La clase `ArrayList`

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1

La clase `ArrayList`

Características importantes de la clase ArrayList son:

- Puede aumentar su capacidad interna tanto como se requiera.
- Cuando se agregan más elementos, simplemente crea el espacio necesario para ellos.
- Mantiene su propia cuenta privada de la cantidad de elementos almacenados, que se obtiene mediante size.
- Mantiene el orden de los elementos que se agregan (secuencial).
- El método add almacena cada nuevo elemento al final de la lista.
- Posteriormente se pueden recuperar en el mismo orden.

Tipos primitivos y la clase **`ArrayList`**

Los elementos de los **`ArrayList`** son objetos (**`Object`**).Los enteros, por ejemplo,no son objetos. Por tanto, un **`int`** no se puede insertar en un **`ArrayList`**. ¿Qué hacemos entonces?. Utilizar la clase envoltorio **`Integer`**.

```
import java.util.ArrayList;
        public class DemoEnvoltorio {
                public static void main(String[] args) {
                      ArrayList<Integer> lista = new ArrayList<Integer>();
                      lista.add((Integer) 42);      // También vale lista.add(42);
                                                    // Antes era: lista.add(new Integer(42));
                                                    // Ahora deprecated
                      Integer n;
                      n = lista.get(0);
                      int miEntero = n;
                      System.out.println(n);
                }
        }
```

Otras clases envoltorio son **`Byte`**,**`Float`**,**`Double`**,**`Long`** y **`Short`**.

La interfaz `Iterator`

Los iteradores permiten recorrer colecciones sin preocuparse de la implementación subyacente.

`hasNext()`:¿hay un elemento siguiente?

`next()`:devuelve el siguiente elemento de la colección.

`remove()`:elimina el último elemento devuelto por el iterador.

```
public void imprime(Collection col) {
    Iterator it = col.iterator();
    while (it.hasNext())
        System.out.println(it.next());
}
```

Iteradores sobre distintos tipos de colecciones

```
import java.util.*;
public class PruebaColecciones {
      public static void prueba(Collection<String> c) {
      String[] lista = { "uno", "dos", "tres", "cuatro", "tres" };
      for (int i = 0; i < lista.length; i++)
            c.add(lista[i]);
      Iterator<String> it = c.iterator();
      while (it.hasNext())
            System.out.println(it.next());
      System.out.println("------------------------");
      }
      public static void main(String args[]) {
      Collection<String> c;
      c = new ArrayList<String>();
      prueba(c);
      c = new LinkedList<String>();
      prueba(c);
      c = new HashSet<String>();
      prueba(c);
      c = new TreeSet<String>();
prueba(c);
}
}
```

Más formas para recorrer colecciones

Bucle for each:

Se incorpora en la versión 5 de Java. Esta estructura nos permite recorrer una Colección o un array de elementos de una forma sencilla. Evita el uso de Iteradores o de un bucle for normal.

```
//Declaro un ArrayListArrayList<String>pelis = new ArrayList<String>();
//Estructura for each
for (String nombre: pelis)
System.out.println(nombre);
```

El resultado es más legible. Sin embargo en ocasiones los iteradores aportan cosas interesantes que los bucles for each no pueden abordar. Vamos a borrar todos los objetos de una lista con un nombre concreto. La operación parece tan sencilla como hacer lo siguiente:

```
for (String nombre:pelis){
   if (nombre.equals("Casablanca")){
    pelis.remove("Casablanca");
   }
}
```

El código no funciona y lanza una excepción.

La interfaz Iterator dispone de un método adicional que permite eliminar objetos de una lista mientras la recorremos. El método remove.

La clase `LinkedList`

Se puede usar para crear una estructura de datos Cola o Pila. Añadir datos a la Cola se hace con addLast(). Eliminar con removeFirst(). Obtener el primer elemento con getFirst().

En cuanto a ArrayList y LinkedList vemos que las inserciones en posiciones iniciales o finales son, por regla general, más rápidas con LinkedList. Igualmente pasaría con las eliminaciones o borrados. Aunque observamos que las búsquedas o las inserciones en posiciones intermedias de una lista con gran número de elementos es más rápida en el caso de ArrayList. Esto es debido a que esta accede a la posición de manera directa.

Para finalizar esta comparación, vemos que el método add por defecto sin posición es más rápido que insertar en posiciones iniciales o intermedias en ambas implementaciones.