Programski Sistemi

Generičko programiranje

Generičko programiranje

- Pisanje koda kojim se obezbeđuje uniformna obrada struktura podataka različitih tipova
- Izbegava se pisanje koda koji u suštini radi isto, ali sa podacima razlitih tipova: poštovanje principa DRY
- Klase, interfejsi i metodi u Javi mogu biti definisani s parametrima koji predstavljaju (klasne) tipove podataka.

Generičko programiranje

- Dinamički niz tipa **`ArrayList`** može imati elemente bilo kog tipa, jer se svaki niz tipa **`ArrayList`** sastoji zapravo od elemenata tipa **`Object`**.
- Tip elemenata dinamičkog niza može se suziti, tj. parametrizovati tako da bude npr. **`String`**, tako što se konstruiše dinamički niz tipa **`ArrayList<String>`**

Generičko programiranje

- Prednosti parametrizovanja:
  - Poboljšava čitljivost programa
  - omogućava rano otkrivanje grešaka i tako program čini pouzdanijim.

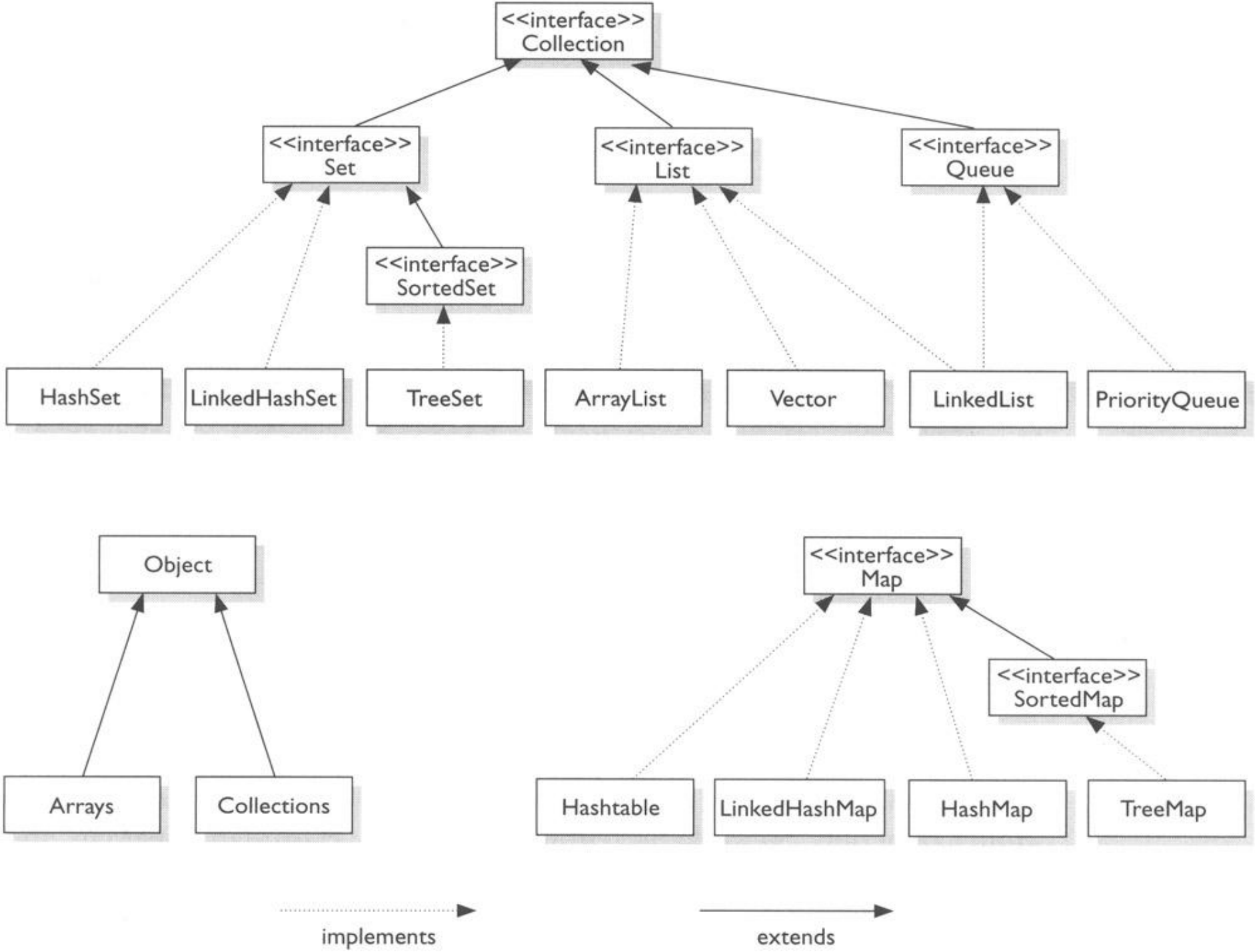
```
ArrayList<Integer> a =
  new ArrayList<Integer>();
a[0] = "greška"; //greška koja se
  otkriva još u fazi prevođenja
```

Generičko programiranje

- Klasa **`ArrayList`** je samo jedna od brojnih klasa iz Javinog API-ja generičkih struktura podataka,
  tzv. kolekcija
- Od verzije 1.5 parametrizovane su sve klase kolekcija, ali je i dalje moguće koristiti neparametrizovane verzije

Generičke strukture podataka

- Dele se u dve grupe: kolekcije i mape
- Kolekcija je skup objekata, bez obraćanja mnogo pažnje na njihove moguće dodatne međusobne odnose.
- Mapa je struktura podataka u kojoj se može prepoznati preslikavanje objekata jednog skupa u objekte drugog skupa.


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
<<interface>>
SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
Arrays
Collections
<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends

Kolekcije

- Kolekcije objekata u Javi se dalje dele u dve podvrste: liste i skupove.
- Lista je kolekcija u kojoj je redosled elemenata definisan.
- Osnovno svojstvo skupa kao kolekcije objekata je da ne postoje duplikati objekata u skupu.

Kolekcije

- Liste i skupovi su predstavljeni parametrizovanim interfejsima **`List<T>`** i **`Set<T>`** koji implementiraju interfejs **`Collection<T>`**
- Interfejs **`Collection<T>`** definiše opšte operacije koje se mogu primeniti na svaku kolekciju
- interfejsi **`List<T>`** i **`Set<T>`** definišu dodatne operacije koje su specifične samo za liste i skupove.

Kolekcije

- Generičkim kolekcijama u Javi ne mogu pripadati vrednosti primitivnih tipova nego samo klasnih tipova.
- Sistematično „posećivanje” svih objekata u kolekciji nekim redom, počinjući od nekog objekta u kolekciji i prelazeći s jednog objekta na drugi dok se ne iskoriste svi objekti postiže se pomoću iteratora

Kolekcije

- Iterator je objekat koji služi za prelazak s jednog objekta na drugi u kolekciji.
- Ako je k kolekcija, onda poziv **`k.iterator()`** kao rezultat daje iterator koji se može koristiti za pristup svim objektima kolekcije **k.**

```
// Dobijanje iteratera za kolekciju
Iterator<String> iter = k.iterator();

// Pristupanje svakom elementu kolekcije po redu
while (iter.hasNext()) {
    String elem = iter.next();
    System.out.println(elem);
}
```

Kolekcije

- Uklanjanje elemenata iz kolekcije:

```
Iterator<File> iter = k.iterator();

while (iter.hasNext()) {
    File elem = iter.next();
    if (elem == null)
        iter.remove()
}
```

Kolekcije

- Od verzije 1.5 umesto iteratora bolje je koristiti for each petlju
- Prikazivanje imena svih datoteka u kolekciji dir tipa **`Collection<File>`**:

```
for (File datoteka : dir) {
   if (datoteka != null)
      System.out.println(datoteka.getName());
}
```

Liste

- Konkretna implementacija interfejsa List: **`ArrayList`** i **`LinkedList`**
- **`ArrayList<T>`** je linearno uređena lista objekata tipa T koji se čuvaju kao elementi dinamičkog niza, odnosno dužina niza se automatski povećava prilikom dodavanja novih objekata u listu.

Liste

- Objekat tipa **`LinkedList<T>`** takođe predstavlja linearno uređenu listu objekata tipa **`T`**, ali se ti objekti čuvaju kao čvorovi koji su međusobno povezani pokazivačima (referencama).
- **`LinkedList<T>`** je efikasnija od **`ArrayList<T>`** u primenama kod kojih se elementi liste često dodaju ili uklanjaju u sredini liste.

Skupovi

- Skup je kolekcija objekata u kojoj nema duplikata, odnosno nijedan objekat u skupu se ne pojavljuje dva ili više puta.
- Ako je **s** objekat tipa **Set<T>**, onda naredba **s.add(o)** ne proizvodi nikakav efekat ukoliko se objekat **o** već nalazi u skupu **s**

Skupovi

- Praktična reprezentacija opšteg pojma skupa elemenata u Javi zasniva se na binarnim stablima i heš tabelama.
- Ova dva načina obuhvaćena su dvema konkretnim generičkimklasama **`TreeSet<T>`** i **`HashSet<T>`** u paketu **`java.util`**

Skupovi

- Skup predstavljen klasom **`TreeSet`**
  ima dodatnu osobinu da su njegovi elementi uređeni u rastućem redosledu. Pored toga, iterator za takav skup uvek prolazi kroz elemente skupa u ovom rastućem redosledu.
- To ograničava vrstu objekata koji mogu pripadati skupu tipa **`TreeSet`**,
  jer se njegovi objekti moraju upoređivati

TreeSet

- Praktično, objekti u skupu tipa **`TreeSet<T>`** moraju implementirati interfejs **`Comparable<T>`** tako da relacija **`o1.compareTo(o2)`** bude definisana na razuman način za svaka dva objekta **`o1`** i **`o2`** u skupu.

HashSet

- Elementi skupa u strukturi **`HashSet<T>`** čuvaju se u strukturi podatak koja se naziva *heš tabela.* Ona obezbeđuje vrlo efikasne operacije nalaženja, dodavanja i uklanjanja elemenata, mnogo brže od sličnih operacija u **`TreeSet`**.
- Objekti skupa **`HashSet`** nisu uređeni u posebnom redosledu i ne moraju da implementiraju interfejs **`Comparable`**.

Mape

- Mapa je uopštenje matematičkog pojma preslikavanja ili funkcije.
- Mapa koncepcijski podseća na niz, osim što za indekse ne služe celi brojevi nego objekti proizvoljnog tipa.
- Objekti koji u mapi služe kao indeksi nazivaju se *ključevi,* dok se objekti koji su pridruženi ključevima nazivaju *vrednosti.*

Mape

- Svaki ključ može odgovarati najviše jednoj vrednosti, ali jedna vrednost može biti pridružena većembroju različitih ključeva.
- U Javi, mape su predstavljene interfejsom **`Map<T,S>`** iz paketa **`java.util`**.

Mape

- **`Map<T,S>`** je parametrizovan dvoma tipovima: prvi parametar T određuje tip objekata koji predstavljaju ključeve mape, a drugi parametar S određuje tip objekata koji su vrednosti mape.
- Npr. mapa tipa **`Map<Date,Boolean>`** definiše preslikavanje ključeva tipa **`Date`** u vrednosti tipa **`Boolean`**.

Mape

- Dve praktične implementacije interfejsa **`Map<T,S>`** u Javi obuhvaćene su klasama **`TreeMap<T,S>`** i **`HashMap<T,S>`**
- Obično je bolje koristiti klasu **`HashMap`** ukoliko u programu nema potrebe za sortiranim redosledom ključeva koji obezbeđuje klasa **`TreeMap`**.

Telefonski imenik

```
import java.util.*;

public class TelImenik {

   private Map<String,String> imenik;

   // Konstruktor
   public TelImenik() {
      imenik = new HashMap<String,String>();
   }

   public String nađiBroj(String imeOsobe) {
      return imenik.get(imeOsobe);
   }

   public void dodajStavku(String imeOsobe, String brojOsobe) {
      imenik.put(imeOsobe,brojOsobe);
   }

   public void ukloniStavku(String imeOsobe) {
      imenik.remove(imeOsobe);
   }
}
```

Mape

```
System.out.println("Mapa sadrži sledeće parove ključ/vrednost:");

//Redom za svaki ključ u skupu ključeva mape prikazati (ključ,vrednost)
for (String ključ : m.keySet()) {
   Double vrednost = m.get(ključ);
   System.out.println("(" + ključ + "," + vrednost + ")");
}
```

Generičke metode

- Razmotrimo problem pretrage niza radi određivanja da li se data vrednost nalazi u datom nizu:

```
public static boolean nađi(T x, T[] a) {
    for (T elem : a)
        if (x.equals(elem))
            return true;
    return false;
}
```

Generičke metode

- Ovako napisan metod neće proći sintaksnu kontrolu, jer će Java prevodilac podrazumevati da je **T** neki konkretan tip čija definicija nije navedena. Zbog toga se mora na neki način ukazati da je **T** parametar tipa, a ne konkretan tip.

Generičke metode

- Kod generičkih metoda, oznaka **<T>** navodi se u zaglavlju metoda iza svih modifikatora i ispred tipa rezultata metoda:

```
public static <T> boolean nađi(T x, T[] a) {
   for (T elem : a)
      if (x.equals(elem))
         return true;
   return false;
}
```

Generičke metode

- Vrlo sličan metod može se napisati i za nalaženje objekta u bilo kojoj kolekciji:

```
public static <T> boolean nadi(T x, Collection<T> k) {
   for (T elem : k)
      if (x.equals(elem))
         return true;
   return false;
}
```

Generičke metode

- Opšte pravilo kod generičkih klasa je da ne postoji nikakva veza između **GenKlasa<A>** i **GenKlasa<B>** bez obzira na to u kakvoj su vezi konkretne klase **A** i **B**.
- Ovo pravilo u nekim slučajevima ima negativan efekat na prednosti generičkog programiranja.

Ograničenje tipova

- Rešenje su džoker tipovi:
  `public static void prikažiKolege (Par<? extends Službenik> p) . . .`
- Zapis **„`? extends Službenik`”** u tipu parametra ovogmetoda označava svaki tip koji je jednak klasi **`Službenik`** ili je podklasa od klase **`Službenik`**.

Džoker tipovi

- U zapisu oblika „`? Extends T`” umesto `T` može stajati interfejs, a ne isključivo klasa.
- Reč **`extends`**, a ne **`implements`**, koristi se čak i ako je `T` interfejs.