Jesenski ispitni rok OOP 03.09.2019.

Ispit nosi ukupno 50 bodova i piše se 150 minuta.
NAPOMENA: Nije potrebno pisati javadoc i importanje paketa. Sva rješenja upisuju se izravno na ispit u predviđena polja (bez upotrebe dodatnih papira)

1. zadatak (15 bodova)

U sklopu programskog rješenja za simulaciju restorana potrebno je implementirati dio sustava. Sustav se sastoji od klase `Chef` kojim je jedinstveno opisan svaki kuhar/kuharica i to sa imenom `firstName` i prezimenom `lastName` te skupom svojih specijaliteta. Specijaliteti su modelirani klasom `SpecialtyDish` koja pohranjuje informacije o namirnicama potrebnim za pripravku tog jela te tipu jela `Type` koje može biti `VEGAN` i `NONVEGAN`. Također potrebno je napraviti metodu `addDish(SpecialtyDish dish)` s kojom je svakom kuharu/kuharici omogućeno dodavanje njegovih specijaliteta. Namirnice su modelirane klasom `FoodItem` a sadrže samo naziv `name` namirnice. Jednom stvorena namirnica ne može više mijenjati svoje ime, te ono također ne smije biti `null`. Količina i mjerna jedinica količine modelirani su klasom `AmountAndUnit`. Tom je klasom modeliran uređeni par količine `double amount` i mjerne jedinice `Unit`, koji također jednom nakon što su stvoreni se ne mogu mijenjati. Mjerna jedinica `Unit` može biti `GRAM` ili `LITER`. U klasi `SpecialtyDish` mora postojati metoda `addFoodItem` koja prima namirnicu, jedinicu i količinu te ju pohranjuje u svoje interno spremište. Također treba postojati metoda checkVeganFriendly koja za jelo provjerava je li to jelo sadrži ili ne sadrži meso. Ukoliko jelo nije prikladno za vegetarijance baca se neprovjeravana iznimka `WrongFoodTypeException`, koju je također potrebno modelirati. Dva su kuhara/kuharice jednaki ako su im jednaka jela koja kuhaju, te je s toga potrebno implementirati metode `equals` i `hashCode`. Također za kuhare mora postojati prirodan poredak po prezimenu, a ako su isti po imenu.
U unutrašnjosti košuljice ili na zasebne papire potrebno je napisati programski kod (atribute, konstruktore, metode, gettere i settere) svih klasa, iznimki i enumova, tj. `Chef`, `SpecialtyDish`, `FoodItem`, `AmountAndUnit`, `Unit`, `Type`, `WrongFoodException`. Gdje je potrebno, slobodno koristite klase i sučelja iz *Java Collection Frameworka*.

2. Zadatak (10 bodova)

Ulazna datoteka `F1.csv` sadrži podatke o plasmanima vozača Formule 1 u pojedinim utrkama kroz godine. U svakom retku datoteke nalazi se jedan zapis (opisuje jedan plasman):

```
godina;redni broj utrke;imeprezime_vozača;plasman
```

Iz tih zapisa je potrebno stvoriti primjerke razreda Plasman te ih dodati u odgovarajuću listu (`listaPlasmana`). Razred Plasman ima niže navedeni konstruktor, pri čemu parametri konstruktora služe za inicijalizaciju vrijednosti istoimenih atributa razreda, a razred još ima i odgovarajuće metode za postavljanje i dohvaćanje vrijednosti svih atributa.

```
public Plasman(int godina, int redniBrojUtrke, String vozac, int
plasman)
```

Koristeći kolekcijske tokove (Stream API) potrebno je na izlaznu jedinicu sortirano uzlazno po imenu ispisati sve vozače koji su pobijedili barem jednom nakon 2000.

```
public class PlasmanExplorer {

    List<Plasman> listaPlasmana=new ArrayList<Plasman>();

    public void loadPlasmani(String filepath) {

// TO DO: napišite metodu koja učitava plasmane iz datoteke









    }
    public void printPobjednike()
    {
// TO DO: napišite kod koji ispisuje uzlazno sortiranu listu svih pobjednika nakon
// 2000. koristeći Stream API

            listaPlasmana.stream()









      }
```

3. Zadatak (10 bodova)

Program prolazi kroz strukturu direktorija počevši od specificiranog direktorija koji je zadan konstantom `directory` na početku programa. Pri prolasku program traži identične datoteke i briše duplikate koji se nalaze „dublje" u hijerarhiji direktorija. Također, program bilježi ukupnu vrijednost oslobođene memorije, te broj pobrisanih datoteka. Program treba napisati oslanjajući se na `SimpleFileVisitor`.

Po završetku rada program treba na zaslon ispisati ukupni iznos oslobođene memorije i broj pobrisanih datoteka. Za ispis morate koristiti Javine kolekcijske tokove. Primjer formata ispisa:

```
Freed disk space 462551 Deleted files 4
```

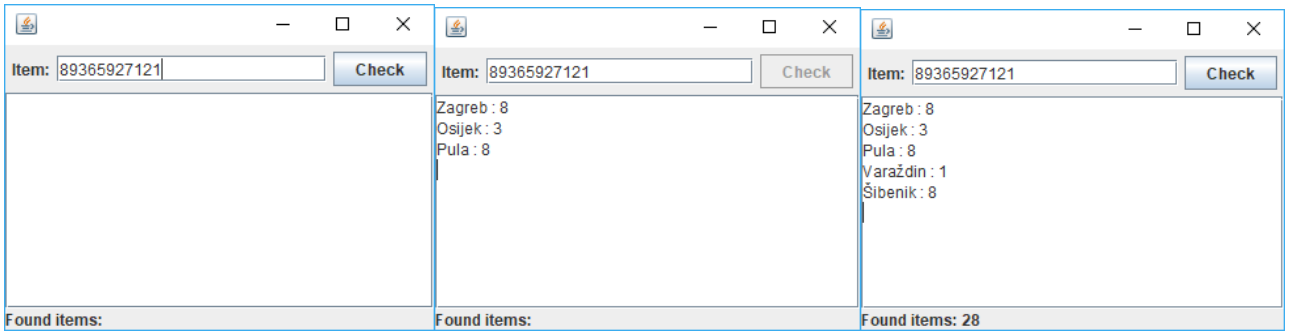
```
public class Main {
	public static void main(String[] args) {
		final String directory = "D:/_OOP";
		Path root = Paths.get(directory);
		DuplicateFileVisitor visitor = new DuplicateFileVisitor();
	    try {
	        Files.walkFileTree(root, visitor);
	    } catch (IOException e) {
	        e.printStackTrace();
	    }

	// TO DO: napisati kod za ispis ukupnog iznosa oslobođene memorije
	// i broja obrisanih datoteka - morate koristiti kolekcijske tokove

	}

}
```

Morate nadjačati odgovarajuće metode `FileVisitora`. Brisanje datoteke i spremanje podataka o brisanju izdvojeni su u posebnu metodu `deleteFile` koju morate napisati. Imate na raspolaganju statičku metodu `String Digester.MD5(Path file)` koja za datoteku vraća jedinstveni sažetak njenog sadržaja.

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

       private Map<String, Long> data = new HashMap<String, Long>();
       private Map<String, String> repo = new HashMap<String, String>();

       public Map<String, Long> getDeletedData() {return data;} //dohvaćanje pobrisanih

       public Map<String, String> getDataRepo() {return repo;} //dohvaćanje unikata

// TO DO: nadjačavanjem definirajte potrebne metode FileVisitora

    private void deleteFile(Path file, long size) {
       //TO DO napisati ostatak metode u kojem se spremaju podatci o brisanju

       //TO DO: obrisati datoteku

    }
}
```

4. Zadatak (15 bodova)

Potrebno je dovršiti programski kod aplikacije s grafičkim korisničkim sučeljem, koja za unesenu šifru artikla (niz znakova) provjerava na svim skladištima (tj. u svim udaljenim bazama podataka) neke tvrtke, koliko artikala s tom šifrom se nalazi na lageru. Primjer izgleda aplikacije prije, tijekom i nakon izvođenja prikazan je sljedećim slikama:



Nadopunite programski kod tako da vaša aplikacija vizualno i funkcionalno odgovara aplikaciji prikazanoj na slikama. Uočite da je dohvat podataka sa udaljenih skladišta potencijalno vremenski zahtjevna operacija. Aplikacija tijekom provjere treba ispisivati rezultate kako pristižu, tj. nakon što se provjeri neko skladište, taj se rezultat treba pojaviti u aplikaciji (slika 2). Radi jednostavnosti, međurezultate možete vraćati kao nizove znakova. Nakon što se provjere sva skladišta, aplikacija treba prikazati ukupni broj pronađenih artikala („Found items:" na slici 3). Treba spriječiti pokretanje nove provjere dok postojeća nije završila (slike .2 i 3.).
Za rad s udaljenim bazama koristite statičke metode iz klase **`RemoteDatabaseChecker`**.

```
public class OnStockCheckerFrame _____________________________________________ {

      private static final long serialVersionUID = 1L;
      private JLabel lbl1 = new JLabel("Item:");
      private JTextField tf = new JTextField(20);
      private JButton btn = new JButton("Check");
      private JTextArea ta = new JTextArea(10, 0);
      private JLabel lbl2 = new JLabel("Found items: ");

      public OnStockCheckerFrame() {

      btn.addActionListener(new ActionListener() {

       });

   } //public OnStockCheckerFrame()
```

```
private class OnStockCheckerWorker _____________________________________{

    public OnStockCheckerWorker _____________________________ {

    }

    @Override
    protected Integer doInBackground() throws Exception {

    }

    @Override
    protected void process(_____________________________________________) {

    }
```

```
        @Override
        protected void done() {









        }

    }  //private class OnStockCheckerWorker


       public static void main (String[] args) {
         SwingUtilities.invokeLater(new Runnable() {
             @Override public void run() {
                OnStockCheckerFrame frame = new OnStockCheckerFrame();
                frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);
             }
         });
       }
}


public class RemoteDatabaseChecker{
   public static List<String> getRemoteWarehouses() {
       return List.of("Zagreb", "Osijek", "Pula", "Varaždin", "Šibenik");
   }

   public static Integer checkRemoteForItem(String warehouse, String
itemId){
       //simulate long job
       try {
             Thread.sleep(1500);
       } catch (InterruptedException e) {
             e.printStackTrace();
       }
       return (int) (Math.random()*10);
   }
}
```