Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 7

Εβδομάδα 7: Κλάσεις συλλογών δεδομένων στην Java

Υλοποιήσεις

Τύποι συλλογών δεδομένων

- List
- Set
- Map

Τεχνικές υλοποίησης

- linked
- array

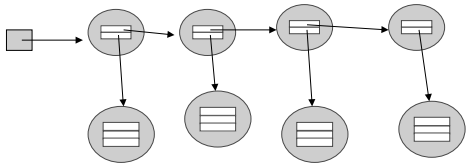
Δομές δεδομένων συλλογών Java

Η Java χρησιμοποιεί τις παρακάτω δομές δεδομένων στις υλοποιήσεις των συλλογών:

- Συνδεδεμένη λίστα [linked list]
- Δυναμικό διάνυσμα [resizable array]
- «Ισοσκελισμένο» δένδρο [balnanced tree]
- Πίνακα κατακερματισμού [hash table]

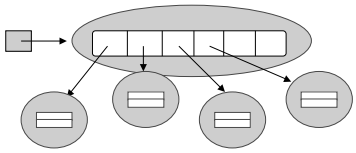
Συνδεδεμένη λίστα [linked list]

- Δομή βασισμένη αποκλειστικά σε «συνδέσμους» [links]



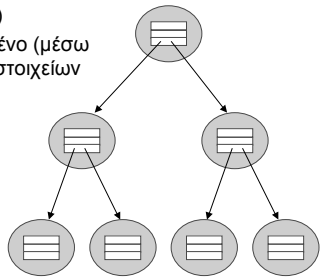
Δυναμικό διάνυσμα [resizable array]

- Βασίζεται σε απλά διανύσματα
- Χρησιμοποιεί επανακαταχώρηση [reallocation]



«Ισοσκελισμένο» δένδρο [balnanced tree]

- Συνδεδεμένη δομή
- Δυαδικό (στην Java)
- Πάντοτε ισοσκελισμένο (μέσω ανακατανομής των στοιχείων του [reshuffling])

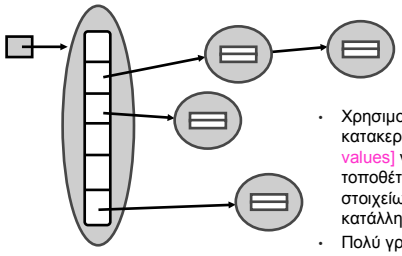


Αντώνιος Συμβώνης, Πανεπιστήμιο Ιωαννίνων

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 7

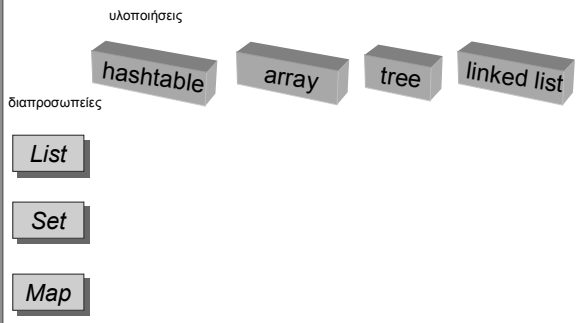
Πίνακας κατακερματισμού [hash table]

- Συνδυασμός διανύσματος και συνδεδεμένης δομή

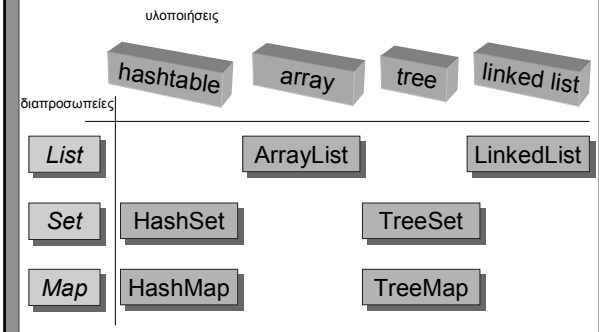


- Χρησιμοποιεί τιμές κατακερματισμού [hash values] για την τοποθέτηση των στοιχείων στην κατάλληλη θέση
- Πολύ γρήγορη ανάκτηση στοιχείων

Υλοποιήσεις



Υλοποιήσεις συλλογών δεδομένων Java



Υλοποιήσεις συλλογών δεδομένων Java

- **List**:
  - **`ArrayList`** (σταθερός χρόνος προσπέλασης, χρησιμοποιείται στις περισσότερες περιπτώσεις)
  - **`LinkedList`** (γρήγορη εισαγωγή στοιχείων)
- **Set**:
  - **`HashSet`** (γρήγορη δομή, χρησιμοποιείται στις περισσότερες περιπτώσεις)
  - **`TreeSet`** (ταξινομημένη)
- **Map**:
  - **`HashMap`** (γρήγορη δομή)
  - **`TreeMap`** (ταξινομημένη)

Οι συλλογές στην Java 1.1

Τι απέγιναν οι κλάσεις **`Vector`** και **`Hashtable`**;

Στην Java 2 (JDK 1.2)

- Η κλάση **`Vector`** έχει αντικατασταθεί από την **`ArrayList`**
- Η κλάση **`Hashtable`** έχει αντικατασταθεί από την **`HashMap`**

Στην Java 2 πρέπει να χρησιμοποιούμε τις νέες κλάσεις συλλογής δεδομένων

Χρήση των συλλογών δεδομένων…

- Αρχικά: επιλέξτε τη διαπροσωπία (λειτουργικότητα)
- Κατόπιν: επιλέξτε την υλοποίηση (κλάση)

- Δημιουργήστε μια κλάση
- Καταχωρείστε αντικείμενα συλλογών σε μεταβλητές τύπου διαπροσωπίας

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 7

Χρήση διαπροσωπίας ως τύπου

Παράδειγμα: χρήση της **LinkedList** ως τύπο **List**

```
LinkedList things;
things = new LinkedList();
process(things);

void process(LinkedList listToProcess)
{
   ...
}
```


Χρήση εναλλακτικής υλοποίησης

```
LinkedList things;
things = new LinkedList();
process(things);

void process(LinkedList listToProcess)
{
   ...
}
```

Εάν αλλάξουμε τη συνδεδεμένη λίστα σε ArrayList, κάθε χρήση της LinkedList πρέπει να αντικατασταθεί με ArrayList (δηλώσεις μεταβλητών, δηλώσεις παραμέτρων, δημιουργία αντικειμένων)

Χρήση εναλλακτικής υλοποίησης (2)

```
ArrayList things;
things = new ArrayList();
process(things);

void process(ArrayList listToProcess)
{
   ...
}
```

Η εύρεση αναφορών της αρχικής κλάσης (**LinkedList**) και η αντικατάσταση τους με αναφορές στη νέα κλάση (**ArrayList**) είναι μια χρονοβόρα και επιρρεπής στα λάθη διαδικασία.

Χρήση εναλλακτικής υλοποίησης (3)

Η λύση: χρησιμοποιήστε μόνο διαπροσωπείες (List) στις δηλώσεις. Χρησιμοποιήστε την κλάση υλοποίησης μια μόνο φορά για τη δημιουργία του αντικειμένου

```
List things = new LinkedList();
process(things);

void process(List listToProcess)
{
   ...
}
```

«Διαπροσπελαστές» [iterators]

Οι **διαπροσπελαστές** [iterators] χρησιμοποιούνται για να προσπελαστούν τα αντικείμενα μίας συλλογής δεδομένων

Επιστρέφει ένα αντικείμενο διαπροσπελαστή προς μια συλλογή

```
public interface Collection
{
      ...
      Iterator iterator();
}

public interface Iterator
{
      boolean hasNext();
      Object next();
      void remove();
}
```


Χρήση διαπροσπελαστών [iterators]

Συνήθης κώδικας για την προσπέλαση των στοιχείων μιας συλλογής:

```
void printAll(Collection people)
{
   for (Iterator i = people.iterator();
        i.hasNext(); )
   {
      Person person = (Person)i.next();
      person.print();
   }
}
```

Πολυμορφικός

(Όμοιοι με τους «enumerations» της Java 1.1, με σωστή λειτουργία κατά τη διαγραφή στοιχείων)

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 7

«Μαζικές» [bulk] λειτουργίες

`boolean containsAll(Collection c);`
Επιστρέφει true εάν η συλλογή περιέχει όλα τα στοιχεία της συλλογής `c` που δίδεται σαν παράμετρος.

`boolean addAll(Collection c);`
Εισάγει όλα τα στοιχεία της συλλογής `c` που δίδεται σαν παράμετρος στην παρούσα συλλογή.

`boolean removeAll(Collection c);`
Διαγράφει από την συλλογή τα στοιχεία τα οποία περιέχονται επίσης στην συλλογή `c` που δίδεται σαν παράμετρος

«Μαζικές» [bulk] λειτουργίες (2)

`boolean retainAll(Collection c);`
Κρατάει στη συλλογή τα στοιχεία τα οποία περιέχονται επίσης στην συλλογή `c` που δίδεται σαν παράμετρος και διαγράφει τα υπόλοιπα.

`void clear();`
Διαγράφει όλα τα στοιχεία από τη συλλογή.

Λειτουργίες διανυσμάτων

Για συμβατότητα με προηγούμενες εκδόσεις [backwards compatibility] οι συλλογές μπορεί να μετατραπούν σε διανύσματα:

```
Object[] toArray();
Object[] toArray(Object a[]);
```

Μετατροπή σε διανυσμα από `Object`s:

```
Object[] a = c.toArray();
```

Γεμίζει προϋπάρχων διάνυσμα από `String`s

```
c.toArray(new String[c.size()]);
```

Μετατρέπει σε νέο διανυσμα από `String`s:

```
String[] a = (String[])c.toArray(new String[0]);
```


Αλγόριθμοι

Στην κλάση **`Collections`** περιέχονται διάφοροι γενικής χρήσης αλγόριθμοι (στατικές μέθοδοι):

static int binarySearch(List list, Object key)
Searches the specified list for the specified object using the binary search algorithm.

static void copy(List dest, List src)
Copies all of the elements from one list into another.

static void fill(List list, Object o)
Replaces all of the elements of the specified list with the specified element.

static Object max(Collection coll)
Returns the maximum element of the given collection, according to the natural ordering of its elements.

static Object min(Collection coll)
Returns the minimum element of the given collection, according to the natural ordering of its elements.

Αλγόριθμοι (2)

static void reverse(List l)
Reverses the order of the elements in the specified list.

static void shuffle(List list)
Randomly permutes the specified list using a default source of randomness.

static void sort(List list)
Sorts the specified list into ascending order, according to the natural ordering of its elements.

static Set singleton(Object o)
Returns an immutable set containing only the specified object.

```
c.removeAll(Collections.singleton(e));
```


Η διαπροσωπία `Set`

Ήδη δηλωμένα στην `Collection`

```
public interface Set {
      int size();
      boolean isEmpty();
      boolean contains(Object element);
      boolean add(Object element);
      boolean remove(Object element);
      Iterator iterator();

      boolean containsAll(Collection c);
      boolean addAll(Collection c);
      boolean removeAll(Collection c);
      boolean retainAll(Collection c);
      void clear();

      Object[] toArray();
      Object[] toArray(Object a[]);
}
```

Η διαπροσωπία `List`

```
public interface List extends Collection {
      // Positional Access
      Object get(int index);
      Object set(int index, Object element);
      void add(int index, Object element);
      Object remove(int index);
      abstract boolean addAll(int index, Collection c);

      // Search
      int indexOf(Object o);
      int lastIndexOf(Object o);

      ListIterator listIterator();
      ListIterator listIterator(int index);
      List subList(int from, int to);
}
```


Η διαπροσωπία `Map` (όχι πλήρης...)

```
public interface Map {
      Object put(Object key, Object value);
      Object get(Object key);
      boolean containsKey(Object key);
      boolean containsValue(Object value);

      public Set keySet();
      public Collection values();
      public Set entrySet();

      // Interface for entrySet elements
      public interface Entry {
            Object getKey();
            Object getValue();
            Object setValue(Object value);
      }
}
```