# Lab: Polymorphism

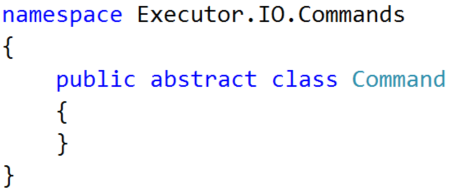
This document defines the lab overview for the ["C# OOP Basics" course @ Software University](https://softuni.bg/trainings/1636/c-sharp-oop-basics-june-2017). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1375/java-basics-oop-june-2016).

# 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 in the end is to provide a source code which is a lot more extensible and readable than at the moment. 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 solve several problems:

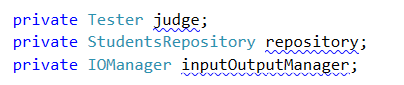
# Making an abstract class called Command

Start by making a **sub-folder** in the **IO** namespace called **Commands.** This is the place where we will store our first class **Command,** which we will define **abstract.**



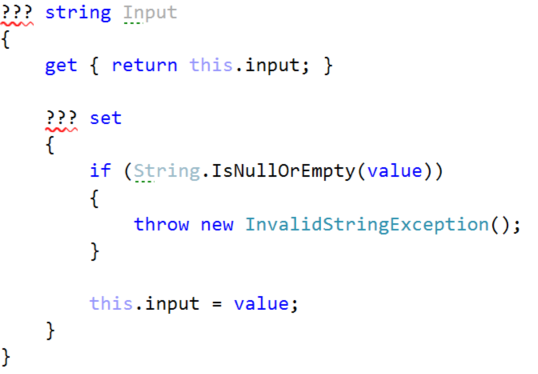
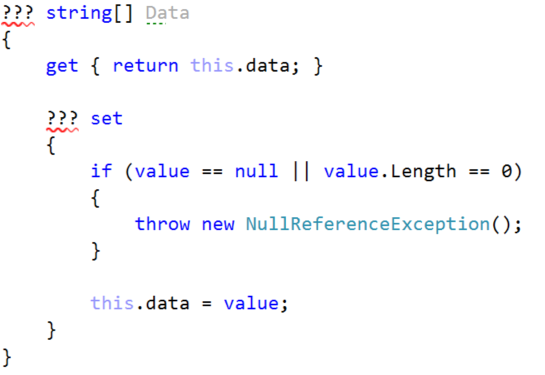
It will have **two fields** corresponding to the parameters we used to pass to each command so far – **input** and **data**:

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

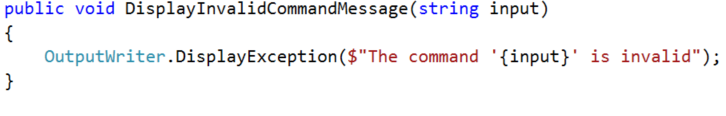


As you can see all command methods have these exact two parameters so make such fields.

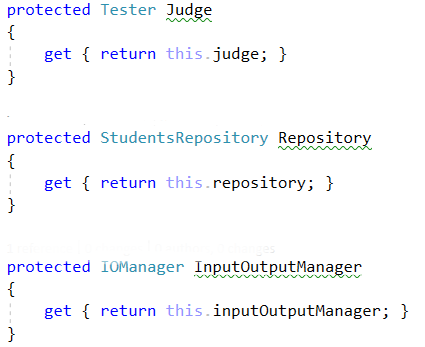
Now **encapsulate** the fields trough getters and setters. Think about the appropriate access modifiers - weather the **getters** will be used in the rest of Command classes which will **extend** our **abstract** Command and **setters** that will only be used in the current class.

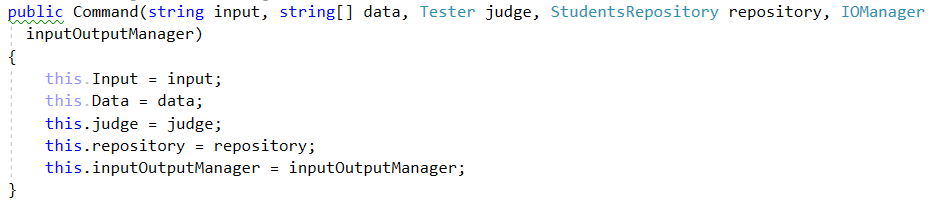
As for the **InvalidCommandException –** create one to replace the purpose of the **DisplayInvalidCommandMessage** method. It should extend the class **Exception** once again:



Also provide **getters** for the utility classes. They should be useable **only by classes that extend our abstract** class:



Don't forget to make a constructor that will set all the fields. Use the setters of the fields where possible:



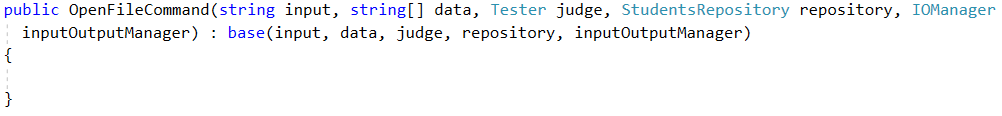
As you can see the constructor is quite big and it receives utility objects that not every command really needs. Still we are leave it like this because we need all the commands to have the same parameters in their constructors. You will find out why that is so in the **Reflection** **lab** in the **next 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**.

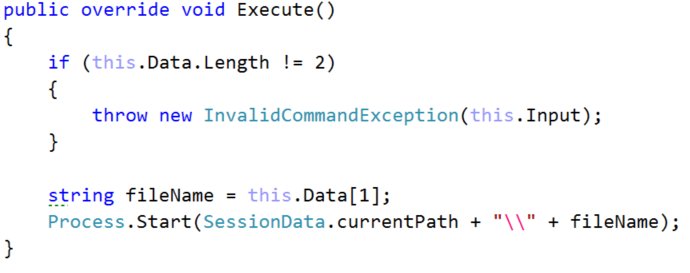


# Making the OpenFileCommand class

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



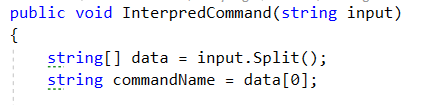
Then **override** the abstract method from the base class. Inside copy the code from the **TryOpenFile** method with a few small changes: throw a new **InvalidInputException** instead of using the **DisplayInvalidCommandMessage** method; replace data with its corresponding field getter.



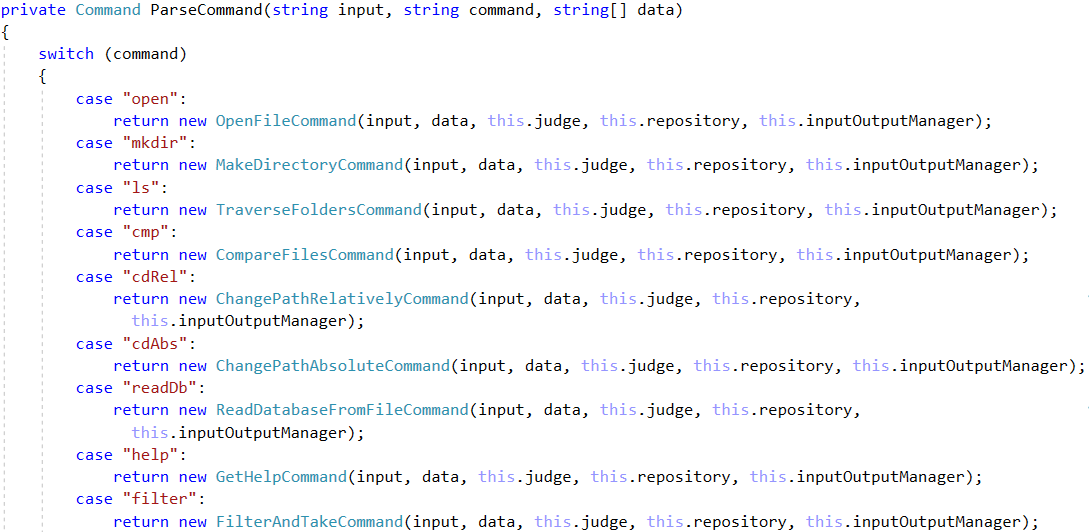
# Accommodate the CommandInterpreter to work with our new commands.

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:



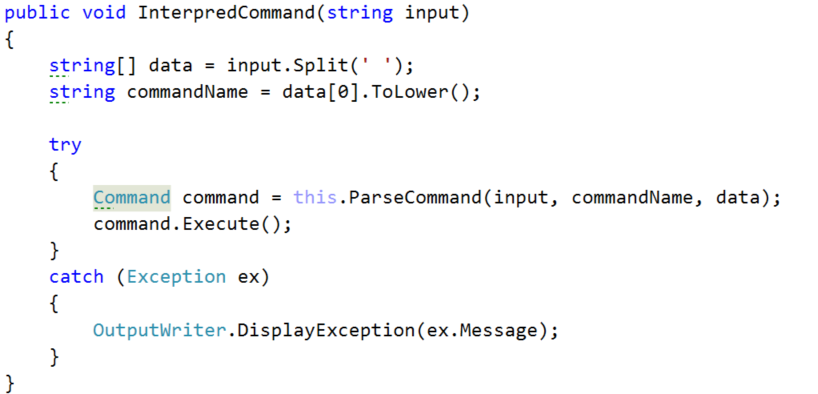
You will later create classes for all the other commands. The **default case** throws a new **InvalidCommandException**:



The command classes' names are the following:

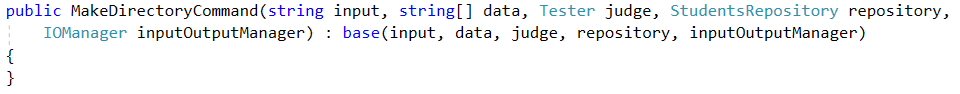
* “open” -> OpenFileCommand
* “mkdir” -> MakeDirectoryCommand
* “ls” -> TraverseFoldersCommand
* “cmp” -> CompareFilesCommand
* “cdrel” -> ChangeRelativePathCommand
* “cdabs” -> ChangeAbsolutePathCommand
* “readdb” -> ReadDatabaseCommand
* “help” -> GetHelpCommand
* “show” -> ShowCourseCommand
* “filter”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

Finally in the try block of the InterpedCommand method creat a 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 type **Exception**. In this way we are sure that everything will be caught and we will still print the right message thus we reduce code clutter. Here is the final look of the **interpretCommand** method:

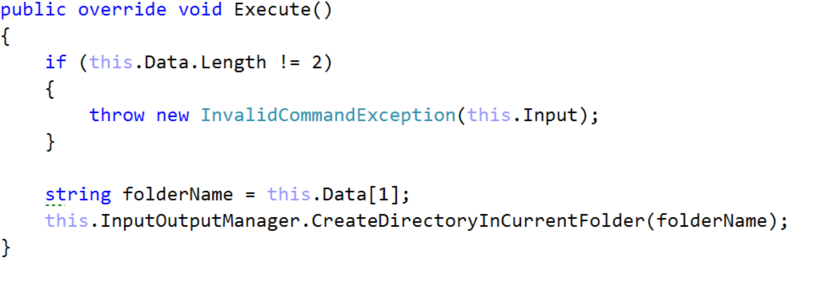


# Creating the MakeDirectoryCommand

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

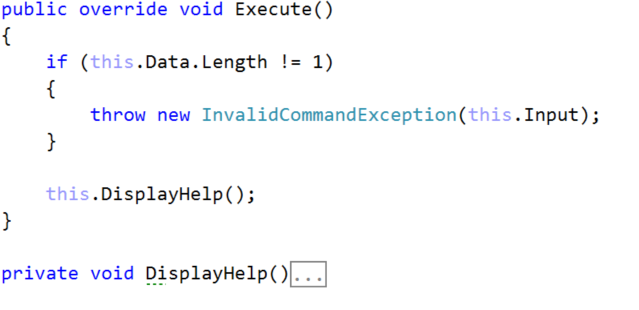


When copying the code from **TryCreateDirectory** to the **Execute** methoddon't forget to change the usage of the **inputOutputManager** field with the corresponding **getter:**



# Creating the GetHelpCommand

In order to **encapsulate** the **main logic** behind some commands as this one, we use **private helper methods**. 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 by yourself:



The other Commands that have such helper methods are: **PrintFilteredStudentsCommand** and **PrintOrderedStudentsCommand.**

# Finish all the other Command classes by yourself

We went through the basic logic behind the transition from a **Try<DoSomething>** method to a **<DoSomthing>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 P**arseCommand.**

Before we consider the job done, do some testing in order to confirm all the changes. If everything works as previously - congratulations you have completed the lab exercise for **Polymorphism!**