Lab Exercise 9
CS 2334

October 22, 2015

Introduction

In this lab, you will begin experimenting with Graphical User Interfaces (GUIs) in Java. GUIs are defined as a hierarchical set of graphical components (hierarchy, here, is in a *has-a* sense). At the top of hierarchy is a component that is a container of other components – in our case, this component is a **JFrame**. Other components, such as **JButton**, **JLabel**, **JTextField**, and **JRadioButton**, provide the individual pieces of the GUI with which the user interacts. **JPanel** objects are components that also act as containers for multiple components, giving us a convenient, logical way of grouping different parts of the GUI together.

Your task for the lab is to create a GUI for a simple base conversion calculator. The GUI will give the user a single text box in which to type an operand (value) and a set of radio buttons for selecting the type of operator. After performing the calculation, the GUI will display the result in a non-editable text field.

Learning Objectives

By the end of this laboratory exercise, you should be able to implement a simple GUI by:

1. Creating a window
2. Adding various graphical components to the window
3. Responding to window events in a meaningful way

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Import the existing lab9 implementation into your eclipse workspace.

   (a) Download the lab9 implementation:
   `http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab9/lab9-initial.zip`

   (b) In Eclipse, select *File/Import*

   (c) Select *General/Existing projects into workspace.* Click *Next*

   (d) Select *Select archive file.* Browse to the lab9.zip file. Click *Finish*

Number Representations

We are used to representing numbers in the *decimal* (or base 10) number system. What we mean by this is that a single digit can take on one of ten values (0...9). Furthermore, when we write the number: 237, we mean:

$$237 = 2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0. \tag{1}$$

However, a number can be written in terms of any positive base. For example, in the *binary* (base 2) number system, we only have two digits: 0 and 1. Then, when we write 101, we mean:

$$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0, \tag{2}$$

which is equivalent to 5 in decimal.

The binary number system is important because it is fundamentally how we represent **all** information inside of a digital computer. The downside to this representation is that it is hard to keep track of so many digits. As a result, it is common for us to use either octal (base 8) or hexadecimal (base 16) representations. In the case of hexadecimal, we still use the digits 0...9. However, after 9, we need a single digit to represent the next value. The standard is to use the letters $a...f$ to capture the values 10...15. For example, when we write $2bd$, we mean:

$$2bd = 2 \times 16^2 + 11 \times 16^1 + 13 \times 16^0, \quad (3)$$

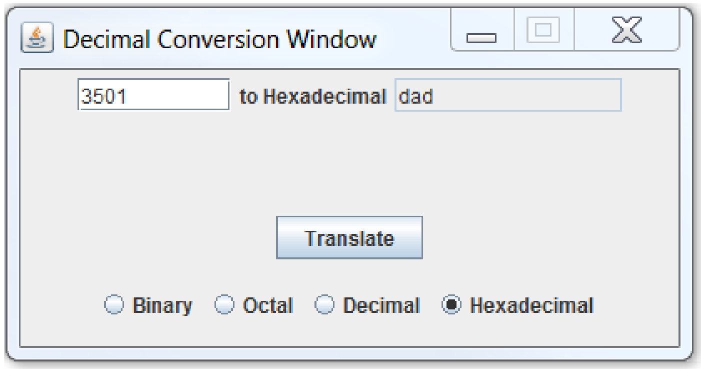
which is equivalent to decimal 701.

Simple Base Conversion Calculator

Below is an illustration of the graphical user interface for the calculator that we are creating. The user will enter a decimal number into the first text field. There are four base conversions provided to user: binary, octal, decimal, and hexadecimal. When the program is first started, the binary button should be selected.

When the user clicks on the button *Translate,* depending on the base selected by the user, the answer should be displayed to the right of the “to XXXX” label, where XXXX represents the user’s selected base.

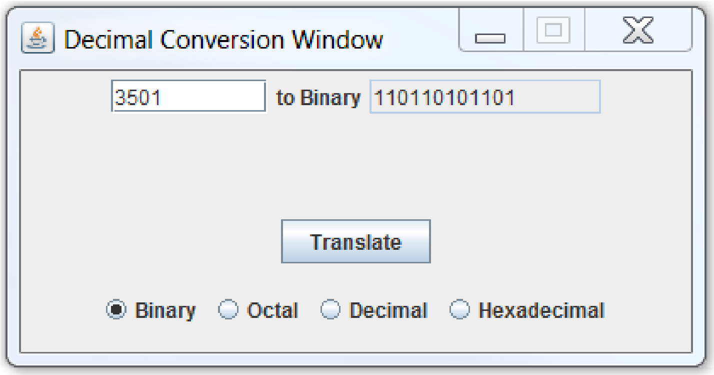
In the following example, the user types the value *3501* into the input text field, selects *Hexadecimal* and clicks *Translate.* The GUI then presents the result of converting decimal 3501 into hexadecimal: *dad.* Note that the result uses lower case letters.



3501 to Hexadecimal dad

Binary Octal Decimal [✓ Hexadecimal]

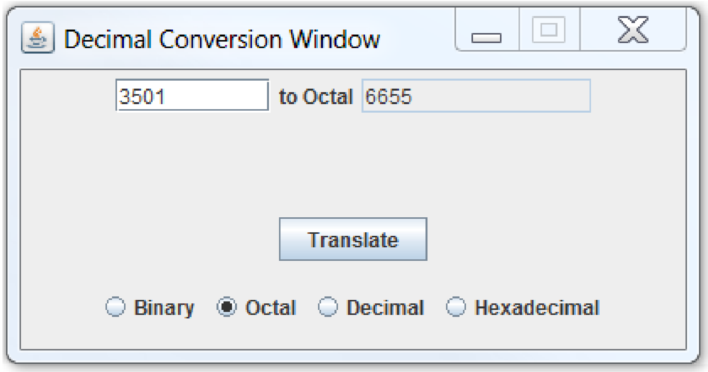
Likewise, when *Binary* is selected and *Translate* is clicked, the GUI presents the binary result: *110110101101*.


Decimal Conversion Window

3501 to Binary 110110101101

[✓ Binary] Octal Decimal Hexadecimal

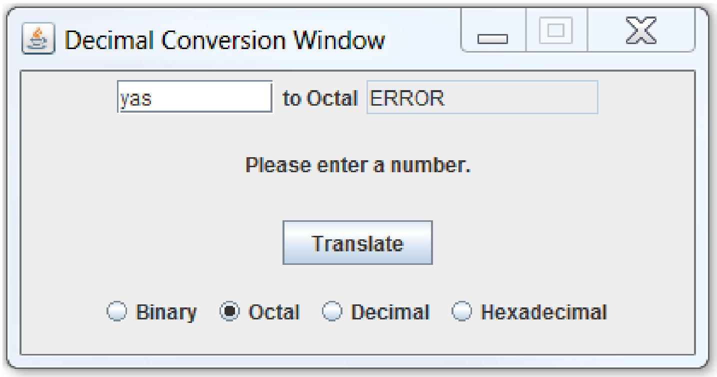
Finally, the *Octal* result is: *6655*.


Decimal Conversion Window

3501 to Octal 6655

Binary [✓ Octal] Decimal Hexadecimal

If the user enters an invalid number or leaves the text field blank, and then clicks *Translate*, then an error message should be displayed, just below the operand box.


Decimal Conversion Window

yas to Octal ERROR

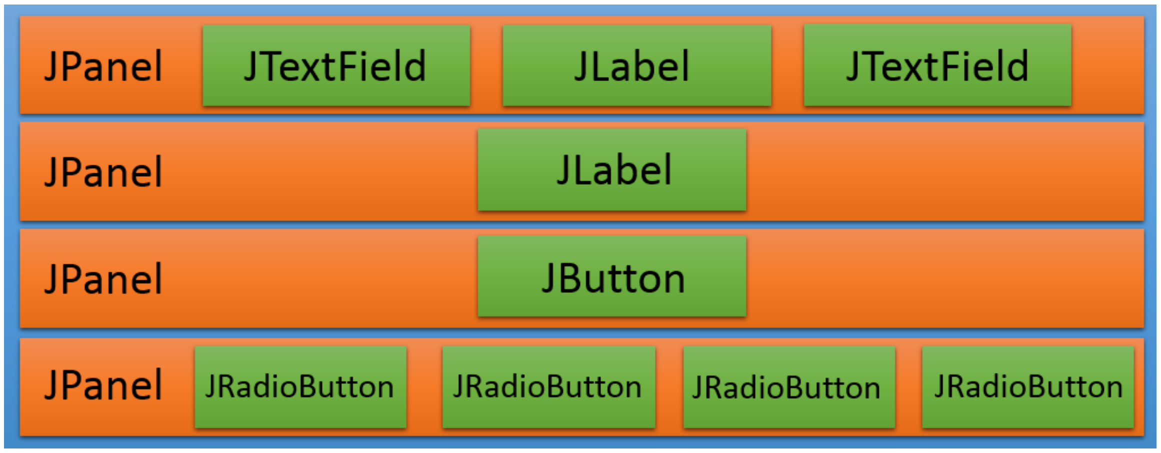
Please enter a number.

Binary [✓ Octal] Decimal Hexadecimal

When the user closes the window, the program should terminate.

GUI Organization

Our calculator is organized using a set of 4 horizontal rows, as shown below:



The rows, each encapsulated by a **JPanel**, contain the following information:

1. The operand, operator, and result
2. An error message (which can be blank)
3. A *Translate* button
4. A set of radio buttons for operator selection

Here, we have used a **GridLayout** of dimension 4×0 to organize the four **JPanel** objects within the **JFrame**.

Use the following components in the design of your GUI:

- **JFrame** for the window
- **JPanel** for the sub-panels
- **JButton** for the button
- **JTextField** instances for providing an *editable* input text box and a *non-editable* output text box

- **JLabel** instances for displaying the intended base conversion and the error message
- **ButtonGroup** to tell the Java windowing system that only one operator button can be selected at any one time
- **JRadioButton** instances for the radio buttons

Lab 9: Specific Instructions

There is only one class file, CalculatorFrame.java, the skeleton of which is provided in lab9.zip.

1. Modify the class according to the instructions given in the comments
   - Within the **CalculatorFrame** class, there is only the constructor and the *main()* method
   - Be sure that the class name is exactly as given in the initial zip file
   - You must use the default package, meaning that the package field must be left blank
   - Do not change the name for instance variables and method name
   - Do not add functionality to the classes beyond what has been specified
   - Don't forget to document as you go!
2. Test your program by compiling and testing all possible inputs and conversions.
   - Be sure all of the conversions are accurate
   - Make sure that the error message appears when the user tries to translate erroneous input
   - You do not have any unit tests to write for this lab. All of your testing will be done by interacting with the GUI. However, we will be testing your code with a set of unit tests

Hints

- Remember that your class is-a **JFrame**. This means that it will provide all of the methods that are made available by **JFrame**. This is why we call *super()* in Calculator Frame constructor.
- Your constructor should create all of the GUI components, hook them together, and attach the appropriate **ActionListener**s
- Your main function creates a single instance of **CalculatorFrame** and then does nothing else
- All of your inner, anonymous classes can "see" the instance variables
- See the **JRadioButton** API documentation for information about how to ask the radio button about whether it has been selected
- See the **JLabel** API documentation for information about how to change the text contained within the label
- See the **JTextField** API documentation for information about making a text field un-editable.
- There are many number conversion tools available on the web that you can use to check your results
- Look carefully at *Integer.toString()* for something that will help with your implementation.

Final Steps

1. Generate Javadoc using Eclipse.
    - Select *Project/Generate Javadoc...*
    - Make sure that your project is selected, as is the CalculatorFrame
    - Select *Private* visibility
    - Use the default destination folder
    - Click *Finish*

2. Open the *lab9/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that all of your classes are listed and that all of your documented methods have the necessary documentation.

3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, October 21st.

- Method 1: Submit through Eclipse

  1. From the *Window* menu, select *Preferences/Configured Assignment.*
  2. Select your project.
  3. From the Project menu, select *Submit Assignment.*
  4. Under *Select the assignment to submit*, select *Lab 9: Lab 9: Graphical User Interfaces.*
  5. Click *Change Username or Password....* Enter your Web-Cat username and password. Click *OK*. You should only need to do this step once per session.
  6. Click *Finish.*
  7. Your browser should automatically open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

- Method 2: Submit directly to the Web-Cat server

  1. From the File menu, select *Export.*
  2. Select *Java/JAR File.* Click *Next.*
  3. Select and expand your project folder.
  4. Select your *src* and *doc* folders.
  5. Select *Export Java source files and resources.*

6. Select an export destination location (e.g., your *Documents* folder/directory). This file should end in *.jar*
7. Select *Add directory entries*.
8. Click *Finish*.
9. In your web browser, login to the Web-Cat2 server.
10. Click the *Submit* button.
11. Browse to your jar file.
12. Click the *Upload Submission* button.
13. The next page will give you a list of all files that you are uploading. If you selected the correct jar file, then click the *Confirm* button.
14. Your browser will then open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Correctness/Testing: 45 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). **NOTE:** The tests used for this lab will focus on button interactions and values inside textFields - a style of testing different than what we have done before. These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation (up to 10 points)
- Missing or inappropriate inline documentation (2 points per violation; up to 10 points)
- Inappropriate choice of variable or method names (2 points per violation; up to 10 points)
- Inefficient implementation of an algorithm (minor errors: 2 points each; up to 10 points)
- Incorrect implementation of an algorithm (minor errors: 2 points each; up to 10 points)

- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code (up to 15 points)

If you do not submit compiled Javadoc for your lab, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions (where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late.