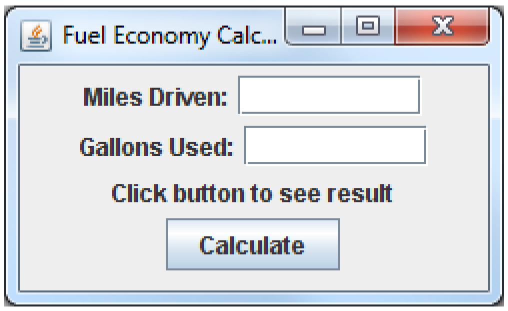
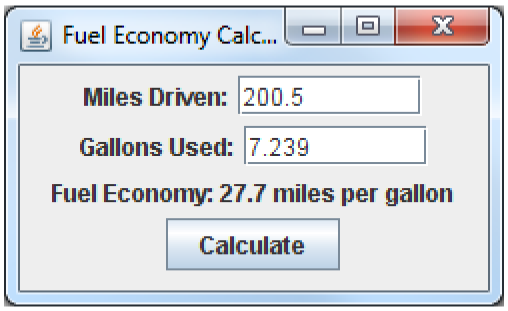
CIS 255 • Java Programming
Laboratory Assignment 11: GUI Applications

In this laboratory assignment you will work with Java GUI syntax to create a window that allows the user to enter the number of miles driven on a trip and the number of gallons of gas consumed and displays the fuel economy for the trip when the user clicks a button.

Background

The fuel economy of a car is dependent on the number of miles driven and the gallons of gas used in the process of traveling to a particular destination. Your goal is to create a window that allows the user to determine the fuel economy for a particular trip. The window might appear like the following (example from Windows 7):





Miles Driven: 200.5
Gallons Used: 7.239
Fuel Economy: 27.7 miles per gallon

Step 1: Download the Starter File and Launch TextPad

On Blackboard, click the link for the starter file to download it. Make sure you insert your first initial and last name and the laboratory assignment number where indicated (**FLast_Lab11.java**) and that you specify the file location. Once you have downloaded the file, open it in TextPad (or a similar Java code editor).

Step 2: Customize Your Code, Add `import` Statements, and Use Inheritance

Edit the comments at the top of the code to include your name, the course number, the laboratory assignment number, and a brief description of the program's contents. You may also wish to include the date (perhaps after the laboratory assignment number). Also, modify the class name to match your file name. As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire class definition to add comments.

This program requires `import` statements for two packages as well as one individual class. Use wildcard `import` statements for the packages `javax.swing` and `java.awt.event`; then, in order for the result to be displayed with one digit after the decimal point, import the `DecimalFormat` class explicitly.

This class should be derived from `JFrame` so that it can execute all of the window-related methods within the constructor without having to use a calling object. Add an `extends` clause for `JFrame` to the end of the class header.

Although the program uses the `main` method in the file as an embedded `main` for displaying the window, the majority of the code will go in the class brackets before the `main` method header.

Step 3: Declare Member Variables

The following variables should be declared as `private` members of the class (not as local variables in a method):

Description	Data Form
Panel to contain the window's contents	Panel
Label for miles text field	Label
Text field for entry of miles traveled	Text Field
Label for gallons text field	Label
Text field for entry of gallons of gas used	Text Field
Label to hold result	Label
Button to trigger calculation	Button

Do not put these declarations inside a method! They should only be within the brackets for the class.

Don't call the constructors for these objects yet – just declare the variables that will store the addresses of the objects. You'll call the constructors later.

Step 4: Begin the Constructor with Window Setup Methods

Begin writing the constructor by writing its header: it is a `public` method with no return type and the same name as the class. This is a no-arg constructor.

The first statements within the method should set up some of the initial characteristics of the window:

Window Attribute	Value
Title	Fuel Economy Calculator
Size	250 pixels wide, 150 pixels high
Default Close Operation	Exit on close

Call the inherited mutator methods that apply these attributes to the window.

Since your class extends `JFrame`, there is no need for a calling `JFrame` object: the object being constructed is the automatic (implied) calling object.

Step 5: Define a Helper Method to Construct and Add the GUI Components

In the examples in the textbook, a separate helper method sets up the GUI components contained within a panel. After adding a closing bracket for the constructor, write the header for a `private void` method named `buildPanel` with no parameters. In the body of this method, call the constructor for each of the GUI components declared as class members with an appropriate argument:

- The labels should be constructed with their initial text as the arguments to their constructor calls (the label that eventually displays the results should initially display "Click button to see result")
- Each text field should be eight characters wide
- The button should have a caption similar to "Calculate"
- The panel requires no arguments

Do not write declarations for the GUI components you have already declared in the class here, as these local variables would shadow the fields. Instead, assign the constructor calls to the fields already declared. (The assignment statement should not include the data type at the beginning.)

Instead of adding the labels, text fields, and button directly to the window, invoke the `add` method on the panel object six times within this helper method, with the GUI objects added in the order in which they are to appear in the window.

Make sure to use the `JPanel` object as the calling object for the `add` method here. You are not adding the GUI components directly to the window.

The default layout manager for a `JPanel` object is `FlowLayout`, so the GUI components added to the panel will be displayed row by row, left to right, and centered within the window.

Step 6: Define the Private Inner Listener Class

In order for the program to respond to the event generated when the user clicks the button, the class must include an event listener class from which it can register a listener object with the button. Within the class definition (after the closing bracket for `buildPanel` but before the header for `main`), write the header for a `private` inner class that implements the `ActionListener` interface. You may name this class anything that you want, although it is best to include the word `Listener` as part of the class name.

Chapter 7 mentions that any class that implements an interface must contain definitions for any methods included in that interface. In this case, a class that implements the `ActionListener` interface must include a `public void` method named `actionPerformed` with an `ActionEvent` parameter. Write the header for this method within the `private` inner class definition.

This listener method requires several local variables: at least one `String` variable to temporarily store the input gathered from the text fields, a `String` variable in which to "build" the result text, `double` variables to store each quantity from input as well as the result, and a `DecimalFormat` object that requires one digit before and after the decimal place. Declare these variables in the body of the method.

To obtain the input from each text field, invoke the method `getText()` on each text field object. Remember that this value is returned as a `String`, so it cannot be assigned directly to a `double` variable. You will need to parse the value from the text field in order to use it as a `double`.

Fuel economy is equal to the number of miles driven divided by the gallons of gas used. Calculate this result; then, build the result text with labels around the formatted value (e.g., "`Fuel Economy:` *value* `miles per gallon`").

Assign the text that will be displayed (the labels and the value) to the `String` variable you declared to hold this text. Use the concatenation operator to separate these values as you would in a call to one of the `System.out` methods or to `showMessageDialog`.

Invoke the `DecimalFormat` method `format` when building the `String` for the result text. This ensures that the `String` contains the value of the `double` with proper formatting.

Instead of displaying this result in a separate dialog, the program can use the label above the button to show the result. A GUI program changes the text displayed in a label by invoking the method `setText()` on the label object with the new text as the argument. Pass the `String` that contains the result text to a call to this method, using the third label as the calling object.

If the result text is not long enough to fill the width of the row, the button may move up to same row as this text. Remember that a container with a `FlowLayout` may adjust the placement of items based on changes to the width or contents of the window.

Don't call `showMessageDialog` in this program. You are to display the result within the same window, not in a separate dialog box.

Step 7: Register the Event Listener with the Button

Now that the class contains a listener class, the `buildPanel` method can register a listener of this type with the button object. In `buildPanel`, invoke the method `addActionListener()` on the button object with a new listener object of your class type as the argument.

Call the constructor for the listener object inside the argument list for the listener registration call. Your program does not need to assign the listener object to a variable, as it does not refer to the object explicitly: the registration takes care of the actual event handling. The listener object is called an *anonymous object* because it does not have a variable name for itself.

Step 8: Invoke the Helper Method and Make the Window Visible

Go back to the constructor. Call the helper method that sets up the GUI components and adds them to the panel. The components are now part of the panel, but they aren't part of the window yet. In the constructor, invoke the method `add` one time with the panel as the argument.

When adding a GUI component to the window itself, `add` does not require a calling object.

Finally, invoke the method that causes the window to become visible. By packaging all of these calls within the class constructor, the `main` method will not need to contain detailed code to set up the window.

Step 9: Construct a Window Object

The only code needed inside the `main` method is a call to the constructor for your class. You can create a variable associated with this object, or you can simply use the keyword `new` followed by the call to the default constructor (creating an anonymous window object).

If a window class does not contain an embedded `main` method, it cannot be executed. Another program that contains a `main` method can call the class's constructor to display its window.

Step 10: Compile Your Code and Execute the Byte Code

Compile your Java program, correcting any syntax errors as the compiler finds them, and then execute the compiled application byte code. Enter some test values to verify that the results match what you believe they should be. You can test the window multiple times with different values within the same run of the program.

Step 11: Add Inline Comments and Submit Your Code

Make sure that your code has **three or more inline comments** near the statements that you believe warrant the additional description (especially those related to GUI programming and events). These can be simple line comments at the end of a line of code that needs explanation, or block comments where each precedes a series of statements described in the comment.

Follow the instructions on the course website to attach and submit your source code.

Make sure you complete the submission process! If you attach your file without submitting the assignment, your file will not be received!