Podatkovne strukture

Kdaj uporabiti katero podatkovno strukturo?

Podatkovne strukture

1. Podatkovna struktura je organizacija podatkov v pomnilniku računalnika.

2. Pod tem razumemo sezname, sklad, binarna drevesa, “hash” tabele itd.

3. Algoritmi obdelujejo podatke v teh strukturah na različne načine, na primer z iskanjem in sortiranjem.

Prednosti in slabosti podatkovnih struktur

Tip podatkovne strukture	Prednosti	Slabosti
Array	Quick insertion, very fast access if index known.	Slow search, slow deletion, and fixed size.
Ordered array	Quicker search than unsorted array	Slow insertion and deletion, fixed size
Stack	Last in first out	Slow access to other items
Queue	First in first out access.	Slow access to other items
Linked list	Quick insertion, quick deletion	Slow search
Array List	Random Access	Slow insertion, deletion

Kolekcije

Kolekcija (collection , pri C++ pa container) je objekt, , ki združuje več elementov v eno enoto.

• Kolekcije potrebujemo za pomnenje, pridobivanje in rokovanje s podatki in za prenos podatkov od ene metode k drugi.
• Kolekcije pomnijo:
– Specifični podatkovni tip
– generični podatkovni tip

Java Collections Framework

Algoritmi:
Metode, ki na objektih, ki implementirajo kolekcijske vmesnike, izvajajo uporabna rokovanja, kot na primer iskanje in sortiranje.

Algoritmi so polimorfni, ker lahko enako metodo uporabljamo na različnih implementacijah ustreznega kolekcijskega vmesnika.

Imamo ponovno uporabljivo funkcionalnost.

Kaj je “Collections framework”?

- Ogrodje (framework) je prostrana množica vmesnikov, abstraktnih razredov in konkretnih razredov skupaj s podpornimi orodji.
- Ogrodje je podano v paketu java.util in ga sestavljajo trije deli:
    1. Osnovni vmesniki
    2. Množica implementacij.
    3. Uporabne metode

“**Java Collections Framework “**

•Ponuja vmesnike: abstraktne podatkovne tipe, ki predstavljajo kolekcije.

•Omogoča rokovanje s kolekcijami neodvisno od podrobnosti njihovih predstavitev.

•Ponuja implementacije: konkretne implementacije kolekcijskih vmesnikov.

•Ponuja ponovno uporabljive podatkovne strukture

Le zakaj bi rabili tako ogrodje?

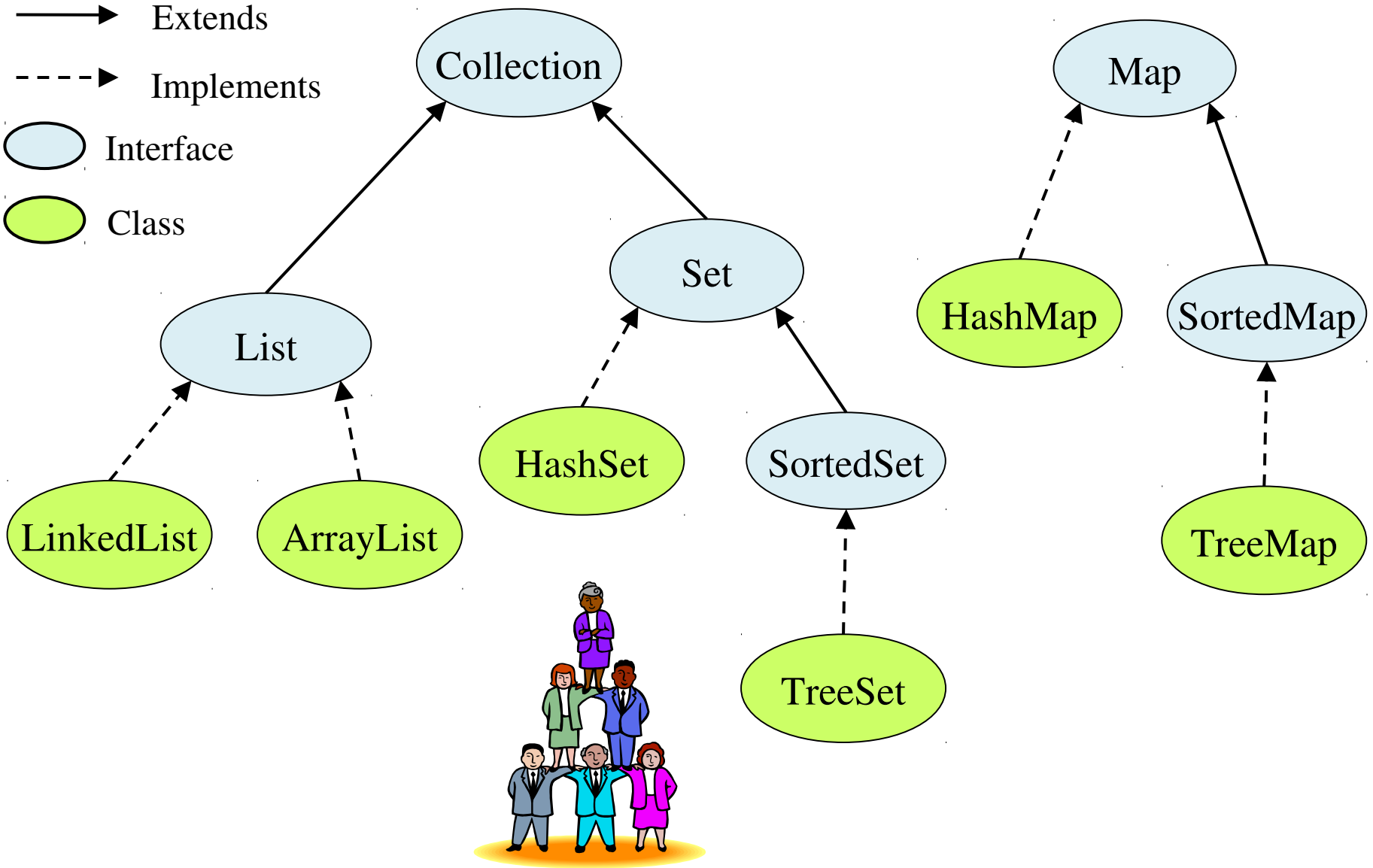
- Pred Javo SDK1.2, smo imeli povsem uporabne podatkovne strukture:
  - Hash Table
  - Vector
  - Stack

- Bile so dobre, njihova uporaba je bila enostavna. Vendar niso bile organizirane v bolj splošno ogrodje.
- Pomanjkljiva je bila tudi interoperabilnost.

Značilnosti kolekcijskega ogrodja

- Zmanjša napor pri programiranju.
- Poveča učinkovitost.
- Zagotavlja interoperabilnost med nevezanimi API-ji (Application Programming Interfaces).
- Hitrejša ponovna uporaba progra

Vmesniki in implementacija



Vmesnik Collection

`add(o)`	Add a new element
`addAll(c)`	Add a collection
`clear()`	Remove all elements
`contains(o)`	Membership checking.
`containsAll(c)`	Inclusion checking
`isEmpty()`	Whether it is empty
`iterator()`	Return an iterator
`remove(o)`	Remove an element
`removeAll(c)`	Remove a collection
`retainAll(c)`	Keep the elements
`size()`	The number of elements

Množice (Sets)

Skupina unikatov, ki nima duplikatov

Nekaj primerov

- Množica velikih črk ‘A’ do ‘Z’
- Množica nenegativnih celih števil { 0, 1, 2, … }
- Prazna množica{ }

Osnovne lastnosti množic

- Ima le po en primerek vsake postavke
- Lahko je končna ali neskončna
- Lahko definira abstraktne pojme
- Lahko je urejena ali neurejena

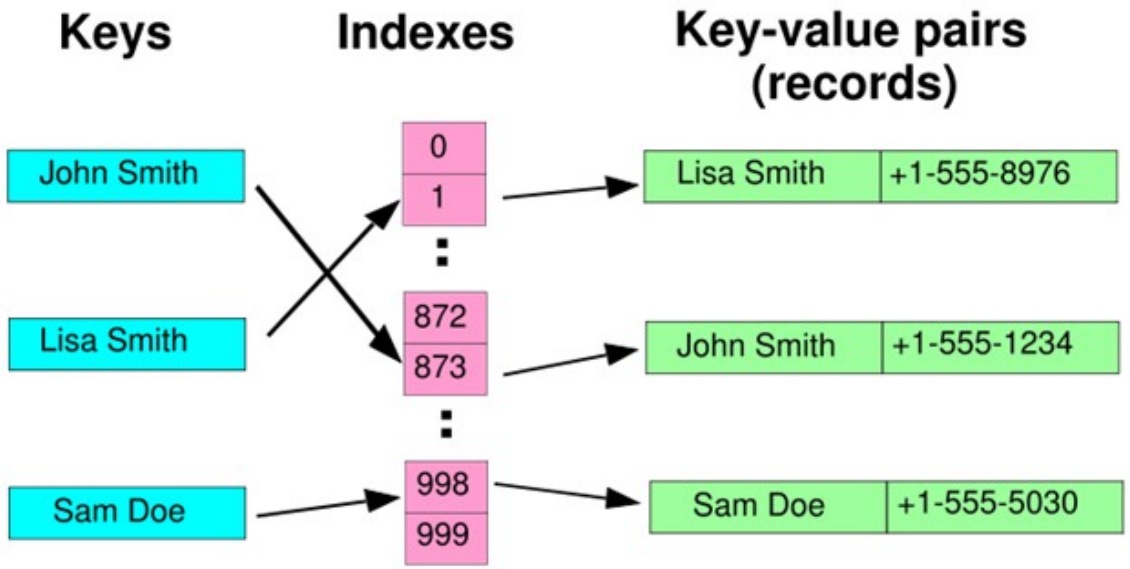
Kdaj sta dve množici enaki?

{1, 2, 3} == {1, 3, 4} ?
{A, a, c} == {A, a, c} ?

HashSet

HashSet: implementira vmesnik Set, naslonjen na zgoščeno tabelo (hash table).

- *Ne zagotavlja, da bo vrstni red ves čas enak.*
- *Skoraj konstantna časovna performansa*
- *Kapaciteta – število zajetij v zgoščeni tabeli*
- *Faktor obremenitve– Razmerje števila elementov, pomnjenih v tabeli, v primerjavi z njeno tekočo kapaciteto.*



Majhen telefonski imenik kot zgoščena tabela

Uporaba HashSet

```
Set set = new HashSet(); // instantiate a concrete set
// ...
set.add(obj); // insert an elements
// ...
int n = set.size(); // get size
// ...
if (set.contains(obj)) {...} // check membership

// iterate through the set
Iterator iter = set.iterator();
while (iter.hasNext()) {
  Object e = iter.next();
  // downcast e
  // ...
}
```

Primer: štetje različnih besed (1)

```
public class CountWords {
  static public void main(String[] args) {
    Set words = new HashSet();
    BufferedReader in =
          new BufferedReader(
              new InputStreamReader(System.in));
    String delim = " \t\n.,:;?!-/()[]\"\'";
    String line;
    int count = 0;
```

Primer: štetje različnih besed (2)

```
    try {
      while ((line = in.readLine()) != null) {
        StringTokenizer st =
           new StringTokenizer(line, delim);
        while (st.hasMoreTokens()) {
          count++;
          words.add(
            st.nextToken().toLowerCase());
        }
      }
    } catch (IOException e) {}
    System.out.println("Total number of words: "
                       + count);
    System.out.println("Number of different words: "
                       + words.size());
  }
}
```

List (seznam)

Urejena kolekcija elementov (pravimo ji tudi zaporedje):

- Lahko vsebuje duplikate.
- Do elementov dostopamo glede na njihov položaj v seznamu.
- Prvi element ima indeks nič.

Vmesnik List

- `add(i,o)` Insert `o` at position `i`
- *`add(o)`* Append `o` to the end
- `get(i)` Return the `i`-th element
- `remove(i)` Remove the `i`-th element
- `set(i,o)` Replace the `i`-th element with `o`
- `indexOf(o)`
- `lastIndexOf(o)`
- `listIterator()`
- `sublist(i,j)`

TreeSet

TreeSet implementira vmesnik Set in je podprt z instanco TreeMap:

- Zagotavlja, da bo urejena množica z elementi v naraščajočem vrstnem redu.
- TreeMap je rdeče-črno drevo, ki implementira vmesnik SortedMap.

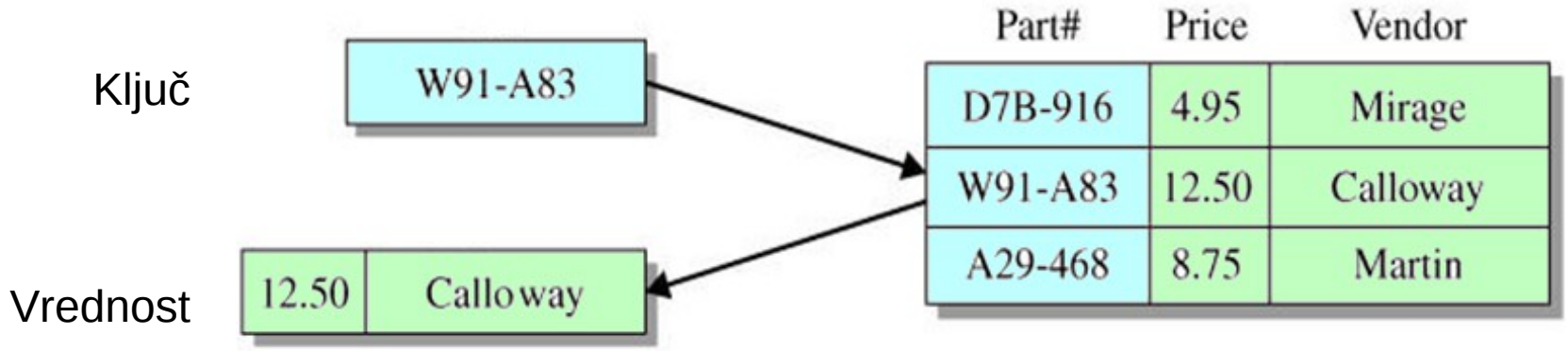
Map (karta, slovar)

Karta je posebna oblika množice.

- Množica parov ključ- vrednost.
- Vsak ključ preslika v največ eno vrednost.
- Le enkratni ključi vendar večkratne vrednosti.

Nekaj primerov:

- Preslikava ključev v zapise podatkovne baze
- Slovar (besede preslikane v pomen)



Vmesnik Map

`clear()`	Remove all mappings
`containsKey(k)`	Whether contains a mapping for `k`
`containsValue(v)`	Whether contains a mapping to `v`
`entrySet()`	Set of key-value pairs
`get(k)`	The value associated with `k`
`isEmpty()`	Whether it is empty
`keySet()`	Set of keys
`put(k,v)`	Associate `v` with `k`
`remove(k)`	Remove the mapping for `k`
`size()`	The number of pairs
`values()`	The collection of values

Uporaba Map

```
Map map = new HashMap(); // instantiate a concrete map
// ...
map.put(key, val); // insert a key-value pair
// ...
// get the value associated with key
Object val = map.get(key);
map.remove(key); // remove a key-value pair
// ...
if (map.containsValue(val)) { ... }
if (map.containsKey(key)) { ... }
Set keys = map.keySet(); // get the set of keys
// iterate through the set of keys
Iterator iter = keys.iterator();
while (iter.hasNext()) {
  Key key = (Key) iter.next();
  // ...
}
```

Primer: frekvenca besed (1)

```
public class Count {
  public Count(String word, int i) {
    this.word = word;
    this.i = i;
  }

  public String word;
  public int i;
}
```

Primer: frekvenca besed (2)

```
public class WordFrequency {

  static public void main(String[] args) {
    Map words = new HashMap();
    String delim = " \t\n.,:;?!-/()[]\"\'";
    BufferedReader in =
      new BufferedReader(
        new InputStreamReader(System.in));
    String line, word;
    Count count;
```

Primer: frekvenca besed (3)

(class `WordFrequency` continued.)

```
    try {
      while ((line = in.readLine()) != null) {
        StringTokenizer st =
          new StringTokenizer(line, delim);
        while (st.hasMoreTokens()) {
          word = st.nextToken().toLowerCase();
          count = (Count) words.get(word);
          if (count == null) {
            words.put(word,
                      new Count(word, 1));
          } else {
            count.i++;
          }
        }
      }
    } catch (IOException e) {}
```

Primer: frekvenca besed (4)

(class `WordFrequency` continued.)

```
    Set set = words.entrySet();
    Iterator iter = set.iterator();
    while (iter.hasNext()) {
      Map.Entry entry =
                  (Map.Entry) iter.next();
      word = (String) entry.getKey();
      count = (Count) entry.getValue();
      System.out.println(word +
          (word.length() < 8 ? "\t\t" : "\t") +
          count.i);
    }
  }
}
```

Primer: frekvenca besed: izpis

Using President Lincoln's *The Gettysburg Address* as the input, the output is:

```
devotion        2
years           1
civil           1
place           1
gave            2
they            3
struggled       1
......
men             2
remember        1
who             3
did             1
work            1
rather          2
fathers         1
```

The Gettysburg Address

Four score and seven years ago, our fathers brought forth upon this continent a new nation: conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war. . .testing whether that nation, or any nation so conceived and so dedicated. . . can long endure. We are met on a great battlefield of that war.

We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we cannot dedicate. . .we cannot consecrate. . . we cannot hallow this ground. The brave men, living and dead, who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here.

It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us. . .that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion. . . that we here highly resolve that these dead shall not have died in vain. . . that this nation, under God, shall have a new birth of freedom. . . and that government of the people. . .by the people. . .for the people. . . shall not perish from the earth.

Razlika med kolekcijami in kartami

Kolekcije (collections)

Lahko dodajamo, brišemo, gledamo izolirane postavke v kolekciji.

Karte (Maps)

- Na voljo imamo operacije na kolekcijah, ki pa delujejo na parih ključ-vrednost namesto na izoliranih elementih.
- Tipična uporaba kart je dostop do vrednosti, shranjenih s ključem.

Kolekcije so vmesniki

- Kolekcija je pravzaprav vmesnik
- Vsaka vrsta kolekcije ima eno ali več implementacij
- Tvorimo lahko nove vrste kolekcij
- Ko implementiramo vmesnik, obljubimo, da bomo zagotovili zahtevane metode
- Nekatere metode kolekcije so opcijske (neobvezne)
  - Kako lahko vmesnik deklarira neobvezno metodo?

Konkretne kolekcije

concrete collection	implements	description
`HashSet`	`Set`	hash table
`TreeSet`	`SortedSet`	balanced binary tree
`ArrayList`	`List`	resizable-array
`LinkedList`	`List`	linked list
`Vector`	`List`	resizable-array
`HashMap`	`Map`	hash table
`TreeMap`	`SortedMap`	balanced binary tree
`Hashtable`	`Map`	hash table

Metode kolekcij

Collection
+add(element : Object) : boolean +addAll(collection : Collection) : boolean +clear() : void +contains(element : Object) : boolean +containsAll(collection : Collection) : boolean +equals(object : Object) : boolean +hashCode() : int +iterator() : Iterator +remove(element : Object) : boolean +removeAll(collection : Collection) : boolean +retainAll(collection : Collection) : boolean +size() : int +toArray() : Object[ ] +toArray(array : Object[ ]) : Object[ ]

Iteratorji

Kolekcija nudi iterator, ki omogoča zaporedni dostop do elementov v kolekciji.

Metode:

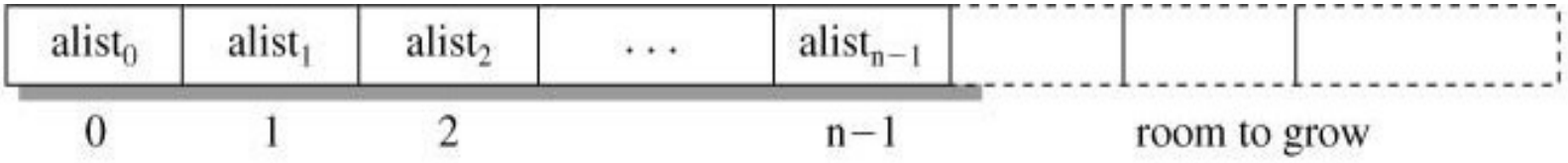
- has Next() – test, če imamo še kaj elementov
- next() – vrne naslednji objekt in napreduje
- remove() – odstrani trenutno kazani objekt

Konkretne implementacije list

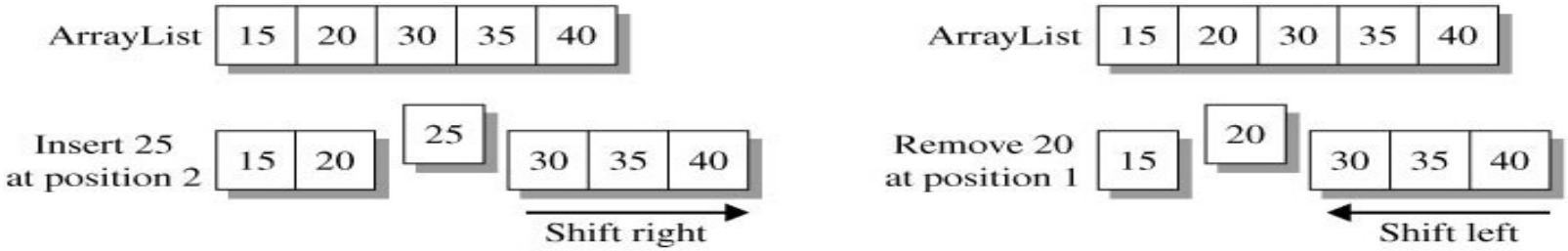
- Imamo dve konkretni implementaciji vmesnika List
  - LinkedList
  - ArrayList
- Katero naj bi uporabili, je odvisno od naših potreb.

Array List

- Shranjuje elemente v celovit kos pomnilnika, ki ga lahko avtomatično podaljšujemo.



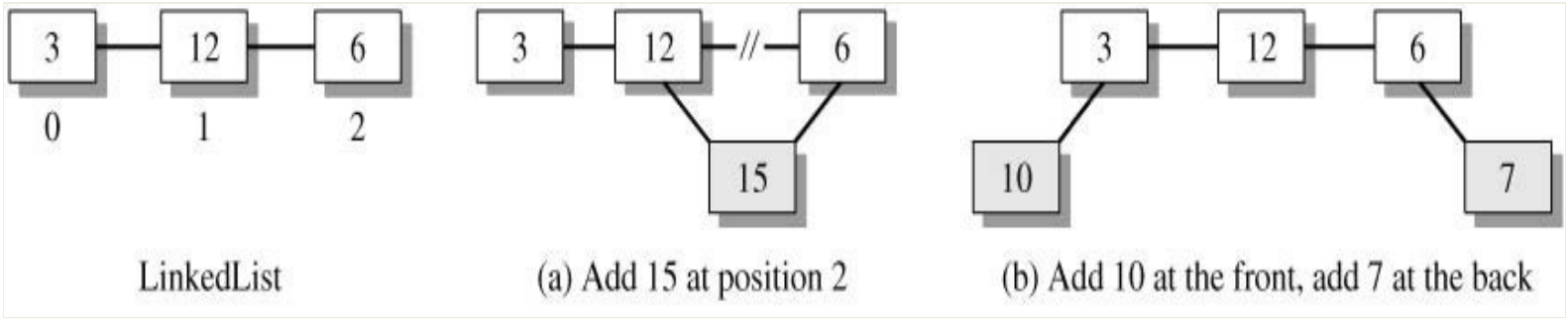
- Kolekcija učinkovito dodaja in briše elemente na koncu seznama.
- Operacije na vmesnih pozicijah so manj hitre.

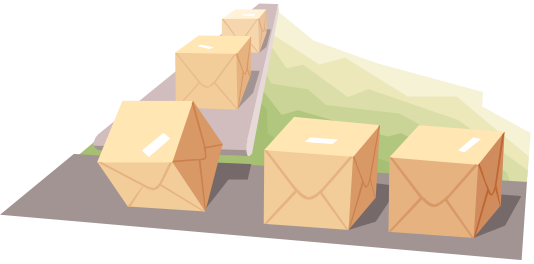


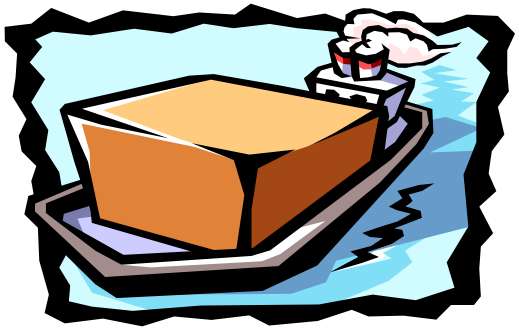
Shifting blocks of elements to insert or remove an ArrayList item.

Link List

- Elementi imajo vrednost in povezave, ki identificirajo sosednje elemente v zaporedju.
- Vrivanje in brisanje elementov ni zelo hitro.



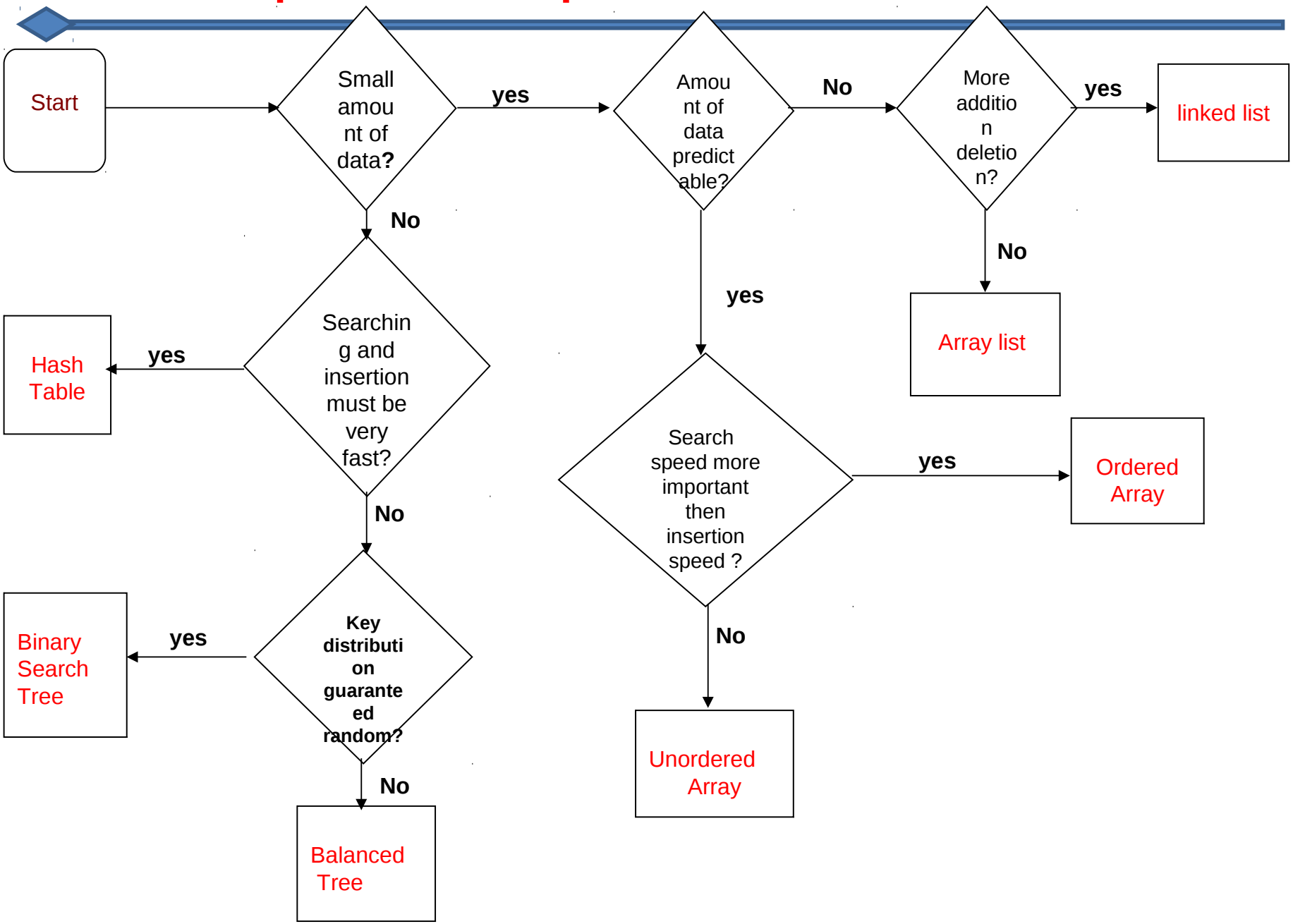




Primerjava Link list : Array List

Link List	Array List
Ni naključnega dostopa dostop	Hiter naključni
Hitro rokovanje	ovanje

Izbira primerne podatkovne strukture



Urejanje oziroma sortiranje

Imamo dva načina definiranja ureditve objektov:

- Vsak razred lahko definira “naravno ureditev” med instancami z uporabo vmesnika Comparable.

```
int compareTo(Object o)
```

- Poljubno ureditev med objekti lahko definiramo s “komparatorji”, razredi, ki implementirajo vmesnik Comparator.

```
int compare(Object o1, Object o2)
```

Ureditev, definirana s strani uporabnika

Nizi, urejeni po obratnem abecednem redu

```
public class StringComparator   implements Comparator  {
        public int compare(Object o1, Object o2) {
            if (o1 != null &&
                o2 != null &&
                o1 instanceof String &&
                o2 instanceof String) {
                    String s1 = (String) o1;
                    String s2 = (String) o2;
                 return - (s1.compareTo(s2));
            } else {  return 0;  }
    }
}
```

Sortiranje

```
public class CountComparator    implements Comparator {
  public int compare(Object o1, Object o2) {
    if (o1 != null &&
        o2 != null &&
        o1 instanceof Count &&
        o2 instanceof Count) {
      Count c1 = (Count) o1;
      Count c2 = (Count) o2;
      return (c2.i - c1.i);
    } else { return 0; }
  }
}
```

Frekvenca besed malo drugače

```
public class WordFrequency4 {
  static public void main(String[] args) {
    <same as WordFrequency>
    List list = new ArrayList(words.values());
    Collections.sort(list, new CountComparator());
    Iterator iter = list.iterator();
    while (iter.hasNext()) {
        count = (Count) iter.next();
        word = count.word;
        System.out.println(word + (word.length() < 8 ? "\t\t" : "\t") +  count.i);
    }
  }
}
```

In še izpis v tem primeru

Če uporabimo besedilo “President Lincoln's The Gettysburg Address”, dobimo:

```
the             13
that            12
we              10
here            8
to              8
a               7
and             6
......
consecrate      1
world           1
consecrated     1
remember        1
did             1
work            1
fathers         1
```

Uvod v generike

Javanska kolekcija je fleksibilna podatkovna struktura, ki laahko vsebuje heterogene objekte in imajo lahko elementi različen tip reference.

Skrb programerja je, da vodi evidenco o tem, kakšne tipe objektov vsebuje kolekcija.

Primer: V kolekcijo želimo dodajati cela števila. Vendar vanjo lahko vstavljamo le objekte. Zato moramo vsako celoštevilčno spremenljivko prej pretvoriti v ustrezen referenčni tip (torej Integer). Ko pa uporabljamo elemente iz kolekcije splošnih objektov, moramo spet zagotoviti pravi tip (torej uporabiti kasto (Integer))

Zato so taki javanski programi težje berljivi in bolj pogosto pride do napak v času izvajanja.

Uporabimo generike

Z uporabo generikov kolekcije ne obravnavamo več kot seznam referenc na objekte in lahko razločujemo med kolekcijami referenc na Integer ali referenc na Byte itd.

Kolekcija z generičnim tipom ima parameter tipa, ki specificira tip elementov, ki so v taki kolekciji pomnjeni.

Primer

Imejmo povezani seznam najprej brez uporabe generikov. Dodajmo mu celoštevilčno vrednost in jo nato preberimo nazaj:

```
LinkedList list = new LinkedList();
list.add(new Integer(1));
   . . . . .
Integer num = (Integer) list.get(0);
```

To je sicer zaradi eksplicitne pretvorbe (kaste) varno, vendar bi v času izvajanja lahko prišlo do napake, če bi brali objekt nekega drugega tipa. Z uporabo generikov pa bi to zgledalo tako:

```
LinkedList<Integer> list = new LinkedList<Integer>();
list.add(new Integer(1));
Integer num = list.get(0);
```

Primer (nadaljevanje)

Nered zmanjšamo, če primer zapišemo malo drugače, z uporabo opredmetenja (autoboxing):

```
LinkedList<Integer> list = new
LinkedList<Integer>();
list.add(1);
int num = list.get(0);
```

Še en primer brez generikov

Poglejmo še primer, ko tvorimo kolekcijo z dvema nizoma in enim celoštevilčnim objektom ter nato kolekcijo izpišemo:

```
import java.util.*;
public class Ex1 {
    private void testCollection() {
        List list = new ArrayList();
        list.add(new String(“Dober dan!"));
        list.add(new String(“Na svidenje!"));
        list.add(new Integer(95));
        printCollection(list);
    }
    private void printCollection(Collection c) {
        Iterator i = c.iterator();
        while(i.hasNext()) {
            String item = (String) i.next(); System.out.println("Item: "+item);
    } }
    public static void main(String argv[]) {
    Ex1 e = new Ex1(); e.testCollection(); } }
```

Potrebna je eksplicitna pretvorba.

Pri izpisu tretjega elementa pride do napake

Primer z generiki

```
import java.util.*;
public class Ex2 {
    private void testCollection() {
        List<String> list = new ArrayList<String>();
        list.add(new String(“Dober dan!"));
        list.add(new String("Good bye!"));
        list.add(new Integer(95));
        printCollection(list);
    }
    private void printCollection(Collection c) {
        Iterator<String> i = c.iterator(); while(i.hasNext()) {
            System.out.println("Item: "+i.next());
    } }

    public static void main(String argv[]) {
        Ex2 e = new Ex2(); e.testCollection();
} }
```

O napaki nas bo obvestil že prevajalnik (ne moremo Integer elementa vstaviti v kolekcijo elementov String)

Za konec

“Uporaba kolekcij je stil programiranja, ki omogoča ponovno uporabljivost algoritmov z različnimi tipi podatkov”

“Objektno usmerjeno programiranje je dalo večjo moč tipom... Generično programiranje daje večjo moč algoritmom”