Mouse Tracking

Use MouseListener MouseMotionListener to get notifications about mouse events over a component.
The component itself is the source of the notifications -- add the listener to the component.

Listener vs. Adapter Style

Problem
- Listener has a bunch of abstract methods -- e.g. 5 in MouseListener.
- You typically only care about one or two, so implementing all 5 is a bore.

Solution
- "Adpater" class has empty { } definitions of all the methods
- Then you only need to implement the ones you care about -- the adapter catches the others.

Bug
- If you type the prototype slightly wrong, your method will be ignored -- e.g. MousePressed() instead of the correct mousePressed()

MouseListener Interface

```
public interface MouseListener extends EventListener {

    /**
     * Invoked when the mouse has been clicked on a component.
       (press+release)
     */
    public void mouseClicked(MouseEvent e);

    /**
     * Invoked when a mouse button has been pressed on a component.
     */
    public void mousePressed(MouseEvent e);

    /**
     * Invoked when a mouse button has been released on a component.
     */
    public void mouseReleased(MouseEvent e);

    /**
     * Invoked when the mouse enters a component.
     */
    public void mouseEntered(MouseEvent e);

    /**
     * Invoked when the mouse exits a component.
     */
    public void mouseExited(MouseEvent e);
}
```

Mouse Adapter Class

```
public abstract class MouseAdapter implements MouseListener {
    /**
     * Invoked when the mouse has been clicked on a component.
     */
    public void mouseClicked(MouseEvent e) {}

    /**
     * Invoked when a mouse button has been pressed on a component.
     */
    public void mousePressed(MouseEvent e) {}

    /**
     * Invoked when a mouse button has been released on a component.
     */
    public void mouseReleased(MouseEvent e) {}

    /**
     * Invoked when the mouse enters a component.
     */
    public void mouseEntered(MouseEvent e) {}

    /**
     * Invoked when the mouse exits a component.
     */
    public void mouseExited(MouseEvent e) {}
}
```

Press : MouseListener

How to hear about a mouse press on a component...

```
component.addMouseListener( new MouseAdapter() {
   public void mousePressed(MouseEvent e) {
      // called when mouse button first pressed on component
```

Motion: MouseMotionListener

How to hear about a mouse gesture with mouse button held down...

```
component.addMouseMotionListener( new MouseMotionAdapter() {
   public void mouseDragged(MouseEvent e) {
      // called as mouse is dragged, after initial click
```

JComponent = source

The JComponent where the click began is the "source" object for the mouse events. Register with the component to hear about clicks on it.

Local Co-Ords

Notifications about the mouse event will use the local co-ord system of the component where they happened. (This is similar to the way paintComponent() works -- using the local co-ord system.)

The "delta" rule for mouse motion

Wrong: absolute

Use the current co-ords of the mouse--
Set the position of whatever it is to those co-ords

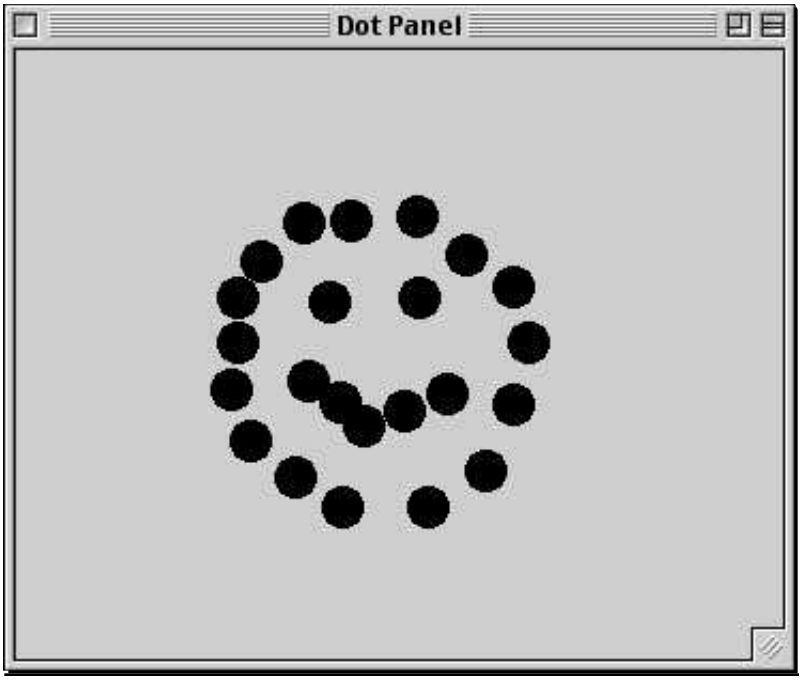
Right: relative

Get the current co-ords
Compare the last co-ords
Apply that delta to whatever it is

Test case

Aclick-release with no motion should not change any state -- relative mouse tracking gets this right.

DotPanel Example




```
// DotPanel.java
/**
 The DotPanel class demonstrates a few things...

 -Mouse tracking -- clicking makes a new point, clicking
 on an existing point moves it. The data model is the collection
 of points where there is a dot on screen.

 -Smart repaint -- only repaints the needed rectangle when a dot moves

*/

import java.awt.*;
import javax.swing.*;
import java.util.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;
```

```
class DotPanel extends JPanel  {
   private ArrayList dots; // represent each dot by its center point
   public final int SIZE = 20;   // diameter of one dot

   // remember the last point for mouse tracking
   private int lastX, lastY;
   private Point lastPoint;

   public boolean smartRepaint = true;

    // we'll use this later
    // dirty = changed from disk version
   private boolean dirty;



   /**
    Utility test-main creates a DotPanel in a window.
   */
   public static void main(String[] args) {
      JFrame frame = new JFrame("Dot Panel");

      JComponent container = (JComponent) frame.getContentPane();

      DotPanel dotPanel = new DotPanel(300, 300, null);

      container.add(dotPanel);



      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);
   }


   /**
    Create an empty DotPanel. Load the contents of the
    given File if it is non-null.
   */
   public DotPanel(int width, int height, File file) {
      super();
      setPreferredSize(new Dimension(width, height));
      setOpaque(true);
      setBackground(Color.white);

      dirty = false;
      dots = new ArrayList();

      if (file != null) {
         load(file);
      }
```

```
   /*
    Mouse Strategy:
    -if the click is not on an existing dot, then make a dot
    -note where the first click is into lastX, lastY
    -then in MouseMotion: compute the delta of this position
    vs. the last
    -Use the delta to change things (not the abs coordinates)
   */

   addMouseListener( new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
         //System.out.println("press:" + e.getX() + " " + e.getY());

         Point point = findDot(e.getX(), e.getY());
         if (point == null) { // make a dot if nothing there
            point = addDot(e.getX(), e.getY());
         }

         // Note the starting setup to compute deltas later
         lastPoint = point;
         lastX = e.getX();
         lastY = e.getY();

      }
   });


   addMouseMotionListener( new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent e) {
         //System.out.println("drag:" + e.getX() + " " + e.getY());

         if (lastPoint != null) {
            // compute delta from last point
            int dx = e.getX()-lastX;
            int dy = e.getY()-lastY;
            lastX = e.getX();
            lastY = e.getY();

            // apply the delta to that point
            moveDot(lastPoint, dx, dy);
         }
      }
   });

}

/**
 Generates a repaint for the rect around one dot
 smart: repaint the rect just around the dot
 standard: repaint the whole panel
*/
public void repaintDot(Point point) {
   if (smartRepaint) {
      repaint(point.x-SIZE/2, point.y-SIZE/2, SIZE, SIZE);
   }
   else {
      repaint();
   }
```

```
}


/**
 Moves a dot from one place to another.
 Trick: needs to repaint both the old and new locations
 Moving components get this right automatically --
 see component.setBounds().
*/
public void moveDot(Point point, int dx, int dy) {
   repaintDot(point);   // repaint its old rectangle
   point.x += dx;
   point.y += dy;
   repaintDot(point);   // repaint its new rectangle

   setDirty(true);
}


/**
 Private utility -- adds a dot to the data model.
*/
private Point addDot(int x, int y) {
   Point point = new Point(x, y);
   dots.add(point);
   repaintDot(point);

   setDirty(true);

   return(point);
}


/**
 Finds a dot in the data model that contains
 the given point, or return null.
*/
public Point findDot(int x, int y) {
   Iterator it = dots.iterator();
   while (it.hasNext()) {
      Point point = (Point)it.next();
      int left = point.x-SIZE/2;
      int top = point.y-SIZE/2;
      if (left<=x && x<left+SIZE &&
         top<=y && y<top+SIZE) {
         return(point);
      }
   }
   return(null);
}
```

```
/**
 Standard override -- draws all the dots.
*/
public void paintComponent(Graphics g) {
   // As a JPanel subclass we need call super.paintComponent()
   // so JPanel will draw the background for us.
   super.paintComponent(g);

   Iterator it = dots.iterator();

   // standard draw: just iterate through and draw them all.
   while (it.hasNext()) {
      Point point = (Point)it.next();
      g.fillOval(point.x - SIZE/2, point.y-SIZE/2, SIZE, SIZE);
   }

}
```