OPRPP 1 - 2022./2023. - LJIR

===== TEORIJA =====

1. Zadani su razredi R i S, te metoda main u nekom trećem razredu u istom paketu. Detaljno nacrtajte stanje u memoriji nakon izvođenja zadnje naredbe u metodi main.

```
class R {
	String a;
	int b;
	R(String a, int b) {
		this.a = a;
		this.b = b;
	}
}
class S {
	R r;
	int c;
	R(R r, int c) {
		this.r = r;
		this.c = c;
	}
}
public static void main(String …args) {
	R r = new R(„aaa“, 5);
	S[] s = {new S(r, 12), new S(new R(„bbb“, 7), 14)};
	S s = ss[1];
}
```

Na stogu (Stack): U programskom jeziku Java, lokalne varijable i reference na objekte nalaze se na stogu. Varijable na stogu su brže za pristup jer je stog strukturiran tako da uvijek daje brzi pristup vrhu stoga. U našem slučaju, na stogu bi se našle sljedeće varijable:

- Referenca r koja pokazuje na objekt klase R u hrpi.
- Referenca ss koja pokazuje na polje objekata klase S u hrpi.
- Referenca s1 koja pokazuje na drugi objekt u polju ss u hrpi.

Na hrpi (Heap): U hrpi se nalaze svi objekti. Svaki put kada se koristi new za stvaranje objekta, memorija se alocira na hrpi. U ovom slučaju, na hrpi bismo imali:

- Objekt r klase R s vrijednostima a = "aaa" i b = 5.
- Polje ss koje sadrži dva objekta klase S.
- Prvi objekt u polju ss ima referencu na objekt r i vrijednost c = 12.
- Drugi objekt u polju ss ima svoj vlastiti objekt klase R s vrijednostima a = "bbb" i b = 7, te vrijednost c = 14.

2. Potrebno je ostvariti neki posao na temelju datoteka koje se nalaze u nekom podstablu direktorija. Objasnite kako to možemo (najsmislenije) napraviti koristeći paket java.io, a kako koristeći paket java.nio.file? Koja je prednost drugog načina?

Java.io: Metoda listFilesRecursively prima objekt File koji predstavlja direktorij. Koristi se metoda listFiles za dobivanje svih datoteka (i direktorija) unutar zadanog direktorija. Za svaku datoteku, ako je direktorij, rekurzivno se poziva listFilesRecursively; inače se obavlja posao na datoteci.

Java.nio.file: Metoda listFilesRecursively koristi metodu Files.walkFileTree za obilazak datotečnog stabla počevši od zadane putanje. SimpleFileVisitor je korisna apstraktna klasa koja omogućuje lako definiranje ponašanja pri posjeti datoteci ili direktoriju.

Prednosti java.nio.file nad java.io:

- java.nio.file pruža bolje performanse kroz bolje korištenje sustavskih IO mogućnosti.
- java.nio.file pruža bolju kontrolu nad datotekama i direktorijima, uključujući simboličke poveznice, datotečne dozvole i druge atribute datoteka.

By: CrazyFreak (& ChatGPT)

- java.nio.file omogućuje jednostavnije i učinkovitije obilazak direktorija pomoću Files.walkFileTree metode, dok java.io zahtijeva rekurzivni obilazak.
- java.nio.file ima bolju podršku za rad s velikim datotekama, zahvaljujući klasama poput FileChannel i MappedByteBuffer.

Ilustrirajte to Java pseudokodom na nekom primjeru.

Java.io:

```
public void listFilesRecursively(File dir) {
    File[] files = dir.listFiles();
    if (files != null) {  // Direktorij može biti prazan, što bi rezultiralo null poljem.
        for (File file : files) {
            if (file.isDirectory()) {
                listFilesRecursively(file);
            } else {
                // Ovdje se obavlja posao na datoteci.
                System.out.println(file.getName());
            }
        }
    }
}
```

Java.nio.file:

```
public void listFilesRecursively(Path startPath) throws IOException {
    Files.walkFileTree(startPath, new SimpleFileVisitor<Path>() {
        @Override
        public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
            // Ovdje se obavlja posao na datoteci.
            System.out.println(file.getFileName());
            return FileVisitResult.CONTINUE;
        }
    });
}
```

3. Imamo razred Par i sljedeći kod:

```
class Par {
       String s1;
       String s2;
       String min() {…}
       String max() {…}
       Par(String s1, String s2) {
              this.s1 = s2;
              this.s1 = s2;
       }
}
public static void main(String …args) {
       Par p = new Par („Zagreb“, „Krapina“);
       String manji = p.min(); // Krapina
       String veci = p.max(); // Zagreb
}
```

U ovom primjeru metoda min će vratiti Krapina jer se stringovi prirodno uspoređuju leksikografski pa je Krapina „manja“ od Zagreba.

Proširite razred Par tako da može reprezentirati i parove drugih vrsta podataka (ne samo stringove), a proširenje treba podržavati tipsku korektnost pri prevođenju programa. Napišite novu implementaciju tog razreda te primjer kako bismo napravili par brojeva 7 i 4.

```
class Par<T extends Comparable<T>> {
    T prvi;
    T drugi;

    Par(T prvi, T drugi) {
        this.prvi = prvi;
```

```
        this.drugi = drugi;
    }

    T min() {
        if (prvi.compareTo(drugi) < 0) {
            return prvi;
        } else {
            return drugi;
        }
    }

    T max() {
        if (prvi.compareTo(drugi) > 0) {
            return prvi;
        } else {
            return drugi;
        }
    }
}
public static void main(String …args) {
       Par<Integer> par = new Par<>(7, 4);
       int manji = par.min(); // 4
       int veci = par.max(); // 7
}
```

4. Pretpostavite da imamo polje arr koje sadrži objekte tipa S iz prvog zadatka. Koristeći tokovni API:

a) napišite naredbu koja će vratiti sumu svih b-ova

```
Arrays.stream(arr).mapToInt(s -> s.r.b).sum();
```

b) napišite naredbu koja će vratiti sumu svih a-ova koji počinju slovom „P“

```
Arrays.stream(arr).filter(s -> s.r.a.startsWith("P")).map(s -> s.r.a).collect(Collectors.joining());
```

c) objasnite razliku između metoda .map i .flatmap te dajte ilustrativni primjer

map je metoda koja primjenjuje funkciju na svaki element toka i rezultira tokom koji se sastoji od rezultata tih funkcija. flatMap je sličan, ali očekuje da će funkcija za svaki element toka vratiti tok, a zatim "spljošti" sve te tokove u jedan tok.

Primjer korištenja map:

```
List<String> words = Arrays.asList("Hello", "World");
List<Integer> wordLengths = words.stream().map(String::length).collect(Collectors.toList());
```

Ovaj kod transformira tok riječi u tok duljina tih riječi.

Primjer korištenja flatMap:

```
List<String> sentences = Arrays.asList("Hello World", "Java Stream API");
List<String>   words   =   sentences.stream().flatMap(sentence   ->   Arrays.stream(sentence.split("
")));.collect(Collectors.toList());
```

Ovaj kod transformira tok rečenica u tok riječi. Za svaku rečenicu, Arrays.stream(sentence.split(" ")) vraća tok riječi, a flatMap spljošti te tokove u jedan tok.

5. Paralelno želimo pokrenuti dva posla i svaki od njih puni listu s 10 slučajno generiranih brojeva. U metodi main potom želimo ispisati ukupnu sumu svih brojeva. Napišite sav potreban kod.

```
import java.util.*;

public class Main {
    static List<Integer> list1 = new ArrayList<>();
    static List<Integer> list2 = new ArrayList<>();

    static class RandomNumbersRunnable implements Runnable {
        private final List<Integer> list;
```

```
        RandomNumbersRunnable(List<Integer> list) {
            this.list = list;
        }

        @Override
        public void run() {
            Random random = new Random();
            for (int i = 0; i < 10; i++) {
                synchronized(list) {
                    list.add(random.nextInt(100)); // generiramo brojeve od 0 do 99
                }
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Thread thread1 = new Thread(new RandomNumbersRunnable(list1));
        Thread thread2 = new Thread(new RandomNumbersRunnable(list2));

        thread1.start();
        thread2.start();

        thread1.join();
        thread2.join();

        int sum = 0;
        for (int num : list1) {
            sum += num;
        }
        for (int num : list2) {
            sum += num;
        }

        System.out.println("Ukupna suma: " + sum);
    }
}
```

===== ZADACI =====

1. Informacije o pjevačima pop glazbe čuvaju se u tekstovnoj datoteci pjevaci.txt. Jedan zapis proteže se u jednom retku; elementi su razdvojeni tabovima, a o svakom pjevaču pamti se njegovo ime, trenutna ocjena popularnosti (od 1 do 5) te je li pobijedio na Euroviziji ili nije. Primjer:

Janko 3.5 true
Vesna 4.2 false
Ante 4.3 false
Ana 3.6 true

Napravite razred Pjevac kojim modelirate jednog pjevača. Potom napravite razred BazaPjevaca kojim modelirate bazu pjevača.
Razred treba omogućiti stvaranje prazne baze, zatim učitavanje sadržaja baze na temelju staze do datoteke iz koje će pročitati trenutno stanje (ili bazu ostaviti praznom ako datoteka ne postoji), spremanje sadržaja baze natrag u datoteku te dodavanje novih zapisa.
Kako radimo s pojednostavljenim zapisima, ništa nije jedinstveno, tj. može postojati više pjevača s istim imenom, no tretiramo ih kao različite pjevače.

Napravite grafičku aplikaciju. Ista treba ponuditi izbornik File -> Load i File -> Save.
File -> Load korisnika pita da odabere datoteku iz koje će učitati sadržaj baze i potom ga učitava.
File -> Save pohranjuje bazu u zadnju datoteku koja je bila otvorena za čitanje, a ako se ništa nije prethodno učitivalo, tada se korisnika pita gdje da pohrani sadržaj.

Aplikacija treba imati gumb Dodaj na čiji će se klik otvoriti dijalog u koji korisnik može unijeti sve podatke o jednom pjevaču, i koja će potom stvoriti jedan primjerak razreda Pjevac te pozvati baza.dodaj(pjevac).

Aplikacija treba korisniku u lijevoj polovici prozora prikazati listu svih imena pjevača, a u desnoj polovici prozora treba prikazati listu stavki gdje svaka stvaka prikazuje informacije o pjevaču u formatu "ime (ocjena, pobijedio)" gdje vrijednost "pobijedio" može biti "DA" ili "NE", npr. za Janka bi to bilo "Janko (3.5, DA)".

Napravite vlastiti upravljač razmještajem koji će osigurati poboljšano razmještanje lijeve i desne liste na način da se obje protežu po čitavoj raspoloživoj visini. Lijeva strana treba uvijek biti širka 200 piksela, dok desna treba zauzeti čitav preostali prostor.