

Tema: Java collection Framework

Integrantes:

Mendez Castro Giusseppe Abraham

Granados Henrich

Rojas Alexis

Camavilca Gonzalez Gabriel

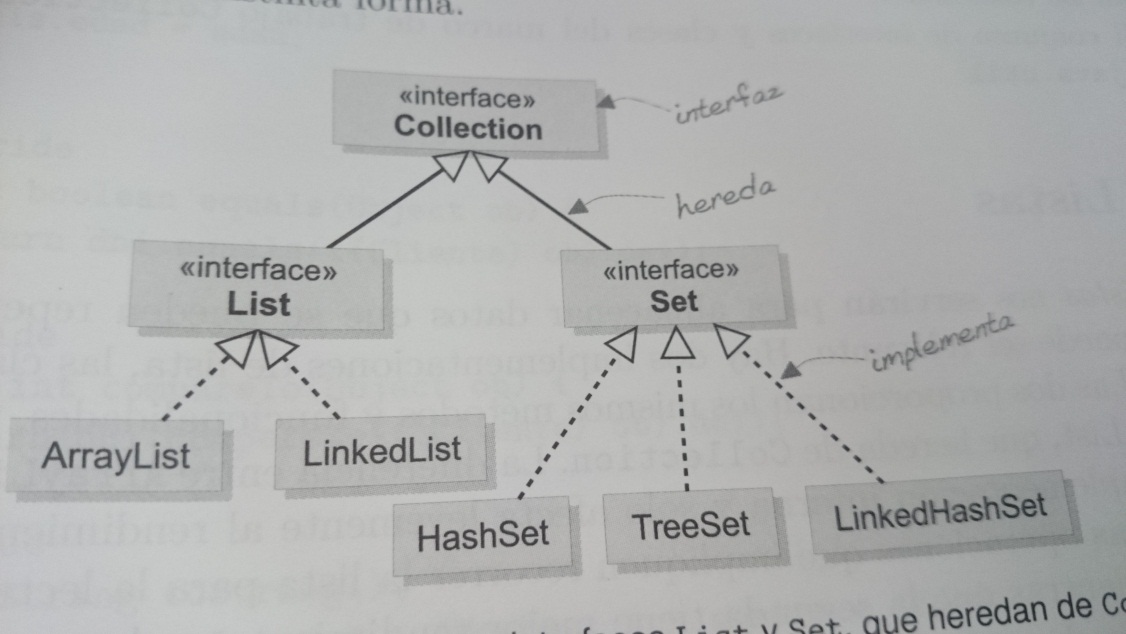
Sede: Ate

Turno tarde

2017 102ª

**¿QUE ES UN FRAMEWORK?**

**Un *framework,* entorno de trabajo**[**1**](https://es.wikipedia.org/wiki/Framework)**​ o marco de trabajo**[**2**](https://es.wikipedia.org/wiki/Framework)**​ es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.**

****

**Hay tres tipos fundamentales de estructuras ligadas a la interfaz *Collection,***

* **Listas: responden a la necesidad de manejar sucesiones de datos que pueden estar repetidos y cuyo orden puede ser importante.**
* **Conjuntos: el orden de los datos no es relevante y lo que realmente importa es la mera pertenencia, o no, de un dato a la estructura, con lo que las repeticiones tampoco tienen sentido.**
* **Mapas o diccionarios: relacionados con la interfaz *Collection,* aunque no la implementan. Sirven para guardar datos identificados por claves que no se repiten.**

**Todas estructuras son dinamicas, ya que permiten añadir y quitar unidades de informacion (objetos llamados nodos o elementos) en tiempo de ejecucion, cambiando el tamaño total sin tener que volverlas a declarar. Tambien, llamaremos colecciones a todas las clases que implementen la interfaz *Collection*, aunque tambien usaremos el nombre de la clase implementada (*ArrayList, etc).***

**Una particularidad de *Collection* es que trabaja con datos de tipo generico. Esto quiere decir que cuando declaremos una de estas estructuras, especificaremos la clase de objetos que se pueden insertar en ellas, lo que permite hacer un control de tipo mas eficiento.**

El conjunto de interfaces y clases del marco de trabajo *Collection se halla en el paquete java.util*

*1.1 Listas*

Las *listas* nos serviran para almacenar datos que se pueden repetir y cuyo orden de insercion puede ser relevante. Hay dos implementaciones de lista, las clases *ArrayList* y *LinkedList.* La diferencia entre *ArrayList y LinkedList* radica en la implementacion interna y solo afecta levemente al rendimiento.

En todo el codigo que vamos a escribir, podemos sustituir *ArrayList* por *LinkedList* sin alterar ningun resultado.

La sintaxis general para construir un *ArrayList* con un tipo generico de datos E ( sea Cliente, Empleado, Integer…) es,

ArrayList <E> lista = new ArrayList<>();

E debe ser siempre el nombre de una clase, nunca un tipo primitivo.

La declaracion de un objeto de la clase *ArrayList* que nos sirva para guardar objetos de tipo Cliente, seria:

ArrayList<Cliente> clie1 = new ArrayList<Cliente>();

1.1.2 Métodos básicos de la interfaz *Collection*

Los métodos de la interfaz *Collection* son de dos tipos. Unos afectan a elementos individuales y otros agrupos de elementos. A los primeros los llamaremos métodos básicos y a los segundos métodos globales.

Metodos de inserción

Son aquellos que sirven para añadir elementos nuevos en una colección.

Boolean add(E elem): sirve para insertar un elemento en una lista. Se le pasa el objeto a insertar. Si la insercion tiene éxito, devuelve true. En caso contrario, false. El nuevo nodo queda insertado al final de la lista. E es el tipo generico con que se ha declarado, en nuestro caso, Cliente.

Probemos con nuestra lista de clientes,

Cliente cliente = new Cliente(«111», «Marta», 20);

clie1.add(cliente);

Metodos de eliminacion

boolean remove(Object ob): elimina un nodo ob de una lista. Devuelve true si la elimina en caso contrario false. No se exige a ob que sea del tipo generico E con el que se ha declarado, ya que no va a añadir ningun nodo, por tanto queda cerrado de cualquier objeto no permitido. Sintaxis a continuacion;

clie1.remove(cliente);

Metodos de comprobacion

Insertemos algunos nodos para seguir experimentando,

clie1.add(new Cliente(«111», «Marta», 20));

clie1.add(new Cliente(«115», «Jorge», 21));

clie1.add(new Cliente(«112», «Carlos», 18));

Int size(): nos permite saber en cada momento el numero de elementos (o nodos) insertados en una lista. Por ejemplos

clie1.size() //devuelve 3

Boolean isEmpty(): permite saber si una lista esta vacia. Devuelve true si la lista esta vacía y false en caso contrario. La expresión,

clie1.isEmpty() //devolverá false

Otros metodos

String toString(): devuelve una cadena que representa la colección. Todas las colecciones, tienen el metooo implementado. Cada nodo se muestra, a su vez,según la implementacion toString() que tenga la clase generica E. Por tanto los elementos de clie1,

System.out.printIn(clie1);

Con nuestra implementacion de Cliente, obtendriamos por pantalla,

[Dni: 111 Nombre: Marta Edad: 20

, Dni: 115 Nombre: Jorge Edad: 21

, Dni: 112 Nombre: Carlos Edad: 18

]

Para recorrer una lista nodo por nodo, se utiliza un iterator. El metodo iterator(), invocado por la lista nos devuelve un iterador asociado a ella,

Iterator iterator()

Por ejemplo, si queremos recorrer nuestra lista de clientes, creamos el iterador con la sentencia,

iterator it = clie1.iterator();

1.1.3 Metodos globales de la interfaz *Collection*

Existen otros metodos, llamados metodos globales, en los que intervienen otras colecciones.

Boolean conteinsAll(Collection <?> c): La expresion <?> significa que la colección c que se pasa como parametro de un tipo generico cualquiera (de ahí el signo de interrogacion ?, que se conoce como*comodin).* Devuelve true si es o false si no sale.

Para ilustrarlo vamos a crear una segunda lista de objetos Cliente,

ArrayList<Cliente> clie2 = new ArrayList<>();

clie2.add(new Cliente(«111», «Marta», 20));

clie2.add(new Cliente(«112», «Carlos», 18));

Boolean addAll(Collection <? Extends E> c):

En nuestro caso, como clie1 se ha declarado como una lista del tipo generico Cliente (eso seria E), podemos pasarle como parametro cualquier colección que sea del tipo Cliente.

clie1.addAll(clie2);

Si mostramos clie1 tal como ha quedado se obtiene,

[Dni: 111 Nombre: Marta Edad: 20

, Dni: 112 Nombre: Carlos Edad: 18

, Dni: 211 Nombre: Ana Edad: 19

, Dni: 111 Nombre: Marta Edad: 20

, Dni: 112 Nombre: Carlos Edad: 18

]

**1.2 Interfaz Set**

Todos sus metodos los hereda de ***COLLECTION.*** Lo unico que añade es la restriccion de no permitir duplicados. Esto significa que si intentamos insertar un nodo que ya existe, no lo hara.

El conjunto de metodos disponibles es el mismo que vimos en los apartados de metodos basicos y globales de las colecciones:

**int size()**

**boolean isEmpty()**

**boolean contains(Object element)**

**boolean add(E element)**

**boolean remove(Object element)**

**iterator<E> iterator()**

**boolean containsAll(Collection<?> c)**

**boolean addAll(Collection<? Extends E> c)**

Implementacion Set:

* **HashSet:** tiene un buen rendimiento, aunque no garantiza ningun orden en la insercion.
* **TreeSet:** El criterio de ordenacion se lo proporciona un comparador en el constructor el proporcionado por el metodo **compareTo()**
* **LinkedHashSet:** garantiza el orden basado en la insercion, ya que siempre inserta los nodos al final.

**La interfaz Map**

Un Map (correspondencia) es un objeto que asocia una clave a un valor. También se denomina Diccionario.

Métodos para añadir y borrar:

" **put(Object key, Object value)**

" **remove(Object key)**

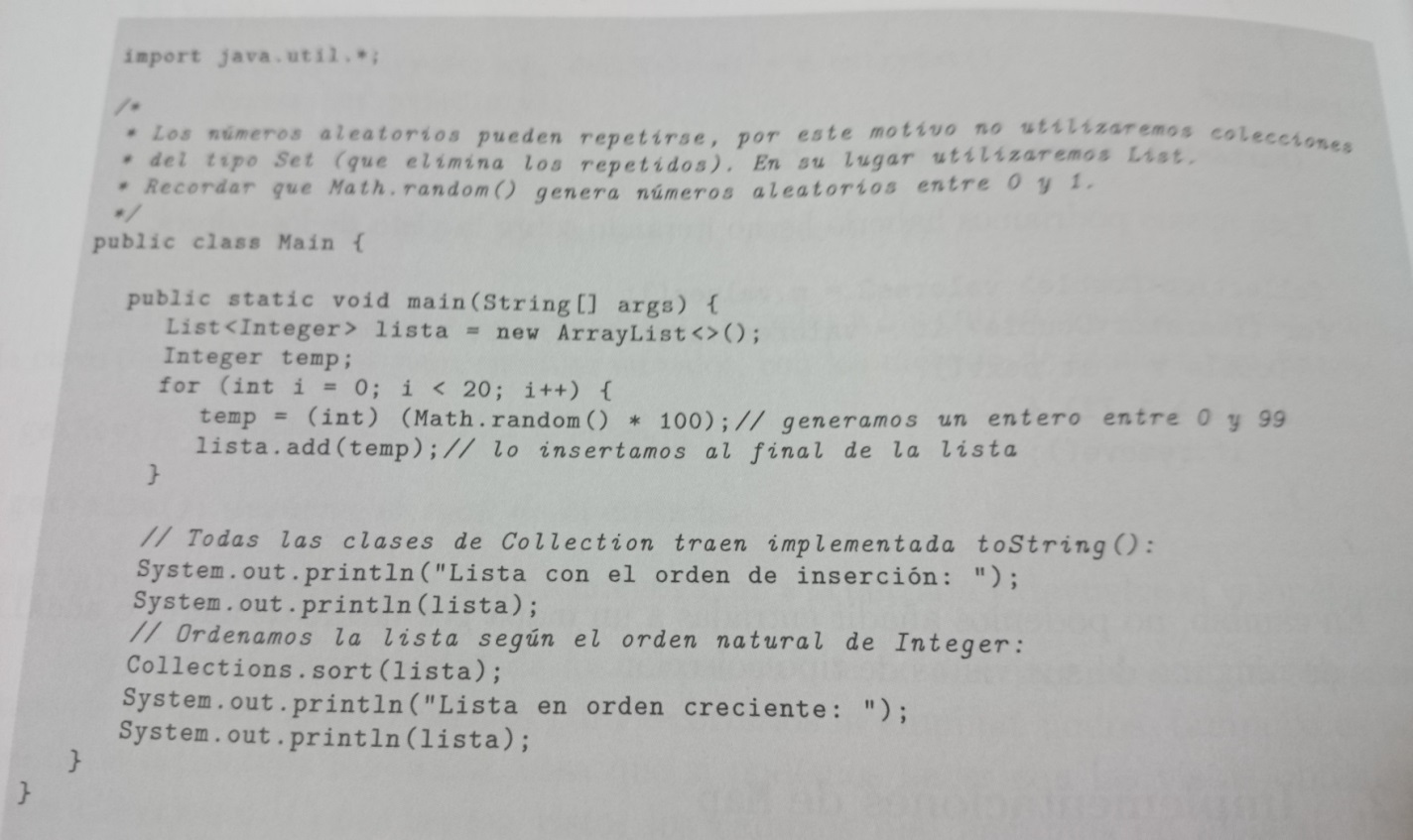
Métodos para la extracción de objetos

" **get(Object key)**

Métodos para obtener las claves, los valores y las parejas (clave,valor) como un conjunto

* " **keySet() // devuelve un set**
* " **values() // devuelve un collection**
* " **entrySet() // devuelve un set**

Ejercicios de Collections

1.Crear una colección de 20 numeros enteros aleatorios menores que 100, y guardalos en el orden en que se vayan generando; mostrar por pantalla dicha lista una vez creada, Ordenarla en sentido creciente y volverla a mostrar por pantalla.