Capítulo 5

Almacenamiento de la información en estructuras de datos dinámicas (ArrayList)

5.1. Resultados de aprendizaje

6. Escribe programas que manipulen información, seleccionando y utilizando tipos avanzados de datos.

b) Se han reconocido las librerías de clases relacionadas con tipos de datos avanzados.

c) Se han utilizado listas para almacenar y procesar información.

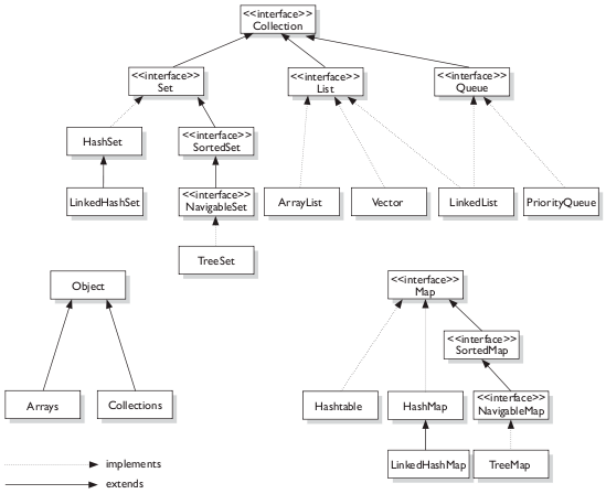
d) Se han utilizado iteradores para recorrer los elementos de las listas.

e) Se han reconocido las características y ventajas de cada una de la colecciones de datos disponibles.

5.2. Colecciones

Cuando se necesitan características más sofisticadas para almacenar datos que las que proporciona un array, Java pone a disposición del programador el interface Collection.

El siguiente diagrama muestra la jerarquía de clases que integran las colecciones Java.



*Una **interfaz** en Java es una colección de métodos abstractos y propiedades constantes. En las interfaces se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describan la lógica del comportamiento de los métodos*

- La interface **Collection** comparte un grupo de objetos desordenados y permite duplicados.
- La interface **Set** extiende a Collection, pero prohíbe la presencia de duplicados.
- La interface **List** extiende a Collection, permite duplicados e introduce un índice posicional.
- La interface **Map** es independiente de Set y Collection, y está orientada a su uso con parejas de tipo clave-valor.

http://www.androidcurso.com/index.php/tutoriales-java-esencial/462-las-colecciones-en-java

5.2.1. ArrayList

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

Los ArrayList son de todas las colecciones de Java la más usada.

Un ArrayList es un array dinámico. No tiene restricciones de capacidad. Su tamaño se ajusta de forma dinámica.

La clase ArrayList dispone de varios constructores:

El operador se denomina operador diamante

ArrayList()

Construye un arraylist con capacidad para 10 elementos. Su número de elementos (size() es cero).

```
1 ArrayList<Integer> lista = new ArrayList();
2 System.out.println(lista.size()); // 0
```

ArrayList(Collection c)

Construye un arraylist a partir de otra colección.

ArrayList(int initialCapacity)

Construye un arraylist indicando la capacidad inicial.

Los elementos dentro de un ArrayList son **objetos**.

La clase ArrayList forma parte del paquete java.util

Para añadir un elemento a esta estructura, usamos el método **add** y para recoger un elemento usamos el método **get**.

```
1  // instanciamos un nuevo ArrayList formado por objetos de tipo String
2
3  ArrayList<String> miLista = new ArrayList<String>();
4  List<String> miListaDos = new ArrayList();
5
6  // Objetos de tipo Object
7  ArrayList miLista = new ArrayList();
8
9  // agregamos el elemento por el final
10 // si la lista está vacía será el primer elemento
11
12 miLista.add("elemento 1");
```

También se pueden insertar elementos en posiciones concretas, desplazando parte de los elementos.

```
1  // Instanciamos un nuevo ArrayList
2  ArrayList<String> miLista = new ArrayList();
3  miLista.add("e1");
4  miLista.add("e2");
5  miLista.add("e3");
6
7  // agregamos un elementos en la posición uno
8  miLista.add(1, "elemento A");
9
10 //e1, elementoA, e2,e3
```

Para eliminar elementos se dispone del método **remove()** que nos permitirá borrar por contenido o por posición.

Otras métodos importantes son:

Para determinar la cantidad de elementos de la lista solo se llama al método **size()**.

Para vaciar la lista y dejarla sin ningún elemento en ella se usa el método **clear()**.

Para saber si un objeto ésta o no en la lista usamos el método **contains()** que devuelve true si existe o false si sucede lo contrario.

Para saber si la lista está **vacía**, es decir, si no tiene elementos usamos **isEmpty()** que devuelve true si NO hay elementos o false si contiene algún elemento.

Para pasar o copiar los elementos a un array de objetos (TipoObjeto objetos []) se usa el método **toArray()**.

El uso de **indexOf(Object o)** retorna la posición de un elemento que está en el array o -1 sino está.

El método **addAll(Collection c)** se encarga de anexar un arraylist a otro ya creado.

```
1  // Declaración de un ArrayList de "String". Puede ser de cualquier
       otro Elemento u Objeto (float, Boolean, Object, ...)
2  ArrayList<String> nombreArrayList = new ArrayList<String>();
3
4  // Añade el elemento al ArrayList
5  nombreArrayList.add("Elemento");
6
7  // Añade el elemento al ArrayList en la posición 'n'
8  nombreArrayList.add(n, "Elemento 2");
9
10 // Devuelve el numero de elementos del ArrayList
11 nombreArrayList.size();
12
13 // Devuelve el elemento que esta en la posición '2' del ArrayList
14 nombreArrayList.get(2);
15
16 // Comprueba se existe del elemento ('Elemento') que se le pasa como
       parametro
17 nombreArrayList.contains("Elemento");
18
19 // Devuelve la posición de la primera ocurrencia ('Elemento') en el
       ArrayList
20 nombreArrayList.indexOf("Elemento");
21
22 // Devuelve la posición de la última ocurrencia ('Elemento') en el
       ArrayList
23 nombreArrayList.lastIndexOf("Elemento");
24
25 // Borra el elemento de la posición '5' del ArrayList
26 nombreArrayList.remove(5);
```

```
27
28  // Borra la primera ocurrencia del 'Elemento' que se le pasa como
        parametro.
29  nombreArrayList.remove("Elemento");
30
31  //Borra todos los elementos de ArrayList
32  nombreArrayList.clear();
33
34  // Devuelve True si el ArrayList esta vacio. Sino Devuelve False
35  nombreArrayList.isEmpty();
36
37  // Copiar un ArrayList
38  ArrayList arrayListCopia = (ArrayList) nombreArrayList.clone();
39
40  // Pasa el ArrayList a un Array
41  Object[] array = nombreArrayList.toArray();
```

5.3. ArrayList y repetitivas

Para llevar a cabo acciones con cada uno de los elementos de un arrayList, además de las repetitivas vistas hasta ahora también existen otras clases y métodos.

Los **Iteradores** sirven para recorrer los ArrayList y poder trabajar con ellos. Los Iteradores solo tienen tres métodos que son el **hasNext()** para comprobar que siguen quedando elementos en el iterador, el **next()** para que nos de el siguiente elemento; y el **remove()** que sirve para eliminar el elemento del Iterador.

La interfaz Iterator, tal y como hemos visto en el tema anterior, también se puede usar con los arrays.

```
1   // Declaración el ArrayList
2   ArrayList<String> nombreArrayList = new ArrayList();
3
4   // Añadimos 10 Elementos en el ArrayList
5   for (int i=1; i<=10; i++){
6     nombreArrayList.add("Elemento "+i);
7   }
8
9   // Añadimos un nuevo elemento al ArrayList en la posición 2
10  nombreArrayList.add(2, "Elemento 3");
11
12  // Declaramos el Iterador e imprimimos los Elementos del ArrayList
13  Iterator<String> nombreIterator = nombreArrayList.iterator();
14  while(nombreIterator.hasNext()){
15    String elemento = nombreIterator.next();
16    System.out.print(elemento+" / ");
17  }
```

También se pueden usar los for mejorados.

```
List<String> lista = new ArrayList<>();
lista.add("A");
lista.add("B");
lista.add("C");

for(String elemento : lista){
  System.out.println(elemento);
}
```

5.3.1. Método forEach

```
    ArrayList<Integer> lista = new ArrayList();
    lista.add(3);
    lista.add(4);
    // For "normal"
    for (int x = 0; x < lista.size(); x++)
        System.out.println(lista.get(x));

    // For mejorado
    for(int elemento : lista)
        System.out.println(elemento);

    // forEach
    lista.stream().forEach((elemento)->
        System.out.println(elemento));
```

El foreach y stream() se explican en el tema de las ultimas novedades