UNIVERSIDAD DEL BÍO-BÍO

Paradigmas de la Programación

Colección de Objetos

UNIVERSIDAD DEL BÍO-BÍO

Colección de objetos

Una colección de objetos es un objeto que puede almacenar un número variable de elementos siendo cada elemento otro objeto totalmente distinto.

UNIVERSIDAD DEL BÍO-BÍO

ArrayList<T>

La clase ArrayList en Java, es un tipo clase que permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja de que el número de elementos que almacena, lo hace de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los Arrays.

Obs.: En la imagen de abajo se crea una lista con capacidad indeterminada de tipo **String**.

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

UNIVERSIDAD DEL BÍO-BÍO

Utilizando ArrayList

```
ArrayList<String> textos = new ArrayList<String>();
textos.add("Hola Mundo");//Agregando elementos
textos.add(":)");
textos.add("Tengo sueño");
textos.add("¿Qué hora es?");

String texto = textos.get(3);//Obteniendo valor de la posición 3

for(int i=0; i < textos.size(); i++)//Imprimiendo cada valor.
    System.out.println(textos.get(i));
```

UNIVERSIDAD DEL BÍO-BÍO

ArrayList con datos primitivos

ArrayList trabaja solamente con declaraciones de Clases, pero no con datos primitivos(int,float,long,….).
Para poder manipular números por ejemplo, es necesario recurrir a la clase Wrapper asociada(Integer).
Las clases Wrapper son la versión en objetos de los datos primitivos.

- Byte para byte.
- Short para short.
- Integer para int.
- Long para long.
- Boolean para boolean
- Float para float.
- Double para double
- Character para char.

UNIVERSIDAD DEL BÍO-BÍO

ArrayList con datos primitivos

Si se desea tener una lista de números, se tendrá que utilizar su variante Integer y no int.

```
ArrayList<Integer> numeros = new ArrayList<Integer>();
numeros.add(10);//Agregando elementos
numeros.add(15);
numeros.add(20);
numeros.add(25);
```

UNIVERSIDAD DEL BÍO-BÍO

ArrayList – Métodos

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1

UNIVERSIDAD DEL BÍO-BÍO

HashMap<K,V>

HashMap es una clase de colección basada en Map que se utiliza para almacenar claves asociadas a un valor, se denota como HashMap <Clave, Valor> o HashMap <K, V>. Esta clase no ofrece garantías en cuanto al orden.

Como no es una colección ordenada, no devolverá las claves y los valores en el mismo orden en que se insertaron. Para poder utilizar esta clase se requiere la importación de **java.util.HashMap.**

```
HashMap<String, Persona> map = new HashMap<>();
```

UNIVERSIDAD DEL BÍO-BÍO

Agregando valores

Para agregar un valor se ocupará el método put de la instancia.
En el primer parámetro se ingresa la clave asociado al valor del segundo parámetro.

```
HashMap<String, Persona> map = new HashMap<>();

map.put("12.222.222-2", new Persona("Mario"));
map.put("15.333.333-3", new Persona("Juan"));
map.put("17.555.555-5", new Persona("Diego"));
map.put("18.000.000-0", new Persona("Daniela"));
```

UNIVERSIDAD DEL BÍO-BÍO

Consultando un valor

Antes de obtener un valor es necesario preguntar si se encuentra disponible, para ello se necesitará ingresar en el método **containsKey** la **clave** asociada al objeto. Si obtenemos verdadero utilizaremos el método **get** ingresando la **clave**.

```
Persona persona = null;
if (map.containsKey("??.???.???-?"))
    persona = map.get("??.???.???-?");
```

Recorriendo un HashMap

Una forma para recorrer el HashMap, es crear un ciclo "foreach" asociado a las claves y de acuerdo a ello obtener los objetos.

```
for(String clave : map.keySet())
    System.out.println(map.get(clave));
```

```
run:
Daniela
Diego
Juan
Mario
BUILD SUCCESSFUL (total time: 0 seconds)
```

UNIVERSIDAD DEL BÍO-BÍO

Métodos más usados

Tipo de retorno	Método - Descripción
void	clear()Removes all of the mappings from this map.
Object	clone()Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
boolean	containsKey(Object key)Returns true if this map contains a mapping for the specified key.
boolean	containsValue(Object value)Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>>	entrySet()Returns a Set view of the mappings contained in this map.
V	get(Object key)Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
boolean	isEmpty()Returns true if this map contains no key-value mappings.
Set<K>	keySet()Returns a Set view of the keys contained in this map.
V	put(K key, V value)Associates the specified value with the specified key in this map.
V	remove(Object key)Removes the mapping for the specified key from this map if present.
int	size()Returns the number of key-value mappings in this map.
Collection<V>	values()Returns a Collection view of the values contained in this map.

Para mayor información visitar:

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

UNIVERSIDAD DEL BÍO-BÍO

TreeSet<E>

TreeSet es una de las implementaciones más importantes de la interfaz SortedSet la cual utiliza un Árbol para el almacenamiento, su costo de operaciones es de O(log n).

La interfaz SortedSet proporciona funcionalidades para mantener los elementos ordenados. Incluye también la interfaz NavigableSet proporcionando funcionalidades para recorrer los datos almacenados. Por ejemplo, encontrar un elemento mayor o menor respecto a otro, encontrar el primer y último elemento para un conjunto de datos.

UNIVERSIDAD DEL BÍO-BÍO

Características

1. TreeSet implementa la interfaz SortedSet , por lo que **no se permiten valores duplicados en la comparación**.
2. Los objetos en un TreeSet se almacenan en **orden ascendente** (De menor a mayor).
3. TreeSet no conserva el orden de inserción de los elementos, pero los elementos se ordenan por claves.
4. No permite insertar elementos de diferentes tipos. Lanzará ClassCastException si se intenta agregar objetos ajenos a la colección.

UNIVERSIDAD DEL BÍO-BÍO

Características

1. Sirve como una excelente opción para almacenar grandes cantidades de información ordenada a la que se supone que se debe acceder rápidamente debido a su mayor tiempo de acceso y recuperación.
2. Su implementación está basada en un árbol de búsqueda binaria Equilibrado como Rojo-Negro/AVL. Por lo tanto, sus operaciones como agregar, eliminar y buscar costarán O(Log n). Por último, operaciones como imprimir n elementos en orden tomará O(n).

UNIVERSIDAD DEL BÍO-BÍO

Importante

Para utilizar TreeSet en nuestras clases, será necesario implementar la interfaz **Comparable**.

UNIVERSIDAD DEL BÍO-BÍO

Agregando valores

Para agregar un valor se ocupará el método add asociado a TreeSet.

```
public static void main(String[] args) {

    TreeSet<Persona> tree = new TreeSet<>();

    tree.add(new Persona("Diego", 27));
    tree.add(new Persona("Matías", 25));
    tree.add(new Persona("Juan", 30));
    tree.add(new Persona("Daniela", 24));

    System.out.println("Árbol : " + tree);

}
```

```
run:
Árbol : [Daniela - 24, Matías - 25, Diego - 27, Juan - 30]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Recorriendo un TreeSet – Forma 1

Una forma para recorrer el TreeSet, es crear un ciclo "foreach", también se puede utilizar el método descendingSet(), el cual retornar un NavigableSet, encargado de entregar los valores de manera descendente.

```
public static void main(String[] args) {

    TreeSet<Persona> tree = new TreeSet<>();

    tree.add(new Persona("Diego", 27));
    tree.add(new Persona("Matías", 25));
    tree.add(new Persona("Juan", 30));
    tree.add(new Persona("Daniela", 24));

    for (Persona persona : tree.descendingSet())
        System.out.println(persona);

}
```

```
run:
Juan - 30
Diego - 27
Matías - 25
Daniela - 24
BUILD SUCCESSFUL (total time: 0 seconds)
```

Recorriendo un TreeSet – Forma 2

Una forma para recorrer el TreeSet, es crear un ciclo "while" iterando los elementos por medio de **Iterator** (Los elementos se obtiene de forma ascendente).

Métodos:

hasNext() : Retorna Verdadero/Falso si aún hay elementos.

next(): Entrega un elemento.

```
public static void main(String[] args) {

    TreeSet<Persona> tree = new TreeSet<>();

    tree.add(new Persona("Diego", 27));
    tree.add(new Persona("Matías", 25));
    tree.add(new Persona("Juan", 30));
    tree.add(new Persona("Daniela", 24));

    Iterator e = tree.iterator();

    while(e.hasNext())
        System.out.println(e.next());

}
```

```
run:
Daniela - 24
Matías - 25
Diego - 27
Juan - 30
BUILD SUCCESSFUL (total time: 0 seconds)
```

UNIVERSIDAD DEL BÍO-BÍO

Utilizando algunos métodos

```
public static void main(String[] args) {

    TreeSet<Persona> tree = new TreeSet<>();

    Persona diego = null;

    tree.add(diego = new Persona("Diego", 27));
    tree.add(new Persona("Matías", 25));
    tree.add(new Persona("Juan", 30));
    tree.add(new Persona("Daniela", 24));

    System.out.println("Árbol :" + tree);
    System.out.println("1.- Elemento más cercano a : " + diego);
    System.out.println("1.1.- Por valor más bajo (Comparación): " + tree.lower(diego));
    System.out.println("1.2.- Por valor más alto (Comparación): " + tree.higher(diego));
    System.out.println("2.1.- Primer elemento: " + tree.first());
    System.out.println("2.2.- Último elemento: " + tree.last());

}
```

```
run:
Árbol :[Daniela - 24, Matías - 25, Diego - 27, Juan - 30]
1.- Elemento más cercano a : Diego - 27
1.1.- Por valor más bajo (Comparación): Matías - 25
1.2.- Por valor más alto (Comparación): Juan - 30
2.1.- Primer elemento: Daniela - 24
2.2.- Último elemento: Juan - 30
```

UNIVERSIDAD DEL BÍO-BÍO

Métodos más usados

Tipo de retorno	Método - Descripción
boolean	**add**(**E** e)Adds the specified element to this set if it is not already present.
void	**clear**()Removes all of the elements from this set.
boolean	**contains**(**Object** o)Returns true if this set contains the specified element.
Iterator<**E**>	**descendingIterator**()Returns an iterator over the elements in this set in descending order.
NavigableSet<**E**>	**descendingSet**()Returns a reverse order view of the elements contained in this set.
E	**first**()Returns the first (lowest) element currently in this set.
NavigableSet<**E**>	**headSet**(**E** toElement, boolean inclusive)Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
E	**higher**(**E** e)Returns the least element in this set strictly greater than the given element, or null if there is no such element.
boolean	**isEmpty**()Returns true if this set contains no elements.

UNIVERSIDAD DEL BÍO-BÍO

Métodos más usados

Tipo de retorno	Método - Descripción
Iterator<E>	iterator()Returns an iterator over the elements in this set in ascending order.
E	last()Returns the last (highest) element currently in this set.
E	lower(E e)Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	pollFirst()Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	pollLast()Retrieves and removes the last (highest) element, or returns null if this set is empty.
boolean	remove(Object o)Removes the specified element from this set if it is present.
int	size()Returns the number of elements in this set (its cardinality).
SortedSet<E>	tailSet(E fromElement)Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
NavigableSet<E>	tailSet(E fromElement, boolean inclusive)Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

Para mayor información visitar:

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