Java Graphics

* and an unrelated bit about anonymous classes

Krysta Yousoufian

CSE 331 Section, 3/1/2012

With material from Marty Stepp

Custom Graphics

- Sometimes you need to draw custom graphics in your GUI
  - Displaying an image
  - Drawing geometric shapes and lines
- For this, you need a custom component
  - Often called a *canvas* (not to be confused with the `Canvas` class)
  - Override `paintComponent` to tell Java how to render it

Creating a Canvas

- Write a class that extends JComponent
- Override its paintComponent method

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

- In paintComponent:
  - First, call super.paintComponent
  - Then, call `Graphics` methods to draw what you want
  - (Actually, often want `Graphics2D` … more later)

Graphics methods

- drawImage
- drawLine
- drawOval
- drawRect
- setColor
- etc…

http://docs.oracle.com/javase/6/docs/api/java/awt/Graphics.htm

Example

- `SimpleCanvas.java`

Repainting

- Want to redraw the canvas in response to user input
- Can't call paintComponent() without a reference to its graphics object
- Instead, call the canvas's built-in repaint() method
  - Internally calls paintComponent()

Graphics2D

- `Graphics2D`: subclass of `Graphics`
- More powerful
- `Graphics` objects in your canvas are really `Graphics2D` objects
- Simply cast `Graphics` object to `Graphics2D`:

```
public void paintComponent(Graphics g)
{
      Graphics2D g2d = (Graphics2D)g;
```

Graphics2D methods

- http://docs.oracle.com/javase/6/docs/api/java/awt/Graphics2D.html

Drawing images

- Use the `drawImage` method in `Graphics`
- Load the image into an Image object:

```
Image img =
    Toolkit.getDefaultToolkit()
    .getImage(IMAGE_PATH);
```

- Pass Image object into Graphics.drawImage:

```
g.drawImage(img, …)
```

Example

- `ImageCanvas.java`

And now, for something completely different…

ANONYMOUS CLASSES

Motivation

- Need a small, single-use class to pass into a method
  - Usually class has one short method
  - addActionListener(ActionListener listener)
- Why not write an ordinary inner class?
  - Less readable - separates action from where it's used
  - Clutters up top-level class

Implementation

- Where you would normally put a reference to a variable, you write:

```
new SomeClassName() {
    public void someMethod() {
          // your implementation here
    }
}
```

where SomeClassName is an abstract class or interface to extend/implement

Example

- `Timer` **takes a** `TimerTask` **to schedule:**

```
public void schedule(TimerTask task,
    long delay)
```

```
timer.schedule(new TimerTask() {
    public void run() {
        System.out.println(“Time’s up!”);
    }
}, 1000);
```

Caveats

- Better or worse than regular inner classes? It depends
- Anonymous classes can make code cleaner and easier to follow
- Or they can have the opposite effect
- Good for classes which are…
  - Very small (only a few lines, usually one method)
  - Only used once in the program
- Bad for…
  - Classes of any length (i.e. most classes)
  - Classes for which an object is constructed more than once (need to redefine anonymous class every time)

Demo

- AnonClassGUI.java