Practical Exercise 11 – Mouse events

The purpose of this practical is to learn how to use the MouseListener and MouseMotionListener in your programs.

## MouseListener

To use the MouseListener:

1. import java.awt.event.\*
2. add “implements MouseListener” to the class header
3. include addMouseListener(this) in init()
4. add all of the following methods (whether you use them or not):

* mousePressed() is called when the mouse button is pressed on a component
* mouseClicked() is called when mouse button is clicked (pressed and released) on a component
* mouseReleased() is called when the mouse button has been released on a component
* mouseEntered() is called when the mouse enters a component
* mouseExited() is called when the mouse exits a component.

## MouseMotionListener

MouseMotionListener is implemented in the same way, and includes the following methods:

* mouseDragged() is called when the mouse is pressed on a component and then dragged
* mouseMoved() is called when the mouse cursor has been moved onto a component but no buttons have been pushed.

# MouseDemo

The following sample code displays the coordinates of the pointer whenever you click on the window with the mouse.

You will notice a bunch of new techniques in this code, most notably:

1. Image and Graphics2D objects are used to display the output on the screen in a "buffered" way. This means we can access the Graphics2D methods (which are in many cases are the same as the Graphics methods we're familiar with) to build up the next image to be displayed when the paint() method is next called.
2. The main() class creates the window, a JFrame, and adds two JPanels to it. One is for the main drawing area, and a second one contains the Clear button.
3. The ActionListener is added to the Clear button in a way that we haven't seen before.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class MyCanvas extends JPanel implements MouseListener

{

Image image;

Graphics2D graphics2D;

int x, y;

public void init()

{

addMouseListener(this);

}

public void paint(Graphics g)

{

if (image == null)

{

image = createImage(getSize().width, getSize().height);

graphics2D = (Graphics2D)image.getGraphics();

graphics2D.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

clear();

}

g.drawImage(image, 0, 0, null);

}

public void clear()

{

graphics2D.setColor(Color.white);

graphics2D.fillRect(0, 0, getSize().width, getSize().height);

graphics2D.setColor(Color.black);

repaint();

}

public void mousePressed(MouseEvent e)

{

clear();

x = e.getX();

y = e.getY();

graphics2D.drawString("Mouse is at (" + x + "," + y + ")", x, y);

repaint();

}

public void mouseClicked(MouseEvent e)

{

}

public void mouseReleased(MouseEvent e)

{

}

public void mouseEntered(MouseEvent e)

{

}

public void mouseExited(MouseEvent e)

{

}

}

public class MouseDemo

{

public static void main(String[] args)

{

JFrame window = new JFrame("Mouse Demo");

window.setLayout(new BorderLayout());

// Create and add the sketch area to the window

MyCanvas sketchArea = new MyCanvas();

sketchArea.init();

window.add(sketchArea, BorderLayout.CENTER);

// Create and add a button to the window

JPanel panel = new JPanel();

JButton clearButton = new JButton("Clear");

clearButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

sketchArea.clear();

}

});

panel.add(clearButton);

window.add(panel, BorderLayout.SOUTH);

// Finish setting up the window

window.setSize(280, 300);

window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

window.setVisible(true); }

}

# Task 1 – Basic mouse events

1. Get the above MouseDemo program working.
2. Add to this program to make use of the other mouse methods, so that the position of the mouse, as well as the type of event, is displayed (either on the window itself or in the system console) for any event.

For example:

public void mouseClicked(MouseEvent e)

{

graphics2D.drawString("Mouse clicked at (" + x + "," + y + ")", x, y);

repaint();

}

public void mouseEntered(MouseEvent e)

{

System.out.println("The mouse has entered");

}

# Task 2 – CS Paint

1. Create a simple drawing program that allows you to create works of art like this. The idea is that as you drag the mouse it leaves a trail of dots on the window.
2. To use these methods you must implement MouseMotionListener by including it in the class header and you must call addMouseMotionListener() in init().

For example:

class MyCanvas extends JPanel implements MouseListener, MouseMotionListener

...

public void init()

{

addMouseListener(this);

addMouseMotionListener(this);

}

1. I'll leave you to work out what needs to happen with mousePressed(), but here's a suggestion for mouseDragged():

public void mouseDragged(MouseEvent e)

{

x = e.getX();

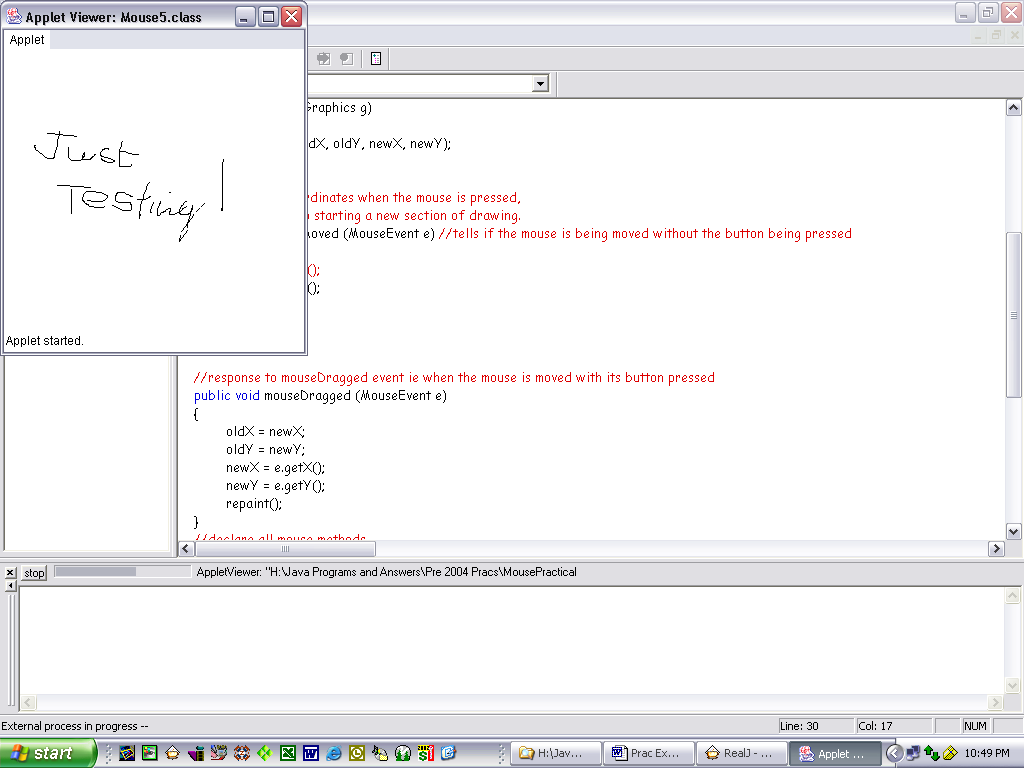
y = e.getY();

graphics2D.fillOval(x-2, y-2, 4, 4);

repaint();

}

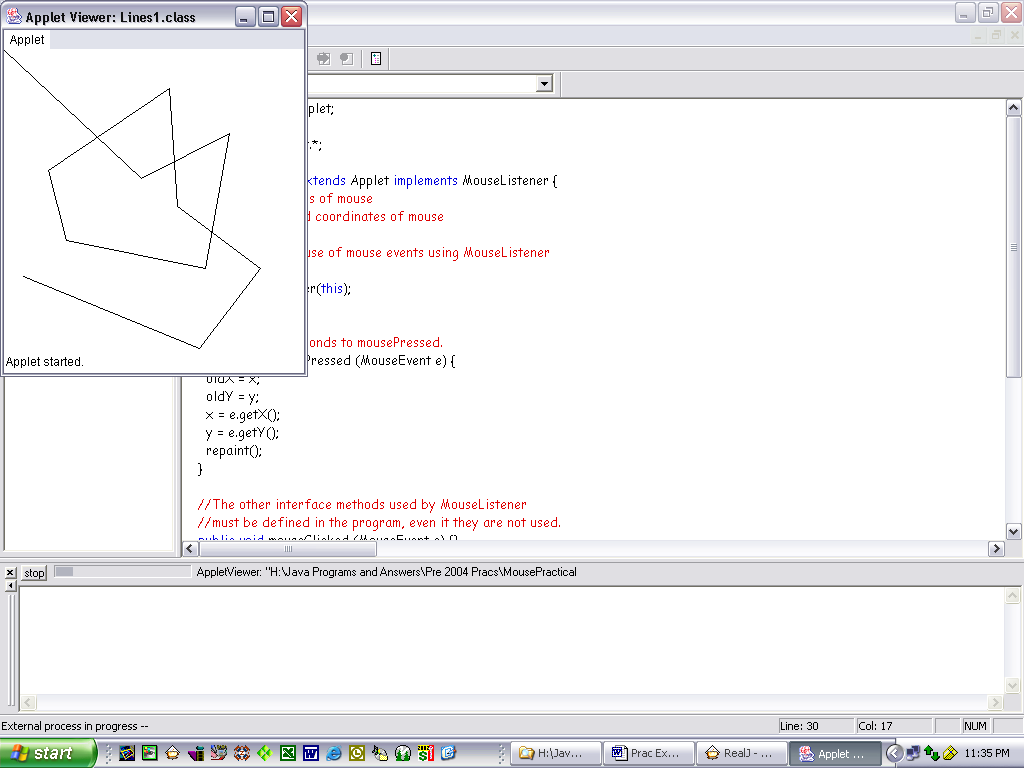
# Task 3 – CS Paint v2.0 Drag and Click

Modify your basic paint program so that:

1. when the mouse is dragged, a continuous (curved) line is drawn, and
2. when the mouse is clicked, a straight line is drawn (i.e. from where the mouse was last dragged/clicked to the current mouse location).

To draw a line, you can use the following code:

graphics2D.drawLine(oldX, oldY, currentX, currentY);

Where:

* currentX and currentY are the current x and y coordinates of the mouse:

currentX = e.getX();

currentY = e.getY();

* oldX and oldY give the previous x and y coordinates:

oldX = currentX;

oldY = currentY;

Hint: Depending on how you implement this behaviour, it may be necessary to set the oldX and oldY to the mouse coordinates when the mouse is pressed (which corresponds to starting a new section of drawing).

# Task 4 – Brush selection (optional stretch goal – bonus mark)

Improve your paint program so that the user can select different brush shapes, sizes and/or colours. (e.g. by pressing a key on the keyboard).