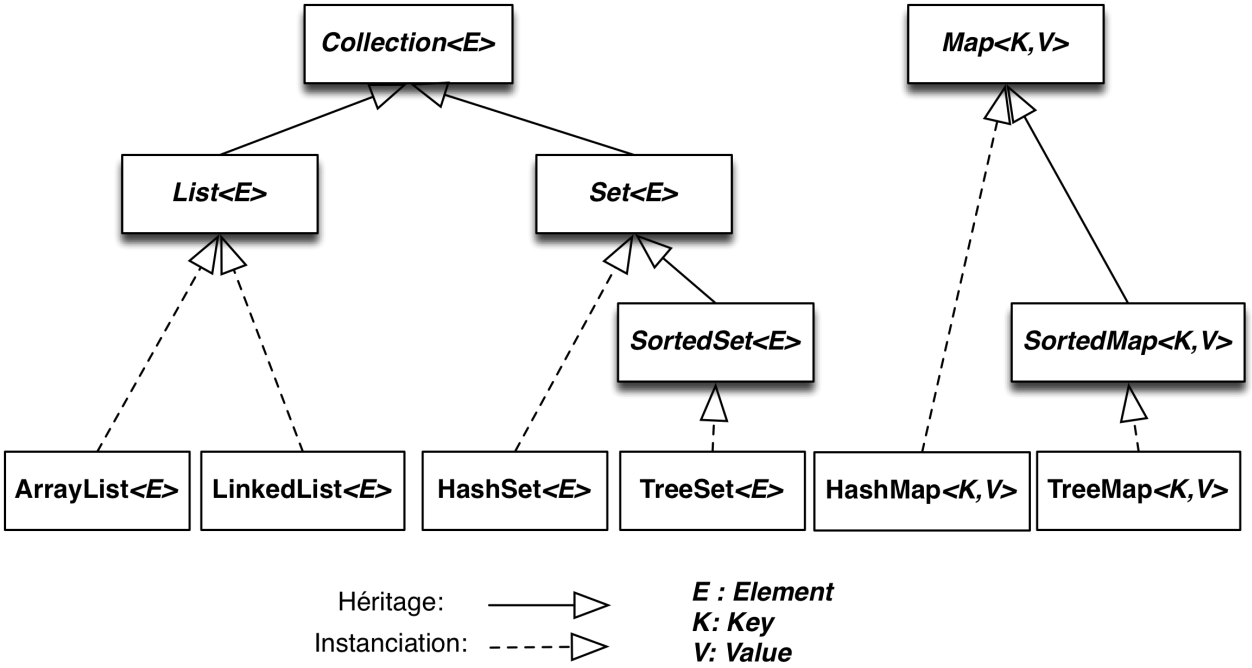
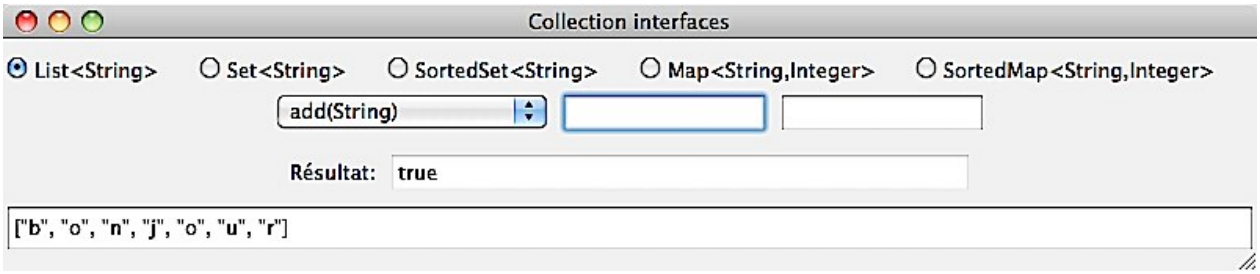
Collection Framework de Java




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

    // Opérations de groupe
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();

    // Transformations en tableaux
    Object[] toArray();
    <T> T[] toArray(T a[]);
}
```

```
public interface Iterator<E> {
    boolean hasNext();
    <E> next();
    void remove();
}

public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();

    boolean hasPrevious();
    E previous();

    int nextIndex();
    int previousIndex();

    void remove();
    void set(E o);
    void add(E o);
}

public interface Comparable<T> {
    public int compareTo(T o);
}

public interface Comparator<T> {
    public int compare(T o1, T o2);
}
```

List

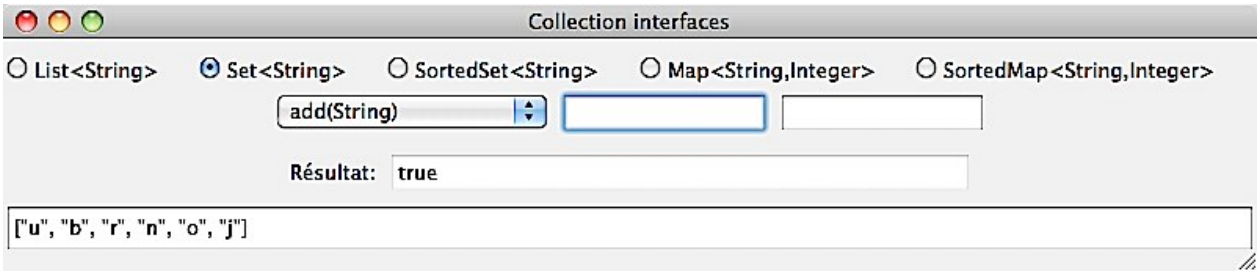



```
public interface List<E> extends Collection<E> {
    // accès par position
    E get(int index);
    E set(int index, E element);
    void add(int index, E element);
    E remove(int index);
    boolean addAll(int index, Collection <? extends <E>>);

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

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

    // vue comme sous-liste
    List<E> subList (int from, int to);
}
```

Exemples de création

```
List<String> l = new ArrayList<String>();
List<Integer> l = new LinkedList<Integer>();
```

Set

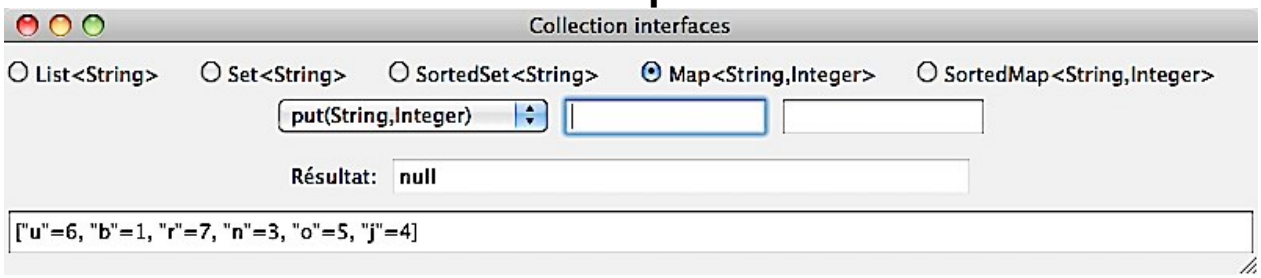



```
public interface Set<E> extends Collection<E>{
    // aucune nouvelle méthode
    // mais s'assure que tous les éléments sont distincts
}
```

Exemple de création

```
Set<String> s = new HashSet<String>();
```

Map

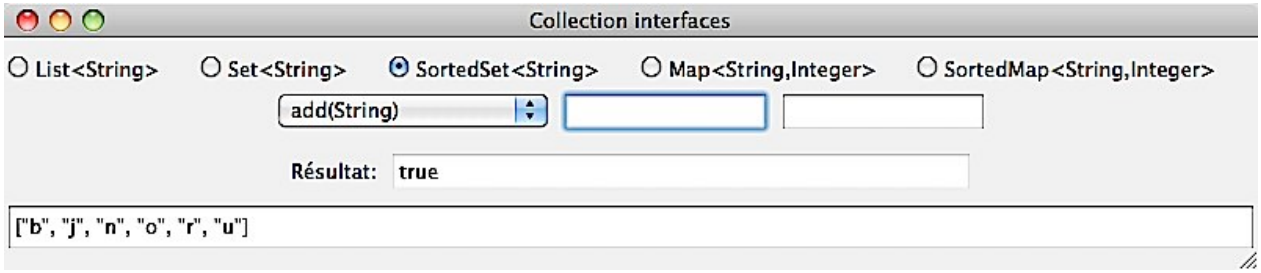



```
public interface Map<K,V> {
    // Opérations de base
    V put(K key, V value);
    V get(K key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
//  Opérations de groupe
    void putAll(Map<? Extends K, ? extends V> m);
    void clear();
//  Vues comme des Collections
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();
// Interface pour les entrées de la table (entrySet)
//      on la désigne par Map.Entry
    public interface Entry<K,V> {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Création

```
Map<String,Integer> m = new HashMap<String,Integer>();
Map<String,Integer> m = new LinkedHashMap<String,Integer>();
```

SortedSet

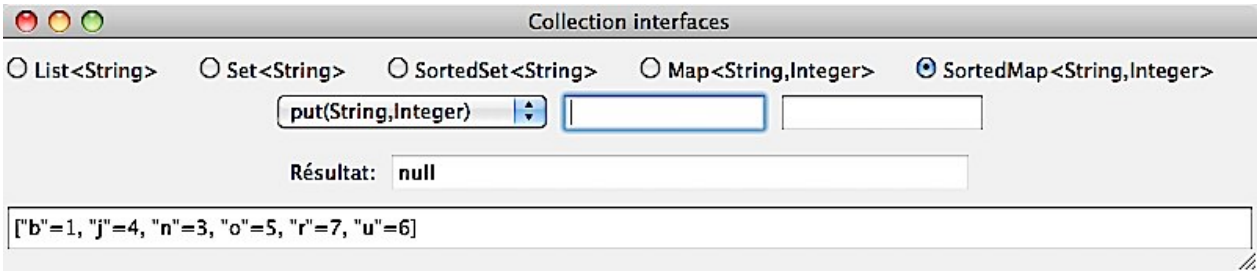



```
public interface SortedSet<E> extends Set<E> {
    // vue comme sous-ensemble
    SortedSet<E> subSet(Object fromElement, Object toElement);
    SortedSet<E> headSet(Object toElement);
    SortedSet<E> tailSet(Object fromElement);

    // points extrêmes
    E first();
    E last();

    // accès au comparateur
    Comparator<? super E> comparator();
}
```

Création

```
SortedSet<String> ss = new TreeSet<String>();
```

SortedMap




```
public interface SortedMap<K,V> extends Map<K,V> {

    // vue comme sous-table
    SortedMap<K,V> subMap(K fromKey, K toKey);
    SortedMap<K,V> headMap(K toKey);
    SortedMap<K,V> tailMap(K fromKey);

    // accès aux clés extrêmes
    K firstKey();
    K lastKey();

    // accès au comparateur
    Comparator<? super K> comparator();
}
```

Création

```
SortedMap<String,Integer> sm = new TreeMap<String,Integer>();
```

Collections

Contient des méthodes statiques qui opèrent et retournent des collections. On ne présente ici que les méthodes qui nous semblent les plus couramment utilisées.

```
public class Collections {
    // listes vides immuables
    public static <T> List<T> emptyList(){…}
    public static <T> Set<T>  emptySet(){…}
    public static <K,V> Map<K,V> emptyMap(){…}

    // Tri
    public static <T extends Comparable<? super T>>
                  void sort(List<T> list){…}
    public static <T> void sort(List<T> list,
                                Comparator<? Super T> c){…}

    // Recherche
    public static <T> int binarySearch (
       List<? extends Comparable<? super T>>list, T key){…}
    public static <T> int binarySearch(
                          List<? extends T> list, T key,
                          Comparator<? super T> c){…}

    public static <T extends Object & Comparable<? super T>>
                     T min(Collection<? extends T> coll){…}
    public static <T extends Object & Comparable<? super T>>
                     T min(Collection<? extends T> coll,
                          Comparator<? super T> comp){…}
    // équivalent pour max

    // Remplissage
    public static <T>void fill(List<? super T> list, T o){ …}
    public static <T>void copy (List<? super T> dest,
                                List<? extends T> src){…}
    // Brassage
    public static void reverse(List<?> l) { ... }
    public static void shuffle(List<?> list) { ... }
    public static void shuffle(List<?> list, Random rnd){…}
```

Arrays

Algorithmes génériques sur les tableaux dont les éléments sont les types prédéfinis (long, int, short, char, byte, float, double) ainsi que Object. Dans la description suivante, ces types sont notés ***T***

```
public class Arrays {
    public static <T>List<T> asList(T... a) {…}

    // Tri
    public static void sort( int[] a) { ... }
    public static void sort( int[] a, int fromIndex,
                                      int toIndex) {…}
    // idem pour boolean, char, double, long, short, float
    public static <T>void sort(T[] a,
                               Comparator<? super T> c){…}
    public static <T>void sort(T[] a,
                             int fromIndex, int toIndex,
                               Comparator<? super T> c){…}

    // Recherche
    public static int binarySearch(int[] a, int key) { ... }
    // idem pour byte, char, double, long, short, float
    public static <T>int binarySearch(T[] a, T key,
                               Comparator<? Super T> c){…}
    // Test d'égalité de tous les éléments des tableaux
    public static boolean equals(int[] a, int[] a2) { ... }

    // Remplissage
    public static void fill( int[] a, int[] val){…}
    public static void fill( int[] a,
                             int fromIndex, int toIndex,
                             int val) { ... }
}
```