Indice

1. ArrayList

   1.1 Definición

   1.2 Metodos

   1.3 Añadir

   1.4 Eliminar

   1.5 Recorrer

   1.6 Find

   1.7 Ordenar

   1.8 Otro

2. HashMap

1. ArrayList

Volver al indice

1.1 Definición y creación de una colección

ArrayList en Java es una implementación de la interfaz List que utiliza un arreglo dinámico para almacenar elementos. Proporciona métodos para manipular elementos dinámicamente.

Constructores utiles

```
// Constructor vacío
ArrayList<String> A = new ArrayList<>();

// Constructor con capacidad inicial
// ArrayList<T> A = new ArrayList<>(n) --> A = {null, null, ...} hasta n veces
ArrayList<String> A = new ArrayList<>(3); // A = {null, null, null}

// Constructor que acepta otra coleccion
ArrayList<String> B = new ArrayList<>();
ArrayList<String> A = new ArrayList<>(B);  // A = B
```

Volver al indice

1.2 Metodos/Propiedades Generales

Tamaño

```
// size() es un metodo que devuelve el tamaño de la coleccion (sin tener en
// cuenta si sus elementos tambien son colecciones, solo cuenta la "primera"
// dimensión)
ArrayList<String> A = new ArrayList<>();

List<String> B = new List<>();
saludos.add("A"); // B = {"A"}
saludos.add("B"); // B = {"A", "B"}

A.add(B);
int tamanoA = A.size(); // --> tamanoA = 1
int tamanoB = B.size(); // --> tamanoB = 2
```

Acceso por índice/Clave

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}

// String elementoA = A.get(indiceElementoA);
String elementoA = A.get(0); // Devolverá "A"
```

Volver al indice

1.3. Añadir Datos a la Colección

Añadir Elementos desde el Constructor

```
ArrayList<String> A = new ArrayList<>(Arrays.asList("A","B","C")); // A = {"A"
, "B", "C"}
```

Añadir Elementos desde Otras Colecciones

```
ArrayList<String> B = new ArrayList<>();
B.add("A"); // B = {"A"}
B.add("B"); // B = {"A", "B"}

ArrayList<String> A = new ArrayList<>();
A.addAll(B); // A = {"A" , "B"}
```

Añadir Elementos Mediante Código

```
ArrayList<String> A = new ArrayList<>();
A.add("A") // A = {"A"}

ArrayList<ArrayList<String>> B = new ArrayList<>();
B.add(A) // B = {{"A"}}
```

Volver al indice

1.4 Eliminar datos de la Colección

Eliminar Elementos Mediante Código

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}

// Eliminación a través del elemento que contiene
A.remove("B"); // A = {"A", "C"}

// Eliminación a través del indice del elemento
A.remove(1); // A = {"A"}
```

> **Nota importante:** La eliminación a través del elemento **solo elimina el primer elemento** que coincida con el elemento a borrar:
> `A = {"A", "B", "C", "B"} ---> A.remove("B") ---> A = {"A", "C", "B"}`

Eliminar Elementos usando Otra Colección

Haciendo uso del metodo ***removeAll*** eliminamos **todos** los elementos que coincidan con la colección que le introducimos en el metódo.

```
ArrayList<String> B = new ArrayList<>();
B.add("A"); // B = {"A"}
B.add("C"); // B = {"A", "C"}

ArrayList<String> A = new ArrayList<>();
A.addAll(B); // A = {"A", "C"}
A.add("B"); // A = {"A", "C", "B"}

A.removeAll(B); // A = {"B"}
```

Eliminar Elementos usando un Predicado

```
ArrayList<int> A = new ArrayList<>();
A.add(1); // A = {1}
A.add(3); // A = {1, 3}
A.add(5); // A = {1, 3, 5}
A.add(7); // A = {1, 3, 5, 7}

// A.removeIf(element -> condicion) ---> condicion = true ---> elimina
"element"
A.removeIf(element -> element > 5) // A = {1, 3}
```

Volver al indice

1.5 Recorrer la Colección

Recorrer haciendo uso de un for

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}
A.add("D"); // A = {"A", "B", "C", "D"}

for(int i = 0; i < A.size(); i++){
    String elemento = A.get(i); // Siendo elemento el String devuelto de la
colección
    System.out.println(elemento);
}
```

Recorrer haciendo uso de un foreach

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}
A.add("D"); // A = {"A", "B", "C", "D"}

for(String s : A){
    System.out.println(s); // Siendo s el String devuelto de A
}
```

Recorrer haciendo uso de Iterator

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
```

```
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}

Iterator<String> iteratorA = A.iterator(); // Creamos el iterador
while (iteratorA.hasNext()) {
    String elemento = iteratorA.next(); // Siendo elemento el String devuelto
de A
    System.out.println(elemento);
}
```

Recorrer haciendo uso de programacion funcional

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}

A.forEach(elemento -> {
    System.out.println(elemento); // Siendo elemento el String devuelto de A
});
```

> **Nota importante:** si se realiza alguna modificación sobre la lista o alguno de sus elementos mientras se recorre usando un *foreach* o usando *Iterator*, se lanzará la una ***ConcurrentModificationException***.

Volver al indice

1.6 Búsqueda de Elementos

Realizar una búsqueda con un bucle (For/Foreach/Iterator)

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}

for(String elemento : A){
    if(elemento.equals("C")){
        // Se ha encontrado el elemento
    }
}
```

Realizar una búsqueda con Expresiones Lambda (Stream)

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}

boolean encontrado = A.stream().anyMatch(elemento -> elemento.equals("C"));
// Devuelve true si hay alguna coincidencia
```

> **Nota importante:** el tipo ***Stream***, al que se convierte el *ArrayList* permite hacer uso de metodos de programación funcional. En caso de querer retornar un ArrayList una vez se convierte es necesario usar el metodo collect especificando como parametro el tipo a devolver haciendo uso de los metodos estáticos de conversión disponibles en *Collectors*.

Realizar una búsqueda con Api Stream (Filtrado/Collect)

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}
A.add("A"); // A = {"A", "B", "C", "A"}

List<String> B = A.stream().filter(element ->
element.equals("A")).collect(Collectors.toList()); // A = {"A", "A"}
```

Volver al indice

1.7 Ordenación de elementos

Ordenar haciendo uso de Funciones de Collection

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("C"); // A = {"A", "C"}
A.add("B"); // A = {"A", "C", "B"}
A.add("A"); // A = {"A", "C", "B", "A"}

Collections.sort(A); // A = {"A", "A", "B", "C"}
```

Ordenar haciendo uso de Expresiones Lambda

```
ArrayList<Integer> A = new ArrayList<>();
A.add(1); // A = {1}
A.add(3); // A = {1, 3}
```

```
A.add(2); // A = {1, 3, 2}

// ArrayList.sort((current, next) -> current.compareTo(b))
A.sort((a, b) -> a.compareTo(b)); // A = {1, 2, 3}
```

Ordenar haciendo uso de la API Stream

```
ArrayList<String> A = new ArrayList<>();
A.add("A"); // A = {"A"}
A.add("B"); // A = {"A", "B"}
A.add("C"); // A = {"A", "B", "C"}
A.add("A"); // A = {"A", "B", "C", "A"}

List<String> B = A.stream().filter(element ->
element.equals("A")).collect(Collectors.toList()); // A = {"A", "A"}
```

Volver al indice

1.9 Otros Aspectos

- **Uso de Genéricos:** ArrayList en Java utiliza genéricos para asegurar el tipo de elementos almacenados.
- **Capacidad Dinámica:** ArrayList ajusta automáticamente su capacidad interna según sea necesario.
- **Eficiencia en Inserciones/Eliminaciones:** Las inserciones y eliminaciones en el medio de la lista pueden ser lentas, ya que requieren desplazar los elementos.
- **Uso de Métodos Equals y HashCode:** ArrayList utiliza el método equals() para comparar elementos y hashCode() para mejorar el rendimiento en ciertas operaciones.
- **Sincronización:** *ArrayList* no es sincronizado, lo que significa que no es seguro para subprocesos. Si es necesario, puedes usar Collections.synchronizedList(lista) para hacerlo sincronizado.

Volver al indice

HashMap