UNIVERSIDAD AMERICANA

Facultad de ingeniería y arquitectura

POO

Colecciones investigación

Integrantes:

- Franco Xavier Aguilera Ortez
- David Joel Sanchez Acevedo
- Andrea Johanna Duarte Guerrero
- Solieth Valentina Trejos Perez
- Kimberly Maria Zapata Espinoza

Docente: Jose Duran Garcia

Managua, Nicaragua (10/09/25)

Índice

1. ArrayList

El ArrayList en Java es una clase que forma parte del **framework de colecciones** y que implementa la interfaz List.. A diferencia de los arreglos tradicionales, cuyo tamaño es fijo desde el momento en que se crean, los ArrayList pueden **redimensionarse dinámicamente** conforme se agregan o eliminan elementos. Esto los convierte en una

```
public static void main (String[] args) {
```

opción mucho más flexible cuando no se conoce de antemano la cantidad exacta de datos que se almacenarán. Según la documentación oficial de Oracle (Java SE 8), esta clase permite almacenar cualquier tipo de objeto, incluyendo valores null, lo que refuerza su versatilidad en diversos contextos de programación.

El acceso a sus elementos se realiza mediante **índices**, al igual que en un arreglo, lo cual lo hace muy eficiente para lecturas rápidas. Sin embargo, las operaciones de inserción o eliminación en el medio de la lista pueden ser más costosas en comparación con otras estructuras, porque los elementos deben moverse internamente.

- **Características principales**:
    - Permite **elementos duplicados**.
    - Mantiene el **orden de inserción**.
    - Se accede a los elementos mediante índices numéricos.
    - Es más rápido para operaciones de lectura que para inserciones en posiciones intermedias.

- **Casos de uso comunes**:

    - Manejo de listas dinámicas como inventarios de productos, listas de estudiantes, tareas pendientes, etc.

    - Situaciones donde se necesita acceder a los elementos frecuentemente por índice.

1.1- Diferentes implementaciones de List

En algunos casos los usuarios o programadores pueden llegar a creer que existen distintos tipos de listas pero estas en verdad son colecciones distintas que implementan la misma interfaz List:

ArrayList → lista basada en un arreglo dinámico (tipo del cual estamos hablando principalmente en el documento).

LinkedList → lista doblemente enlazada (más eficiente en inserciones/eliminaciones intermedias).

Vector → similar a ArrayList pero **sincronizado** (seguro para hilos).

Stack → hereda de Vector y se usa como pila (LIFO).

1.2- Diferencia entre ArrayLits y Arrays

Características	**Arreglos**	**Listas**
Tamaño	Fijo: Al momento de crearlo. No se puede cambiar sin crear uno nuevo.	Dinámico; puede crecer o reducirse según se agreguen o eliminen elementos.
Tipo de datos	Homogéneo: todos los elementos deben ser del mismo tipo (en Java).	Puede ser homogéneo o heterogéneo (en algunos lenguajes como Python).
Métodos y operaciones	Limitadas; normalmente solo se puede acceder, asignar o recorrer elementos.	Más funcionalidades integradas: agregar, eliminar, buscar, ordenar, etc.
Rendimiento	Acceso a elementos muy rápido (O(1)), pero cambios de tamaño son costosos.	Acceso un poco más lento que un arreglo puro, pero muy flexible para modificar tamaño.

1.3- ¿Por qué no usar Listas en vez de Arreglos?

Rendimiento

- Acceder a un elemento en un arreglo es **muy rápido** (O(1)), porque la memoria está contigua y no hay capa adicional.

- Las listas, especialmente las dinámicas como ArrayList, pueden ser un poco más lentas porque internamente manejan redimensionamiento y métodos extra.

- En aplicaciones donde se hacen **millones de accesos rápidos**, un arreglo puro puede ser más eficiente.

Uso de memoria

- Un arreglo ocupa exactamente lo que necesita para sus elementos.

- Una lista dinámica puede reservar memoria extra para permitir crecimiento, lo que puede ser **menos eficiente** si se tiene memoria limitada.

Simplicidad y control

- Si sabes que tu colección de datos **no va a cambiar de tamaño**, un arreglo es más simple y directo.

- Con listas, hay más abstracción y métodos, lo que a veces no es necesario y puede complicar el código.

- Algunas librerías o funciones esperan específicamente **arreglos**, no listas.

2. Set

En Java, un Set es una colección que se caracteriza por **no permitir elementos duplicados**, ya que su principal objetivo es garantizar la unicidad de los datos. A diferencia de las listas, los Set no mantienen un índice explícito para acceder a sus elementos, lo que significa que no es posible obtener un valor mediante una posición específica, sino a través de iteraciones o búsquedas. Existen varias implementaciones de Set:

- **HashSet**: Almacena los elementos sin un orden específico y es muy eficiente en búsquedas.

  HashSet ofrece operaciones básicas (add, remove, contains) en **tiempo constante promedio** (O(1)) si la función hash distribuye bien los elementos. Pero su iterador, al no tener un orden garantizado, puede depender también de la capacidad interna del HashMap que lo respalda.

```
Set<String> s = new HashSet<>();
```

- **LinkedHashSet**: Mantiene el orden de inserción de los elementos.

  LinkedHashSet es ligeramente más lento que HashSet para insertar porque además de la estructura hash, tiene que mantener el enlace de la lista para preservar el órden de inserción. Sin embargo, la diferencia generalmente es pequeña.

```
LinkedHashSet<String> lh = new LinkedHashSet<>(
```

- **TreeSet**: Ordena automáticamente los elementos de acuerdo con su orden natural o con un comparador definido.

  Un comparador definido hace referencia a establecer un parámetro ya sea: ordenar en base al números de letras de una palabra, los números divisibles entre dos se imprimen de primero, entre otros.

```
TreeSet<String> t = new TreeSet<>();
```

TreeSet tiene costo logarítmico para operaciones básicas (O(log n)) ya que está basado en árbol rojo-negro.

```
import java.util.*;
class Geeks {

    public static void main(String[] args)
    {
        // Declaring object of Set of type String
        Set<String> h = new HashSet<String>();

        // Elements are added using add() method, Custom input elements
        h.add("A");
        h.add("B");
        h.add("C");
        h.add("B");
        h.add("D");
        h.add("E");

        System.out.println("Initial HashSet " + h);

        // Removing custom element using remove() method
        h.remove("B");
        System.out.println("After removing element " + h);
    }
}
```

- **Características principales**:
    - No permite duplicados.
    - Puede o no mantener un orden (según la implementación).
    - Ideal para operaciones de pertenencia (verificar si un elemento está en el conjunto).
- **Casos de uso comunes**:
    - Listados de correos electrónicos únicos.

- Registro de identificadores sin duplicados.

- Conjuntos de datos donde el orden no es tan importante como la unicidad.

2.1-Implementaciones especiales

- **EnumSet**: especializada para enums, muy eficiente, implementada internamente como un bit-vector.
- Oracle menciona otras implementaciones útiles aparte de HashSet, LinkedHashSet y TreeSet
- **CopyOnWriteArraySet**: versión segura para concurrencia, donde al modificar la colección, en lugar de mutar se copia internamente (algo costosa para muchas modificaciones).

Si bien estas no son tan ocupadas como las más “comunes”, también son necesarias en casos especiales como los ya mencio

3. Map (Diccionario)

```java
import java.util.HashMap;
import java.util.Map;

public class Dictionary {
  public static void main(String[] args) {
    System.out.println("Dictionary in Java");
    System.out.println("Using HashMap ");
    Map<String, String> map = new HashMap<>();
    map.put("1", "Letters with alphabetical Order with key A");
    map.put("2", "Letters with alphabetical Order with key B");
    System.out.println(map.get("2"));
  }
```

Un Map en Java es una estructura que almacena elementos en pares **clave → valor**. Cada clave es única dentro del mapa, pero los valores asociados pueden repetirse. Es una estructura muy similar a los **diccionarios** en otros lenguajes de programación.

El Map no forma parte de la interfaz Collection, sino que es una interfaz propia dentro del framework de colecciones. Algunas de sus implementaciones más utilizadas son:

```
Map<String, Student> map = new HashMap();
```

- **HashMap**: La implementación más común, que permite acceso rápido a los elementos pero no garantiza un orden.

- **LinkedHashMap**: Extiende HashMap y además mantiene el **orden de inserción**, lo que lo hace útil cuando se necesita consistencia en la iteración.

- **TreeMap**: Ordena los pares clave → valor según el orden natural de las claves o un comparador.

- **EnumMap**: una implementación especializada para usar **enumeraciones (enum) como claves**. Es muy eficiente en rendimiento y memoria.

- **ConcurrentHashMap**: una variante segura para entornos multihilo (*thread-safe*) que permite operaciones de concurrencia sin necesidad de bloquear toda la estructura, lo que mejora el rendimiento en aplicaciones concurrentes.

3.1- Métodos de la interfaz Map


```
myMap.size(); // Devuelve el numero de elementos del Map
myMap.isEmpty(); // Devuelve true si no hay elementos en el Map y false si si los hay
myMap.put("68274736E", "Paco Soria"); // Añade un elemento al Map
```

3.2- Características principales

- Cada **clave es única**, pero los valores pueden repetirse.
- Permite acceder a los valores a través de las claves.
- Tiene métodos útiles como put(), get(), remove(), containsKey().

3.2- Casos de uso comunes

- Directorios telefónicos (nombre → número).
- Configuraciones de aplicaciones (opción → valor).
- Inventarios (código → cantidad disponible).
- Sesiones de usuario en aplicaciones web (ID de sesión → datos de usuario).
- Cache de datos para mejorar rendimiento en sistemas de alto tráfico.

4. Comparación entre ArrayList, Set y Map

List	Set	Map
La interfaz de lista permite elementos duplicados	Set no permite elemento duplicados	Map no permite elemento duplicados.
La lista mantiene del orden de inserción	No tiene un orden de inserción.	No tiene un orden de inserción
Podemos agregar cualquier número de valores nulos	En el conjunto casi solo un valor nulo	El mapa solo permite una sola clave null como máximo.
Las clases de implementación de List son: ArrayList y LinkedList	Las clases de implementación de Set son: HashSet, LinkedSet, TreeSet	Las clases de implementación de Map son: HashMap, HashTable,TreeMap, CurrentHashMap y LinkedHashMap
Si se necesita acceder a los	Si desea crear una colección	Si desea almacenar los datos

datos de la lista, se hace mediante un índice.	de elementos únicos, se puede usar set.	en forma de par clave/valor, podemos usar Map.

5. Conclusión

Las colecciones en Java como ArrayList, Set y Map ofrecen soluciones eficientes para diferentes necesidades de programación. Mientras que ArrayList se enfoca en listas dinámicas con acceso por índice, Set garantiza unicidad de los elementos y Map permite asociar claves con valores. La elección de cuál utilizar depende de los requisitos de unicidad, orden y forma de acceso a los datos.

Referencias

- Oracle. (2025). *Java Platform, Standard Edition 21 API Specification*. Disponible en: https://docs.oracle.com/en/java/javase/21/docs/api/
- GeeksforGeeks. (2024). *Collections in Java*. Disponible en: https://www.geeksforgeeks.org/collections-in-java-2/
- Baeldung. (2024). *Guide to the Java Collections Framework*. Disponible en: https://www.baeldung.com/java-collections

- TutorialsPoint. (2024). *Java Data Structures*. Disponible en: https://www.tutorialspoint.com/java/java_data_structures.htm