Arreglos y Archivos (Parte I)

Introducción a Arrays (Arreglos)

Competencias

- Reconocer los arrays en el lenguaje de programación para sus distintas implementaciones y uso.
- Construir programas utilizando arrays para manejar y tratar la información en volúmenes de datos.

Introducción

En este capítulo conoceremos e implementaremos los arreglos, sus diferentes formas de declaración, inicialización y llenado.

Conocer los arreglos es necesario para adentrarnos en el mundo de las base de datos que, a nivel de lógica, es lo más cercano a manejar grandes volúmenes de información. Para lograr este manejo de información aprenderemos distintos métodos que nos ayudarán a implementar los arreglos de manera oportuna y efectiva consiguiendo un mejor trato de este tipo de datos.

¿Para qué sirven los Arrays?

Los arrays se utilizan mucho dentro de la programación. Java trata un arreglo como una variable normal, esto quiere decir que se declara, inicializa y utiliza. Los corchetes [ ] le indican al compilador que esa variable es un arreglo de un tipo de dato en específico. Nos permite resolver diversos tipos de problemas.

Ejemplo:

```
int [] a = {2,4,5,6};
```

```
String[ ] nombre = {"Juan", "Pedro"};   //Array de 2 elementos
```

Algunos posibles usos son:

- En una aplicación web podemos traer los datos de la base de datos en un array y luego mostrar los datos en la página.
- Obtener los datos desde una API (Aplication Programming Interface), los datos devueltos de una API pueden venir como una colección y podemos guardarlos en una base de datos o archivos.
- Traer información guardada en uno o más archivos.

Tipos de arreglos

En Java existen dos clasificaciones para los arreglos:

- **De tipo de datos primitivos (int, char ,double, boolean…):** Son arreglos de tamaño estático, al cual se le define el tamaño al inicializarlo, y pueden almacenar tipos de datos primitivos como objetos. Se almacena en la memoria stack.
- **De tipo objeto:** Java a ese tipo de variables las trabaja como objeto y no como variable de dato primitivo, estas variables se almacenan en la memoria HEAP. En tiempo de ejecución de nuestro programa pueden ir cambiando su tamaño (agregando o quitando elementos). Estos son algunos tipos de Arrays: Arraylist, LinkedList, HashSet, List, entre otros.

Creando un Array

Para crear un array utilizaremos la siguiente sintaxis dependiendo de cada caso.

Estáticos

Para definir una variable como array, debemos especificar el tipo de dato, los corchetes y el nombre de la variable.

Al momento de definir el array, los corchetes pueden estar antes o después del nombre de la variable.

```
int[] a;
int b[];
```

Para definir el tamaño del array, ahora debemos decirle de qué tamaño será dentro de los corchetes.

```
a = new int [4]; //array llamado a, de tipo enteros, de tamaño 4
```

Para introducir un valor podemos escribir:

```
a[0] = 4; // en la posición 0 agrega el valor 4
```

O bien, al momento de declarar un arreglo, asignarle los valores iniciales:

```
int[] ba = {2,4,5,6};//arreglo llamado ba, de tipo enteros, tamaño 4
```

Dinámicos

Para el caso de crear un arreglo de tipo dinámico, usaremos `ArrayList`, agregando previamente la librería `import java.util.ArrayList;`

```
ArrayList <Integer> arrayInt = new ArrayList<Integer> ();
```

Índices

Cada elemento del arreglo tiene una posición determinada, a la cual se le denomina índice. El índice nos permite acceder al elemento que está dentro del arreglo.

Por ejemplo, tenemos:

```
System.out.printf("%d\n", a[0]) // 1
```

Si queremos acceder al primer elemento, debemos acceder a la posición 0.

```
System.out.printf("%d\n", a[0]) // 1
```

Recorrido

Para recorrer un array se debe ocupar sentencias de bucle (for, for each, While), esto se recorre desde 0 hasta `n-1` , donde `n` es el tamaño del arreglo.

Con la propiedad `length` de un arreglo de corchetes podemos obtener el tamaño:

```
int i;
int[] a = {1,2,3,4,5};
int n = a.length;
for(i=0;i<n;i++){
 System.out.printf("%d\n",a[i]);
}
```

Salida

```
1
2
3
4
5
```

Tipos de errores con los índices en un Arrays

Los errores más comunes cuando se trabaja con arrays son los de tipo índices, esto quiere decir que se intenta agregar más elementos a un arrays definido y se accede a un índice que no existe; como por ejemplo un índice negativo.

```
int[] a = {1,2,3,4,5}; System.out.printf("%d\n", a[5]);
```

Pero al ejecutar el código obtenemos el error que nos dice que el índice 5 está fuera del límite del arreglo.

```
------------------------------------------------------------------------
---
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for
length 5 at .(#33:1)
```

¿Y si colocamos índices negativos?

```
System.out.printf("%d\n", a[-1]);
```

Ocurre el mismo error:

```
------------------------------------------------------------------------
---
 java.lang.ArrayIndexOutOfBoundsException: Index -1 out of bounds for
length 5 at .(#34:1)
```

Ejercicio guiado: Sumar dentro de un arreglo

Construir un programa el cual nos permita sumar los valores que estén entre 1 y 10 dentro de un arreglo.

Paso 1: Creamos un método llamado Suma que retorna un número entero de la suma.

```
// Paso 1
public static int suma() {

}
```

Paso 2: Dentro del método suma, declaramos una variable local llamada suma que parte en cero.

```
public static int suma() {
    //Paso 2
    int suma = 0;
}
```

Paso 3: Inicializamos el arreglo con valores aleatorios.

```
public static int suma() {
    int suma = 0;
    //Paso 3
    int[] arreglo = { 1, 5, 11, 33, 4, 6, 7, 44, 6, 1, -1 };
}
```

Paso 4: Recorremos el arreglo con un ciclo `for`.

```
public static int suma() {
    int suma = 0;
    int[] arreglo = { 1, 5, 11, 33, 4, 6, 7, 44, 6, 1, -1 };
    //Paso 4
    for (int x = 0; x < arreglo.length; x++) {
    }
}
```

Paso 5: Dentro del ciclo, realizamos condición `if` donde preguntamos por los valores que están en el intervalo solicitado.

```
public static int suma() {
    int suma = 0;
    int[] arreglo = { 1, 5, 11, 33, 4, 6, 7, 44, 6, 1, -1 };
    for (int x = 0; x < arreglo.length; x++) {
        //Paso 5
        if (arreglo[x] >= 1 && arreglo[x] <= 10) {

        }
    }
}
```

Paso 6: Si la condición se cumple, sumará todos los valores.

```
public static int suma() {
    int suma = 0;
    int[] arreglo = { 1, 5, 11, 33, 4, 6, 7, 44, 6, 1, -1 };
    for (int x = 0; x < arreglo.length; x++) {
        if (arreglo[x] >= 1 && arreglo[x] <= 10) {
           //Paso 6
           suma = suma + arreglo[x];
        }
    }
}
```

Paso 7: Este método lo llamamos dentro del método `main`.

```
public static void main(String[] args) {
    //Paso 7
    System.out.println("La suma es: " + suma());
}
```

Finalmente el ejercicio quedaría de la siguiente manera:

```
public static void main(String[] args) {
            System.out.println("La suma es: " + suma());
      }

      public static int suma() {
            int suma = 0;
            int[] arreglo = { 1, 5, 11, 33, 4, 6, 7, 44, 6, 1, -1 };
            for (int x = 0; x < arreglo.length; x++) {
                  if (arreglo[x] >= 1 && arreglo[x] <= 10) {
                        suma = suma + arreglo[x];
                  }
            }
            return suma;
      }
```

Ejercicio propuesto (1)

Construir un programa en el cual se requiere lo siguiente:

- Crear un método el cual retorna el promedio de sueldo en tipo de dato `double`.
- Tenemos el siguiente arreglo como ejemplo:

```
int[] sueldos = { 400000, 760000, 1100000, 650000, 654980, 987300,
700450, 442300 };
```

Se requiere promediar los sueldos que sean `mayores a 500000` y retornar este valor.

Operaciones básicas en un Array dinámico

Competencias

- Comprender la documentación de la clase ArrayList para hacer uso de sus métodos y codificar de manera rápida.
- Aplicar operaciones de un array dinámico para “agregar”, “eliminar”, “ordenar”, “contar” para conocer los métodos esenciales de un ArrayList.

Introducción

Los arreglos tienen muchos métodos que nos permiten realizar operaciones básicas como, por ejemplo, saber si un elemento está dentro de un arreglo, agregar elementos, borrar elementos y contar elementos. Por lo mismo es importante saber leer e interpretar su documentación.

Los arreglos dinámicos nos ayudan a tener una imagen a grandes rasgos de cómo es el comportamiento de una base de datos.

Para tener más información se recomienda acceder a Arreglos dinámicos (ArrayList) ubicado en “Material Complementario”.

Comprender e interpretar la documentación de la clase ArrayList

Primero, vemos a qué módulo y paquete pertenece la clase ArrayList, cómo se define una descripción, entre otros.

Luego, vienen las definiciones de los constructores y todos los métodos que tiene la clase.

A continuación se detalla cada uno de sus constructores y métodos:

Agregar un elemento

Modificador y Tipo	Método y descripción.
boolean	add(E e) Agrega un elemento específico al final de la lista.
void	add(int index, E element) Inserta un elemento específico en la posición indicada.

Tabla 1. Agrega elementos a un ArrayList.
Fuente: Desafío Latam.

El método principal para agregar elementos a una Arraylist es `add`, a medida que se vayan agregando, el índice va aumentando de forma secuencial, partiendo como valor inicial en cero.

```
ArrayList <String> a = new ArrayList <String> ();
a.add(1);
a.add(2);
a.add(3);
```

```
System.out.println(a);
```

A diferencia de los array estáticos, el `ArrayList` permite mostrar todos sus elementos sin recorrer.

Salida del ArrayList

```
[1,2,3]
```

Ejercicio guiado: Agregar elementos a un arreglo

Dado un arreglo llamado "ingredientes" se nos pide crear un programa donde el usuario pueda consultar si un ingrediente existe en la pizza, y si no existe debe ser añadido a la lista de ingredientes.

```
ingredientes // [piña, jamón, salsa, queso]
```

- ***Paso 1:*** Importar la clase ArrayList

```
import java.util.ArrayList;
```

- ***Paso 2:*** Crear un ArrayList de tipo String llamado "ingredientes" el cual se le agrega los valores por defectos citados en el ejercicio.

```
ArrayList<String> ingredientes = new ArrayList<String> ();
ingredientes.add("piña");
ingredientes.add("jamón");
ingredientes.add("salsa");
ingredientes.add("queso");
```

- ***Paso 3:*** Crear un objeto Scanner el cual nos permitirá leer los datos ingresados por consola.

```
Scanner sc = new Scanner(System.in);
String ingrediente = sc.nextLine();
```

- ***Paso 4:*** En la condición if, preguntar si el ingrediente ingresado está en el arreglo, si esto existe, mostramos por consola el mensaje correspondiente.

```
if(ingredientes.contains(ingrediente)){
 System.out.printf("El ingrediente ya se encuentra dentro de la
pizza\n");
}
```

- ***Paso 5:*** Si el ingrediente no existe dentro del arreglo se mostrará el mensaje correspondiente más el arreglo completo.

```
else {
 ingredientes.add(ingrediente);
 System.out.printf("El ingrediente %s fue
agregado\n",ingrediente);
}
System.out.println(ingredientes);
```

Solución completa

```
// Paso 1
import java.util.ArrayList;

// Paso 2
ArrayList<String> ingredientes = new ArrayList<String> ();
ingredientes.add("piña");
ingredientes.add("jamón");
ingredientes.add("salsa");
ingredientes.add("queso");

// Paso 3
Scanner sc = new Scanner(System.in);
String ingrediente = sc.nextLine();

// Paso 4
if(ingredientes.contains(ingrediente)){
 System.out.printf("El ingrediente ya se encuentra dentro de la
pizza\n");
}

// Paso 5
else {
 ingredientes.add(ingrediente);
 System.out.printf("El ingrediente %s fue
agregado\n",ingrediente);
}
System.out.println(ingredientes);
```

Si agregamos el elemento "champiñon", esta sería la salida:

```
El ingrediente champiñón fue agregado
[piña, jamón, salsa, queso, champiñón]
```

Remover elementos

Existen muchas formas de eliminar uno o varios elementos de un Arraylist.

Eliminar todos los elementos de un arreglo

Si queremos eliminar todos los elementos del arreglo, usaremos el método clear();

```
// Crear arreglo nombres
ArrayList<String> nombres = new ArrayList<String> ();

// Añadir “Juan” al arreglo nombres
nombres.add("Juan");

// Eliminar el arreglo nombres
nombres.clear();

//Imprimir mensaje de salida
System.out.println(" Valores en el arreglo" + nombres);
```

```
Valores en el arreglo []
```

Eliminar elemento según índice

El método `remove(int index)` es el encargado de eliminar elementos dentro del ArrayList según su índice como parámetro de entrada.

```
ArrayList<String> nombres = new ArrayList <String>();
nombres.add("Juan");
nombres.add("Pedro");
nombres.add("Luis");
nombres.remove(1); // "Pedro"
System.out.println(nombres);
```

Cuando realizamos esta operación, el valor de retorno del método es el valor eliminado, por lo que podríamos hacer lo siguiente:

Eliminar el elemento que coincida con el valor entregado.

¿Qué pasa ahora si entregamos el valor del elemento que queremos eliminar?

Si queremos eliminar un elemento, por ejemplo, el valor "a".

En este caso, el método `remove()` nos retornará `true` o `false`, si se hizo o no la eliminación.

```
ArrayList<String> a = new ArrayList <String>();
a.add("a");
a.add("b");
a.add("c");
a.add("d");
System.out.println(a);
String borrado = a.remove(1); // "b"
System.out.println(a);
System.out.println("Elemento borrado: " + borrado);
```

```
[a, b, c, d]
[a, c, d]
```

```
Elemento borrado: b
```

Como dato importante, antes de eliminar un elemento necesitas validar si existen elementos dentro del arreglo, esto para que el índice que se desea borrar al menos exista y el programa no se caiga .

Eliminar elemento que coincida con el valor entregado

```
ArrayList<String> a = new ArrayList <String>();
a.add("a");
a.add("b");
a.add("c");
a.add("d");
System.out.println(a); //[a,b,c,d]
a.remove("a");
System.out.println(a); //[b,c,d]
```

```
[a, b, c, d]
[b, c, d]
```

En este caso, el método `remove()` nos retornará `true` o `false`, dependiendo si se realiza o no la eliminación.

¿Qué pasa si tenemos más de un elemento con el mismo valor?

```
ArrayList<String> a = new ArrayList <String>();
a.add("a");
a.add("b");
a.add("c");
a.add("c");
a.add("c");
a.add("c");
a.add("a");
System.out.println(a); //[a, b, c, c, c, c, a]
a.remove("a");
System.out.println(a); //[b, c, c, c, c, a]
```

```
[a, b, c, c, c, c, a]
[b, c, c, c, c, a]
```

Va a eliminar solo la primera ocurrencia de este.

Eliminar todos los elementos dentro de una colección

Si queremos eliminar todos los elementos que coincidan con "a" del ejemplo anterior, tenemos el método `public boolean removeAll(Collection c);` que recibe como parámetro una colección.

Por lo que podemos crear un ArrayList con los elementos que queremos eliminar de nuestro arreglo:

```
ArrayList<String> a = new ArrayList <String>();
a.add("a");
a.add("b");
a.add("c");
a.add("c");
a.add("c");
a.add("c");
a.add("a");
a.add("d");
System.out.println(a); //[a, b, c, c, c, c, a, d]
ArrayList<String> elementosABorrar = new ArrayList<String>();
elementosABorrar.add("a");
elementosABorrar.add("c");
a.removeAll(elementosABorrar);
System.out.println(a); //[b, d]
```

```
[a, b, c, c, c, c, a, d]
[b, d]
```

Ejercicio guiado: Agregar número par

Crear un método que permita agregar solo números pares a un ArrayList y mostrar el o los elementos del ArrayList.

Paso 1: Se crea un método llamado `agregarNumeroPar` que recibe como parámetro de entrada un número de tipo entero.

Paso 2: Crear una variable local de tipo ArrayList llamada números.

Paso 3: Realizamos la condición if para validar si el número ingresado es un número par.

Paso 4: Si la condición se cumple, agregamos el elemento al ArrayList.

Paso 5: Mostramos el resultado con la sentencia `System.out.println`.

```
public static void main(String[] args) {
            agregarNumeroPar(3);
      }

public static  void agregarNumeroPar(int numero) {
            ArrayList<Integer>  numeros = new ArrayList<Integer>();
            if(numero%2 == 0) {
                  numeros.add(numero);
            }
            System.out.println(numeros);
      }
```

Ejercicio propuesto (2)

Crear un método llamado `agregaElemento` el cual nos permite agregar un elemento de una casa a un `ArrayList` , este elemento se ingresa como un parámetro de entrada.

Se requiere validar si el elemento que se ingresará no existe en el arreglo.

- Si no existe se agregará.
- Si existe se mostrará el mensaje “Elemento ya existe”.

Se entrega este ArrayList como base para validar el campo de entrada.

```
ArrayList<String> elementos = new ArrayList<String>();
elementos.add("mesa");
elementos.add("Refrigerador");
elementos.add("Cocina");
elementos.add("lavadora");
```

Otros Métodos utilizados en ArrayList

Competencias

- Aplicar métodos importantes como size(), sort(), entre otros, para manejar fácilmente volúmenes de información.

Introducción

Aplicaremos nuevos métodos de un ArrayList para utilizar buenas prácticas de programación, agilizando el tiempo y el código dentro del programa.

Cada método que aplicaremos permitirá el mejor desempeño del estudiante a la hora de abordar casos complejos con volúmenes grandes de información. Para esto es necesario aplicar y reforzar métodos ya aprendidos en conjunto con los nuevos, porque los ArrayList nos disponibiliza métodos para cada caso.

Reemplazar elementos según índice

Usaremos el método `set(int index, E element)` para reemplazar un elemento del arreglo. Este método retorna el elemento previo en dicha posición.

```
ArrayList<String> a = new ArrayList <String>();
a.add("a");
a.add("b");
a.add("c");
a.add("d");
System.out.println(a); //[a, b, c, d]
a.set(1, "k");
System.out.println(a); //[a, k, c, d]
```

```
String elementoCambiado = a.set(0,"j");
System.out.println("elemento cambiado" + elementoCambiado);
elemento cambiado a
```

Contar elementos del `ArrayList` con el método `size()`

Podemos contar la cantidad de elementos de un array con el método `size()`. Cuando no existen elementos en el arreglo, su valor es cero.

```
ArrayList<String> a = new ArrayList <String>();
a.add("a");
a.add("b");
a.add("c");
a.add("d");
System.out.println(a.size()); //4
```

Importar ArrayList

En Java podemos aplicar otros métodos sobre los ArrayList que permiten facilitar el trabajo al momento de operar sobre los elementos. Para estas operaciones utilizaremos la librería.

```
import java.util.ArrayList;
```

Ordenar los elementos con el método `sort()`

Nos permite ordenar la lista de manera ascendente. En el caso de los Strings, los ordena comparando carácter a carácter:

```
ArrayList<String> paises = new ArrayList<String>();
paises.add("Chile");
paises.add("Argentina");
paises.add("Colombia");
paises.add("Perú");
paises.add("Venezuela");
Collections.sort(paises);
System.out.println(paises); //[Argentina, Chile, Colombia, Perú,
Venezuela]
```

Al ordenarse quedan en orden alfabético:

```
[Argentina, Chile, Colombia, Perú, Venezuela]
```

Agregaremos un nuevo elemento a la lista, pero con minúscula la primera letra y llamaremos al método `sort()`.

```
paises.add("chile");
Collections.sort(paises);
System.out.println(paises);
```

```
[Argentina, Chile, Colombia, Perú, Venezuela, chile]
```

Podemos ver que “chile”, luego de ordenar el arreglo, queda al final cuando debería haber quedado en la posición 1. Para solucionar esto escribiremos lo siguiente:

```
Collections.sort(paises,String.CASE_INSENSITIVE_ORDER);
System.out.println(paises);
```

```
[Argentina, Chile, chile, Colombia, Perú, Venezuela]
```

¿Y si ahora queremos tener el orden descendente?

Invertir los elementos del ArrayList con el método `reverse()`.

Para invertir una lista, existe el método `reverse()` que se encarga de invertir una ArrayList.

```
Collections.reverse(paises);
System.out.println(paises);
```

```
[Venezuela, Perú, Colombia, chile, Chile, Argentina]
```

Obtener el mínimo y máximo valor del ArrayList

- `min ()`: Retorna el valor mínimo dentro del ArrayList.
- `max()`: Retorna el valor máximo dentro del ArrayList.

```
ArrayList<Integer> numeros = new ArrayList<Integer>();
numeros.add(5);
numeros.add(1);
numeros.add(4);
numeros.add(1);
numeros.add(2);
numeros.add(6);
System.out.println(Collections.min(numeros)); //1
System.out.println(Collections.max(numeros)); //6
```

Obtener la frecuencia de un elemento en el ArrayList

Si queremos saber cuántas veces existe la ocurrencia de un elemento en el ArrayList, podemos usar el método `frequency()`.

```
System.out.println(Collections.frequency(numeros, 1)); //2
```

Ejercicio guiado: Lista de platos

Crear un método llamado "ordenar" que nos permita ordenar alfabéticamente una lista de platos de un restaurante, también se debe mostrar lista ordenada por pantalla.

Esta lista cuenta con los siguientes datos:

- Cazuela.
- Porotos.
- Pastel de Choclo.
- Ají de gallina.
- Ceviche.
- Arepas.

Paso 1: Creamos el método estático llamado ordenar.

```
public static void ordenar() {
}
```

Paso 2: Se crea una variable local de tipo ArrayList llamada lista.

```
public static void ordenar() {
   ArrayList<String> lista = new ArrayList<String>();
}
```

Paso 3: Agregamos cada elemento al arreglo.

```
public static void ordenar() {
   ArrayList<String> lista = new ArrayList<String>();
   lista.add("Cazuela");
   lista.add("Porotos");
   lista.add("Pastel de Choclo");
   lista.add("Ají de Gallina");
   lista.add("Ceviche");
   lista.add("Arepas");
}
```

Paso 4: Utilizamos el método sort para ordenar la lista.

```
Collections.sort(lista);
```

Paso 5: Se muestra por consola la lista.

```
System.out.println("La lista de comida es " + lista);
```

La solución completa quedaría así:

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

	// Paso 1
	public  static  void ordenar() {

		// Paso 2
		ArrayList<String> lista = new ArrayList<String>();

		// Paso 3
		lista.add("Cazuela");
		lista.add("Porotos");
		lista.add("Pastel de Choclo");
		lista.add("Aji de Gallina");
		lista.add("Ceviche");
		lista.add("Arepas");

		// Paso 4
		Collections.sort(lista);

		// Paso 5
		System.out.println("La lista de comida es " + lista);
	}
```

Ejercicio propuesto (3)

Crear un método que nos permita obtener la nota máxima y mínima de un conjunto de exámenes del curso Programación Java.

Estas notas deben ser mostradas por consola.

Se entregan las siguientes notas :

- 4.7
- 2.2
- 5.4
- 6.9
- 4.4
- 2.6

Iterando a partir del índice

Competencias

- 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.

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.

Ejercicio Propuesto (4) :

Crear un método que permita buscar un nombre ingresado por parámetro dentro de un `Arraylist`. y comparar el elemento ingresado con el `ArrayList`.

Las reglas son las siguientes:

- Si el elemento existe, se debe mostrar por consola el elemento.
- Si no, mostrar el o los elementos que no fueron encontrados y eliminar de la lista.

Soluciones ejercicios propuestos

Ejercicio propuesto (1)

1. Creamos un método double llamado promedio.

```
public static double promedio() {
    // Código a ejecutar dentro del método.
}
```

2. Inicializamos el arreglo `sueldos`.

```
public static double promedio() {
    int[] sueldos = { 400000, 760000, 1100000, 650000, 654980, 987300,
700450, 442300 };
}
```

3. Recorremos el arreglo.

```
public static double promedio() {
    int[] sueldos = { 400000, 760000, 1100000, 650000, 654980, 987300,
700450, 442300 };
    for (int i = 0; i < sueldos.length; i++) {
         // Código a ejecutar dentro del ciclo for
     }
}
```

4. Sumaremos solo los sueldos mayores o igual a 500000.

```
public static double promedio() {
    int[] sueldos = { 400000, 760000, 1100000, 650000, 654980, 987300,
700450, 442300 };

    double suma = 0;
    for (int i = 0; i < sueldos.length; i++) {
         if (sueldos[i] >= 500000) {
            suma += sueldos[i];
         }
     }
}
```

5. Finalmente, promediamos el sueldo con el largo del arreglo, quedando el código completo resuelto de la siguiente manera.

```
public static double promedio() {
    int[] sueldos = { 400000, 760000, 1100000, 650000, 654980, 987300,
700450, 442300 };

    double suma = 0;
    double count = 0;
    for (int i = 0; i < sueldos.length; i++) {
         if (sueldos[i] >= 500000) {
            suma += sueldos[i];
            count++;
         }
     }
     return suma / count;
}
```

Ejercicio propuesto (2)

1. Creamos un método llamado agregarElemento (String nuevoElemento).
2. Inicializamos el ArrayList.
3. Recorremos el ArrayList con Iterator.
4. Preguntamos con condición IF si el elemento existe.
    a. Mostramos mensaje.
    b. Si no existe, agregamos el nuevo elemento.

```
// Paso 1
public static void agregarElemento(String nuevoElemento) {
	// Paso 2
	ArrayList<String> elementos = new ArrayList<String>();
	elementos.add("mesa");
	elementos.add("Refrigerador");
	elementos.add("Cocina");
	elementos.add("lavadora");
	// Paso 3
	for (Iterator iterator = elementos.iterator();
iterator.hasNext();) {
		String element = (String) iterator.next();
		// Paso 4
		if(element.contains(nuevoElemento)) {
			System.out.println("Elemento ya existe");
		}
		else {
			elementos.add(nuevoElemento);
		}
	}
}
```

Ejercicio propuesto (3)

1. Creamos un método llamado notas.
2. Inicializamos el ArrayList.
3. Utilizaremos la clase Collections con sus métodos min y max para rescatar la nota máxima y mínima.

```
// Paso 1
public static void notas() {
	// Paso 2
	ArrayList<Double> notas = new ArrayList<Double>();
	notas.add(4.7);
	notas.add(2.2);
	notas.add(5.4);
	notas.add(6.9);
	notas.add(4.4);
	notas.add(2.6);
	// Paso 3
	System.out.println(Collections.min(notas));
	System.out.println(Collections.max(notas));
}
```

Ejercicio propuesto (4)

1. Se declara e inicializa un `ArrayList` de tipo String.

2. Se agregan elementos al `ArrayList`.

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

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

5. 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.

6. Con la variable elementos podemos realizar condiciones de búsqueda. En este caso, dentro del bucle, preguntaremos si el nombre que se ingresa por parámetro está dentro del `ArrayList`.

    a. Si lo encuentra, entrará a la condición `if`.
    b. Si no lo encuentra, entrará en condición `else`.

7. Luego, se muestra por consola el elemento dentro del bucle.

8. El método `remove()` elimina del iterator todas las coincidencias no encontradas en el `if`.

El código de la solución quedaría de la siguiente manera:

```
static void filtroNombre(String parametro) {
	// Paso 1
	ArrayList<String> nombres = new ArrayList<String>();
	// Paso 2
	nombres.add("Juan");
	nombres.add("Pedro");
	nombres.add("Luis");
	nombres.add("Ana");
	// Paso 3 y 4
	for (Iterator iterator = nombres.iterator(); iterator.hasNext();)
{
		// Paso 5
		String elemento = (String) iterator.next();
		// Paso 6
		if (elemento.equals(parametro)) {
			// Paso 7
			System.out.println("Elemento encontrado -> " +
elemento);
		} else {
			Paso // 8
			iterator.remove();
			System.out.println("Elemento no encontrado -> " +
elemento);
		}
	}
}
```

Salida del código

```
Elemento no encontrado -> Juan
Elemento no encontrado -> Pedro
Elemento encontrado -> Luis
Elemento no encontrado -> Ana
```