CONJUNTOS

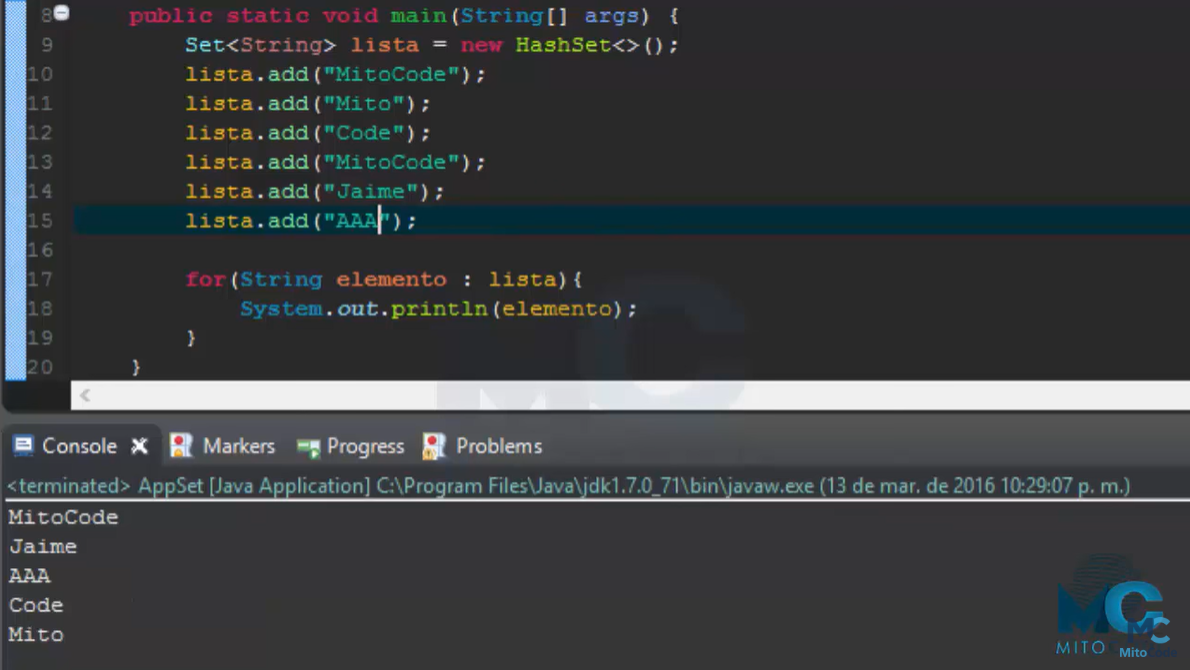
No pueden tener elementos duplicados. A diferencia de una lista donde los elementos podían repetirse.

**Tipos de conjuntos**

• HashSet: se implementa utilizando una tabla hash para darle un valor único a cada elemento y de esa manera evitar los duplicados. Tiene mejor performance que el Treeset.

Se preocupa por tener valores únicos;

Los guarda de manera aleatoria;

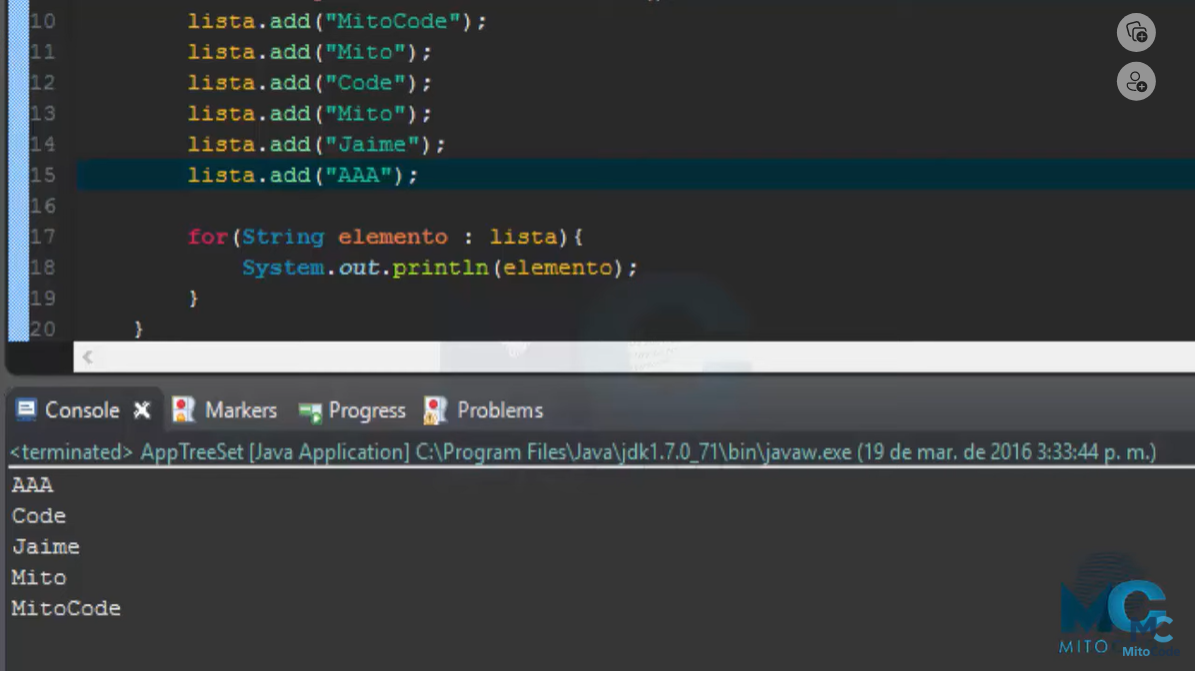


|||||No se preocupa por respetar el orden en el que se ingresa los datos|||||

[Tutorial Java 7 SE Avanzado - 13 HashSet - YouTube](https://www.youtube.com/watch?v=YRf9k0aHtus)

Ejemplo: HashSet <String>nombres = new HashSet(); (DECLARACIÓN)

• TreeSet: se implementa utilizando una estructura de árbol (árbol rojo-negro en el libro de algoritmos). Mantiene sus elementos de manera ordenada(**forma ascendente**).Los métodos de agregar, eliminar son más lentos(**que cada vez que le entra un elemento debe posicionarlo** **para que quede ordenado).**

****

Se preocupa por tener valores únicos;

Los guarda de manera ordenada ascedente;

[Tutorial Java 7 SE Avanzado - 14 TreeSet - YouTube](https://www.youtube.com/watch?v=Gclb5YHIi0o)

Ejemplo: TreeSet <Integer>numeros = new TreeSet(); (DECLARACIÓN)

LinkedHashSet: Está entre HashSet y TreeSet. Se implementa como una tabla hash con una lista vinculada que se ejecuta a través de ella.

Se preocupa por tener valores únicos;

Los guarda por orden de inserción

Ejemplo: LinkedHashSet <String> frases = new LinkedHashSet(); (DECLARACION)

MAPAS

Los mapas modelan un objeto a través de una llave y un valor. Un ejemplo seria una persona que tiene su dni/rut (llave única) y como valor puede ser su nombre completo, puede haber dos personas con el mismo nombre, pero distinto dni/rut(**el primer tipo dato será el valor de la llave y el segundo el valor del valor (valga la redundancia)**.

HashMap: Las llaves se almacenan utilizando un algoritmo de hash para las llaves y evitar que se repitan.

HashMap <Integer,String>personas = new HashMap(); DECLARACIÓN ;

TreeMap: es un mapa que ordena los elementos de manera ascendente a través de las llaves.

TreeMap <Integer,String> personas = new TreeMap(); DECLARACIÓN;

LinkedHashMap: es un HashMap que conserva el orden de inserción.

LinkedHashMap <Integer,String> personas = new LinkedHashMap(); DECLARACIÓN

AÑADIR UN ELEMENTO A UNA COLECCIÓN

Si queremos añadir un elemento a las listas o sets vamos a tener que utilizar la función add(T), pero para los mapas vamos a utilizar la función put(llave,valor).

Listas:

ArrayList <Integer> numeros = new ArrayList();

int num = 20;

numeros.**add**(num);

Conjuntos:

HashSet <Integer> numeros = new HashSet();

int num = 20;

numeros.**add**(num);

Mapas:

HashMap <Integer,String> alumnos = new HashMap();

int dni = 34576189;

String nombreAlumno = “Pepe”;

alumnos**.put**(dni, nombreAlumno);

RECORRER UNA COLECCIÓN

Para recorrer mapas vamos a tener que usar el objeto Map.Entry en el for each. A traves de el entry vamos a traer los valores y las llaves, si no podemos tener un for each para cada parte de nuestro mapa sin utlilizar el objeto Map.Entry

Listas y Conjuntos:

ArrayList <String>lista = new ArrayList();

for (String cadena : lista) {

System.out.println(cadena); // mostramos los elementos a traves de la variable

}

Mapas:

HashMap <Integer,String> estudiantes = new HashMap();

Recorrer las dos partes del mapa:

/// Se usa Map.Entry es cada elemento que tiene el mapa(clave,valor);

/// Es una interfaz anidada dentro de la interfaz Map;

for (**Map.Entry <Intenger,String> entry : estudiantes.entrySet())** {

System.out.println("documento=" + **entry.getKey()** + ", nombre=" **+ entry.getValue());**

// entry.getKey trae la llave y entry.getValue trae los valores del mapa

// Mostrar solo las llaves :

for (Integer dni : **estudiantes.keySet())** {

System.out.println("Documento = " + dni);

}

// Mostrar solo los valores:

for (String nombres : **estudiantes.values())** {

System.out.println("Nombre: " + nombres);

}

ITERATOR

Este, nos permite recorrer, acceder a la información y eliminar algún elemento de una colección. Ya que, cuando queremos eliminar algún elemento, mediante el for each, nos va a tirar un error. Todas las colecciones vienen con un metodo iterator().

Para poder usar el Iterator es importante crear el objeto de tipo Iterator, con el mismo tipo de dato que nuestra colección.

**El iterator contiene tres metodos muy utiles para lograr esto:**

1. boolean hasNext(): Retorna verdadero si al iterator le quedan elementos por iterar

2. Object next(): Devuelve el siguiente elemento en la coleccion, mientras le metodo hasNext() retorne true. Este metodo es el que nos sirve para mostrar el elemento.

3. void remove(): Elimina el elemento actual de la colección.

Ejemplo:

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

lista.add("A");

lista.add("B");

// Iterator para recorrer la lista

**Iterator** iterator = lista.iterator(); Devolvemos el iterador

System.out.println("Elementos de la lista : ");

// Usamos un while para recorrer la lista, siempre que el hasnext() devuelva true.

while (**iterator.hasNext**())

System.out.print(**iterator.next()** + " ");

// Mostramos los elementos con el iterator.next()

System.out.println();

}

ELIMINAR UN ELEMENTO DE UNA COLECCIÓN

Vamos a tener que utilizar la función remove() del **Iterator**, esto se aplica para el resto de nuestras colecciones. Para los mapas, vamos a utilizar **el Map. Entry para usar con el iterator**. Además, **podemos elegir sobre si queremos preguntar por la llave o por el valor**, para validar que elemento eliminar.

**Ejemplo de como recorrer un mapa utilizando el “Map.Entry”:**

while (iterador.hasNext()) {

Map.Entry <String,Integer> entry = iterador.next();

si la llave es igual al producto

if (entry.getKey().equalsIgnoreCase(modificarProducto)) {

System.out.println("Ingrese nuevo precio del producto: ");

precionuevo = entrada.nextInt();

entry.setValue(precionuevo);

System.out.println("Precio modificado correctamente");

}

}

Conjuntos:

HashSet<Integer> numerosSet = new HashSet();

Iterator<Integer> it = numerosSet.iterator();

while (it.hasNext()) {

if (it.next() == 3) { // Borramos si está el numero 3

it.remove();

(**Borrara los elementos que se encuentren en esa “Posición”)**

}

}

Mapas:

Con los mapas vamos a tener que usar el **remove propio de los Mapas pero no podemos**

**recorrerlos**, este remove, solo sirve **para eliminar elementos del mapa según su llave**.

Ejemplo:

HashMap<Integer, String> estudiantes = new HashMap();

Int llaveABorrar = 123;

estudiantes.remove(llaveABorrar)

}

ORDENAR UNA COLECCIÓN

Para ordenar una colección, vamos a tener que utilizar la función **Collections.sort(colección)**. La función recibe la colección y la ordena para después poder mostrarla ordenada de manera ascendente,

Algunas colecciones, como los conjuntos **o los mapas no pueden utilizar el sort().** **Ya que por ejemplo los HashSet, manejan valores Hash y el sort() no sabe ordenar por hash, si no por elementos.** Por otro lado, **los mapas al tener dos datos, el sort() no sabe por cual de sos datos ordenar.**

Nota**: recordemos que los TreeSet y TreeMap se ordenan por si mismos.**

Ordenar ArrayList:

ArrayList <Integer> numeros = new ArrayList(); Collections.sort(numeros);

Collections.sort(numeros);

Conjuntos:

HashSet<Integer> numerosSet = new HashSet();

ArrayList<Integer> numerosLista = new ArrayList(numerosSet);

// Se convierte el HashSet a Lista

Collections.sort(numerosLista);

Mapas:

Mapas:

HashMap<Integer, String> alumnos = new HashMap();

TreeMap<Integer, String> alumnosTree = new TreeMap(alumnos);

// Se convierte el HashMap a TreeMap

COMPARATOR

A la hora de querer ordenar una colección de objetos en Java, no podemos utilizar la función sort, ya que el sort se utiliza para ordenar colecciones con elementos uniformes.

Para esto, utilizamos la interfaz Comparator con su función compare en nuestra clase entidad.

Primero debemos pasarlo a ArrayList, y luego usar los comparadores.

Ejemplo de declaración:

public class Comparadores {

public static Comparator <Pelicula> ordenarporDuracionAscendente = new Comparator<Pelicula>() {

@Override

public int compare(Pelicula t, Pelicula t1) {

return t.getHoras().compareTo(t1.getHoras());

}

ArrayList:

ArrayList<Perro> perros = new ArrayList();

perros.sort(Perro.compararEdad); Se llama al metodo estatico a traves de la

clase y se pone la lista a ordenar.

Conjuntos:

HashSet<Perro> perrosSet = new HashSet();

ArrayList<Perro> perrosLista = new ArrayList(perrosSet);

perrosLista.sort(Perro.compararEdad);

Mapas:

HashMap<Integer, Alumno> alumnos = new HashMap();

ArrayList<Alumno> nombres = new ArrayList(map.values());

//Se usa una función de los mapas para traer todos valores.

nombres.sort(Alumno.compararDni);

Treeset:

En los TreeSet necesitamos crearlos con el comparator porque como el TreeSet se ordena

solo, necesitamos decirle al TreeSet, bajo que atributo se va a ordenar, por eso le pasamos

el comparator en el constructor.

TreeSet<Perro> perros = new TreeSet(Perro.compararEdad);

Perro perro = new Perro();

perros.add(perro);

COLECCIÓNES EN FUNCIONES

A la hora de querer pasar una colección a una función, deberemos recordar que Java es fuertemente tipado, por lo que deberemos poner el tipo de dato de la colección y que tipo de colección es cuando la pongamos como argumento.

Listas:

Public void llenarLista(ArrayList<Integer> numeros){

numeros.add(20);

}

Desde el Main:

ArrayList<Integer> numeros = new ArrayList();

llenarLista(numeros); // Le pasamos la lista a la función

Conjuntos:

Public void llenarHashSet(HashSet<String> palabras){

palabras.add(“Hola”)

}

Desde el Main:

HashSet<String> palabras = new HashSet();

llenarHashSet(palabras); // Le pasamos el conjunto a la función

Mapas:

Public void llenarMapa(HashMap<Integer, String> alumnos){

alumnos.add(1, “Pepe”);

}

Desde el Main:

HashMap<Integer, String> alumnos = new HashMap();

llenarMapa(alumnos); // Le pasamos el conjunto a la función

DEVOLVER UNA COLECCIÓN EN FUNCIONES

Listas:

Public ArrayList<Integer> llenarLista(){

ArrayList<Integer> numeros = new ArrayList();

numeros.add(20);

return numeros; // Devolvemos la lista llena.

}

Conjuntos:

Public HashSet<String> llenarHashSet(){

HashSet<String> palabras = new HashSet();

palabras.add(“Hola”)

return palabras }

Mapas:

Public HashMap<Integer,String> llenarMapa(){

HashMap<Integer, String> alumnos = new HashMap();

alumnos.add(1, “Pepe”);

return alumnos;

}