Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

по дисциплине “СПП”

Вариант 7

Выполнил:

Студентка 3 курса

Группы ПО-8

Гордейчук М.В.

Проверила:

Крощенко А.А.

2024

Лабораторная работа №3

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

Задание 1

Множество символов ограниченной мощности – Предусмотреть возможность объединения 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. Мощность множества задается при создании объекта. Реализацию множества осуществить на базе одномерного массива. Реализовать метод equals, выполняющий сравнение объектов данного типа.

Код программы:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nВыберите действие:");
            System.out.println("1. Объединение множеств");
            System.out.println("2. Проверка принадлежности элемента первому
множеству");
            System.out.println("3. Удаление элемента из первого множества");
            System.out.println("4. Вывод множеств на экран");
            System.out.println("5. Выйти из программы");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    performUnion(scanner);
                    break;
                case 2:
                    performContainsCheck(scanner);
                    break;
                case 3:
                    performRemoval(scanner);
                    break;
                case 4:
                    printSets(scanner);
                    break;
                case 5:
                    System.out.println("Программа завершена.");
                    scanner.close();
                    return;
                default:
                    System.out.println("Некорректный выбор.");
            }
        }
    }

    private static void performUnion(Scanner scanner) {
        System.out.println("Введите мощность первого множества: ");
        int capacity1 = scanner.nextInt();
        SymbolSet set1 = new SymbolSet(capacity1);
```

```
        System.out.println("Введите элементы первого множества:");
        for (int i = 0; i < capacity1; i++) {
            char element = scanner.next().charAt(0);
            set1.add(element);
        }

        System.out.println("Введите мощность второго множества: ");
        int capacity2 = scanner.nextInt();
        SymbolSet set2 = new SymbolSet(capacity2);

        System.out.println("Введите элементы второго множества:");
        for (int i = 0; i < capacity2; i++) {
            char element = scanner.next().charAt(0);
            set2.add(element);
        }

        SymbolSet unionSet = set1.union(set2);
        System.out.println("Объединение множеств: " + unionSet);
    }

    private static void performContainsCheck(Scanner scanner) {
        System.out.println("Введите элементы множества для проверки:");
        int capacity = scanner.nextInt();
        SymbolSet set = new SymbolSet(capacity);

        System.out.println("Введите элементы множества:");
        for (int i = 0; i < capacity; i++) {
            char element = scanner.next().charAt(0);
            set.add(element);
        }

        System.out.print("Введите элемент для проверки принадлежности
множеству: ");
        char element = scanner.next().charAt(0);
        System.out.println("Принадлежит ли элемент множеству: " +
set.contains(element));
    }

    private static void performRemoval(Scanner scanner) {
        System.out.println("Введите мощность множества:");
        int capacity = scanner.nextInt();
        SymbolSet set = new SymbolSet(capacity);

        System.out.println("Введите элементы множества:");
        for (int i = 0; i < capacity; i++) {
            char element = scanner.next().charAt(0);
            set.add(element);
        }

        System.out.print("Введите элемент для удаления из множества: ");
        char elementToRemove = scanner.next().charAt(0);
        set.remove(elementToRemove);
        System.out.println("Множество после удаления элемента: " + set);
    }

    private static void printSets(Scanner scanner) {
        System.out.println("Введите элементы первого множества (через
пробел): ");
        String input1 = scanner.nextLine();
        char[] elements1 = input1.toCharArray();
        SymbolSet set1 = new SymbolSet(elements1);

        System.out.println("Введите элементы второго множества (через
пробел): ");
```

```
        String input2 = scanner.nextLine();
        char[] elements2 = input2.toCharArray();
        SymbolSet set2 = new SymbolSet(elements2);

        System.out.println("Первое множество: " + set1);
        System.out.println("Второе множество: " + set2);
    }
}
```

```
Выберите действие:
1. Объединение множеств
2. Проверка принадлежности элемента первому множеству
3. Удаление элемента из первого множества
4. Вывод множеств на экран
5. Выйти из программы
1
Введите мощность первого множества:
2
Введите элементы первого множества:
1
2
Введите мощность второго множества:
2
Введите элементы второго множества:
3
4
Объединение множеств: { 1, 2, 3, 4 }
```

Задание 2

Система оповещений на дорожном вокзале

Автоматизированная информационная система на железнодорожном вокзале содержит сведения об отправлении поездов дальнего следования. Составить программу, которая должна хранить расписание поездов в структурированном, отсортированном по времени отправления виде (используя бинарное дерево).

• Обеспечивает первоначальный ввод данных в информационную систему о текущем расписании из файла и формирование дерева;

• Печатает все расписание на экран по команде;

• Выводит информацию о поезде по номеру поезда;

• По названию станции назначения выводит данные обо всех поездах, которые следуют до этой станции;

• Список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

• Список поездов, отправляющихся до заданного пункта назначения и имеющих общие места;

• За 10, 5, 3 минуты до отправления поезда показывает информационное сообщение об отправлении поезда.

Код программы:

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

// Класс, представляющий информацию о поезде
class Train {
    String trainNumber;
    String destinationStation;
    Date departureTime;
    int availableSeats;

    public Train(String trainNumber, String destinationStation, Date
departureTime, int availableSeats) {
        this.trainNumber = trainNumber;
        this.destinationStation = destinationStation;
        this.departureTime = departureTime;
        this.availableSeats = availableSeats;
    }

    @Override
    public String toString() {
        return "Train{" +
                "trainNumber='" + trainNumber + '\'' +
                ", destinationStation='" + destinationStation + '\'' +
                ", departureTime=" + departureTime +
                ", availableSeats=" + availableSeats +
                '}';
    }
}

// Класс, представляющий узел бинарного дерева
class TreeNode {
    Train train;
    TreeNode left;
    TreeNode right;

    public TreeNode(Train train) {
        this.train = train;
        left = null;
        right = null;
    }
}

// Класс, представляющий расписание и операции с ним
class TrainSchedule {
    TreeNode root;

    public void addTrain(Train train) {
        root = addRecursive(root, train);
    }

    private TreeNode addRecursive(TreeNode current, Train train) {
        if (current == null) {
            return new TreeNode(train);
```

```
        }

        if (train.departureTime.compareTo(current.train.departureTime) < 0) {
            current.left = addRecursive(current.left, train);
        } else if (train.departureTime.compareTo(current.train.departureTime)
> 0) {
            current.right = addRecursive(current.right, train);
        }

        return current;
    }

    public void printSchedule() {
        printInOrder(root);
    }

    private void printInOrder(TreeNode node) {
        if (node != null) {
            printInOrder(node.left);
            System.out.println(node.train);
            printInOrder(node.right);
        }
    }

    public Train findTrainByNumber(String trainNumber) {
        return findTrainByNumber(root, trainNumber);
    }

    private Train findTrainByNumber(TreeNode node, String trainNumber) {
        if (node == null) {
            return null;
        }

        if (trainNumber.equals(node.train.trainNumber)) {
            return node.train;
        } else if (trainNumber.compareTo(node.train.trainNumber) < 0) {
            return findTrainByNumber(node.left, trainNumber);
        } else {
            return findTrainByNumber(node.right, trainNumber);
        }
    }

    public List<Train> getTrainsToDestinationAfterTime(String destination,
Date time) {
        List<Train> trains = new ArrayList<>();
        getTrainsToDestinationAfterTime(root, destination, time, trains);
        return trains;
    }

    private void getTrainsToDestinationAfterTime(TreeNode node, String
destination, Date time, List<Train> result) {
        if (node == null) {
            return;
        }

        getTrainsToDestinationAfterTime(node.left, destination, time,
result);

        if (node.train.destinationStation.equals(destination) &&
node.train.departureTime.after(time)) {
            result.add(node.train);
        }

        getTrainsToDestinationAfterTime(node.right, destination, time,
```

```
result);
    }

    public List<Train> getTrainsToDestination(String destination) {
        List<Train> trains = new ArrayList<>();
        getTrainsToDestination(root, destination, trains);
        return trains;
    }

    private void getTrainsToDestination(TreeNode node, String destination,
List<Train> result) {
        if (node == null) {
            return;
        }

        getTrainsToDestination(node.left, destination, result);

        if (node.train.destinationStation.equals(destination)) {
            result.add(node.train);
        }

        getTrainsToDestination(node.right, destination, result);
    }

    public List<Train> getTrainsToDestinationWithSeats(String destination,
int seats) {
        List<Train> trains = new ArrayList<>();
        getTrainsToDestinationWithSeats(root, destination, seats, trains);
        return trains;
    }

    private void getTrainsToDestinationWithSeats(TreeNode node, String
destination, int seats, List<Train> result) {
        if (node == null) {
            return;
        }

        getTrainsToDestinationWithSeats(node.left, destination, seats,
result);

        if (node.train.destinationStation.equals(destination) &&
node.train.availableSeats >= seats) {
            result.add(node.train);
        }

        getTrainsToDestinationWithSeats(node.right, destination, seats,
result);
    }

    public void showDepartureMessage() {
        showDepartureMessage(root);
    }

    private void showDepartureMessage(TreeNode node) {
        if (node == null) {
            return;
        }

        showDepartureMessage(node.left);

        long currentTimeMillis = System.currentTimeMillis();
        long timeDifference = node.train.departureTime.getTime() -
currentTimeMillis;
        long minutesDifference = timeDifference / (1000 * 60);
```

```
        if (minutesDifference == 10 || minutesDifference == 5 ||
minutesDifference == 3) {
            System.out.println("Train " + node.train.trainNumber + " to " +
node.train.destinationStation +
                    " will depart in " + minutesDifference + " minutes.");
        }

        showDepartureMessage(node.right);
    }
}

public class Main {
    public static void main(String[] args) {
        TrainSchedule schedule = new TrainSchedule();

        // Загрузка данных из файла
        loadScheduleFromFile(schedule, "schedule.txt");

        // Меню действий
        Scanner scanner = new Scanner(System.in);
        int choice;
        do {
            System.out.println("\n*** Train Schedule Menu ***");
            System.out.println("1. Print schedule");
            System.out.println("2. Find train by number");
            System.out.println("3. Find trains to destination after time");
            System.out.println("4. Find trains to destination");
            System.out.println("5. Find trains to destination with seats");
            System.out.println("6. Show departure message");
            System.out.println("7. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.println("\nTrain Schedule:");
                    schedule.printSchedule();
                    break;
                case 2:
                    System.out.print("Enter train number: ");
                    String trainNumber = scanner.nextLine();
                    Train train = schedule.findTrainByNumber(trainNumber);
                    if (train != null) {
                        System.out.println("Train found: " + train);
                    } else {
                        System.out.println("Train not found");
                    }
                    break;
                case 3:
                    System.out.print("Enter destination station: ");
                    String destination = scanner.nextLine();
                    System.out.print("Enter time (yyyy-MM-dd HH:mm): ");
                    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm");
                    try {
                        Date time = sdf.parse(scanner.nextLine());
                        List<Train> trains =
schedule.getTrainsToDestinationAfterTime(destination, time);
                        System.out.println("\nTrains to " + destination + "
after " + sdf.format(time) + ":");
                        for (Train t : trains) {
                            System.out.println(t);
```

```
                        }
                    } catch (ParseException e) {
                        System.out.println("Invalid date format");
                    }
                    break;
                case 4:
                    System.out.print("Enter destination station: ");
                    String destStation = scanner.nextLine();
                    List<Train> trainsToDestination =
schedule.getTrainsToDestination(destStation);
                    System.out.println("\nTrains to " + destStation + ":");
                    for (Train t : trainsToDestination) {
                        System.out.println(t);
                    }
                    break;
                case 5:
                    System.out.print("Enter destination station: ");
                    String dest = scanner.nextLine();
                    System.out.print("Enter number of seats: ");
                    int seats = scanner.nextInt();
                    List<Train> trainsWithSeats =
schedule.getTrainsToDestinationWithSeats(dest, seats);
                    System.out.println("\nTrains to " + dest + " with at
least " + seats + " seats:");
                    for (Train t : trainsWithSeats) {
                        System.out.println(t);
                    }
                    break;
                case 6:
                    System.out.println("\nDeparture Messages:");
                    schedule.showDepartureMessage();
                    break;
                case 7:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a number
between 1 and 7.");
            }
        } while (choice != 7);
    }

    public static void loadScheduleFromFile(TrainSchedule schedule, String
fileName) {
        File file = new File(fileName);
        try {
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split(",");
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm");
                Date departureTime = sdf.parse(parts[2]);
                Train train = new Train(parts[0], parts[1], departureTime,
Integer.parseInt(parts[3]));
                schedule.addTrain(train);
            }
            scanner.close();
        } catch (FileNotFoundException | ParseException e) {
            e.printStackTrace();
        }
    }
}
```

```
*** Train Schedule Menu ***
1. Print schedule
2. Find train by number
3. Find trains to destination after time
4. Find trains to destination
5. Find trains to destination with seats
6. Show departure message
7. Exit
Enter your choice: 1

Train Schedule:
Train{trainNumber='123', destinationStation='Station A', departureTime=Sun Apr 07 09:00:00 MSK 2024, availableSeats=100}
Train{trainNumber='456', destinationStation='Station B', departureTime=Sun Apr 07 10:30:00 MSK 2024, availableSeats=80}
Train{trainNumber='789', destinationStation='Station C', departureTime=Sun Apr 07 11:45:00 MSK 2024, availableSeats=120}
Train{trainNumber='234', destinationStation='Station A', departureTime=Sun Apr 07 12:15:00 MSK 2024, availableSeats=90}
Train{trainNumber='567', destinationStation='Station B', departureTime=Sun Apr 07 14:00:00 MSK 2024, availableSeats=110}
Train{trainNumber='890', destinationStation='Station C', departureTime=Sun Apr 07 15:30:00 MSK 2024, availableSeats=70}
```