An Introduction to Java GUI Design

While simple I/O in Java can seem unnecessarily complex, coding a simple GUI is relatively easy. To code a GUI application in Java, you will need to import javax.swing.* and java.awt.event.* . UI controls in Java originally were coded using heavyweight (drawn using the native OS graphic toolkit) Abstract Window Toolkit components; now the AWT is used primarily only to handle events, with Swing components, which are lightweight and can take on a variety of visual styles ("look-and-feel"s), being used for the visuals.

You'll need to be able to refer to the online Java docs website for information about fields and methods of Swing objects. The Sun Java website also has a plethora of Swing tutorials with sample source code.

http://java.sun.com/javase/6/docs/api/
http://java.sun.com/docs/books/tutorial/ui/index.html

In simple GUI applications, typically one has a class that overrides a JFrame, the base class for a window, and a main function that initializes your frame class and calls setVisible(true) on the window. Or you can have a private member variable JFrame and modify that in your main function, instead of having your main class extend the JFrame. Several important methods used in the constructor of your JFrame class include:

.setTitle(name);
.setSize(int width, int height).
(Toolkit.getDefaultToolkit.getScreenSize() returns the screensize, as a Dimension object. You could then set the size of your window based on a fraction of the dimension.width and dimension.height.)

.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
.setResizable(false);

.setLayout(LAYOUT_STYLE);

The default layout style is flow, where components are shown in the window in the order in which they are added, left to right, then top to bottom. Another simple one is Border Layout, where there are center, north, south, east, and west panels to which you can add components:

.setLayout(new BorderLayout());

You would then add components to your window (more on this later) specifying as an extra parameter their destination in the layout, e.g.:

p.add(new JButton("Example"), BorderLayout.SOUTH);

GridLayout is another layout, where you specify the number of rows and columns, and items added to the panel are automatically added sequentially to the grid.

p.setLayout(new GridLayout(4,4));

As for the look-and-feel—the default, Metal, is rather ugly. If you're familiar with Limewire, that's the look-and-feel that program uses. You can set your program to use your OS visual style by using the following code:

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

The names of the other styles are available at http://java.sun.com/docs/books/tutorial/uiswing/lookandfeel/plaf.html.

Swing visual objects typically derive from a base object called a Component. A Container is a name for a component that can contain other components. You won't add visual objects to your JFrame window itself—you'll use the .getContentPane() method on your window, which returns a container to which you can add items. For the container that this function returns, you can declare and initialize a new container or a new JPanel (which you can override and create your own panel to customize), which inherits from a container and can contain other components, panels, and containers.

Important methods used with components include:

.setBackgroundColor(Color.RED);
Other color names can be found at http://java.sun.com/javase/6/docs/api/java/awt/Color.html.
.setForegroundColor(new Color(red, green, blue)); where the saturation of red, green, and blue is represented with integers 0-255, or with floats 0-1. Another interesting color function is .brighter() and .darker()—Color.RED.brighter() equals pink.
.setPreferredSize(int width, int height);
.setPreferredSize(dimensionObject);

You can declare UI components as member variables in your frame class and in your constructor initialize them and add them to the container or panel that is your frame's contentpane—your components will automatically be repainted when the window is repainted—or you can call revalidate() if you somehow change the size or layout of your components or repaint() if otherwise necessary. You can also draw strings and graphics directly on your panel, by overriding the paintComponent(Graphics g) function. (So you are going to have to make your own panel class that extends jPanel if you want to draw on it besides adding components to it.) You will need to call super.paint() at the beginning of this function, so that painting the background and similar aspects of the window are handled automatically. (When you're just getting started, don't worry too much about the why and how of the graphics device object. The system handles the nitty-gritty—you just draw on it.) In your paintComponent you will then call functions like:

g.setColor(Color name or rgb value, as
above); g.drawRect(x,y,width,height);
g.fillRect(x,y,width,height);

Different shapes, fills, and borders are available for drawing. You can draw rounded rectangles, polygons, and lines. A list can be found at http://java.sun.com/javase/6/docs/api/java/awt/Graphics.html. For reference, (0,0) is the top-left corner of the window.

As for drawing text, you'll want to first specify a font:

Font blah = new Font(Font.SANS_SERIF, Font.BOLD + Font.ITALIC, 14);

Font blah = new Font(Font.SERIF, Font.PLAIN, 12);

Using a particular true-type font is tricky, as you may not be certain whether the user has the font installed. Relying on the font-name constants is easier.

g.setFont(font)

g.drawString(text,x,y);

The width of text shown on the string is given in letter widths (kerning), not in pixels like for graphics. For future reference, you can determine the width of a certain string on the screen using the FontMetrics class.

FontMetrics = g.getFontMetrics(font);
integer = Fm.stringWidth(stringToBeWrittenInFont);

g.drawImage(imageObject,x,y,null);

null represents an optional MediaTracker for the image.

You can make a new Image object to be drawn in several ways. If the file is in your current working directory on the local machine, you can use Image image = Toolkit.getDefaultToolkit().getImage(ImagefileNameString); If it's an URL, you can use: URL u = new URL(relativeOrAbsoluteURLofImageFile); Image image = Toolkit.getDefaultToolkit().getImage(u); If you are using a Jar file (more on this later) that includes your image, you can use Image image = Toolkit.getDefaultToolkit().getImage(getClass().getResource(imageFileName));

Getting back to UI components, other ones you may wish to add to your containers include the following. You'll declare and initialize each component then use the panel.add(component, optional layout parameters); to add it.

JButton(String label);

JLabel(String text);

JTextField(String textCanBeEmpty, int ShownWidthInChars);
JTextField.setEditable(false)
JTextField.getText(); JTextField.setText("");

JTextArea(String optionalText, int rows, int columns);
JTextArea has the same setEditable, getText(), and setText() functions.

To add scroll bars to your text area, you must make a new JScrollPane to put your JTextArea inside.

yourPanel.add(yourJScrollPane(yourJTextArea); or
yourPanel.add(new JScrollPane(yourJTextArea));

Making groups of checkboxes or radio buttons is easy. With radio buttons, only one can be selected at a time, so you must group them together in a buttonGroup.

JCheckbox("label");

ButtonGroup();
JRadioButton("label",false);
buttonGroup.add(radioButtonName);

Both checkboxes and radio buttons have an .isSelected() function that returns a Boolean.

You can group your buttons together visually by adding them to a panel, putting a border on this panel, then adding this panel to your base panel.

yourPanel.addBorder(BorderFactory.createEtchedBorder());
createLineBorder(), createMatteBorder(), createLoweredBevelBorder() are some other BorderFactory methods. BorderFactory constructs a new Border type for you.

You can add a title to a border by calling createTitledBorder(a border,"title"):
yourPanel.addBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(),"my panel"));

Other components you may wish to look into include JMenuBar, JMenu, and JMenuItem; and also JList, a listbox component, which you can also add to a JScrollPane.

http://java.sun.com/docs/books/tutorial/uiswing/components/menu.html

As you will see tomorrow, Java handles button presses, mouse clicks, and other such phenomena using the Observer design pattern, where listener objects can be added to objects that receive user events, and these listener objects implement methods to process these events when they happen. Tomorrow you will learn how to make your GUIs do more than just look pretty.