Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

APPOO

Laboratory work#1

Principles of Object Oriented Programming

Author:
Denis POSTICA

Supervisor:
Anastasia SERSUN

May 25, 2017

Laboratory work #1

1 Purpose of the laboratory work

Create a program that allows users to play a game. Use this program to demonstrate principles of Object Oriented programming:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

2 Laboratory work implementation

To show the basic use of the principles of Object Oriented Programming I implemented the 'Sea Battle' game. It's a two players command line game. When game start both player have to introduce a nickname. The board with placed ships is created randomly for each player. To choose a location to attack player has to enter a pair of two integers (between 1 and 10). If the location is valid player will see the updated board, if not valid player will have to re-enter a location. The player that destroys all the ships of the opponent wins. I chose Java programming language to perform the laboratory work task. Therefore, in most of the times I will refer to Java principles, definitions and so on. The code of the program can be found at Git repository `https://github.com/PosticaDenis/APPOO`

Laboratory work analysis

ENCAPSULATION PRINCIPLE

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

```java
package APPOO.lab1;

import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

/**
 * Created by Dennis on 29-Apr-17.
 **/
public class Board {

    private List<Ship> ships = new ArrayList<>();

    private int[][] availableOptionsBoard = new int[12][12]; // used to identify
        available options when generating ships
    private int[][] shipLocationsBoard = new int[12][12]; // used to store the
        information about actual state of the Board

    private int destroyedBlocks = 0;
    private boolean winner = false;

    private int startPosX, startPosY, endPosX, endPosY;
    private boolean myTurn;

    Board(){
        generateAllShips();
    }

  //...

    public int[][] getAvailableOptionsBoard(){
        return availableOptionsBoard;
    }

    public void setAvailableOptionsBoard(int[][] availableOptionsBoard){
        this.availableOptionsBoard = availableOptionsBoard;
    }

    public int[][] getShipLocationsBoard() {
        return shipLocationsBoard;
    }

    public void setShipLocationsBoard(int[][] shipLocationsBoard) {
        this.shipLocationsBoard = shipLocationsBoard;
    }
//...
```

In the example above (Board class) we can clearly see that all the variables are hidden from other classes (are private) and to get/update these values I created getters and setters.

POLYMORPHISM PRINCIPLE

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

ABSTRACTION PRINCIPLE

In Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

```
package APPOO.lab1;

/**
 * Created by Dennis on 29-Apr-17.
 **/

public interface ThePlayer {
    Board getBoard();
    int[] chooseLocation();
    int[] getLocationToAttack(boolean repeat);
    String setPlayerNickname();
    String getPlayerNickname();
}
```

```
package APPOO.lab1;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.util.Scanner;

/**
 * Created by Dennis on 29-Apr-17.
 **/
public class RealPlayer implements ThePlayer{

    private String playerNickname;
    private Board board;

    RealPlayer(boolean myTurn) {

        //...
    }

    public Board getBoard() {
        return board;
    }
    public int[] chooseLocation() {

        //...
    }

    public String setPlayerNickname() {
        //...
    }

    public String getPlayerNickname() {
        return playerNickname;
    }
}
```

As example of abstraction, also polymorphism, I created the interface ThePlayer in which user can just see what a player can do not how he does it.

INHERITANCE PRINCIPLE

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

In my program I didn't use inheritance but an example where I could use it is: class Ship, I could create four classes for each available length of the ship, these classes would extend the superclass.

Conclusions

These principles I talked about are what mostly define a object oriented programming language. To show the better understanding of these basic principles I wrote a program that, mostly, had some practical example of each principle. If making use of Encapsulation, Abstraction, Inheritance and Polymorphism the written program will be more easy to re-use, the secure level of the data in the program would increase, it would be easier to understand the code of the program and many other nice things.

References

1 Java - Inheritance, `https://www.tutorialspoint.com/java/java_inheritance.htm`

2 Java - Abstraction, `https://www.tutorialspoint.com/java/java_abstraction.htm`

3 Java - Encapsulation, `https://www.tutorialspoint.com/java/java_encapsulation.htm`

4 Java - Polymorphism, `https://www.tutorialspoint.com/java/java_inheritance.htm`