JAVA Lab
Lab Experiment No 12

Name: Aamir Ansari **Batch:** A **Roll no:** 01

Aim: Java program to illustrate the usage of one of the swing components.
Problem statement: Write a Java program to take user response using JCheckBox component from swing.

Theory:
Event handling:
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. For example, clicking on a button, moving the mouse, entering a character through a keyboard, selecting an item from a list, scrolling the page are the activities that cause an event to happen.

Types of Event
The events can be broadly classified into two categories:

Foreground Events - Those events which require the direct interaction of the 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 the keyboard, selecting an item from a list, scrolling the page etc.

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

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 an event handler that is executed when an event occurs.

Event handling has the following key participants namely:

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 provides classes for source objects.

Listener - It is also known as event handler. Listeners are responsible for generating responses 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 processes the event and then returns.
The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

1. The User clicks the button and the event is generated.

2. Now the object of the concerned event class is created automatically and information about the source and the event get populated within the same object.
3. Event object is forwarded to the method of registered listener class.
4. The method is now executed and returns.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener

ContainerEvent	ContainerListener
FocusEvent	FocusListener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

Button

```
public void addActionListener(ActionListener a){}
```

MenuItem

```
public void addActionListener(ActionListener a){}
```

TextField

```
public void addActionListener(ActionListener a){}
public void addTextListener(TextListener a){}
```

TextArea

```
public void addTextListener(TextListener a){}
```

Checkbox

```
public void addItemListener(ItemListener a){}
```

Choice

```
public void addItemListener(ItemListener a){}
```

List

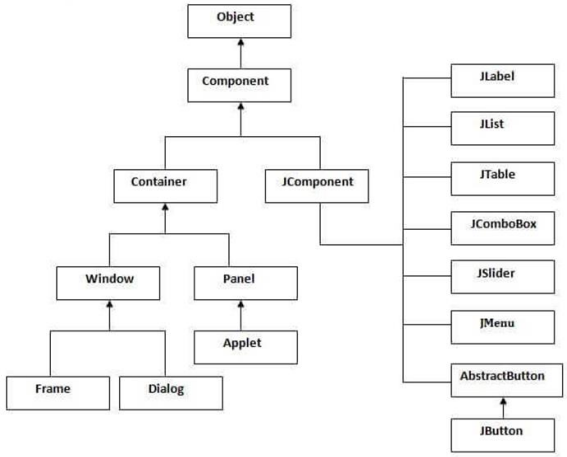
```
public void addActionListener(ActionListener a){}
public void addItemListener(ItemListener a){}
```

Java Swing

1. Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.

2. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
3. Unlike AWT, Java Swing provides platform-independent and lightweight components.
4. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Swing class Hierarchy



Difference between AWT and Swing

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent.**	Java swing components are **platform-independent.**
2)	AWT components are **heavyweight.**	Swing components are **lightweight.**
3)	AWT **doesn't support pluggable look and feel.**	Swing **supports pluggable look and feel.**
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC.**

Features of Swing

Light Weight − Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.

Rich Controls − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.

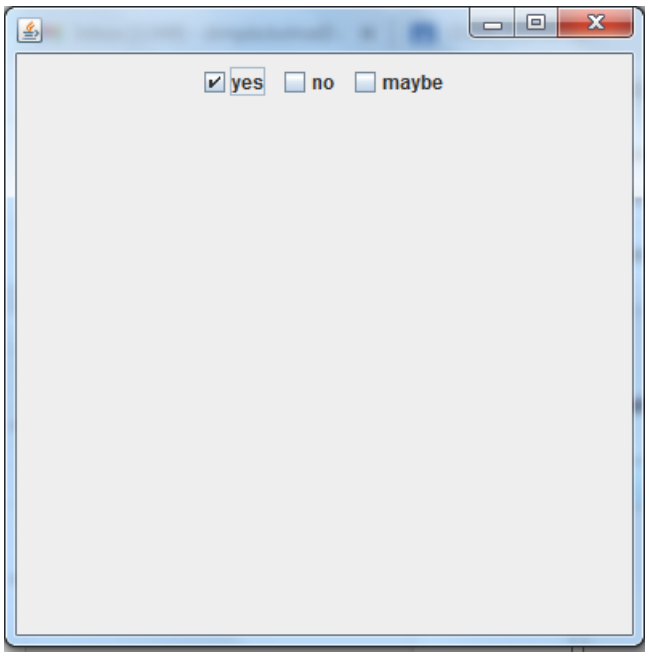
Highly Customizable − Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.

Pluggable look-and-feel − SWING based GUI Application look and feel can be changed at run-time, based on available values.

Program:

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Test_swing extends JFrame
{
  public Test_swing()
  {
    JCheckBox jcb = new JCheckBox("yes");   //creating JCheckBox.
    add(jcb);                               //adding JCheckBox to frame.
    jcb = new JCheckBox("no");              //creating JCheckBox.
    add(jcb);                               //adding JCheckBox to frame.
    jcb = new JCheckBox("maybe");           //creating JCheckBox.
    add(jcb);                               //adding JCheckBox to frame.
    setLayout(new FlowLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 400);
    setVisible(true);
  }
  public static void main(String[] args)
  {
    new Test_swing();
  }
}
```

Output:


yes
no
maybe