

# Java Programming 2 – Lab Sheet 1

---

This Lab Sheet contains material based on Lectures 1—2.

**The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 1 October 2020.**

## Aims and objectives

- Familiarisation with JShell REPL for writing, testing, loading, and saving fragments of Java code
- Writing Java methods to solve simple problems

If you are already familiar with Java, you may want to skip to the section entitled **Submission material** – however, if you have not used JShell before, it may still be instructive to follow the initial instructions below.

The instructions here should provide you enough information about JShell to complete the lab assignment. If you want more details, the official documentation is available at <https://docs.oracle.com/en/java/javase/14/jshell/introduction-jshell.html>.

## Installing Java and JShell

If you are using your own laptop, you will first need to ensure that the correct version of Java is installed before following the rest of the instructions. The following links describe how to install OpenJDK 14.0.2 on various operating systems; there is also a video walkthrough on the course Moodle site of installing Java (and Eclipse) on Windows 10.

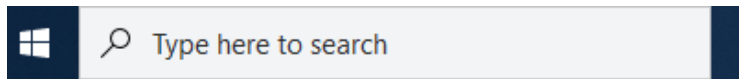
Windows 10: <https://java.tutorials24x7.com/blog/how-to-install-openjdk-14-on-windows>

MacOS: <https://java.tutorials24x7.com/blog/how-to-install-openjdk-14-on-mac>

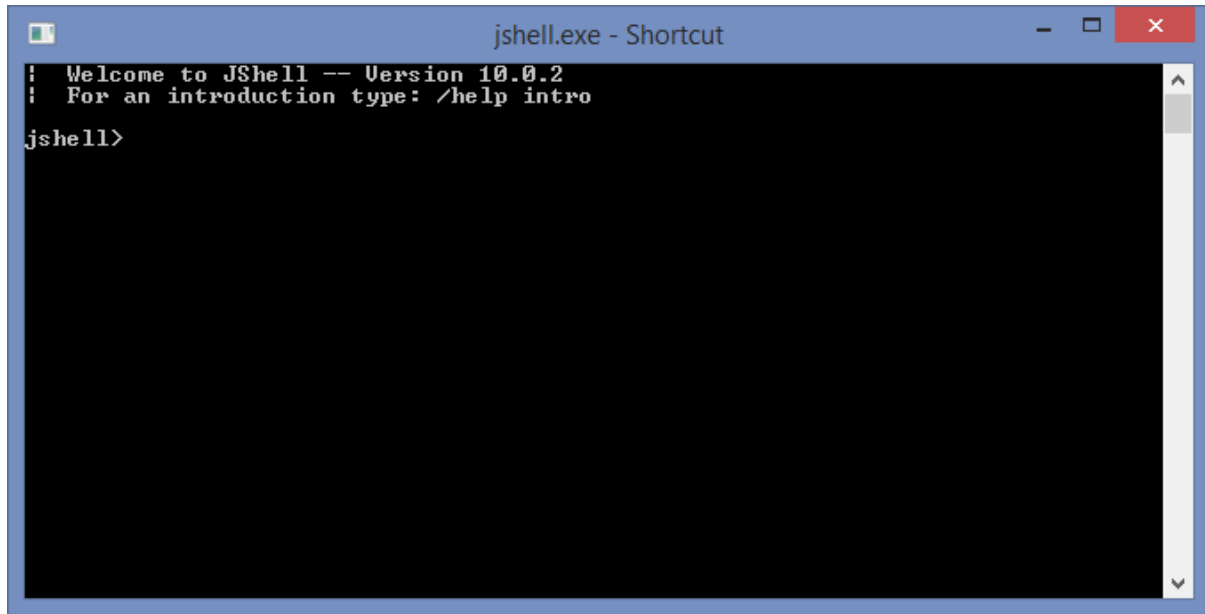
Linux: <https://java.tutorials24x7.com/blog/how-to-install-openjdk-14-on-ubuntu>

## Using JShell

To launch JShell on Windows, click the **Start** menu, click the magnifying glass to search, and then type **jshell** and press enter. The process for launching it on other operating systems should be similar.



A console window will eventually appear that looks similar to the following:



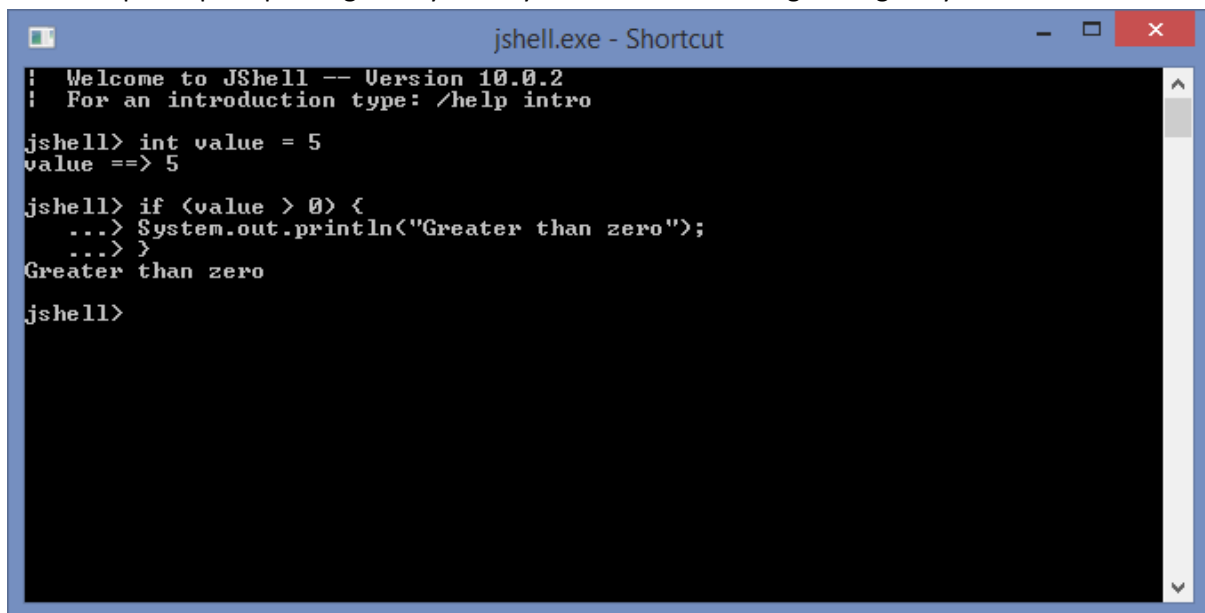
When you want to quit JShell, you can type **/exit** or just close the window it's running in.

## Declaring and modifying variables

As I have shown in lectures, the JShell REPL allows you to enter and run fragments of Java code. The following are some suggestions for things to try – you can also enter your own Java code and see what happens.

- Declare an integer variable **value** (**int value**) and assign it a value using the = operator
- Try assigning other values to **value** and see what happens
  - Different integer values, positive and negative?
  - Extremely large or small integer values (e.g., 1000000000)?
  - Decimal numbers (e.g., 0.05)?
  - Character values? (remember to use single quotes, like 'c')
  - String values? (remember to use double quotes, like "string")
- Print output to the screen with **System.out.println** – try updating and printing the value of a variable
  - Print a string like "hello world"
  - Print the current value of the variable **value**
- Declare a new **String** variable called **value** – note that this variable will replace the previous **int** variable
- Try all of the above assignments again with the new version of **value** and see what happens now

You can also enter loops and if/else statements into JShell. When you open a new block of code (that is, when you enter a new opening curly bracket), JShell will show a prompt that looks like this: ...>. This special prompt will go away when you enter the matching closing curly bracket. Like this:

A screenshot of a Windows command prompt window titled "jshell.exe - Shortcut". The window has a black background with white text. The text shows the JShell version 10.0.2 and a welcome message. The user enters "int value = 5", and the prompt changes to "value ==> 5". Then, the user enters an if statement: "if (value > 0) { ...> System.out.println(\"Greater than zero\"); ...> }". The output "Greater than zero" is displayed. The prompt returns to "jshell>".

```
jshell.exe - Shortcut
! Welcome to JShell -- Version 10.0.2
! For an introduction type: /help intro

jshell> int value = 5
value ==> 5

jshell> if (value > 0) {
...> System.out.println("Greater than zero");
...> }
Greater than zero
jshell>
```

Try entering more complex code – for example:

- Write an **if/else** statement to check if the **value** is positive, negative, or zero, and print **“positive”**, **“negative”**, or **“zero”** respectively

You can also try any other Java code that you might be interested in writing, or type in any code that you see on the lecture slides – don’t worry, it’s not possible to break anything, whatever you type!

Other JShell tips:

- You can press the up and down arrows to retrieve commands from your command history if you want to try to run something again
- You can use the **<TAB>** key to automatically complete variable names, method names, file paths, and lots of other things – give it a try on the command line!
- JShell includes reasonably extensive help documentation; type **/help intro** to get started

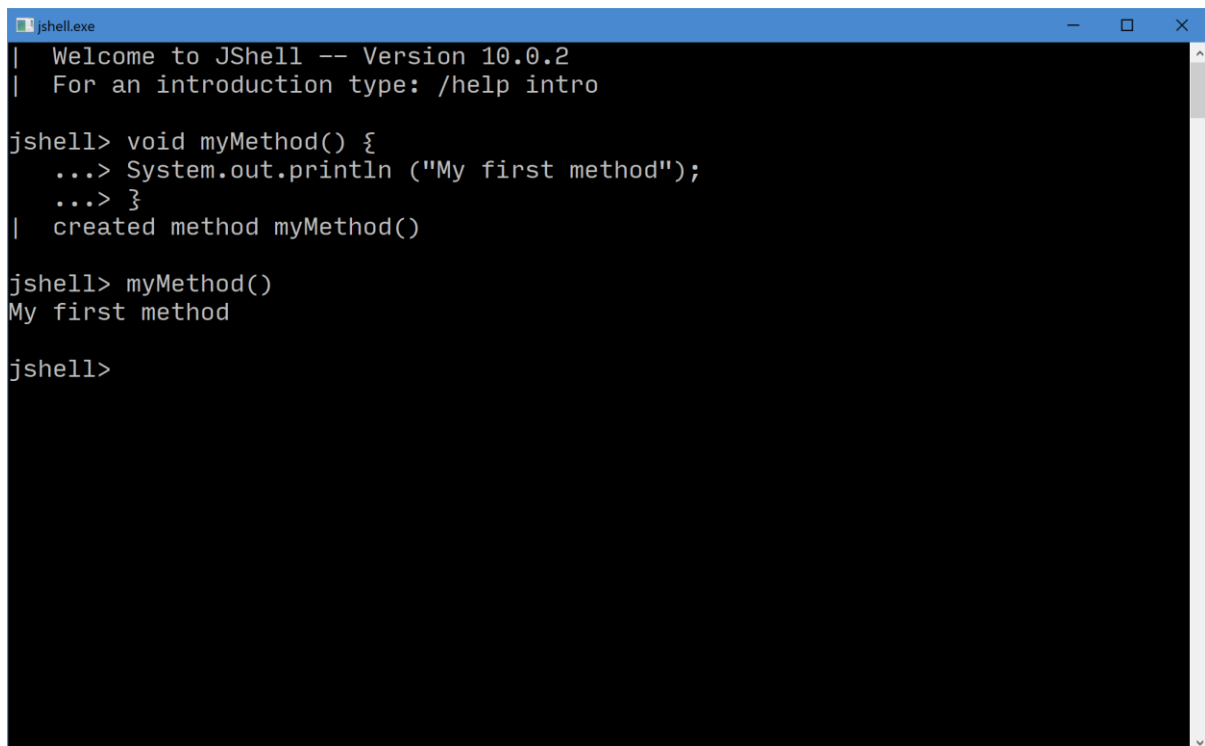
## Defining and calling methods

You can also write new methods in JShell and call them. For example, enter the following method into JShell:

```
void myMethod() {  
    System.out.println("My first method");  
}
```

Just like when entering a for or while loop, you will see the `...>` prompt until you finish writing the body of the method. If you make a mistake in entering the method, the JShell window will show the details of the error. You can re-type it, or you can use the arrow keys to modify the lines you previously entered and fix the errors.

Once you have defined a method in JShell, you can call it; for example, type **myMethod()** to call the method that you have just defined. In fact, this is a case where the **<TAB>** key is useful – try typing **my<TAB>** and it should fill in the rest of the line for you.



```
jshell.exe  
| Welcome to JShell -- Version 10.0.2  
| For an introduction type: /help intro  
  
jshell> void myMethod() {  
    ...> System.out.println ("My first method");  
    ...> }  
| created method myMethod()  
  
jshell> myMethod()  
My first method  
  
jshell>
```

## Editing, loading, and saving files

The process above is useful for testing small fragments of Java code; however, it is also useful sometimes to edit larger pieces of code into a file using an external editor and to load them into JShell to run them. This is what you will do in the assessed portion of the lab; in this section, I will describe how to do this.

First, download the file **Laboratory01.zip** from the JP2 Moodle site (the same place you got this lab sheet). Open the zip file and extract the contents. After the file has been extracted, you should have a directory **Laboratory01** somewhere on your hard drive containing a single file, **firstProgram.java**.

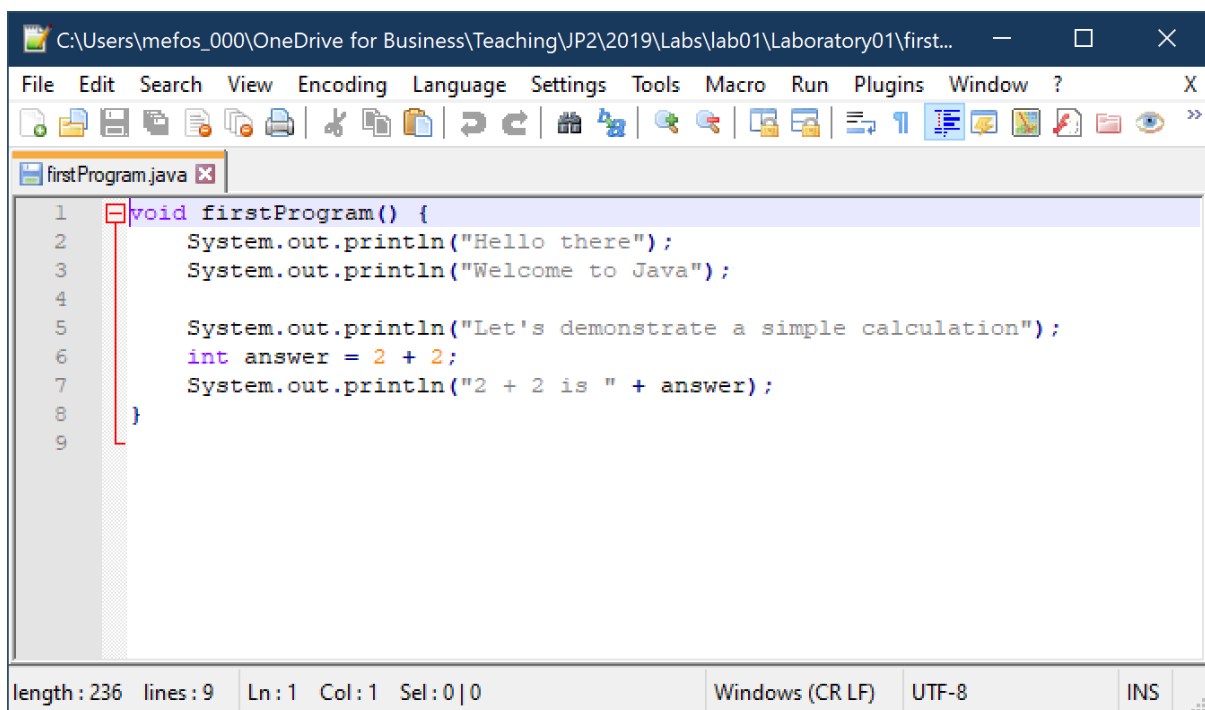
Now launch JShell and load the file **firstProgram.java** with the following command (replace **/path/to** with the actual location where you downloaded the file on your computer):

```
/open /path/to/Laboratory01/firstProgram.java
```

You can use <TAB>-completion to specify the path to the file. **You need to use forward slashes "/" rather than backward slashes "\" to specify the path.**

After this command, JShell will have a new method called **firstProgram**. You can view the contents of the method by typing **/list firstProgram** and call the method by typing **firstProgram()**.

To edit Java code, you can use any editor that you like. A good programmer's editor is **Notepad++**,<sup>1</sup> which looks like this:



```
1 void firstProgram() {
2     System.out.println("Hello there");
3     System.out.println("Welcome to Java");
4
5     System.out.println("Let's demonstrate a simple calculation");
6     int answer = 2 + 2;
7     System.out.println("2 + 2 is " + answer);
8 }
9
```

length: 236 lines: 9 Ln: 1 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS

Open **firstProgram.java** and try making some changes, then rerun the **/open** command to reload the updated method. If there is an error in the code, jshell will show it when the file is loaded.

<sup>1</sup> <https://notepad-plus-plus.org/>

If you want to save a method to a new file, you can use the **/save** command as follows:

**/save firstProgram /path/to/Laboratory1/firstProgram2.java**

This saves the method called **firstProgram** to the file at the given location.

## Submission material

You are now ready to work on the programming exercise below that forms the submission material for this lab. Your task in each case is to write a method meeting the specification, and then to save that method to a file in order to submit it.

### Task 1: Printing prime numbers

Your task is to write a Java method to compute and print the list of prime numbers less than a given parameter. The method signature should be as follows:

**void printPrimes (int max)**

The suggested implementation of the method is as follows:<sup>2</sup>

1. For each candidate number **n** between 2 and max (inclusive), do the following:
  - a. First, create a **boolean** variable (for example, call it **prime**) to record whether **n** is prime and initialise it to **true**
  - b. For each candidate factor **f** between 2 and the square root of **n**, do the following
    - i. If **n** is evenly divisible by **f**, set the Boolean variable to **false**
  - c. After checking all candidate factors, if **prime** is still true, print out the string "**n is prime**" (using the actual value of **n**)

Hints:

- Use **Math.sqrt(n)** to compute the square root of an integer **n**
- Use **n % f** to compute the remainder when **n** is divided by **f**
- If you have difficulty getting your code to work, you may want to add appropriate **System.out.println** statements to help you trace what is happening. **Don't forget to remove any such statements before submitting!**

One possible execution trace of the program would appear as follows in the JShell window:

```
jshell> printPrimes(12)
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
jshell>
```

---

<sup>2</sup> Other methods of computing prime numbers exist – if you want to try some other implementation, feel free, but we will not help you with any bugs in the implementation and the output must be as described above.

Your code will be marked automatically, so be sure that your method name and output are exactly as shown above. In particular, if you use `System.out.println` statements to help with debugging, you must remove them before submitting.

Be sure that your `printPrimes` method is saved in a file called `printPrimes.java` for submission.

## Task 2: Computing Fibonacci numbers

Your task is to write a Java method to compute and return the Nth Fibonacci number, which is a sequence of numbers where each number is the sum of the preceding two (see [https://en.wikipedia.org/wiki/Fibonacci\\_number](https://en.wikipedia.org/wiki/Fibonacci_number) if you can't remember the details). The method signature should be as follows:

**int computeFibonacci (int n)**

The suggested implementation of the method is as follows:<sup>3</sup>

1. Create two integer variables, **result** and **lastResult**, and give both of them the initial value 1
2. For every number between 0 and **n – 2**, do the following:
  - a. Store the current value of **result** in a new integer value, perhaps called **temp**
  - b. Add **lastResult** to **result**
  - c. Set the value of **lastResult** to the value stored in **temp**
3. Once the loop has finished, return the value of **result**

Again, you may find it helpful to use `System.out.println` statements to help in debugging your method, but don't forget to remove the statements before submitting.

One possible execution trace of the program would appear as follows in the JShell window; the line after the method call shows the result of the method, and the number after the dollar sign will vary depending on how many commands you have given to JShell.

```
jshell> computeFibonacci(10)
$56 ==> 55
```

```
jshell> computeFibonacci(2)
$57 ==> 1
```

```
jshell>
```

Be sure that your `computeFibonacci` method is saved in a file called `computeFibonacci.java` for submission.

**As with the preceding task, your code will be marked automatically, so be sure that your method name and output are exactly as shown above. In particular, your method should not produce any output.**

---

<sup>3</sup> Again, feel free to use other implementations if you want, but the tutors will not be able to help you if you choose to do this.

## How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JP2 moodle site. Click on Laboratory 1 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code and drag only the following files into the drag-and-drop submission area:

- **printPrimes.java**
- **computeFibonacci.java**

Then click the blue **Save Changes** button. Check the two .java files are uploaded to the system. Then click **Submit Assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you via moodle.

## Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (\*\*\*\*\*) to "Very poor" (\*). We will automatically execute your submitted code and check its output; we will also look at the code before choosing a final mark.

Example scores might be:

**5\***: you complete both tasks correctly with no bugs

**4\***: you complete one of the two tasks correctly and have made an attempt at the other

**3\***: you completed only one of the tasks, or have made a reasonable attempt at both

**2\***: you have made some attempt at both tasks

**1\***: minimal effort

**For this assignment only, we will not deduct for inappropriate code formatting or commenting – however, the tutors may comment on such stylistic issues and will deduct for them in the future.**