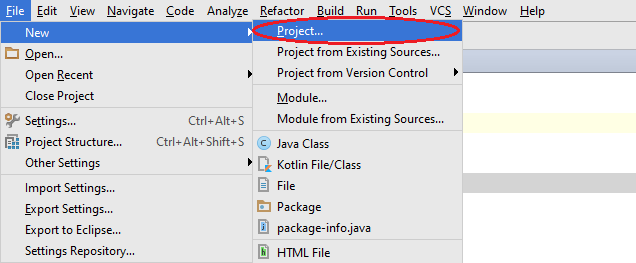
# Lab: Stacks and Queues

This document defines the lab exercise for the ["Java Advanced" course @ Software University](https://softuni.bg/trainings/1377/advanced-java-may-2016). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1377/advanced-java-may-2016).

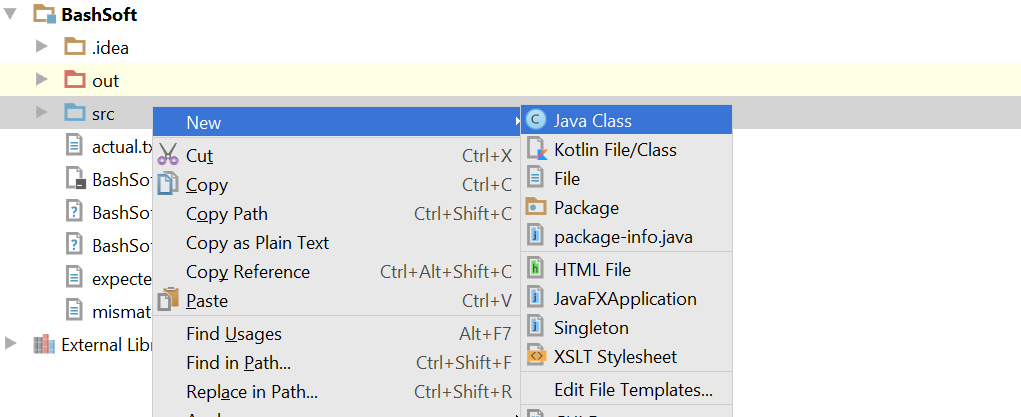
# Part I: Creating a Project

## Create IntelliJ IDEA Project

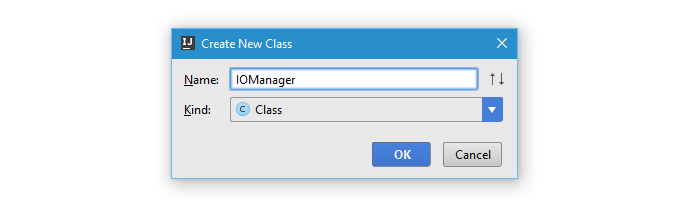
Our first task is to create a project called BashSoft, which we will extend until the end of the course so you might want to save it somewhere, where you can easily find it and where you can be sure you won’t delete it. You can call the class with the Main() method, Program, because from it we will only call the specific functions we want to execute, but our execution logic will be in other classes. (Don’t panic if you do not know anything from OOP)



Once you have created the project, you have to add a class that we will call IOManager and it will give us the functionality for traversing the folders and other behaviors.



In the next menu you have to choose to create a new class with the name “IOManager”



So now your class should look something like this:

## Create Class for Displaying Output

Before anything else, we need to decide **how** are we going **to** **communicate with the user efficiently** and if this is something that we have to use in many places, how can we change it or replace it easily using doing only a few changes in one place. The solution behind such a problem give us one of the [**Design Patterns**](https://en.wikipedia.org/wiki/Software_design_pattern) which are a topic of the **next course**, but the main idea of this one is that we can **hide** some **functionality** (The writing to the console, which can easily be changed for writing in a file), **by using** a **class** that only gives us **base functionality** for communication with the user.

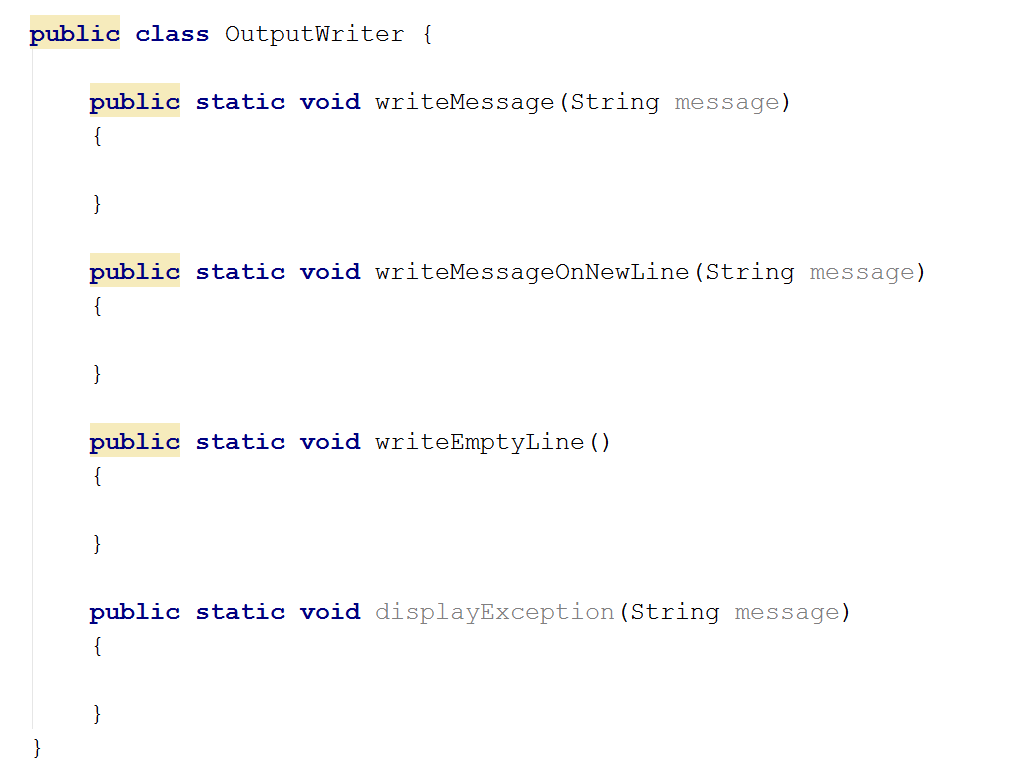
Our new class can be called **OutputWriter** and you should make it following the steps above as described for the **IOManager**. The **new class** has to be **public** and after you’ve created it, it should look something like this:



So now we can add a few methods that we will use throughout our whole app that write to the currently set output.

* The first method gives us the ability to **write a message.**
* The second method to implement is a method for **writing a message on a new line**.
* The third method is to **write a new empty line**.
* The fourth method is to **write a different kind of message** which is an **error/exception**.

The class with the three methods inside it should look something like this:



The implementation these methods is pretty common. The **first** one only **writes the message on the console**, the **second** **writes the message** and goes to the **next line**. The **third** only **writes an empty line on the console**. The **fourth** method is just like the first one.

Now that we are ready with the user output. It’s time to implement the traversal of the folders and in the future, **if** we want to **change** the **output destination**, we **only** need to **change** it here in the **class** we **just made**, and not everywhere where we’ve written **System.out.print();**

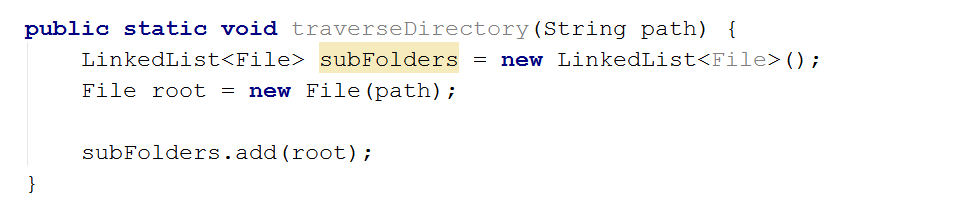
# Part II: Traversing Folders

## Implement Breadth First Search Algorithm

We will traverse the Program files (x86) files using queue with a technique called [BFS](https://en.wikipedia.org/wiki/Breadth-first_search). [Here](https://upload.wikimedia.org/wikipedia/commons/5/5d/Breadth-First-Search-Algorithm.gif) is a animation that can probably help you understand how BFS works.

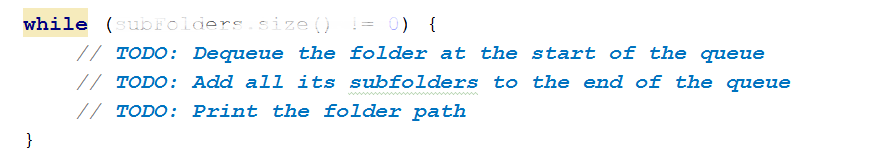
Shortly we will create a method **traverseFolder(String path)**. How does it traverse a folder? First it enqueues the folder that we pass as parameter in the method signature. After that it dequeues every folder in the queue one at a time until the queue becomes empty, while at the same time enqueues all its subfolders at the end of the queue.

For our purposes we will use the class java.io.File, which will get us all the information we need for the directories we work with, don’t worry you’ll get familiar with it in a few lectures. Here is the initialization of the method with the queue. We enqueue the root folder we wanted to traverse first.

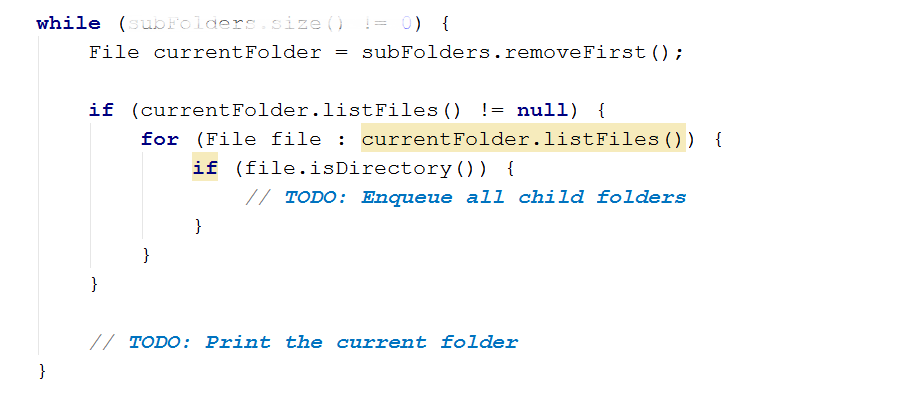


Next we need to make sure we will traverse all the subfolder that we have in the queue so we will traverse while the queue is not empty (that is why we push the initial element in the queue).

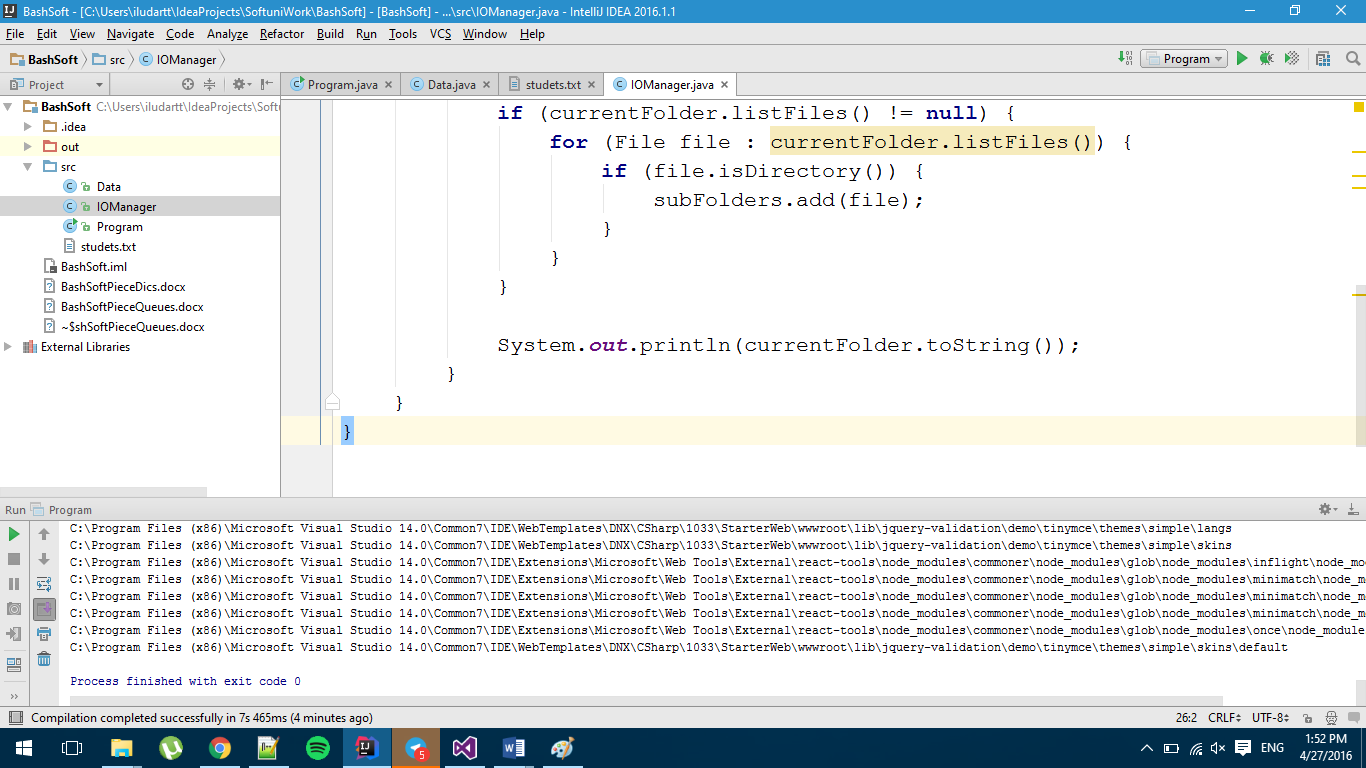
For each iteration of the while we want to dequeue a folder that we are going to traverse and to print its path.



For each iteration of the while we want to dequeue a folder that we are going to traverse and to print its path. Note that we have to check if the current folder has any files in it.



Now all that is left to do is to print the current folder. You can do this with the following line of code:



Next thing we need is a try-catch block, because if we try to touch and manage folders that we have no access to, our program will throw an exception. If we catch such an exception, we just print “Access denied”.

Now we should be ready and if we call the method with a path to our Program files folder, the result on the console should be something like this.

