МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ОТЧЁТ

по лабораторной работе №3

Выполнила:
студентка группы ПО-9
Кот А. А.

Проверил:
Крощенко А. А.

Брест 2024

Цель работы: научиться создавать и использовать классы в программах на языке программирования Java.

Вариант 8.
Ход работы

Задание 1.

Реализовать простой класс.

Требования к выполнению
• Реализовать пользовательский класс по варианту:
Множество целых чисел переменной мощности – Предусмотреть возможность пересечения двух множеств, вывода на печать элементов множества, а так же метод, определяющий, принадлежит ли указанное значение множеству. Класс должен содержать методы, позволяющие добавлять и удалять элемент в/из множества. Конструктор должен позволить создавать объекты с начальной инициализацией. Реализацию множества осуществить на базе структуры ArrayList. Реализовать метод equals, выполняющий сравнение объектов данного типа.

• Создать другой класс с методом main, в котором будут находится примеры использования пользовательского класса.

Для каждого класса
• Создать поля классов
• Создать методы классов
• Добавьте необходимые get и set методы (по необходимости)
• Укажите соответствующие модификаторы видимости
• Добавьте конструкторы
• Переопределить методы toString() и equals()

Работа программы:

```
Set 1: [1, 2, 3]
Set 2: [2, 3, 4]
Is 2 in set1? true
Is 4 in set1? false
Intersection of set1 and set2: [2, 3]
Set1 == Set2? false
Set 1 after adding 4: [1, 2, 3, 4]
Set 2 after removing 4: [2, 3]

Process finished with exit code 0
```

Код программы:

Main.java

```
import java.math.BigInteger;
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
```

```
        ArrayList<BigInteger> set1Elements = new ArrayList<>();
        set1Elements.add(BigInteger.valueOf(1));
        set1Elements.add(BigInteger.valueOf(2));
        set1Elements.add(BigInteger.valueOf(3));
        BigIntegerSet set1 = new BigIntegerSet(set1Elements);
        System.out.println("Set 1: " + set1.toString());

        BigIntegerSet set2 = new BigIntegerSet();
        set2.addElement(BigInteger.valueOf(2));
        set2.addElement(BigInteger.valueOf(3));
        set2.addElement(BigInteger.valueOf(4));
        System.out.println("Set 2: " + set2.toString());

        System.out.println("Is 2 in set1? "
                + set1.contains(BigInteger.valueOf(2)));
        System.out.println("Is 4 in set1? "
                + set1.contains(BigInteger.valueOf(4)));
        System.out.println("Intersection of set1 and set2: "
                + set1.intersection(set2));
        System.out.println("Set1 == Set2? " + set1.equals(set2));

        set1.addElement(BigInteger.valueOf(4));
        System.out.println("Set 1 after adding 4: " + set1.toString());

        set2.removeElement(BigInteger.valueOf(4));
        System.out.println("Set 2 after removing 4: " + set2.toString());
    }
}
```

BigIntegerSet.java

```
import java.math.BigInteger;
import java.util.ArrayList;

public class BigIntegerSet {
    private ArrayList<BigInteger> elements;

    public BigIntegerSet() {
        this.elements = new ArrayList<>();
    }

    public BigIntegerSet(ArrayList<BigInteger> initialElements) {
        this.elements = new ArrayList<>(initialElements);
    }

    public void addElement(BigInteger element) {
        if (!elements.contains(element)) {
            elements.add(element);
        }
    }

    public void removeElement(BigInteger element) {
        elements.remove(element);
    }

    public boolean contains(BigInteger element) {
        return elements.contains(element);
    }

    public BigIntegerSet intersection(BigIntegerSet otherSet) {
        ArrayList<BigInteger> intersectionElements = new ArrayList<>();
        for (BigInteger element : elements) {
            if (otherSet.contains(element)) {
                intersectionElements.add(element);
            }
```

```
        }
        return new BigIntegerSet(intersectionElements);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof BigIntegerSet)) return false;
        BigIntegerSet otherSet = (BigIntegerSet) obj;
        return this.elements.equals(otherSet.elements);
    }

    @Override
    public String toString() {
        return elements.toString();
    }
}
```

Задание 2.

Разработать на Java автоматизированную систему на основе некоторой структуры данных, манипулирующей объектами пользовательского класса. Реализовать требуемые функции обработки данных.

Требования к выполнению
• Задание посвящено написанию классов, решающих определенную задачу автоматизации;
• Данные для программы загружаются из файла (формат произволен). Файл создать и написать вручную.

Вариант: Автоматизированная система обработки информации об авиарейсах

Написать программу для обработки информации об авиарейсах (Airlines): Каждый рейс имеет следующие характеристики:
• Пункт назначения;
• Номер рейса;
• Тип самолета;
• Время вылета;
• Дни недели, по которым совершаются рейсы.

Программа должна обеспечить:
• Генерацию списка рейсов;
• Вывод списка рейсов для заданного пункта назначения;
• Вывод списка рейсов для заданного дня недели;
• Вывод списка рейсов для заданного дня недели, время вылета для которых больше заданного;
• Все рейсы самолетов некоторого типа;
• Группировка рейсов по числу пассажиров (маломестные - 1-100 чел, средместные (100-200), крупные рейсы (200-350));
• Все рейсы самолетов туда-обратно.

Работа программы:

```
Flights to London:
101 - Tue Apr 02 08:00:00 MSK 2024
104 - Fri Apr 05 14:00:00 MSK 2024

Flights on Monday:
101 - Tue Apr 02 08:00:00 MSK 2024
103 - Thu Apr 04 12:00:00 MSK 2024
106 - Sun Apr 07 18:00:00 MSK 2024
107 - Mon Apr 08 20:00:00 MSK 2024

Monday flights after 12:00:
103 - Thu Apr 04 12:00:00 MSK 2024
106 - Sun Apr 07 18:00:00 MSK 2024
107 - Mon Apr 08 20:00:00 MSK 2024

Boeing 737 flights:
101 - Tue Apr 02 08:00:00 MSK 2024
104 - Fri Apr 05 14:00:00 MSK 2024
107 - Mon Apr 08 20:00:00 MSK 2024
```

```
Grouped flights by passenger capacity:
Small capacity flights:
103 - Thu Apr 04 12:00:00 MSK 2024 - Passenger capacity = 100
106 - Sun Apr 07 18:00:00 MSK 2024 - Passenger capacity = 100

Medium capacity flights:
101 - Tue Apr 02 08:00:00 MSK 2024 - Passenger capacity = 180
104 - Fri Apr 05 14:00:00 MSK 2024 - Passenger capacity = 180
107 - Mon Apr 08 20:00:00 MSK 2024 - Passenger capacity = 180

Large capacity flights:
102 - Wed Apr 03 10:00:00 MSK 2024 - Passenger capacity = 300
105 - Sat Apr 06 16:00:00 MSK 2024 - Passenger capacity = 300

Round trip flights:
101 - Tue Apr 02 08:00:00 MSK 2024
104 - Fri Apr 05 14:00:00 MSK 2024

Process finished with exit code 0
```

Код программы (классы Flight и FlightManager, класс Main можно найти на GitHub):

Flight.java

```
import java.util.Date;
import java.util.List;

public class Flight {
    private String destination;
    private int flightNumber;
    private String aircraftType;
    private Date departureTime;
    private List<String> operatingDays;

    public Flight(String destination, int flightNumber,
                  String aircraftType, Date departureTime,
                  List<String> operatingDays) {
        this.destination = destination;
        this.flightNumber = flightNumber;
        this.aircraftType = aircraftType;
        this.departureTime = departureTime;
        this.operatingDays = operatingDays;
    }

    public int calculatePassengerCapacity(String aircraftType) {
        if (aircraftType.equalsIgnoreCase("Boeing 737")) {
            return 180;
        } else if (aircraftType.equalsIgnoreCase("Airbus A320")) {
            return 100;
        } else if (aircraftType.equalsIgnoreCase("Boeing 777")) {
            return 300;
        } else {
            return 0;
        }
    }

    public String getDestination() {
        return destination;
    }

    public int getFlightNumber() {
```

```
        return flightNumber;
    }

    public String getAircraftType() {
        return aircraftType;
    }

    public Date getDepartureTime() {
        return departureTime;
    }

    public List<String> getOperatingDays() {
        return operatingDays;
    }
}
```

FlightManager.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

public class FlightManager {
    private List<Flight> flights;

    public FlightManager() {
        this.flights = new ArrayList<>();
    }

    public void loadFlightsFromFile(String filename) throws
FileNotFoundException, ParseException {
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm");
        File file = new File(filename);
        Scanner scanner = new Scanner(file);
        while (scanner.hasNextLine()) {
            String[] parts = scanner.nextLine().split(",");
            String destination = parts[0];
            int flightNumber = Integer.parseInt(parts[1]);
            String aircraftType = parts[2];
            Date departureTime = dateFormat.parse(parts[3]);
            String[] daysArray = parts[4].split(";");
            List<String> operatingDays = new ArrayList<>();
            operatingDays.addAll(Arrays.asList(daysArray));
            Flight flight = new Flight(destination, flightNumber,
aircraftType, departureTime, operatingDays);
            flights.add(flight);
        }
        scanner.close();
    }

    public List<Flight> getFlightsByDestination(String destination) {
        List<Flight> filteredFlights = new ArrayList<>();
        for (Flight flight : flights) {
            if (flight.getDestination().equalsIgnoreCase(destination)) {
                filteredFlights.add(flight);
            }
        }
        return filteredFlights;
    }

    public List<Flight> getFlightsByDay(String day) {
        List<Flight> filteredFlights = new ArrayList<>();
```

```
        for (Flight flight : flights) {
            if (flight.getOperatingDays().contains(day)) {
                filteredFlights.add(flight);
            }
        }
        return filteredFlights;
    }

    public List<Flight> getFlightsByDayAndTime(String day, Date time) {
        List<Flight> filteredFlights = new ArrayList<>();
        for (Flight flight : flights) {
            if (flight.getOperatingDays().contains(day) &&
flight.getDepartureTime().after(time)) {
                filteredFlights.add(flight);
            }
        }
        return filteredFlights;
    }

    public List<Flight> getFlightsByAircraftType(String aircraftType) {
        List<Flight> filteredFlights = new ArrayList<>();
        for (Flight flight : flights) {
            if (flight.getAircraftType().equalsIgnoreCase(aircraftType)) {
                filteredFlights.add(flight);
            }
        }
        return filteredFlights;
    }

    public List<List<Flight>> groupFlightsByPassengerCapacity() {
        List<List<Flight>> groupedFlights = new ArrayList<>();
        List<Flight> smallCapacityFlights = new ArrayList<>();
        List<Flight> mediumCapacityFlights = new ArrayList<>();
        List<Flight> largeCapacityFlights = new ArrayList<>();
        for (Flight flight : flights) {
            int passengerCapacity =
calculatePassengerCapacity(flight.getAircraftType());
            if (passengerCapacity >= 1 && passengerCapacity <= 100) {
                smallCapacityFlights.add(flight);
            } else if (passengerCapacity > 100 && passengerCapacity <= 200) {
                mediumCapacityFlights.add(flight);
            } else if (passengerCapacity > 200 && passengerCapacity <= 350) {
                largeCapacityFlights.add(flight);
            }
        }
        groupedFlights.add(smallCapacityFlights);
        groupedFlights.add(mediumCapacityFlights);
        groupedFlights.add(largeCapacityFlights);
        return groupedFlights;
    }

    private int calculatePassengerCapacity(String aircraftType) {
        if (aircraftType.equalsIgnoreCase("Boeing 737")) {
            return 180;
        } else if (aircraftType.equalsIgnoreCase("Airbus A320")) {
            return 100;
        } else if (aircraftType.equalsIgnoreCase("Boeing 777")) {
            return 300;
        } else {
            return 0;
        }
    }

    public List<Flight> getRoundTripFlights() {
        List<Flight> roundTripFlights = new ArrayList<>();
        for (Flight outboundFlight : flights) {
```

```
            for (Flight returnFlight : flights) {
                if
(outboundFlight.getDestination().equalsIgnoreCase(returnFlight.getDestination
()) &&
                        outboundFlight.getFlightNumber() !=
returnFlight.getFlightNumber()) {
                    roundTripFlights.add(outboundFlight);
                }
            }
        }
        return roundTripFlights;
    }
}
```

Вывод: в ходе лабораторной работы мы научились создавать и использовать классы в программах на языке программирования Java.