CS324e - Elements of Graphics and Visualization

More Java2D Graphics

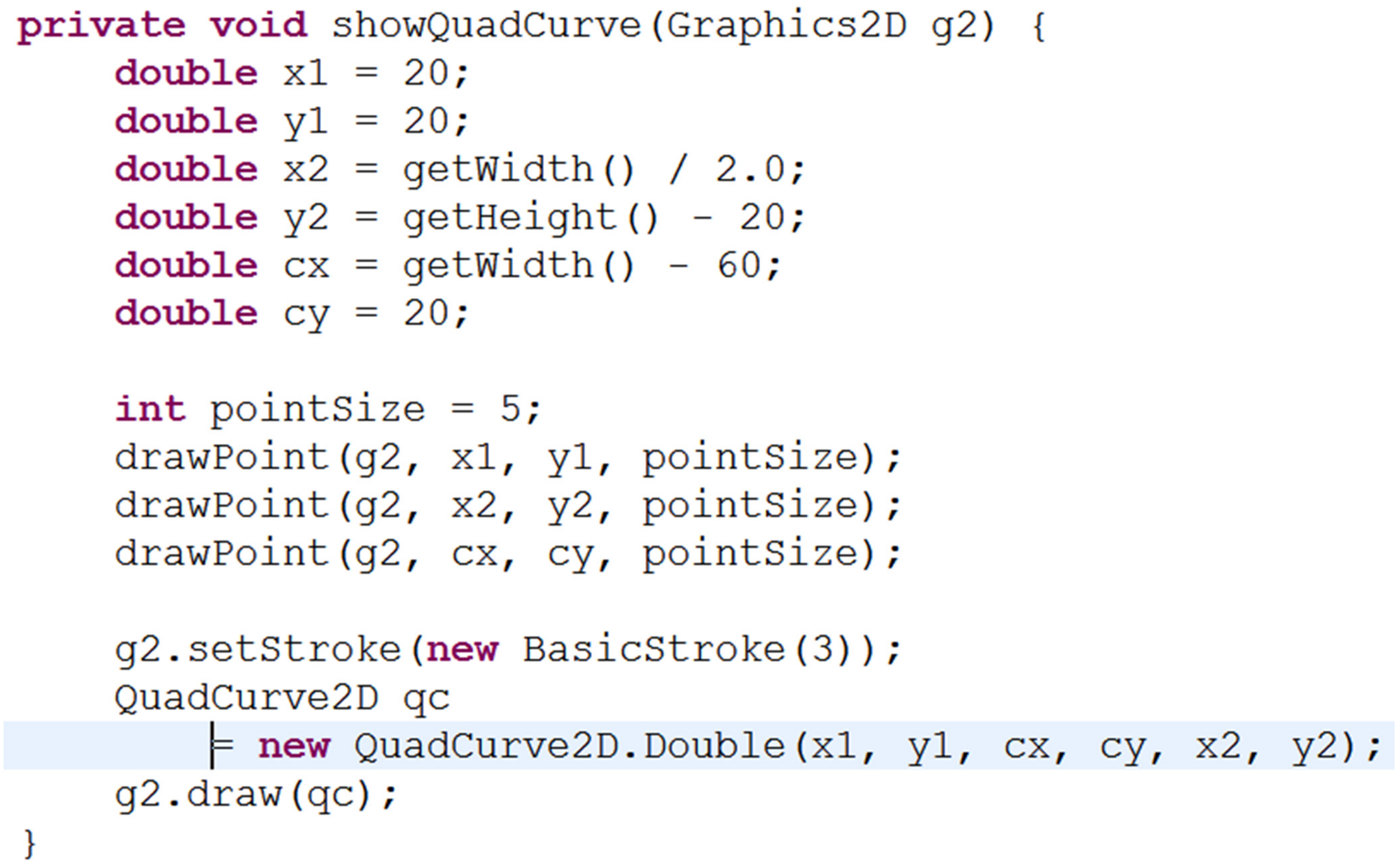
More 2D Graphics "Primitives"

- We have already seen:
  - rectangles, ellipses, arcs, lines
- Today:
  - curves, polygons, areas, paths

Quad Curves

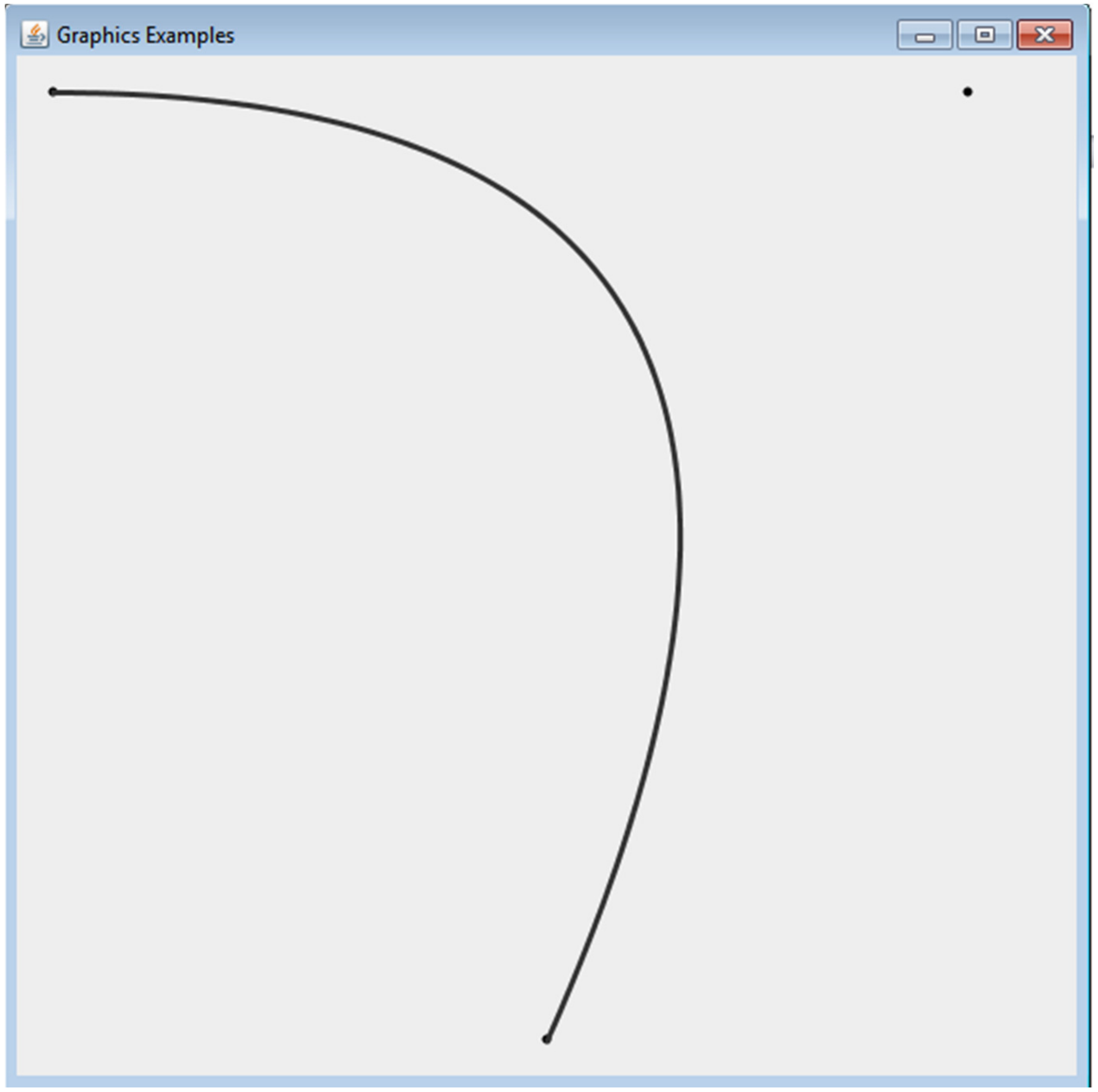
- Quadratic curves
- Defined with 2 end points and a control point
- A type of *Bézier curve*
- A way to model smooth curves
- Given ends points and control points, points on the curve are calculated
  - popularized by Pierre Bézier for designing automobile bodies, based on early work of Paul de Casteljau

Code to Draw QuadCurve

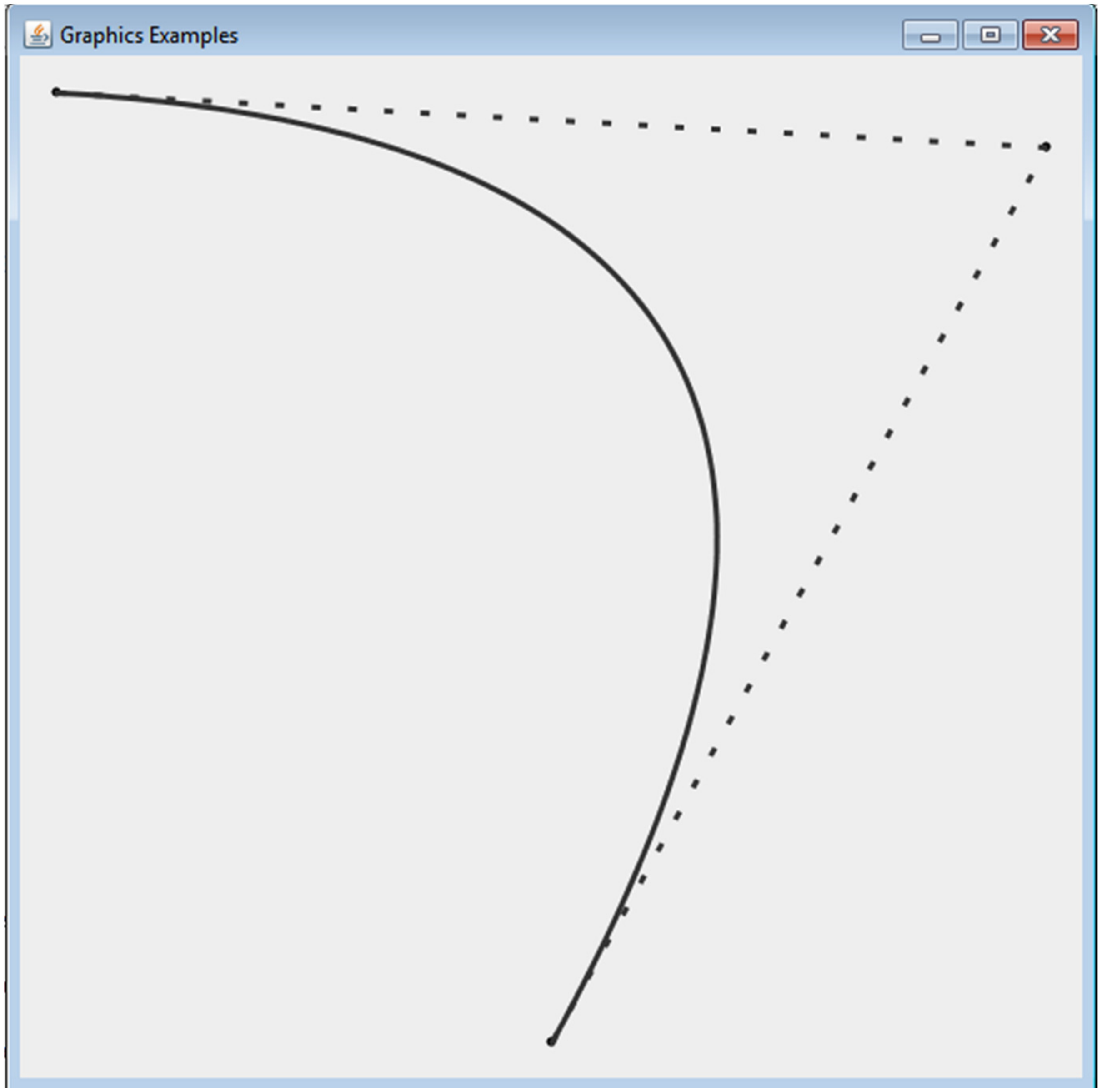


```java
private void showQuadCurve(Graphics2D g2) {
    double x1 = 20;
    double y1 = 20;
    double x2 = getWidth() / 2.0;
    double y2 = getHeight() - 20;
    double cx = getWidth() - 60;
    double cy = 20;

    int pointSize = 5;
    drawPoint(g2, x1, y1, pointSize);
    drawPoint(g2, x2, y2, pointSize);
    drawPoint(g2, cx, cy, pointSize);

    g2.setStroke(new BasicStroke(3));
    QuadCurve2D qc
        = new QuadCurve2D.Double(x1, y1, cx, cy, x2, y2);
    g2.draw(qc);
}
```

Result

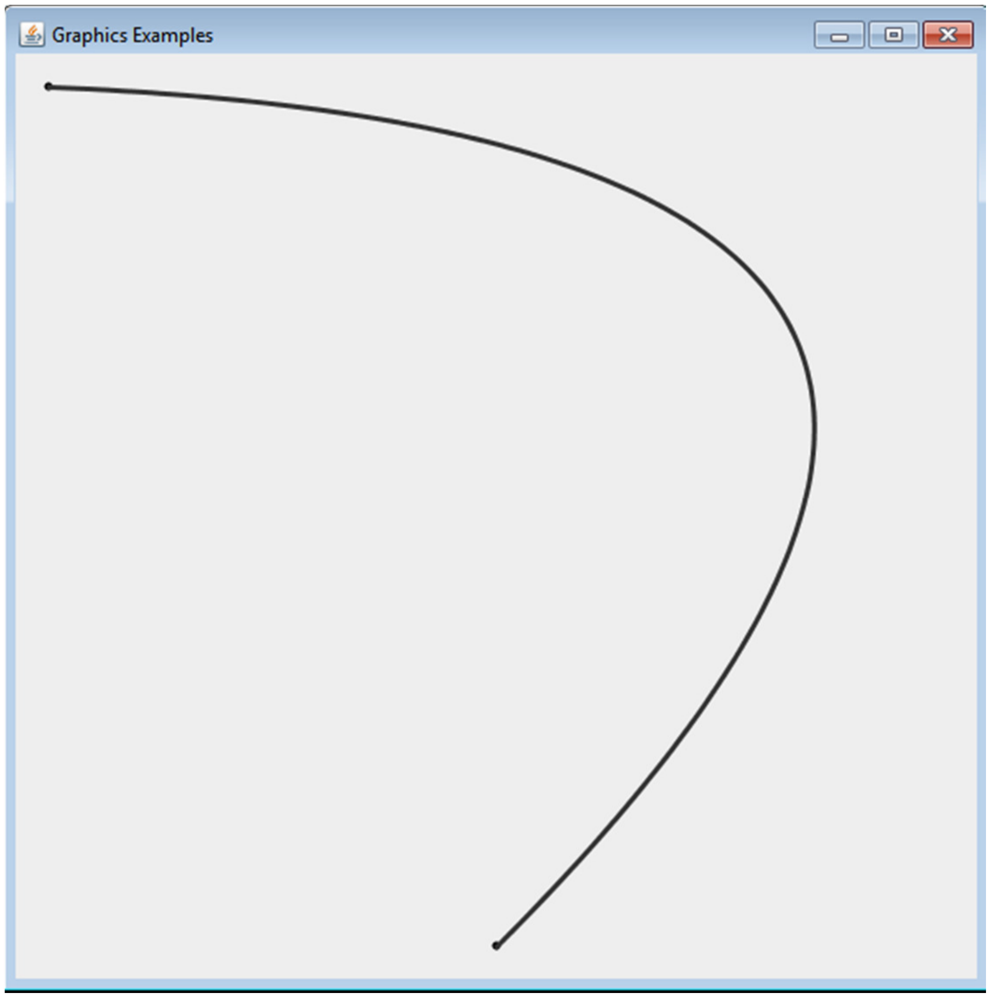


Lines from End Points to Control Point

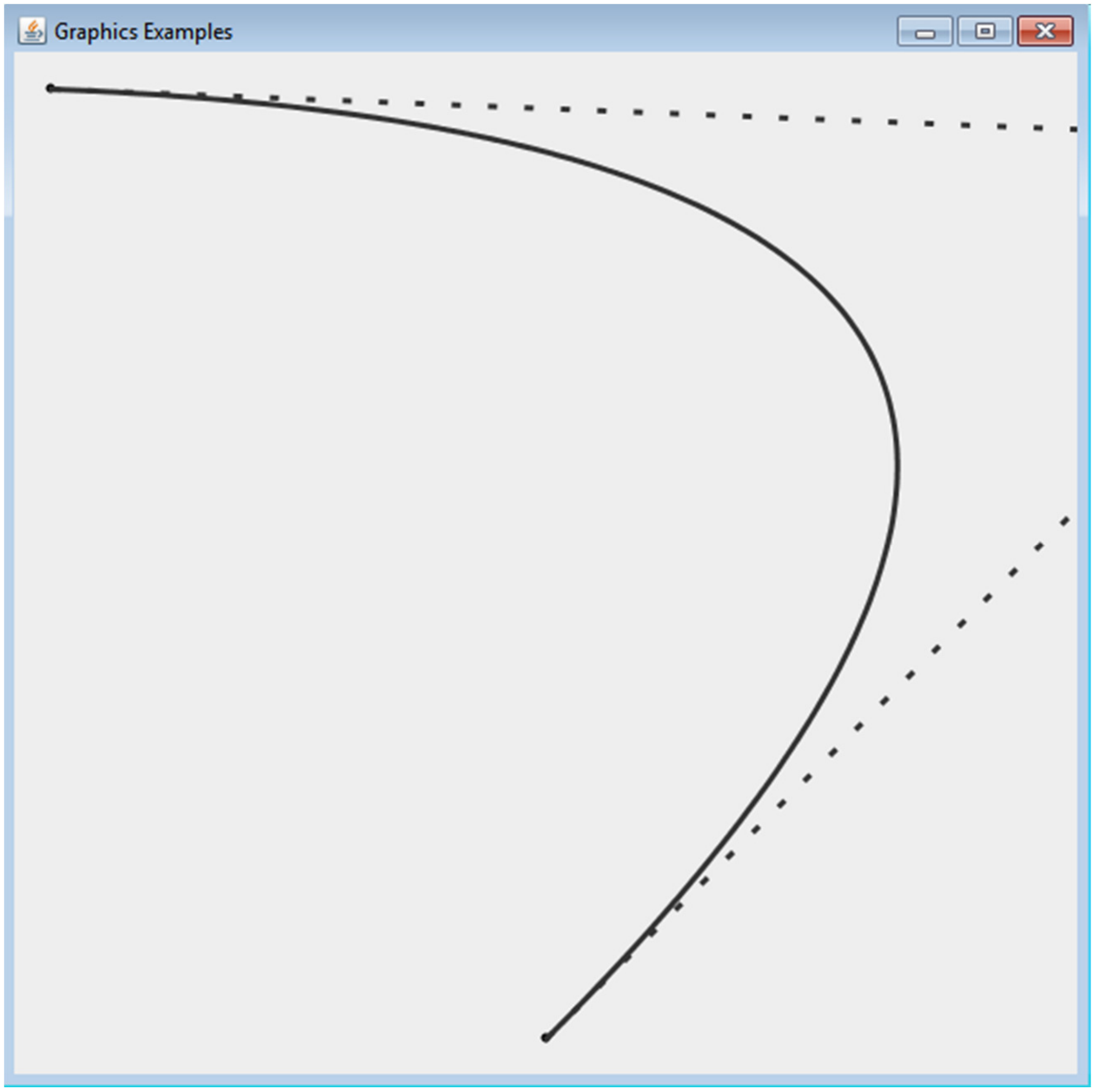


Another QuadCurve

- Control point does not need to be on screen

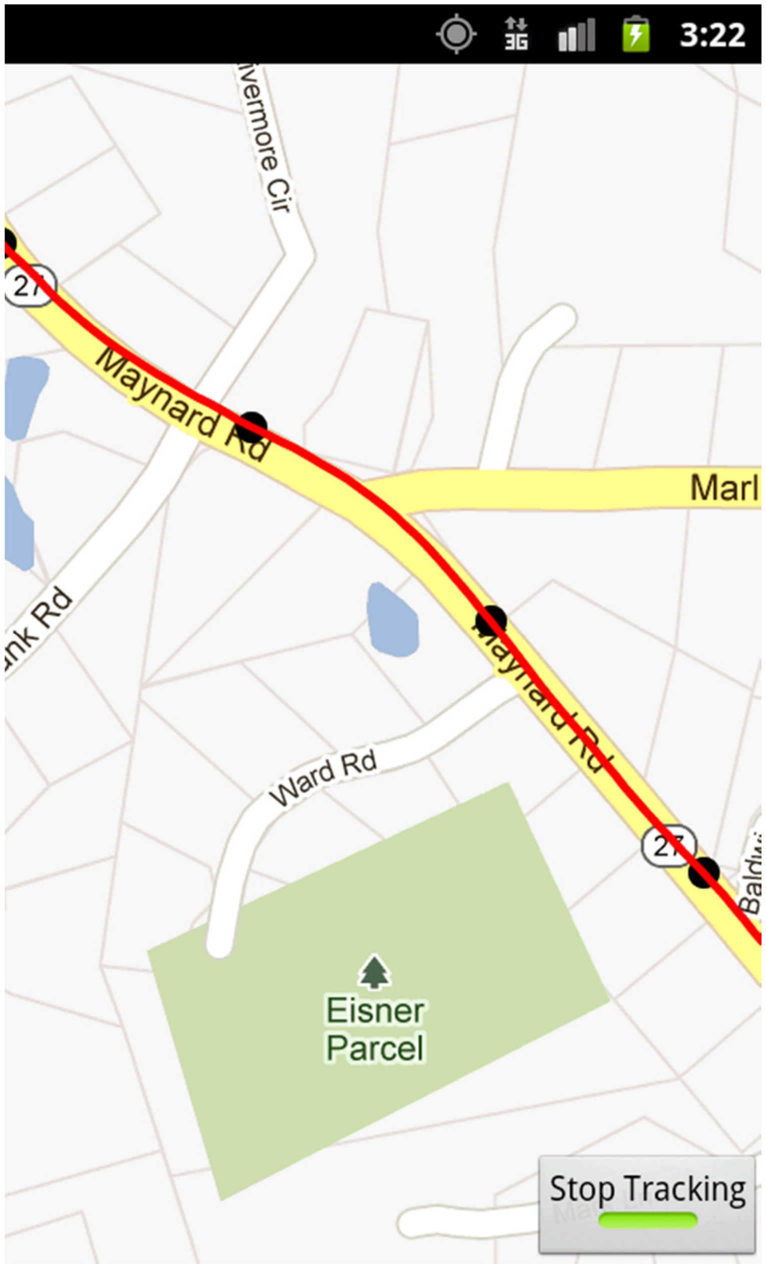


Showing Lines from End Points to Control Point



Use of QuadCurve

- Mapping Application
- Drawing lines (curves) between track points
- Uses QuadCurves to connect points

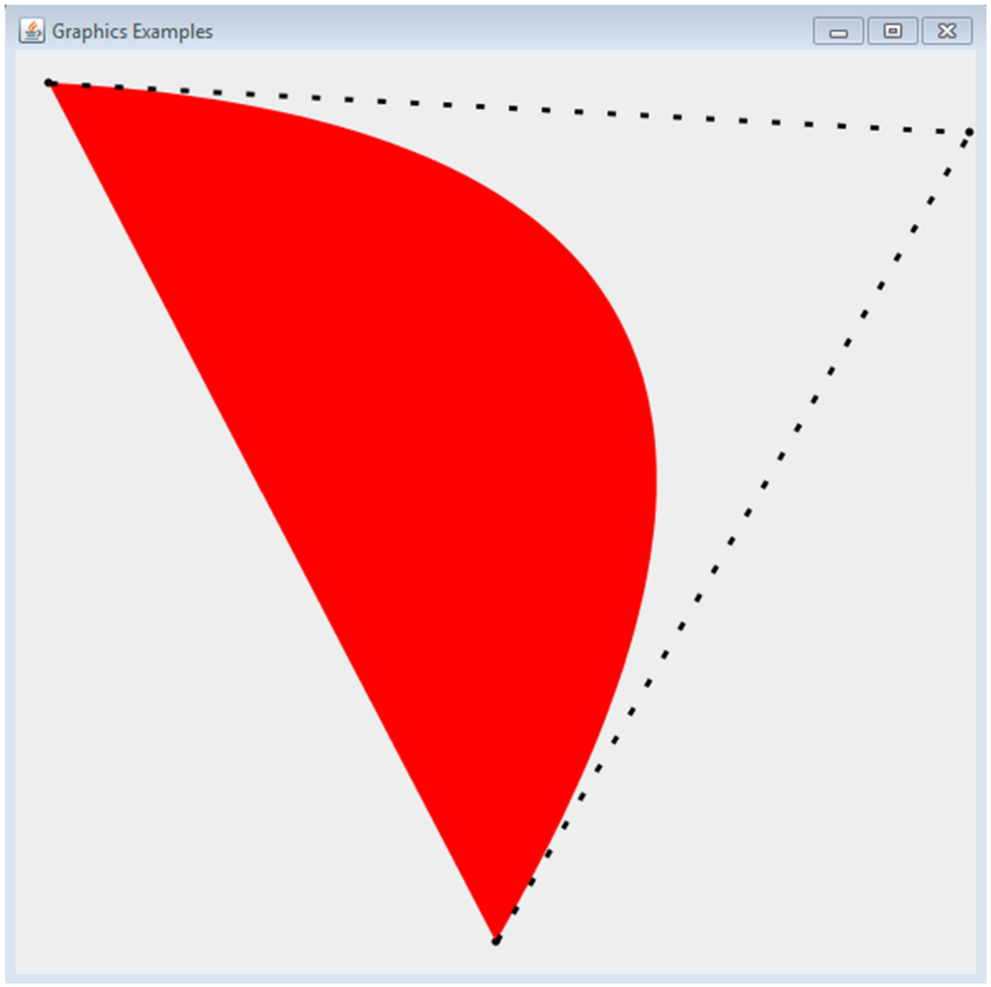


Aside - Responding to MouseEvent

- Alter program so a mouse click changes the control point for the curve
- cx and cy become instance variables
- Create a MouseListener to respond to mouse clicks
- add listener to the panel

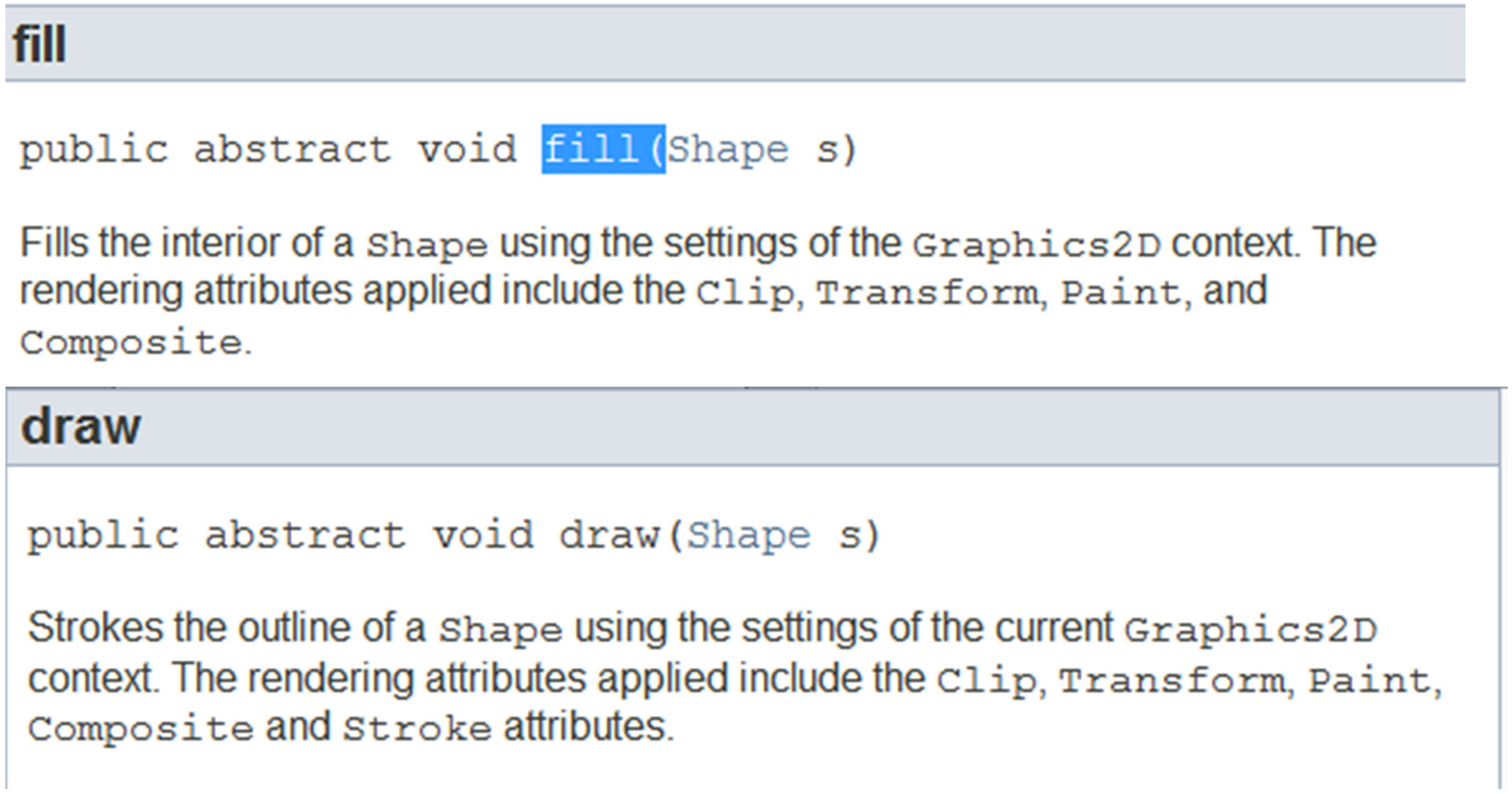
Graphics Fill

- result of g2.fill(quadCurve)



Aside fill and draw

- Methods in the Graphics2D class



fill

```
public abstract void fill(Shape s)
```

Fills the interior of a `Shape` using the settings of the `Graphics2D` context. The rendering attributes applied include the `Clip`, `Transform`, `Paint`, and `Composite`.

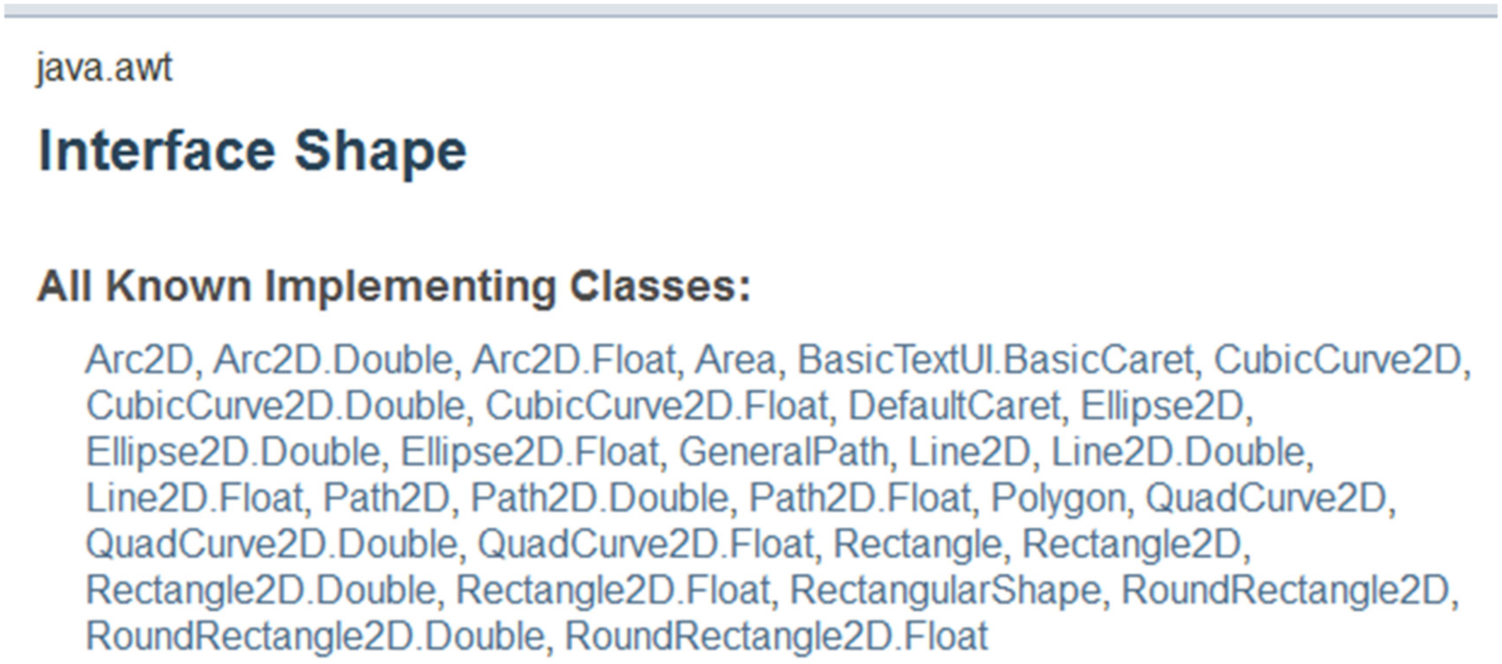
draw

```
public abstract void draw(Shape s)
```

Strokes the outline of a `Shape` using the settings of the current `Graphics2D` context. The rendering attributes applied include the `Clip`, `Transform`, `Paint`, `Composite` and `Stroke` attributes.

Polymorphism

- Shape is an interface in Java
  - the to do list
- Any class that implements the Shape interface can be sent as an argument to draw and fill



java.awt

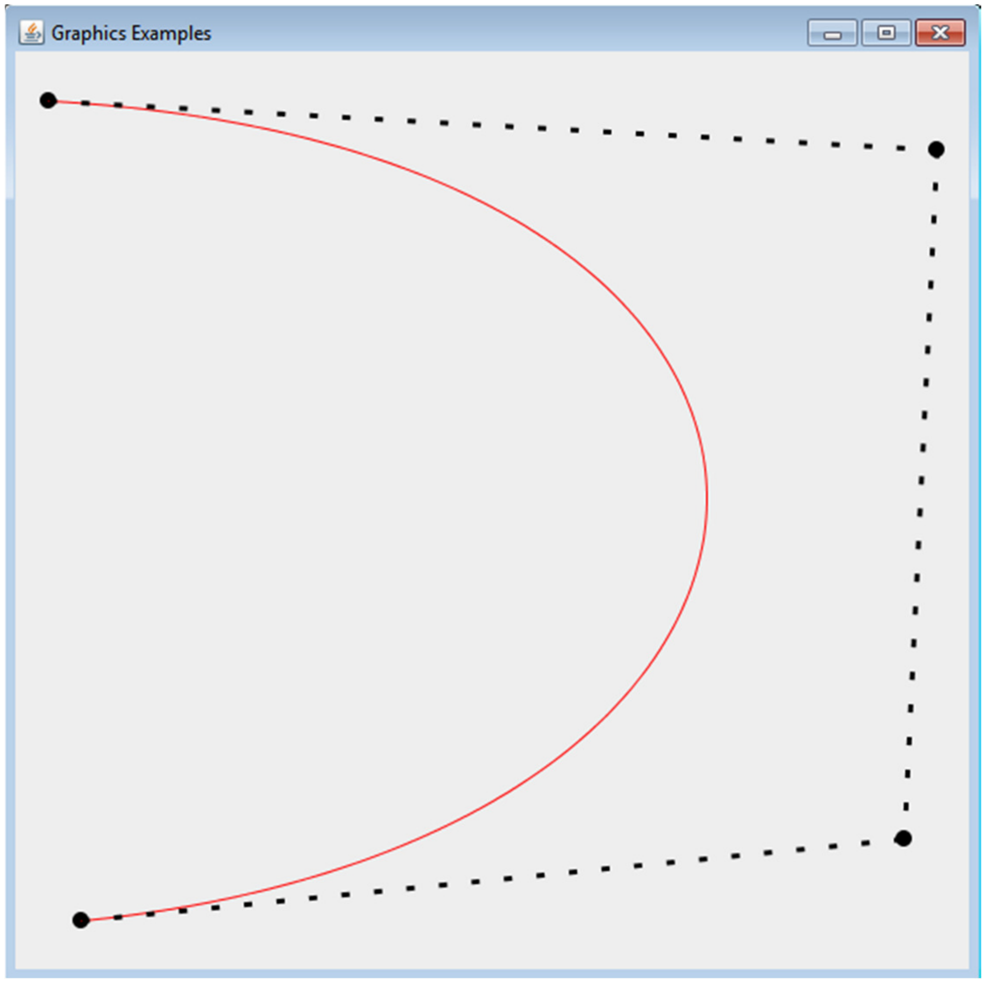
Interface Shape

All Known Implementing Classes:

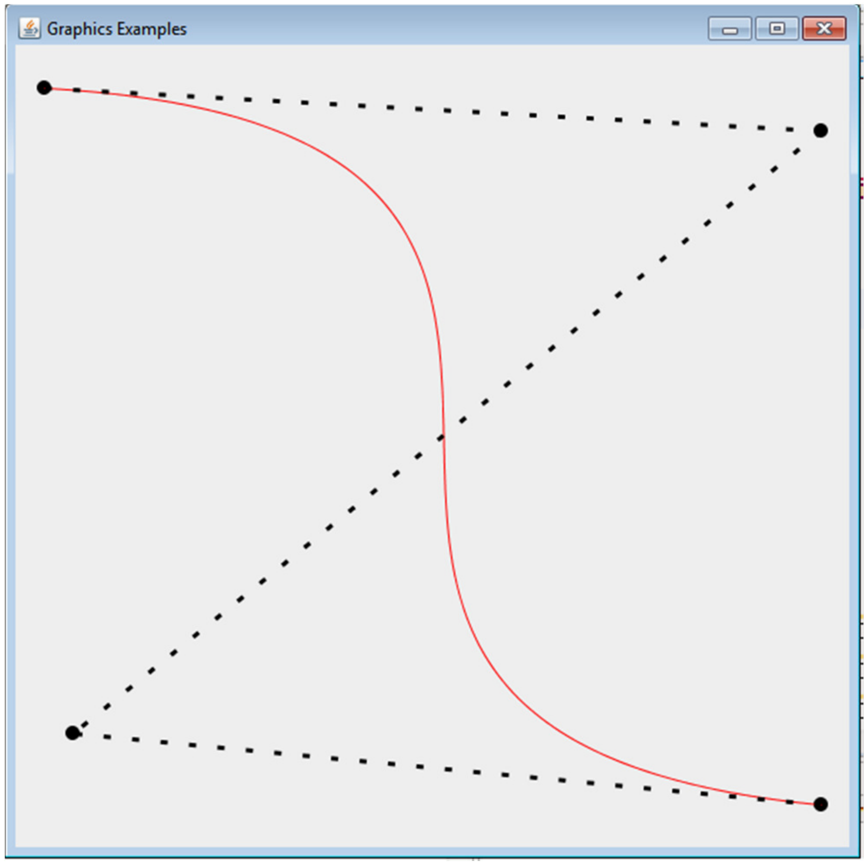
Arc2D, Arc2D.Double, Arc2D.Float, Area, BasicTextUI.BasicCaret, CubicCurve2D, CubicCurve2D.Double, CubicCurve2D.Float, DefaultCaret, Ellipse2D, Ellipse2D.Double, Ellipse2D.Float, GeneralPath, Line2D, Line2D.Double, Line2D.Float, Path2D, Path2D.Double, Path2D.Float, Polygon, QuadCurve2D, QuadCurve2D.Double, QuadCurve2D.Float, Rectangle, Rectangle2D, Rectangle2D.Double, Rectangle2D.Float, RectangularShape, RoundRectangle2D, RoundRectangle2D.Double, RoundRectangle2D.Float

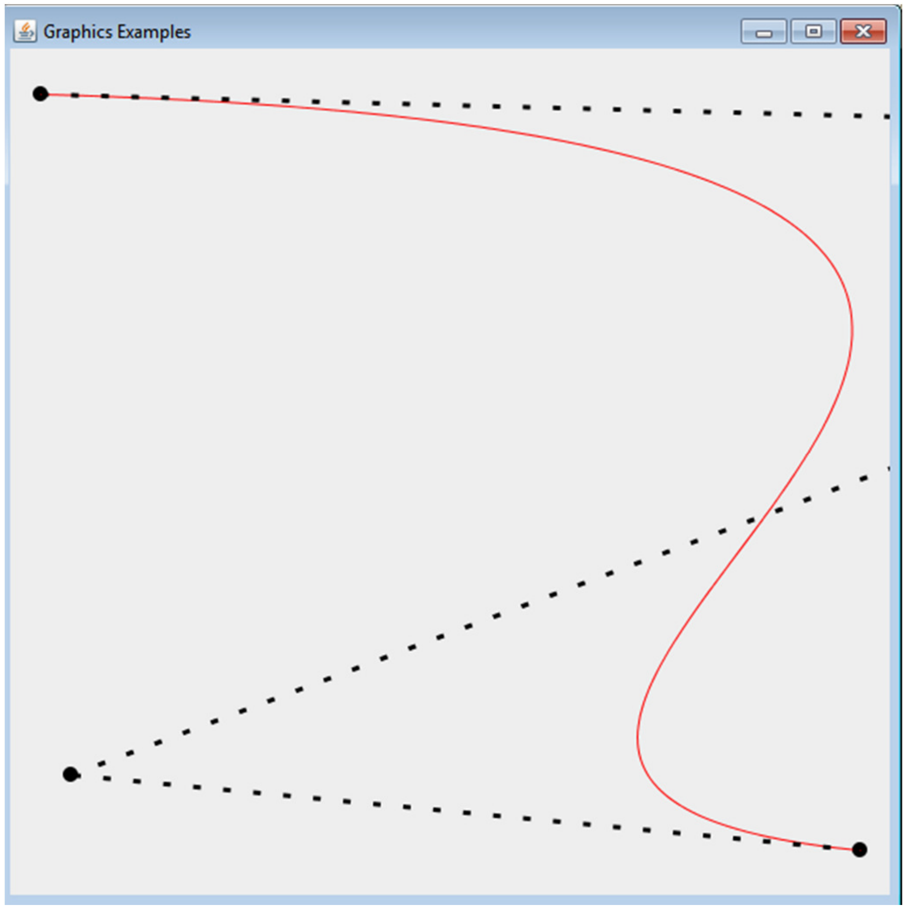
Cubic Curve

- Another *Bézier* curve, but with 2 control points
- draw or fill
- s curve if control points on opposite sides of endpoints



Cubic Curves

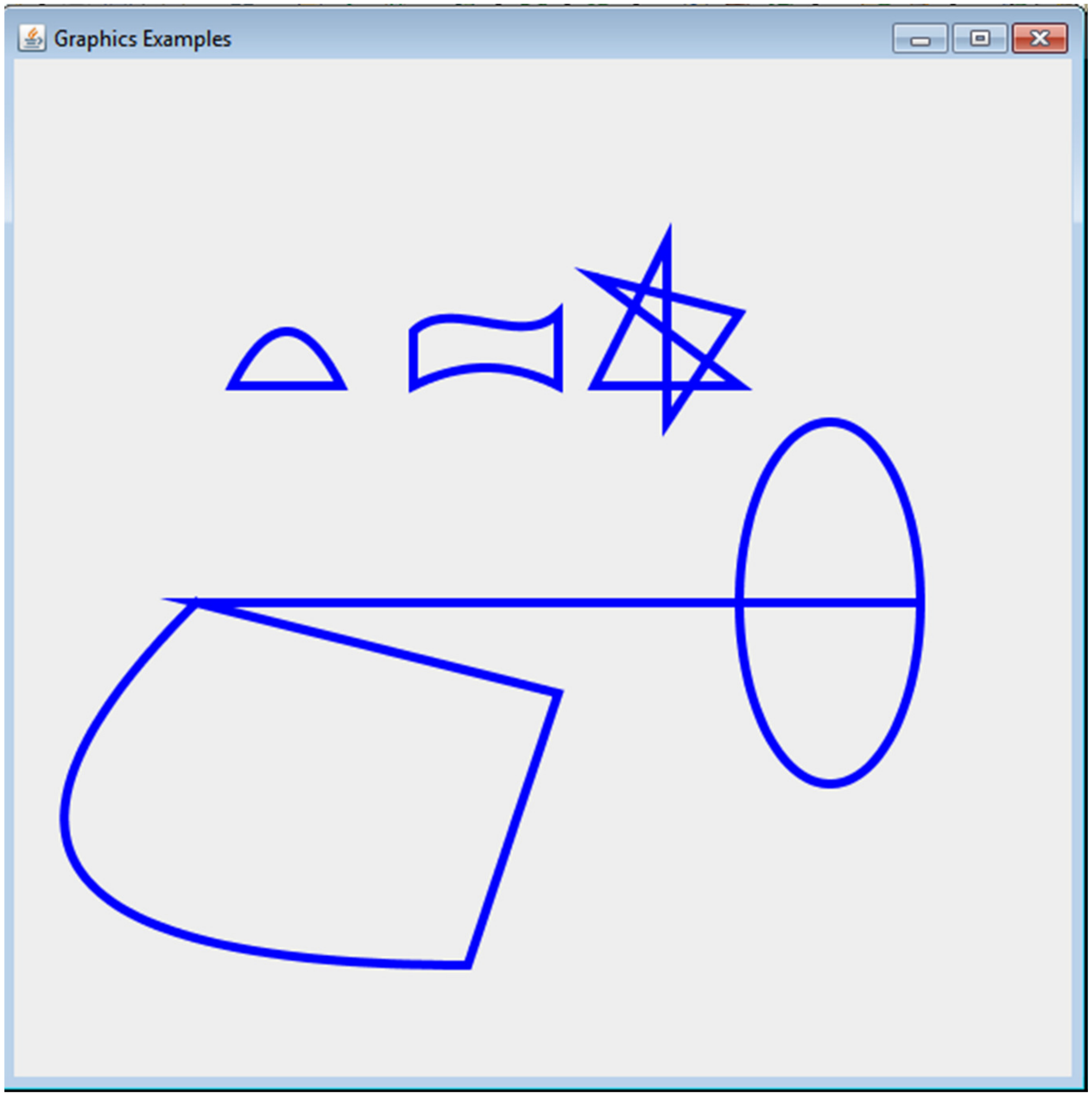




General Path

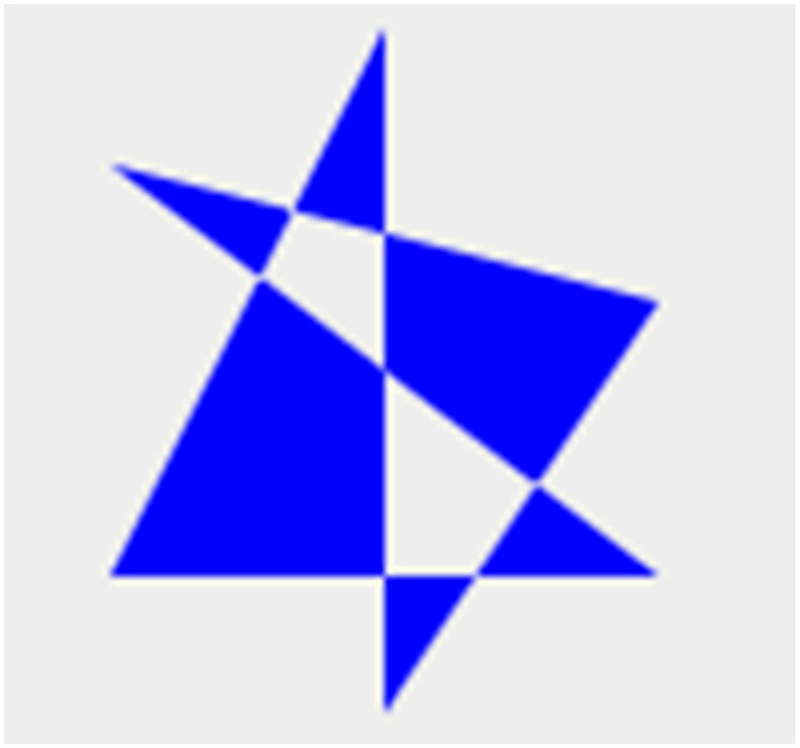
- Combine lines, quad curves, and cubic curves into a general path
- can create with a Shape or empty
- methods to moveTo, lineTo, quadTo, curveTo
  - similar to turtle graphics
- can be drawn or filled

General Paths



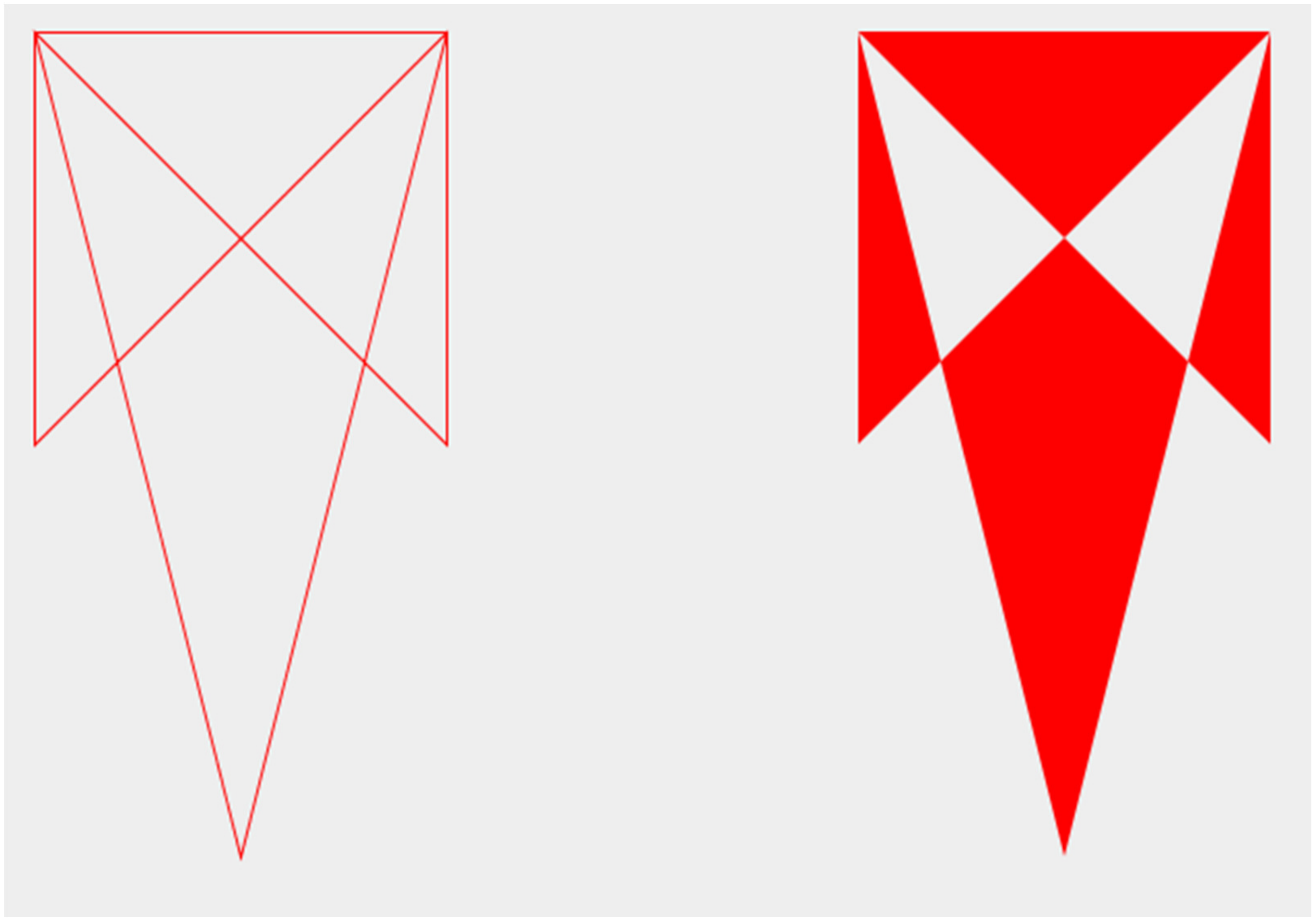
Filling General Paths

- Filling of a general path depends on the *winding rule* set for the path
- Two winding rules:
  - Path2D.WIND_EVEN_ODD
  - Path2D.WIND_NON_ZERO



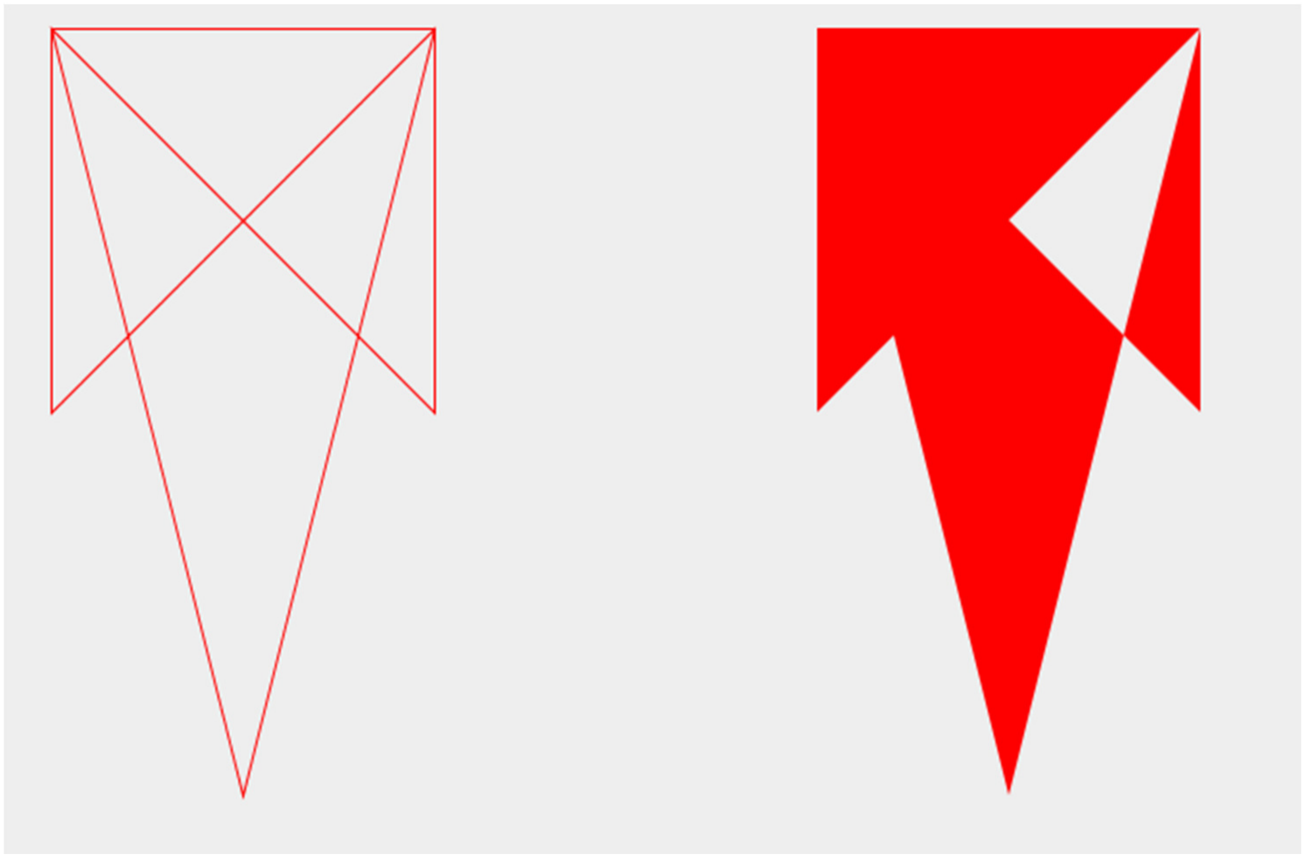
Sample Path

- Path2D.WIND_EVEN_ODD



Sample Path

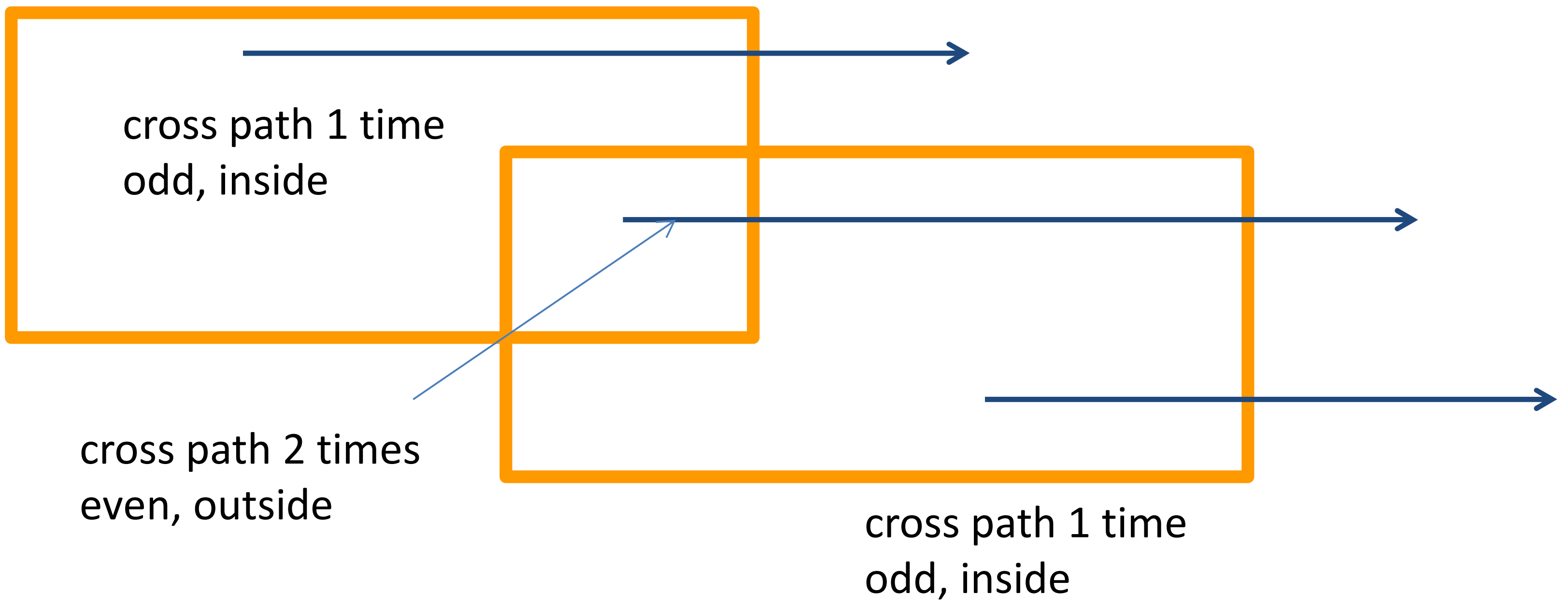
- Path2D.WIND_NON_ZERO
- (Must know direction path drawn)



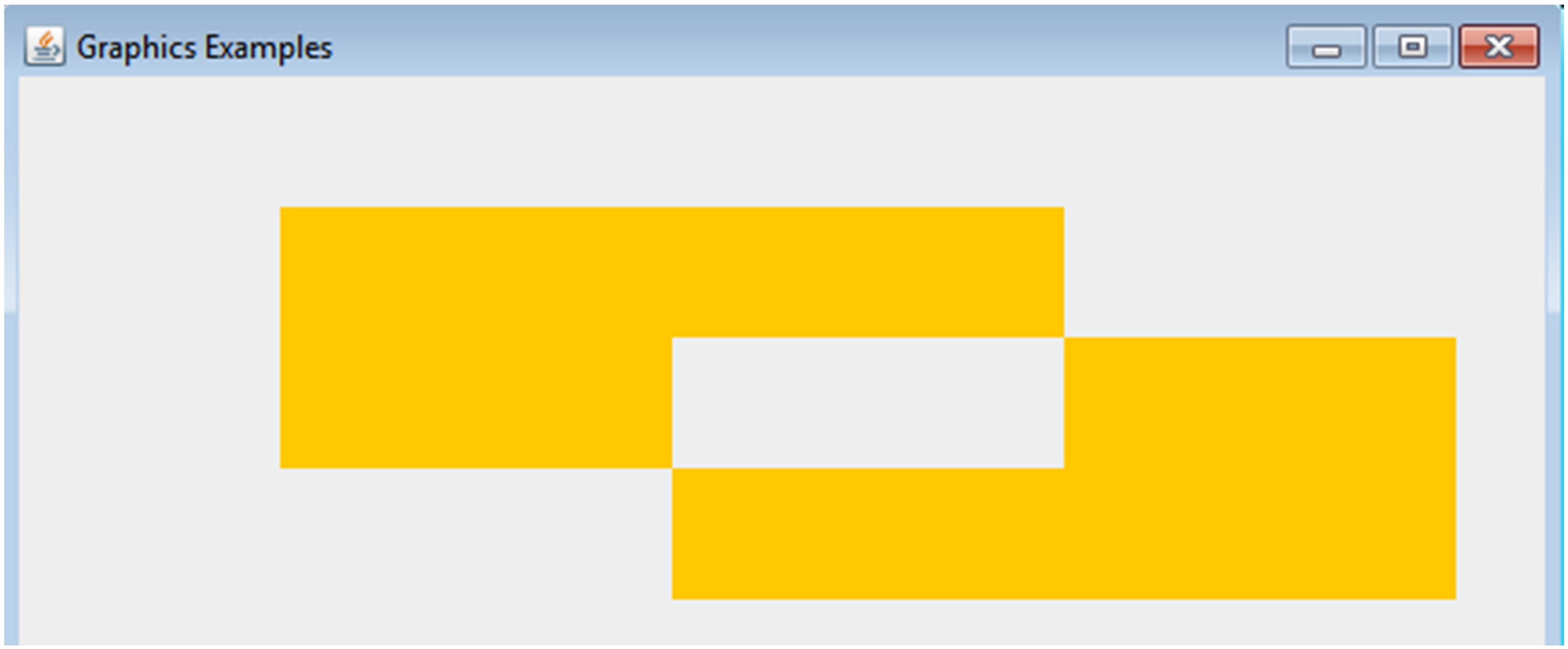
WIND_EVEN_ODD

- To determine if region is inside or outside the path draw a line from inside the region to outside the path (infinity)
- If the number of crossings is odd then the region is inside the path.
- If the number of crossings is even then the region is outside the path.

Even Odd Example



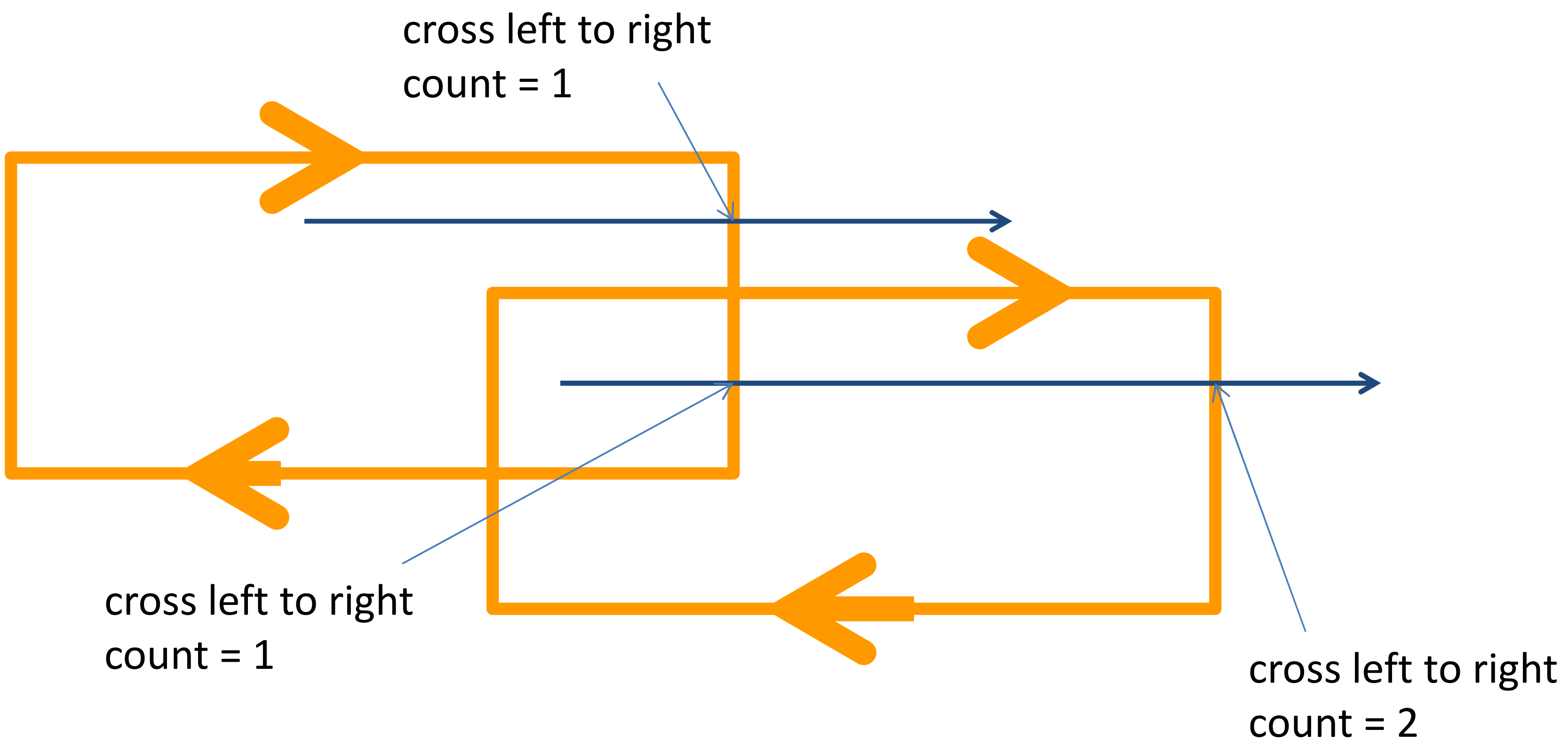
Even Odd Result



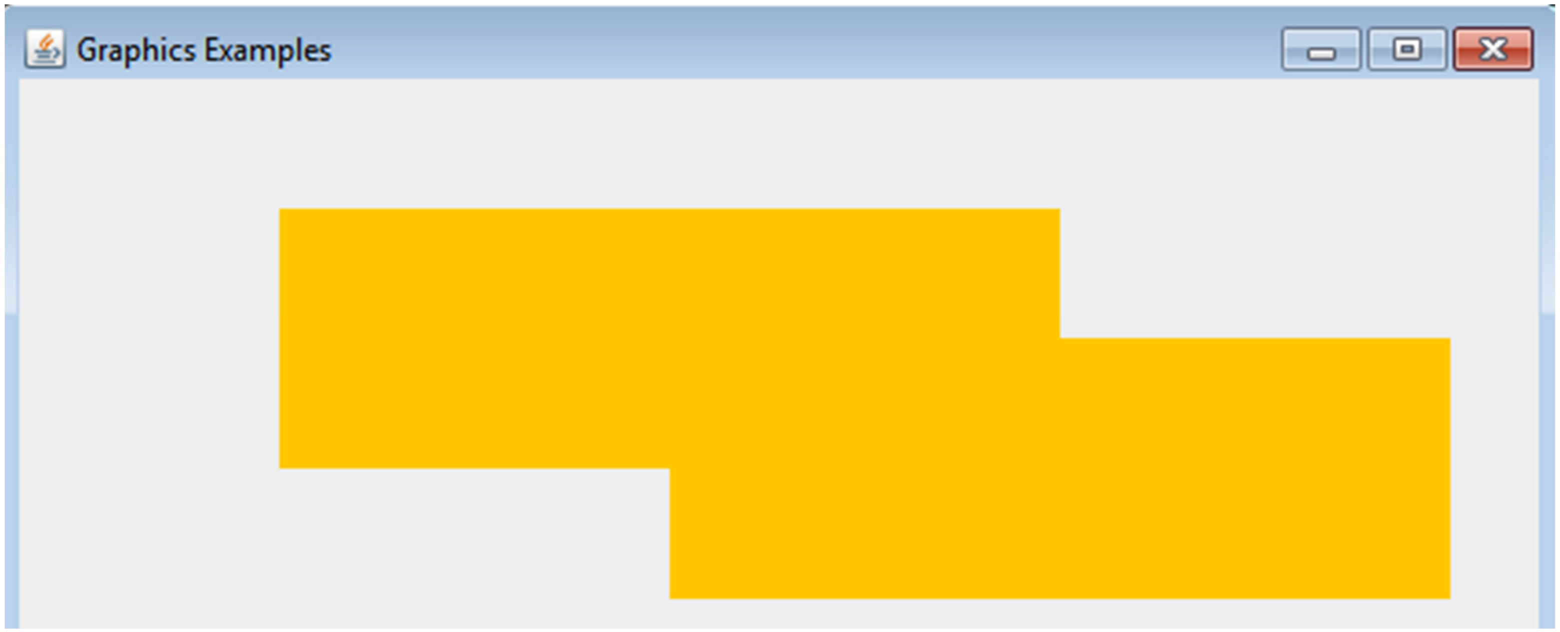
Non Zero Rule

- The direction of the path crossed is considered
- Draw line from region to infinity
- Initialize counter to 0
- Every time path crossed "left to right" add 1
- Every time path crossed "right to left" subtract 1
- Interior regions have a total not equal to 0

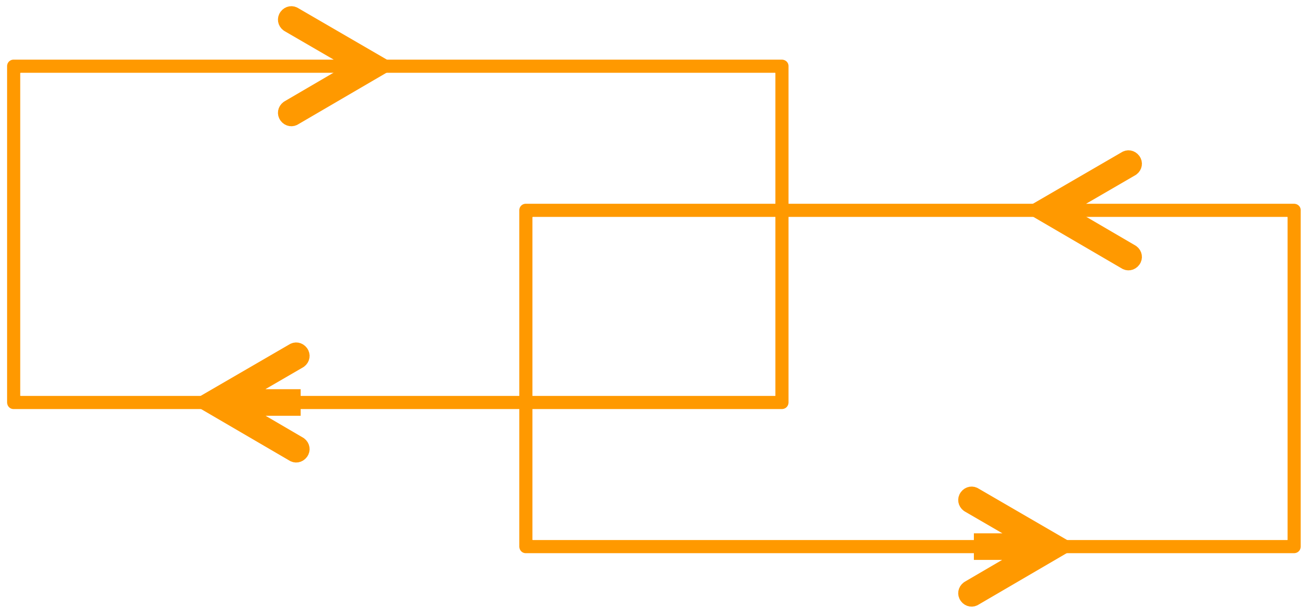
Non Zero Example



Non Zero Result

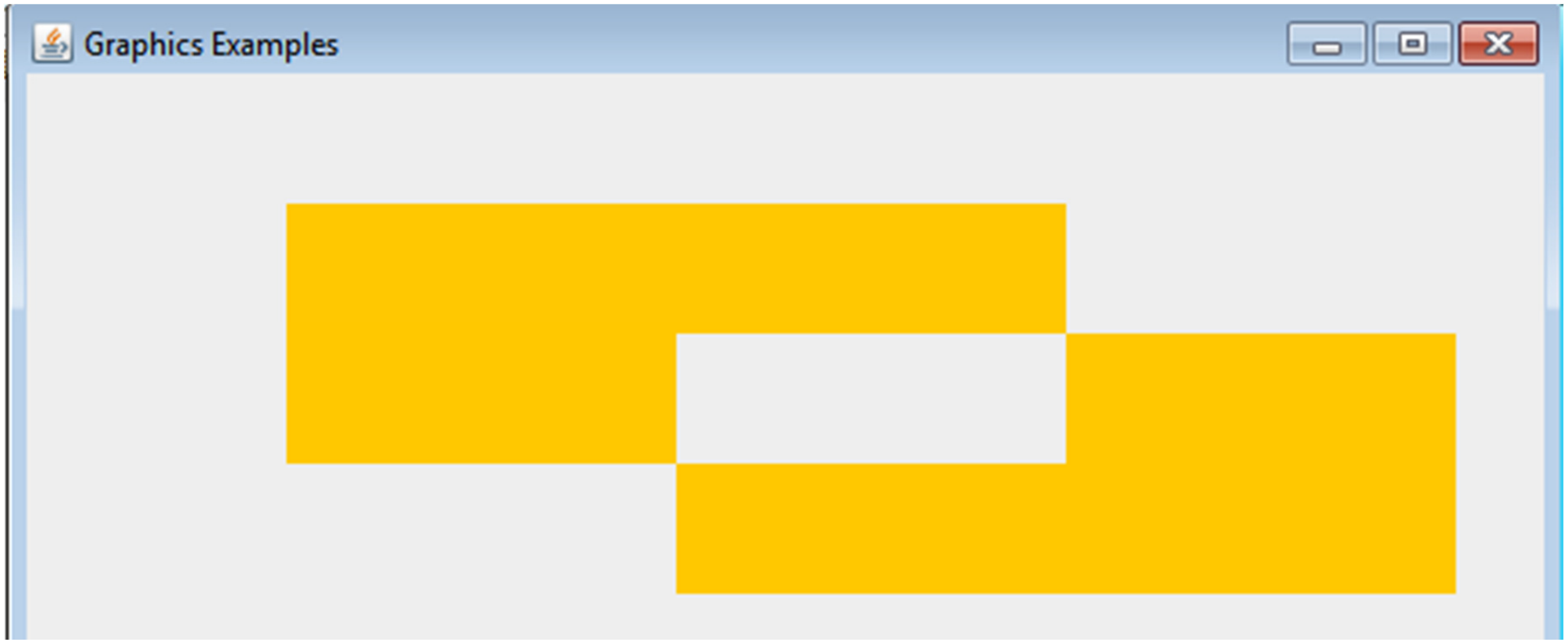


Change Direction of One Path



Result?

Result



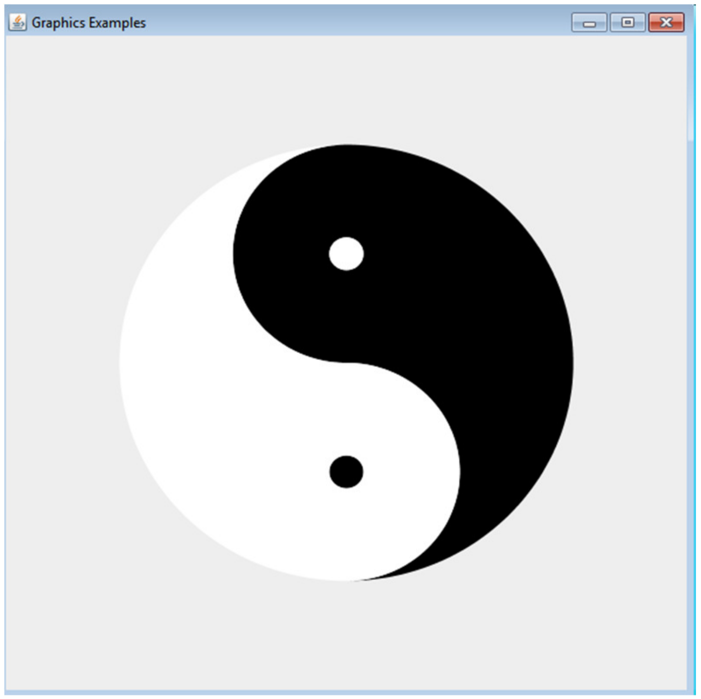
- Default of GeneralPath is NON_ZERO
- Does direction of path affect interior regions for EVEN_ODD ruler?

Areas

- Areas are to General Paths as Rectangles and Ellipses, are to Lines and Curves
- Build an area out of multiple shapes
- *Constructive Area Geometry* - CAG
- Alter area by
  - add (union)
  - subtract
  - intersection
  - exclusive or (union minus intersection)

Sample CAG

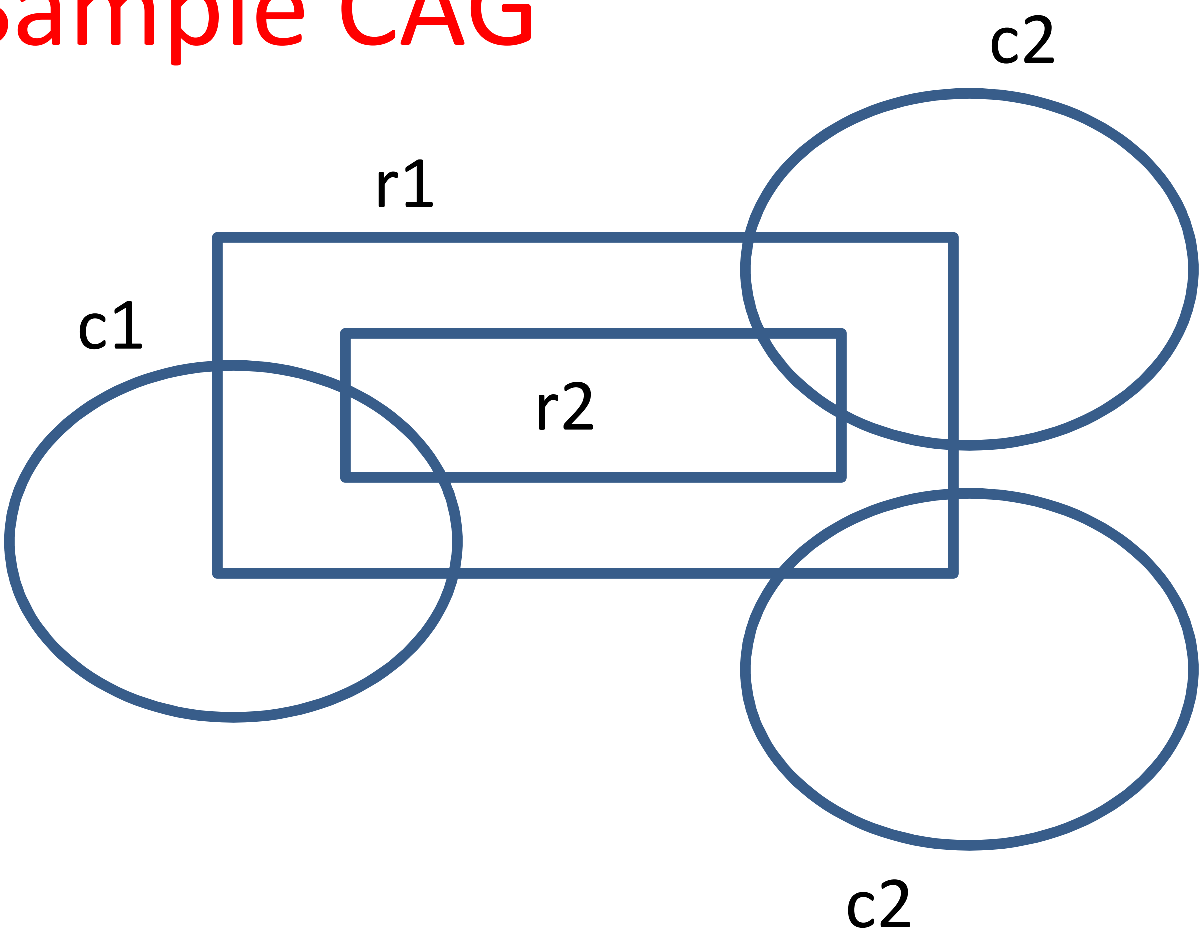
```java
Area ying = createHalf(new Shape[] {leftArc, bottomCircle,
        bottomSmallCircle, topCircle, topSmallCircle});
Area yang = createHalf(new Shape[] {rightArc, topCircle,
        topSmallCircle, bottomCircle, bottomSmallCircle});
```



```java
private Area createHalf(Shape[] parts){
    Area result = new Area(parts[0]);
    result.add(new Area(parts[1]));
    result.subtract(new Area(parts[2]));
    result.subtract(new Area(parts[3]));
    result.add(new Area(parts[4]));
    return result;
}
```

Sample CAG

```
Area a1 = new Area(r1);
Area a2 = new Area(r2);
Area a3 = new Area(c1);
Area a4 = new Area(c2);
Area a5 = new Area(c3);
a1.subtract(a2);
a1.add(a3);
a1.exclusiveOr(a4);
a1.subtract(a5);

// result??
```


c2
r1
c1
r2
c2