

# El uso de Listas en Java

Por **Antony García González** - enero 14, 2015



El uso de listas en Java es una forma útil de almacenar y manipular grandes volúmenes de datos, tal como haríamos en una matriz o arreglo, pero con una serie de ventajas que hacen de este tipo de variables las preferidas para el procesamiento de grandes cantidades de información.

Las listas en Java son variables que permiten almacenar grandes cantidades de datos. Son similares a los Array o a las Matrices. Sin embargo, conversando con mi amigo [Aristides Villarreal](#) (miembro de [Netbeans Dream Team](#) y uno de los mejores programadores en Java en la actualidad) me dice que las nuevas tendencias en la programación van orientadas hacia el uso de Listas para el manejo de grandes volúmenes de datos. De hecho Java 8, dentro de sus nuevas herramientas de programación (específicamente las expresiones Lambda), posee muchas funcionalidades interesantes implementando listas, lo cuales nos permitirá lograr muchas cosas fabulosas de una manera mucho más rápida y fácil de lo que haríamos antes (sin embargo, esto es contenido para otro post).

Esto me ha llevado a buscar formas de agregar las Listas en Java a mis herramientas de programación. En realidad no soy muy bueno utilizando este tipo de variables ya que apenas estoy empezando a utilizarlas, pero siento la necesidad de compartir lo poco que sé ya que con ello ayudo a otras personas a la vez que fabrico mi propio cuaderno de apuntes para futuras auto consultas.

Algo que me llama mucho la atención de las listas en Java es el hecho de que no es necesario establecer un tamaño específico para la variable, a diferencia de las tradicionales matrices o arreglos. Las listas son sumamente versátiles y mucho más fáciles de manejar que otros tipos de variables de agrupación de datos.

Entonces... ¿cómo declaramos una lista? Pues seguimos la siguiente estructura:

```
1 List ejemploLista = new ArrayList();
```

Este tipo de lista puede almacenar cualquier tipo de dato, pero este enfoque ya ha quedado obsoleto. Se prefiere que se designe el tipo de dato que se va a almacenar. Entonces para declarar una lista donde guardaremos datos tipo **String**, hacemos lo siguiente:

```
1 List<String> ejemploLista = new ArrayList<String>();
```

Con nuestra lista creada podemos empezar a introducir datos en ella.

Supongamos que queremos agregar los siguientes nombres: **Juan, Pedro, José, María, Sofía**.

Hacemos lo siguiente:

```
1 List<String> ejemploLista = new ArrayList<String>();
2     ejemploLista.add("Juan");
3     ejemploLista.add("Pedro");
4     ejemploLista.add("José");
5     ejemploLista.add("María");
6     ejemploLista.add("Sofía");
```

Es posible además agregar el índice en el que queremos agregar dicho elemento. Podemos obtener la cantidad de elementos que posee la lista:

```
1 ejemploLista.size();
```

Para consultar la lista se utiliza:

```
1 ejemploLista.get(0);
```

Donde 0 es el índice en el que se encuentra la información que queremos. En este caso, el índice 0 vendría siendo **Pedro**.

Si queremos eliminar determinado elemento:

```
1 ejemploLista.remove(0);
```

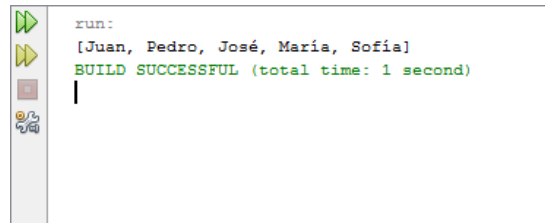
Donde nuevamente el 0 representa el índice que queremos eliminar. Otra forma de eliminar un registro es por su nombre:

```
1 ejemploLista.remove("Juan");
```

Si deseamos imprimir en consola los elementos de la lista:

```
1 System.out.println(ejemploLista);
```

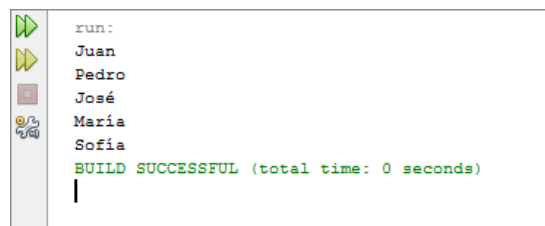
Esto nos producirá la siguiente salida:



```
run:
[Juan, Pedro, José, María, Sofía]
BUILD SUCCESSFUL (total time: 1 second)
```

También podemos imprimir todos los elementos de la lista de forma individual con la ayuda de un ciclo for.

```
1 for (int i = 0; i <= ejemploLista.size() - 1; i++) {
2     System.out.println(ejemploLista.get(i));
3 }
```



```
run:
Juan
Pedro
José
María
Sofía
BUILD SUCCESSFUL (total time: 0 seconds)
```

También podríamos usar un iterador para recorrer la lista e imprimir todos sus valores:

```
1 Iterator i = ejemploLista.iterator();
2 while(i.hasNext())
3 {
4     System.out.println(i.next());
5 }
```

Para eliminar todos los elementos de la lista usamos:

```
1 ejemploLista.clear();
```

Si deseamos saber si nuestra lista contiene algún elemento utilizamos:

```
1 ejemploLista.isEmpty();
```

Esto nos devolverá un **true** o un **false**. En caso de que contenga algún elemento podemos verificar si entre esos elementos tenemos alguno en específico. Por ejemplo si queremos saber si en nuestra lista está escrito el nombre de José, utilizamos:

```
1 ejemploLista.contains("José");
```

Esto también nos devolverá un **true** o un **false**. Y si por alguna razón queremos modificar algún dato de nuestra lista, por ejemplo el índice 1 que contiene el nombre de Pedro, utilizamos el siguiente método:

```
1 ejemploLista.set(1, "Félix");
```

Habremos cambiado el nombre en el índice 1 (Pedro) por el nuevo nombre (Félix).

Si queremos extraer una lista que contenga los nombres entre un índice y otro podemos utilizar:

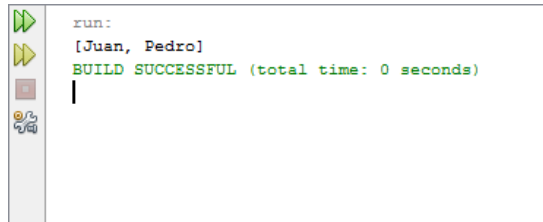
```
1 ejemploLista.subList(0, 2)
```

Veamos el siguiente ejemplo:

```
List<String> ejemploLista = new ArrayList<String>();
ejemploLista.add("Juan");
ejemploLista.add("Pedro");
ejemploLista.add("José");
ejemploLista.add("María");
ejemploLista.add("Sofía");

System.out.println(ejemploLista.subList(0, 2));
```

El resultado de esto es:



```
run:
[Juan, Pedro]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Nótese que se transforman los elementos desde el índice inicial (0) hasta el elemento anterior al índice final (2), despreciando el elemento que se encuentra en el índice final.

Estos son los usos elementales que le podemos dar a las listas. En próximos aportes estaré explicando cómo crear listas de más de una dimensión y el uso de expresiones Lambda, en Java 8.



[report this ad](#)

Por lo pronto espero que esta información sea de utilidad para ustedes.

Saludos.

**Antony García González**

Ingeniero Electromecánico, graduado de la Universidad Tecnológica de Panamá. Miembro fundador de Panama Hitek.  
Entusiasta de la electrónica y la programación.



