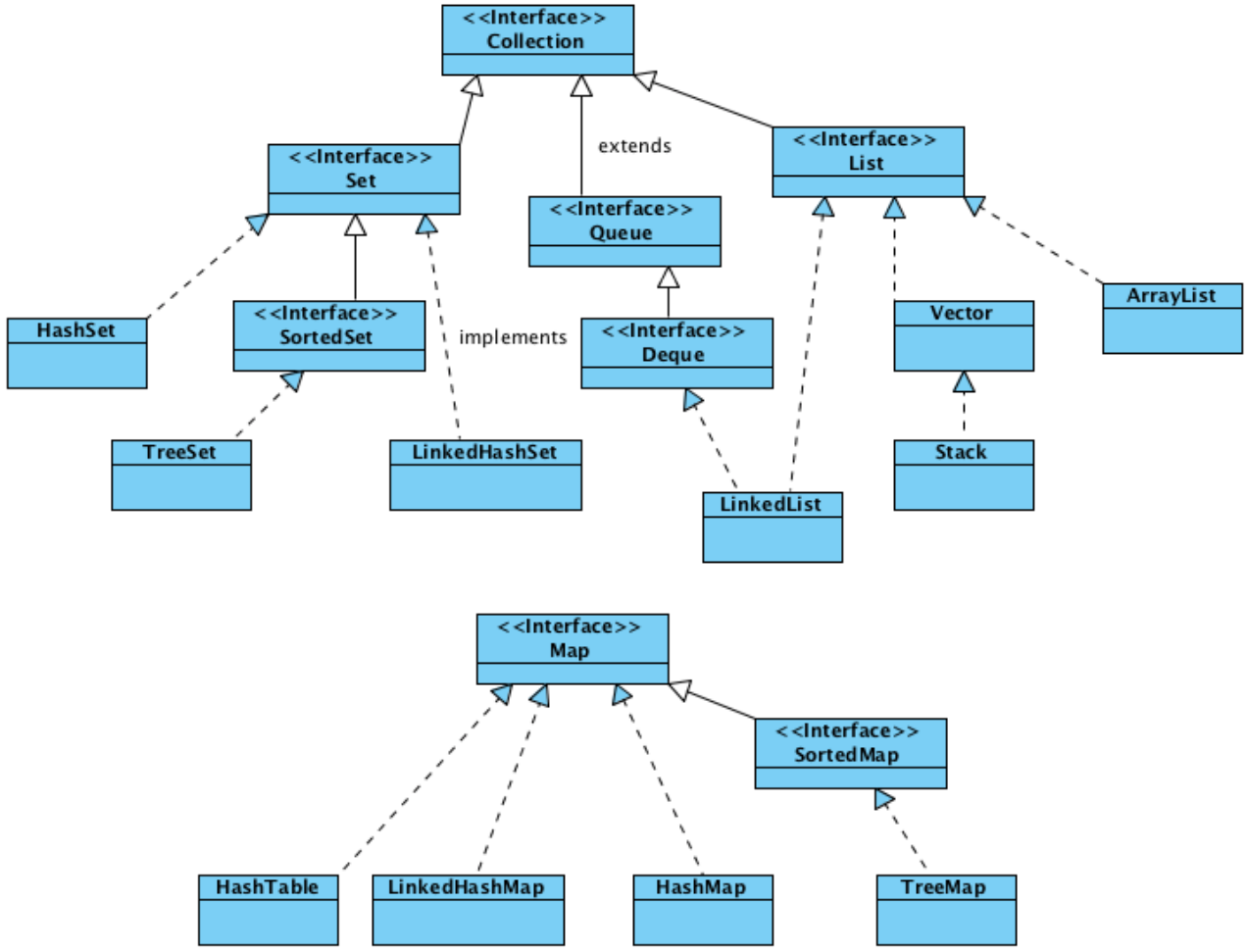
Java kolekcije

Od Java 2 platforme (Java verzija 1.2), Java programiski jezik poseduje robusnu platformu (API) za rukovanje kolekcijama podataka nazvanu *Java Collections Framework*. Platforma se zasniva na skupu osnovnih interfejsa i implementacija koji omogućavaju upotrebu standardnih struktura podataka.

Osim implementacije struktura podataka, platforma omogućava i implementaciju osnovnih algoritama za manipulaciju podacima. Struktura platforme je prikazana na slici ispod.



Korenski interfejs Collection pruža sledeće metode za rukovanje podacima:

Ime metode	Opis
`add(Object obj)`	Dodavanje elementa u kolekciju.
`addAll(Collection c)`	Dodaje sve elemente prosleđene kolekcije.
`clear()`	Brisanje svih elemenata.
`contains(Object obj)`	Provera da li kolekcija sadrži prosleđeni element.
`containsAll(Collection c)`	Provera da li kolekcija sadrži sve elemente prosleđene kolekcije.
`isEmpty()`	Provera da li je kolekcija prazna.
`iterator()`	Vraća iterator kroz elemente kolekcije.
`remove(Object obj)`	Uklanjanje elementa. Postoji i varijanta `removeAll(Collection c)`
`size()`	Broj elemenata u kolekciji.
`toArray()`	Pretvaranje kolekcije u niz.

Osnovni tipovi struktura podataka su predstavljeni interfejsima koji direktno implementiraju korenski interfejs `Collection`. To su:

1. **Set** – Predstavlja neuredjeni skup jedinstvenih podataka.
2. **List** – Struktura podataka tipa liste
3. **Map** – Struktura podataka tipa mape (rečnika)

Liste

Liste predstavljaju dinamičke strukture uređenih podatka pri čemu elementi liste ne moraju da budu jedinstveni. Postoji nekoliko implementacija List interfejsa, neki od njih su:

- **`ArrayList`** – Implementacija liste koja je bazirana na dinamičkom nizu. U sebi sadrži niz objekata (`Object[]`) koji služi za smeštanje podataka. Lista se brine o ažuriranju dimenzije niza prilikom dodavanja i uklanjanja elemenata. Zbog brzine pristupa elementima često se koristi kao kontejner podataka opšteg tipa, međutim odlikuje je nešto sporije umetanje i brisanje podataka unutar liste.
- **`LinkedList`** – Implementacija liste koja je optimizovana za sekvencijanli pristup elementima. U odnosu na `ArrayList`, ova implementacija pruža bolje performanse prilikom dodavanja i uklanjanja elemenata iz sredine liste, ali je sporija prilikom direktnog pristupa.

Primer rukovanja ArrayList kolekcijom

1. **Kreiranje nove liste**

```
ArrayList<String> listaStringova = new ArrayList<String>();
```

2. **Dodavanje elemenata**

```
listaStringova.add("Prvi string");
listaStringova.add("Drugi string");

// Može se dodadi i referenca na postojeći objekat
String treci = "Treci string";
listaStringova.add(treci);
```

3. **Uklanjanje elemenata**

```
// Može preko indeksa
listaStringova.remove(0);

// Može i preko reference
listaStringova.remove(treci);
```

4. **Pristup elementima**

```
// Može preko klasične for petlje
for (int i = 0; i < listaStringova.size(); i++) {
	String el = listaStringova.get(i);
	System.out.println(el);
}

// Može preko for each petlje
for(String el : listaStringova) {
	System.out.println(el);
}

// Može preko Iterator intefejsa
Iterator<String> iterator = listaStringova.iterator();
while(iterator.hasNext()) {
	String el = iterator.next();
	System.out.println(el);
}
```

Mape

Mape predstavljaju strukture podataka koje elemente čuvaju u parovima ključ-vrednost, pri čemu vrednost predstavlja konkretan element mape, dok ključ jedinstveno određuje elemente. Zbog ovoga, elementima mape se ne pristupa preko indeksa, već preko ključa.

Najčešće korišćene implementacije mapa u Javi su:

- **`HashMap`** – Vrlo efikasna prilikom pristupa pojedinačnim elementima za čije proračunavanje koristi mapu heš kodova. Postoji i varijanta **`LinkedHashMap`**.
- **`TreeMap`** – Implementacija sortirane mape. Način sortiranja je određenen Sortable objektom koji se može proslediti kao parametar konstruktora. Zbog brige o sortitanju podataka ovaj tim mape pokazuje slabije performanse od **`HashMap`** varijante.

Primer rukovanja `HashMap` kolekcijom

1. Kreiranje mape

```
HashMap<Integer, String> mapaStringova = new HashMap<Integer, String>();
```

2. Dodavanje elemenata

```
mapaStringova.put(0, "prvi string");
mapaStringova.put(1, "drugi string");

String treci = "treci string";
mapaStringova.put(2, treci);
```

NAPOMENA: Metoda **`put()`** dodaje novi element sa zadanim ključem u mapu. Ukoliko prosleđeni ključ već postoji u mapi, vrednost elementa sa tim ključem će biti zamenjena prosleđenom vrednošću.

3. Uklanjanje elemenata

```
// Ukanja element sa zadanim ključem
// Vraća null ako ne postoji
mapaStringova.remove(2);

// Uklanja element sa zadanim ključem i vrednošću
// Vraća false ako taj par ne postoji
mapaStringova.remove(1, "drugi string");
```

4. Iteracija kroz elemente

1. Skup ključeva u mapi možemo dobiti pozivom metode **`keySet()`**. Ukoliko nam je potrebna iteracija samo kroz ključeve mape, kroz ovaj skup možemo proći nekom od standardnih metoda za iteraciju (**`for`**, **`foreach`**, **`while`**):

```
for(Integer key : mapaStringova.keySet()) {
	System.out.println(key);
}
```

2. Listu vrednosti u mapi možemo dobiti pozivom metode **`values()`**. Iteracija kroz vrednosti se vrši iteracijom kroz ovu listu:

```
for(String value : mapaStringova.values()) {
	System.out.println(value);
}
```

3. Ukoliko su nam potrebni celi parovi ključ-vrednost (**`Entry`**), skup ovih vrednosti

možemo dobiti pozivom metode **`entrySet()`**:

```
Iterator iterator = mapaStringova.entrySet().iterator();
while(iterator.hasNext()) {
    Entry<Integer, String> entry =
                 (Entry<Integer, String>) iterator.next();
    Integer key = entry.getKey();
    String       val = entry.getValue();
    System.out.println("Ključ: " + key + ", Vrednost: " + val);
 }
```

4. Alternativno, za dobijanje parova ključ-vrednost možemo iskoristiti iteraciju kroz ključeve i u svakoj iteraciji dobaviti vrednost za zadani ključ.

```
for(Integer key : mapaStringova.keySet()) {
    String value = mapaStringova.get(key);
    System.out.println(key + ":" + value);
}
```

Zadatak

1. Kreirati program za učitavanje korisnika u listu iz tekstualne datoteke. Svaki korisnik je predstavljen imenom, prezimenom, jmbg-om, korisničkim imenom i šifrom. Podaci o korisnicima se čuvaju u tekstualnoj datoteci u pojedinačnim linijama.

   Prilikom pokretanja programa, na osnovu podataka iz datoteke kreirati listu objekata klase Korisnik i ispisati u konzolu sadržaj liste.

Algoritmi

Sortiranje kolekcija

Java Collections API poseduje određeni broj statičkih funkcija koje predstavljaju implementaciju osnovnih algoritama za rad sa funkcijama. Jedan od primera je funkcija **`sort()`** koja služi za sortiranje elemenata u kolekciji.

Sortiranje kolekcije primitivnih tipova

Za sortiranje kolekcije primitivnih tipova (uključujući i tip **`String`**), dovoljno je proslediti željenu kolekciju metodi **`sort()`**:

```
Collections.sort(listaStringova);
```

Sortiranje liste proizvoljnih objekata

Ukoliko imamo listu objekata neke klase koju smo mi napisali, potrebno je da ručno obezbedimo implementaciju kriterijuma za sortiranje.

Pretpostavimo da imamo klasu `Student` koja između ostalih atributa ima atribut prosek. Ukoliko bi hteli da sortiramo listu objekata klase `Student` po proseku, prvi korak je da klasu `Student` proširimo implementacijom interfejsa `Comparable` i u njoj redefinišemo metodu `compareTo()`.

```
class Student implements Comparable<Student> {
	...

	@Override
	public int compareTo(Student o) {
		// Za opadajuće sortiranje, obrnemo operande
		return (int)(this.prosek - o.prosek);
	}
}
```

Ukoliko želimo da sortiramo po tekstualnom obeležju, možemo iskoristiti metodu compareTo() klase String:

```
@Override
public int compareTo(Student o) {
	return this.getIme().compareTo(o.getIme());
}
```

Nakon ovoga možemo sortirati listu studenata po proseku ili imenu:

```
Collections.sort(listaStudenata);
```

DODATAK: Rad sa datumima i enumeracijama

Datumi

Tip podataka koji predstavljaju datume (sa ili bez vremenske koponente) je predstavljen klasom **`GregorianCalendar`** iz paketa **`java.util`**. Kreiranje objekta ove klase upotrebom konstruktora bez parametara će kreirati **`GregorianCalendar`** instancu sa postavljenim trenutnim datumom i vremenom očitanim sa lokalnog računara;

```
GregorianCalendar danas = new GregorianCalendar();
```

Osim ovoga, moguće je kreirati i **GregorianCalendar** objekat sa željenim datumom i vremenom prosleđivanjem ciljanih vrednosti konstruktoru:

```
// Postavlja datum i vreme na 15.02.2015 15:45
GregorianCalendar dan = new GregorianCalendar(2015, 2, 15, 15, 45);
```

Klasa **GregorianCalendar** omogućava različite manipulacije datumima, kao što su upoređivanje, dodavanje vrednosti na pojedinačne koponente datuma i slično.

Za ispis datuma u željenom formatu koristi se klasa **`SimpleDateFormat`**:

```
SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
GregorianCalendar danas = new GregorianCalendar();
format.format(danas.getTime());
```

Kompletan spisak oznaka za pojedinačne komponente datuma možete pogledati ovde.

Ukoliko želimo dobiti **`GregorianCalendar`** instancu iz Stringa, koristimo **`parse`** metodu **`SimpleDateFormat`** klase:

```
String danString = "15.02.2015";
GregorianCalendar dan = new GregorianCalendar();
try {
	dan.setTime(format.parse(danString));
} catch (ParseException e) {
	e.printStackTrace();
}
```

NAPOMENA Prilikom parsiranja datuma iz teksta format u tekstualnom obliku mora odgovarati formatu navedenom u konstruktoru `SimpleDateFormat` klase inače se dobija `ParseException` izuzetak.

Enumeracije

Enumeracije predstavljaju tipove podataka koji mogu da uzimaju ograničen skup predefinisanih vrednosti (tzv. *nabrojivi tipovi*). Kako bi programski kod u projektu bio što modularniji, sve enumeracije ćemo pisati u posebnim datotekama.

Primer enumeracije koja predstavlja pol osobe (datoteka `Pol.java`):

```
public enum Pol {
	ZENSKI,
	MUSKI
}
```

Zbog lakšeg čuvanja vrednosti enumeracija u tekstualnim datotekama i njihovog čitanja, umesto konkretnih vrednosti enumeracija, čuvaćemo njihove redne brojeve (indekse) počevši od nule. Za ove potrebe biće nam korisne dve funkcije Java enumeracija:

- **`values()`** – Funkcija koja vraća Java niz sa svim vrednostima enumeracije nad kojom je pozvana,
- **`ordinal()`** – Funkcija koja vraća indeks vrednosti enumeracija nad kojom je pozvana

Primeri:

- Ako imamo indeks 0 koji smo pročitali iz fajla, vrednost `ZENSKI` enumeracije `Pol` mozemo dobiti na sledeći način:

```
Pol.values()[0]
```

- Ukoliko nam treba indeks vrednosti `MUSKI` enumeracije `Pol`:

```
Pol.MUSKI.ordinal()
```

Enumeracije se mogu kreirati putem *File* menija Eclipse okruženja (*File* → *New* → *Enum*).

Zadaci

1. Proširiti zadatak sa korisnicima tako da omogućava vođenje evidencije o prometu jedne prodavnice računara. Korisnici aplikacije su zaposleni u prodavnici, a prodavnica prodaje pojedinačne komponente i gotove konfiguracije računara.

   Svi podaci se čuvaju u tekstualnim datotekama (korisnici.txt, komponente.txt, konfiguracije.txt).

   Kreirati sledeće klase:

   1. **Komponenta** - Opisana sledećim atributima: šifra, naziv, cena, raspoloživa količina, opis, kategorija (moguće kategorije su: računarske komponente, periferni uređaji, dodatna oprema).
   2. **Konfiguracija** - Opisana atributima: šifra, naziv, cena, raspoloživa količina, opis, lista komponenti
   3. **Prodavnica** – Klasa koja sadrži liste zaposlenih i proizvoda i metode za učitavanje I snimanje podataka u datoteke.
   4. **ProdavnicaMain** – Glavna klasa sa main metodom.

   Kada se program pokrene, učitati sve podatke iz datoteka i tražiti od korisnika da se prijavi na sistem. Nakon prijave, ispisati korisniku listu svih proizvoda u radnji.