Colecciones

Índice:

1. Arrays redimensionables: ArrayList.

La clase ArrayList es una implementación de la interfaz List y permite que usemos arrays redimensionables en nuestras aplicaciones. Estos arrays, si no indicamos lo contrario, parten con una capacidad de 10 elementos y van creciendo automáticamente a medida que se supera ese tamaño.

Algunas características:

- Puede almacenar un número indefinido de elementos.
- Cada elemento se almacena en una posición del array.
- Permite valores repetidos
- Esta clase está en el paquete java.utils.
- Declaración:
    - ArrayList <clase> identificador-lista;
    - List<clase> identificador-lista;

- Inicialización:
    - ArrayList <Vehiculo> vehiculos = new ArrayList <>();
    - List<Vehiculo> vehiculos = new ArrayList<>();

Métodos más comunes:

- add (Object o). añade un elemento a la lista.
  vehiculos.add (new Vehiculo(“2040 GHT”,”VW”,”GOLF”,”ROJO”, 90.2, false));

- addAll(int index, Colección). Añade al arrayList todos los elementos de la colección a partir de la posición indicada en index. Los elementos existentes se desplazan a la derecha.

- get (int posicion). Extrae el elemento almacenado en la posición indicada. La primera posición es la cero.
  v = vehiculos.get (0);

- size(). Devuelve el número de elementos de la lista.
  int totalVehiculos = vehiculos.size();

- remove(posición). Elimina de la lista el objeto situado en la posición indicada. Ojo, no podemos recorrer un arrayList e ir eliminando varios objetos del mismo, esto genera una excepción de la clase ConcurrentModificationException. Para realizar este tipo de eliminaciones es necesario usar la interfaz Iterator como se explica en el punto 1.1.

- clear(). Vaciar el arrayList.

- isEmpty(). Devuelve true si el arrayList está vacío

- set(posicion, objeto). Coloca el objeto en la posición indicada, sustituyendo el objeto existente por el nuevo. La posición debe contener un objeto.

- trimToSize(). Ajusta el tamaño el arrayList al número de objetos que tiene (size).

- toArray(T[] array). Devuelve un array con el contenido del del arrayList.

```
ArrayList<Integer> b = new ArrayList<>();
b.add(12);b.add(13);b.add(14);
Integer [] c = new Integer[b.size()];
c = b.toArray(c);
```

- Podemos recorrerlo con un for tradicional o con un for-each.

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

a.add(0);
a.add(10);
a.add(15);
a.add(20);

System.out.println("Contenido del array");
for (Integer integer : a) {
    System.out.println(integer);
}
```

1.1. Recorrer un arrayList mediante Iterator

Mediante la interfaz Iterator pueden recorrerse colecciones de objetos para ser tratados de uno en uno. Pueden eliminarse objetos del iterador y el resultado se refleja en el ArrayList. De esta forma pueden evitarse errores en el tratamiento de colecciones.

Los métodos más comunes de un iterador son:

- hasNext(). Devuelve true si hay un elemento siguiente en la colección.
- next(). Devuelve el siguiente elemento. Lanza una excepción si no hay más elementos.
- remove(). Elimina el objeto actual.

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

a.add(11);a.add(10);a.add(15);a.add(20);a.add(17);

System.out.println("Contenido del arrayList antes");
System.out.println("*****************************");
for (Integer integer : a) {
    System.out.println(integer);
}

//eliminar los valores impares del arrayList
Iterator iterator = a.iterator();
while(iterator.hasNext()) {
    int valor = (int) iterator.next();
    if (valor%2==1)
        iterator.remove();
}

System.out.println("Contenido del arrayList después");
System.out.println("******************************");
for (Integer integer : a) {
    System.out.println(integer);
}
```

1.2. Ordenar un arrayList.

- **Ordenar un arrayList de objetos de clases envoltorio (wrappers), es decir, clases asociadas a los tipos primitivos (Integer, Double, …).**
    - Mediante la clase Collections: Collections.sort(nombreArrayList);
    - Mediante la clase Comparator.

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

a.add(11);a.add(10);a.add(15);a.add(20);a.add(17);

System.out.println("Contenido del arrayList antes");
System.out.println("*****************************");
for (Integer integer : a) {
    System.out.println(integer);
}

a.sort(new Comparator<Integer>() {
    public int compare(Integer o1, Integer o2) {
        return o1.compareTo(o2);
    }
});

System.out.println("Contenido del arrayList después");
System.out.println("*****************************");
for (Integer integer : a) {
    System.out.println(integer);
}
```

- **Ordenar un ArrayList de clases definidas por el usuario mediante el uso de la interfaz Comparator.**

    - En este caso al llamar a Collections.sort(), debemos pasar como parámetro el arrayList y además la forma de ordenarlo. Esta forma de ordenarlo consiste en crear un objeto Comparator parametrizado con la clase Persona, que es la clase base del arrayList. Como Comparator es una interface, nos obliga a implementar el método compare, y ahí es donde definimos qué atributo de la clase sirve para ordenar, en nuestro caso la edad.
    - El método compareTo, recibe dos objetos Persona y los compara por su edad, devolviendo 1, 0 ó -1 según el primero sea mayor, sean iguales o el segundo sea mayor respectivamente.

```java
ArrayList<Persona> personas = new ArrayList<>();
personas.add(new Persona(3, "Emi", 23));
personas.add(new Persona(1, "Elena", 11));
personas.add(new Persona(2, "Sara", 19));

System.out.println("Datos antes de ordenar");
System.out.println("**********************");
for (Persona persona : personas) {
    System.out.println(persona);
}
Collections.sort(personas, new Comparator<Persona>() {
    @Override
    public int compare(Persona o1, Persona o2) {
        return o1.getEdad().compareTo(o2.getEdad());
        //dev 1 si o1.getEdad() es mayor
        //0 si son iguales
        //-1 si o2.getEdad() es mayor
    }
});

System.out.println("Datos después de ordenar");
System.out.println("************************");
for (Persona persona : personas) {
    System.out.println(persona);
}
```

2. Arrays asociativos: HashMap.

HashMap es una clase que implementa la interfaz Map. Se utiliza para crear arrays asociativos, que son estructuras formadas por parejas del tipo (clave, valor). Dicho de otra forma, es una colección donde los objetos se almacenan asociados a un valor y no a una posición indicada por un índice numérico. Imagina que queremos localizar clientes por su DNI, países por su código, artículos por su referencia…

- Un HashMap permite guardar null tanto en la clave como en el valor.
- No garantiza el orden de los elementos.

En Java, HashMap es una clase y su funcionamiento es como sigue:

- Declaración: HashMap <ClaseClave, ClaseValor> nombreHash = new HashMap<>();

  HashMap <String, String> miHash = new HashMap<>();
  Map<String, String> miHash = new HashMap<>();

- Object put (Object key, Object valor). Si el key existe en la colección, el objeto se sustituye por el nuevo, sino se añade.
  miHash.put(“33978833S”, “José Luis”);

- boolean containsKey(Object key). Devuelve verdadero si existe la clave en la colección.
  miHash.containsKey(“33978833S”)

- Object get (Object key). Devuelve el valor asociado a la key o null si la key no existe.
  String nombre=miHash.get(“33978833S”);

- Object remove(Object key). Si existe, elimina la pareja y devuelve el valor asociado a la clave eliminada, en caso contrario devuelve null.
  String cadena=miHash.remove(“33978833S”);

- Object remove(Object key, Object o). Si existe el objeto lo elimina y retorna lo eliminado. Si no existe retorna null.

- int size(). Devuelve el número de elementos de la colección.

- boolean containsValue(Object valor). Devuelve true o false, según el valor exista o no en la colección.
  miHash.containsValue(“12341234”);

- Recorrer la estructura mediante un iterador (Iterator).

```
HashMap<String,String> datos = new HashMap<>();

datos.put("1", "Sara");
datos.put("2", "Vicente");
datos.put("3", "Alex");

Iterator it = datos.keySet().iterator();
while(it.hasNext()) {
    String clave = (String) it.next();
    String valor = datos.get(clave);
    System.out.printf("\nClave: %s, Valor: %s",clave, valor);
}
```

- Recorrer mediante un for-each

```
HashMap<String,String> datos = new HashMap<>();

datos.put("1", "Sara");
datos.put("2", "Vicente");
datos.put("3", "Alex");

for (String clave : datos.keySet()) {
    String valor = datos.get(clave);
    System.out.printf("\nClave: %s, Valor: %s",clave, valor);
}
```

2.1. Ordenar un HashMap.

- ordenar por clave keys. Con esta opción podemos ordenar las claves extraídas previamente y guardadas en un ArrayList o array. El arraylist es ordenado mediante sort y hecho esto, accederemos a los valores (get) a partir de esas keys ordenadas.
- ordenar por valor. Este proceso es más complejo.
    - Obtener una lista (ArrayList) con las parejas clave valor del HashMap.
    - Ordenar esa lista mediante sort y un Comparator.
    - Guardar los objetos de esa lista en un LinkedHashMap. Funciona igual que HashMap pero mantiene el orden en que se han guardado sus elementos.
    - Recorrer el LinkedHashMap y mostrar.
    - **En realidad no ordenamos el HashMap sino una estructura paralela.**

```
HashMap<String, Double> notas = new HashMap<>();
notas.put("FOL", 4.5);
notas.put("BBDD", 5.2);
notas.put("SSII", 4.4);
notas.put("Programación", 7.3);

ArrayList<Map.Entry<String, Double>> list =
        new ArrayList<Map.Entry<String, Double> >(notas.entrySet());

// Sort the list
Collections.sort(list, new Comparator<Map.Entry<String, Double>>() {
    public int compare(Map.Entry<String, Double> o1,
                       Map.Entry<String, Double> o2)
    {
        return (o1.getValue()).compareTo(o2.getValue());
    }
});

//guardamos los datos en un LinkedHashMap que si conserva el orden
//en el que se guardan lo datos
HashMap<String, Double> temporal = new LinkedHashMap<String, Double>();
for (Map.Entry<String, Double> aa : list) {
    temporal.put(aa.getKey(), aa.getValue());
}
//mostrar
for (String key : temporal.keySet()) {
    System.out.println(temporal.get(key));
}
```

3. HashSet

Esta clase es una implementación de la interfaz Set, que se utiliza para almacenar colecciones sin valores repetidos (métodos equals() y hashCode()). No garantiza el orden de los elementos.

Los métodos son los mismos que los usados en la clase ArrayList: add, addAll, clear, contains, remove, size, toArray. En este caso no existe el método get(posicion), ya que no se trata de un array.

```
Set<Persona> personas = new HashSet<>();

personas.add(new Persona(1,"Elena", LocalDate.of(2009, 11, 02)));
personas.add(new Persona(2,"Carmen", LocalDate.of(2009, 1, 12)));
personas.add(new Persona(3,"Dani", LocalDate.of(2008, 1, 1)));
personas.add(new Persona(3,"Dani", LocalDate.of(2008, 1, 1)));

for (Persona persona : personas) {
    System.out.println(persona);
}
```

```
Persona [id=2, nombre=Carmen, fechaNac=2009-01-12]
Persona [id=3, nombre=Dani, fechaNac=2008-01-01]
Persona [id=1, nombre=Elena, fechaNac=2009-11-02]
```