CSC420

Swing Painting and Graphics

Painting

- The process by which the app updates the display
- Can involve:
  - Swing internal code for repainting standard components
  - Some of your code, if you have custom painting
- Originates in one of two ways
  - Swing/AWT libs post a repaint request
  - App code posts such a request
- Posting a paint/repaint request is different from custom painting!

Painting

- In general, happens automatically
  - Swing detects changes and issues repaint requests
- What happens of Swing fails to do it?
  - Example: a change to an internal property controlling translucency (as opposed to something that is part of a component's data such as the text of a label)
- Two categories of app-controlled paint requests

Asynchronous

- Tell Swing what needs to be updated
- Let Swing handle the scheduling on the EDT
- All of these are variants of Component.repaint()
- Component.repaint()
  - Swing repaints the entire component
  - Important: repaint requests get coalesced! (only one can be in the EDT)
  - Downside: overhead (entire component for a small change)
- Component.repaint(int x, int y, int width, int height)
  - Repaint a rectangle in the component
  - Again, coalesced
  - repaint() == repaint(0, 0, getWidth(), getHeight())

Synchronous

- Execute immediately, in the current thread
- Careful: <u>must</u> be on EDT!
- Jcomponent.paintImmediately(int x, int y, int w, int h)
  - Does not coalesce (i.e., overhead)
- Component.paint(Graphics g)
  - Execute if you want to render a component to an image (or another non-standard Graphics object

Swing Rendering

- A paint request goes on the event queue
- Sometime later the EDT dispatches it to the Swing RepaintManager object
- That object calls paint() on the component
- The component paints first its own content, then its border, then recurses the call to its children
- Painter's algorithm
- Jcomponent.paintComponent(Graphics)
- Component.paint(Graphics)
- JComponent.setOpaque(boolean)

paintComponent()

- Override for most kinds of custom painting
- Careful: you are on the EDT!
- Example:

```
public class OvalComponent extends JComponent {

  public void paintComponent(Graphics g) {
    g.setColor(getBackground());
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(Color.GRAY);
    g.fillOval(0, 0, getWidth(), getHeight());
  }
```

Overriding paint()

- Not a good idea, because might forget to call all necessary functionality
- Sometimes is unavoidable, however
- Example: changing the Composite attribute of a Graphics object to achieve translucency

Opacity and setOpaque()

- Java2D opacity != Swing opacity
- Java2D opacity:
  - Rendering concept
  - Combination of an alpha value and a Composite mode
  - Describes the degree of blending (half-translucent = half existing color + half new color)
- Swing opacity:
  - Refers to visibility
  - Anything rectangular and non-translucent (Java2D sense) is opaque
  - A translucent button is not Swing-opaque
  - A rounded button is not Swing-opaque
  - Reason: performance (Painter's algorithm is very slow)

Double-buffering

- Finally true double-buffering in Java 6 Swing
- Techique for reducing flicker
- Uses an off-screen image to render screen contents (called a back buffer)
- At appropriate times, the back buffer is copied to the screen (in a single operation)
- Swing benefits also because of its rendering pipeline (opacity) – otherwise, rendering artifacts on screen