**ISTE120-01 Computer Problem Solving in the Information Domain I**

Fall 2018 Semester

**Lab 13: Debugging and tracing**

# Background

Have you ever heard that unforgettable children's song *100 bottles of pop* on the wall? Your task is to write a program that will "sing" a few verses of this song. Here's an example [recording on Youtube](http://www.youtube.com/watch?v=L5Oy3w4fBm0), but beware it may get stuck in your head the rest of the day!

# Objectives

Step through code using a debugger. Trace execution ﬂow with a debugger.

Compare iteration with recursion.

# Part 1: Bottles of Pop

Download Bottles.java as a starting point for the lab. Write your name and today's date in the Javadoc comment. Here is what the output should look like when you ﬁnish the lab:

100 bottles of pop on the wall 100 bottles of pop

If one of those bottles should happen to fall 99 bottles of pop on the wall

99 bottles of pop on the wall 99 bottles of pop

If one of those bottles should happen to fall 98 bottles of pop on the wall

*(repeat until 1)*

1. Most of the algorithm has already been implemented in Bottles.java. Figure out how to complete the sing method.

**Do NOT edit any of the other methods.**

1. Test your solution on a small number of verses. You may want to run this lab from the command line (i.e., java Bottles 5) for convenience.

Is it working right? If not, go back and ﬁx it! Do you have a blank line after each verse?

1. Modify your code to print "1 bottle" instead of "1 bottles" when applicable.

Don't forget the ending of the "2 bottles of pop" verse.

1. Modify your code to print "No more bottles" instead of "0 bottles" when applicable.
2. Make sure your solution works in the following cases. Test each one via the command line.

java Bottles 0 *nothing should print* java Bottles 1 *the last verse prints* java Bottles 5 *only ﬁve verses print*

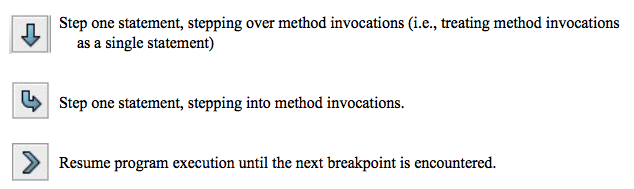
# Part 2: Debugging with JGrasp

Most [integrated development environments](http://en.wikipedia.org/wiki/Integrated_development_environment) (like JGrasp, DrJava, Eclipse, and NetBeans) come with a debugger that allows you to "step through" your code while it is running.

1. Move your cursor to the following line of code in JGrasp (approximately line 58):

if (args.length > 0) {

1. Make sure that the menu item **Build – Debug Mode** is checked. Compile all source code files required by your program.
2. Set a breakpoint (to paus program execution) by left-clicking at the left edge of the line where you want the program to pause on Current Line. Note, you may set as many breakpoints as you want.
3. Run your program in the debugger by clicking on the  button (located next to the regular run button). Your program should stop on the if statement.
4. At this point, you can take any of the following steps from the Debugger menu (or use the keyboard shortcuts):



1. Practice stepping through your code one line at a time. Each time the program ends, you will need to restart the debugger.
2. jGrasp has an "Automatic Trace" feature that animates the entire execution of a program. Can you ﬁgure out how it works? Record the steps to make it work here.

Running the debugger and watching the variable and thread window

1. Change the main method to invoke recursive instead of iterative. Then trace the program execution using the debugger. How many times does the recursive method return?

10

# Part 3: Using Debugger

1. Depending on your development environment, create either a directory or a project for this part of the lab.
2. Setup your development environment by including AreaCalculator.java, Geometry.java, and JMUConsole.class in an appropriate directory/folder.
3. Run jGrasp. “Clean up” jGrasp by clicking on **File** and pulling down to **Close All + Windows**.
4. Open Geometry.java by clicking on **File**, pulling down to **Open,** and selecting the appropriate file and directory.
5. Open AreaCalculator.java in the same fashion.

**Review:** This part of the lab will review a few topics related to programming in Java.

1. Where is the **source code** for the AreaCalculator class?  
   AreaCAulator.java
2. Where is the source code for the Geometry class?

Geometry.java

1. Where is the source code for the JMUConsole class?  
   We don’t have it
2. Do you need the source code or the byte code to **execute** a program?  
   byte code

1. Where can you find information about the methods in the JMUConsole class?  
   JMUConsole.pdf

**Setting-Up jGrasp for Debugging:** Debuggers require special version of .class files that are created by “compiling in debug mode.” This part of the lab will help you learn how to do this.

1. To make JGrasp compile in debug mode, click on **Build** on the menu bar and “check” **Debug Mode** (if it is not already “checked”).
2. Click on the “button” for Geomtry.java to make sure that it has the focus.
3. Click on **Build** and pull down to **Compile**.
4. Click on the “button” for AreaCalculator.java to make sure that it has the focus.
5. Click on **Build** and pull down to **Compile.**

**Setting a Breakpoint:** One of the nice things about running an application in a debugger is that you can stop the execution at one or more pre-defined locations (called *breakpoints)*. This part of the lab will help you become comfortable working with breakpoints.

1. Click on the “button” for AreaCalculator.java to make sure that it has the focus.
2. Click on line 25 of AreaCalculator.java (the if statement) to move the cursor to that line. Note, don’t worry if you don’t yet know what the if statement does, it doesn’t matter at this point and we’ll discuss it soon.
3. Right-click on line 25 of AreaCalculator.java and pull down to **Toggle Breakpoint**. Note, you can also set a breakpoint by clicking on the left-most edge of a line.
4. What happened?  
   created a breakpoint icon
5. Click on **Build** and pull down to **Debug** and enter 4 when prompted.
6. What happened?  
   nothing but push me to a new line

**Checking State Information:** Another nice thing about running an application in a debugger is that, once you stop the execution at a breakpoint, you can check state information (e.g., the value of attributes and variables). For example:

1. Check on the **Variables** tab on the left side of the jGrasp Window.
2. Click on the “expand icon” next to **Locals** to expand it, if it isn’t already expanded.
3. What is the current value of shape?  
   4
4. Enter an expression involving shape in the “Eval” tab (e.g., shape/2).
5. Note that the value of this expression will change whenever shape changes. How might this kind of capability help when debugging? Helps you restrict where errors can be. This can be used to check your entered data when before assigning it.
6. Click on the “Variables” tab.

**Stepping Over Lines:** When running an application in a debugger, once you stop the execution at a breakpoint, you can continue the execution one “step” at a time. This part of the lab will help you learn how to do this.

1. Click on the  button.
2. What happened?  
   Pushed me out of the if else statement
3. Why did it move to this line?  
   It had completed all it needs to do within that if else
4. Click on the  button to run to the end of the application and respond to both prompts with the value 10.

**Stepping Into Lines:** So far, all of the “stepping” you have done has been in one method in one class. This is called “stepping over.” You can also “step into” a line of code to see what happens there. This part of the lab will help you learn how to do this.

1. Click on **Build** and pull down to **Debug**. Enter 0 when prompted.
2. Click on the  button twice. You should be prompted for the diameter. Enter 10 and click on .
3. What happened?  
   it pushed you to the nextline down and calulates the area
4. Now instead of clicking on the  button, click on the  button instead.
5. What happened?  
   it adds JMUConsole.readDouble to the stack
6. Look at the “Call Stack.” It tells you what class and method you are in and where this method was called from.
7. What method is currently being executed and what class is it in?  
   readDouble() its in JMUConsole class
8. What line is currently being executed?  
   28
9. Where was this method called from?  
   AreaCalulator main
10. Click on the “Variables” tab.
11. What are the current values of the local variables?  
    n/a
12. Click on the  button, watching how the local variables change, until control returns from the circleArea() method in Geometry class to the main() method in the AreaCalculator class.
13. What is the current value of area in the main() method in the AreaCalculator class?  
    its not storing anything until the other method from Geomtry returns
14. Click on the  button. What is the current value of area?  
    78.54
15. Click on the  button.
16. Run the program again, but this time, try and step into the statement that calls JMUConsole.print().
17. What happened?
18. Why?
19. Click on the  button and complete the run.

**Stepping to the Cursor:** In this part of the lab you will learn how to step to a particular line without setting another breakpoint.

1. Click on line 40.
2. Click on the  button. This will run the application to the cursor.
3. What happened?  
   displays what the area is but its editable
4. Click on the  button to run to the end.
5. How else could you have accomplished the same thing?

More breakpoints

**Gaining Experience with the Debugger:** To understand the value of a debugger, you need to use it to trace the execution of a larger program than the one used in the previous sections. This part of the lab will give you some experience with this.

1. Does a debugger actually debug code?   
   it doesn’t directly debug code but helps the creator follow the program easily
2. What does a debugger do?  
   helps the user understand the complexity of the code in pieces with step by step directions
3. You may have been unable to remove all of the errors from the code you wrote for a programming assignment. If so, use the debugger to step through your code to try and identify the errors. Obviously, you should execute the code on a test that is failed, not a test that is passed. If all of your code you have written thus far has not contained any errors, use the debugger to step through the most complicated program you’ve written this semester.

# Part 4: CodingBat Problems

For the remainder of today's lab, see how many of the [Recursion-1](http://codingbat.com/java/Recursion-1) problems you can solve. **Remember to log into CodingBat ﬁrst so you will receive credit, and verify your name is properly in the memo field.** At a minimum, solve the following: [sumDigits](http://codingbat.com/prob/p163932) and [countX](http://codingbat.com/prob/p170371). Be sure to read the paragraph at the top of the [Recursion-1](http://codingbat.com/java/Recursion-1) page.

# Part 5: Submission

* Submit your completed Bottles.java ﬁle and your Lab13 document with your responses to the question above via myCourses **by the end of the class** **today (@ 1:50pm), late submissions will NOT be accepted**.
* About half of your lab grade will be based on your CodingBat results which are due by **tonight @ 11:59pm**.