

Ljetni ispitni rok OOP 09.07.2019.

Ispit nosi ukupno 50 bodova i piše se 150 minuta. Svaki zadatak nosi 10 bodova.

NAPOMENA: Nije potrebno pisati javadoc i import paketa. Sva rješenja upisuju se izravno na ispit u predviđena polja (bez upotrebe dodatnih papira).

1. Zadatak

1. i 2. zadatak su povezani. Potrebno je dopuniti definicije klase koje se koriste za modeliranje prilika u dućanu na način da se omogući potpuna funkcionalnost klase `Main` koja je priložena u nastavku.

```
public class Main {
    public static void main(String[] args) {
        List<Item> items =.getItems();
        Item[] kosarica = new Item[items.size()];
        items.toArray(kosarica);

        // odnosi se na 2. zadatak
        Salesperson s = new Salesperson();
        s.printItems(items, i->i.barcode().startsWith("385"));
        s.printTheBill(kosarica);

        // odnosi se na 2. zadatak
        Map<String, OriginOfProduct> origins = s.getOrigin(items);
        for(Map.Entry<String, OriginOfProduct> entry: origins.entrySet())
            System.out.println(entry);

        Set<Item> skup = new LinkedHashSet<>();
        skup.add(new Food("Dukat jogurt", "385000909", 2.20, 0.1));
        skup.add(new Food("Dukat jogurt", "385000909", 2.20, 0.1));
        skup.addAll(items);
        for(Item i: skup)
            System.out.println(i.toString());
    }
    public static List<Item> getItems() {
        List<Item> items = new ArrayList<>();
        items.add(new Beverage("Jana voda", "38523456", 5.90, 1.5));
        items.add(new Food("Lay's", "492345678", 10.00, 0.3));
        Item [] itemsInPack = new Item [6];
        for(int i=0; i < itemsInPack.length; i++)
            itemsInPack[i] = new Food("Dukat jogurt", "385000909", 2.20, 0.1);
        items.add(new Pack("Dukat jogurt pack", "385876543", itemsInPack));
        return items;
    }
}
```

Izvođenje `main` metode rezultira sljedećim ispisom:

```
Dukat jogurt pack 385876543 Net:13,20 Sale:13,86
Jana voda 38523456 Net:5,90 Sale:7,38

Jana voda                7.38
Lay's                    10.50
Dukat jogurt pack 6x2.31
                        13.86
TOTAL: 31.74

38523456=DOMESTIC
492345678=FOREIGN
385876543=DOMESTIC

Dukat jogurt 385000909 Net:2,20 Sale:2,31
Jana voda 38523456 Net:5,90 Sale:7,38
Lay's 492345678 Net:10,00 Sale:10,50
Dukat jogurt pack 385876543 Net:13,20 Sale:13,86
```

U aplikaciji postoje primjerci objekata koji predstavljaju piće (`Beverage`), hranu (`Food`), i pakete artikala (`Pack`). Piće, hrana i paketi artikala jesu artikli (`Item`). Primjerke artikala nije moguće samostalno kreirati.

Prodajna cijena svakog artikla može se izračunati pozivom metode `getSalePrice` koja vraća prodajnu cijenu kao realan broj. Za piće i hranu se prodajna cijena računa tako da se neto cijena uveća za odgovarajući postotak poreza (TAX), a za paket artikala se prodajna cijena računa tako da se prodajna cijena jednog artikla pomnoži s brojem artikala u paketu. Za paket se, prilikom konstruiranja objekta računa i neto cijena paketa kao zbroj neto cijena artikala sadržanih u paketu. Paket može sadržavati samo jednake artikale. Artikli su jednaki ako su im jednaki *naziv*, *barkod* i *neto cijena*. Uzmite to u obzir prilikom pisanja metoda `hashCode` i `equals`. Nadalje, svi artikli su barkodirani jer implementiraju sučelje (`Barcoded`). Konstruktori navedenih klasa služe za inicijaliziranje vrijednosti atributa. Vrijednosti atributa u svim klasama se nakon inicijalizacije više ne mogu mijenjati. Vrijednosti se mogu dohvaćati preko standardnih getterskih metoda koje postoje i koje ne trebate pisati u klasama.

```
public interface Barcoded { String barcode(); // metoda vraća barkod }

// TO DO: dopuniti zaglavlje klase
public _____ Item _____ {
    private String name;
    private String barcode;
    protected double netPrice; // neto cijena
    public String getName() { return name; }
    public Item(String name, String barcode, double netPrice) {
        this.name = name; this.barcode = barcode; this.netPrice = netPrice;
    }
    // TO DO: napisati metode klase Item koje nedostaju
}
```

```
// TO DO: dopuniti zaglavlje klase
public class Beverage _____ {
    private double volume;
    private final int TAX = 25; //(%)
    // TO DO: napisati metode klase Beverage koje nedostaju

}
```

```
// TO DO: dopuniti zaglavlje klase
public class Food _____ {
    private double weight;
    private final int TAX = 5; //(%)
    // TO DO: napisati metode klase Food koje nedostaju

}
```

```
// TO DO: dopuniti zaglavlje klase
public class Pack _____ {
    private Item [] itemsInPack;
    public Item[] getItemsInPack() { return itemsInPack; }
    // TO DO: napisati metode klase Pack koje nedostaju

}
```

2. Zadatak

Za klasu prodavač (Salesperson) napišite metodu `printItems` koja ispisuje filtrirano samo one artikle iz liste koji zadovoljavaju predikat i to sortirano po nazivu artikla u formatu kako je prikazano u ispisu main metode (vidi zadatak 1). Za implementaciju ove metode morate koristiti Javine koleksijske tokove. Također, dopunite metodu `printTheBill` za ispis računa prema formatu kako je prikazano ranije u rezultatu izvođenja main metode. Napišite i metodu `getOrigin` koja za primjenu listu barkodiranih `Barcoded`, ali i za sve podtipove koji implementiraju `Barcoded` sučelje, vraća mapu u kojoj su ključevi barkodovi, a vrijednosti su enumeracijske konstante `DOMESTIC` ili `FOREIGN` koje morate sami definirati (prostor za definiciju enumeracije `OriginOfProduct` nalazi se u nastavku). Ako barkod počinje sa "385" proizvod je domaći (`DOMESTIC`), inače je strani (`FOREIGN`).

```
//TO DO definirajte enumeraciju OriginOfProduct s dvije konstante: DOMESTIC i FOREIGN
```

```
public class Salesperson {
    // TO DO: napišite ovu metodu tako da koristite koleksijske tokove
    public void printItems(List<Item> list, Predicate<Item> p) {

    }

    // TO DO: dopunite zaglavlje metode
    public void printTheBill(_____){
        double total = 0.;
        for(Item i: basket){
            // TO DO: napišite metodu tako da ispis odgovara prikazanom
            // (npr. sa System.out.format)

        }
        System.out.format("%30s%6.2f\n", "TOTAL: ", total);
    }
}
```

```
// TO DO: dopunite zaglavlje metode
public Map<String, OriginOfProduct>
    getOrigin(_____) {

    // TO DO: napišite metodu

}
}
```

Stream	
Modifier and Type	Method and Description
long	<code>count()</code> Returns the count of elements in this stream.
<code>Stream<T></code>	<code>distinct()</code> Returns a stream consisting of the distinct elements (according to <code>Object.equals(Object)</code>) of this stream.
<code><R> <code>Stream<R></code></code>	<code>map(Function<? super T, ? extends R> mapper)</code> Returns a stream consisting of the results of applying the given function to the elements of this stream.
<code>DoubleStream</code>	<code>mapToDouble(ToDoubleFunction<? super T> mapper)</code> Returns a <code>DoubleStream</code> consisting of the results of applying the given function to the elements of this stream.
void	<code>forEach(Consumer<? super T> action)</code> Performs an action for each element of this stream.
<code>Stream<T></code>	<code>filter(Predicate<? super T> predicate)</code> Returns a stream consisting of the elements of this stream that match the given predicate.
<code>Stream<T></code>	<code>sorted(Comparator<? super T> comparator)</code> Returns a stream consisting of the elements of this stream, sorted according to the provided <code>Comparator</code> .
<code>Stream<T></code>	<code>sorted()</code> Returns a stream consisting of the elements of this stream, sorted according to natural order.
DoubleStream	
Modifier and Type	Method and Description
<code>OptionalDouble</code>	<code>average()</code> Returns an <code>OptionalDouble</code> describing the arithmetic mean of elements of this stream, or an empty optional if this stream is empty.

Klasa OptionalDouble

void	<code>ifPresent(DoubleConsumer consumer)</code> Have the specified consumer accept the value if a value is present, otherwise do nothing.
boolean	<code>isPresent()</code> Return true if there is a value present, otherwise false.
double	<code>getAsDouble()</code> If a value is present in this <code>OptionalDouble</code> , returns the value, otherwise throws <code>NoSuchElementException</code> .

3. Zadatak

U iscrtkani prostor upišite u točnom redoslijedu što se ispisuje kao rezultat izvođenja metode `main`.

```
public class Main {
    public static void main(String[] args) {
        Set<Panda> family1 = new TreeSet<Panda>(Arrays.asList(new Panda("Sheng", 87), new Panda("Tai", 88)));
        Set<Panda> family2 = new TreeSet<Panda>(Arrays.asList(new Panda("Po", 102), new Panda("Tian", 99)));
        List<Set<Panda>> zoo = new LinkedList<Set<Panda>>();
        zoo.add(family1);
        zoo.add(family2);

        int i = 1;
        for (Set<Panda> family : zoo) {
            for (Panda p: family) {
                try(VirtualFood f = new VirtualFood(i)){
                    try {
                        p.eat(f);
                    } catch (OverweightException e) {
                        System.out.println(e.getMessage());
                    }
                } catch (Exception e) {
                    System.out.println(e.getMessage());
                    i=Integer.parseInt(e.getMessage());
                } finally {
                    i++;
                    System.out.println(p.getName() + "... feeding finished.");
                }
            }
        }
        System.out.println("main - end");
    }
}

public class Panda implements Comparable<Panda> {
    private String name;
    private Integer weight;
    public Panda(String name, Integer weight) { this.name = name; this.weight = weight; }
    public Integer getWeight() {return weight;}
    public void eat(VirtualFood f) throws OverweightException {
        if (weight > 100) {
            f.setConsumed(false);
            throw new OverweightException(this.name + " has to go on a diet.");
        }
        f.setConsumed(true);
        this.weight = this.weight + 10;
        System.out.println(this.name + " gained weight and now has " + this.weight + " kg");
    }
    @Override public int compareTo(Panda o) {
        return Integer.compare(this.getWeight(), o.getWeight());
    }
    @Override public String toString() {return name + ":" + weight;}

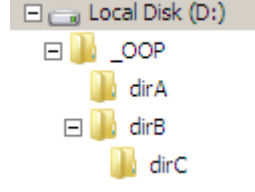
    public String getName() { return name; }
}

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

public class VirtualFood implements Closeable {
    int id;
    boolean consumed;
    public void setConsumed(boolean consumed) {
        this.consumed = consumed;
    }
    public VirtualFood(int id) {
        this.id = id;
    }
    @Override public void close() throws IOException {
        System.out.println("Closing food resource "
            + id);
        if(!consumed) throw new RuntimeException("Food "
            + id + " is not consumed");
    }
    public int getId() { return id; }
}
```

4. Zadatak

Program bilježi informacije o (pod)direktorijima i odabranim datotekama počevši od specificiranog direktorija koji je zadan konstantom `directory` na početku programa. Bilježe se informacije o početnom direktoriju, o svim poddirektorijima i svim odabranim datotekama. Konstanta `extension` može imati vrijednost "*" što znači da program mora bilježiti informacije o svim pronađenim datotekama neovisno o njihovim ekstenzijama. Druga je mogućnost da konstanta specificira neki konkretan tip datoteke npr. "pdf". U tom slučaju treba bilježiti informacije samo o datotekama s ekstenzijom .pdf (neovisno o velikim ili malim slovima), dok se ostale ignoriraju. O početnom direktoriju, postojećim poddirektorijima i odabranim datotekama prikupljaju se podaci o: *nazivu*, *vremenu stvaranja* i *veličini* direktorija/datoteke. Za svaki direktorij veličinu zabilježite kao 0 po čemu će se oni razlikovati od datoteka. Podaci se spremaju u listu. Element liste predstavlja string u kojem su navedene vrijednosti odvojene zarezom (CSV). Primjer liste za vrijednost `extension="*"` i `extension="ppt"` prikazan je u tablici. (Prepostavite da nema direktorija i datoteka koji sadrže zarez u nazivu.) Program treba napisati oslanjajući se na `SimpleFileVisitor`.

	<code>_OOP,2019-07-03 16:50,0</code> <code>dirA,2019-07-03 16:50,0</code> <code>file3.doc,2019-07-03 16:52,156672</code> <code>file4.ppt,2019-07-03 16:52,1911808</code> <code>dirB,2019-07-03 16:50,0</code> <code>dirC,2019-07-03 16:51,0</code> <code>file5,2019-07-03 16:53,27140</code> <code>file2.docx,2019-07-03 16:52,91567</code> <code>file1.pptx,2019-07-03 16:52,125602</code>	<code>_OOP,2019-07-03 16:50,0</code> <code>dirA,2019-07-03 16:50,0</code> <code>file4.ppt,2019-07-03 16:52,1911808</code> <code>dirB,2019-07-03 16:50,0</code> <code>dirC,2019-07-03 16:51,0</code>
---	---	---

Po završetku rada program treba na zaslon ispisati srednju vrijednost veličina svih datoteka o kojima su zabilježeni podaci u listi. Za ispis morate koristiti Javine koleksijske tokove. Za gore navedeni primjer lijeve liste ispis izgleda ovako: Average file size 462557.80

```
public class Main {
    public static void main(String[] args) {
        final String directory = "D:/_OOP";
        final String extension = "*"; // ili "pdf" ili "ppt" ili "java" ...
        Path root = Paths.get(directory);
        MetadataFileVisitor visitor = new MetadataFileVisitor(extension);
        try {
            Files.walkFileTree(root, visitor);
        } catch (IOException e) {
            e.printStackTrace();
        }
        // TO DO: napisati kod za ispis srednje vrijednosti veličina pronađenih
        //datoteka, morate koristiti koleksijske tokove
    }
}
```

Morate nadjačati odgovarajuće metode `FileVisitora`. Pripema i spremanje podataka u listu izdvojeni su u posebnu metodu `fillList` koju morate napisati. Podatak o vremenu stvaranja nalazi se u objektu `FileTime` (vidi JavaDoc na kraju ispita) i ima string reprezentaciju u formatu "1970-01-01T00:00:00Z". Vi trebate napisati funkciju `Function<String, String>` kojom se navedeni format pretvara u "1970-01-01 00:00" koji se koristi u listi (umjesto slova T staviti razmak i izbaciti sve iza druge dvotočke uključivši i nju).

```
public class MetadataFileVisitor extends SimpleFileVisitor<Path> {
    private List<String> data = new LinkedList<>();
    public List<String> getMetaData() {return data;} //dohvaćanje liste
    private String fileExtension;
    public MetadataFileVisitor(String fileExtension) {
        this.fileExtension = fileExtension;
    }
    // TO DO: nadjačavanjem definirajte potrebne metode FileVisitora
```

```
private void fillList(Path file, BasicFileAttributes attr) {
    StringBuilder sb = new StringBuilder();
    //TO DO: napisati funkciju za pretvorbu formata vremena

    Function<String, String> f =
```

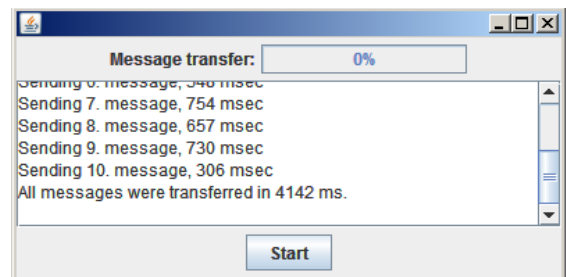
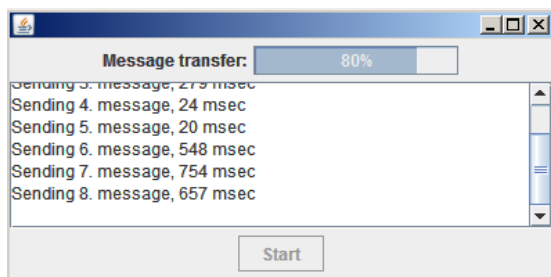


```
// TO DO napisati ostatak fillList metode u kojem se priprema string (sb) za  
// spremanje u listu
```

```
data.add(sb.toString());  
}  
}
```

5. Zadatak

Java aplikacija čije je grafičko sučelje prikazano na slici pokreće se na standardan način (main metoda nije prikazana) i predstavlja simulator slanja poruka. Lijeva slika prikazuje izgled sučelja tijekom slanja poruka, a desna prikazuje sučelje nakon obavljenog posla.



Vaš je zadatak da dopunite definiciju klase `SimulatedTaskFrame`. Najprije trebate dodati odgovarajuće panele i napraviti raspored komponenata (layout). Po pritisku na gumb "Start" treba gumb učiniti neaktivnim, podesiti vrijednost progress bara, sadržaj text komponente se treba isprazniti i na kraju se kreira i pokreće `MessageWorker` objekt. Nadalje, trebate nadjačati odgovarajuće metode u `MessageWorker` klasi. `MessageWorker` je `SwingWorker` koji u pozadini obavlja slanje ukupno `numMessages` poruka. Slanje poruke simulirano je pozivom metode `sendMessage`. Svaki put se prije poziva metode zabilježi vrijeme početka slanja pozivom `System.currentTimeMillis()`. To isto se učini nakon što završi slanje poruke. Razlika vremena kraja i početka slanja poruke predstavlja vrijeme potrebno za slanje poruke i ono se objavljuje u formi teksta kao međurezultat. Točan sadržaj teksta je prikazan na slikama sučelja. Također, u sklopu pozadinskog posla se zbrajaju vremena koja su bila potrebna za slanje svih `numMessages` poruka i to ukupno vrijeme se vraća kao rezultat pozadinskog posla (`Long`). Tekstovi objavljenih međurezultata se prikazuju u `textArea` komponenti, a pritom se ažurira i vrijednost progress bara. Na kraju obavljenog posla u `textArea` komponenti se prikazuje linija teksta u kojoj je vidljivo ukupno vrijeme koje je bilo potrebno za slanje svih poruka, a vrijednost progress bara i gumb se postavljaju u inicijalno stanje kako je prikazano na slici desno.

```

public class SimulatedMessageFrame extends JFrame {
    private SwingWorker<Long, String> worker;
    private int numMessages = 10;
    private JLabel label = new JLabel("Message transfer:", JLabel.CENTER);
    private JProgressBar progressBar = new JProgressBar();
    private JButton startButton = new JButton("Start");
    private JTextArea textArea = new JTextArea();
    public SimulatedMessageFrame() {
        // progress bar (0, numMessages) => inkrement vrijednosti za svaku poruku
        progressBar.setIndeterminate(false);
        progressBar.setStringPainted(true);
        progressBar.setMinimum(0);
        progressBar.setMaximum(numMessages);
        // TO DO: Dodaj odgovarajuće panele, layout, ...

        startButton.addActionListener((ActionEvent e) -> {
            // TO DO on Start

        });
    }
}

```

```
// TO DO MessageWorker dopunite zaglavlje klase
private class MessageWorker _____ {
    private int numMessages;
    public MessageWorker(int numMessages) {
        this.numMessages = numMessages;
    }
    // TO DO dopisati potrebne metode MessageWorkera
```

```

}
//Simulate a long run time needed for encryption and sending message
void sendMessage() throws InterruptedException {
    Random rnd = new Random(System.currentTimeMillis());
    long millis = rnd.nextInt(1000);
    Thread.sleep(millis);
}
}

```

Class `SwingWorker<T,V>` Type Parameters: T- the result type returned by thisSwingWorker's `doInBackground` and `get` methods. V- the type used for carrying out intermediate results by thisSwingWorker's `publish` and `process` methods

Modifier and Type	Method and Description
boolean	<code>cancel()</code> (boolean mayInterruptIfRunning) Attempts to cancel execution of this task.
protected abstract <code>T</code>	<code>doInBackground()</code> Computes a result, or throws an exception if unable to do so.
protected void	<code>done()</code> Executed on the <i>Event Dispatch Thread</i> after the <code>doInBackground</code> method is finished.
void	<code>execute()</code> Schedules this <code>SwingWorker</code> for execution on a <i>worker</i> thread.
<code>T</code>	<code>get()</code> Waits if necessary for the computation to complete, and then retrieves its result.
protected void	<code>process(List<V> chunks)</code> Receives data chunks from the <code>publish</code> method asynchronously on the <i>Event Dispatch Thread</i> .
protected void	<code>publish(V... chunks)</code> Sends data chunks to the <code>process(java.util.List<V>)</code> method.

Class System

Modifier and Type	Method and Description
public static long	<code>currentTimeMillis()</code> Returns the current time in milliseconds.

Sučelje `FileVisitor`:

Modifier and Type	Method and Description
<code>FileVisitResult</code>	<code>postVisitDirectory(T dir, IOException exc)</code> Invoked for a directory after entries in the directory, and all of their descendants, have been visited.
<code>FileVisitResult</code>	<code>preVisitDirectory(T dir, BasicFileAttributes attrs)</code> Invoked for a directory before entries in the directory are visited.
<code>FileVisitResult</code>	<code>visitFile(T file, BasicFileAttributes attrs)</code> Invoked for a file in a directory.
<code>FileVisitResult</code>	<code>visitFileFailed(T file, IOException exc)</code> Invoked for a file that could not be visited.

Enum `FileVisitResult`:

Enum Constant and Description
<code>CONTINUE</code> Continue.
<code>SKIP_SIBLINGS</code> Continue without visiting the <i>siblings</i> of this file or directory.
<code>SKIP_SUBTREE</code> Continue without visiting the entries in this directory.
<code>TERMINATE</code> Terminate.

Interface `BasicFileAttributes`

Modifier and Type	Method and Description
<code>FileTime</code>	<code>creationTime()</code> Returns the creation time. The creation time is the time that the file was created. If the file system implementation does not support a time stamp to indicate the time when the file was created then this method returns an implementation specific default value, typically the <i>last-modified-time</i> or a <code>FileTime</code> representing the epoch (1970-01-01T00:00:00Z)
long	<code>size()</code> Returns the size of the file (in bytes). The size may differ from the actual size on the file system due to compression, support for sparse files, or other reasons. The size of files that are not <i>regular</i> files is implementation specific and therefore unspecified.
boolean	<code>isDirectory()</code> Tells whether the file is a directory. Returns: true if the file is a directory

Class `FileTime`

Modifier and Type	Method and Description
public <code>String</code>	<code>toString()</code> Returns the string representation of this <code>FileTime</code> . The string is returned in the ISO 8601 format: YYYY-MM-DDThh:mm:ss[.s+]Z where "[.s+]" represents a dot followed by one or more digits for the decimal fraction of a second. It is only present when the decimal fraction of a second is not zero. For example, <code>FileTime.fromMillis(1234567890000L).toString()</code> yields "2009-02-13T23:31:30Z", and <code>FileTime.fromMillis(1234567890123L).toString()</code> yields "2009-02-13T23:31:30.123Z".

Interface `Function<T,R>`

Modifier and Type	Method and Description
<code>R</code>	<code>apply(T t)</code> Applies this function to the given argument.