Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ярославский государственный технический университет»

Кафедра «Информационных систем и технологий»

Отчет по лабораторной работе защищен

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_ В.С.Пашичев

«\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2022

**СИСТЕМЫ СБОРКИ, UNIT-ТЕСТИРОВАНИЕ**

Отчет по лабораторной работе №4 по курсу «Объектно-ориентированное программирование»

ЯГТУ 09.03.04-009 ЛР

|  |  |  |
| --- | --- | --- |
|  |  | Отчет выполнила  студентка группы ЦПИ-21    К.В.Мельникова  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2022 |
|  |  |  |

2022

**Цель работы**Разнести свой код по классам/методам так, чтобы для него можно было написать автоматические Unit-тесты. Unit-тесты должны работать по принципам «черного ящика» — проверка на выходящий результат при входящих данных.

**Исходный код**

Main.java

// Var 9  
package com.company;  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
  
 int[] componentTypes = {1, 2, 1, 4, 3, 3, 3, 4, 5};  
 int[] componentPrices = {10, 15, 20, 25, 20, 15, 30, 20, 10};  
 int[] componentPower = {30, 40, 20, 15, 20, 40, 50, 15, 10};  
  
 int[] wireTypes = {1, 2, 1, 4, 3, 3, 3, 4, 5};  
 int[] wirePrices = {10, 15, 20, 25, 20, 15, 30, 30, 5};  
  
 int[] computerPartTypes = {4, 1, 2};  
 int[] computerPartPrices = {50, 45, 55};  
 int[] computerPartPower = {40, 50, 40};  
  
 int maxType = 1;  
 int maxPrice = componentPrices[0];  
 int maxPower = componentPower[0];  
 int powerSum = 0;  
 int priceSum = 0;  
  
 int componentN = componentTypes.length;  
 ArrayList<Component> parts = new ArrayList<>();  
 for (int i = 0; i < componentN; i++) {  
  
 parts.add(new Component(componentTypes[i], componentPrices[i], componentPower[i]));  
  
 maxType = Math.max(maxType, parts.get(i).type);  
 maxPrice = Math.max(maxPrice, parts.get(i).price);  
 maxPower = Math.max(maxPower, parts.get(i).power);  
  
 priceSum += parts.get(i).price;  
 powerSum += parts.get(i).power;  
 }  
 int wireN = wireTypes.length;  
 for (int i = 0; i < wireN; i++) {  
 parts.add(new Wire(wireTypes[i], wirePrices[i]));  
  
 maxType = Math.max(maxType, parts.get(i).type);  
 maxPrice = Math.max(maxPrice, parts.get(i).price);  
  
 priceSum += parts.get(i).price;  
 }  
  
 System.out.println("Средняя стоимость: " + (double) priceSum/(componentN + wireN));  
 System.out.println("Средняя мощность: " + (double) powerSum/componentN);  
 System.out.println("Максимальная стоимость: " + maxPrice);  
 System.out.println("Максимальная мощность: " + maxPower);  
 System.out.println();  
  
 int computerPartN = computerPartTypes.length;  
 ArrayList<ComputerPart> computerParts = new ArrayList<>();  
 for (int i = 0; i < computerPartN; i++) {  
 computerParts.add(new ComputerPart(computerPartTypes[i], computerPartPrices[i], computerPartPower[i]));  
 }  
  
 ArrayList<Detail> details = new ArrayList<>();  
 for (int i = 0; i < componentN + wireN; i++) {  
 details.add(parts.get(i));  
 }  
  
 for (int i = 0; i < computerPartN; i++) {  
 details.add(computerParts.get(i));  
 }  
  
 int priceSumOfType = 0;  
 int powerSumOfType = 0;  
 details.sort(Comparator.comparing(Detail::getType));  
  
 int componentAmountPT = 0;  
 int wireAmountPT = 0;  
 int computerPartAmountPT = 0;  
  
 for (int i = 0; i < componentN+wireN+computerPartN-1; i++) {  
 priceSumOfType += details.get(i).price;  
  
 if (details.get(i).getClass().getSimpleName().equals("Component")) {  
 componentAmountPT ++ ;  
 powerSumOfType += details.get(i).power;  
 }  
 else if (details.get(i).getClass().getSimpleName().equals("ComputerPart")) {  
 computerPartAmountPT++;  
 powerSumOfType += details.get(i).power;  
 }  
 else  
 wireAmountPT ++ ;  
  
 if (!Objects.equals(details.get(i).type, details.get(i+1).type)) {  
 System.out.println("Комплектующая типа " + details.get(i).type +  
 " имеет " + componentAmountPT + " деталей, " + wireAmountPT + " проводов и " +  
 computerPartAmountPT + " компьютерных частей" +  
 ": средняя стоимость: " + (double)priceSumOfType/(componentAmountPT+wireAmountPT+computerPartAmountPT) +  
 "; средняя мощность: " + (double)powerSumOfType/(componentAmountPT+computerPartAmountPT));  
  
 priceSumOfType = 0;  
 powerSumOfType = 0;  
 componentAmountPT = 0;  
 wireAmountPT = 0;  
 computerPartAmountPT = 0;  
 }  
  
 if (i == details.size()-2) {  
 priceSumOfType += details.get(i+1).price;  
  
 if (details.get(i+1).getClass().getSimpleName().equals("Component")) {  
 componentAmountPT ++ ;  
 powerSumOfType += details.get(i+1).power;  
 }  
 else if (details.get(i+1).getClass().getSimpleName().equals("ComputerPart")) {  
 computerPartAmountPT++;  
 powerSumOfType += details.get(i+1).power;  
 }  
 else  
 wireAmountPT ++ ;  
  
 System.out.println("Комплектующая типа " + details.get(i+1).type +  
 " имеет " + componentAmountPT + " деталей, " + wireAmountPT + " проводов и " +  
 computerPartAmountPT + " компьютерных частей" +  
 ": средняя стоимость: " + (double)priceSumOfType/(componentAmountPT+wireAmountPT+computerPartAmountPT) +  
 "; средняя мощность: " + (double)powerSumOfType/(componentAmountPT+computerPartAmountPT));  
  
 }  
 }  
 System.out.println();  
  
 for (Detail detail : details) {  
 detail.printInfo();  
 }  
 System.out.println();  
  
 List<Component> price = DataProcessor.top3Price(parts);  
 for (Component component : price) {  
 component.printInfo();  
 }  
 System.out.println();  
  
 System.out.print("Введите цену: ");  
 while (true) {  
 try {  
 Scanner scanner = new Scanner(System.in);  
 DataProcessor.searchPrice(parts, scanner.nextInt()).printInfo();  
 break;  
 }  
 catch (InputMismatchException e) {  
 System.out.print("Вы ввели строку, а не число.\nПожалуйста, введите число: ");  
 }  
 catch (NegativeInput e) {  
 System.out.print("Вы ввели отрицательное число.\nПожалуйста, введите число: ");  
 }  
 catch (IndexOutOfBoundsException e) {  
 System.out.print("Объект с таким ценником не найден.");  
 break;  
 }  
 }  
 }  
}

Detail.java

package com.company;  
  
public abstract class Detail {  
 public int type;  
 public int price;  
 public int power;  
  
 public Integer getType() { return type; }  
  
 public Integer getPrice() { return price; }  
  
 public void printInfo() {}  
  
 @Override public boolean equals(Object obj)  
 {  
 if (this == obj)  
 return true;  
  
 if (obj == null || this.getClass() != obj.getClass())  
 return false;  
  
 Detail p1 = (Detail)obj;  
  
 return this.type == p1.type  
 && this.price == p1.price  
 && this.power == p1.power;  
 }  
}

Component.java

package com.company;  
  
public class Component extends Detail {  
 public Component(int type, int price, int power) {  
 this.type = type;  
 this.price = price;  
 this.power = power;  
 }  
  
 public void printInfo() {  
 System.out.println("Компонент: тип " + type + ", цена " + price + ", мощность: " + power);  
 }  
}

Wire.java

package com.company;  
  
class Wire extends Component {  
 public Wire(int type, int price) {  
 super(type, price, 0);  
 }  
  
 public String getInfo() {  
 return "Провод: тип " + type + ", цена " + price;  
 }  
  
 public void printInfo() {  
 System.out.println("Провод: тип " + type + ", цена " + price);  
 }  
}

ComputerPart.java

package com.company;  
  
public class ComputerPart extends Detail {  
 public ComputerPart(int type, int price, int power) {  
 this.type = type;  
 this.price = price;  
 this.power = power;  
 }  
  
 public void printInfo() {  
 System.out.println("Компьютерная часть: тип " + type + ", цена " + price + ", мощность: " + power);  
 }  
}

DataProcessor.java

package com.company;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Comparator;  
import java.util.stream.\*;  
  
public class DataProcessor {  
 public static List<Component> top3Price(ArrayList<Component> list) {  
 list.sort(Comparator.comparing(Detail::getPrice).reversed());  
 List<Component> top3 = list.stream()  
 .filter(element -> element.price < 30)  
 .filter(element -> element.price > 10)  
 .limit(3)  
 .collect(Collectors.toList());  
  
 return top3;  
 }  
  
 public static Component searchPrice(ArrayList<Component> list, int price) throws NegativeInput {  
 if (price < 0) { throw new NegativeInput("positive number expected"); }  
  
 List<Component> exactPrice = list.stream()  
 .filter(element -> element.price == price).toList();  
  
 return exactPrice.get(0);  
 }  
}

NegativeInput.java

package com.company;  
  
public class NegativeInput extends Exception {  
 public NegativeInput(String message) { super(message); }  
}

DataProcessorTest.java

package com.company;  
  
import org.junit.jupiter.api.Test;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class DataProcessorTest {  
 @Test  
 void top3Price() {  
 ArrayList<Component> list = new ArrayList<>();  
 list.add(new Component(1, 10, 30));  
 list.add(new Component(4, 20, 20));  
 list.add(new Component(3, 25, 15));  
 list.add(new Wire(2, 15));  
 list.add(new Wire(3, 30));  
 list.add(new Wire(5, 5));  
 List<Component> result = DataProcessor.top3Price(list);  
  
 List<Component> rightList = new ArrayList<>();  
 rightList.add(new Component(3, 25, 15));  
 rightList.add(new Component(4, 20, 20));  
 rightList.add(new Wire(2, 15));  
  
 List<Component> wrongList = new ArrayList<>();  
 wrongList.add(new Component(1, 10, 30));  
 wrongList.add(new Wire(3, 30));  
 wrongList.add(new Wire(5, 5));  
  
 assertEquals(rightList.get(0), result.get(0));  
 assertEquals(rightList.get(1), result.get(1));  
 assertEquals(rightList.get(2), result.get(2));  
  
 assertNotEquals(wrongList.get(0), result.get(0));  
 assertNotEquals(wrongList.get(1), result.get(1));  
 assertNotEquals(wrongList.get(2), result.get(2));  
 }  
  
 @Test  
 void searchPrice() throws NegativeInput {  
 ArrayList<Component> list = new ArrayList<>();  
 list.add(new Component(1, 10, 30));  
 list.add(new Component(4, 20, 20));  
 list.add(new Component(3, 25, 15));  
 list.add(new Wire(2, 15));  
 list.add(new Wire(3, 30));  
 list.add(new Wire(5, 5));  
 Component result = DataProcessor.searchPrice(list, 5);  
  
 Component rightAnswer = new Wire(5, 5);  
 Component wrongAnswer = new Component(1, 10, 30);  
  
 assertEquals(rightAnswer, result);  
 assertNotEquals(wrongAnswer, result);  
 }  
}

WireTest.java

package com.company;  
  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class WireTest {  
 @Test  
 void getInfo() {  
 Wire wire = new Wire(5, 5);  
  
 assertEquals("Провод: тип 5, цена 5", wire.getInfo());  
 assertNotEquals("Компонент: тип 4, цена 20, мощность 20", wire.getInfo());  
 }  
}

**Вывод:** я разнесла свой код по классам/методам так, чтобы для него можно было написать автоматические Unit-тесты. Unit-тесты работают по принципам «черного ящика» — проверка на выходящий результат при входящих данных.