Date, enum, kolekcije, sortiranje

NAPOMENA: Vežbe podrazumevaju da je student ovladao teorijom iz dela “Uvod u objektno orijentisano programiranje - Objektno orijentisano programiranje” (1_uvod.ppt, 2_uvod.ppt i 3_uvod.ppt)

Date

Primer 1. Definisati različite datume uz pomoć klasa *LocalDate*, *LocalTime*, *LocalDateTime*, *ZonedDateTime* i formatirati ispis.

```
LocalDate currentLocalDate = LocalDate.now();
System.out.println("Trenutni datum:" + currentLocalDate);

LocalDate someLocalDate = LocalDate.of(2013, 10, 1);

someLocalDate = someLocalDate.withYear(2015).withMonth(11).withDayOfMonth(11);
System.out.println("Pojedinacna izmena datuma: " + someLocalDate);

LocalDate parseLocalDate = LocalDate.parse("17-07-2017", DateTimeFormatter.ofPattern("dd-MM-yyyy"));
System.out.println("Parsiranje datuma: " + parseLocalDate);

parseLocalDate = parseLocalDate.plusDays(1).minusYears(5);
System.out.println("Pojedinacna izmena datuma (dodavanjem, oduzimanjem) i formatiranje: "
        + parseLocalDate.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM)));

LocalTime currentLocalTime = LocalTime.now();
System.out.println("Trenutno vreme:" + currentLocalTime);

// LocalTime parseLocalTime = LocalTime.parse("12:44");
LocalTime parseLocalTime = LocalTime.parse("12:44:25");
System.out.println("Parsiranje vremena: " + parseLocalTime);

LocalDateTime currentLocalDateTime = LocalDateTime.now();
System.out.println("Trenutni datum i vreme:" + currentLocalDateTime);

LocalDateTime someLocalDateTime = LocalDateTime.of(2015, Month.OCTOBER, 11, 11, 11);

someLocalDateTime = someLocalDateTime.withYear(2015).withMonth(11).withDayOfMonth(11);
System.out.println("Pojedinacna izmena datuma i vremena: " + someLocalDateTime);

LocalDateTime parseLocalDateTime = LocalDateTime.parse("17.07.2017 17.33",
        DateTimeFormatter.ofPattern("dd.MM.yyyy HH.mm"));
System.out.println("Parsiranje datuma i vremena: " + parseLocalDateTime);

parseLocalDateTime = parseLocalDateTime.plusDays(1).minusYears(5).plusHours(10);
System.out.println("Pojedinacna izmena datuma i vremena (dodavanjem, oduzimanjem) i formatiranje: "
        + parseLocalDateTime.format(DateTimeFormatter.ofPattern("<dd.MM.yyyy HH:mm>")));

ZonedDateTime zonedDateTime = ZonedDateTime.of(LocalDateTime.now(), ZoneId.systemDefault());
System.out.println("Datum i vreme sa sistemskom zonom: " + zonedDateTime);

// Pogledati dostupne zone: System.out.println(ZoneId.getAvailableZoneIds());
System.out.println("Datum i vreme sa odabranom zonom: "
        + ZonedDateTime.of(LocalDateTime.now(), ZoneId.of("America/Barbados")));
```

Enum

Tip enum je specijalni tip podataka koji omogućava da se vrednost neke varijable ograniči na predefinisan skup konstanti. Svaki tip enumeracije sadrži numeričku i *String* reprezentaciju konstante. Pored konstanti, *enum* tip može da ima konstruktore i metode. Može se definisati u zasebnoj datoteci ili u okviru datoteke neke klase ali se onda vidljivost ograničava na paket.

Primer 2. Definisati enum tip za godišnja doba. Napomena: eksplicitno pozivanje konstruktora u *Enum*-u nije obavezujuće.

```
public enum GodisnjaDoba {
    ZIMA(1), PROLECE(2), LETO(3), JESEN(4);
    int doba;

    private GodisnjaDoba() {}
    private GodisnjaDoba(int i) { this.doba = i;}

    private String [] opis = {"zima", "proleće", "leto", "jesen"};

    @Override
    public String toString() {
        return opis[this.ordinal()];
    }
}

class Main{
    public static void main(String[] args) {
        GodisnjaDoba gd = GodisnjaDoba.PROLECE;
        System.out.println("Trenutno je "+gd);
         //vrednost varijable nema veze sa poretkom
        gd.doba++;
        System.out.println("I dalje je "+gd);
        System.out.println(GodisnjaDoba.LETO.ordinal());
        System.out.println(GodisnjaDoba.valueOf("LETO"));

        for (GodisnjaDoba doba : GodisnjaDoba.values()) {
            System.out.print(doba+" ");
        }
        switch (gd) {
        case LETO:
            System.out.println("Konačno leto!");
            break;
        default:
            System.out.println("Nije još leto.");
        }
    }
}
```

Kolekcije

Nizovi u *Javi* su posebna vrsta objekata čiji elementi nemaju nazive nego su numerisani indeksima. Elementi niza mogu biti varijable ili objekti istog tipa i smešteni su u niz sekvencijalnih memorijskih lokacija. Prednost nizova predstavlja brzina pristupa i iteriranja kroz njegove elemente. Mana nizova je njihova fiksna dužina koja se zadaje pri instanciranju. U slučaju da se niz ispuni i da se želi dodati još jedan element, mora se zauzeti drugi niz veće veličine, prekopirati prethodni sadržaj i tek onda dodati novi element.

Kolekcije rešavaju problem fiksne dužine nizova uvođenjem različitih načina organizacije podataka. Poseduju mehanizme skladištenja i manipulisanja (dodavanje, brisanje, pretraga, sortiranje itd.) grupe objekata određenog tipa. Postoji više koncepata i implementacija kolekcija, neke od najčešće korišćenih su prikazane na sledećoj slici:

Implementacija / Koncept	Hash table	Resizable Array	Balanced Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Implementacije kolekcija zasnovane na *Set* konceptu nemaju poredak dodatih elemenata i sadrže samo jedinstvene vrednosti koje se isčitavaju iteratorom a ne indeksom. Jedna od implementacija je *HashSet* koji unetim vrednostima pristupa putem heš (engl. *hash*) koda. Za razliku od *HashSet* implementacije, *LinkedHashSet* unutar sebe čuva poredak dodavanja i omogućava iteriranje po redosledu dodavanja. Implementacija *TreeSet* sortira elemente kako se dodaju.

Ukoliko se dodaju elementi adresnog tipa, onda se unutar date klase moraju preklopiti metode *equals* i *hasCode*(za *HashSet i LinkedHashSet*) i implementirati *Comparable* interfejs (za *TreeSet*).

Primer 3. Napraviti kolekcije proizvoljnih imena zasnovanu na *Set* konceptu. Isprobati metode za manipulaciju elementima kolekcije. Ispisati uneta imena koristeći klasu *Iterator*.

```
HashSet<String> imena = new HashSet<String>();
//LinkedHashSet<String> imena = new LinkedHashSet<String>();
//TreeSet<String> imena = new TreeSet<String>();

imena.add("Marko");
imena.add("Janko");
imena.add("Marko");
imena.add("Branko");

Iterator<String> itr = imena.iterator();
while (itr.hasNext()) {
    System.out.println(itr.next());
}
```

Implementacije kolekcija zasnovane na *List* konceptu imaju uređen poredak dodatih elemenata i vrednosti se isčitavaju indeksom. Jedna od implementacija je jednostruko spregnuta lista *ArrayList* koja koristi dinamički niz za smeštanje vrednosti koje mogu biti i duplirane. Iščitavanje vrednosti se vrši veoma brzo ali je dodavanje na kraj uslovno brže nego na početak ili sredinu liste. *LinkedList* je implementirana kao dvostruko spregnuta lista. Pogodna za manipulaciju vrednosti tokom sekvencijalnog iteriranja od početka ili kraja, dok je nasumičan pristup veoma spor.

Primer 4. Napraviti kolekcije proizvoljnih imena zasnovanu na *List* konceptu. Isprobati metode za manipulaciju elementima kolekcije. Ispisati uneta imena koristeći klasu *Iterator* ili for petlju.

```
//ArrayList<String> imena = new ArrayList<String>();
LinkedList<String> imena=new LinkedList<String>();

imena.add("Marko");
imena.addFirst("Janko");
imena.addLast("Marko");
imena.add("Branko");
imena.removeFirst();

Iterator<String> itr = imena.iterator();
while (itr.hasNext()) {
    System.out.println(itr.next());
}
```

Implementacije kolekcija zasnovane na *Map* konceptu sadrže podatke bazirane na paru ključ-vrednost. Ključevi su namapirani na heš kod koji jednoznačno određuje memorijsku lokaciju traženog podatka. Pristupanje podacima se vrši po ključu koji je jedinstven i brzo se razlučuje postojanje određenog elementa. Jedna od implementacija je

HashMap koja ne čuva poredak dodatih elementa i samim tim se ne može sortirati. Za razliku od *HashMap* implementacije, *LinkedHashMap* unutar sebe čuva poredak dodavanja dok implementacija *TreeMap* sortira elemente kako se dodaju.

Primer 5. Napraviti kolekcije studenata zasnovanu na *Map* konceptu. Za ključeve uzeti studentske indekse a za vrednost čuvati imena studenata. Isprobati metode za manipulaciju elementima kolekcije. Ispisati uneta imena koristeći klasu *Iterator* ili foreach petlju.

```
//HashMap<String, String> studenti = new HashMap<String, String>();
// LinkedHashMap<String,String> studenti=new LinkedHashMap<String,String>();
TreeMap<String,String> studenti=new TreeMap<String,String>();

studenti.put("SW12345/2015", "Marko Markovic");
studenti.put("SW12345/2015", "Marko Markovicc");
studenti.put("SW12346/2015", "Petar Petrovic");
studenti.put("SW12347/2015", "Jovan Jovanovic");
String indeks = "SW12345/2015";
System.out.println("Student sa indeksom " + indeks + " je " + studenti.get(indeks));
studenti.remove(indeks);

for (String s : studenti.values()) {
    System.out.println(s);
}

for (Map.Entry<String,String> m : studenti.entrySet()) {
    System.out.println(m.getKey() + " " + m.getValue());
}
```

Sortiranje

Podaci koji se unose u kolekcije mogu biti smešteni u različitom poredku u odnosu na dodavanje elemenata. Nekada je potrebno te podatke sortirati po određenom redosledu i tek nakon toga iščitavati podatke. Postoje kolekcije koje su implementirane tako da elemente pri dodavanju ujedno i sortiraju u rastućem redosledu. Mada i takvi tipovi kolekcija neće uspešno obaviti sortiranje ako se radi sa adresnim tipovima. Potrebno je implementirati interfejs *Comparable* i redefinisati metodu *compareTo* da poredi vrednosti atributa neke klase a ne njenu adresu. Drugi način je da se definiše zasebna klasa za sortiranje koja će implementirati interfejs *Comparator* i redefinisati metodu *compare* da poredi više različitih tipova objekata i njenih atributa.

Primer 6. Napraviti klasu Knjiga koja sadrži atribute šifra, naziv, cena i godina publikovanja. Sortirati knjige po nazivu, u rastućem i opadajućem redosledu, korišćenjem metode *sort* klase *Collections*.

```
public class Knjiga implements Comparable{

    protected String    sifra;
    protected String    naziv;
    protected Double    cena;
    protected Integer   godinaPublikovanja;

    public Knjiga(String sifra, String naziv, double cena,
            Integer godinaPublikovanja) {
        super();
        this.sifra = sifra;
        this.naziv = naziv;
        this.cena = cena;
        this.godinaPublikovanja = godinaPublikovanja;
    }

    @Override
    public int compareTo(Object o) {
        Knjiga objKnjiga = (Knjiga) o;
        return this.naziv.compareTo(objKnjiga.naziv);
    }
}

class Main {
    @SuppressWarnings("unchecked")
    public static void main(String[] args) {
        ArrayList <Knjiga> list = new ArrayList<Knjiga>();
        list.add(new Knjiga("001","Harry Potter and the Goblet of Fire", 1400.0, 2000));
        list.add(new Knjiga("002","Harry Potter and the Deathly Hallows", 2000.0, 2007));
        list.add(new Knjiga("003","A Game of Thrones", 1200.00, 1998));

        System.out.println("Sortiranje knjiga rastuce");
        Collections.sort(list);
        for (Knjiga knjiga : list) {
            System.out.println(knjiga);
        }
    }
}
```

Primer 7. Modifikovati prethodni primer (primer 6) tako da se zasebno definiše klasa za sortiranje koja će sortirati više tipova objekata po više tipova parametara. U tom slučaju koristiti drugu implementaciju metode *sort* klase *Collections*.

```
public class OsnovniSorter implements Comparator{

    int direction = 1;

    public int compare(Object o1, Object o2) {
        int retVal = 0;
        if(o1!= null && o2!=null && o1 instanceof Knjiga && o2 instanceof Knjiga){
            Knjiga objKnjiga1 = (Knjiga) o1;
            Knjiga objKnjiga2 = (Knjiga) o2;
            retVal = objKnjiga1.getNaziv().compareTo(objKnjiga2.getNaziv());
        }
        return retVal * direction;
    }
}
```

Zadaci

Kreirati klase Osoba, Student, Profesor, Ocena, Predmet i obezbediti da su svi atributi zaštićeni, da postoje više konstuktora (bez parametara, parametri koji su atributi klase, referenca na objekat), korisničke metode i set/get metode za atibute klase i metodu koja parsira liniju stringa i vraća objekat. Sve klase sadrže atribut aktivnost (tipa boolean) koji označava logičko brisanje. Pri brisanju objekta obratiti pažnju (zabraniti brisanje) na slučaj kad neki drugi objekat ima referencu na brisani objekat.

Klasa Osoba predstavlja apstrakciju svih osoba i potrebno je da ima sledeće atribute: ime (tipa *String*), prezime (tipa *String*), jmbg (tipa *String*).

Klasa Student predstavlja specijalizaciju osobe i potrebno je da ima sledeće dodatne atribute: indeks (tipa *String*), datum upisa (tipa *Date*), predmeti (tipa lista Predmeta), ocene (tipa lista Ocena), prosek (tipa double).

Klasa Profesor predstavlja specijalizaciju osobe i potrebno je da ima sledeće dodatne atribute: broj lične karte (tipa *String*), datum zapošljenja (tipa *Date*), titula (tipa *enum*) i predmeti (tipa lista Predmeta).

Klasa Predmet potrebno je da ima sledeće atribute: sifra predmeta(tipa *String*), naziv predmeta (tipa *String*), semestar (tipa *enum*), profesor (tipa *Profesor*).

Klasa Ocena potrebno je da ima sledeće atribute: predmet (tipa *Predmet*) i ocena (tipa *int*).

Testirati klase na primeru fakulteta i omogućiti izbor 4 osnovne opcije - dodavanje, brisanje, izmena i prikaz. Dodatno omogućiti opciju pretrage studenata (po indeksu, godini upisa, proseku i imenu i prezimenu) i predmeta (po šifri, semestru i profesoru). Podatke čuvati u bar tri različite kolekcije (implementacije) i sortirati po ugledu na primer 7.

Omogućiti sledeće funkcionalnosti:

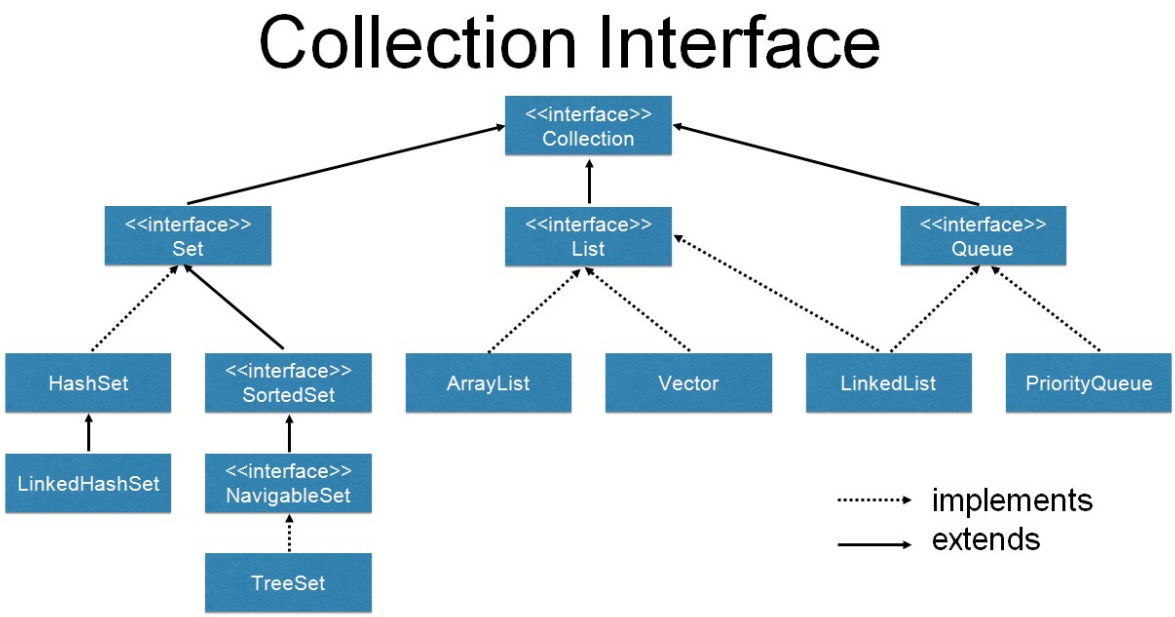
1. rad sa predmetima
   a. unos podataka o novom predmetu
   b. izmena podataka o predmetu (odabrati predmet na osnovu šifre)
   c. brisanje podataka o predmetu (logicko brisanje, odabrati predmet na osnovu šifre)
   d. ispis podataka svih predmeta u formi (naziv, šifra, profesor, semestar) u odnosu na status zapisa:
      i. svi predmeti
      ii. samo aktivni
      iii. samo obrisani
   e. ispis podataka o određenom predmetu u formi (naziv, šifra, profesor, semestar) (odabrati predmet na osnovu šifre)
2. rad sa studentima

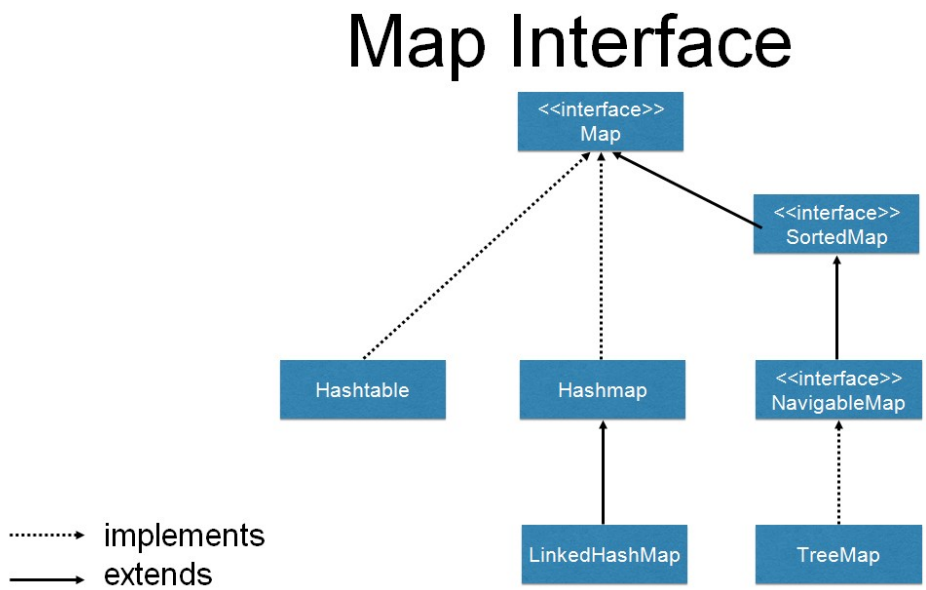
a. unos podataka o novom studentu
b. izmena podataka o studentu (odabrati studenta na osnovu indeksa)
c. brisanje podataka o studentu (logicko brisanje, odabrati studenta na osnovu indeksa)
d. ispis podataka svih studenata u formi (broj indeksa, ime i prezime, godina upisa studija, prosek) u odnosu na status zapisa:
    i. svi studenti
    ii. samo aktivni
    iii. samo obrisani
e. ispis podataka o određenom studentu u formi (broj indeksa, ime i prezime, godina upisa studija, prosek) i ispis položenih i nepoloženih ispita za studenta u formi (ocena, predmet, student) (odabrati studenta na osnovu indeksa).
f. unos ocena studenta (odabrati studenta na osnovu indeksa, odabrati predmet na osnovu šifre predmeta), računanje proseka studenata
g. sotiranje studenata po
    i. datumu upisa studija (rastuće)
    ii. imenu i prezimenu (rastuće)
    iii. broju položenih ispita (opadajuće)
    iv. broju nepoloženih ispita (rastuće)
    v. prosecnoj oceni (opadajuće)

3.rad sa profesorima

h. unos podataka o novom profesoru
i. izmena podataka o profesoru (odabrati profesora na osnovu broju lične karte)
j. brisanje podataka o profesoru (logicko brisanje, odabrati profesora na osnovu broja lične karte)
k. ispis podataka svih profesora u formi (broj lične karte, titula, ime i prezime) u odnosu na status zapisa:
    iv. svi profesori
    v. samo aktivni
    vi. samo obrisani
l. ispis podataka o određenom profesoru u formi broj lične karte, titula, ime i prezime) i ispis predmenta koje predaje profesor u formi (naziv, šifra, semestar) (odabrati profesora na osnovu broja lične karte).
m. unos predmeta profesora (odabrati predmet na osnovu šifre predmeta)
n. sotiranje profesora po
    vi. datum zapošljenja (opadajuće)
    vii. imenu i prezimenu (rastuće)
    viii. broju predmeta koje predaje (rastuće)
    ix. semestru prvog predmeta (opadajuće)

Dodatno


Collection Interface
<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
HashSet
<<interface>>
SortedSet
ArrayList
Vector
LinkedList
PriorityQueue
LinkedHashSet
<<interface>>
NavigableSet
TreeSet
implements
extends


Map Interface
<<interface>>
Map
<<interface>>
SortedMap
Hashtable
Hashmap
<<interface>>
NavigableMap
LinkedHashMap
TreeMap
implements
extends

* ilustracije preuzete sa adrese *www.dzone.com*