Component

- A component is an object that can be displayed on the screen and can interact with the user.
- It is like a button or scroll bar that has visual representation in a screen window It has special property like background color and font color.
- Component is an abstract class that encapsulate all of the attributes of a visual component.
- •All user interface elements that are displayed on the screen and that interact with the user are subclasses of “Component”.
- A component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container

- Container:- It is a subclass of Component class. It has additional methods that allow other Component objects to be nested within it.
- Other Container objects can be stored inside of a Container (since they are themselves instance of Component) It uses various Layout Managers.

Panel

- Panel:- It is a concrete subclass of a Container.
- It doesn’t add any new methods; it simply implements Container.
- Panel is a superclass for Applet.
- When screen output is directed to an applet,it is drawn on the surface of a Panel object.
- Panel is a window that does not contain a title bar, Menu bar, or border.
- When you run an applet using an appletviewer, the applet viewer provides the title and border.

Panel

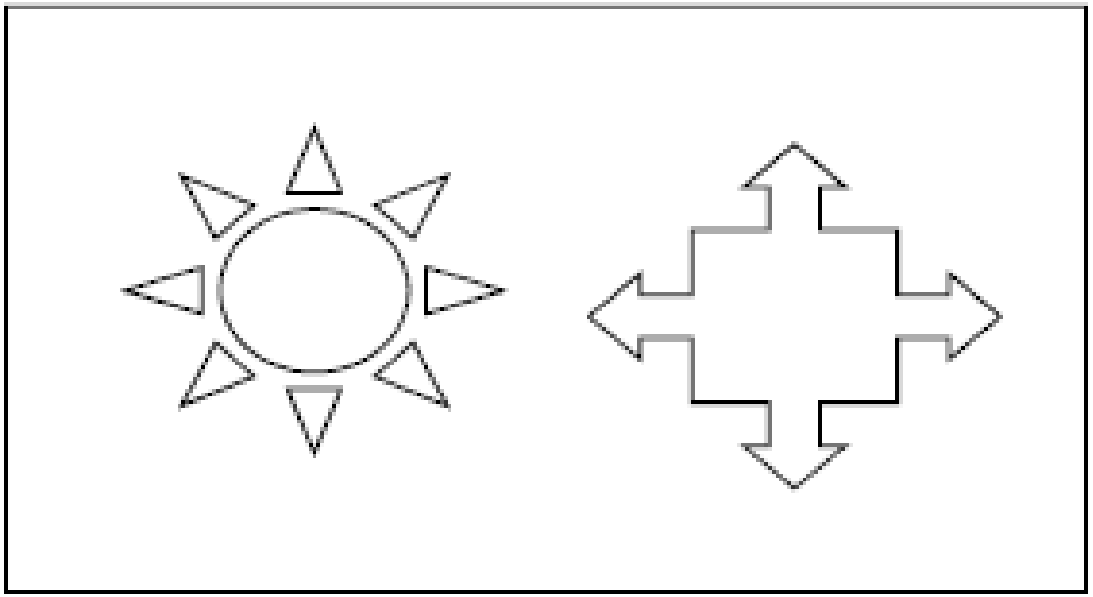
- Other components can be added to a panel object by its add() method (inherited from container).
- Once these components have been added,you can position and resize them manually using the setLocation(), setSize(), or setBounds() methods defined by Component.

Windows

- Window:- The window class creates a top-level window.
- A top-level window is not contained within any other object; it sits directly on the desktop
- Generally we won’t create window objects directly, instead will use a subclass of Window called Frame.
-

Canvas

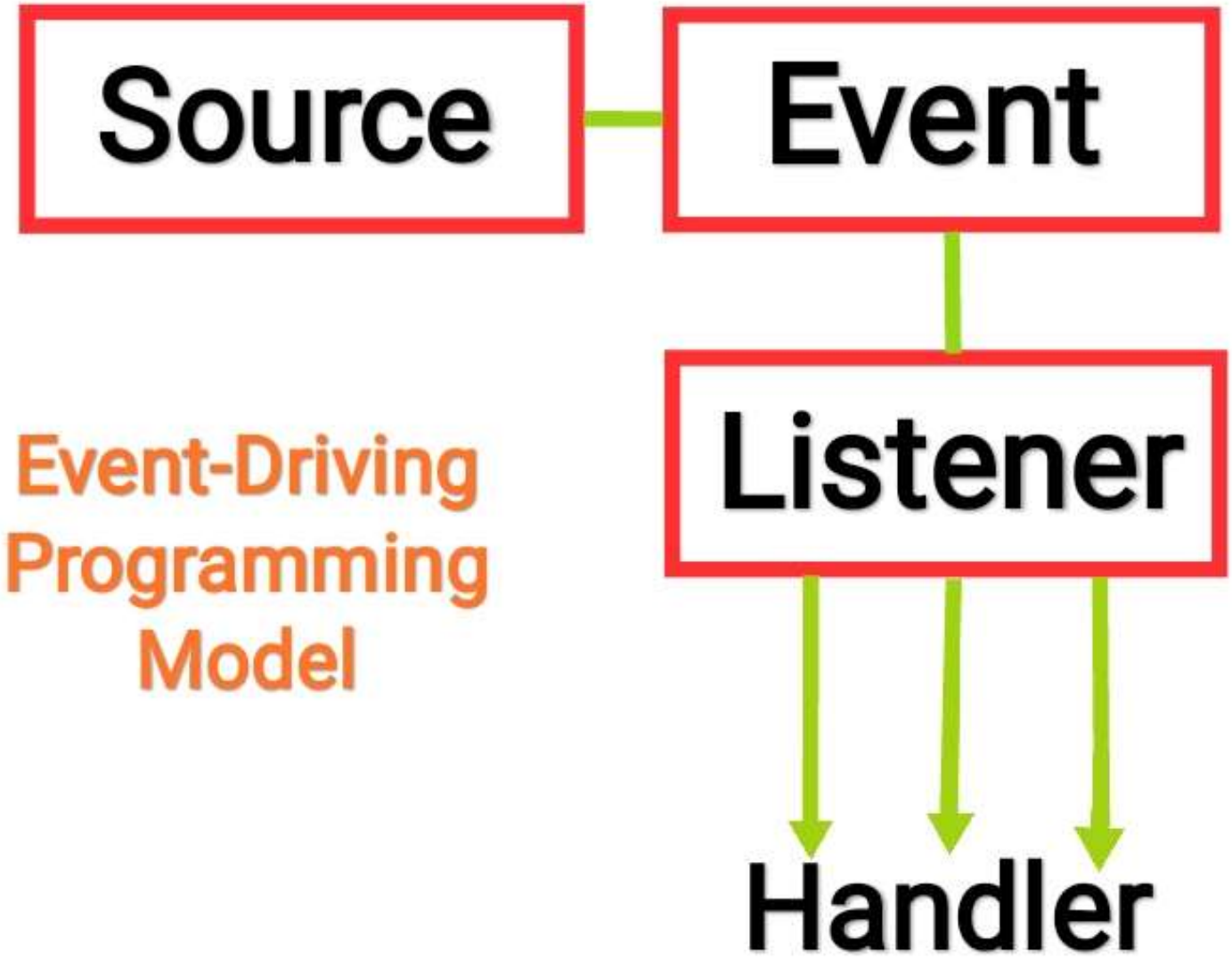
- Canvas:- Although it is not part of the hierarchy for applet or frame window, there is one another type of window called Canvas.
- Canvas encapsulates a blank window upon which you can draw.



Delegation Event Model

- The delegation event model, which defines standard and consistent mechanisms to generate and process events.
- Here a source generates an event and sends it to one or more listeners. In this,the listener simply waits until it receives an event.
- Once an event is received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
- A user interface element is able to “delegate” the processing of an event to a separate piece of code.

Delegation Event Model



Event-Driving Programming Model

Delegation Event Model

- The reasons for coming out with the Java Delegation Event Model are:

1. A source generates an event and sends it to one or more listeners.
2. In this scheme, the listener simply waits until it receives an event.
3. Once received, the listener processes the event and then return.

Delegation Event Model

4. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
5. A user interface element is able to “delegate” the processing of an event to a separate piece of code.
6. In this event model, listener must register with a source in order to receive an event notification. This provide an important benefit: notification are sent only to listeners that want to receive them.

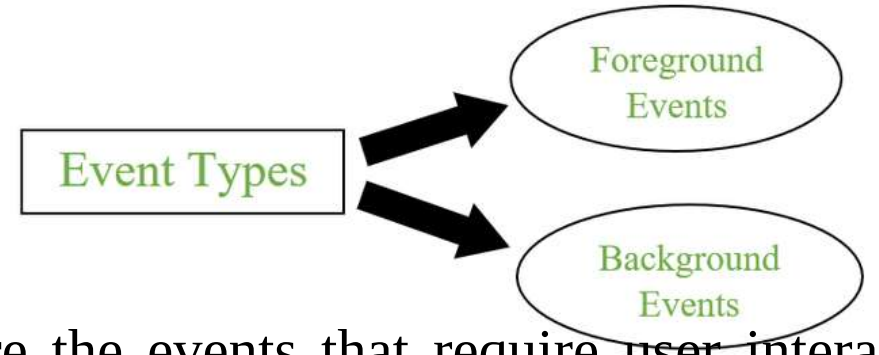
Delegation Event Model

1. What is An Event ?

- It is an object which describes a change in a source.
- Basically they are associated with a component.
- Some examples of the events are Java Button Pressed Event, entering a character via keyboard, selecting an item in a list , clicking the mouse etc.
- Events are supported by number of packages including java.util, java.awt, java.awt.event .

Delegation Event Model

1. What Classification of Events



- Foreground Events

1. 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.

Delegation Event Model

1. Event Source.
2. A source is an object that generates an Event .
3. This happens when the internal state of the object changes in some way.
4. A source can generate one or more types of events.
5. A method getSource() is associated with all events which returns the object on which the event is initially occurred.

Delegation Event Model

1. Event Listener

- It is an object which is notified when an event occurs.

Listeners are Java interfaces and when we implement an interface by a class we must implement all interface's methods because all the methods in an interface are abstract .

- Implementing an interface without adding its required abstract methods result in a syntax error .
- A listener has two major requirements,

1. It must have been registered with one or more sources to receive notifications about specific types of events.
2. It must implement methods to receive and process these notifications.

Methods of component class

Methods of component class-

1. void setVisible(boolean);
2. void show();
3. void hide();
4. void enable(boolean)
5. void disable(boolean)
6. void setSize(int w,int height);
7. void setLocation(int left, int top);
8. void setBounds(int left,int top, int w,int h);
9. Add(object of control)

Steps for creating GUI

Steps for creating GUI:

- Create object of container
- Create objects of control
- Customize appearance of container and controls
- Add controls on container
- Show container
- Event Delegation Model is based on four concepts:

1. The Event Classes
2. The Event Listeners
3. Explicit Event Enabling
4. Adapters

Event Model : four concepts

- At the root of the java event class hierarchy is EventObject, which is an java.util.
- It is the superclass for all events.
- EventObject contains two methods: getSource() and toString()
- getSource() - method returns the source of the event.
- toString() - returns the string equivalent of the event.

1. Event Class

- ActionEvent : generated by component activation
- AdjustmentEvent : generated by adjustment of adjustable components such as scroll bars
- ContainerEvent : generated when components are added to or removed from a container
- FocusEvent : generated when a component receives input focus

1. Event Class

- ItemEvent : generated when an item is selected from a list,choice or check box
- KeyEvent : generated by keyboard activity
- MouseEvent : generated by mouse activity
- PaintEvent : generated when a component is painted
- TextEvent :generated when a text component is modified
- WindowEvent : generated by window activity like minimizing or maximizing

2. Event Listener

- Listeners for Different Events
- For each event type that can occur, the application can add event listeners, that have methods invoked when the event occurs.
- The listeners are defined as interfaces so that an actual listener has to implement these methods.
- The names of the listener classes are simply derived from the names of the events they handle, • MouseEvent's have two different types of listener for efficiency reasons.

(MouseListener and MouseMotionListener)

Event Handling Steps

- Listener- Compare and matches the application with its event.
- Every listener in java is an interface .
- Hence, need to implement.

Steps:

1. Create a Listener class.
2. Give body to all the methods of listener Interfaces.
3. Register the control on which the event is generated. (b.addActionlistner(this)).

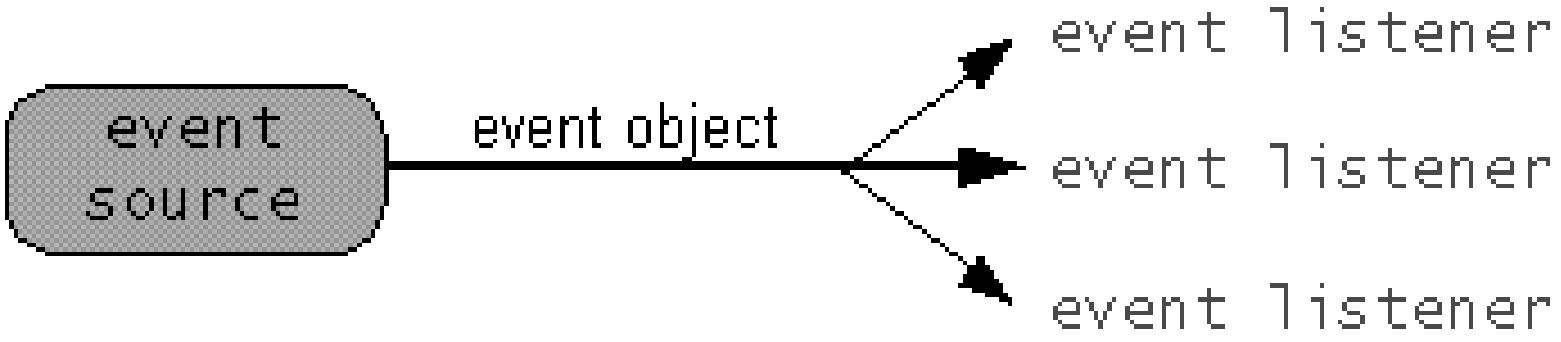
Event Handling Steps

- An event listener may be removed from an Event Source's list of interested Listeners by calling a remove...Listener() method, passing in the listener object to be removed.
- For example in the above code fragment the code below removes the action listener object listenerObject from the button m_Button.
- m_Button.removeActionListener(listenerObject);

Listeners

- Any number of event listener objects can listen for all kinds of events from any number of event source objects.
- e.g. a program might create one listener per event source or a program might have a single listener for all events from all sources.
- Multiple listeners can register to be notified of events of a particular type from a particular source.
- Also, the same listener can listen to notifications from different objects.

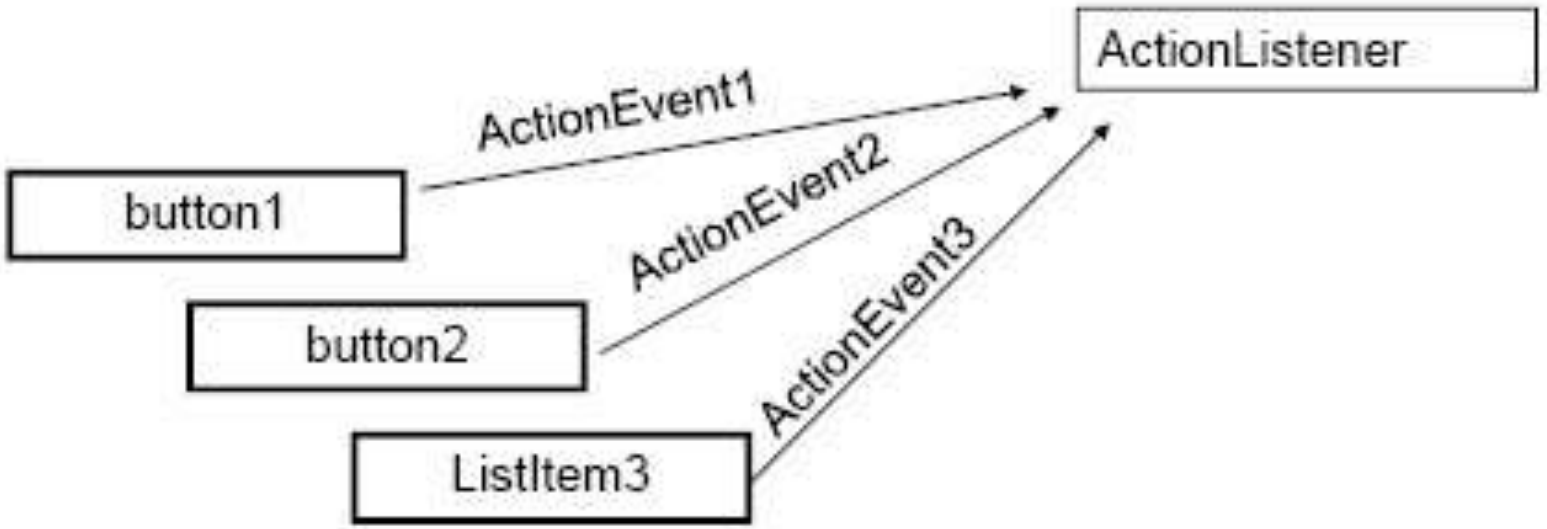
Listeners



•Each type of listeners can be notified only for its corresponding types of events which can be generated by specific types of sources.

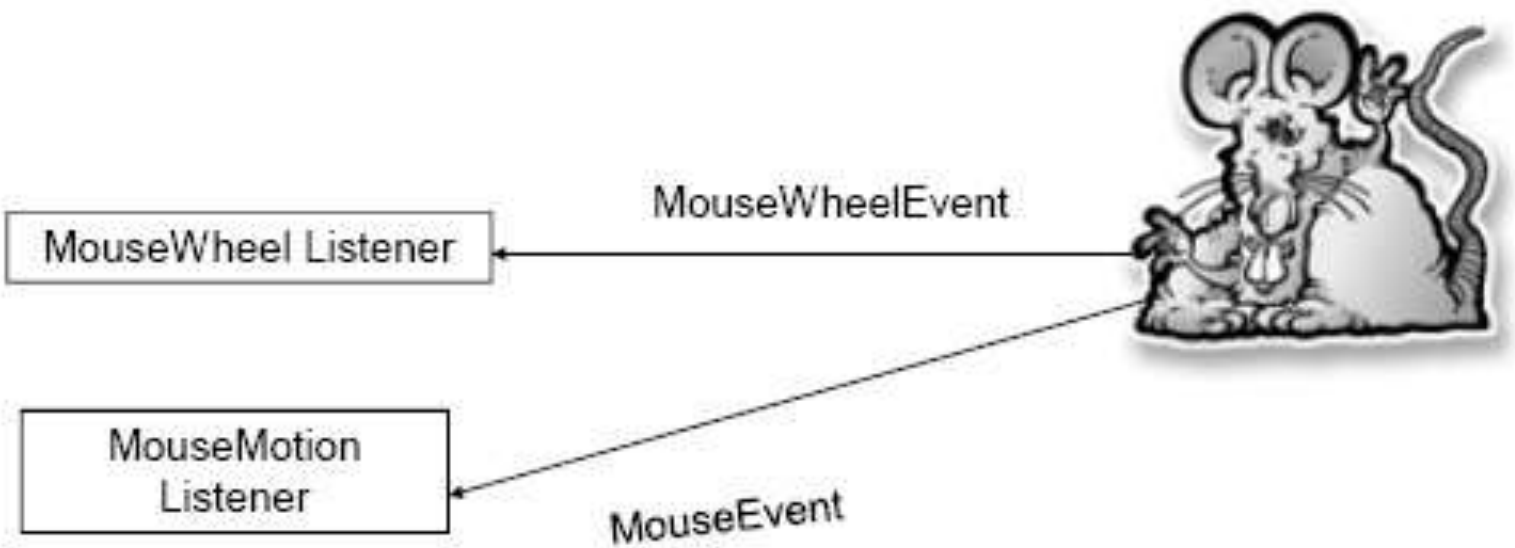
Multiple sources, single listener

- Many buttons can register the same listener since all buttons generate the same type of event.
- This type of event may be generated by other types of sources as well.



Single source, multiple listeners

- A single source may generate different types of events and thus register multiple listeners.



Listeners as interfaces

- You implement an interface to create a listener.
- In the case of a single source that generates multiple types of events you can create a single listener that implements all interfaces
- (remember: a class may extend only one superclass but implement more than one interfaces).
- You register a listener using the corresponding add function in the form:

```
component.addSomeListener(listener_object);
```

- You can find the source of an event by using the getSource method.

```
event_object.getSource();
```

Listeners for Each Event

EVENT	Listener	Method
ActionEvent	ActionListener	actionPerformed()
AdjustmentEvent	AdjustmentListener	adjustmentValueChang Ed()
ComponentEvent	ComponentListener	componentResized() componentMoved() componentShown()
FocusEvent	FocusListener	focusGained() focusLost()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyTyped() keyPressed() keyReleased()

Listeners for Each Event

MouseEvent	**MouseListener**	**mouseClicked()** **mouseEntered()** **mouseExited()** **mousePressed()** **mouseReleased()**
	MouseMotionListener	**mouseDragged()** **mouseMoved()**
WindowEvent	**WindowListener**	**windowClosed()** **windowClosing()** **windowDeiconified()** **windowIconified()** **windowOpened()**

Steps To Use Swing

- Create a container
  - Frame, Dialog, Window, Panel, ScrollPane
- Select a LayoutManager
  - Flow, Border, Grid, GridBag, Card, none (null)
- Create components
  - JButton, JCheckbox, JLabel, JList, JTextArea,JTextField etc.
- Add components to container
- Specify event handling
  - listeners are objects interested in events
  - sources are objects that “fire” events
  - register listeners with sources