Introducción a List

¡Comencemos!

¿Qué aprenderás?

- Aplicar el uso de List para resolver problemas cotidianos dentro del mundo de la programación.
- Crear distintas implementaciones de tipo List para ocupar métodos como agregar, eliminar y recorrer datos.

Introducción

Una lista es una colección ordenada, a veces llamada secuencia, que contiene elementos en su interior. Las listas pueden contener elementos duplicados.

La plataforma Java contiene dos implementaciones de Lista de propósito general: ArrayList, que suele ser la implementación con mejor rendimiento; y LinkedList, que ofrece un mejor rendimiento en determinadas circunstancias. Ambas fueron definidas previamente en los gráficos anteriores.

Además de algunas operaciones heredadas desde Collection, la interfaz List también incluye operaciones de uso propio. Algunas de estas son:

- Acceso posicional (get, set, add, addAll, remove y size)
- Búsqueda (indexOf y lastIndexOf)
- Iteración (listIterator)

Más adelante veremos cómo se usa cada una de estas operaciones, pero primero partiremos creando un ArrayList para entender cómo funcionan y desde dónde se implementan.

¡Vamos con todo!

Creando un ArrayList

Un arrayList es una estructura de datos que puede estirarse para acomodar elementos adicionales dentro de sí mismos y reducirse a un tamaño más pequeño cuando se eliminan elementos. Es una implementación de matriz-redimensionable de la interfaz List. Además, permite a todos los elementos, incluyendo el valor nulo. Se puede ver en la imagen la implementación de este método y que implementa desde List.

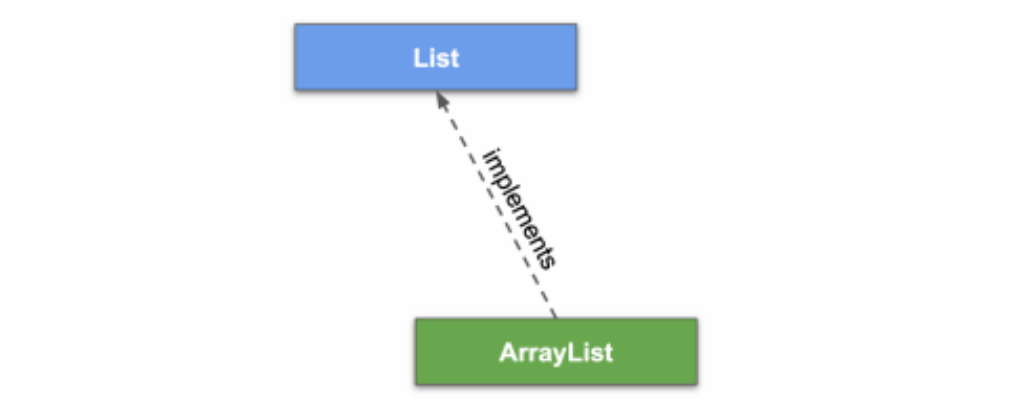

Imagen 1. Implementación ArrayList en List
Fuente: Desafío Latam

Ahora, vayamos a un ejemplo de cómo crear un ArrayList en Java. Para esto, partiremos definiendo una List del tipo String, cuyo nombre será list e instanciamos como una ArrayList.

```
ArrayList<String> list = new ArrayList<>();
list.add("Java");
list.add("Scala");
list.add("Kotlin");
System.out.println(list); //[Java, Scala, Kotlin]
```

Es importante destacar que los elementos de la lista tienen una posición y se les llama índices. Se puede acceder a los valores pasando el índice como parámetro del método get. En el ejemplo podemos ver como al llamar la posición número cero esta llama al elemento agregado “Java”.

```
System.out.println(list.get(0)); //Java
```

Los índices van de cero hasta (n-1), donde `n` es la cantidad de elementos de la colección. Si una lista contiene 5 elementos, el primer elemento estará en la posición cero, y el último en la cuarta posición. En caso de que el índice sea mayor o igual a la cantidad de elementos se obtendrá una excepción `ArrayIndexOutOfBoundsException`.

Ejercicio guiado: Métodos de acceso posicional

Como se mencionó previamente, tenemos distintos métodos para manejar el acceso posicional de elementos. La colección List permite agregar, quitar, obtener y establecer operaciones basadas en posiciones numéricas de elementos en la lista. A continuación, veremos algunos de los métodos más usados sobre estas listas con un ejemplo de las ciudades de Chile:

- **Paso 1:** Para crear una ArrayList<>() hay que importar su implementación desde "`util.java.ArrayList`" en la parte superior de la clase y luego instanciarla como se muestra en la siguiente imagen.

```
import java.util.ArrayList

      ArrayList<String> ciudades = new ArrayList<>();
```

- **Paso 2:** Para incorporar elementos a la lista, debemos ocupar el método void `add()`, el cual ocupa el nombre de la lista para ir incorporando elementos al ArrayList previamente definido.

```
      ArrayList<String> ciudades = new ArrayList<>();
      ciudades.add("Santiago");
      ciudades.add("Iquique");
      ciudades.add("Arica");
      ciudades.add("Concepción");
      ciudades.add("La Serena");
      ciudades.add("Puerto Montt");
      System.out.print(ciudades);

      ---------------------------
      Impresión en pantalla:

      [Santiago, Iquique, Arica, Concepción, La Serena, Puerto Montt]
```

- **Paso 3:** Para incorporar elementos desde una colección específica a la lista de ciudades que tenemos, podemos ocupar el método `addAll()`. Este método agrega todos los elementos de la colección a la lista. El primer elemento se inserta en el índice dado. Si ya hay un elemento en esa posición, ese elemento y los otros elementos posteriores (si los hay) se desplazan hacia la derecha al aumentar su índice.

```
        //Incorporación de ciudades al ArrayList
        ArrayList<String> ciudades = new ArrayList<>();
        ciudades.add("Santiago");
        ciudades.add("Iquique");
        ciudades.add("Arica");
        ciudades.add("Concepción");
        ciudades.add("La Serena");
        ciudades.add("Puerto Montt");

        //Incorporación de más ciudades desde una colección distinta
llamada otrasCiudades

        ArrayList<String> otrasCiudades = new ArrayList();
        otrasCiudades.add("Rancagua");
        otrasCiudades.add("Punta Arenas");
        ciudades.addAll(otrasCiudades);
        System.out.print(ciudades);

        ---------------------------
        Impresión en pantalla:

        [Santiago, Iquique, Arica, Concepción, La Serena, Puerto Montt,
Rancagua, Punta Arenas]
```

- **Paso 4:** Para obtener un elemento en base a su posición, podemos hacer uso del get y buscar este índice al interior de la lista. Si queremos saber de qué elemento se está hablando, usaremos el método `get()`. Este método devuelve el elemento en el índice especificado partiendo. Cabe recordar que la lista de los índices parte desde el número cero.

```
System.out.print(list.get(0));
System.out.print(list.get(4));
---------------------------
Impresión en pantalla:

[Santiago]
[Puerto Montt]
```

- **Paso 5:** Para remover un elemento específico desde la ArrayList previamente hecha, podemos utilizar el método `remove()`. Este método elimina un elemento del índice especificado. Desplaza los elementos posteriores (si los hay) a la izquierda y disminuye sus índices en 1.

```
//Para eliminar a Puerto Montt por ejemplo, se puede remover usando su posición que la número 4 al interior de la lista
ciudades.remove(4);
System.out.print(ciudades);
---------------------------
Impresión en pantalla:

[Santiago, Iquique, Arica, Concepción, La Serena, Rancagua, Punta Arenas]

//Para comprobar si se elimina el elemento, podemos usar el mismo método usando su nombre y este nos arrojará false si no lo encontró o true si lo elimino

ciudades.remove("Puerto Montt");
```

- **Paso 6:** Para modificar un elemento al interior de la lista en base a su índice correspondiente, podemos usar el método `set()`. Este método reemplaza el elemento en un índice dado con un nuevo elemento. Esta función devuelve el elemento que acaba de ser reemplazado por un nuevo elemento.

```
      ciudades.set(2, "Talca");
      System.out.print(ciudades);
      --------------------------
      Impresión en pantalla:

      [Santiago, Iquique, Talca, Concepción, La Serena, Rancagua, Punta
Arenas]
```

- **Paso 7:** Para encontrar la cantidad exacta de elementos que contiene la lista, podemos utilizar el método `size()`. Este método devuelve la cantidad de elementos al interior de la lista.

```
      System.out.print(ciudades.size());
      --------------------------
      Impresión en pantalla:

      [7]
```

Ejercicio guiado: Métodos de búsqueda

Por su parte, List también proporciona métodos para buscar elementos y los devuelve en base a su posición numérica. Los siguientes dos métodos son compatibles con List para esta operación:

- **Paso 8:** Para buscar en base a contenido de un elemento, podemos usar el método `indexOf()`. Este método devuelve la posición del elemento dado, si el elemento aún está en la lista o (-1) si el elemento no está presente en la lista.

```
System.out.print(ciudades.indexOf("Puerto Montt"));
System.out.print(ciudades.indexOf("Santiago"));
--------------------------
Impresión en pantalla:

[-1] //Fuera de la lista
[0]  //Devuelve su posición que es 0
```

- **Paso 9:** Para buscar en base al último contenido que tuvo un elemento, podemos usar el método `lastIndexOf()`. Este método devuelve la posición del elemento dado, si el elemento aún está en la lista o (-1) si el elemento no está presente en la lista.

```
System.out.print(ciudades.lastIndexOf("Puerto Montt"));
System.out.print(ciudades.lastIndexOf("Santiago"));
--------------------------
Impresión en pantalla:

[-1] //Fuera de la lista
[0]  //Devuelve su posición que es 0
```

Ejercicio guiado: Métodos de iteración

Como se mencionó también al inicio de este capítulo, contamos con un método para iterar en una lista mediante `listIterator()`. A continuación, veremos cómo ocupamos este método siguiendo con el ejemplo de las ciudades de Chile:

- **Paso 10:** Para crear una ArrayList<>() hay que importar su implementación desde "util.java.ArrayList" en la parte superior de la clase y luego instanciarla como se muestra en la siguiente imagen:

```
import java.util.ListIterator;

	//Devuelve la lista ordenada tal cual llega
	List<String> ciudades = new ArrayList<>();
	ciudades.add("Santiago");
	ciudades.add("Iquique");
	ciudades.add("Arica");
	ciudades.add("Concepción");
	ciudades.add("La Serena");
	ciudades.add("Puerto Montt");

	ListIterator<String> ciudadesIterator = ciudades.listIterator();
	System.out.print(ciudadesIterator.hasNext());
	---------------------------
	Impresión en pantalla:
	true
	//El true es el equivalente a la lista recorrida y ordenada
	//[Santiago, Iquique, Arica, Concepción, La Serena, Puerto Montt]



	//Devuelve la lista ordenada al revés
	List<String> ciudades = new ArrayList<>();
	ciudades.add("Santiago");
	ciudades.add("Iquique");
	ciudades.add("Arica");
	ciudades.add("Concepción");
	ciudades.add("La Serena");
	ciudades.add("Puerto Montt");

	ListIterator ciudadesIterator = ciudades.listIterator();
	System.out.print(ciudadesIterator.hasPrevious());
```

```
        ---------------------------
        Impresión en pantalla:
        [false]
        //El false es equivalente a la lista recorrida pero ordenada de
manera inversa
        //[Puerto Montt,La Serena, Concepción,  Arica,  Iquique,
Santiago]
```