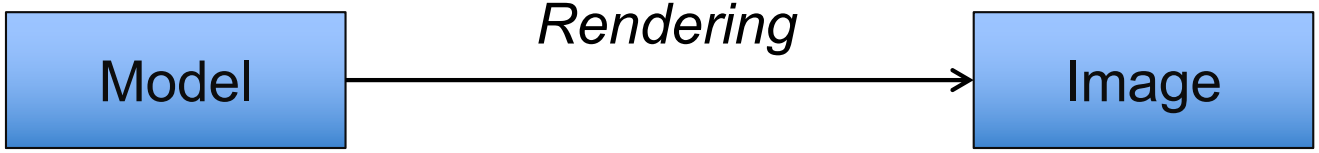
2D Graphics

Shape Models, Drawing, Selection

Graphic Models vs. Images

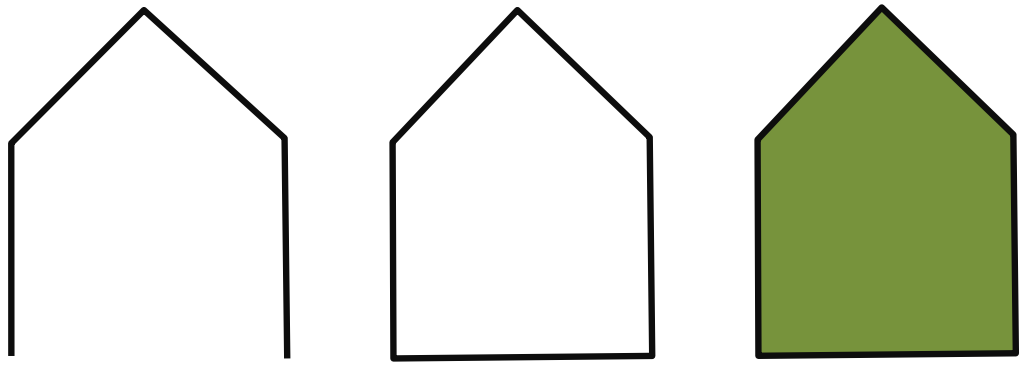
- Computer Graphics: the creation, storage, and manipulation of images and their models

- Model: a mathematical representation of an image containing the important properties of an object (location, size, orientation, color, texture, etc.) in data structures
- Rendering: Using the properties of the model to create an image to display on the screen
- Image: the rendered model



Example Shape Models

- A "shape model" is a type of model that includes all of the data that we need to draw a particular shape.
  - An array of points: {P1, P2, …, Pn}
  - Can be open, closed, filled …
  - Includes any other data that we need to draw
- Allows for reuse of shape models in code

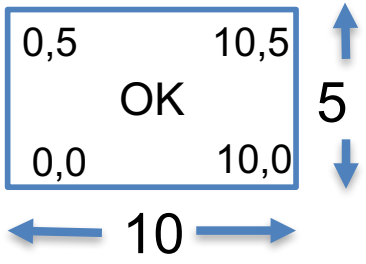


- Java supports primitive shapes (Lines, Ellipses, Rectangles) with tools for building others (CubicCurves, Paths, drawPolygon, drawPolyline, fillPolygon)

Example Shape Models

- We can also think of widgets as shape models
  - Consist of an array of points {P1, P2, …, Pn}
  - Properties describing how to draw it

- e.g. Button
  - Points
    - (0,0), (10,0), (5,0), (10,5)
  - Properties
    - Text: “OK”
    - Border width: 1 pixel
    - Border color: blue



Typical Interaction

- What do we need to support interaction with user interfaces?
  - Ability to draw shapes/widgets on the screen
    - at specified position, size, orientation
    - perhaps draw many copies, each different from the others
  - Ability to test when a shape/widget is “selected” or “clicked”
    - could be a filled or outlined polygon or a polyline
    - selections that “just miss” the shape should “snap” to shape

- Programmer tasks:
  - create a model of the shape
  - draw it
  - choose a “selection” paradigm
  - implement shape hit tests and/or inside tests (with snapping)
  - respond to events

Now: what we did in X, in Java
Next: 2D transformations

SimpleDraw: Drawing in Java

```
package twoD_graphics;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Path2D;

public class SimpleDraw {
    public static void main(String[] args) {
        JFrame f = new JFrame("SimpleDraw");          // jframe is the window
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(400, 400);                          // window size
        f.setContentPane(new Canvas());               // add canvas to jframe
        f.setVisible(true);                           // show the window
    }
}

// JComponent is a base class for custom components
class Canvas extends JComponent {

    // custom graphics drawing
    public void paintComponent(Graphics g) {
   …
    }
}
```

SimpleDraw: Drawing in Java

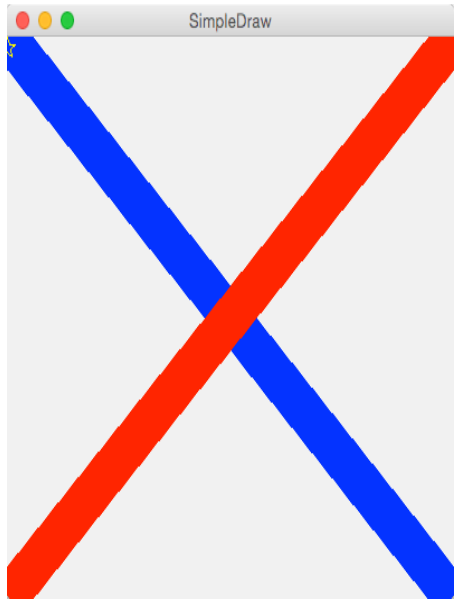
```
package twoD_graphics;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Path2D;

public class SimpleDraw { … }

// JComponent is a base class for custom components
class Canvas extends JComponent {

    // custom graphics drawing
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;          // cast to get 2D drawing funcs
        g2.setStroke(new BasicStroke(32));       // 32 pixel thick stroke
        g2.setColor(Color.BLUE);                 // make it blue
        g2.drawLine(0, 0, getWidth(), getHeight()); // draw line
        g2.setColor(Color.RED);
        g2.drawLine(getWidth(), 0, 0, getHeight());
        g2.setColor(Color.BLACK);
        g2.setStroke(new BasicStroke(1));
        g2.draw(new Star());
    }
}
```



SimpleDraw: Drawing in Java

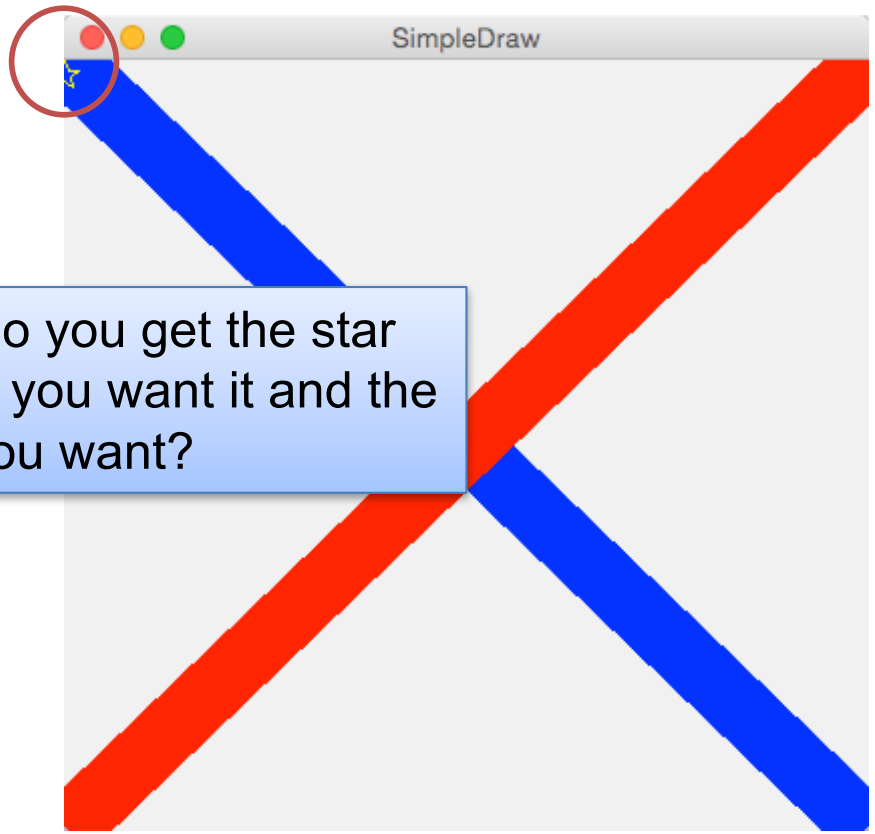
```
package twoD_graphics;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Path2D;

public class SimpleDraw { … }

class Canvas extends JComponent {…}
```

```
class Star extends Path2D.Double {
    public Star() {
        super(WIND_EVEN_ODD);
        this.moveTo(0, 0);
        this.lineTo(-1.5, 5);
        this.lineTo(-7, 5);
        this.lineTo(-2.5, 8);
        this.lineTo(-4.2, 13);
        this.lineTo(0, 10);
        this.lineTo(4.2, 13);
        this.lineTo(2.5, 8);
        this.lineTo(7, 5);
        this.lineTo(1.5, 5);
        this.lineTo(0, 0);
    }
}
```



Selection Paradigms

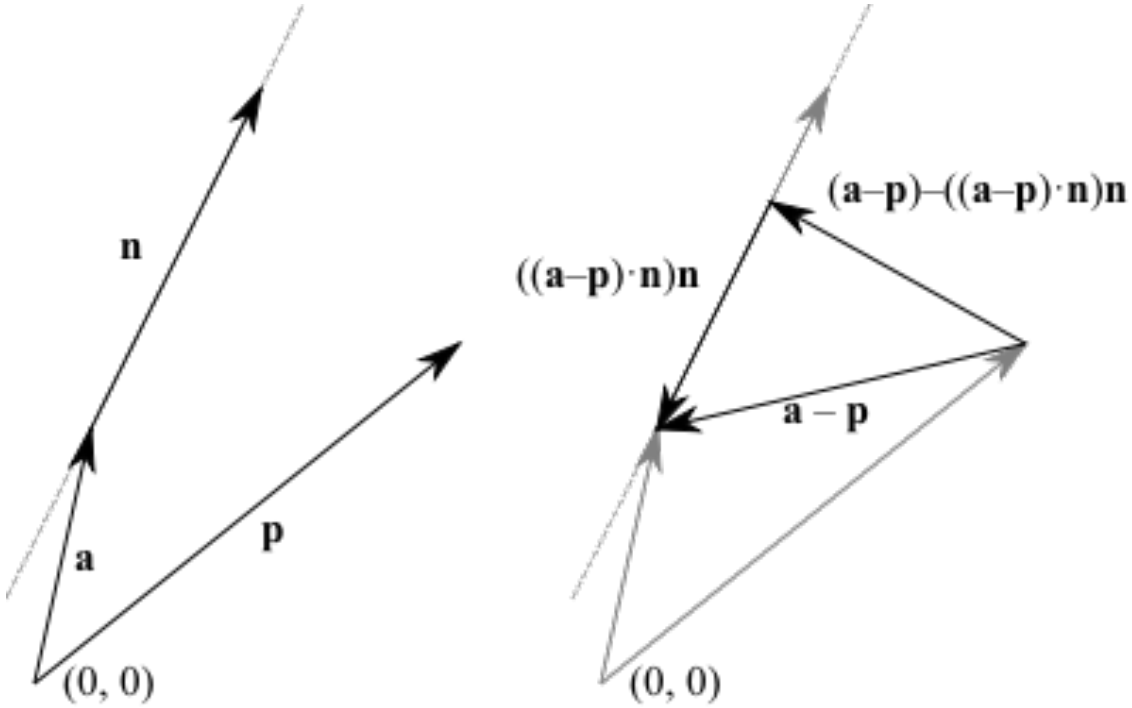
- Users need to be able to interact with drawn shapes.
- e.g.
  - Selecting or manipulating drawings
  - Interaction with widgets

- Approach that we'll use
  - Click selection
    - Works differently for lines vs. closed shapes
    - Example of each

- Alternate approaches not covered
  - Rubberband rectangle
  - Lasso (e.g. Chapter 14 of Olsen)

Closest Shape to Mouse Test

- Check distance from every line segment of every shape to mouse position (can be optimized …)
- Check distance from mouse to line segment using vector projection
- ClosestPointDemo.java



http://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line

NOTE ON ClosestPointDemo EXAMPLE

- The ClosestPointDemo example will NOT RUN ON YOUR MACHINE USING THE STANDARD INSTALL!!!!

- Need vecmath.jar
  - Download it online, place it in your directory, and use the –cp command line switch to include it
  - Or try installing java3d – ymmv.
  - javac -cp vecmath.jar ClosestPointDemo.java
  - java –cp .;vecmath.jar ClosestPointDemo

  - **NOTE THE ".;" IN THE SECOND COMMAND**

- The makefile included the samples will build and run it properly

Distance from Line to Point

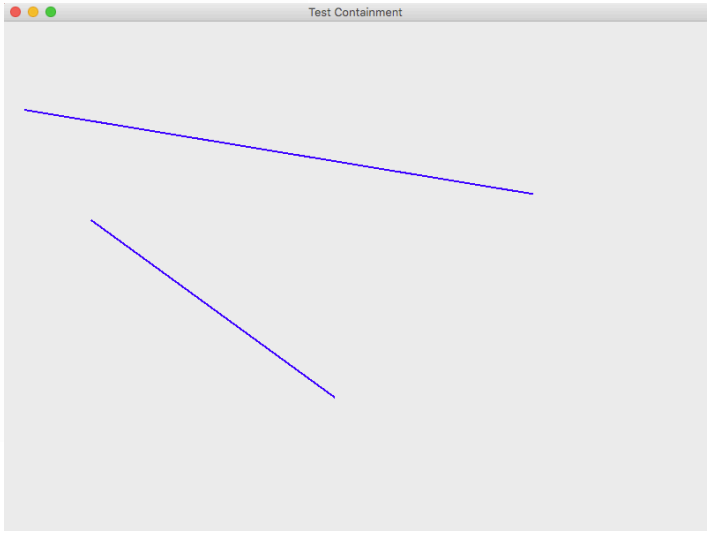
TestLine.java

```
int[] l1xvals = {25, 600};
int[] l1yvals = {100, 195};
int[] l2xvals = {100, 375};
int[] l2yvals = {225, 425};
```

```
g2.drawLine(l1xvals[0], l1yvals[0], l1xvals[1], l1yvals[1]);
g2.drawLine(l2xvals[0], l2yvals[0], l2xvals[1], l2yvals[1]);
```

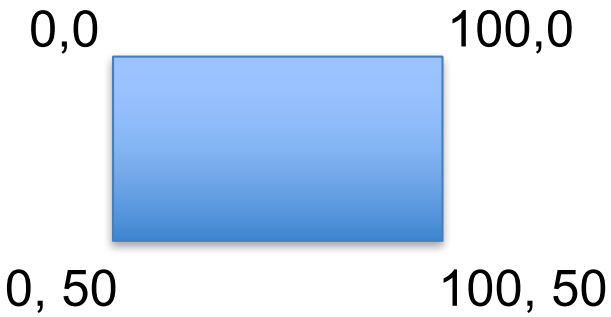
```
protected void testContainment(){
    double d1 = Line2D.ptSegDist(
        l1xvals[0], l1yvals[0],
        l1xvals[1], l1yvals[1],
        pos_x, pos_y);
    color1 = (d1 < 5.0); // 5 pixels distance

    double d2 = Line2D.ptSegDist(
        l2xvals[0], l2yvals[0],
        l2xvals[1], l2yvals[1],
        pos_x, pos_y);
    color2 = (d2 < 5.0);
}
```



Simplest : Selection of regular shapes

- The easiest thing to detect is a selection within a regular, closed shape.
  - e.g. imagine that these are widgets, and the mouse is clicked at point (x,y)

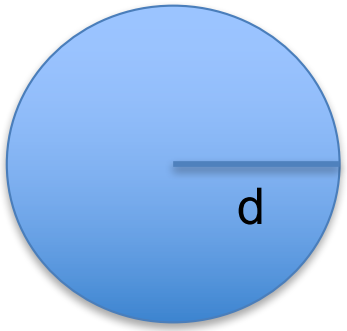


Point (x, y) is contained within the rectangular region if:

$0 < x < 100$ and $0 < y < 50$

Point (x, y) is contained within the circular region if:

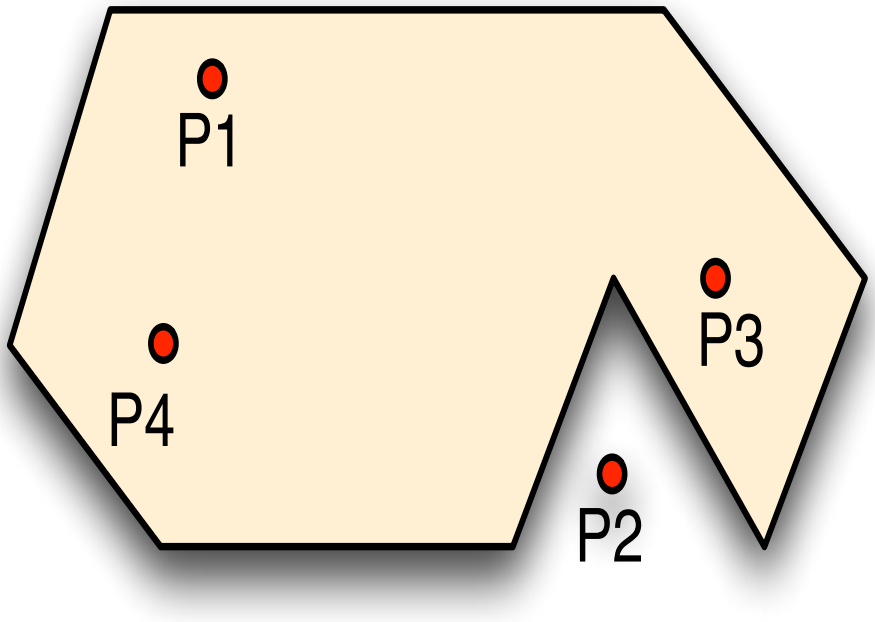
distance (center, point) < d



Harder: Selection of arbitrary shapes

Polygons are harder.

Is a point inside or outside a polygon?



See Polygon class in java.awt
See TestClick.java