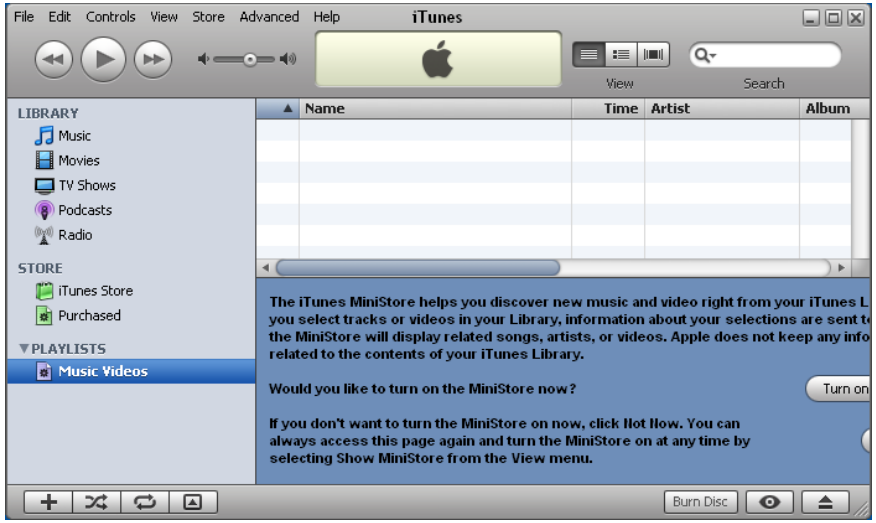
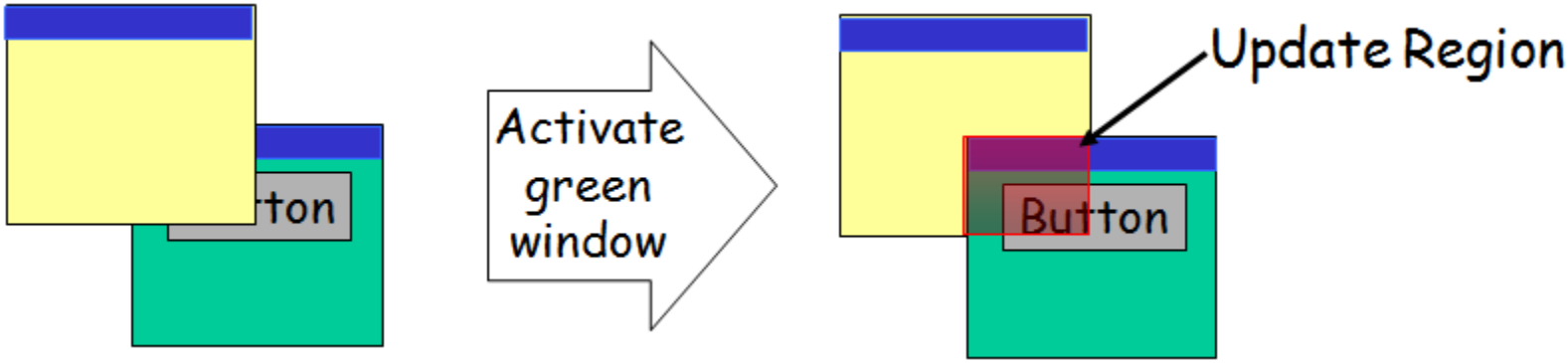
# Custom Widgets and Drawing



# Recap: Rendering of Widgets

Widgets have a visual representation

- Widgets define "paint" event listener: draws the widget by sending commands to the windowing system
- Widget gets paint events (aka. "update events") from the windowing system (through GUI framework)
- Often not complete redrawing, but "update region"
- Application can send "invalidate" events to the windowing system if redrawing necessary (to trigger paint events)



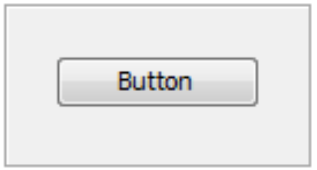
# How Swing Widgets are Painted

All Swing widgets inherit from **`JComponent`**

- **`JComponent`** defines **`paint(Graphics g)`**
- **`paint()`** called by the system whenever drawing is necessary

**`paint()`** calls other methods of **`JComponent`**

- **`paintComponent()`** paints the widget (override this!)
- **`paintBorder()`** paints a border the widget may have (see **`border`** property)
- **`paintChildren()`** paints the children of the widget, if it is a container (don't override this!)




You never call **`paint()`** directly

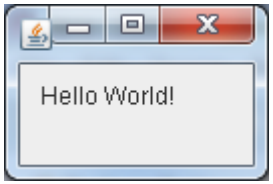
- Instead invalidate the widget region by calling **`repaint()`**
- **`repaint()`** asynchronously calls **`paint()`** (through windowing system)
- You can give dirty region as argument: **`repaint(Rectangle r)`**

# Creating a Custom Widget

1. Create new class that extends **`JPanel`**
2. Override **`paintComponent(Graphics g)`** with custom drawing code
   - Make sure to honor the **`width`** and **`height`** of the widget
   - Possibly call **`super.paint(g)`** to draw the superclass widget (e.g. unicolored background)
3. Override **`getPreferredSize()`** to return the right preferred size for your widget

```
class MyPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello World!",10,20);
    }

    public Dimension getPreferredSize() {
        return new Dimension(100,50);
}   }
```



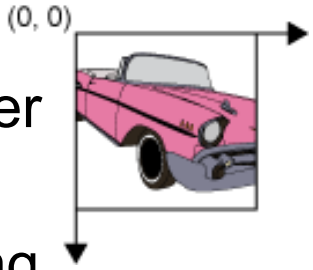
# Drawing with Java

Always draw in a graphics context (**`Graphics / Graphics2D`**):

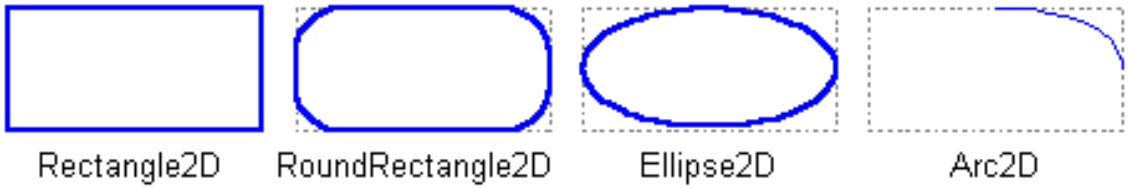
- Uniform way to draw on different devices, like a universal canvas
- Properties of the current pen used for drawing
  - **`color`**, **`background`**, **`font`**
  - **`stroke`**, i.e. pen size and shape
  - **`paint`**, i.e. a color pattern to use
  - **`composite`** type, i.e. how it looks when shapes are drawn onto existing shapes (e.g. blending them together)
  - **`clip`**ping rectangle to limit painting area

- Drawing methods, e.g. **`draw(Shape)`**, **`fill(Shape)`**, **`drawString()`**, **`drawImage()`**
- Transformation methods to apply to the drawing operations, e.g. **`scale()`**, **`rotate()`**, **`translate()`**

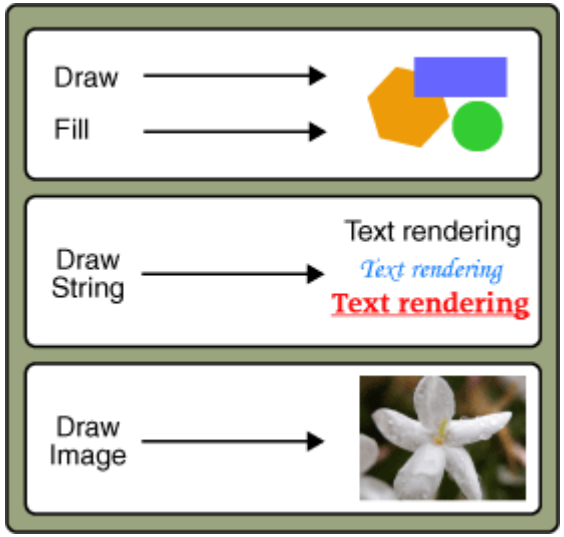
# Drawing Basics




- By default, coordinate space origin in the top-left corner
- Shapes and lines represented as classes implementing interface **`Shape`**




  - **`draw(Shape s)`** draws shape outline using **`color`** and **`stroke`**
  - **`fill(Shape s)`** draws solid shape using **`color`** / **`paint`**




- **`drawString(String s, float x, float y)`** draws text using **`font`**
- **`drawImage(Image i, int x, int y, ...)`**

http://docs.oracle.com/javase/tutorial/2d/overview/index.html

# RoundedButton Part 1


```
class RoundedButton extends JButton {
  public void paintComponent(Graphics g) {
    // Argument of paint() is actually a Graphics2D object,
    // which has more functionality than Graphics
    Graphics2D g2 = (Graphics2D) g;

    // Switch on anti-aliasing, which looks better
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
            RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
```

Without Anti-Aliasing:


With Anti-Aliasing:


```
    g2.setColor(getBackground());
    g2.fill(new Rectangle2D.Float(
          0, 0, getWidth(), getHeight()));
```

```
    g2.setColor(new Color(110, 120, 210));
    g2.fill(new RoundRectangle2D.Float(
          0, 0, getWidth(), getHeight(), 50, 50));
```

```
    ...
```

# RoundedButton Part 2

```
    g2.setColor(new Color(120, 130, 255));
    g2.setStroke(new BasicStroke(5));
    g2.draw(new RoundRectangle2D.Float(
        2, 2, getWidth() - 4, getHeight() - 4, 50, 50));
    g2.setStroke(new BasicStroke(1));

    FontMetrics metrics = g2.getFontMetrics(getFont());
    int h = metrics.getAscent();
    int w = metrics.stringWidth(getText());

    g2.setColor(getForeground());
    g2.drawString(getText(),
        (getWidth() - w) / 2, (getHeight() + h) / 2);
  }

  public static void main(String[] args) {
    JFrame frame = new JFrame();
    RoundedButton r = new RoundedButton(); r.setText
("Hello!");
    r.setFont(new Font("Comic Sans MS", Font.PLAIN, 16));
    frame.getContentPane().add(r);
    frame.pack(); frame.setVisible(true);
}   }
```


# Summary

- **Drawing** can be performed using graphics objects
  - A graphics context (**`Graphics2D`**)
  - **`Stroke`**s, **`Font`**s, **`Color`**s...
  - **`Shape`** objects that can be **`draw`**n or **`fill`**ed
- **Custom components** can be created by overriding the method paintComponent(Graphics g) of a widget

References:


- The Java Tutorials: 2D Graphics. http://docs.oracle.com/javase/tutorial/2d/
- The Java Tutorials: Performing custom painting. http://docs.oracle.com/javase/tutorial/uiswing/painting/