GUIs, Graphics

Sara Sprenkle

July 11, 2006

Announcements

- Project 1
  - Due Thursday
  - Sign up for demos in CPM
  - Test plan
    - Sketch of “interesting” test cases/scenarios for demo
- No class next Tuesday (July 18)
  - Allow time for demos
  - Work on assignment 4
- Quiz

GUI Review

- Many Swing components
  - Don't memorize these components or APIs
  - Get to know them from online documentation
  - See example programs
- Implement/extend appropriate event handlers

Swing: Look and Feel

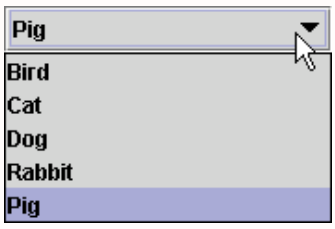
- Graphics in Swing are highly customizable
  - Swing's "theme" or "skin" is called a look and feel
- Possible but difficult to create your own look and feel
  - Very useful for building UI for the disabled
- A look and feel is identified by a class
  - Provides definitions of defaults, such as colors and fonts
  - e.g., javax.swing.plaf.metal.MetalLookAndFeel

Swing: Look and Feel: UIManager

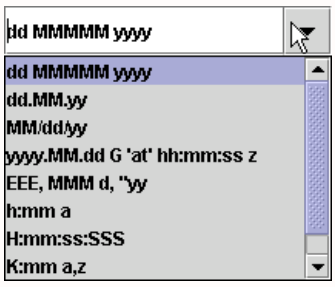
- Handles Look and Feel information
- UIManager.getInstalledLookAndFeels();
  - Find available, installed look and feels
  - returns an array of UIManager.LookAndFeelInfo objects
- UIManager.LookAndFeelInfo class
  - String getName()
  - String getClassName()
- UIManager.setLookAndFeel ("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
- Installed on my Mac
  - LookAndFeel.java

JComboBox

- Allows user to choose among several options
- Two different forms
  - Uneditable
  - Editable



ComboBoxDemo.java



ComboBoxDemo2.java

MVC: Swing Components

- Loosely based on MVC
  - Separate the data (model) from view/controller
- Each Swing component can be given a custom model
  - Model object must implement an interface for the model
- The model for JComboBox is ComboBoxModel
  - Constructor: JComboBox(ComboBoxModel amodel)
  - Default implementation of model used if not specified

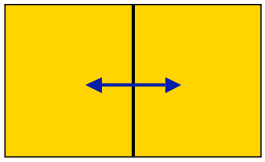
MVC: Swing Components

- Model
  - The data
- Views
  - Graphics
  - Look and Feel
- Controllers
  - Handle/process events
    - Event **model** is already written
  - Listeners
  - Call repaint (repaints the views)

Swing Layouts

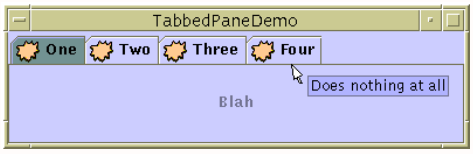
JSplitPane

- A container with a top/bottom or left/right and a draggable separator
- Pass the orientation to the constructor with the constants HORIZONTAL_SPLIT and VERTICAL_SPLIT
  - Alternatively, can specify the two components in the constructor (left/top comes first)
- Example get/set methods:
  - **setLeftComponent(Component)** or **setTopComponent(Component)**
- Can collapse components if setOneTouchExpandable is true



SplitPaneDividerDemo.java

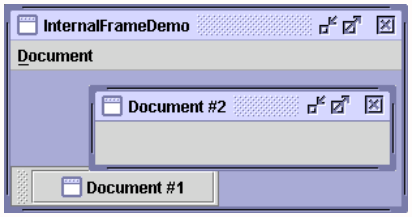
JTabbedPane



- A component with tabs
  - allow the user to switch between a group of components by clicking on a tab with a given title and/or icon
- Several components share the same space
  - In AWT, you'd use a CardLayout
- Each tab has an associated index
- Special-purpose add methods:
  - void addTab(String title, Component component)
  - void addTab(String title, Icon icon, Component component)
  - void addTab(String title, Icon icon, Component component, String tooltip)

TabbedPaneDemo.java

JDesktopPane



- Emulate desktop within a frame using internal frames
  - multiple-document interface
  - virtual desktop

JDesktopPane

- Set the content pane of your root window to a desktop pane

```
JFrame rootFrame = new JFrame("Root Frame");
JDesktopPane desktop = new JDesktopPane();
rootFrame.setContentPane(desktop);
JInternalFrame internalFrame = new
  JInternalFrame("Internal Frame");
desktop.add(internalFrame);
```

- Internal frames: invisible and small by default
- They do not use WindowListener and WindowEvent, they use InternalFrameListener and InternalFrameEvent in javax.swing.event

InternalFrameDemo.java

Dialogs

JOptionPane

- Quick user input
- Input a single string
  - String inputValue = JOptionPane.showInputDialog("Please input a value");
- Show a message
  - JOptionPane.showMessageDialog(null, "The program crashed.");
- Confirm something
  - int option = JOptionPane.showConfirmDialog(null, "Are you sure?");
  - The option can be checked with the constants NO_OPTION and YES_OPTION

JOptionPane

- Normally, the enclosing container is provided as the first argument
  - If null, use default frame
- Display methods have overloaded versions that allow you to choose the icon displayed (question icon, exclamation icon, etc.)
- Used in FreecellDemo.java

JOptionPaneDemo.java

JFileChooser

- Dialog for navigating file system
- Specify the starting directory in constructor as String or File
  - Or use setCurrentDirectory(File)
- Call int showSaveDialog(Component parent) or int showOpenDialog(Component parent)
  - Returns a status code: JFileChooser.APPROVE_OPTION, CANCEL_OPTION, or ERROR_OPTION)

JFileChooser.java

JColorChooser

- Color dialog
  - Actually, a component
- What layouts/components do you think the JColorChooser uses?
- For simplicity, can just call the static method
  - Color showDialog(Component parent, String title, Color initialColor)

ColorChooserDemo.java

Assignment 4

- Creating a GUI from scratch
  - add a GUI to the media library
- Due Thursday, July 20
- Execution: Screen shots
  - 6-10 screen shots of your GUI
  - Demonstrate the GUI and how you tested

Graphics Programming

Graphics Programming Overview

- Big draw to Java early on
- In AWT
  - extend the **Canvas** class
  - override the paint(Graphics g) method
- In Swing
  - extend the **JPanel** class
  - override the paintComponent(Graphics g) method
- The **Graphics** class allows you to draw lines, rectangles, etc. with methods such as drawLine, fillRect, drawRect

Drawing on a Panel

- To draw on a panel:
  - Define a new class that extends the JPanel class
  - Override the paintComponent() method in derived class
- paintComponent() method takes one parameter
  - an object of type **Graphics**
- Graphics object
  - Abstract class
    - Implementation different for each platform
  - a collection of settings for drawing images and text, such as colors and fonts
- All drawing in Java must go through a Graphics object

Drawing on a Panel

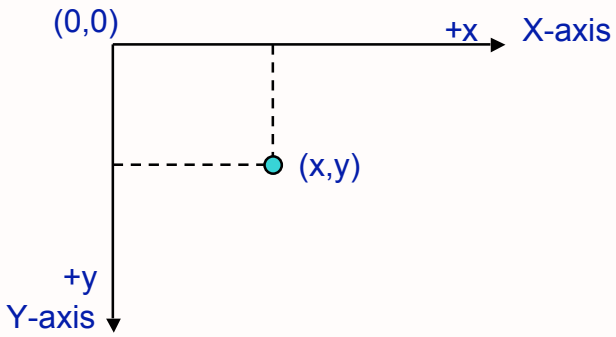
```
class MyPanel extends JPanel
{
      public void paintComponent(Graphics g)
      {

            // code for drawing goes here

      }
}
```


The paintComponent Method()

- paintComponent() is called automatically by the system whenever the container needs to be redrawn on the screen
  - Do not call this method yourself
  - It will be called when it needs to be
- If you need to force repainting of the screen, call the repaint() method
  - causes paintComponent() to be called for all needed components with appropriate Graphics objects
  - Do not override repaint - does system stuff

Graphics object

- Measurements on a Graphics object is in pixels, as an offset from the top-left corner
  - The (0,0) coordinates represent the top-left corner of the container on which you are drawing



Drawing on a Panel

- Displaying text is a special type of drawing, called rendering text
- To render text on a panel, call drawString()

```
class HelloWorldPanel extends JPanel
{
      public static final int MESSAGE_X = 75;
      public static final int MESSAGE_Y = 100;

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

            g.drawString("Hello World.",
                  MESSAGE_X, MESSAGE_Y);
      }
}
```

Drawing on a Panel

- Code calls the superclass (JPanel) paintComponent() method
- For JPanel class, draw/paint the panel means fill in the background color
- To make sure background color gets filled, call the superclass paintComponent() method
  - Every JPanel should color its background

ColorJPanel.java: Example of Drawing

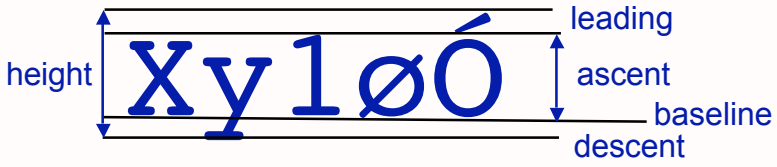
- Colors made up of three components
  - Red, Green, Blue component
  - RGB values
    - Components: either 0 to 255 or 0.0 to 1.0
- Thirteen predefined colors in the Color class:
  - black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, and yellow
  - Also defined in all caps
  - See API

FontJPanel: Drawing on a Panel

- Fonts
  - Font name (getFamily -- platform-specific)
    - SansSerif, Monospaced, Serif
  - Font style (getStyle)
    - Font.PLAIN, Font.ITALIC, Font.BOLD
  - Font size
    - Measured in points

FontMetrics

- Height
- Descent
  - Amount character dips below the baseline
- Leading
  - Difference between the descent of one line of text and the ascent of the line of text below it

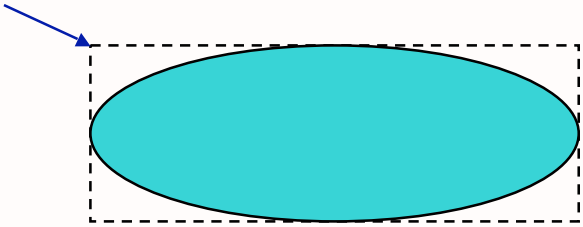


MetricsJPanel

Drawing Lines, Rectangles, Ovals

- Draw ovals, rounded rectangles within bounding rectangle

Starting Position of oval



- Can also draw arcs, polygons, polylines

LinesRectsOvalsJPanel

2-Dimensional Shapes

- Java includes the *Java2D* library
  - Advanced, powerful set of graphics operations
  - version 1.2 of the Java 2 SDK and higher
- To use the Java2D library, you need an object of the **Graphics2D** class
  - Graphics2D is derived from the Graphics class
  - Need to cast object
- paintComponent() actually receives an object of class Graphics2D

Getting a Graphics2D Object

- To obtain a Graphics2D object…

```
public void paintComponent(Graphics g)
{
      Graphics2D g2 = (Graphics2D)g;

      . . .
      // drawing code goes here
}
```


Shape Objects

- With a Graphics2D object, you can draw objects that implement the **Shape** interface
- The Java2D library includes a number of Shape objects, including
  - Line2D
  - Rectangle2D
  - Ellipse2D

Shape Objects

- To draw a Shape
  - create an object of a class that implements the Shape interface
  - call draw() on the Graphics2D object you want to draw…

```
Rectangle2D rect  = . . .;
Ellipse2D   ellip = . . .;
g2.draw(rect);
g2.draw(ellip);
```

ShapesJPanel.java

An Issue with Shape Objects

- Shape objects represent coordinates as single precision floating-point numbers
- All constant floating-point numbers are doubles
  - Problematic
  - Since Java is strongly-typed, cast **double** value into a **float** value
- Add a suffix to a constant floating-point number to indicate a float…

```
float f_num1 = 1.2;   // WILL NOT WORK
float f_num2 = 1.2F;  // WILL WORK
```

An Issue with Shape Objects

- Why is this a problem for Shape objects?

```
float f_num = shape1.getWidth();
```

  - will not work
  - getWidth() returns a double and f_num is a float
  - We need to cast the return value

```
float f_num = (float)shape1.getWidth();
```


An Issue with Shape Objects

- The Shape object stores the width as a float
  - We want it as a float
  - Its accessor method returns a double!
- The 2D library supplies two versions of every Shape object
  - one with float coordinates
  - one with double coordinates
- For example, Rectangle2D is an abstract class with two concrete (inner) subclasses, Rectangle2D.Float and Rectangle2D.Double

Creating Shape Objects

- We can make two rectangle objects:

```
Rectangle2D.Float floatR = new Rectangle2D.Float(
      10.0F, 25.0F, 22.5F, 20.0F);
Rectangle2D.Double doubleR = new Rectangle2D.Double(
      10.0, 25.0, 22.5, 20.0);
```

- The .Float and .Double classes are subclasses of Rectangle2D
  - After we create the object, we do not have to remember its type (they behave the same)

```
Rectangle2D floatR = new Rectangle2D.Float(
      10.0F, 25.0F, 22.5F, 20.0F);
Rectangle2D doubleR = new Rectangle2D.Double(
      10.0, 25.0, 22.5, 20.0);
```


Creating Shape Objects

- Construction is only difference in .Float and .Double versions is
  - one takes float values and one takes double values
- If you want to make a rectangle and you have double values (or constants), make it a Rectangle2D.Double object
- If you have float values, make it a Rectangle2D.Float object
- After you've made the object, the difference does not matter

Point Objects

- In addition to lines, rectangles, and ellipses, Java2D provides Point objects
- Everything that holds for Rectangles holds for other Shapes, including Points…

```
Point2D p1 = new Point2D.Double(10, 20);
Point2D p2 = new Point2D.Float(10F, 20F);
```


Creating Rectangular Shapes

- The Rectangle2D and Ellipse2D classes derive from a base class, **RectangularShape**
- These classes of Shapes are easy to construct
  - Constructor takes the x and y coordinates of the top-left corner and the width and the height

```
Rectangle2D rect = new Rectangle2D.Float(10,20,100,200);
      // init pos = (10,20) size = 100x200
Ellipse2D ellip  = new Ellipse2D.Float(50,75,200,300);
      // init pos = (50,75) size = 200x300
```

Creating a Line Shape

- Lines are very easy to make
  - provide the start and end coordinates, as either Point2D objects or as XY coordinates:

```
Point2D start_point = new Point2D.Double(5,10);
Point2D end_point   = new Point2D.Double(25,75);
Line2D line1 = new Line2D.Double(start_point,end_point);

Line2D line2 = new Line2D.Double(5,10,25,75);
```


Creating Shapes – Other Ways

- An ellipse can be constructed by passing the ellipse.setFrame() method a Rectangle2D object
  - method uses the Rectangle2D object as the bounding rectangle and constructs the ellipse inside of it
- All of the different methods of creating these types of objects are documented in the online API documentation

Changing Colors

- Shapes are drawn in the default color
  - To change the color, call setPaint() on Graphics2D object and pass it a Color object
- To use one of the predefined colors:

```
g2.setPaint(Color.lightGrey);
g2.draw(. . .);        // will draw this in light grey
g2.setPaint(Color.magenta);
// will render this in magenta
g2.drawString("Hello",100,100);
```


Color Filling

- Fill a Shape using the Graphics2D fill()

```
Rectangle2D rect1 = . . .;
Ellipse2D   ellip = . . .;

g2.setPaint(Color.cyan);
g2.fill(rect1);     // fills the rectangle with cyan
g2.setPaint(Color.red);
g2.fill(ellip);     // fills the ellipse with red
```

General Path

- A shape constructed from straight lines and complex curves
- java.awt.geom.GeneralPath

Shapes2JPanel.java

External Images

- You can now create simple images by drawing lines and shapes
- Complex images, such as photographs, flowcharts, etc., are usually generated externally
  - from scanners, Photoshop, etc.
- After an image is stored in a file (local or on the Internet), you can read and display it on a Graphics object

Reading External Image Files

- To read an image file, use a **Toolkit** object
- Toolkit reads GIF and JPEG image file formats

```
String filename = "/Users/sarasprenkle/Documents/" +
      "UDel/CISC370/java.gif";
Image java_image = Toolkit.getDefaultToolkit()
      .getImage(filename);
```

- gets the default Toolkit object
  - For system-dependent tasks
- getImage() loads the file and creates a new **Image** object that encapsulates the GIF file
  - Object variable `java_image` refers to created object

Drawing the External Image

- The image can be displayed on a Graphics object using drawImage()
  - drawImage() takes the Image object variable and the X and Y coordinates to place image

```
String filename = "/Users/sarasprenkle/Documents/" +
      "UDel/CISC370/java.gif";
Image java_image = Toolkit.getDefaultToolkit()
      .getImage(filename);
g.drawImage(java_image, 25, 100, null);
```

External Images – A Slight Problem

- Java (in particular, the AWT) recognizes that loading an image from disk (or network) can take a long time
- When you call the drawImage() method, your program may not have completely loaded the image from the disk!

Non-Blocking Image Loading

- When you first call drawImage(), Java starts a separate thread of execution that loads the image
  - drawImage() returns after it has started to load the image, **not** when it is **done**!
  - drawImage() does a non-blocking read
- Loading the image from disk occurs while your program continues

Monitoring Image Loading

```
String filename = "/Users/sarasprenkle/Documents/" +
      "UDel/CISC370/java.gif";
Image java_image = Toolkit.getDefaultToolkit()
      .getImage(filename);
g.drawImage(java_image, 25, 100, null);
```

- The fourth parameter to drawImage() call is an **ImageObserver** object
  - ImageObserver receives notifications during image loading about the current state

Waiting for Complete Loading

- For now, simply wait until the entire image has finished loading
  - Use a **MediaTracker** object
    - Tracks the acquisition of images
  - Process:
    - create a MediaTracker object
    - add Images to it, using the addImage() method, assigning each an integer ID
    - call the waitForID() method
    - Execution pauses until that Image has been loaded

Waiting for Complete Loading

```
MediaTracker tracker = new MediaTracker();

String filename = "c:\Images\my_new_truck.jpeg";
Image truck_image = Toolkit.getDefaultToolkit()
      .getImage(filename);

int id = 1;
// adds this image to the tracker w/ ID of 1
tracker.addImage(truck_image, id);
try {
// waits for image w/ ID of 1 to finish loading
 tracker.waitForID(id);
}
catch (InterruptedException exp) { }
```


Waiting for Complete Loading

- If you are loading an Image, you can wait for it to be completely loaded into memory before you do anything else
- Normally, this is not necessary
- The drawImage() method knows that the image may take awhile to load and it can handle that
  - You only need to wait for the image to completely load if you are going to copy the Image object or otherwise manipulate it

Random Numbers

- Create a random number generator
  - Random generator;
- Initialize it
  - generator = new Random();
- Generate Random Numbers from 0 to 8
  - x = generator.nextInt(9);
- How do you get random numbers from 1 to 10?