Iterando a partir del índice

¡Comencemos!

¿Qué aprenderás?

- Comprender la interfaz Iterator y sus principales métodos para tener una mejor claridad del uso y lo que nos facilita como programador.
- Aplicar las distintas operaciones de un arreglo para “buscar”, “agregar”, “eliminar” y “mostrar” agilizando el manejo y dando utilidad al interfaz Iterator.

Introducción

Estudiaremos una nueva forma de recorrer un `Arraylist`, esto nos permitirá entender y aplicar la interfaz `Iterator`.

Con Iterator manejamos el arreglo de una forma más fácil ya que se puede realizar una transformación de datos en tiempo de ejecución del programa.

¡Vamos con todo!

Usando Iterator

Es una interfaz que permite recorrer un arreglo en un bucle, nos permite recorrer cualquier arreglo de tipo dinámico ArrayList, List, LinkedList, entre otros.

El Iterator tiene métodos especiales para su uso y son únicos para esta interfaz.

- `hasNext()`: Comprueba que siguen quedando elementos en el iterador.
- `next()`: Nos da el siguiente elemento del iterador.
- `remove()`: Elimina el elemento del Iterador.

Para trabajar con la interfaz Iterator se necesita importar su librería:

```
import java.util.Iterator;
```

Ejercicio Guiado: Iterator

A continuación, se presenta un ejemplo completo del uso de `Iterator` con Arraylist:

Paso 1: Se declara e inicializa un ArrayList de tipo String.

```
ArrayList<String> random = new ArrayList<String>();
```

Paso 2: Se agregan elementos al ArrayList.

```
random.add("Primero");
random.add("Segundo");
random.add("Tercero");
random.add("Cuarto");
random.add("Quinto");
```

Paso 3: Se declara un ciclo `for` donde se crea una variable temporal llamado `iterator` que es de tipo `Iterator`.

La variable `random` de tipo `ArrayList` usa su método `iterator()`, el cual permite crear una variable de tipo `Iterator`. Esta sentencia es crucial ya que al realizar toda la información que está dentro del arreglo ahora queda en la variable `iterator`.

```
for (Iterator iterator = random.iterator(); iterator.hasNext();) {

}
```

Paso 4: La sentencia `String elementos = (String) iterator.next();` rescata el primer elemento del arreglo en el `Iterator` y lo asigna a la variable local llamada `elementos`.

```
for (Iterator iterator = random.iterator(); iterator.hasNext();) {
     String elemento = (String) iterator.next();
}
```

Paso 5: Luego se muestra por consola el elemento dentro del bucle.

```
System.out.println("El elemento es"+ elemento);
```

La solución completa es la siguiente:

```
// Paso 1
ArrayList<String> random = new ArrayList<String>();

// Paso 2
random.add("Primero");
random.add("Segundo");
random.add("Tercero");
random.add("Cuarto");
random.add("Quinto");

// Paso 3
for (Iterator iterator = random.iterator(); iterator.hasNext();) {

     // 4
     String elemento = (String) iterator.next();
     // 5
     System.out.println("El elemento es"+ elemento);
}
```

Este código nos permite recorrer e interpretar el ciclo dentro de un ArrayList.

- Con la variable `elementos` podemos realizar condiciones de búsqueda dentro del bucle.
- El método `remove()` elimina elementos dentro del iterator, pero no del Arraylist.

Matrices o arreglos de múltiples dimensiones

- Explicar qué son a partir de un ejemplo o caso de uso real.
- Ejercicio de iteración a partir de un arreglo bidimensional.

Debemos partir definiendo que los arreglos son estructuras de datos homogéneas (todos los datos son del mismo tipo) que permiten almacenar un determinado número de datos bajo un mismo identificador, es decir, podemos pensar en estos arreglos o "Arrays" como vectores, matrices, etc. Sin embargo, es importante también destacar que dependiendo del tamaño del arreglo será la dimensión que este tenga.

La dimensión de un arreglo será el tamaño que este posea y que se define como la forma de organizar los datos. Por ejemplo, podemos tener un arreglo unidimensional (1 dimensión), bidimensional (2 dimensiones) y tridimensional (3 dimensiones). En el caso de Java no hay restricciones en cuanto al número de dimensiones que puede tener un arreglo, pero para los/las programadores/as es más compleja su manipulación ante un aumento significativo del número de dimensiones.

Para ejemplificar lo que hemos hablado previamente, usaremos el mismo ejercicio anterior donde íbamos incorporando los datos: "Primero", "Segundo", "Tercero", "Cuarto", "Quinto" a un ArrayList llamado "random". Si pudiéramos graficar el ejercicio sería algo similar a lo que se muestra a continuación:

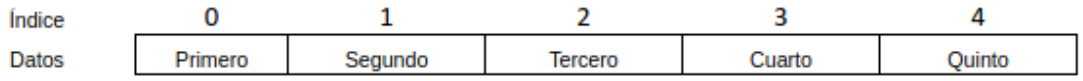

Imagen 1. Referencia arreglo de 1 fila
Fuente: Desafío Latam

Donde el índice nos daría a conocer la posición con la cual van llegando estos datos en el ArrayList del tipo unidimensional o también conocido como vector. Es importante destacar que si siguiéramos agregando datos al ArrayList, estos se seguirán agregando a la derecha del último elemento incorporado (que sería "Quinto").

Por lo tanto sería un ArrayList con 1 fila y 5 columnas o dicho de otra forma, un vector (matriz unidimensional) de 5 columnas. Si agregamos una segunda fila a continuación de ésta, podríamos hablar del origen de una matriz, ya que hay más de una fila y una columna involucradas al mismo tiempo. El índice también será para contar los datos que van llegando desde la segunda fila.

Índice	0	1	2	3	4
0	Primero	Segundo	Tercero	Cuarto	Quinto
1	Sexto	Séptimo	Octavo	Novemo	Décimo

Imagen 2. Referencia arreglo de 2 filas
Fuente: Desafío Latam

Declaración y construcción

La declaración de un arreglo bidimensional en Java se muestra a continuación:

```
tipo[][] nombre;
```

Donde los corchetes cuadrados `[ ]` indican que se trata de un arreglo bidimensional o matriz (cada corchete corresponde a una dimensión distinta), el `tipo` hace referencia a los tipo de datos que se usarán (String, int, float, etc) y el `nombre` corresponde al alias que se le colocará a este arreglo.

Otra forma de ver la declaración es igual que los arreglos unidimensionales, donde el proceso de declaración y construcción puede realizarse 2 sentencias, tal como se muestra a continuación:

```
nombre = new tipo[Filas][Columnas];
```

Donde `nombre` es el alias, **`new`** es la creación del arreglo, `tipo` es el tipo de datos, `[Filas]` será el número total de filas y `[Columnas]` será el número total de columnas.

También podemos encontrarlas mediante la expresión de constantes simbólicas, ya sea las 2 dimensiones o sola una de éstas.

```
public static final R = 10;

public static final C = 6;

matriz = new boolean [R][C]
```

Y también pueden usarse variables enteras previamente inicializadas:

```
int filas = 12;
```

```
int columnas = 8;

matriz = new double [filas][columnas]
```

La primera dimensión se refiere al total de filas de la matriz y la segunda se refiere al total de columnas.

Propiedad Length

En el caso de un arreglo de cualquier tipo de elementos, su única propiedad es `length`, la cual contiene el número de elementos en el arreglo, pero para un arreglo bidimensional, esta propiedad puede indicar el número de filas como el número de columnas, dependiendo de cómo se utilice. Por ejemplo:

```
int [] [] matriz = new int [5][3];
int x = matriz.length;
int y = matriz[2].length;
```

En la primera línea se declara una matriz con enteros de 5 filas y 3 columnas, luego declaramos una variable entera en la segunda línea y se almacena el resultado de la propiedad length. Por lo que en este punto x=5 que son el número de filas. Luego, en la línea 3 vemos que se vuelve a utilizar length, pero ahora se establece que sea en la fila 2, por lo que se obtendrá el total de columnas que tiene la matriz hasta la fila 2. Por lo tanto, ¿Es posible que existan matrices que no sean exactamente iguales en dimensiones? La respuesta es sí.

La matrices pueden ser cuadradas:

10	3	4	5
4	8	3	2
2	5	9	0
7	5	9	6

Imagen 3. Matriz cuadrada 4x4 (Filas x Columnas)
Fuente: Desafío Latam

La matrices pueden ser no cuadradas:

10	3	4	5	8
4	8	3	2	2
2	5	9	0	5

Imagen 4. Matriz no cuadrada 3x5 (Filas x Columnas)
Fuente: Desafío Latam

La matrices pueden ser irregulares:

1	2		
3	4	5	
6	7		
8	9	10	11

Imagen 5. Matriz irregular
Fuente: Desafío Latam

Para la declaración de la última matriz, se puede realizar lo siguiente:

```
int[][] matriz = new int[4][];  // Sólo se indica el número de
renglones, pero no el de columnas, porque será diferente en cada
renglón.
matriz[0] = new int[2]; // En el renglón 0, se construyen 2 columnas
matriz[1] = new int[3]; // El renglón 1 contiene 3 columnas
matriz[2] = new int[2]; // El renglón 2 contiene 2 columnas
matriz[3] = new int[4]; // El renglón 3 contiene 4 columnas.
```

Además se puede declarar, construir e inicializar esta matriz realizando lo siguiente:

```
int[][] matriz = { {1, 2}, {3, 4, 5}, {6, 7}, { 8, 9, 10, 11} };
```

Introducción de datos a un arreglo bidimensional

Para introducir un dato a una casilla específica de un arreglo usaremos:

```
arreglo [fila][columna] = valor;
```

Donde `arreglo` es el nombre del ArrayList, `[fila]` es la dimensión de la fila, `[columna]` es la dimensión de la columna y `valor` es el valor a almacenar en esa casilla.

Búsqueda de un dato en un arreglo bidimensional

Si queremos acceder a los elementos de un arreglo bidimensional debemos hacer el conteo de índices que inicia desde 0, por lo que si un arreglo ha sido declarado y construido de la siguiente forma:

```
int[][] arreglo = new int[5][4];
```

Significa que tiene 20 casillas, elementos o espacios donde almacenará datos del tipo entero (int).

- Tendrá 5 filas, donde la dimensión va desde 0 hasta 4.
- Tendrá 4 columnas, donde dimensión va desde 0 hasta 3

En el ejemplo anterior, podemos agregar elementos haciendo lo siguiente:

```
int[][] arregloMatriz = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12},
{13,14,15,16}, {17,18,19,20}}
```

Por lo que la matriz sería la siguiente:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

Imagen 6. Matriz de 20 casillas
Fuente: Desafío Latam

Si buscamos por ejemplo, el elemento que está ubicado en la fila 3 y la columna 2, debemos hacer lo siguiente:

```
System.out.println(arregloMatriz[3][2]);
```

El cual nos dará como resultado

```
15
```

Ejercicio de iteración por fila:

Crearemos una matriz llamada “matrizEjemplo” de 3x3, incorporando números del 1 al 9. Por lo que si queremos ver los primeros datos de cada fila recorriendo hasta la columna 3 de la fila 3, debemos hacer lo que se muestra a continuación:

```
int n=3;
int matrizEjemplo[][] = {{1,2,3},{4,5,6},{7,8,9}}

for (int i=0; i<auxiliar.length; i++){
            for (int j=0; j<auxiliar.length; j++){
          System.out.println(matrizEjemplo[i][j] + "\t")
            }
      }
```

El resultado es: 1, 4, 7.

Ejercicio de iteración por columna:

Si buscamos ver los elementos que están en la primera fila recorriendo cada columna, podemos usar la siguiente iteración:

```
int n=3;
int matrizEjemplo[][] = {{1,2,3},{4,5,6},{7,8,9}}

for (int i=0; i<auxiliar.length; i++){
            for (int j=0; j<auxiliar.length; j++){
          System.out.println(matrizEjemplo[j][i] + "\t")
            }
      }
```

El resultado es: 1, 2, 3.