EECS 1720
Building Interactive Systems

Lecture 19 :: Event Handling [5]

Mouse Motion and Graphics

YORK UNIVERSITÉ UNIVERSITY

This Lecture:

Associate Action, Mouse & Key events with:

- Mouse Motion
- Images
- Canvas
- Graphical elements / drawing (revisited – the swing version!)

Mouse Motion Events

- Recall: *MouseListener* interface adds methods:
  - public void **mouseEntered**(MouseEvent e);
  - public void **mouseExited**(MouseEvent e);
  - public void **mouseClicked**(MouseEvent e);
  - public void **mousePressed**(MouseEvent e);
  - public void **mouseReleased**(MouseEvent e);

- **MouseMotionListener** interface adds these additional methods:
  - public void **mouseMoved**(MouseEvent e);
  - public void **mouseDragged**(MouseEvent e);

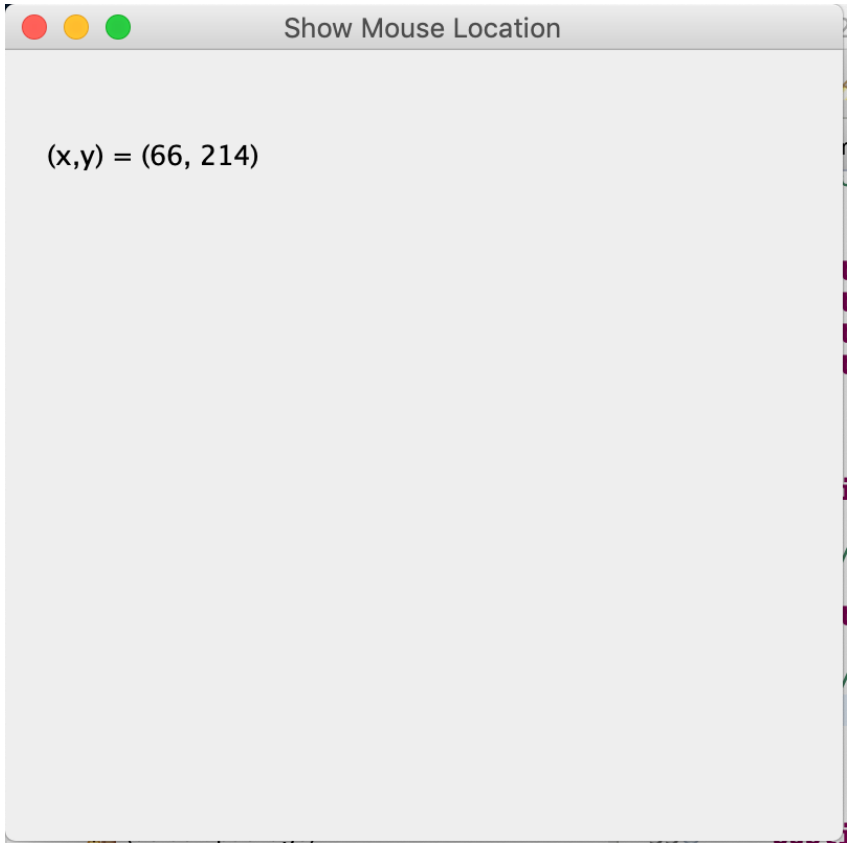
- mouseMoved
  - Invoked when a mouse button is moving on/within a component it is registered to (but no buttons on the mouse have been pushed)
  - MOUSE_MOVED events will continue to be delivered to the component on which a move is registering

mouseMoved example 1:

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;


import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class MouseLocationPaint extends JFrame implements MouseMotionListener {

     private JLabel mousePosition;
     private Color pen = Color.BLUE;

     public MouseLocationPaint(String title){     // constructor
          super(title);

          this.mousePosition = new JLabel();      // JLabel to show mouse position
          this.mousePosition.setBounds(20,40,150,20);

          this.add(this.mousePosition);           // register this MouseLocationPaint
          this.addMouseMotionListener(this);      // instance as MouseMotionListener

          this.setSize(400,400);
          this.setLayout(null);
          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          this.setVisible(true);
     }
```

mouseMoved example 1:

```
        // ...

    public void mouseMoved(MouseEvent e) {
        // whenever mouse is moved over anything in the JFrame
        this.mousePosition.setText("(x,y) = ("
                                   + e.getX() + ", " + e.getY() + ")");

    }

    public void mouseDragged(MouseEvent e) {
        // NOT USED IN THIS EXAMPLE
    }


    public static void main(String[] args) {
        MouseLocationPaint frame =
                new MouseLocationPaint("Show Mouse Location");
    }


}
```



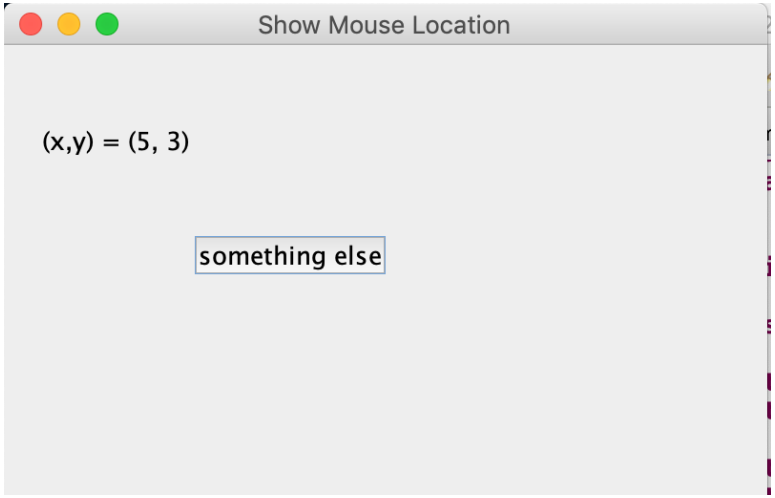
Recall:

what is getX()
and getY()
relative to?

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class MouseLocationPaint extends JFrame        nts MouseMotionListener {

    private JLabel mousePosition;
    private Color pen = Color.BLUE;

    public MouseLocationPaint(String title
        super(title);

        this.mousePosition = new JLabel      ;  // JLabel to show mouse position
        this.mousePosition.setBounds(2  ,40,150,20);

        this.add(this.mousePosition);      // register this MouseLocationPaint
        this.addMouseMotionListener(this); // instance as MouseMotionListener

        // ...
```

Attached to MouseLocationPaint instance (which is a subclass of JFrame)

```
// Attach to the Content Pane ?

Container pane = this.getContentPane();
pane.addMouseMotionListener(this);


// OR … Attach to another JComponent?

JButton button = new JButton("something else");
button.setBounds(100, 100, 100, 20);
button.addMouseMotionListener(this);
pane.add(button);


// …  inside constructor
```

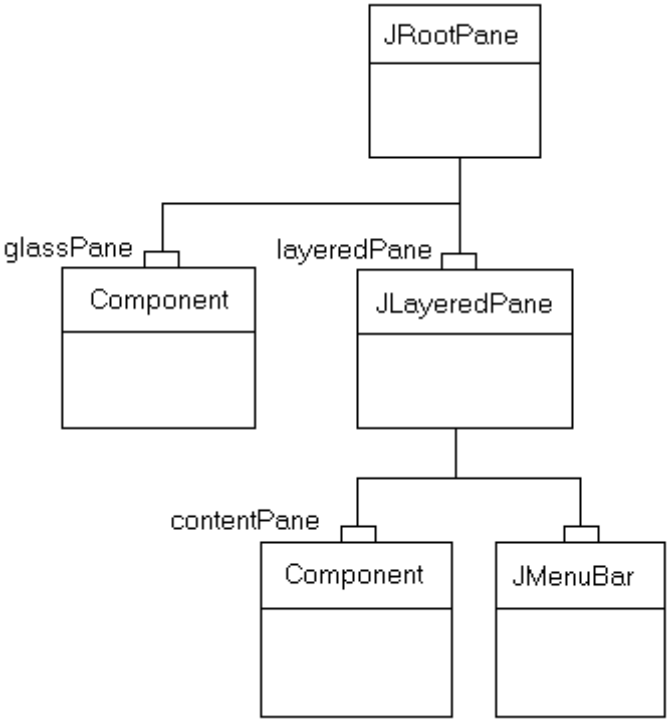


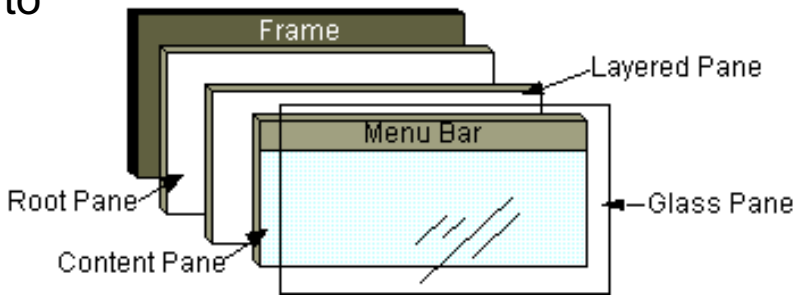

```
public void mouseMoved(MouseEvent e) {
     // SHOW RELATIVE MOUSE POSITION
     this.mousePosition.setText("(x,y) = (" + e.getX() + ", " + e.getY() +
     ")");

     // SHOW ABSOLUTE SCREEN MOUSE POSITION
     this.mousePosition.setText("(x,y) = (" + e.getXOnScreen() + ", " +
     e.getYOnScreen() + ")");

     // CONSOLE MESSAGE
     System.out.println(e);
}
```

YORK UNIVERSITÉ UNIVERSITY

Recall…

- JFrame has-a JRootPane object
- JRootPane has several parts:
  - layered pane
    - positions both the content pane and optional menu bar
  - menu bar (optional)
  - **content pane**
    - container for root pane's visible components (excl. menu bar)
    - this is where most of your GUI components will go
  - **glass pane**
    - transparent, can be switched on to intercept interaction events



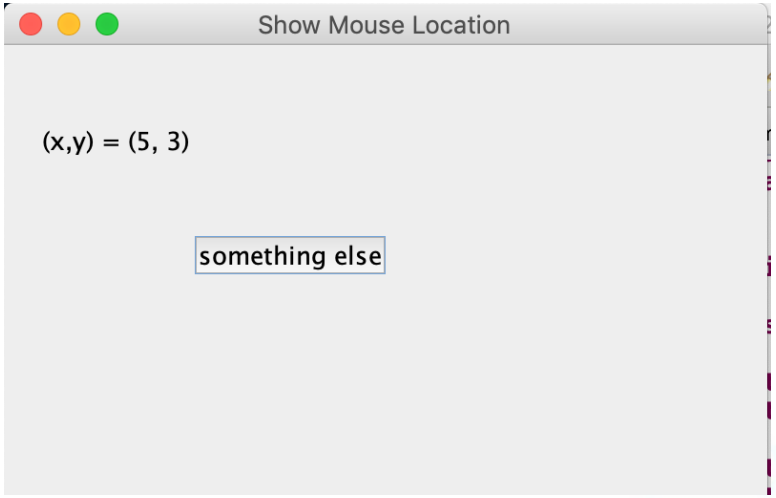


What if we

- remove listener from button?
- set/enable a glassPane (with the listener)?

```
this.add(this.mousePosition);           // register this MouseLocationPaint
//this.addMouseMotionListener(this);    // instance as MouseMotionListener

Container pane = this.getContentPane();
pane.addMouseMotionListener(this);

JButton button = new JButton("something else");
button.setBounds(100, 100, 100, 20);
//button.addMouseMotionListener(this);
pane.add(button);

Component glass = this.getGlassPane();
glass.addMouseMotionListener(this);
glass.setVisible(true);
this.setGlassPane(glass);
```

can intercept mouse events



- MouseDragged
  - Invoked **when a mouse button is pressed on a component and then dragged**.
  - MOUSE_DRAGGED events will continue to be delivered to the component where the drag originated until the mouse button is released (regardless of whether the mouse position is within the bounds of the component).

Outputting MOUSE_DRAGGED and MOUSE_MOVED events (using println)

```
Event e =
java.awt.event.MouseEvent[MOUSE_DRAGGED,(153,271),absolute(153,293),
modifiers=Button1,extModifiers=Button1,clickCount=1] on frame0

Event e =
java.awt.event.MouseEvent[MOUSE_DRAGGED,(152,271),absolute(152,293),
modifiers=Button1,extModifiers=Button1,clickCount=1] on frame0



Event e =
java.awt.event.MouseEvent[MOUSE_MOVED,(65,24),absolute(65,46),
clickCount=0] on frame0
Event e =
java.awt.event.MouseEvent[MOUSE_MOVED,(65,24),absolute(65,46),
clickCount=0] on frame0
```

Event Handling Demos

Simple interactive paint
Keys+Mouse Events (move-a-line)
Drag-based Selection (crop & filter)
Using Adaptors (vs. Listeners)

Very Simple Interactive Paint (swing version)

- Here:
  - A JFrame instance has a Graphics instance

- Recall:
  - Canvas object (added to the main content pane)
  - We can also access a Graphics instance within a Canvas (if we want to restrict drawing to part of the GUI)

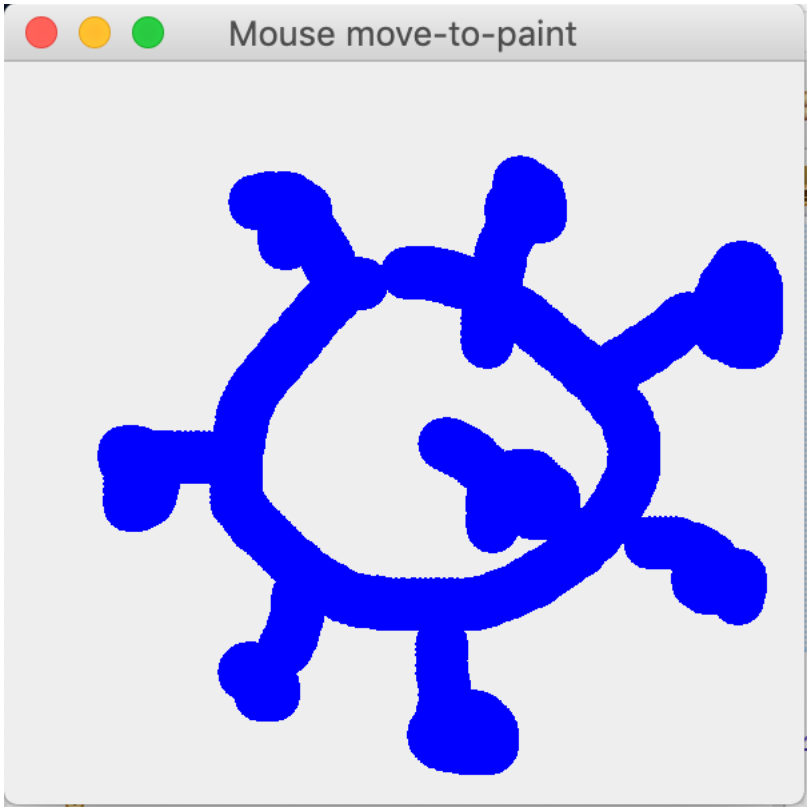
Mouse Motion + Graphics

- Inheriting from JPanel
- Inheriting from Canvas
  - i.e. create your own specialized type of Canvas (subclass)

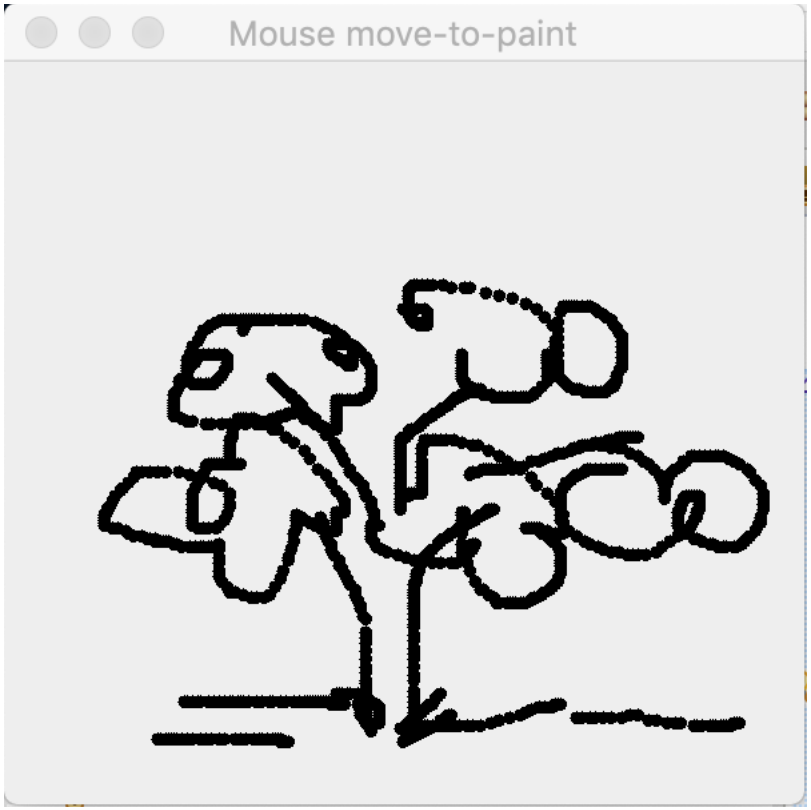
```
import java.awt.*;
import java.awt.event.*;


import javax.swing.JFrame;

public class MouseCursorPaint extends JFrame implements MouseMotionListener {


    public MouseCursorPaint(String title){
        super(title);
        // note there are NO GUI Panels/Canvases or Components added
        // this is because a JFrame has a  Graphics component
        // we can get a reference to directly

        this.addMouseMotionListener(this);


        this.setSize(300,300);
        this.setLayout(null);
        this.setResizable(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
```

```
    public void mouseDragged(MouseEvent e) {

        // if MOUSE_DRAGGED event continues

        System.out.println("Event e = " + e); // shows drag events

        // note that g may be cast down to (Graphics2D) - since Graphics2D
        is a child of Graphics

        Graphics g = this.getGraphics();
        Graphics2D g2d = (Graphics2D) g;            // not used here

        // Paint a black oval at mouse position (if mouse is being dragged)
        g.setColor(Color.BLACK);
        g.fillOval(e.getX(),e.getY(),5,5);

    }

    public void mouseMoved(MouseEvent e) {
        // Does nothing at this point
        System.out.println("Event e = " + e); // shows move events
    }


    public static void main(String[] args) {
        new MouseCursorPaint("Mouse move-to-paint");
    }

}
```

Move to paint



fillOval (size 20x20)



fillOval (size 5x5)

YORK UNIVERSITÉ UNIVERSITY

Other things you can draw at mouse location

- An Image?
  - ImageIcon objects have a method .getImage() which will return an image object (which can be passed to g.drawImage() - where g is a Graphics2D reference)
  - g.drawImage(…);

In fact, you can use:

mousePressed to capture a start position

mouseDragged to create a dynamic line/arc/shape

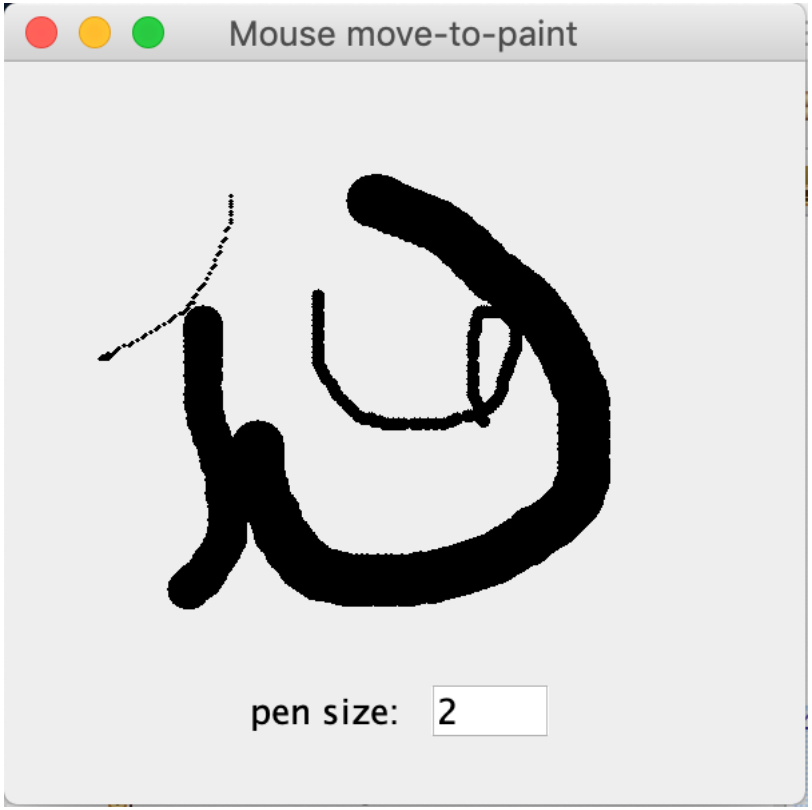
mouseReleased to stop

Drawing within a Canvas

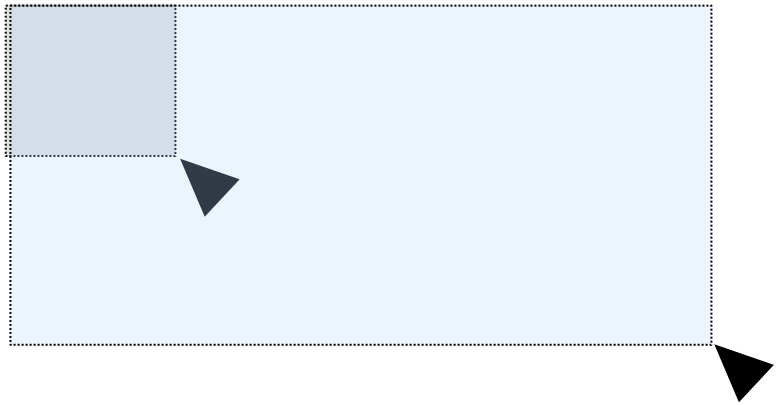
```
// imports not shown

public class MouseCursorPaintCanvas extends JFrame
                        implements MouseMotionListener {

        private JTextField penSize;
        private Canvas drawArea;

        public MouseCursorPaintCanvas(String title){

                super(title);
                // drawPanel (holds canvas)
                JPanel drawPanel = new JPanel();

                this.drawArea = new Canvas();
                this.drawArea.setSize(400, 300);
                this.drawArea.addMouseMotionListener(this);
                drawPanel.add(drawArea);

                // controlPanel holds GUIs for pen settings
                JPanel controlPanel = new JPanel();
                controlPanel.setLayout(new FlowLayout());
                controlPanel.setPreferredSize(new Dimension(400,50));
```

```
        JLabel penLabel = new JLabel("pen size: ");
        this.penSize = new JTextField("5");
        this.penSize.setPreferredSize(new Dimension(50,20));

        // controlPanel holds pen textfield and label

        controlPanel.add(penLabel);
        controlPanel.add(this.penSize);

        this.add(drawPanel, BorderLayout.CENTER);
        this.add(controlPanel, BorderLayout.SOUTH);
        this.setSize(300,300);


        this.setResizable(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);

    }
```

```
    public void mouseDragged(MouseEvent e) {

        // if MOUSE_DRAGGED event continues
        System.out.println("Event e = " + e); // shows drag events

        // get graphics reference from Canvas (drawArea)
        Graphics g = drawArea.getGraphics();
        Graphics2D g2d = (Graphics2D) g; // not used here

        // Paint a black oval at the mouse position (if mouse is being dragged)
        g.setColor(Color.BLACK);

        g.fillOval(e.getX(),e.getY(),
                Integer.parseInt(penSize.getText()),
                Integer.parseInt(penSize.getText()));

    }

    public void mouseMoved(MouseEvent e) {
        // Does nothing at this point
        System.out.println("Event e = " + e); // shows move events
    }


    public static void main(String[] args) {
        new MouseCursorPaintCanvas("Mouse move-to-paint");
    }

}
```



Selection Rectangle (controlled by mouse press & drag)



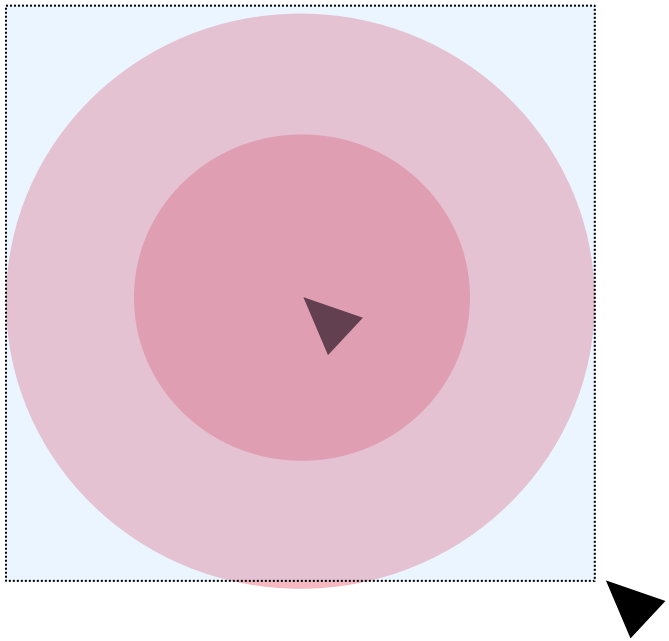
// use fields to save and modify rectangles

// as the mouse moves, update them and redraw

```
Rectangle currentRect = null;
Rectangle rectToDraw = null;
```

Painting shapes?

- Same idea
- Usually need MouseListener & MouseMotionListener for these scenarios



Custom Painting (in Swing)

A number of ways to achieve:

- Retrieve the graphics context of any **`JComponent`** explicitly, and use it directly (as we have done previously)
  - e.g. from a Canvas or JPanel

```
Canvas c = new Canvas(…);
Graphics g = c.getGraphics();
```

- Extend Canvas (old way with AWT) ..
  - Override Frame's →
    - **`paint(Graphics g) or paintComponent(Graphics g)`** method
  - OR explicitly invoke repaint() to update graphics

YORK UNIVERSITÉ UNIVERSITY

Custom Painting (extending any JComponent)

- e.g. Create a class (or inner class) that extends JPanel
  - Override its **`paint (Graphics g)`** method:

```
@Override
public void paint (Graphics g) {
  super.paint(g);  // paint parent's background
  ...
}
```

preferred

Why do it this way?

- At times, we need to *explicitly refresh the display* (e.g., in game and animation).
  - we invoke the JComponent's **repaint()** method
    - The Windowing Subsystem will in turn call back the **paint()** with the current Graphics context
  - You can **repaint()** a particular **JComponent** such as a **JPanel** or the entire **JFrame**
    - The children contained within the **JComponent** will also be automatically repainted!

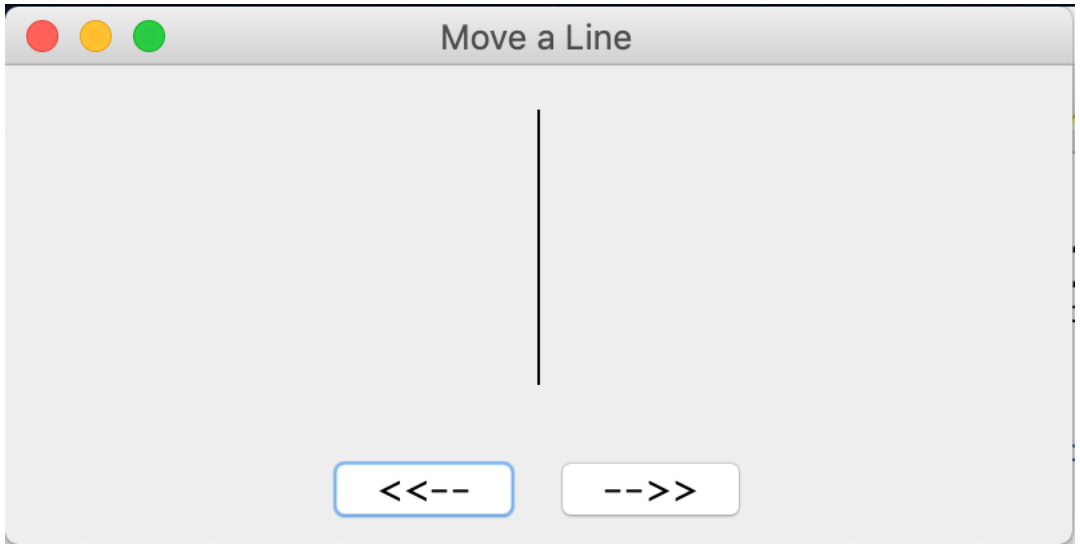
Event Handling Demos

Keys+Mouse Events (move-a-line)

Using Adaptors (vs. Listeners)

Drag-based Selection (crop & filter)

Example 1: overriding paint()



Use buttons to move an object on screen (use repaint and special version of a component for draw area)

Example 1:

```
// Class fields
// buttons
private JButton btnLeft;
private JButton btnRight;

// a custom drawing canvas (an inner class extends JPanel)
private DrawCanvas canvas;
```

```
public SimpleMoveALine(String title) {

    super(title);
    // Set up a panel for the buttons
    JPanel btnPanel = new JPanel(new FlowLayout());

    ButtonListener actionHandler = new ButtonListener();

    this.btnLeft = new JButton("<<--");
    this.btnRight = new JButton("-->>");
    this.btnLeft.addActionListener(actionHandler);
    this.btnRight.addActionListener(actionHandler);
    btnPanel.add(btnLeft);
    btnPanel.add(btnRight);

    // Set up a custom drawing JPanel
    canvas = new DrawCanvas();
    canvas.setPreferredSize(new Dimension(CANVAS_WIDTH,
                                          CANVAS_HEIGHT));
    // ...
}
```

Specialized JPanel

```
/**
* Define inner class DrawCanvas, which is a JPanel used
* for custom drawing.
*/
private class DrawCanvas extends JPanel {

    @Override
    public void paint(Graphics g) {

        super.paint(g);

        g.setColor(LINE_COLOR);
        g.drawLine(x1, y1, x2, y2); // Draw the line

    }

}
```

YORK UNIVERSITÉ UNIVERSITY

ActionListener

```
private class ButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {

        JButton clicked = (JButton) e.getSource();

        if (clicked == btnLeft) {
        x1 -= 10;
        x2 -= 10;
        }
        if (clicked == btnRight) {
        x1 += 10;
        x2 += 10;
        }
        canvas.repaint();
    }

}
```

What's happening?

- repaint() call is invoked on *canvas*, which is an instance of our specialized JPanel class "DrawCanvas"

- More exactly.. when you invoke the repaint() method to repaint any JComponent:
  - the Windowing subsystem *calls* paint() method.
  - The paint() method then *calls* three methods:
    - paintComponent()
    - paintBorder() and
    - paintChilden().

What's happening?

- repaint() call is invoked on *canvas*, which is an instance of our specialized JPanel class "DrawCanvas"

- More exactly.. when you invoke the repaint() method to repaint any JComponent:
  - the Windowing subsystem *calls* paint() method.
  - The paint() method then *calls* three methods:
    - paintComponent()
    - paintBorder() and
    - **paintChilden().**

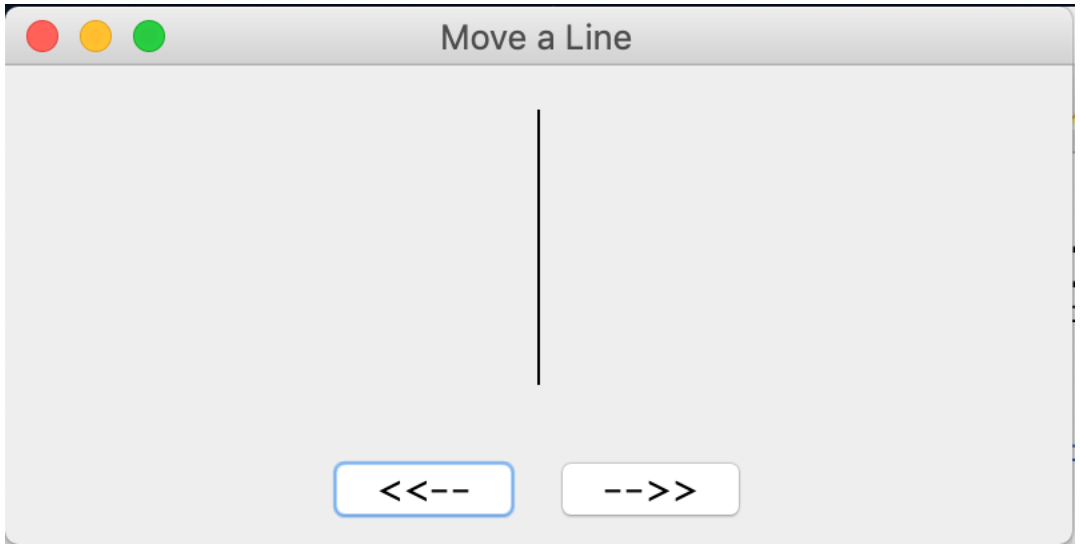
This will invoke paint() on any child objects (sub JPanels, or other components added to the current JComponent)

Our paint() method

- Has access to its own Graphics reference (g)
- Invokes the paint() method of the super class
  - Paints background of parent JComponent (in our case the background of the JPanel)
    - If not called.. can also draw background explicitly
      - e.g. use a fillRect(..) call; or
      - using setOpaque(false) to make component transparent
- Then does any specialized drawing that we provide!
  - Set the colour and draw a line in our case (using x1,y1,x2,y2)
- This is constantly invoked whenever something is changed with the window (position/resize)… or invoked explicitly (by our ActionListener)

Using alternative inputs to invoke same behaviour (keys or buttons)

- Add alternative key events to move line (sprite)

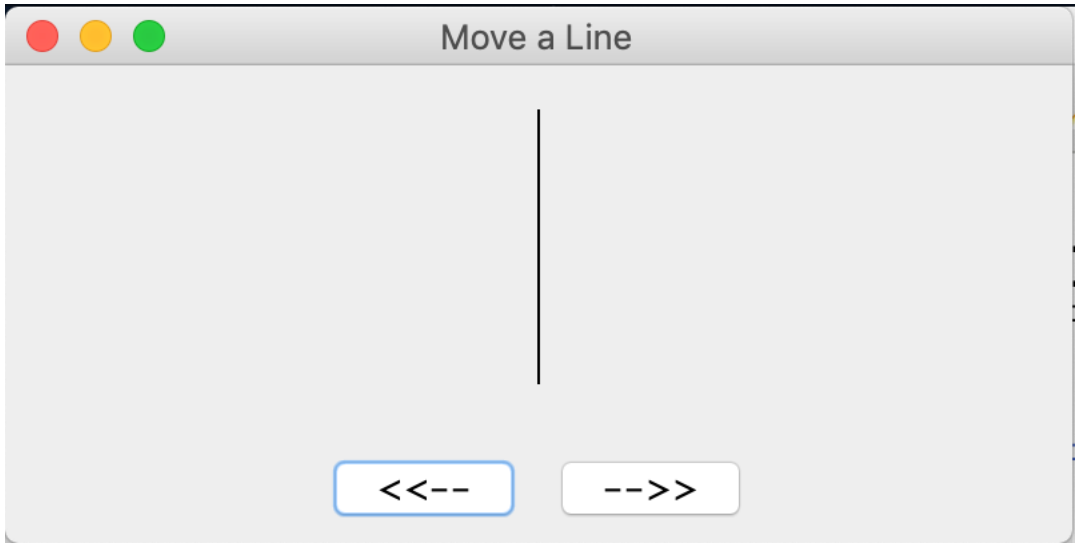


YORK UNIVERSITÉ UNIVERSITY

To include key events…

```
// INSIDE constructor

// let "super" JFrame fire the KeyEvent

this.addKeyListener(new MoveKeyListener());


// after setting up Jframe...
// set the focus to JFrame to receive KeyEvent

this.requestFocus();
```

Which listener is connected to what?



```
private class MoveKeyListener implements KeyListener {

    @Override
    public void keyPressed(KeyEvent e) {

        switch(e.getKeyCode()) {

            case KeyEvent.VK_LEFT:
            x1 -= 10;
            x2 -= 10;
            canvas.repaint();
            break;

            case KeyEvent.VK_RIGHT:
            x1 += 10;
            x2 += 10;
            canvas.repaint();
            break;
        }
    }

 // ...
```

```
    // ...


    @Override
    public void
    keyTyped(KeyEvent e) {
    // do nothing
    }

    @Override
    public void
    keyReleased(KeyEvent e) {
    // do nothing

    }

}
```

ActionListener (modified)

```
private class ButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {

        JButton clicked = (JButton) e.getSource();

        if (clicked == btnLeft) {
        x1 -= 10;
        x2 -= 10;
        }
        if (clicked == btnRight) {
        x1 += 10;
        x2 += 10;
        }
        canvas.repaint();
        requestFocus();
    }

}
```

Return focus to application JFrame

Using Adaptors

What is an adaptor?

- A class that implements a listener
  - Think of it as a class that provides empty implementations for the methods of that interface

- By extending an adaptor:
  - Only need to override the method you want to provide an implementation of (instead of providing all implementations)

MouseInputAdapter (only have to define mouse methods we need)

```
private class MyListener extends MouseInputAdapter {

    public void mousePressed(MouseEvent e) {

        // capture initial position of mouse
        int x = e.getX();
        int y = e.getY();

        // create a new Rectangle after capturing initial
        // position of mouse
        currentRect = new Rectangle(x, y, 0, 0);

        // update drawable rectangle
        updateDrawableRect(getWidth(), getHeight());

        repaint(); // calls JFrame's repaint
    }
```

```
    public void mouseDragged(MouseEvent e) {

        updateSize(e);
    }


    public void mouseReleased(MouseEvent e) {

        updateSize(e);

    }

}
```

```
// Cleans up rectangle to be drawn (if negative width/height or
rect off screen)
private void updateDrawableRect(int compWidth, int
                                                    compHeight) {

    // get position and size of currentRect (to create rectToDraw)
    int x = currentRect.x;
    int y = currentRect.y;
    int width = currentRect.width;
    int height = currentRect.height;

    //Make the width and height positive, if necessary. (resets x,y
    to top left)
    if (width < 0) {
        width = 0 - width;
        x = x - width + 1;
        if (x < 0) {
            width += x;
            x = 0;
        }
    }

    // similarly for y (not shown)
```

```
    //The rectangle shouldn't extend past the drawing area.
    if ((x + width) > compWidth) {
        width = compWidth - x;
    }

    if ((y + height) > compHeight) {
        height = compHeight - y;
    }

    //Update rectToDraw after saving old value.

    if (rectToDraw != null) {
        rectToDraw.setBounds(x, y, width, height);
    }

    else {
        rectToDraw = new Rectangle(x, y, width, height);
    }

}
```

Triggered by mouseDragged and mouseReleased

```
void updateSize(MouseEvent e) {


    int x = e.getX();
    int y = e.getY();


    // modify currentRect's width and height only
    currentRect.setSize(x - currentRect.x,
                        y - currentRect.y);

    // clean up rectToDraw
    updateDrawableRect(getWidth(), getHeight());

    repaint();

}
```

Mouse-based Selection

- Use MouseListener and MouseMotionListener events

Demo

Use JLabel as the JComponent we will interact with

- So we can turn a JLabel into our specialized “canvas”

- Why?
  - JLabel easily can hold an image (ImageIcon)
  - JLabel is a JComponent, so we can override its paint (…) method
  - JLabel can register mouse events

```
public class SimpleMouseSelect extends JFrame {

    // class fields

    private JLabel rectBounds;

    private SelectionCanvas canvas;


    // ...
```

```
public SimpleMouseSelect(String title, String imgFile) {

    super(title);
    Container pane = this.getContentPane();
    pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));

    ImageIcon img = new ImageIcon(imgFile);

    this.canvas = new SelectionCanvas(img);

    this.add(this.canvas);

    this.rectBounds = new JLabel("Drag within the image");
    this.rectBounds.setLabelFor(this.canvas);
    this.add(this.rectBounds);

    this.canvas.setAlignmentX(LEFT_ALIGNMENT);
    this.rectBounds.setAlignmentX(LEFT_ALIGNMENT);

    // setup Jframe (not shown)

}
```

```
private class SelectionCanvas extends JLabel {

    private Rectangle currentRect = null;
    private Rectangle rectToDraw = null;
    private Rectangle previousRectDrawn = new Rectangle();

    public SelectionCanvas(ImageIcon image) {

        super(image);    //This component displays an image

        setOpaque(true);
        setMinimumSize(new Dimension(10,10));

        MyListener myListener = new MyListener();
        addMouseListener(myListener);
        addMouseMotionListener(myListener);

    }
// ...
```

- currentRect -> dynamic rect (with mouse)
- rectToDraw -> cleaned up rect (one to draw/use)

  ** note a rect can flip if we drag in -x or –y

```
public void paint(Graphics g) {

    //paints the background and image
    super.paint(g);

    //If currentRect exists, paint a box on top.
    if (currentRect != null) {

        //Draw a rectangle on top of the image.
        g.setXORMode(Color.white); //Color of line varies

        g.drawRect(rectToDraw.x, rectToDraw.y,
                rectToDraw.width - 1, rectToDraw.height - 1);

        // update rectBounds (with rectToDraw):
        int width = rectToDraw.width;
        int height = rectToDraw.height;


        // ...
```

YORK UNIVERSITÉ UNIVERSITY

```
        //Make the coordinates look OK if a dimension is 0.
        if (width == 0) {
        width = 1;
        }
        if (height == 0) {
        height = 1;
        }

        rectBounds.setText("Rectangle goes from ("
        + rectToDraw.x + ", " + rectToDraw.y + ") to ("
        + (rectToDraw.x + width - 1) + ", "
        + (rectToDraw.y + height - 1) + ").");

    }
}
```

Doing something with the selection:

```
public void mouseReleased(MouseEvent e) {

    updateSize(e);

    // launch task to use selection here
    Picture cropped = ImageManip.crop(new Picture(origImage),
                      rectToDraw.x, rectToDraw.y,
                      rectToDraw.width, rectToDraw.height);

    Picture sepia = Question03.sepiaImage(new Picture(cropped));
    Picture edge = Question03.edgeImage(new Picture(cropped), 10);

    cropped.show();
    sepia.show();
    edge.show();

}
```