Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Curso de **Programación** en **Java**

Juan Francisco Maldonado León
Arquitecto de Software

Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Java

Collections

Juan Francisco Maldonado León
Arquitecto de Software

Collections Frameworks

Una colección es simplemente un grupo de objetos de varios elementos representado en una sola unidad.

Las colecciones se utilizan para almacenar, recuperar, manipular y comunicar los datos agregados.

Colección de Contactos Telefonicos.

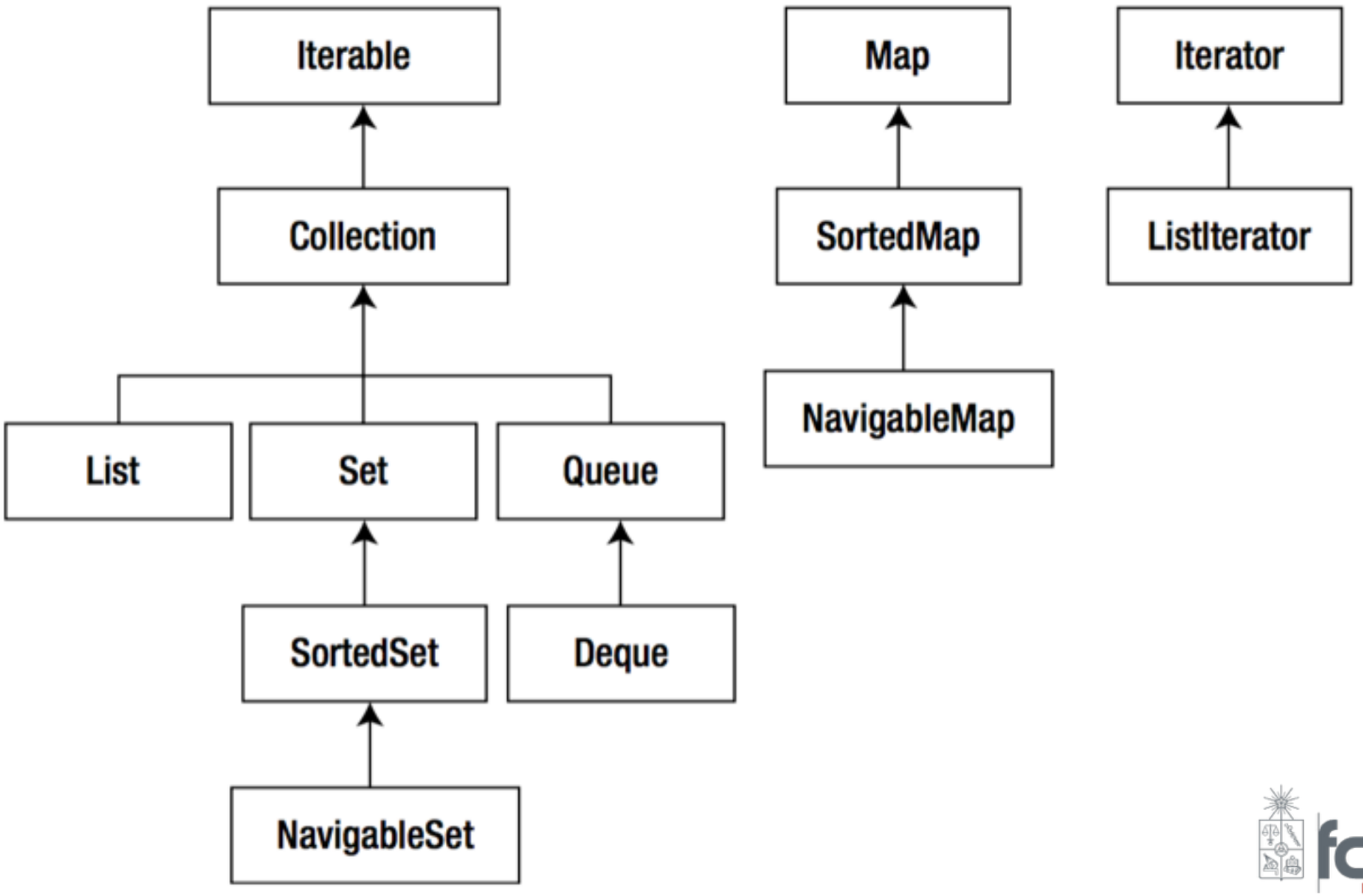
Componentes Principales del Framework

- **Clases abstractas e interfaces**: El Framework de la colecciones tiene muchas clases abstractas e interfaces que proporcionan funcionalidad general. (List, Set, Map, etc)

- **Clases concretas**: Estos son los casos reales de contenedores que va a utilizar en los programas.

- **Algoritmos**: funcionalidades como la clasificación, búsqueda, etc.

Interfaces


Iterable
Collection
List
Set
Queue
SortedSet
Deque
NavigableSet
Map
SortedMap
NavigableMap
Iterator
ListIterator

Introducción a Java

Collections Framework

Abstract Class/Interface	Short Description
`Iterable`	A class implementing this interface can be used for iterating with a `for each` statement.
`Collection`	Common base interface for classes in the collection hierarchy. When you want to write methods that are very general, you can pass the `Collection` interface. For example, `max()` method in `java.util.Collections` takes a `Collection` and returns an object.
`List`	Base interface for containers that store a sequence of elements. You can access the elements using an index, and retrieve the same element later (so that it maintains the insertion order). You can store duplicate elements in a `List`.
`Set`, `SortedSet`, `NavigableSet`	Interfaces for containers that don't allow duplicate elements. `SortedSet` maintains the set elements in a sorted order. `NavigableSet` allows searching the set for the closest matches.
`Queue`, `Deque`	`Queue` is a base interface for containers that holds a sequence of elements for processing. For example, the classes implementing `Queue` can be LIFO (last in, first out— as in stack data structure) or FIFO (first in, first out—as in queue data structure). In a `Deque` you can insert or remove elements from *both* the ends.
`Map`, `SortedMap`, `NavigableMap`	Base class for containers that map keys to values. In `SortedMap`, the keys are in a sorted order. A `NavigableMap` allows you to search and return the closest match for given search criteria. Note that `Map` hierarchy does *not* extend the `Collection` interface.

Interfaz Collection

Method	Short description
`boolean add(Element elem)`	Adds `elem` into the underlying container.
`void clear()`	Removes all elements from the container.
`boolean isEmpty()`	Checks whether the container has any elements or not.
`Iterator<Element> iterator()`	Returns an `Iterator<Element>` object for iterating over the container.
`boolean remove(Object obj)`	Removes the element if `obj` is present in the container.
`int size()`	Returns the number of elements in the container.
`Object[] toArray()`	Returns an array that has all elements in the container.

Clases Concretas

Concrete Class	Short Description
`ArrayList`	Internally implemented as a resizable array. This is one of the most widely used concrete classes. Fast to search, but slow to insert or delete. Allows duplicates.
`LinkedList`	Internally implements a doubly-linked list data structure. Fast to insert or delete elements, but slow for searching elements. Additionally, `LinkedList` can be used when you need a stack (LIFO) or queue (FIFO) data structure. Allows duplicates.
`HashSet`	Internally implemented as a hash-table data structure. Used for storing a set of elements—it does not allow storing duplicate elements. Fast for searching and retrieving elements. It does *not* maintain any order for stored elements.
`TreeSet`	Internally implements a red-black tree data structure. Like `HashSet`, `TreeSet` does not allow storing duplicates. However, unlike `HashSet`, it stores the elements in a sorted order. It uses a tree data structure to decide where to store or search the elements, and the position is decided by the sorting order.
`HashMap`	Internally implemented as a hash-table data structure. Stores key and value pairs. Uses hashing for finding a place to search or store a pair. Searching or inserting is very fast. It does *not* store the elements in any order.
`TreeMap`	Internally implemented using a red-black tree data structure. Unlike `HashMap`, `TreeMap` stores the elements in a sorted order. It uses a tree data structure to decide where to store or search for keys, and the position is decided by the sorting order.

Introducción a Java

Collections Framework

Map

Collections

Los objetos Map asocian claves a valores y no pueden contener claves duplicadas (es decir, cada clave puede asociarse solamente con un valor; a este tipo de asociación se le conoce como asociación de uno a uno.

Tres de las muchas clases que implementan a la interfaz Map son HashTable, HashMap y TreeMap.

HashMap

Collections

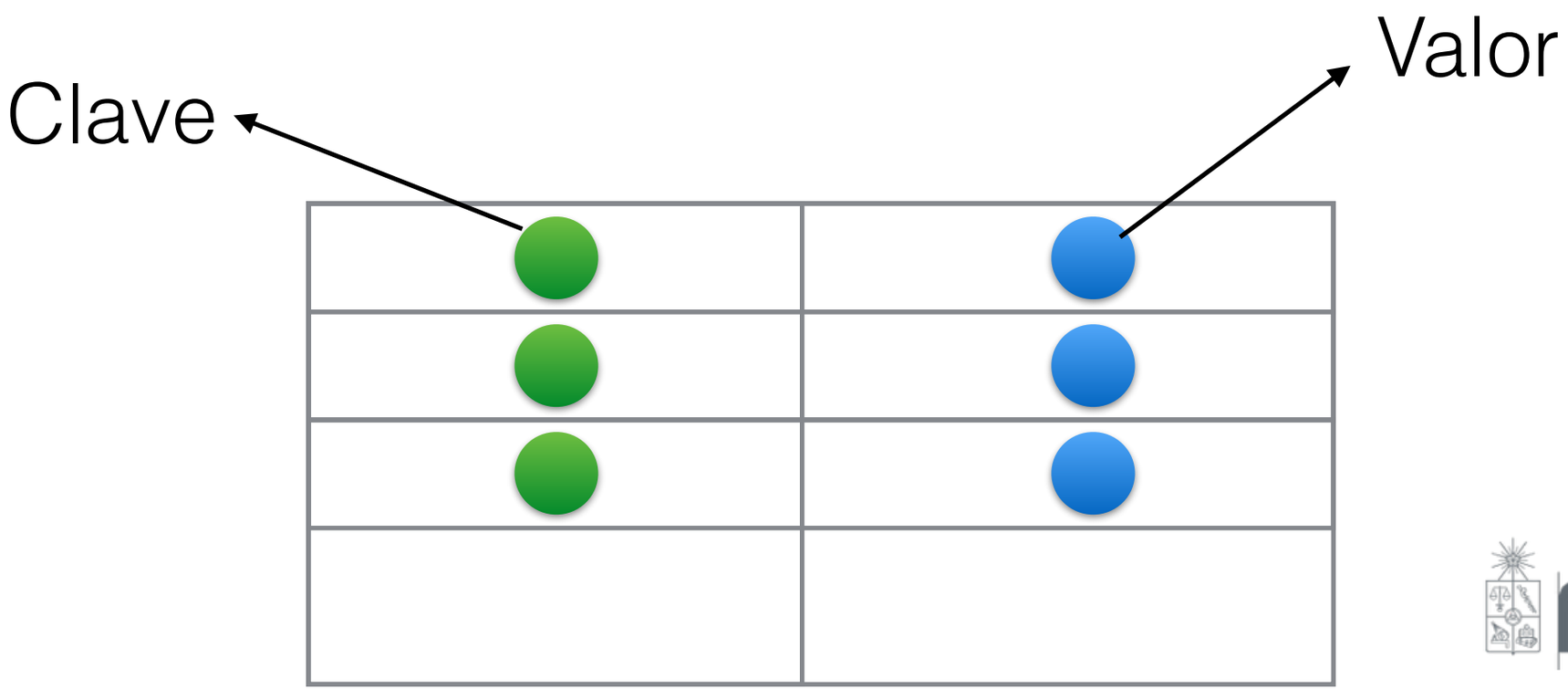
HashMap es una colección *No ordenada.* es decir no devuelve los elemento en el mismo orden en que se almacenan.

La posición en la que se almacena el elemento dependerá del código hash de la clave.

HashMap

Collections

HashMap es una colección *no sincronizada.*
Son eficientes para almacenar y para realizar búsquedas según su clave.



Introducción a Java

Collections Framework

HashMap

Collections

Tipo Clave

Inicializar un HashMap

```
Map<String, String> mapa = new HashMap<String, String>();
```

Tipo Valor

HashMap vacío con una capacidad inicial predeterminada (16 elementos)

HashMap

Collections

Inicializar un HashMap con capacidad inicial

```
Map<String, String> mapa = new HashMap<String, String>(5);
```

HashMap

Collections

La clave es única, es decir no puede repetirse.

Para verificar si dos objetos son iguales se utiliza el método hashCode que hereda de la clase Object

El método hash se encargan de representar un conjunto de datos unívocamente en forma numérica.

HashMap

Collections

Sobreescribir el método hashcode de la clase Object

```java
@Override
public int hashCode() {
    return this.getRut().hashCode();
}
```

HashMap

Collections

```
* s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char val[] = value;

        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
```

Agregar un elemento

Collections

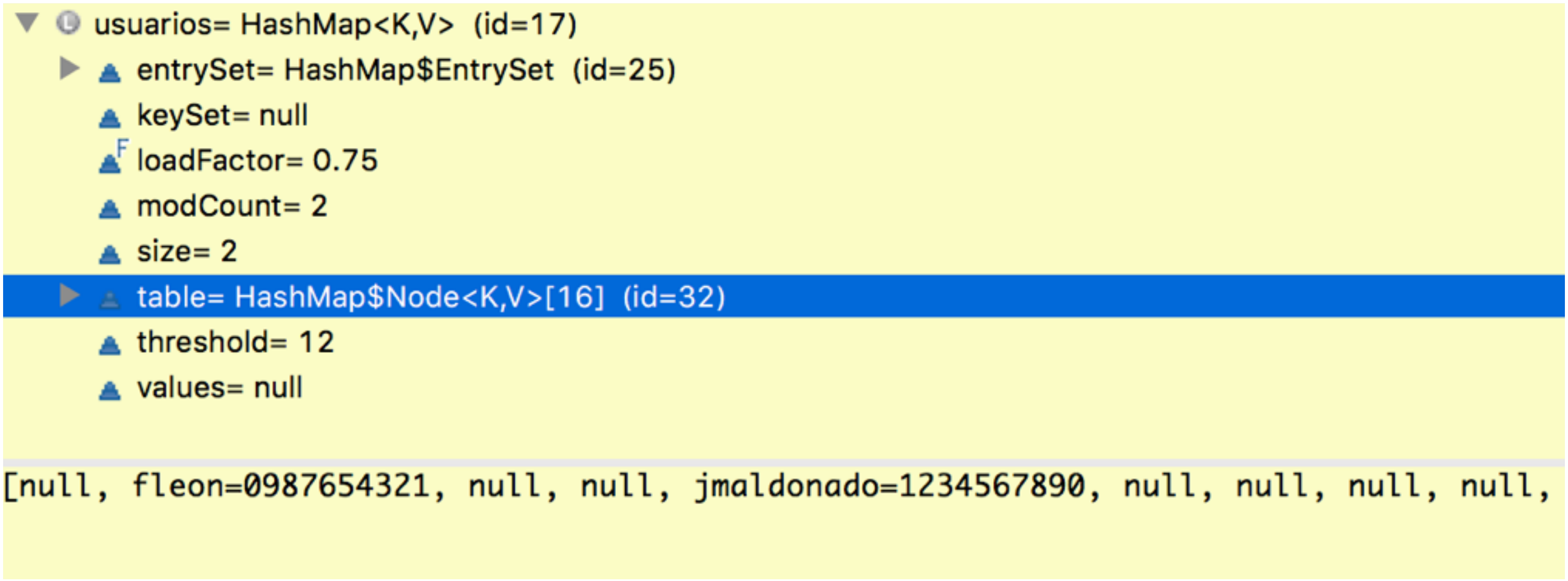
```
Map<String, String> usuarios = new HashMap<String, String>();
usuarios.put("jmaldonado", "1234567890");
usuarios.put("fleon", "0987654321");
```

Clave

Valor

Agregar un elemento

Collections



Agregar un elemento

Collections

```
Map<String, String> usuarios = new HashMap<String, String>();
usuarios.put("jmaldonado", "1234567890");
usuarios.put("jmaldonado", "0987654321");
System.out.println( usuarios.size() );
```

El método size() obtiene el total de elementos, como las claves no pueden repetirse, solo tiene un elemento

Obtener un elemento

Collections

```
Map<String, String> usuarios = new HashMap<String, String>();
usuarios.put("jmaldonado", "1234567890");
usuarios.put("jmaldonado", "0987654321");
System.out.println( usuarios.get("jmaldonado") );
```

Obtiene el valor según la clave ingresada como parámetro

Eliminar un elemento

Collections

```
Map<String, String> usuarios = new HashMap<String, String>();
usuarios.put("jmaldonado", "1234567890");
usuarios.put("fleon", "0987654321");
usuarios.remove("fleon");
```

Elimina el valor según la clave ingresada como parámetro y devuelve el valor asociado.

Métodos

Collections

void clear():Elimina todos los registro del mapa
boolean containsKey(Object key): Retorna un boleano si existe o no la clave.
boolean containsValue(Object Value):Retorna un boleado si existe o no el Valor
boolean isEmpty(): It checks whether the map is empty. If there are no key-value mapping present in the map then this function returns true else false.
Set keySet(): It returns the Set of the keys fetched from the map.
int size(): Returns the size of the map – Number of key-value mappings.

Iterar un Mapa

Collections

```
Iterator<Entry<String, String>> it = usuarios.entrySet().iterator();
while (it.hasNext()) {
    Entry<String, String> e = it.next();
    System.out.println(e.getKey() + " " + e.getValue());
}
```

TreeMap

Collections

Ordena los elementos según el orden natural del tipo de clave, Es decir si la clave es numérica ordenara los elemento de mayor a menor.

Para lograr este orden con cualquier objeto, la clave tiene que implementar la interfaz comparable, para indicar cual será el orden.

TreeMap

Collections

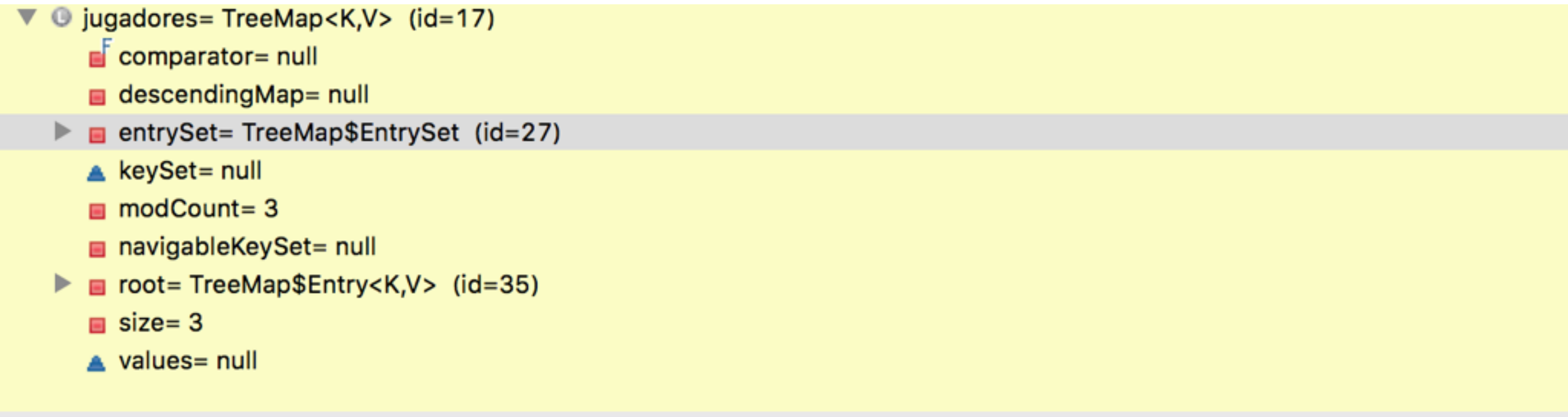
```
Map<Integer, String> jugadores = new TreeMap<Integer, String>();
jugadores.put(23 , "Michael Jordan");
jugadores.put(3 , "Allen Iverson");
jugadores.put(55 , "Jason Williams");
```

Introducción a Java

Collections Framework

TreeMap

Collections




```
[3=Allen Iverson, 23=Michael Jordan, 55=Jason Williams]
```

TreeMap

Collections

```
Persona per1 = new Persona("juan", "16.942.331-6");
Persona per2 = new Persona("francisco", "16.942.333-6");
Persona per3 = new Persona("williams", "16.942.335-6");
Map<Persona, String> personas = new TreeMap<Persona, String>();
personas.put(per1, "1234567890");
personas.put(per2, "0987654321");
personas.put(per3, "0987654321");
```

En este ejemplo la clase Persona, es la clave y debería implementar la interfaz comparable y definir por que atributo se realizara el orden