Java Collections Framework

Γιώργος Θάνος
gthanos@uth.gr
Γραφείο: Γ5/8, 3ος όροφος
Γκλαβάνη 37

Εισαγωγή

Το Java Collections Framework υλοποιεί δομές αποθήκευσης και ανάκτησης δεδομένων καθώς και αλγορίθμους εύρεσης και ταξινόμησης. Συνοπτικά απαρτίζεται από τα εξής:

- **Interfaces:** Γενικοί τύποι δεδομένων που προτυποποιούν τη συμπεριφορά των κλάσεων που τα υλοποιούν. Για παράδειγμα προτυποποιούνται τα interfaces `List`, `Queue`, `Map`, `Set`, `SortedSet`
- **Interface Implementations:** Οι κλάσεις οι οποίες υλοποιούν τα συγκεκριμένα Interfaces. Για παράδειγμα, το interface `List` υλοποιείται από τις κλάσεις `ArrayList` και `LinkedList`.
- **Αλγόριθμοι:** υλοποιήσεις αλγορίθμων, όπως αλγόριθμοι αναζητήσεως και αλγόριθμοι ταξινομήσεως.

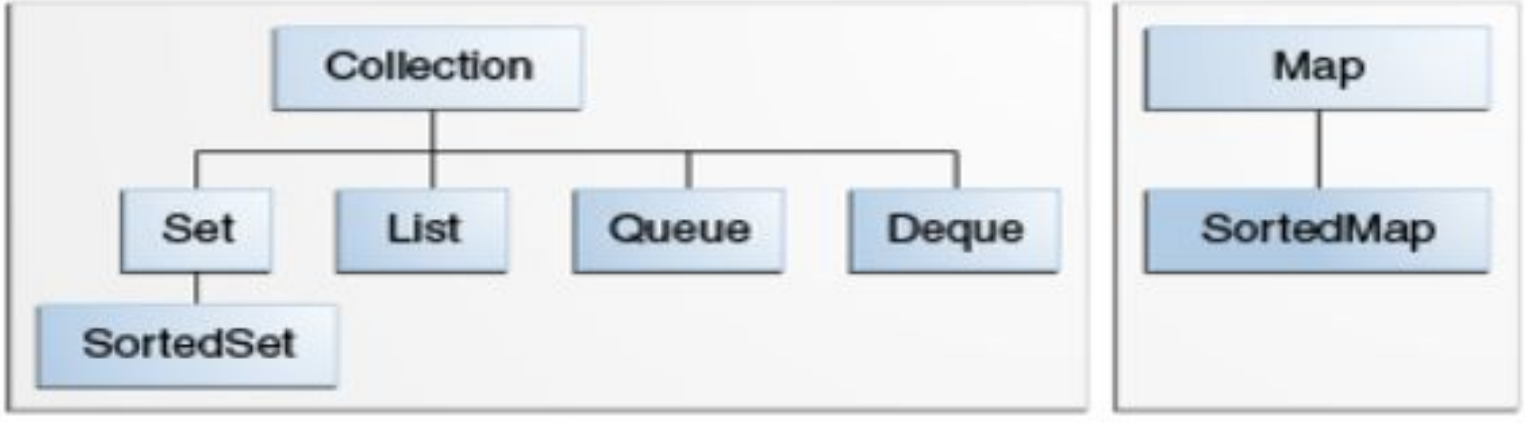
Πλεονεκτήματα χρήσης του JCF

Λιγότερος κώδικας προς ανάπτυξη: Κάθε φορά που χρειάζεστε μία διασυνδεδεμένη λίστα ή ένα δένδρο είναι ευκολότερο να το πάρετε έτοιμο.

Αύξηση της ταχύτητας και της ποιότητας του τελικού προγράμματος: Καθώς το JCF αποτελείται από βέλτιστες υλοποιήσεις είναι μάλλον απίθανο να υλοποιήσετε καλύτερα δομές δεδομένων και αλγορίθμους που υλοποιεί το framework.

Ενοποίηση σχετικά ανομοιογενών APIs: Τα APIs που παρέχει το JCF διατηρούν κοινά χαρακτηριστικά που κάνουν ευκολότερη την χρήση των δομών δεδομένων και των αλγορίθμων. Κατά συνέπεια, η εκμάθηση του JCF είναι σχετικά εύκολη.

JCF Interfaces



Collection: Το βασικό interface στην ιεραρχία των interfaces. Το συγκεκριμένο interface παρέχει κάποιες μεθόδους οι οποίες με την σειρά τους υλοποιούνται από άλλα Interfaces. Αποτελεί ένα σύνολο κοινών μεθόδων τις οποίες πρέπει να μοιράζονται τα παρακάτω interfaces που το κληρονομούν. Δεν υπάρχουν υλοποιήσεις κλάσεων για το συγκεκριμένο interface, αλλά μόνο για τους απογόνους αυτού.

Set: Πρόκειται για ένα Collection που δεν μπορεί να αποθηκεύσει δύο φορές το ίδιο αντικείμενο (τα στοιχεία του είναι μοναδικά).

Sorted Set: Ένα Set που διατηρεί την σειρά των δεδομένων του με βάση συγκεκριμένους κανόνες ταξινόμησης.

List: Μία διασυνδεδεμένη λίστα. Χαρακτηριστικό της λίστας είναι ότι μπορούμε να διατρέξουμε τα στοιχεία με την σειρά που εισάγονται. Πρόκειται για διπλά διασυνδεδεμένη λίστα.

Queue: Υλοποίησης ενός FIFO Queue.

Dequeue: Υλοποίηση ενός LIFO (Last In, First Out) Queue.

Map: Ένα αντικείμενο που αντιστοιχεί κλειδιά σε δεδομένα. Κάθε κλειδί οδηγεί μοναδικά στο αντίστοιχο δεδομένο, επομένως δεν μπορούμε να έχουμε πολλαπλές καταχωρήσεις κλειδιών.

Sorted Map: Ένα Map που διατηρεί την σειρά των δεδομένων του με βάση συγκεκριμένους κανόνες ταξινόμησης.

Interface java.util.Collection

Interface java.util.Collection - Παράδειγμα

```
import java.util.*;
import java.lang.*;
public class StudentCollection {
  private Collection<Student> students;
  public StudentCollection() {
    students = new LinkedList<Student>();
    populateList();
  }
  public final void populateList() {
    students.add(new Student("John",
"Smith"));
    students.add(new Student("Stanley",
"Peters"));
  }
```

```
  public void iterateList() {
    for(Student st: students) {
      System.out.println(st.toString());
    }
  }

  public static void main(String args[])
{
    StudentCollection stl =
         new StudentCollection();
    stl.iterateList();
  }
}
```

Interface java.util.Collection - Διάτρεξη

Διάτρεξη μέσω for

```
public void iterateList() {
   for(Student st: students) {
      System.out.println(st);
   }
}
```

Διάτρεξη μέσω iterator

```
public void iterateList() {
   Iterator<Student> it =
              students.iterator();
   while( it.hasNext() ) {

 System.out.println(it.next());
     }
   }
```

Interface java.util.Collection - Μέθοδοι

isEmpty(): Εξετάζει αν το Collection έχει περιεχόμενα ή όχι επιστρέφοντας true/false.

size(): Επιστρέφει τον αριθμό των αντικειμένων που περιέχει το Collection ή μηδέν αν είναι άδειο.

iterator(): Επιστρέφει ένα Iterator<E> object για την διάτρεξη του Collection.

clear(): Διαγράφει όλα τα περιεχόμενα του Collection.

containsAll(Collection<?> c): Επιστρέφει true εάν υπάρχουν όλα τα μέλη της c στην αρχική λίστα. Διαφορετικά επιστρέφει false.

addAll(Collection<?> c): Προσθέτει όλες τις εγγραφές που περιέχονται στο Collection c.

removeAll(Collection<?> c): Αφαιρεί όλες τις εγγραφές που περιέχονται στο Collection c.

Interfaces java.util.Set & java.util.SortedSet

Το Set είναι μία λίστα αντικειμένων με τον επιπλέον περιορισμό ότι όλα τα στοιχεία στο Set είναι μοναδικά.

Το SortedSet είναι ένα Set του οποίου τα στοιχεία είναι μοναδικά και ταξινομημένα.

Interface java.util.Set - Παράδειγμα

```
public class FindDups2 {
  public static void main(String[] args) {
    Set<String> uniques = new HashSet<String>();
    Set<String> dups = new HashSet<String>();
    for (String a : args)
      if (!uniques.add(a))
        dups.add(a);
    uniques.removeAll(dups);
    System.out.println("Unique words:    " + uniques);
    System.out.println("Duplicate words: " + dups);
  }
}
```

Το Set είναι ένα Collection το οποίο δεν επιτρέπει διπλές εγγραφές.

Υλοποιήσεις του Interface Set

HashSet: Υλοποιεί το Set μέσα από ένα HashTable. Γρήγορο στην αναζήτηση. Δεν εγγυάται ότι η σειρά διάτρεξης είναι η σειρά με την οποία εισήχθησαν τα δεδομένα. Απαιτεί κατά κανόνα περισσότερο χώρο αποθήκευσης από τον στοιχεία που περιέχει το Set.

TreeSet: Υλοποιεί το Set μέσα από ένα Red-Black tree. Πιο αργό στην αναζήτηση, αλλά επίσης αρκετά γρήγορο. Δεν εγγυάται ότι η σειρά διάτρεξης είναι η σειρά με την οποία εισάγαμε τα δεδομένα. Η σειρά διάτρεξης είναι η σειρά κατάταξης των στοιχείων (υλοποιεί τον interface SortedSet).

LinkedHashSet: Υλοποιεί το Set μέσα από ένα HashTable με παράλληλη χρήση διπλά διασυνδεδεομένης λίστας. Γρήγορο στην αναζήτηση. Εγγυάται ότι η σειρά διάτρεξης του *keySet* είναι η σειρά με την οποία εισάγαμε τα δεδομένα, λόγω της ύπαρξης της λίστας. Απαιτεί κατά κανόνα περισσότερο χώρο αποθήκευσης από τον στοιχεία που περιέχει το Set.

Interface java.util.SortedSet

Το interface **SortedSet** είναι ένα **Set** που διατηρεί τα στοιχεία του σε αύξουσα σειρά με βάση την υλοποίηση του interface Comparable ή ενός Comparator για τον συγκεκριμένο τύπο δεδομένων.

Κλάσεις που υλοποιούν το συγκεκριμένο interface είναι οι εξής:

- **TreeSet**: Υλοποιεί το interface με τη βοήθεια ενός **RedBlackTree**.
- **ConcurrentSkipListSet**: Υλοποιεί το interface με την βοήθεια ενός συγχρονισμένου **SkipList**.

Interface java.util.SortedSet

Επιπλέον μέθοδοι σε σχέση με το interface java.util.Set είναι οι εξής:

- **first()** - Επιστρέφει το 1ο στοιχείο από το Set.
- **last()** - Επιστρέφει το τελευταίο στοιχείο από το Set.
- **headSet(E toElement)** - Επιστρέφει το υποσύνολο του Set που τα στοιχεία του είναι μικρότερα από την τιμή *toElement*.
- **tailSet(E fromElement)** - Επιστρέφει το υποσύνολο του Set που τα στοιχεία του είναι μεγαλύτερα ή ίσα από την τιμή *fromElement*.
- **subSet(E fromElement, E toElement)** - Επιστρέφει το υποσύνολο του Set από *fromElement* (μαζί με το *fromElement*) έως *toElement* (χωρίς το *toElement*).

Interface java.util.List

Υλοποιεί μία διπλά διασυνδεδεμένη λίστα.

Interface java.util.List

Το interface **List** υλοποιεί μία διπλά διασυνδεδεμένη λίστα. Η βασική διαφορά σε σχέση με το **Set** interface είναι ότι ένα αντικείμενο μπορεί να εμφανίζεται περισσότερες από μία φορές μέσα στη λίστα.

Η σειρά των αντικειμένων στη λίστα διατηρείται και είναι η σειρά με την οποία εισήχθησαν (όλα τα νέα αντικείμενα εισάγονται στο τέλος της λίστας).

Μπορείτε να σκεφτείτε την λίστα σαν ένα πίνακα, όπου έχετε πρόσβαση στο i-στο στοιχείο του.

Επιπλέον μέθοδοι σε σχέση με το interface java.util.Collection

get(int index) - Επιστρέφει την τιμή στην θέση index.

indexOf(Object o) - Επιστρέφει την 1η θέση του αντικειμένου στην λίστα ή -1 αν δεν περιέχεται το αντικείμενο.

lastIndexOf(Object o) - Επιστρέφει την τελευταία θέση του αντικειμένου στην λίστα ή -1 αν δεν περιέχεται το αντικείμενο.

subList(int fromIndex, int toIndex) - Επιστρέφει μία υπολίστα της αρχικής λίστας, από fromIndex έως toIndex.

set(int index, E element) - Αντικαθιστά το αντικείμενο στην θέση index με το element επιστρέφοντας το αντικείμενο που ήταν αρχικά αποθηκευμένο. Εάν το index που δίνεται είναι εκτός των ορίων της λίστας throws IndexOutOfBoundsException.

Η μέθοδος **listIterator()** επιστρέφει ένα **ListIterator** object που επιτρέπει την διάτρεξη της λίστας από το τέλος προς την αρχή, όπως παρακάτω.

Υλοποιήσεις του interface java.util.List

Δύο βασικές κλάσεις που υλοποιούν το συγκεκριμένο interface **ArrayList** και **LinkedList**.

Χρόνος αναζήτησης στοιχείου: Η **ArrayList** έχει σταθερό χρόνο επιστροφής του i-στου στοιχείου, ενώ η **LinkedList** έχει γραμμικό χρόνο.

Εισαγωγή στοιχείου στο τέλος: Και οι δύο έχουν τον ίδιο χρόνο εισαγωγής.

Εισαγωγή στοιχείου στην αρχή ή στην μέση: Η **ArrayList** έχει γραμμικό χρόνο εισαγωγής ως προς το μέγεθος της λίστας, ενώ η **LinkedList** έχει σταθερό χρόνο.

Η **LinkedList** έχει τις επιπλέον μεθόδους **addFirst**, **getFirst**, **removeFirst**, **addLast**, **getLast** και **removeLast**.

Η **LinkedList** υλοποιεί το **Queue** interface.

```
import java.util.*;
import java.io.*;
public class JCFCollectionExample {
  public static void main(String []args) {
    try (Scanner sc = new Scanner(new File(args[0]))) {
      Collection<String> coll = new LinkedHashSet<>();
      while(sc.hasNext()) {
        coll.add(sc.next());
      }
      Iterator<String> it = coll.iterator();
      while(it.hasNext()) {
        System.out.println(it.next());
      }
    }
    catch(Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Collection Example

Διαβάζουμε μία σειρά από λέξεις από αρχείο λεξικού τις οποίες στη συνέχεια εκτυπώνουμε. Ανάλογα με την δομή που θα επιλέξουμε, αλλάζει η σειρά με την οποία διατρέχει τα δεδομένα ο iterator(). Οι επιλογές που έχουμε είναι 3:

1. Χρήση λίστας: Τα δεδομένα διατρέχονται με τη σειρά που εισήχθησαν.
2. Χρήση HashTable: Τα δεδομένα διατρέχονται με τυχαία σειρά.
3. Χρήση δομής με ταξινόμηση (SortedSet): Τα δεδομένα διατρέχονται ταξινομημένα.

Interface java.util.Map

Υλοποιεί μία δομή αποθήκευσης ζευγαριών δεδομένων που αποτελούνται από ένα μοναδικό κλειδί και μία τιμή που αντιστοιχεί στο κλειδί (key, value pair).

Τα κλειδιά είναι μοναδικά, οι τιμές που αντιστοιχούν στα κλειδιά μπορεί να επναλαμβάνονται.

Interface java.util.Map - Παράδειγμα

Ένα αντικείμενο τύπου **Map.Entry** αντιστοιχεί "τιμές" (*values*) σε "κλειδιά" (*keys*). Τόσο οι τιμές όσο και τα κλειδιά μπορεί να είναι οποιουδήποτε τύπου. Ο στόχος της συγκεκριμένης δομής είναι να μπορούμε έχοντας το κλειδί να λάβουμε την τιμή που αντιστοιχεί σε αυτό. Οι μέθοδοι του interface **Map.Entry** δίνονται στον παρακάτω πίνακα.

Μία δομή τύπου **Map** είναι μία δομή που περιέχει εγγραφές τύπου **Map.Entry** με την ιδιαιτερότητα ότι δεν μπορεί να διαθέτει δύο εγγραφές με το ίδιο κλειδί (μπορεί όμως να διαθέτει δύο εγγραφές με διαφορετικά κλειδιά, αλλά ίδιες τιμές).

Υλοποιήσεις του java.util.Map interface

HashMap: Υλοποιεί το Map μέσα από ένα HashTable. Γρήγορο στην αναζήτηση. Δεν εγγυάται ότι η σειρά διάτρεξης είναι η σειρά με την οποία εισάγαμε τα δεδομένα. Απαιτεί κατά κανόνα περισσότερο χώρο αποθήκευσης από τον στοιχεία που περιέχει το Map.

TreeMap: Υλοποιεί το Map μέσα από ένα Red-Black tree. Πιο αργό στην αναζήτηση, αλλά επίσης αρκετά γρήγορο. Δεν εγγυάται ότι η σειρά διάτρεξης είναι η σειρά με την οποία εισάγαμε τα δεδομένα. Η σειρά διάτρεξης είναι η σειρά κατάταξης των στοιχείων (υλοποιεί τον interface SortedMap).

LinkedHashMap: Υλοποιεί το Map μέσα από ένα HashTable με παράλληλη χρήση διπλά διασυνδεδεμένης λίστας. Γρήγορο στην αναζήτηση. Εγγυάται ότι η σειρά διάτρεξης είναι η σειρά με την οποία εισάγαμε τα δεδομένα, λόγω της ύπαρξης της λίστας. Απαιτεί κατά κανόνα περισσότερο χώρο αποθήκευσης από τον στοιχεία που περιέχει το Map.

Βασικές μέθοδοι του interface java.util.Map

containsKey(Object key) - Επιστρέφει true αν η δομή περιέχει το συγκεκριμένο κλειδί.

containsValue(Object value) - Επιστρέφει true αν η δομή περιέχει την συγκεκριμένη τιμή αντιστοιχισμένη με ένα ή περισσότερα κλειδιά.

remove(Object key) - Διαγράφει την εγγραφή που αντιστοιχεί στο κλειδί key επιστρέφοντας την τιμή V ή null αν δεν βρέθηκε το κλειδί.

replace(K key, V value) - Αντικαθιστά την εγγραφή που αντιστοιχεί στο κλειδί key επιστρέφοντας την τιμή V. Δεν γίνεται αντικατάσταση αν δεν βρεθεί το κλειδί στο Map.

put(K key, V value) - Α Διαγράφει την εγγραφή που αντιστοιχεί στο κλειδί key επιστρέφοντας την τιμή V ή null αν δεν υπήρχε προηγούμενη καταχώρηση για το συγκεκριμένο κλειδί.

get(Object key) - Επιστρέφει την τιμή που αντιστοιχεί στο δεδομένο κλειδί ή null αν δεν υπάρχει καταχώρηση για το συγκεκριμένο κλειδί.

Μετατροπή του Map σε μορφή που να μπορείτε να το διατρέξετε (Collection ή Set)

entrySet() - Επιστρέφει ένα Set από καταχωρήσεις τύπου Entry.Map όπου κάθε καταχώρηση περιέχει το συνδυασμό κλειδί-τιμή.

values() - Επιστρέφει ένα Collection με τις τιμές του Map δίχως τα κλειδιά. Μία τιμή μπορεί να εμφανίζεται περισσότερες από μία φορές αν αντιστοιχεί σε περισσότερα του ενός κλειδιά.

keySet() - Επιστρέφει ένα Set με τα κλειδιά.

```
public class JCFMapExample {
  public static final int ALPHABET_SIZE = 26;
  public static void main(String []args) {
    try (Scanner sc = new Scanner(new File(args[0]))) {
      Map<Character,Collection<String>> map = new LinkedHashMap<>();
      while(sc.hasNext()) {
        String word = sc.next();
        if(map.containsKey(word.charAt(0))) {
          Collection<String> coll = map.get(new Character(word.charAt(0)));
          coll.add(word); map.put(new Character(word.charAt(0)), coll);
        }
        else {
          Collection<String> coll = new TreeSet<>();
          coll.add(word); map.put(word.charAt(0), coll);
        }
      }
      for(int i=0; i<ALPHABET_SIZE; i++) {
        Collection<String> coll = map.get(new Character((char)('a' + i)));
        if(coll == null) continue;
        Iterator<String> it = coll.iterator();
        while(it.hasNext())
          System.out.println(it.next());
      }
    }
    catch(Exception ex) {
      ex.printStackTrace();
    }
  }
```

Map Example

Διαβάζουμε μία σειρά από λέξεις από αρχείο λεξικού τις οποίες στη συνέχεια αποθηκεύουμε σε ένα **`Map<Character, Collection<String>`**. Το κλειδί για το Map είναι ο πρώτος χαρακτήρας κάθε λέξης.

Το interface **`Map`** δεν μας παρέχει δυνατότητα διάτρεξης των δεδομένων του. Ανάλογα με την δομή που θα επιλέξουμε για τα Collections, αλλάζει η σειρά με την οποία διατρέχει τα δεδομένα ο iterator() (*δες σχετικά στο προηγούμενο παράδειγμα*)

Interface java.util.SortedMap

Το interface SortedMap είναι ένα Map που διατηρεί τα κλειδιά του σε αύξουσα σειρά με βάση την υλοποίηση του interface Comparable ή της κλάσης Comparator τα κλειδιά.

Κλάσεις που υλοποιούν το συγκεκριμένο *interface* είναι οι εξής:

- **TreeMap**: Υλοποιεί το interface με τη βοήθεια ενός **RedBlackTree**.
- **ConcurrentSkipListMap**: Υλοποιεί το interface με την βοήθεια ενός συγχρονισμένου **SkipList**.

Interface java.util.SortedMap

Επιπλέον μέθοδοι σε σχέση με το interface java.util.Set είναι οι εξής:

- **firstKey()** - Επιστρέφει το 1ο κλειδί.
- **lastKey()** - Επιστρέφει το τελευταίο κλειδί.
- **headMap(K toKey)** - Επιστρέφει το υποσύνολο του Map που τα κλειδιά του είναι μικρότερα από την τιμή toKey.
- **tailMap(K fromKey)** - Επιστρέφει το υποσύνολο του Map που τα κλειδιά του του είναι μεγαλύτερα ή ίσα από την τιμή fromKey.
- **subMap(K fromKey, K toKey)** - Επιστρέφει το υποσύνολο του Set από fromKey (μαζί με το fromKey) έως toKey (χωρίς το toKey).

Αλγόριθμοι

Οι βασικοί αλγόριθμοι του JFC υλοποιούνται στην κλάση Collections

Sorting - N Log(n) performance

```
import java.util.*;

public class Sort {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

Shuffling - Ανακατεύει στα στοιχεία της λίστας σε τυχαία σειρά

```
import java.util.*;

public class Shuffle {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.shuffle(list);
        System.out.println(list);
    }
}
```

Reverse - Αντιστρέφει στα στοιχεία της λίστας

```
import java.util.*;

public class Reverse {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.reverse(list);
        System.out.println(list);
    }
}
```

Copy - Αντιγράφει τη μία λίστα πάνω στην άλλη

```
import java.util.*;


public class Copy {
  public static void main(String args[])
 {
    // create two lists
    List<String> srclst = new
                ArrayList<String>(5);
    List<String> destlst = new
                ArrayList<String>(10);
    // populate two lists
    srclst.add("Java");
    srclst.add("is");
    srclst.add("best");
    destlst.add("C++");
    destlst.add("is");
    destlst.add("older");
    destlst.add("and");
    destlst.add("fast!");
    // copy into dest list
    Collections.copy(destlst, srclst);
    System.out.println("Value of source list:
"+srclst);
    System.out.println("Value of destination
list: "+destlst);
  }
}
```

προϋπόθεση: η λίστα προορισμού να έχει περισσότερα στοιχεία από την λίστα που θα αντιγραφεί

Swap - Αντιμεταθέτει τα στοιχεία μεταξύ δύο θέσεων της λίστας.


```
import java.util.*;

public class Swap {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    System.out.println("List elements before swap: "+list);
    Collections.swap(list, 0, 2);
    System.out.println("List elements after swap: "+list);
  }
}
```

AddAll - Προσθέτει επιπλέον στοιχεία στη λίστα


```
import java.util.*;

public class AddAll {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    Collections.addAll(list, "how", "are", "you?");
    System.out.println(list);
  }
}
```

Min & Max

```
import java.util.*;


public class MinMax {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);


    System.out.println("Min value: "+Collections.min(list,null)+" | Max
value: "+ Collections.max(list,null) );
  }
}
```