# FORO1

Noé Marcelo Ortega Urías **OU232690**
Oscar Armando Munguía Sotelo **MS232185**
Diego Alberto Muñoz Chavez **MC191763**
Fernando Samuel Quijada Arévalo **QA190088**
David Guillermo Cardona Pérez **CP121738**

P00941 | 02/23

# FORO1 – POO941

## Colecciones JAVA

En Java, **Collection**, **List** y **Map** son interfaces que forman parte del framework de colecciones, que proporciona una manera conveniente y eficiente de manejar grupos de objetos. Aquí te presento definiciones básicas para cada una de estas interfaces:

1. **Collection: Collection** es la interfaz raíz en la jerarquía de colecciones. Representa un grupo de elementos individuales, donde cada elemento puede aparecer una o más veces. No se garantiza ningún orden específico ni unicidad de elementos en una colección. Algunos métodos clave en la interfaz **Collection** incluyen **add**, **remove**, **size**, **isEmpty**, **contains** y **iterator**.

2. **List: List** es una interfaz que extiende **Collection** y define una colección ordenada de elementos donde los elementos pueden repetirse. Cada elemento en una lista tiene una posición indexada, lo que permite acceder a elementos por su posición. Las implementaciones populares de **List** incluyen **ArrayList** (respaldada por un arreglo dinámico), **LinkedList** (respaldada por una lista doblemente enlazada) y **Vector** (similar a **ArrayList**, pero sincronizada).

3. **Map: Map** es una interfaz que no extiende **Collection** y representa una estructura de datos que almacena pares clave-valor únicos. Cada clave está asociada a un valor, y se accede a los valores a través de sus claves. Las claves en un mapa deben ser únicas, y los valores pueden estar duplicados. Algunas implementaciones comunes de **Map** son **HashMap** (basado en tablas hash), **TreeMap** (ordenado según el orden natural de las claves o un comparador personalizado) y **LinkedHashMap** (mantiene el orden de inserción).

Cabe señalar que estas son interfaces y no se pueden instanciar directamente. Para utilizar estas colecciones, debemos trabajar con implementaciones concretas proporcionadas por la biblioteca Java, como **ArrayList**, **HashMap**, etc. Estas implementaciones proporcionan la funcionalidad real y los detalles de almacenamiento y manipulación de datos.

## ¿Cómo se declaran estas colecciones?

En Java, las colecciones no se pueden instanciar directamente, ya que las interfaces como Collection, List y Map son solo plantillas de métodos y definiciones de comportamiento. Para utilizar estas colecciones, necesitas crear

instancias de las clases concretas que implementan estas interfaces. Aquí se muestra cómo declarar e inicializar algunas de estas colecciones:

ArrayList (implementación de List):

```
import java.util.*;
List<String> arrayList = new ArrayList<>();
```

HashMap (implementación de Map):

```
import java.util.*;
Map<String, Integer> hashMap = new HashMap<>();
```

Se pueden reemplazar los tipos de datos (String y Integer en los ejemplos anteriores) con los tipos de datos que se necesitan para los casos de uso específicos. Además, asegúrar de importar la clase apropiada desde el paquete java.util.

Si se quiere usar otras implementaciones, como LinkedList, TreeMap, etc., el proceso es similar. Solo se tiene que reemplazar el nombre de la clase con la implementación que se quiere usar.

Una vez que se ha declarado e inicializado estas colecciones, se puede usar los métodos y propiedades proporcionados por las interfaces y las clases concretas para agregar, eliminar y manipular elementos según las necesidades.

## Metodología de asignación de valores

En los siguientes esquemas se explica la forma con la cual se asignan valores a las diferentes colecciones mencionadas.

**Procedimiento para Asignar Valores en Mapas de Java:**

El proceso de asignación de valores en mapas en Java sigue una serie de etapas bien definidas. A continuación, se detalla el procedimiento para asignar valores a los mapas utilizando la interfaz Map y la implementación HashMap como ejemplo:

Paso 1 - Importar Clases Requeridas:

Primero, es necesario importar la clase relevante del paquete java.util al comienzo del código:

```
import java.util.*;
```

Paso 2 - Establecer el Mapa:

Se debe establecer la instancia del mapa utilizando la implementación deseada, como HashMap:

```
Map<String, Integer> miMapa = new HashMap<>();
```

Paso 3 - Ingresar Datos Clave-Valor:

Los datos clave-valor se ingresan en el mapa mediante el método put. Aquí se proporcionan ejemplos de esta acción:

```
miMapa.put("Clave1", 100);
miMapa.put("Clave2", 200);
miMapa.put("Clave3", 300);
```

Paso 4 - Acceder a los Valores del Mapa: Para acceder a los valores almacenados, se utilizan las claves correspondientes. A modo de ejemplo, para obtener el valor asociado con la clave "Clave2":

```
int valor = miMapa.get("Clave2");
```

Este enfoque se puede aplicar a otras variantes de mapas en Java. Un entendimiento sólido sobre la asignación de valores en mapas es fundamental para utilizar plenamente sus capacidades dentro del contexto de la programación Java.

**Procedimiento para Asignar Valores en Listas de Java:**

El proceso para asignar valores en listas en Java se compone de pasos bien definidos. A continuación, se describe el procedimiento para asignar valores a las listas utilizando la interfaz **List** y la implementación **ArrayList** como ejemplo:

Paso 1 - Importar Clases Necesarias:

Inicia importando la clase requerida del paquete **java.util** al principio del código:

```
import java.util.*;
```

Paso 2 - Preparar la Lista:

Debes preparar la instancia de la lista mediante la implementación deseada, como ArrayList:

```
List<String> miLista = new ArrayList<>();
```

Paso 3 - Agregar Elementos a la Lista:

Se utilizan métodos de la interfaz List, como add, para incorporar elementos a la lista. Aquí se proporciona un ejemplo:

```
miLista.add("Elemento 1");
miLista.add("Elemento 2");
miLista.add("Elemento 3");
```

Paso 4 - Acceder a Elementos en la Lista:

Para acceder a los elementos almacenados, se emplean índices. Por ejemplo, para obtener el primer elemento de la lista:

```
String primerElemento = miLista.get(0);
```

Este proceso es aplicable a otras variantes de listas en Java. Comprender cómo asignar valores en listas es crucial para optimizar la utilización de estas estructuras de datos en la programación Java.

**Procedimiento para Asignar Valores en Colecciones de Java:**

Asignar valores en colecciones en Java sigue una serie de pasos bien establecidos. A continuación, se presenta el procedimiento para asignar valores a las colecciones utilizando la interfaz Collection y la implementación ArrayList como ejemplo:

Paso 1 - Importar Clases Necesarias:

Es necesario importar la clase adecuada del paquete java.util al principio del código:

```
import java.util.*;
```

Paso 2 - Preparar la Colección:

Prepara la instancia de la colección utilizando la implementación deseada, en este caso, ArrayList:

```
Collection<String> miColeccion = new ArrayList<>();
```

Paso 3 - Agregar Elementos a la Colección:

Utiliza los métodos de la interfaz Collection, como add, para agregar elementos a la colección. Aquí tienes un ejemplo:

```
miColeccion.add("Elemento 1");
miColeccion.add("Elemento 2");
miColeccion.add("Elemento 3");
```

Paso 4 - Iterar sobre la Colección:

Para acceder y trabajar con los elementos almacenados, se utiliza un iterador o bucle. Por ejemplo:

```
for (String elemento : miColeccion) {
    // Realizar acciones con cada elemento
}
```

Este proceso es aplicable a otras implementaciones de la interfaz Collection en Java. Entender cómo asignar valores a las colecciones es esencial para manejar datos de manera eficiente en la programación Java.

## Eliminación de valores

Eliminar valores de colecciones en Java implica seguir un proceso específico según el tipo de colección. A continuación, se describe cómo eliminar valores en los tres casos utilizando las interfaces Collection, List y Map.

1. Eliminación de Valores en Collection:

Para eliminar valores en una colección, como ArrayList, se puede utilizar el método remove. Aquí hay un ejemplo:

```
miColeccion.remove("Elemento a eliminar");
```

2. Eliminación de Valores en List:

En una lista, como ArrayList, se puede usar el método remove para eliminar un elemento por índice:

```
miLista.remove(índice);
```

3. Eliminación de Valores en Map:

En un mapa, como HashMap, se puede emplear el método remove para eliminar un par clave-valor mediante la clave:

```
miMapa.remove("Clave a eliminar");
```

**Consideraciones Especiales:**

En el caso de List y Map, la eliminación afecta el orden o la relación clave-valor. En Collection, solo se elimina una instancia del valor si existe.

Si se eliminan elementos en un bucle, se debe tener cuidado para evitar problemas de índice cambiante o ConcurrentModificationException.

Es importante recordar que estas son solo guías generales. La elección de qué elementos eliminar y cómo hacerlo dependerá de los requisitos específicos de tu aplicación y de la lógica que estés implementando.