**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П.О.СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

### ЛАБОРАТОРНАЯ РАБОТА № 3

по дисциплине: «Введение в разработку программного обеспечения»

на тему: **«**Моделирование и алгоритмизация как средства проектирования программного обеспечения**»**

Выполнил: студент гр. ИТИ-21

Павлович Д. Н.

Принял преподаватель:

Карась О. В.

Гомель 2025

**Цель работы:** изучить использование моделирования и алгоритмизации как средства проектирования программного обеспечения в *Java.*

**Задание:**

* Разработать *UML*-диаграмму иерархии классов, согласно варианта (таблица 1).
* При наименовании компонентов руководствоваться соглашением о наименовании
* При описании иерархии использовать наследование и композицию.
* На основе *UML*-диаграммы разработать иерархию классов на языке *Java*.
* Весь код должен быть снабжен элементами документирования.
* Разработанную иерархию поместить в *jar* файл для дальнейшего использования в качестве библиотечных классов.
* Создать консольное приложение для демонстрации работы созданных классов.
* Составить отчет о проделанной работе.

Вариант 1:

1.1 Создать иерархию классов для учёта покупок хот-догов (*HunterDog*, *MasterDog* и *Berlinka*);

1.2 При этом все компоненты могут продаваться отдельно.

1.3 Продать каждый вид хот-дога.

1.4 Подсчитать общую сумму всех заказов.

1.5 Подсчитать количество полных заказов.

1.6 Подсчитать среднюю стоимость заказов.

**Ход работы:**

Проектирование иерархии классов

На основе анализа требований была спроектирована объектно-ориентированная система классов. Разработана *UML*-диаграмма (см. Рисунок 1, Рисунок 2), отображающая структуру наследования между классами. Создан абстрактный базовый класс *HotDog*, содержащий общие свойства всех хот-догов: название и цену. От этого класса унаследованы три конкретных реализации: *HunterDog*, *MasterDog* и *Berlinka*, каждая со своей фиксированной ценой.

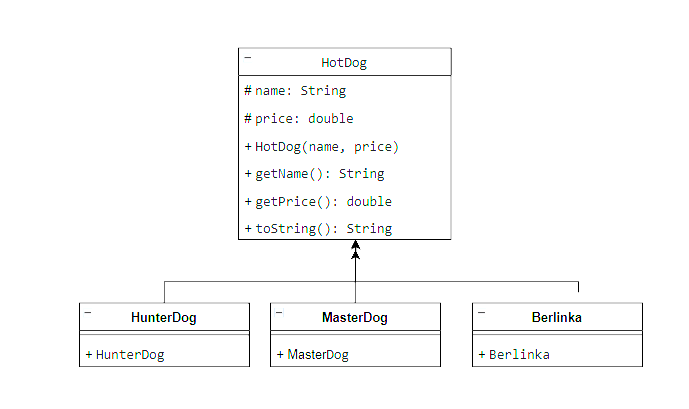


Рисунок 1 – *UML*-диаграмма

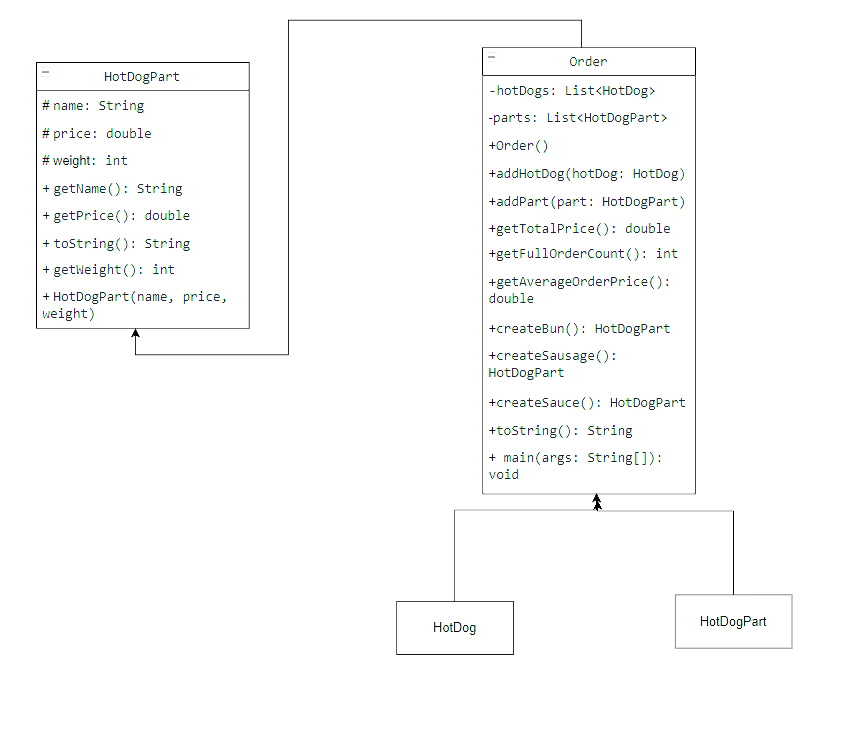


Рисунок 2 – *UML*-диаграмма

Реализация компонентов

Для реализации требования о продаже компонентов отдельно создан класс *HotDogPart*, который представляет отдельные части хот-дога: булку, сосиску и соус. На диаграмме классов показано, что *HotDogPart* является независимым классом, не связанным отношением наследования с основной иерархией хот-догов. Каждый компонент имеет название, цену и вес.

Создание системы заказов

Разработан класс *Order*, который управляет составом заказа. *UML*-диаграмма показывает, что класс *Order* содержит агрегацию классов *HotDog* и *HotDogPart*, что отражает возможность включения в заказ как целых хот-догов, так и отдельных компонентов. Класс предоставляет методы для добавления целых хот-догов и отдельных компонентов в заказ, а также методы для расчета общей стоимости, количества полных заказов и средней стоимости.

Реализация бизнес-логики

В классе *Order* созданы фабричные методы для удобного создания компонентов хот-дога. Реализованы методы для анализа заказов: вычисление общей суммы, подсчет количества целых хот-догов и расчет средней стоимости элементов заказа.

Тестирование системы

В методе *main* проведено тестирование созданной системы: добавлены все три вида хот-догов и несколько отдельных компонентов, после чего выведена полная информация о заказе и статистические данные.

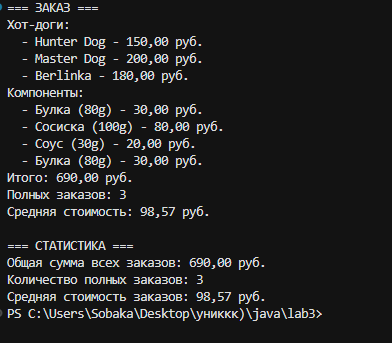


Рисунок 3 – Результат выполнения программы

Процесс создания *JAR*-файла

Подготовка исходных файлов

Все *Java*-файлы были размещены в директории *src*. Исходные коды классов включали основную иерархию хот-догов и систему заказов.

Компиляция *Java*-файлов

Выполнена компиляция исходных кодов с помощью команды *javac*. Все файлы были успешно скомпилированы без ошибок, что подтвердило корректность синтаксиса и отсутствие конфликтов между классами.

Создание *JAR*-архива

Для упаковки скомпилированных классов в *JAR*-файл использована команда *jar* *cf*. Был создан архив *hotdog*-*library*.*jar*, содержащий все необходимые классы и автоматически сгенерированный файл манифеста *META*-*INF*/*MANIFEST*.*MF*.

Проверка содержимого *JAR*-файла

С помощью команды *jar* *tf* выполнена проверка содержимого архива. В результате подтверждено наличие всех целевых классов: *Berlinka*.*class*, *HotDog*.*class*, *HotDogPart*.*class*, *HunterDog*.*class*, *MasterDog*.*class* и *Order*.*class*.

Результат

В результате успешно создан *JAR*-файл *hotdog*-*library*.*jar*, готовый к использованию в качестве библиотеки в других *Java*-проектах. Архив содержит полную иерархию классов для системы учета продаж хот-догов и может быть подключен к другим приложениям через *classpath*.

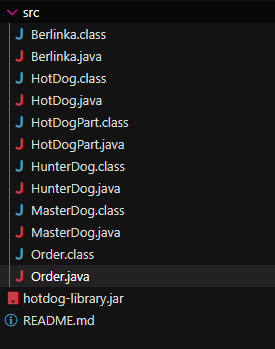


Рисунок 4 – Создание библиотеки классов

**Вывод**: в ходе лабораторной работы была успешно создана объектно-ориентированная система учета продаж хот-догов. Разработанная *UML*-диаграмма наглядно демонстрирует структуру системы, включая иерархию наследования классов хот-догов и отношения агрегации между классом заказа и другими классами системы. Реализованная архитектура позволяет учитывать как продажи целых хот-догов разных видов, так и отдельных их компонентов, а также автоматически рассчитывать ключевые финансовые показатели. Диаграмма классов подтверждает корректность проектных решений и обеспечивает наглядное представление о взаимосвязях между компонентами системы.

**ПРИЛОЖЕНИЕ А**

**Листинг файла Order.java:**

import java.util.ArrayList;

import java.util.List;

/\*\*

 \* Класс, представляющий заказ в системе учета покупок

 \* Содержит коллекции хот-догов и компонентов, а также методы для работы с заказом

 \*/

public class Order {

    // Список целых хот-догов в заказе (неизменяемая ссылка)

    private final List<HotDog> hotDogs;

    // Список отдельных компонентов в заказе (неизменяемая ссылка)

    private final List<HotDogPart> parts;

    /\*\*

     \* Конструктор заказа

     \* Инициализирует пустые списки для хот-догов и компонентов

     \*/

    public Order() {

        this.hotDogs = new ArrayList<>();

        this.parts = new ArrayList<>();

    }

    /\*\*

     \* Добавить хот-дог в заказ

     \* @param hotDog объект хот-дога для добавления

     \*/

    public void addHotDog(HotDog hotDog) {

        hotDogs.add(hotDog);

    }

    /\*\*

     \* Добавить компонент в заказ

     \* @param part объект компонента для добавления

     \*/

    public void addPart(HotDogPart part) {

        parts.add(part);

    }

    /\*\*

     \* Рассчитать общую стоимость заказа

     \* Суммирует цены всех хот-догов и компонентов

     \* @return общая стоимость заказа в рублях

     \*/

    public double getTotalPrice() {

        double total = 0;

        // Суммируем цены всех хот-догов

        for (HotDog hotDog : hotDogs) {

            total += hotDog.getPrice();

        }

        // Суммируем цены всех компонентов

        for (HotDogPart part : parts) {

            total += part.getPrice();

        }

        return total;

    }

    /\*\*

     \* Получить количество полных заказов (целых хот-догов)

     \* @return количество целых хот-догов в заказе

     \*/

    public int getFullOrderCount() {

        return hotDogs.size();

    }

    /\*\*

     \* Рассчитать среднюю стоимость одного элемента заказа

     \* Делит общую стоимость на общее количество элементов

     \* @return средняя стоимость элемента заказа в рублях

     \*/

    public double getAverageOrderPrice() {

        int totalItems = hotDogs.size() + parts.size();

        if (totalItems == 0) return 0; // Защита от деления на ноль

        return getTotalPrice() / totalItems;

    }

    // === Фабричные методы для создания компонентов ===

    /\*\*

     \* Создать объект булки для хот-дога

     \* @return объект булки с фиксированными параметрами

     \*/

    public HotDogPart createBun() {

        return new HotDogPart("Булка", 30.0, 80);

    }

    /\*\*

     \* Создать объект сосиски для хот-дога

     \* @return объект сосиски с фиксированными параметрами

     \*/

    public HotDogPart createSausage() {

        return new HotDogPart("Сосиска", 80.0, 100);

    }

    /\*\*

     \* Создать объект соуса для хот-дога

     \* @return объект соуса с фиксированными параметрами

     \*/

    public HotDogPart createSauce() {

        return new HotDogPart("Соус", 20.0, 30);

    }

    /\*\*

     \* Преобразование заказа в строку для вывода

     \* Форматирует информацию о заказе в читаемом виде

     \* @return строковое представление заказа

     \*/

    @Override

    public String toString() {

        StringBuilder sb = new StringBuilder();

        sb.append("=== ЗАКАЗ ===\n");

        // Вывод списка хот-догов, если они есть

        if (!hotDogs.isEmpty()) {

            sb.append("Хот-доги:\n");

            for (HotDog hotDog : hotDogs) {

                sb.append("  - ").append(hotDog).append("\n");

            }

        }

        // Вывод списка компонентов, если они есть

        if (!parts.isEmpty()) {

            sb.append("Компоненты:\n");

            for (HotDogPart part : parts) {

                sb.append("  - ").append(part).append("\n");

            }

        }

        // Вывод итоговой информации

        sb.append(String.format("Итого: %.2f руб.\n", getTotalPrice()));

        sb.append(String.format("Полных заказов: %d\n", getFullOrderCount()));

        sb.append(String.format("Средняя стоимость: %.2f руб.", getAverageOrderPrice()));

        return sb.toString();

    }

    /\*\*

     \* Главный метод демонстрационного приложения

     \* Создает заказ, добавляет товары и выводит результаты

     \* @param args аргументы командной строки (не используются)

     \*/

    public static void main(String[] args) {

        // Создаем новый заказ

        Order order = new Order();

        // Продаем каждый вид хот-дога (требование п.1.3)

        order.addHotDog(new HunterDog());

        order.addHotDog(new MasterDog());

        order.addHotDog(new Berlinka());

        // Продаем компоненты отдельно (требование п.1.2)

        order.addPart(order.createBun());

        order.addPart(order.createSausage());

        order.addPart(order.createSauce());

        order.addPart(order.createBun()); // Еще одна булка

        // Выводим полную информацию о заказе

        System.out.println(order);

        // Выводим отдельную статистику (требования п.1.4-1.6)

        System.out.println("\n=== СТАТИСТИКА ===");

        System.out.printf("Общая сумма всех заказов: %.2f руб.\n", order.getTotalPrice());

        System.out.printf("Количество полных заказов: %d\n", order.getFullOrderCount());

        System.out.printf("Средняя стоимость заказов: %.2f руб.\n", order.getAverageOrderPrice());

    }

}

**Листинг файла MasterDog.java:**

/\*\*

 \* Класс MasterDog

 \*/

public class MasterDog extends HotDog {

    public MasterDog() {

        super("Master Dog", 200.0);

    }

}

**Листинг файла HunterDog.java:**

/\*\*

 \* Конкретный класс хот-дога "Hunter Dog"

 \* Наследует свойства и методы от базового класса HotDog

 \*/

public class HunterDog extends HotDog {

    /\*\*

     \* Конструктор хот-дога Hunter Dog

     \* Устанавливает фиксированные название и цену

     \*/

    public HunterDog() {

        // Вызов конструктора родительского класса с конкретными значениями

        super("Hunter Dog", 150.0);

    }

}

**Листинг файла HotDogPart.java:**

/\*\*

 \* Класс, представляющий отдельный компонент хот-дога

 \* Может продаваться отдельно от целых хот-догов

 \*/

public class HotDogPart {

    // Название компонента

    protected String name;

    // Цена компонента в рублях

    protected double price;

    // Вес компонента в граммах

    protected int weight;

    /\*\*

     \* Конструктор компонента хот-дога

     \* @param name название компонента

     \* @param price цена компонента

     \* @param weight вес компонента в граммах

     \*/

    public HotDogPart(String name, double price, int weight) {

        this.name = name;

        this.price = price;

        this.weight = weight;

    }

    /\*\*

     \* Получить название компонента

     \* @return название компонента

     \*/

    public String getName() {

        return name;

    }

    /\*\*

     \* Получить цену компонента

     \* @return цена компонента

     \*/

    public double getPrice() {

        return price;

    }

    /\*\*

     \* Получить вес компонента

     \* @return вес компонента в граммах

     \*/

    public int getWeight() {

        return weight;

    }

    /\*\*

     \* Преобразование компонента в строку для вывода

     \* @return строковое представление компонента

     \*/

    @Override

    public String toString() {

        return String.format("%s (%dg) - %.2f руб.", name, weight, price);

    }

}

**Листинг файла HotDog.java:**

/\*\*

 \* Абстрактный класс, представляющий базовый хот-дог

 \* Содержит общие свойства и методы для всех видов хот-догов

 \*/

public abstract class HotDog {

    // Название хот-дога

    protected String name;

    // Цена хот-дога в рублях

    protected double price;

    /\*\*

     \* Конструктор хот-дога

     \* @param name название хот-дога

     \* @param price цена хот-дога

     \*/

    public HotDog(String name, double price) {

        this.name = name;

        this.price = price;

    }

    /\*\*

     \* Получить название хот-дога

     \* @return название хот-дога

     \*/

    public String getName() {

        return name;

    }

    /\*\*

     \* Получить цену хот-дога

     \* @return цена хот-дога

     \*/

    public double getPrice() {

        return price;

    }

    /\*\*

     \* Преобразование хот-дога в строку для вывода

     \* @return строковое представление хот-дога

     \*/

    @Override

    public String toString() {

        return String.format("%s - %.2f руб.", name, price);

    }

}

**Листинг файла Berlinka.java:**

/\*\*

 \* Класс Berlinka

 \*/

public class Berlinka extends HotDog {

    public Berlinka() {

        super("Berlinka", 180.0);

    }

}