Objektno orijentirano programiranje

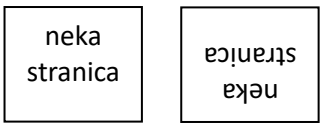
Ispitni rok

4.9.2020.

Ispit nosi ukupno 50 bodova i piše se 150 minuta.
U zadacima nije potrebno pisati dio u kojem se uključuju klase ili paketi klasa (import).
***Rješenja je potrebno pisati isključivo na predviđena mjesta**. Pišite čitko i uredno. Nije dozvoljeno šarati po QR kodovima. **Obavezno treba predati sve stranice ispita.** One stranice koje nisu predane neće biti ocijenjene.* Na predzadnjoj stranici pronađite podsjetnik za odabrane dijelove gradiva. Zadnju praznu stranicu možete koristiti za skiciranje rješenja te se ona ni u kojem slučaju neće ocjenjivati. Ispiti će se skenirati pa obratite pažnju da koristite olovku koja ostavlja tamniji trag.

UPUTE ZA PREDAJU:

- provjeriti jesu li svi papiri na broju
- provjeriti je li potpisana izjava
- urediti papire da svi budu „normalno“ rotirani (drugim riječima, nemojte dozvoliti da imate papir „naglavačke“)



OK LOŠE

- sortirati ih tako da prva stranica bude na vrhu

IZJAVA

Tijekom ove provjere znanja neću od drugoga primiti niti drugome pružiti pomoć te se neću koristiti nedopuštenim sredstvima.
Ove su radnje povreda Kodeksa ponašanja te mogu uzrokovati trajno isključenje s Fakulteta.
Zdravstveno stanje dozvoljava mi pisanje ovog ispita.

Vlastoručni potpis studenta: _____

1. zadatak (10 bodova)

Potrebno je modelirati pojednostavljeni sustav za evidenciju proizvoda slastičarnice. **Potrebno je nadopuniti predložak UML dijagrama klasa ovog sustava koji se nalazi na sljedećoj stranici**. Pritom **možete dodati nove klase** (ili sučelja) ako smatrate da je to potrebno**. Programski kod nije potrebno pisati**. Na ovom predlošku pažljivo navedite:

- oznake za sučelje (**I**), enumeraciju (**E**), apstraktnu (**A**) ili običnu klasu (**C**)
- oznake za apstraktnu metodu (**A**) te oznake za statičku (**S**) i finalnu (**F**) metodu ili atribut
- modifikatore vidljivosti metoda i atributa (**+** public, **#** protected, **-** private, **~** package-private)
- tipove povratnih vrijednosti i argumenata za metode te tipove atributa
- odgovarajuće strelice da naznačite nasljeđivanje klasa (puna linija ⟶▷) ili implementaciju sučelja (iscrtkana linija ⇢▷)

Svaki proizvod `Item` opisuju ime `name` (String), opis `description` (String), cijena `price` (double) i rok trajanja `expDate` (LocalDate). Za sve atribute potrebno je napisati odgovarajuće gettere i settere. Nije moguće stvoriti objekt tipa Item. Konkretni proizvodi koje je moguće stvoriti su kolač `Cake`, sladoled `Icecream` i slatka košarica `SweetBox` (kutija koja sadrži proizvoljan broj proizvoda). Kolač je opisan načinom pripreme `Preparation` (`BAKED`, `COOKED`, `RAW`). Sladoled opisuje temperatura na kojoj se skladišti `storageTemp` (double), a ima i dvije izdvojene podvrste: čokoladni `ChocolateIcream` i voćni `FruitIcecream`. Čokoladni sladoled sadrži informaciju o udjelu čokolade `chocolatePercentage` (short), kao i tip korištene čokolade `chocolateType` (`WHITE`, `MILK`, `DARK`). Voćni sladoled sadrži informaciju o vrsti voća `fruitType` (String).

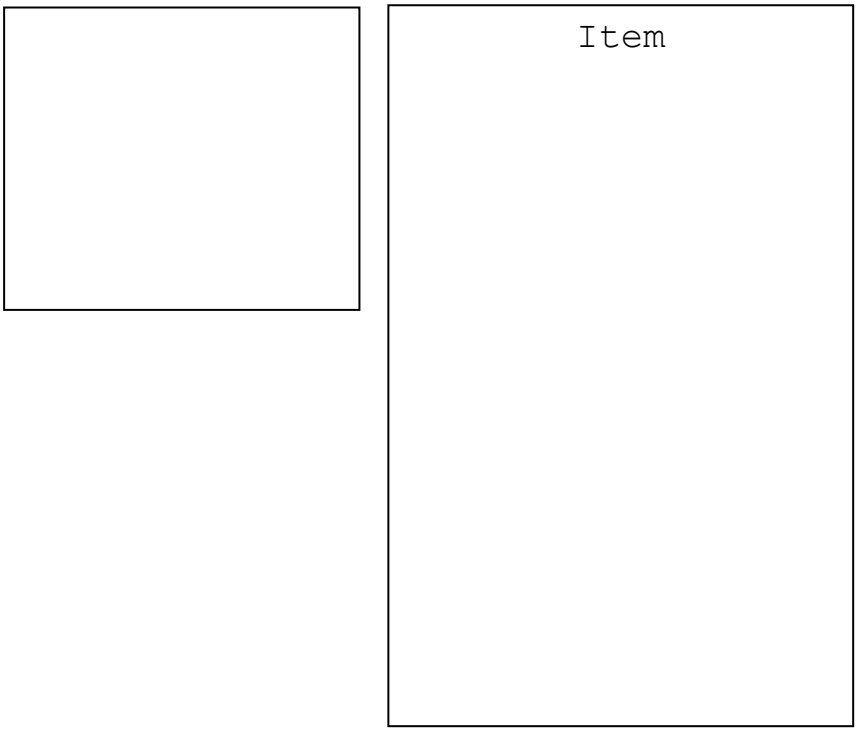
Slatka košarica sadrži metodu `addItems`. Kod metode `addItems` je sljedeći:

```
public void addItems(Item ... items) {
      this.items.addAll(Arrays.asList(items));
}
```

Nadalje, cijena košarice je jednaka sumi cijena svih proizvoda koji se trenutno nalaze u košarici, a rok trajanja košarice je rok trajanja najkvarljivijeg proizvoda u košarici.

Kolači, voćni sladoled i slatka košarica spadaju u dijetalne proizvode `DietProduct`, tj. implementiraju metode getN`oOfCalories()` (vraća int) i `isSugarFree()` (vraća boolean).

Svi **proizvodi** se mogu **uspoređivati**, a dva su proizvoda jednaka ako imaju isti **naziv i cijenu**. Iznimno, čokoladni sladoledi su jednaki ako sadrže i isti tip čokolade.


Item

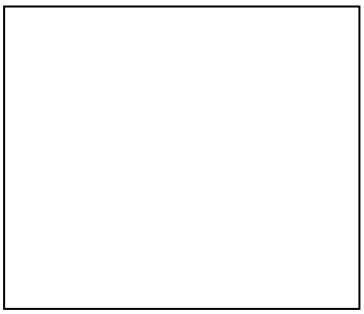
2. zadatak (10 bodova)

Potrebno je implementirati klasu `CustomFileVisitor` koja nasljeđuje `SimpleFileVisitor<Path>` i to na način da njene javne metode po izvođenju metode `Files.walkFileTree` ostvaruju sljedeće zadaće:

- `public Path getMaxFilePath()` - vraća putanju do najveće datoteke;
- `public long getMaxFileSize()` - *vraća veličinu najveće datoteke;*
- `public Path getMaxFilesDirPath()` - *vraća putanju do direktorija koji sadrži najveći broj datoteka u sebi.*

U slučaju da dvije datoteke imaju istu veličinu ili ako dva direktorija sadrže jednak broj datoteka, metoda treba vratiti onaj rezultat koji je abecedno prvi po redu (npr. ako direktoriji "aa" i "bb" imaju jednak broj datoteka, metoda treba vratiti putanju do direktorija "aa").

```
public class CustomFileVisitor extends SimpleFileVisitor<Path> {

	private Path maxFilePath, maxFilesPerDirPath;
	private long maxFileSize = -1;
	private int maxFilesPerDir = 0;

	public Path getMaxFilePath() {
		return maxFilePath;
	}

	public long getMaxFileSize() {
		return maxFileSize;
	}

	public Path getMaxFilesDirPath() {
		return maxFilesPerDirPath;
	}

	@Override
	public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs) {









		return FileVisitResult.CONTINUE;
	}

	@Override
	public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {









		return FileVisitResult.CONTINUE;
	}
}
```



Također, potrebno je implementirati i drugu klasu `CustomFileVisitor2` koja nasljeđuje `SimpleFileVisitor<Path>` a ostvaruje sljedeću funkcionalnost:

- ispisuje posjećenu putanju i to od najdublje razine prema višoj,
- u slučaju da je posjećen direktorij koji u imenu sadrži riječ "`dump`"

potrebno je preskočiti sve datoteke koje se nalaze u njemu.

Dakle, ako je zadan početni direktorij sa sljedećom strukturom:

```
folder-------aa--------a.png
                |--------b.png
                |--------dump--------s.png
                                |----------z.png
        |--------an------c.png
                |-------d.png
```

Ispis po pozivu metode `walkFilesTree` treba biti sljedeći:

/folder/aa/b.png
/folder/aa/a.png
/folder/aa
/folder/an/c.png
/folder/an/d.png
/folder/an
/folder

```
public class CustomFileVisitor2 extends SimpleFileVisitor<Path>{

	@Override
	public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs)
	throws IOException {
		if(_____________________________)return FileVisitResult.SKIP_SUBTREE;



	}

	@Override
	public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
	throws IOException {







	}

	@Override
	public FileVisitResult postVisitDirectory(Path dir, IOException exc)
	throws IOException {







	}
}
```

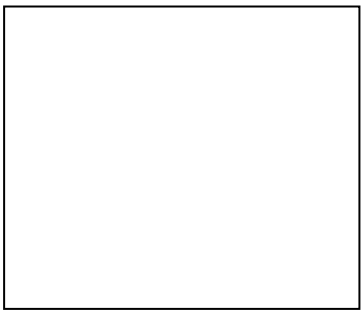
3. zadatak (10 bodova)

Postoji klasa **`Drink`** koja predstavlja piće u nekom baru. Klasa sadrži atribute (i odgovarajuće gettere i settere) `name` (`String`) koji predstavlja ime pića, `amountInMl` (`int`) koji predstavlja koliko ml tekućine to piće sadrži (npr. 330 ml) te `containsAlcohol` (`boolean`) koji predstavlja informaciju sadrži li to piće alkohol. Klasom **`Bar`** modeliran je bar. Klasa bar sadrži mapu `Map<Drink, Double> priceList` koja predstavlja cjenik tj. za svako piće govori kolika mu je cijena. Također u klasi `Bar` nalazi se i mapa `Map<Drink,Integer> drinksSold` koja za svako pojedino piće sadrži podatak o broju prodanih primjeraka tog pića. Obje mape imaju odgovarajuće gettere i settere. Klasa sadrži još dvije pomoćne `metode public void loadDrink(Drink d, Double price)` za dodavanje pića u cjenik i `public void buyDrink(Drink d, int amount)` za dodavanje pića u mapu prodanih.

Koristeći kolekcijske tokove, potrebno je napisati:

- metodu `long getNumberOfAlcoholicDrinks(Bar bar)` koja vraća broj alkoholnih pića koja se prodaju u nekom baru;
- metodu `double getDailyReceipts(Bar bar)` koja vraća ukupnu količinu novaca koja je taj dan zarađena u baru od prodaje pića. (Pretpostavite da se na dnevnoj bazi svi brojači resetiraju)
- metodu `Map<String,Double> getPricePerMlOfDrink(Bar bar)` koja vraća mapu čiji je ključ ime pića a vrijednost je cijena **jednog** mililitra pića.

```
public class Main {

	public static void main(String[] args) {

		Drink cola = new Drink("cola", 250, false);
		Drink juice = new Drink("juice", 330, false);
		Drink coffee = new Drink("coffee", 100, false);
		Drink irishCoffee = new Drink("Irish coffee", 150, true);
		Drink wine = new Drink("wine", 100, true);
		Drink beer = new Drink("beer", 500, true);

		Bar bar = new Bar();
		bar.loadDrink(cola, 15.00);
		bar.loadDrink(juice, 16.50);
		bar.loadDrink(coffee, 9.00);
		bar.loadDrink(irishCoffee, 13.00);
		bar.loadDrink(wine, 15.50);
		bar.loadDrink(beer, 18.00);

		bar.buyDrink(cola, 2);
		bar.buyDrink(beer, 10);

		System.out.println(getNumberOfAlcoholicDrinks(bar));

		//prints: 3



	Map<String, Double> map = getPricePerMlOfDrink(bar);
		for (String s : map.keySet()) {
		System.out.println(s + " : " + map.get(s));
		}
	//prints:
		//coffee : 0.09
		//juice : 0.05
```



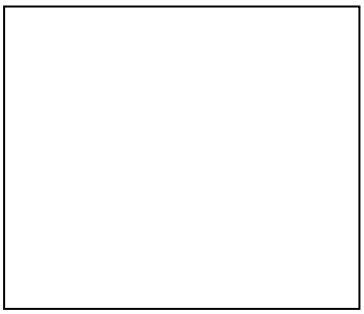
```
        //Irish coffee : 0.08666666666666667
        //beer : 0.036
        //wine : 0.155
        //cola : 0.06

            System.out.println(getDailyReceipts(bar));

            //prints 210.0

    }

    public static long getNumberOfAlcoholicDrinks(Bar bar) {

        return bar.









    }

    public static double getDailyReceipts(Bar bar) {

        return bar.









    }

    public static Map<String, Double> getPricePerMlOfDrink(Bar bar) {

        return bar.









    }
}
```

4. zadatak (10 bodova) Pretpostavite da postoji klasa `Main` koja proširuje `JFrame`, ima konstruktor i metodu `main` prema dostupnom predlošku. Nadalje, pretpostavite da postoji klasa `SlowArrayList` koja se ponaša kao `ArrayList` samo je izvođenje njezinih metoda vremenski zahtjevno. Klasa `StringWorker` je specijalizirani tip klase `SwingWorker` koja u metodi `doInBackground()` prolazi kroz listu Stringova (`slowList`), razlama ih po `Stringu` (`delimiter`), objavljuje međurezultate (primjeri međurezultata: „You“, „say“…) te računa i vraća ukupan broj razlomljenih dijelova (npr. 7). Međurezultati se ispisuju na standardni izlaz putem `chunksProcessor`, dok se konačni rezultat ispisuje na standardni izlaz putem `onDone`. Ispis mora odgovarati dolje-navedenom primjeru. **Na ovoj stranici popunite metodu `main` s rješenjem za `chunkProcessor` i onDone. Na idućoj stranici implementirajte `StringWorker`.**

```
Main() {
setSize(200, 100);
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}

static void main(String[] args) throws Exception {
  SwingUtilities.invokeAndWait(() -> {
       Main window = new Main();
       window.setLocation(20, 20);
       window.setVisible(true);

       List<String> slowList = new SlowArrayList<>();
       slowList.add("You say you");
       slowList.add("wander your own land");
       String delimiter = " ";
       // implementirati chunksProcessor i onDone









       StringWorker worker = new StringWorker(slowList, delimiter,
       chunksProcessor, onDone);
       worker.execute();
  });
}
```

ISPIS:

```
Found: You
Found: say
Found: you
Found: wander
Found: your
Found: own
Found: land
Total: 7
```



Napomena: U klasi `StringWorker` nije dozvoljeno ispisivati na standardni izlaz. Dohvat rezultata metode `doInBackground()` moguće je putem metode `get()` koja može vratiti iznimku ako se nešto loše dogodi. Potrebno je popuniti: članske varijable, konstruktor i njegove argumente, metode `doInBackground, process i done`.

```
class StringWorker extends SwingWorker<Integer, String> {









	public StringWorker(List<String> slowList, String delimiter,

                                                                    ){









	}
	@Override
	protected Integer doInBackground() throws Exception {

















	}
	@Override
	protected void process(List<String> chunks) {



	}
	@Override
	protected void done() {









	}

}
```

5. zadatak (10 bodova)

Zadan je razred `C`, koji implementira sučelje `Closeable`, i sadrži metodu `m`. Što će program ispisati nakon što se izvrši metoda `main`?

Rješenje upisati u iscrtkani okvir na ovom ispitu.

```
public class C implements Closeable {
	private Integer i = 0;

	public C() {
		System.out.println("created: " + i);
	}

	public C(Integer i) {
		this.i = i;
		System.out.println("created: " + i);
	}

	@Override
	public void close() throws IOException {
		System.out.println("close: " + this.i);
	}

	public void m(C other) {
		System.out.println("m: " + this.i / other.i);
	}

	public static void main(String[] args) {
		C c1 = new C(1);
		try (C c2 = new C(2)) {
			try (C c0 = new C(0); C c4 = null) {
				c1.m(c2);
				c1.m(c0);
				c2.m(new C());
				c0.m(c4);
			} catch (NullPointerException e) {
				System.out.println("NP exception");
			}
		} catch (Exception e) {
			System.out.println("exception");
		} finally {
			System.out.println("finally");
		}

		try {
			c1.close();
		} catch (IOException e) {
			System.out.println("IO exception");
		}

		System.out.println("end");
	}
}
```

Podsjetnik

Napomena**:** na ovoj stranici nije dozvoljeno pisanje rješenje zadataka.

(not all) functional interfaces (name and method signature)

BiConsumer<T,U> - void accept(T t, U u)
BiFunction<T,U,R> - R apply(T t, U u)
Consumer<T> - void accept(T t)
Runnable – void run()
Function<T,R> - R apply(T t)
Predicate<T> - boolean test(T t)
Supplier<T> - T get()

BasicFileAttributes

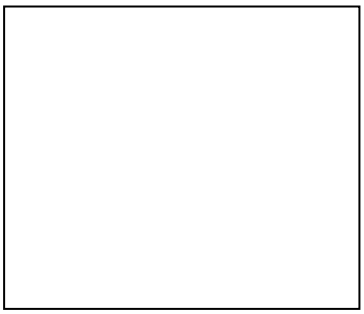
FileTime creationTime() - Returns the creation time.
Object fileKey() - Returns an object that uniquely identifies the given file, or null if a file key is not available.
boolean isDirectory() - Tells whether the file is a directory.
boolean isOther() - Tells whether the file is something other than a regular file, directory, or symbolic link.
boolean isRegularFile() - Tells whether the file is a regular file with opaque content.
boolean isSymbolicLink() - Tells whether the file is a symbolic link.
FileTime lastAccessTime() - Returns the time of last access.
FileTime lastModifiedTime() - Returns the time of last modification.
long size() - Returns the size of the file (in bytes).

Map<K,V>

default void forEach(BiConsumer<? super K,? super V> action) - Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
V get(Object key) - Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
Set<Map.Entry<K,V>> entrySet() - Returns a Set view of the mappings contained in this map.
default V getOrDefault(Object key, V defaultValue) - Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
Set<K> keySet() - Returns a Set view of the keys contained in this map.
default V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction) - If the specified key is not already associated with a value or is associated **with null, associates it with the given non-null value.**
default V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) - Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).

Collectors

static <T> Collector<T,?,Set<T>> toSet() - Returns a Collector that accumulates the input elements into a new Set.
static <T,K,U> Collector<T,?,Map<K,U>> toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper) - Returns a Collector that accumulates elements into a Map whose keys and values are the result of applying the provided mapping functions to the input elements.
static <T> Collector<T,?,List<T>> toList() - Returns a Collector that accumulates the input elements into a new List.
static <T> Collector<T,?,Long> counting() - Returns a Collector accepting elements of type T that counts the number of input elements.



Slobodni prostor za skiciranje rješenja.
Napomena: ova stranica se neće ocjenjivati, rješenja zadataka je potrebno upisati isključivo u za to predviđena mjesta.