Contenedores o Colecciones

Interface Collection

La interface ***Collection*** es implementada por los ***conjuntos*** (*sets*) y las ***listas*** (*lists*). Esta interface declara una serie de métodos generales utilizables con ***Sets*** y ***Lists***

```
public interface java.util.Collection
{
public abstract boolean add(java.lang.Object); // opcional
public abstract boolean addAll(java.util.Collection); // opcional
public abstract void clear(); // opcional
public abstract boolean contains(java.lang.Object);
public abstract boolean containsAll(java.util.Collection);
public abstract boolean equals(java.lang.Object);
public abstract int hashCode();
public abstract boolean isEmpty();
public abstract java.util.Iterator iterator();
public abstract boolean remove(java.lang.Object); // opcional
public abstract boolean removeAll(java.util.Collection); // opcional
public abstract boolean retainAll(java.util.Collection); // opcional
public abstract int size();
public abstract java.lang.Object toArray()[];
public abstract java.lang.Object toArray(java.lang.Object[])[];
}
```

Los métodos indicados como "// opcional" pueden no estar disponibles en algunas implementaciones, como por ejemplo en las clases que no permiten modificar sus objetos. Por supuesto dichos métodos deben ser definidos, pero lo que hacen al ser llamados es lanzar una ***UnsupportedOperationException***.

El método ***add()*** trata de añadir un objeto a una colección, pero puede que no lo consiga si la colección es un ***set*** que ya tiene ese elemento. Devuelve ***true*** si el método ha llegado a modificar la colección. Lo mismo sucede con ***addAll()***. El método ***remove()*** elimina un único elemento (si lo encuentra), y devuelve ***true*** si la colección ha sido modificada.
El método ***iterator()*** devuelve una referencia ***Iterator*** que permite recorrer una colección con los métodos ***next()*** y ***hasNext()***. Permite también borrar el elemento actual con ***remove()***.
Los dos métodos ***toArray()*** permiten convertir una colección en un array.

Interface List

La interfaz ***List*** define métodos para operar con colecciones ordenadas y que pueden tener elementos repetidos. Por ello, dicha interfaz declara métodos adicionales que tienen que ver con el orden y con el acceso a elementos o rangos de elementos. Además de los métodos de ***Collection***, la interfaz ***List*** declara los métodos siguientes:

```
Compiled from List.java
public interface java.util.List extends java.util.Collection
{
public abstract void add(int, java.lang.Object);
public abstract boolean addAll(int, java.util.Collection);
public abstract java.lang.Object get(int);
public abstract int indexOf(java.lang.Object);
public abstract int lastIndexOf(java.lang.Object);
public abstract java.util.ListIterator listIterator();
public abstract java.util.ListIterator listIterator(int);
public abstract java.lang.Object remove(int);
public abstract java.lang.Object set(int, java.lang.Object);
public abstract java.util.List subList(int, int);
}
```

List miLista = new ArrayList(); (sin restricciones)
List<Persona> miLista = new ArrayList<Persona>(); (con restricción)
Métodos: add(objeto); size(), get(posición), remove(índice u objeto)

Existen dos implementaciones de la interface ***List***, que son las clases ***ArrayList*** y ***LinkedList***.
La diferencia está en que la primera almacena los elementos de la colección en un ***array*** de ***Objects***, mientras que la segunda los almacena en una ***lista vinculada***. Los ***arrays*** proporcionan una forma de acceder a los elementos mucho más eficiente que las listas vinculadas. Sin embargo tienen dificultades para crecer (hay que reservar memoria nueva, copiar los elementos del array antiguo y liberar la memoria) y para insertar y/o borrar elementos (hay que desplazar en un sentido o en otro los elementos que están detrás del elemento borrado o insertado). Las ***listas vinculadas*** sólo permiten acceso secuencial, pero tienen una gran flexibilidad para crecer, para borrar y para insertar elementos. El optar por una implementación u otra depende del caso concreto de que se trate.
Interface Map

Interfaz Map: →HashMap
→TreeMap

Cada entrada tiene llave y valor (las llaves son objetos = índice)

Un ***Map*** es una estructura de datos agrupados en parejas ***clave/valor***. Pueden ser considerados como una tabla de dos columnas. La ***clave*** debe ser única y se utiliza para acceder al ***valor***.
Aunque la interfaz ***Map*** no deriva de ***Collection***, es posible ver los ***Maps*** como colecciones de ***claves***, de ***valores*** o de parejas ***clave/valor***. A continuación se muestran los métodos de la interfaz ***Map*** (comando > ***javap java.util.Map***):

```
Compiled from Map.java
public interface java.util.Map {
public abstract void clear();
public abstract boolean containsKey(java.lang.Object);
public abstract boolean containsValue(java.lang.Object);
public abstract java.util.Set entrySet();
public abstract boolean equals(java.lang.Object);
public abstract java.lang.Object get(java.lang.Object);
public abstract int hashCode();
public abstract boolean isEmpty();
public abstract java.util.Set keySet();
public abstract java.lang.Object put(java.lang.Object, java.lang.Object);
public abstract void putAll(java.util.Map);
public abstract java.lang.Object remove(java.lang.Object);
public abstract int size();
public abstract java.util.Collection values();
public static interface java.util.Map.Entry   {      ← Interfaz interna de Map
       public abstract boolean equals(java.lang.Object);
       public abstract java.lang.Object getKey();
       public abstract java.lang.Object getValue();
       public abstract int hashCode();
       public abstract java.lang.Object setValue(java.lang.Object);
}   }
```

La interface ***SortedMap*** añade los siguientes métodos, similares a los de ***SortedSet***:

```
Compiled from SortedMap.java
public interface java.util.SortedMap extends java.util.Map
{
public abstract java.util.Comparator comparator();
public abstract java.lang.Object firstKey();
public abstract java.util.SortedMap headMap(java.lang.Object);
public abstract java.lang.Object lastKey();
public abstract java.util.SortedMap subMap(java.lang.Object,
java.lang.Object);
public abstract java.util.SortedMap tailMap(java.lang.Object);
}
```

La clase ***HashMap*** implementa la interface ***Map*** y está basada en una hash table, mientras que ***TreeMap*** implementa ***SortedMap*** y está basada en un árbol binario.

Declaración:
Map<String,String> tabla = new TreeMap<String,String>();
Métodos: put(añade llave-valor), remove(objeto), clear() (vacío, borra todo), get(objeto), contains(llave) contiene llave?

Métodos	Función que realizan
Vector(), Vector(int), Vector(int, int)	Constructores que crean un vector vacío, un vector de la capacidad indicada y un vector de la capacidad e incremento indicados
void addElement(Object obj)	Añade un objeto al final
boolean removeElement(Object obj)	Elimina el primer objeto que encuentra como su argumento y desplaza los restantes. Si no lo encuentra devuelve false
void removeAllElements()	Elimina todos los elementos
Object clone()	Devuelve una copia del vector
void copyInto(Object anArray[])	Copia un vector en un array
void trimToSize()	Ajusta el tamaño a los elementos que tiene
void setSize(int newSize)	Establece un nuevo tamaño
int capacity()	Devuelve el tamaño (capacidad) del vector
int size()	Devuelve el número de elementos
boolean isEmpty()	Devuelve true si no tiene elementos
Enumeration elements()	Devuelve una Enumeración con los elementos
boolean contains(Object elem)	Indica si contiene o no un objeto
int indexOf(Object elem, int index)	Devuelve la posición de la primera vez que aparece un objeto a partir de una posición dada
int lastIndexOf(Object elem, int index)	Devuelve la posición de la última vez que aparece un objeto a partir de una posición, hacia atrás
Object elementAt(int index)	Devuelve el objeto en una determinada posición
Object firstElement()	Devuelve el primer elemento
Object lastElement()	Devuelve el último elemento
void setElementAt(Object obj, int index)	Cambia el elemento que está en una determinada posición
void removeElementAt(int index)	Elimina el elemento que está en una determinada posición
void insertElementAt(Object obj, int index)	Inserta un elemento por delante de una determinada posición

Tabla 4.6. Métodos de la clase Vector.

Las clases ***Vector*** y ***Hashtable y*** la interface ***Enumeration*** están presentes en lenguaje desde la primera versión. Después la ***Java Collections Framework***, fue introducida en la versión JDK 1.2.

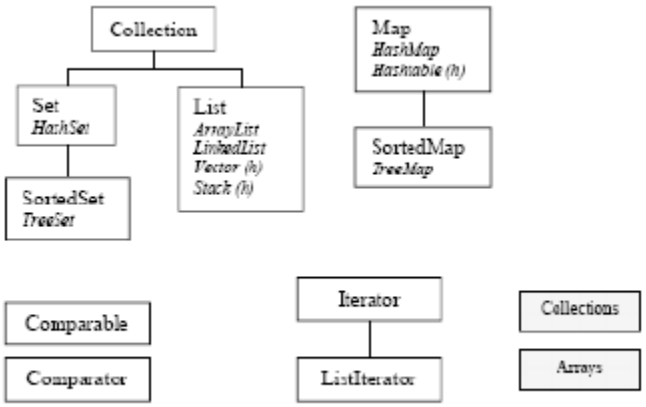
Vector: La clase ***java.util.Vector*** deriva de ***Object***, implementa ***Cloneable*** (para poder sacar copias con el método ***clone()***), y ***Serializable*** (para poder ser convertida en cadena de caracteres).

Interfaz Enumeration
La interfaz ***java.util.Enumeration*** define métodos útiles para recorrer una colección de objetos.
Puede haber distintas clases que implementen esta interfaz y todas tendrán un comportamiento similar.
La interfaz ***Enumeration*** declara dos métodos:
1. ***public boolean hasMoreElements()***. Indica si hay más elementos en la colección o si se ha llegado ya al final.
2. ***public Object nextElement()***. Devuelve el siguiente objeto de la colección. Lanza una *NoSuchElementException* si se llama y ya no hay más elementos.
Ejemplo: Para imprimir los elementos de un vector ***vec*** se pueden utilizar las siguientes sentencias:

```
for (Enumeration e = vec.elements(); e.hasMoreElements(); ) {
System.out.println(e.nextElement());
}
```

El Collections Framework de Java 1.2

En la versión 1.2 del JDK se introdujo el ***Java Framework Collections*** o "estructura de colecciones de Java" (en adelante JCF). Se trata de un conjunto de clases e interfaces que mejoran notablemente las capacidades del lenguaje respecto a estructuras de datos. La Figura muestra la jerarquía de interfaces de la ***Java Collection Framework*** (JCF). En letra cursiva se indican las clases que implementan las correspondientes interfaces. Por ejemplo, hay dos clases que implementan la interface ***Map***: ***HashMap*** y ***Hashtable***. Las clases "históricas", es decir, clases que existían antes de la versión JDK 1.2, se denotan en la Figura 4.1 con la letra "h" entre paréntesis. Aunque dichas clases se han mantenido por motivos de compatibilidad, sus métodos no siguen las reglas del diseño general del JCF; en la medida de lo posible se recomienda utilizar las nuevas clases.



Interfaces de la Collection Framework.

Clases históricas: Son las clases ***Vector*** y ***Hashtable*** presentes desde las primeras versiones de ***Java***. En las versiones actuales, implementan respectivamente las interfaces ***List*** y ***Map***, aunque conservan también los métodos anteriores.

Algoritmos: La clase ***Collections*** dispone de métodos ***static*** para ordenar, desordenar, invertir orden, realizar búsquedas, llenar, copiar, hallar el mínimo y hallar el máximo. La clase ***Arrays*** dispone de métodos ***static*** para manipulación de arreglos.