CHAPTER

3

Event Handling

Syllabus Topics

The delegation Event Model : Event sources, Event listeners, Event classes : The Action Event class, the Item Event class, the Key Event class, the Mouse Event class, the Text Event class, the Window Event class. Adapter classes, Inner classes, Event listener interfaces : ActionListener Interface, ItemListener Interface, KeyListener Interface, MouseListener Interface, MouseMotion Interface, TextListener Interface, WindowsListener Interface, WindowFocusListener Interface.

Syllabus Topic : The Delegation Event Model - Event Sources, Event Listeners

3.1 The Delegation Event Model : Event Sources, Event Listeners

Q. 3.1.1 Write note on Event Handling. (Ref. Sec. 3.1) **(Likely Ques)**

- The delegation event model is based on the **Event Source** and **Event Listeners**. We will see all the concepts regarding delegation event model :

☞ **Event**

- Changing the state of an object is known as an **event**.
- It can be triggered by typing in a text field, selecting an item from the menu etc.
- Events may also be triggered when timer expires, hardware or software failure occurs, operation completes, counter is increased or decreased by a value etc.
- **Event handler :** The code which gets executed in response to an event is known as event handler.
- **Event handling :** The mechanism of responding to events is called as Event Handling.
- **Event Source :** It is an object which is responsible for generating the event(s).
- More often the event source is a button or any other component on which user can click but sometimes it may also be swing component.
- The work of the event source is to accept registrations, get events from the user and invoke the listener's event handling method.
- **Event Listener :** It is an object which listens the events and handles them with proper mechanism.
- It can be considered as a consumer which receives events from the source.
- The work of an event listener is to implement the interface, register with the source and provide the event handling.
- **Listener interface :** It is an interface having the methods which the listener has to implement.

Syllabus Topic : Event Classes

3.2 Event Classes

The java.awt.event package provides number of event classes to handle events.

Syllabus Topic : The Action Event Class

3.2.1 The Action Event Class

Q. 3.2.1 Enlist constructors and methods of ActionEvent class. (Ref. Sec. 3.2.1) **(Likely Ques)**

- Action Event class is member of java.awt.event package.
- The ActionEvent is generated when button is clicked or the list item is double clicked.
- This respective event is transferred to every ActionListener object that is registered to receive such events using the component's addActionListener method.

Constructors

1. **ActionEvent(java.lang.Object source, int id, java.lang.String command)**

// Creates an ActionEvent object.

2. **ActionEvent(java.lang.Object source, int id, java.lang.String command, int modifiers)**

// Creates an ActionEvent object with specified modifier keys.

3. **ActionEvent(java.lang.Object source, int id, java.lang.String command, long when, int modifiers)**

// Creates an ActionEvent object with the given modifier keys and timestamp.

Methods

1. **String getActionCommand()**

// Returns the command string associated with this action.

2. **int getModifiers()**

// Returns the modifier keys held down during this action event.

3. **long getWhen()**

// Returns the timestamp of when this event occurred.

4. **String paramString()**

// Returns a parameter string identifying this action event.

3.2.2 The Component Event Class

Q. 3.2.2 Enlist constructors and methods of Component Event class. (Ref. Sec. 3.2.2) **(Likely Ques)**

- This is considered as a low level event which implies that a component has been moved, changed size, or changed visibility.
- This respective event is transferred to every ComponentListener or ComponentAdapter object which has been registered to receive these types of events with the help of the component's addComponentListener method.

Constructors

1. **ComponentEvent(Component source, int id)**

// Constructs a ComponentEvent object.

Methods

1. **Component getComponent()**

// Returns the originator of the event.

2. **String paramString()**

// Returns a parameter string identifying this event.

3.2.3 The Container Event Class

Q. 3.2.3 Enlist constructors and methods of ContainerEvent class. (Ref. Sec. 3.2.3) **(Likely Ques)**

- This is considered as a low level event which implies that a container's contents have been changed since a component was added or removed.
- Container events are provided for notification.

☞ **Constructors**

1. **ContainerEvent(Component source, int id, Component child)**

// Constructs a ContainerEvent object.

☞ **Methods**

1. **Component getChild()**

// Returns the component that was affected by the event.

2. **Container getContainer()**

// Returns the originator of the event.

3. **String paramString()**

// Returns a parameter string identifying this event.

3.2.4 The Focus Event Class

Q. 3.2.4 Enlist constructors and methods of FocusEvent class. (Ref. Sec. 3.2.4) **(Likely Ques)**

- This is considered as a low level event which implies that a Component has gained or lost the input focus.
- This low-level event is generated by a Component (such as a TextField).
- This respective event is transferred to every FocusListener or FocusAdapter object which has been registered to receive these types of events with the help of the Component's addFocusListener method.
- (FocusAdapter objects implement the FocusListener interface.) Each such listener object gets this FocusEvent when this event is fired.

☞ **Constructors**

1. **FocusEvent(Component source, int id)**

// Constructs a FocusEvent object and identifies it as a permanent change in focus.

2. **FocusEvent(Component source, int id, boolean temporary)**

// Constructs a FocusEvent object and identifies whether or not the change is temporary.

3. **FocusEvent(Component source, int id, boolean temporary, Component opposite)**

// Constructs a FocusEvent object with the specified temporary state and opposite Component.

☞ **Methods**

1. **Component getOppositeComponent()**

// Returns other Component involved in this focus change.

2. **boolean is Temporary()**

// Identifies the focus change event as temporary or permanent.

3. **String paramString()**

// Returns a parameter string identifying this event.

Syllabus Topic : The Item Event Class

3.2.5 The Item Event Class

Q. 3.2.5 Enlist constructors and methods of ItemEvent class. (Ref. Sec. 3.2.5) **(Likely Ques)**

- A semantic event which indicates that an item was selected or deselected.
- This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.
- This respective event is transferred to every ItemListener object which has been registered to receive these types of events with the help of the component's addItemListener method.
- The object that implements the ItemListener interface gets this ItemEvent when this event is fired.

☞ **Constructors**

1. **ItemEvent(ItemSelectable source, int id, Object item, int stateChange)**

// Constructs an ItemEvent object.

☞ **Methods**

1. **Object getItem()**

// Returns the item affected by the event.

2. **ItemSelectable getItemSelectable()**

// Returns the originator of the event.

3. **int getStateChange()**

// Returns the type of state change (selected or deselected).

4. **String paramString()**

// Returns a parameter string identifying this item event.

Syllabus Topic : The Key Event Class

3.2.6 The Key Event Class

Q. 3.2.6 Enlist constructors and methods of KeyEvent class. (Ref. Sec. 3.2.6) **(Likely Ques)**

- An event which indicates that a keystroke is occurred in a component.
- This low-level event is generated by a component object (such as a text field) when a key is pressed, released, or typed.
- This respective event is transferred to every KeyListener or KeyAdapterobject which has been registered to receive these types of events with the help of the component's addKeyListener method.
- KeyAdapter objects implement the KeyListener interface. Each such listener object gets this KeyEvent when this event is fired.

☞ **Constructors**

1. **KeyEvent(Component source, int id, long when, int modifiers, int keyCode, char keyChar)**

// Constructs a KeyEvent object.

2. **KeyEvent(Component source, int id, long when, int modifiers, int keyCode, char keyChar, int keyLocation)**

// Constructs a KeyEvent object.

☞ **Methods**

1. **int getExtendedKeyCode()**

// Returns an extended key code for the event.

2. **static int getExtendedKeyCodeForChar(int c)**

// Returns an extended key code for a unicode character.

3. **char getKeyChar()**

// Returns the character associated with the key in this event.

4. **int getKeyCode()**

// Returns the integer keyCode associated with the key in this event.

5. **int getKeyLocation()**

// Returns the location of the key that originated this key event.

6. **static String getKeyModifiersText(int modifiers)**

// Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".

7. **static String getKeyText(int keyCode)**

// Returns a String describing the keyCode, such as "HOME", "F1" or "A".

8. **boolean isActionKey()**

// Returns whether the key in this event is an "action" key.

9. **string paramString()**

// Returns a parameter string identifying this event.

10. **void setKeyChar(char keyChar)**

// Set the keyChar value to indicate a logical character.

11. **void setKeyCode(int keyCode)**

// Set the keyCode value to indicate a physical key.

Syllabus Topic : The Mouse Event Class

3.2.7 The Mouse Event Class

Q. 3.2.7 Enlist constructors and methods of MouseEvent class. (Ref. Sec. 3.2.7) **(Likely Ques)**

- An event which indicates that a mouse action is occurred in a component.
- A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the un-obscured part of the component's bounds when the action happens.
- A mouse event type is enabled by adding the appropriate mouse-based EventListener to the component (MouseListener or MouseMotionListener)

Constructors

1. **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger)**

// Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, click count, and popupTrigger flag.

2. **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, int button)**

// Constructs a MouseEvent object with the specified source component, type, time, modifiers, coordinates, click count, popupTrigger flag, and button number.

3. **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int xAbs, int yAbs, int clickCount, boolean popupTrigger, int button)**

// Constructs a MouseEvent object with the specified source component, type, time, modifiers, coordinates, absolute coordinates, click count, popupTrigger flag, and button number.

Methods

1. **int getButton()**

// Returns which, if any, of the mouse buttons has changed state.

2. **int getClickCount()**

// Returns the number of mouse clicks associated with this event.

3. **Point getLocationOnScreen()**

// Returns the absolute x, y position of the event.

4. **int getModifiersEx()**

// Returns the extended modifier mask for this event.

5. **static String getMouseModifiersText(int modifiers)**

// Returns a String instance describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".

6. **Point getPoint()**

// Returns the x,y position of the event relative to the source component.

7. **int getX()**

// Returns the horizontal x position of the event relative to the source component.

8. **int getXOnScreen()**

// Returns the absolute horizontal x position of the event.

9. **int getY()**

// Returns the vertical y position of the event relative to the source component.

Syllabus Topic : The Text Event Class

3.2.8 The Text Event Class

Q. 3.2.8 Enlist constructors and methods of TextEvent class. (Ref. Sec. 3.2.8) **(Likely Ques)**

- A semantic event which indicates that an object's text has been changed. This high-level event is generated by

an object (such as a TextComponent) when its text changes.

- This respective event is transferred to every TextListener object which has been registered to receive these types of events with the help of the component's addTextListener method.
- The object that implements the TextListener interface gets this TextEvent when this event is fired.

☞ **Constructors**

1. **TextEvent(Object source, int id)**

// Constructs a TextEvent object.

☞ **Methods**

1. **String paramString()**

// Returns a parameter string identifying this text event.

Syllabus Topic : The Window Event Class

3.2.9 The Window Event Class

Q. 3.2.9 Enlist constructors and methods of WindowEvent class. (Ref. Sec. 3.2.9) **(Likely Ques)**

- A low-level event that indicates that a window has changed its status.
- This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.
- This respective event is transferred to every WindowListener or WindowAdapter object which has been registered to receive these types of events with the help of the window's addWindowListener method. (WindowAdapterobjects implement the WindowListener interface.)
- Each such listener object gets this WindowEvent when this event is fired.

☞ **Constructors**

1. **WindowEvent(Window source, int id)**

// Constructs a WindowEvent object.

2. **WindowEvent(Window source, int id, int oldState, int newState)**

// Constructs a WindowEvent object with the specified previous and new window states.

3. **WindowEvent(Window source, int id, Window opposite)**

// Constructs a WindowEvent object with the specified opposite Window.

4. **WindowEvent(Window source, int id, Window opposite, int oldState, int newState)**

// Constructs a WindowEvent object.

☞ **Methods**

1. **int getNewState()**

// For WINDOW_STATE_CHANGED events returns the new state of the window.

2. **int getOldState()**

// For WINDOW_STATE_CHANGED events returns the previous state of the window.

3. **Window getOppositeWindow()**

// Returns the other Window involved in this focus or activation change.

4. **Window getWindow()**

// Returns the originator of the event.

5. **String paramString()**

// Returns a parameter string identifying this event.

Syllabus Topic : Adapter Classes

3.3 Adapter Classes

Q. 3.3.1 What is an Adapter Class? (Ref. Sec. 3.3) **(Likely Ques)**

- Java provides adapter classes which gives the default implementation of listener interfaces.

- When an Adapter class is extended, it is not compulsory to provide the implementation of all the methods of listener interfaces. Hence it saves code.
- The adapter classes are provided in java.awt.event, java.awt.dnd and javax.swing.event packages.

☞ java.awt.event Adapter classes

Sr. No.	Adapter class	Listener interface
1.	WindowAdapter	WindowListener
2.	KeyAdapter	KeyListener
3.	MouseAdapter	MouseListener
4.	MouseMotionAdapter	MouseMotionListener
5.	FocusAdapter	FocusListener
6.	ComponentAdapter	ComponentListener
7.	ContainerAdapter	ContainerListener
8.	HierarchyBoundsAdapter	HierarchyBoundsListener

☞ java.awt.dnd Adapter classes

Sr. No.	Adapter class	Listener interface
1.	DragSourceAdapter	DragSourceListener
2.	DragTargetAdapter	DragTargetListener

☞ javax.swing.event Adapter classes

Sr. No.	Adapter class	Listener interface
1.	MouseInputAdapter	MouseInputListener
2.	InternalFrameAdapter	InternalFrameListener

Syllabus Topic : Inner Classes

3.4 Inner Classes

Q. 3.4.1 What is a nested class? Explain its types with suitable examples. (Ref. Sec. 3.4) **(Likely Ques)**

- Writing a class inside another class is known as nesting of classes.
- The class written within is known as inner class or **nested class** while the class that holds the inner (nested) class is known as the **outer class**.

☞ Some important points regarding inner class

- The scope of a nested class is bounded by the scope of its enclosing class. Thus class *NestedClass* does not exist independently of class *OuterClass*.
- A nested class has access to the members, including private members, of the class in which it is nested. However, reverse is not true i.e. the enclosing class does not have access to the members of the nested class.
- A nested class is also a member of its enclosing class.
- As a member of its enclosing class, a nested class can be declared *private*, *public*, *protected*, or *package private* (default).

☞ Syntax

```
class outerClass
{
   class innerClass
   {
        ...
   }
}
```

☞ Example (Program)

- We will see a Java program which handles the mouse click event using inner class and adapter class.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Demo extends JApplet
{
   JLabel lbl;
```

```
public void init()
{
    setSize(400,200);
    setLayout(new FlowLayout());
    lbl = new JLabel();
    add(lbl);
    addMouseListener(new MyAdapter());
}
class MyAdapter extends MouseAdapter
{
    public void mouseClicked(MouseEvent me)
    {
        lbl.setText("Mouse get clicked");
    }
}
}
```

Width and height for applet

Add label to an Applet

Advantages of inner / nested classes

Q. 3.4.2 Write any two advantages of inner class. (Ref. Sec. 3.4) **(Likely Ques)**

(1) Nested classes can access all the members (data members and methods) of outer class including private.

(2) Nested classes helps to develop more readable as well as maintainable code since it can logically group classes and interfaces in one place.

(3) Code Optimization : It requires less code to write.

3.4.1 Anonymous Inner Class

Q. 3.4.3 Write note on Anonymous Inner Class with suitable example. (Ref. Sec. 3.4.1) **(Likely Ques)**

- An inner class which is created without giving any name is called as an **anonymous inner class**.
- These classes are declared and instantiated at the same time
- Generally, they are used when there is requirement of overriding any method of a class or an interface.

Syntax

- The of an anonymous inner class is as follows :

```
AnonymousInner an_inner = new AnonymousInner()
{
  public void my_method()
    {
        ........
        ........
    }
};
```

Example (Anonymous inner class)

- In the previous mouse event handling program we can use an anonymous inner class as to simplify it :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Demo extends JApplet
{
    JLabel lbl;
    public void init()
    {
        setSize(400,200);
        setLayout(new FlowLayout());
        lbl = new JLabel();
        add(lbl);
        addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent me)
            {
                lbl.setText("Mouse get clicked");
            }
        }
        );
    }
}
```

Syllabus Topic : Event Listener Interfaces

3.5 Event Listener Interfaces

Java provides various interfaces for handling events.

☞ Java Event classes and Listener interfaces

Event Classes	Description	Listener Interface
ActionEvent	This event is triggered when button is pressed, menu-item is selected, list-item is double clicked.	ActionListener
MouseEvent	This event is triggered when mouse is dragged, moved, clicked,pressed or released and also when it enters or exit a component.	MouseListener
KeyEvent	This event is triggered when input is received from keyboard..	KeyListener
ItemEvent	This event is triggered when check-box or list item is clicked.	ItemListener
TextEvent	This event is triggered when value of textarea or textfield is changed.	TextListener
MouseWheelEvent	This event is triggered when mouse wheel is moved.	MouseWheelListener
WindowEvent	This event is triggered when window is activated, deactivated, deiconified, iconified, opened or closed.	WindowListener
ComponentEvent	This event is triggered when component is hidden, moved, resized or set visible.	ComponentEventListener
ContainerEvent	This event is triggered when component is added or removed from container.	ContainerListener
AdjustmentEvent	This event is triggered when scroll bar is manipulated.	AdjustmentListener
FocusEvent	This event is triggered when component gains or loses keyboard focus.	FocusListener

- The important step in handling event is to register the component with the Listener.
- There are different Registration Methods for different components :

1. Button

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

2. MenuItem

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

3. TextField

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

4. **TextArea**

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

5. **Checkbox**

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

6. **Choice**

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

7. **List**

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

Syllabus Topic : The ActionListener Interface

3.5.1 The ActionListener Interface

Q. 3.5.1 Enlist methods of ActionListener Interface. (Ref. Sec. 3.5.1) **(Likely Ques)**

- This is a listener interface which is used to receive action events.
- The respective class which wants to handle action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method.
- When the action event occurs, that object's action Performed method is invoked.

Methods

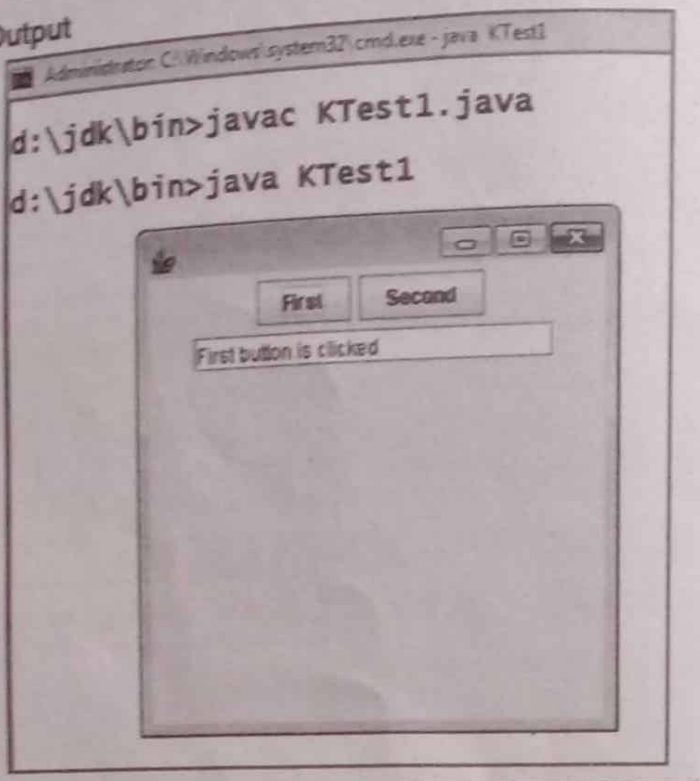
1. **void actionPerformed(ActionEvent e)**

// Invoked when an action occurs.

Example on JButton and JTextField

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class KTest1 extends JFrame implements ActionListener
{
   JButton b1,b2;
   JTextField jtf;
   KTest1 ()
   {
b1 = new JButton("First");        Creating a First Button.
      b2 = new JButton("Second");
                                  Creating a Second Button.
      jtf = new JTextField(20);   //creating JTextField.
      setLayout(new FlowLayout());
                                  setting layout using
                                  FlowLayout object
      setSize(300, 300);          //setting size of Jframe
      add(b1);
      add(b2);
      add(jtf);
      b1.addActionListener(this);
      b2.addActionListener(this);
      setVisible(true);
      setDefaultCloseOperation
      (JFrame.EXIT_ON_CLOSE);
                                  //setting close operation.
   }
   public void actionPerformed(ActionEvent e)
   {
      if (e.getSource()==b1)
      jtf.setText("First button is clicked");
      else
      jtf.setText("Second button is clicked");
   }
   public static void main(String[] args)
   {
      new KTest1();
   }
}
```

Output



Program 3.5.1 : Design a swing program to display three labels, two text boxes and two buttons as "Addition" and "Subtraction" respectively. In first label display text as "First Number" and in second label display text as "Second Number". Accept two numbers from two text box perform addition and subtraction operations, display the result in third label. **(5 Marks)**

Solution :

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class KTest1 extends JFrame implements
ActionListener
{
    JButton b1,b2;
    JTextField t1,t2;
    JLabel l1,l2,l3;
    KTest1 ()
    {
        b1 = new JButton("Addition");
        b2 = new JButton("subtraction");
        t1 = new JTextField(20);
        t2 = new JTextField(20);
        l1 = new JLabel("First Number");
        l2 = new JLabel("Second Number");
        l3 = new JLabel("  ");
        setLayout(new FlowLayout());
        setSize(300, 300);
        add(l1);
        add(t1);
        add(l2);
        add(t2);
        add(l3);
        add(b1);
        add(b2);
        b1.addActionListener(this);
        b2.addActionListener(this);
        setVisible(true);
        setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
                                    //setting close operation.
    }
    public void actionPerformed(ActionEvent e)
    {
        int n1,n2,r;
        n1 = Integer.parseInt(t1.getText());
        n2 = Integer.parseInt(t2.getText());
        if (e.getSource()==b1)
        {
            r = n1 + n2;
            l3.setText(""+r);
        }
        else
        {
            r = n1 - n2;
            l3.setText(""+r);
```

Interface

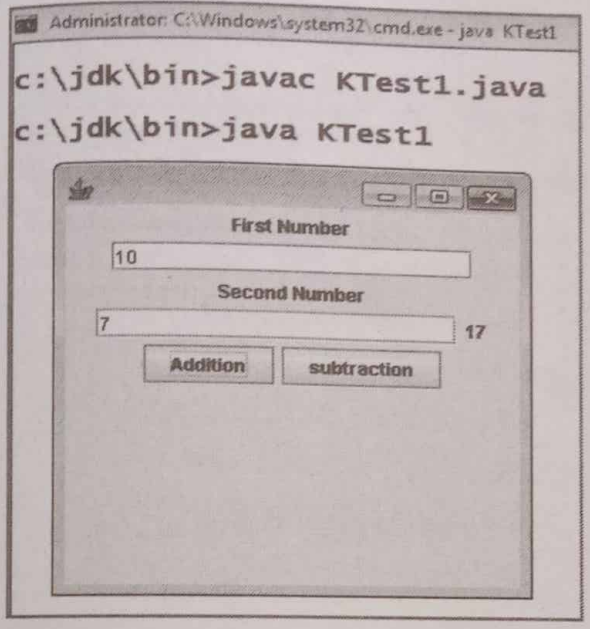
Adding controls on frame

Attaching listeners to buttons action preferred ( ) will be called

Convert string to int

```
    }
  }
  public static void main(String[] args)
  {
     new KTest1();
  }
}
```

Output



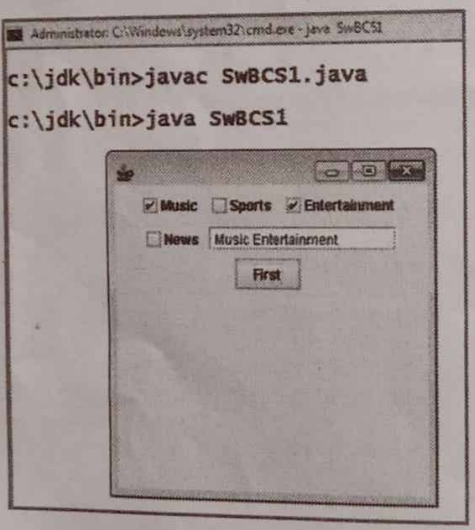
Program 3.5.2 : Write a program using swing to display the names of four hobbies using check boxes. Display the selected hobbies in a text field when the user clicks the "OK" button.

Solution :

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

public class SwBCS1 extends JFrame implements
ActionListener
{
   JCheckBox cb1,cb2,cb3,cb4;
   JTextField jtf;
   JButton btn;
   SwBCS1()
   {
      setLayout(new FlowLayout());   //setting layout using FlowLayout object
      setSize(300, 300);             //setting size of Jframe
      cb1=new JCheckBox("Music");
      add(cb1);

      cb2=new JCheckBox("Sports");
      add(cb2);

      cb3=new JCheckBox("Entertainment");
      add(cb3);

      cb4=new JCheckBox("News");
      add(cb4);
      jtf=new JTextField(15);
      add(jtf);
      btn = new JButton("First");
      btn.addActionListener(this);
      add(btn);
      setVisible(true);
      setDefaultCloseOperation
      (JFrame.EXIT_ON_CLOSE);        //setting close operation.
   }
   public void actionPerformed(ActionEvent e)
   {
      string str = "";
      if (cb1.isSelected())
      str = str + " Music";

      if (cb2.isSelected())
      str = str + " Sports";
```

```
        if (cb3.isSelected())
        str = str + " Entertainment";

        if (cb4.isSelected())
        str = str + " News";

        jtf.setText(str);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        JCheckBox cb=(JCheckBox)ie.getItem();
        jtf.setText(cb.getText());
    }
    public static void main(String[] args)
    {
        new SwBCS1();
    }
}
```

Output



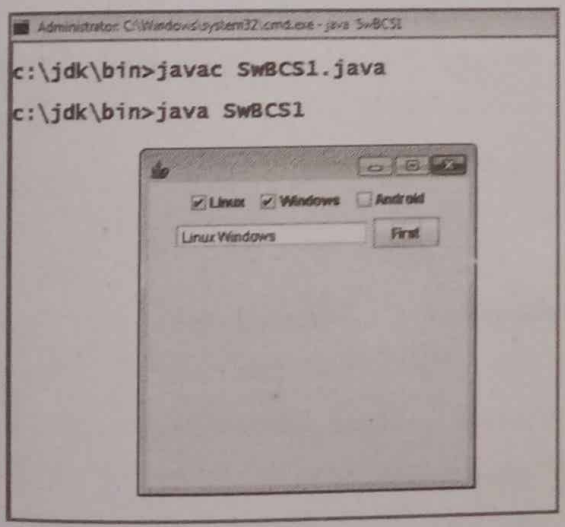
Program 3.5.3 : Write a java program to create a screen which contains three checkboxes (Linux, Windows, Android) and displays selected items in a textbox.

Solution :

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

public class SwBCS1 extends JFrame implements
ActionListener
{
    JCheckBox cb1,cb2,cb3;
    JTextField jtf;
    JButton btn;
    SwBCS1()
    {
        setLayout(new FlowLayout());
                    //setting layout using FlowLayout object
        setSize(300, 300);   //setting size of Jframe
        cb1=new JCheckBox("Linux");
        add(cb1);
        cb2=new JCheckBox("Windows");
        add(cb2);
        cb3=new JCheckBox("Android");
        add(cb3);

        jtf=new JTextField(15);
        add(jtf);   //Add text field
        btn = new JButton("First");
        btn.addActionListener(this);
        add(btn);
        setVisible(true);
        setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
                    //setting close operation.
    }
    public void actionPerformed(ActionEvent e)
    {
        string str = "";
        if (cb1.isSelected())
        str = str + " Linux";
```

```
        if (cb2.isSelected())
        str = str + " Windows";

        if (cb3.isSelected())
        str = str + " Android";

        jtf.setText(str);
    }

    public void itemStateChanged(ItemEvent ie)
    {
        JCheckBox cb=(JCheckBox)ie.getItem();
        jtf.setText(cb.getText());
    }

    public static void main(String[] args)
    {
        new SwBCS1();
    }
}
```

Output



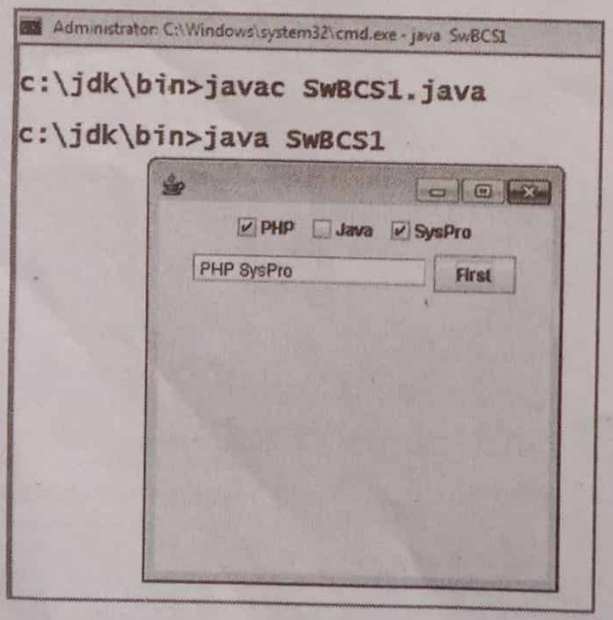
Program 3.5.4 : Write a program in java to create a screen which contains three checkboxes (PHP, Java, Syspro) and displays the selected items in a textbox.

Solution :

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

public class SwBCS1 extends JFrame implements
ActionListener
{
    JCheckBox cb1,cb2,cb3;
    JTextField jtf;
    JButton btn;
    SwBCS1()
    {
        setLayout(new FlowLayout());
                //setting layout using FlowLayout object
        setSize(300, 300);
                //setting size of Jframe
        cb1=new JCheckBox("PHP");
        add(cb1);

        cb2=new JCheckBox("Java");
        add(cb2);

        cb3=new JCheckBox("SysPro");
        add(cb3);

        //Add text field
        jtf=new JTextField(15);
        add(jtf);
                //Add text field
        btn = new JButton("First");
        btn.addActionListener(this);
        add(btn);
        setVisible(true);
        setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
                //setting close operation.
    }
    public void actionPerformed(ActionEvent e)
    {
```

```
        string str = "";
        if (cb1.isSelected())
        str = str + " PHP";

        if (cb2.isSelected())
        str = str + " Java";

        if (cb3.isSelected())
        str = str + " SysPro";


        jtf.setText(str);
    }

    public void itemStateChanged(ItemEvent ie)
    {
        JCheckBox cb=(JCheckBox)ie.getItem();
        jtf.setText(cb.getText());
    }

    public static void main(String[] args)
    {
        new SwBCS1();
    }
}
```

Output

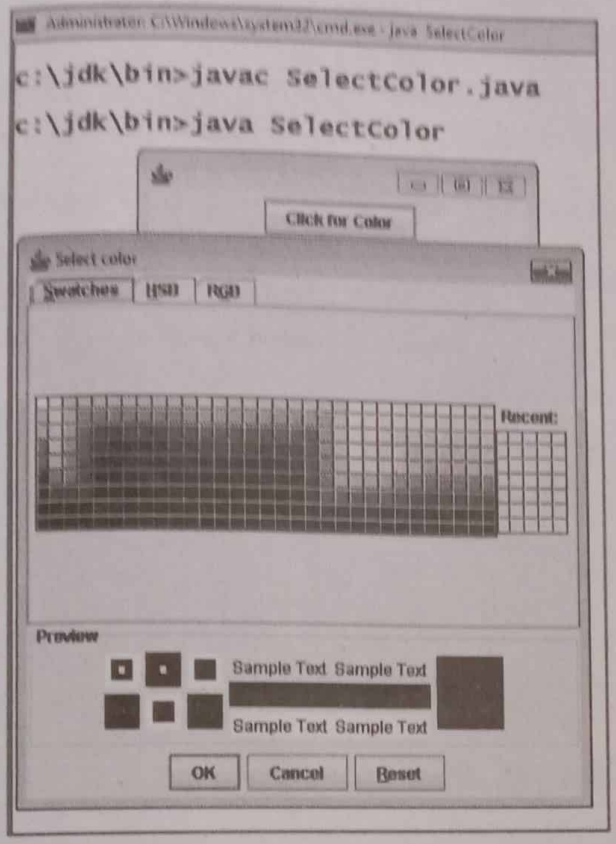


Program 3.5.5 : Write a Java program to set background colour of dialog box using Jcolor chooser.

Solution :

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class SelectColor second option is to extend JFrame
implements ActionListener
{
    JButton btn;
    Container cnt;
    SelectColor()
    {
        cnt=getContentPane();
        cnt.setLayout(new FlowLayout());
        btn=new JButton("Click for Color");
        btn.addActionListener(this);
        cnt.add(btn);
    }
    public void actionPerformed(ActionEvent e)
    {
        Color defcolor=Color.RED;
        Color c = JColorChooser.showDialog(this,"Select
        color",defcolor);
        cnt.setBackground(c);
    }

    public static void main(String[] args)
    {
        SelectColor obj=new SelectColor();
        obj.setSize(300,300);
        obj.setVisible(true);
        obj.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Output



3.5.2 The ComponentListener Interface

Q. 3.5.2 Enlist methods of ComponentListener Interface. (Ref. Sec. 3.5.2) **(Likely Ques)**

- This is a listener interface which is used to receive component events.
- The class which wants to handle the component event has one option as implement the respective interface and write definitions for all its methods or second option is to extend the abstract ComponentAdapter class (can override only required methods).
- The listener object which has been created for the class is further registered with a component with the help of component's addComponentListener method.
- When the component's size, location, or visibility changes, the respective method present in the listener object has been called, and the ComponentEvent is transferred to it.

Methods

1. **void componentHidden(ComponentEvent e)**

// Invoked when the component has been made invisible.

2. **void componentMoved(ComponentEvent e)**

// Invoked when the component's position changes.

3. **void componentResized(ComponentEvent e)**

// Invoked when the component's size changes.

4. **void componentShown(ComponentEvent e)**

// Invoked when the component has been made visible.

3.5.3 The ContainerListener Interface

Q. 3.5.3 Enlist methods of ContainerListener Interface. (Ref. Sec. 3.5.3) **(Likely Ques)**

- This is a listener interface which is used to receive container events.
- The respective class which wants to handle the container event has one option as implement the respective interface and write definitions for all its methods or second option is to extend the abstract ContainerAdapter class (can override only required methods).
- The listener object which has been created for the class is further registered with a component with the help of component's addContainerListener method.
- When the container's contents change since a component has been added or removed, the respective method present in the listener object has been called, and the ContainerEvent is transferred to it.

Methods

1. **void componentAdded(ContainerEvent e)**

// Invoked when a component has been added to the container.

2. **void componentRemoved(ContainerEvent e)**

// Invoked when a component has been removed from the container.

3.5.4 The FocusListener Interface

Q. 3.5.4 Enlist methods of FocusListener Interface. (Ref. Sec. 3.5.4) **(Likely Ques)**

- This is a listener interface which is used to receive keyboard focus events on a component.
- The respective class which wants to handle the focus event has one option as implement the respective interface and write definitions for all its methods or second option is to extend the abstract FocusAdapter class (can override only required methods).
- The listener object which has been created for the class is further registered with a component with the help of component's addFocusListener method.
- When the component gains or loses the keyboard focus, the respective method present in the listener object has been called, and the FocusEvent is transferred to it.

Methods

1. **void focusGained(FocusEvent e)**

// Invoked when a component gains the keyboard focus.

2. **void focusLost(FocusEvent e)**

// Invoked when a component loses the keyboard focus.

Syllabus Topic : The ItemListener Interface

3.5.5 The ItemListener Interface

Q. 3.5.5 Enlist methods of ItemListener Interface. (Ref. Sec. 3.5.5) **(Likely Ques)**

- This is a listener interface which is used to receive item events.
- The respective class which wants to handle then item event implements this interface. The object created with that class is then registered with a component using the component's addItemListener method. When an item-selection event occurs, the listener object's itemStateChanged method is invoked.

Method

void itemStateChanged(ItemEvent e)

// Invoked when an item has been selected or deselected by the user.

Program 3.5.6 : Write a program in java to create a screen which contains three radio buttons (Hindi, Marathi, English) and displays the selected items in a textbox.

Solution :

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

public class SwBCS2 extends JFrame implements
ItemListener
{
    JTextField jtf;
    SwBCS2()
    {
        setLayout(new FlowLayout());
                  //setting layout using FlowLayout object
        setSize(300, 300);   //setting size of Jframe
        JRadioButton opt1=new JRadioButton("Hindi");
        opt1.addItemListener(this);
        add(opt1);
        JRadioButton opt2=new JRadioButton("English");
        opt2.addItemListener(this);
        add(opt2);
        JRadioButton opt3=new JRadioButton("Marathi");
        opt3.addItemListener(this);
        add(opt3);
        ButtonGroup bg = new ButtonGroup();
        bg.add(opt1);
        bg.add(opt2);
        bg.add(opt3);
```

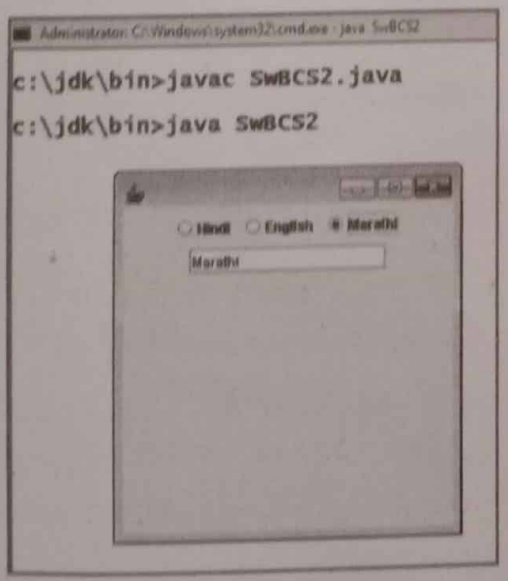
```
        jtf=new JTextField(15);

        add(jtf);          //Add text field
        setVisible(true);
        setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
                           //setting close operation.
    }
    public void itemStateChanged(ItemEvent ie)
    {
        JRadioButton cb=(JRadioButton)ie.getItem();
        jtf.setText(cb.getText());
    }
    public static void main(String[] args)
    {
        new SwBCS2();
    }
}
```

Explanation

Q. 3.5.6 What is the use of class ButtonGroup ? (Ref. Sec. 3.5.5) **(Likely Ques)**

- Here the ButtonGroup class is used to create group of buttons in which user can select a single option.

Output



Program 3.5.7 : Write a Java program to create an applet which has a list of radio buttons with titles of various colors. Set the background color of the applet to the selected color.

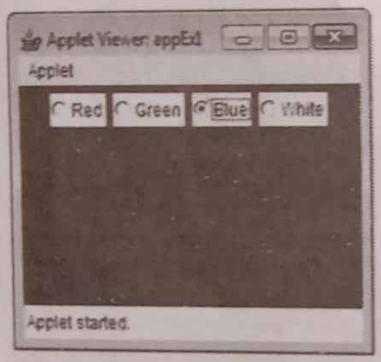
Solution :

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

/*
<applet code="appEx1" width=250 height=150>
</applet>
*/

public class appEx1 extends Applet implements
ItemListener
{
    CheckboxGroup cbg1;
    Checkbox cb1,cb2,cb3,cb4;

    public void init()
    {
        cbg1 = new CheckboxGroup();

        cb1 = new Checkbox("Red",false,cbg1);
        cb2 = new Checkbox("Green",false,cbg1);
        cb3 = new Checkbox("Blue",true,cbg1);
        cb4 = new Checkbox("White",false,cbg1);
                          text, default stato, group

        add(cb1);
        cb1.addItemListener(this);
        add(cb2);
        cb2.addItemListener(this);
        add(cb3);
        cb3.addItemListener(this);
        add(cb4);
        cb4.addItemListener(this);
        setBackground(Color.BLUE);
```

```
}

public void itemStateChanged(ItemEvent e)
{
 if(cb1.getState())
  setBackground(Color.RED);
 else if(e.getItemSelectable()==cb2)
  setBackground(Color.GREEN);
 else if(e.getItemSelectable()==cb3)
  setBackground(Color.BLUE);
 else if(e.getItemSelectable()==cb4)
  setBackground(Color.WHITE);
 }
}
```

Output

```
Administrator: C:\Windows\system32\cmd.exe - appletviewer appEx1.java

c:\jdk\bin>javac appEx1.java
c:\jdk\bin>appletviewer appEx1.java
```



Syllabus Topic : The KeyListener Interface

3.5.6 The KeyListener Interface

Q. 3.5.7 Enlist methods of KeyListener Interface. (Ref. Sec. 3.5.6) **(Likely Ques)**

- The KeyListener is notified the state of key gets changed.
- It is notified against KeyEvent.
- The KeyListener interface is member of java.awt.even package.

Methods

- It has following three methods :

1. **public abstract void keyPressed(KeyEvent e);**
2. **public abstract void keyReleased(KeyEvent e);**
3. **public abstract void keyTyped(KeyEvent e);**

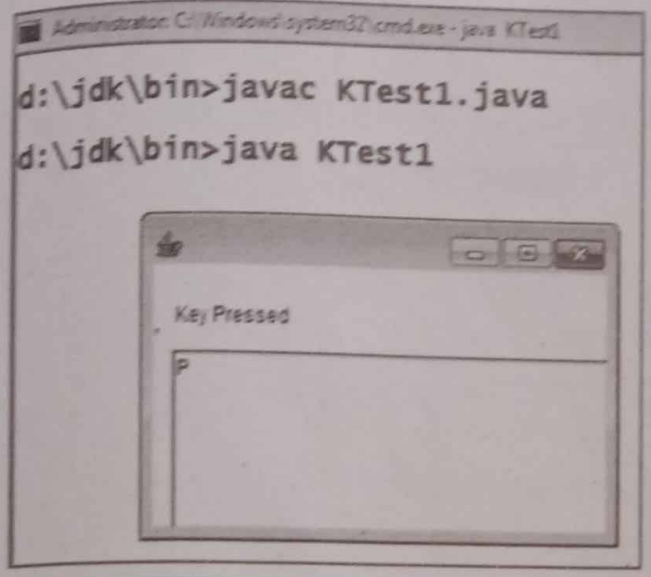
Example

```
import java.awt.*;
import java.awt.event.*;
public class KTest1 extends Frame implements
KeyListener{
  Label lbl;
  TextArea ta;
  KTest1(){

    lbl=new Label();
    lbl.setBounds(20,50,100,20);
    ta=new TextArea(5,20);
    ta.setBounds(20,80,300, 300);
    ta.addKeyListener(this);

    add(lbl);add(ta);
    setSize(350,350);
    setLayout(null);
    setVisible(true);
  }
  public void keyPressed(KeyEvent e) {
    lbl.setText("Key Pressed");
  }
  public void keyReleased(KeyEvent e) {
    lbl.setText("Key Released");
  }
  public void keyTyped(KeyEvent e) {
    lbl.setText("Key Typed");
  }
```

Attaching listener to text area

```
    public static void main(String[] args) {
        new KTest1();
    }
}
```

Output



Syllabus Topic : The MouseListener Interface

3.5.7 The MouseListener Interface

Q. 3.5.8 Enlist methods of MouseListener Interface. (Ref. Sec. 3.5.7) **(Likely Ques)**

- The MouseListener gets notified when mouse state is changed.
- That means it is notified against MouseEvent.
- The MouseListener interface is member of java.awt.event package.

Methods

- It has following five methods :

1. **public abstract void mouseClicked(MouseEvent e);**
2. **public abstract void mouseEntered(MouseEvent e);**
3. **public abstract void mouseExited(MouseEvent e);**
4. **public abstract void mousePressed(MouseEvent e);**
5. **public abstract void mouseReleased(MouseEvent e);**

Mouse Event Handling Example

Program 3.5.8 : Design a screen in java using AWT to handle the mouse events and display the position of mouse click in textbook.

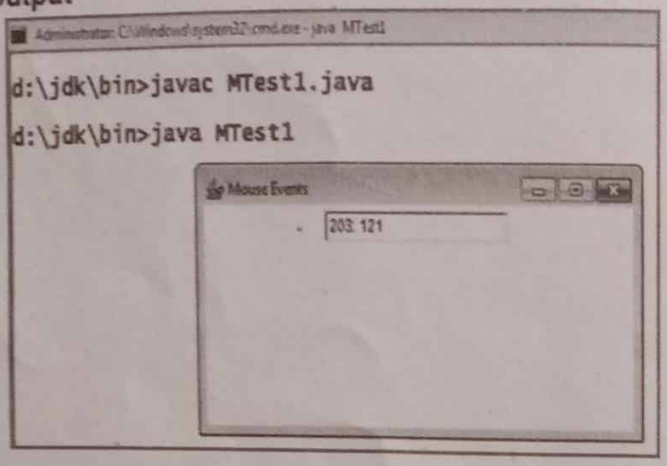
Solution :

```
import java.awt.*;
import java.awt.event.*;
public class MTest1 extends Frame
{
    TextField a;
    public MTest1()
    {
        super("Mouse Events");
        setSize(400,200);
        setLayout(new FlowLayout());
        a= new TextField(20);
        add(a);

        addMouseListener(new MouseAdapter()
        {
            public void mousePressed(MouseEvent e)
            {
                a.setText(String.valueOf(e.getX()) + ":" +
                String.valueOf(e.getY()));
            }
        });
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e){
            setVisible(false);
            dispose();
            System.exit(0);
        }
    });
}
public static void main(String[]args){
```

Displaying x, y co-ordinates in textbox

```
MTest1 app=new MTest1();
app.setVisible(true);
}
}
```

Output



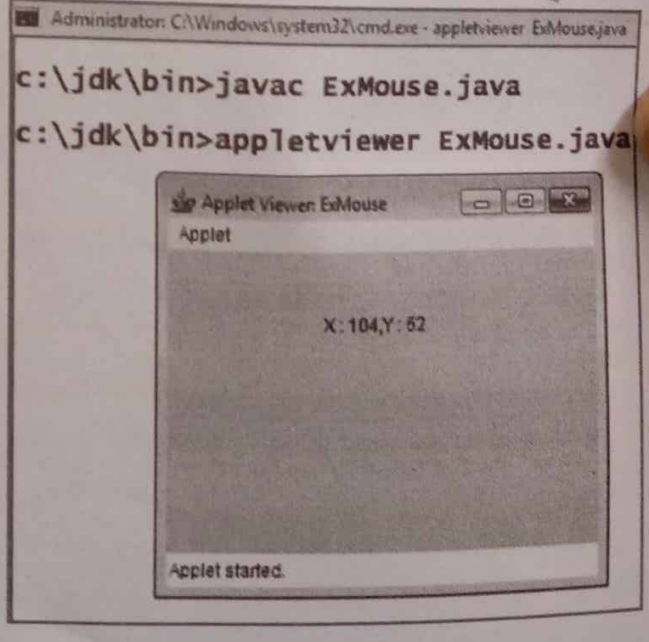
Example 1

- Display the x and y coordinates of the location where the mouse button is clicked in the applet window.

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

/*<applet code="ExMouse" width="300" height="200">
</applet>*/

public class ExMouse extends Applet implements
MouseListener
{
    private int mouseX, mouseY;
    private boolean mouseclicked = false;
    public void init()
    {
     setBackground(Color.PINK);
     addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
      mouseX = e.getX();
      mouseY=e.getY();
      mouseclicked = true;
      repaint();
    }

    public void mouseEntered(MouseEvent e){};
    public void mousePressed(MouseEvent e){};
    public void mouseReleased(MouseEvent e){};
    public void mouseExited(MouseEvent e){};
    public void paint( Graphics g )
    {
      String msg;
      g.setColor(Color.BLUE);
      if (mouseclicked)
      {
        msg = "X : "+ mouseX + "," + "Y : "+ mouseY ;
        g.drawString(msg,mouseX,mouseY);
        mouseclicked = false;
      }
    }
}
```

Geting x, y co-ordinates of mouse pointer

Output

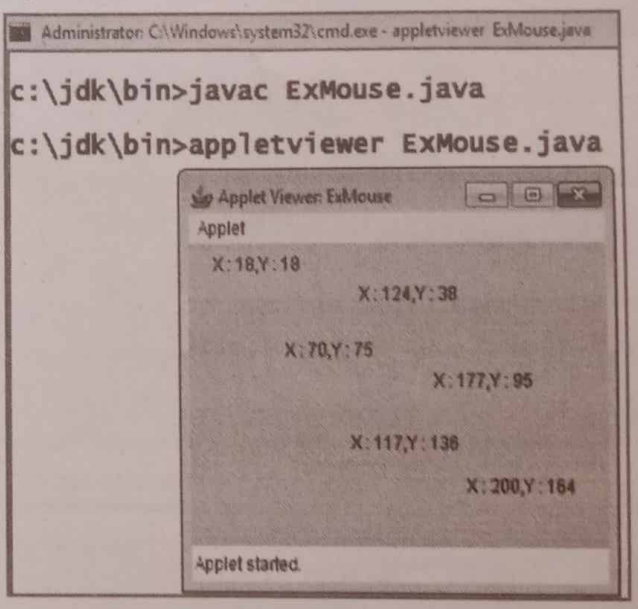


Example 2

– Display the x and y coordinates of the location where the mouse button is clicked in the applet window. (All the coordinates should display wherever user click without erasing any).

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
/*<applet code="ExMouse" width="300" height="200">
</applet>*/
public class ExMouse extends Applet implements
MouseListener
{
    private int mouseX, mouseY;
    private boolean mouseclicked = false;
    public void init()
    {
     setBackground(Color.PINK);
     addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
     mouseX = e.getX();
     mouseY=e.getY();
     mouseclicked = true;
     repaint();
    }
    public void mouseEntered(MouseEvent e){};
    public void mousePressed(MouseEvent e){};
    public void mouseReleased(MouseEvent e){};
    public void mouseExited(MouseEvent e){};
    public void update(Graphics g)
    {
     paint(g);
    }
    public void paint( Graphics g )
    {
     String msg;
     g.setColor(Color.BLUE);
     if (mouseclicked)
     {
      msg = "X : "+ mouseX + "," + "Y : "+ mouseY ;
      g.drawString(msg,mouseX,mouseY);
      mouseclicked = false;
     }
    }
}
```

Output



Syllabus Topic : The MouseMotionListener Interface

3.5.8 The MouseMotionListener Interface

Q. 3.5.9 Enlist methods of MouseMotionListener Interface. (Ref. Sec. 3.5.8) **(Likely Ques)**

– This is a listener interface which is used to receive mouse motion events on a component.

– The respective class which wants to handle the mouse motion event has one option to implement the respective interface and write definitions for all its methods or second option is to extend the abstract

MouseMotionAdapter class (can override only required methods).

- The listener object which has been created for the class is further registered with a component with the help of component's addMouseMotionListener method.
- A mouse motion event is generated when the mouse is moved or dragged. (Many such events will be generated).
- When a mouse motion event occurs, the respective method present in the listener object has been called, and the MouseEvent is transferred to it.

☞ **Methods**

1. **void mouseDragged(MouseEvent e)**

// Invoked when a mouse button is pressed on a component and then dragged.

2. **void mouseMoved(MouseEvent e)**

// Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Syllabus Topic : The TextListener Interface

3.5.9 The TextListener Interface

Q. 3.5.10 Enlist methods of TextListener Interface. (Ref. Sec. 3.5.9) **(Likely Ques)**

- This is a listener interface which is used to receive text events.
- The respective class which wants to handle the text event implements this interface.
- The object created with that class is then registered with a component using the component's addTextListener method. When the component's text changes, the listener object's textValueChanged method is invoked.

☞ **Methods**

1. **void textValueChanged(TextEvent e)**

// Invoked when the value of the text has changed.

Syllabus Topic : The WindowListener Interface

3.5.10 The WindowListener Interface

Q. 3.5.11 Enlist methods of WindowListener Interface. (Ref. Sec. 3.5.10) **(Likely Ques)**

- This is a listener interface which is used to receive window events.
- The respective class which wants to handle the window event has one option to implement the respective interface and write definitions for all its methods or second option is to extend the abstract WindowAdapter class (can override only required methods).
- The listener object created from that class is then registered with a Window using the window's addWindowListener method. When the window's status changes by virtue of being opened, closed, activated or deactivated, iconified or deiconified, the respective method present in the listener object has been called, and the WindowEvent is transferred to it.

☞ **Methods**

1. **void windowActivated(WindowEvent e)**

// Invoked when the Window is set to be the active Window.

2. **void windowClosed(WindowEvent e)**

// Invoked when a window has been closed as the result of calling dispose on the window.

3. **void windowClosing(WindowEvent e)**

// Invoked when the user attempts to close the window from the window's system menu.

4. **void windowDeactivated(WindowEvent e)**

// Invoked when a Window is no longer the active Window.

5. **void windowDeiconified(WindowEvent e)**

// Invoked when a window is changed from a minimized to a normal state.

6. **void windowIconified(WindowEvent e)**

// Invoked when a window is changed from a normal to a minimized state.

7. **void windowOpened(WindowEvent e)**

// Invoked the first time a window is made visible.

3.5.11 The WindowFocusListener Interface

Q. 3.5.12 Enlist methods of WindowFocusListener Interface. (Ref. Sec. 3.5.11) **(Likely Ques)**

- This is a listener interface which is used to receive WindowEvents, including WINDOW_GAINED_FOCUS and WINDOW_LOST_FOCUS events.
- The respective class which wants to handle the WindowEvent has one option to implement the respective interface and write definitions for all its methods or second option is to extend the abstract WindowAdapter class (can override only required methods).
- The listener object created from that class is then registered with a Window using the Window's addWindowFocusListener method.
- When the Window's status changes by virtue of it being opened, closed, activated, deactivated, iconified, or deiconified, or by focus being transfered into or out of the Window, the respective method present in the listener object has been called, and the WindowEvent is transferred to it.

Methods

1. **void windowGainedFocus(WindowEvent e)**

// Invoked when the Window is set to be the focused Window, which means that the Window, or one of its subcomponents, will receive keyboard events.

2. **void windowLostFocus(WindowEvent e)**

// Invoked when the Window is no longer the focused Window, which means that keyboard events will no longer be delivered to the Window or any of its subcomponents.

3.6 Solved Programs

Program 3.6.1 : Write a Java Applet that accepts two numbers in text box. When user clicks Calculate, the applet displays the GCD of the 2 numbers and the mean of all the numbers that lie between the two ?

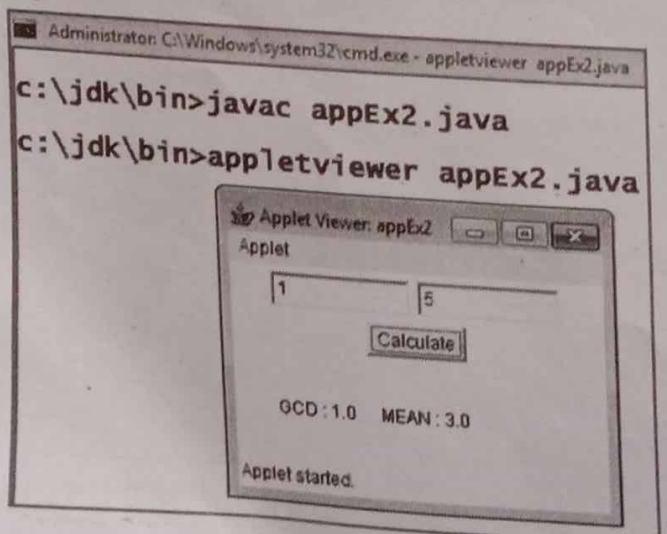
Solution :

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

/* <applet code="appEx2" width=250 height=200>
</applet> */

public class appEx2 extends Applet implements
ActionListener
{
    public double n1,n2,gcd,mean,sum,i;
    TextField t1;
    TextField t2;
    Button b;

    public void init()
    {
        t1 = new TextField(10);
        t2 = new TextField(10);
        b = new Button("Calculate");

        add(t1);
        add(t2);
        add(b);
        b.addActionListener(this);
    }
```

Adding listener

```
public void paint(Graphics g)
{
    g.drawString("GCD : "+gcd, 30,100);
    g.drawString("MEAN : "+mean, 100,100);
}

public void actionPerformed(ActionEvent e)
{

    n1 = Double.parseDouble(t1.getText());
    n2 = Double.parseDouble(t2.getText());
    for(i=1; i <= n1 && i <= n2; ++i)
    {
        if(n1%i==0 && n2%i==0)
// Checks if i is factor of both integers
            gcd = i;
    }
    sum = 0;
    i = 0;
    for(;n1<=n2;n1++)
    {
        sum = sum + n1; i++;
    }
mean = sum / i;
repaint();
  }
}
```

Output



Program 3.6.2 : Write a Java program to create an applet which contains a list of courses. Display the selected course in a text box.

Solution :

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/* <applet code="appEx3" width=350 height=130>
</applet>*/

public class appEx3 extends Applet implements
ActionListener
{
    TextField t;
    List L;

    public void init()
    {
        t = new TextField(10);
        L = new List(5, false);
        add(L);
        add(t);
        L.addActionListener(this);

        L.add("C");
        L.add("C++");
        L.add("DS");
        L.add("Java");
        L.add("VB");
    }
    public void actionPerformed(ActionEvent e)
    {
        t.setText(L.getSelectedItem());
    }
}
```