CODING, TESTING, DEBUGGING IN JAVA

It would be nice if, having developed an algorithm for your program, you could relax, press a button, and get a perfectly working program. Unfortunately, the process of turning an algorithm into Java source code doesn't always go smoothly. And when you do get to the stage of a working program, it's often only working in the sense that it does **something**. Unfortunately not what you want it to do.

After program design comes coding: translating the design into a program written in Java or some other language. Usually, no matter how careful you are, a few syntax errors will creep in from somewhere, and the Java compiler will reject your program with some kind of error message. Unfortunately, while a compiler will always detect syntax errors, it's not very good about telling you exactly what's wrong. Sometimes, it's not even good about telling you where the real error is. A spelling error or missing "{" on line 45 might cause the compiler to choke on line 105. You can avoid lots of errors by making sure that you really understand the syntax rules of the language and by following some basic programming guidelines. For example, I never type a "{" without typing the matching "}". Then I go back and fill in the statements between the braces. A missing or extra brace can be one of the hardest errors to find in a large program. Always, always indent your program nicely. If you change the program, change the indentation to match. It's worth the trouble. Use a consistent naming scheme, so you don't have to struggle to remember whether you called that variable `interestrate` or `interestRate`. In general, when the compiler gives multiple error messages, don't try to fix the second error message from the compiler until you've fixed the first one. Once the compiler hits an error in your program, it can get confused, and the rest of the error messages might just be guesses. Maybe the best

advice is: Take the time to understand the error before you try to fix it. Programming is not an experimental science.

When your program compiles without error, you are still not done. You have to test the program to make sure it works correctly. Remember that the goal is not to get the right output for the two sample inputs that the professor gave in class. The goal is a program that will work correctly for all reasonable inputs. Ideally, when faced with an unreasonable input, it should respond by gently chiding the user rather than by crashing. Test your program on a wide variety of inputs. Try to find a set of inputs that will test the full range of functionality that you've coded into your program. As you begin writing larger programs, write them in stages and test each stage along the way. You might even have to write some extra code to do the testing -- for example to call a subroutine that you've just written. You don't want to be faced, if you can avoid it, with 500 newly written lines of code that have an error in there *somewhere.*

The point of testing is to find bugs -- semantic errors that show up as incorrect behavior rather than as compilation errors. And the sad fact is that you will probably find them. Again, you can minimize bugs by careful design and careful coding, but no one has found a way to avoid them altogether. Once you've detected a bug, it's time for debugging. You have to track down the cause of the bug in the program's source code and eliminate it. Debugging is a skill that, like other aspects of programming, requires practice to master. So don't be afraid of bugs. Learn from them. One essential debugging skill is the ability to read source code -- the ability to put aside preconceptions about what you *think* it does and to follow it the way the computer does -- mechanically, step-by-step -- to see what it really does. This is hard. I can still remember the time I spent hours looking for a bug only to find that a line of code that I had looked at ten times had a "1" where it should have had an "i", or the time when I wrote a subroutine named`WindowClosing` which would have done exactly what I

wanted except that the computer was looking for `windowClosing` (with a lower case "w"). Sometimes it can help to have someone who doesn't share your preconceptions look at your code.

Often, it's a problem just to find the part of the program that contains the error. Most programming environments come with a debugger, which is a program that can help you find bugs. Typically, your program can be run under the control of the debugger. The debugger allows you to set "breakpoints" in your program. A breakpoint is a point in the program where the debugger will pause the program so you can look at the values of the program's variables. The idea is to track down exactly when things start to go wrong during the program's execution. The debugger will also let you execute your program one line at a time, so that you can watch what happens in detail once you know the general area in the program where the bug is lurking.

I will confess that I only occasionally use debuggers myself. A more traditional approach to debugging is to insert debugging statements into your program. These are output statements that print out information about the state of the program. Typically, a debugging statement would say something like

```
System.out.println("At start of while loop, N = " + N);
```

You need to be able to tell from the output where in your program the output is coming from, and you want to know the value of important variables. Sometimes, you will find that the computer isn't even getting to a part of the program that you think it should be executing. Remember that the goal is to find the first point in the program where the state is not what you expect it to be. That's where the bug is.

And finally, remember the golden rule of debugging: If you are absolutely sure that everything in your program is right, and if it still doesn't work, then one of the things that you are absolutely sure of is wrong.

Source : http://math.hws.edu/javanotes/c3/s2.html