•Collection



**Collection (conteneur)** est un objet qui regroupe plusieurs éléments en une seule unité

une collection peut être utilisée pour stocker et manipuler des données et pour transmettre des données d'une méthode à une autre

réduit l'effort de programmation
améliore la qualité et les performances du programme
permet l'interopérabilité d'API

**dans le package java.util**

**Collection :** groupe d'objets(conteneur)

- **Interfaces**
- **Implementations**
- **Algorithmes**

**Interfaces**

- Collection
- Set
  - SortedSet
- List
- Queue

- Map
  - SortedMap



## INTERFACE COLLECTION

```
public interface Collection<E> extends Iterable<E> {
    // Basic Operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);         // Optional
    boolean remove(Object element); // Optional
    Iterator<E> iterator();

    // Bulk Operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); // Optional
    boolean removeAll(Collection<?> c);        // Optional
    boolean retainAll(Collection<?> c);        // Optional
    void clear(); // Optional

    // Array Operations
    Object[] toArray();
    }
```

*méthode optionnelle*

```
public boolean add (E element)
{throw new  UnsupportedOperationException();
}
```

*Il s'agit d'une exception dérivée de RunTimeException*
*il n'y a pas throws et catch*

## Iterator

Un itérateur permet de parcourir une collection
La notion d'itérateur est implantée en Java par l'interface Iterator

public boolean hasNext()
public Object next()
public void remove()  (enlève le dernier élément retourné)

for (Iterator i = collec.iterator(); i.hasNext(); ) {
Object o = i.next(); // Recupere l'élément et passe au suivant
// ...
}



**SET** est une collection sans duplication d'éléments
*(les mêmes méthodes mais en tenant compte duplications)*
Implémentations:
HashSet (sans ordre)
TreeSet ( avec un ordre)

```
import java.util.*;

public class FindDups {
    public static void main(String args[]) {
        Set<String> s = new HashSet<String>();
        for (String a : args)
            if (!s.add(a))
                System.out.println("Duplicate detected: "+a);

        System.out.println(s.size()+" distinct words detected: "+s);
    }
}
```

```
import java.util.*;

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);  // Destructive set-difference

        System.out.println("Unique words:    " + uniques);
        System.out.println("Duplicate words: " + dups);
    }
}
```

When run with the same same argument list used earlier (`i came i saw i left`), the program yields the output:

```
Unique words:    [left, saw, came]
Duplicate words: [i]
```



# **LIST** :un ensemble ordonné, duplication possible

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

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

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```

## Implémentations

ArrayList, Vector
LinkedList

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove(); // Optional
    void set(E o); // Optional
    void add(E o); // Optional
}
```

```
public int indexOf(E o) {
    for (ListIterator<E> i = listIterator(); i.hasNext(); )
        if (o==null ? i.next()==null : o.equals(i.next()))
            return i.previousIndex();
    return -1; // Object not found
}
```

```
public static <E> void replace(List<E> s, E val, E newVal) {
   for (ListIterator<E> i = s.listIterator(); i.hasNext(); )
      if (val==null ? i.next()==null : val.equals(i.next()))
         i.set(newVal);
}
```



### Algorithms

- `sort`: Sorts a List using a merge sort algorithm, which provides a fast, stable sort. (A *stable sort* is one that does not reorder equal elements.)
- `shuffle`: Randomly permutes the elements in a `List`.
- `reverse`: Reverses the order of the elements in a `List`.
- `rotate`: Rotates all of the elements in a `List` by a specified distance.
- `swap`: Swaps the elements at specified positions in in a `List`.
- `replaceAll`: Replaces all occurrences of one specified value with another.
- `fill`: Overwrites every element in a `List` with the specified value.
- `copy`: Copies the source `List` into the destination List.
- `binarySearch`: Searches for an element in an ordered `List` using the binary search algorithm.
- `indexOfSubList`: Returns the index of the first sublist of one `List` that is equal to another.
- `lastIndexOfSubList`: Returns the index of the last sublist of one `List` that is equal to another.

**MAP** est une collection qui permet de stocker des associations (clé, valeur)

- Une clé peut correspondre au plus à une valeur
- Deux tableaux associatifs sont égaux s'ils représentent les mêmes associations clé/valeur

**Implémentations** HashMap<K,V>

HashTable



```
// Basic Operations
boolean is Empty()
V put(Object key, Object value); //Optional
V get(Object key)
boolean containsKey(Object key);
boolean containsValue(Object value);
int size();

// Bulk Operations
void putAll(Map<?extends K, ,?extends V> t);
V remove(Object key); //Optional

//View operations
Collection<V> values()
```

```
import java.util.*;

public class Freq {

    public static void main(String args[]) {
        Map<String, Integer> m = new HashMap<String, Integer>();

        // Initialize frequency table from command line
        for (String a : args) {
            int freq =  m.get(a);
            m.put(a, freq==0 ? 1 : freq + 1);
        }

        System.out.println(m.size() + " distinct words detected:");
        System.out.println(m);
    }
}
```