

---

# 2D Graphics

CSE 331

Lecturer: Michael Hotan

slides created by Marty Stepp

based on materials by M. Ernst, S. Reges, D. Notkin, R. Mercer, Wikipedia

<http://www.cs.washington.edu/331/>

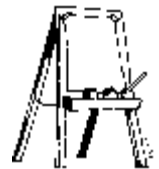
# Custom components

---

- AWT/Swing come with lots of components that you can use to implement a fully-featured GUI.
- But there are cases when you need a custom component.
  - Usually this is when you want to paint custom 2-D graphics.
  - We often call a custom painted component a *canvas*.
- To do so, write a class that extends `JComponent`.
  - Override method `paintComponent` to tell Java how to draw it:

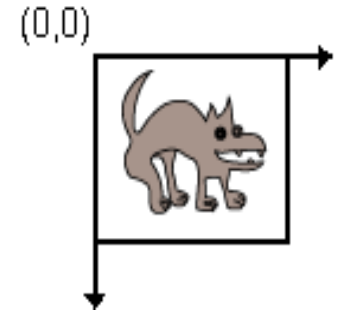
```
public void paintComponent (Graphics g)
```

- Some programmers extend `JPanel` rather than `JComponent`.



# A drawing canvas

- Coordinate system: (0, 0) at top-left, x-axis increases rightward, y-axis **downward**.
- Component's surface is *transparent* unless drawn on.
- JComponent's `paintComponent` does important things that we don't want to lose. (e.g. paints the component's background)
  - So call the method `super.paintComponent` first thing.



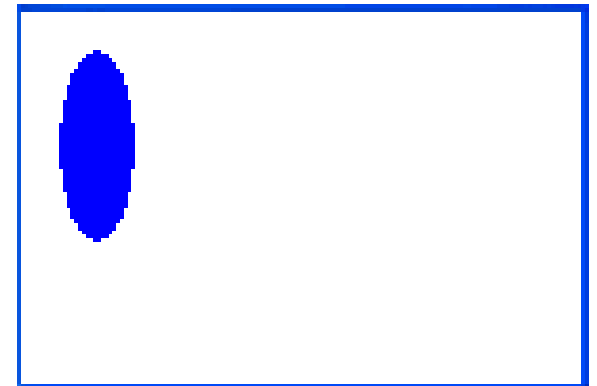
```
public void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    ...  
}
```

# Quick drawing example

---

```
public class MyCanvas extends JComponent {
    public MyCanvas() {
        this.setBackground(Color.WHITE);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setPaint(Color.BLUE);
        g.fillOval(10, 10, 20, 50);
    }
}
```



# Graphics methods



Method name	Description
<code>drawImage (Image, x, y, [w, h], panel)</code>	an image at the given x/y position and size
<code>drawLine (x1, y1, x2, y2)</code>	line between points (x1, y1), (x2, y2)
<code>drawOval (x, y, width, height)</code>	outline largest oval that fits in a box of size <i>width</i> * <i>height</i> with top-left at (x, y)
<code>drawRect (x, y, width, height)</code>	outline of rectangle of size <i>width</i> * <i>height</i> with top-left at (x, y)
<code>drawString (text, x, y)</code>	text with bottom-left at (x, y)
<code>fillOval (x, y, width, height)</code>	fill largest oval that fits in a box of size <i>width</i> * <i>height</i> with top-left at (x, y)
<code>fillRect (x, y, width, height)</code>	fill rectangle of size <i>width</i> * <i>height</i> with top-left at (x, y)
<code>setColor (color)</code>	paint any following shapes in the given color
<code>setFont (font)</code>	draw any following text with the given font

# Graphics2D

---

- The `Graphics` object `g` passed to `paintComponent` is a "graphical context" object to draw on the component.

- The actual object passed in is a `Graphics2D` (can cast).

```
Graphics2D g2 = (Graphics2D) g;
```



- `Graphics2D` is a subclass of `Graphics` that adds new features, new shapes, matrix transformations, color gradients, etc.
  - Added to Java in v1.2 to improve on the features of `Graphics`.
  - Why didn't they just add the new methods and features to `Graphics` directly? Why did they bother to make it a separate class?
    - *Answer:* Open-Closed Principle. `Graphics` already worked just fine. Why risk breaking it by adding new features to the same file?

# Graphics2D methods

---

method name	description
<code>draw (Shape)</code>	draws the outline of a given shape object ( <i>replaces drawRect, etc.</i> )
<code>fill (Shape)</code>	draws the outline and interior of a given shape object
<code>getPaint () ,</code> <code>setPaint (Paint)</code>	returns or sets the current paint used for drawing (Color is one kind of Paint, but there are others)
<code>getStroke () ,</code> <code>setStroke (Stroke)</code>	returns or sets the current line stroke style used for drawing (can be thin/thick, solid/dashed/dotted, etc.)
<code>rotate (angle)</code>	rotates any future drawn shapes by the given angle (radians)
<code>scale (sx, sy)</code>	resizes any future drawn shapes by the given x/y factors
<code>translate (dx, dy)</code>	moves any future drawn shapes by the given x/y amounts
<code>setRenderingHint (</code> <code>key, value)</code>	sets "rendering hints" such as anti-aliasing and smoothing
<code>shear (shx, shy)</code>	gives a slanted perspective to future drawn shapes
<code>transform (t)</code>	adds a transformation that will be applied to all shapes

# Shapes (java.awt.geom)

---

- `Arc2D.Double(x, y, w, h, start, extent, type)`

An arc, which is a portion of an ellipse.



- `Ellipse2D.Double(x, y, w, h)`



- `Line2D.Double(x1, y1, x2, y2)`

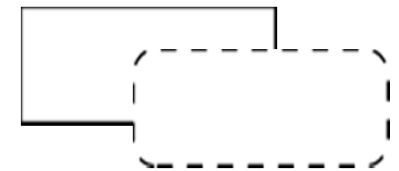
`Line2D.Double(p1, p2)`

A line between two points.



- `Rectangle2D.Double(x, y, w, h)`

- `RoundRectangle2D.Double(x, y, w, h, arcx, arcy)`



- `GeneralPath()`

A customizable polygon.





# Methods of all shapes

---

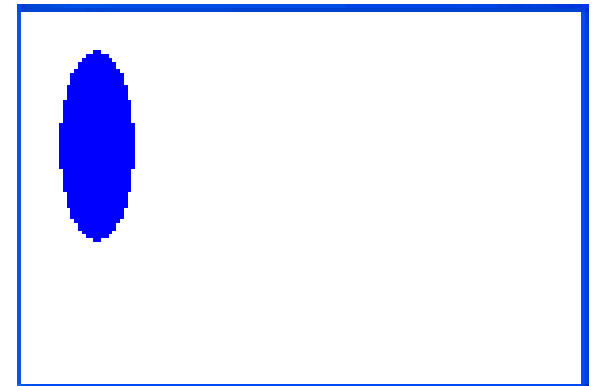
method name	description
<code>contains (x, y)</code> <code>contains (point)</code> <code>contains (rectangle)</code>	whether the given point is inside the bounds of this shape
<code>getBounds ()</code>	a rectangle representing the bounding box around this shape
<code>getCenterX/Y ()</code> <code>getMinX/Y ()</code> <code>getMaxX/Y ()</code>	various corner or center coordinates within the shape
<code>intersects (x, y, w, h)</code> <code>intersects (rectangle)</code>	whether this shape touches the given rectangular region

# Drawing with objects

---

```
public class MyCanvas extends JComponent {
    public MyCanvas() {
        this.setBackground(Color.WHITE);
    }

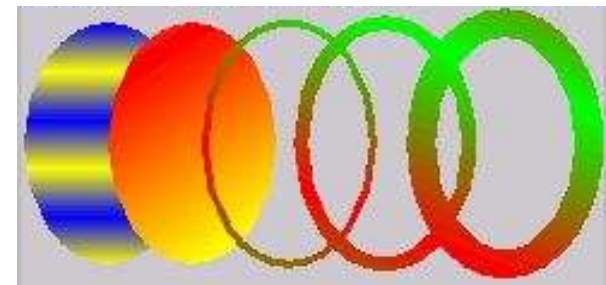
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        Shape shape = new Ellipse2D.Double(10, 10, 20, 50);
        g2.setPaint(Color.BLUE);
        g2.fill(shape);
    }
}
```



# Colors and paints

---

- **Color** (a simple single-colored paint)
  - `Color.RED`
  - `public Color(int r, int g, int b)`
  - `public Color(int r, int g, int b, int alpha)`
    - a partially-transparent color (range 0-255, 0=transparent)
- **GradientPaint** (a smooth transition between 2 colors)
  - `public GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)`
- `java.awt.TexturePaint`  
(use an image as a "paint" background)



# Strokes (pen styles)

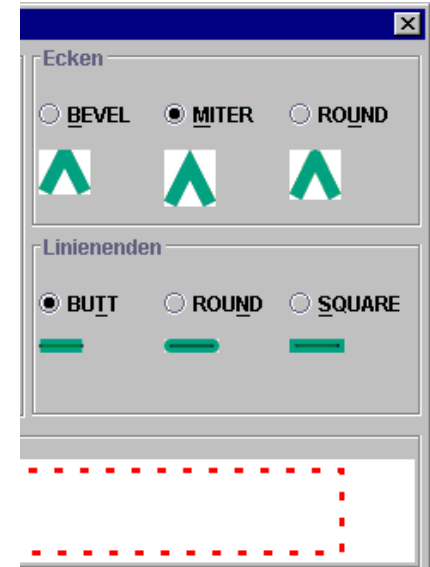
## Graphics2D

- `public void setStroke(Stroke s)`  
Sets type of drawing pen (color, width, style) that will be used by this Graphics2D.

- **BasicStroke**

*A pen stroke for drawing outlines.*

- `public BasicStroke(float width)`
- `public BasicStroke(float width, int cap, int join)`
- `public BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)`
  - cap can be: CAP\_BUTT, CAP\_ROUND, CAP\_SQUARE
  - join can be: JOIN\_BEVEL, JOIN\_MITER, JOIN\_ROUND



# Repainting

---

- Most canvases are drawing the state of fields, a model, etc.
  - When the state updates, you must tell the canvas to re-draw itself.
  - But you can't call its `paintComponent` method, because you don't have the `Graphics g` to pass.
  - The proper way is to call `repaint` on the canvas instead:

```
public void repaint()
```

```
...
```

```
public void update(Observable o, Object arg) {  
    myView.repaint(); // perhaps this.repaint();  
}
```

# Anti-aliasing

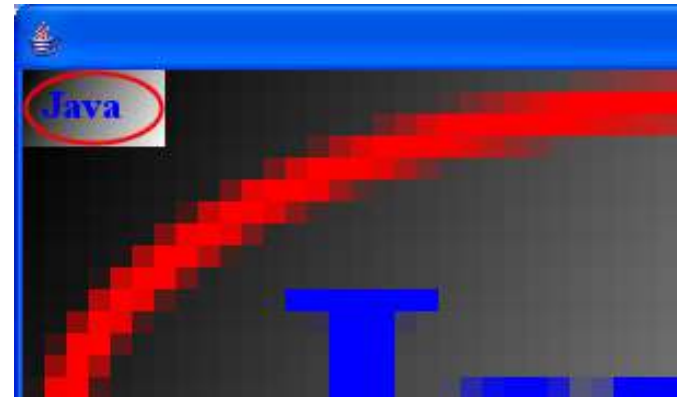
---

- Onscreen text and shapes can have jagged edges, or *aliases*. These can be removed by smoothing, or *anti-aliasing*, the component.

- `public void setRenderingHint(key, value)`

- **Example:**

```
g2.setRenderingHint(  
    RenderingHints.KEY_ANTIALIASING,  
    RenderingHints.VALUE_ANTIALIAS_ON);
```



# Creating images

---

```
// import java.awt.image.*;
```

BufferedImage

*A blank graphic image buffer surface onto which you can draw*

- `public BufferedImage(int w, int h, int type)`
  - where `type` is a constant such as `BufferedImage.TYPE_INT_ARGB`
- `public Graphics getGraphics()`
  - returns a graphical pen for "drawing on" this image
- you can draw a `BufferedImage` onto the screen from within the `paintComponent` method of your canvas:
  - `g.drawImage(BufferedImage, x, y, this);`

# Upload Images

---

- ImageIO : class that makes it convenient to upload images.

*Public static BufferedImage read (File input) throws IOException*

*Public static BufferedImage read(URL input) throws IOException*



# Example

---

- FunImageDisplayer.java
- Graph Application