# Argentina Programa

# Clase 21: Estructura de datos Parte 2 - Map - Set - Colas - Pilas

# Agenda

Familiarizarse con los conceptos básicos relacionados a Map - Set - Colas - Pilas

- Estructura de datos Parte 2 - Map - Set - Colas - Pilas.
- Práctica Map - Set - Colas - Pilas.

# Map

Map es una interfaz en Java que representa una estructura de datos de asociación clave-valor.

Con un Map, puedes almacenar y recuperar pares de valores clave-valor, donde la clave es un objeto único que identifica el valor asociado.

# Map

Algunos ejemplos de uso de Map incluyen:

- Almacenamiento de preferencias de usuario en una aplicación.
- Almacenamiento de traducciones de texto en una aplicación multilingüe.
- Almacenamiento de información de una base de datos, donde las claves son nombres de columna y los valores son los valores correspondientes.

# Map

Java proporciona varias implementaciones de la interfaz Map, incluyendo:

- HashMap
- TreeMap
- LinkedHashMap.

# Map - HashMap

Es una clase en Java que implementa la interfaz Map y almacena datos en pares clave-valor.

Utiliza una tabla hash para almacenar los elementos, lo que lo hace eficiente para la búsqueda y recuperación de valores basados en sus claves.

# Map - HashMap

Una tabla hash es una estructura de datos que se utiliza para implementar un mapa (también conocido como diccionario) en la que las claves se utilizan para acceder a los valores asociados.

Una tabla hash funciona transformando las claves en índices numéricos que se utilizan para acceder a los valores en una matriz o tabla.

# Map - HashMap

```
public class Example {
   public static void main(String[] args) {
      // Creamos  HashMap
      HashMap<String, Integer> map = new HashMap<>();

      // Agregamos Elementos al HashMap
      map.put("John", 26);
      map.put("Jane", 30);
      map.put("Jim", 35);

      // Obtenemos el valor de una clave específica
      int age = map.get("Jane");
      System.out.println("Jane's age: " + age);

      // Chequeamos si la clave existe en HashMap
      if (map.containsKey("Jim")) {
         System.out.println("Jim is in the map");
      }
   }
}
```

# Map - TreeMap

TreeMap es una clase en Java que implementa la interfaz Map y almacena datos en una estructura de árbol ordenada.

A diferencia del HashMap, que utiliza una tabla hash y no garantiza ningún orden específico para los elementos, TreeMap garantiza que los elementos estén ordenados según su clave.

# Map - TreeMap

```
import java.util.TreeMap;

public class Example {
   public static void main(String[] args) {
       TreeMap<String, Integer> map = new TreeMap<>();

        map.put("John", 26);
       map.put("Jane", 30);
       map.put("Jim", 35);

       int age = map.get("Jane");
       System.out.println("Jane's age: " + age);

       if (map.containsKey("Jim")) {
          System.out.println("Jim is in the map");
       }
        System.out.println("First key: " + map.firstKey());
       System.out.println("Last key: " + map.lastKey());
   }
}
```

# Map - HashMap

Es una clase en Java que implementa la interfaz Map y almacena datos en pares clave-valor.

Utiliza una tabla hash para almacenar los elementos, lo que lo hace eficiente para la búsqueda y recuperación de valores basados en sus claves.

# Map - LinkedHashMap

Es una clase en Java que implementa la interfaz Map y almacena los datos en pares clave-valor como HashMap.

Sin embargo, a diferencia de HashMap, LinkedHashMap mantiene el orden en el que se insertaron los elementos, lo que significa que los elementos se recuperan en el orden en el que se insertaron.

# Map - LinkedHashMap

```
import java.util.LinkedHashMap;

public class Example {
   public static void main(String[] args) {
      LinkedHashMap<String, Integer> map = new LinkedHashMap<>();
      map.put("John", 26);
      map.put("Jane", 30);
      map.put("Jim", 35);
      int age = map.get("Jane");
      System.out.println("Jane's age: " + age);

      if (map.containsKey("Jim")) {
         System.out.println("Jim is in the map");
      }

      for (String key : map.keySet()) {
         System.out.println(key + ": " + map.get(key));
      }
   }
}
```

# Set

Es una estructura de datos que permite almacenar elementos únicos sin repetir. Es similar a una lista, pero no permite elementos repetidos.

Hay varios tipos de objetos Set disponibles en Java, cada uno con sus propias características y funcionalidades, como:

- HashSet
- TreeSet
- LinkedHashSet

# Set - HashSet

Es una clase en Java que implementa la interfaz Set y utiliza una tabla hash para el almacenamiento.

No permite elementos duplicados y no garantiza un orden específico para los elementos.

# Set - HashSet

```
import java.util.HashSet;

public class Example {
   public static void main(String[] args) {
      HashSet<String> set = new HashSet<>();
      set.add("John");
      set.add("Jane");
      set.add("Jim");
      set.add("John");
      System.out.println("Number of elements: " + set.size());
      if (set.contains("Jim")) {
         System.out.println("Jim is in the set");
      }
      set.remove("John");
      for (String element : set) {
         System.out.println(element);
      }
   }
}
```

# Set - TreeSet

Es una clase en Java que implementa la interfaz Set y almacena los elementos en un árbol binario de búsqueda (BST, por sus siglas en inglés).

Garantiza un orden natural para los elementos, lo que significa que los elementos se ordenan en forma ascendente según su orden natural o según un comparador especificado.

# Set - TreeSet

```
import java.util.TreeSet;

public class Example {
   public static void main(String[] args) {
      TreeSet<Integer> set = new TreeSet<>();
      set.add(3);
      set.add(1);
      set.add(2);
      set.add(1);
      System.out.println("Number of elements: " + set.size());
      if (set.contains(2)) {
         System.out.println("2 is in the set");
      }
      set.remove(1);
      for (Integer element : set) {
         System.out.println(element);
      }
   }
}
```

# Set - LinkedHashSet

Es una clase en Java que extiende la clase HashSet e implementa la interfaz Set.

Utiliza una tabla hash y una lista enlazada para almacenar los elementos, permitiendo un acceso rápido a los elementos y preservando el orden en que fueron insertados.

# Set - LinkedHashSet

```
import java.util.LinkedHashSet;

public class Example {
   public static void main(String[] args) {
      LinkedHashSet<String> set = new LinkedHashSet<>();
      set.add("John");
      set.add("Jane");
      set.add("Jim");
      set.add("John");
      System.out.println("Number of elements: " + set.size());
      if (set.contains("Jim")) {
         System.out.println("Jim is in the set");
      }
      set.remove("John");
      for (String element : set) {
         System.out.println(element);
      }
   }
}
```

# Colas - Queue

Una cola es una estructura de datos que permite almacenar elementos de acuerdo a una regla de FIFO (First In, First Out), es decir, el primer elemento en ser agregado es el primero en ser removido.

# Colas - Queue

```
import java.util.LinkedList;
import java.util.Queue;

public class Main {
	public static void main(String[] args) {
		Queue<String> queue = new LinkedList<>();
		queue.offer("A");
		queue.offer("B");
		queue.offer("C");
		queue.offer("D");
		queue.offer("E");
		System.out.println("Elementos en la cola: ");
		while (!queue.isEmpty()) {
			System.out.println(queue.poll());
		}
	}
}
```

# Pilas - Stack

Es una estructura de datos que permite almacenar elementos de acuerdo a una regla de LIFO (Last In, First Out), es decir, el último elemento en ser agregado es el primero en ser removido.

# Pilas - Stack

```
import java.util.Stack;

public class Main {
	public static void main(String[] args) {
		Stack<String> stack = new Stack<>();
		stack.push("A");
		stack.push("B");
		stack.push("C");
		stack.push("D");
		stack.push("E");
		System.out.println("Elementos en la pila: ");
		while (!stack.isEmpty()) {
			System.out.println(stack.pop());
		}
	}
}
```

# CONSULTAS?

Muchas Gracias!