Experiment No.: 11

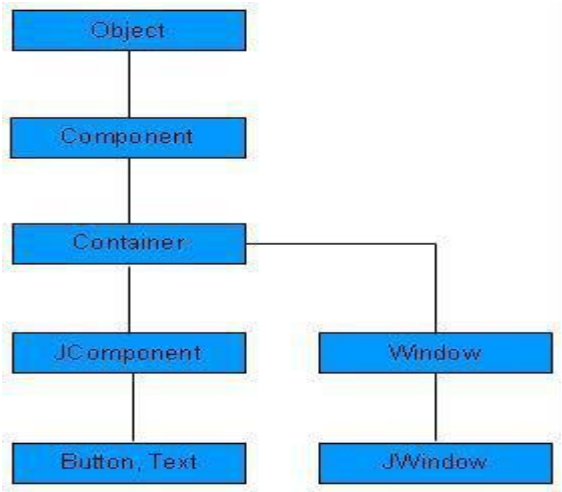
Title: Write a program to develop Swing GUI based standard calculator.

Objectives: 1. To learn how to create windows using swing classes.
2. To learn how to create button using Swing.
3. To learn how to handle event.

Theory:

Swing API is set of extensible GUI Components to ease developer's life to create JAVA based Front End/ GUI Applications. It is built upon top of AWT API and acts as replacement of AWT API as it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfill the following criteria.

- A single API is to be sufficient to support multiple look and feel.
- API is to model driven so that highest level API is not required to have the data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers to use it.



Component

A Container is the abstract base class for the non-menu user-interface controls of SWING. Component represents an object with graphical representation

Container

A Container is a component that can contain other SWING components.

JComponent

A JComponent is a base class for all swing UI components. In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.

Swing UI Elements

Sr.No.	**Control & Description**
1	JLabel A JLabel object is a component for placing text in a
2	Container. JButton This class creates a labeled button.
3	JCheck Box A JCheckBox is a graphical component that can be in either an **on** (true) or **off** (false) state
4	JRadioButton The JRadioButton class is a graphical component that can be in either an **on** (true) or **off** (false) state. in a group.
5	JList A JList component presents the user with a scrolling list of text items.
6	JTextField It is object is a text component that allows for the editing of a single line
7	JTextArea A JTextArea object is a text component that allows for the editing of a multiple lines of text.
8	JColorChooser A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.
9	JPasswordField A JPasswordField object is a text component specialized for password entry
10	ImageIcon A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
11	JScrollbar A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12	JProgressBar As the task progresses towards completion, the progress bar displays the task's percentage of completion.

Events

An *event* is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.

Many other user operations could also be cited as examples. Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, software or hardware failure occurs, or an operation is completed. You are free to define events that are appropriate for your application.

Event Sources

A *source* is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method. Here is the general form:

public void add*Type*Listener(*Type*Listener *el*)

Here, *Type* is the name of the event and *el* is a reference to the event listener. For example, the method that registers a keyboard event listener is called **addKeyListener( )**. The method that registers a mouse motion listener is called **addMouseMotionListener ( )**. When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as *multicasting* the event. In all cases, notifications are sent only to listeners that register to receive them. Some sources may allow only one listener to register. The general form of such a method is this:

public void add*Type*Listener(*Type*Listener *el*)

throws java.util.TooManyListenersException

Here, *Type* is the name of the event and *el* is a reference to the event listener. When such an event occurs, the registered listener is notified. This is known as *unicasting* the event.

A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:

public void remove*Type*Listener(*Type*Listener *el*)

Here, *Type* is the name of the event and *el* is a reference to the event listener.

For example, to remove a keyboard listener, you would call **removeKeyListener( )**.The methods that add or remove listeners are provided by the source that generates events. For example, the **Component** class provides methods to add and remove keyboard and mouse event listeners.

Event Listeners

A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive

notifications about specific types of events. Second, it must implement methods to receive and process these notifications.

The methods that receive and process events are defined in a set of interfaces found in **java.awt.event**. For example, the **MouseMotionListener** interface defines two methods to receive notifications when the mouse is dragged or moved. Any object may receive and process one or both of these events if it provides an implementation of this interface. Many other listener interfaces are discussed later in this and other chapters.

Key Concepts: Awt,Swing , Buttons, setLayout, EventHandling.

Algorithm:

1) Create a class that implements ActionListener.
2) Create frame by creating object of JFrame class
3) Create buttons and text field using the JButton and JTextField class.
4) Using setBounds methods specify location of the buttons and textField in the frame.
5) Add the buttons and textfield in the frame using frame class object.
6) In actionPerformed method define actions to be performed when buttons are pressed.
7) If button pressed is equal button perform appropriate function and display results in textfield.

Note: Please follow the naming conventions while writing the program.