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

Напредно програмирање

Примери од прв парцијален испит

Верзија 1.0, 20 Септември, 2016

Содржина

1. Time Table (1 парцијален испит 2014)

Да се имплементира класа `TimeTable` која ќе чита од влезен тек (стандарден влез, датотека, ...) податоци за времиња во 24-часовен формат. Сите времиња се разделени со едно празно место, а во самото време часот и минутите може да бидат разделени со `:` или `.`. Пример за форматот на податоците:

```
11:15 0.45 23:12 15:29 18.46
```

Ваша задача е да ги имплементирате методите:

- `TimeTable()` - default конструктор
- `void readTimes(InputStream inputStream)` - метод за читање на податоците
- `void writeTimes(OutputStream outputStream, TimeFormat format)` - метод кој ги печати сите времиња сортирани во растечки редослед во зададениот формат (24 часовен или AM/PM).

Методот за читање `readTimes` фрла исклучоци од тип `UnsupportedFormatException` ако времињата се разделени со нешто друго што не е `:` или `.` и `InvalidTimeException` ако времето (часот или минутите) е надвор од дозволениот опсег (0-23, 0-59). И двата исклучоци во пораката `getMessage()` треба да го вратат влезниот податок кој го предизвикал исклучокот. Сите времиња до моментот кога ќе се фрли некој од овие два исклучоци треба да си останат вчитани. Правила за конверзија од 24-часовен формат во AM/PM:

1. за првиот час од денот (0:00 - 0:59), додадете 12 и направете го "AM"
2. од 1:00 до 11:59, само направето го "AM"
3. од 12:00 до 12:59, само направето го "PM"
4. од 13:00 до 23:59 одземете 12 и направете го "PM"

```
package mk.ukim.finki.np.mt1;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Comparator;
```

```
import java.util.List;
import java.util.Scanner;
import java.util.function.Function;
import java.util.stream.Collectors;

public class TimesTest {

    public static void main(String[] args) {
        TimeTable timeTable = new TimeTable();
        try {
            timeTable.readTimes(System.in);
        } catch (UnsupportedFormatException e) {
            System.out.println("UnsupportedFormatException: " + e.getMessage());
        } catch (InvalidTimeException e) {
            System.out.println("InvalidTimeException: " + e.getMessage());
        }
        System.out.println("24 HOUR FORMAT");
        timeTable.writeTimes(System.out, TimeFormat.FORMAT_24);
        System.out.println("AM/PM FORMAT");
        timeTable.writeTimes(System.out, TimeFormat.FORMAT_AMPM);
    }

}

enum TimeFormat {
    FORMAT_24, FORMAT_AMPM
}

class UnsupportedFormatException extends Exception {
    public UnsupportedFormatException(String msg) {
        super(msg);
    }
}

class InvalidTimeException extends Exception {
    public InvalidTimeException(String msg) {
        super(msg);
    }
}

class TimeTable {
    List<Time> times;

    public TimeTable() {
        times = new ArrayList<>();
    }

    public void readTimes(InputStream input) throws UnsupportedFormatException,
            InvalidTimeException {
        Scanner scanner = new Scanner(input);
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split("\\s+");
            for (String p : parts) {
                Time time = new Time(p);
                times.add(time);
            }
        }
    }

    public void writeTimes(OutputStream output, TimeFormat format) {
        PrintWriter writer = new PrintWriter(output);
        final Function<Time, String> toString = time -> {
            if (format.equals(TimeFormat.FORMAT_24)) return time.toString();
            else return time.toStringAMPM();
        };
        List<String> timesToPrint = times.stream().sorted(Comparator.comparing(Time:
:getHour)
                .thenComparing(Time::getMinute))
                .map(toString)
                .collect(Collectors.toList());
        timesToPrint.forEach(writer::println);
        writer.flush();
    }
}
```

```
class Time {
    private final int hour;
    private final int minute;

    public Time(String time) throws UnsupportedFormatException,
            InvalidTimeException {
        String[] parts = time.split("\\.");
        if (parts.length == 1) {
            parts = time.split(":");
        }
        if (parts.length == 1)
            throw new UnsupportedFormatException(time);
        this.hour = Integer.parseInt(parts[0]);
        this.minute = Integer.parseInt(parts[1]);
        if (hour < 0 || hour > 23 || minute < 0 || minute > 59)
            throw new InvalidTimeException(time);
    }

    public String toStringAMPM() {
        String part = "AM";
        int h = hour;
        if (h == 0) {
            h += 12;
        } else if (h == 12) {
            part = "PM";
        } else if (h > 12) {
            h -= 12;
            part = "PM";
        }
        return String.format("%2d:%02d %s", h, minute, part);
    }

    public int getHour() {
        return hour;
    }

    public int getMinute() {
        return minute;
    }

    @Override
    public String toString() {
        return String.format("%2d:%02d", hour, minute);
    }

}
```

2. MinMax (1 парцијален испит 2014)

Да се имплемнтира генеричка класа `MinMax` од два споредливи објекти (минимум/максимум). За класата да се имплементираат:

- `MinMax()` - default конструктор
- `void update(T element)` - метод за ажурирање на тековните минимум/максимум
- `T max()` - го враќа најголемиот елемент
- `T min()` - го враќа најмалиот елемент
- да се преоптовари методот `toString()` кој враќа стринг составен од минималниот и максималниот елемент и бројот на елементи обработени во

методот `update` кои се различни од тековниот минимум/максимум, разделени со празно место.

Во класата не смеат да се чуваат елементите кои се обработуваат во методот `update`, освен тековниот минимум/максимум.

```
package mk.ukim.finki.np.mt1;

public class MinAndMax {
    public static void main(String[] args) throws ClassNotFoundException {
        MinMax<String> strings = new MinMax<>();
        strings.add("b");
        strings.add("b");
        strings.add("a");

        strings.add("a");
        strings.add("a");
        strings.add("c");
        strings.add("c");
        strings.add("c");
        strings.add("d");
        System.out.println(strings);
    }
}

class MinMax<T extends Comparable<T>> {
    T min;
    T max;
    int total;
    int minCount;
    int maxCount;

    public MinMax() {
        total = 0;
        minCount = 0;
        maxCount = 0;
    }

    public void add(T element) {
        if (total == 0) {
            min = element;
            max = element;
        }
        ++total;
        if (element.compareTo(min) < 0) {
            minCount = 1;
            min = element;
        } else {
            if (element.compareTo(min) == 0) {
                minCount++;
            }
        }
        if (element.compareTo(max) > 0) {
            maxCount = 1;
            max = element;
        } else {
            if (element.compareTo(max) == 0) {
                maxCount++;
            }
        }
    }

    public T min() {
        return min;
    }

    public T max() {
```

```
            return max;
        }

        @Override
        public String toString() {
            StringBuilder sb = new StringBuilder();
            T min = min();
            T max = max();
            sb.append(String.format("%s %s %d\n", min, max, total
                    - (maxCount + minCount)));
            return sb.toString();
        }

    }
```

3. Stacked Canvas

Да се имплементира класа `Canvas` на која ќе чуваат различни форми. За секоја форма се чува:

- `id:String`
- `color:Color` (enum дадена)

Притоа сите форми треба да имплментираат два интерфејси:

- `Scalable` - дефиниран со еден метод `void scale(float scaleFactor)` за соодветно зголемување/намалување на формата за дадениот фактор
- `Stackable` - дефиниран со еден метод `float weight()` кој враќа тежината на формата (се пресметува како плоштина на соодветната форма)

Во класата `Canvas` да се имплементираат следните методи:

- `void add(String id, Color color, float radius)` за додавање круг
- `void add(String id, Color color, float width, float height)` за додавање правоаголник

При додавањето на нова форма, во листата со форми таа треба да се смести на соодветното место според нејзината тежина. Елементите постојано се подредени според тежината во опаѓачки редослед.

- `void scale(String id, float scaleFactor)` - метод кој ја скалира формата со даденото id за соодветниот scaleFactor. Притоа ако има потреба, треба да се изврши преместување на соодветните форми, за да се задржи подреденоста на елементите.

Не смее да се користи сортирање на листата.

- `toString()` - враќа стринг составен од сите фигури во нов ред. За секоја фигура се додава:
  - `C: [id:5 места од лево] [color:10 места од десно] [weight:10.2 места од десно]` ако е круг
  - `R: [id:5 места од лево] [color:10 места од десно] [weight:10.2 места од десно]` ако е правоаголник

Користење на `instanceof` ќе се смета за неточно решение

4. Изворен код од примери и задачи

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

Source Code ZIP