Graphical User Interfaces (II)

1. Overview

The learning objectives of this laboratory session are:

- Understand the use of mouse for user interaction
- Acquire knowledge on the use of important classes and interfaces used with handling of mouse gestures
- Acquire hands-on experience with handling of mouse events

The mouse is handled automatically by most components, so you never have to know about it. For example, if someone clicks on a button (**`JButton`**), you will receive an **`ActionEvent`**, but you don't need to know (and shouldn't care) whether this was from a mouse click on the button, or from a keyboard shortcut.

Graphics. If you are drawing your own graphics (e.g., in a **`JPanel`**) and need to know where the user clicks, then you need to know about mouse events. You can easily add a mouse listener to a **`JPanel`**.

2. Important Classes and Interfaces

These classes are defined in **`java.awt.event`**. The first three are the most commonly used.

- **`MouseEvent`** – A `MouseEvent` object is passed to all mouse listeners. The most useful information in a `MouseEvent` is the x and y coordinates of the mouse cursor.
- **`MouseListener`** – Interface for mouse presses, releases, clicks, enters, and exits.
- **`MouseMotionListener`** – Interface for mouse moves and drags.
- **`MouseInputListener`** – Interface combination of MouseListener and MouseMotionListener.
- **`MouseAdapter`** – Class useful for writing anonymous listener for mouse button presses, entering, ...
- **`MouseMotionAdapter`** – Class useful for writing anonymous listener for mouse movement.

2.1. MouseListener - Handles presses, releases, clicks, enters, and exits.

This type of mouse listener is for events which typically don't happen very often – a mouse button is pressed, released, or the mouse enters or leaves the area of the component with a listener. Table 1 lists the actions that a **MouseListener** catches.

Action	Explanation
press	one of the mouse buttons is pressed.
release	one of the mouse buttons is released.
click	a mouse button was pressed and released without moving the mouse. This is perhaps the most commonly used.
enter	mouse cursor enters the component. Often used to change cursor.
exit	mouse cursor exits the component. Often used to restore cursor.

Table 1. Actions (events) that a **`MouseListener`** catches.

To listen for these events you will use **`addMouseListener`**.

2.1.1. `MouseListener` Interface

To implement a **MouseListener** interface, you must define the following methods. You can copy these definitions into your program and only make a meaningful body for those methods that are of interest.

```
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

This method is called	When the user does this action
`mouseClicked`	A click is the result of a press and a release. This is probably the most common method to write.
`mousePressed`	A mouse button is pressed (any of three possible mouse buttons)
`mouseReleased`	A mouse button is released.
`mouseEntered`	The mouse cursor enters a component. You might write this to change the cursor.
`mouseExited`	The mouse cursor leaves a component. You might write this to restore the cursor.

To Get the Mouse Coordinates. All coordinates are relative to the upper left corner of the component with the mouse listener. Use the following **`MouseEvent`** methods to get x and y coordinates of where the mouse event occurred.

```
int getX() // returns the x coordinate of the event.
int getY() // returns the y coordinate of the event.
```

To Check for Double Clicks. Use the following **`MouseEvent`** method to get a count of the number of clicks.

```
int getClickCount() // number of mouse clicks
```

2.1.2. `MouseMotionListener` - Handles moves and drags.

Moves and drags. When you move the mouse, the interface generates events very rapidly. If a mouse button is depressed, then this is called a drag. Events from a move or drag are generated very quickly, and can be listed to by adding a mouse motion listener.

MouseMotionListener methods

To implement a `MouseMotionListener`, you must define the following methods.

```
public void mouseMoved(MouseEvent e) {}
public void mouseDragged(MouseEvent e) {}
```

This method is called	When the user does this action
`mouseMoved(...)`	The mouse is moved while over the component.
`mouseDragged(...)`	The mouse is dragged (moved with a button pressed).

2.2. Mouse Listeners - How and where to write mouse listeners.

There are several styles for using the mouse listeners. They are usually added to a graphics panel with a **`paintComponent`** method.

2.2.1. Listening within the panel itself

It is common to have a panel listen to its own events. For example,

```
class DrawingPanel extends JPanel implements MouseListener
{

   public DrawingPanel()
   { // Constructor
      this.addMouseListener(this);
      . . .
   }

   public void paintComponent(Graphics g)
   {
      . . .
   }
   . . .
   public void mousePressed(MouseEvent e) {. . .}
   public void mouseReleased(MouseEvent e) {. . .}
   public void mouseClicked(MouseEvent e) {. . .}
   . . .
}
```

It can communicate changes with the outside by (1) making it a subclass, (2) supplying *getter* methods, or (3) supplying a "model" object to the constructor.

2.2.2. Listening from outside the panel

You may create a panel and want the listeners outside it because it is more convenient to interact with them that way. If you only have one such panel, then you can implement the mouse listener interfaces in your non-panel class and write all of the listener methods. For example,

```
public class MyClass implements MouseListener {
   . . .
   DrawingPanel drawing = new DrawingPanel();
   drawing.addMouseListener(this);
   . . .

   public void mousePressed(MouseEvent e) {. . .}
   public void mouseReleased(MouseEvent e) {. . .}
   public void mouseClicked(MouseEvent e) {. . .}
   . . .
}

class DrawingPanel extends JPanel {
   public void paintComponent(Graphics g) {
      . . .
   }
   . . .
}
```

This requires *setter* methods in the `DrawingPanel` class so that what is drawn can be changed. Or a constructor for `DrawingPanel` could be passed an object for the "model" that would allow it to get values needed by `paintComponent`.

2.2.3. As above with anonymous listeners

If you only want to listen for one kind of event, it's easy to use the **`MouseAdapter`** or **`MouseMotionAdapter`** classes to create an anonymous listener. For example, to listen for mouse clicks,

```
p.addMouseListener(new MouseAdapter()
{
   public void mouseClicked(MouseEvent e)
   {
      x = e.getX();
      y = e.getY();
   }
});
```

2.3. Mouse Buttons, Modifier Keys - How to check which mouse buttons are pressed.

Mouse Buttons. Java supports up to three mouse buttons. Even if your mouse doesn't have three separate buttons, you can simulate some of the buttons by pressing modifier keys when pressing the mouse button.

The **`MouseEvent`** object that is passed to the listener contains information that allows you to ask which combinations of buttons were pressed when the event occurred. *Mouse scroll controls* were first supported in Java 2 SDK 1.4.

There are two ways to test the mouse buttons and modifier keys:

- Use *methods* to find the status of the mouse buttons and modifier keys.
- Use *bit masks* which are defined in the `InputEvent` class to look at the `MouseEvent` modifier bits. This is a *very fast* way to check, especially for complex combinations of modifiers and buttons, but you need to understand the bit operators (| & ^ ~ >> >>> <<).

2.3.1. To Use Methods to Check Mouse Buttons

To check which mouse button is pressed, call one of the static methods in `SwingUtilities`. These methods return true if the corresponding button is being used. Note that more than one of them will be true if more than one button is in use at the same time.

- `boolean SwingUtilities.isLeftMouseButton(MouseEvent anEvent)`
- `boolean SwingUtilities.isMiddleMouseButton(MouseEvent anEvent)`
- `boolean SwingUtilities.isRightMouseButton(MouseEvent anEvent)`

2.3.2. To Use Methods to Check Modifier Keys

To check which modifier keys are pressed, use these methods in the MouseEvent class:

```
boolean isAltDown()     // true if Alt key middle mouse button
boolean isControlDown() // true if Control key is pressed
boolean isShiftDown()   // true if Shift key is pressed
boolean isAltGraphDown()// true if Alt Graphics key (found on some
keyboards) is pressed
boolean isMetaDown()    // true if Meta key or right mouse button
```

For example, inside the mouse listener we could make a test like the following to see if the right mouse button is pressed while the shift key is down. Assume that `e` is a `MouseEvent` object.

```
if (SwingUtilities.isRightMouseButton(e) && e.isShiftDown())
   ...
```

2.3.3. To Use Bit Masks to Check Mouse Buttons and Modifier Keys

Use the `MouseEvent getModifiers()` method to get the a bitmask which tells which buttons were pressed when the event occurred. The masks for each of the mouse buttons as well as modifier keys are:

Mask	Meaning
InputEvent.BUTTON1_MASK	mouse button1
InputEvent.BUTTON2_MASK	mouse button2
InputEvent.BUTTON3_MASK	mouse button3
InputEvent.ALT_MASK	alt key
InputEvent.CTRL_MASK	control key
InputEvent.SHIFT_MASK	shift key
InputEvent.META_MASK	meta key
InputEvent.ALT_GRAPH_MASK	alt-graph key

To rewrite the previous example using bit masks to test whether the right mouse button is pressed while the shift key is down, we could do the following:

```
int RIGHT_SHIFT_MASK = InputEvent.BUTTON3_MASK + InputEvent.SHIFT_MASK;
. . .
if ((e.getModifiers() & RIGHT_SHIFT_MASK) == RIGHT_SHIFT_MASK) {
   ...
```

2.4. Example - MouseTest.java - Example shows mouse coordinates.

This is a simple demonstration of listening to mouse events on a panel. This displays two panels to show how the mouse listener depends on the component
The main program

```
// File:   mousedemo/MouseTest.java
// Description: Main program/applet to demo mouse listeners.
// Author: Fred Swartz
// Date:   2005-02-03, 2000-11-29...2002-11-21
// Possible enhancements: Show other mouse events.


package mousedemo;
import javax.swing.*;

////////////////////////////////////////////////////// class MouseTest
public class MouseTest extends JApplet{
    //================================================== constructor
    public MouseTest() {
        add(new DualMousePanel());
    }

    //================================================== method main
    public static void main(String[] args) {
        JFrame window = new JFrame("Mouse Demo");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setContentPane(new DualMousePanel());
        window.pack();
        window.setVisible(true);
    }
}
```

The content pane of the GUI

```
// File:   mousedemo/MousePanel.java
// Description: Panel holding two MousePanels.
// Author: Fred Swartz
// Date:   2005-02-03, 2000-11-29...2002-11-21

package mousedemo;

import java.awt.*;
```

```
import javax.swing.*;
import javax.swing.border.*;

////////////////////////////////////////////////// class DualMousePanel
class DualMousePanel extends JPanel {
    //======================================================= constructor
    public DualMousePanel() {
        //--- Create two MousePanels
        MousePanel mp1 = new MousePanel();
        MousePanel mp2 = new MousePanel();

        //--- Add borders (note: borders are inside panel)
        Border etched = BorderFactory.createEtchedBorder();
        mp1.setBorder(BorderFactory.createTitledBorder(etched, "Panel
1"));
        mp2.setBorder(BorderFactory.createTitledBorder(etched, "Panel
2"));

        //--- Layout the panels
        this.setLayout(new GridLayout(1, 2));
        this.add(mp1);
        this.add(mp2);
    }//end constructor
}
```

The panel which listens to and shows mouse events

```
// File:   mousedemo/MousePanel.java
// Description: Panel that listens to mouse events and displays info
//              about some of them.
// Author: Fred Swartz
// Date:   2005-02-03, 2000-11-29...2002-11-21

package mousedemo;

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

////////////////////////////////////////////////////// class MousePanel
class MousePanel extends JPanel implements MouseListener, MouseMotionListener {

   //================================================ instance variables
    private int m_lastClickedX = 0;  // x coord of mouse click
    private int m_lastClickedY = 0;  // y coord of mouse click
    private int m_lastMovedX   = 0;  // x coord of mouse move
    private int m_lastMovedY   = 0;  // y coord of mouse move


    //======================================================= constructor
    public MousePanel() {
        this.setBackground(Color.white);
        this.setPreferredSize(new Dimension(200, 200));
        //--- Add the mouse listeners.
        this.addMouseListener(this);       // listen to mouse events
        this.addMouseMotionListener(this); // listen to moves and drags
    }


    //========================================== method paintComponent
    public void paintComponent(Graphics g) {
        super.paintComponent(g);  // paint background and borders
        g.drawString("Last click: x=" + m_lastClickedX
```

```
                        + ", y=" + m_lastClickedY , 10, 30);
            g.drawString("x=" + m_lastMovedX   + ", y=" + m_lastMovedY
                        , m_lastMovedX, m_lastMovedY);
        }


        //============================================ listener mouseClicked
        public void mouseClicked(MouseEvent e) {
            m_lastClickedX = e.getX(); // Save the x coordinate of the click
            m_lastClickedY = e.getY(); // Save the y coordinate of the click
            this.repaint();            // Paint the panel with the new values.
        }


        //============================================ listener mouseMoved
        public void mouseMoved(MouseEvent e) {
            m_lastMovedX = e.getX();
            m_lastMovedY = e.getY();
            this.repaint();
        }


        //============================================================ ignored
        //==== the other motion events must be here, but do nothing.
        public void mouseDragged (MouseEvent e) {}  // ignore
        //==== these "slow" mouse events are ignored.
        public void mouseEntered (MouseEvent e) {}  // ignore
        public void mouseExited  (MouseEvent e) {}  // ignore
        public void mousePressed (MouseEvent e) {}  // ignore
        public void mouseReleased(MouseEvent e) {}  // ignore
    }
```

2.5. Example - DragDemo.java - Example shows mouse dragging.

Drag the ball around on the applet to the left. The source for this program is given below. It's written as both an applet (subclass `JApplet`) and application (it has a `main` method). The two source files are given below.
The image is drawn with the Graphics method (`fillOval`), but an image could be easily displayed.

The main/applet framework

```
/** DragDemo.java - Mouse drag example dual application/applet
    @author Fred Swartz
    @version 2004-04-15
*/
// "appletviewer DragDemo.java" works because of the following line.
// <applet code="DragDemo.class" height="200" width="200"></applet>
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
//////////////////////////////////////////////////////////// class DragDemo
/** This is an application because it has a main method.
    It's also an applet because it extends JApplet.
*/
public class DragDemo extends JApplet {
    //============================================================ method main
    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Drag Demo");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setContentPane(new DragBallPanel());
        window.pack();
```

```
        window.setVisible(true);
    }//end main

    //======================================================== applet constructor
    public DragDemo() {
        this.setContentPane(new DragBallPanel());
    }
}//endclass DragDemo
```

The panel that is used for the graphics

```
/** DragBallPanel.java - Panel that allows dragging a ball around.
    @author Fred Swartz
    @version 2004-04-15
*/

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

////////////////////////////////////////////////////////// class DragBallPanel
/** When the mousePressed listener is called to position is tested
    to see if it's in the area of the ball.  If it is,
    (1) _canDrag is set true meaning pay attention to the MouseDragged events.
    (2) Record where in the ball (relative to the upper left coordinates)
        the mouse was clicked, because it looks best if we drag from there.
*/
public class DragBallPanel extends JPanel implements MouseListener,
MouseMotionListener {

    private static final int BALL_DIAMETER = 40; // Diameter of ball
    //--- instance variables
    /** Ball coords.  Changed by mouse listeners.  Used by paintComponent. */
    private int _ballX     = 50;   // x coord - set from drag
    private int _ballY     = 50;   // y coord - set from drag

    /** Position in ball of mouse press to make dragging look better. */
    private int _dragFromX = 0;    // pressed this far inside ball's
    private int _dragFromY = 0;    // bounding box.

    /** true means mouse was pressed in ball and still in panel.*/
    private boolean _canDrag  = false;

    //============================================================== constructor
    /** Constructor sets size, colors, and adds mouse listeners.*/
    public DragBallPanel() {
        setPreferredSize(new Dimension(300, 300));
        setBackground(Color.blue);
        setForeground(Color.yellow);
        //--- Add the mouse listeners.
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
    }//endconstructor

    //================================================== method paintComponent
    /** Ball is drawn at the last recorded mouse listener coordinates. */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);   // Required for background.
        g.fillOval(_ballX, _ballY, BALL_DIAMETER, BALL_DIAMETER);
    }//end paintComponent

    //================================================== method mousePressed
    /** Set _canDrag if the click is in the ball (or in the bounding
        box, which is lazy, but close enuf for this program).
        Remember displacement (dragFromX and Y) in the ball
```

```
        to use as relative point to display while dragging.
    */
    public void mousePressed(MouseEvent e) {
        int x = e.getX();   // Save the x coord of the click
        int y = e.getY();   // Save the y coord of the click

        if (x >= _ballX && x <= (_ballX + BALL_DIAMETER)
                && y >= _ballY && y <= (_ballY + BALL_DIAMETER)) {
            _canDrag = true;
            _dragFromX = x - _ballX;  // how far from left
            _dragFromY = y - _ballY;  // how far from top
        } else {
            _canDrag = false;
        }
    }//end mousePressed

    //============================================================ mouseDragged
    /** Set x,y  to mouse position and repaint. */
    public void mouseDragged(MouseEvent e) {
        if (_canDrag) {   // True only if button was pressed inside ball.
            //--- Ball pos from mouse and original click displacement
            _ballX = e.getX() - _dragFromX;
            _ballY = e.getY() - _dragFromY;

            //--- Don't move the ball off the screen sides
            _ballX = Math.max(_ballX, 0);
            _ballX = Math.min(_ballX, getWidth() - BALL_DIAMETER);

            //--- Don't move the ball off top or bottom
            _ballY = Math.max(_ballY, 0);
            _ballY = Math.min(_ballY, getHeight() - BALL_DIAMETER);

            this.repaint(); // Repaint because position changed.
        }
    }//end mouseDragged

    //=================================================== method mouseExited
    /** Turn off dragging if mouse exits panel. */
    public void mouseExited(MouseEvent e) {
        _canDrag = false;
    }//end mouseExited

    //============================================== Ignore other mouse events.
    public void mouseMoved   (MouseEvent e) {}  // ignore these events
    public void mouseEntered (MouseEvent e) {}  // ignore these events
    public void mouseClicked (MouseEvent e) {}  // ignore these events
    public void mouseReleased(MouseEvent e) {}  // ignore these events
}//endclass DragBallPanel
```

2.6. Animation using the `Timer` class

For `Timer` you also have to implement an `actionPerformed()` method specified in the `ActionListener` interface. To start/stop animations, you should invoke the `start()` and `stop()` methods in class `Timer`.

Un exemplu simplu de animație:

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

```
public class TimerEx extends JPanel implements ActionListener{
    JLabel l;
    Timer t;
    int x = 10;
    int y = 300;
   TimerEx(){
       ImageIcon img = new ImageIcon("Mario.gif");
       l = new JLabel(img);
       l.setLocation(x, y);
       this.add(l);
       setBackground(Color.white);
       t = new Timer(100, this);
       t.addActionListener(this);
       t.start();
   }
// @override
   public void actionPerformed(ActionEvent e){
       x+=20;
       if (x>800) x = 50;
       l.setLocation(x,y);
   }
   public static void main(){
       JFrame frame = new JFrame("Timer Example");
       frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
       frame.setSize(800, 800);
       TimerEx pane= new TimerEx();
       frame.setContentPane(pane);
       frame.setVisible(true);
    }
}
```

3. Lab Tasks

3.1. Study and execute the code samples provided.

3.2. Add animation to the `DragBall` example:

- When the user clicks inside the ball, generate random coordinates for the ball at after a fixed time interval has passed.
- Display, in the top left corner of the panel the balls current coordinates. Hint: use the `drawString(String s, int x, int y)` of the class `Graphics`. The code should be written inside the `paintComponent(Graphics g) method.`
- A more advanced version of this requirement is to generate a direction which the ball must follow till it meets a panel margin, the direction should change such a way that the ball does not leave the panel.

3.3. Change the code (e.g. add methods). Notice the effects of your changes.