Section 1 - Problems with JOptionPane

Introduction to Week Ten

This week we will plunge head first into true GUI programming. Fasten your seatbelts.

Important Note for Early Readers

I am making this module available earlier than the scheduled 10th week for students who want to play with GUIs sooner. If you are reading this early in the course, some of the OOP terminology will be new to you, and you'll have to fill in those gaps on your own. By the 10th week, when this material is required reading, you will have had all the prerequisites and should be able to read this with full comprehension.

Reading

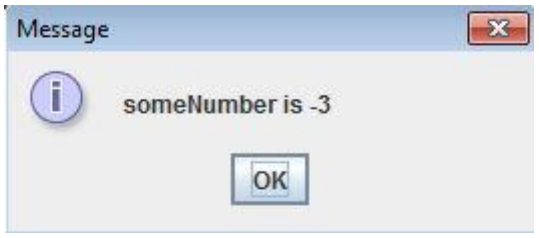
The optional textbook reading is handled as we have been doing all along: look up the terms with which you want more coverage in the index and see where it leads.

10A.1.1 JOptionPane.showMessageDialog()

We modify one of our previous console programs by changing all the **println()** calls to **showMessageDialog()** calls.

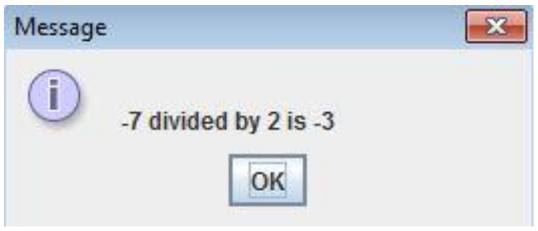
```
import javax.swing.*;

public class Experiment_1
{
   public static void main(String[] args)
   {
      int someNumber;

      someNumber = -7;
      someNumber = (someNumber-2)/3;
      JOptionPane.showMessageDialog(null, "someNumber is " + someNumber);

      // just some more stuff to demo / and %
      JOptionPane.showMessageDialog(null, "\n-7 divided by 2 is "
         + (-7 / 2)  );
      JOptionPane.showMessageDialog(null, "\n-7 mod 2 is "
         + (-7 % 2)  );
   }
}
```

When you run this, you'll notice that between each **showMessageDialog()** method call, the user must click **OK** to move the program execution forward.



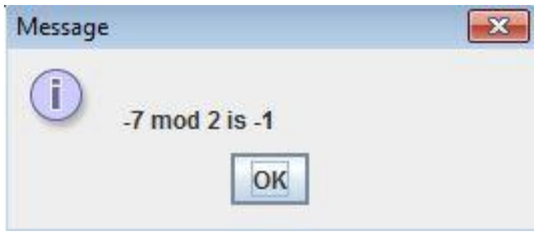
```
someNumber is -3
```

→



```
-7 divided by 2 is -3
```

→



```
-7 mod 2 is -1
```

What about the '**\n**'**?** In the console app we needed it to cause output to be placed on a new line. But here each output statement emits a new message dialog box, so it is unnecessary. We can remove the '**\n**'s from the program. But just to see how closely you are following along, what effect, if any, do these '**\n**'s have in this GUI version of our program?

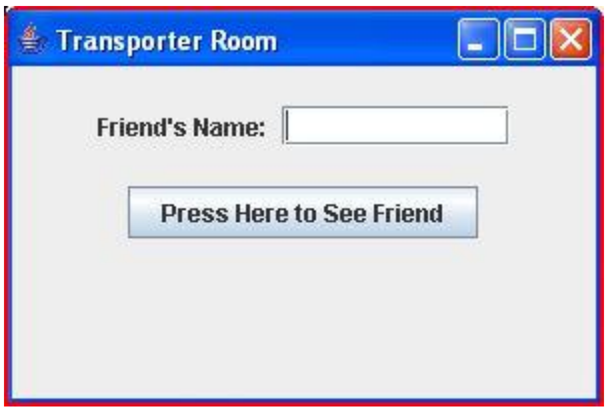
Can you see now, why sometimes the less beautiful ***console apps*** are more convenient if we have a large quantity of output?

10A.1.2 Not Quite a GUI ... Yet

Actually, what we just did (and have been doing) is not truly GUI. Placing a bunch of **showMessageDialogs** onto the screen in rapid succession is really just a confused form of mixing console and graphics. To do things right, we have to learn the "New School" way to do GUI, and that's what we study next.

Section 2 - New School GUIs

10A.2.1 Magic with Java



I've been holding out on you. Java can do absolutely incredible things, but I've been boring you with numbers and arithmetic. Okay, I'll give you a "taste."

What if you could push a button and see any one of your friends, any time, night or day? Before computers, this was impossible. Now, however, we can beam our friends right into our homes and offices with just the push of a button. Today we will learn how to write the GUI you see on the right. Nothing fancy... simply type your friend's name into the upper text box, go comb your hair and rinse your mouth with mouthwash, and when you're ready, come back and click the button "Press Here to See Friend."

In the next few pages, we will learn how to do this. First we look at the **Swing** library.

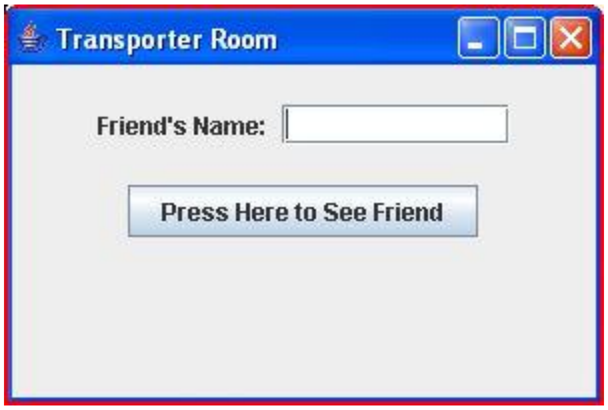
10A.2.2 The Swing Windows Library

As you saw with the two output methods we've studied, Java contains many built-in tools that do things for us. These tools are grouped in functional units called ***packages***. Did you notice that, in the GUI version of the ***Hello World!*** program, you had to include an **import** statement at the top: **import javax.swing.*;**? That was to bring in a GUI support ***package*** called **Swing**. We needed **Swing** in order to use the **JOptionPane** ***class*** that we used to display the message dialog.

There are other ***packages*** that we'll need as we progress. But this package serves a large group of GUI ***classes*** and ***methods*** that help us draw GUI onto the screen. You can usually tell which ***classes*** belong to this package because their names begin with the letter capital "J" ... **JOptionPane**, for example. We're about to learn a few other classes in the Swing library.

Section 3 The JFrame

10A.3.1 JFrame



The window that we are going to open this week is not the same **JOptionPane** style object that we used in previous weeks. This one is called a **JFrame** and it is much more customizable.

With the **JOptionPane** we could only give it one message and had to accept the "**OK**" button. With **JFrame** we can place as many different controls as we wish. As you can see there are three such controls inside the **JFrame**, and we'll discuss those shortly.

The way one places a **JFrame** on the screen is as follows:

```
   // establish main frame in which program will run
   JFrame frmMyWindow = new JFrame("Transporter Room");
   frmMyWindow.setSize(300, 200);
   frmMyWindow.setLocationRelativeTo(null);
   frmMyWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Let's explore this line-by-line.

```
   // establish main frame in which program will run
```

This is, of course, just a comment to help us understand what the next group of lines does. As usual, we aligned the comment up with the lines directly below it. Also notice that we did not skip any lines. We waited until the last statement in this group before skipping a line, then opened the next group with another comment.

```
   JFrame frmMyWindow = new JFrame("Transporter Room");
```

This line creates a new **JFrame** for us to work on. The **JFrame** object is called **frmMyWindow**, and when displayed on the screen it will have the words **"Transporter Room"** printed in the top title bar. This line is shorthand for the two longer statements we could have used instead:

```
   JFrame frmMyWindow;
   frmMyWindow = new JFrame("Transporter Room");
```

Either version will create a new **JFrame** object for us.

Terminology

We are using the **JFrame *class*** to help us define a specific **JFrame *object*** which we are naming frmMyWindow. In other words, we are instantiating a **JFrame** object and calling it **frmMyWindow**;

Next:

```
   frmMyWindow.setSize(300, 200);
```

This line sets the size of the **JFrame** to be 300 pixels wide and 200 pixels tall. A pixel is one pixel element, a very small size. Your computer screen might be 1024 pixels wide, for example. More optional terminology: We are using the **frmMyWindow *object*** to call the **setSize() *method***. Another way to say it is that we are ***dereferencing*** the **setSize()** method through the **object *frmMyWindow***.

```
   frmMyWindow.setLocationRelativeTo(null);
```

This line above is how we force the new **JFrame** object to be centered on the computer screen. (Optional terminology: We use the **frmMyWindow *object*** to ***dereference*** a call to the **setLocationRelativeTo() *method***.)

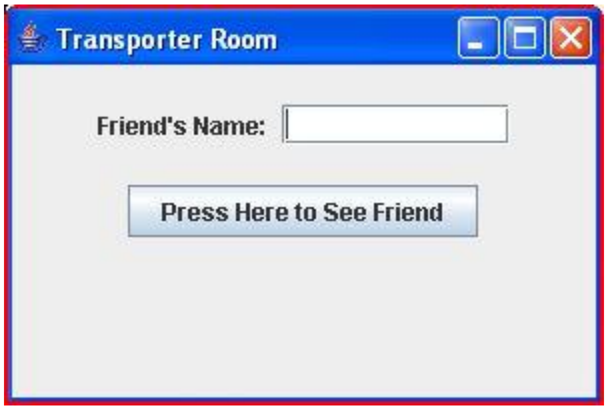
```
   frmMyWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

This last line of the group tells Java that when the user clicks the little **x** box in the upper right of the **JFrame**, the box will close and the program will end. It would be possible to allow the user to close a **JFrame** without ending our program. We want to avoid that in this program, which is why we include this line.

Normally, for your early GUI applications, you'll want to use these four statements, making minor adjustments to the size, object name, and text that goes on the top of the frame. You can even write a short program now that has only these four lines and try to run it. (If you do, don't forget to import javax.swing.*).

Section 4 The Layout

10A.4.1 FlowLayout



For reference, I am redisplaying a screen shot of the GUI that we are trying to build on this page (and the following pages).

Our next group of statements is dedicated to setting up what we call a ***layout manager*** for the **JFrame**. There are different ***layout managers*** in **Java**, and each specifies how the ***controls*** (***buttons***, ***labels***, etc.) will appear on the **JFrame**. The simplest is called **FlowLayout** because the ***controls*** will ***flow*** onto your **JFrame**, from left-to-right and top-to-bottom.

It is helpful to use **FlowLayout** with some ***gaps*** -- a small amount of space above, below, left and right of each control. This assures that the controls aren't placed too close to one another. We specify a vertical gap (in pixels) and a horizontal gap (in pixels). The horizontal gap will be 5, and the vertical gap will be 20. You'll see how these numbers are specified in the code below.

Here is how one creates a **FlowLayout** with the gaps described and uses it on a **JFrame**:

```
   // set up layout which will control placement of buttons, etc.
   FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 20);
   frmMyWindow.setLayout(layout);
```

Again, we start with a comment that documents what we are about to do:

```
   // set up layout which will control placement of buttons, etc.
```

Next we see this:

```
    FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 20);
```

This line above creates a new **FlowLayout** for us, gives it the name **layout** and tells Java that we should center the controls (rather than left or right justifying them), and use a horizontal gap of 5 pixels and a vertical gap of 20 pixels. This line can, like the **JFrame** line, be broken up into two separate statements for ease of readability:

```
FlowLayout layout ;
layout = new FlowLayout (FlowLayout.CENTER, 5, 20);
```

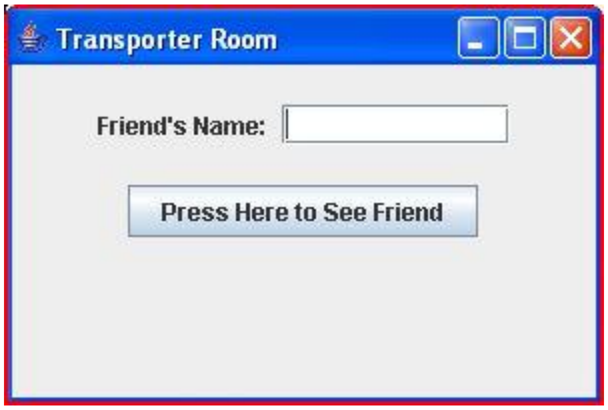
Again, you don't have to understand the exact meaning of each of these statements; just know that they create a flow layout.

```
 frmMyWindow.setLayout(layout);
```

This line sets the layout manager of the **JFrame** to the layout manager object, **layout**, that you created in the statement before. Notice the similarity to the statements that we used to initialize the **JFrame** in the last section.

Section 5 The Controls

10A.5.1 Buttons, Labels and Text Fields



Our next group of statements is dedicated to creating the three controls that we see on the screen. The text **Friend's Name** is called a ***label***, and for that we need to create an object of the class **JLabel**. The text box to the right of that label is called a **JTextField**. And the button at the bottom is called a **JButton**.

Here is how one creates these three controls:

```
// 3 controls: a label, a text field and a button
JLabel lblMyLabel = new JLabel("Friend's Name: ");
JTextField txtMyTextArea = new JTextField(10);
JButton btnMyButton = new JButton("Press Here to See Friend");
```

I'll skip the comment line, since you should know what this does by now. Let's look at the first of these three lines:

```
JLabel lblMyLabel = new JLabel("Friend's Name: ");
```

This uses syntax similar to what we've seen so far. It creates a **JLabel** object called **lblMyLabel** and places the text "Friend's Name: " on the **Label**. By the 10th week, you should be able to expand this shortened statement into two longer ones that perform the same task, stepwise. Normally, of course, we don't want to make things longer, but you do need to understand that this is a combination of two statements.

The next line is:

```
JTextField txtMyTextArea = new JTextField(10);
```

That line creates the text input area, called a **JTextField**. We name that field **txtMyTextArea** and we make sure that it has space for 10 letters in it.

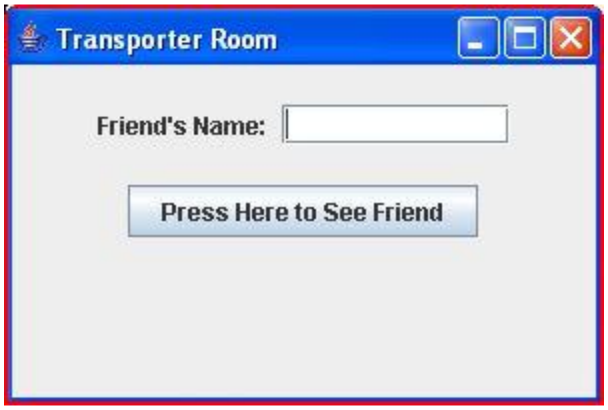
The last line is:

```
   JButton btnMyButton = new JButton("Press Here to See Friend");
```

That line creates a **JButton** . We name that **JButton btnMyButton** and we put some text on the button.

Section 6 Visibility

10A.6.1 Adding the Controls to the JFrame



You might think that the statements in the previous group would have been enough to place the controls into the **JFrame**, but they are not. Just creating controls like **JTextAreas** and **JButtons** don't give them a home. For example, there may be more than one **JFrame** in our program, and we have to tell Java on which **JFrame** to place the controls. We also have to say in what order we want the controls to be placed (they don't have to be in the same order in which we declared them, in the last section).

Here is the group that does what we want:

```
   // place your 3 controls into frame
   frmMyWindow.add(lblMyLabel);
   frmMyWindow.add(txtMyTextArea);
   frmMyWindow.add(btnMyButton);
```

These lines are self-explanatory.

10A.6.2 Showing the JFrame

Just as we had to do something extra to make our controls appear on the **JFrame**, we must also tell Java that we want to see the **JFrame**. This might seem ridiculous. After all, why *wouldn't* we want to see the **JFrame**. Well, there are lots of times in which we want frames to be built, but stay invisible until we are ready for them. We are ready for this one as soon as we build it, so we issue the statement:

```
   // show everything to the user
   frmMyWindow.setVisible(true);
```

This is that last statement for the GUI of this application.

Section 7 The GUI Listing

10A.7.1 Trying It Out

Here is the entire GUI listing. You can create a project, add a class named "Transporter", and into that class .java file, you can paste the following. When you have done that, run the application to see what happens.

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

public class Transporter
{
   public static void main(String[] args)
   {
      // establish main frame in which program will run
      JFrame frmMyWindow = new JFrame("Transporter Room");
      frmMyWindow.setSize(300, 200);
      frmMyWindow.setLocationRelativeTo(null);
      frmMyWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // set up layout which will control placement of buttons, etc.
      FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 20);
      frmMyWindow.setLayout(layout);

      // 3 controls: a label, a text field and a button
      JLabel lblMyLabel = new JLabel("Friend's Name: ");
      JTextField txtMyTextArea = new JTextField(10);
      JButton btnMyButton = new JButton("Press Here to See Friend");

      // place your 3 controls into frame
      frmMyWindow.add(lblMyLabel);
      frmMyWindow.add(txtMyTextArea);
      frmMyWindow.add(btnMyButton);

      // show everything to the user
      frmMyWindow.setVisible(true);
   }
}
```

Try it out.

10A.7.2 Expectations

So what happened when you entered your friend's name and pushed the button?

It might not have been as spectacular as you expected, but remember, this is just the beginning. We have to learn a little more GUI programming before we can move your friend all the way across the galaxy to see you.