Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

Event Handling in JDK 1.1, MouseMotion Events

Java Programming, Lecture Notes # 92, Revised 02/24/99.

Preface

Students in Prof. Baldwin's **Intermediate Java Programming** classes at ACC are responsible for knowing and understanding all of the material in this lesson.

On 2/24/99, the sample program in this lesson was confirmed to operate properly under JDK 1.2.

Introduction

JDK 1.1 was formally released on February 18, 1997. This lesson was originally written on February 21, 1997 using the software in JDK 1.1. A number of potential bugs in the **MouseMotion** capability of JDK 1.1 for Win95 were identified at that time.

This lesson, and the example program, were completely rewritten in September of 1997 based on JDK 1.1.3 for Win95. As you will see, although the system seems to be much cleaner now, there appear to be a couple of remaining bugs having to do with the very rapid generation of **MouseMotion** events. When these events are generated rapidly and randomly over an extended period of time, they suddenly cease to be generated. Whether or not this is a problem depends on the application.

When we discussed mouse events in an earlier lesson, we didn't take into account that there are actually two different mouse interfaces:

- MouseListener
- MouseMotionListener

In that lesson, we only discussed the implementation of methods in the **MouseListener** interface.

In this lesson, we will also implement the **mouseDragged()** and the **mouseMoved()** methods in the **MouseMotionListener** interface to implement a program which can move a **Label** object on a **Panel** object by dragging the **Label** object with the mouse.

This lesson is not intended to be a polished demonstration of drag-and-drop. Rather, it is intended only to illustrate how to instantiate, register, listen for, and use events declared in the methods of the **MouseMotionListener** interface.

This lesson will also make use of the **mousePressed()** method of the **MouseListener** interface.

Overview

The **MouseMotionListener** interface declares two methods:

```
• public abstract void mouseDragged(MouseEvent e)
• public abstract void mouseMoved(MouseEvent e)
```

The **mouseDragged()** method is invoked when a mouse button is pressed on a component and then dragged.

According to the documentation, mouse drag events will continue to be delivered to the component where they first originated until the mouse button is released (regardless of whether the mouse position is within the bounds of the component).

However, by disabling the code in the following sample program that moves the Label object, and observing the printed output on the screen, it can be seen that mouse drag events cease to be generated as soon as the mouse leaves the boundaries of the object for which the listener was registered. Furthermore, moving the mouse back inside the boundaries of the object does not cause mouse drag events to resume until the mouse button is released and pressed again.

It is unclear at this time whether this is a bug, a documentation error, or simply a misinterpretation of the documentation on the part of this author.

The **mouseMoved()** method is invoked when the mouse is moved on a component (with no buttons down). A **System.out.println()** statement in the **mouseMoved()** method in the following program indicates that a stream of events is generated when the mouse is moved inside the label and the events stop being generated when the mouse leaves the label. The stream of events resumes when the mouse re-enters the label.

The Sample Program

This section will present a short discussion of the program followed by the program listing.

Discussion

The program illustrates the **mouseDragged()** event used in conjunction with the **mousePressed()**.

The program uses a combination of **mousePressed()** and **mouseDragged()** events to implement a crude form of *Drag-and-Drop*.

A yellow Label is placed on a **Panel,** which is placed in a **Frame**.

A **MouseListener** object and a **MouseMotionListener** object are instantiated and registered to receive mouse events on the **Label** object.

The user can drag the **Label** object by dragging the mouse internal to the **Label** object. The algorithm which uses the information provided by a **mousePressed()** event and the **mouseDragged()** events to drag the **Label** object on the screen is explained in the comments in the program.

The purpose of this program is simply to demonstrate the use of the **MouseMotionListener** interface as compared to the **MouseListener** interface. It is not intended to be a polished *Drag-and-Drop* program.

The program also produces screen output showing the stream of events produced by the **mouseDragged()** and **mouseMoved()** methods. By viewing this output while moving or dragging the mouse, you can get an idea of how the event-handling system is behaving.

Finally, a **WindowListener** object is instantiated and registered to terminate the program when the user closes the Frame object.

These results were produced using JDK 1.1.3 running under Windows 95.

Program Listing

A listing of the program with additional comments follows:

```
/*File Event21A.java Copyright 1997, R.G.Baldwin
This program is designed to be compiled and run
under JDK 1.1.3

The program illustrates the mouseDragged event used in
conjunction with the mousePressed() event to drag a label
object on a Panel object within a Frame object in a
crude form of Drag-and-Drop.

The program also illustrates the mouseMoved event by
displaying the contents of the object passed as a parameter
to the event when the mouse is moved within the label
```

```
without pressing a mouse button.

A yellow Label is placed on a Panel which is placed in a
Frame.

A MouseListener object and a MouseMotionListener object are
instantiated and registered to receive mouse events on the
Label object.

The user can drag the yellow Label object by dragging the
mouse internal to the Label object.

Comments within the code explain the algorithm which uses
the information provided by the mousePressed and mouseDragged
events to move the label object.

The purpose of this program is simply to demonstrate the
use of the MouseMotionListener interface as compared to the
MouseListener interface.  It is not intended to be a
polished Drag-and-Drop program.

Finally, a WindowListener object is instantiated and
registered to terminate the program when the user closes
the Frame object.

These results were produced using JDK 1.1.3 running under
Windows 95.

Possible bugs in JDK 1.1.3 for Win 95 relative to the
mouseDragged and mouseMoved events are described by
comments in the code.  Briefly, when the mouse is moved
in such a manner as to cause a large number of such events
to be generated over an extended period, the system
simply stops generating events.
*/
//=========================================================

import java.awt.*;
import java.awt.event.*;

public class Event21A {
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class Event21A
//=========================================================

class GUI {
  Label myLabel;//reference variables passed as parameters
  LabelInfo myLabelInfo;
  int initialX = 50; //initial location of the label object
  int initialY = 25;

  public GUI(){//constructor
    //Create visual components
    myLabel = new Label("LabelObject");//create a label
```

```
    myLabel.setBackground(Color.yellow);//make it yellow

    //create a panel to place the panel on
    Panel myPanel = new Panel();
    myPanel.setLayout(null); //no layout manager wanted
    myPanel.add(myLabel); //place the label on the panel
    //Adjust x,y,width & height
    myLabel.setBounds(initialX,initialY,125,100);

    Frame myFrame = new Frame();//frame to contain it all
    myFrame.setSize(600,600);
    myFrame.setTitle("Copyright 1997, R.G.Baldwin");
    myFrame.add("Center",myPanel); //place panel in frame
    myFrame.setVisible(true);//make it all visible

    //Create object to maintain info about the label
    // while it is being dragged about.
    myLabelInfo = new LabelInfo();
    myLabelInfo.labelX = initialX;
    myLabelInfo.labelY = initialY;

    //Instantiate and register MouseListener
    // and MouseMotionListener
    myLabel.addMouseMotionListener(
          new MyMouseMotionProcessor(myLabelInfo,myLabel));
    myLabel.addMouseListener(
                new MyMouseProcessor(myLabelInfo,myLabel));

    //Instantiate and register a WindowListener object
    // which will terminate the program when the user
    // closes the Frame object
    WProc1 winProcCmd1 = new WProc1();
    myFrame.addWindowListener(winProcCmd1);
  }//end constructor
}//end class GUI definition
//=========================================================

//This is a simple wrapper class used to maintain info for
// several position parameters used while dragging the
// label object.

class LabelInfo{
  int labelX;
  int labelY;
  int mousePressedX;
  int mousePressedY;
  int mouseDraggedX;
  int mouseDraggedY;
}//end class LabelInfo

//=========================================================

//This class recognizes mousePressed().  This method is
// used to determine the starting position of the mouse
// pointer.
class MyMouseProcessor extends MouseAdapter{
```

```
  LabelInfo theLabelInfo;
  Label theLabel;

  //Constructor
  MyMouseProcessor(LabelInfo inLabelInfo, Label inLabel){
    //save references to the input objects
    theLabelInfo = inLabelInfo;
    theLabel = inLabel;
  }//end constructor

  public void mousePressed(MouseEvent e){
    //save starting position of mouse pointer
    theLabelInfo.mousePressedX = e.getX();
    theLabelInfo.mousePressedY = e.getY();
  }//end mousePressed()

}//end MyMouseProcessor

//=======================================================
class MyMouseMotionProcessor extends MouseMotionAdapter{
  LabelInfo theLabelInfo;
  Label theLabel;

  //Constructor
  MyMouseMotionProcessor(
                     LabelInfo inLabelInfo, Label inLabel){
    //save incoming object reference
    theLabelInfo = inLabelInfo;
    theLabel = inLabel;
  }// end constructor

  public void mouseDragged(MouseEvent e){
    System.out.println("Drag = " + e);
    //Save mouse coordinates during the drag operation.
    theLabelInfo.mouseDraggedX = e.getX();
    theLabelInfo.mouseDraggedY = e.getY();


    /* How does this work?
    Dragging the mouse on the yellow label produces a
    stream of events of this type.  This method is called
    once for each event. When an event occurs, if the
    mouse has been dragged a distance equal to a full
    pixel (or more), in either or both directions,
    the X and Y values of the mouseDragged event
    are used to reposition the label by that distance in
    one or both of those directions.  This causes the
    label to slide under the mouse pointer so that the
    X and Y values of the next mouseDragged event revert
    back to the original Y and Y values of the original
    mousePressed event.  In other words, the motion of the
    label is such as to cause the coordinates of the
    mouseDragged event to continue to be equal to the
    coordinates of the original mousePressed event, or to
    cause the difference between the two to be zero.
```

```
    Think of the following expressions as a feedback system
    that strives to drive an error signal to zero where
    the error signal is the difference between the
    coordinates of the current mouseDragged event and the
    coordinates of the original mousePressed event.

    Note that when the yellow label is dragged rapidly
    and randomly for an extended period of time, the system
    simply quits generating mouseDragged events.  This may
    be a bug in JDK 1.1.3 for Win95.
    */

    int newX = theLabelInfo.labelX
                              + theLabelInfo.mouseDraggedX
                              - theLabelInfo.mousePressedX;
    int newY = theLabelInfo.labelY
                              + theLabelInfo.mouseDraggedY
                              - theLabelInfo.mousePressedY;

    //move label to the new location
    theLabel.setLocation(newX,newY);

    //save the new location information
    theLabelInfo.labelX = newX;
    theLabelInfo.labelY = newY;

  }//end mouseDragged()

  public void mouseMoved(MouseEvent e){
  /*This method is implemented here simply for
    illustration.  It displays the contents of the
    object passed in for each mouseMoved event.

    It also illustrates a possible bug in JDK 1.1.3
    for Win 95.  In particular, when the mouse is moved
    rapidly and randomly within the yellow label for
    an extended period, at some point, mouseMoved
    events cease to be generated.
    */
    System.out.println("Move = " + e);
  }//
}//end class MyMouseMotionProcessor
//=======================================================

//The following listener is used to terminate the
// program when the user closes the Frame object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
//=======================================================
```

Review

Q - Without viewing the following solution, write a Java application that meets the specifications given in the comments at the beginning of the application.

A - See specifications and solution below.

```
/*File SampProg126.java  from lesson 92
Copyright 1997, R. G. Baldwin
Without viewing the following solution, write a Java
application that meets the specifications given below.

The solution shown below was successfully tested using
JDK 1.1.3 under Win95.

Write an application that displays a Frame object with
a width of 300 pixels and a height of 200 pixels.  Put
your name in the banner at the top of the Frame.

When you move your mouse pointer into the client area of
the Frame, the coordinates of the mouse pointer appear
directly above the pointer.  As you move the mouse pointer
around in the client area, the coordinates of the pointer
continue to be displayed above the pointer.

When you move the mouse pointer out of the client area, the
coordinates of the exit point appear and remain in view
until you again move the pointer into the client area.

If you press one of the mouse buttons while moving the
mouse in the client area, the coordinates of the point
where you pressed the button appear and remain there until
you release the mouse button, at which time the
coordinates of the mouse pointer resume being displayed.

Closing the frame terminates the program and returns
control to the operating system.

End of specifications.

*/
//=========================================================
import java.awt.*;
import java.awt.event.*;

public class SampProg126 {
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class SampProg126
//=========================================================

//Subclass Frame in order to override the paint method.
class MyFrame extends Frame{
  int xCoor;
  int yCoor;
```

```
  MyFrame(String name){//constructor
    this.setTitle("Copyright 1997, R.G.Baldwin");
    this.setSize(300,200);
    this.setName(name);
  }//end constructor

  public void paint(Graphics g){
    //display coordinate information on the Frame
    g.drawString("" + xCoor + ", " + yCoor, xCoor, yCoor);
  }//end paint()
}//end class MyFrame
//=========================================================

class GUI {
  public GUI(){//constructor
    //Create a visual object of type MyFrame named Frame1
    MyFrame myFrame1 = new MyFrame("Frame1");
    myFrame1.setVisible(true);

    //Instantiate and register Listener object which will
    // terminate the program when the user closes
    // the Frame.
    WProc1 winProcCmd1 = new WProc1();
    myFrame1.addWindowListener(winProcCmd1);

    //Instantiate and register Listener object which will
    // process mouseMoved events to display coordinate
    // information on the Frame object named myFrame1.
    MyMouseMotionProcessor mouseMotionProc =
                      new MyMouseMotionProcessor(myFrame1);
    myFrame1.addMouseMotionListener(mouseMotionProc);

  }//end constructor
}//end class GUI definition
//=========================================================

//This listener class monitors for mouseMove events and
// displays the coordinates of the mouse pointer when the
// mouse is moved inside the client area of the Frame.
class MyMouseMotionProcessor extends MouseMotionAdapter{
  MyFrame refToFrame1; //save references to the Frame

  //Constructor
  MyMouseMotionProcessor(MyFrame inFrame1){
    //save incoming object reference
    refToFrame1 = inFrame1;
  }// end constructor

  public void mouseMoved(MouseEvent e){
    //Get X and Y coordinates of mouse pointer and store
    // in instance variables of the Frame object
    refToFrame1.xCoor = e.getX();
    refToFrame1.yCoor = e.getY();
    //Force a repaint to display the coordinate information
    refToFrame1.repaint();
```

```
  }//end mouseMoved()

}//end class MyMouseMotionProcessor

//=========================================================

//The following listener is used to terminate the program
// when the user closes the frame object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
//=========================================================
```

-end-