**Міністерство освіти і науки, молоді та спорту України**

**Національний технічний університет України**

**«Київський політехнічний інститут»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №6**

**Тема: «Наслідування та поліморфізм.»**

**Виконав: Студент групи ІО-33**

**Заколенко Р.К.**

**Перевірив: Викладач**

**Невдащенко М.В.**

**Київ 2014**

1. **Завдання**

C13 = 6 ( 3308 mod 13 = 6 )

Завдання: визначити ієрархію амуніції лицаря. Екіпірувати лицаря. Порахувати вартість амуніції. Провести сортування амуніції за вагою. Знайти елементи амуніції, що відповідає заданому діапазону цін.

1. **Опис програми**

Програма містить п’ять класів:

1. Клас Munitions;
2. Клас Armor (extends Munitions);
3. Клас Helmet (extends Munitions);
4. Клас Shield (extends Munitions);
5. Клас Main (із виконавчим методом).

У виконавчому методі ми створюємо об’єкти класів Helmet, Armor, Shield. Завдяки тому що дані класи наслідують єдиний клас Munitions, утворюємо масив типу Munitions із даних об’єктів.

Далі виконуємо завдання:

1. Рахуємо загальну вартість амуніцій лицаря сумуючи поля “price” елементів масиву типу Munitions за допомогою метода cost.
2. Сортуємо амуніцію лицаря за полем “weight” за допомогою метода sortByWeight.
3. Знаходимо амуніцію, яка знаходиться у даному діапазоні вартості за допомогою метода priceRange.
4. **Код програми**

package ua.kpi.io33.zakolenko.lab6;

import java.util.Arrays;

import java.util.Comparator;

/\*\*

\* This class describe data about object Munitions.

\*

\* @author Roman Zakolenko

\* @version 1,6 05 May 2014

\*/

public class Munitions {

/\*\*

\* The field "weight" contains value of weight munition

\*/

private int weight;

/\*\*

\* The field "price" contains value of price munition

\*/

private int price;

/\*\*

\* THe field "material" contains value of material munition

\*/

private String material;

/\*\*

\* @return value of field weight

\*/

public int getWeight() {

return weight;

}

/\*\*

\* @param weight

\* assigned with field weight

\*/

public void setWeight(int weight) {

this.weight = weight;

}

/\*\*

\* @return value of field price

\*/

public int getPrice() {

return price;

}

/\*\*

\* @param price

\* assigned with field price

\*/

public void setPrice(int price) {

this.price = price;

}

/\*\*

\* @return value of fields material

\*/

public String getMaterial() {

return new String(material);

}

/\*\*

\* @param material

\* assigned with field material

\*/

public void setMaterial(String material) {

this.material = new String(material);

}

/\*\*

\* @param w

\* assigned with field weight

\* @param p

\* assigned with field price

\* @param m

\* assigned with field material

\*/

public Munitions(int w, int p, String m) {

setWeight(w);

setPrice(p);

setMaterial(m);

}

/\*\*

\* @param obj

\* contains the field whose values ​​will be installed at the new

\* object of class Munitions

\*/

public Munitions(Munitions obj) {

setWeight(obj.getWeight());

setPrice(obj.getPrice());

setMaterial(obj.getMaterial());

}

/\*\*

\* @param m

\* contains elements of class Munitions

\* @return the sum of the fields price of objects of class Munitions that

\* contains in array m

\*/

public static int cost(Munitions[] m) {

int price = 0;

for (int i = 0; i < m.length; i++) {

price = price + m[i].getPrice();

}

return price;

}

/\*\*

\* @param m

\* contains elements of class Munitions

\*/

public static void sortedByWeight(Munitions[] m) {

Arrays.sort(m, new Comparator<Munitions>() {

@Override

public int compare(Munitions m1, Munitions m2) {

return m1.getWeight() - m2.getWeight();

}

});

}

/\*\*

\* @param m

\* contains elements of class Munitions

\* @param a

\* is initial value range

\* @param b

\* is end of the range

\* @return array of objects of class Munitions that go to the specified

\* range

\*/

public static Munitions[] priceRange(Munitions[] m, int a, int b) {

int length = 0;

for (int i = 0; i < m.length; i++) {

if (m[i].getPrice() >= a && m[i].getPrice() <= b) {

length++;

}

}

Munitions[] m1 = new Munitions[length];

for (int i = 0; i < m.length; i++) {

if (m[i].getPrice() >= a && m[i].getPrice() <= b) {

m1[i] = m[i];

}

}

if (length == 0) {

return null;

} else {

return m1;

}

}

}

package ua.kpi.io33.zakolenko.lab6;

/\*\*

\* This class extends class Munitions and consist data about object Armor.

\*

\* @author Roman Zakolenko

\* @version 1,6 05 May 2014

\*/

public class Armor extends Munitions {

/\*\*

\* @param w

\* assigned with field weight

\* @param p

\* assigned with field price

\* @param m

\* assigned with field material

\*/

public Armor(int w, int p, String m) {

super(w, p, m);

}

}

package ua.kpi.io33.zakolenko.lab6;

/\*\*

\* This class extends class Munitions and describe data about object Helmet.

\*

\* @author Roman Zakolenko

\* @version 1,6 05 May 2014

\*/

public class Helmet extends Munitions {

/\*\*

\* @param w

\* assigned with field weight

\* @param p

\* assigned with field price

\* @param m

\* assigned with field material

\*/

public Helmet(int w, int p, String m) {

super(w, p, m);

}

}

package ua.kpi.io33.zakolenko.lab6;

/\*\*

\* This class extends class Munitions and describe data about object Shield.

\*

\* @author Roman Zakolenko

\* @version 1,6 05 May 2014

\*/

public class Shield extends Munitions {

/\*\*

\* @param w

\* assigned with field weight

\* @param p

\* assigned with field price

\* @param m

\* assigned with field material

\*/

public Shield(int w, int p, String m) {

super(w, p, m);

}

}

package ua.kpi.io33.zakolenko.lab6;

/\*\*

\* Is used to create objects of the class Helmet, Armor and Shield which are

\* then sorted by the specified field.

\*

\* @author Roman Zakolenko

\* @version 1,7 05 May 2014

\* @see ua.kpi.io33.zakolenko.lab6.Munitions ua.kpi.io33.zakolenko.lab6.Shield

\* ua.kpi.io33.zakolenko.lab6.Armor ua.kpi.io33.zakolenko.lab6.Helmet

\*/

public class Main {

public static void main(String[] args) {

Helmet h = new Helmet(35, 115, "Железо");

Armor a = new Armor(25, 200, "Сталь");

Shield s = new Shield(10, 110, "Дерево");

Munitions[] munitions = { h, a, s };

System.*out*.println("Общая стоимость амуниции рыцаря: "

+ Munitions.*cost*(munitions));

Munitions.*sortedByWeight*(munitions);

System.*out*.println("Отсортированная по весу аммуниция рыцаря:");

for (int i = 0; i < munitions.length; i++) {

System.*out*.print(munitions[i].getWeight() + " ");

}

System.*out*.println();

Munitions d[];

d = Munitions.*priceRange*(munitions, 50, 200);

System.*out*.println("Аммуниция из данного диапозона цен:");

for (int i = 0; i < d.length; i++) {

System.*out*.print(d[i].getPrice() + " ");

}

}

}

1. **Висновок**

В результаті виконання лабораторної роботи я ознайомився за такими механізмами об’єктно-орієнтованого програмування, як наслідування та поліморфізм. Був створений узагальнений клас Munition та 3 дочірні класи.

Для виконання вказаного варіантом завдання були створені три методи:

1. Статичний метод класу Munition cost, параметром якого є масив з об’єктів класу Munition. Метода повертає значення типу int (загальна вартість амуніуії).
2. Статичний метод класу Munition sortedByWeight, який за допомогою стандартних методів сортування мови програмування Java (Arrays.sort) сортує масив об’єктів класу Munition за полем “weight“.
3. Статичний метод класу Munition priceRange, який повертає масив об’єктів класу Munition, які потраплять у діапазон цін, інакше, якщо таких не було знайдено, то метод повертає null.