Университет ИТМО, кафедра ВТ

Лабораторная работа №2 по “Программированию Интернет-Приложений”

Вариант 703

Работу выполнил

студент группы P3200

Рогов Я. С.

Преподаватель:

Харитонова А.Е.

Санкт-Петербург, 2016

Задание: скомпилировать и запустить исходный код на языке программирования Java, выданный в соответствии с вариантом. Разобраться том, как реализуются принципы объектно-ориентированного программирования в получившейся программе, и том, почему она выдаёт такой результат. Добавить комментарии в ключевые фрагменты программы, поясняющие её поведение.

Код программы:

```
// Вариант 703

public class Lab2 {
	public static void main(String[] args) {
		Metapod mother = new Metapod();
		Butterfree daughter = new Butterfree();

		// @brother can call methods defined in Metapod class
		// static methods - Metapod's implementation: shieldDust()
		// non-static methods - Butterfree's implementation: battleArmor()
		Metapod brother = new Butterfree();

		mother.battleArmor(); //Metapod.battleArmor()
		daughter.harden(brother); //Butterfree.harden(Metapod)
		mother.harden(daughter); //Metapod.harden(Butterfree)
		mother.machSpeed(); //Metapod.machSpeed()
		daughter.growth(); //Butterfree.growth()
		brother.shieldDust(); //Metapod.shieldDust()
		mother.shieldDust(); //Metapod.shieldDust()

		//Casted object uses Butterfree's methods and fields
		((Butterfree)brother).swordsDance(); //Butterfree.swordDance()
		daughter.workUp(); //Butterfree.workUp()

		brother.foresight(); //Metapod.foresight()
		daughter.battleArmor(); //Butterfree.battleArmor()
		daughter.focusEnergy(); //Metapod.focusEnergy()
		daughter.harden(mother); //Butterfree.harden(Metapod)
		brother.battleArmor(); //Butterfree.battleArmor()
		mother.harden(brother); //Metapod.harden(Metapod)
		daughter.shieldDust(); //Butterfree.shieldDust()

	}
}

class Metapod {
	double defense = 8.3;
	//Butterfree class inherits all non-private fields
	//Only childs and Metapod could access protected fields
	protected String rockFighting = "RockFighting";
	protected String fighting = "Fighting";
	protected String rock = "Rock";
	int power;
	public static int inflatable = 88;
	protected int threaded;

	public Metapod() {
		power = 88;
		threaded = 072; // <- Octal number. Decimal: 58
	}

	// this is instance initializer
	// it executes on class' instance creation
	// so @threaded will always be 072!
	{
		threaded = 63;
	}

	// static method : all instances of class use one instance of method
	public static void shieldDust() {
		System.out.println("Metapod casts Shield Dust");
	}
```

```
	public void foresight() {
		// @equals method compares ACTUAL content of strings!
		System.out.println(rockFighting.equals("Rock"+"Fighting")); //true

		// == operator compares string POINTERS of two objects! Bot their content!
		System.out.println(rockFighting == "Rock"+fighting); //false

		// Now that's tricky. Java compiler computes all constants
		// So constants of the same content will point to the same memory
		System.out.println(rockFighting == "Rock"+"Fighting"); //true
		System.out.println(rockFighting.equals(rock+"Fighting")); //true
		System.out.println(rockFighting == rock+fighting); //false
		System.out.println(rockFighting.equals(rock+fighting)); //true
	}

	public void harden(Butterfree p) {
		System.out.println("Metapod attacks Butterfree with Harden");
	}

	public void harden(Metapod p) {
		System.out.println("Metapod attacks Metapod with Harden");
	}

	public void machSpeed() {
		double weight = 6.7;

		System.out.println((defense - weight) == 1.6);
	}

	public void battleArmor() {
		System.out.println("Metapod casts Battle Armor");
	}

	public void focusEnergy() {
		System.out.println(inflatable - threaded);
		System.out.println(power - inflatable);
		System.out.println(threaded + power);
	}
}


// Butterfree inherits Metapod,
// i.e. it has all non-private fields and methods Metapod has
class Butterfree extends Metapod {
	// Private field : only Butterfree will be able to access this field!
	private byte fragile;
	private String ground = "Ground";
	private String groundFighting = "GroundFighting";
	double accuracy = 5.1;


	// another instance initializer
	// but there's no reassignment of @fragile!
	// So it will always be 0x88
	{
		// hexadecimal. Decimal : 136
		// But! byte type, so it interpreted as -(256-136) = -120
		fragile = (byte) 0x88;
	}

	public void workUp() {
		//And you can actually access Metapod's non-private fields!
		System.out.println(fragile - power);
		System.out.println(fragile + inflatable);
		System.out.println(threaded - fragile);
	}

	public static void shieldDust() {
		System.out.println("Butterfree casts Shield Dust");
	}

	public void harden(Butterfree p) {
		System.out.println("Butterfree attacks Butterfree with Harden");
	}

	public void harden(Metapod p) {
		System.out.println("Butterfree attacks Metapod with Harden");
	}
```

```
	public void battleArmor() {
		System.out.println("Butterfree casts Battle Armor");
	}

	public void growth() {
		//speed variable is accessible only in this method!
		double speed = 7.7;
		System.out.println((speed - accuracy) == 2.6);
	}

	public void swordsDance() {
		//invokes "canonical representation" of the string from the memory
		//i.e. any string of the same content point to the very same memory
		//if it's "canonical representation"
		System.out.println(groundFighting == (ground+fighting).intern());
		//but this are two different strings (different memory pointers)
		System.out.println(groundFighting == ground+fighting);
		//and this
		System.out.println(groundFighting == new String("GroundFighting"));
		//and this
		System.out.println(groundFighting == new String("Ground"+"Fighting"));
	}

}
```

Результат выполнения программы:

```
Metapod casts Battle Armor
Butterfree attacks Metapod with Harden
Metapod attacks Butterfree with Harden
false
false
Metapod casts Shield Dust
Metapod casts Shield Dust
true
false
false
false
-208
-32
178
true
false
true
true
false
true
Butterfree casts Battle Armor
30
0
146
Butterfree attacks Metapod with Harden
Butterfree casts Battle Armor
Metapod attacks Metapod with Harden
Butterfree casts Shield Dust
```

Вывод: в ходе выполнения данной лабораторной работы мной были закреплены основные принципы ООП, изучены наиболее используемые модификаторы и особенности областей видимости, а также некоторые правила работы со строками.