Listas en Java

ÍNDICE

1. Introducción

Las listas en Java nos van a permitir crear colecciones dinámicas de datos, todos del mismo tipo, en los que puede haber repetidos. La ventaja es que no hace falta definir, en tiempo de compilación, el número de elementos que va a tener la lista sino que puede crecer y decrecer en tiempo de ejecución (a diferencia de los arrays).

Una peculiaridad de Java es que las listas no funcionan para trabajar con datos que sean de tipo primitivo (int, float, boolean, etcétera), sino con objetos. Es por ello, que para trabajar con dichos valores necesitamos utilizar sus clases *wrapper*, es decir, `Integer`, `Float`, `Boolean`, etcétera.

Nosotros, para el trabajo con listas nos vamos a centrar en el uso de `ArrayList`. A continuación se indica la manera de trabajar con lista y una serie de métodos pero hay muchos más… ¡investiga!.

2. Declaración e inicialización

El trabajo con listas en Java no se limita a la interfaz `ArrayList`. Sin embargo, esta es la más utilizada y, por ello, nos centraremos en ella por ahora.

Para declarar listas en Java podemos utilizar la siguiente sintaxis:

```
List<tipoDeDato> nombreLista = new ArrayList<tipoDeDato>();
```

Para poder trabajar con listas, en Java, es necesario importar el paquete `java.util` que es donde se encuentra la interfaz List (entre otras). Esto te va a permitir hacer programas más complejos y fáciles de mantener en comparación con el uso de arrays.

Por ejemplo, si queremos almacenar la estatura de un grupo de personas, podríamos implementar el siguiente código:

```
List<Float> estaturas = new ArrayList<Float>();
```

3. Insertar elementos

3.1 Inserción por defecto

Insertar o agregar elementos a las listas en Java de tipo ArrayList es muy sencillo. Para ello podemos utilizar el método `add()` de la siguiente forma:

```
// Inserción directa
estaturas.add(1.79f);

// Inserción mediante variable de tipo primitivo
float estatura1 = 1.85f;
estaturas.add(estatura1);

// Inserción mediante wrapper de variable de tipo primitivo
Float estatura2 = 1.9f;
estaturas.add(estatura2);
```

Cada elemento que agreguemos **se insertará al final de la lista**.

3.2 Inserción en posición

Puede que nos interese insertar un elemento en una determinada posición de la lista, para ello, haremos uso de una sobreescritura del propio método `add()` indicando la posición a insertar (además del elemento a insertar).

En el siguiente ejemplo, insertamos una nueva estatura al comienzo de la lista (posición **0** ya que es *zero-base*).

```
estaturas.add(0, 1.84f);
```

4. Recuperar elementos

Si queremos obtener el elemento de una determinado posición, podemos hacer uso del método `get()` indicando el índice de la lista:

```
Float segundoElemento = estaturas.get(1); // Base 0
System.out.println(segundoElemento);
```

5. Reemplazar elementos

Si necesitamos reemplazar/sustituir un elemento por otro dentro de una lista, podemos hacer uso del método `set()` de la siguiente forma. El primer parámetro es el índice a modificar y el segundo parámetro es el nuevo valor:

```
estaturas.set(2, 2.05f);
```

6. Número de elementos

Una de las cosas habituales es saber el número de elementos que tiene nuestra lista. Para ello, haremos uso del método size() de la propia lista.

```
int numElementos = estaturas.size();
```

7. Actividades

Actividad 1. Debes implementar, de todas las formas que se te ocurran, la siguiente funcionalidad: pide al usuario 10 números enteros y muestra por pantalla la suma de todos ellos con el siguiente formato: num1 + num2 + … + num10 = suma.

Por ejemplo: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55

Actividad 2. Implementa la funcionalidad anterior pero sin saber el número de enteros que introducirá el usuario. Sabremos que el usuario no quiere introducir más números enteros cuando introduzca un cero (0).