Lab4 part A: Debugging
Debugging Using Java & Eclipse*

CSSSKL162: Programming Methodology Skills

Summary

The purpose of this lab is to get in a bit of practice doing base conversion, and then hone your debugging skills. Start by downloading the following files and adding them to an eclipse project: {DebugMe.java, Box.java, Debug.java, TestComplieTime.java, Account.java, DebuggingExercise4.java. You will also need AssertDemo.java for the last part of this lab}

Download the Buggy Software

Download the java files associated with this lab from our class website. We'll start with DebugMe.java and Box.java; read the comments in both classes for further instructions. Keep in mind that it might be helpful to comment out parts of the code that you aren't currently working on. As an example, you might wish to comment out the compareBoxes() method so you can focus on first fixing the printSums() method. As you explore the software, note that not all errors will be as basic as syntax errors – some of the logical errors will be more complex to find.

(1) Debug the two programs (DebugMe.java and Box.java) by following the comments and fixing the errors. Note that there are both syntax and logic errors.
(2) We'll use the graphical debugger built into Eclipse
    a. Set a breakpoint on the first line in main.
        i. Do this by clicking in the "gutter" area for Eclipse and BlueJ.
        ii. These appear as blue circles in Eclipse and red stop signs in BlueJ.
    b. Click on the "Bug" rather than the "Play button".
        i. This begins your debug session – say "yes" to change to debug view in eclipse
    c. Hover over variables to see their values.

Working with Debug Flags

Using visual debuggers is one methodology used to trace and evaluate execution; next we'll consider using boolean flags to indicate our software should produce additional reporting output or alter the way in which the program proceeds. For example, consider the code that will only print to the console if the boolean variable "DEBUG" is set to true.

```
if(DEBUG)  {
        System.out.println( “Debug Info while “ + doing + “ with x=” + x + “ and y=” + y);
}
```

In this next section, you will fix errors by uncovering them using debug flags.

(1) Compile Debug.java and TestCompileTime.java, and run TestCompileTime's main() method.
(2) Change the debug flag in Debug.java to be true and rerun your code.
    a. Using the DEBUG output, fix the errors in this code until the program is correct
(3) Fix the TestCompileTime.java class so the debug flag actually works.
    a. You should be able to turn on and off all debug messages (rather than all output, as it is currently).
    b. You should still see non-debug messages when the flag is false.

Using a Debugger

No matter which editor you're using, there are debugging tools that can help simplify the process of hunting down logic errors. In Eclipse and BlueJ, you just need to toggle a breakpoint in the gutter of the text area; this should place a blue circle next to the line of code your debugger will execute up to. Then to debug, instead *of play*, click on the "little bug" icon next to play. Say yes to the perspective change dialog, and watch as your code executes up until the breakpoint. Once your code is paused, you now have the option to inspect variables and their values (hover over them or use a **watch**). You can ***step over*** whole lines at a time (such as function calls) or ***step into*** a function call to examine the execution further. If you're using JEdit, you may have to configure the plug-in manager to include a debugger. If on linux, consider using gdb, which offers command-line debugging (and there are GUIs to wrap this). Answer the following questions in your comments in any file and submit that.

(1) Using your debugger, debug Account.java and DebuggingExercise4.java
(2) Hover over variables to see their values.
    a. Where is the variable watch window on your screen?
        i. What information does this present to you?
            1. Describe this in comments in your code.
    b. Where is the method call stack on your screen?
        i. What information does this describe?
(3) Find the shortcut keys for the following debugging commands, and describe what each does in comments.
    a. Step over
        i. What does this do?
    b. Step into
        i. What does this do?

ii. How is it different from step over?

c. Step out
   i. What does this do?
   ii. How is it different from step over or step into?

d. Continue
   i. What does this do?
   ii. How is it different than moving in steps?

Debugging With Asserts

Yet another technique used in combination with console output, flags, and graphical debugging is Asserts. This technique allows you to convert **Class Invariants** and **Preconditions/Postconditions** into actual code that will **fail fast** if we find anything wrong. While asserts are terribly useful in many programming languages, they can incur runtime overhead (depending on implementation) and so are disabled by default in Java. When you run your code on the command line, you invoke the java virtual machine using "java.exe". If we provide the VM with the command line argument "-ea" (as in "java.exe –ea"), then we can make use of assertions in our code.

1. Download the **AssertDemo.java** driver and run it
2. It should run with no problems, ***but this is incorrect***. We must enable asserts in our IDEs.
   a. If you compile using the command line, then use "java.exe –ea YourClass.java"
   b. For BlueJ:
      i. http://www.cs.uwlax.edu/~riley/CS120F09/Handouts/7_assert_in_BlueJ.pdf
   c. For Eclipse:
      i. Click on "Run Configurations" in Eclipse. (a little arrow next to the play button)
      ii. Choose the current project (AssertDemo.java) from the list to the left
      iii. Choose the second tab "x=Arguments"
      iv. In the textarea labeled "VM arguments", add "-ea"
         1. "-ea" tells "java.exe" to "enable assertions"
      v. Click Ok or Apply and rerun AssertsDemo.java
   d. For any other editor (NetBeans, JGrasp, etc), simply Google the name of the editor and "how to enable asserts" for instructions.
3. Now with asserts enabled ("-ea"), your code should fail in the first two lines of asserts.
4. Look at and explain the first two asserts in main; which of the two is breaking the program?
   a. Comment out the offending assert line and rerun your code
5. In main, start by commenting out everything except what you're working on
6. Find the function warmUpAsserts() and look at the assert examples in the code
   a. At the end of this function, find the TODO and build two new assert code
7. Next, look at the function called assertWithPrimitives() and uncomment

    a. At the end of this method, find the TODO and build a couple of new asserts for use with primitives.
8. In the third function assertWithObjects(), notice how we still construct boolean expressions for use in the assert statement.
    a. In place of "==" for primitives, we observe ".equals()" or ".compareTo()".
    b. Find the checkAddress() function and explain in comments what it does
        i. To what address does "this" map to?
        ii. To what address does "input" or "ad" map to?
            1. What does this mean? Explain this in your comments.
    c. At the end of this subroutine, build two new asserts that test Objects.
        i. How would we test immutable objects? How is this similar to primitives?
        ii. When building these new asserts, consider testing Objects from previous labs, such as a Point2D or a Date.
9. Finally, examine the homeworkRelatedAsserts() function.
    a. Explore the code already provided that could help you test your homeworks.
    b. Build two new asserts relative to the homework you're currently working on.

Reflection and Analysis Questions

Put the answers to this questions in comments inside of your code (**AssertDemo.java).** Of the debugging techniques covered here…

(1) Which technique do you prefer?
    a. Why?
(2) Are some debugging techniques more (or less) appropriate for longer programs?
(3) What are the advantages to using a debugger with a GUI?
    a. What can you inspect here that you couldn't when just printing to the console?
(4) How can proper documentation help in finding and avoiding bugs?

** This lab is almost entirely based on Rob Nash's solution with minor modifications by A.Retik*