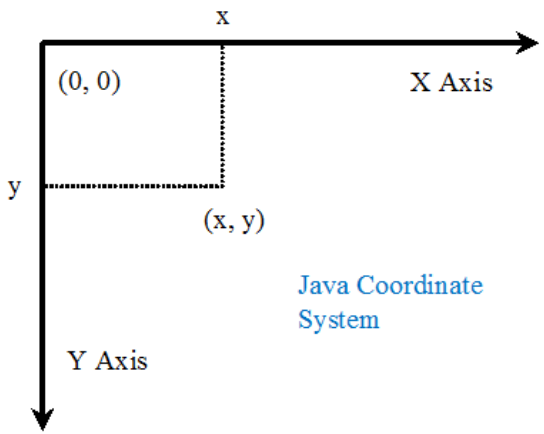
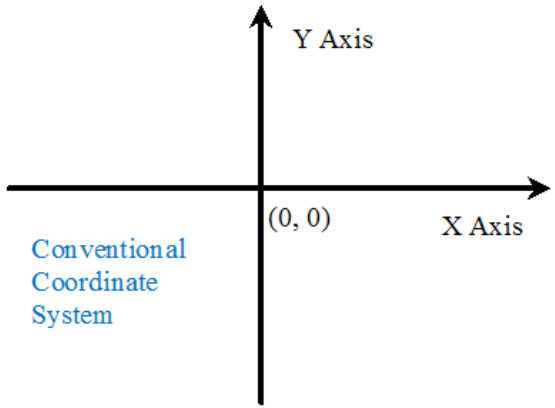
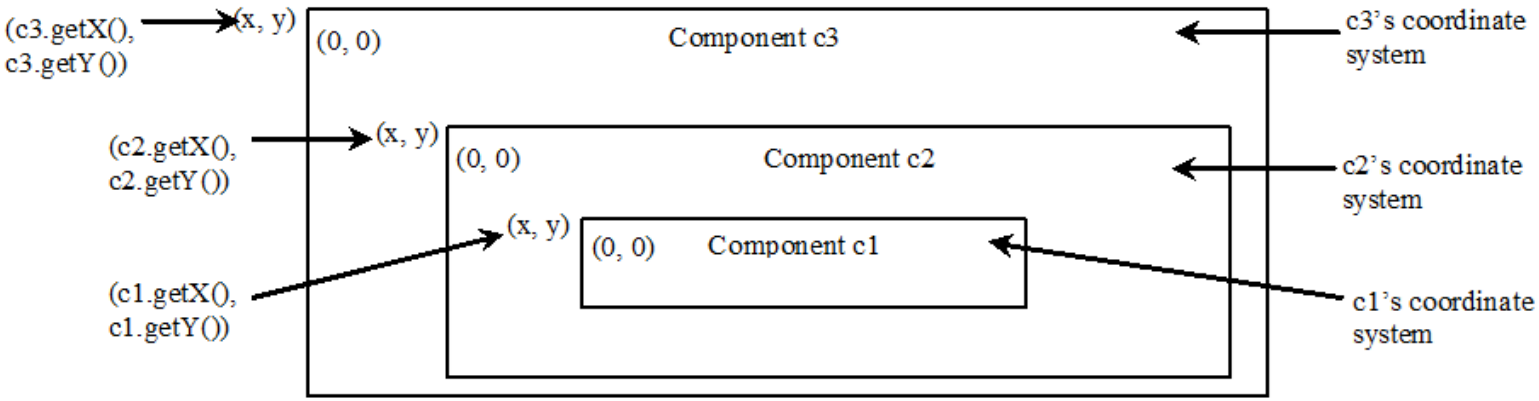
Graphics Programming

Graphic Coordinate Systems







Graphics Classes

- The graphics context of a component provides a device-dependent graphics settings for painting/drawing strings, shapes and images
  - **`Graphics`** class since 1.0
  - **`Graphics2D`** class extending **`Graphics`** class since 1.2
- Whenever a component is displayed, the JVM automatically creates a graphics object for the component on the native platform
  - Think of a component as a piece of paper and the graphics object as a pencil or paintbrush
- All painting/drawing in Java must go through a graphics object

Example: Test Draw Using getGraphics()

- A graphics object can be obtained using the **`getGraphics`**() method
  - You can apply methods in the graphics class to draw on the component's graphics context
  - You do not normally draw graphics this way !

```
public static void main(String[] args) {
  EventQueue.invokeLater(new Runnable() {
    public void run() {

      DrawFrame frame = new DrawFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);

      JOptionPane.showMessageDialog(null, "Delay is necessary");

      frame.drawAll();
    }
  });
}
```

?

Example: Test Draw Using getGraphics()

```
class DrawComponent extends JComponent {

  void drawAll() {

    Graphics2D g2 = (Graphics2D) getGraphics();

    // draw a rectangle
    double leftX = 100;
    double topY = 100;
    double width = 200;
    double height = 150;
    Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
    g2.draw(rect);
    // draw the enclosed ellipse
    Ellipse2D ellipse = new Ellipse2D.Double();
    ellipse.setFrame(rect);
    g2.draw(ellipse);
    // draw a diagonal line
    g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
    // draw a circle with the same center
    double centerX = rect.getCenterX();
    double centerY = rect.getCenterY();
    double radius = 150;
    Ellipse2D circle = new Ellipse2D.Double();
    circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
    g2.draw(circle);
  }
}
```

`paintComponent` Method

- **`paintComponent()`** method of a Swing component is **automatically** invoked by JVM when **necessary**:
  - A frame is displayed
  - A frame is resized
  - A component or a part of a component is displayed or redisplayed
- To permanently display anything (strings / shapes / images), you need to draw them in the **`paintComponent`** method

```
protected void paintComponent(Graphics g)
```

- Never call the **`paintComponent`**() method directly
  - Call **`repaint`**() method if you need to force repainting

Example: Test Draw Using paintComponent()

```
public static void main(String[] args) {
  EventQueue.invokeLater(new Runnable() {
    public void run() {

      DrawFrame frame = new DrawFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);

    }
  });
}

class DrawFrame extends Jframe {

   public static final int DEFAULT_WIDTH = 400;
   public static final int DEFAULT_HEIGHT = 400;

   public DrawFrame() {
      setTitle("DrawTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      DrawComponent component = new DrawComponent();
      add(component);

   }
}
```

Example: Test Draw Using paintComponent()

```
class DrawComponent extends JComponent {

    public void paintComponent(Graphics g) {

    Graphics2D g2 = (Graphics2D) g;

    // draw a rectangle
    double leftX = 100;
    double topY = 100;
    double width = 200;
    double height = 150;
    Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
    g2.draw(rect);
    // draw the enclosed ellipse
    Ellipse2D ellipse = new Ellipse2D.Double();
    ellipse.setFrame(rect);
    g2.draw(ellipse);
    // draw a diagonal line
    g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
    // draw a circle with the same center
    double centerX = rect.getCenterX();
    double centerY = rect.getCenterY();
    double radius = 150;
    Ellipse2D circle = new Ellipse2D.Double();
    circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
    g2.draw(circle);
  }
}
```

Make a Component to Draw On

- Define a class that extends **`JComponent`** and override the **`paintComponent`**() method
  - Some people draw on a JPanel
    - A JPanel is intended to be a container for other components
    - Do you really need a layout manager provided by the JPanel for your drawing?

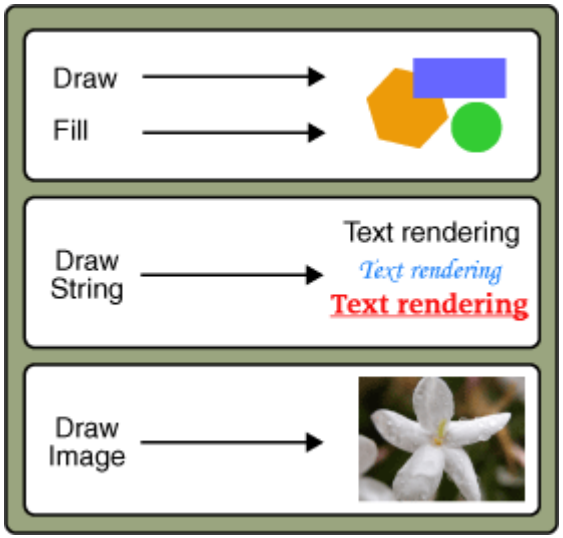
```
Class MyComponent extends JComponent {

    paintComponent(Graphics g) {

        Graphics2D g2 = (Graphics2D) g;

        // code for drawing
    }
}
```

Drawing

- Draw and fill methods to render text, shapes and images
- Attributes setting methods affect how that drawing and filling appears
- For drawing and filling geometric shapes:

```
drawLine(), drawArc(), fillArc(),
drawRect(), fillRect(), drawOval(),
fillOval(), drawPolygon(), fillPolygon()
```

e.g. `g2.draw(new Line2D.Double(0, 0, 30, 40));`



- For drawing text: `drawString()`

e.g. `g.drawString("Hello", 10, 10);`

- For drawing images: `drawImage()`

e.g.
```
g.drawImage(img, 0, 0, width, height,
            0, 0, imageWidth, imageHeight,
            null);
```

`Graphics2D` Class

- Access to the enhanced graphics and rendering features of the Java 2D API
  - Rendering the outline of any geometric primitive, using the stroke and paint attributes (`draw()` method).
  - Rendering any geometric primitive by filling its interior with the color or pattern specified by the paint attributes (`fill()` method).
  - Rendering any text string (`drawString()` method). The font attribute is used to convert the string to glyphs, which are then filled with the color or pattern specified by the paint attributes.
  - Rendering the specified image (the `drawImage()` method).

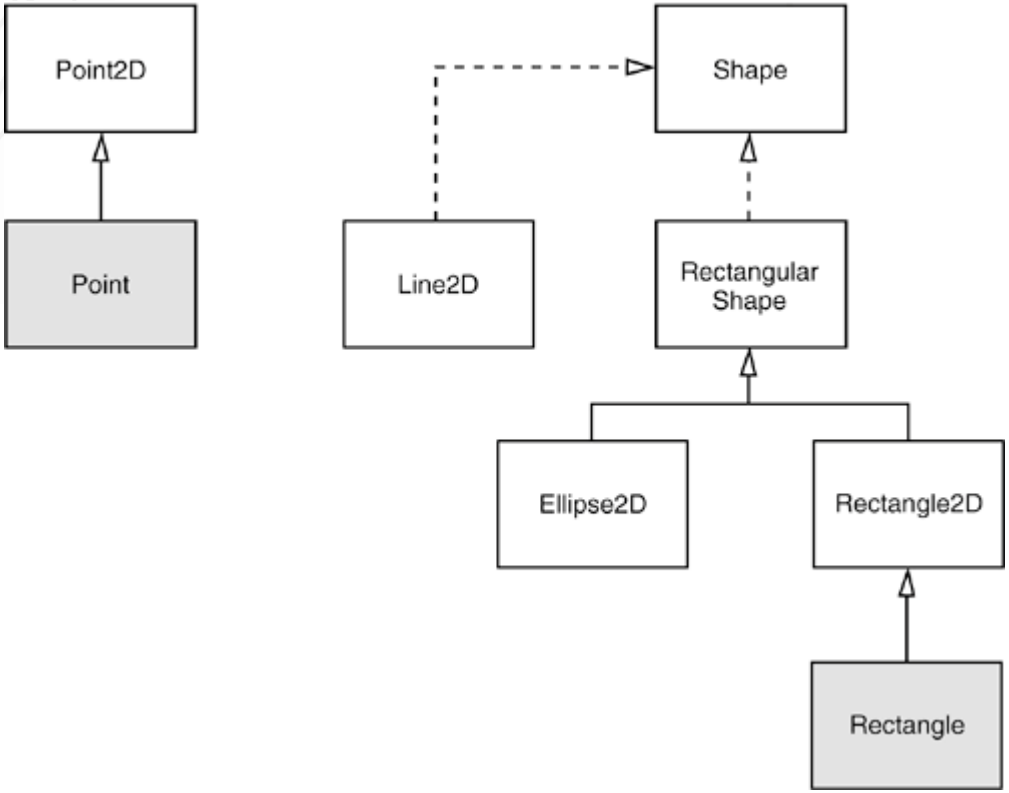
Graphics2D Class

- A uniform rendering model for display devices and printers
- A wide range of geometric primitives
  - such as curves, rectangles, and ellipses, as well as a mechanism for rendering virtually any geometric shape
- Mechanisms for performing hit detection on shapes, text, and images
- A compositing model that provides control over how overlapping objects are rendered

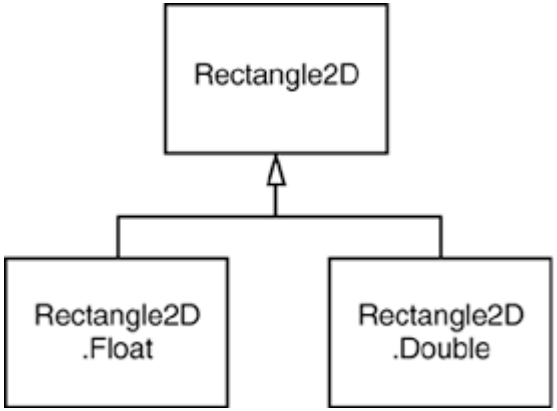
- Enhanced color support that facilitates color management
- Control of the quality of the rendering through the use of rendering hints

We do not discuss all of them in the class

2D Shapes

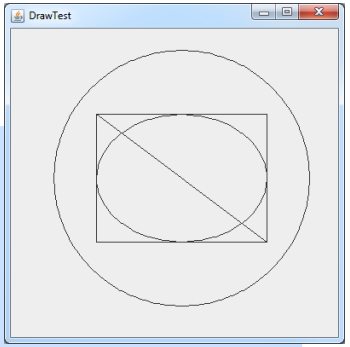

Point2D
Shape
Point
Line2D
Rectangular Shape
Ellipse2D
Rectangle2D
Rectangle

2D Rectangle Classes



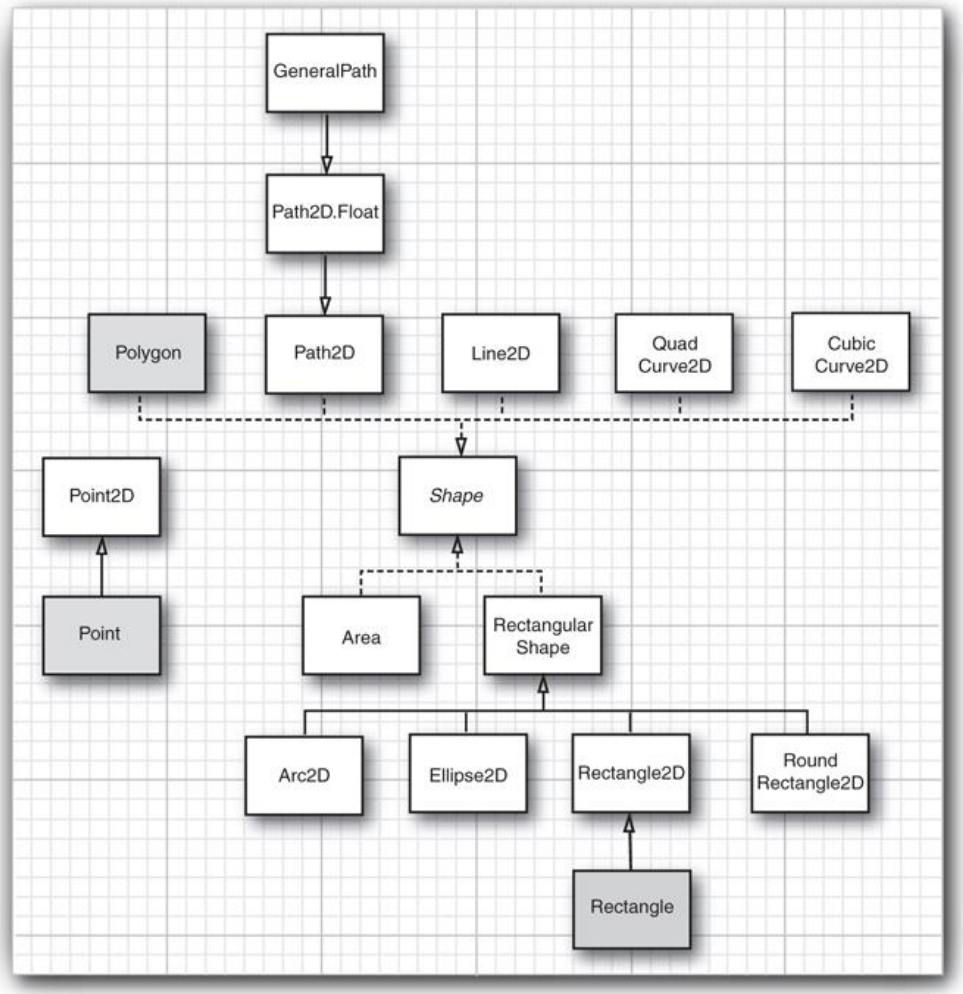

```
Rectangle2D floatRect = new Rectangle2D.Float(10.0F, 25.0F, 22.5F, 20.0F);

Rectangle2D doubleRect = new Rectangle2D.Double(10.0, 25.0, 22.5, 20.0);
```

Example: Test Draw Revisited




```
class DrawComponent extends JComponent {
   public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
     // draw a rectangle
     double leftX = 100;
     double topY = 100;
     double width = 200;
     double height = 150;
     Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
     g2.draw(rect);
     // draw the enclosed ellipse
     Ellipse2D ellipse = new Ellipse2D.Double();
     ellipse.setFrame(rect);
     g2.draw(ellipse);
     // draw a diagonal line
     g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
     // draw a circle with the same center
     double centerX = rect.getCenterX();
     double centerY = rect.getCenterY();
     double radius = 150;
     Ellipse2D circle = new Ellipse2D.Double();
     circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
     g2.draw(circle);
  }
}
```

More Shapes



Quadratic Curve

Cubic Curve

General Path

Arc

Using Colors

- **`setPaint`**() method of the Graphics2D class allows selecting a color

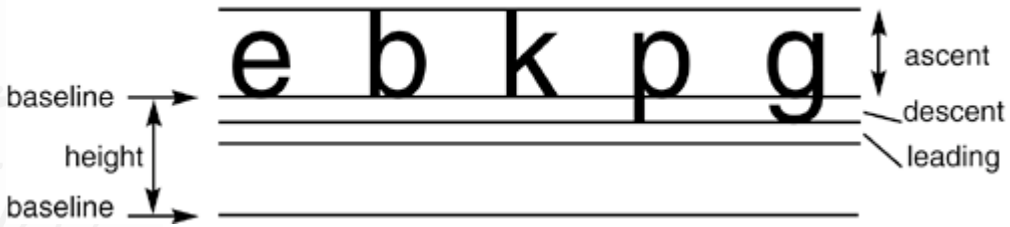
```
g2.setPaint(Color.RED);
g2.drawString("Warning!", 100, 100);

g2.fill(rect);
```

Using Fonts for Text

- Set a font by its *font face name* using **`setFont`**() method
  - A font face name is composed of a *font family name*, such as "**`Helvetica`**," and an optional *suffix* such as "**`Bold`**."
- AWT logical font names, to be mapped to fonts on the platform
  - **`SansSerif, Serif, Monospaced, Dialog, DialogInput`**
- Find out available fonts
  - **`getAvailableFontFamilyNames`**() method of the **`GraphicsEnvironment`** class

```
String[] fontNames = GraphicsEnvironment
                     .getLocalGraphicsEnvironment()
                     .getAvailableFontFamilyNames();
```

Typesetting Terms



Height of the enclosing rectangle = ascent + descent + leading

- The **`FontRenderContext`** object represents the font characteristics of the screen device
  - Example for obtaining a rectangle that encloses a string

```
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);
```

- Get string width, height and ascent

```
double stringWidth = bounds.getWidth();
double stringHeight = bounds.getHeight();
double ascent = -bounds.getY();
```

Displaying Images

- Loading images
  - Supported image file formats, as of 1.6
    - GIF, JPEG, PNG, BMP WBMP
  - From local file

```
String filename = "...";
Image image = ImageIO.read(new File(filename));
```

  - From URL

```
String urlname = "...";
Image image = ImageIO.read(new URL(urlname));
```

- Display images

The observer parameter is usually null

```
g.drawImage(image, x, y, null);
```

Supporting User Interaction

- Problem: determine when the user clicks on one of the displayed graphics
  - **`contains`**() method of a shape class (a class that implements the **`Shape`** interface) determines whether a mouse click was within the bounds of the shape

e.g.

```
public void mousePressed(MouseEvent e){
    if(rect.contains(e.getX(), e.getY())) updateLocation(e);
    . . .
}
```

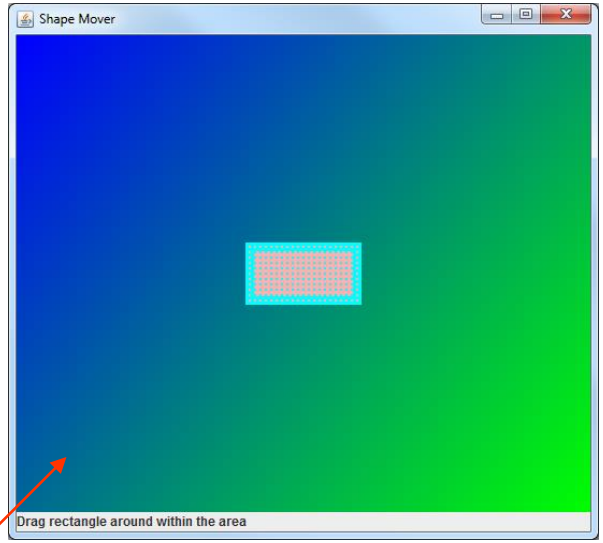
Example: Shape Mover

- Check whether the mouse is clicked inside of the rectangle

```
public void mousePressed(MouseEvent e){
    if ( rect.contains(e.getX(), e.getY())){
        updateLocation(e);
}
```

- Update the location of the rectangle

```
rect.setLocation(last_x + e.getX(),
                 last_y + e.getY());
```



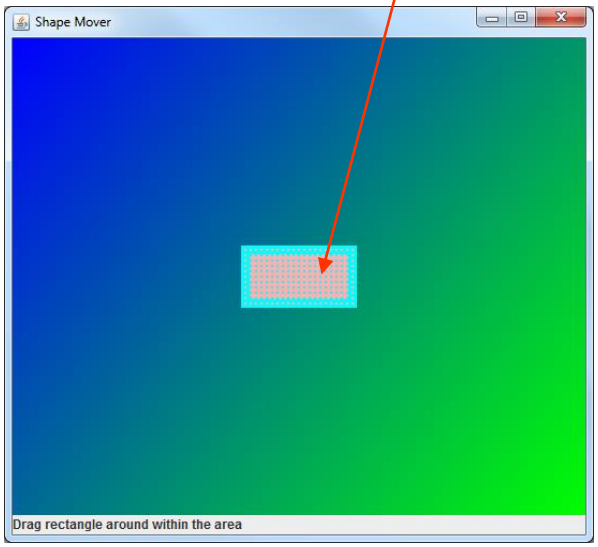
- Create a rectangle with gradient paint

```
g2.setPaint(new GradientPaint(0f,0f,Color.blue,(float)w,(float)h,Color.green));
g2.fillRect(0, 0, w, h);
```

Example: Shape Mover *continued*

- Using the **`BufferedImage`** object to create the ***fill texture*** paint pattern and fill the rectangle

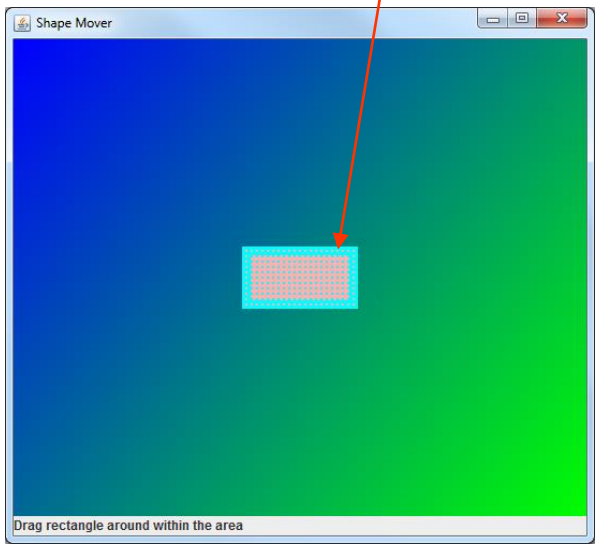
```
BufferedImage bi = new BufferedImage(5, 5, BufferedImage.TYPE_INT_RGB);
Graphics2D big = bi.createGraphics();

big.setColor(Color.pink);
big.fillRect(0, 0, 7, 7);
big.setColor(Color.cyan);
big.fillOval(0, 0, 3, 3);
Rectangle r = new Rectangle(0,0,5,5);
TexturePaint fillPolka = new TexturePaint(bi, r);
big.dispose();

g2.setPaint(fillPolka);
g2.fill(rect);
```



Example: Shape Mover *continued*

- Using the **`BufferedImage`** object to create the ***stroke texture*** paint pattern and draw the rectangle

```
BufferedImage bi = new BufferedImage(5, 5, BufferedImage.TYPE_INT_RGB);
Graphics2D big = bi.createGraphics();

big.setColor(Color.cyan);
big.fillRect(0, 0, 7, 7);
big.setColor(Color.pink);
big.fillOval(0, 0, 3, 3);
Rectangle r = new Rectangle(0,0,5,5);
TexturePaint strokePolka = new TexturePaint(bi, r);
big.dispose();

g2.setStroke(new BasicStroke(8.0f));
g2.setPaint(strokePolka);
g2.draw(rect);
```



Animation

- Using Swing Timer
  - The **`actionPerformed`**() runs on the event-dispatching thread

```
// Create an instance of an anonymous subclass of ActionListener
ActionListener updateTask = new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        // update the location, then
        repaint();
    }
};

// Start and run the task at regular interval
new Timer(UPDATE_INTERVAL, updateTask).start();
```

Animation

- Using an update thread

```
// Create a new anonymous and inner runnable class to update at regular interval
Thread updateThread = new Thread() {
    public void run() {
        while (true) {

            // update the location, then
            repaint();

            try {
                // Delay and give other thread a chance to run
                Thread.sleep(UPDATE_INTERVAL);
            } catch (InterruptedException ignore) {}
        }
    }
};

updateThread.start();
```