

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 importanje 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 klasa 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);
        Salesperson s = new Salesperson();
        s.printItems(items, i->i.barcode().startsWith("385"));

        s.printTheBill(kosarica);

        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 abstract class Item implements Barcoded {
    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
    public abstract double getSalePrice();
    //public double getSalePrice() {return 0;} // priznati i ovakvo rješenje

    @Override public String barcode() { return barcode; }

    @Override public int hashCode() {
        int result = ((barcode == null) ? 0 : barcode.hashCode())
            + ((name == null) ? 0 : name.hashCode())
            + Double.valueOf(netPrice).hashCode();
        return result;
    }

    @Override public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (!(obj instanceof Item))
            return false;
        Item other = (Item) obj;

        return this.barcode.equals(other.barcode) &&
            this.name.equals(other.name) &&
            Double.valueOf(this.netPrice).equals(Double.valueOf(other.netPrice));
    }

    @Override public String toString() {
        return String.format("%s %s Net: %.2f Sale: %.2f",
            name, barcode, netPrice, getSalePrice());
    }
}
```

```
// TO DO: dopuniti zaglavlje klase
public class Beverage extends Item {
    private double volume;
    private final int TAX = 25; //(%)
    // TO DO: napisati metode klase Beverage koje nedostaju
    public Beverage(String name, String barcode, double netPrice, double volume) {
        super(name, barcode, netPrice);
        this.volume = volume;
    }
    @Override public double getSalePrice() {
        return netPrice + (TAX/100.)*netPrice;
    }
}
```

```
// TO DO: dopuniti zaglavlje klase
public class Food extends Item {
    private double weight;
    private final int TAX = 5; //(%)
    // TO DO: napisati metode klase Food koje nedostaju
    public Food(String name, String barcode, double netPrice, double weight) {
        super(name, barcode, netPrice);
        this.weight = weight;
    }
    @Override public double getSalePrice() {
        return netPrice + (TAX/100.)*netPrice;
    }
}
```

```
// TO DO: dopuniti zaglavlje klase
public class Pack extends Item {
    private Item [] itemsInPack;
    public Item[] getItemsInPack() { return itemsInPack; }
    // TO DO: napisati metode klase Pack koje nedostaju
    public Pack(String name, String barcode, Item [] itemsInPack) {
        super(name, barcode, itemsInPack[0].netPrice*itemsInPack.length);
        this.itemsInPack = itemsInPack;
    }
    @Override public double getSalePrice() {
        return itemsInPack.length*itemsInPack[0].getSalePrice();
    }
}
```

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 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 enum OriginOfProduct {  
    DOMESTIC, FOREIGN  
}
```

```
public class Salesperson {  
    // TO DO: napišite metodu tako da koristite koleksijske streamove  
    public void printItems(List<Item> list, Predicate<Item> p) {  
        list.stream()  
            .filter(p)  
            .sorted((i1, i2)->i1.getName().compareTo(i2.getName()))  
            .forEach(i->System.out.println(i));  
    }  
    // TO DO: dopunite zaglavlje metode  
    public void printTheBill(Item [] basket){  
    //public void printTheBill(Item ... basket){      // OK  
        double total = 0.;  
        for(Item i: basket){  
            // TO DO: napišite metodu tako da ispis odgovara prikazanom  
            if(i instanceof Pack)  
                System.out.format("%s %dx%.2f\n%-30s%.2f\n", i.getName(),  
                    ((Pack) i).getItemsInPack().length,  
                    ((Pack) i).getItemsInPack()[0].getSalePrice(), "", i.getSalePrice() );  
            else  
                System.out.format("%-30s%.2f\n", i.getName(), i.getSalePrice());  
            total += i.getSalePrice();  
        }  
        System.out.format("%30s%.2f\n", "TOTAL: ", total);  
    }  
    // TO DO: dopunite zaglavlje metode  
    public Map<String, OriginOfProduct> getOrigin(List<? extends Barcoded> list) {  
        // TO DO: napišite metodu  
        Map<String, OriginOfProduct> origins = new LinkedHashMap<>();  
        for(Barcoded b: list){  
            if(b.barcode().startsWith("385"))  
                origins.put(b.barcode(), OriginOfProduct.DOMESTIC);  
            else  
                origins.put(b.barcode(), OriginOfProduct.FOREIGN);  
        }  
        return origins;  
    }  
}
```

3. Zadatak

U iscrtkani prostor upišite u točnom redoslijedu što se ispisiuje 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; }
}
```

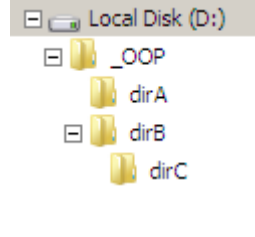
```
public class OverweightException extends Exception {
    public OverweightException(String msg) { super(msg); }
}

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; }
}
```

```
Sheng gained weight and now has 97 kg
Closing food resource 1
Sheng... feeding finished.
Tai gained weight and now has 98 kg
Closing food resource 2
Tai... feeding finished.
Tian gained weight and now has 109 kg
Closing food resource 3
Tian... feeding finished.
Po has to go on a diet.
Closing food resource 4
Food 4 is not consumed
Po... feeding finished.
Exception in thread "main"
java.lang.NumberFormatException
```

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 veličine svih datoteka o kojima su zabilježeni podaci u listi. Za ispis morate koristiti Javine koleksijske tokove. Za gore navedeni primjer lijeve liste ispis izgleda obako: `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

        try{
            OptionalDouble avg = visitor.getMetaData().stream()
                .map(s ->s.split(","))
                .filter(s->!s[2].equals("0"))
                .mapToDouble(s->Integer.parseInt(s[2]))
                .average();
            System.out.format("\nAverage file size %.2f", avg.getAsDouble());
        }catch(Exception e){
            System.out.println(e);
        }

    }
}
```

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" (umjesto slova T staviti razmak i izbaciti sve iza druge dvotočke uključivši i nju) koji se koristi u listi.

```
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
    @Override
    public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attr)
        throws IOException {
        fillList(dir, attr);
        return CONTINUE;
    }
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attr) {
        if(fileExtension.equals("")) {
            fillList(file, attr);
            return CONTINUE;
        }

        String fileName = file.getFileName().toString();
        String extension="";
        int i = fileName.lastIndexOf('.');
        if (i > 0)
            extension = fileName.substring(i+1).toUpperCase();

        if(extension.equals(fileExtension.toUpperCase()))
            fillList(file, attr);
        return CONTINUE;
    }

    private void fillList(Path file, BasicFileAttributes attr) {
        StringBuilder sb = new StringBuilder();
        //TO DO: napisati funkciju za pretvorbu formata vremena
        Function<String, String> f = new Function<String, String>(){
            @Override public String apply(String s) {
                return s.substring(0, 10) + " " + s.substring(11, 16);
            }
        };

        //TO DO napisati ostatak metode u kojem se priprema string za spremanje u listu
        sb.append(file.getFileName().toString()+",");

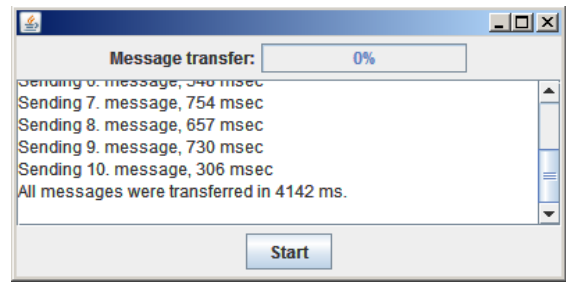
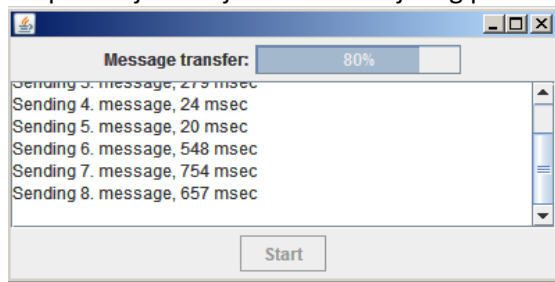
        String timeStr = attr.creationTime().toString();
        sb.append(f.apply(timeStr)+",");

        if (!attr.isDirectory()) {
            sb.append(String.valueOf(attr.size()));
        } else
            sb.append(0);

        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 ove 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 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 tekst u kojem 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 vrijednostiza svaku poruku
        progressBar.setIndeterminate(false);
        progressBar.setStringPainted(true);
        progressBar.setMinimum(0);
        progressBar.setMaximum(numMessages);
        // TO DO Dodaj odgovarajuće panele, layout, ...
        this.setLayout(new BorderLayout()); // nepotrebno default
        JPanel topArea = new JPanel();
        topArea.setLayout(new FlowLayout()); // default
        topArea.add(label);
        topArea.add(progressBar);
        add(topArea, BorderLayout.NORTH);
        JPanel buttonsArea = new JPanel();
        buttonsArea.setLayout(new FlowLayout()); // default
        buttonsArea.add(startButton);
        add(buttonsArea, BorderLayout.SOUTH);
        textArea.setEditable(false);
        JScrollPane scroll = new JScrollPane(textArea);
        add(scroll, BorderLayout.CENTER); //add(scroll); // OK
    }
}
```



```

        startButton.addActionListener((ActionEvent e) -> {
            // TO DO on Start
            startButton.setEnabled(false);
            progressBar.setValue(0);
            textArea.setText("");
            worker = new MessageWorker(numMessages);
            worker.execute();
        });
    }

    // TO DO MessageWorker dopunite
    private class MessageWorker extends SwingWorker<Long, String> {
        private int numMessages;
        public MessageWorker(int numMessages) {
            this.numMessages = numMessages;
        }
        // TO DO dopisati potrebne metode MessageWorkera
        @Override protected Long doInBackground() throws Exception {
            long totalTime = 0L;
            for (int i = 0; i < numMessages; i++) {
                long elapsedTime = System.currentTimeMillis();
                sendMessage();
                elapsedTime = System.currentTimeMillis() - elapsedTime;
                String status = "Sending " + (i+1) + ". message, " + elapsedTime
                    + " msec";
                publish(status);
                totalTime += elapsedTime;
            }
            return totalTime;
        }
        @Override protected void process(List<String> chunks) {
            for (String chunk : chunks) {
                textArea.append(chunk + "\n");
                progressBar.setValue(progressBar.getValue() + 1);
            }
        }
        @Override protected void done() {
            try {
                textArea.append("All messages were transferred in " + get()
                    + " ms.\n");
            } catch (Exception e) { e.printStackTrace(); }
            progressBar.setValue(0);
            startButton.setEnabled(true);
        }
    }

    /**
     * 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);
    }
}

```