



Objektno orijentirano programiranje Jesenski ispitni rok 2b

25.8.2020.

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 na stranice ispita. *Ako na nekoj stranici ne budete imali dovoljno mjesta, ostatak možete napisati na zadnju (praznu) stranicu ispita.*

Obavezno treba predati sve stranice ispita. *One stranice koje nisu predane neće biti ocjenjene.*

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 vođenja statistike za sportske igre mladih na temelju opisa koji slijedi u nastavku. **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 -.-.-▷)

U sustavu za evidenciju sportskih natjecanja evidentira se svaka osoba `Person`. Osoba ima ime `name` tipa `String`, prezime `surname` tipa `String`, datum rođenja `dateOfBirth` tipa `LocalDate` te visinu i tjelesnu masu (oboje tipa `int`). Svi atributi imaju pripadne getterske i setterske metode. Pretpostavlja se da je klasa `LocalDate` uvezena gdje je potrebno. Osoba može, ali i ne mora biti natjecatelj. Natjecatelj `Competitor` ima svoj identifikator `competitorID`. Dva natjecatelja su jednaka ako im je jednak identifikator. Pretpostavite da se čovjek može natjecati samo u jednom sportu. Neke osobe organizirane su pomoću liste `teammates` u ekipu `Team`. Svaka ekipa također ima svoje ime `String` `teamName` i identifikator `int` `teamID`, pri čemu se `teamName` može postaviti i dohvatiti. Također postoji metoda za dodavanje natjecatelja u tim `addCompetitor` te metoda `getTeammates` koja dohvaća listu s ekipom.

Klasa `Sport` sadrži datum te kategorije po dobi (`AgeCategory`: `YOUTH`, `JUNIOR`, `SENIOR`) i spolu (`GenderCategory`: `MALE`, `FEMALE`). Osim toga, klasa `Sport` u sebi sadrži mapu `scoreMap`, u kojoj će biti zapisani ključ `Competitor` i vrijednost `String` te metoda za dodavanje u mapu `addToScoreMap`. U `Stringu` će se dodavati formatirani zapis učinka natjecatelja za jedan sportski događaj. Sve neapstraktne klase koje nasljeđuju `Sport` moraju imati metodu: `calculateStatistics`, koja vraća novu mapu `<Competitor, String>` s izračunatom sportskom statistikom (za svaki sport na različit način). Sport može biti ekipni (`TeamSport`) i individualni (`IndividualSport`). Za ekipne sportove evidentira se ekipa `Team` na domaćem terenu i ekipa u gostima (`homeTeam` i `awayTeam`). Za postojeće klase sportskih disciplina dodajte `FootballGame` i `BasketballGame` te njima potrebne metode. Od individualnih sportova postoji klasa `RunningEvent` koja ima duljinu staze `int` `trackLength`. Sve mape koje se koriste su implementirane pomoću klase `HashMap`.

Person	
--------	--

Team	
------	--

Sport	
-------	--

--

Identifiable	
--------------	--

TeamSport	
-----------	--

IndividualSport	
-----------------	--

Competitor	
------------	--

BasketballGame	
----------------	--

FootballGame	
--------------	--

GenderCategory	
----------------	--

AgeCategory	
-------------	--

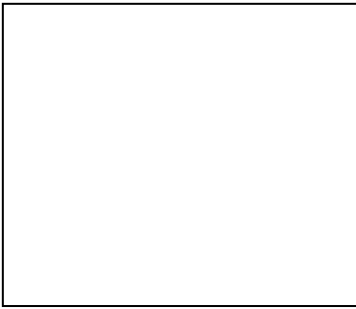
RunningEvent	
--------------	--

2. zadatak (10 bodova) Klasom `Hotel` modeliramo hotelske zgrade, s atributima kako slijedi: "oib" je porezni broj vlasnika; "city", "street" i "streetNo" predstavljaju grad, ulicu i kućni broj gdje se zgrada nalazi; "rooms" bilježi broj soba u zgradi; dok "pricePerNight" označava cijenu noćenja. Nadopunite kod kako bi omogućili dodavanje objekata klase `Hotel` u sortirane kolekcije, prirodni poredak je leksikografski po oib-u zatim po adresi (prvo grad, zatim ulica i broj). Osigurajte da je jednakost objekata konzistentna s navedenim prirodnim poretom.

U klasu nadopišite i komparatore `BY_MAX_REVENUE`, `BY_CITY` koji objekte uspoređuju po maksimalnom mogućem dnevnom prometu, tj. po gradu u kojem se nalazi.

Napomena: nije potrebno brinuti o jeziku po kojem je definiran uređaj nad tipom `String`.

```
class Hotel _____ {  
    private long oib;  
    private String city;  
    private String street;  
    private String streetNo;  
    private long rooms;  
    private double pricePerNight;  
  
    //TODO: implementirajte sve potrebne metode
```



```
public static final _____ BY_MAX_REVENUE  
= new _____ {
```

```
};  
public static final _____ BY_CITY = _____
```

Ne definirajući novi komparator nadopunite kod kako bi skup bio sortiran silazno po gradu, a zatim po maksimalnom prometu.

```
Set<Hotel> sortedHotels = new  
TreeSet<> (_____)
```



3. zadatak (10 bodova) Napišite program koji za zadani direktorij "root", kolekciju ekstenzija "ext" i direktorij "skip", briše sve datoteke danih ekstenzija u rootu i svim njegovim poddirektorijima osim u onima čije ime sadrži string "skip". Ukoliko je direktorij nakon brisanja datoteka prazan program ga treba obrisati. Na kraju izvođenja program ispisuje broj i ukupnu veličinu obrisanih datoteka kao i broj obrisanih direktorija.

NAPOMENA: Izvadak iz dokumentacije za FileVisitor se nalazi na kraju ispita.

```
public class Main {

    public static void main(String[] args) {
        Path root = Path.of("c:\\test");
        Collection<String> ext = Arrays.asList(".txt", ".clj", ".java");
        String skip = "skip";

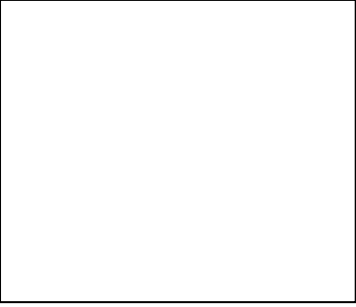
        FileVisitor<Path> visitor = new DeleteExtensions(ext,skip);

        try {
            Files._____(root,visitor);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        // TODO: ispis statistike

    }

}

public class DeleteExtensions _____ {
    // TODO: Nadopuni
}
```



```
        public FileVisitResult preVisitDirectory(Path dir,  
        BasicFileAttributes attrs {
```

```
    }
```

```
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {
```

```
    }
```

```
    public FileVisitResult postVisitDirectory(Path dir, IOException exc) {
```

```
    }
```

```
}
```

4. zadatak (10 bodova)

Imamo klasu **Item** koja predstavlja kupovni artikl. **Item** ima privatne članske varijable `String name` (naziv artikla) i `Double price` (cijena) te metode za dohvat tih varijabli: `public String getName()` i `public Double getPrice()`.

Klasa **Store** predstavlja trgovinu. **Store** ima privatne članske varijable `String name` (naziv trgovine) i `Map<Item, Integer> items` (mapa svih artikala u dućanu i njihova količina). U mapi `items` ključ je artikl (`Item`), a vrijednost je broj tih artikala prisutnih u trgovini.

Imamo na raspolaganju klasu `Loader` sa javnom statičkom metodom `Map<Integer, Store> loadMall()` koja vraća mapu parcela u trgovačkom centru i trgovina na tim parcelama. Ključ je redni broj parcele, a vrijednost je trgovina na toj parceli.

***Potrebno je napisati metodu** `double getStoreItemsValue(Store store)` koja vraća ukupnu vrijednost svih artikala u trgovini `store` (ukupna vrijednost = cijena artikla * broj artikala).

***Potrebno je napisati metodu** `List<String> getFilteredItemNames(Store store, int minQuantity)` koja vraća listu s imenima artikala kojih ima više ili jednako vrijednosti `minQuantity` u trgovini `store`.

***Potrebno je napisati metodu** `Map<String, Double> mallItemsTotalValue(Map<Integer, Store> mall)` koja vraća mapu svih artikala koji se nalaze u svim dućanima unutar trgovačkog centra `mall` i njihovu ukupnu vrijednost. Ključ mape je naziv artikla, a vrijednost je ukupna vrijednost svih tih artikala u trgovačkom centru. Metodu je potrebno implementirati tako da se napuni pomoćna mapa `itemsTotalValue` tipa `Map<String, Double>` **korištenjem metode `merge`** i da se ta mapa vrati iz metode.

JavaDoc metode `merge` iz sučelja `java.util.Map` je sljedeći:

```
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. Otherwise, replaces the associated value with the results of the given remapping function, or removes it if the result is null.

```
public class Loader {
    public static Map<Integer, Store> loadMall() {
        Map<Integer, Store> mall = new HashMap<>();
        Item tv = new Item("lg", 2500);
        Item iphone = new Item("iphone", 4999.99);
        Item mac = new Item("macBook", 12000);
        Item scooter = new Item("xiaomi", 1290);
        Item ps = new Item("ps4", 900);
        Store apple = new Store("Apple");
        Store ronis = new Store("Ronis");
        apple.addItem(iphone, 5);
        apple.addItem(mac, 3);
        ronis.addItem(tv, 12);
        ronis.addItem(ps, 8);
        ronis.addItem(scooter, 10);
    }
}
```




```
        ronis.addItem(iphone, 3);
        mall.put(1, apple);
        mall.put(2, ronis);
        return mall;
    }
}
```

```

public class Main {
    public static void main(String[] args) {
        Map<Integer, Store> mall = Loader.loadMall();
        Store apple = mall.get(1);
        Store ronis = mall.get(2);
        System.out.println("Store items total value: " + getStoreItemsValue(apple));
        //Store items total value: 60999.95
        System.out.println("Store items with quantity higher then 10: " +
            getFilteredItemNames(ronis, 10));
        //Store items with quantity larger or equal to 10: [lg, xiaomi]
        System.out.println("Total value of items in mall: "+ mallItemsTotalValue(mall));
        //Total value of items in mall: {ps4=7200.0, xiaomi=12900.0, macBook=36000.0,
        lg=30000.0, iphone=39999.92}
    }
    private static double getStoreItemsValue(Store store) {
        return store.
    }

    private static List<String> getFilteredItemNames(Store store, int minQuantity) {
        return store.
    }

    private static Map<String, Double> mallItemsTotalValue(Map<String, Store> mall) {
        Map<String, Double> itemsTotalValue = new HashMap<>();
        mall.

        return itemsTotalValue;
    }
}

```

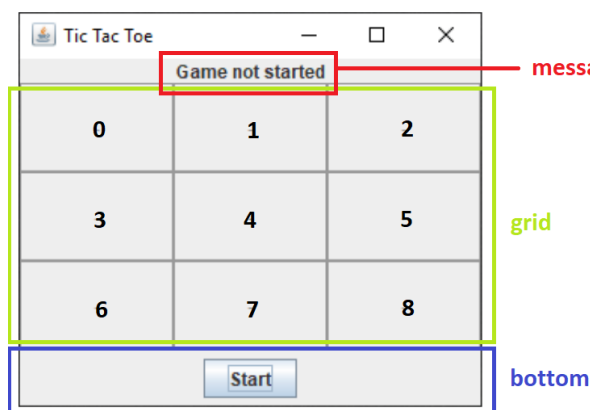
5. zadatak (10 bodova)

Imamo klasu **TicTacToe** pomoću koje je omogućeno igranje igre križić-kružić. **TicTacToe** ima privatne članske varijable:

- `List<String> fields`
- `JLabel message` - poruka korisniku na vrhu ekrana
- `JPanel grid` - polje za igru s `GridLayout` razmještajem koje u sebi sadrži 9 `JButton`-a – tipke koje korisnik treba pritisnuti kako bi odigao

potez

- `JPanel bottom` - komponenta u kojoj stoji tipka (`JButton`) za pokretanje igre
- `String player`



`List<String> fields` je polje veličine 9 u kojem program pamti odigrane poteze. Indeksi polja odgovaraju brojkama na poljima u gridu na slici lijevo. Vrijednosti polja su moguće vrijednosti varijable, koja pamti igrača trenutno na potezu - `String player`, koje mogu biti ' ', 'X' ili 'O'.

Prilikom inicijalizacije programa, tipke koje predstavljaju polja za igranje u isključene, poruka na vrhu glasi „Game not started“. Igrač mora pritisnuti tipku Start kako bi

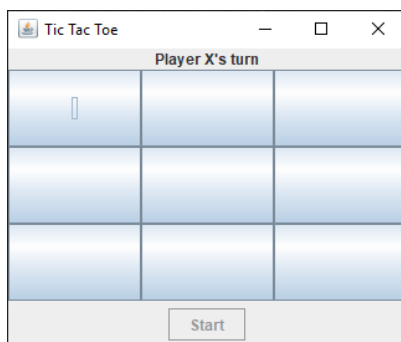
pokrenuo partiju. (Slika 1.)

Pritiskom na Start, program uključuje tipke za igranje i postavlja igrača. Poruka na vrhu glasi „Player X's turn“ ili „Player O's turn“ ovisno koji je igrač na potezu. (Slika 2.)

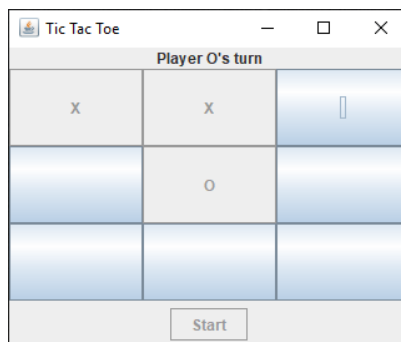
Pritiskom na tipku za igranje, tipka se isključuje, na nju se postavlja String trenutnog igrača, te se mijenja igrač. (Slika 3.)



Slika 1.



Slika 2.

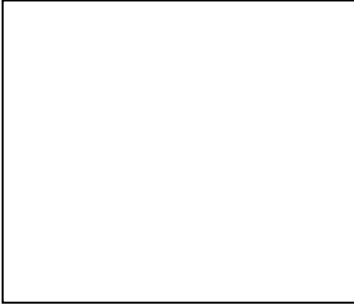


Slika 3.

Ako odigrano polje rezultira pobjedom ili izjednačenom partijom, prikazuje se pop-up s odgovarajućom porukom, te igra opet ulazi u „Game not started“ stanje. (Slika 1.)

Dio funkcionalnosti je već implementiran. Metoda `private void switchPlayer()` postavlja varijablu `player` na sljedećeg igrača i postavlja tekst poruke.

Metoda `private boolean checkWinner()` vraća `true` ako je potez rezultirao pobjedom ili `false` ako nije, tako da provjerava stanje članske varijable `fields`.



Metoda `private void gameEnd(String msg)` prikazuje pop-up s porukom `msg` za kraj igre te vraća igru na početak. (Slika 1.)

***Potrebno je dopuniti konstruktor** klase `TicTacToe` da se dobije opisani razmještaj i funkcionalnost (prepoznavanje koje je polje pritisnuto, listener, itd.)

***Potrebno je dopuniti metodu** koja se pokreće prilikom pritiska na igrača polje `private void fieldBtnListener(ActionEvent e)`. Metoda mora prepoznati koje je polje pritisnuto, koji indeks u listi `fields` odgovara tom polju, mora postaviti simbol trenutnog igrača na odigrano polje i na dobru poziciju u listi `fields` te isključiti to polje da se ne može opet pritisnuti.

***Potrebno je dopuniti metodu** `private void startBtnListener(ActionEvent e)` koja se pokreće prilikom pritiska na tipku `Start`. Metoda mora isključiti tipku `Start` i uključiti svih 9 tipki igračeg polja.

```
public class TicTacToe extends JFrame {
    private List<String> fields;
    private JPanel grid;
    private JLabel message;
    private JPanel bottom;
    private String player;

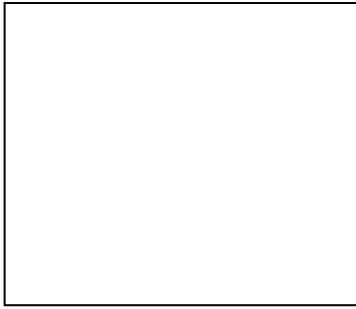
    public TicTacToe() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        setTitle("Tic Tac Toe");
        fields = Arrays.asList(" ", " ", " ", " ", " ", " ", " ", " ", " ");
        player = " ";

        message = new JLabel("Game not started");
        message.setHorizontalAlignment(SwingConstants.CENTER);
        add(message, BorderLayout.NORTH);

        grid = new JPanel();
        // TODO: postavljanje razmještaja i tipki u igračem polju

        bottom = new JPanel();
        // TODO: postavljanje donje Start tipke

    }
```



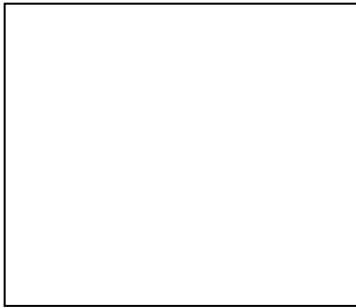
```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(() -> {  
        TicTacToe frame = new TicTacToe();  
        frame.pack();  
        frame.setVisible(true);  
    });  
}
```

```
private void fieldBtnListener(ActionEvent e) {  
    // TODO: obrada pritiska na tipku igračeg polja
```

```
    boolean win = checkWinner();  
    if(win) {  
        gameEnd("Player " + player + " won!");  
    }else if(fields.stream().filter(str -> str != " ").count() == 9){  
        gameEnd("Draw!");  
    }else {  
        switchPlayer();  
    }  
}
```

```
private void startBtnListener(ActionEvent e) {  
    // TODO: obrada pritiska na Start tipku
```

```
        switchPlayer();  
    }
```



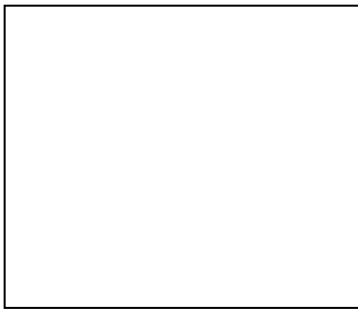
Interface FileVisitor<T>

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

Enum FileVisitResult

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

Na ovoj stranici možete napisati kod koji vam nije stao na prethodne stranice.

A large, empty rectangular box with a thin black border, intended for the user to write code that did not fit on the previous pages.