Національний Технічний Університет України

«Київський Політехнічний Інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №6

*Наслідування та поліморфізм*

|  |  |
| --- | --- |
| Прийняв  Ст.в. Невдащенко М. В.  «\_\_»\_\_\_\_\_\_\_\_ 2014 р. | Виконала Студентка 1-ого курсу ФІОТ  групи ІО-32  Шапран К.О. |

1. ***Варіант завдання***

Номер залікової книжки  *3229*

Варіант: *C3 = 3229 mod 13 = 5*

1. Визначити ієрархію дорогоцінного та напівкоштовного каміння. Відібрати

камені для намиста. Порахувати загальну вагу (у каратах) і вартість намиста.

Провести сортування каміння намиста за цінністю. Знайти каміння в намисто,

що відповідає заданому діапазону параметрів прозорості.

1. Створити узагальнений клас та не менше 3 класів-нащадків, що описують задану варіантом (п.2) область знань. Створити клас, що складається з масиву об’єктів, з яким можна виконати вказані варіантом дії. Необхідно обробити всі виключні ситуації, що можуть виникнути під час виконання програмного коду. Всі змінні повинні бути описані та значення їх задані у виконавчому методі. Код повинен відповідати стандартам JCC та бути детально задокументований.
2. ***Опис алгоритму програми***

Створено клас каміння, що складається з 4 полів: назва каміння, вага, прозорість та ціна, і 3 нащадки цього класу: клас дорогоцінного каміння, клас напівдорогоцінного каміння та клас штучного каміння. Клас намиста містить 1 поле – масив каміння. У ньому описано метод додавання нових елементів у масив, метод визначення загальної ваги та вартості намиста, пошуку каміння в намисті, що відповідає заданому діапазону параметрів прозорості, а також сортування каміння намиста за цінністю. Результати виконання цих методів виводяться на екран.

1. ***Програмний код***

Class Main

/\*\*

\* @(#) Main.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Main class, which has executive method main

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Main {

public static void main(String[] args){

/\*@param n object of class Stones

\* @param sNumber length array of stones

\*/

int sNumber = 6;

Necklace n = new Necklace(sNumber);

/\*Filling an array of stones\*/

n.addStone(new Stones ("Rubin", 3.95, 600, 2.1));

n.addStone(new Semiprecious\_stones("Agate", 5.89, 200, 1.7, 82));

n.addStone(new Artificial\_stones("Quartz ", 10.2, 573, 5.3, 0.17));

n.addStone(new Precious\_stones("Diamond", 2.72, 11830, 15.1, 28));

n.addStone(new Semiprecious\_stones("Hyacinth", 3.91, 167, 2.56, 52));

n.addStone(new Artificial\_stones("Onyx", 8.34, 720, 4.21, 3.57));

/\*Output the total weight of necklace\*/

System.out.println("Weight of the necklace: "+n.totalW()+" ct.");

/\*Output the total price of necklace\*/

System.out.println("Weight of the necklace: "+n.totalP()+" grn.");

/\*@param start,finish - range of clarity\*/

double start = 2;

double finish = 16;

/\*@param chooseClarity array of stones with the clarity of the specified range\*/

Stones[] chooseClarity = n.searchStone(start,finish);

System.out.println('\n'+"Stones with the defined clarity (from "+start+" to "+finish+"): ");

for(int i = 0;chooseClarity[i] != null && i<chooseClarity.length;i++){

System.out.println("Name: "+chooseClarity[i].getName());

System.out.println("Clarity of the stone: "+chooseClarity[i].getClarity());

}

/\*Sorting an array of stones by value and display the sorted array\*/

n.sortPrice();

System.out.println(n);

}

}

Class Stones

/\*\*

\* @(#) Stones.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Class Stones, which describes the stone

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Stones {

/\*\*

\* @param name

\* @param weight

\* @param price

\* @param clarity

\*/

private double weight, clarity;

private int price;

private String name;

/\*\*Constructor of Stones\*/

Stones(String name, double weight, int price, double clarity){

this.name = name;

this.weight = weight;

this.price = price;

this.clarity = clarity;

}

/\*\*

\* @return name

\*/

public String getName(){

return name;

}

/\*\*

\* @return weight

\*/

public double getWeight(){

return weight;

}

/\*\*

\* @return price

\*/

public int getPrice(){

return price;

}

/\*\*

\* @return clarity

\*/

public double getClarity(){

return clarity;

}

}

Class Precious\_stones

/\*\*

\* @(#) Precious\_stones.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Class Precious\_stones, which describes

\* the precious stone

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Precious\_stones extends Stones {

/\*\*

\* @param rare

\*/

private int rare;

Precious\_stones(String name, double weight, int price, double clarity, int rare){

super(name, weight, price, clarity);

this.rare = rare;

}

}

Class Semiprecious\_stones

/\*\*

\* @(#) Semiprecious\_stones.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Class Semiprecious\_stones, which describes

\* the semiprecious stone

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Semiprecious\_stones extends Stones{

/\*\*

\* @param durability//долговечность

\*/

private int durability;

Semiprecious\_stones(String name, double weight, int price, double clarity, int durability){

super(name, weight, price, clarity);

this.durability = durability;

}

}

Class Artificial\_stones

/\*\*

\* @(#) Artificial\_stones.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Class Artificial\_stones, which describes

\* the artificial stone

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Artificial\_stones extends Stones{

/\*\*

\* @param number\_of\_defects //количество дефектов

\*/

private double number\_of\_defects;

Artificial\_stones(String name, double weight, int price, double clarity, double number\_of\_defects){

super(name, weight, price, clarity);

this.number\_of\_defects =number\_of\_defects ;

}

}

Class Necklace

import java.util.Arrays;

/\*\*

\* @(#) Necklace.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Class Necklace, which describes

\* the necklace of stones

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Necklace {

/\*\*

\* @param stone

\*/

private Stones[] stone;

/\*\*

\* Constructor of class Necklace

\* @param number length array of stones

\*/

Necklace(int number){

stone = new Stones[number];

}

/\*\*Add the stones in a necklace\*/

void addStone(Stones stn){

int k = 0;

boolean t = true;

for(int i=0;(i<stone.length)&&t;i++){

if (stone[i] == null){

k = i;

t = false;

}

}

if (k != stone.length)

stone[k] = stn;

}

/\*\*The method returns total weight of a necklace \*/

public double totalW(){

double total\_weight=0.0;

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

total\_weight+=stone[i].getWeight();

}

return total\_weight;

}

/\*\*The method returns total price of a necklace\*/

public double totalP(){

double total\_price=0.0;

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

total\_price+=stone[i].getPrice();

}

return total\_price;

}

/\*\*Sort by stone value\*/

public void sortPrice(){

Arrays.sort(stone, new Sort());

}

/\*\*The method sorts the array of stones for clarity (from smallest to largest)\*/

public Stones[] searchStone(double start, double finish){

Stones[] stn = new Stones[stone.length];

int k = 0;

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

if(stone[i].getClarity()>=start && stone[i].getClarity()<=finish){

stn[k] = stone[i];

k++;

}

}

return stn;

}

@Override

public String toString(){

String s = '\n'+"Stones sorted by price:";

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

s += '\n'+"Name: "+stone[i].getName()+'\n'+"Price: "+stone[i].getPrice();

}

return s;

}

}

Class Sort

import java.util.Comparator;

/\*\*

\* @(#) Sort.java 1.0 07/05/14

\*

\* Copyright (c) 2014 Karina Shapran

\*/

/\*\* Class Sort implements Comparator interface

\* to sort an array of objects of class Stones

\* by value from smallest to largest

\*

\* @version 1.0 07 May 2014

\* @author Karina Shapran

\* @since 1.0

\*/

public class Sort implements Comparator <Stones>{

@Override

public int compare(Stones obj1, Stones obj2){

return obj2.getPrice() - obj1.getPrice();

}

}

1. ***Висновок***

Під час лабораторної роботи я ознайомилася із механізмом наслідування і принципом поліморфізму в мові програмування Java, створивши ієрархію класів та перевизначивши один із стандартних методів toString().

Код був оформлений і задокументований відповідно до стандартів JCC.