Programación orientada a objetos

Introducción a colecciones y List

{desafío} latam_

Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje Java para resolver un problema de baja complejidad

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Distinguir las distintas interfaces de colecciones para implementarlas en problemas de código.*
- *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.*

{desafío} latam_

¿Qué son las colecciones?

{desafío} latam_

/* Collection & List */

Colecciones

Es un grupo de objetos individuales representados como una sola unidad.

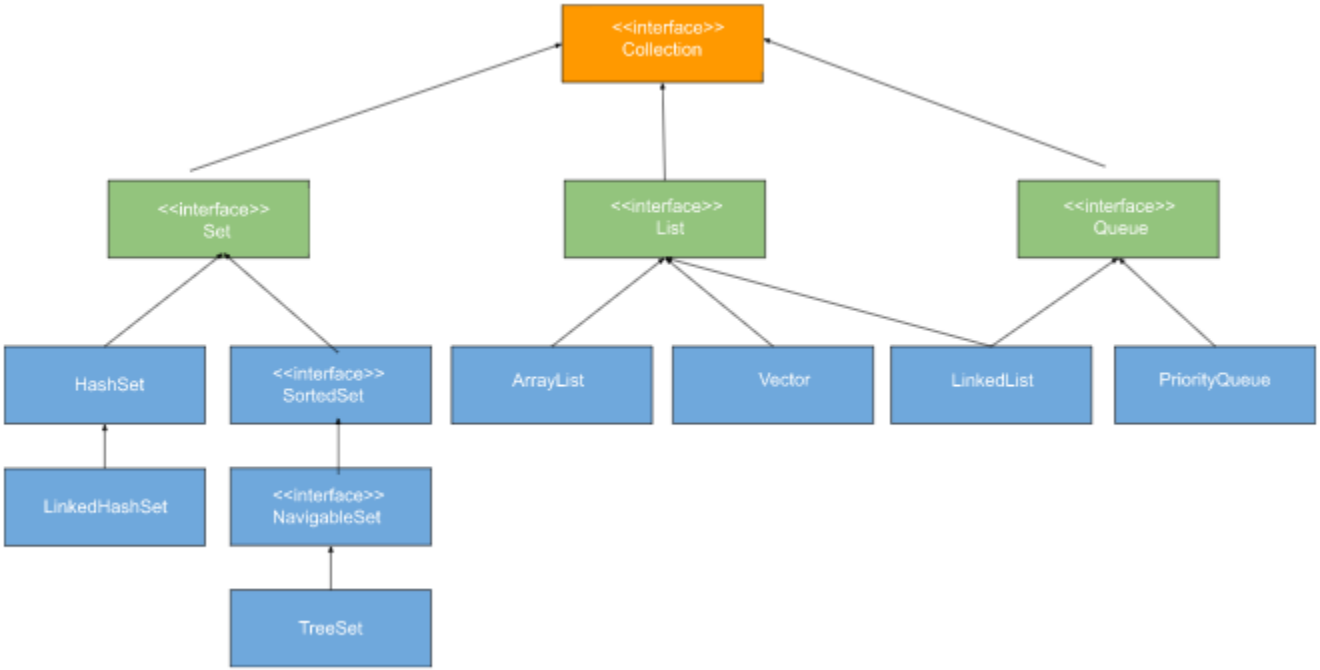
Nos proporciona Frameworks de colecciones mediante clases e interfaces para representar un grupo de objetos como una sola unidad.

Principales interfaces "raíz" de las clases de recopilación de Java:

- La interfaz Collection (`java.util.Collection`)
- La interfaz Map (`java.util.Map`) son las

Jerarquía de Collection Framework

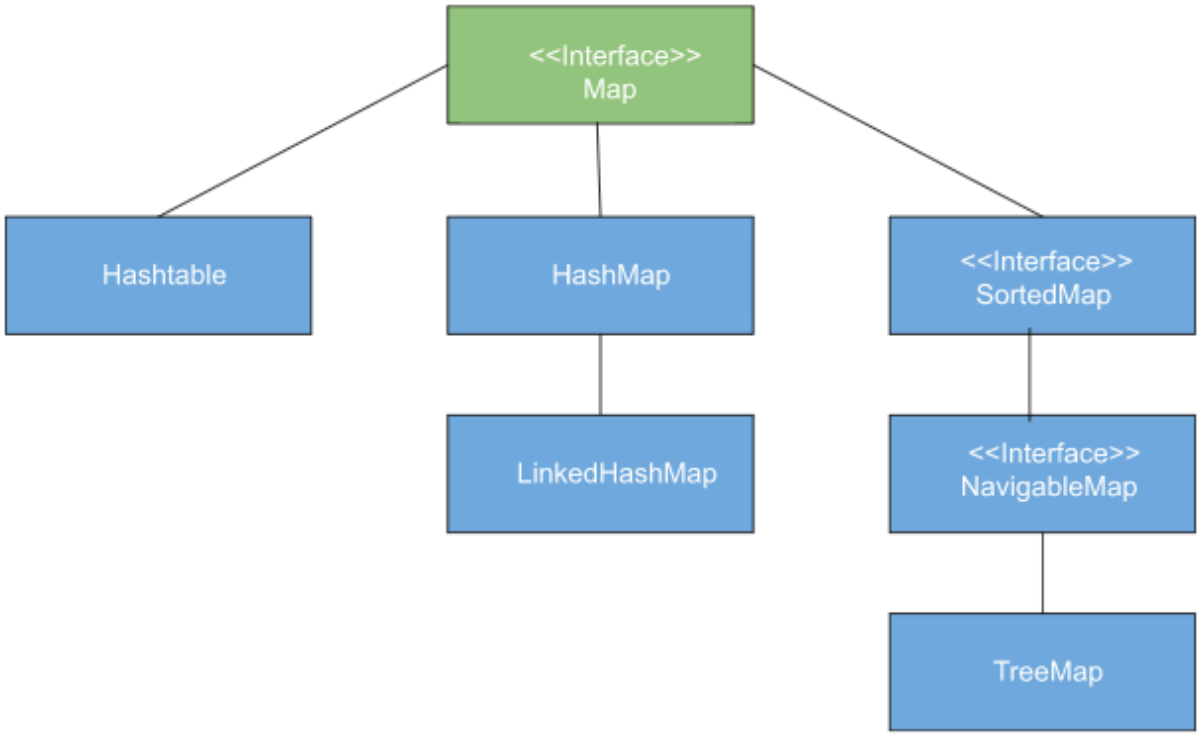
Elementos que contiene la interfaz Collections en Java



{desafío}
latam_

Jerarquía de Collection Framework

Elementos que contiene la interfaz Map en Java



List

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

- ArrayList, que suele ser la implementación con mejor rendimiento
- LinkedList, que ofrece un mejor rendimiento en determinadas circunstancias.

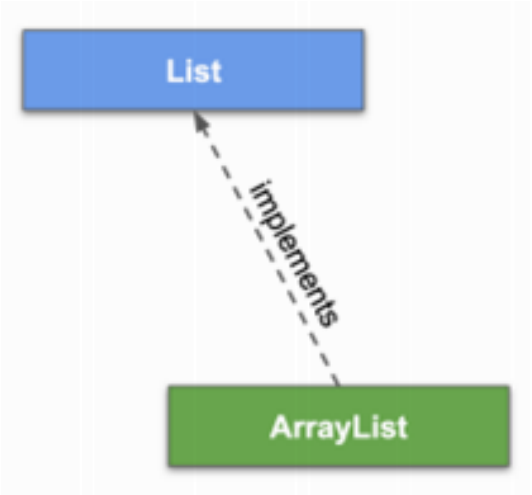
Además de algunas operaciones heredadas desde Collection, la interfaz List también incluye operaciones de uso propio. Algunas de estas son:

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

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.



Crear un ArrayList

Comenzaremos 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]
```

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.

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

Ejercicio guiado

Métodos de acceso posicional

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

```
import java.util.ArrayList
      ArrayList<String> ciudades = new ArrayList<>();
```

Métodos de acceso posicional

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]
```

Métodos de acceso posicional

Paso 3

Para incorporar elementos desde una colección específica a la lista de ciudades que tenemos, podemos ocupar el método addAll().

```
//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]
```

Métodos de acceso posicional

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().

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

    [Santiago]
    [Puerto Montt]
```

Métodos de acceso posicional

Paso 5

Para remover un elemento específico desde la ArrayList previamente hecha, podemos utilizar el método remove()

```
//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");
```

Métodos de acceso posicional

Paso 6

Para modificar un elemento al interior de la lista en base a su índice correspondiente, podemos usar el método set().

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

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

Métodos de acceso posicional

Paso 7

Para encontrar la cantidad exacta de elementos que contiene la lista, podemos utilizar el método size().

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

[7]
```

Métodos de búsqueda

Paso 8

Para buscar en base al contenido de un elemento, podemos usar el método indexOf().

```
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
```

Métodos de búsqueda

Paso 9

Para buscar en base al último contenido que tuvo un elemento, podemos usar el método lastIndexOf().

```
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
```

Métodos de iteración

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.

¿Cuál es el código resultante?

Según lo aprendido, ¿para qué usamos las colecciones?

A partir de lo estudiado, ¿cómo se define una List?

Próxima sesión...

- *Aplicar el uso de Set para resolver problemas cotidianos dentro del mundo de la programación.*
- *Aplicar el uso de Queue para resolver problemas cotidianos dentro del mundo de la programación.*
- *Aplicar el uso de Map para resolver problemas cotidianos dentro del mundo de la programación.*

{desafío}
latam_

Academia de
talentos digitales