Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Computer Science

Object-Oriented Programming

Laboratory work #2

SOLID principles

Authors:
Daniel Surdu

Supervisor:
Anastasia Șerșun

Chișinău - 2018

1 Work purpose:

The study and implementation of the two of the SOLID object-oriented design principles (S and I).

2 Task implementation

2.1 Single-Responsibility Principle

A class should have one and only one reason to change, meaning that a class should have only one job. For example, in the code below, we have two classes lichid and alcohol both of them have functions like changeDensity() and change_sweetness() and they also have function like output_values(). Here we have the situation when the class have multiple jobs, to change data and to output them on the screen.

```
1  class alcohol: public lichid
2  {
3      int content;
4      int sweetness;
5  public:
6      alcohol(char *name, float density, int content, int sweetness) :
           lichid(name, density)
7      {
8          this->content = content;
9          this->sweetness = sweetness;
10     }
11     void change_sweetness(int new_value);
12     void change_content(int new_value);
13     void output_values();
14 };
```

So, I created a third class OutputData that is responsible of the output, and first two classes remains only with the possibility to change data in the class.

```
1  class OutputData
2  {
3  public:
4      void output(Alcohol data);
5      void output(Lichid data);
6  };
```

2.2 Interface segregation principle

A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use. In the next snippet of code, the user wants to create an object of alcohol class lets say "Vodka" and this tipe of alcohol doesn't have such thing as sweetness and the function change_sweetness remains unused.

```
1  class alcohol: public lichid
2  {
3      int content;
4      int sweetness;
5  public:
6      alcohol(char *name, float density, int content, int sweetness) :
           lichid(name, density)
7      {
8          this->content = content;
9          this->sweetness = sweetness;
10     }
11     void change_sweetness(int new_value);
12     void change_content(int new_value);
13     void output_values();
14 };
```

In order to solve this situation, I created another class named Wine that inherits the Alcohol class and I extracted all extra data from Alcohol class.

```
1  class Alcohol: public Lichid
2  {
3  protected:
4      int content;
5  public:
6      Alcohol(char *name, float density, int content) : Lichid(name,
           density)
7      {
8          this->content = content;
9      }
10     void change_content(int new_value);
11     int get_content();
12 };
13 class Wine: public Alcohol
14 {
15     int sweetness;
16 public:
17     Wine(char *name, float density, int content, int sweetness) :
           Alcohol(name, density, content)
18     {
19         this->sweetness = sweetness;
20     }
21     void change_sweetness(int new_value);
22     int get_sweetness();
23 };
```

Conclusion

During this laboratory work I learned the 2 of 5 SOLID principles which are Single responsibility principle and Interface segregation principle. Those two principles make the program easier to debug and less dependent one class of another.

References

1 SOLID Principles, https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-