Objektno orijentirano programiranje

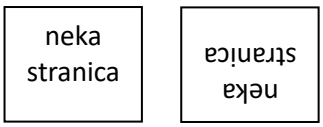
Ispitni rok

15.9.2020.

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)

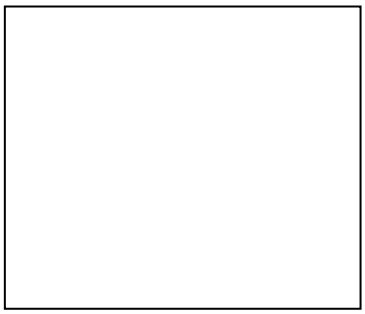
Potrebno je modelirati sustav telekomunikacijskog operatera koji je opisan nastavku teksta. **Vaš zadatak je nadopuniti predložak UML dijagrama klasa ovog sustava koji se nalazi na sljedećoj stranici**. Pritom **možete dodati nove klase** (ili sučelja) ako smatrate da je to potrebno**. Nije potrebno pisati programski kod sustava i to neće donositi nikakve bodove**. Na ovom predlošku pažljivo navedite:

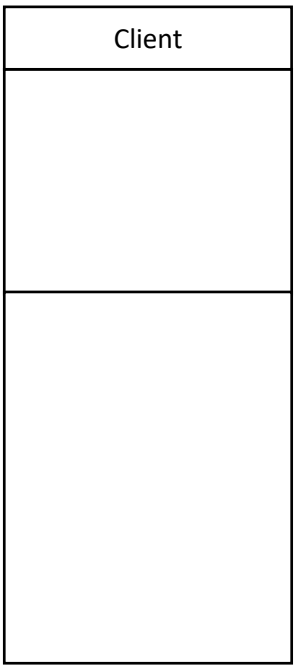
- oznake za sučelje (**I**), enumeraciju (**E**), apstraktnu (**A**) ili običnu klasu (**C**)
- oznake za apstraktnu metodu (**A**) te oznake za statičku (**S**) i finalnu (**F**) metodu ili atribut
- modifikatore vidljivosti metoda i atributa (**+** public, **#** protected, **-** private, **~** package-private)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

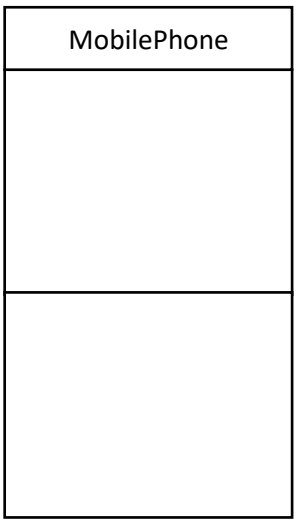
Telekomunikacijski operater ima ponudu vlastitih proizvoda (ProductOffering), pri čemu svaki proizvod predstavlja paket usluga. Za svaki proizvod se bilježi njegov naziv (name). Svaki ponuđeni proizvod ima svoj cjenik (Pricing) koji ima datum početka važenja (validFrom) i datum završetka važenja (validTo). Jedan cjenik može se koristiti na više proizvoda. Svaki cjenik sastoji se od barem jedne stavke (PriceItem) koja konkretno može biti cijena jedinice usluge (ChargePriceItem) ili besplatne jedinice usluge (AllowancePriceItem). Tip usluge na koju se odnosi stavka može biti glasovna (Voice), tekstualna poruka (SMS), te podaci (Data). Cjenik ne može sadržavati više jednakih stavki, a to su stavke koje se odnose na isti tip usluge. Na primjer, cjenik ne može sadržavati više stavki koje bi definirale cijene za glasovnu (Voice) uslugu. Ali, cjenik može sadržavati stavku koja definira cijenu i stavku koja definira besplatne jedinice za jednu uslugu, te se na taj način definira broj besplatnih jedinica za svaki mjesec i puna cijena usluge koja se plaća jednom kada se besplatne jedinice potroše. Stavka cijene usluge definira cijenu (unitPrice) po jedinici usluge, dok stavka besplatnih jedinica usluge definira broj jedinica usluge (units) koje su besplatne na mjesečnoj razini.

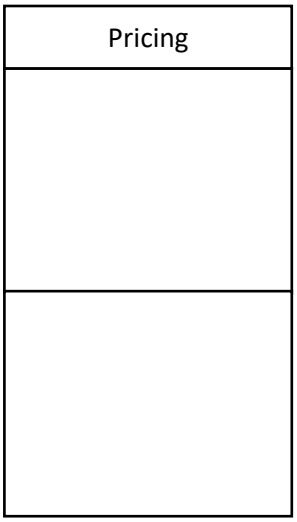
Klijent telekomunikacijskog operatora (Client) je fizička osoba za koju se bilježi OIB (ID), ime (name), prezime (lastName), te datum rođenja (birthDate), te može imati više mobilnih telefona (mobilePhones). Svaki od mobilnih telefona (MobilePhone) ima svoj jedinstveni broj (MSISDN) i proizvod koji koristi (product). Za svaki mobilni telefon se bilježi lista ostvarenih usluga (serviceLog) čije stavke (ServiceLogItem) sadržavaju vrijeme ostvarenja usluge (timestamp), neki od ranije navedenih tipova usluge (serviceType), da li je usluga ulazna (incoming = false/true), to jest da li je uspostavu usluge zatražio korespondent, broj korespondenta (correspondentISDN), te broj utrošenih jedinica usluge (units).

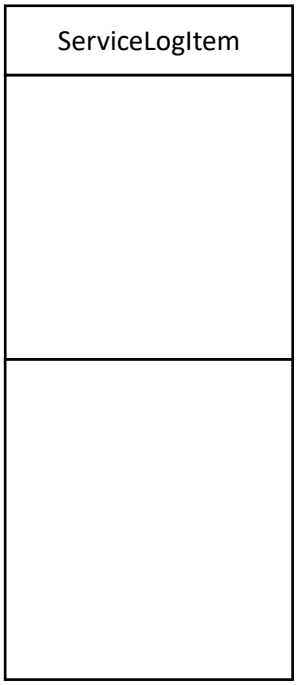
Za svaku klasu definirajte konstruktor koji prima sve vrijednosti svojstava te klase koje nisu kolekcije. Za sva svojstva klasa dodati gettere. Za svaku kolekciju definirajte metodu za dodavanje elemenata kolekcije (add****), na primjer *addPriceItem* za dodavanje stavki cjenika.

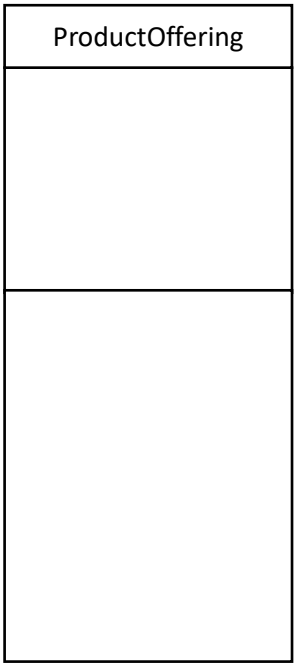


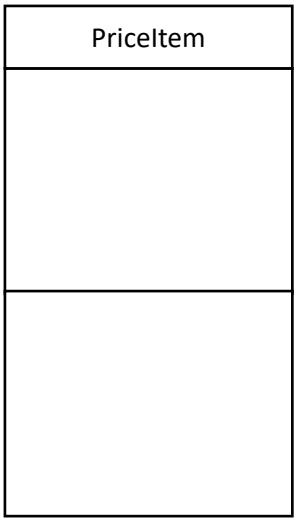

Client

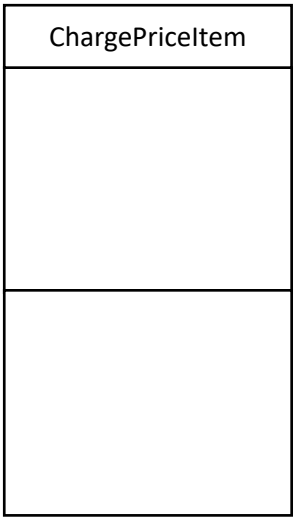

MobilePhone

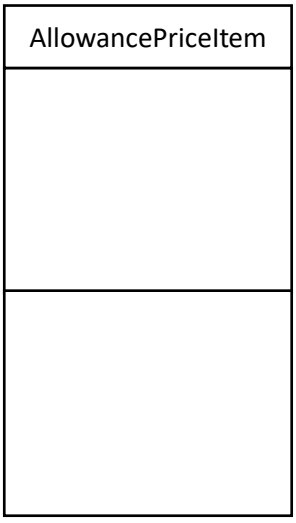

Pricing


ServiceLogItem

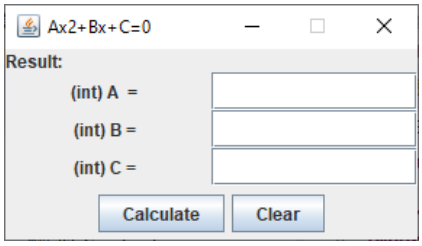

ProductOffering


PriceItem

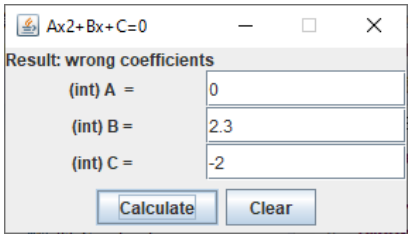

ChargePriceItem


AllowancePriceItem

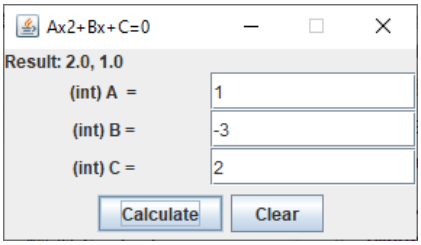
2. zadatak (10 bodova) Potrebno je dopuniti konstruktor klase `QuadraticEquationCalculator` koja predstavlja GUI kalkulatora za rješavanje kvadratnih jednadžbi $Ax^2 + Bx + C = 0$ čiji su koeficijenti *A*, *B* i *C* cjelobrojne vrijednosti. Sl. 1 prikazuje grafičko sučelje po pokretanju aplikacije. Sl. 2, 3 i 4 prikazuju sučelje nakon stiskanja `Calculate` gumba: - ako koeficijenti nisu pravilno upisani, - ako su koeficijenti ispravni, a rješenja su realni brojevi, - ako su koeficijenti ispravni a rješenja su kompleksni brojevi. Nakon pritiska na gumb `Clear` sučelje mora izgleda jednako kao i prilikom pokretanja aplikacije (sl. 1).



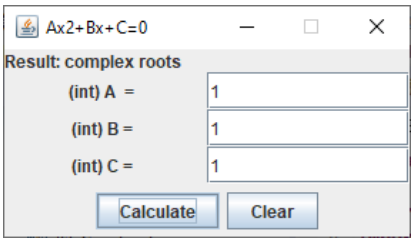
sl.1



sl. 2



sl.3

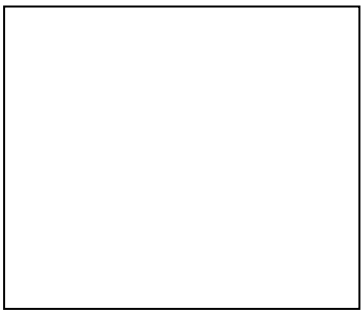


sl. 4

U klasi `QuadraticEquationCalculator` definirane su sve potrebne članske varijable. Kod dopunite sljedećim redoslijedom: 1) Formiranje (priprema) panela gumba. 2) Formiranje panela koeficijenata koji sadrži labele s nazivima koeficijenata i komponente za upis vrijednosti koeficijenata (text fields). 3) Dodavanje labele (`resultLabel`) za ispis rezultata/poruka, panela koeficijenata i panela gumba na prozor aplikacije. 4) Obrada događaja „pritisak na gumb `Clear`". 5) Obrada događaja „pritisak na gumb `Calculate`", pri čemu ako se jedna ili više upisanih vrijednosti koeficijenata ne može parsirati u integer ispisuje se poruka `Result: wrong coefficients` (sl. 2). Ista poruka se ispisuje ako je vrijednost koeficijenta A = 0. Ako su vrijednosti koeficijenata takve da su rješenja kvadratne jednadžbe kompleksni brojevi, rješenja se ne računaju već se ispisuje poruka `Result: complex roots` (sl.4). Ako su rješenja jednadžbe realni brojevi ona se računaju i ispisuju u formatu `Result: x1, x2` (sl. 3). Podsjetnik: $x_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$.

```
public class QuadraticEquationCalculator extends JFrame {
        private final int COEFF = 3;
        private final String PREFIX = "Result: ";
        private JPanel coeffPanel, buttonPanel;
        private JLabel resultLabel = new JLabel(PREFIX, JLabel.LEFT);
        private JLabel labels[] = new JLabel[COEFF];
        private JTextField inputs[] = new JTextField[COEFF];
        private String lebelTexts[]= {"(int) A  =  ", "(int) B =  ","(int) C =  "};
        private JButton calculateButton = new JButton("Calculate");
        private JButton clearButton = new JButton("Clear");

        public QuadraticEquationCalculator() {
            super("Ax2+Bx+C=0");
            setSize(300,170);
            setLayout(new BorderLayout());
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // TO DO: Formiranje panela gumba
```



// TO DO: Formiranje panela koeficijenata

// TO DO: Dodavanje komponenata na prozor

// TO DO: Obrada događaja „Clear“

// TO DO: Obrada događaja „Calculate“

```
        }// kraj konstruktora
        public static void main(String[] args) {
           SwingUtilities.invokeLater(() -> {
               QuadraticEquationCalculator qec = new QuadraticEquationCalculator();
               qec.setResizable(false);qec.setVisible(true);
            });
        }
}
```

3. zadatak (10 bodova)

Potrebno je implementirati klasu `DiskAnalyzer` koja nasljeđuje `SimpleFileVisitor<Path>` i to na način da njena javna metoda `public Map<String, Long> getSizesByExtension()` po izvođenju metode `Files.walkFileTree` ostvaruje sljedeću zadaću:

- metoda treba vratiti mapu u kojoj su ključevi ekstenzije datoteka, a vrijednost je zbirni iznos veličina svih pronađenih datoteka koje imaju dotičnu ekstenziju gledano od početnog direktorija i dublje (početni direktorij se zadaje u pozivu walkFileTree metode). Pretpostavite da sve datoteke sadrže ekstenziju. Pri tome datoteke čija imena sadrže podstring **`copy`** trebaju se **preskočiti**, odnosno njihove veličine se **ne smiju pojaviti u ukupnom zbroju gledano po ekstenzijama.** Također, **direktoriji čija imena sadrže podstring `copy` i sav njihov sadržaj** trebaju se preskočiti.

Dodatno, po izvođenju metode `Files.walkFileTree` potrebno je:

- Za svaki direktorij izračunati **ukupnu veličinu** svih **datoteka (ali ne i njegovih poddirektorija)** koje se nalaze u tom direktoriju.
- Kako je već navedeno, **direktoriji čija imena sadrže podstring `copy` i sav njihov sadržaj** trebaju se preskočiti (zanemariti). Također, datoteke čija imena sadrže podstring **`copy`** trebaju se **preskočiti**, odnosno njihove veličine se **ne smiju pojaviti u ukupnom zbroju**
- Veličinu direktorija potrebno je ispisivati na standardni izlaz počevši od dna stabla prema vrhu.

Dakle, ako je zadan početni direktorij sa sljedećom strukturom:

```
folder-------aa--------a.png  // Veličine 1
                |--------b.png // Veličine 2
                |--------c-copy.png // Veličine 2
      |--------dd.png // Veličine 4
      |--------ee-copy
                |--------f.png // Veličine 2
```

Ispis po pozivu metode `walkFileTree` treba biti sljedeći:

```
/folder/aa 3
/folder 4
```

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

  private final Map<String, Long> sizesByExtension;
  private static final String skipString = "copy";

  public Map<String, Long> getSizesByExtension() {
    return sizesByExtension;
  }

  public DiskAnalyzer() {
    sizesByExtension = new HashMap<>();
  }

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




  }

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









  }

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









  }
}
```

4. zadatak (10 bodova)

Dopunite klasu **`Car`** koja opisuje automobile. Za svaki automobil pohranjuje se registracija, proizvođač (npr. Toyota), model (npr. Yaris), zapremnina motora (u ccm) te podaci o dodatnoj opremi. **Dva su automobila jednaka ako imaju jednaku registraciju.** Prilikom implementacije vodite računa da se objekti klase **Car** moraju moći ispravno koristiti u Javinim kolekcijama (npr. **`HashMap`** i **`TreeSet`**). **Prirodni poredak automobila je prvo po proizvođaču, zatim po modelu i na kraju po registraciji.**

Klasa Car ima i dva javna konstantna komparatora: po registraciji (komparator **`BY_REG`**) i po zapremnini motora (komparator **`BY_ENG`**). **Komparatore definirati pomoću lambda izraza.**
Možete pretpostaviti da su definirani getteri i setteri za sve privatne varijable, te metoda toString koja ispisuje sve podatke o automobilu.
U glavnom programu definirano je polje **`cars`** koje sadrži 10 automobila. U prvom dijelu glavnog programa polje je potrebno prepisati u HashMapu **`mCars`** u kojoj je ključ registracija, a vrijednost odgovarajući objekt tipa **`Car`**, te tu mapu napuniti svim objektima iz polja.
U drugom dijelu glavnog programa potrebno je definirati skup **`sCars`** i napuniti ga svim objektima iz polja **`cars`**. Ispis svih elemenata iz skupa mora biti prema prirodnom poretku.

```
public class Car _____{
    private String registration;
    private String manufacturer;
    private String model;
    private int engine;
    private String equipment;

    /* Getteri i setteri - NE TREBA PISATI*/

    public Car(String t_reg, String t_man, String t_mod, int t_eng, String t_equ) {
        . . .
    }

    public String toString() {
        . . .
    }

    /* Implementacija prirodnog poretka */
```

```
    /* Metode potrebne za ispravno korištenje u kolekcijama */
```

```
    public static final Comparator<Car>

      BY_REG = ____________________________________________________________;

    public static final Comparator<Car>

      BY_ENG = ____________________________________________________________;


    public static void main(String[] args) {
        Car [] cars = new Car[10];
        /* Punjenje polja elementima – NE TREBA PISATI*/


        /* Definicija mape mCars i punjenje mape automobilima iz polja cars */









        System.out.println("\nMAPA:");
        for (String reg : mCars.keySet()) System.out.println(mCars.get(reg));

         /* Definicija skupa sCars i punjenje skupa automobilima iz polja cars */









        System.out.println("\nSKUP - prirodni poredak:");
        for (Car car : sCars) System.out.println(car);

    }
}
```

5. zadatak (10 bodova)

Zadani su razredi `ResourceException` i `Resource` koji implementira sučelje `Closeable`. Što će program ispisati tijekom izvođenja metode `main` ?

Rješenje upisati u iscrtkani okvir na ovom ispitu.

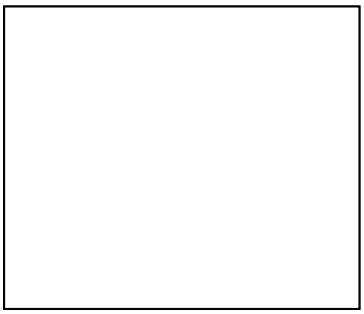
```
public class ResourceException extends RuntimeException {}

public class Resource implements Closeable {
    int p;
    public Resource(int p) {
       this.p = p;
       System.out.println(p + " created");
    }
    @Override public void close() throws IOException {
       System.out.println(p + " closed");
    }
    public void work() throws ResourceException {
       if(p == 2) throw new ResourceException();
       System.out.println(p + " work ends " + 30/p );
    }

    public static void main(String[] args) {
       Set<Integer> priorities = new TreeSet<>();
       priorities.add(2);
       priorities.add(0);
       priorities.add(2);
       List<Resource> resources = new ArrayList<>();
       for(Integer p: priorities)
              resources.add(new Resource(p));

       for(Resource r : resources) {
              try(r) {
                     r.work();
              }catch(ResourceException e) {
                     System.out.println("Resource Exception");
              }catch(IOException e) {
                     System.out.println("IO Exception");
              }catch(RuntimeException e) {
                     System.out.println("Runtime Exception");
              }
       }

       try(Resource r3 = new Resource(3);
           Resource r1 = new Resource(1);) {
           Resource r0 = new Resource(0);
           r3.work();
           r1.work();
           r0.work();
           System.out.println("try ends");
       }catch(Exception e) {
           System.out.println("exception");
           return;
       }finally {
           System.out.println("finally");
       }
       System.out.println("end");
    }
}
```



Podsjetnik

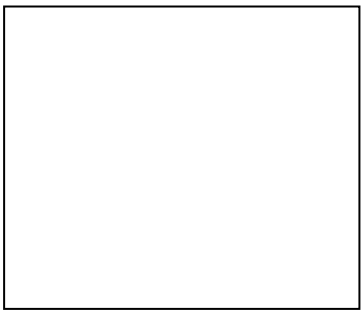
Napomena: na ovoj stranici nije dozvoljeno pisanje rješenja zadataka.

Interface FileVisitor<T>

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

Enum FileVisitResult

Enum Constant and Description
CONTINUE Continue.
SKIP_SIBLINGS Continue without visiting the *siblings* of this file or directory.
SKIP_SUBTREE Continue without visiting the entries in this directory.
TERMINATE Terminate.



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