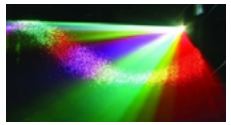
Chapter 6
Described Static Visual Content

The Design and Implementation of Multimedia Software

David Bernstein

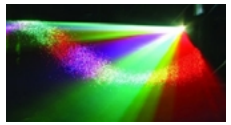
Jones and Bartlett Publishers

www.jbpub.com



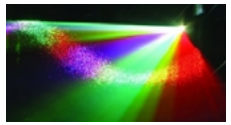
About this Chapter

- The description of static visual content the description of *geometric shapes* and the way in which they should be rendered.
- Two common examples of described static visual content are *scalable graphics* and *outline fonts*



Some Common (Simple) Geometric Shapes

0-Dimensional	1-Dimensional	2-Dimensional
Point	Line	Rectangle
	Curve	Polygon
		Ellipse



Some Participants in Java

- Requirements of Geometric Shapes:

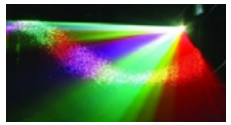
  `Shape`

- A Realization:

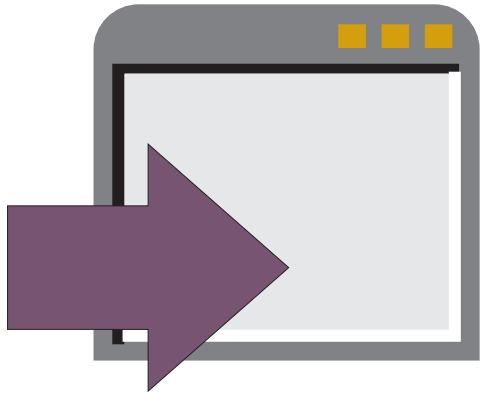
  `Rectangle`

- Rendering:

  Use the `draw()` method in the `Graphics2D` class



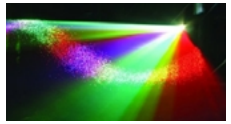
A Simple Example – Demonstration



In `examples/chapter`:

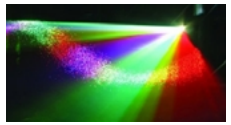
RandomRectangle.html

```
java -cp RandomRectangle.jar RandomRectangleApplication
```



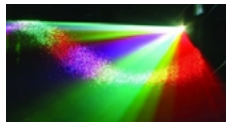
A Simple Example – The Structure

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

public class RandomRectangleCanvas extends JComponent
{
    private Random      generator;

    public RandomRectangleCanvas()
    {
       super();
       generator = new Random(System.currentTimeMillis());
    }
}
```



A Simple Example – paint()

```
public void paint(Graphics g)
{
   Graphics2D      g2;
   int             height, maxHeight, maxWidth, width, x, y;
   Rectangle       rectangle;


   g2 = (Graphics2D)g;

   maxHeight = getHeight();
   maxWidth  = getWidth();

   x      = generator.nextInt(maxWidth  - 1);
   y      = generator.nextInt(maxHeight - 1);
   width  = generator.nextInt(maxWidth  - x - 1);
   height = generator.nextInt(maxHeight - y - 1);

   rectangle = new Rectangle(x, y, width, height);

   g2.draw(rectangle);
}
```



Some Java Classes

- 0-Dimensional:

  `Point2D`

- 1-Dimensional:

  `Line2D`

  `CubicCurve2D`

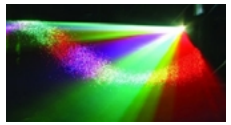
  `QuadCurve2D`

- 2-Dimensional:

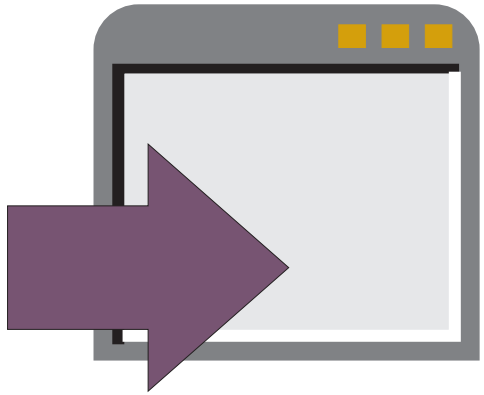
  `Rectangle2D`

  `RoundRectangle2D`

  `Ellipse2D`



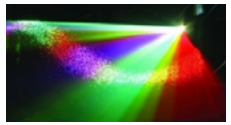
Some Java Classes – A Demonstration



In `examples/chapter`:

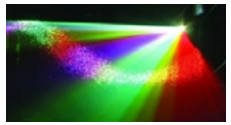
SimpleShapeCanvas.html

```
java -cp SimpleShapeCanvas.jar SimpleShapeCanvasApplication
```



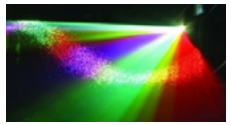
Defining Points

- A point is defined by its coordinates.
- On the plane, a point has two coordinates, usually called the x-coordinate (i.e., horizontal coordinate) and y-coordinate (i.e., vertical coordinate).



An Example of a Point

```
// Construct a point
point = new Point2D.Double(20.0, 30.0);
```



Explicit Form of a Line

- Defined:

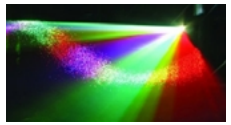
$$y = Ax + B \tag{1}$$

- Shortcomings:

  Does not enable us to represent vertical lines.

  Second, one is generally interested in line segments, not lines.

  Though familiar, it is not very intuitive.

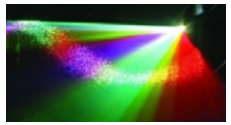


Parametric Form

The *parametric form of a line segment* in terms of the coefficients $a = (a_x, a_y)$ and $b = (b_x, b_y)$ is:

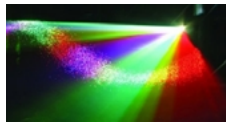
$$x(u) = u^0 a_x + u^1 b_x = a_x + u b_x \quad (2)$$

$$y(u) = u^0 a_y + u^1 b_y = a_y + u b_y \quad (3)$$



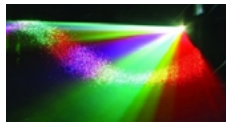
Parametric Form (cont.)

- In Java, a line segment is defined in terms of its two endpoints, p and q.
- To ensure that $(x(0), y(0))$ equals p, set $a_x = p_x$ and $a_y = p_y$.
- To ensure that $(x(1), y(1))$ equals q, set $b_x = q_x - p_x$ and $b_y = q_y - p_y$.
- Java's `Line2D` class does this calculation.



An Example of a Line

```
// Construct a line
line = new Line2D.Double(0.0, 0.0, 50.0, 75.0);
```

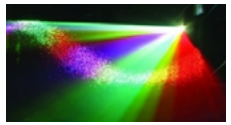


Parametric Form (cont.)

The parametric form can also be written as:

$$x(u) = p_x + uq_x - up_x = (1-u)p_x + uq_x \quad (4)$$
$$y(u) = p_y + uq_y - up_y = (1-u)p_y + uq_y \quad (5)$$

That is, $x(u)$ is a *convex combination* of p_x and p_y, and $y(u)$ is a convex combination of q_x and q_y [with weights $(1-u)$ and u].



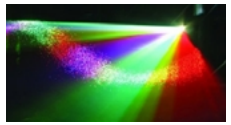
Quadratic Bézier Curve Segments

A quadratic Bézier curve segments is defined in terms of the two end points, p and q, and one *control point*, r, as follows:

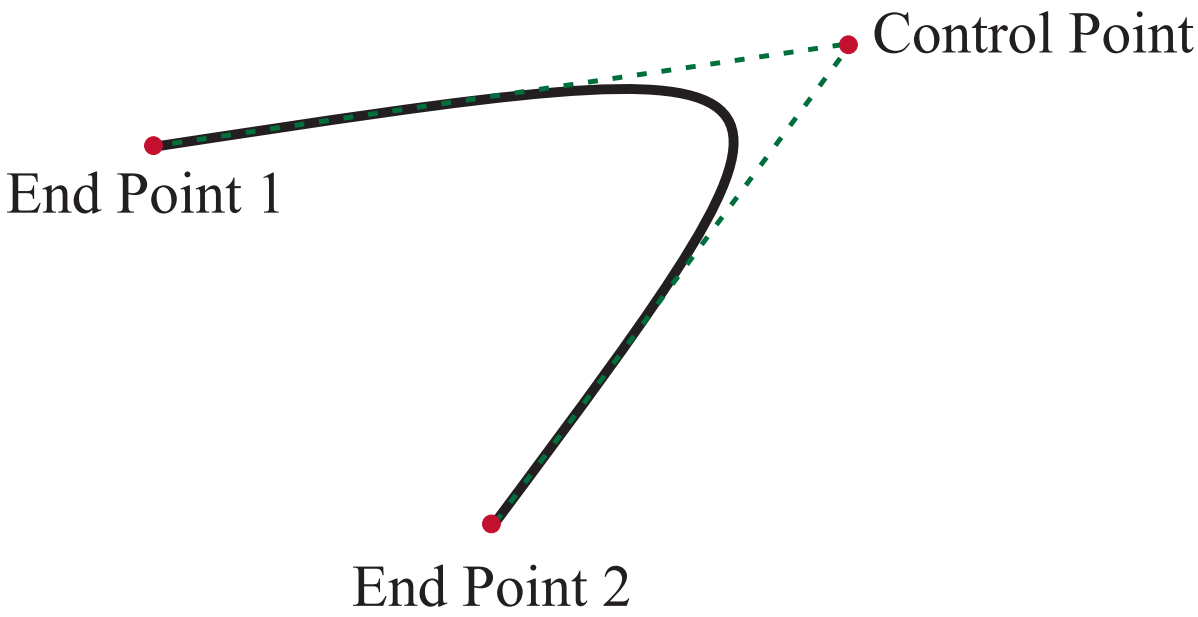
$$x(u) = (1-u)^2 p_x + 2(1-u)u r_x + u^2 q_x \quad (6)$$

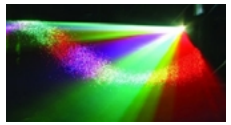
$$y(u) = (1-u)^2 p_y + 2(1-u)u r_y + u^2 q_y \quad (7)$$

for $u \in [0, 1]$.



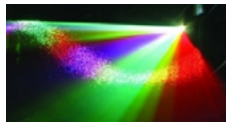
Quadratic Bézier Curve Segments (cont.)





An Example of a Quadratic Bézier Curve Segment

```
// Construct a quadratic curve
quadraticCurve = new QuadCurve2D.Double(
                                  120.0, 120.0,  // End 1
                                  300.0, 180.0,  // Control
                                  130.0, 190.0); // End 2
```



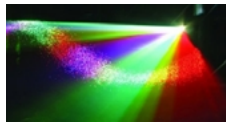
Cubic Bézier Curve Segments

Finally, a *cubic Bézier curve segment* is defined in terms of the two end points, p and q, and two control points, r and s, as follows:

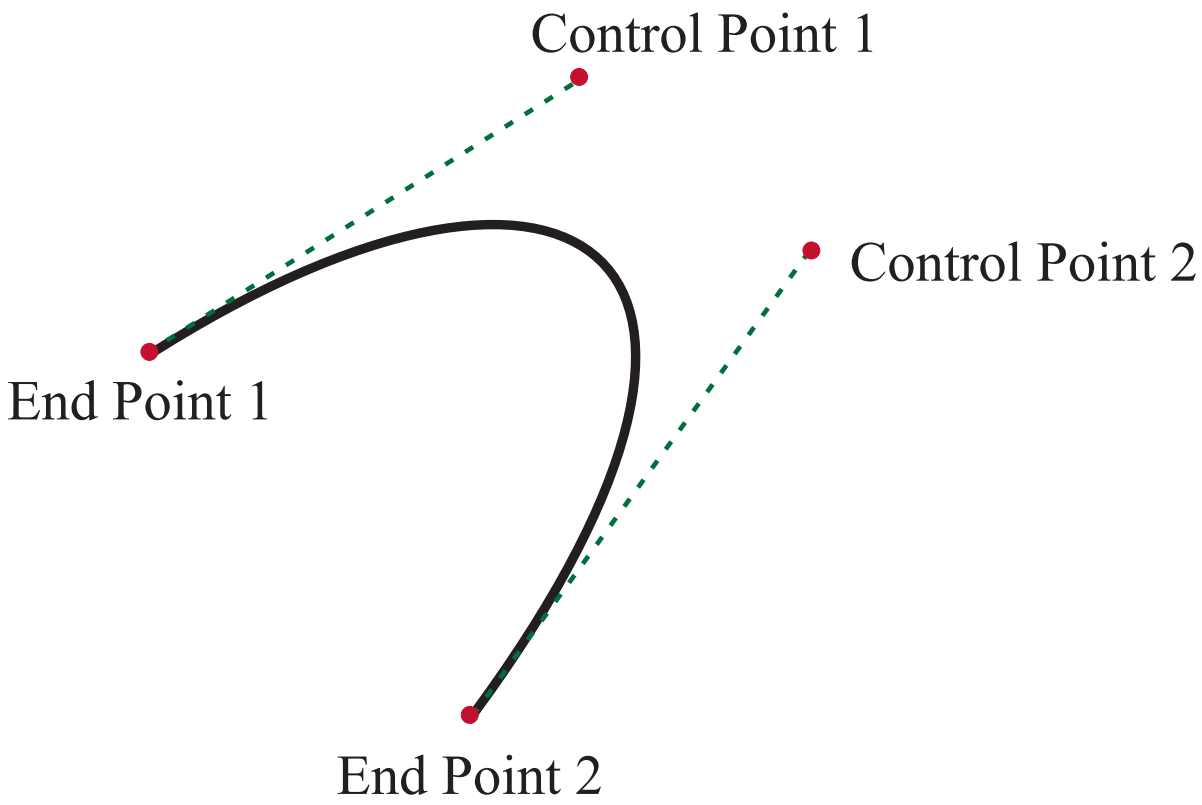
$$x(u) = (1-u)^3 p_x + 3(1-u)^2 u r_x + 2(1-u)^2 u s_x + u^3 q_x \quad (8)$$

$$y(u) = (1-u)^3 p_y + 3(1-u)^2 u r_y + 2(1-u)^2 u s_y + u^3 q_y \quad (9)$$

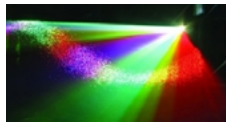
for $u \in [0, 1]$.



Cubic Bézier Curve Segments (cont.)

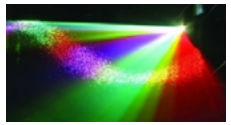


Note that the slope of the curve at each end point is the slope of the line segment connecting that end point and its associated control point.



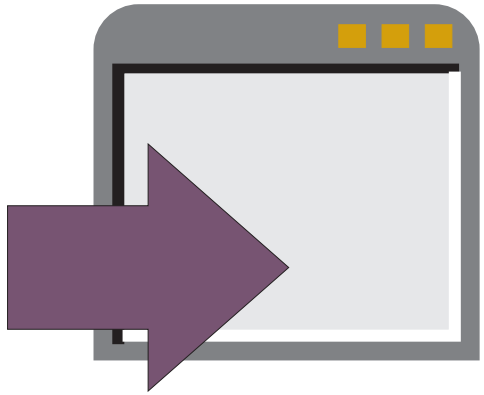
An Example of a Cubic Bézier Curve Segment

```
// Construct a cubic curve
cubicCurve = new CubicCurve2D.Double(320.0, 320.0,  // End 1
                                     300.0, 180.0,  // Control 1
                                     330.0, 370.0,  // End 2
                                     360.0, 390.0); // Control 2
```



An Advantage of Bézier Curves

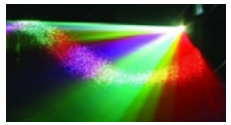
They can be manipulated using the end points and the control point(s).



In `extras`:

Curve.html

```
java -cp Curve.jar CurveApplication
```



Rectangles

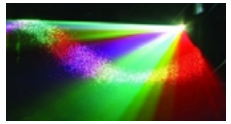
- Loose Definition:

  A four-sided shape with four right angles in which the sides are parallel to the coordinate axes.

- Specification:

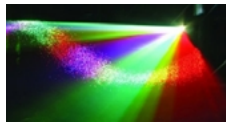
  Use the four corners.

  Use one corner and the width and height.



An Example of a Rectangle

```
// Construct a rectangle
rectangle = new Rectangle2D.Double(10.0, 20.0, // Upper Left
                                   30.0,       // Width
                                   60.0);      // Height
```



Ellipses

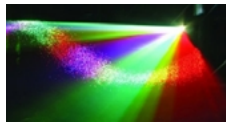
- Loose Definition:

  A flattened circle.

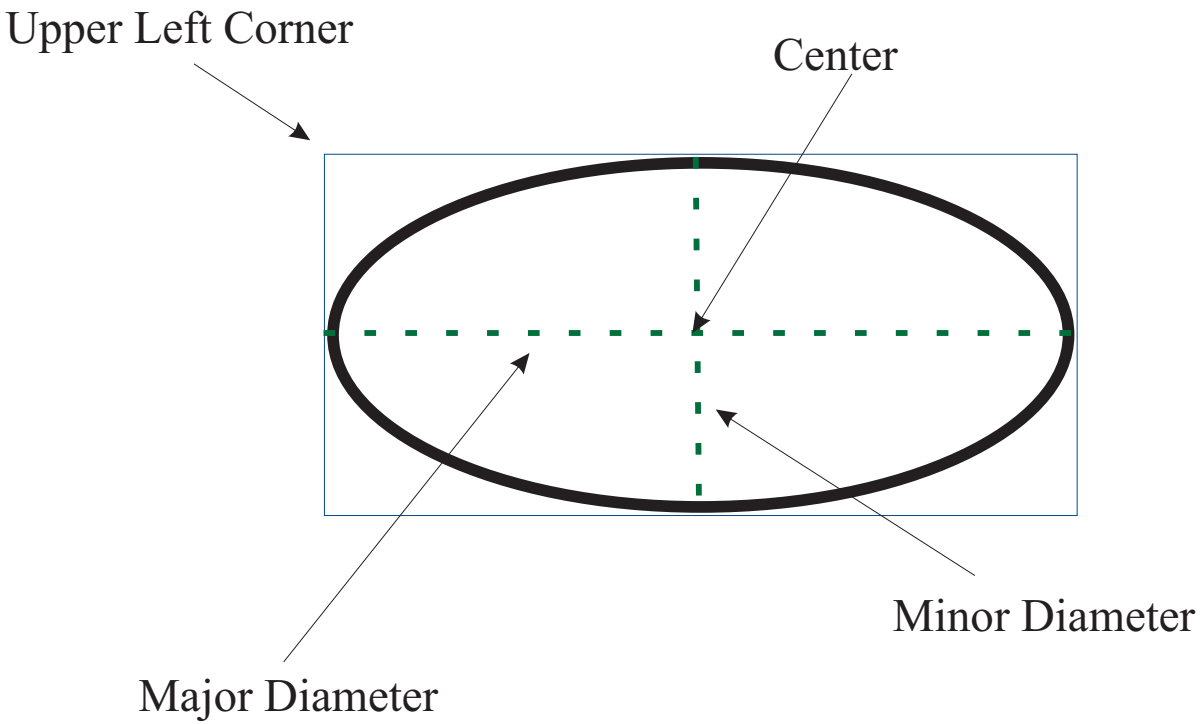
- Specification:

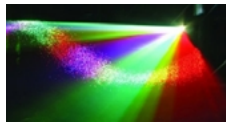
  Use the center and major and minor diameters (or major and minor axes).

  Use one corner and the width and height.



Ellipses (cont.)





Arcs

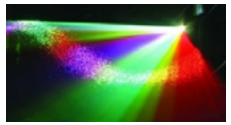
- Loose Definition:

  A part of an ellipse.

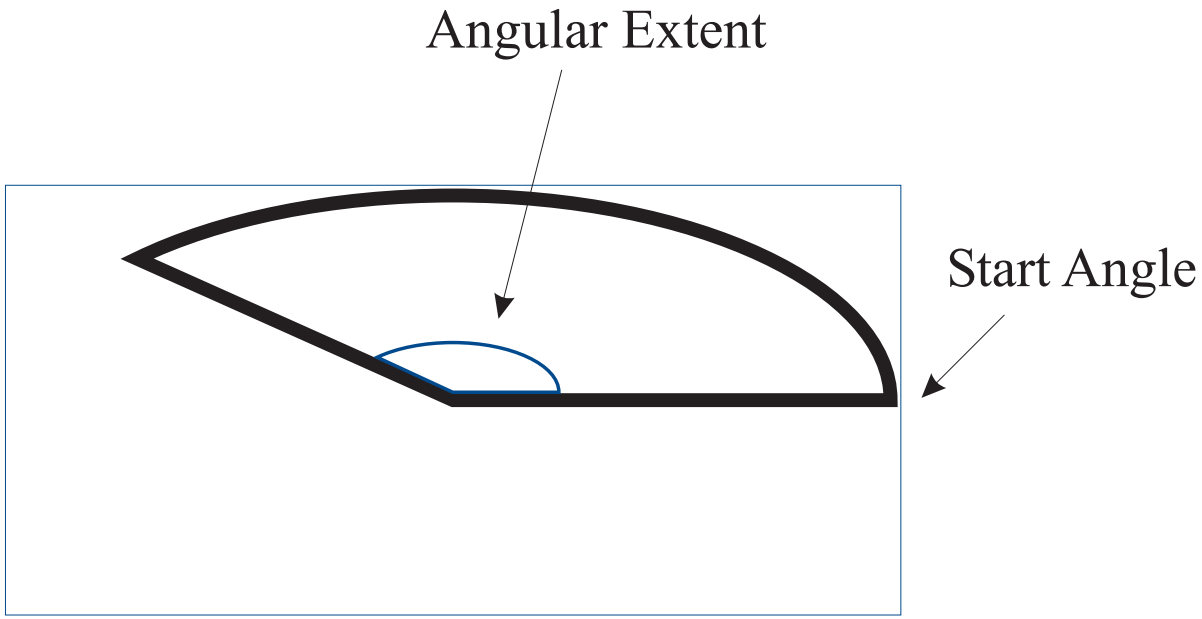
- Specification:

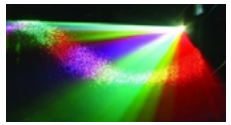
  Use an ellipse and the starting and ending angles.

  Use an ellipse and the starting and angular extent.



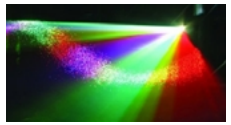
Arcs (cont.)





An Example of an Arc

```
// Construct an arc
arc = new Arc2D.Double(10.0, 200.0, // Upper Left
                       150.0,       // Width
                       100.0,       // Height
                       0.0,         // Starting Angle
                       135.0,       // Angular Extent
                       Arc2D.PIE);  // Type
```



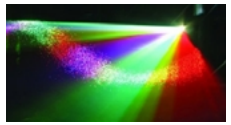
Categorizing Shapes

- Closed/Open:

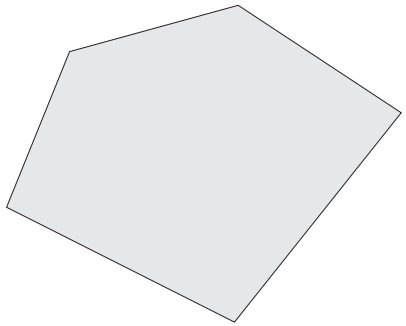
  Closed shapes have a well-defined interior and exterior.

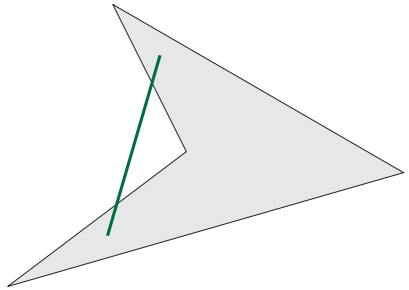
- Convex/Nonconvex:

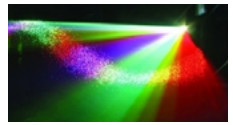
  A shape is convex if a line segment drawn between any two points in the shape lies entirely inside the shape.



Categorizing Shapes (cont.)



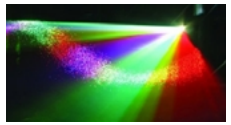




Glyphs

Definition

A *glyph* is a shape that represents one or more characters (in a character set).

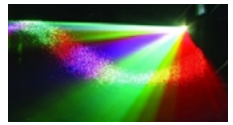


Ligatures

Most of the time, a glyph represents a single character. However, for aesthetic reasons, multiple characters are sometimes represented by a single glyph called a *ligature*.

Two glyphs

One glyph
(a ligature)

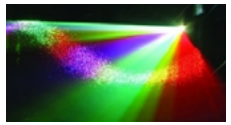


Fonts

Definition

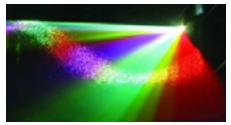
A *font* is a set of glyphs.

Fonts are usually said to have a *face* or *family* (e.g., Garamond, Times New Roman) and a *style* (e.g., italic, bold).

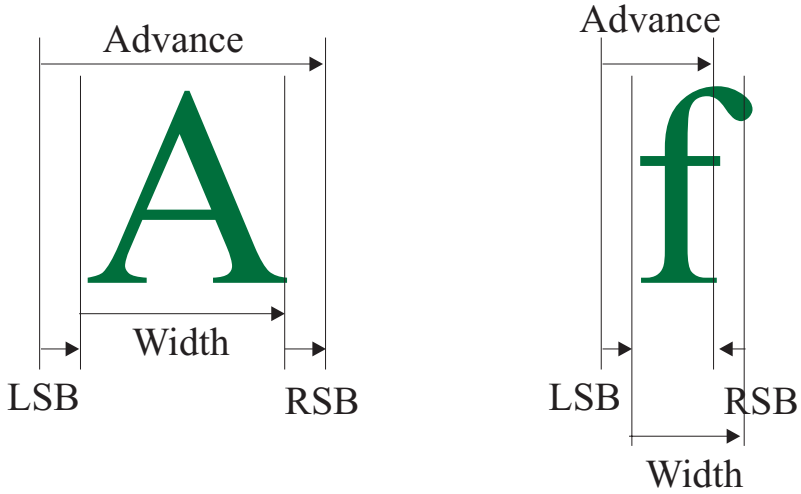


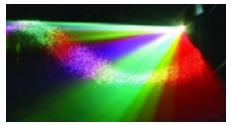
Properties of Glyphs

- *Width*
- *Left-Side Bearing*
- *Right-Side Bearing*
- *Advance* – the sum of the three



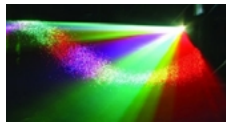
Positive and Negative Right-Side Bearings





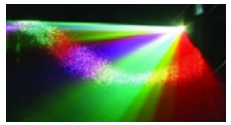
Kerning

The use of different bearings for different glyphs is called kerning.



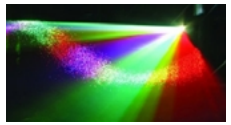
Some Details in Java

- The rendering engine keeps information about fonts in a `FontRenderContext` object.
- Given a `FontRenderContext`, a `Font` object can create a `GlyphVector`.
- `GlyphVector` has several `getOutline()` methods, each of which returns a `Shape`.

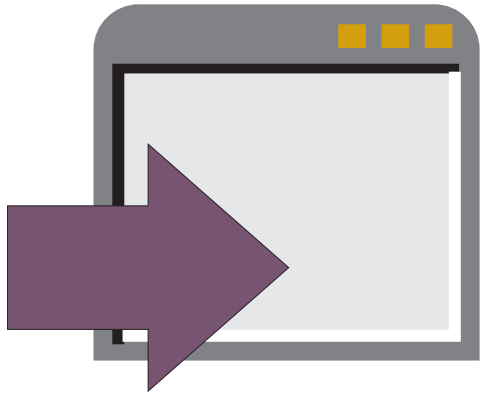


An Example of Glyphs

```
protected void paintGlyphs(Graphics2D g2, String text)
{
    FontRenderContext       frc;
    GlyphVector             glyphs;
    Shape                   shape;

    frc = g2.getFontRenderContext();

    glyphs = font.createGlyphVector(frc, text);
    shape = glyphs.getOutline(0.0f, 100.0f);

    g2.setColor(Color.BLACK);
    g2.draw(shape);
}
```



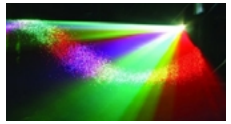
An Example of Glyphs – Demonstration



In `examples/chapter`:

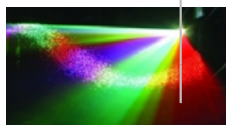
Glyph.html

```
java -cp Glyph.jar GlyphApplication GlyphCanvas Media
```



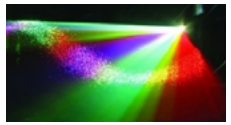
An Example of Fonts

```
public FontCanvas(int fSize)
{
   GraphicsEnvironment    ge;
   int                    i;

   this.fSize = fSize;
   current    = 0;

   // Get the available font families
   ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
   fontFamilies = ge.getAvailableFontFamilyNames();

   // Construct the fonts and text samples
   fonts  = new Font[fontFamilies.length];
   sample = new String[fontFamilies.length];

   for (i=0; i < fontFamilies.length; i++)
   {
      fonts[i] = new Font(fontFamilies[i],
                          Font.PLAIN, fSize);

      sample[i] = fontFamilies[i]+
         "     abcdefABCDEF123456!@#^" +
         "     Av     fi    " +
         "     \u00c0 \u00c1 \u00c2 \u00c7";

      if (fontFamilies[i].equalsIgnoreCase("Dialog"))
      {
```

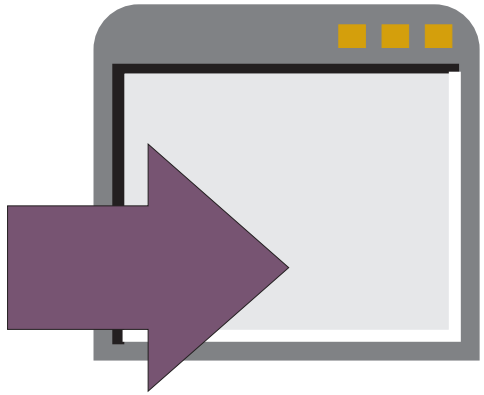


An Example of Fonts (cont.)

```
            current = i;
        }
    }
}
```



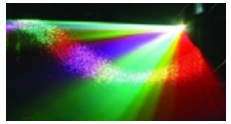
An Example of Fonts – Demonstration



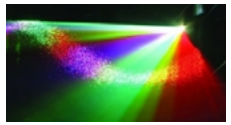
In `extras`:

FontCanvas.html

```
java -cp FontCanvas.jar FontCanvasApplication
```

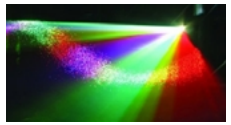


Terminology



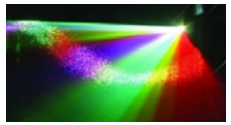
Font Metrics in Java

- Given a `FontRenderContext`, a `Font` can provide a `LineMetrics` object.
- This object can be used to get information about the font's metrics.

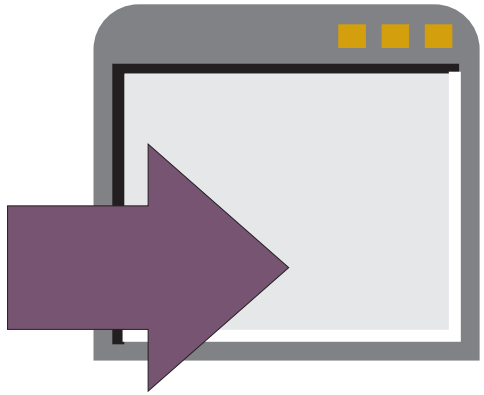


An Example Using LineMetrics

```
FontRenderContext       frc;
GlyphVector             glyphs;

frc     = g2.getFontRenderContext();
glyphs  = font.createGlyphVector(frc, text);
LineMetrics             lm;

lm      = font.getLineMetrics(text, frc);
float                   ascent, descent, height, leading;

// Get the various metrics
ascent  = lm.getAscent();
descent = lm.getDescent();
height  = lm.getHeight();
leading = lm.getLeading();
```



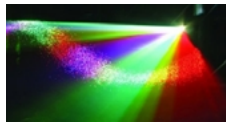
An Example Using `LineMetrics` – Demonstration



In `examples/chapter`:

GlyphMetrics.html

```
java -cp Glyph.jar GlyphApplication GlyphMetricsCanvas abjMZ
```



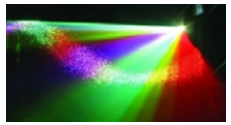
Using `LineMetrics` to Center Text

Get the Bounds

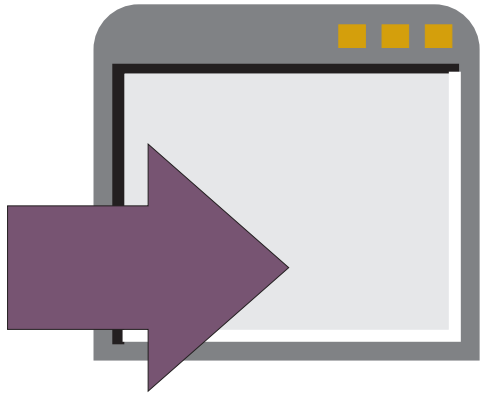
```
// Get the bounding box
bounds = glyphs.getVisualBounds();
```

Center the Shape

```
Dimension               d;
float                   x, y;

d = getSize();

// Center the text
x = (float)(d.width/2.  - bounds.getWidth()/2. );
y = (float)(d.height/2. + bounds.getHeight()/2.);

// Get the outline when centered
shape = glyphs.getOutline(x,y);
```



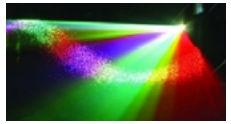
Centering Text – Demonstration



In `examples/chapter`:

CenteredGlyph.html

```
java -cp Glyph.jar GlyphApplication CenteredGlyphCanvas Media
```



There Must be an Easier Way

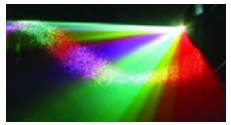
- For Plain Text:

  ```
  drawGlyphVector(GlyphVector, float, float)
  ```

  ```
  drawString(String, float, float)
  ```

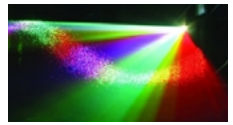
- For Attributed Text:

  ```
  drawString(AttributedCharacterIterator, float, float)
  ```

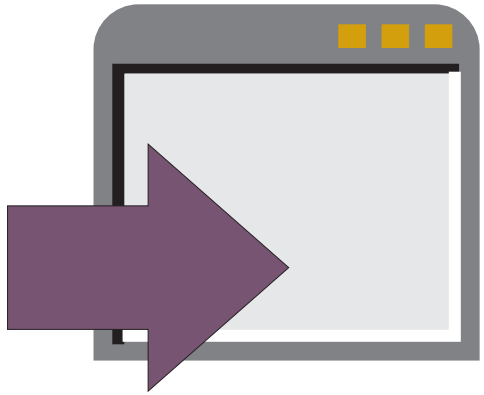


An Example of Attributed Text

```
AttributedString        as;

as = new AttributedString(text);
for (int i=0; i < text.length(); i++)
{
   as.addAttribute(TextAttribute.FONT,
                   new Font("Serif", Font.PLAIN, (i+10)),
                   i, i+1);
}
g2.setColor(Color.BLACK);
g2.drawString(as.getIterator(), 0, 100);
```



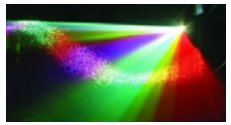
Attributed Text – Demonstration



In `examples/chapter`:

AttributedString.html

```
java -cp Glyph.jar GlyphApplication AttributedStringCanvas MultimediaSoftware
```



Think Like a Pen Plotter

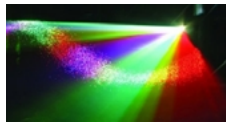
- The Idea:

  A common way to describe complicated geometric shapes is with a sequence of "move to", "line to", "quadratic-curve to", and/or "cubic-curve to" segments.

- In Java:

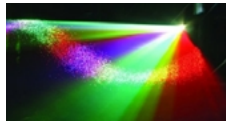
  Create a `Path2D` object.

  All objects that implement the `Shape` interface have a `getPathIterator()` method that returns a `PathIterator` object



An Example of a Complicated Shape

```
bodyShape = new Path2D.Float();
bodyShape.moveTo( 20, 50);
bodyShape.lineTo( 20, 70);
bodyShape.lineTo( 20, 90);
bodyShape.lineTo( 10, 90);
bodyShape.lineTo( 10,100);
bodyShape.lineTo( 80,100);
bodyShape.lineTo( 80, 90);
bodyShape.lineTo( 40, 90);
bodyShape.lineTo( 40, 70);
bodyShape.lineTo( 40, 50);
bodyShape.closePath();
```

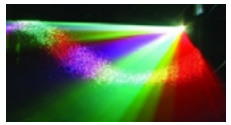


Constructive Area Geometry

Definition

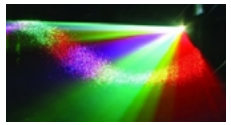
Constructive area geometry (CAG) is the process of performing set-theoretic operations (e.g., union, intersection, difference, symmetric difference) on two-dimensional shapes.

In Java, constructive area geometry functionality is provided by the `Area` class, which can be used to decorate any object that implements the `Shape` interface.

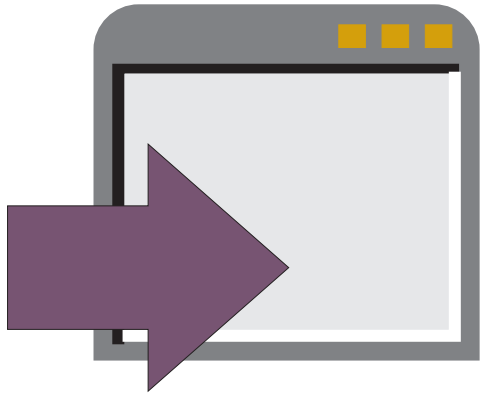


An Example of CAG

```
e = new Area(ellipse);
r = new Area(rectangle);

if      (op.equalsIgnoreCase("Union"))
                e.add(r);
else if (op.equalsIgnoreCase("Intersection"))
                e.intersect(r);
else if (op.equalsIgnoreCase("Difference"))
                e.subtract(r);
else
                e.exclusiveOr(r);

g2.setPaint(GOLD);
g2.fill(e);
g2.setPaint(PURPLE);
g2.draw(e);
```



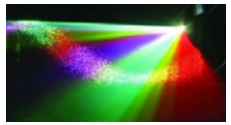
An Example of CAG – Demonstration



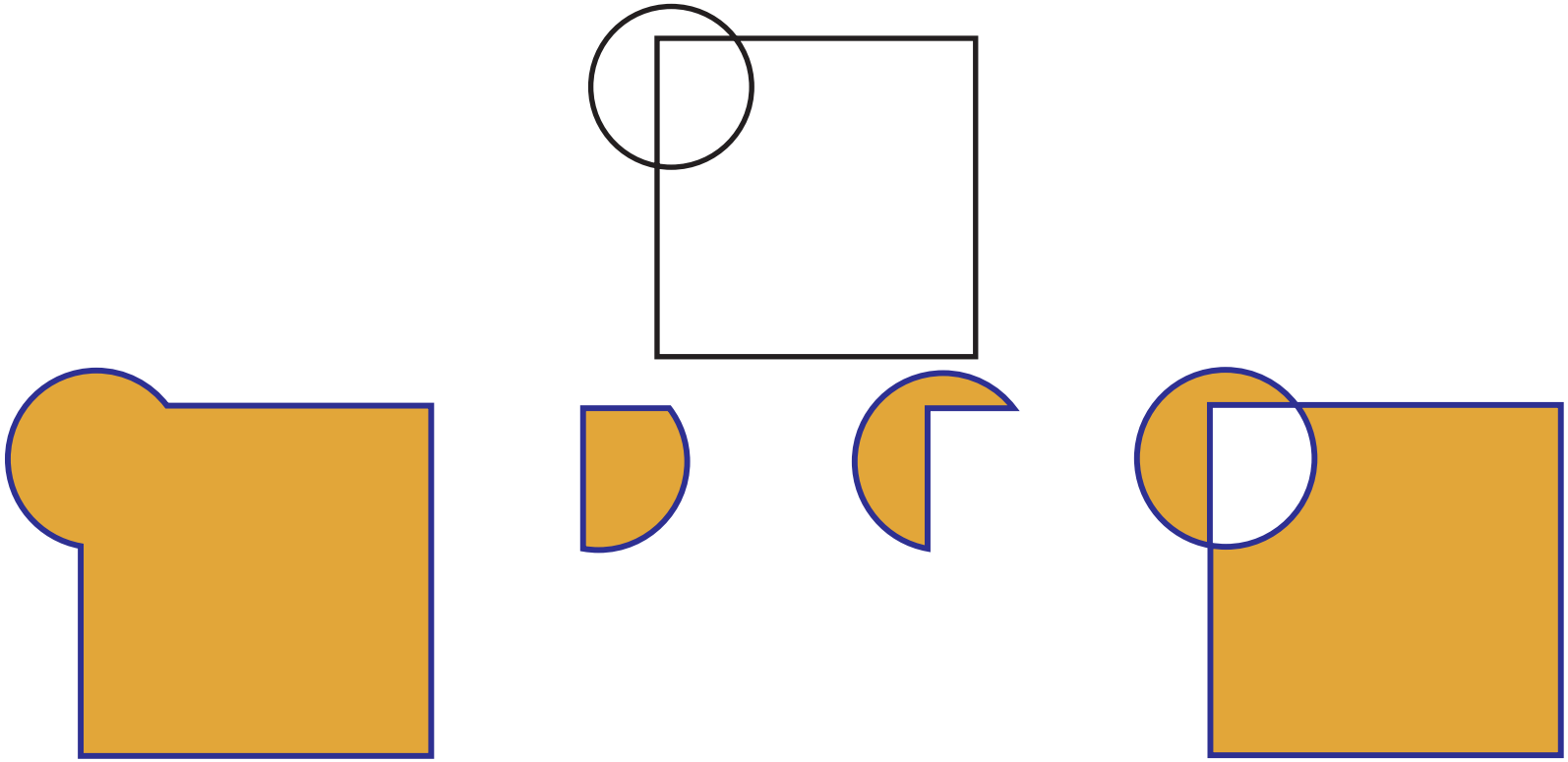
In `examples/chapter`:

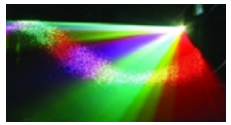
ConstructiveAreaGeometry.html

```
java -cp ConstructiveAreaGeometry.jar ConstructiveAreaGeometryApplication
```



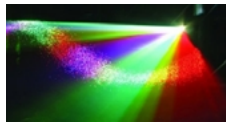
An Example of CAG (cont.)





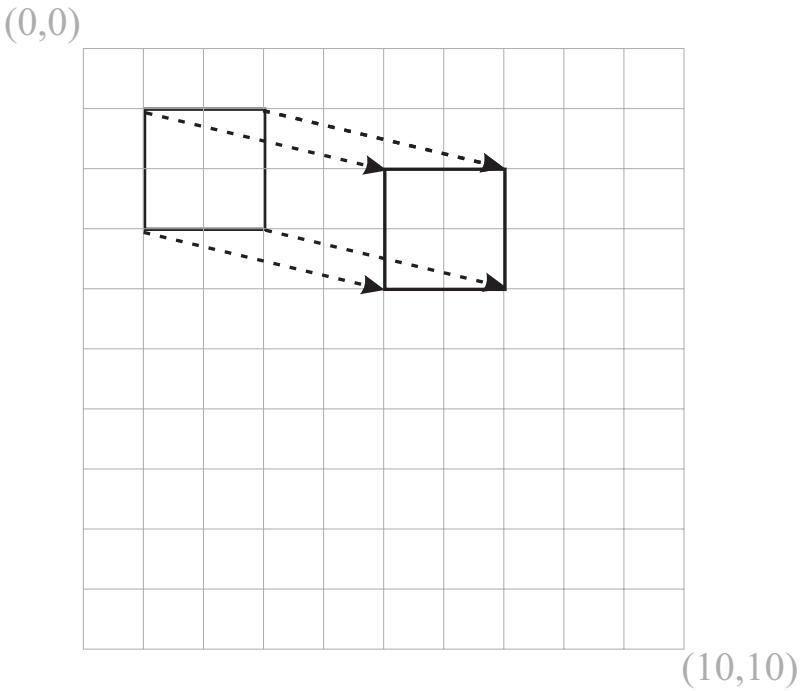
Affine Transformations

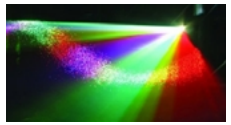
- In many situations, one doesn't want to transform the coordinate space, instead one wants to transform the content itself.
- Again, the most common transforms are translation, scaling, rotation and reflection.
- In Java, transforms are implemented in the `AffineTransform` class.



Translation

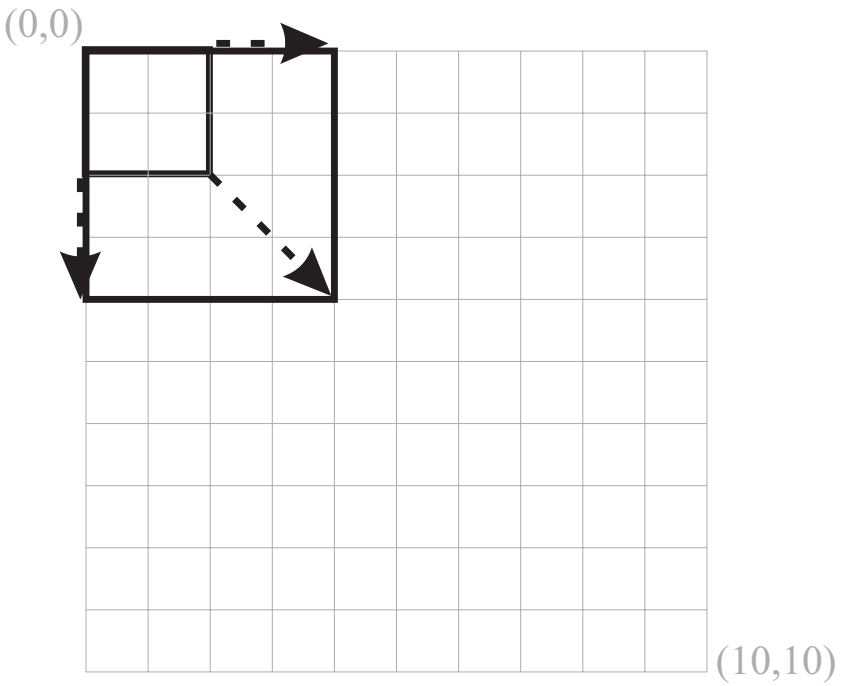
`AffineTransform.getTranslateInstance(double, double)`

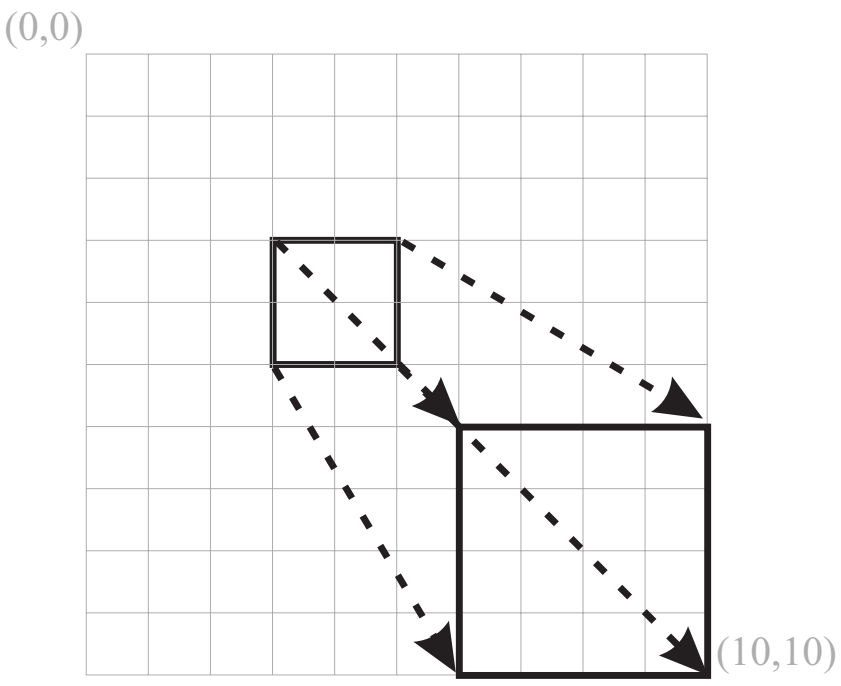


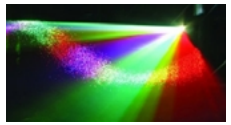


Scaling

`AffineTransform.getScaleInstance(double, double)`

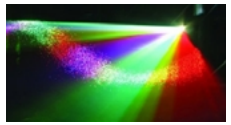




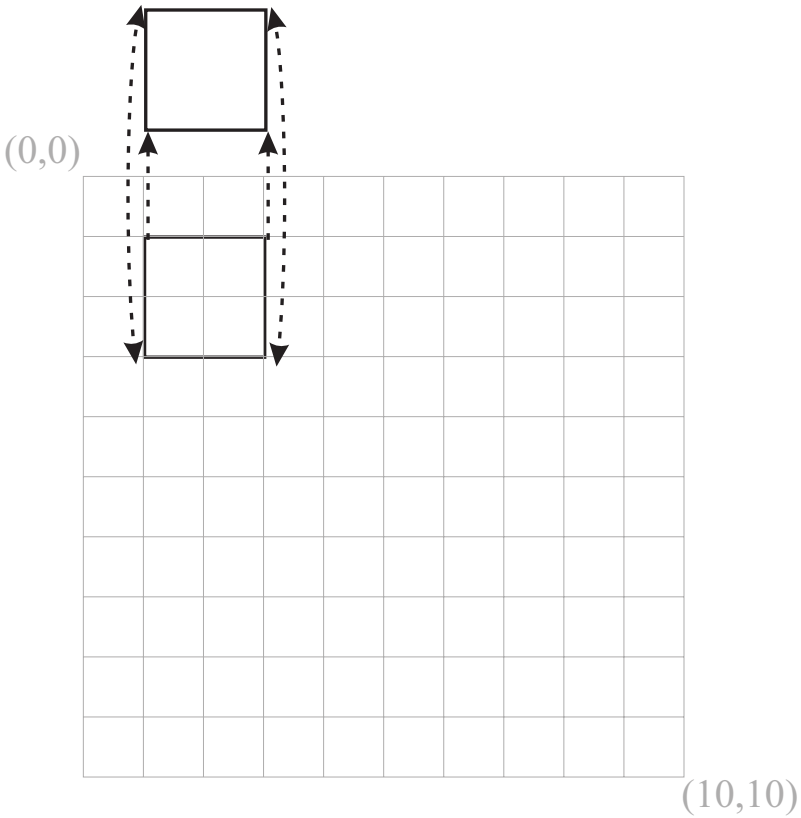


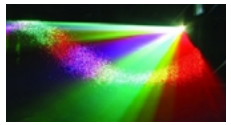
Reflection

```
AffineTransform      aroundX, aroundY;

aroundX = new AffineTransform( 1.0, 0.0, 0.0,-1.0, 0.0, 0.0);
aroundY = new AffineTransform(-1.0, 0.0, 0.0, 1.0, 0.0, 0.0);
```



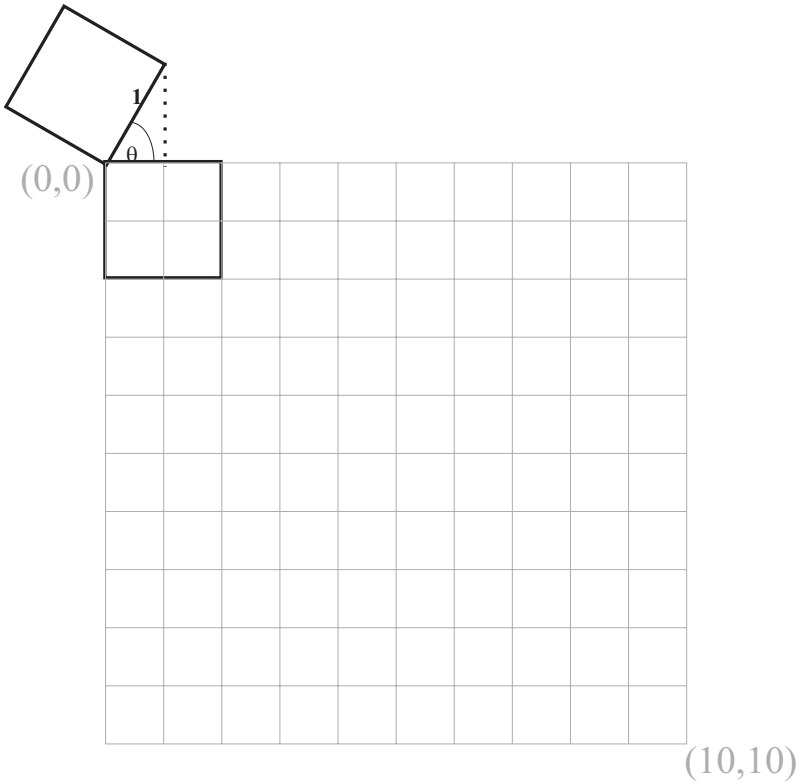
Reflection (cont.)

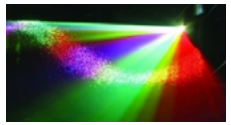




Rotation

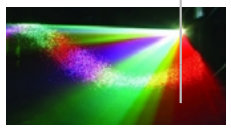
`AffineTransform.getRotateInstance(double radians)` creates a rotational transform for rotating around the point (0,0). In Java, the static method





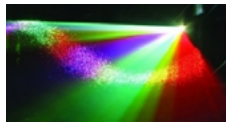
Iteratively Rotating a Complicated Shape

```
protected void paintGlyphs(Graphics2D g2, String text)
{
   AffineTransform         at, trans;
   AlphaComposite          alpha;
   Dimension               d;
   float                   angle, x, y;
   FontRenderContext       frc;
   GlyphVector             glyphs;
   int                     i;
   Shape                   shape, transformedShape;

   d = getSize();

   frc = g2.getFontRenderContext();
   glyphs = font.createGlyphVector(frc, text);
   shape = glyphs.getOutline(0.0f, 100.0f);
   g2.setColor(Color.BLACK);

   for (i=0; i < 6; i++)
   {
      angle = (float)(Math.PI/6.0 * i);
      x = (float)(d.width/2.0);
      y = (float)(d.height/2.0);
      at = AffineTransform.getRotateInstance(angle,x,y);
      trans = AffineTransform.getTranslateInstance(x,y);
      at.concatenate(trans);

      transformedShape = at.createTransformedShape(shape);
```

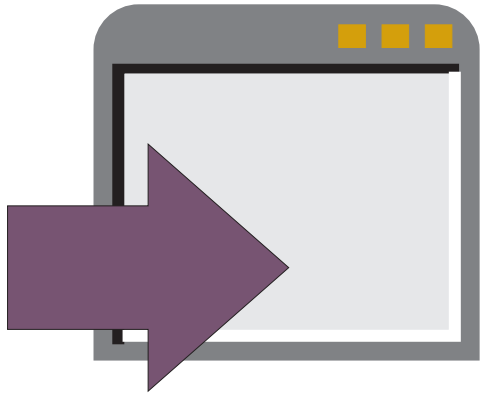


Iteratively Rotating a Complicated Shape (cont.)

```
            g2.fill(transformedShape);
        }
    }
```



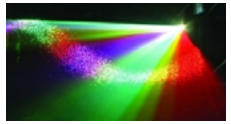
Iteratively Rotating a Shape – Demonstration



In `examples/chapter`:

SpiralGlyph.html

```
java -cp Glyph.jar GlyphApplication SpiralGlyphCanvas Text
```



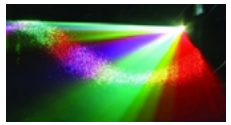
Steps in the Process

JMU

Stroke

JMU

Fill



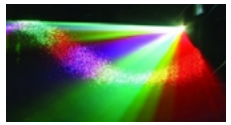
Stroking

Definition

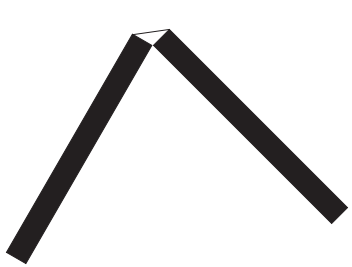
Stroking is the rendering of the border (i.e., outline) of the described content.

`Stroke` Interface

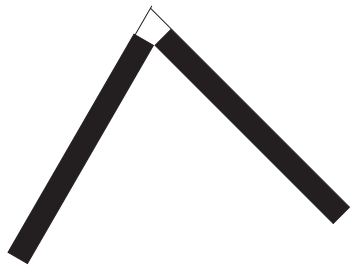
`BasicStroke` Class



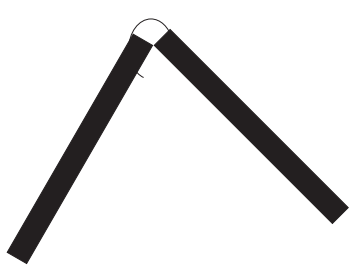
Stroking - Joins



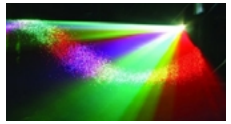
Bevel Join



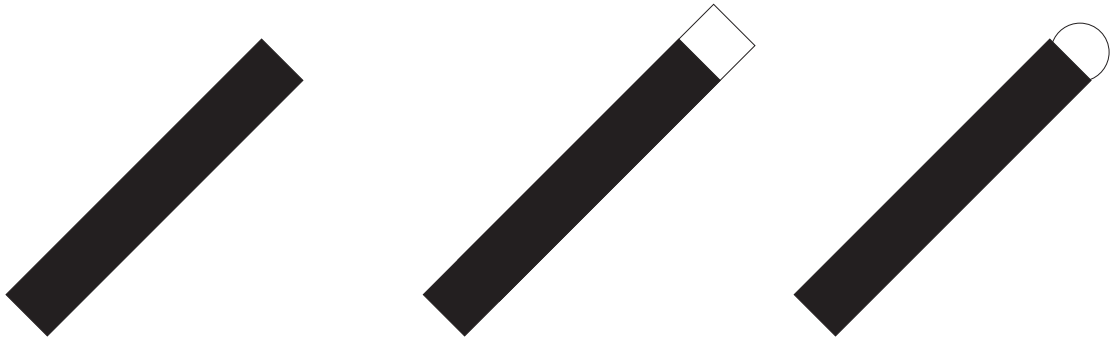
Mitre Join



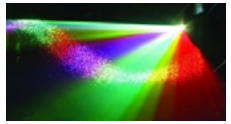
Round Join



Stroking - Caps



Butt Cap Square Cap Round Cap



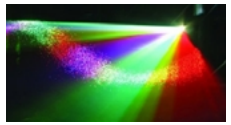
Filling

Definition

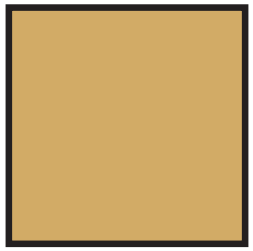
Filling is the rendering of the interior of the content.

`Paint` Interface

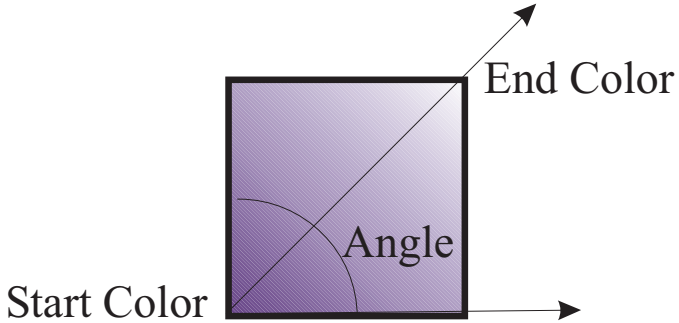
`Color`, `GradientPaint` and `TexturePaint` Classes



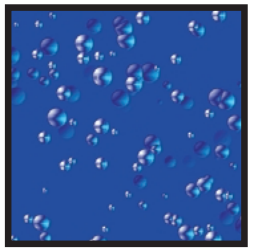
Fill Techniques



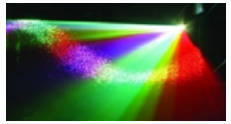
Solid Fill



Gradient Fill

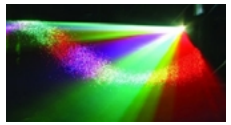


Sampled Fill



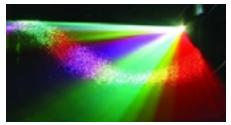
Some Rendering Examples

```
// The first rectangle
rectangle = new Rectangle2D.Double( 10.0,  20.0,
                                    100.0,
                                    150.0);

// Fill in JMU Gold
g2.setPaint(new Color(0xC2, 0xA1, 0x4D));
g2.fill(rectangle);


// Stroke in JMU purple
stroke = new BasicStroke(5.0f,
                         BasicStroke.CAP_BUTT,
                         BasicStroke.JOIN_MITER);
g2.setStroke(stroke);
g2.setColor(new Color(0x45, 0x00, 0x84));
g2.draw(rectangle);
```



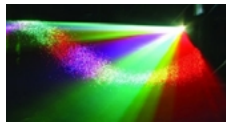
Some Rendering Examples (cont.)

```
// The second rectangle
rectangle = new Rectangle2D.Double( 200.0, 200.0,
                                    100.0,
                                    150.0);

// Fill using a gradient
gradient=new GradientPaint(200.0f, 275.0f, Color.CYAN,
                           300.0f, 275.0f, Color.WHITE);
g2.setPaint(gradient);
g2.fill(rectangle);

// Stroke in black
stroke = new BasicStroke(10.0f,
                         BasicStroke.CAP_BUTT,
                         BasicStroke.JOIN_ROUND);
g2.setStroke(stroke);
g2.setColor(Color.BLACK);
g2.draw(rectangle);
```

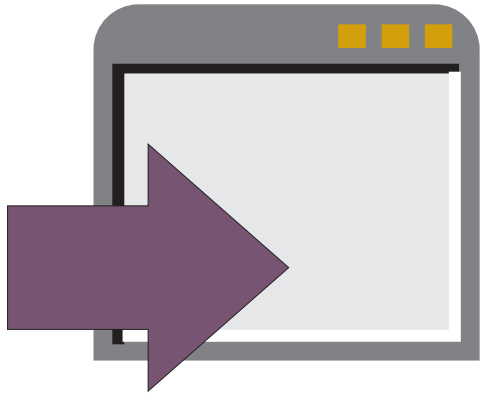


Some Rendering Examples (cont.)

```
// The third rectangle
rectangle = new Rectangle2D.Double( 50.0, 50.0,
                                    200.0,
                                    250.0);

// Use alpha blending to achieve a transparency effect
composite = AlphaComposite.getInstance(
                                AlphaComposite.SRC_OVER,
                                0.5f);
g2.setComposite(composite);

// Fill in gray
g2.setPaint(Color.YELLOW);
g2.fill(rectangle);

// Don't stroke
```



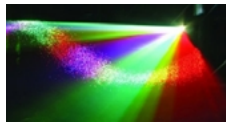
Some Rendering Examples – Demonstration



In `examples/chapter`:

RenderingExampleCanvas.html

```
java -cp RenderingExampleCanvas.jar RenderingExampleCanvasApplication
```



Requirements

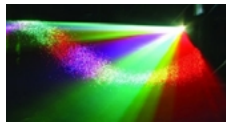
F6.1 Support the addition of described static visual content.

F6.2 Support the removal of described static visual content.

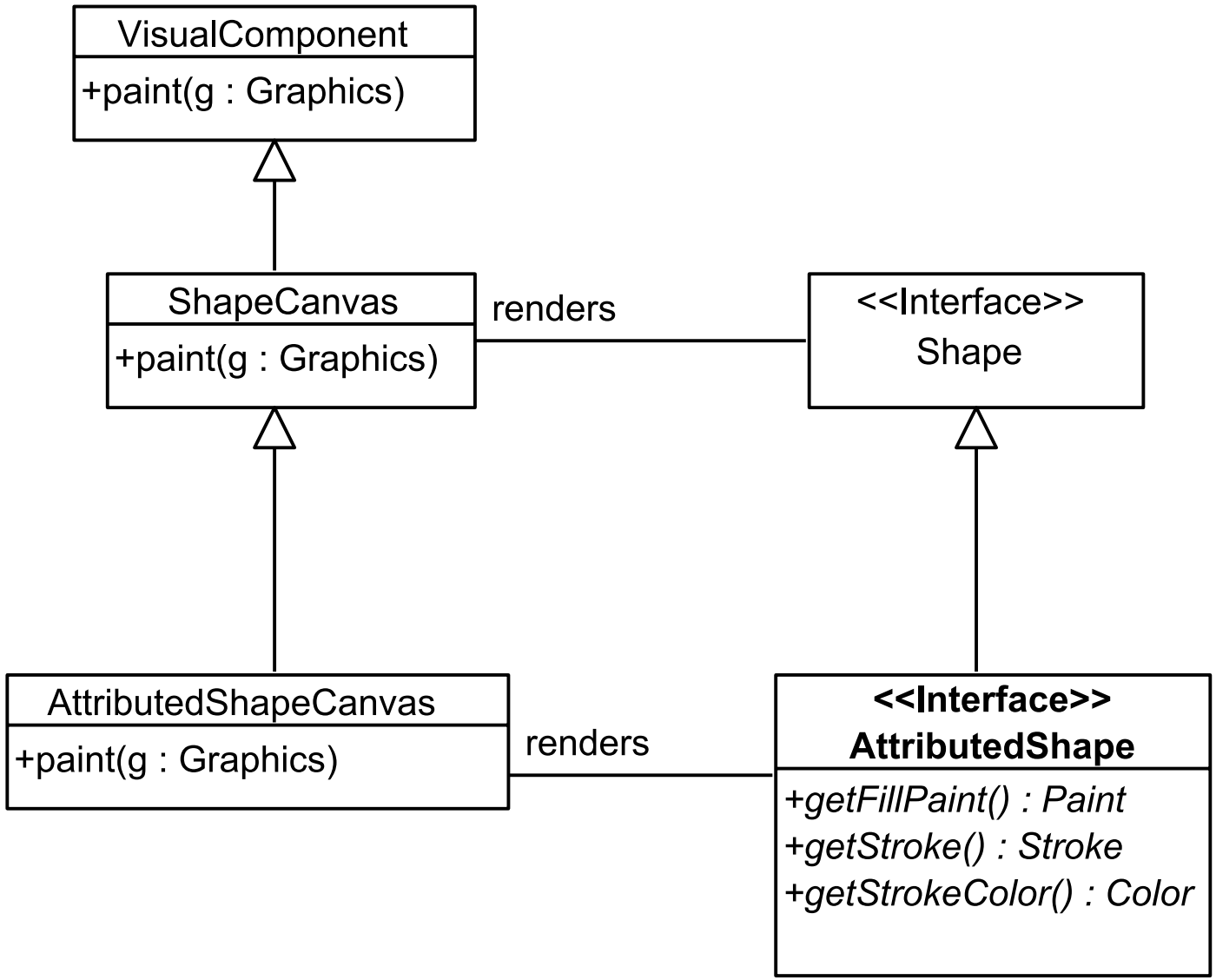
F6.3 Support z-ordering of described static visual content.

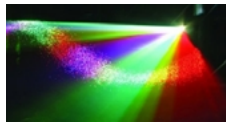
F6.4 Support the transformation of described static visual content.

F6.5 Support the rendering of described static visual content.

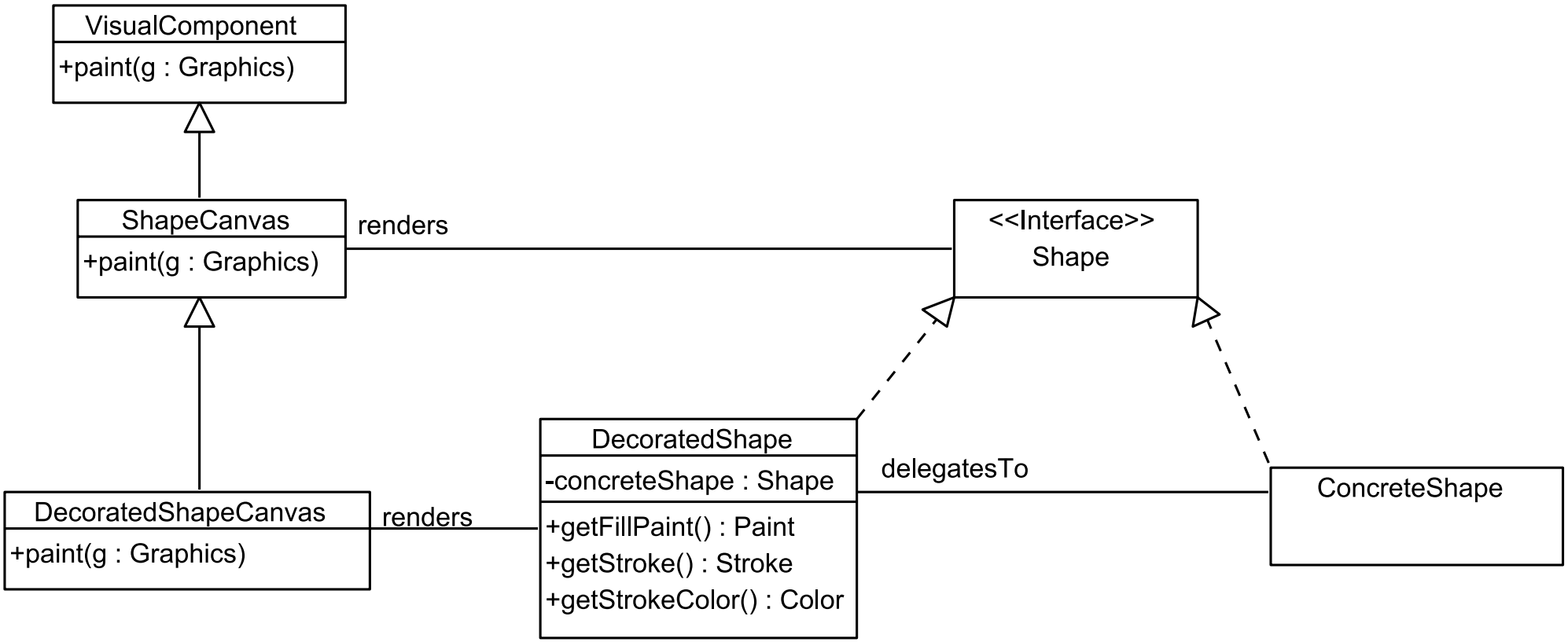


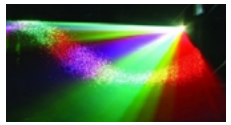
Alternative 1





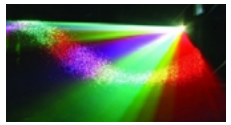
Alternative 2





Shortcomings of Alternatives 1 and 2

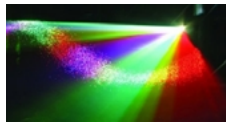
What are the shortcomings?



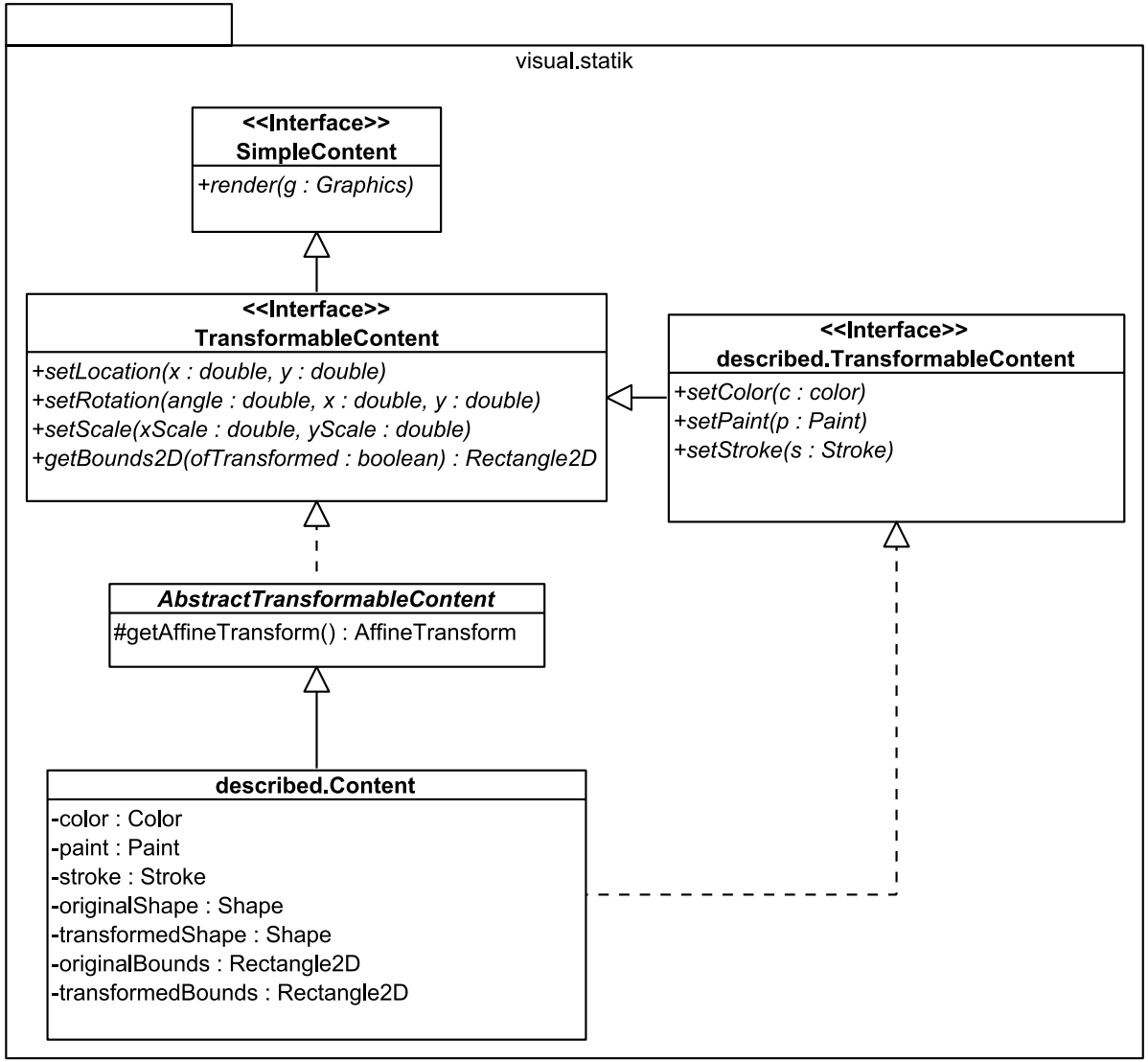
Shortcomings of Alternatives 1 and 2

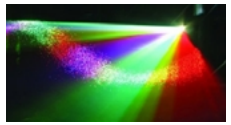
`DecoratedShape` are not cohesive.

The specialization of the `JComponent` class must both manage `Shape` objects and render them.



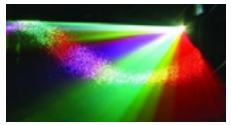
Alternative 3





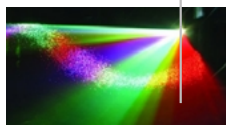
statik.TransformableContent

```
package visual.statik.described;

import java.awt.*;
import java.awt.geom.*;

public interface TransformableContent
       extends   visual.statik.TransformableContent
{
    public abstract void setColor(Color color);

    public abstract void setPaint(Paint paint);

    public abstract void setStroke(Stroke stroke);
}
```



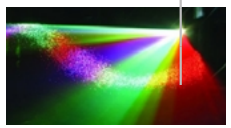
Structure of statik.described.Content

```
package visual.statik.described;

import java.awt.*;
import java.awt.geom.*;

public class Content
             extends visual.statik.AbstractTransformableContent
             implements TransformableContent
{
    private Color                color;
    private Paint                paint;
    private Rectangle2D.Double   originalBounds;
    private Rectangle2D.Double   transformedBounds;
    private Shape                originalShape;
    private Shape                transformedShape;
    private Stroke               stroke;

    private final Stroke          DEFAULT_STROKE =
                                   new BasicStroke();

    private final AffineTransform IDENTITY =
                                   new AffineTransform();

    public Content()
    {
       this(null, null, null, null);
    }
```



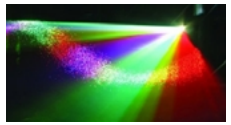
Structure of statik.described.Content (cont.)

```
public Content(Shape  shape,  Color color, Paint paint,
               Stroke stroke)
{
   super();

   setColor(color);
   setPaint(paint);
   setStroke(stroke);

   setShape(shape);
}

private void getBoundsFor(Rectangle2D.Double r, Shape s)
{
   Rectangle2D       temp;

   if (s != null)
   {
      temp = s.getBounds2D();

      if (temp != null)
      {
         r.x      = temp.getX();
         r.y      = temp.getY();
         r.width  = temp.getWidth();
         r.height = temp.getHeight();
      }
   }
```



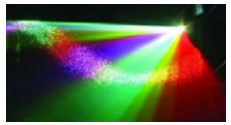
Structure of `statik.described.Content` (cont.)

```
    }
}
```



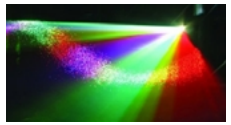
Getters in statik.described.Content

```
public Color getColor()
{
   return color;
}

public Paint getPaint()
{
   return paint;
}

public Stroke getStroke()
{
   return stroke;
}
```



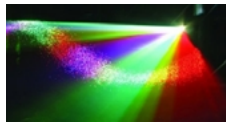
Setters in statik.described.Content

```
public void setColor(Color color)
{
   this.color = color;
}

public void setPaint(Paint paint)
{
   this.paint = paint;
}
```



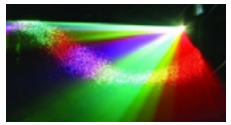
Transformations in statik.described.Content

```
private void createTransformedContent(AffineTransform at)
{
   transformedShape = at.createTransformedShape(
                                          originalShape);
   setTransformationRequired(false);

   getBoundsFor(transformedBounds, transformedShape);
}
```



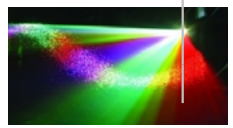
Bounds in statik.described.Content

```
public Rectangle2D getBounds2D(boolean transformed)
{
   if   (transformed) return transformedBounds;
   else               return originalBounds;
}
```



Rendering in statik.described.Content

```
public void render(Graphics g)
{
   Color       oldColor;
   Graphics2D  g2;
   Paint       oldPaint;
   Stroke      oldStroke;

   g2 = (Graphics2D)g;

   // Transform the Shape (if necessary)
   if (isTransformationRequired())
   {
      createTransformedContent();
   }

   if (transformedShape != null)
   {
      // Save the state
      oldColor     = g2.getColor();
      oldPaint     = g2.getPaint();
      oldStroke    = g2.getStroke();


      // Fill the Shape (if appropriate)
      if (paint != null)
      {
         g2.setPaint(paint);
         g2.fill(transformedShape);
```



Rendering in statik.described.Content (cont.)

```
        }

        // Stroke the Shape if appropriate
        if (color != null)
        {
            if (stroke != null) g2.setStroke(stroke);
            g2.setColor(color);
            g2.draw(transformedShape);
        }

        // Restore the state
        g2.setColor(oldColor);
        g2.setPaint(oldPaint);
        g2.setStroke(oldStroke);
    }
}
```