

# Chapter 2: Java and GUIs

## Java and GUIs

Java includes a lot of GUI power in its class library. All that power is in a group of classes called the *Java Foundation Classes*, or just *JFC*. JFC includes three smaller groups of classes: the *Abstract Window Toolkit* (or *AWT*), *Swing*, and *Java 2D*.

AWT was Sun's first group of GUI classes, and Sun included it in the first release of Java in 1995. AWT uses its host platform's graphic tools, meaning if you run an AWT application in Windows, it looks like a native Windows application, and if you run an AWT application in the Mac OS or Linux, it looks like a native Mac or Linux application. While some developers liked this, others wanted their applications to look the same on every platform.

Swing arrived with Java version 1.2, replacing AWT for the most part. Swing not only provides more GUI tools but also allows users to choose a native platform *look-and-feel*—which refers to the interface's general appearance and operation—or the cross-platform Java look-and-feel. The native look-and-feel adapts to its host operating system, while the Java look-and-feel stays the same no matter where it's running.

Java 2D is a group of classes for drawing two-dimensional graphics. It was originally introduced with AWT, but it has evolved since then to interact with OpenGL, giving it 3-D capabilities as well as the 2-D tools its name implies. Java 2D gives us tools for drawing shapes in a window. Swing actually uses Java 2D to draw many of its components, and we'll learn in another lesson how to use it to draw our own shapes directly onto the screen.

Today, we're going to start with Swing and learn how to build basic GUI windows and applications. Later in the course, we'll add Java 2D to our repertoire when we learn how to draw our own shapes in those windows.

### "Hello, World!" Again!

If you took my *Introduction to Java Programming* course, you probably remember the "Hello, world!" Java program we created. We're going to start out with a simple "Hello, world!" program again. But this time, we'll make the message appear in its own window instead of in the console window. Here's what the window will look like:



A GUI Application

First, I'm going to show you the program that created that window, and then we'll talk about how it works:

```
import java.awt.*;
import javax.swing.*;

public class GUIApp1
{
    private JFrame frame;

    public static void main (String[] args)
    {
        GUIApp1 gui = new GUIApp1();
        gui.start();
    }

    public void start()
    {
        frame = new JFrame("My First GUI App");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPane = frame.getContentPane();

        JLabel label = new JLabel("Hello, world!");
        contentPane.add(label);

        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Now let's talk about how this program did what it did.

First, we imported two packages. The first one, *java.awt.\**, contains all the AWT classes. Swing replaced many of the AWT classes, but not all of them, and we'll still need some of them for our GUI programs. The second package we imported, *javax.swing.\**, contains all the Swing classes, which we'll use for most of our GUI tasks.

We only have one instance variable in our class, *frame*, a JFrame variable. Java uses the JFrame class to build windows, and we'll use the frame variable to create and manage our window.

Our main method is very simple. It creates a new GUIApp1 object named *gui* and calls its start() method.

The start() method is where all the work takes place. The first line creates our JFrame object. The JFrame constructor uses the string we give it ("My First GUI App") as the title for the window. Without that string, we'd get a window with an empty title bar.

The second line tells Java to stop running the program if the user closes the window. Without that, the program would keep running even after the window closes.

The third line creates a *Container* named contentPane. A container is a Java GUI component designed to hold other GUI components. We have to use one here because we can't add things directly to the frame. If this seems strange, let me explain.

Think of the frame as the wood frame around the glass of a window in a building. A Java frame holds the title bar with its icon and buttons, but we can't add items to it. We add them to the *pane* of the window, which will display all our GUI *components*. We call the frame's getContentPane() method to get the pane we will add our components to. The contentPane container is like the pane of glass in a window.

The following figure shows the parts of the window frame:



Parts of the window frame

If we continue along with our program, we see that the fourth line of the `start()` method creates another Swing object, a `JLabel` named *label*. You might see a pattern here. Swing class names start with "J" to separate them from the older AWT classes. `Container`, for example, comes from the AWT package, not the Swing package.

Anyway, a `JLabel` holds text to display in a window. The string we give its constructor is the string it will display.

After we create the `JLabel`, we add it to the pane in the fifth line of the method so Java can display it.

The sixth line sets the size of our window, 300 pixels wide and 300 pixels tall, in the call to frame's `setSize()` method.

The last line tells Java that it's finally time to display the window by calling frame's `setVisible()` method with an argument of `true`.

Okay, we've created a window. But it doesn't do much, does it? How do we make it do more? With more GUI components, of course! Java has lots of them: menus, buttons, radio buttons, check boxes, dialogs, sliders, scroll bars, and on and on. We'll not only learn to display them, but we'll learn to interact with them, too.