UT6. ARRAYS DE OBJETOS, LA CLASE ARRAYS, ARRAYLIST y OTROS

Índice de contenido

1 Arrays de objetos

Los arrays no solo pueden almacenar tipos primitivos. También pueden almacenar objetos. El funcionamiento es similar en los dos casos. Conviene destacar si no se especifica ningún valor los elementos de un array de objetos se inicializan automáticamente a null.

2 La clase Arrays

Proporciona métodos static para la manipulación de arrays. Estos métodos incluyen:

- **sort**: para ordenar un array en forma ascendente
- **binarySearch**: para buscar en un array
- **equals**: para comparar arrays
- **fill**: para rellenar valores en un array.

Ejercicio de la clase Arrays Deitel Pag. 281 9 edición.

3 Introducción a las colecciones y la clase ArrayList

La API de Java provee de varias estructuras de datos predefinidas conocidas como colecciones, que se utilizan para almacenar grupos de objetos

relacionados. Estas clases proporcionan métodos eficientes que organizan, almacenan y obtiene datos sin necesidad de saber cómo se almacenan éstos. Gracias a esto se reduce el tiempo de desarrollo de aplicaciones.

Los arrays tradicionales no cambian de manera automática su tamaño en tiempo de ejecución para dar cabida a elementos adicionales. La clase de colección ArrayList<T> proporciona una solución a este problema, puede cambiar su tamaño de forma dinámica para dar cabida a más elementos. La T en Java representa un tipo genérico por convención. Al declara el nuevo objeto ArrayList, hay que reemplazarlo por el tipo de elementos que deseamos que contenga. Es similar a especificar el tipo cuando declaramos un array, pero en el caso del ArrayList no es posible utilizar tipos primitivos.

La clase ArrayList debe importarse del paquete java.util.

3.1 Declaración de ArrayList

Un objeto de la clase ArrayList se declara de la siguiente manera:

```
ArrayList<clase> nombre_lista;
```

Ejemplo:

```
ArrayList<String> lista;
```

3.2 Creación de ArrayList

Tenemos varios constructores:

- public ArrayList(): Construye una lista vacía con una capacidad inicial de 10.
- public ArrayList(int initialCapacity) : Construye una lista vacía con la capacidad inicial indicada como parámetro.

```
nombre_lista = new ArrayList<>();
```

Puede unirse la declaración y la creación:

```
ArrayList<clase> nombre_lista = new ArrayList<>();
```

Ejemplo:

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

La clase ArrayList no permite el uso de tipos primitivos, en el caso de que necesitemos almacenar int, long, byte … podemos utilizar los Wrappers. Al final del capítulo hablamos de ellos brevemente.

A continuación se muestran algunos de los métodos comunes de la clase ArrayList.

Métodos	Descripción

boolean add(Object o)	Agrega un elemento al final de la lista del objeto ArrayList.
void clear()	Elimina todos los elementos del objeto ArrayList.
boolean contains(Object elem)	Devuelve true si el objeto ArrayList contiene el elemento especificado, en caso contrario devuelve false.
Object get(int index)	Devuelve el elemento en el índice especificado.
int indexOf(Object elem)	Devuelve el índice de la primera ocurrencia del elemento especificado en el objeto ArrayList. Si no está devuelve -1.
boolean remove(int i)	Sobrecargado. Elimina la primera ocurrencia del valor especificado o del elemento en el subíndice especificado.
boolean remove(Object o)	Sobrecargado. Este método borra el objeto que se pasa como parámetro si éste está presente dentro de la lista.
int size()	Devuelve el número de elementos almacenados en el objeto ArrayList.
boolean isEmpty()	Este método comprueba si hay elementos en el ArrayList.

El resto de los métodos pueden encontrarse en la documentación que aporta la web de Java:

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

4 Instrucción for mejorada

La instrucción for mejorada nos permite iterar a través de los elementos de un array o de una colección, sin utilizar un contador, La sintaxis de una instrucción for mejorada es:

```
for (parámetro:nombreArreglo)
        instrucción
```

en donde parámetro tiene un tipo y un identificador (por ejemplo, *int* numero) y *nombreArreglo* es el arreglo a través del cual se iterará.

5 El parámetro args de main en Java

En muchos sistemas, es posible pasar argumentos desde la línea de comandos (a éstos se les conoce como argumentos de línea de comandos) a una aplicación, para lo cual se incluye un parámetro de tipo String[ ] (es decir, un arreglo de objetos String) en la lista de parámetros de main. Por convención, a este parámetro se le llama args.

Cuando se ejecuta una aplicación usando el comando java , Java pasa los argumentos de línea de comandos que aparecen después del nombre de la clase en el comando java al método main de la aplicación, en forma de objetos String en el arreglo args . El número de argumentos que se pasan desde la línea de comandos se obtiene accediendo al atributo length del arreglo.

Ejemplo:

```
/*
Ejercicio2
*/
import java.util.*;
public class Ejercicio2 {
        public static void main(String[] args) {
                //DATOS
                //Entradas
                int numero;

                //Salidas
                int cuadrado;
                int cubo;

                //Leer un numero
```

```
            numero = Integer.parseInt(args[0]);
            System.out.println("El numero leido es:"+args[0]);
            //Calcular cubo y cuadrado
            cuadrado = numero*numero;
            cubo = numero*numero*numero;
            System.out.println("El cuadrado del numero es:"+cuadrado);
            System.out.println(" y el cubo del numero es: "+cubo);
      }
}
```

6 Wrappers

Los Wrappers (envoltorios) son clases diseñadas para ser un complemento de los tipos primitivos.

Los tipos primitivos son los únicos elementos de Java que no son objetos. Esto tiene algunas ventajas desde el punto de vista de la eficiencia, pero algunos inconvenientes desde el punto de vista de la funcionalidad.

Las clases Wrapper también proporcionan métodos para realizar otras tareas con lo tipos primitivos, tales como conversión con cadenas de caracteres en uno y otro sentido.

Primitive type	**Wrapper class**
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Existe una clase Wrapper para cada uno de los tipos primitivos numéricos que extienden de la clase Number, esto es, existen las clases Byte, Short, Integer, Long, Character, Float y Double (obsérvese que los nombres empiezan por mayúscula, siguiendo la nomenclatura típica de Java).