Ministerul Educației, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

**Raport**

**Lucrarea de laborator Nr. 3**

*Disciplina: Programarea Orientata pe Obiecte*

**Tema: Polimorfism și Abstractizare**

A efectuat: Toma Cezar, FI-191

A verificat: Eladii Vadim

Chișinău – 2020

**Polimorfism**

Acest principiu poate fi aplicat prin *Overriding* și *Overloading*.

1. Clasa *Pattern* are 3 metode *show().*Aici am implementat polimorfism prin ***Overloading****.*

package com.lab3.polymorphism;  
  
public class Pattern {  
  
 public void show(){  
 System.*out*.println('A');  
 }  
  
 public void show(String word){  
 System.*out*.println(word);  
 }  
  
 public void show(int num, int num2){  
 System.*out*.println(num \* num2);  
 }  
}

1. Clasa *Language* are doar o singură metoda *displayInfo().* Aici am implementat polimorfism prin ***Overriding****.*

package com.lab3.polymorphism;  
  
public class Language {  
 public void displayInfo() {  
 System.*out*.println("Pick a language");  
 }  
}

Mai apoi, avem clasa *French* care extinde clasa *Language*, dar în care se face *Override* la funcția displayInfo(), pentru a avea o metoda specifică clasei respective.

package com.lab3.polymorphism;  
  
public class French extends Language {  
 @Override  
 public void displayInfo() {  
 System.*out*.println("Bonjour");  
 }  
}

Același lucru a fost făcut și la clasa *English*.

package com.lab3.polymorphism;  
  
public class English extends Language{  
 @Override  
 public void displayInfo() {  
 System.*out*.println("Hello");  
 }  
}

**Abstractizare**

Putem utiliza clasa de tip *abstract* și *interface* pentru a implementa acest principiu.

1. Avem o clasă *Computer* de tip *abstract* unde vor fi implementate două metode *abstract* și una *non-abstract*. Acele *abstract* trebuie să fie declarate în *child classes* cu ajutorul la *Override*.

package com.lab3.abstraction;  
  
public abstract class Computer {  
  
 public abstract void speed();  
 public abstract void power(int p);  
 public void utility() {  
 System.*out*.println("Allows me to create Java Code");  
 }  
}

Clasa *Lenovo* extinde *Computer* și acolo vor fi declarate metodele care sunt *abstract* in clasa *Computer*.

package com.lab3.abstraction;  
  
public class Lenovo extends Computer {  
  
 @Override  
 public void speed() {  
 System.*out*.println("4.1ghz");  
 }  
  
 @Override  
 public void power(int p) {  
 System.*out*.println("Number of processors : " + p \* 4);  
 }  
}

1. Interfațele (avem două interfațe, *Polygon* și *Form*).

package com.lab3.abstraction;  
  
public interface Polygon {  
 String *name* = "Polygon";  
  
 void width(int width);  
 void getArea(int length, int width);  
}

package com.lab3.abstraction;  
  
public interface Form {  
 void typeOfForm();  
 void typeOfColor(String color);  
}

Clasa *Square* va implementa aceste două Interfațe.

package com.lab3.abstraction;  
  
public class Square implements Polygon, Form{  
 //Polygon Forms  
 @Override  
 public void width(int width) {  
 System.*out*.println("Width is " + width);  
 }  
  
 @Override  
 public void getArea(int length, int width){  
 System.*out*.println("Area is " + (length\*width));  
 }  
  
 //Form methods  
 @Override  
 public void typeOfForm(){  
 System.*out*.println("this is a square");  
 }  
  
 @Override  
 public void typeOfColor(String color){  
 System.*out*.println(color);  
 }  
}

**Main.java**

package com.lab3;  
  
import com.lab3.abstraction.Computer;  
import com.lab3.abstraction.Lenovo;  
import com.lab3.abstraction.Square;  
import com.lab3.polymorphism.English;  
import com.lab3.polymorphism.French;  
import com.lab3.polymorphism.Language;  
import com.lab3.polymorphism.Pattern;  
  
public class Main {  
  
 public static void main(String[] args) {  
 //Polymorphism using Method Overriding  
 Language l1 = new Language();  
 l1.displayInfo();  
  
 English en = new English();  
 en.displayInfo();  
  
 French fr = new French();  
 fr.displayInfo();  
  
 System.*out*.println("\n---Loop in an array of Languages---\n");  
  
 Language[] set = new Language[3];  
 set[0] = l1;  
 set[1] = en;  
 set[2] = fr;  
  
 for (Language a : set) {  
 a.displayInfo();  
 }  
  
 System.*out*.println("\n---Polymorphism using Method Overloading---\n");  
 Pattern p1 = new Pattern();  
 p1.show();  
 p1.show("Second method");  
 p1.show(5, 5);  
  
  
 System.*out*.println("\n---Abstraction using an abstract class---\n");  
 Lenovo user1 = new Lenovo();  
 user1.speed();  
 user1.power(2);  
 user1.utility();  
 //We cannot instantiate an abstract class  
 //Computer comp1 = new Computer();  
  
 System.*out*.println("\n---Abstraction using an interface class---\n");  
 Square sq1 = new Square();  
 sq1.getArea(4,4);  
 sq1.width(4);  
 sq1.typeOfForm();  
 sq1.typeOfColor("Red");  
 }  
}

**Concluzie**

În acest laborator, am aplicat principiile ale POO de polimorfism și de abstractizare. Am demonstrat cum se implementează polimorfismul prin Overriding și *Overloading*. *Overloading*-ul are loc în aceeași clasă, dar *Overriding*-ul are loc in subclase.

In cazul abstractizării, am creat o clasă abstractă în care au fost declarate metode *abstract* și *non-abstract*. În subclasa ei, am implementat blocul de cod pentru fiecare funcție în parte. În asta și constă de fapt acest principiu : fiecare subclasă trebuie să aibă un *Override* la funcțiile *abstract* din clasa generică.

Am creat și o *Interface*, care este în legătură strânsă cu clasele *abstract.* Ele se diferențiază prin faptul că toate metodele declarate în interiorul său sunt de tip *public abstract* by default. În acest caz, suntem siguri că vom avea metode abstracte, deci subclasele sunt obligate să aibă fiecare implementarea ei proprie. Iar în cazul subclaselor, ele vor utiliza cuvântul-cheie *implements* pentru a moșteni aceste clase.