

# VS Code Primer

An *Integrated Development Environment* (IDE) is a software suite that includes various tools such as an editor, a compiler, a debugger, and a runtime environment. VS Code is a free, simple, and powerful IDE which is becoming increasingly popular among developers. VS Code will be the IDE that we will use in our course. You are welcome to use other IDEs, like Eclipse or IntelliJ (or any text editor).

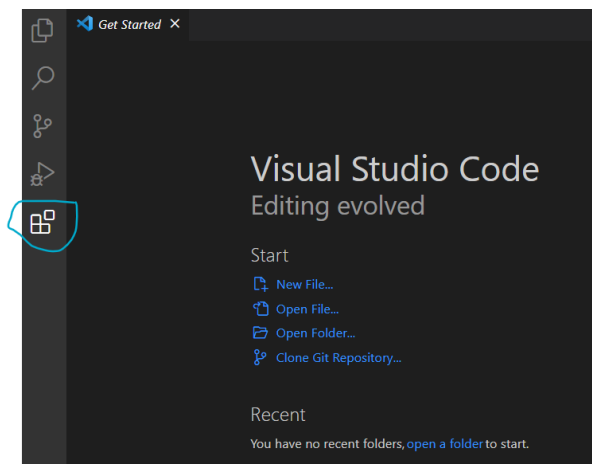
## 0. Installation

Download the VS Code software from <https://code.visualstudio.com> (make sure to select the relevant version for your PC/OS), and follow the installation guidelines (which depend on your PC/OS).

Notice that “VS Code” is also named “Visual Studio Code”, and that’s how the app may appear in your file system / application folder.

Choose the color scheme that you like, then click MARK DONE at the bottom.

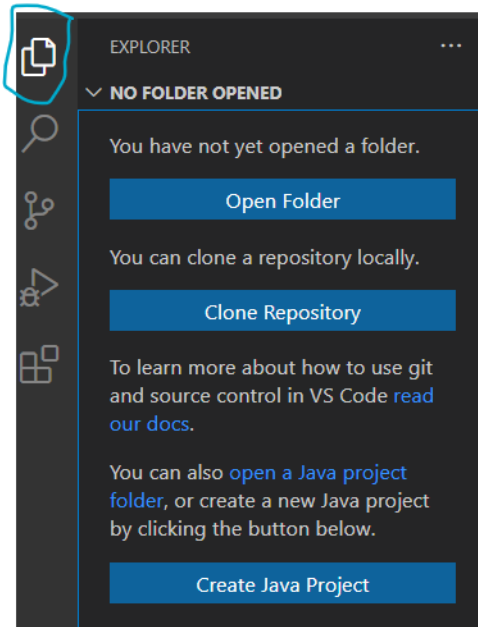
Next, click the EXTENSIONS icon (marked below):



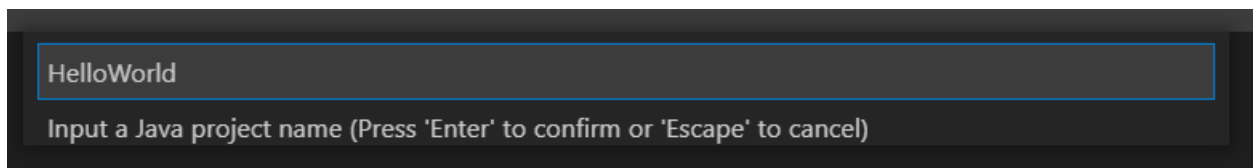
And install the “Extension Pack for Java” (it may already be installed. If it is, no need to reinstall it).

## 1. Writing your first program

A Java program is a set of one or more class files. In most IDE’s, including VS code, a Java program is written and maintained in an environment called “Java Project” (similar to the notion of a folder). Let’s practice creating a Java project. Click the EXPLORER icon (marked below). Then click “Create Java project” and choose “no build tools”.



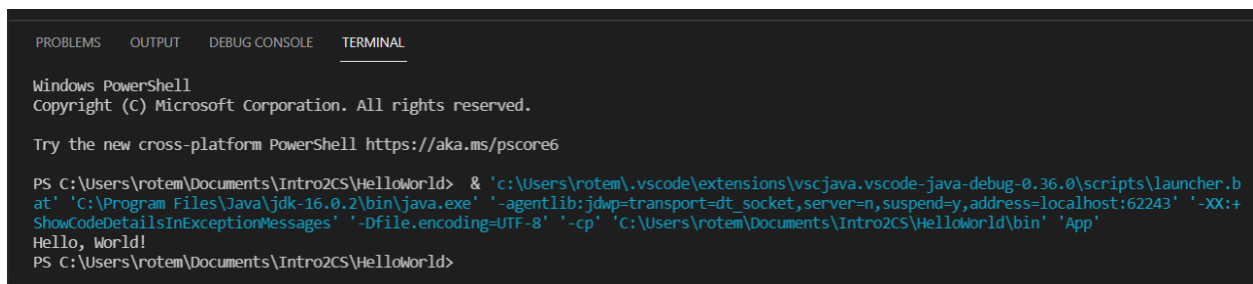
We assume that your PC has a folder named “Intro2CS”, or something like this. Choose this folder as the project location, and name the new project, say, “HelloWorld”:



The explorer will now show your new project, with several sub-folders like “bin” and “src” under it. The latter folder is where your source code will be stored. By default, this code will be stored in a new class file named “App.java”:



Write the code shown above, and try to run it using the “play” button on the right of the screen. If the program contains no errors, it should output “Hello, World!” on the screen:



This window is similar to the terminal / command com shell that you used up to this point in the course. But now you no longer have to compile and execute programs by typing textual

commands. Instead, the PLAY button will do the job. The TERMINAL tab displays the program's outputs, and the PROBLEMS tab displays compilation and runtime errors.

**It may be a good idea to rename the class** to a better name than the default "App". For simplicity, we skip this renaming action.

Note that within the IDE editor, Java code is highlighted with colors, making it easier to read and debug. For example:

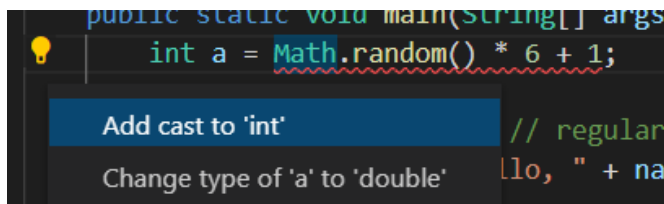
```
if (true) {  
    int a = 1;  
    while (a < 5) {  
        a = (int) (Math.random() * 6) + 1;  
        System.out.println(a);  
    }  
}
```

Control flow words like *if*, *while*, *for* are colored in one color, types/classes like *int*, *double*, *Math* in a second color, reserved words like *true*, *public*, *static* in a third color, etc.

Another important coloring convention is a red curly underline, which marks errors in your code. For example, consider the following attempt to assign a double value to an int variable:

```
int a = Math.random() * 6 + 1;
```

To inspect an error, roll the mouse over the marked line, and you'll get an explanation. In many cases the editor will also display a lightbulb next to the error. If you click it, you will get a tip on how to fix the error. In some cases the tip is good, and in other cases, less so. Here is an example of a tip which is right on target:



The screenshot shows a code editor with a lightbulb icon next to the line `int a = Math.random() * 6 + 1;`. A tooltip is displayed with two suggestions: "Add cast to 'int'" and "Change type of 'a' to 'double'". The first suggestion is highlighted in blue.

The editor also displays warnings (using a gray line) under unused variables, and many more useful tips. Read and use these tips in order to debug your code.

## 1. Command-line arguments

Suppose you want your program to operate on one command-line argument. Suppose you want the value of this command-line argument to be "Rotem". Click run->add configurations (or open configurations, if you already have one). A file named "launch.json" will pop up, or you can find it through the path explorer->.vscode->launch.json.

At the end of this configuration file, add the line: "args": "Rotem", like this:

```

4 // For more information, visit: https://go.microsoft.com/fwlink/?LinkId=723250
5 "version": "0.2.0",
6 "configurations": [
7   {
8     "type": "java",
9     "name": "Launch Current File",
10    "request": "launch",
11    "mainClass": "${file}"
12  },
13  {
14    "type": "java",
15    "name": "Launch App",
16    "request": "launch",
17    "mainClass": "App",
18    "projectName": "HelloWorld_b4bb5fb3",
19    "args": "Rotem"
20  }
21 ]
22 }

```

(You are welcome to enter your name, instead of “Rotem”)

Don’t forget to save! (ctrl+s works)

Now change the HelloWorld code to:

```

public static void main(String[] args) throws Exception {
    String name = args[0];
    System.out.println("Hello, " + name + "!");
}

```

Now run the program, and watch it printing “Hello Rotem”, or “Hello yourName”.

## 2. Debugging

Programs can be debugged by marking “breakpoints” in the code. Let’s add a breakpoint next to the print line. The breakpoint is marked with a red circle:

```

2  ✓ public static void main(String[] args) throws Exception {
3      String name = args[0];
4      System.out.println("Hello, " + name + "!");
5  }
6  }
7

```

Now launch the debugger: On the left side of the play button, click the small arrow and then “Debug Java”:

Run the program. Note that the debugger displays the values of the variables at the moment at which the break occurred:

```

2   public static void main(String[] args) throws Exception {
3       String name = args[0]; name = "Rotem", args = String[1]@9
4       System.out.println("Hello, " + name + "!"); name = "Rotem"
5   }
6   }

```

Suppose you want to experiment with possible code changes, and you want to check how these changes impact the program's execution. Click the "DEBUG CONSOLE" tab, and try out the code you want to evaluate. For example:


```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
→ name
> "Rotem"
→ "Bye" + name
> "ByeRotem"
→ "Bye " + name
> "Bye Rotem"
→ 1 + name
> "1Rotem"
→ 3 * 5
15
> Type your eval command here...

```

Once you are done playing and you want to move on with the program's execution, click CONTINUE (marked below) on the upper bar:

App.java - HelloWorld - Visual Studio Code



(String[])

The program will continue running, until the next breakpoint.

If you want to continue executing line by line, click the STEP OVER button (on the right of the icon marked above).

There are lots of other options to explore, but that's enough for now.

### 3. Javadoc

To create Javadoc for a function, start typing `/**`. The editor will open a field for entering a javadoc comment, like so:

```

/** */
Run  Javadoc comment
public static void main(String[] args) throws Exception {

```

Click ENTER (for an empty comment) and you will get the following:

```

/**
 *
 * @param args
 * @throws Exception
 */
Run | Debug
public static void main(String[] args) throws Exception {
    String name = args[0];
    System.out.println("Hello, " + name + "!");
}

```

Note that the editor automatically fills in the skeletons of the documentation of the function's arguments, and the exceptions thrown by the function (more on exceptions – later in the course). At this point you can start writing the documentation itself. For example:

```

/**
 * Prints a greeting to the given name
 * @param args a String representing a name
 * @throws Exception if no args are given
 */
Run | Debug
public static void main(String[] args) throws Exception {
    String name = args[0];
    System.out.println("Hello, " + name + "!");
}

```

To create an HTML version of the Javadoc that you wrote, go to the TERMINAL and enter the following command:

```
javadoc -d "javadoc" "src\App.java"
```

The editor will create a new folder named “javadoc” in your HelloWorld folder. To inspect the Javadoc, open the generated “App.html” file in a browser. If you scroll down to “Method Details” you will see the function documentation:

## Method Details

### main

```
public static void main(String[] args)
    throws Exception
```

Prints a greeting to the given name

#### Parameters:

args - a String representing a name

#### Throws:

Exception - if no args are given

Let's go back to TERMINAL. Note that there is a warning, saying that a class comment is missing:

```
PS C:\Users\rotem\Documents\Intro2CS\HelloWorld> javadoc -d "javadoc" "src\App.java"
Loading source file src\App.java...
Constructing Javadoc information...
Building index for all the packages and classes...
Standard Doclet version 16.0.2+7-67
Building tree for all the packages and classes...
Generating javadoc\App.html...
src\App.java:1: warning: no comment
public class App {
      ^
Generating javadoc\package-summary.html...
Generating javadoc\package-tree.html...
Generating javadoc\overview-tree.html...
Building index for all classes...
Generating javadoc\allclasses-index.html...
Generating javadoc\allpackages-index.html...
Generating javadoc\index-all.html...
Generating javadoc\index.html...
Generating javadoc\help-doc.html...
1 warning
```

To fix it, write a class description:

```
/**
 * This is a description of the class
 */
```

Now save the file and create the HTML, as we did before. Inspect the generated App.html, to see the changes that you made.

Note that “regular” comments (i.e comments that are within the function, for example using `“//”`) don’t appear in the javadoc. These internal comments are meant to programmers who maintain and extend the class code. In contrast, the Javadoc is written for programmers who will use your code, without worrying how the code is written. All they want to know is how to use the class functions: Which argument to pass, and what to expect to get when the function returns.