Java – Mouse Actions

C&G criteria: 5.2.1, 5.4.1, 5.4.2, 5.6.2.

The events so far have depended on creating Objects and detecting when they receive the event. The position of the mouse on the screen can also be used to fire code. If a program is run in the applet viewer the Window can be dragged to any size. If the relative mouse position is important it may be preferable to run code in a browser where the size of the running program can be set in HTML. The 1.1 mouse events are very different to those in Java 1.0, 5.42 in the C&G task list refers to the 1.0 model. This program with draw the current mouse coordinates as a String.

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


public class MouseList extends Applet implements MouseListener
{  //MouseListener is much like ActionListener and the rest
        int downx, downy;
        public void init()
        { //do not miss out this essential line
                this.addMouseListener(this);
        }
        public void mouseClicked(MouseEvent e)
        {
                downx = e.getX(); //get x and y
                downy = e.getY();
                repaint();
        }
        public void mouseReleased(MouseEvent e)
        {  //note that all 5 mouse actions must be present in code even if only 1 is used
        }
        public void mousePressed(MouseEvent e)
        {
        }
        public void mouseEntered(MouseEvent e)
        {
        }
        public void mouseExited(MouseEvent e)
        {
        }
        public void paint(Graphics g)
        {//draw the current coordinates at their position
                g.drawString(downx+", "+downy,downx,downy);
        }
}
```

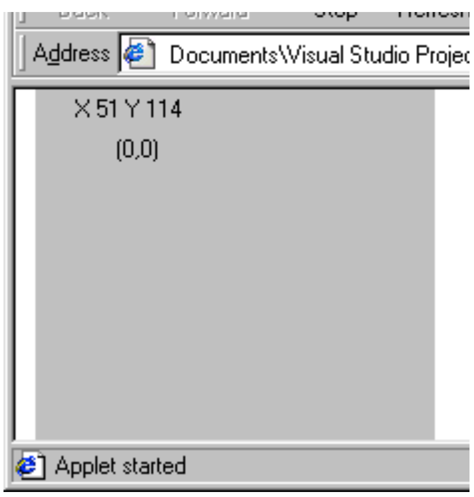
Run the code as it stands and with 1 of the mouse actions commented out.

Modifications

Use Java.Math.Random to create a random mouse coordinate and have the user click on screen to find the hidden spot. A 200 by 200 Window is large enough for the applet. Inform the user when within 10 of the hidden spot. For the sake of testing it is useful to show the hidden coordinates. This ensures that the correct result is achieved.

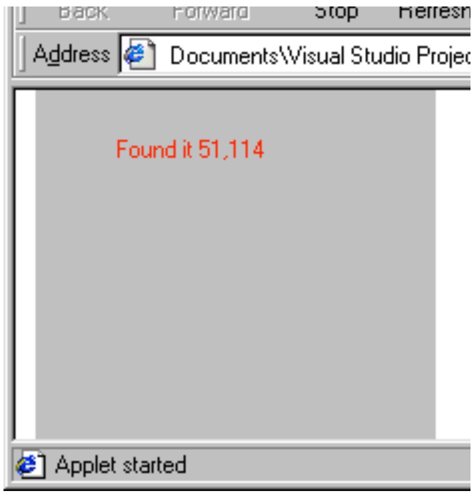
The following code will calculate the positive offset value from the point clicked (myX) and that chosen randomly (javaX) and store it as the integer offX.

```
                    offX=java.lang.Math.abs(myX-javaX);
```



```
X 51 Y 114
   (0,0)
```

Applet started



```
Found it 51,114
```

Applet started

Drawing lines

If Java knows 2 sets of points it can draw a line between them. Dragging achieved by
```
          public void mouseDragged(MouseEvent e)
```

This must be accompanied by
```
          public void mouseMoved(MouseEvent e)
```

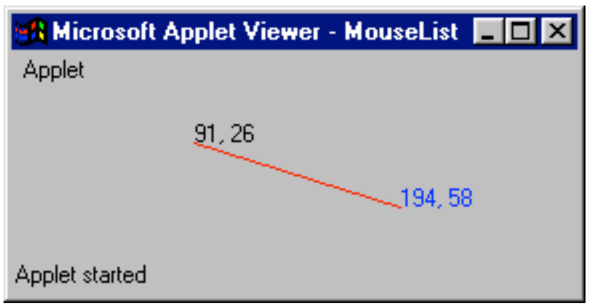
In the same way that all the clicking events have to be together. They are not implemented by MouseListener but by MouseMotionListener. There is likely to be some clicking in the same program to a possible start is:

```
public class MouseList extends Applet implements MouseMotionListener, MouseListener
{
          public void init()
          {
                    this.addMouseListener(this);
                    this.addMouseMotionListener(this);
          }
}
```

To drag the initial and final mouse coordinates are required. So 2 sets of x and y values will have to be assigned to variables. A Boolean variable drawing will detect if dragging is starting or stopping.

```
public void mousePressed(MouseEvent e)
{ // mousePressed detects the start of dragging.
        if(!drawing)
        {
                downx = e.getX();
                downy = e.getY();
                repaint();
        }
}

public void mouseReleased(MouseEvent e)
{  //stop of dragging
        drawing = false;
}

public void mouseDragged(MouseEvent e)
{  //the drag constantly updates mx1 and my1.
        mx1 =e.getX();
        my1=e.getY();
        drawing = true;
        repaint();
}

public void paint(Graphics g)
{//last mx, my are where string is drawn
        g.drawString(downx+", "+downy,downx,downy);
        if (drawing ==true)
        {
                g.setColor(Color.red);
                g.drawLine(downx,downy,mx1,my1);
                g.setColor(Color.blue);
                g.drawString(mx1+", "+my1,mx1,my1);
        }
}
```

Complete this code to draw lines.

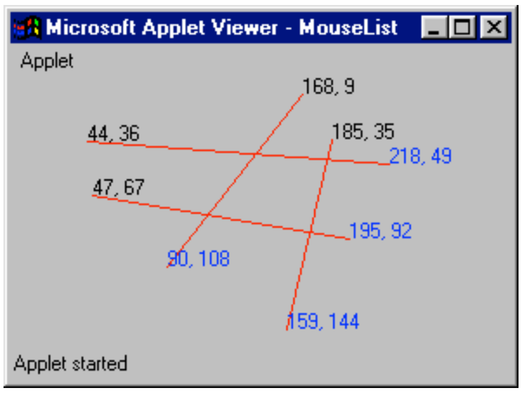


Modifications

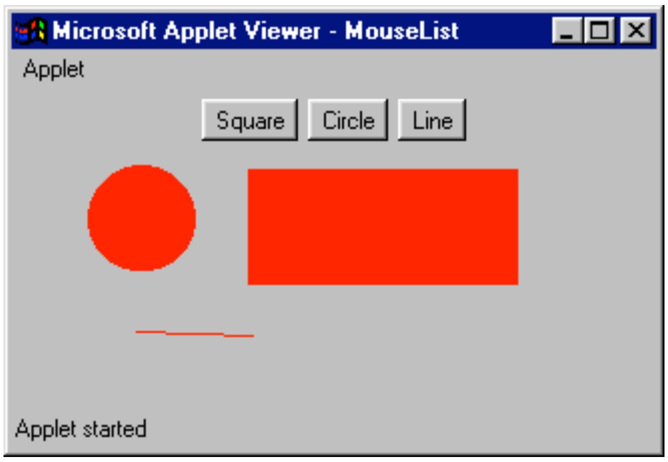
1. When a new line is drawn the old one is lost because paint redraws the canvas based on what information it has. To redraw existing data their details could be placed in variables and accessed by paint but there is an easier way, using getGraphics(). The following method adapts the paint method from the above program. It is called instead of repaint.

To draw multiple lines some changes will have to be made to when true and false are set.

```
public void redraw()
{
        Graphics g = getGraphics();
        g.drawString(downx+",
"+downy,downx,downy);
        if (drawing ==true)
        {
                g.setColor(Color.red);
                g.drawLine(downx,downy,mx1,my1);
                g.setColor(Color.blue);
                g.drawString(mx1+",
"+my1,mx1,my1);
                drawing=false;
        }
}
```



2. Use alternative graphics methods to draw circles, ovals and squares. The redraw method can be adapted for the various shapes. Allow the user to choose what shape to draw, Buttons can handle this.

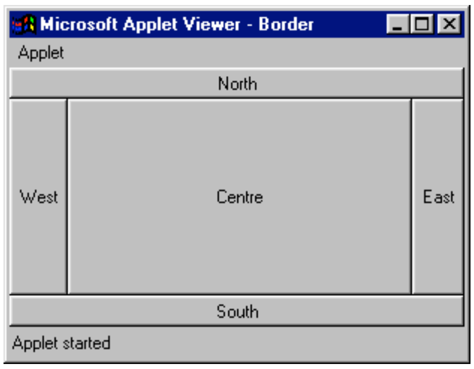


Layouts

The layout in all programs so far is the default or flow layout. The controls flow across the Window in the order that they were created. If the Window shape changes the controls move about.

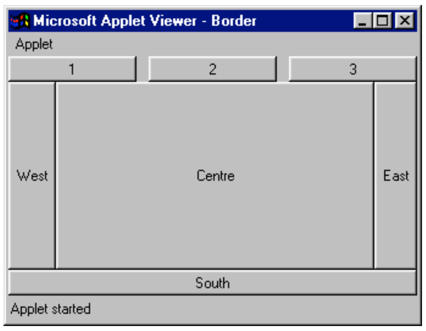
The simplest layout is by the use of BorderLayout that allows controls to be placed in the 4 compass points of the Window or in the centre. The following code places 5 buttons in the 5 positions. It will not work without - setLayout(new BorderLayout());

```
Button bNorth, bSouth, bEast, bWest, bCentre;

public void init()
{
        setLayout(new BorderLayout());
        bNorth= new Button("North");
        add(bNorth,"North");
        bSouth= new Button("South");
        add(bSouth,"South");
        bEast= new Button("East");
        add(bEast,"East");
        bWest= new Button("West");
        add(bWest,"West");
        bCentre= new Button("Centre");
        add(bCentre,"Center");
}
```



This is not a lot of use but each of the 5 positions can hold more than 1 control through the use of a Panel object. The following code uses a Panel to distribute 3 Buttons in the North area.

```
Button part1, part2, part3;
Panel layout;
public void init()
{
        setLayout(new BorderLayout()); //must have this
        layout =new Panel();//create panel 1st
        //the GridLayout specifies (rows, cols, horiz spacing, vert spacing)
        layout.setLayout(new GridLayout(1,3,10,10));
        part1= new Button("1");
        layout.add(part1); //add controls to the Panel
        part2= new Button("2");
        layout.add(part2);
        part3= new Button("3");
        layout.add(part3);
        add(layout,"North"); //finally add the Panel to
        //the Window
        //add other controls
}
```



Drawing on a Canvas

To allow a program to have a drawing area and controls without the controls sitting on the drawing area Java allows a Canvas to be used. A Canvas is an object like other controls but the various drawing methods relating to it have to be specified for the Canvas. This requires a Canvas Class to be created and an object of that Class added to the user interface Class. So we end up with more than 1 Class in the program. This is the norm in Java programming.

This is an entire Canvas Class that is used to draw shapes. There are a number of key points that would not normally be followed in a user defined Class. All the methods are public and some public variables from the user interface Class are used by this Class (int mx, my, mx1, my1).

```
class MyCanvas extends Canvas implements MouseMotionListener, MouseListener
{ // the Class extends Canvas so as to make use of the methods of Class Canvas.
        public MyCanvas()
        {//constructor method – here is adds the mouse listeners
                this.addMouseListener(this);
                this.addMouseMotionListener(this);
        }
        public void mouseClicked(MouseEvent e)
        {       }
        public void mouseReleased(MouseEvent e)
        {  //redraws the screen
                repaint();
        }
        public void mousePressed(MouseEvent e)
        {
                mx=e.getX();
                my=e.getY();
                mx1=e.getX();
                my1=e.getY();
        }
        public void mouseEntered(MouseEvent e)
        { }
        public void mouseExited(MouseEvent e)
        { }
        public void mouseDragged(MouseEvent e)
        {
                mx1=e.getX();
                my1=e.getY();
        }
        public void mouseMoved(MouseEvent e)
        { }
        public void update(Graphics g)
        {
                g.setColor(currentColour);  //currentColour is set by the interface Class
                if(currentTool==0)  //currentTool is set by the interface Class
                        g.drawLine(mx,my,mx1,my1);
                else if(currentTool==1)
```

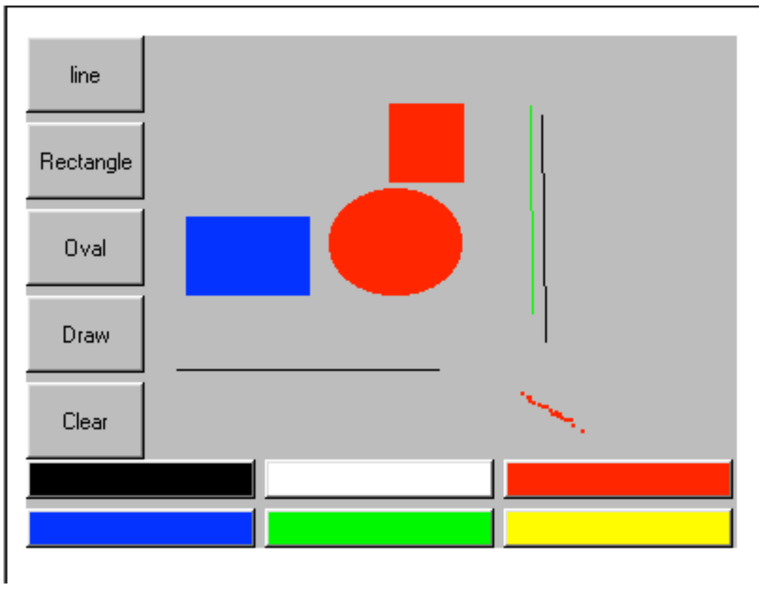
```
                    g.fillRect(mx,my, mx1-mx,my1-my);
            else if (currentTool==2)
                    g.fillOval(mx,my,mx1-mx,my1-my);
            else if(currentTool==3) //draw a very small rectangle
                    g.fillRect(mx1,my1,2,2);  //approximates a paint brush
            else if(currentTool==4) // if the drawing area is 300 x 300
                    g.clearRect(0,0,300,300);
        }
}
```

This Class is of no use without an appropriate interface Class to set the draw type, draw colour, starting and ending positions. It is not possible to shove all this code into a single Class due to the nature of the Canvas object.

```
public class Draw2 extends Applet implements ActionListener
{ //these are suggested controls – any names and functions can be used.
        Panel palette,tools;//2 distinct panels
        Button line, rect,oval,draw,clear;
        Button black, white, red, blue, green, yellow;
        MyCanvas Drawarea; //instance of sub class

        int mx=0;
        int my=0;
        int mx1=0;
        int my1=0;
        int currentTool=0;//0=line, 1=rect, 2 = oval
        Color currentColour = Color.White;

        public void init()
        {       //set controls and layout for palette
                setLayout(new BorderLayout());
                //add controls in a suitable layout

                drawarea = new MyCanvas();//add the class instance
                add(Drawarea,"Center");
        }
        public void actionPerformed(ActionEvent evt)
        {// Button actions
        }
//Type the Canvas Class here –note Canvas is not public
}
```

Use the Panels and GridLayout to set up an interface similar to that on the next page. The exact positions and colours are not important. There is no need to implement action code until a reasonable interface is created. This will ensure that any interface set up code is correct before writing any actions.

actionPerformed(ActionEvent evt) only sets the currentColour and currentTool. All the drawing actions are already handled by the Canvas object.



Modifications

1. Add another 2 colours to draw with.

2. Go back to the Calculator program and improve the layout. All the number Buttons could be part of 1 panel and the operator Buttons on another.