**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине: «Технологии разработки программного обеспечения»

на тему: «Развитие представлений о разработке программ. Объектно-ориентированное программирование в *Java*»

Выполнил: студент гр. ИТИ-21

Тимошенко И.Ю.

Принял: преподаватель

Стефановский И. Л.

Гомель 2024

**Цель работы:** изучить основы проектирования и создания программ при помощи объектно-ориентированного языка программирования *Java*.

**Задание:**

1. Разработать UML-диаграмму иерархии классов, согласно варианта

|  |
| --- |
| 1.1 Создать класс Sneakers (Кроссовки) с полями name, cost, type, producer. Кроссовки могут быть футбольные, баскетбольные и т.п. (реализовать через наследование)  1.2 Создать приватный внутренний класс Producer с полями name и country.  1.3 Создать не менее 10 объектов, содержащих информацию о различных кроссовках  1.4 Подсчитать количество производителей.  1.5 Подсчитать среднюю стоимость обуви по каждому производителю.  1.6 Подсчитать среднюю стоимость обуви по каждому типу.  1.7 Дополнить иерархию классов новым типом кроссовок, при этом нельзя изменять методы, реализующие пункты 1.4 – 1.6 |

2. При наименовании компонентов руководствоваться соглашением о наименовании.

3. При описании иерархии использовать наследование и композицию.

4. На основе *UML*-диаграммы разработать иерархию классов на языке Java.

5. Весь код должен быть снабжен элементами документирования.

6. Разработанную иерархию поместить в .*jar* файл для дальнейшего использования в качестве библиотечных классов.

7. Создать консольное приложение для демонстрации работы созданных классов.

8. Составить отчет о проделанной работе.

**Ход работы:**

Разработаем *UML*-диаграмму классов, включающую в себя суперкласс *Sneakers*, от которого будут наследоваться все остальные кроссовки. Дополнительно полями являются перечисление типов кроссовок, объект приватного внутреннего класса *Producer*. Также опишем классы, реализующие сами наследуемые виды кроссовок. Полученная диаграмма представлена на рисунке 1.

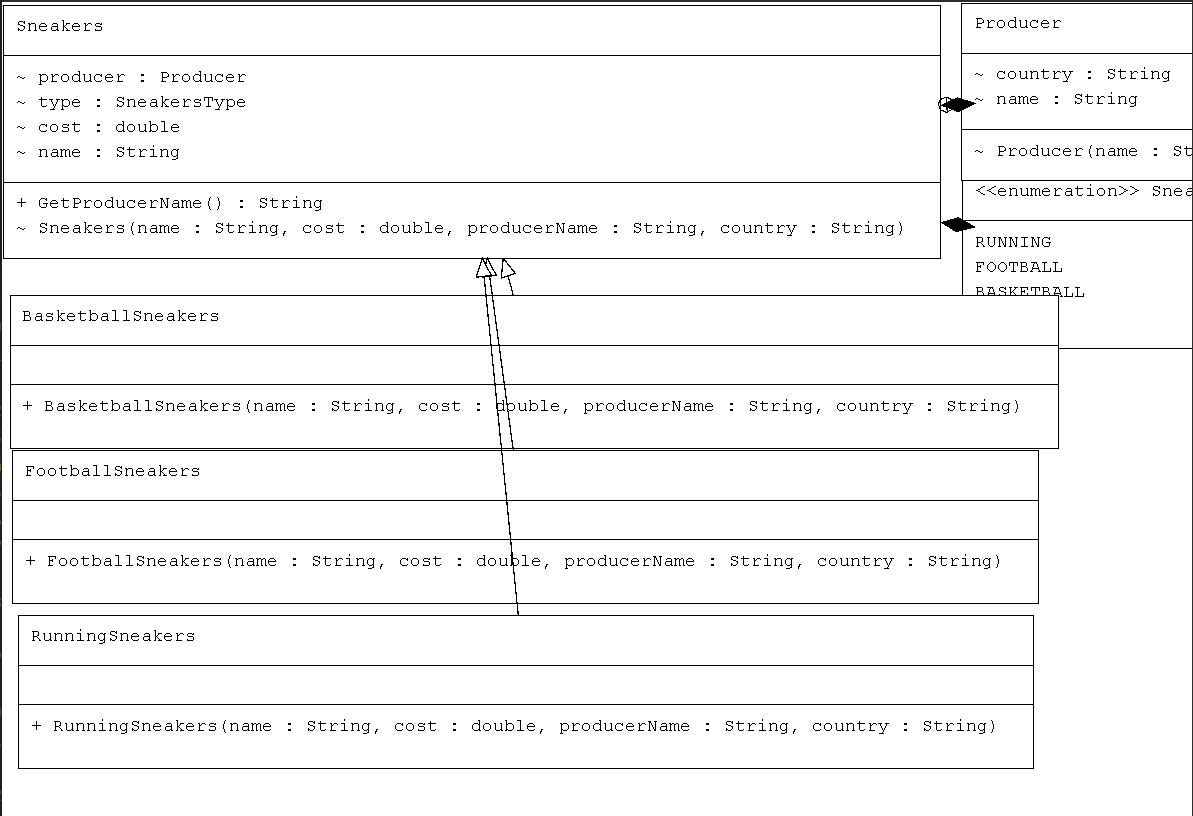


Рисунок 1 – Диаграмма классов

Реализуем эти классы на языке *Java*. В отдельном проекте создадим класс Main, в котором будет проводиться работа с элементами данных классов. Для того, чтобы работать с классами в другом проекте, необходимо создать .*jar* файл из библиотеки классов и подключить его ко второму проекту. Последовательность этих действий представлена на рисунках 2 – 6.

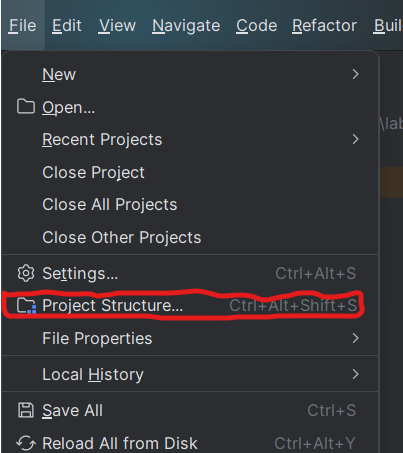


Рисунок 2 – Структура проекта

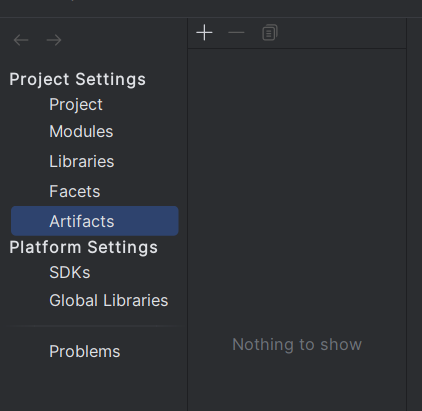


Рисунок 3 – Артефакты

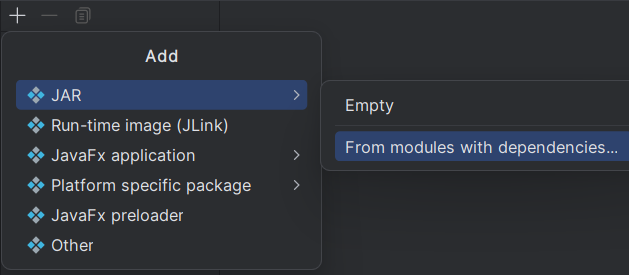


Рисунок 4 – Добавление *jar* файла

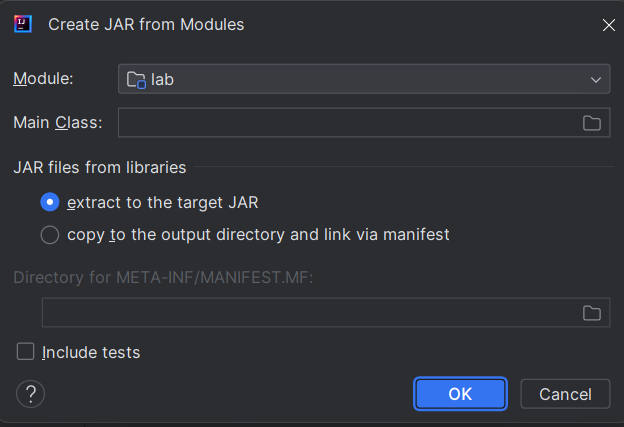


Рисунок 5 – Создание файла

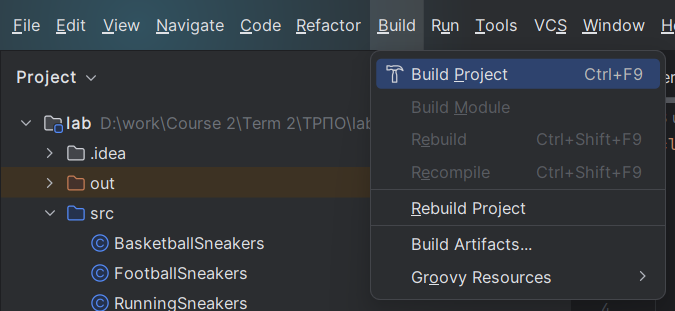


Рисунок 6 – Создание библиотеки классов

После создания файла библиотеки необходимо подключить его в проект с классом *Main*. Для этого в *Main*-проекте нужно зайти в раздел Структура проекта и выбрать раздел Библиотеки. После этого выбираем файл с библиотекой. Его путь представлен на рисунке 7.

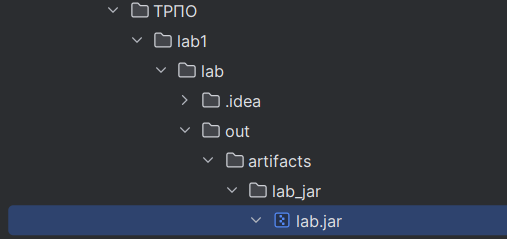


Рисунок 7 – Путь к библиотеке классов

После добавления становится возможным работать со всеми доступными классами из данной библиотеки. Пропишем в классе *Main* методы работы с массивом кроссовок. Сам массив представлен на рисунке 8.



Рисунок 8 – Массив кроссовок

На рисунке 9 показан результат выполнения всех методов.

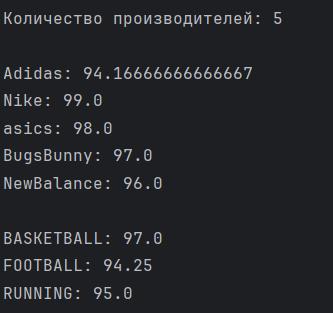


Рисунок 9 – Вывод результатов

В приложении А представлен код всех файлов.

**Выводы:** в результате выполнения данной лабораторной работы были изучены основы проектирования и создания программ при помощи объектно-ориентированного языка программирования *Java*.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Sneakers.java**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

class Sneakers {

String name;

double cost;

SneakersType type;

Producer producer;

/\*\*

\*

\* @param name Название кроссовок

\* @param cost Цена кроссовок

\* @param producerName Название производителя

\* @param country Страна производителя

\*/

Sneakers(String name, double cost, String producerName, String country) {

this.name = name;

this.cost = cost;

this.producer = new Producer(producerName, country);

}

/\*\*

\* @return Страна производителя кроссовок

\*/

public String GetProducerName()

{

return producer.name;

}

public static SneakersType[] GetTypeArray()

{

return SneakersType.values();

}

private class Producer {

String name;

String country;

/\*\*

\*

\* @param name Название бренда

\* @param country Страна производителя

\*/

Producer(String name, String country) {

this.name = name;

this.country = country;

}

}

}

**BasketballSneakers.java**

public class BasketballSneakers extends Sneakers{

public BasketballSneakers(String name, double cost, String producerName, String country){

super(name, cost, producerName, country);

type = SneakersType.BASKETBALL;

}

}

**FootballSneakers.java**

public class FootballSneakers extends Sneakers{

public FootballSneakers(String name, double cost, String producerName, String country){

super(name, cost, producerName, country);

type = SneakersType.FOOTBALL;

}

}

**RunningSneakers.java**

public class RunningSneakers extends Sneakers{

public RunningSneakers(String name, double cost, String producerName, String country){

super(name, cost, producerName, country);

type = SneakersType.RUNNING;

}

}

**SneakersType.java**

/\*\*

\* <li>Баскетбольные</li>

\* <li>Футбольные</li>

\* <li>Воллейбольные</li>

\*/

public enum SneakersType {

BASKETBALL,

FOOTBALL,

RUNNING

}

**Main.java**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or

// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.

public class Main {

public static void main(String[] args)

{

Sneakers[] sneakers = new Sneakers[]{

new BasketballSneakers("Мокасины",100,"Adidas","Belarus"),

new FootballSneakers("Туфли",99,"Nike","Belarus"),

new BasketballSneakers("Шлепки",98,"asics","Belarus"),

new RunningSneakers("Кеды",97,"BugsBunny","Belarus"),

new BasketballSneakers("Тапки",96,"NewBalance","Belarus"),

new FootballSneakers("Ботинки",95,"Adidas","Belarus"),

new BasketballSneakers("Шлепки",94,"Adidas","Belarus"),

new RunningSneakers("Кроссовки",93,"Adidas","Belarus"),

new FootballSneakers("Кеды",92,"Adidas","Belarus"),

new FootballSneakers("Мокасины",91,"Adidas","Belarus")

};

System.out.println("Количество производителей: "+ProducersQuantity(sneakers)+"\n");

double[] prodAvgs = AverageProducerSum(sneakers);

String[] producers = GetAllProducers(sneakers).toArray(new String[0]);

double[] typeAvgs = AverageTypeSum(sneakers);

SneakersType[] types = Sneakers.GetTypeArray();

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

System.out.println(producers[i]+": "+prodAvgs[i]);

}

System.out.println();

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

System.out.println(types[i]+": "+typeAvgs[i]);

}

}

/\*\*

\*

\* @param sneakers Массив кроссовок.

\* @return Список, содержащий названия всех производителей.

\*/

public static List<String> GetAllProducers(Sneakers[] sneakers)

{

List<String> producersList = new ArrayList<String>();

for( Sneakers sneaker : sneakers)

{

if(!producersList.contains(sneaker.GetProducerName()))

producersList.add(sneaker.GetProducerName());

}

return producersList;

}

/\*\*

\*

\* @param sneakers Массив кроссовок.

\* @return Количество производителей.

\*/

public static int ProducersQuantity(Sneakers[] sneakers)

{

return GetAllProducers(sneakers).size();

}

/\*\*

\*

\* @param sneakers Массив кроссовок.

\* @return Средняя цена у каждого производителя.

\*/

public static double[] AverageProducerSum(Sneakers[] sneakers)

{

List<String> producers = GetAllProducers(sneakers);

List<Double> avgs = new ArrayList<Double>();

for(String producer : producers)

{

avgs.add(Arrays.stream(sneakers).filter(sneaker -> sneaker.GetProducerName() == producer).mapToDouble(a->a.cost).average().orElse(0));

}

return avgs.stream().mapToDouble(i->i).toArray();

}

public static double[] AverageTypeSum(Sneakers[] sneakers)

{

SneakersType[] types = Sneakers.GetTypeArray();

List<Double> avgs = new ArrayList<Double>();

for(SneakersType type : types)

{

avgs.add(Arrays.stream(sneakers).filter(sneaker -> sneaker.type == type).mapToDouble(a->a.cost).average().orElse(0));

}

return avgs.stream().mapToDouble(i->i).toArray();

}

}