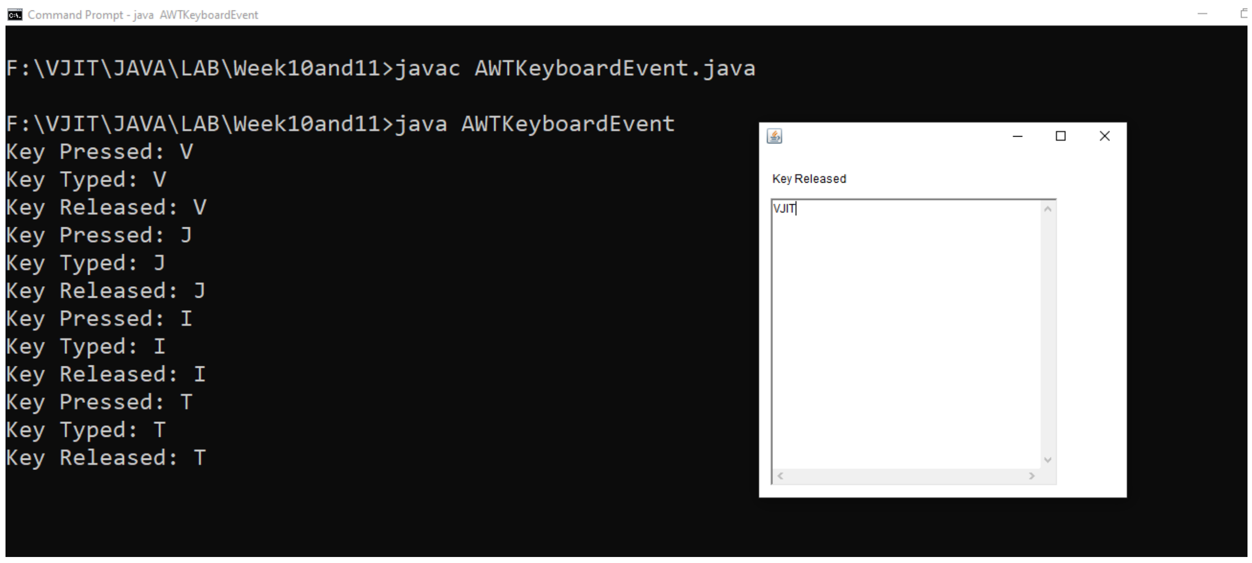
Common Methods in KeyListener Interface

Method	Description
keyPressed(KeyEvent e)	Called when a key is pressed.
keyReleased(KeyEvent e)	Called when a key is released.
keyTyped(KeyEvent e)	Called when a key is typed (only for characters, not special keys).

Program to illustrate **Event Handling** for **Keyboard** Events

```
// importing awt libraries
import java.awt.*;
import java.awt.event.*;

public class AWTKeyboardEvent extends Frame implements KeyListener
{
          Label l;
      AWTKeyboardEvent()
     {
         // creating the label
         l = new Label();
// setting the location of the label in frame
        l.setBounds (20, 50, 100, 20);
// creating the text area
       TextArea area = new TextArea();
// setting the location of text area
        area.setBounds (20, 80, 300, 300);
// adding the KeyListener to the text area
        area.addKeyListener(this);
// adding the label and text area to the frame
        add(l);
       add(area);
// setting the size, layout and visibility of frame
        setSize (400, 400);
        setLayout (null);
        setVisible (true);
        addWindowListener(new WindowAdapter()
          {
            public void windowClosing(WindowEvent we)
              {
               System.exit(0);  // Close the application
             }
        });
    }
// overriding the keyPressed() method of KeyListener interface
    public void keyPressed (KeyEvent e)
   {
       l.setText ("Key Pressed");
       int keyCode = e.getKeyCode();
       System.out.println("Key Pressed: " + KeyEvent.getKeyText(keyCode));

    }

// overriding the keyReleased() method of KeyListener interface
    public void keyReleased (KeyEvent e)
         {
       l.setText ("Key Released");
       int keyCode = e.getKeyCode();
       System.out.println("Key Released: " + KeyEvent.getKeyText(keyCode));

    }

// overriding the keyTyped() method of KeyListener interface
    public void keyTyped (KeyEvent e)
   {
       l.setText ("Key Typed");
             char keyChar = e.getKeyChar();
       System.out.println("Key Typed: " + keyChar);
    }

  // main method
    public static void main(String[] args)
    {
       new AWTKeyboardEvent();
    }
}
```

Output:



```
F:\VJIT\JAVA\LAB\Week10and11>javac AWTKeyboardEvent.java

F:\VJIT\JAVA\LAB\Week10and11>java AWTKeyboardEvent
Key Pressed: V
Key Typed: V
Key Released: V
Key Pressed: J
Key Typed: J
Key Released: J
Key Pressed: I
Key Typed: I
Key Released: I
Key Pressed: T
Key Typed: T
Key Released: T
```

```
VJIT
```

Handling Mouse Events

Mouse Events are triggered when a user interacts with the mouse such as,

A **mouse event** occurs when:

- The mouse is **clicked** (mouseClicked)
- A mouse button is **pressed** (mousePressed)
- A mouse button is **released** (mouseReleased)
- The mouse **enters** a component (mouseEntered)
- The mouse **exits** a component (mouseExited)
- The mouse **moves** (mouseMoved)
- The mouse **drags** (mouseDragged)

These events can be handled using **two interfaces**:

1. **MouseListener** → Handles button clicks and entry/exit events.
2. **MouseMotionListener** → Handles mouse movement and dragging.

1. MouseEvent Class

The MouseEvent class provides methods to detect **mouse actions**.

Common Methods in MouseEvent Class

Method	Description
getX()	Returns the X coordinate of the mouse pointer.
getY()	Returns the Y coordinate of the mouse pointer.
getPoint()	Returns the mouse pointer position as a Point object.
getClickCount()	Returns the number of times the mouse was clicked.
isMetaDown()	Checks if the **right mouse button** was pressed.
isAltDown()	Checks if the **Alt key** was pressed while clicking.

2. MouseListener Interface

The MouseListener interface has **five methods**:

Method	Description
mouseClicked(MouseEvent e)	Called when the mouse is clicked.
mousePressed(MouseEvent e)	Called when a mouse button is pressed.
mouseReleased(MouseEvent e)	Called when a mouse button is released.
mouseEntered(MouseEvent e)	Called when the mouse enters a component.
mouseExited(MouseEvent e)	Called when the mouse exits a component.

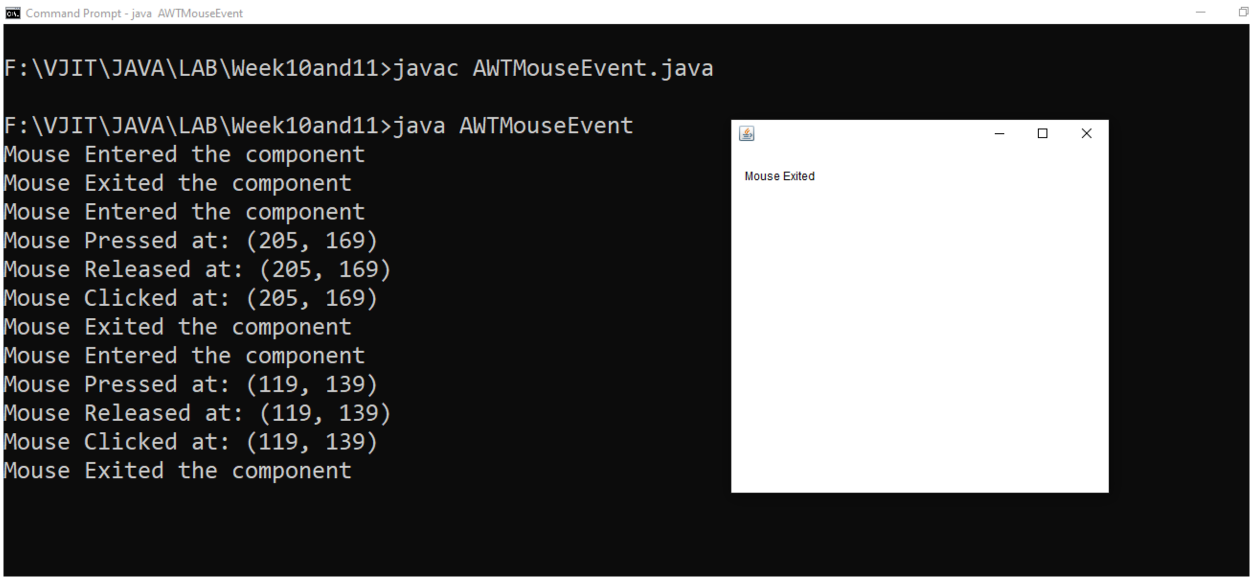
3. MouseMotionListener Interface

Method	Description
mouseMoved(MouseEvent e)	Called when the mouse is moved.
mouseDragged(MouseEvent e)	Called when the mouse is dragged (moved while a button is pressed).

Program to illustrate **Event Handling** for **Mouse** Events

```
import java.awt.*;
import java.awt.event.*;
public class AWTMouseEvent extends Frame implements MouseListener
{
    Label l;
    AWTMouseEvent()
     {
        addMouseListener(this);

        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);  // Close the application
            }
        });
    }
    public void mouseEntered(MouseEvent e)
    {
        l.setText("Mouse Entered");
        System.out.println("Mouse Entered the component");
    }
    public void mouseExited(MouseEvent e)
   {
        l.setText("Mouse Exited");
        System.out.println("Mouse Exited the component");
    }
```

```
  public void mousePressed(MouseEvent e)
  {
      l.setText("Mouse Pressed");
      System.out.println("Mouse Pressed at: (" + e.getX() + ", " + e.getY() + ")");
 }
   public void mouseReleased(MouseEvent e)
  {
      l.setText("Mouse Released");
      System.out.println("Mouse Released at: (" + e.getX() + ", " + e.getY() + ")");
   }
        public void mouseClicked(MouseEvent e)
         {
      l.setText("Mouse Clicked");
      System.out.println("Mouse Clicked at: (" + e.getX() + ", " + e.getY() + ")");
       }

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



```
F:\VJIT\JAVA\LAB\Week10and11>javac AWTMouseEvent.java

F:\VJIT\JAVA\LAB\Week10and11>java AWTMouseEvent
Mouse Entered the component
Mouse Exited the component
Mouse Entered the component
Mouse Pressed at: (205, 169)
Mouse Released at: (205, 169)
Mouse Clicked at: (205, 169)
Mouse Exited the component
Mouse Entered the component
Mouse Pressed at: (119, 139)
Mouse Released at: (119, 139)
Mouse Clicked at: (119, 139)
Mouse Exited the component
```

Adapter Classes

An **Adapter Class** is a class that provides **empty implementations** of methods in an *Event Listener Interface*. These adapter classes help in handling **only the required methods** instead of implementing all methods from an interface.

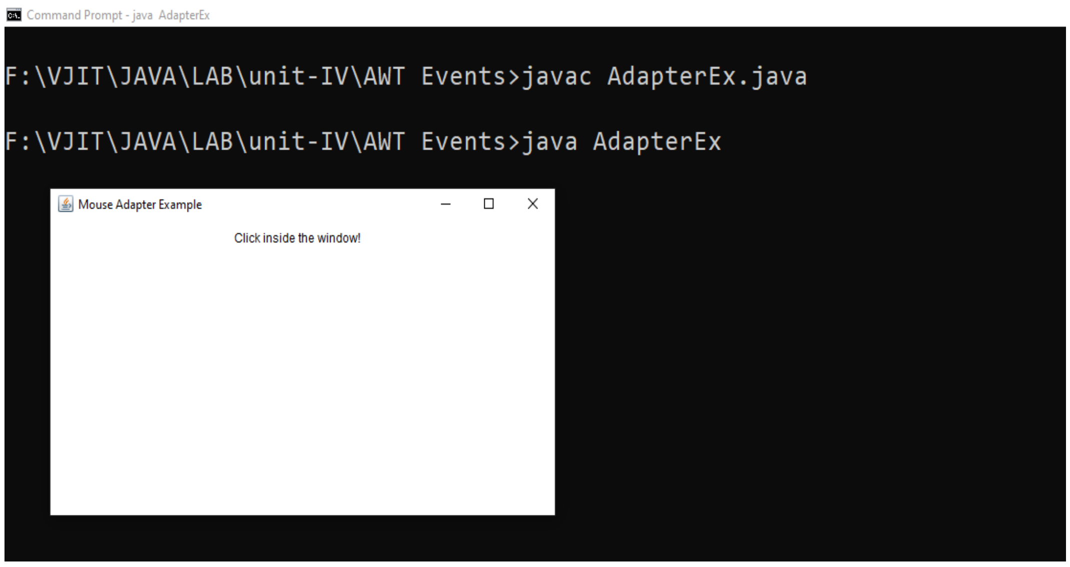
Use of Adapter Classes

- When using event listeners, **all methods** of an interface must be implemented, even if they are not needed.
- Adapter classes provide **default (empty) implementations**, allowing you to override only the methods you need.
- This makes the code **cleaner** and **more readable**.

1. List of Adapter Classes in Java AWT

Adapter Class	Implements This Interface	Purpose
WindowAdapter	WindowListener	Handles window events (closing, opening, etc.).
MouseAdapter	MouseListener	Handles mouse button events.
MouseMotionAdapter	MouseMotionListener	Handles mouse motion (dragging/moving).
KeyAdapter	KeyListener	Handles keyboard events.
FocusAdapter	FocusListener	Handles focus events.
ComponentAdapter	ComponentListener	Handles component resize, move, or visibility changes.
ContainerAdapter	ContainerListener	Handles container events (adding/removing components).

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

public class AdapterEx extends Frame
{

    Label label;

    public AdapterEx()
   {
        setTitle("Mouse Adapter Example");
        setSize(400, 400);
        setLayout(new FlowLayout());

        label = new Label("Click inside the window!");
        add(label);

        // Adding MouseAdapter to handle mouse events
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e)
           {
                label.setText("Mouse Clicked at: (" + e.getX() + ", " + e.getY() + ")");
            }
        });

        // Close window on exit
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
           {
                dispose();
            }
        });

        setVisible(true);
    }
    public static void main(String[] args)
      {
        new AdapterEx();
    }
}
```

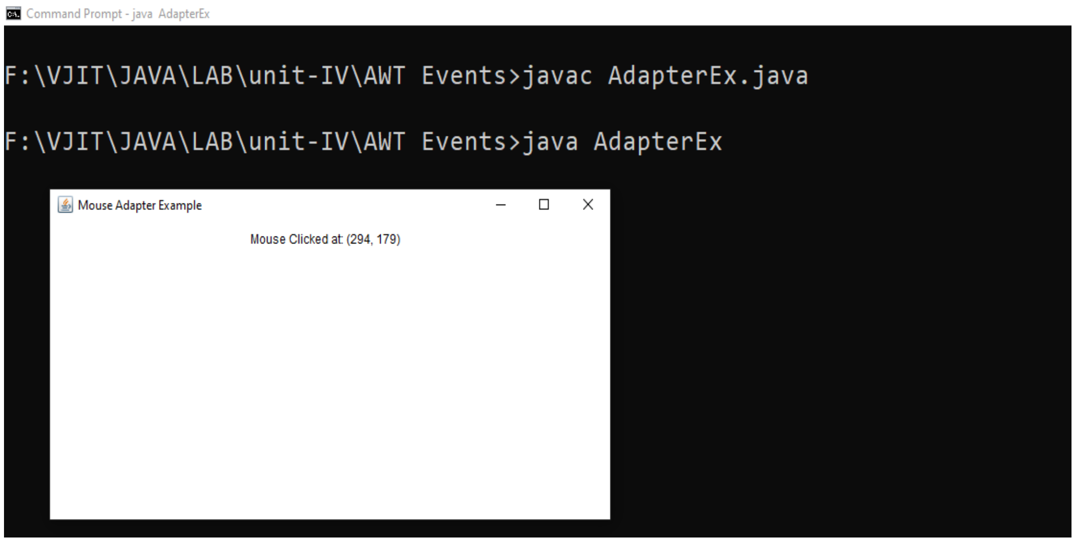


Command Prompt - java AdapterEx

```
F:\VJIT\JAVA\LAB\unit-IV\AWT Events>javac AdapterEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Events>java AdapterEx
```

Mouse Adapter Example

Click inside the window!



Command Prompt - java AdapterEx

```
F:\VJIT\JAVA\LAB\unit-IV\AWT Events>javac AdapterEx.java

F:\VJIT\JAVA\LAB\unit-IV\AWT Events>java AdapterEx
```

Mouse Adapter Example

Mouse Clicked at: (294, 179)