Java2D Graphics – Part 4
• Coordinate Transformations in Java2D
  ◦ Java2D allows you to easily translate, rotate, scale, or shear the coordinate system

  ◦ This is very convenient: it is often much easier to move the coordinate system than to calculate new coordinates for each of your points

  ◦ Besides, for some data structures like ellipses and strings there is no other way to get rotated or stretched versions
Coordinate Transformations in Java2D

- The meanings of translate, rotate, and scale are: to move, to spin, or to stretch/shrink evenly in the x and/or y direction.

- *Shear* means to stretch unevenly: an x shear moves points to the right based on how far they are from the y axis; a y shear moves points down based on how far they are from the x axis.
Coordinate Transformations in Java2D

- The easiest way to picture what is happening is to imagine that the person doing the drawing has a picture frame that he lays down on top of a sheet of paper.

- The drawer always sits at the bottom of the frame. To apply a translation, you move the frame (moving the drawer with it), and do the drawing in the new location.
Coordinate Transformations in Java2D

- You then move the frame back to its original location, and what you now see is the final result

- Similarly, for a rotation, you spin the frame (and the drawer), draw, then spin back to see the result

- Similarly for scaling and shears; modify the frame without touching the underlying sheet of paper, draw, then reverse the process to see the final result
Coordinate Transformations in Java2D

- An outside observer watching this process would see the frame move in the direction specified by the transformation, but see the sheet of paper stay fixed.

- This is illustrated in the second column in the diagram below.

- The dotted rectangle represents the frame, while the gray rectangle represents the sheet of paper.
Coordinate Transformations in Java2D

- On the other hand, to the person doing the drawing it would appear that the sheet of paper moved in the opposite way from that specified in the transformation, but that he didn't move at all.

- This is illustrated in the third column in the following diagram.

- The first column illustrates the starting configuration, and the fourth illustrates the final result.
### Coordinate Transformations in Java2D

<table>
<thead>
<tr>
<th>Before Transformation</th>
<th>Outsider's Perspective</th>
<th>Drawer's Perspective</th>
<th>Final Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
</tbody>
</table>

**Java2D Graphics**
Coordinate Transformations in Java2D

- You can also perform more complex transformations (e.g. creating a mirror image by flipping around a line) by directly manipulating the underlying arrays that control the transformations.

- This is a bit more complicated to envision than the basic translation, rotation, scaling, and shear transformations.
Coordinate Transformations in Java2D

The idea is that a new point \((x_2,y_2)\) can be derived from an original point \((x_1,y_1)\) as follows:

\[
\begin{bmatrix}
  x_2 \\
  y_2
\end{bmatrix}
= \begin{bmatrix}
m_{00} & m_{01} & m_{02} \\
m_{10} & m_{11} & m_{12}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
y_1
\end{bmatrix}
= \begin{bmatrix}
m_{10}x_1 + m_{11}y_1 + m_{12}
\end{bmatrix}
\]
Coordinate Transformations in Java2D

- Note that you can only supply six of the nine values in the transformation array (the \( m_{xx} \) values)

- The bottom row is fixed at \([0 0 1]\) to guarantee that the transformations preserve "straightness" and "parallelness" of lines

- There are several ways of supplying this array to the AffineTransform constructor; see the AffineTransform API for details
Coordinate Transformations in Java2D

- There are two basic ways to use transformations:
  - You can create an AffineTransform object, set its parameters, and then assign that AffineTransform to the Graphics2D object via setTransform().
  - This is your only choice if you want to do the more complex transformations permitted by setting explicit transformation matrices.
  - Alternatively, for the basic transformations you can call translate, rotate, scale, and shear directly on the Graphics2D object.
Coordinate Transformations in Java2D: Translations and Rotations

import java.awt.*;

// An example of coordinate translations and rotations
// with Java2D in Java 1.2
public class RotationExample extends StrokeThicknessExample {
    private Color[] colors = {Color.white, Color.black};

    public void paintComponent (Graphics g) {
        clear (g);
        Graphics2D g2d = (Graphics2D) g;
        drawGradientCircle (g2d);
        drawThickCircleOutline (g2d); // Move the origin to the
        // center of the circle
        g2d.translate (185.0, 185.0);
Coordinate Transformations in Java2D: Translations and Rotations

for (int i = 0; i < 16; i++) // Rotate the coordinate system
    // around current origin, which is
    // at he center of the circle
{
    g2d.rotate (Math.PI / 8.0);
    g22d.setPaint (colors[i % 2]);
    g2d.drawString ("Java", 0, 0);
}

public static void main (String [] args)
{
    WindowUtilities.openInJFrame (new RotationExample (), 380, 400);
}

Java2D Graphics
Coordinate Transformations in Java2D: Translations and Rotations

Result:
Coordinate Transformations in Java2D: Shear Transformations

- If you specify a non-zero x shear, then x values will be more and more shifted to the right the farther they are away from the y axis.

- For example, an x shear of 0.1 means that the x value will be shifted 10% of the distance the point is away from the y axis.

- y shears are similar: points are shifted down in proportion to the distance they are away from the x axis.
Coordinate Transformations in Java2D: Shear Transformations

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

// An example of shear transformations with Java2D in Java 1.2
public class ShearExample extends JPanel
{
    private static int gap = 10;
    private static int width = 100;

    private Rectangle rect = new Rectangle (gap, gap, 100, 100);

    public void paintComponent (Graphics g)
    {
        super.paintComponent (g);
        Graphics2D g2d = (Graphics2D) g;
```
Coordinate Transformations in Java2D: Shear Transformations

```java
for (int i = 0; i < 5; i++)
{
    g2d.setPaint (Color.red);
    g2d.fill (rect); // Each new square gets 0.2 more x shear
    g2d.shear (0.2, 0.0);
    g2d.translate (2 * gap + width, 0);
}
```

```java
public static void main (String[] args)
{
    String title = "Shear: x shear ranges from 0.0 for the leftmost 'square' " + "to 0.8 for the rightmost one."
    WindowUtilities.openInJFrame (new ShearExample (), 20 * gap + 5 * width, 5 * gap + width, title);
}
```
Coordinate Transformations in Java2D: Shear Transformations

- Result:
• Additional Information
  ◦ Requesting More Accurate Drawing: Rendering Hints
    • Since Java2D already does a lot of calculations compared to the old AWT, there are several optional features that the designers chose to disable by default in order to improve performance

    • Turning them on results in crisper drawing, especially for rotated text
Additional Information

- Requesting More Accurate Drawing: Rendering Hints
  - For example, the JTable used for the transformations example above resulted in excessively jagged text using the default settings
  - The most important two settings are to turn on anti-aliasing (smooth jagged lines by blending colors) and to simply request the highest-quality rendering
  - This approach is illustrated in the following code:
Additional Information

- Requesting More Accurate Drawing: Rendering Hints

    RenderingHints renderHints = new RenderingHints
            (RenderingHints.KEY_ANTIALIASING,
             RenderingHints.VALUE_ANTIALIAS_ON);
    renderHints.put (RenderingHints.KEY_RENDERING,
                      RenderingHints.VALUE_RENDER_QUALITY);
    ...

    public void paintComponent (Graphics g)
    {
        super.paintComponent (g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHints (renderHints);
        ...
    }

Java2D Graphics
• Other Capabilities of Java2D
  ◦ High-quality printing
  ◦ Improved XOR mode
  ◦ Custom color mixing (implement Composite and CompositeContext interfaces)
  ◦ Bounds/hit testing (see contains and intersects methods of Shape)
• Other Capabilities of Java2D

  ◦ Fancy text operations
    • Create new fonts by transforming old ones (use Font.deriveFont ( ))
    • Draw multi-font or multi-color strings (use the draw () method of TextLayout)
    • Draw outlines of fonts, or fill fonts with images or gradient colors (use the getOutline () method of TextLayout)

  ◦ Low-level image processing and color model manipulation
Questions/comments/A string walks into a bar...

Java2D Graphics