TEMA 6. ALMACENAMIENTO DE LA INFORMACIÓN EN ESTRUCTURAS DE DATOS. SEGUNDA PARTE.

INDICE

COLECCIONES DE OBJETOS

Las colecciones en Java son unas estructuras de datos cuya principal característica es que los datos que almacenan son del mismo tipo.

En Java existe un **interfaz genérico** de nombre **Collection** que es la raíz de las clases que se usan para manipular colecciones. De esa clase derivan tres interfaces genéricos

List. Es una estructura de datos secuencial en la que cada elemento se almacena en un índice o posición determinado. Permite elementos repetidos. Se usa para crear listas de datos.

Set. Es una estructura que se usa para almacenar un conjunto de datos. Los datos no ocupan una posición determinada. NO se permiten elementos repetidos. Se usa para crear conjuntos de datos.

Queue. Es una estructura de datos en la que los elementos se manipulan siguiendo el esquema de una cola o esquema FIFO (First Input First Output) es decir, que el primer elemento que se almacena en la cola es el primer elemento que sale de ella. Se usa para crear cola de datos.

Cada uno de estos tres interfaces están implementado en una clase abstracta genérica de nombre **AbstractList**, **AbstractSet**, y **AbstractQueue** respectivamente.

Existen otros tipos de colecciones como las pilas (**Stack**) que son un tipo especial de lista en la que los elementos se manipulan siguiendo el esquema de una pila o esquema LIFO (Last Input First Output) es decir, que el último elemento que se almacena en la pila es el primer elemento que sale de ella.

Para crear una aplicación que use colecciones Java proporciona el marco de trabajo (Framework) Collections cuyas clases principales se encuentran en los paquetes **java.util** y **java.util.concurrent**.

ESTRUCTURAS DINÁMICAS: LISTAS. ARRAYLIST

Las listas de datos son un tipo de colección que se usa con mucha frecuencia. En Java podemos crear una lista usando la clase genérica **ArrayList** que permite crear una lista que almacene cualquier tipo de objetos.

La principal ventaja de los ArrayList frente a los arrays tradicionales es que el **tamaño** de los ArrayList varía dinámicamente en función de las necesidades del programa por lo que no se desperdicia memoria como pasa con los arrays tradicionales.

Para definir un ArrayList en Java debemos escribir ArrayList y entre <> el tipo de los objetos que va a contener el ArrayList. Si no se especifica el tipo entre <> se crea un **ArrayList** de objetos de tipo **Object**. Esto puede ser útil en casos dónde nos interese aprovechar el **polimorfismo** (que veremos más adelante).

Para poder usar la clase ArrayList en una aplicación Java debemos importarla. Para ello al comienzo de la clase en la que vamos a usar ArrayList escribimos

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

Si queremos crear un nuevo ArrayList de nombre arrayListString que contenga objetos de tipo String escribimos

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

Para crear un ArrayList de cualquiera de los tipos predefinidos del sistema (int, double, ...) debemos usar sus clases envolventes (Integer, Double, ...). Por ejemplo, si queremos crear un nuevo ArrayList de nombre arrayListInteger que contenga objetos de tipo Integer escribimos

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

Si queremos crear un nuevo ArrayList de nombre arrayListDouble que contenga objetos de tipo Double escribimos

```
ArrayList<Double> arrayListDouble = new ArrayList<Double>();
```

OPERACIONES CON LISTAS LINEALES. INSERCIÓN, BÚSQUEDA, BORRADO ...

Java nos proporciona una serie de métodos para facilitar la manipulación de objetos de tipo ArrayList. Éstos métodos funcionan igual sea cual sea el tipo de objeto que contiene el ArrayList ya que los ArrayList, al igual que los otros tipo de colecciones, son estructuras denominadas contenedoras.

Si partimos del ArrayList de String anterior de nombre arrayListString podemos usar los siguientes métodos

```
arrayListString.add("Elemento");  // Añade el elemento al ArrayList
arrayListString.add(n, "Elemento 2");  // Añade el elemento al ArrayList en la posición 'n'
arrayListString.size();  // Devuelve el número de elementos del ArrayList
arrayListString.get(n);  // Devuelve el elemento que está en la posición n del ArrayList
arrayListString.set(n, "NuevoValor");  // Cambia el valor del elemento que está en la posición n del ArrayList
arrayListString.contains("Elemento");  // Comprueba se existe del elemento ('Elemento') que se le pasa como parametro
arrayListString.indexOf("Elemento");  // Devuelve la posición de la primera ocurrencia ("Elemento") en el ArrayList
arrayListString.lastIndexOf("Elemento");  // Devuelve la posición de la última ocurrencia ("Elemento") en el ArrayList
arrayListString.remove(n);  // Borra el elemento de la posición n del ArrayList
arrayListString.remove("Elemento");  // Borra la primera vez que parece el "Elemento" que se le pasa como parametro.
arrayListString.clear();  //Borra todos los elementos de ArrayList
arrayListString.isEmpty();  // Devuelve true si el ArrayList esta vacio. Sino Devuelve false
ArrayList arrayListCopia = (ArrayList) arrayListString.clone();  // Crea una copia de un ArrayList
Object[] array = arrayListString.toArray();  // Convierte el ArrayList a un Array
```

RECORRIDO DE LISTAS LINEALES

Para recorrer un ArrayList podemos hacerlo del modo tradicional usando una repetitiva que recorra todas las posiciones hasta la última cuyo valor obtenemos con el método **size**.

Por ejemplo, para recorrer un ArrayList de nombre arrayListString escribimos

```
for (int posicion=0; posicion < arrayListString.size(); posicion++) {...}
```

Para recorrer cualquier tipo de colección, en este caso un ArrayList, Java proporciona una estructura repetitiva de nombre **for each**. La sintaxis para recorrer un ArrayList de tipo String y de nombre arrayListString usando una variable temporal del mismo tipo String y de nombre cadena es

```
for (String cadena : arrayListString) {...}
```

Además, Java proporciona otro método para recorrer los ArrayList. Este método utiliza un objeto de tipo **Iterator**. Los objetos de tipo Iterator nos permiten recorrer una colección sin saber cuál es el número de elementos que hay. Para poder usar un objeto Iterator debemos importar **java.util.Iterator**. Para ello utiliza los siguientes métodos

hasNext. Indica si hay algún elemento más en la colección o hemos llegado ya al final.
Next. Devuelve el valor del siguiente elemento de la colección.

Por ejemplo, si queremos crear un nuevo objeto Iterator para el ArrayList de nombre arrayListString que contiene objetos de tipo String escribimos

```
Iterator<String> it = arrayListString.iterator();
```

Al haber definido el objeto Iterator ahora podemos usar el siguiente código para recorrer el ArrayList de nombre arrayListString

```
String s="";
while(it.hasNext()){
 s = it.next();
...
}
```

EJERCICIOS CON LISTAS. ARRAYLIST

1. Crea la clase ArrayListCadenas que recibe Strings por teclado hasta que se introduce un String en blanco y los va almacenando en un ArrayList. Cuando finaliza la introducción muestra el contenido del array por pantalla.

2. Crea la clase ArrayListCadenasForeach que recibe Strings por teclado hasta que se introduce un String en blanco y los va almacenando en un ArrayList. Cuando finaliza la introducción muestra el contenido del array por pantalla usando for each.

3. Crea la clase ArrayListCadenasIterator que recibe Strings por teclado hasta que se introduce un String en blanco y los va almacenando en un ArrayList. Cuando finaliza la introducción muestra el contenido del array por pantalla usando un objeto de tipo Iterator.

4. Crea la clase ArrayListNumveces que pide números del 1 al 5 por pantalla y calcula el número de veces que han sido introducidos utilizando un ArrayList. Finaliza la introducción de datos cuando se introduzca 0, y muestra por pantalla el número de veces que se ha introducido cada número.

5. Crea la clase ArrayListMedia que pide números enteros por pantalla y los almacena en un ArrayList mientras que no se introduzca un número negativo. Después calcula la Media y la muestra.

6. Crea la clase ArrayListMediaTemperaturas que pide temperaturas por pantalla y las almacena en un ArrayList mientras que no se introduzca -999. Después calcula la temperatura media y muestra el número de temperaturas que son mayores que la media, iguales que la media, e inferiores a la media.

7. Crea la clase ArrayListCadenasMenu que muestra un menú por pantalla con las siguientes opciones para manipular Strings
   - Añadir String. Pide un String y lo añade al array.

- Buscar String. Pide un String y lo busca en el array. Muestra un mensaje con la posición en que se encuentra o un mensaje de error si no se encuentra en el array.
- Borrar String. Pide un String y lo elimina, si es que existe, del array.
- Listar Array. Muestra todos los elementos del array por pantalla, si es que tiene elementos.
- Salir. Realiza las operaciones necesarias para la correcta finalización del programa.

8. Crea la clase ArrayListEnterosMenu que muestra un menú por pantalla con las siguientes opciones para manipular datos de tipo Integer

- Añadir Entero. Pide un número entero y lo añade al array.
- Buscar Entero. Pide un número entero y lo busca en el array. Muestra un mensaje con la posición en que se encuentra o un mensaje de error si no se encuentra en el array.
- Borrar Entero. Pide un número entero y lo elimina, si es que existe, del array.
- Listar Array. Muestra todos los elementos del array por pantalla, si es que tiene elementos.
- Salir. Realiza las operaciones necesarias para la correcta finalización del programa.

LISTAS ORDENADAS

Para ordenar un ArrayList **de menor a mayor** podemos usar el método **sort** de la clase **Collections**. El método sort se basa en el método **compareTo** de la clase del ArrayList (String en este caso) para realizar la ordenación. Para poderlo usar lo debemos importar mediante la sentencia

```
import java.util.Collections;
```

Por ejemplo, para ordenar de menor a mayor un ArrayList de nombre arrayListString escribimos

```
Collections.sort(arrayListString);
```

Para ordenar un ArrayList **de mayor a menor** debemos usar un objeto de la clase **Comparator** al que se asigna el resultado de ordenar la colección usando el método **reverseOrder.** Este objeto se encargará de realizar la comparación de los objetos y de irlos ordenando de mayor a menor. El método **reverseOrder** se basa en el método **compareTo** de la clase del ArrayList (String en este caso) para realizar la ordenación invirtiendo el resultado, cuando compareTo devuelve un valor mayor que 0 reverseOrder devuelve un valor menor que 0 y cuando compareTo devuelve un valor menor que 0 reverseOrder devuelve un valor mayor que 0. Para poderlo usar lo debemos importar mediante la sentencia

```
import java.util.Comparator;
```

Por ejemplo, para ordenar de mayor a menor un ArrayList de nombre arrayListString escribimos

```
Comparator<String> comparador = Collections.reverseOrder();
Collections.sort(arrayListString, comparador);
```

Para ordenar un ArrayList de objetos de una clase que no tiene definida el método compareTo o cuando queremos ordenar por otro criterio que no sea el que usa compareTo debemos sobrescribir el método **compare** de la clase **Comparator**. Dentro del método compare escribiremos el código que realiza la comparación de los objetos.

Por ejemplo, si queremos comparar dos objetos de la clase Complejo tomando como base de la comparación sólo su parte real (por simplificar el proceso), dentro de la clase Complejo escribimos

```
Collections.sort(arrayListComplejos, new Comparator<Complejo>() {
 @Override
```

```
 public int compare(Complejo c1, Complejo c2) {
  return new Integer((c1.getReal()).compareTo(c2.getReal()));
 }
});
```

EJERCICIOS CON LISTAS ORDENADAS

9. Crea la clase ArrayListCadenasOrdenado que modifica la clase ArrayListCadenas para que las cadenas aparezcan por pantalla en orden ascendente (de menor a mayor).

10. Crea la clase ArrayListCadenasOrdenadoDescendente que modifica la clase ArrayListCadenasOrdenado para que las cadenas aparezcan por pantalla en orden descendente (de mayor a menor).

11. Crea la clase ArrayListDiccionario que muestra un menú por pantalla con las siguientes opciones
    - Añadir Palabra. Pide una palabra y la añade al array, si todavía no existe, en la posición que le corresponda alfabéticamente.
    - Buscar Palabra. Pide una palabra y la busca en el array. Si la encuentra la muestra y si no la encuentra muestra un mensaje de error.
    - Borrar Palabra. Pide una palabra y la elimina, si es que existe, del array. Si no la encuentra muestra un mensaje de error.
    - Listar Diccionario. Muestra todos los elementos del diccionario por pantalla, si es que tiene elementos, ordenados alfabéticamente.
    - Salir. Realiza las operaciones necesarias para la correcta finalización del programa.

12. Crea la clase ArrayListDiccionarioInvertido que modifica la clase ArrayListDiccionario para que ahora las palabras se añadan y se muestren en orden alfabético inverso (de mayor a menor).

13. Crea la clase ArrayListEnterosMenuOrdenado que muestra un menú por pantalla con las siguientes opciones para manipular datos de tipo Integer
    - Añadir Entero. Pide un número entero y lo añade al array, si todavía no existe, en la posición que le corresponda.
    - Buscar Entero. Pide un número entero y lo busca en el array. Si lo encuentra lo muestra y si no lo encuentra muestra un mensaje de error.
    - Borrar Entero. Pide un número entero y lo elimina, si es que existe, del array.
    - Listar Array. Muestra todos los elementos del array por pantalla, si es que tiene elementos, ordenados de menor a mayor.
    - Salir. Realiza las operaciones necesarias para la correcta finalización del programa.

14. Crea la clase ArrayListEnterosMenuInvertido que modifica la clase ArrayListEnterosMenu para que ahora los números se añadan y se muestren en orden descendente (de mayor a menor).

15. Crea la clase ArrayListComplejosMenuOrdenado que muestra un menú por pantalla con las siguientes opciones para manipular datos de tipo Complejo
    - Añadir Complejo. Pide un número Complejo y lo añade al array, si todavía no existe, en la posición que le corresponda.
    - Buscar Complejo. Pide un número Complejo y lo busca en el array. Si lo encuentra lo muestra y si no lo encuentra muestra un mensaje de error.
    - Borrar Complejo. Pide un número Complejo y lo elimina, si es que existe, del array.
    - Listar Array. Muestra todos los elementos del array por pantalla, si es que tiene elementos, ordenados de menor a mayor.
    - Salir. Realiza las operaciones necesarias para la correcta finalización del programa.

16. Crea la clase ArrayListComplejosMenuImaginaria que modifica la clase ArrayListComplejosMenuOrdenado para que ahora los datos de tipo Complejo se muestren en orden ascendente (de mayor a menor) en función de su parte imaginaria.

LISTAS CIRCULARES Y DOBLEMENTE ENLAZADAS.

En muchos lenguajes de programación, por ejemplo C y C++, existe la posibilidad de crear listas a las que se puede acceder al principio y al final para facilitar la inserción o el borrado de los datos e incluso alguno de los elementos de la lista puede contar con un enlace a su elemento anterior y otro a su elemento posterior.

En Java no hace falta utilizar ese tipo de estructuras ya que la clase ArrayList proporciona los métodos necesarios para trabajar de manera sencilla con listas.

COLAS

Una cola es un tipo especial de lista de tipo FIFO (First Input First Output, el primero que entra es el primero que sale) en el que el primer elemento que entra es el primero que sale. Un ejemplo típico de cola es la cola del cine.

En Java se puede implementar una cola derivando la clase ArrayList y adaptando los métodos necesarios para que el ArrayList se comporte como una cola.

PILAS. STACKS

Una pila es un tipo especial de lista de tipo LIFO (Last Input First Output, el último que entra es el primero que sale) en el que el último elemento que entra es el primero que sale. Un ejemplo típico de pila es una torre o pila de libros en la que el libro que podemos coger es siempre el que está encima de la pila.

En Java se puede implementar una pila derivando la clase ArrayList y adaptando los métodos necesarios para que el ArrayList se comporte como una pila.

EJERCICIOS CON PILAS Y COLAS.

17. Crea la clase ArrayListCadenasCola que muestra un menú por pantalla con las siguientes opciones
    - Ponerse a la cola. Pide un nombre y lo añade al array en la última posición.
    - Coger Entrada. Muestra el nombre del primer cliente y lo elimina de la cola, si es que el array no está vacío.
    - Listar Cola. Muestra todos los nombres de los clientes de la cola por pantalla, si es que tiene.
    - Salir. Realiza las operaciones necesarias para la correcta finalización del programa.

18. Crea la clase ArrayListCadenasPila que muestra un menú por pantalla con las siguientes opciones
    - Poner Libro. Pide un título de libro y lo añade al array en la última posición.
    - Coger Libro. Muestra el título del último libro y lo elimina de la pila, si es que el array no está vacío.
    - Listar Pila. Muestra todos los títulos de los libros de la pila por pantalla, si es que tiene.
    - Salir. Realiza las operaciones necesarias para la correcta finalización del programa.