Graphic Interface Programming II

Some First Graphics Tricks
Advanced Graphics

- NOTE!
- Some advanced Graphics features are not available on all platforms
- One such feature is transparency
  - Some kinds may still work!

- It is necessary to test for this property when we implement more fancy features
From Simple to...

- Need to know more about the basic drawing mechanisms in Swing

- From the Basics
  - To the Skies
    - In small small steps
Window drawing

- `paintComponent()` draws everything
  - Decorations
    - Borders
    - Title Bars

- `setUndecorated(true);`
  - By default Frames are decorated
  - If undecorated the titlebar disappears!
  - Moving the window becomes difficult…
Graphics(2D)

- Most of the time Swing uses 2D graphics
  - (even when we have a Graphics g component)

- We never create a Graphics object
  - We get it in the call to paintComponent()

- We manipulate the given Graphics object in the method
2D Rendering

1. Get a Graphics (Graphics2D) object

2. Set attributes on the Graphics object

3. Draw graphics primitives with the Graphics object

4. Repeat 2 and 3 until satisfied
Example

```java
Protected void paintComponent(Graphics g) {
    g.setColor(Color.RED);
    g.fillRect(0,0,width, height);
    g.setColor(Color.BLUE);
    g.fillOval(10,20,width/2, height/2);
}
```

- Items are drawn in execution order from the bottom up!
- Last item is "on top"
  - The Rectangle will be seen as "behind" the Oval.
Mixing Components and Paint

- Difficult, they don’t form "mixable" layers
setOpaque()?

- JComponent.setOpaque( boolean ) is misnamed
- Does not do transparency!!!
- Collaborates with super.paintComponent

- It really should be called JComponent.wipeBackground( boolean )
Simple Transparent Button

class TransparentButton extends JButton {
    public TransparentButton(String text) {
        super(text);
        setOpaque(false);
    }

    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setComposite(
            AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f)
        );
        super.paintComponent(g2);  // After changing!!!
        g2.dispose();
    }
}
Transparent PopupMenu

class TransparentPopupMenu extends JPopupMenu {
    public TransparentPopupMenu() {
        super();
        setOpaque(false);
    }
}
class TransparentMenuItem extends JMenuItem {
    public TransparentMenuItem(String text) {
        super(text);
        setOpaque(false);
    }
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f));
        super.paint(g2);
        g2.dispose();
    }
}
setComposite(Composite c)

- Specifies HOW:
  - the new pixels
  - are combined with
  - the existing pixels
  - on the graphics device (e.g. Graphics g)

- E.g. As more transparent colors
  - Don’t have to work on the colors directly!

- Composite is an interface!
Color

- Colors can also be transparent
- Color stores a color in 32 bits
  - 8 bits for each color
  - Bits $8 \times 3 = 24$; 8 bits missing:

- Alpha channels
  - Values 0 to 255 define transparency of the colour
Example

Color almostTransparentDarkOrange =
    new Color( 0xff /* red */, 0x8c /* green */, 0x00 /* blue */, 0x1f /* alpha */ );

- #ff8c00 = DarkOrange as a colour code
Dynamic!

\[ \text{AlphaComposite.getInstance(AlphaComposite.SRC\_OVER, 0.5f)} \]

- Note that this can be changed during runtime
  - Since it is set in the paintComponent method
- New Colors can also be defined in paintComponent
  - But that creates a lot of instances (if we use loops and many colours)
Experiment!