# Lab: Interfaces and Abstraction

This document defines the lab overview for the ["Java OOP Advanced" course @ Software University](https://softuni.bg/java-advanced-oop). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1376/java-advanced-oop-july-2016).

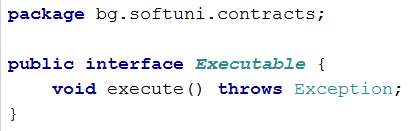
# Introduction

So far we've learned about **classes**, their **members** and the first three principles of OOP - **Encapsulation**, **Inheritance** and **Polymophism**. We've employed all of that and more in refactoring our **BashSoft**. Now it is time to go one step further and employ the fourth OOP principle in our code - namely **Abstraction**. We will do this by creating several **interfaces** that will correspond to the classes we have. We will also try to **segregate** them somewhat into more than one **interface** per class where we see fit.

# Creating Interfaces for the IO package

As you've learned during the lecture the purpose of an **interface** is to define what methods a certain class needs to have in order to be sufficient for a certain task. Thus all methods in an interface are considered public and abstract.

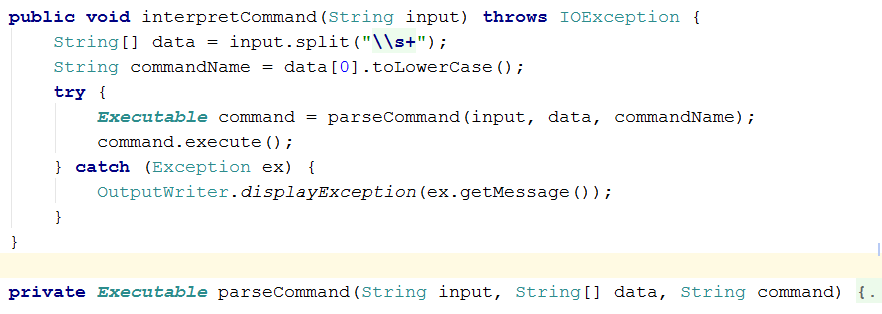
Let's start with creating our first **interface** that will define what a command should have. Create a new package called "**contracts"** and in it an **interface** called "**Executable**". Then define the proper method signature for the execute method our commands have:



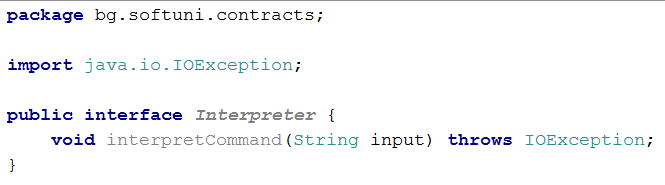
Don't forget to put "throws Exception" in the methods signature because without it, the method won't be the same as the one we have in our Command classes. Now implement the method in our abstract **Command** class:



If you defined the **interface** correctly there shouldn't be any issues. You could also implement the interface in the derived commands, although would be redundant, the code will be considered more readable. Finally replace the abstract **Command** in every **field**, **method** or **constructor** **return type / parameter**. This way we will raise the **abstraction** level of our application:

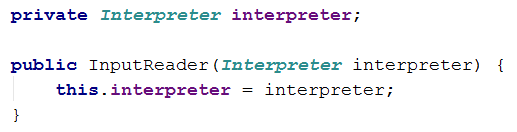


Our next **Interface** will be for the **CommandInterpreter**. You can call it "Interpreter" and define in it the public method of our class:



Don't forget to **implement** the **interface** in the **CommandInterpreter** and raise the abstraction level in **fields**, **methods** or **constructor** **return types / parameters**.

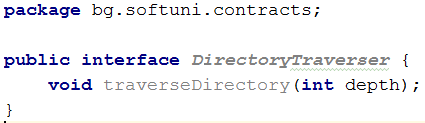
In the **InputReader:**



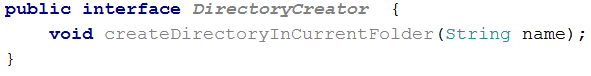
In the **main** method:



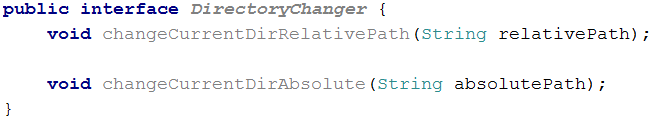
Next up is the **IOManager -** it is a good place to employ some **interface segregation.** First create an **interface** called **DirectoryTraverser** - it will hold the **traverseDirectory** method:



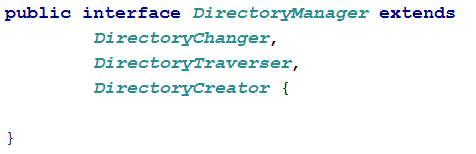
Create an **interface** called **DirectoryCreator** for the **createDirectoryInCurrentFolder** method:



Finally for this class create an **interface** called **DirectoryChanger** for the remaining two methods:



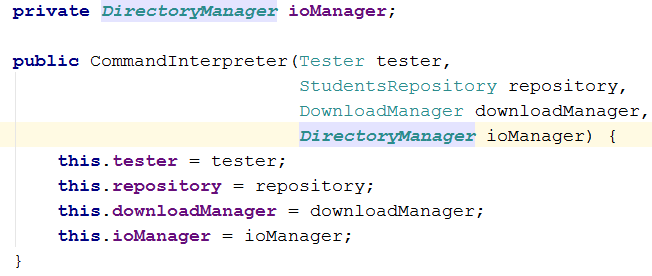
Creating these three **interfaces** would allow us to make classes that does **only one** of the things we described by each **interface**. However in the case of our **IOManager** we will need all three, so we will combine the three interfaces in a single one:

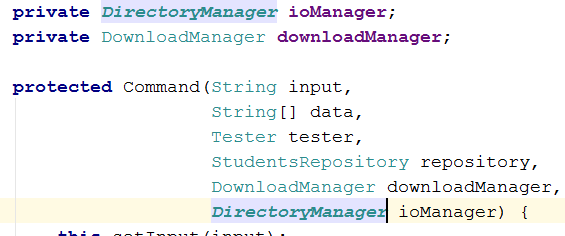


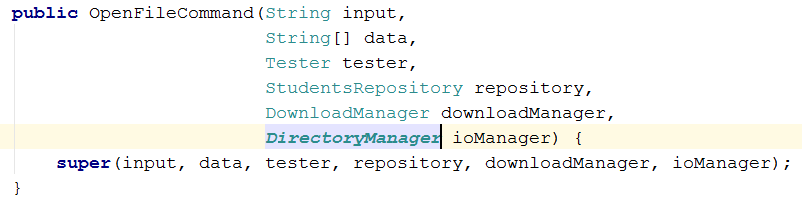
Implement it:



Finally replace **IOManager** with **DirectoryManager** in all **fields**, **methods** or **constructor** **return types / parameters** in the **CommandInterpreter** and all the **Command** classes:





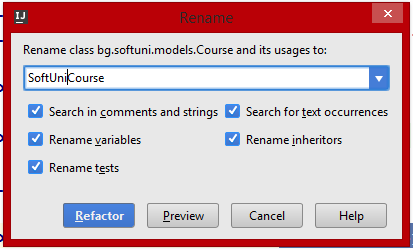
Finish the rest of the command classes yourself.

Make an interface for the **InputReader** yourself. You can call it **Reader** with method **readCommands.** Don't forget to implement it and replace **InputReader** with its interface where possible (psst - the main method).

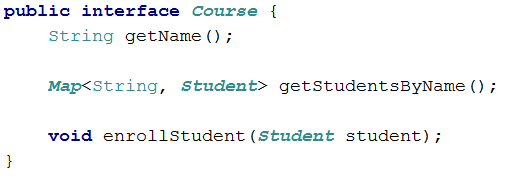
We will not make an interface for our **OutputWriter** because it is not meant to be an instance class - it only has static methods and those don't belong in interfaces.

# Creating Interfaces for the Models package

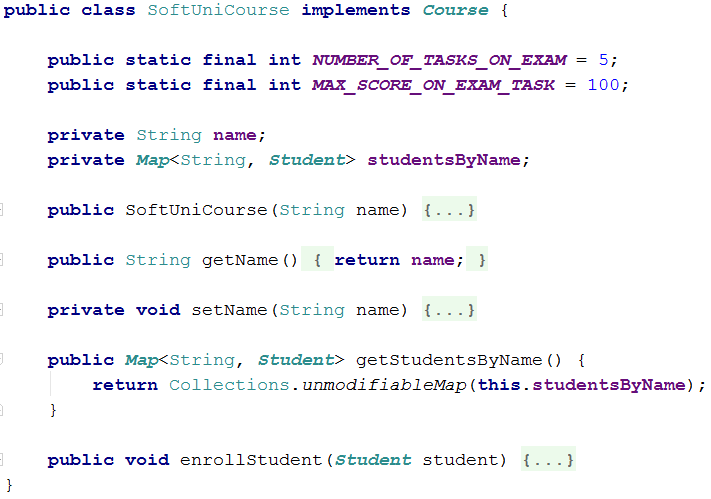
Let's begin by changing the name of both the classes to **SoftUniStudent** and **SoftUniCourse**:

****

We are doing this because we need the old more **abstract** names for the **underlying interfaces** we are about to create. Now let's do just that - create interfaces called **Course** and **Student.** They will have all the public methods of our **SoftUni** classes. This is how the **Course** interface should look (notice how we use the **Student** interface instead of the **SoftUniStudent** class in the method signatures):



And the **SoftUniCourse** class should now look like this:



Do the **Student** interface and **SoftUniStudent** classes yourself, following the pattern:

1. Put all **public** method signatures of the **class** in the **interface**.
2. Implement the interface in the class
3. Change every **SoftUniStudent** with **Student** in both of them.
4. Change every **SoftUniCourse** with **Course** in both of them.
5. Fix the **StudentRepository** to work through both the **Student** and **Course** **interfaces -** you can start off with the fields:

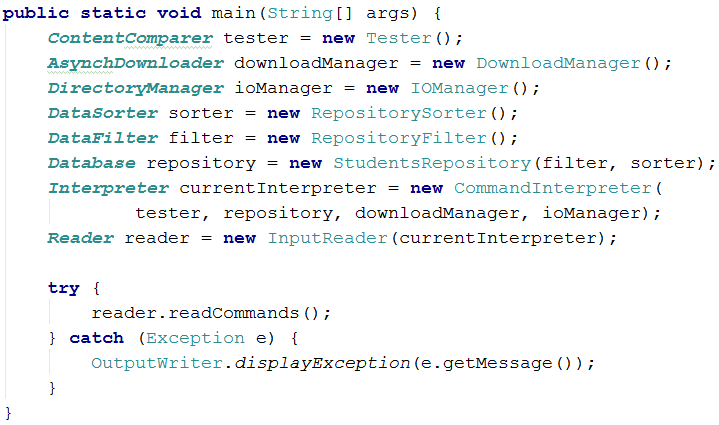


# Creating the remaining Interfaces

Create the following interfaces yourself:

* For the **Tester** class
  + **ContentComparer** with method **compareContent**
* For the **DownloadManager**
  + **Downloader** with method **download**
    - **AsynchDownloader** extending **Downloader** with method **downloadOnNewThread**
* For the **StudentsRepository**
  + **Database** with methods: **loadData, unloadData** and extending sub-interfaces:
    - **Requester** with methods: **getStudentMarkInCourse, getStudentsByCourse**
    - **FilteredTaker** with methods: **filterAndTake**
    - **OrderedTaker** with methods: **orderAndTake**
* For the **RepositorySorter**
  + **DataSorter** with methods: **printSortedStudents**
* For the **RepositoryFilter**
  + **DataFilter** with methods: **printFilteredStudents**

Once you are done your main method should look something like this:

****

If the application still compiles at the end - congratulations you have completed the lab for **Interfaces** **and Abstraction**!