Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «Современные платформы программирования»

Выполнил:
студент 3 курса
группы ПО-8
Сорока В.С.

Проверил:
Крощенко А.А.

Брест, 2024

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования

Задание 1 (Вариант 4)
Реализовать указанный класс, включив в него вспомогательный внутренний класс или классы.

Реализовать 2-3 метода (на выбор). Продемонстрировать использование реализованных классов.

Создать класс Зачетная Книжка с внутренним классом, с помощью объектов которого можно хранить информацию о сессиях, зачетах, экзаменах.

Спецификации ввода-вывода программы:
1. Класс ZachetnayaKnizhka:
   - Содержит внутренний класс Session для управления данными о сессии студента.
2. Класс Session:
   - Содержит поля для названия предмета, списков зачетов и экзаменов, а также списков оценок зачетов и экзаменов.
   - Метод Session(String name) создает новую сессию с указанным названием предмета.
   - Методы addZacheti(String discipline, int grade) и addEkzameny(String discipline, int grade) добавляют соответственно зачеты и экзамены с их оценками.
   - Метод creditGrade(int grade) возвращает строку, указывающую, зачтен ли предмет на основе оценки.
   - Метод averageGrade() вычисляет и возвращает средний балл за сессию.
   - Метод printSessionResult() выводит результаты сессии на экран, включая название предмета, оценки за зачеты и экзамены, и средний балл.
3. Использование классов:
   - Создается объект ZachetnayaKnizhka, затем создается объект Session с указанием названия семестра.
   - Для сессии добавляются зачеты и экзамены с их оценками при помощи методов addZacheti и addEkzameny.
   - Вызывается метод printSessionResult() для вывода результатов сессии на экран.

Текст программы

```
public class ZachetnayaKnizhka {
    private class Session
    {
        private String nazvaniePredmeta;
        private List<String> zachety;
        private List<String> ekzameny;
        private List<Integer> ocenkiZachet;
        private List<Integer> ocenkiEkzamen;
```

```
        public Session(String name) {
            this.nazvaniePredmeta = name;
            zachety = new ArrayList<>();
            ekzameny = new ArrayList<>();
            ocenkiZachet = new ArrayList<>();
            ocenkiEkzamen = new ArrayList<>();
        }

        public void addEkzameny(String discipline, int grade) {
            ekzameny.add(discipline);
            ocenkiEkzamen.add(grade);
        }

        public void addZacheti(String discipline, int grade) {
            zachety.add(discipline);
            ocenkiZachet.add(grade);
        }

        public String creditGrade(int grade) {
            if (grade >= 4) {
                return "зачтено";
            } else {
                return "не зачтено";
            }
        }

        public double averageGrade() {
            double sum = 0;
            int count = 0;

            for (int grade : ocenkiZachet) {
                sum += grade;
                count++;
            }

            for (int grade : ocenkiEkzamen) {
                sum += grade;
                count++;
            }

            if (count == 0) {
                return 0;
            }
            return Math.round((sum / count) * 10) / 10.0;
        }

        public void printSessionResult() {
            StringBuilder result = new StringBuilder();
            result.append("\nСессия:
").append(nazvaniePredmeta).append("\n");
            System.out.print(result.toString());
            System.out.println("");
            System.out.println("Зачёты:");
            System.out.println("_______");
            for (String zachet : zachety) {
                result.setLength(0);
                result.append("Дисциплина: ").append(zachet).append("\n");
                int index = zachety.indexOf(zachet);
                int ocenka = ocenkiZachet.get(index);
                result.append("Оценка:
").append(creditGrade(ocenka)).append(" (").append(ocenka).append(")");
                System.out.println(result.toString());
            }
```

```
            System.out.println("\nЭкзамены:");
            System.out.println("_________");
            for (String ekzamen : ekzameny) {
                result.setLength(0);
                result.append("Дисциплина: ").append(ekzamen).append("\n");
                int index = ekzameny.indexOf(ekzamen);
                int ocenka = ocenkiEkzamen.get(index);
                if (ocenka == -1) {
                    result.append("Оценка: " + "оценка отсутствует");
                } else {
                    result.append("Оценка: " + ocenka);
                }
                System.out.println(result.toString());
            }
            System.out.println(" ");
            double average = averageGrade();
            System.out.println("Средний балл: " + average);
        }
    }
```

```
ZachetnayaKnizhka zahtnayaKnizhka = new ZachetnayaKnizhka();
Session session = zahtnayaKnizhka.new Session("№6, семестр №6 (лето 2024):");

session.addZacheti("Современные платформы программирования (СПП)", 7);
session.addZacheti("Проектирование интернет-систем (ПИС)", 6);
session.addZacheti("Надежность программного обеспечения (НПО)", 6);
session.addZacheti("Физическая культура", 8);
session.addZacheti("Технологическая практика", 6);

session.addEkzameny("Программное обеспечение мобильной робототехники (ПОМР)",
7);
session.addEkzameny("Разработка программного обеспечения для мобильных
платформ (РПОМП)", 8);
session.addEkzameny("Базы данных (БД)", 6);
session.addEkzameny("Основы бизнеса и права в сфере ИКТ", 7);

session.printSessionResult();
```

Пример работы программы

```
================================================================================
Сессия: №6, семестр №6 (лето 2024):
--------------------------------------------------------------------------------
Зачёты:

-------
Дисциплина: Современные платформы программирования (СПП)
Оценка: зачтено (7)
Дисциплина: Проектирование интернет-систем (ПИС)
Оценка: зачтено (6)
Дисциплина: Надежность программного обеспечения (НПО)
Оценка: зачтено (6)
Дисциплина: Физическая культура
Оценка: зачтено (8)
Дисциплина: Технологическая практика
Оценка: зачтено (6)
```

```
Экзамены:

---------
Дисциплина: Программное обеспечение мобильной робототехники (ПОМР)
Оценка: 7
Дисциплина: Разработка программного обеспечения для мобильных платформ (РПОМП)
Оценка: 8
Дисциплина: Базы данных (БД)
Оценка: 6
Дисциплина: Основы бизнеса и права в сфере ИКТ
Оценка: 7
--------------------------------------------------------------------------------
Средний балл: 6.8
================================================================================
```

Задание 2 (Вариант 10)
Реализовать агрегирование. При создании класса агрегируемый класс объявляется как атрибут
(локальная переменная, параметр метода). Включить в каждый класс 2-3 метода на выбор. Продемонстрировать использование разработанных классов.

Создать класс Планета, используя класс Материк.

Спецификации ввода-вывода программы:
1. Класс Materick:
   - Создает объект, представляющий материк.
   - Метод description() выводит информацию о материке.
   - Метод populationDensity(double area, int population) вычисляет и выводит плотность населения материка.
   - Метод economySize(double gdp) выводит размер экономики материка.
2. Класс Planet:
   - Создает объект, представляющий планету с списком материков.
   - Метод description() выводит информацию о планете и ее материках.
   - Метод launchRocket() выводит сообщение о запуске ракеты с планеты.
   - Метод explorePlanet() выводит сообщение о начале исследования планеты.
   - Метод launchSatellite() выводит сообщение о запуске спутника с планеты.
   - Метод weatherForecast(String date) выводит прогноз погоды на указанную дату.
3. Вывод информации:
   - Создание объекта Planet Earth с списком материков.
   - Вывод описания планеты и ее материков.
   - Вызов методов для запуска ракеты, исследования планеты, запуска спутника и вывода прогноза погоды.

- Вызов методов populationDensity() и economySize() для каждого материка с соответствующими значениями параметров.

4. Исследование планеты:
   - Программа проводит исследование планеты Земля, выводя информацию о её материках, запуске ракеты и спутника, а также прогнозе погоды.
   - Для каждого материка выводится информация о плотности населения и размере экономики.

Текст программы

```
static class Materick
{
    private String name;
    public Materick(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void description() {
        System.out.println("Это " + name + " - один из материков планеты.");
    }
    public void populationDensity(double area, int population) {
        double density = population / area;
        double roundedDensity = Math.round(density * 10.0) / 10.0;
        System.out.println("Плотность населения " + name +" составляет " +
roundedDensity +" человек на квадратную единицу.");
    }
    public void economySize(double gdp) {
        double roundedGdp = Math.round(gdp * 10.0) / 10.0;
        System.out.println("Размер экономики " + name + " составляет " +
roundedGdp + " единиц.");
    }
}

static class Planet
{
    private List<Materick> matericks;
    public Planet(List<Materick> matericks) {
        this.matericks = matericks;
    }
    public List<Materick> getMatericks() {
        return matericks;
    }
    public void setMatericks(List<Materick> matericks) {
        this.matericks = matericks;
    }
    public void description() {
        System.out.print("Планета Земля состоит из следующих материков: ");
        for (Materick materick : matericks) {
            System.out.print(materick.getName() + ", ");
        }
        System.out.println();
    }
    public void launchRocket() {
        System.out.println("Ракета успешно запущена с планеты Земля!");
    }
    public void explorePlanet() {
```

```
        System.out.println("Начато исследование планеты Земля.");
    }
    public void launchSatellite() {
        System.out.println("Спутник запущен с планеты Земля!");
    }
    public void weatherForecast(String date) {
        System.out.println("Прогноз погоды на " + date + " - солнечно.");
        System.out.println();
    }
}
```

```
Materick eurasia = new Materick("Евразия");
Materick africa = new Materick("Африка");
Materick northAmerica = new Materick("Северная Америка");

List<Materick> matericks = new ArrayList<>();
matericks.add(eurasia);
matericks.add(africa);
matericks.add(northAmerica);

Planet Earth = new Planet(matericks);
Earth.description();
Earth.launchRocket();
Earth.explorePlanet();
Earth.launchSatellite();
Earth.weatherForecast("22-03-2024");

eurasia.populationDensity(1000, 505005);
africa.populationDensity(2000, 3030003);
northAmerica.populationDensity(1500, 4040004);

eurasia.economySize(5000540.5434);
africa.economySize(3004300.25);
northAmerica.economySize(4045000.92);
```

Пример работы программы:

```
================================================================================
Планета Земля состоит из следующих материков: Евразия, Африка, Северная Америка,
Ракета успешно запущена с планеты Земля!
Начато исследование планеты Земля.
Спутник запущен с планеты Земля!
Прогноз погоды на 22-03-2024 - солнечно.

Плотность населения Евразия составляет 505.0 человек на квадратную единицу.
Плотность населения Африка составляет 1515.0 человек на квадратную единицу.
Плотность населения Северная Америка составляет 2693.3 человек на квадратную единицу.
Размер экономики Евразия составляет 5000540.5 единиц.
Размер экономики Африка составляет 3004300.3 единиц.
Размер экономики Северная Америка составляет 4045000.9 единиц.
================================================================================
```

Задание 3 (Вариант 11)
Построить модель программной системы с применением отношений (обобщения, агрегации, ассоциации, реализации) между классами. Задать атрибуты и методы классов. Реализовать (если необходимо) дополнительные классы. Продемонстрировать работу разработанной системы.

Система Аэрофлот
Администратор формирует летную Бригаду (пилоты, штурман, радист, стюардессы) на Рейс. Каждый Рейс выполняется Самолетом с определенной вместимостью и дальностью полета. Рейс может быть отменен из-за погодных условий в Аэропорту отлета или назначения. Аэропорт назначения может быть изменен в полете из-за технических неисправностей, о которых сообщил командир.

Спецификации ввода-вывода программы:
1. Классы:
   - Airport (Аэропорт): хранит информацию об аэропорте, его названии и позволяет изменять место назначения рейса.
   - Aircraft (Самолет): хранит информацию о вместимости, дальности полета и модели самолета.
   - CrewMember (Член Экипажа): базовый класс для членов экипажа, содержит информацию о имени и возрасте.
   - Pilot (Пилот), Navigator (Штурман), RadioOperator (Радист), Stewardess (Стюардесса): производные классы от CrewMember, представляют соответствующие члены экипажа с уникальными методами.
2. Методы:
   1) Pilot:
      - flyAircraft(): Управляет самолетом.
      - getVoice(): Объявляет информацию пассажирам.
   2) Navigator:
      - navigate(): Определяет маршрут полета.
   3) RadioOperator:
      - speakWithEarth(): Связывается с диспетчерским центром.
   4) Stewardess:
      - servePassengers(): Обслуживает пассажиров.
      - conductBriefing(): Проводит инструктаж пассажиров.
   5) Flight (Рейс):
      - Хранит информацию о вылете и прилете, использует объекты Airport и Aircraft.
      - Методы changeDestination(String s) и setDestination(String newDestination) позволяют изменять место назначения рейса.
   6) Метод getJob():
      - Принимает список пилотов (pilotList), штурмана (navigator), радиооператора (radioOperator), списки стюардесс (stewardessList) и аэропорт назначения (destinationAirport).

- Ожидает 500 миллисекунд (просто для имитации задержки).
Если имя пилота равно "Сорока Вадим Сергеевич":
  - Вызывается метод flyAircraft() для этого пилота.
  - Иначе вызывается метод getVoice() для этого пилота.
- Происходит задержка в 500 миллисекунд.
- Если штурман (navigator) является экземпляром класса Navigator:
  - Происходит приведение типа к Navigator.
  - Вызывается метод navigate() для штурмана.
- Происходит задержка в 500 миллисекунд.
- Если радиооператор (radioOperator) является экземпляром класса RadioOperator:
  - Происходит приведение типа к RadioOperator.
  - Вызывается метод speakWithEarth() для радиооператора.
- Происходит задержка в 500 миллисекунд.
- Для каждой стюардессы в списке стюардесс (stewardessList):
Если текущий член экипажа является экземпляром класса Stewardess:
  - Происходит приведение типа к Stewardess.
  - Если имя стюардессы равно "Петрова Александра Сергеевна", вызывается метод conductBriefing().
  - Иначе вызывается метод servePassengers().
- Происходит задержка выполнения кода на 3 секунды (Thread.sleep(2000)).
- Случайным образом выбирается число от 0 до 2.
  - Если случайное число равно 0:
    1) Если погода хорошая (метод isGoodWeather() возвращает true):
       - Выводится сообщение о резком ухудшении погодных условий и временной недоступности аэропорта назначения.
       - Создается новый объект Airport и конечный пункт прибытия изменяется на новый аэропорт.
       - Выводится сообщение о завершении рейса в новой конечной точке.
    2) Иначе выводится сообщение о успешном завершении рейса.
  - Если случайное число равно 1:
    1) Выводится сообщение о технической неисправности воздушного судна и изменении курса на ближайший аэропорт.
    2) Создается новый объект Airport и конечный пункт прибытия изменяется на новый аэропорт.

7) Метод scheduleFlight:
- Принимает списки stewardessList, pilotList, объекты navigator, radioOperator и destinationAirport.
- Ожидает изменения погоды на протяжении 5 секунд.
- Если погода становится хорошей, начинает подготовку к рейсу.
- Если погода остается плохой, повторяет проверку через 5 секунд. При этом, если срабатывает условие "плохая погода" дважды, рейс отменяется.

8) Метод getWeatherCondition:

- Возвращает случайное число от 0 до 2, представляющее погодные условия.

9) Метод isGoodWeather:
- Возвращает случайное булево значение, указывающее на хорошую или плохую погоду.

10) Переменная badWeatherCount:
- Используется для отслеживания количества срабатываний условия "плохая погода" в методе scheduleFlight.

3. Вывод информации о формировании лётной бригады:
- Выводится сообщение о формировании лётной бригады.
- Последовательно выводится информация о пилотах, навигаторе, радиооператоре и стюардессах.
- Для пилотов и стюардесс выводится их имя с соответствующим номером в списке.
- Для навигатора и радиооператора выводится только их имя.

4. Вывод информации о рейсе и погодных условиях:
- Выводится информация о рейсе (номер рейса, аэропорт отправления и назначения, модель и характеристики самолёта).
- В зависимости от погодных условий, выводится соответствующее сообщение:
А. При солнечной погоде в обоих аэропортах рейс разрешён.
Б. При наличии сильных осадков и штормового предупреждения в аэропорту отправления или назначения, рейс задерживается.
В. В случае непредвиденного состояния погоды также происходит задержка рейса.

4. Вызов методов для управления рейсом:
- При солнечной погоде вызывается метод getJob для назначения обязанностей членам экипажа и начала рейса.
- При наличии плохих погодных условий вызывается метод scheduleFlight для повторной проверки погоды и управления рейсом в зависимости от неё.

Текст программы

```
    // Класс Аэропорт
    static class Airport {
        private String name;

        public Airport(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }

        public void changeDestination(Flight flight, String newDestination) {
            System.out.println("Рейс изменяет место назначения из " +
flight.destinationAirport.getName() + " в " + newDestination);
            flight.setDestination(newDestination);
            flight.destinationAirport = this; // Обновляем аэропорт
назначения рейса
        }
    }

    // Класс Самолет
    static class Aircraft
    {
        private int capacity;
        private int range;
        private String model;

        public Aircraft(int capacity, int range, String model) {
            this.capacity = capacity;
            this.range = range;
            this.model = model;
        }

        public int getCapacity() {
            return capacity;
        }
        public int getRange() {
            return range;
        }
    }

    // Класс ЧленЭкипажа
    class CrewMember
    {
        private String name;
        private int age;

        public CrewMember(String name, int age) {
            this.name = name;
            this.age = age;
        }

        public String getName() {
            return name;
        }
        public int getAge() {
            return age;
        }
    }
```

```
    // Класс Пилот (производный класс от ЧленЭкипажа)
    class Pilot extends CrewMember
    {
        public Pilot(String name, int age) {
            super(name, age);
        }

        public void flyAircraft() {
            System.out.println("Главный пилот " + getName() + " управляет
самолётом");
        }

        public void getVoice() {
            System.out.println("Помощник пилота " + getName() + " объявляет
необходимую информацию пассажирам по громкоговорителю");
        }
    }

    // Класс Штурман (производный класс от ЧленЭкипажа)
    class Navigator extends CrewMember
    {
        public Navigator(String name, int age) {
            super(name, age);
        }

        public void navigate() {
            System.out.println("Штурман определяет маршрут полёта...");
        }
    }

    // Класс Радист (производный класс от ЧленЭкипажа)
    class RadioOperator extends CrewMember
    {
        public RadioOperator(String name, int age) {
            super(name, age);
        }

        public void speakWithEarth() {
            System.out.println("Радист " + getName() + " докладывает
обстановку по рации в диспетчерский центр");
        }
    }

    // Класс Стюардесса (производный класс от ЧленЭкипажа)
    class Stewardess extends CrewMember
    {
        public Stewardess(String name, int age) {
            super(name, age);
        }

        public void servePassengers() {
            System.out.println("Стюардесса " + getName() + " обслуживает
пассажиров в самолете");
        }

        public void conductBriefing() {
            System.out.println("Стюардесса " + getName() + " проводит
инструктаж пассажиров");
        }
    }

    // Класс Рейс
    static class Flight
    {
```

```
        private Airport departureAirport;
        private Airport destinationAirport;
        private Aircraft aircraft;

        public Flight(Airport departureAirport, Airport destinationAirport,
Aircraft aircraft) {
            this.departureAirport = departureAirport;
            this.destinationAirport = destinationAirport;
            this.aircraft = aircraft;
        }

        public void changeDestination(String s) {
        }

        public void setDestination(String newDestination) {
        }
    }

    private static void getJob(List<CrewMember> pilotList, CrewMember
navigator, CrewMember radioOperator, List<CrewMember> stewardessList, Airport
destinationAirport) throws InterruptedException {

        Thread.sleep(500);
        for (CrewMember crewMember : pilotList)
        {
            if (crewMember instanceof Pilot) {
                Pilot pilot = (Pilot) crewMember;
                if (pilot.getName().equals("Сорока Вадим Сергеевич")) {
                    pilot.flyAircraft();
                } else {
                    pilot.getVoice();
                }
            }
        }

        Thread.sleep(500);
        if (navigator instanceof Navigator) {
            Navigator castedNavigator = (Navigator) navigator;
            castedNavigator.navigate();
        }

        Thread.sleep(500);
        if (radioOperator instanceof RadioOperator) {
            RadioOperator castedOperator = (RadioOperator) radioOperator;
            castedOperator.speakWithEarth();
        }
        Thread.sleep(500);
        for (CrewMember crewMember : stewardessList)
        {
            if (crewMember instanceof Stewardess) {
                Stewardess stewardess = (Stewardess) crewMember;
                if (stewardess.getName().equals("Петрова Александра
Сергеевна")) {
                    stewardess.conductBriefing();
                } else {
                    stewardess.servePassengers();
                }
            }
        }

        try {
            // Задержка выполнения кода на 3 секунды
            Thread.sleep(2000);
            System.out.println("\nРейс проходит в штатном режиме...");
```

```
            Thread.sleep(3000);

            Random random = new Random();
            int randomNumber = random.nextInt(3); // 0, 1, 2
            if (randomNumber == 0)
            {
                if (isGoodWeather())
                {
                    Thread.sleep(500);
                    System.out.println("\nВнимание!!! В процессе рейса
произошло резкое ухудшение погодных условий. Конечный аэропорт " +
destinationAirport.getName() + " временно недоступен для посадки!");

                    // Создаем новый объект Airport
                    Airport newDestinationAirport = new Airport("Аэропорт
Иваново-Северный");
                    Thread.sleep(500);
                    System.out.println("Конечный пункт прибытия изменён с " +
destinationAirport.getName() + " на " + newDestinationAirport.getName());

                    // Обновляем значение destinationAirport
                    destinationAirport = newDestinationAirport;

                    try {
                        System.out.println("\nРейс проходит в штатном
режиме...");
                        // Задержка выполнения кода на 5 секунд
                        Thread.sleep(2000);
                        System.out.println("\nСамолёт прибыл в " +
destinationAirport.getName());
                        Thread.sleep(500);
                        System.out.println("Рейс №228 был успешно завершён в
другой конечной точке!");
                    } catch (InterruptedException e) {
                        // Обработка исключения, если прерывание произошло
                        e.printStackTrace();
                    }
                } else {
                    System.out.println("\nСамолёт прибыл в " +
destinationAirport.getName());
                    Thread.sleep(500);
                    System.out.println("Рейс №228 успешно завершён!");
                }

            } else if (randomNumber == 1) {
                Thread.sleep(500);
                System.out.println("\nГоворит главный пилот самолёта " +
pilotList.get(0).getName() + "! В связи с технической неисправностью
воздушного судна мы не сможем долететь до аэропорта назначения: " +
destinationAirport.getName() + ". ");

                // Создаем новый объект Airport
                Airport newDestinationAirport = new Airport("Аэропорт
Кобылёво");
                Thread.sleep(500);
                System.out.println("Поэтому мы изменяем курс на ближайший к
нам аэропорт: " + newDestinationAirport.getName() + ". Авиакомпания заранее
приносит свои извенения, все подробности будут позже!");

                // Обновляем значение destinationAirport
                destinationAirport = newDestinationAirport;

                try {
                    System.out.println("\nРейс проходит в аварийном
```

```
режиме...");
                    // Задержка выполнения кода на 5 секунд
                    Thread.sleep(2000);
                    System.out.println("\nСамолёт прибыл в " +
destinationAirport.getName());
                    Thread.sleep(500);
                    System.out.println("Рейс №228 завершён преждевременно из-
за технической неисправности!");
                } catch (InterruptedException e) {
                    // Обработка исключения, если прерывание произошло
                    e.printStackTrace();
                }

            } else {
                System.out.println("\nСамолёт прибыл в " +
destinationAirport.getName());
                Thread.sleep(500);
                System.out.println("Рейс №228 успешно завершён!");
            }

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Переменная для отслеживания количества срабатываний условия else
    private static int badWeatherCount = 0;

    private static void scheduleFlight(List<CrewMember> stewardessList,
List<CrewMember> pilotList, CrewMember navigator, CrewMember radioOperator,
Airport destinationAirport)
    {
        System.out.println("Ожидаем изменения погоды...");
        try {
            // Пауза на 5 секунд
            long delay = 5000;
            long startTime = System.currentTimeMillis();
            while (System.currentTimeMillis() - startTime < delay) {
                Thread.sleep(1000); // Можно делать проверки каждую секунду
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        if (isGoodWeather()) {
            System.out.println("Погода улучшилась. Начинаем подготовку к
рейсу... \n");
            try {
                getJob(pilotList, navigator, radioOperator, stewardessList,
destinationAirport);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        } else {
            // Увеличиваем счетчик при каждом срабатывании условия else
            badWeatherCount++;

            if (badWeatherCount < 2) {
                System.out.println("Продолжается плохая погода. Повторная
проверка через 5 секунд...");
                scheduleFlight(stewardessList, pilotList, navigator,
radioOperator, destinationAirport);
            } else {
                System.out.println("Продолжается плохая погода. Рейс
```

```
отменён.");
            }
        }
    }


    private static int getWeatherCondition()
    {
        Random random = new Random();
        return  random.nextInt(3);
    }

    private static boolean isGoodWeather()
    {
        Random random = new Random();
        return random.nextBoolean();
    }
```

```
List<CrewMember> stewardessList = new ArrayList<>();
List<CrewMember> pilotList = new ArrayList<>();

// Создаем объекты классов
Aircraft aircraft = new Aircraft(200, 1500, "Boeing 737 №555");
Airport departureAirport = new Airport("Аэропорт Брестский ()");
Airport destinationAirport = new Airport("Аэропорт Иваново (IWA)");

// Создаем рейс
Flight flight = new Flight(departureAirport, destinationAirport, aircraft);

// Создаем и добавляем членов экипажа
CrewMember first_pilot = zachetnayaKnizhka.new Pilot("Сорока Вадим
Сергеевич", 30);
pilotList.add(first_pilot);
CrewMember second_pilot = zachetnayaKnizhka.new Pilot("Капитонов Максим
Игоревич", 29);
pilotList.add(second_pilot);

CrewMember navigator = zachetnayaKnizhka.new Navigator("Таразевич Никита
Александрович", 29);
CrewMember radioOperator = zachetnayaKnizhka.new RadioOperator("Дорошков
Александр Дмитриевич", 29);

CrewMember first_stewardess = zachetnayaKnizhka.new Stewardess("Сидорова
Елена Николаевна", 24);
stewardessList.add(first_stewardess);
CrewMember second_stewardess = zachetnayaKnizhka.new Stewardess("Петрова
Александра Сергеевна", 23);
stewardessList.add(second_stewardess);

// Вывод сообщения о формировании лётной бригады
Thread.sleep(500);
System.out.println("Администратор сформировал лётную бригаду:");

// Вывод списка пилотов с нумерацией
Thread.sleep(500);
System.out.println("Пилоты:");
for (int i = 0; i < pilotList.size(); i++) {
    System.out.println((i + 1) + ") " + pilotList.get(i).getName());
}

// Вывод навигатора и радиооператора
Thread.sleep(500);
```

```
System.out.println("Навигатор: " + navigator.getName());
System.out.println("Радиооператор: " + radioOperator.getName());

// Вывод списка стюардесс с нумерацией
Thread.sleep(500);
System.out.println("Стюардессы:");
for (int i = 0; i < stewardessList.size(); i++)
{
    System.out.println((i + 1) + ") " + stewardessList.get(i).getName());
}
Thread.sleep(500);
System.out.println("\nРейс №228: " + departureAirport.getName() + " -- " +
destinationAirport.getName() + "\nСамолёт: " + aircraft.model +
"\nВместимость: " + aircraft.capacity + "\nДальность рейса: " +
aircraft.range);
System.out.println("\n==========================================================
=====\n");

int weatherCondition = getWeatherCondition();
switch (weatherCondition) {
    case 0:
        System.out.println("Солнечная погода в обоих аэропортах. Рейс
разрешен.\n");
        try {
            getJob(pilotList, navigator, radioOperator, stewardessList,
destinationAirport);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        break;
    case 1:
        System.out.println("Сильные осадки, штормовое предупреждение в
аэропорту отправления. Рейс задерживается.");
        scheduleFlight(stewardessList, pilotList , navigator, radioOperator,
destinationAirport);
        break;
    case 2:
        System.out.println("Сильные осадки, штормовое предупреждение в
аэропорту назначения. Рейс задерживается.");
        scheduleFlight(stewardessList, pilotList , navigator, radioOperator,
destinationAirport);
        break;
    default:
        System.out.println("Непредвиденное состояние погоды.");
        scheduleFlight(stewardessList, pilotList , navigator, radioOperator,
destinationAirport);
        break;
}
```

Пример работы программы

А) хорошая погода без происшествий

```
Солнечная погода в обоих аэропортах. Рейс разрешен.

Главный пилот Сорока Вадим Сергеевич управляет самолётом
Помощник пилота Капитонов Максим Игоревич объявляет необходимую информацию пассажирам по громкоговорителю
Штурман определяет маршрут полёта...
Радист Дорошков Александр Дмитриевич докладывает обстановку по рации в диспетчерский центр
Стюардесса Сидорова Елена Николаевна обслуживает пассажиров в самолете
Стюардесса Петрова Александра Сергеевна проводит инструктаж пассажиров

Рейс проходит в штатном режиме...

Самолёт прибыл в Аэропорт Иваново (IWA)
Рейс №228 успешно завершён!
```

Б) хорошая погода в обоих аэропортах + техническая неисправность в полёте

```
================================================================================
Администратор сформировал лётную бригаду:
Пилоты:
1) Сорока Вадим Сергеевич
2) Капитонов Максим Игоревич
Навигатор: Таразевич Никита Александрович
Радиооператор: Дорошков Александр Дмитриевич
Стюардессы:
1) Сидорова Елена Николаевна
2) Петрова Александра Сергеевна

Рейс №228: Аэропорт Брестский () -- Аэропорт Иваново (IWA)
Самолёт: Boeing 737 №555
Вместимость: 200
Дальность рейса: 1500

============================================================

Солнечная погода в обоих аэропортах. Рейс разрешен.

Главный пилот Сорока Вадим Сергеевич управляет самолётом
Помощник пилота Капитонов Максим Игоревич объявляет необходимую информацию пассажирам по громкоговорителю
Штурман определяет маршрут полёта...
Радист Дорошков Александр Дмитриевич докладывает обстановку по рации в диспетчерский центр
Стюардесса Сидорова Елена Николаевна обслуживает пассажиров в самолете
Стюардесса Петрова Александра Сергеевна проводит инструктаж пассажиров

Рейс проходит в штатном режиме...
Говорит главный пилот самолёта Сорока Вадим Сергеевич! В связи с технической неисправностью воздушного судна мы не сможем долететь до аэропорта назначения: Аэропорт Иваново (IWA).
Поэтому мы изменяем курс на ближайший к нам аэропорт: Аэропорт Кобылёво. Авиакомпания заранее приносит свои извенения, все подробности будут позже!

Рейс проходит в аварийном режиме...

Самолёт прибыл в Аэропорт Кобылёво
Рейс №228 завершён преждевременно из-за технической неисправности!
================================================================================
```

В) плохая погода в аэропорте отправления + плохая погода в ходе рейса

```
Сильные осадки, штормовое предупреждение в аэропорту отправления. Рейс задерживается.
Ожидаем изменения погоды...
Погода улучшилась. Начинаем подготовку к рейсу...

Главный пилот Сорока Вадим Сергеевич управляет самолётом
Помощник пилота Капитонов Максим Игоревич объявляет необходимую информацию пассажирам по громкоговорителю
Штурман определяет маршрут полёта...
Радист Дорошков Александр Дмитриевич докладывает обстановку по рации в диспетчерский центр
Стюардесса Сидорова Елена Николаевна обслуживает пассажиров в самолете
Стюардесса Петрова Александра Сергеевна проводит инструктаж пассажиров

Рейс проходит в штатном режиме...

Внимание!!! В процессе рейса произошло резкое ухудшение погодных условий. Конечный аэропорт Аэропорт Иваново (IWA) временно недоступен для посадки!
Конечный пункт прибытия изменён с Аэропорт Иваново (IWA) на Аэропорт Иваново-Северный

Рейс проходит в штатном режиме...

Самолёт прибыл в Аэропорт Иваново-Северный
Рейс №228 был успешно завершён в другой конечной точке!
```

Г) хорошая погода + изменение погоды в ходе рейса

```
Солнечная погода в обоих аэропортах. Рейс разрешен.

Главный пилот Сорока Вадим Сергеевич управляет самолётом
Помощник пилота Капитонов Максим Игоревич объявляет необходимую информацию пассажирам по громкоговорителю
Штурман определяет маршрут полёта...
Радист Дорошков Александр Дмитриевич докладывает обстановку по рации в диспетчерский центр
Стюардесса Сидорова Елена Николаевна обслуживает пассажиров в самолете
Стюардесса Петрова Александра Сергеевна проводит инструктаж пассажиров

Рейс проходит в штатном режиме...

Внимание!!! В процессе рейса произошло резкое ухудшение погодных условий. Конечный аэропорт Аэропорт Иваново (IWA) временно недоступен для посадки!
Конечный пункт прибытия изменён с Аэропорт Иваново (IWA) на Аэропорт Иваново-Северный

Рейс проходит в штатном режиме...

Самолёт прибыл в Аэропорт Иваново-Северный
Рейс №228 был успешно завершён в другой конечной точке!
```

Д) отмена рейса из-за длительной плохой погоды

```
Сильные осадки, штормовое предупреждение в аэропорту назначения. Рейс задерживается.
Ожидаем изменения погоды...
Продолжается плохая погода. Повторная проверка через 5 секунд...
Ожидаем изменения погоды...
Продолжается плохая погода. Рейс отменён.
```

Вариантов различных исходов больше.

Вывод: приобрёл практические навыки в области ООА языка программирования Java при решении практических задач.