

Asignatura

Programación



UNIVERSAE
Instituto Superior de FP

Asignatura

Programación

UNIDAD 9

Colecciones y tipos abstractos de datos

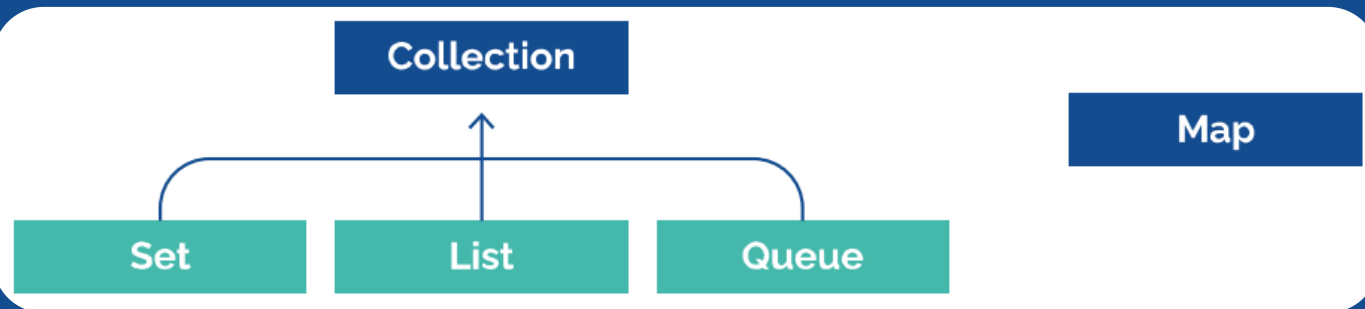


UNIVERSAE
Instituto Superior de FP

Las colecciones

¿Que hacer cuando tenemos muchos datos? Uso de colecciones

- Son estructuras dinámicas
- Agrupan elementos de una misma tipología
- Su tamaño puede variar en tiempo de ejecución
- Contienen métodos específicos:
 - ❑ INSERCIÓN
 - ❑ ELIMINACIÓN
 - ❑ ORDENACIÓN



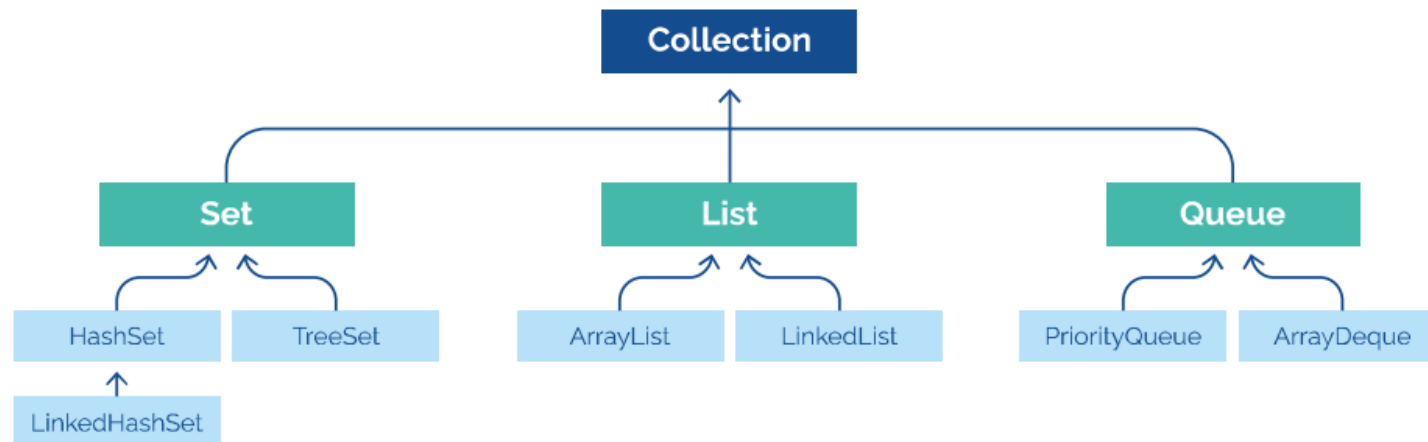
Interfaz Collection



Características

- Es la interfaz base de las colecciones
- Los elementos se identifican por si mismo
- Pueden estar duplicados o no dependiendo del tipo
- Pueden estar ordenados o no dependiendo del tipo


Método	Descripción
<code>int size()</code>	Devuelve el número de elementos contenido en la colección.
<code>boolean isEmpty()</code>	Devuelve true si no hay elementos. false si hay elementos
<code>void clear()</code>	Borra todos los elementos
<code>boolean contains(E elemento)</code>	Devuelve true si existe el elemento, false en caso contrario
<code>boolean add(E elemento)</code>	Añade el elemento. Devuelve true si ha sido añadido
<code>boolean remove(E elemento)</code>	Elimina el elemento. Devuelve true si lo ha borrado.
<code>Iterator<E> iterator()</code>	Devuelve un iterador de la colección para recorrerla



Set. HashSet

Características

- Uso de una tabla hash
- No admite valores duplicados
- Permite valores nulos
- Los elementos no están ordenados
- Rendimiento bastante bueno



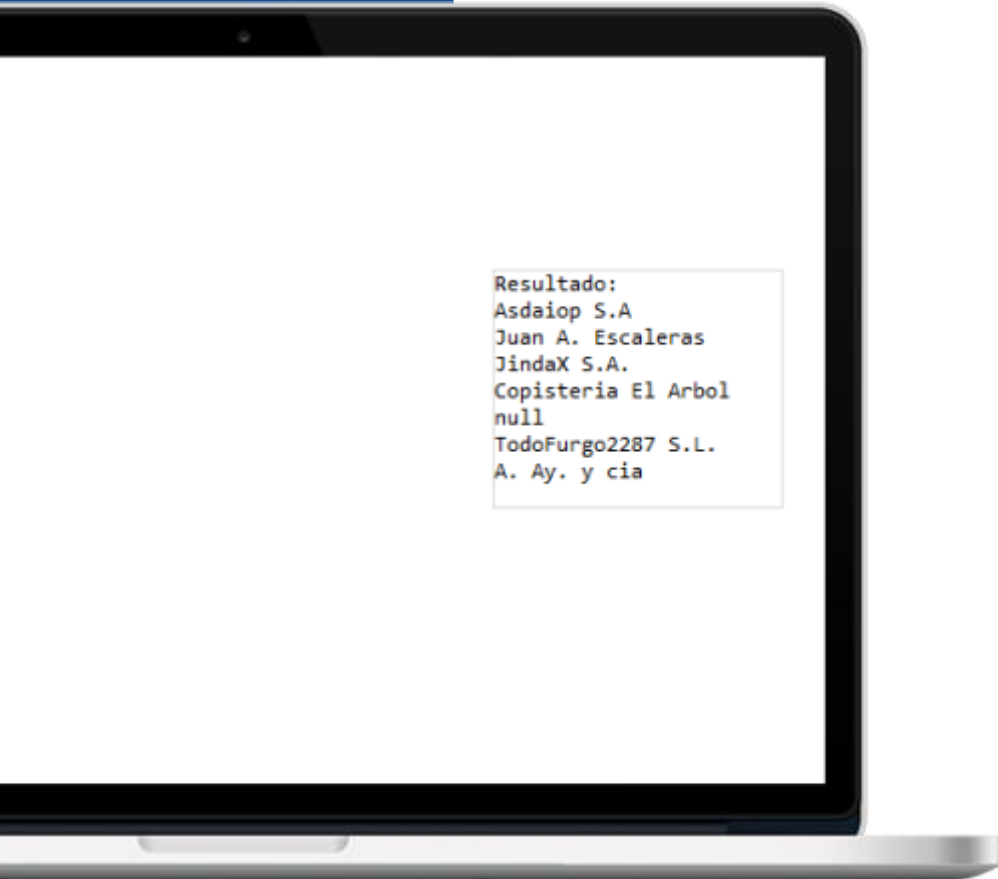
```
Resultado:  
null  
TodoFurgo2287 S.L.  
Juan A. Escaleras  
JindaX S.A.  
A. Ay. y cia  
Copisteria El Arbol  
Asdaiop S.A
```

```
HashSet<String> clientes = new HashSet<>();  
clientes.add("Asdaiop S.A");  
clientes.add("Juan A. Escaleras");  
clientes.add("JindaX S.A.");  
clientes.add("Copisteria El Arbol");  
clientes.add(null);  
clientes.add("TodoFurgo2287 S.L.");  
clientes.add("JindaX S.A.");  
clientes.add("A. Ay. y cia");  
  
Iterator<String> iterador = clientes.iterator();  
  
System.out.println("Resultado:");  
  
while(iterador.hasNext()) {  
    System.out.println(iterador.next());  
}
```

Set. LinkedHashSet

Características

- Uso de una tabla hash
- Contiene enlaces al elemento siguiente y anterior
- No admite valores duplicados
- Permite valores nulos
- Los elementos están ordenados según se insertan
- Rendimiento es inferior a HashSet



```
Resultado:  
Asdaiop S.A  
Juan A. Escaleras  
JindaX S.A.  
Copisteria El Arbol  
null  
TodoFurgo2287 S.L.  
A. Ay. y cia
```

```
LinkedHashSet<String> clientes = new LinkedHashSet<>();  
clientes.add("Asdaiop S.A");  
clientes.add("Juan A. Escaleras");  
clientes.add("JindaX S.A.");  
clientes.add("Copisteria El Arbol");  
clientes.add(null);  
clientes.add("TodoFurgo2287 S.L.");  
clientes.add("JindaX S.A.");  
clientes.add("A. Ay. y cia");  
  
Iterator<String> iterador = clientes.iterator();  
  
System.out.println("Resultado:");  
  
while(iterador.hasNext()) {  
    System.out.println(iterador.next());  
}
```




Set. TreeSet

Características

- Uso de una estructura en árbol
- No admite valores duplicados
- No permite valores nulos
- Los elementos están ordenados según sus valores
- Rendimiento es inferior para insertar
- Las búsquedas son rápidas
- Se puede definir el criterio de ordenación con la interfaz **Comparator**

Método	Descripción
E first()	Devuelve el menor elemento.
E last()	Devuelve el mayor elemento.
E floor(E e)	Devuelve el mayor elemento que sea menor o igual al elemento dado, null si no existe.
E ceiling(E e)	Devuelve el menor elemento que sea mayor o igual al elemento dado, null si no existe.
E higher(E e)	Devuelve el menor elemento que sea mayor al elemento dado, null si no existe.
E lower(E e)	Devuelve el mayor elemento que sea menor al elemento dado, null si no existe.
E pollFirst()	Recupera y elimina el primer elemento menor.
E pollLast()	Recupera y elimina el último elemento mayor.

Resultado:
A. Ay. y cia
Asdaiop S.A
Copisteria El Arbol
JindaX S.A.
Juan A. Escaleras
TodoFurgo2287 S.L.

```
TreeSet<String> clientesTree = new TreeSet<>();
clientesTree.add("Asdaiop S.A");
clientesTree.add("Juan A. Escaleras");
clientesTree.add("JindaX S.A.");
clientesTree.add("Copisteria El Arbol");
clientesTree.add("TodoFurgo2287 S.L.");
clientesTree.add("JindaX S.A.");
clientesTree.add("A. Ay. y cia");

System.out.println("Resultado:");
for (String cliente : clientesTree) {
    System.out.println(cliente);
}
```

Conceptos fundamentales para colecciones List y Queue



FIFO

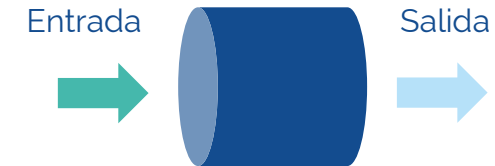
Representa una cola.

Primero en entrar primero en salir

Extraer: se hace del principio.

Insertar: se hace del final.

Ejemplo: Para tratar productos con fecha de caducidad



LIFO

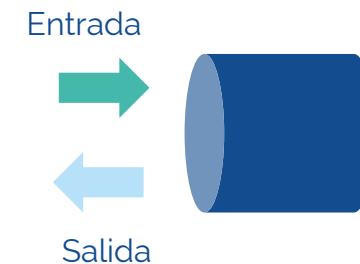
Representa una pila

Último en entrar primero en salir

Extraer: se hace del final

Insertar: se hace del final.

Ejemplo: Producto de la construcción





List. ArrayList

Características

- Es un array
- Acceso secuencial
- Puede tener elementos null
- Permite valores duplicados
- Los elementos están ordenados según se insertan
- Rendimiento bueno para el acceso
- Penaliza la inserción o eliminación

Método	Descripción
E get(int posición)	Devuelve el elemento según la posición dada
int indexOf(E Elemento)	Devuelve la posición de la primera ocurrencia del elemento
int LastIndexOf(E Elemento)	Devuelve la posición de la última ocurrencia del elemento
E set(int posición, E elemento)	Reemplaza el elemento según la posición dada
sort(comparator<? super E> c)	Ordena los elementos según el comparador facilitado.

Resultado:
Asdaiop S.A
Juan A. Escaleras
JindaX S.A.
Copisteria El Arbol
null
TodoFurgo2287 S.L.
JindaX S.A.
A. Ay. y cia

```
ArrayList<String> clientes = new ArrayList<>();
clientes.add("Asdaiop S.A");
clientes.add("Juan A. Escaleras");
clientes.add("JindaX S.A.");
clientes.add("Copisteria El Arbol");
clientes.add(null);
clientes.add("TodoFurgo2287 S.L.");
clientes.add("JindaX S.A.");
clientes.add("A. Ay. y cia");

System.out.println("Resultado:");
for (String cliente : clientes) {
    System.out.println(cliente);
}
```



List. LinkedList

Características

- Es un array. Contiene enlaces al elemento siguiente y anterior
- Acceso secuencial
- Puede tener elementos null
- Permite valores duplicados
- Los elementos están ordenados según se insertan
- Rendimiento bueno para el acceso
- Mejora la inserción o eliminación

Método	Descripción
addFirst(E e)	Inserta el elemento al principio.
push(E e)	Inserta el elemento al principio.
addLast(E e)	Inserta el elemento al final.
boolean offer(E e)	Inserta el elemento al final.
E peek()	Devuelve el elemento del principio
E peekLast()	Devuelve el elemento del final
E poll()	Devuelve y elimina el elemento del principio
E pollLast()	Devuelve y elimina el elemento del final

Resultado:
Asdaiop S.A
Juan A. Escaleras
JindaX S.A.
Copisteria El Arbol
null
TodoFurgo2287 S.L.
JindaX S.A.
A. Ay. y cia

```
LinkedList<String> clientes = new LinkedList<>();
clientes.add("Asdaiop S.A");
clientes.add("Juan A. Escaleras");
clientes.add("JindaX S.A.");
clientes.add("Copisteria El Arbol");
clientes.add(null);
clientes.add("TodoFurgo2287 S.L.");
clientes.add("JindaX S.A.");
clientes.add("A. Ay. y cia");

System.out.println("Resultado:");
for (String cliente : clientes) {
    System.out.println(cliente);
}
```



Queue. PriorityQueue

Características

- Es un símil a la interfaz List aplicando FIFO
- No puede tener elementos null
- Permite valores duplicados
- Los elementos están ordenados según se insertan
- O los elemento se ordenan por el orden natural del tipo o definir uno propio con la interfaz

Comparable

Resultado:

1
3
4
9
55
67

```
PriorityQueue<Integer> turnoClientes = new PriorityQueue<>();
turnoClientes.add(9);
turnoClientes.add(55);
turnoClientes.add(4);
turnoClientes.add(1);
turnoClientes.add(67);
turnoClientes.add(3);

Integer elemento = turnoClientes.poll();
while(elemento!=null) {
    System.out.println(elemento);
    elemento = turnoClientes.poll();
}
```

Queue. ArrayDeque

Características

- Es un símil a la interfaz ArrayList aplicando FIFO o LIFO
- No puede tener elementos null
- Permite valores duplicados
- Los elementos están ordenados según se insertan

Resultado:
9
55
4
1
67
3

```
ArrayDeque<Integer> turnoClientes = new ArrayDeque<>();
turnoClientes.add(9);
turnoClientes.add(55);
turnoClientes.add(4);
turnoClientes.add(1);
turnoClientes.add(67);
turnoClientes.add(3);

System.out.println("Resultado: ");
Integer elemento = turnoClientes.poll();
while(elemento!=null) {
    System.out.println(elemento);
    elemento = turnoClientes.poll();
}
```

Interfaz Map



Características

- Se utiliza un par clave-valor por cada elemento
- No admite claves duplicadas
- Si puede admitir valores duplicados



Tipos de mapa

- HashMap
- LinkedHashMap
- TreeMap

Método	Descripción
int size()	Devuelve el número de elementos contenido en el mapa.
boolean isEmpty()	Devuelve true si el mapa está vacío. false si hay algún par clave-valor.
void clear()	Borra todos los pares clave-valor.
set<K> keySet()	Obtiene todas las claves del mapa.
Collection<V> values()	Obtiene todos los valores del mapa.
boolean containsKey(K clave)	Devuelve true si existe la clave, false en caso contrario.
boolean containsValue(V valor)	Devuelve true si existe el valor, false en caso contrario.
Object get(Object clave)	Obtiene el valor según su clave.
Object put(K clave, V valor)	Añade el par clave-valor. Si el par ya existía devuelve el objeto, si no, devuelve null
Object remove(K clave)	Elimina el par clave-valor según su clave. Y lo devuelve si lo ha podido eliminar.
boolean remove(K clave, V valor)	Elimina el par clave-valor según su clave y valor. Devuelve true si lo ha eliminado.
E replace(K clave, V valor)	Reemplaza el valor del par clave-valor. Devuelve el valor anterior si lo reemplaza, null, en caso contrario.



HashMap

Características

- Guarda las claves en una tabla hash
- Los elementos no están ordenados
- Permite poner valores nulos
- El rendimiento es bueno.

Resultado:
null
Asdaiop S.A
Juan A. Escaleras
A. Ay. y cia
JindaX S.A.
TodoFurgo2287 S.L.
JindaX S.A.
Copisteria El Arbol

```
HashMap<Integer, String> clientes = new HashMap<Integer, String>();  
clientes.put(1, "Asdaiop S.A");  
clientes.put(2, "Juan A. Escaleras");  
clientes.put(4, "JindaX S.A.");  
clientes.put(7, "Copisteria El Arbol");  
clientes.put(null, null);  
clientes.put(5, "TodoFurgo2287 S.L.");  
clientes.put(6, "JindaX S.A.");  
clientes.put(3, "A. Ay. y cia");  
  
System.out.println("Resultado:");  
for (String cliente : clientes.values()) {  
    System.out.println(cliente);  
}
```


LinkedHashMap

Características

- Es un HashMap
- Guarda los enlaces del elemento siguiente y anteriores
- Los elementos están ordenados según el orden de inserción
- Permite poner valores nulos
- El rendimiento es inferior a HashMap


Resultado:
Asdaiop S.A
Juan A. Escaleras
JindaX S.A.
Copisteria El Arbol
null
TodoFurgo2287 S.L.
JindaX S.A.
A. Ay. y cia

```
LinkedHashMap<Integer, String> clientes =  
    new LinkedHashMap<Integer, String>();  
clientes.put(0, "Asdaiop S.A");  
clientes.put(1, "Juan A. Escaleras");  
clientes.put(3, "JindaX S.A.");  
clientes.put(4, "Copisteria El Arbol");  
clientes.put(null, null);  
clientes.put(5, "TodoFurgo2287 S.L.");  
clientes.put(7, "JindaX S.A.");  
clientes.put(6, "A. Ay. y cia");  
  
System.out.println("Resultado");  
for (int i= 0; i<clientes.size(); i++) {  
    System.out.println(clientes.get(i));  
}
```

TreeMap

Características

- Estructura de árbol
- Ordena por las claves
- No permite valores nulos en las claves
- Búsquedas muy rápidas
- Rendimiento inferior para insertar



```
Resultado:  
Asdaiop S.A  
Juan A. Escaleras  
null  
JindaX S.A.  
Copisteria El Arbol  
TodoFurgo2287 S.L.  
A. Ay. y cia  
JindaX S.A.
```

```
TreeMap<Integer, String> clientesTreeMap = new TreeMap<Integer, String>();  
clientesTreeMap.put(0, "Asdaiop S.A");  
clientesTreeMap.put(1, "Juan A. Escaleras");  
clientesTreeMap.put(3, "JindaX S.A.");  
clientesTreeMap.put(4, "Copisteria El Arbol");  
clientesTreeMap.put(2, null);  
clientesTreeMap.put(5, "TodoFurgo2287 S.L.");  
clientesTreeMap.put(7, "JindaX S.A.");  
clientesTreeMap.put(6, "A. Ay. y cia");  
  
System.out.println("Resultado:");  
for (int i = 0; i < clientesTreeMap.size(); i++) {  
    System.out.println(clientesTreeMap.get(i));  
}
```



¿Qué colección elegir?

- ¿Qué tipo de elemento vamos a guardar?
- ¿Puede estar repetido?
- ¿Se van a realizar búsquedas?
- ¿Con qué frecuencia se insertaran los datos?
- ¿Necesitamos que sea rápido?
- Otras cuestiones.



Resumen

1. Las colecciones
2. Interfaz Collection
3. Set. HashSet
4. Set. LinkedHashSet
5. Set. TreeSet
6. Conceptos fundamentales para List y Queue
7. List. ArrayList
8. List. LinkedList
9. Queue. PriorityQueue
10. Queue. ArrayDeque
11. Interfaz Map
12. HashMap
13. LinkedHashMap
14. TreeMap
15. ¿Qué colección elegir?

The background is a solid blue color. Overlaid on this are several faint, light-blue geometric patterns. These include a grid of small squares that form larger, irregular shapes, and numerous small, light-blue arrows pointing in various directions. The overall effect is a sense of movement and digital connectivity.

UNIVERSAE

— CHANGE YOUR WAY —