JFRAME AND JPANEL

In a Java GUI program, each GUI component in the interface is represented by an object in the program. One of the most fundamental types of component is the window. Windows have many behaviors. They can be opened and closed. They can be resized. They have "titles" that are displayed in the title bar above the window. And most important, they can contain other GUI components such as buttons and menus.

Java, of course, has a built-in class to represent windows. There are actually several different types of window, but the most common type is represented by the *JFrame* class (which is included in the package`javax.swing`). A *JFrame* is an independent window that can, for example, act as the main window of an application. One of the most important things to understand is that a *JFrame* object comes with many of the behaviors of windows already programmed in. In particular, it comes with the basic properties shared by all windows, such as a titlebar and the ability to be opened and closed. Since a *JFrame* comes with these behaviors, you don't have to program them yourself! This is, of course, one of the central ideas of object-oriented programming. What a *JFrame* doesn't come with, of course, is content, the stuff that is contained in the window. If you don't add any other content to a *JFrame*, it will just display a blank area. You can add content either by creating a *JFrame* object and then adding the content to it or by creating a subclass of *JFrame* and adding the content in the constructor of that subclass.

The main program above declares a variable, `window`, of type *JFrame* and sets it to refer to a new window object with the statement:

```
JFrame window = new JFrame("GUI Test");
```

The parameter in the constructor, "GUI Test", specifies the title that will be displayed in the titlebar of the window. This line creates the window object, but the window itself is not yet visible on the screen. Before making the window visible, some of its properties are set with these statements:

```
window.setContentPane(content);
window.setSize(250,100);
window.setLocation(100,100);
```

The first line here sets the content of the window. (The content itself was created earlier in the main program.) The second line says that the window will be 250 pixels wide and 100 pixels high. The third line says that the upper left corner of the window will be 100 pixels over from the left edge of the screen and 100 pixels down from the top. Once all this has been set up, the window is actually made visible on the screen with the command:

```
window.setVisible(true);
```

It might look as if the program ends at that point, and, in fact, the `main()` routine does end. However, the window is still on the screen and the program as a whole does not end until the user clicks the OK button. Once the window was opened, a new thread was created to manage the graphical user interface, and that thread continues to run even after `main()` has finished.

The content that is displayed in a *JFrame* is called its content pane. (In addition to its content pane, a *JFrame* can also have a menu bar, which is a separate thing that I will talk about later.) A basic *JFrame*already has a blank content pane; you can either add things to that pane or you can replace the basic content pane entirely. In my sample program, the line `window.setContentPane(content)` replaces the original

blank content pane with a different component. (Remember that a "component" is just a visual element of a graphical user interface.) In this case, the new content is a component of type *JPanel*.

JPanel is another of the fundamental classes in Swing. The basic *JPanel* is, again, just a blank rectangle. There are two ways to make a useful *JPanel*: The first is to **add other components** to the panel; the second is to **draw something** in the panel. Both of these techniques are illustrated in the sample program. In fact, you will find two *JPanels* in the program: `content`, which is used to contain other components, and `displayPanel`, which is used as a drawing surface.

Let's look more closely at `displayPanel`. This variable is of type *HelloWorldDisplay*, which is a nested static class inside the *HelloWorldGUI2* class. (Nested classes were introduced in Subsection 5.7.2.) This class defines just one instance method, `paintComponent()`, which overrides a method of the same name in the *JPanel* class:

```
private static class HelloWorldDisplay extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      g.drawString( "Hello World!", 20, 30 );
   }
}
```

The `paintComponent()` method is called by the system when a component needs to be painted on the screen. In the *JPanel* class, the `paintComponent` method simply fills the panel with the panel's background color. The `paintComponent()` method in *HelloWorldDisplay* begins by calling `super.paintComponent(g)`. This calls the version of `paintComponent()` that is defined in the superclass,*JPanel*; that is, it fills the

panel with the background color. (See Subsection 5.6.2 for a discussion of the special variable `super`.) Then it calls `g.drawString()` to paint the string "Hello World!" onto the panel. The net result is that whenever a *HelloWorldDisplay* is shown on the screen, it displays the string "Hello World!".

We will often use *JPanels* in this way, as drawing surfaces. Usually, when we do this, we will define a nested class that is a subclass of *JPanel* and we will write a `paintComponent` method in that class to draw the desired content in the panel.

Source : http://math.hws.edu/javanotes/c6/s1.html