CSCE 155 - Java

Lab 8.0 - Debugging

Prior to Lab

Before attending this lab:

1. Read and familiarize yourself with this handout.
2. Review the following tutorial on debugging Java using Eclipse:
   http://www.vogella.com/articles/EclipseDebugging/article.html

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- Be able to understand the error messages generated by the Java compiler and Eclipse
- Understand the types of errors that occur in a program
- Debug a program using Eclipse's debugger

2 Background

During these first few weeks of classes, you've likely had an error in one of your programs. The error might have been something that made your program produce incorrect output, crash while it was running, or even fail to compile at all. Unfortunately, errors like these are common, and it's a rare (and fantastic) moment when code gets written the first time without any errors. Fortunately, because errors are so common, there are a myriad of tools designed to ease the pain of correcting them. Errors tend to fall into one of three categories: compilation errors, runtime errors, and logic errors.

Compilation Errors

Compilation errors occur when the compiler encounters something that it doesn't understand in the code you're attempting to compile. This will cause the compiler to stop and print a message to the screen describing the problem that it encountered. An example might look something like:

```
PaycheckCalc.java:52: cannot find symbol
symbol  : variable grossPay
location: class PaycheckCalc
        grossPay = payRate * hoursWorked;
        ^
```

The output indicates the filename where the error occurred, followed by the line number, and a message about the problem. Thus, the compiler in this case was unable to finish and the Java bytecode for this program was never produced.

Runtime Errors

Runtime errors are errors that cause the program to crash (i.e., end in an unexpected place and manner). Depending on the operating system you're running the program on, a variety of error messages might be displayed.

In Java, an Exception can also be an indication of a problem that occurs during a programs execution. A common runtime error is an `ArrayIndexOutOfBoundsException` that Java will throw when a program attempts to access an element outside of the bounds of an array.

The compiler will not catch these errors, but this lab will give you a couple of methods to help determine where and why the error occurred.

Logic Errors

Logic errors are errors that cause the program to operate incorrectly (such as producing incorrect output) but will compile and run without error. An example of such an error would be writing a function `add(int x, int y)` in the following manner:

```
1  public int add( int x, int y ) {
2    return x * y;
3  }
```

The code above would compile and execute without error, but obviously when you call a function named `add`, you expect it to return the sum of the numbers and not the product.

3 Activities

There are many tools and techniques used to detect and fix bugs. This lab will familiarize you with several of these tools and techniques for debugging Java programs. Clone this lab's project code from GitHub using the following URL: https://github.com/cbourke/CSCE155-Java-Lab08.

3.1 Compilation & Syntax Errors

For this part we're going to attempt to compile a Java program from the command line.

1. Login to CSE through PuTTY or SSH, and navigate to the folder where you stored `PaycheckCalc.java`

2. From the cse prompt attempt to compile the program by typing: `javac PaycheckCalc.java`
3. Read the errors that the compiler gives you, but do not fix them. Look at the code and with your partner identify all the syntax/compiler errors in the actual code.
4. Answer the questions on your worksheet (there are still runtime errors that we'll fix in the next part).

3.2 Fixing Runtime Errors in an IDE

IDEs such as Eclipse automatically compile and build the project as you type code and provide immediate feedback and can even suggest potential ways to fix the problem.

1. Return to Eclipse. The syntax errors should be more obvious in Eclipse's code editor: they are highlighted in red. Hover your cursor over a few of them and observe that Eclipse tells you what the error is and offers several suggestions on how to fix them. Do not fix them directly, instead choose the appropriate suggested fix from Eclipse and observe what was changed
2. Run the program and enter some input. Observe that there are still several runtime errors still to be fixed. When an error is encountered, a stack trace is printed to the standard error (in red) in the console window. The line(s) that caused the error are highlighted in blue and clickable. Click each one and fix the runtime errors.
3. Demonstrate your working program to an instructor and answer the questions in your worksheet.

3.3 Using a Full Featured Debugger

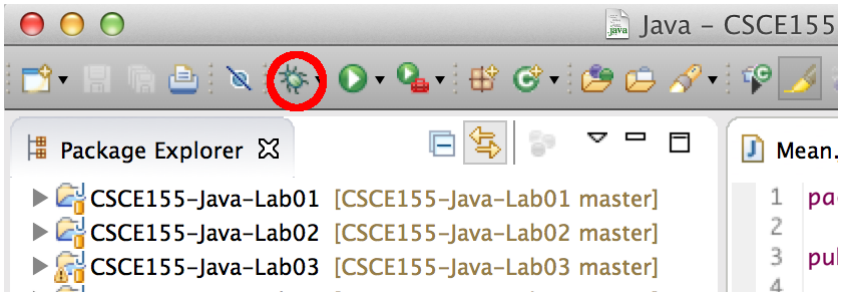
Eclipse is a full featured development environment that provides several tools to aid the programming activity. The workbench is the entire desktop development environment. It is comprised of several windows (called views) that enable a programmer to see the resources and features of their project. A perspective in Eclipse is simply a pre-specified layout of different views. Most likely up to this point you've been using the Java perspective. For the remainder of this lab we're going to use the Debug perspective. Eclipse provides several different perspectives depending upon which resources a programmer wants to see while working on a project. You are encouraged to explore (or customize) the other perspectives and find one that you prefer most. Note that changing the perspective does not change your code or project; it simply displays a different layout of your project with different views.

In the next few exercises we will use Eclipse's built-in debugger to find and fix several runtime errors.

1. Open the `Mean.java` source file

2. Run the program and observe the resulting error (clicking the errant line will not provide immediate help)

3. Rerun the program in Debug mode by clicking the Debug icon:



Eclipse will prompt you to switch to the Debug Perspective, select yes.

4. To understand what the problem is; set a breakpoint on line 9 (double click the left side of the code editor on line 9 until you see the small dot)

5. Run the program in debug mode: click the "bug" next to the play button. The program will execute until it reaches the breakpoint you set.

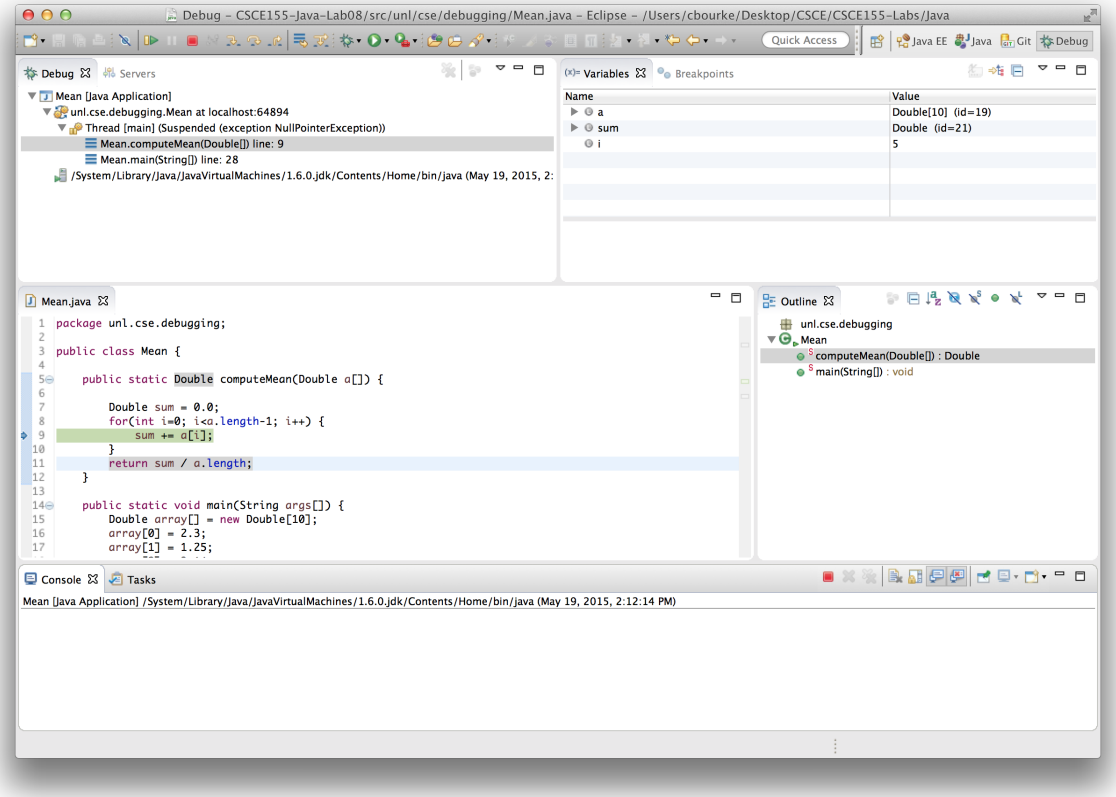


Figure 1: Eclipse Debug Perspective

6. Observe the windows (views) and tools (in the toolbar) you have available to you. An example screen shot can be found in Figure 1.
    - The upper left contains the program's stack (the order of methods that have been called so far)
    - The upper right contains all local variables and their values; clicking on them also displays their string representation
    - The code (and its current line) are displayed at the left

Figure 2: Eclipse Debug Toolbar

Debug controls are provided in the toolbar of the program stack window as depicted in Figure 2. As marked in the figure, from left-to-right, each icon does the following:

- The Play button resumes the program until the next breakpoint
- The Stop button aborts the program execution
- The "Step into" arrow steps to the immediate next command
- The "Step Over" arrow resumes execution to the end of the next command (so if a method is called it will execute the entire method)

7. Step through the program (click the step over or step into button) and observe what value the index i takes prior to when an exception is thrown. Deduce what the problem is and fix it.

The program still contains a logic error, though it is not apparent.

1. Manually add the values and verify that the answer the program gives is incorrect.
2. Run the debugger again and step through the entire for-loop. What value of `i` does the loop end on?
3. Fix the problem and demonstrate the debugged program to a lab instructor.

The debugger can also be used to detect where a program may get stuck.

1. Change your perspective back to Java (upper right corner)
2. Drag and drop the TenHellos.java program into the same package as before
3. Run the program (it is intended to print `"Hello!"` ten times) and observe that it does not execute

4. Kill the program by clicking the "stop" button in the console window
5. Set an appropriate break point to see where the program is getting stuck
6. Fix the program and demonstrate it to a lab instructor

4 Handin/Grader Instructions

1. Hand in your completed files:
   - `Mean.java`
   - `TenHellos.java`
   - `worksheet.md`

   through the webhandin (https://cse-apps.unl.edu/handin) using your cse login and password.
2. Even if you worked with a partner, you *both* should turn in all files.
3. Verify your program by grading yourself through the webgrader (https://cse.unl.edu/~cse155h/grade/) using the same credentials.
4. Recall that both expected output and your program's output will be displayed. The formatting may differ slightly which is fine. As long as your program successfully compiles, runs and outputs the *same values*, it is considered correct.

5 Advanced Activity (Optional)

Eclipse like all IDEs can make programming much easier. Challenge yourself by using a command line debugger for Java; specifically jdb. See the following link: (http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/jdb.html) and run through this lab again using jdb.