Arraylist

Arrays de objetos

Los arrays no solo pueden almacenar tipos primitivos. También pueden almacenar objetos. El funcionamiento es similar en los dos casos. Conviene destacar si no se especifica ningún valor los elementos de un array de objetos se inicializan automáticamente a null.

La clase Arrays

Proporciona métodos static para la manipulación de arrays. Estos métodos incluyen:

- sort: para ordenar un array en forma ascendente
- binarySearch: para buscar en un array
- equals: para comparar arrays
- fill: para rellenar valores en un array.
- toString: para visualizar el array

Introducción a las colecciones y la clase ArrayList

La API de Java provee de varias estructuras de datos predefinidas conocidas como colecciones, que se utilizan para almacenar grupos de objetos relacionados.

Estas clases proporcionan métodos eficientes que organizan, almacenan y obtiene datos sin necesidad de saber cómo se almacenan éstos.

Gracias a esto se reduce el tiempo de desarrollo de aplicaciones. Los arrays tradicionales no cambian de manera automática su tamaño en tiempo de ejecución para dar cabida a elementos adicionales.

La clase de colección ArrayList proporciona una solución a este problema, puede cambiar su tamaño de forma dinámica para dar cabida a más elementos. La T en Java representa un tipo genérico por convención. Al declarar el nuevo objeto ArrayList, hay que reemplazarlo por el tipo de elementos que deseamos que contenga. Es similar a especificar el tipo cuando declaramos un array, pero en el caso del ArrayList no es posible utilizar tipos primitivos. La clase ArrayList debe importarse del paquete java.util.

Declaración de ArrayList

Un objeto de la clase ArrayList se declara de la siguiente manera:

ArrayList<clase> nombre_lista;

Ejemplo:

ArrayList<String> lista;

Creación de ArrayList

Tenemos varios constructores:

- **public ArrayList():** Construye una lista vacía con una capacidad inicial de 10.
- **public ArrayList(int initialCapacity) :** Construye una lista vacía con la capacidad inicial indicada como parámetro.

nombre_lista = new ArrayList<T>();

Puede unirse la declaración y la creación:

ArrayList<clase> nombre_lista = new ArrayList<T>();

Ejemplo: ArrayList lista = new ArrayList<T>();

La clase ArrayList no permite el uso de tipos primitivos, en el caso de que necesitemos almacenar int, long, byte … podemos utilizar los Wrappers.

Al final del capítulo hablamos de ellos brevemente. A continuación se muestran algunos de los métodos comunes de la clase ArrayList.

	Métodos Descripción	Métodos Descripción
Añadir	boolean add(Object o)	Agrega un elemento al final de la lista del objeto ArrayList.
Eliminar	void clear()	Elimina todos los elementos del objeto ArrayList.
Contiene	boolean contains(Object elem)	Devuelve true si el objeto ArrayList contiene el elemento especificado, en caso contrario devuelve false.
Consultar	Object get(int index)	Devuelve el elemento en el índice especificado.
Consultar	int indexOf(Object elem)	Devuelve el índice de la primera ocurrencia del elemento especificado en el objeto ArrayList. Si no está devuelve -1.

	Métodos	Descripción
Eliminar	boolean remove(int i)	Sobrecargado. Elimina la primera ocurrencia del valor especificado o del elemento en el subíndice especificado.
Eliminar	boolean remove(Object o)	Sobrecargado. Este método borra el objeto que se pasa como parámetro si éste está presente dentro de la lista.
Tamaño	int size()	Devuelve el número de elementos almacenados en el objeto ArrayList.
¿Está Vacia?	boolean isEmpty()	Este método comprueba si hay elementos en el ArrayList.

For each

A partir de Java 5 existe otra forma de recorrer una lista que es mucho más cómoda y compacta , el uso de bucles foreach.

Un bucle foreach se parece mucho a un bucle for con la diferencia de que no hace falta una variable i de inicialización :

```
for ( TipoARecorrer nombreVariableTemporal : nombreDeLaColección ) {

                    Instrucciones

}
```

Algunos puntos importantes de for-each:

- Comienza con la palabra clave **for** al igual que un bucle *for* normal.
- En lugar de declarar e inicializar una variable contador del bucle, declara una variable que es del mismo tipo que del array, seguido de dos puntos y seguido del nombre del array.
- En el cuerpo del bucle, puede usar la variable del bucle que creó en lugar de usar un elemento indexado del array.
- Se usa comúnmente para iterar sobre un array o una clase de colecciones (por ejemplo, **ArrayList**)

```
public void listarTodosLosNombres () {
        for (String nombre: listaDeNombres) {
        System.out.println (nombre);            }
  }
```

Wrappers

Los Wrappers (envoltorios) son clases diseñadas para ser un complemento de los tipos primitivos. Los tipos primitivos son los únicos elementos de Java que no son objetos. Esto tiene algunas ventajas desde el punto de vista de la eficiencia, pero algunos inconvenientes desde el punto de vista de la funcionalidad. Las clases Wrapper también proporcionan métodos para realizar otras tareas con lo tipos primitivos, tales como conversión con cadenas de caracteres en uno y otro sentido.

Wrappers

Existe una clase Wrapper para cada uno de los tipos primitivos numéricos que extienden de la clase Number, esto es, existen las clases Byte, Short, Integer, Long, Character, Float y Double (obsérvese que los nombres empiezan por mayúscula, siguiendo la nomenclatura típica de Java).

Primitive type	Wrapper class	Primitive type	Wrapper class
boolean	Boolean	int	Integer
byte	Byte	long	Long
char	Character	short	Short
float	Float	double	Double