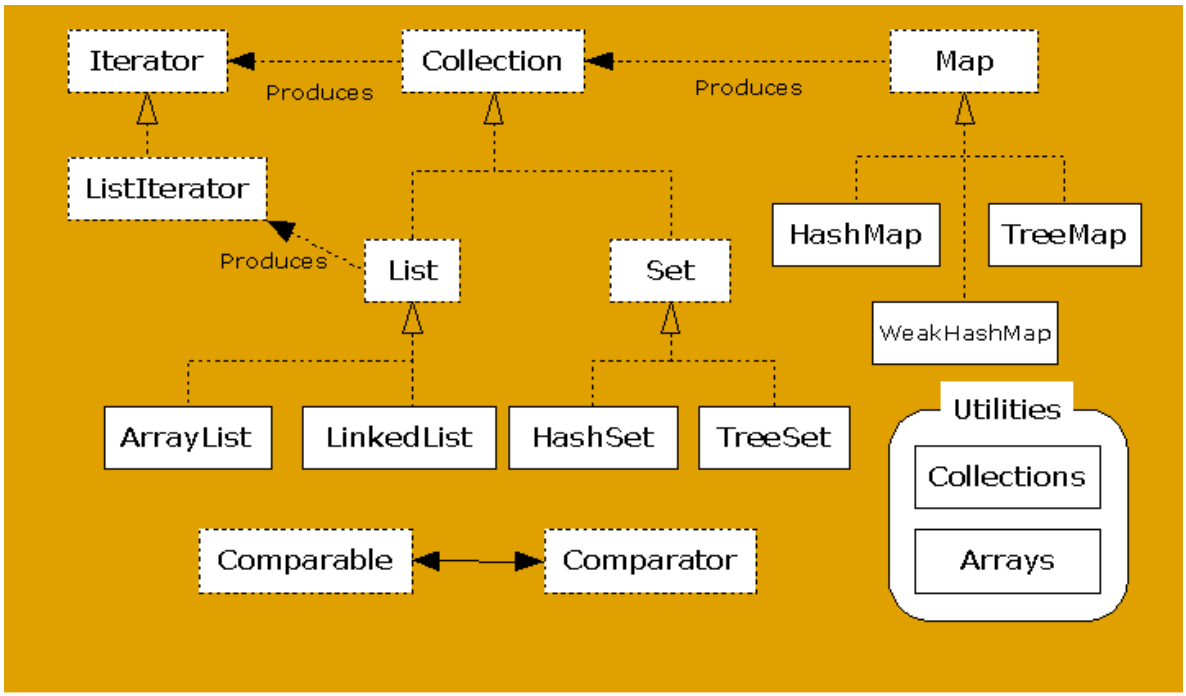
Colecciones en JAVA



La Interface Collection

```
public interface Collection {
        // Basic Operations
        int size();
        boolean isEmpty();
        boolean contains(Object element);
        boolean add(Object element); // Optional
        boolean remove(Object element); // Optional
        Iterator iterator();
        // Bulk Operations
        boolean containsAll(Collection c);
        boolean addAll(Collection c); // Optional
        boolean removeAll(Collection c); // Optional
        boolean retainAll(Collection c); // Optional
        void clear(); // Optional
        // Array Operations
        Object[] toArray();
        Object[] toArray(Object a[]);
}
```

Operaciones sobre listas

UNION
set1.addAll(set2)

INTERSECCION
set1.retainAll(set2)

DIFERENCIA
set1.removeAll(set2)

La Interface List

```
public interface List extends Collection {
        // Positional Access
        Object get(int index);
        Object set(int index, Object element); // Optional
        void add(int index, Object element); // Optional
        Object remove(int index); // Optional
        abstract boolean addAll(int index, Collection c);
        // Optional
        // Search
        int indexOf(Object o);
        int lastIndexOf(Object o);
        // Iteration
        ListIterator listIterator();
        ListIterator listIterator(int index);
        // Range-view
        List subList(int from, int to);
}
```

The **List** interface corresponds to an order group of elements.

Duplicates are allowed.

• Extensions compared to the **Collection** interface
Access to elements via indexes, like arrays

add (int, Object), get(int), remove(int),
set(int, Object) (note set = replace bad name for the method)

Search for elements
indexOf(Object), lastIndexOf(Object)

Specialized **Iterator**, call **ListIterator**
Extraction of sublist
subList(int fromIndex, int toIndex)

La Interface Iterator

```
// an example
public static void main (String[] args){
ArrayList cars = new ArrayList();
for (int i = 0; i < 12; i++)
        cars.add (new Car());
        Iterator it = cats.iterator();
        while (it.hasNext())
        System.out.println ((Car)it.next());
}
```

Classes ArrayList and LinkedList

The classes **ArrayList** and **LinkedList** implement the **List** interface.

• **ArrayList** is an array based implementation where elements can be accessed directly via the **get** and **set** methods.
Default choice for simple sequence.
• **LinkedList** is based on a double linked list Gives better performance on **add** and **remove** compared to **ArrayList**.

Gives poorer performance on **get** and **set** methods compared to **ArrayList**.

Ejemplo ArrayList,

```
import java.util.*;
public class Shuffle {
        public static void main(String args[]) {
                List l = new ArrayList();
                for (int i = 0; i < args.length; i++)
                l.add(args[i]);
                Collections.shuffle(l, new Random());
                System.out.println(l);
        }
}
```

Ejemplo LinkedList,

```
import java.util.*;
```

```
public class MyStack {
        private LinkedList list = new LinkedList();
        public void push(Object o){
                list.addFirst(o);
        }
        public Object top(){
                return list.getFirst();
        }
        public Object pop(){
                return list.removeFirst();
        }

        public static void main(String args[]) {
                Car myCar;
                MyStack s = new MyStack();
                s.push (new Car());
                myCar = (Car)s.pop();
        }
}
```

La Interface Map

```
public interface Map {
        // Basic Operations
        Object put(Object key, Object value);
        Object get(Object key);
        Object remove(Object key);
        boolean containsKey(Object key);
        boolean containsValue(Object value);
        int size();
        boolean isEmpty();
        // Bulk Operations
        void putAll(Map t);
        void clear();
        // Collection Views
        public Set keySet();
        public Collection values();
        public Set entrySet();
        // Interface for entrySet elements
        public interface Entry {
                Object getKey();
                Object getValue();
                Object setValue(Object value);
        }
```

```
}
```

A Map is an object that maps keys to values. Also called an *associative array* or a *dictionary*.

- Methods for adding and deleting
  - **put(Object key, Object value)**
  - **remove (Object key)**
- Methods for extraction objects
  - **get (Object key)**
- Methods to retrieve the keys, the values, and (key, value) pairs
  - **keySet() // returns a Set**
  - **values() // returns a Collection**,
  - **entrySet() // returns a set**

Classes HashMap and TreeMap

The **HashMap** and **HashTree** classes implement the **Map** interface.

- **HashMap**
  - The implementation is based on a hash table.
  - No ordering on (key, value) pairs.
- **TreeMap**
  - The implementation is based on *red-black tree structure*.
  - (key, value) pairs are ordered on the key.

Ejemplo HashMap,

```
import java.util.*;
public class Freq {
private static final Integer ONE = new Integer(1);
        public static void main(String args[]) {
                Map m = new HashMap();
                // Initialize frequency table from command line
                for (int i=0; i < args.length; i++) {
                        Integer freq = (Integer) m.get(args[i]);
                        m.put(args[i], (freq==null ? ONE :
                        new Integer(freq.intValue() + 1)));
                }
        System.out.println(m.size()+" distinct words detected:");
        System.out.println(m);
        }
}
```