La clase ArrayList

Un inconveniente que tienen los arrays es que cuando se crean se establece su tamaño, es decir, el número de elementos que contendrá el array. Dicho tamaño se puede establecer en tiempo de ejecución pero una vez establecido ya no se puede modificar

Pero en muchas circunstancias no es posible determinar de antemano el número de elementos de información que necesitaremos almacenar en el array o simplemente este número no es constante.

En tal caso una solución que se puede adoptar es sobredimensionar el array con el despilfarro de memoria que ello puede suponer. Una solución mejor sería utilizar un objeto que permita almacenar más o menos referencias a objetos según se necesite, es decir, un objeto que pueda contener un número variable de referencias a objetos sin desperdiciar recursos de memoria. Los objetos de la clase ArrayList tienen estas características

La clase ArrayList está definida en el paquete de clases java.util. Un objeto ArrayList es un array que se expande a sí mismo automáticamente.

- La clase ArrayList define métodos **add** para añadir al final de la lista o insertar en una posición una nueva referencia a objeto
- El método **get**(indice) devuelve la referencia a objeto del ArrayList que ocupa la posición indicada por el índice
- El método **remove**(indice) elimina del ArrayList la referencia a objeto que ocupa la posición indicada por el índice y devuelve dicho elemento. Cuando esto ocurre, automáticamente los elementos siguientes se desplazan hacia la izquierda una posición
- El método **set**(i,e): reemplaza la referencia a objeto que ocupa la posición indicada por el índice i por la referencia a objeto e
- El método **clear**() elimina todas las referencias a objeto del ArrayList
- El método **isEmpty**() devuelve *true* cuando el ArrayList no contiene ninguna referencia a objeto
- El método **size**() devuelve el número de referencias a objetos que contiene el ArrayList
- El método **toArray**() devuelve un array que contiene los elementos del ArrayList (un Array es más eficiente que un ArrayList...)
- El método **indexOf**(refObjeto) devuelve el número de posición que ocupa en el ArrayList la referencia a objeto pasado como parámetro
- El método **contains**(refObj) devuelve true si el ArrayList contiene la referencia a objeto pasada como parámetro

Aunque los métodos anteriores son los más utilizados, la clase ArrayList define más métodos (consulta las API)

Un ArrayList es un contenedor de referencias a objetos de la clase Object. Esto significa que un ArrayList puede contener referencias a objetos de cualquier tipo.

Por lo tanto, cuando se llame a un método de la clase ArrayList que devuelve un elemento (como por ejemplo el método get(indice)) lo que devuelve es una referencia a objeto del tipo Object.

El programador debe tener en cuenta el tipo de los objetos cuyas referencias han sido almacenadas en el ArrayList y una vez recuperadas, habrá de hacerse una conversión explícita. Por ejemplo:

```
ArrayList ciudades=new ArrayList();
ciudades.add(new String());// podemos añadir al ArrayList cualquier tipo de objeto en vez de String
String unaCiudad=(String) ciudades.get(i);//tenemos que saber qué tipo de objetos pusimos en la posición i del ArrayList para hacer el cast adecuado
```

TIPOS GENÉRICOS O TIPOS PARAMETRIZADOS

Cuando en un ArrayList o en cualquier otro tipo de colección con la que vamos a trabajar las referencias a objetos que se van a almacenar van a representar SIEMPRE objetos del mismo tipo, al definir el ArrayList o la colección podemos indicar esta circunstancia utilizando un tipo genérico de ArrayList o colección .

Para definir tipos genéricos, tras el nombre de la clase colección se escribe el tipo de los objetos que va a contener entre los símbolos <>. Por ejemplo, supongamos que hemos definido una clase denominada *Persona*. Si queremos definir un ArrayList que va a contener sólo referencias a objetos del tipo *Persona* escribiremos la sentencia:

```
ArrayList<Persona> lasPersonas=new ArrayList< >();
```

Utilizar tipos genéricos tiene la ventaja de que al recuperar objetos del ArrayList no se recupera una referencia a objeto tipo *Object* sino una referencia a objeto del tipo genérico indicado al crear por ejemplo un ArrayList y por lo tanto no se necesita hacer cast de la referencia a objeto devuelta por el método get del ArrayList.

Copiar la información de un ArrayList a un array

Si no se sabe cuántos elementos tendrá en principio una serie de objetos no se puede utilizar un array para representar la serie puesto que al crearlo se tiene que indicar el número de elementos que contiene.

En ese caso se puede utilizar un ArrayList para representar la información pero en cualquier momento podremos construir un array con la misma información del ArrayList utilizando las siguientes instrucciones:

```
ArrayList intermedio=new ArrayList();
// a continuación tendríamos sentencias que añadirían elementos a intermedio
……
//cuando intermedio ya contiene la información que queremos "copiar" a un array:

Persona[] elArray=new Persona[intermedio.size()];
elArray= intermedio.toArray(elArray);
```

Que es equivalente a:

```
ArrayList intermedio=new ArrayList();
// a continuación tendríamos sentencias que añadirían elementos a intermedio
……
//cuando intermedio ya contiene la información que queremos "copiar" a un array:
Persona[] elArray= intermedio.toArray(new Persona[intermedio.size()]);
```

OBSERVA QUE PRIMERO SE CREA UN OBJETO ARRAY DEL TAMAÑO ADECUADO Y DESPUÉS SE DEFINE UNA REFERENCIA A DICHO OBJETO

Los métodos Arrays.asList y Arrays.copyOf

La clase *Arrays* define el método estático *asList* que espera como parámetro un array y devuelve un objeto List que representa la misma información que dicho array. Esto puede sernos útil cuando tenemos la información en un array pero necesitamos invocar algún método propio de colecciones.
El método estático *copyOf* de la clase *Arrays* espera dos parámetros, el primero es un array y el segundo es un número entero. El método devuelve un nuevo array que contiene las n primeras referencias a objeto del array que recibe como primer parámetro, siendo n el valor de su segundo parámetro.
Se pueden utilizar dichos método para obtener un array "desordenado" que represente la misma información que el array original:

```
String[] archivosDeImagen=baraja.archivosDeImagen();
List sorteo = Arrays.asList(archivosDeImagen);
//"revuelve" las referencias a objetos de la lista sorteo
Collections.shuffle(sorteo);
//copyOf genera un nuevo array que contiene las n primeras referencias
//a objetos del array que recibe como primer parámetro
return Arrays.copyOf(sorteo.toArray(new String[sorteo.size()]), n);
```

Método addAll

Puesto que la clase ArrayList implementa Collection se puede utilizar el método *addAll* para añadir referencias a objetos. *addAll* permite añadir a una colección varias referencias a objetos. Hay dos métodos *addAll*, uno estático que espera como primer parámetro una colección y como segundo y siguientes parámetros uno o varios objetos cuyas referencias se añadirán a la colección. Por otra parte,
el método *addAll* no estático espera una colección cuyas referencias a objetos se añadirán a la colección que lo invoca

```
ArrayList<String> uno=new ArrayList<>();
HashSet<String> otro=new HashSet<>();
//a la colección otro le añadimos las dos primeras referencias a objeto de la colección uno
Collections.addAll(otro, uno.get(0),uno.get(1));
// a la colección uno le añadimos todas las referencias a objeto que contiene la colección otro
uno.addAll(otro);
```

Y si las referencias a objetos que queremos añadir al ArrayList son las de un array:

```
String[] arrayPalabras={"Hola","Saludo","Despedida"};
ArrayList<String> uno=new ArrayList<>();
uno.addAll(Arrays.asList(arrayPalabras));
```