Arrays y colecciones

Contenido

Arrays (vectores)
- ¿Qué es un array?
- Creación y uso.
- Matrices.
- La clase Arrays.

Colecciones
- Principales colecciones en Java.
- ArrayList
- LinkedList
- Stack
- Queue

¿Qué es un array?

En Java, los arrays o vectores son una estructura de datos que permiten almacenar múltiples elementos de un mismo tipo.

Una vez creado, el tamaño de un array **no** puede cambiar.
Si se necesita un tamaño dinámico, se debe usar una colección como ***ArrayList***.

Los arrays pueden contener *tipos primitivos* o *referencias a objetos*.

Los arrays de tipos primitivos se inicializan automáticamente con valores predeterminados (cero, false, etc.)

Los arrays de tipos de referencia se inicializan con ***null***.

Creación y uso

```java
// DECLARACIÓN
int[] edades;
String[] nombres;
```

```java
// CREACIÓN
edades = new int[5];
nombres = new String[3];
```

```java
// DECLARACIÓN Y CREACIÓN
int[] codigos = new int[2];
double[] precios = new double[4];
```

```java
// DECLARACIÓN E INICIALIZACIÓN
int[] numeros = {9, 12, 18};
String[] palabras = {"hola", "mundo", "cómo", "estas?"};
```

Creación y uso

```java
// INICIALIZAR POR ÍNDICE
codigos[0] = 3366;
codigos[1] = 4455;
codigos[2] = 7755; //ERROR! FUERA DE LÍMITES!
```

```java
// OBTENER VALORES
int ultimoCodigo = codigos[codigos.length - 1];
System.out.println(ultimoCodigo);
```

```java
// USO DEL BLOQUE FOR
for (int index = 0; index < numeros.length; index++) {
    System.out.println(numeros[index]);
}
```

```java
// USO DEL BLOQUE FOR EACH
for (String palabra : palabras) {
    System.out.println(palabra);
}
```

Matrices

```java
// MATRICES
int[][] matriz = new int[2][3];
// 2 filas y 3 columnas
```

```java
// DECLARACIÓN E INICIALIZACIÓN
float[][] otraMatriz = {
    {3.3f, 5.0f, 7.5f},
    {0, 1.1f, 23}
};
```

```java
// USO DEL BLOQUE FOR
for (int i = 0; i < matriz.length; i++) {
    for (int j = 0; j < matriz[i].length; j++) {
        System.out.println( matriz[i][j]);
    }
}
```

```java
// USO DEL BLOQUE FOR EACH
for (float[] fs : otraMatriz) {
    for (float fs2 : fs) {
        System.out.println(fs2);
    }
}
```

La clase Arrays

Java proporciona la clase ***Arrays*** en el paquete *java.util*, que contiene métodos estáticos para manipular arrays.

```java
// ORDENAMIENTO
int[] valores = {5, 3, 8, 1, 2};
Arrays.sort(valores);
// [1, 2, 3, 5, 8]
```

```java
// BÚSQUEDA DE ÍNDICE
int[] otrosValores = {1, 2, 3, 4, 5};
System.out.println(Arrays.binarySearch(otrosValores, key:3));
// key=>3; índice: 2
```

```java
// COPIA DE ARRAYS
int[] copia = Arrays.copyOf(otrosValores, newLength:2);
System.out.println(copia.length);
// [1, 2]

int[] otraCopia = Arrays.copyOfRange(copia, from:0, to:1);
System.out.println(otraCopia.length);
// [1]
```

Ejercitación

Colecciones

Colecciones

Las colecciones en Java son una parte fundamental de la biblioteca de clases del lenguaje y proporcionan una arquitectura para almacenar y manipular grupos de objetos.

La principal ventaja de usar colecciones es que permiten manejar datos de una manera más eficiente y flexible que los arreglos tradicionales.

- ArrayList
- LinkedList
- HashSet
- HashMap
- Stack
- Queue

ArrayList

Características:

- Basada en un array redimensionable.
- Permite acceso rápido por índice.
- Ideal para operaciones de búsqueda.

Uso:

- Muy utilizada debido a su flexibilidad y rendimiento en operaciones de acceso y modificación de elementos.

```
ArrayList<String> lista = new ArrayList<>();
lista.add(e:"hola ");
lista.add(e:"mundo");
lista.add(e:"!");
```

```
for (String cadena : lista) {
    System.out.print(cadena);
}
// hola mundo!
```

LinkedList

Características:

- Basada en una lista doblemente enlazada.
- Ideal para operaciones frecuentes de inserción y eliminación.
- Implementa las interfaces *List*, *Deque* y *Queue*.

Uso:

- Adecuada para implementaciones de *pilas* y *colas*.

HashSet

Características:

- Basada en una tabla hash.
- No permite elementos duplicados.
- No garantiza el orden de los elementos.

Uso:

- Ideal para colecciones donde la unicidad es crucial.

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

set.add(e:"A");
set.add(e:"B");
set.add(e:"C");
set.add(e:"A"); // NO SE AGREGA
```

```
System.out.println(set);
// [A, B, C]
```

HashMap

Características:

- Basada en una tabla hash.
- Permite almacenar pares clave-valor.
- No permite claves duplicadas.

Uso:

- Muy utilizada para mapas y diccionarios.

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

map.put(key:"A", value:1);
map.put(key:"B", value:2);
map.put(key:"C", value:3);
```

```
System.out.println(map.get(key:"B"));
// 2
```

Stack

Características:

- Basada en la clase *Vector*.
- Sigue el principio LIFO (**L**ast **I**n **F**irst **O**ut).

Uso:

- Ideal para problemas de recursión, deshacer operaciones, etc.

```
Stack<Integer> pila = new Stack<>();

pila.add(e:1);
pila.add(e:2);
pila.add(e:3);
```

```
int cantidad = pila.size();
for (int i = 0; i < cantidad; i++) {
    System.out.println(pila.pop());
}
// 3 2 1
```

Queue

Características:

- Sigue el principio FIFO (**F**irst **I**n **F**irst **O**ut).
- Java proporciona varias implementaciones de la interface *Queue*, como **LinkedList** o **PriorityQueue**, entre otras.

Uso:

- Adecuada para la gestión de trabajos en impresoras, manejo de solicitudes en servidores, algoritmos de búsqueda en grafos, etc.

Queue

LinkedList:

```
Queue<Integer> cola = new LinkedList<>();

cola.add(e:1);
cola.add(e:2);
cola.add(e:3);
```

```
int cantidad = cola.size();
for (int i = 0; i < cantidad; i++) {
    System.out.println(cola.poll());
}
// 1 2 3
```

PriorityQueue:

- Los elementos se ordenan según el orden natural o un comparador.

```
Queue<Integer> pCola = new PriorityQueue<>();

pCola.add(e:3);
pCola.add(e:1);
pCola.add(e:2);
```

```
int cantidad = pCola.size();
for (int i = 0; i < cantidad; i++) {
    System.out.println(pCola.poll());
}
// 1 2 3
```

Ejercitación