Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Напредно програмирање

Примери од втор парцијален испит

Верзија 1.0, 13 Декември, 2016

Содржина

1. Најдобри филмови (2 парцијален испит 2014)

Да се имплементира класа `MoviesList` во која се чува листа од филмови (класа `Movie` за секој филм се дадени неговиот наслов и листа од рејтинзи (цели броеви од 1 до 10) и ги има следните методи:

- `public void addMovie(String title, int[] ratings)` - метод за додавање нов филм во листата (наслов и низа од рејтинзи)
- `public List<Movie> top10ByAvgRating()` - метод кој враќа листа од 10-те филмови со најдобар просечен рејтинг, подредени во опаѓачки редослед според рејтингот (ако два филмови имаат ист просечен рејтинг, се подредуваат лексикографски според името)
- `public List<Movie> top10ByRatingCoef()` - метод кој враќа листа од 10-те филмови со најдобар рејтинг коефициент (се пресметува како просечен ретјтинг на филмот `x` вкупно број на рејтинзи на филмот `/` максимален број на рејтинзи (од сите филмови во листата)

За класата `Movie` да се препокрие `toString()` методот да враќа соодветна репрезентација (погледнете го пример излезот).

```
package mk.ukim.finki.np.mt2;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

/**
 * 2 partial exam 2014
 */
public class MoviesTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        MoviesList moviesList = new MoviesList();
        int n = scanner.nextInt();
        scanner.nextLine();
        for (int i = 0; i < n; ++i) {
            String title = scanner.nextLine();
            int x = scanner.nextInt();
            int[] ratings = new int[x];
            for (int j = 0; j < x; ++j) {
                ratings[j] = scanner.nextInt();
            }
            scanner.nextLine();
            moviesList.addMovie(title, ratings);
        }
        scanner.close();
        List<Movie> movies = moviesList.top10ByAvgRating();
        System.out.println("=== TOP 10 BY AVERAGE RATING ===");
        for (Movie movie : movies) {
            System.out.println(movie);
```

```
        }
        movies = moviesList.top10ByRatingCoef();
        System.out.println("=== TOP 10 BY RATING COEFFICIENT ===");
        for (Movie movie : movies) {
            System.out.println(movie);
        }
    }
}

class MoviesList {
    List<Movie> movies;

    public MoviesList() {
        movies = new ArrayList<>();
    }

    public void addMovie(String title, int[] ratings) {
        Movie movie = new Movie(title, ratings);
        movies.add(movie);
    }

    public List<Movie> top10ByAvgRating() {
        return movies.stream()
                .sorted(Comparator.comparing(Movie::getAvgRating).reversed()
                        .thenComparing(Comparator.comparing(Movie::getTitle)))
                .limit(10)
                .collect(Collectors.toList());
    }

    public List<Movie> top10ByRatingCoef() {
        int maxRatings = movies.stream()
                .map(movie -> movie.ratings.size())
                .reduce(0, Math::max);
        return movies.stream()
                .sorted(new CoefRatingComparator(maxRatings))
                .limit(10)
                .collect(Collectors.toList());
    }
}

class CoefRatingComparator implements Comparator<Movie> {
    int maxRatings;

    public CoefRatingComparator(int maxRatings) {
        this.maxRatings = maxRatings;
    }

    @Override
    public int compare(Movie o1, Movie o2) {
        int ar = Double.compare(o1.avgRating * o1.ratings.size() / maxRatings,
                o2.avgRating * o2.ratings.size() / maxRatings);
        if (ar == 0) {
            return o1.title.compareTo(o2.title);
        }
        return -ar;
    }
}

class Movie {
    String title;
    List<Integer> ratings;
    double avgRating;

    public Movie(String title, int[] ratings) {
        this.title = title;
        this.ratings = IntStream.of(ratings)
                .boxed()
                .collect(Collectors.toList());
        avgRating = this.ratings.stream()
                .mapToDouble(Integer::doubleValue)
                .average().orElse(0);
    }

    public String getTitle() {
        return title;
    }
```

```
    public double getAvgRating() {
        return avgRating;
    }

    @Override
    public String toString() {
        return String.format("%s (%.2f) of %d ratings", title, avgRating, ratings.size());
    }
}
```

2. Мерна станица (2 парцијален испит 2013)

Во една метеролошка станица на секои 5 минути пристигнуваат податоци за временските услови (температура, влажност на воздухот, ветар, видливост, време). Пример за вакви податоци:

- температура: 13 степени
- влажност: 98%
- ветар: 11.2 km/h
- видливост: 14 km
- време: 28.12.2013 14:37:55 (dd.MM.yyyy HH:mm:ss).

Потребно е да се имплементира класа `WeatherStation` која ќе ги чува податоците за временските услови за последните `x` денови (при додавање на податоци за ново мерење, сите мерења чие што време е постаро за `x` денови од новото се бришат). Исто така ако времето на новото мерење кое се додава се разликува за помалку од 2.5 минути од времето на некое претходно додадено мерење, тоа треба да се игнорира (не се додава).

Да се имплементираат следните методи на класата `WeatherStation`:

- `WeatherStation(int days)` - конструктор со аргумент бројот на денови за кои се чуваат мерења
- `public void addMeasurement(float temperature, float wind, float humidity, float visibility, Date date)` - додавање на податоци за ново мерење
- `public int total()` - го враќа вкупниот број на мерења кои се чуваат
- `public void status(Date from, Date to)` - ги печати сите мерења во периодот од `from` до `to` подредени според датумот во растечки редослед и на

крај ја печати просечната температура во овој период. Ако не постојат мерења во овој период се фрла исклучок од тип `RuntimeException` (вграден во Јава).

Пример за форматот на излезот:

```
24.6 80.2 km/h 28.7% 51.7 km Tue Dec 17 23:40:15 CET 2013
23.5 32.2 km/h 16.5% 187.2 km Tue Dec 17 23:45:15 CET 2013
13.2 67.1 km/h 18.9% 135.4 km Tue Dec 17 23:50:15 CET 2013
Avarage temperature: 20.43
```

```
package mk.ukim.finki.np.mt2;

import java.text.ParseException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;
import java.util.TreeSet;
import java.util.function.Predicate;

public class WeatherStationTest {
    public static void main(String[] args) throws ParseException {
        Scanner scanner = new Scanner(System.in);
        DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss");
        int n = scanner.nextInt();
        scanner.nextLine();
        WeatherStation ws = new WeatherStation(n);
        while (true) {
            String line = scanner.nextLine();
            if (line.equals("=====")) {
                break;
            }
            String[] parts = line.split(" ");
            float temp = Float.parseFloat(parts[0]);
            float wind = Float.parseFloat(parts[1]);
            float hum = Float.parseFloat(parts[2]);
            float vis = Float.parseFloat(parts[3]);
            line = scanner.nextLine();
            LocalDateTime date = LocalDateTime.parse(line, timeFormatter);
            ws.addMeasurement(temp, wind, hum, vis, date);
        }
        String line = scanner.nextLine();
        LocalDateTime from = LocalDateTime.parse(line, timeFormatter);
        line = scanner.nextLine();
        LocalDateTime to = LocalDateTime.parse(line, timeFormatter);
        scanner.close();
        System.out.println(ws.total());
        try {
            ws.status(from, to);
        } catch (RuntimeException e) {
            System.out.println(e);
        }
    }
}

class WeatherStation {
    TreeSet<Measurement> measurments;

    int days;

    WeatherStation(int days) {
        this.days = days;
        measurments = new TreeSet<>();
    }

    public void addMeasurement(float temperature, float wind, float humidity,
```

```
                                float visibility, LocalDateTime date) {
        Measurement m = new Measurement(temperature, wind, humidity, visibility, date);
        if (!measurments.add(m)) {
            return;
        }

        Predicate<Measurement> old = measurement -> (measurement.date.minusDays(days)
.isAfter(date));
        measurments.removeIf(old);
    }

    public int total() {
        return measurments.size();
    }

    public void status(LocalDateTime from, LocalDateTime to) {
        Predicate<Measurement> isInRange = measurement -> !(measurement.date.isBefore(
from)
                || measurement.date.isAfter(to));

        double averageTemperature = measurments.stream()
                .filter(isInRange)
                .mapToDouble(measurement -> measurement.temperature)
                .average().orElse(0);
        measurments.stream()
                .filter(isInRange)
                .forEach(System.out::println);
        System.out.printf("Average temperature: %.2f\n", averageTemperature);
    }
}

class Measurement implements Comparable<Measurement> {
    float temperature;
    float wind;
    float humidity;
    float visibility;
    LocalDateTime date;

    public Measurement(float temperature, float wind, float humidity,
                       float visibility, LocalDateTime date) {
        this.temperature = temperature;
        this.wind = wind;
        this.humidity = humidity;
        this.visibility = visibility;
        this.date = date;
    }

    @Override
    public int compareTo(Measurement other) {
        long span = Math.abs(other.date.until(date, ChronoUnit.SECONDS));
        if (span < 150) {
            return 0;
        } else return date.compareTo(other.date);
    }

    @Override
    public String toString() {
        return String.format("%.1f %.1f km/h %.1f%% %.1f km %s", temperature,
                wind, humidity, visibility, date);
    }

}
```

3. Stadium (2 парцијален испит 2014)

Да се имплементира систем за билети за стадион. За таа цел треба да се имплементираат класите:

1. `Sector` во која се чуваат информации за:
   - кодот на секторот `String`
   - бројот на места за седење `int`
   - информации за зафатеност на местата за седење ?
2. `Stadium` во која се чуваат информации за:
   - името на стадионот `String`
   - и сите сектори во стадионот ?

Во класата `Stadium` треба да се имплементираат следните методи:

- `Stadium(String name)` конструктор со аргумент име на стадионот
- `void createSectors(String[] sectorNames, int[] sizes)` креирање на сектори со имиња `String[] sectorNames` и број на места `int[] sizes` (двете низи се со иста големина)
- `void buyTicket(String sectorName, int seat, int type)` за купување билет од проследениот тип (`type`, 0 - неутрален, 1 - домашен, 2 - гостински), во секторот `sectorName` со број на место `seat` (местото секогаш е со вредност во опсег `1 - size`). Ако местото е зафатено (претходно е купен билет на ова место) се фрла исклучок од вид `SeatTakenException`. Исто така ако се обидеме да купиме билет од тип 1, во сектор во кој веќе има купено билет од тип 2 (и обратно) се фрла исклучок од вид `SeatNotAllowedException`.
- `void showSectors()` ги печати сите сектори сортирани според бројот на слободни места во опаѓачки редослед (ако повеќе сектори имаат ист број на слободни места, се подредуваат според името).

```
package mk.ukim.finki.np.mt2;

import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Scanner;


public class StadiumTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine();
        String[] sectorNames = new String[n];
        int[] sectorSizes = new int[n];
        String name = scanner.nextLine();
```

```
        for (int i = 0; i < n; ++i) {
            String line = scanner.nextLine();
            String[] parts = line.split(";");
            sectorNames[i] = parts[0];
            sectorSizes[i] = Integer.parseInt(parts[1]);
        }
        Stadium stadium = new Stadium(name);
        stadium.createSectors(sectorNames, sectorSizes);
        n = scanner.nextInt();
        scanner.nextLine();
        for (int i = 0; i < n; ++i) {
            String line = scanner.nextLine();
            String[] parts = line.split(";");
            try {
                stadium.buyTicket(parts[0], Integer.parseInt(parts[1]),
                        Integer.parseInt(parts[2]));
            } catch (SeatNotAllowedException e) {
                System.out.println("SeatNotAllowedException");
            } catch (SeatTakenException e) {
                System.out.println("SeatTakenException");
            }
        }
        stadium.showSectors();
    }
}

class Stadium {
    String name;
    HashMap<String, Sector> sectors;

    public Stadium(String name) {
        this.name = name;
        sectors = new HashMap<>();
    }

    public void createSectors(String[] sectorNames, int[] sectorSizes) {
        for (int i = 0; i < sectorNames.length; ++i) {
            addSector(sectorNames[i], sectorSizes[i]);
        }
    }

    void addSector(String name, int size) {
        Sector sector = new Sector(name, size);
        sectors.put(name, sector);
    }

    public void buyTicket(String sectorName, int seat, int type)
            throws SeatNotAllowedException, SeatTakenException {
        Sector sector = sectors.get(sectorName);
        if (sector.isTaken(seat))
            throw new SeatTakenException();
        sector.takeSeat(seat, type);
    }

    public void showSectors() {
        sectors.values().stream()
                .sorted(Comparator.comparing(Sector::free)
                        .thenComparing(Sector::getName))
                .forEach(System.out::println);
    }

}

class Sector {
    String name;
    int size;
    HashMap<Integer, Integer> taken;
    HashSet<Integer> types;

    public Sector(String name, int size) {
        this.name = name;
        this.size = size;
        taken = new HashMap<>();
        types = new HashSet<>();
    }
```

```
    public String getName() {
        return name;
    }

    int free() {
        return size - taken.size();
    }

    public void takeSeat(int seat, int type) throws SeatNotAllowedException {
        if (type == 1) {
            if (types.contains(2))
                throw new SeatNotAllowedException();
        } else if (type == 2) {
            if (types.contains(1))
                throw new SeatNotAllowedException();
        }
        types.add(type);
        taken.put(seat, type);
    }

    public boolean isTaken(int seat) {
        return taken.containsKey(seat);
    }

    @Override
    public String toString() {
        return String.format("%s\t%d/%d\t%.1f%%", name, free(), size, (size - free()) *
100.0 / size);
    }

}

class SeatNotAllowedException extends Exception {

}

class SeatTakenException extends Exception {

}
```

4. Изворен код од примери и задачи

https://github.com/finki-mk/NP/

Source Code ZIP