**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

Здановская А. С.

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.
* При именовании компонентов руководствоваться соглашением о наименовании *Java*.
* При описании иерархии использовать наследование и композицию (в нашей диаграмме используется композиция).
* На основе *UML*-диаграммы разработать иерархию классов на языке *Java*.
* Весь код должен быть снабжен элементами документирования (*Javadoc*).
* Разработанную иерархию поместить в .*jar* файл.
* Создать консольное приложение для демонстрации работы.

Вариант 7.

* 1.1 Создать иерархию классов для учёта самолетов в аэропорту.
* 1.2 Создать 3 класса самолетов и минимум 3 экземпляра.
* 1.3 Классы должны содержать поля: количество пассажиров,
* количество топлива, название рейса.
* 1.4 Рассчитать расход топлива.

**Ход работы:**

На первом этапе работы была разработана *UML*-диаграмма иерархии классов, отображающая структуру будущей системы. Диаграмма включала три основных класса: *Car*, *Driver* и *Engine*. Класс *Car* представляет автомобиль и содержит такие атрибуты как марка, класс автомобиля и вес. Класс *Driver* описывает водителя с полями для хранения ФИО и стажа вождения. Класс *Engine* моделирует двигатель автомобиля с характеристиками мощности и производителя.

*UML*-диаграмма иерархии классов представлена на рисунке 1.

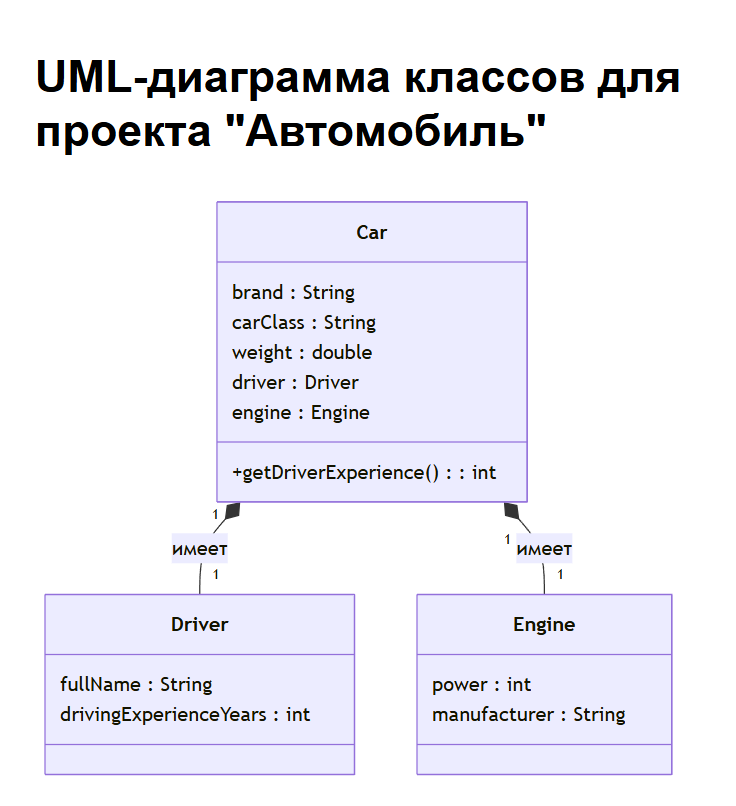


Рисунок 1 – *UML*-диаграмма иерархии классов.

Реализация системы выполнена на языке *Java* с строгим соблюдением соглашений о наименовании Oracle Code Conventions.

Класс Aircraft является абстрактным базовым классом, который определяет общие характеристики и поведение для всех типов самолетов. Он инкапсулирует основные поля: номер рейса, количество пассажиров, объем топлива и норму расхода топлива. Ключевыми методами класса являются calculateFuelConsumption() для расчета расхода топлива на заданное расстояние и calculateMaxRange() для определения максимальной дальности полета. Конструкторы класса позволяют создавать объекты с инициализацией всех необходимых параметров.

Класс PassengerAircraft расширяет базовый класс Aircraft и специализируется для пассажирских авиаперевозок. Он добавляет поле класса обслуживания (эконом, бизнес, первый) и предоставляет соответствующие методы доступа. Наследование от Aircraft обеспечивает соблюдение принципа подстановки Барбары Лисков, позволяя использовать объекты PassengerAircraft везде, где ожидается Aircraft.

Класс CargoAircraft представляет грузовые самолеты и также наследует от Aircraft. Особенностью класса является переопределение метода calculateFuelConsumption(), который учитывает вес груза при расчете расхода топлива. Это демонстрирует принцип полиморфизма - каждый тип самолета может иметь собственную реализацию расчета расхода.

Конкретные классы самолетов (Boeing737, AirbusA320, AntonovAn124) наследуются от соответствующих абстрактных классов и задают специфические параметры для каждой модели. Каждый класс определяет характерный для данной модели расход топлива через константы DEFAULT\_FUEL\_CONSUMPTION, что обеспечивает правильную инициализацию объектов.

Класс Airport реализует композицию для управления коллекцией самолетов. Он содержит список объектов Aircraft и предоставляет методы для добавления, удаления и поиска самолетов по номеру рейса. Важным методом является calculateTotalFuelConsumption(), который вычисляет общий расход топлива для всех самолетов в аэропорту на заданное расстояние.

Для демонстрации функциональности системы было разработано консольное приложение AirportMenu с интерактивным текстовым интерфейсом. Приложение реализует полный цикл управления аэропортом, включая просмотр всех самолетов, добавление новых воздушных судов различных типов, удаление самолетов из системы, поиск по номеру рейса и расчет расходов топлива.

Результат выполнения программы представлен на рисунке 2.

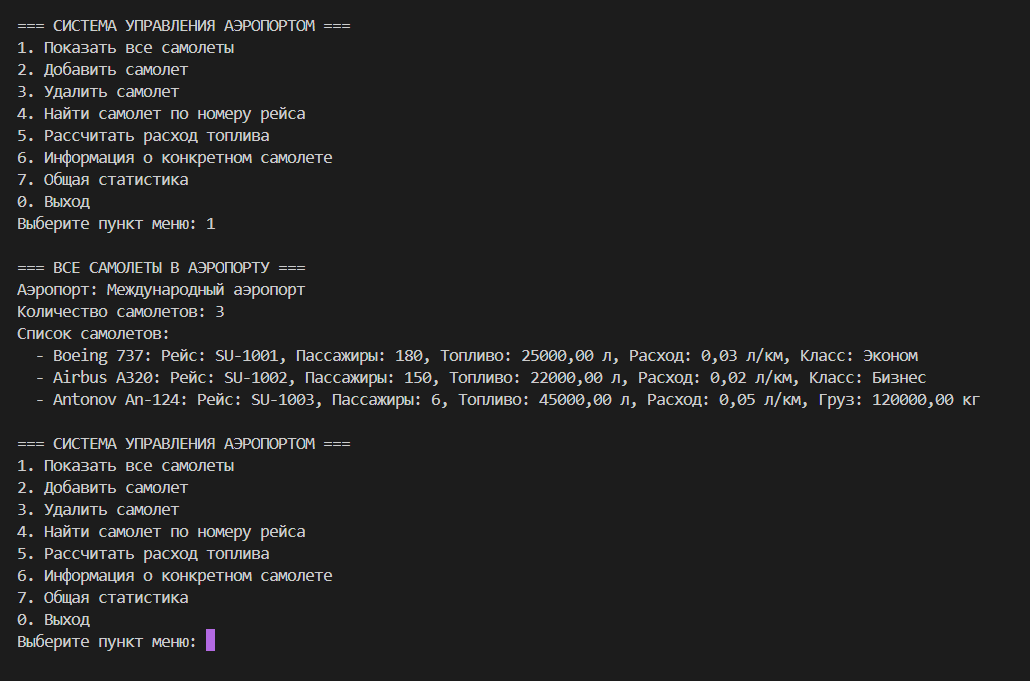


Рисунок 2 – Результат выполнения программы.

Все разработанные классы были упакованы в *JAR*-файл для использования в качестве библиотеки. Это позволяет использовать созданную иерархию классов в других проектах без необходимости копирования исходного кода. Процесс сборки включает компиляцию *Java*-файлов и создание архива с соблюдением структуры пакетов.

**Вывод**: реализованная система демонстрирует все принципы объектно-ориентированного программирования: инкапсуляцию данных в классах, наследование иерархии самолетов, полиморфизм в расчетах расхода топлива и композицию в классе Airport. Архитектура системы позволяет легко расширять функциональность добавлением новых типов самолетов без изменения существующего кода.

**ПРИЛОЖЕНИЕ А**

**Листинг файла AirportDemo*.java:***

/\*\*

 \* Демонстрационное приложение для тестирования иерархии классов самолетов.

 \*/

public class AirportDemo {

    public static void main(String[] args) {

        // Создаем аэропорт

        Airport airport = new Airport("Шереметьево");

        // Создаем 3+ экземпляра самолетов

        Boeing737 boeing1 = new Boeing737("SU-1001", 180, 25000, "Эконом");

        AirbusA320 airbus1 = new AirbusA320("SU-1002", 150, 22000, "Бизнес");

        AntonovAn124 antonov1 = new AntonovAn124("SU-1003", 6, 45000, 120000);

        Boeing737 boeing2 = new Boeing737("SU-1004", 160, 24000, "Первый");

        // Добавляем самолеты в аэропорт

        airport.addAircraft(boeing1);

        airport.addAircraft(airbus1);

        airport.addAircraft(antonov1);

        airport.addAircraft(boeing2);

        // Демонстрация работы

        System.out.println("=== ДЕМОНСТРАЦИЯ РАБОТЫ СИСТЕМЫ АЭРОПОРТА ===\n");

        // Отображаем все самолеты

        airport.displayAllAircraft();

        // Демонстрация расчета расхода топлива

        System.out.println("\n=== РАСЧЕТ РАСХОДА ТОПЛИВА ===");

        double distance = 1000; // 1000 км

        for (Aircraft aircraft : airport.getAircraftList()) {

            double fuelConsumption = aircraft.calculateFuelConsumption(distance);

            double maxRange = aircraft.calculateMaxRange();

            System.out.printf("%s - Расход на %d км: %.2f л, Макс. дальность: %.2f км%n",

                    aircraft.getFlightNumber(), (int)distance, fuelConsumption, maxRange);

        }

        // Общий расход топлива

        double totalConsumption = airport.calculateTotalFuelConsumption(distance);

        System.out.printf("\nОбщий расход топлива для всех самолетов на %d км: %.2f л%n",

                (int)distance, totalConsumption);

        // Поиск самолета

        System.out.println("\n=== ПОИСК САМОЛЕТА ===");

        String searchFlight = "SU-1002";

        Aircraft foundAircraft = airport.findAircraft(searchFlight);

        if (foundAircraft != null) {

            System.out.println("Найден самолет: " + foundAircraft);

        } else {

            System.out.println("Самолет с номером " + searchFlight + " не найден");

        }

        // Удаление самолета

        System.out.println("\n=== УДАЛЕНИЕ САМОЛЕТА ===");

        boolean removed = airport.removeAircraft("SU-1001");

        System.out.println("Самолет SU-1001 " + (removed ? "удален" : "не найден"));

        airport.displayAllAircraft();

    }

}

import java.util.Scanner;

import java.util.List;

/\*\*

 \* Консольное меню для управления аэропортом

 \*/

public class AirportMenu {

    private Airport airport;

    private Scanner scanner;

    public AirportMenu() {

        this.airport = new Airport("Международный аэропорт");

        this.scanner = new Scanner(System.in);

        // Добавляем несколько самолетов для демонстрации

        initializeDemoData();

    }

    /\*\*

     \* Инициализация демонстрационных данных

     \*/

    private void initializeDemoData() {

        airport.addAircraft(new Boeing737("SU-1001", 180, 25000, "Эконом"));

        airport.addAircraft(new AirbusA320("SU-1002", 150, 22000, "Бизнес"));

        airport.addAircraft(new AntonovAn124("SU-1003", 6, 45000, 120000));

    }

    /\*\*

     \* Главное меню

     \*/

    public void showMainMenu() {

        while (true) {

            System.out.println("\n=== СИСТЕМА УПРАВЛЕНИЯ АЭРОПОРТОМ ===");

            System.out.println("1. Показать все самолеты");

            System.out.println("2. Добавить самолет");

            System.out.println("3. Удалить самолет");

            System.out.println("4. Найти самолет по номеру рейса");

            System.out.println("5. Рассчитать расход топлива");

            System.out.println("6. Информация о конкретном самолете");

            System.out.println("7. Общая статистика");

            System.out.println("0. Выход");

            System.out.print("Выберите пункт меню: ");

            int choice = getIntInput();

            switch (choice) {

                case 1:

                    showAllAircraft();

                    break;

                case 2:

                    addAircraftMenu();

                    break;

                case 3:

                    removeAircraftMenu();

                    break;

                case 4:

                    findAircraftMenu();

                    break;

                case 5:

                    calculateFuelMenu();

                    break;

                case 6:

                    showAircraftDetails();

                    break;

                case 7:

                    showStatistics();

                    break;

                case 0:

                    System.out.println("Выход из программы...");

                    return;

                default:

                    System.out.println("Неверный выбор! Попробуйте снова.");

            }

        }

    }

    /\*\*

     \* Показать все самолеты

     \*/

    private void showAllAircraft() {

        System.out.println("\n=== ВСЕ САМОЛЕТЫ В АЭРОПОРТУ ===");

        airport.displayAllAircraft();

    }

    /\*\*

     \* Меню добавления самолета

     \*/

    private void addAircraftMenu() {

        System.out.println("\n=== ДОБАВЛЕНИЕ НОВОГО САМОЛЕТА ===");

        System.out.println("1. Boeing 737 (пассажирский)");

        System.out.println("2. Airbus A320 (пассажирский)");

        System.out.println("3. Antonov An-124 (грузовой)");

        System.out.print("Выберите тип самолета: ");

        int typeChoice = getIntInput();

        System.out.print("Введите номер рейса: ");

        String flightNumber = scanner.nextLine();

        System.out.print("Введите количество пассажиров: ");

        int passengers = getIntInput();

        System.out.print("Введите количество топлива (л): ");

        double fuel = getDoubleInput();

        Aircraft newAircraft = null;

        switch (typeChoice) {

            case 1:

                System.out.print("Введите класс обслуживания: ");

                String seatClass1 = scanner.nextLine();

                newAircraft = new Boeing737(flightNumber, passengers, fuel, seatClass1);

                break;

            case 2:

                System.out.print("Введите класс обслуживания: ");

                String seatClass2 = scanner.nextLine();

                newAircraft = new AirbusA320(flightNumber, passengers, fuel, seatClass2);

                break;

            case 3:

                System.out.print("Введите вес груза (кг): ");

                double cargoWeight = getDoubleInput();

                newAircraft = new AntonovAn124(flightNumber, passengers, fuel, cargoWeight);

                break;

            default:

                System.out.println("Неверный выбор типа самолета!");

                return;

        }

        if (newAircraft != null) {

            airport.addAircraft(newAircraft);

            System.out.println("Самолет успешно добавлен!");

        }

    }

    /\*\*

     \* Меню удаления самолета

     \*/

    private void removeAircraftMenu() {

        System.out.println("\n=== УДАЛЕНИЕ САМОЛЕТА ===");

        System.out.print("Введите номер рейса для удаления: ");

        String flightNumber = scanner.nextLine();

        boolean removed = airport.removeAircraft(flightNumber);

        if (removed) {

            System.out.println("Самолет с номером " + flightNumber + " успешно удален!");

        } else {

            System.out.println("Самолет с номером " + flightNumber + " не найден!");

        }

    }

    /\*\*

     \* Меню поиска самолета

     \*/

    private void findAircraftMenu() {

        System.out.println("\n=== ПОИСК САМОЛЕТА ===");

        System.out.print("Введите номер рейса: ");

        String flightNumber = scanner.nextLine();

        Aircraft aircraft = airport.findAircraft(flightNumber);

        if (aircraft != null) {

            System.out.println("Найден самолет:");

            System.out.println(aircraft);

            // Дополнительная информация в зависимости от типа

            if (aircraft instanceof PassengerAircraft) {

                PassengerAircraft pa = (PassengerAircraft) aircraft;

                System.out.println("Класс обслуживания: " + pa.getSeatClass());

            } else if (aircraft instanceof CargoAircraft) {

                CargoAircraft ca = (CargoAircraft) aircraft;

                System.out.println("Вес груза: " + ca.getCargoWeight() + " кг");

            }

        } else {

            System.out.println("Самолет с номером " + flightNumber + " не найден!");

        }

    }

    /\*\*

     \* Меню расчета расхода топлива

     \*/

    private void calculateFuelMenu() {

        System.out.println("\n=== РАСЧЕТ РАСХОДА ТОПЛИВА ===");

        System.out.print("Введите расстояние (км): ");

        double distance = getDoubleInput();

        System.out.println("\nРезультаты расчета:");

        System.out.println("Расстояние: " + distance + " км");

        System.out.println("----------------------------------------");

        List<Aircraft> aircraftList = airport.getAircraftList();

        double totalConsumption = 0;

        for (Aircraft aircraft : aircraftList) {

            double consumption = aircraft.calculateFuelConsumption(distance);

            double maxRange = aircraft.calculateMaxRange();

            totalConsumption += consumption;

            System.out.printf("%s:%n", aircraft.getFlightNumber());

            System.out.printf("  Расход топлива: %.2f л%n", consumption);

            System.out.printf("  Макс. дальность: %.2f км%n", maxRange);

            System.out.println();

        }

        System.out.printf("ОБЩИЙ РАСХОД ТОПЛИВА: %.2f л%n", totalConsumption);

    }

    /\*\*

     \* Детальная информация о самолете

     \*/

    private void showAircraftDetails() {

        System.out.println("\n=== ИНФОРМАЦИЯ О САМОЛЕТЕ ===");

        System.out.print("Введите номер рейса: ");

        String flightNumber = scanner.nextLine();

        Aircraft aircraft = airport.findAircraft(flightNumber);

        if (aircraft != null) {

            System.out.println("\nДЕТАЛЬНАЯ ИНФОРМАЦИЯ:");

            System.out.println("Тип: " + aircraft.getClass().getSimpleName());

            System.out.println("Номер рейса: " + aircraft.getFlightNumber());

            System.out.println("Количество пассажиров: " + aircraft.getPassengerCount());

            System.out.println("Топливо: " + aircraft.getFuelAmount() + " л");

            System.out.println("Расход топлива: " + aircraft.getFuelConsumptionRate() + " л/км");

            System.out.println("Макс. дальность: " + aircraft.calculateMaxRange() + " км");

            if (aircraft instanceof PassengerAircraft) {

                PassengerAircraft pa = (PassengerAircraft) aircraft;

                System.out.println("Класс обслуживания: " + pa.getSeatClass());

            } else if (aircraft instanceof CargoAircraft) {

                CargoAircraft ca = (CargoAircraft) aircraft;

                System.out.println("Вес груза: " + ca.getCargoWeight() + " кг");

            }

        } else {

            System.out.println("Самолет не найден!");

        }

    }

    /\*\*

     \* Общая статистика аэропорта

     \*/

    private void showStatistics() {

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

        List<Aircraft> aircraftList = airport.getAircraftList();

        int totalPassengers = 0;

        double totalFuel = 0;

        int passengerAircraftCount = 0;

        int cargoAircraftCount = 0;

        for (Aircraft aircraft : aircraftList) {

            totalPassengers += aircraft.getPassengerCount();

            totalFuel += aircraft.getFuelAmount();

            if (aircraft instanceof PassengerAircraft) {

                passengerAircraftCount++;

            } else if (aircraft instanceof CargoAircraft) {

                cargoAircraftCount++;

            }

        }

        System.out.println("Общее количество самолетов: " + aircraftList.size());

        System.out.println("Пассажирских самолетов: " + passengerAircraftCount);

        System.out.println("Грузовых самолетов: " + cargoAircraftCount);

        System.out.println("Общее количество пассажиров: " + totalPassengers);

        System.out.println("Общее количество топлива: " + totalFuel + " л");

        System.out.printf("Среднее количество пассажиров на самолет: %.1f%n",

                         (double) totalPassengers / aircraftList.size());

    }

    /\*\*

     \* Ввод целого числа с проверкой

     \*/

    private int getIntInput() {

        while (true) {

            try {

                return Integer.parseInt(scanner.nextLine());

            } catch (NumberFormatException e) {

                System.out.print("Ошибка! Введите целое число: ");

            }

        }

    }

    /\*\*

     \* Ввод дробного числа с проверкой

     \*/

    private double getDoubleInput() {

        while (true) {

            try {

                return Double.parseDouble(scanner.nextLine());

            } catch (NumberFormatException e) {

                System.out.print("Ошибка! Введите число: ");

            }

        }

    }

    /\*\*

     \* Главный метод программы

     \*/

    public static void main(String[] args) {

        AirportMenu menu = new AirportMenu();

        menu.showMainMenu();

    }

}

/\*\*

 \* Базовый класс для всех типов самолетов.

 \* Содержит общие характеристики и методы для всех самолетов.

 \*/

public abstract class Aircraft {

    private final String flightNumber;

    private int passengerCount;

    private double fuelAmount;

    private final double fuelConsumptionRate;

    /\*\*

     \* Конструктор базового класса Aircraft.

     \*

     \* @param flightNumber номер рейса

     \* @param passengerCount количество пассажиров

     \* @param fuelAmount количество топлива

     \* @param fuelConsumptionRate расход топлива

     \*/

    public Aircraft(String flightNumber, int passengerCount, double fuelAmount, double fuelConsumptionRate) {

        this.flightNumber = flightNumber;

        this.passengerCount = passengerCount;

        this.fuelAmount = fuelAmount;

        this.fuelConsumptionRate = fuelConsumptionRate;

    }

    /\*\*

     \* Рассчитывает расход топлива на основе пройденного расстояния.

     \*

     \* @param distance расстояние в километрах

     \* @return расход топлива

     \*/

    public double calculateFuelConsumption(double distance) {

        return distance \* fuelConsumptionRate;

    }

    /\*\*

     \* Рассчитывает максимальную дальность полета.

     \*

     \* @return максимальная дальность в километрах

     \*/

    public double calculateMaxRange() {

        return fuelAmount / fuelConsumptionRate;

    }

    // Геттеры

    public String getFlightNumber() {

        return flightNumber;

    }

    public int getPassengerCount() {

        return passengerCount;

    }

    public double getFuelAmount() {

        return fuelAmount;

    }

    public double getFuelConsumptionRate() {

        return fuelConsumptionRate;

    }

    // Сеттеры

    public void setPassengerCount(int passengerCount) {

        this.passengerCount = passengerCount;

    }

    public void setFuelAmount(double fuelAmount) {

        this.fuelAmount = fuelAmount;

    }

    /\*\*

     \* Возвращает строковое представление объекта Aircraft.

     \*

     \* @return строковое представление

     \*/

    @Override

    public String toString() {

        return String.format("Рейс: %s, Пассажиры: %d, Топливо: %.2f л, Расход: %.2f л/км",

                flightNumber, passengerCount, fuelAmount, fuelConsumptionRate);

    }

}