Lab 7 Methods and IntelliJ

Part 1 - Entering and Running JAVA programs using IntelliJ

1) Create an IntelliJ project called **CMSC255**

2) Navigate to the scr folder to create a package named **Labs** (Right click on the src folder, select new, then package.)

3) Navigate to the **Labs** package and create another package named **Lab7** inside it.

4) Download **Lab7.java** (If clicking Lab7.java opens a new tab rather than a download window, right click the Lab7.java link and select "Save Link As" – for Firefox and Chrome users, "Download Linked File" – for Safari users.)

5) Navigate to the **Lab7** package and drag and drop **Lab7.java** into the **Lab7** package. You should now see the code inside **Lab7.java**.

6) At this point you should see three words **greeting**, **max**, and **sumTo** colored red. This you will fix soon. Do not run your code at this point since you are calling methods that do not exist.

7) Write a void method called **greeting** which takes three String parameters and formats and prints a title, first name and last name in the following format, using println and prints it out:

   blank line
   Dear <title> <first name> <last name>,
   blank line

   Note that your method goes after the ending curly brace for the main method, but before the ending curly brace for the Lab7 class. Also note that the color red for greeting will go away once this method is created.

8) Write a method called **max** that takes two int parameters, num1 and num2, and returns the larger of the two integers. If the integers are equal, return either integer.

9) Write a method called **sumTo** that takes two int parameters, num1 and num2, and returns the sum of all the integers from the smaller of the two integers to the larger of the two inclusively. (i.e. Given 2 and 4, the method will return 9, since 2 + 3 + 4 = 9.) If the two integers are equal, return that integer value. (i.e. Given 5 and 5, return 5.)

10) Compile and run your code making sure to note that your methods are running correctly with the data you enter. Correct any errors.

11) Download, drag and drop the **Lab7Test** file provided into **Lab7** package. (So that **Lab7Test** and **Lab7** are in the same package) You will need to add JUnit 4 to the classpath to make this file run. (You can do this by locating the first import statement, clicking on the word "JUnit" which will be red text. Once clicked, press the option and return keys. This will bring up an option to import JUnit

4.) Run the JUnit **Lab7Test** file. Use this file to correct any errors you might have in **Lab7.java**. Do not change any code in the **Lab7Test** file.

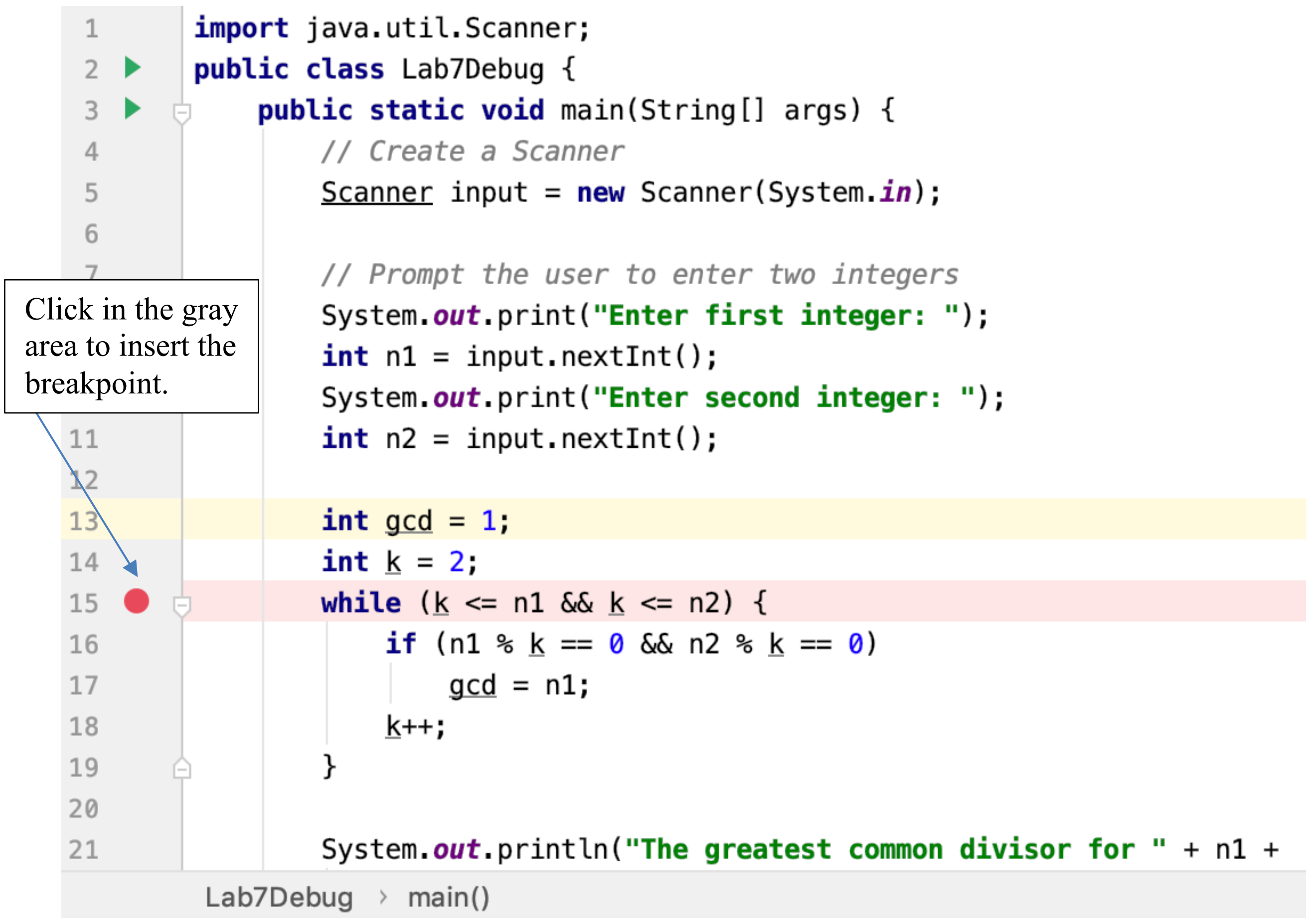
12) Take a screenshot of the **Lab7Test.java** file running without any errors.

Lab 7 Part 2 – Debugging your program using IntelliJ

1) Download and drag the **Lab7Debug** file provided into **Lab7** package

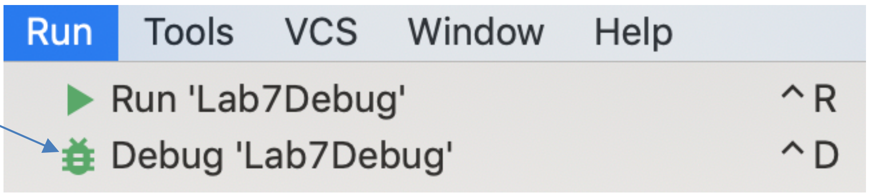
2) Debugging allows a programmer to step though their program one line at a time, allowing a programmer to clearly see the effect each line of code has on their program. A breakpoint is a signal that tells the debugger to temporarily suspend execution of your program at a certain point in the code.

3) To debug **Lab7Debug.java**, place a breakpoint in your code at the line, `while (k <= n1 && k <= n2)`. Here is a tutorial page to help you with this: https://www.jetbrains.com/help/idea/using-breakpoints.html

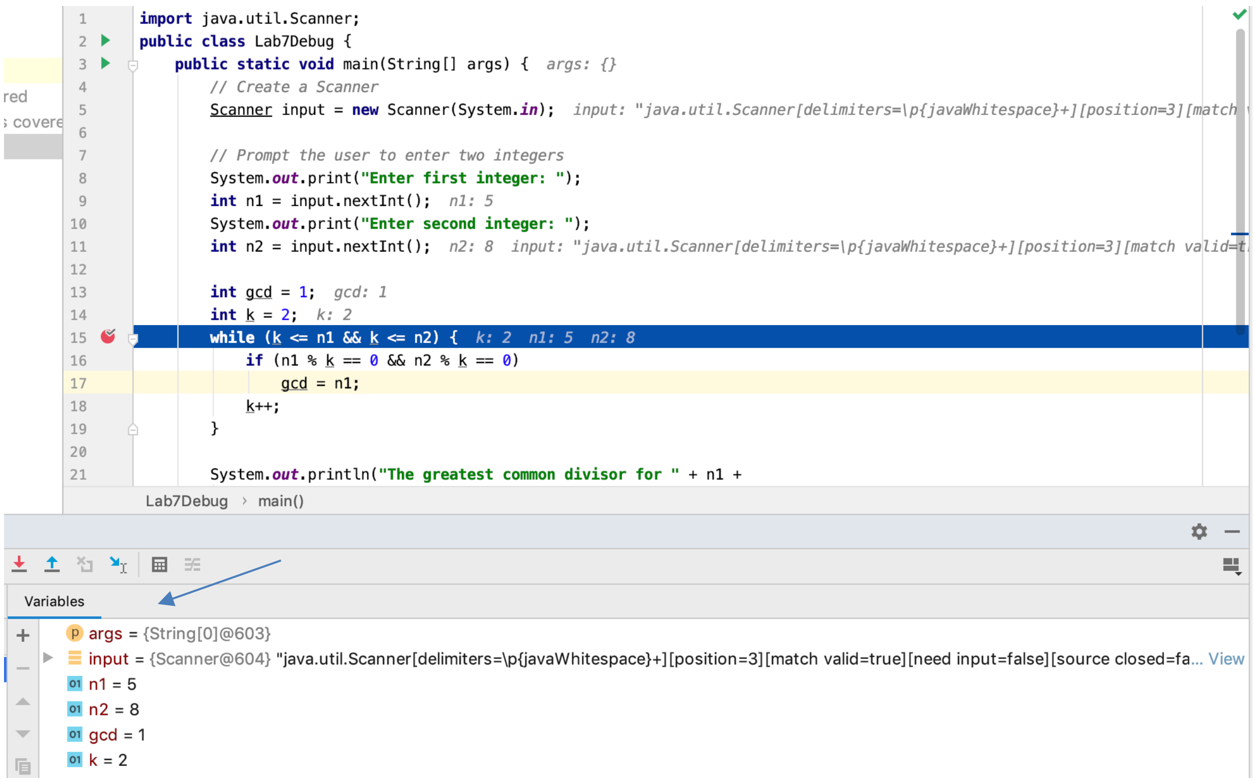


```java
import java.util.Scanner;
public class Lab7Debug {
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter first integer: ");
        int n1 = input.nextInt();
        System.out.print("Enter second integer: ");
        int n2 = input.nextInt();

        int gcd = 1;
        int k = 2;
        while (k <= n1 && k <= n2) {
            if (n1 % k == 0 && n2 % k == 0)
                gcd = n1;
            k++;
        }

        System.out.println("The greatest common divisor for " + n1 +
```

4) Select the line then go to **Run > Debug Lab7Debug**.



5) Before hitting the break point, your program will execute normally. When your program encounters your break point, IntelliJ will stop execution. It will then switch to the variables tab, where you can see what variable values each variable up to this point in the program has. This view can help you understand what is happening with your variables. Here is more from IntelliJ on running the Debugger: https://www.jetbrains.com/help/idea/pausing-and-resuming-the-debugger-session.html



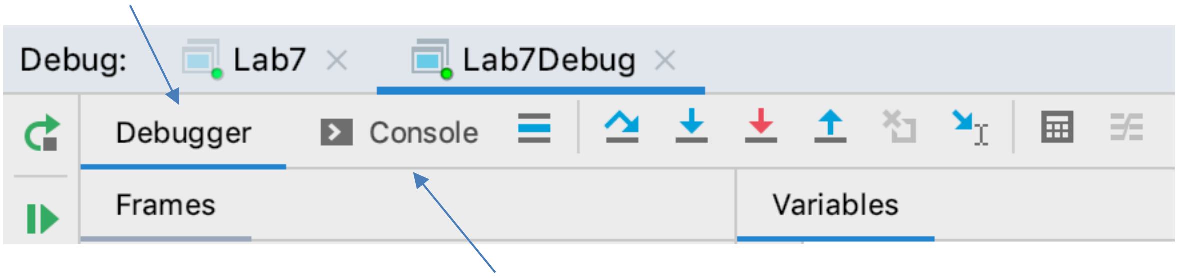
```
import java.util.Scanner;
public class Lab7Debug {
    public static void main(String[] args) {  args: {}
        // Create a Scanner
        Scanner input = new Scanner(System.in);  input: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=3][match

        // Prompt the user to enter two integers
        System.out.print("Enter first integer: ");
        int n1 = input.nextInt();  n1: 5
        System.out.print("Enter second integer: ");
        int n2 = input.nextInt();  n2: 8  input: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=3][match valid=t

        int gcd = 1;  gcd: 1
        int k = 2;  k: 2
        while (k <= n1 && k <= n2) {  k: 2  n1: 5  n2: 8
            if (n1 % k == 0 && n2 % k == 0)
                gcd = n1;
            k++;
        }

        System.out.println("The greatest common divisor for " + n1 +
```

Lab7Debug › main()

Variables

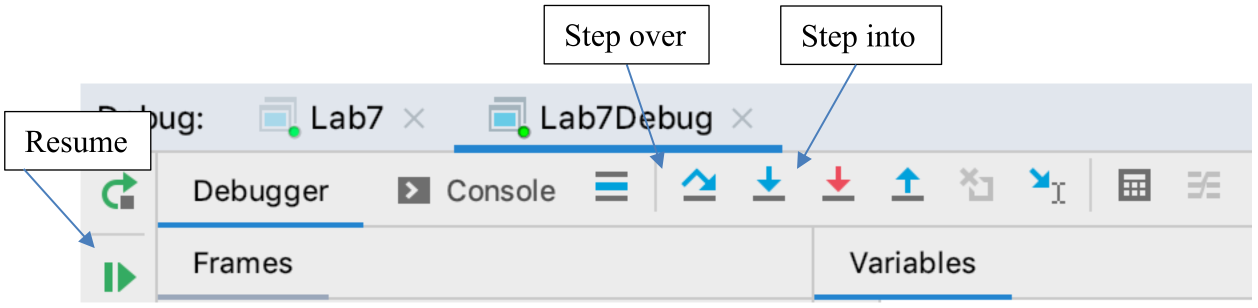
```
args = {String[0]@603}
input = {Scanner@604} "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=3][match valid=true][need input=false][source closed=fa... View
n1 = 5
n2 = 8
gcd = 1
k = 2
```

6) You can manually **switch between perspectives**, the **console**, where you see what you inputted and the output to the screen and the variable, **debugger**, perspective.



7) Stepping Commands
- There are three very important commands controlling the debugger. These are:

- **Step into**, which executes the next statement. If this statement has a method then the line of execution will move to the first statement inside this method.
- **Step over**, which executes the next statement. The line of execution moves to the next statement in the current method.
- **Resume**, which continues the execution of the program either until a breakpoint is found or the program terminates.
- To manually **switch between perspectives**, you can use the debugger and console buttons.



8) With debugging, we can see the state of our application at any given point during its execution. The state of the application is the current line of code being executed, all defined variables and the values stored in those variables. This is an incredibly powerful tool, as it allows us to step though code line by line and see where the actual state of the program deviates from the expected state. The variable window in the debugging perspective is vital for debugging.

9) You will walk through the loop in your **Lab7Debug** file to see if you can find the error in the code. If you continue to press the resume icon, it will loop through to the next loop iteration. You will need to continue to hit the resume icon until the looping ends. Once you have found the error, correct the code and switch back to the Console perspective.

10) Run the corrected Lab7Debug file showing the correct output. Take a screenshot of this output.

11) To submit your files, you right-click on the files **Lab7.java**, and **Lab7Debug.java**, then select the **Reveal in Finder** menu option. (**Reveal in Explorer** for Windows users) This way you can find your files.

12) Submit your **Lab7.java** and **Lab7Debug.java** files to Gradescope. Make sure all the tests pass.

13) Submit your **Lab7.java**, an **Lab7Debug.java** files along with your two screenshots to the submission area in Canvas.