Ministry of education and Higher science of the Russian Federation

Peter The Great St. Petersburg Polytechnic University

1. —
2. Institute of cybersecurity and information security

**LABORATORY WORK №3**

1. **«Patterns»**
2. in the discipline «Object-oriented programming»
3. Perfomed
4. student gr. 4851001/80201 Ageeva А. I.

<*signature*>

1. Verified
2. teaching assistant Chernov А. Y.

<*signature*>

1. Saint-Petersburg
2. 2020
3. **Task**

Purpose: develop a VIM-like text editor in C ++ using patterns.

Tasks:

Write a text editor that meets the following conditions:

1. The program must be written in C ++ using STL.
2. The program must be developed using the MVC pattern.
3. The program must have TUI - Text User Interface. TUI must support cursor and status bar.
4. The program must work with 1-byte text encoding.
5. The program must support 4 modes: navigation mode, input mode, command mode and search mode.
6. To work with text, you need to use your own string class in the form of a static or dynamic library.
7. **Working process**

The text editor was developed in Windows 10. Since the ncurses library does not fit the selected OS, we used its counterpart - PDCurses. Also, the program did not do without using STL.

**2.1 Building a static library**

Since one of the requirements for the developed program is to use its own string class, the first thing the class was transformed into a library.

Libraries should be precompiled for several reasons. First, their code rarely changes. It would be a waste of time to re-compile the library every time you use it in a new program. Second, since all the code is precompiled into machine language, this prevents third parties from gaining access to the source code.

A static library was chosen to work with the MyString class. It was chosen so that this class was already embedded in a text editor and would not have to additionally transfer the .dll when running on someone else's computer. The disadvantages of such a library include, of course, the fact that the size of the executable file increases.

The import of the class into the library did not take much time, as it is easily done in Visual Studio. Steps for importing a class into the library:

1. Create a project of type Windows Desktop Application Wizard;
2. Enter project name and solution name;
3. In the Windows Desktop Application Project dialog box, under Application Type, select Static Library (.lib);
4. Under Advanced Options, clear the Precompiled header check box;

Then it remains only to insert the class code into the created project and compile.

Now that the static library is compiled and ready to use, you need to connect it to a future project. To do this, in the project settings, you need to change the path to the directories of the included files, and then connect the class header file to the project.

**2.2 Patterns**

The following patterns were used in the lab: Observer, MVC and Adapter. Let's dwell on each in more detail.

* + 1. **Observer**

Observer is a behavioral design pattern that creates a subscription mechanism that allows one object to watch and respond to events occurring in other objects.

This pattern is used so that the Model class can notify the View class about any events. Using this pattern, the View itself subscribes to updates, so the Model may not be aware of the View at all. This allows you to write code more flexibly and change the logic of the View class without changing the Model.

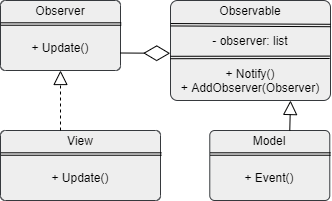
Structure:

Image 1 - Structure Observer

* + 1. **MVC (Model-View-Controller)**

MVC is a programming pattern that allows you to split your application logic into three parts:

* Model Receives data from the controller, performs the necessary operations and transfers them to the view.
* View (view or view). Receives data from the model and outputs it to the user and receives user commands.
* Controller Processes user actions, checks the received data and passes them to the model.

The advantages of this approach are obvious: modularity, extensibility, ease of maintenance and testing.

In the developed program, the Observer pattern was used to implement the MVC pattern. This connection was implemented between the Model and the View. The Model communicates to the View through a notification mechanism how the View should change.

The View stores a pointer to the Controller to inform the Model about changes through it. In turn, the Controller contains a pointer to the Model in order to transfer the necessary data there from the View (for example, the pressed key).

Structure:

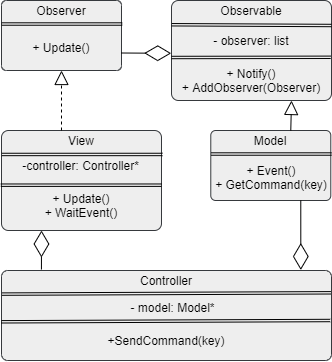


Image 2 - Structure MVC

* + 1. **Adapter**

An adapter is a structural design pattern that allows objects with incompatible interfaces to work together.

This pattern was used to develop a certain wrapper class to use libraries written in C. In the work, a library for working with control was used, namely PDCurses. Unfortunately, this library is written in C.

Structure:

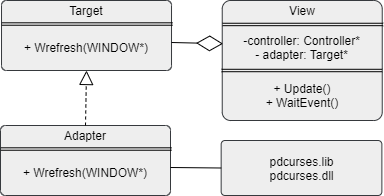


Image 3 - Structure Adapter

* 1. **Development of a text editor**

As it was written earlier, the program is built based on the MVC pattern. Let's consider the algorithm of the program:

1. The user, when interacting with the View, presses any keys. The View receives the code of the pressed key, but at this stage it cannot do anything except to give the received key to the Controller. The pointer to the Controller is stored in the View. It also stores a pointer to the Adapter. The adapter class is needed in order to access functions from the PDCurses library. This is done so that if you need to change the library to work with the console (for example, to another one), then the code would not have to be changed globally, but only to change the Adapter class. In addition, the View stores other fields needed to represent the text, for example, the x and y position in the console, the index of the start of the page and the end, etc.
2. The controller contains a pointer to the Model and passes the code of the given key to it. Basically, the Controller does not perform any function other than passing values ​​from the View to the Model.
3. The model gets a key. Depending on the current mode, one and the same key can be handled differently, so it is the model that knows about the current mode of operation. If you need to edit a specific line (delete a character, add, etc.), then this will execute the model and tell the View that the text has changed and the information on the screen needs to be updated. If the command touches navigation (moving the cursor, switching to command mode from navigation mode, etc.), then the Model passes the string to the View, which will make the necessary changes. The View is accessed through a notification.

The UML diagram is shown in the image below. Some of the methods have been overlooked because there is no communication with other classes through them.

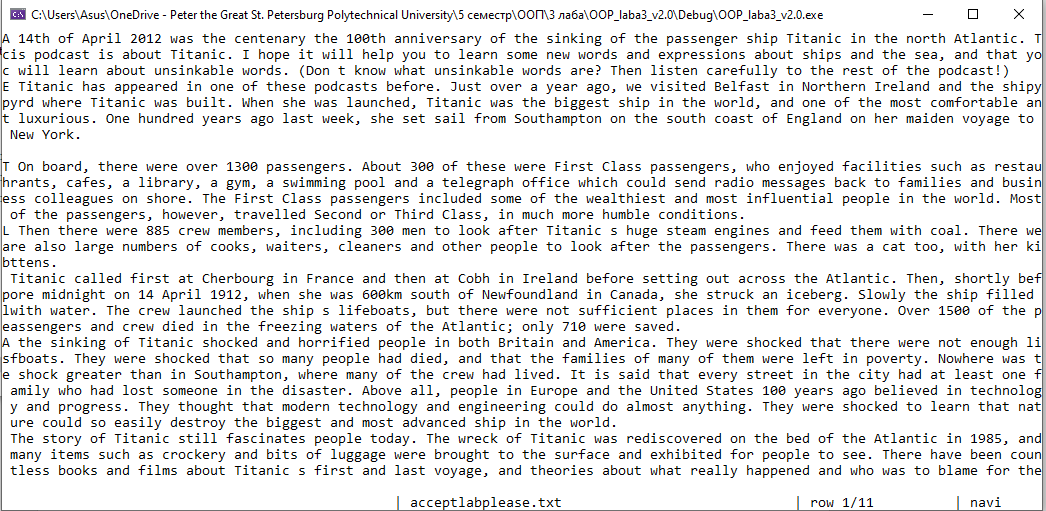
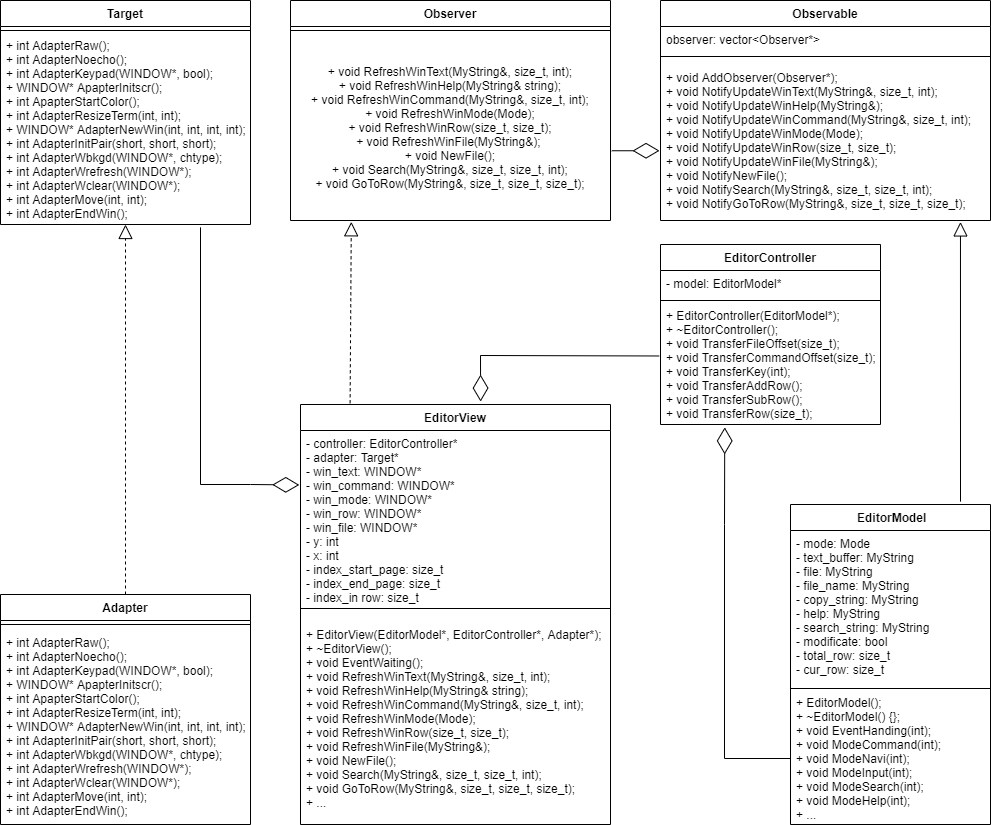
The interface of the developed text editor:

Image 5- Interface text editor

Image 4 - UML diagram

1. **Conclusion**

During the work, we managed to find out what patterns are and why they are needed, and also apply some of them when developing a program.

As a result, a text editor was developed, the work with which is the same as when working with the VIM editor.