# Lab: Polymorphism

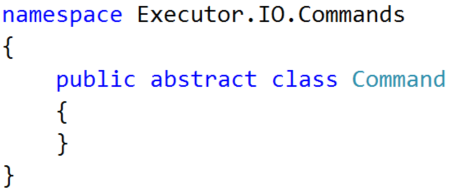
This document defines the lab overview for the ["C# OOP Basics" course @ Software University](https://softuni.bg/c-sharp-basics-oop). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1375/java-basics-oop-june-2016).

# Introduction

After learning about **Inheritance** and **Polymorphism** the time has come to do some more substantial refactoring to our project. In this lab we will employ a **design pattern** called "[**Command pattern**](https://en.wikipedia.org/wiki/Command_pattern)". As you can probably guess from the name it is mainly related to the **CommandInterpreter** and how it parses commands from user input. Our goal at the end is for our code to become more extensible and readable. The whole idea of the **Command pattern** is to replace simple method calls in the **ParseCommand** method with creation of different **Command** objects. This will make the **CommandInterpreter** **much less bulky** as every command will be in a different class. Currently the **CommandInterpreter** is around 300 lines of code and is on the big side for our small project. In order to do this we need to go through several problems:

# Making an abstract class called Command

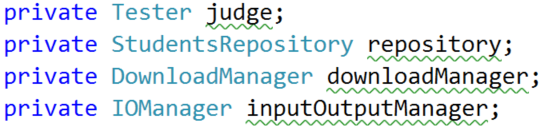
Start by making a sub-folder in the **IO** namespace called **Commands.** There we will create our first class **Command,** which we will define **abstract.**



It will have **two fields** corresponding to the parameters we used to pass to each command so far:

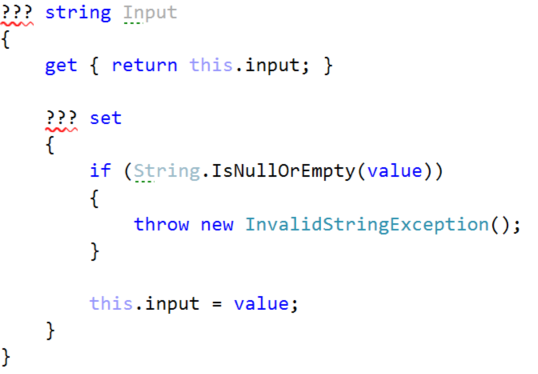
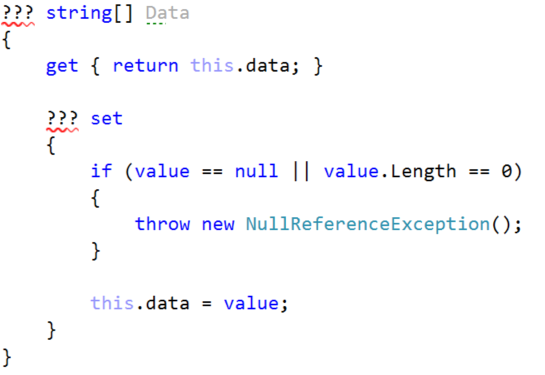


And another **four fields** that will represent all the utility classes we have:

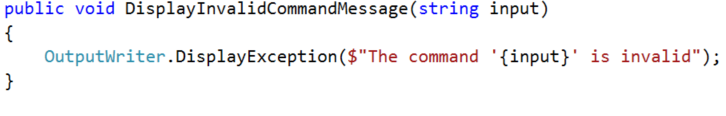


As you can see all command methods we have now have these exact two parameters so make such fields.

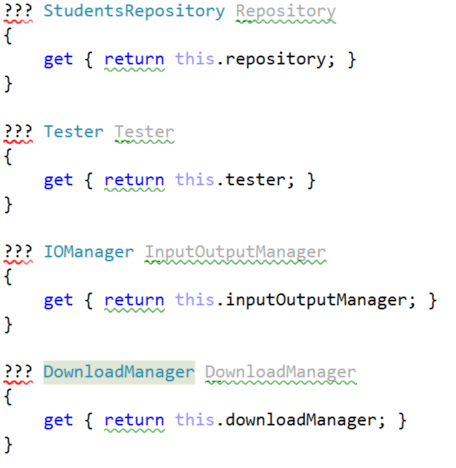
Now **encapsulate** the fields trough getters and setters. Think about what their access modifier will be if we are only going to **getters** in command classes which will **extend** our **abstract** command and **setters** that will only be used in the current class.

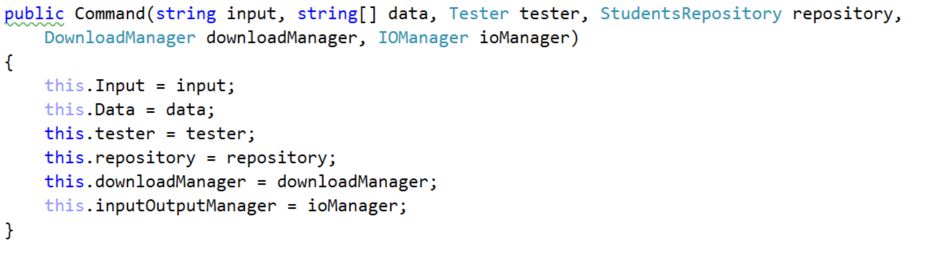
As for the **InvalidCommandException -** make it so it serves the purpose of the **DisplayInvalidCommandMessage** method and it extends **Exception**:



Also make **getters** for the utility classes. They also should be usable **only by classes that extend our abstract Command** class:



Don't forget to make a constructor that will set all the fields. Use setters for fields that have them:



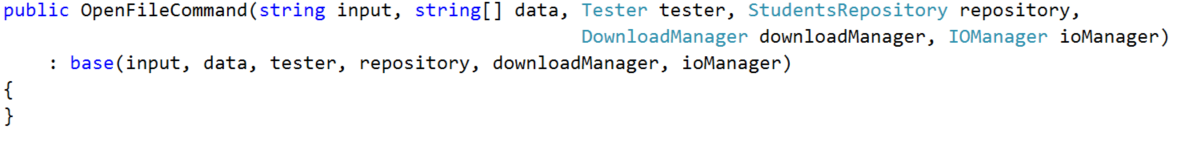
As you can see the constructor is quite big and it passes utility objects that not all commands need. Still we are doing it like this because we need all the commands to have the same parameters in their constructors. You will find out why that is so in the **Reflection** **lab** in the **next course**, so stay tuned with **BashSoft!**

The final thing we will have in our abstract class is an abstract method called **Execute** it will be abstract because we want to force classes that inherit this one to **override** it. It will also throw and **Exception**.

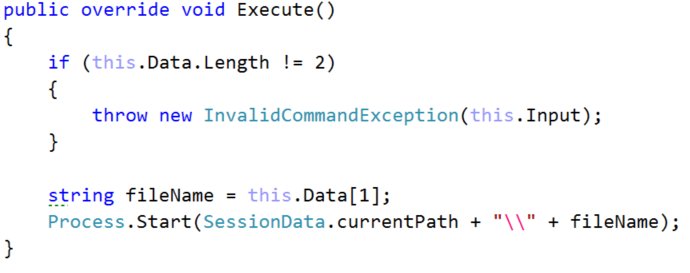


# Making the OpenFileCommand class

Create a class with the name above in the same package as our **Command** class. Add a constructor corresponding to the base class:



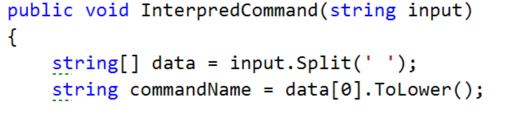
Then **override** the abstract method from the base class. In it copy the code from the **TryOpenFile** method with a few small changes: throw a new **InvalidInputException** instead of using the **DisplayInvalidCommandMessage** method; replace data with its corresponding field getter.



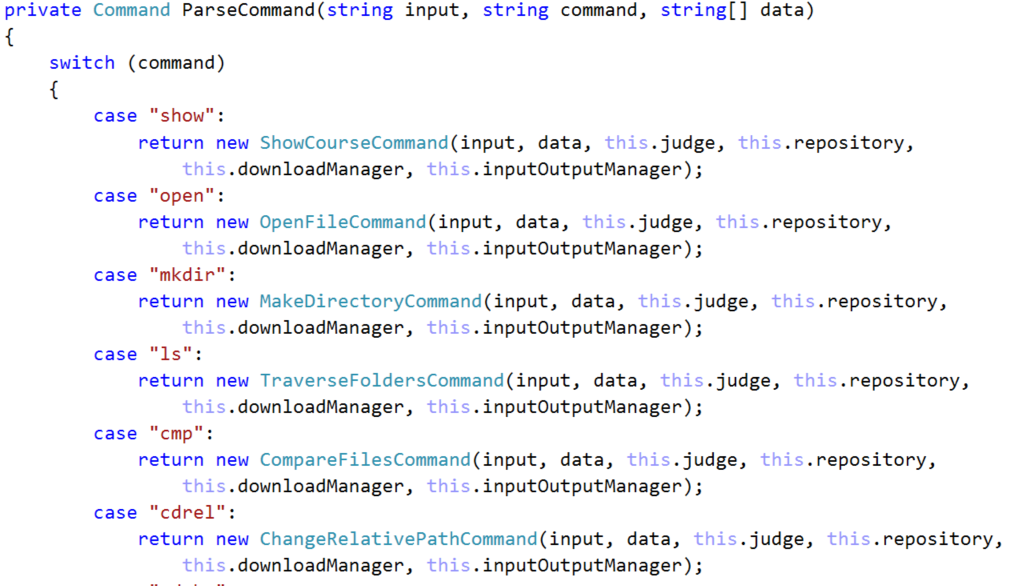
# Accommodate the CommandInterpreter to work with our new commands.

Now that we have the **OpenFileCommand** class we can make the **CommandInterpreter** use it instead of the corresponding method.

Let's start in the **InterpretCommand** method. First - rename the String **command** to **commandName,** we will need that name for the **Command** object.



Now change the return type of the **ParseCommand** method to **Command.** In the **switch-case block** instead of calling methods in every case **return** new **Commands** of the corresponding type:



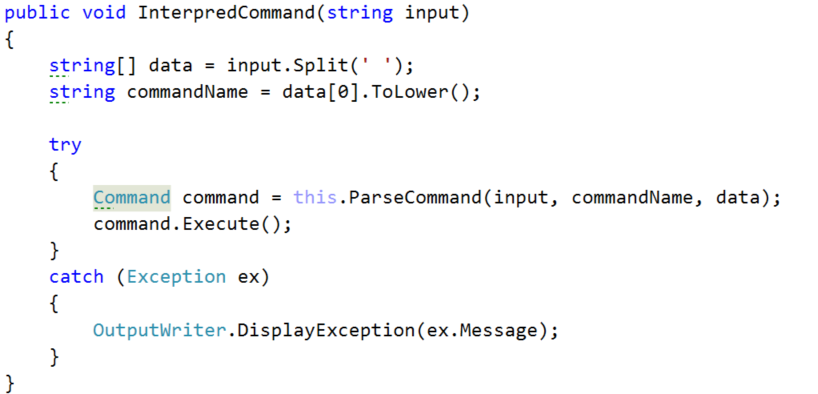
You will later create classes for all the other commands. The **default case** throws a new **InvalidCommandException**:



The command classes' names are the following:

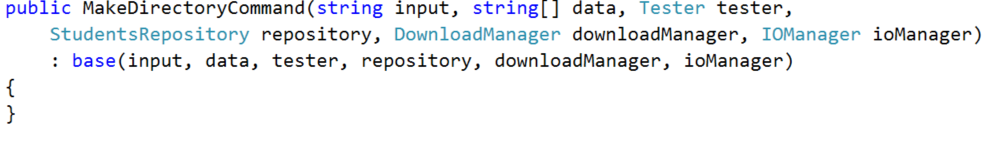
* open -> OpenFileCommand
* mkdir -> MakeDirectoryCommand
* ls -> TraverseFoldersCommand
* cmp -> CompareFilesCommand
* cdrel -> ChangeRelativePathCommand
* cdabs -> ChangeAbsolutePathCommand
* readdb -> ReadDatabaseCommand
* help -> GetHelpCommand
* show -> ShowCourseCommand
* filter -> PrintFilteredStudentsCommand
* order -> PrintOrderedStudentsCommand
* download -> DownloadFileCommand
* downloadasynch -> DownloadAsynchCommand
* dropdb -> DropDatabaseCommand

Finally in the try block create the Command object and set it to **ParseCommand,** then call the **command.Execute**() method. We can also collapse all our catch blocks to a single catch of an **Exception**. This way we are sure to have caught everything, it will still print the right message and we reduce code clutter. Here is the final look of the **interpretCommand** method:

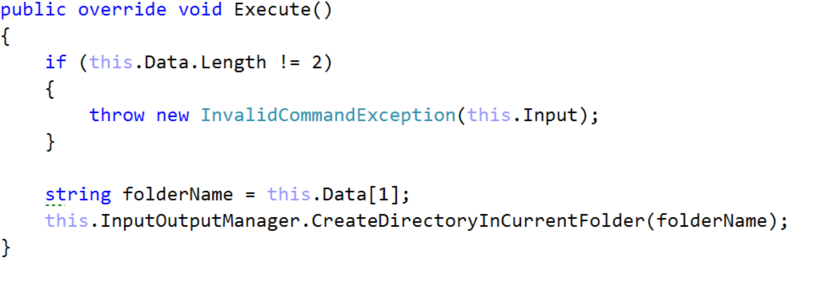


# Creating the MakeDirectoryCommand

Following the same pattern as with creating the **OpenFileCommand** make a constructor matching the base one.

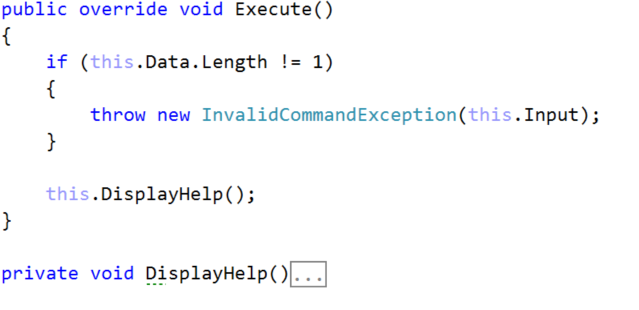


When copying the code from **TryCreateDirectory** to the **execute** method**,** don't forget to change the usage of the ioManager field with the corresponding **getter:**



# Creating the GetHelpCommand

This command and a couple of others use **private helper methods** to **encapsulate** some of their **main logic**. Just copy the whole **DisplayHelp()** **method** alongside the code you move from the **TryGetHelp** method. Here is how it looks in the **GetHelpCommand,** you must do the others yourself:



Commands have such helper methods are only: **PrintFilteredStudentsCommand** and **PrintOrderedStudentsCommand.**

# Finish all the other Command classes yourself

We went through all the basic logic for going from a **try<DoSomething>** method to a **<DoSomething>Command** class. Now it is your turn - finish refactoring the other commands. Once you are done you can delete (or comment if you prefer) all the methods in the **CommandInterpreter** except **InterpretCommand** and **ParseCommand.**

Congratulations you have completed the lab exercise for **Polymorphism!**