Command pattern

Dimitar Markov -ei4s1

**Pros and cons**

**+**

* Decouples the object that invokes the operation from the one that know how to perform it
* This pattern helps in terms of extensible as we can add a new command without changing the existing code.
* It allows you to create a sequence of commands named macro. To run the macro, create a list of Command instances and call the execute method of all commands.
* Ability to undo/redo easily

**-**

* Increase in the number of classes for each individual command

**Introduction to application**

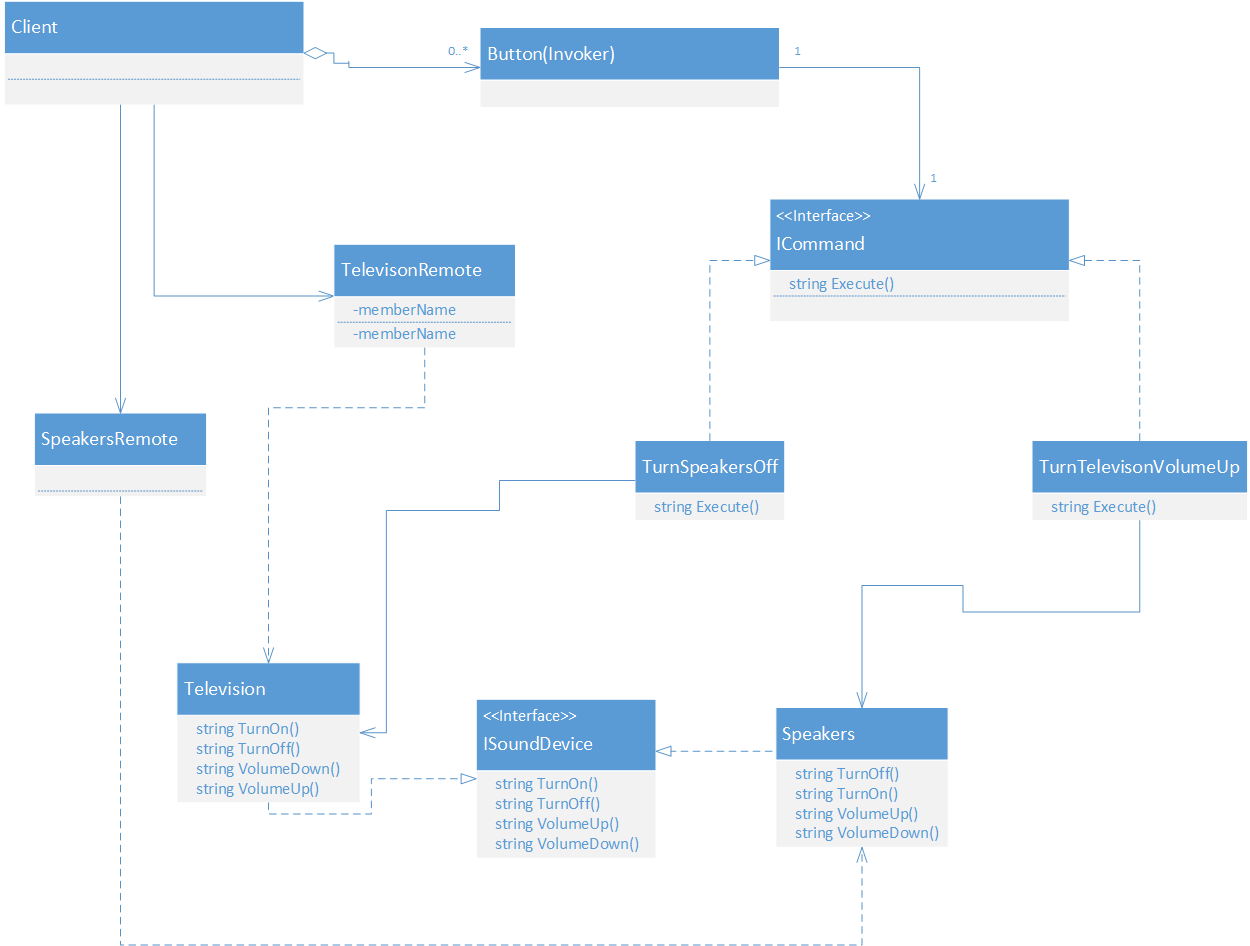
The application is related to remote controls, they can send commands to external sound devices. For now, the app is working with speakers and television. The GUI consists of two remote controls, each has four buttons for on/off and volume up/volume down, by pressing them the executed action is displayed in the log to the side.

**Architecture**

Application consists of few divisions:

* Client – GUI, displays output of the called commands.
* Commands – Stores all command classes, its divided into folders based on the device the command is related to.
* SoundDeviceRemotes – Stores ‘remotes’ classes for the different devices, the classes inside are used to return an instance of device that is related to the given remote.
* SoundDevices – The different kind of devices we can send commands to.
* Tests – Few basic tests for main functionality

After the implementation, the initial class diagram changed a bit and I concluded the following structure: (Sorry for the digital version, it just got too time consuming. It lacks some of the command classes because they are a lot and similar)



We can also relate the classes to the command pattern in that way:

* Button – invoker
* ISoundDevice - receiver
* Television – concrete receiver
* Speakers – concrete receiver
* ICommand - command
* TurnTelevisionVolumeUp – concrete command
* TurnSpeakersOff – concrete command
* SpeakersRemote, TelevisionRemote – external classes for creating objects

**Implementation**

The implementation covers the base requirements for implementation of command pattern. It is simplified and the whole functionality consists of returning strings and displaying them, by going through the whole process of the pattern of course.

What is special about the Client project is that for every button in each particular remote it has only one handler. It achieves that by ‘switching’ between the name of the clicked button in the handler method.

**Example process flow** (just for bigger clearance how the classes interact)

The client starts->Instances of ‘Television’ and ‘Speakers’ are generated by the ‘SpeakersRemote’ and ‘TelevisionRemote’->A certain button is clicked->’Command’ class is instantiated with the created TV or Speakers->The execute method of the command instance called->Returns a string result

**Conclusion**

This solution allows bigger extensibility and flexibility because we can easily create and use new commands without modifying the base and main classes of the application.