**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

# по дисциплине «Введение в разработку программного обеспечения»

на тему: «Развитие представлений о разработке программ.

Объектно-ориентированное программирование в *Java*»

Выполнил: студент гр. ИТП-21

Бондаренко И.А.

Принял: преподаватель-стажёр

Башаримов Ю.С.

Гомель 2024

Цель работы: изучение и анализ развития представлений о разработке программного обеспечения с акцентом на объектно-ориентированное программирование (ООП) на языке *Java.*

**Ход работы**

**Задание.** 1. Разработать *UML*-диаграмму иерархии классов.

2. При наименовании компонентов руководствоваться соглашением о наименовании (https://www.oracle.com/technetwork/java/codeconventions).

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 (<https://www.jetbrains.com/help/idea/working-with-code-documentation.html>).

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.

Условие к заданию 1 приведен на рисунке 1.

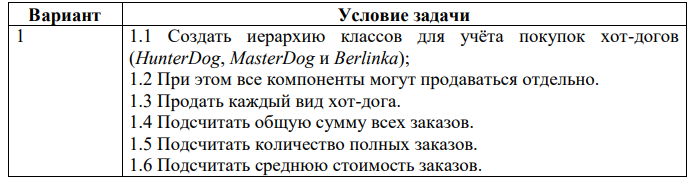


Рисунок 1 – Условие к заданию 1

Разработанная *UML*-диаграмма иерархии классов компонентов представлена на рисунке 2.

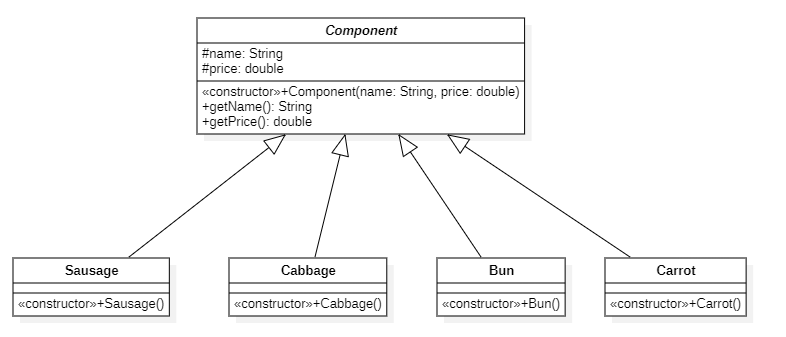


Рисунок 2 – *UML*-диаграмма иерархии классов компонентов

Разработанная *UML*-диаграмма иерархии классов хот-догов представлена на рисунке 3.

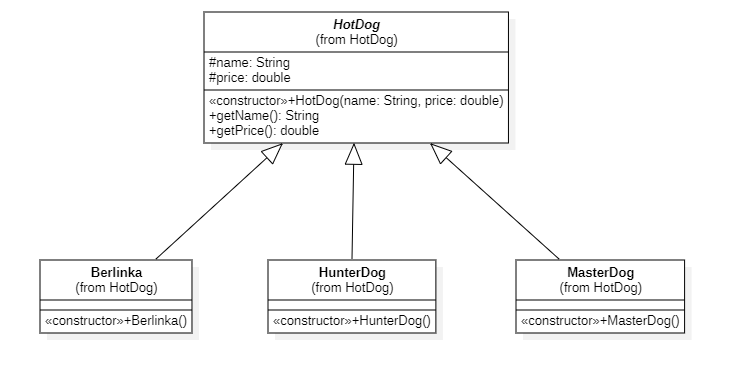


Рисунок 3 – *UML*-диаграмма иерархии классов хот-догов

Код программы находится в приложении А.

Результат размещения разработанной иерархии в *.jar* файл приведен на рисунке 4.



Рисунок 4 – Результат разработки *.jar* файла

Результат выполнения задания 1 представлен на рисунке 5.

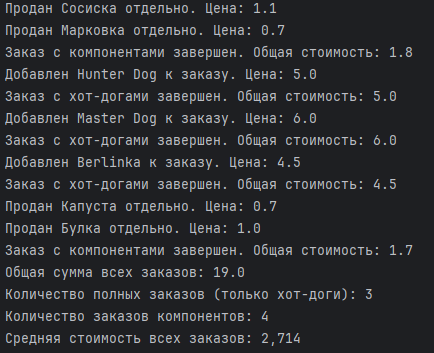


Рисунок 5 – Результат выполнения задания 1

Вывод: в ходе выполнения лабораторной работы мы изучили развитие представлений о разработке программного обеспечения с акцентом на объектно-ориентированное программирование (ООП) на языке *Java.* Таким образом, объектно-ориентированное программирование в *Java* не только продолжает оставаться актуальным, но и активно развивается, соответствуя требованиям современного программирования. Это делает его важным инструментом для разработчиков, стремящихся создавать эффективные и устойчивые программные решения.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

***Bun.java*:**

package by.bondar.component;  
  
*/\*\*  
 \* Класс для булки.  
 \*/*public class Bun extends Component {  
 public Bun() {  
 super("Булка", 1.0);  
 }  
}

***Cabbage.java*:**

package by.bondar.component;  
  
*/\*\*  
 \* Класс для капусты.  
 \*/*public class Cabbage extends Component {  
 public Cabbage() {  
 super("Капуста", 0.7);  
 }  
}

***Carrot.java*:**

package by.bondar.component;  
*/\*\*  
 \* Класс для морковки.  
 \*/*public class Carrot extends Component {  
 public Carrot() {  
 super("Марковка", 0.7);  
 }  
}

***Sausage.java*:**

package by.bondar.component;  
*/\*\*  
 \* Класс для сосиски.  
 \*/*public class Sausage extends Component {  
 public Sausage() {  
 super("Сосиска", 1.1);  
 }  
}

***Component.java*:**

package by.bondar.component;  
  
*/\*\*  
 \* Абстрактный класс для всех компонентов.  
 \*/*public abstract class Component {  
 protected String name;  
 protected double price;  
  
 public Component(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
}

***Berlinka.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Класс Berlinka, наследующий HotDog.  
 \*/*public class Berlinka extends HotDog {  
 public Berlinka() {  
 super("Berlinka", 4.5);  
 }  
}

***HunterDog.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Класс HunterDog, наследующий HotDog.  
 \*/*public class HunterDog extends HotDog {  
 public HunterDog() {  
 super("Hunter Dog", 5.0);  
 }  
}

***MasterDog.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Класс MasterDog, наследующий HotDog.  
 \*/*public class MasterDog extends HotDog {  
 public MasterDog() {  
 super("Master Dog", 6.0);  
 }  
}

***HotDog.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Абстрактный класс для всех видов хот-догов.  
 \*/*public abstract class HotDog {  
 protected String name;  
 protected double price;  
  
 public HotDog(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
}

***Main.java*:**

package by.bondar;  
  
import by.bondar.component.Bun;  
import by.bondar.component.Cabbage;  
import by.bondar.component.Carrot;  
import by.bondar.component.Sausage;  
import by.bondar.hotdog.Berlinka;  
import by.bondar.hotdog.HunterDog;  
import by.bondar.hotdog.MasterDog;  
  
import java.text.DecimalFormat;  
  
*/\*\*  
 \* Основной класс для демонстрации работы приложения.  
 \*/*public class Main {  
 public static void main(String[] args) {  
 // Создание заказа  
 Order order = new Order();  
  
 order.addComponent(new Sausage());  
 order.addComponent(new Carrot());  
 order.finalizeOrder();  
  
 order.addHotDog(new HunterDog());  
 order.finalizeOrder();  
  
 order.addHotDog(new MasterDog());  
 order.finalizeOrder();  
  
 order.addHotDog(new Berlinka());  
 order.finalizeOrder();  
  
 order.addComponent(new Cabbage());  
 order.addComponent(new Bun());  
 order.finalizeOrder();  
  
  
 DecimalFormat df = new DecimalFormat("#.###");  
  
 System.*out*.println("Общая сумма всех заказов: " + order.getTotalRevenue());  
 System.*out*.println("Количество полных заказов (только хот-доги): " + order.getTotalHotDogOrders());  
 System.*out*.println("Количество заказов компонентов: " + order.getTotalComponentOrders());  
 System.*out*.println("Средняя стоимость всех заказов: " + df.format(order.getAverageOrderValue()));  
 }  
}

***Order.java*:**

package by.bondar;  
  
import by.bondar.component.Component;  
import by.bondar.hotdog.HotDog;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс для учета заказа.  
 \*/*class Order {  
 private List<HotDog> hotDogs = new ArrayList<>();  
 private double hotDogTotalPrice = 0.0; // Общая стоимость только хот-догов  
 private double componentTotalPrice = 0.0; // Общая стоимость компонентов  
 private int totalHotDogOrders = 0; // Общее количество заказов с хот-догами  
 private int totalComponentOrders = 0; // Общее количество заказов с компонентами  
 private double totalRevenue = 0.0; // Общая сумма всех заказов  
  
 public void addHotDog(HotDog hotDog) {  
 hotDogs.add(hotDog);  
 hotDogTotalPrice += hotDog.getPrice();  
 System.*out*.println("Добавлен " + hotDog.getName() + " к заказу. Цена: " + hotDog.getPrice());  
 }  
  
 public void addComponent(Component component) {  
 componentTotalPrice += component.getPrice();  
 totalComponentOrders++; // Увеличиваем счетчик заказов компонентов  
 System.*out*.println("Продан " + component.getName() + " отдельно. Цена: " + component.getPrice());  
 }  
  
 public void finalizeOrder() {  
 if (!hotDogs.isEmpty()) {  
 totalHotDogOrders++;  
 totalRevenue += hotDogTotalPrice + componentTotalPrice; // Добавляем цену заказа в общую сумму  
 System.*out*.println("Заказ с хот-догами завершен. Общая стоимость: " + (hotDogTotalPrice + componentTotalPrice));  
 hotDogTotalPrice = 0.0; // Сбрасываем цену текущего заказа  
 componentTotalPrice = 0.0; // Сбрасываем цену компонентов  
 hotDogs.clear(); // Очищаем список хот-догов  
 } else if (componentTotalPrice > 0) {  
 totalRevenue += componentTotalPrice; // Добавляем только стоимость компонентов  
 System.*out*.println("Заказ с компонентами завершен. Общая стоимость: " + componentTotalPrice);  
 componentTotalPrice = 0.0; // Сбрасываем цену компонентов  
 } else {  
 System.*out*.println("Нет хот-догов или компонентов для завершения заказа.");  
 }  
 }  
 public int getTotalHotDogOrders() {  
 return totalHotDogOrders;  
 }  
  
 public int getTotalComponentOrders() {  
 return totalComponentOrders;  
 }  
  
 public double getTotalRevenue() {  
 return totalRevenue;  
 }  
 public double getAverageOrderValue() {  
 int totalOrders = totalHotDogOrders + totalComponentOrders;  
 return totalOrders == 0 ? 0 : totalRevenue / totalOrders;  
 }  
}