

# ITSC 1212 Lab 1 - VSCode & Getting Started with Programming

This lab assumes you have completed the introductory Lab 0. If you haven't, you will need to take some time to complete that lab so that your system is set up and ready for this activity.

## Concepts covered in this lab:

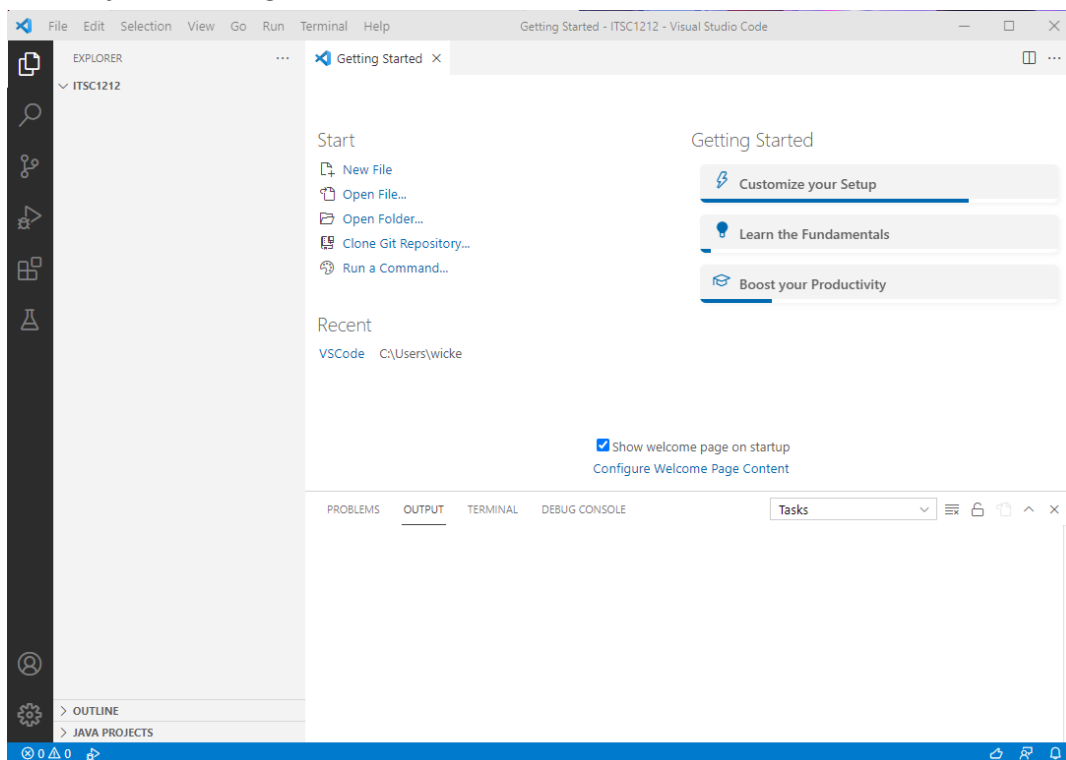
- Creating and running Java classes
- Creating integers and doubles
- Math with integers and doubles
- Flowcharts
- Debugging math errors

## Required files or links

- [draw.io](https://draw.io)

## Part A: Getting started in VSCode

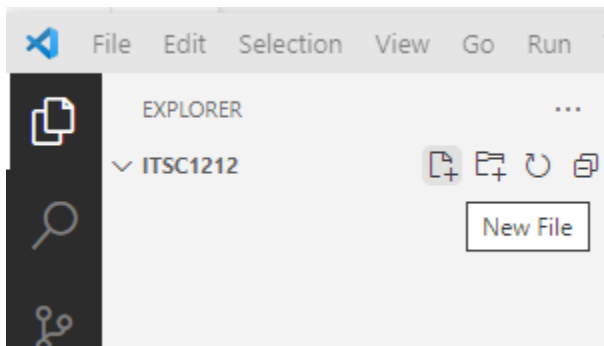
1. Start by launching VSCode. You should see a screen that looks like this.



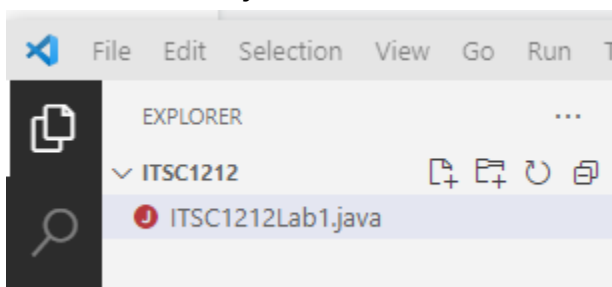
2. VSCode is a wonderfully powerful and versatile editor that can be used for lots of different programming languages, which you most likely

encountered during the installation process. During that process, you hopefully told the installer you would be using VSCode for Java, and it downloaded the necessary modules for that during the installation. If you didn't, you may encounter prompts during this lab to download those modules. Make sure you do those when prompted, and don't hesitate to ask for help if you get stuck.

3. On the right side of the Getting Started screen, you'll see the "Customize your Setup", "Learn the Fundamentals", and "Boost your Productivity" options. These can be great, quick introductions to this editor, and we highly recommend taking a few minutes to watch the video under "Learn the Fundamentals" -> "Lean back and Learn" to help you get acquainted with some of the intricacies of VSCode.
4. Once you're feeling comfortable, Look under "Start" to find the "Open Folder" menu option. In the window that pops up, find and choose the folder you created during Lab 0. Once you've done that, you should now see the folder name in the Explorer sidebar on the left hand side of the screen (You can see what this should look like in the screenshot under step 1). By opening this folder, we have set a workspace where we can easily create and access all the files we'll use this semester.

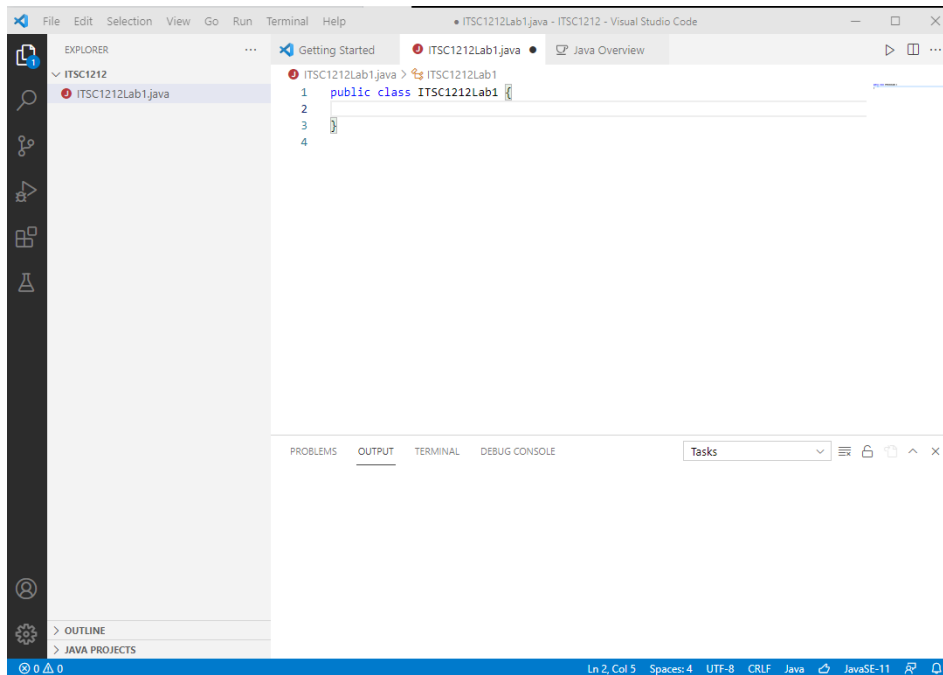


5. Move your cursor to the file name, and notice that several icons will appear. Click on the New File icon, and in the window that pops up type in ITSC1212Lab1.java. Then hit enter.



**\*\*NOTE: File names should never include spaces\*\***

- When you hit enter, two new tabs will appear next to Getting Started. The first one is the file we created, the second one is an introduction to some of the different Java functions that VSCode can do. Most of these functions are fairly advanced topics that you will be introduced to in later courses. If the Getting Started tab didn't appear, that's okay. For now, let's focus on the file we created. It should hopefully look something like this:



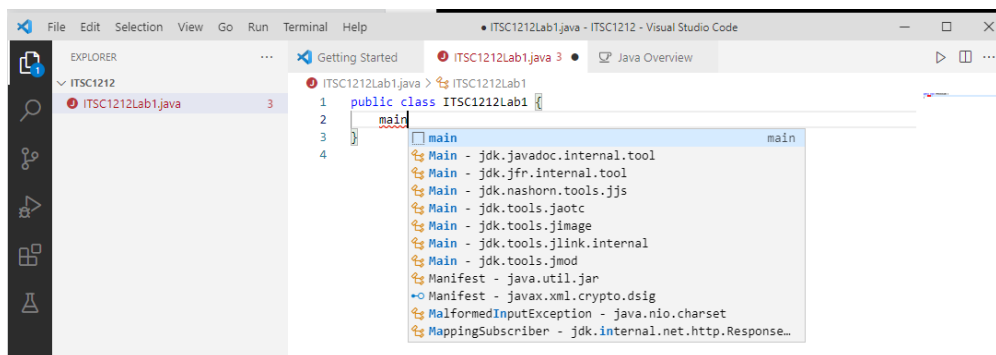
- Once you have this page, **save the file**, then proceed to Part B.

## Part B: Creating and Running your first Java Class

- Now that we have our first class created, there are some essential concepts we need to establish within this small bit of code VSCode wrote for us. Everything we write in Java is going to be contained within Classes. On line one, we see the **class declaration**. These lines will always be the start of our class files, and will always include the words “public”, “class”, and the name of the class. Notice that public and class are colored blue, and the class name is in black (or green, depending on your version/OS) text. These words, public and class, are **reserved keywords** in Java, meaning they have a set function and cannot be used for names. You will encounter many of these keywords during the semester, and VSCode will

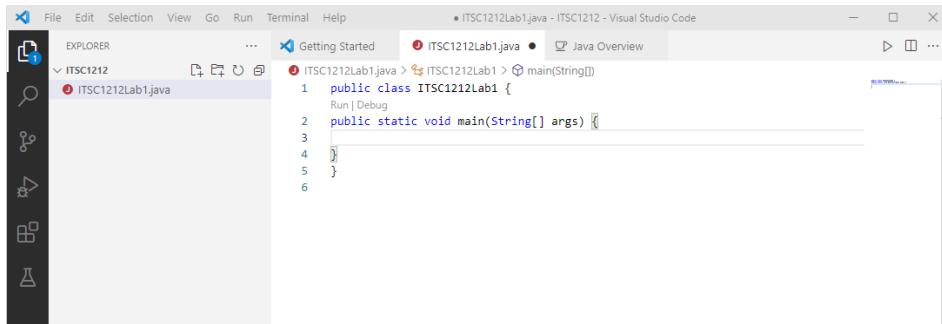
helpfully color these for you so you don't accidentally attempt to use them incorrectly.

2. The second thing we need to establish is incredibly important when it comes to file and class creation. You might have noticed that when VSCode created this file for you, it named your class the same as the file name. This was not by accident, it is actually a strict requirement. The name of the class **MUST** be the exact same as the file name, or the code will not work.
3. Lastly, notice the curly braces `{}` on lines 1 and 3. In Java, curly braces are used to enclose blocks of code. In this case, the curly braces mark the start and end of the code contained by our class. Everything we write must be written inside a class, therefore between our curly braces.
4. Now, we have a class, but it currently doesn't do anything and also isn't runnable. In order for a class to be runnable, it must contain something called a main method. We'll talk about methods a lot in the coming weeks, but in short, methods are executable blocks of code, and the main method is specifically the block that will execute when we press the Run button in VSCode. So, to make our class runnable we need to add a main method.
5. Click into Line 2, and type "main" like so



Notice that when you do, a window will pop up with a lot of confusing looking suggestions. One of the things VSCode will do to help you as you code is offer you auto-complete options, and this is what those look like. You will often see pop-ups like this, and it is extremely important you only choose options that correspond to what it is you are trying to write. Choosing an autocomplete option without fully understanding what it is you are choosing would be a serious mistake and will ultimately make life much harder for you. Even with something as simple as "main", there are a multitude of confusing looking options that aren't at all what we're looking for. The one we do want to use is the first one, so click on that. You can

also press the enter button on your keyboard when the option you want from the pop-up is highlighted.



6. After choosing the top option in the pop-up, you should now see that VSCode has written another set of code that starts with public and has its own set of curly braces. Just like with the class, anything we want to be a part of the main method has to be written between its set of curly braces. Let's test out this code by running it. You can do these several different ways in VSCode, either by pressing the Run Icon in the top right, or by pressing F5 (Windows), or by clicking the "Run" text that is hovering above the main method. No matter which method you go with, as soon as you Run the program lots of things will start happening all at once. Importantly, a lot of text will appear within the Terminal tab at the bottom. If you're not familiar with a Terminal, it is a command line interface, and also the way your code is executed. You won't, at least at the moment, see anything besides some confusing looking lines and directories, because our program doesn't yet have any output. Let's change that.
7. The first bit of code every one writes is called HelloWorld, and we wouldn't want to mess with that tradition. HelloWorld is a program that just outputs a message, "Hello World!". So let's add in a line of code to generate that output.
8. Type in the following line of text, between the curly braces of the main method: (Note: the character immediately after print is a lowercase "L")

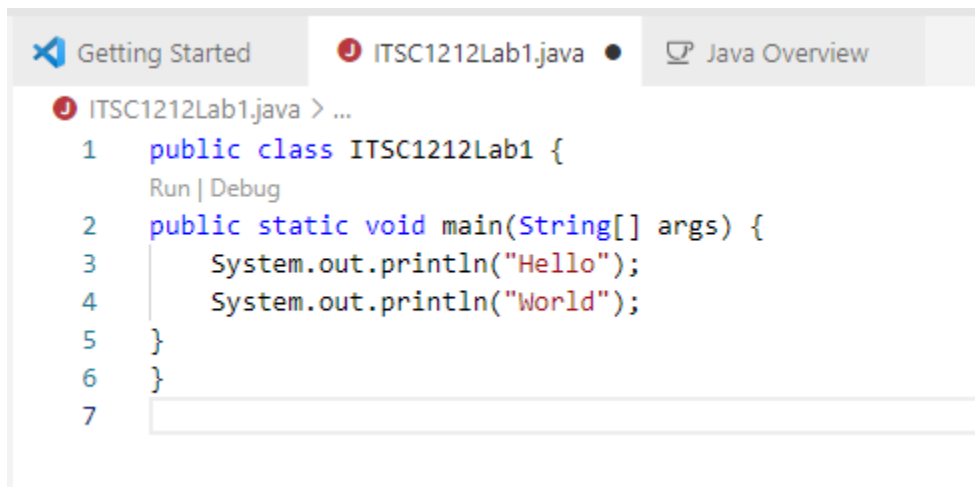
`System.out.println("Hello World");`

9. Run your program. The output in the terminal should now look something like this:

```
PS C:\Users\wicke\Documents\ITSC1212> cd 'c:\Users\wicke\Documents\ITSC1212'; & 'c:\Users\wicke\.vscode\extensions\vscjava.vscode-java-debug-0.34.0\scripts\launcher.bat' 'C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe' -Dfile.encoding=UTF-8 -cp 'C:\Users\wicke\AppData\Roaming\Code\User\workspaceStorage\863f2d6b3abd29c0bb1919bf8e22e4ab\redhat.java\jdt_ws\ITSC1212_17297f77\bin' 'ITSC1212Lab1'
Hello World
PS C:\Users\wicke\Documents\ITSC1212> 
```

You can see that after the commands to run our program, the text “Hello World” has appeared in the terminal window.

10. Let’s make one small change, and split “Hello” and “World” into two different **print statements**. To save time, rather than write out the entire `System.out.println`, simply type `sysout`, and then hit “Enter” to select the first option in the autocomplete pop-up. Once you have your two print statements and have moved the text around, your code should look this:



```
Getting Started ITSC1212Lab1.java Java Overview
ITSC1212Lab1.java > ...
1 public class ITSC1212Lab1 {
  Run | Debug
2 public static void main(String[] args) {
3     System.out.println("Hello");
4     System.out.println("World");
5 }
6 }
7 
```

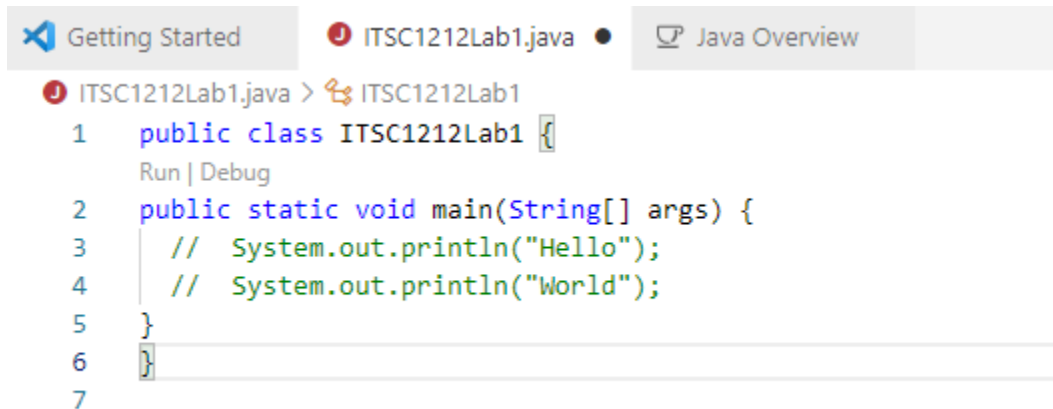
**\*\*Note: shortcuts like “sysout” vary depending on which IDE you’re using\*\***

11. Run your code again. What changed, and did that change match your expectation?
12. Remove the L and N that come after the word print, so that the line reads `System.out.print`, from both of your print statements then run your code again. What changed this time?
13. Run your code several more times, using different combinations of *print* and *println*. You may want to copy and paste your two lines so that you can test multiple combinations. Once you feel like you have a grasp on how to *print* and *println* differ, and how to combine them to produce different

outputs, show your work to an instructor or a TA to receive credit for parts A and B.

## Part C: Variables and Assignment

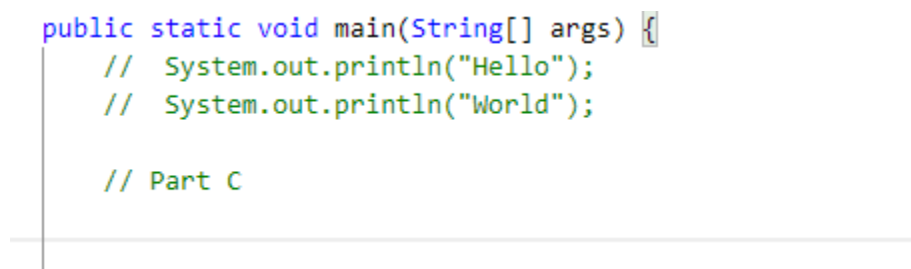
1. We've now written your first bit of Java code, and tested how different outputs can be formatted. But a program that just spits out some text isn't particularly useful or interesting. Before we proceed, let's **comment out** the lines from Part B. Add two forward slashes to the beginning of the lines of code from Part B, such that the lines now look like this:



The screenshot shows an IDE with a tab labeled 'ITSC1212Lab1.java'. The code editor displays the following Java code:

```
1 public class ITSC1212Lab1 {  
    Run | Debug  
2 public static void main(String[] args) {  
3     // System.out.println("Hello");  
4     // System.out.println("World");  
5 }  
6 }  
7
```

2. By adding these slashes, we've turned those from lines of code into **comments**. Comments are lines of text in your program that are ignored at runtime. Comments allow us to make notes to ourselves, others that may use our program, or to disable lines of code we want the system to ignore. Comments are incredibly useful and important, so get used to writing them. Add another comment below these new ones to mark the beginning of Part C, like this:



The screenshot shows the same Java code as before, but with an additional comment added below the previous ones:

```
public static void main(String[] args) {  
    // System.out.println("Hello");  
    // System.out.println("World");  
  
    // Part C  
}
```

3. Before we move on completely, let's think about Hello World one more time. Anytime we write a value directly into our program, whether it's a number or text, we call that a **literal**. Literals have a value, they are essentially whatever value we happened to have typed in, and the program can use that value to do things, but it isn't being stored in any way. If we want our program to remember that value and store it for later use, we have to assign that value to a **variable**.
4. In creating a variable, what we have done is create a memory location to hold a value, as well as a name we can use to reference that value. Let's create a variable and give it a value so we can demonstrate this more fully. Type this into your program below the Part C header.

```
// Part C  
int x = 5;
```

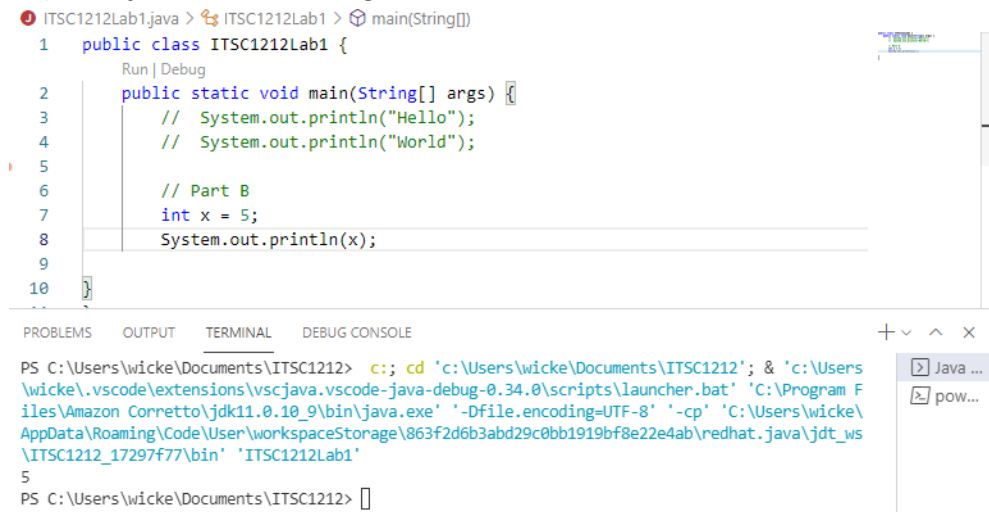
5. Within that line of code, we have created a variable called *x*, **declared** that it will be of type *int*, and **assigned** it a value of 5.
6. Declaration and assignment are the two essential steps of variable creation in Java. On the left side of the = is the declaration, where we give the variable a type and a name, and on the right side of the = is the assignment, where we give it a value. We must do both of these things before we can use that variable. We can break these steps into 2 lines, but we must have both parts before the variable use usable.

```
int x;  
x = 5;
```

7. Add in another print statement, but instead of writing a literal in the parentheses, put our variable *x* in instead. Run the program. You should



hopefully see something like this:



The screenshot shows a Java IDE with a code editor and a terminal window. The code editor displays the following Java code:

```
1 public class ITSC1212Lab1 {  
    Run | Debug  
2     public static void main(String[] args) {  
3         // System.out.println("Hello");  
4         // System.out.println("World");  
5  
6         // Part B  
7         int x = 5;  
8         System.out.println(x);  
9  
10    }  
}
```

The terminal window shows the command prompt output:

```
PS C:\Users\wicke\Documents\ITSC1212> c:; cd 'c:\Users\wicke\Documents\ITSC1212'; & 'c:\Users\wicke\vscode\extensions\vscjava.vsc...  
PS C:\Users\wicke\Documents\ITSC1212>
```

8. Since our variable holds a value of 5, anything we could do with a literal 5 we can do with our variable. To demonstrate that, copy the following code into your main method and use the output to fill in the chart below.

```
// Part B  
int x = 5;  
System.out.println("The value of x");  
System.out.println(x);  
System.out.println("The value of x + 2");  
System.out.println(x + 2);  
System.out.println("The value of x * x");  
System.out.println(x * x);  
System.out.println("The value of x / 2");  
System.out.println(x / 2);
```

Equation	Output
$x + 2$	
$x * x$	
$x / 2$	

Were each of the outputs what you expected? Probably not for the last one!

9. So why is the answer of  $(5/2)$  equal to 2 and not 2.5? Well, the answer is due to an unfortunate quirk of how math works in Java. The value of our variable  $x$  is 5, and we are dividing it by 2. Both the variable and literal in

this case are of type *int*. Ints, along with several other variable types, are only capable of holding whole numbers. Because both of the values in the equation are whole numbers, Java assumes we want our answer to be a whole number as well and will do the math under that assumption. Since we have no way to capture the decimal point, the decimal from 2.5 is dropped, and our final answer is given as 2. Math with integers can seem very goofy, because we as humans tend to want to round numbers either up or down depending on the decimal point value. Java doesn't care what the value of the decimal point is, it just gets rid of it. This means that even something like 2.95 would be converted to 2.

10. Try changing the division line so that it now looks like this:

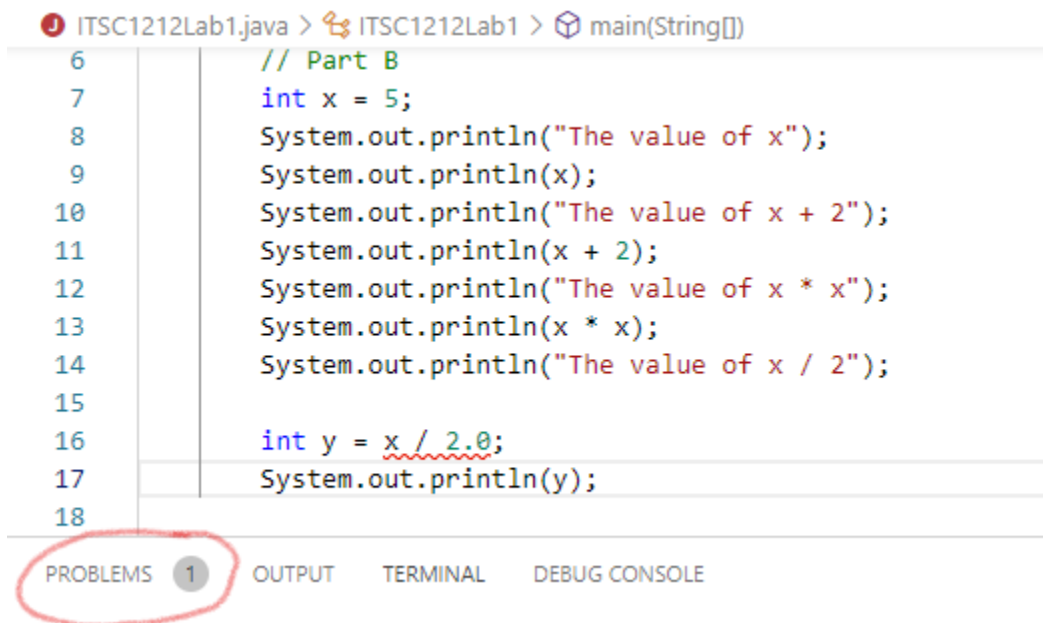
```
System.out.println(x / 2.0);
```

Why do you think the number changed?

---

---

11. Let's make one more change to our program, such that it now looks like this:

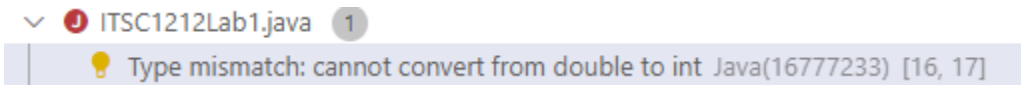


```
ITSC1212Lab1.java > ITSC1212Lab1 > main(String[])
6 // Part B
7 int x = 5;
8 System.out.println("The value of x");
9 System.out.println(x);
10 System.out.println("The value of x + 2");
11 System.out.println(x + 2);
12 System.out.println("The value of x * x");
13 System.out.println(x * x);
14 System.out.println("The value of x / 2");
15
16 int y = x / 2.0;
17 System.out.println(y);
18
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

12. Notice that as you type this in, a red line has appeared under the `x / 2.0`, and VSCode has given us a notification that we now have 1 Problem. If you mouse over the red line, or navigate to the Problems tab, you will

see an error message that says.



13. The reason we see this error is because we have declared the variable to be of type *int* on line 16, which is a whole number, but we have given it a number with a decimal point, or what Java calls a *double*. We also won't be able to run our code until we fix this error.
14. If we want *y* to hold a double, it needs to be declared as one. Change the declaration on line 16 so that *y* is a double instead of an *int*. Once you've made the correct change, the problems tab should be empty and your code will run again.
15. When you are done, show this part to an instructor or a TA to get credit for this section.

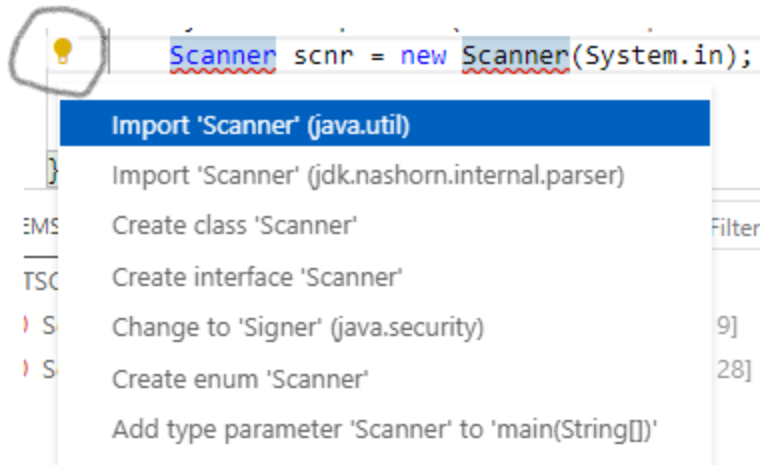
## Part D: Temperature Conversion

A common way of utilizing programs is to perform calculations. As nice as it would be to create a program that does some complex calculations there are a few aspects to that that we aren't ready for just yet. However, we can do input and simple calculation steps. Let's give that a try.

1. Before you do anything else, comment out the code from Part C (You can highlight multiple lines with your mouse, and then use ctrl+/ to add comments to multiple lines at once).
2. Add the following code below a comment header. Try to avoid selecting any of the suggestion pop-ups while you type the line that begins with Scanner

```
//Part D
System.out.println("Enter a temperature in F ");
Scanner scnr = new Scanner(System.in);
```

3. Now that the code has been entered, you should have the red lines under the words Scanner, indicating there is a problem. Scanners are how we get input into our programs, and although they are a part of the Java language, they aren't enabled unless we add an import. Luckily, VSCode can do this for us. Click either instance of the word Scanner, then click on the lightbulb that appears at the start of the line



Choose the first option, which will add the necessary import.

4. Now that we have a Scanner to capture input, we need a variable to capture the value the user types in. Add the following line

```
//Part E
System.out.println("Enter a temperature in F.");
Scanner scnr = new Scanner(System.in);

double tempF = scnr.nextDouble();
```

5. Scanners will be explained in greater detail in future sections, but for now just know that this will capture whatever value is typed in and save it to the variable tempF. Let's test this to make sure our Scanner is working correctly. Add one more line, then run your program.

```
double tempF = scnr.nextDouble();
System.out.println("The value entered was " + tempF);
```

6. When you run your program, you should see something like this in the terminal window.

```
PS C:\Users\wicke\Documents\ITSC1212> c::; cd '
de\extensions\vscjava.vscode-java-debug-0.34.0\
11.0.10_9\bin\java.exe' '-Dfile.encoding=UTF-8'
torage\863f2d6b3abd29c0bb1919bf8e22e4ab\redhat.
Enter a temperature in F.
█
```

What you see here is the program progressing up until the point where the Scanner has stopped to ask for input. Click into the terminal window and type in a number, and then hit Enter. You should see your number printed back to you.

```
Enter a temperature in F.  
33  
The value entered was 33.0
```

If you entered a whole number, you will notice that a decimal was added for you since this is a double-type variable.

**IMPORTANT NOTE:** If you enter anything other than a numerical value here, an error message will appear instead. This error, called an **Exception**, is because you told your program to expect a value that could be saved as a double-type variable. Entering a letter or symbol is not something that can be saved as a double, since doubles are for floating-point numbers, and causes your program to crash. This type of error is also referred to as runtime error or runtime exception.

7. Now we need to convert this value to Celsius. The conversion formula is:  $(\text{Temp} - 32) * (5/9)$ . Fill this in to your program as outlined here

```
System.out.println("The value entered was " + tempF);  
double tempC = (tempF - 32) * (5/9);
```

8. With the formula added, this line of code will create a new double variable, and set its value equal to the converted temperature. Now all that's left to do is test it. Add one more print statement that will print your new temperature value. When you run your program, you should hopefully see something like this

```
Enter a temperature in F.  
32  
The value entered was 32.0  
The temperature in C is: 0.0
```

9. Run your program one more time, but enter a value of 212. This should return a value of 100.0 for your temperature in Celsius.

```
Enter a temperature in F.  
212  
The value entered was 212.0  
The temperature in C is: 0.0
```

*Uh oh...*

10. Clearly something is wrong here. This is why it's important to test your program with multiple inputs where you know what the answer should be.

You can run your program a few more times with different inputs, but no matter what you do the output will always be 0.

11. So why do we always get 0, and how can we fix it? Think back to Part C, and fix your code. Once your code is working properly, show your code to the instructor or a TA to get credit for this section.