Objektno orijentisano programiranje
vežbe

Staša Vujičić Stanković

Kolekcija

- **<u>Kolekcija (lat. collectare)</u>** = zbirka, skupljanje (npr. starog novca, oružja, rukopisa, pisama, maraka itd.).

Generički klasni tipovi

- Pretpostavimo da radimo sa nekim skupom objekata koje želimo da nekako organizujemo.
  U tom slučaju koriste se klase koje se označavaju kao **kolekcije**.
  One definišu objekte koji se koriste za organizaciju drugih objekata odgovarajućeg tipa u kolekcije na odgovarajući način.
- Moguće je u kolekciju dodavati objekte različitog tipa, ali onda imamo problem utvrđivanja tipa objekta prilikom njegovog korišćenja.
  Ako pokušamo da neki objekat iz ovakve kolekcije upotrebimo kao objekat neke druge klase, nailazimo na problem.

Generički klasni tipovi

- Npr. neka smo omogućili da kolekciji mogu da se dodaju objekti klasa Tacka, Duz i String.
  Ako, primera radi, pokušamo da upotrebimo neki objekat klase Duz ili String kao objekat klase Tacka, program neće biti preveden.
- Da bi se izbegli ovakvi problemi,
  treba definisati kolekciju tako da se onemogući slučajno dodavanje objekta pogrešnog tipa.
- Jedan način za rešenje ovog problema je definisati zasebne kolekcije koje mogu da sadrže samo objekte jednog tipa.
  Ali, onda se javlja drugi problem –
  imaćemo za svaki tip objekta po jednu klasu koja definiše odgovarajuću kolekciju.

Generički klasni tipovi

- Drugi, efikasniji način, su upravo **generički tipovi.**
- Oni omogućavaju da se definiše samo jedna klasa kojom je kolekcija predstavljena.
- Svi objekti u kolekciji su istog tipa,
  ali sada postoji mogućnost da taj tip može biti proizvoljan.
- Na ovaj način, imamo jednu kolekciju koja može da se ponaša i kao kolekcija tačaka, i kao kolekcija duži i kao kolekcija stringova itd.

Generički klasni tipovi

- Generički tip je klasni ili interfejsni tip koji ima jedan ili više parametara.
- Definicija konkretne klase ili interfejsa iz generičkog tipa vrši se dodavanjem konkretnog argumenta za svaki od parametara generičkog tipa.

Definicija generičkog klasnog tipa

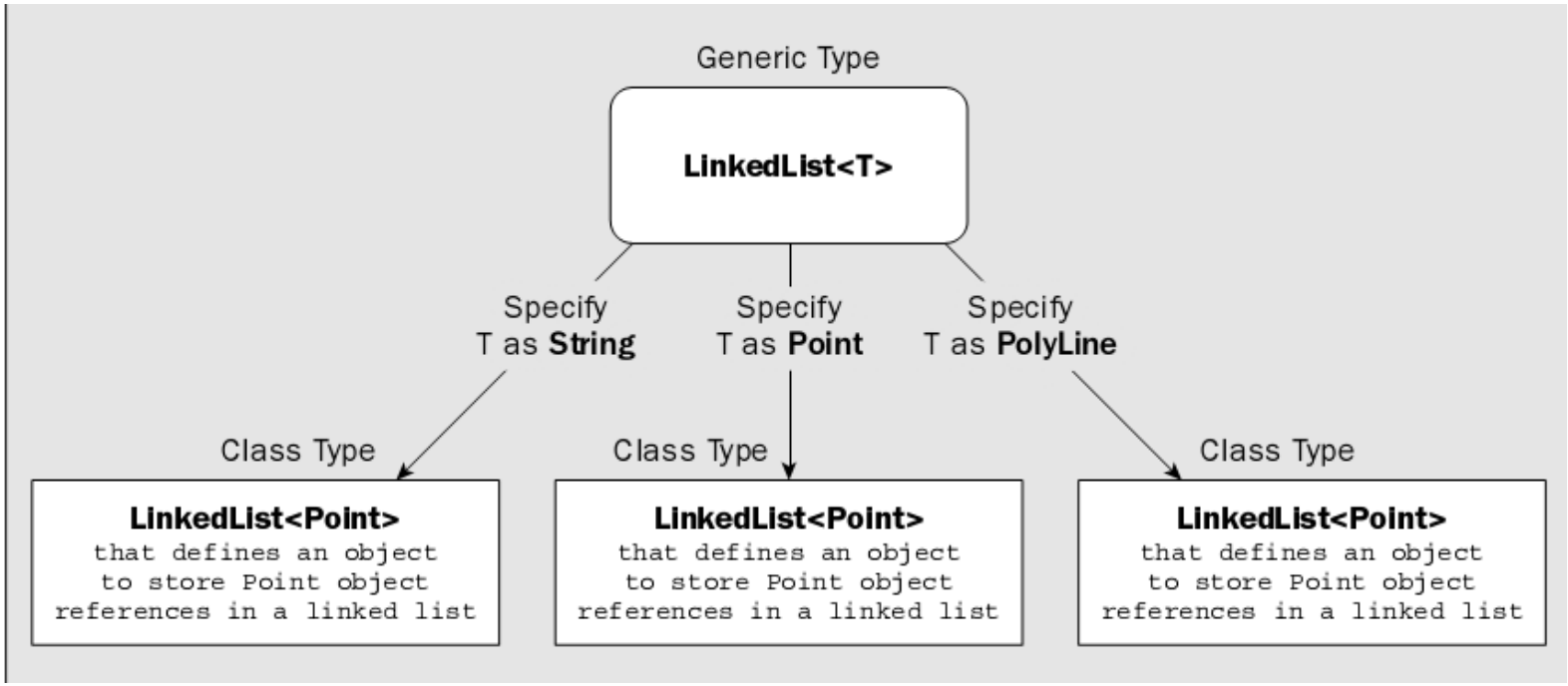
public class LinkedList<T> { ... }

- Parametar T se upotrebljava u definiciji metoda i polja gde postoji zavisnost od tipa ovog argumenta.
- Pojave ovog parametra u definiciji generičkog tipa se označavaju kao **promenljive tipa**, jer će biti zamenjene konkretnom vrednošću tog tipa, na isti način kao što parametri metoda bivaju zamenjeni argumentom koji se prosledi metodu.

Definicija generičkog klasnog tipa

- Kreiranje klase iz datog generičkog tipa vrši se navođenjem odgovarajućeg argumenta za parametar unutar < >.
  Sva pojavljivanja parametra T u definiciji generičkog tipa biće zamenjena datim argumentom.

Generički klasni tipovi



Generički tipovi

- Generički tip na taj način definiše skup tipova, dobijenih za različite vrednosti parametra tipa T.
- **Kao argument za parametar T može se proslediti samo klasni ili interfejsni tip!**
  Primitivni tipovi ne mogu da se koriste.
  Umesto njih koristimo odgovarajuće klase koje ih enkapsuliraju.

Generički klasni tipovi

- Generički tip LinkedList<T>
  koji čuva *double* vrednosti:

```
LinkedList<Double> temp =
        new LinkedList<Double>();
temp.addItem(10.8);
```

- Pošto je tip parametra Double,
  kompajler automatski umeće konverziju
  vrednosti 10.8 iz *double* u *Double*.

Generički tipovi

- Možemo da definišemo generički tip koji definiše skup klasa koje obuhvataju par objekata proizvoljnog tipa.
  U tom slučaju definišemo generički tip sa dva parametra:

```
public class Par<TipKljuca, TipVrednosti>
{
        public Par(TipKljuca kljuc, TipVrednosti vr)      {
                this.kljuc = kljuc;
                this.vr = vr;
        }
    …
        private TipKljuca kljuc;
        private TipVrednosti vr;
}
```

- Upotreba:

```
Par<String,String> unos =
        new Par<String,String>("Milan", "4445555");
```

Generički tipovi

- Oblast važenja parametarskog tipa je cela generička klasa, isključujući statičke članice klase.
- Statička polja ne mogu biti definisana promenljivom tipa, niti statički metodi mogu da imaju parametre ili povratne vrednosti koje su parametarskog tipa.

Generički tipovi

- Ukoliko generička klasa sadrži neko statičko polje, svaki tip proizveden iz datog generičkog tipa, imaće svoju kopiju statičkog polja.
- Neka generička klasa LinkedList<> sadrži statičko polje *count* za brojanje kreiranih objekata.
  Svaka klasa dobijena iz generičke za konkretan argument tipa imaće svoju kopiju ovog polja.
  Na taj način,
  - *count* u klasi LinkedList<String> broji kreirane objekte ove klase,
  - *count* u klasi LinkedList<Point> broji kreirane objekte ove klase itd.

Generički metod

- Možemo da definišemo metod sa svojim nezavisnim skupom od jednog ili više parametara.
  Takav metod naziva se
  parametarski metod ili **generički metod**.
- Generički metod može da postoji i unutar obične klase.

```
public static <T> void ispisiSve(LinkedList<T> lista)
{
        for(T objekat: lista)
                System.out.println(objekat);
}
```

- <T> ispred koga se navodi public ili static ključna reč predstavlja parametarski tip (ili listu tipova) za generički metod.
  Ovde imamo samo jedan parametarski tip za metod ispisiSve, ali ih može biti više.

Generički metod

- Lista parametarskih tipova se uvek pojavljuje između uglastih zagrada i navodi se iza kvalifikatora pristupa,
  a pre povratne vrednosti metoda.
- Argument tipa će biti određen na osnovu parametra tipa koji je prosleđen pri pozivu metoda.

Nizovi i parametrizovani tipovi

- Nizovi elemenata tipa dobijenog od generičkog tipa nisu dopušteni!

  Sledeća naredba će rezultovati porukom kompajlera o grešci:

```
PovezanaLista<String>[] liste =
            new PovezanaLista<String>[10];
```

Kolekcije

- Java **sistem kolekcija** je skup generičkih tipova koje koristimo za kreiranje kolekcijskih klasa.
- **Kolekcijska klasa** je klasa koja organizuje skup objekata datog tipa na određen način (npr. lančane liste, stek, ...).
- Najveći broj ovih klasa koje čine sistem kolekcija definisane su u paketu **java.util**.

Kolekcije

- Razmotrićemo sledeće generičke tipove:
  - **Iterator<T>** interfejsni tip – deklariše metode za iteriranje kroz jedan po jedan element kolekcije.
  - **Vector<T> tip** – ima strukturu dinamičkog niza, broj objekata koje možemo sačuvati se automatski povećava po potrebi.
  - **Stack<T> tip** – podržava smeštanje objekata proizvoljnog tipa u stek.
  - **LinkedList<T>** tip – dvostruko povezana lista, omogućeno je iteriranje u oba smera.

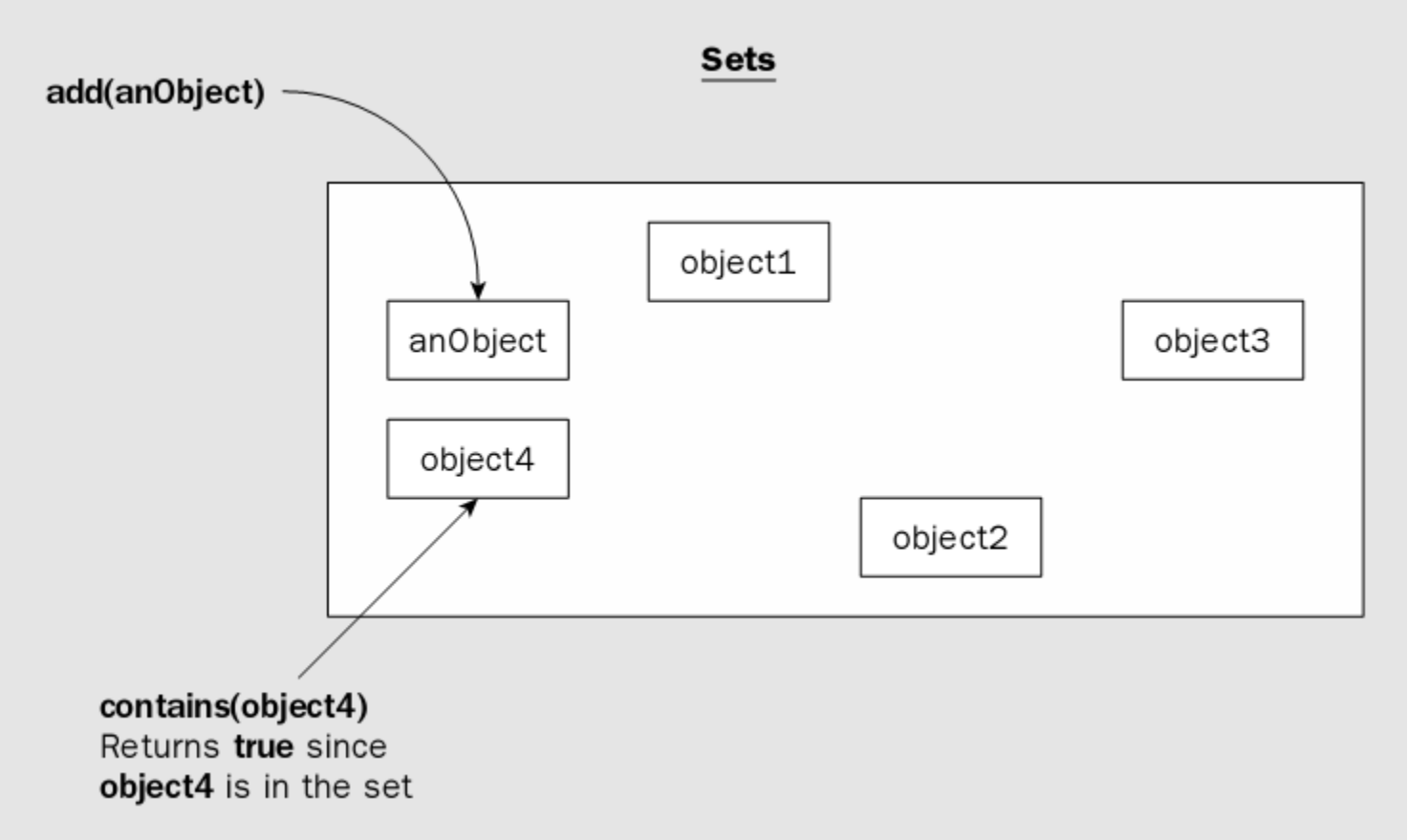
Kolekcije

- Klasu koja definiše kolekciju objekata često nazivamo kontejnerskom klasom.
- Postoje tri osnovna tipa kolekcija koji organizuju objekte na različite načine:
  - **skupovi**
  - **nizovi**
  - **katalozi**.

Kolekcije - skup

- **Skup(Set)** je najjednostavnija kolekcija u kojoj objekti nisu na neki specijalan način uređeni i objekti se jednostavno dodaju skupu, bez ikakve kontrole gde idu.
- Možemo iterirati kroz sve objekte skupa, ispitati da li je neki objekat član skupa. Skup ne može da sadrži duplikate.
- Objekat možemo i obrisati iz skupa, ali samo ako imamo referencu na njega u skupu.

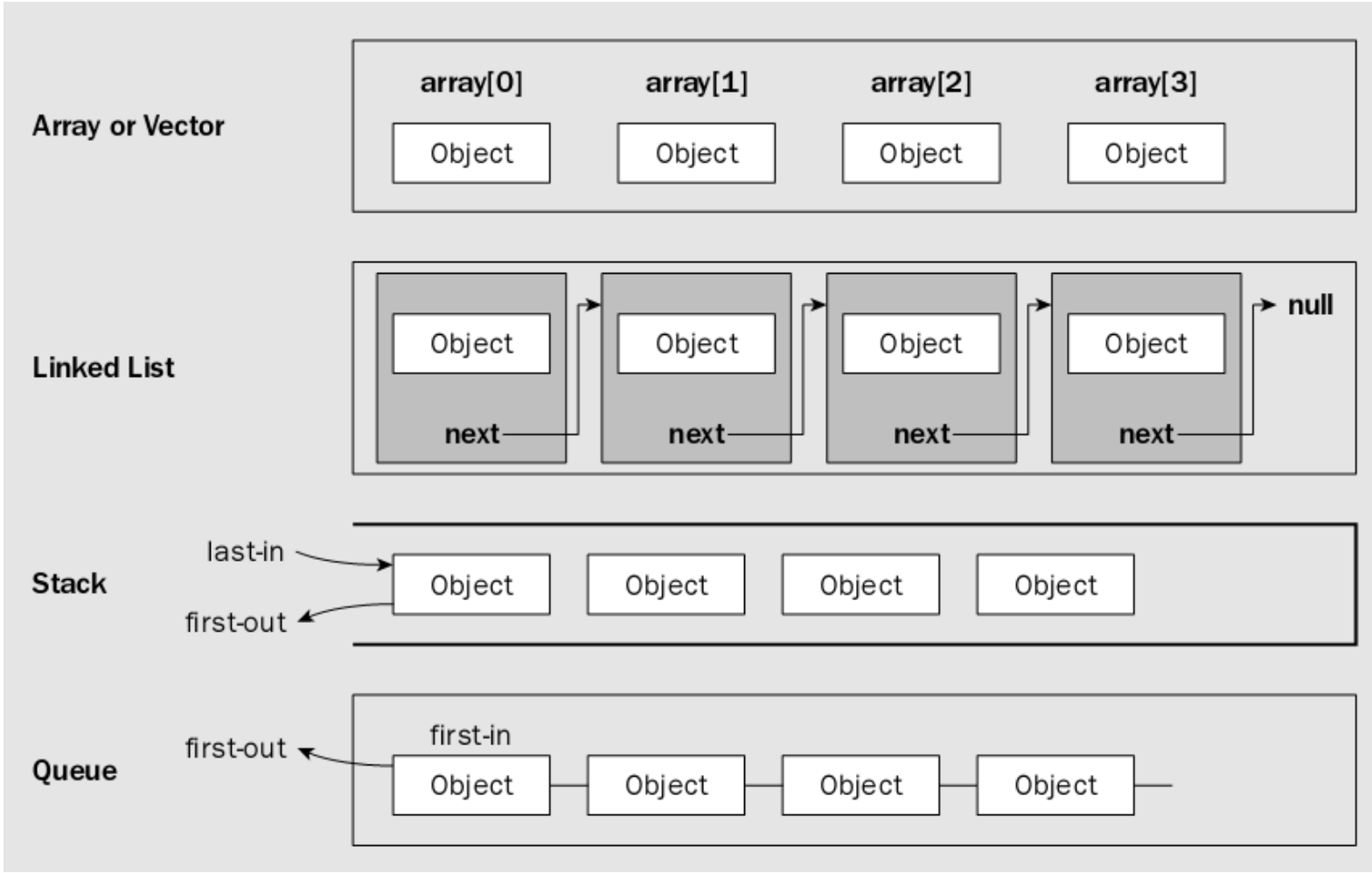
Kolekcije


Sets
add(anObject)
object1
anObject
object3
object4
object2
contains(object4)
Returns true since
object4 is in the set

Kolekcije - niz

- Glavna karakteristika **niza** je da su objekti smešteni na linearan način, u proizvoljnom, fiksnom redosledu gde imamo početak i kraj.
- Kolekcije u opštem slučaju imaju sposobnost da se prošire da bi se smestio potreban broj objekata.
- Primeri:
  - **Nizovi:**
    - Array, Vector
    - LinkedList
  - **Redovi:**
    - Stack(LIFO)
    - Queue(FIFO)

Kolekcije


Array or Vector
array[0]
array[1]
array[2]
array[3]
Object
Object
Object
Object
Linked List
Object
Object
Object
Object
next
next
next
next
null
Stack
last-in
first-out
Object
Object
Object
Object
Queue
first-out
first-in
Object
Object
Object
Object

Kolekcije

- Pošto je niz linearan, moguće je dodati novi objekat na početak ili kraj ili
umetnuti novi objekat iza date pozicije.
- Dobijanje objekta iz niza se može izvršiti na više načina:
  - Može se selektovati prvi ili poslednji
  - Može se dobiti objekat sa date pozicije
  - Može se pretraživati niz unapred ili unazad da bi se ispitalo da li se neki objekat nalazi u nizu i slično.

ITERATORI

- **Iterator** je objekat koji se koristi za iteriranje kroz kolekciju, tj. pristup jednom po jednom objektu kolekcije.
- Upotreba iteratora omogućuje upotrebu oblika for petlje karakterističnog za pretraživanje kolekcija.
- Bilo koji objekat koji predstavlja skup ili niz može da kreira objekat tipa Iterator<>.

ITERATORI

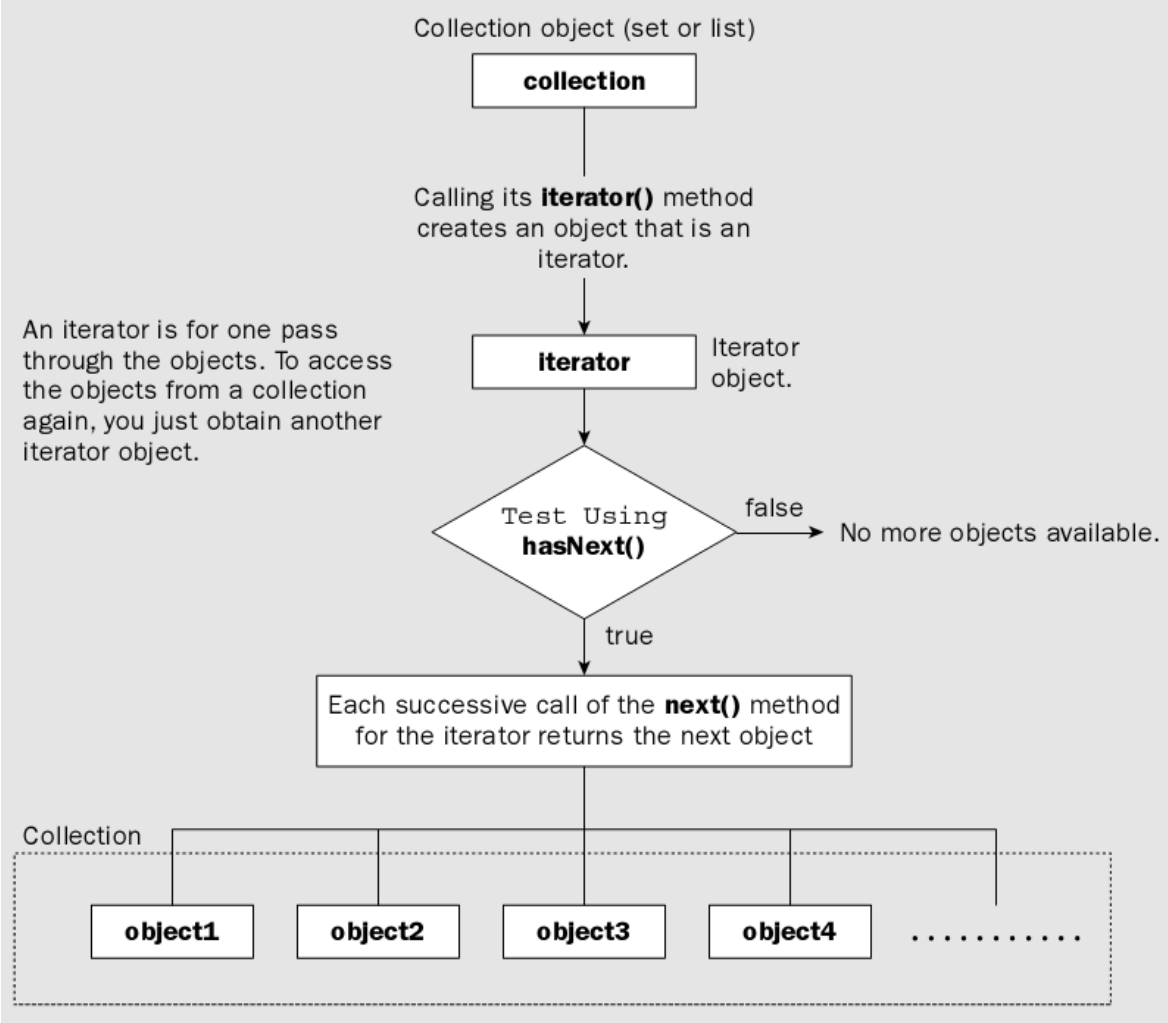
- Objekat tipa Iterator<> sadrži reference na sve objekte kolekcije u nekom redosledu i njima se može pristupiti pomoću metoda interfejsa Iterator<>:
  - **T next()** – vraća objekat tipa T i postavlja Iterator<T> objekat da pri sledećem pozivu ovog metoda referiše na sledeći objekat kolekcije. Ako nema objekta koji će biti vraćen, izbacuje se izuzetak tipa NoSuchElementException
  - **boolean hasNext()** – vraća *true* ako postoji sledeći objekat, inače *false*.
  - **void remove()** – briše poslednji objekat vraćen pozivom metoda next().
    Ako next() nije bio pozvan ili ako se pozove remove() dva puta nakon poziva next(), biće izbačen izuzetak tipa IllegalStateException.

- Primer:

```
Klasa primerak;
while(iterator.hasNext())
        primerak = iterator.next();
```

- Ovde se podrazumeva da je iterator tipa Iterator<Klasa> i da čuva referencu na objekat dobijen iz kolekcije.

ITERATORI


Collection object (set or list)
collection
Calling its iterator() method creates an object that is an iterator.
An iterator is for one pass through the objects. To access the objects from a collection again, you just obtain another iterator object.
iterator
Iterator object.
Test Using hasNext()
false
No more objects available.
true
Each successive call of the next() method for the iterator returns the next object
Collection
object1
object2
object3
object4
. . . . . .

Iteratori

- Jedan objekat koji implementira interfejs Iterator<> je za jednu iteraciju kroz kolekciju. Ukoliko je kasnije potrebno ponovo proći kroz sve elemente kolekcije, neophodno je kreirati novi objekat.

List iteratori

- U paketu java.util definisan je interfejs **ListIterator<>.**
- Deklariše metode koji se koriste za kretanje kroz kolekciju napred i nazad, tako da se do objekta može doći više nego jednom.
- ListIterator<> interfejs nasleđuje Iterator<> interfejs.

Spisak metoda

- **T next()** – kao kod Iterator<>
- **boolean hasNext()** – kao kod Iterator<>
- **int nextIndex()** – vraća indeks objekta koji će biti vraćen pozivom metoda next() kao tip int, ili vraća broj elemenata u listi ako je ListIterator<> objekat na kraju liste.
- **T previous()** – vraća prethodni objekat po redosledu u samoj listi. Koristimo ga za vraćanje kroz listu.

Spisak metoda

- **boolean hasPrevious()** – vraća true ako prethodni poziv previous() vraća objekat.
- **int previousIndex()** – vraća indeks objekta koji će biti vraćen sledećim pozivom metoda previous(), ili -1 ako je ListIterator<> objekat na početku liste.
- **void remove()** – briše poslednji objekat dobijen pozivom metoda *next()* ili *previous()*.
  Ako *next()* ili *previous()* nisu bili pozvani, izbacuje se izuzetak tipa *IllegalStateException*, a ako operacija brisanja nije podržana za datu kolekciju, izbacuje se izuzetak tipa *UnsupportedOperationException*.

Spisak metoda

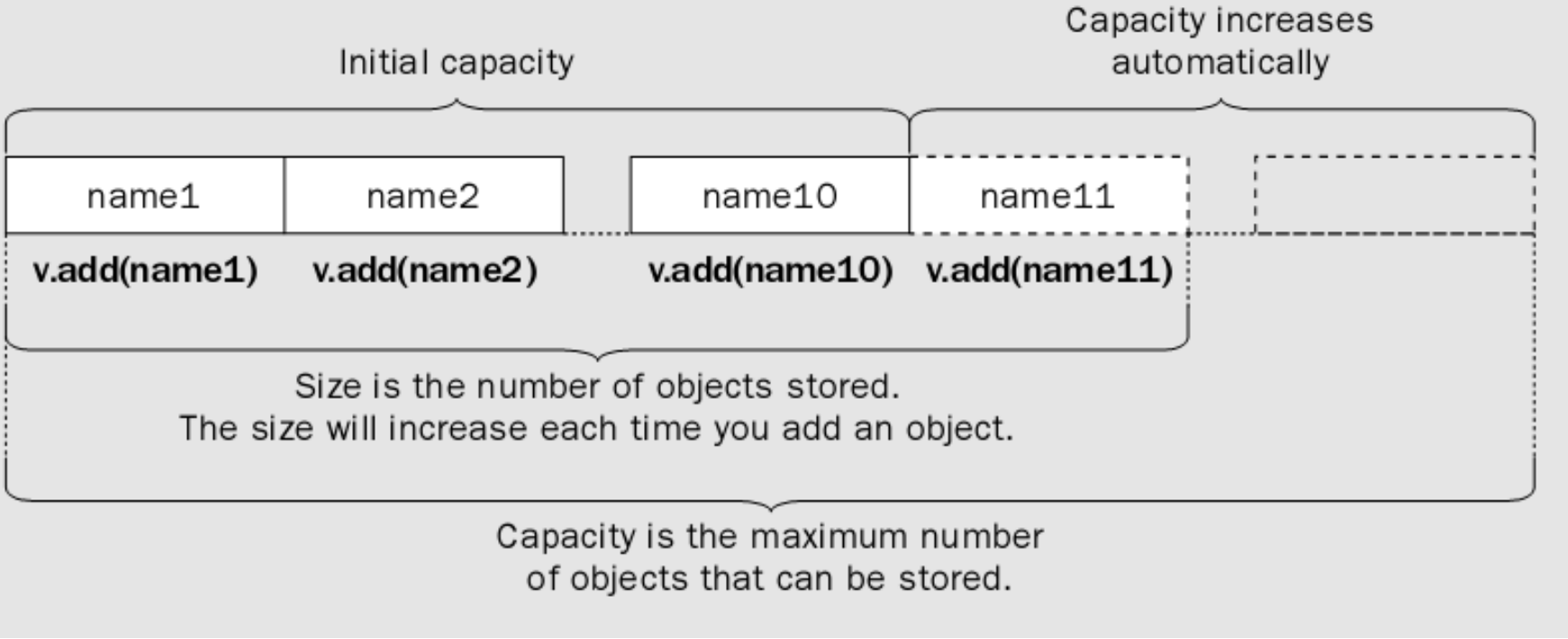
- **void add(T obj)** – dodaje argument neposredno pre objekta koji bi bio vraćen sledećim pozivom metoda *next()* ili iza objekta koji bi bio vraćen sledećim pozivom metoda *previous()*.
  Poziv *next()* nakon dodavanja vraća dodati element, a za *previous()* se ništa ne menja.
  Ako objekat ne može da se doda izbacuje se izuzetak tipa *UnsupportedOperationException*, *ClassCastException* ako klasa argumenta sprečava dodavanje i *IllegalArgumentException* ukoliko postoji neki drugi razlog zbog kojeg dodavanje ne može da se izvrši.
- **void set(T obj)** – menja poslednji objekat dobijen pozivom next() ili previous().
  I ovaj metod može da izbaci izuzetke tipa *UnsupportedOperationException*, *ClassCastException* i IllegalArgumentException.

Vektori

- **Vector<T>** definiše kolekciju sa elementima tipa T.
- Objekat Vector<T> radi kao i niz osim što dodatno raste automatski, kada je potreban veći kapacitet.
- Kao i nizovi, i vektori sadrže reference na objekte, ne same objekte.
  To je pravilo za sve kolekcije.

VEKTORI

- Vektor se, za razliku od niza, karakteriše **<u>veličinom (size)</u>** i **<u>kapacitetom (capacity)</u>**.



VEKTORI

- Kapacitet vektora je maksimalan broj objekata koje vektor može da sadrži.
  Kapacitet je promenljiv, jer se automatski povećava kada se doda novi objekat u već pun vektor.
- Metod **capacity()** vraća kapacitet vektora kao int.
- Metodom **ensureCapacity(int)** postavlja se minimalni kapacitet.
- Primer:

  v.ensureCapacity(150);
  /* ako je kapacitet vektora v manji od 150, kapacitet se uvećava na 150. Ako je 150 ili više, neće biti promenjen ovom naredbom. */

KONSTRUKTORI

- **Vector<>()** – podrazumevani konstruktor kreira prazan Vector<> podrazumevanog kapaciteta 10. Kapacitet se duplira ukoliko se doda element u već pun vektor.

  Vector<String> a = new Vector<String>();

KONSTRUKTORI

- **Vector<String> a =**
  new Vector<String>(100); -

  eksplicitno se zadaje kapacitet.
  Kapacitet se duplira ako se dodaje element u pun vektor. Može da bude neefikasno.

- **Vector<String> a =**
  new Vector<String>(100, 10); -

  zadaje se inicijalni kapacitet i zadaje se za koliko se povećava kapacitet kada je to potrebno.

Metode

- **size()** – vraća broj elemenata koji su smešteni u vektor.
- **setSize(int)** – postavlja se veličina, ostatak do maksimalnog kapaciteta se puni null elementima.
- **add(T)** – dodaje se objekat iza tekućeg poslednjeg objekta u vektoru.
- **add(int, T)** – smešta se objekat na poziciju zadatu prvim argumentom, ostali se pomeraju za jedno mesto u desno.

Metode

- **set(int, T)** – postavljamo objekat u vektor na poziciju zadatu prvim argumentom, briše se objekat koji je prethodno bio na toj poziciji.
- **addAll(lista)** – dodajemo sve objekte druge kolekcije u vektor.
- **addAll(int, lista)**
- **get(int)** – vraća element na zadatoj poziciji
- **firstElement()** – vraća prvi element
- **lastElement()** – vraća poslednji element

Veličina vektora

- Kada se kreira Vector<> date veličine,
  još uvek nijedna referenca ni na jedan objekat nije smeštena u vektor. Dakle, vektor je inicijalno prazan.

- Broj slobodnih elemenata vektora se može dobiti kao:

```
int freeCount = names.capacity() – names.size();
```

Veličina vektora

- Promena veličine vektora:

  names.setSize(50);

  /* veličina vektora *names* promenjena je na 50.
  - Ukoliko vektor ima manje od 50 zauzetih elemenata, ostali elementi do 50 biće postavljeni na *null*.
  - Ako vektor već sadrži više od 50 objekata, sve reference na objekte preko 50 će biti odbačene. Sami objekti, naravno, mogu biti dostupni i dalje, ukoliko postoji neka referenca koja na njih ukazuje. */

Promena veličine vektora

- Činjenica da se kapacitet vektora duplira svaki put kada se vektor popuni utiče na to da se memorija bespotrebno zauzima.
- Međutim, ovo može da se prevaziđe upotrebom metoda **trimToSize().**
- Ovaj metod menja kapacitet tako da odgovara trenutnoj veličini.

```
names.trimToSize();
    /* postavlja kapacitet na trenutnu veličinu. Ako je
    veličina u trenutku izvršavanja metoda 30, i
    kapacitet se postavlja na 30. Naravno, i dalje se
    mogu dodavati novi objekti u vektor. */
```

List iterator nad vektorom

- Možemo dobiti i ListIterator referencu iz vektora pozivanjem metoda listIterator().

```
ListIterator<String> it =
        names.listIterator();
/* sada je moguće kretati se u oba smera kroz
vektor upotrebom metoda iz klase ListIterator. */
```

List iterator nad vektorom

- **ListIterator<String> it =**
  names.listIterator(2);

  /* ovim se dobija ListIterator objekat koji se odnosi samo na deo vektora, pri čemu je argument metoda listIterator() upravo prvi element date sekvence vektora. */

- **List<String> list =**
  names.subList(2,5);

  /* dobija se podskup objekata iz vektora kao kolekcija tipa List<>. Argumetni metoda subList() su početak i kraj sekvence objekata koja čini podskup, pri čemu poslednji element nije uključen. */

List iterator nad vektorom

- Moguće su i kombinacije:

```
ListIterator<String> listIter =
    names.subList(5,15).listIterator(2);
```

/* *subList()* vraća podlistu objekata vektora od pozicije 5 do pozicije 14 zaključno. Nakon toga se za datu listu poziva metod *listIterator()* koji vraća list iterator nad elementima liste počevši od elementa sa indeksom 2, pa do kraja.
To odgovara elementima polaznog vektora sa indeksima od 7 do 14. Ovim iteratorom se kroz datu sekvencu objekata polaznog vektora može kretati u oba smera. */

Konverzija vektora u niz

- Metod **toArray()** omogućava da se elementi vektora dobiju u obliku običnog niza.
- **String[] data =**
  names.toArray(new String[names.size()]);

  /* argument metoda toArray() mora biti niz elemenata istog tipa kao i elementi vektora ili nadtipa. Ako niz nije dovoljne veličine da primi sve elemente vektora, kreiraće se novi niz i referenca na taj niz će biti vraćena. Ovde metod toArray() vraća niz String[] koji sadrži sve elemente vektora names u odgovarajućem redosledu.*/

Konverzija niza u vektor

- **String[] people = {“Bill”, “Barbara”, “Brian”};**
  List<String> nameList =
  java.util.Arrays.asList(people);

  statički parametarski metod asList() definisan u klasi java.util.Arrays konvertuje niz elemenata datog tipa T u List<T> kolekciju. Argument metoda je niz koji se konvertuje i vraća se referenca tipa List<T>. Ova referenca se može proslediti kao argument konstruktoru klase Vector:

- **Vector<String> names =**
  new Vector<String>(nameList);

  /* Time se dobija vektor koji sadrži elemente datog niza. */

Brisanje elemenata

- **remove(int)** – brisanje reference na objekat na zadatoj poziciji. Vraća se referenca na uklonjeni objekat, tako da ostaje mogućnost da se sačuva referenca nakon što se ukloni iz vektora.

  String name = names.remove(3);
- **remove(T)** – brisanje prvog objekta sa zadatom referencom. Ukoliko je pronađen i uklonjen objekat, vraća se true, inače false.
- **removeAll(names.subList(2,5));** - brišu se elementi sa indeksima od 2 do 4.
- **retailAll(names.subList(2,5));** - ostaju samo elementi sa indeksima od 2 do 4.
- **removeAllElements()** – uklanja sve elemente iz vektora.

Brisanje elemenata

- Da li je vektor prazan može se utvrditi metodom **isEmpty().** Ukoliko je veličina vektora 0, metod vraća true, inače false.
- **Napomena:**

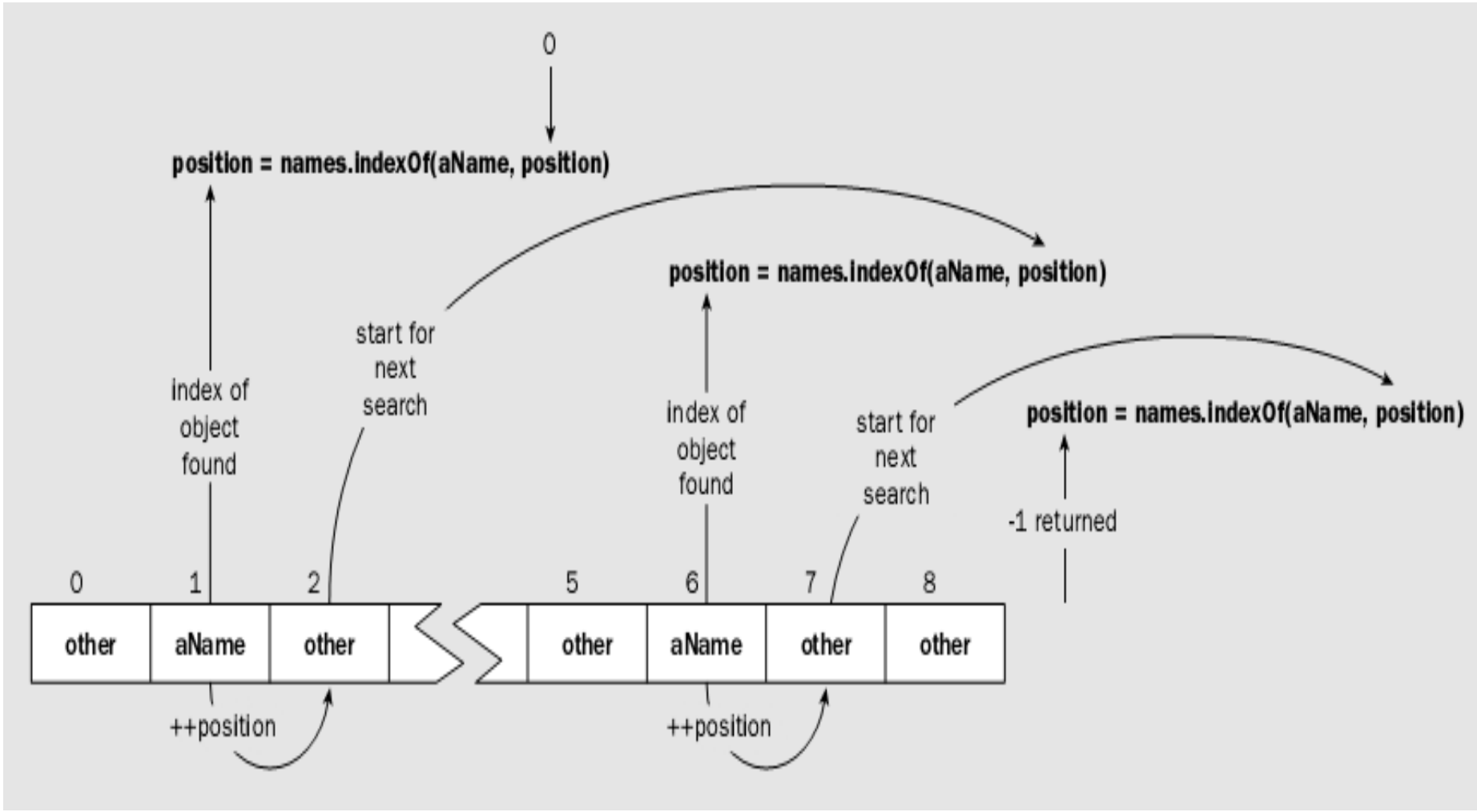
  Ukoliko vektor sadrži samo null elemente, to ne znači da je vektor prazan, tj. da je veličina 0 ili da će metod isEmpty() vratiti true. Da bi se vektor ispraznio elementi, tj. reference na objekte moraju biti uklonjene, a ne samo postavljene na null.

Pretraživanje vektora

- **int indexOf(T)** –
  vraća poziciju datog objekta u vektoru.
- **int indexOf(T, int)** –
  vraća prvu poziciju datog objekta u vektoru, počevši od pozicije zadate kao drugi argument

```
String aName = "Fred";                                    // Name to be found
int count = 0;                                            // Number of occurrences
int position = -1;                                        // Search starting index
while(++position<names.size()) {                          // Search with a valid index
  if(position = names.indexOf(aName, position))<0) {      // Find next
    break;
  }
  ++count;
}
```

Pretraživanje vektora



Primer – Guzva

- Osoba.java
- NekorektanUnosException.java
- Guzva.java
- TestGuzve.java

Sortiranje elemenata

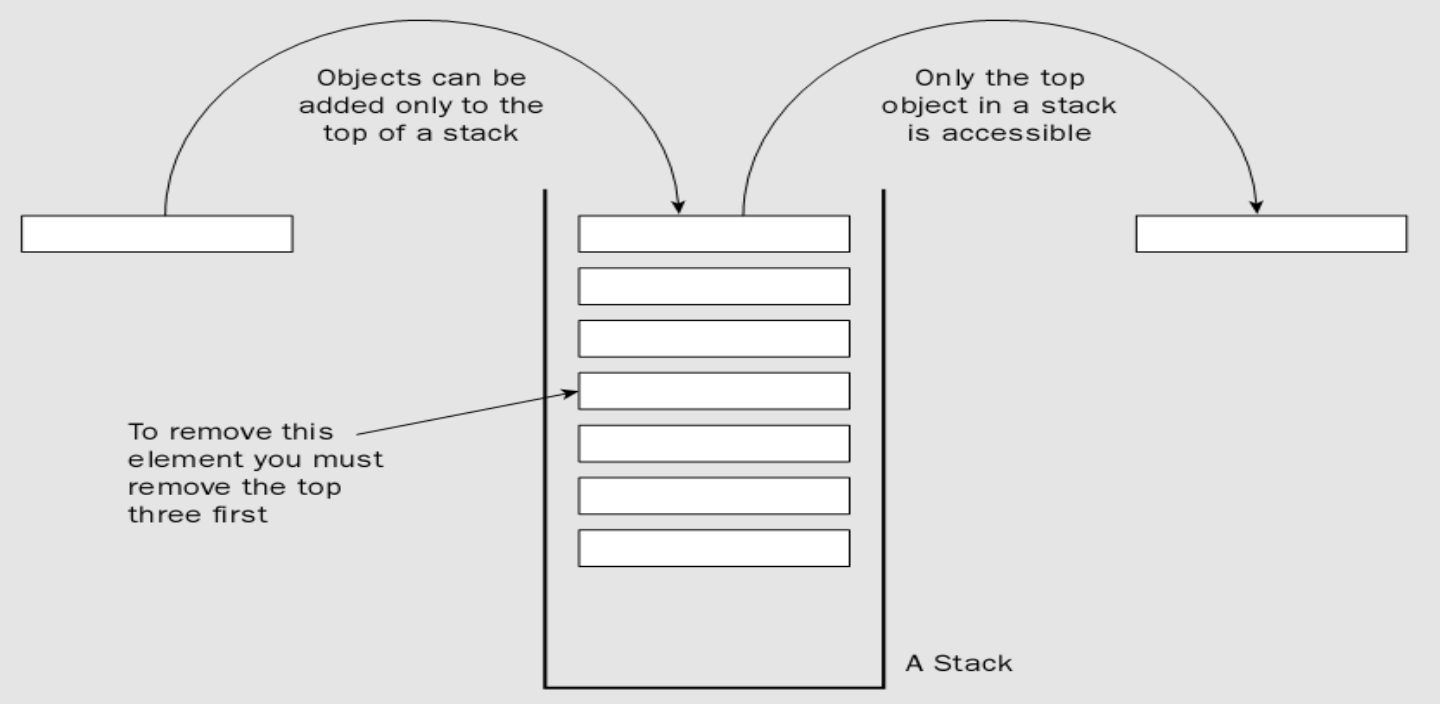
- Za sortiranje elemenata kolekcije najbolje je implementirati interfejs **Comparable<>.**
  On deklariše samo jedan metod
  compareTo() –
  vraća -1, 0 ili 1 ako je objekat
  manji, jednak ili veći od argumenta.
- Ako je Comparable<> interfejs implementiran za tip objekta smešten u kolekciji, možemo samo objekat kolekcije predati kao argument metodu **Collections.sort()** interfejsa Collections<>.

Zadatak – Primer sortiranjeZvezda

- Osoba.java
- NekorektanUnosException.java
- Guzva.java
- TestSortiranjeZvezda.java

Stack

- Generički tip Stack<> izveden je iz generičkog klasnog tipa Vector<>.



Stack

- Svi metodi koji važe za Vector<> važe i za Stack<>. Pored njih, metodi karakteristični samo za stek su:
- **T push(T obj)** – stavlja referencu na objekat obj na vrh steka i vraća je kao povratnu vrednost.
- **T pop()** – skida referencu sa vrha steka i vraća je kao povratnu vrednost.
- **T peek()** – daje referencu na vrhu, ali je ne skida sa steka i vraća je kao povratnu vrednost.
- **int search(Object obj)** – vraća poziciju referisanog objekta obj u steku. Referenca na vrhu steka je na poziciji 1, sledeća je na poziciji 2 itd. Ako objekat nije pronađen u steku, vraća se -1.
- **boolean empty()** – vraća true ako je stek prazan, inače false.
- Jedini konstruktor je konstruktor bez argumenata.

Zadatak – Primer Karte

- Karta.java
- Spil.java
- Ruka.java
- TestKarte.java