# Estructura de datos - Generics - Arrays- List

## Estructura de datos

Las estructuras de datos en Java son conjuntos de valores, variables y tipos de datos organizados para soportar operaciones específicas como agregar y eliminar elementos, buscar elementos, ordenar elementos, etc.

En Java, existen varias estructuras de datos built-in (integradas) que se pueden utilizar para almacenar y organizar información de manera eficiente.

Algunas de las estructuras de datos más comunes en Java incluyen:

- Arrays: Un arreglo es una estructura de datos que permite almacenar una secuencia de elementos del mismo tipo.
- Listas: Una lista es una estructura de datos que permite almacenar una secuencia de elementos de cualquier tipo.
- Pilas: Una pila es una estructura de datos que permite almacenar elementos de manera last-in, first-out (LIFO).
- Colas: Una cola es una estructura de datos que permite almacenar elementos de manera first-in, first-out (FIFO).
- Conjuntos: Un conjunto es una estructura de datos que permite almacenar elementos únicos sin importar el orden.
- Mapas: Un mapa es una estructura de datos que permite asociar un valor a una clave y acceder a él mediante la clave.

Java proporciona implementaciones integradas de muchas de estas estructuras de datos a través de las colecciones de la biblioteca estándar de Java, como ArrayList, LinkedList, Stack, Queue, TreeMap, etc.

Es importante seleccionar la estructura de datos adecuada para una tarea específica, ya que cada estructura de datos tiene fortalezas y debilidades diferentes en términos de rendimiento, capacidad y funcionalidad.

## Generics

En Java, los tipos genéricos permiten escribir código que puede ser utilizado con diferentes tipos de datos, sin necesidad de especificarlos en el momento de escribirlo. Esto mejora la reutilización del código y reduce los errores de tiempo de compilación.

Los tipos genéricos se declaran con angle brackets <> y pueden ser utilizados en clases, interfaces y métodos.

Los tipos genéricos en Java son una herramienta que permite a los desarrolladores escribir código más flexible y reutilizable.

Fueron introducidos en Java 5 y permiten a los desarrolladores crear clases, interfaces y métodos que pueden trabajar con cualquier tipo de datos, sin tener que especificar exactamente qué tipo de datos se utilizará. Esto significa que pueden ser utilizados para crear colecciones de objetos, estructuras de datos y algoritmos, entre otros, que sean genéricos y sean útiles para muchos tipos diferentes de datos.

Antes de la introducción de los tipos genéricos, los desarrolladores tenían que escribir una versión específica de una clase o método para cada tipo de datos que deseaban trabajar. Esto significaba que tenían que escribir mucho código repetitivo y era fácil cometer errores. Con los tipos genéricos, pueden escribir una sola versión de una clase o método que pueda trabajar con cualquier tipo de datos.

Aquí hay algunos ejemplos de usos comunes de genéricos en Java:

- Contenedores de datos: Los genéricos son útiles para crear contenedores de datos, como listas y conjuntos, que puedan almacenar diferentes tipos de datos. Por ejemplo, se puede crear una clase genérica MyList que almacene una lista de objetos de cualquier tipo:

```
public class MyList<T> {
        private List<T> list = new ArrayList<>();

        public void add(T item) {
                list.add(item);
        }

        public T get(int index) {
                return list.get(index);
        }
}
```

- Métodos genéricos: Los genéricos también son útiles para escribir métodos genéricos que puedan funcionar con diferentes tipos de datos. Por ejemplo, se puede escribir un método genérico que encuentre el máximo de dos elementos:

```
public static <T extends Comparable<T>> T max(T a, T b) {
        return a.compareTo(b) > 0 ? a : b;
}
```

- Interfaces genéricas: Los genéricos también se pueden usar en interfaces para hacerlas más genéricas y reutilizables. Por ejemplo, se puede escribir una interfaz genérica MyInterface que declare un método genérico:

```
public interface MyInterface<T> {
        T doSomething(T t);
}
```

- Clases anidadas genéricas: Los genéricos también se pueden usar en clases anidadas para hacerlas más genéricas y reutilizables. Por ejemplo, se puede escribir una clase anidada genérica MyNestedClass dentro de una clase normal:

```
public class MyClass {
        public static class MyNestedClass<T> {
                private T t;

                public MyNestedClass(T t) {
                        this.t = t;
                }
                public T getValue() {
                        return t;
                }
        }
}
```

Estos son solo algunos ejemplos de cómo los genéricos pueden ser útiles en Java.

Los genéricos se pueden usar en muchos otros contextos y son una parte esencial de la programación en Java.

## Arrays

Un array en Java es una estructura de datos que contiene una colección ordenada de elementos del mismo tipo. Cada elemento en el array se identifica con un índice numérico, comenzando desde 0.

Los arrays se pueden crear de dos maneras:

- Especificando el tamaño del array al momento de la creación.

```
public class Main {
        public static void main(String[] args) {
                int[] numbers = new int[5];
                numbers[0] = 3;
                numbers[1] = 1;
                numbers[2] = 4;
                numbers[3] = 2;
                numbers[4] = 5;
                System.out.println("Números en el array: ");
```

```
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Número en la posición " + i + ": " + numbers[i]);
        }
    }
}
```

El resultado de ejecutar este código sería:

```
Números en el array:
Número en la posición 0: 3
Número en la posición 1: 1
Número en la posición 2: 4
Número en la posición 3: 2
Número en la posición 4: 5
```

- Especificando los elementos del array al momento de la creación.

```
int[] numbers = {1, 2, 3, 4, 5};
for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);
}
```

Una vez creado, los elementos de un array pueden ser accedidos y modificados mediante su índice.

Java también proporciona una serie de métodos integrados para trabajar con arrays, como sort (para ordenar los elementos de un array), copyOf (para crear una copia de un array) y binarySearch (para realizar búsquedas binarias en un array ordenado).

Es importante tener en cuenta que los arrays en Java son de tamaño fijo, lo que significa que una vez creado, su tamaño no puede ser cambiado. Para superar esta limitación, Java proporciona otras estructuras de datos dinámicas, como ArrayList.

## List

La clase List en Java es una interfaz que representa una estructura de datos de lista.

Una lista es una colección ordenada de elementos que se pueden acceder por un índice.

La clase List proporciona una serie de métodos para insertar, eliminar, buscar y manipular elementos en la lista.

Algunas implementaciones comunes de la interfaz List incluyen:

- ArrayList: Una lista dinámica que ajusta automáticamente su tamaño según sea necesario.

- LinkedList: Una lista enlazada que permite la inserción y eliminación de elementos de manera eficiente.

- Vector: Una lista que proporciona una sincronización adicional para hacerla segura en entornos multithread.

## ArrayList

ArrayList es una clase de la biblioteca de Java Collections que representa una lista de objetos. A diferencia de los arrays tradicionales en Java, ArrayList es dinámico en términos de tamaño, lo que significa que puedes agregar y eliminar elementos después de que la lista se haya creado.

Aquí están algunas de las características y métodos clave de ArrayList:

- Tamaño dinámico: El tamaño de un ArrayList puede cambiar dinámicamente mientras se usa.

- Tipos genéricos: ArrayList puede contener elementos de cualquier tipo, y puedes especificar el tipo de elementos que deseas al crear un ArrayList.

- Métodos: ArrayList proporciona métodos útiles para manipular sus elementos, como add (para agregar un elemento), remove (para eliminar un elemento), get (para obtener un elemento en una posición específica) y size (para obtener el número de elementos en la lista).

- Iteración: Puedes iterar a través de los elementos de un ArrayList usando un bucle for o un iterador.

- Rendimiento: ArrayList es más lento que los arrays tradicionales para acceder a un elemento en una posición específica, pero es más rápido para agregar y eliminar elementos en el medio de la lista.

Aquí hay un ejemplo de código que demuestra cómo usar ArrayList en Java:

```
import java.util.ArrayList;

public class ArrayListExample {
        public static void main(String[] args) {
                // Creación de una lista de enteros
                ArrayList<Integer> numbers = new ArrayList<>();

                 // Agregar elementos a la lista
                numbers.add(10);
                numbers.add(20);
                numbers.add(30);
                numbers.add(40);
```

```
                // Obtener el tamaño de la lista
                System.out.println("Tamaño de la lista: " + numbers.size());

                // Acceder a un elemento en una posición específica
                System.out.println("Tercer elemento: " + numbers.get(2));

                // Eliminar un elemento
                numbers.remove(3);

                // Recorrer la lista usando un bucle for
                System.out.println("Elementos en la lista:");
                for (int i = 0; i < numbers.size(); i++) {
                        System.out.println(numbers.get(i));
                }
        }
}
```

## LinkedList

LinkedList es una clase de la biblioteca de Java Collections que representa una lista enlazada de objetos.

A diferencia de ArrayList, que almacena sus elementos en una matriz, LinkedList almacena sus elementos en una serie de nodos, donde cada nodo contiene un elemento y un enlace a otro nodo.

Aquí están algunas de las características y métodos clave de LinkedList:

- Lista enlazada: Los elementos de un LinkedList están conectados entre sí mediante enlaces, en lugar de estar almacenados en una matriz.
- Tipos genéricos: LinkedList puede contener elementos de cualquier tipo, y puedes especificar el tipo de elementos que deseas al crear un LinkedList.
- Métodos: LinkedList proporciona métodos útiles para manipular sus elementos, como addFirst (para agregar un elemento al principio de la lista), addLast (para agregar un elemento al final de la lista), getFirst (para obtener el primer elemento en la lista) y getLast (para obtener el último elemento en la lista).
- Iteración: Puedes iterar a través de los elementos de un LinkedList usando un bucle for o un iterador.
- Rendimiento: LinkedList es más rápido que ArrayList para agregar y eliminar elementos en el medio de la lista, pero más lento para acceder a un elemento en una posición específica.

Aquí hay un ejemplo de código que demuestra cómo usar LinkedList:

```
import java.util.LinkedList;

public class LinkedListExample {
	public static void main(String[] args) {
		// Creación de una lista de cadenas
		LinkedList<String> colors = new LinkedList<>();

		// Agregar elementos a la lista
		colors.add("Red");
		colors.add("Green");
		colors.add("Blue");
		colors.add("Yellow");
		colors.add("Purple");

		// Obtener el tamaño de la lista
		System.out.println("Tamaño de la lista: " + colors.size());

		// Agregar un elemento al principio de la lista
		colors.addFirst("Orange");

		// Agregar un elemento al final de la lista
		colors.addLast("Pink");

		// Recorrer la lista usando un bucle for
		System.out.println("Elementos en la lista:");
		for (String color : colors) {
			System.out.println(color);
		}
	}
}
```

## Vector

Vector es una clase de la biblioteca de Java Collections que representa una estructura de datos de tamaño dinámico que permite agregar, eliminar y acceder a sus elementos de manera eficiente. Vector es similar a ArrayList, pero Vector es sincronizado, lo que significa que es seguro para ser usado por varios hilos en forma simultánea.

Aquí están algunas de las características y métodos clave de Vector:

- Tamaño dinámico: El tamaño de un Vector se puede ajustar dinámicamente para adaptarse a las necesidades de su aplicación.

- Tipos genéricos: Vector puede contener elementos de cualquier tipo, y puedes especificar el tipo de elementos que deseas al crear un Vector.

- Sincronización: Vector es sincronizado, lo que significa que es seguro para ser usado por varios hilos en forma simultánea.

- Métodos: Vector proporciona métodos útiles para manipular sus elementos, como addElement (para agregar un elemento a la lista), removeElement (para eliminar un elemento de la lista), elementAt (para obtener el elemento en una posición específica) y size (para obtener el número de elementos en la lista).

- Iteración: Puedes iterar a través de los elementos de un Vector usando un bucle for o un iterador.

Si necesitas una estructura de datos que sea segura para ser usada por varios hilos en forma simultánea, Vector es una buena opción.

Aunque Vector es una clase antigua en Java y se recomienda usar ArrayList en su lugar, debido a su sincronización.

Lectura requerida:
Sznajdleder, Pablo Augusto: El gran libro de Java a Fondo Editorial Marcombo.
Ceballos, Francisco Javier: Java 2. Curso de Programación, Editorial Ra-Ma

Lectura ampliatoria:
https://www.oracle.com/ar/java/