

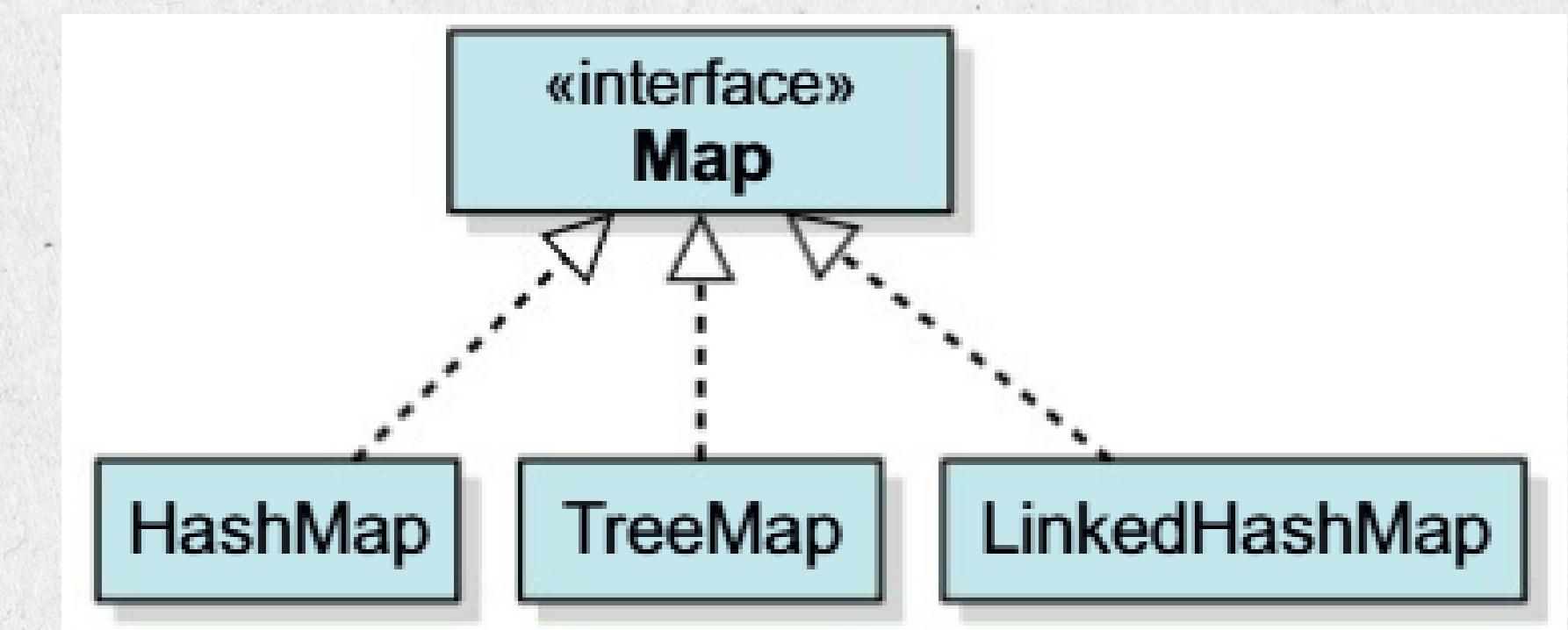
UNIDAD I2: COLECCIONES

CLASE 4: INTERFAZ MAP

INTERFAZ MAP

Los mapas o diccionarios son estructuras dinámicas cuyos elementos son pares claves y valor.

- No heredan de Collection
- Utilizaremos HashMap, TreeMap y LinkedHashMap



INTERFAZ MAP

En los mapas se insertan:

- Claves - Que no pueden repetirse
- Valores - Que si pueden repetirse

Las operaciones fundamentales son la inserción, lectura y eliminación de entradas

INTERFAZ MAP

Inicialmente trabajaremos con HashMap:

```
Map<K,V> m = new HashMap<>();
```

- K es el tipo de las claves
- V es el tipo de los valores

Ambos deben ser clases o interfaces, no primitivas

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

INTERFAZ MAP

- **V put(K clave, V valor):** Inserta el par clave:valor siempre y cuando no exista la clave ya en el mapa, si ya estaba, se sustituye

```
m.put("A", 97.0);
m.put("B", 98.0);
m.put("C", 99.0);
m.put("D", 100.0);
m.put("E", 101.0);
System.out.println(m);
```

{A=97.0, B=98.0, C=99.0, D=100.0, E=101.0}

INTERFAZ MAP

- **void remove(Object k)**: Elimina el par cuya clave es k, si existe. Devuelve el valor asociado

```
System.out.println(m.remove("D"));
System.out.println(m.remove("F"));
```

100.0
null

INTERFAZ MAP

- **V get(Object k):** Devuelve el valor asociado con la clave K

```
System.out.println(m.get("D"));
System.out.println(m.get("F"));
```

```
100.0
null
```

INTERFAZ MAP

- **boolean containsKey(Object k)**: Devuelve true si encuentra la clave K en la colección.

```
System.out.println(m.containsKey("D"));
System.out.println(m.containsKey("F"));
```

true
false

INTERFAZ MAP

- **boolean containsValue(Object v)**: Devuelve true si encuentra algún par con el valor en la colección.

```
System.out.println(m.containsValue(98.0));  
System.out.println(m.containsValue(122.0));
```

true
false

VISTAS COLLECTIONS DE MAPAS

Las vistas son colecciones que respaldan el mapa original de manera que si modificamos las colecciones, modificamos los mapas.

VISTAS COLLECTIONS DE MAPAS

- **KeySet:** Recupera las claves del mapa
 - **Set<K> keyset()**

```
Set<String> claves = m.keySet();
System.out.println(claves);
```

[A, B, C, D, E]

VISTAS COLLECTIONS DE MAPAS

- **Values:** Devuelve una vista Collection (generalmente una lista) con los valores del mapa
 - **Collection<V> values()**

```
Collection<Double> values = m.values();
System.out.println(values);
```

```
[97.0, 98.0, 99.0, 100.0, 101.0]
```

VISTAS COLLECTIONS DE MAPAS

- **EntrySet:** Devuelve una vista de la interfaz con los pares clave:valor con la posibilidad de recuperarlos con getKey(), getValue() y setValue()
 - **Set <Map.Entry<K,V>> entrySet()**
 - **K getKey()**
 - **V getValue()**
 - **V setValue(V nuevo_valor)**

VISTAS COLLECTIONS DE MAPAS

```
Set<Map.Entry<String, Double>> pares = m.entrySet();
for(Map.Entry<String, Double> par : pares) {
    System.out.println("Key: "+par.getKey()+
        " / Value: "+par.getValue());
}
```

```
Key: A / Value: 97.0
Key: B / Value: 98.0
Key: C / Value: 99.0
Key: D / Value: 100.0
Key: E / Value: 101.0
```

VISTAS COLLECTIONS DE MAPAS

VENTAJAS

- Los mapas no permiten la iteración de los elementos, las vistas sí
- Facilita la eliminación en una posición en concreto mediante bucles
- NO podremos hacer inserciones

VISTAS COLLECTIONS DE MAPAS

```
Collection<Double> alum_val = alum.values();
for(Iterator<Double> it = alum_val.iterator();it.hasNext();) {
    Double v = it.next();
    if(v >= 1.70) {
        it.remove();
    }
}
System.out.println(alum);
```

{Ana=1.65, Pedro=1.69, Jorge=1.67}

VISTAS COLLECTIONS DE MAPAS

```
Set<Map.Entry<String, Double>> alum_ent = alum.entrySet();
Iterator<Map.Entry<String, Double>> it = alum_ent.iterator();
for(;it.hasNext();) {
    Map.Entry<String, Double> e = it.next();
    if(e.getValue()>= 1.70) {
        it.remove();
    }
}
System.out.println(alum);
```

{Ana=1.65, Pedro=1.69, Jorge=1.67}

TREEMAP

- **HashMap** no garantiza un orden tras la inserción
- **TreeMap** permite una inserción ordenada y búsqueda eficiente, se insertan en orden natural creciente de claves
 - Podemos modificar el orden en el que se insertan pasando un comparator

TREEMAP

```
Map<String, Double> tm = new TreeMap<>();  
tm.put("Ana", 1.65);  
tm.put("Marta", 1.70);  
tm.put("Luis", 1.73);  
tm.put("Pedro", 1.69);  
tm.put("Lucas", 1.80);  
tm.put("Jorge", 1.67);  
System.out.println(tm);
```

{Ana=1.65, Jorge=1.67, Lucas=1.8, Luis=1.73, Marta=1.7, Pedro=1.69}

LINKEDHASHMAP

- **LinkedHashMap** la información se queda en el orden en el que se van insertando
 - Es muy eficiente en inserciones y eliminación
 - Lento en búsquedas

EJERCICIOS

Ejercicio 1:

Diseña un programa que se comporte como una pequeña agenda. Crea un mapa cuyas claves serán el nombre de la persona y como valor tendrá una lista de teléfonos. Mediante un menú el usuario podrá elegir entre:

- Añadir un nuevo contacto (nombre y uno o varios teléfonos)
- Buscar uno o varios teléfonos de un contacto por el nombre
- Mostrar la información de todos los contactos ordenados alfabéticamente por el nombre.

Puedes elegir que mapa y lista utilizar para almacenar la información teniendo en cuenta que los teléfonos no deberían repetirse en una misma persona y se listarán todos los nombres por orden.

INTRODUCCIÓN

Ejercicio 2:

En un zoo se han establecido diferentes hábitats para cada tipo de animales (ártico, selva, sabana, desierto, océano, bosques..) que se almacenarán en un mapa cuya clave será un número comenzando por el uno.

Los valores serán de la clase Habitat que tendrá:

- Nombre
- Animales que viven en él (una lista con nombres de animales que están en ese habitat en el zoo ordenados alfabéticamente)
- Características
- Km del zoo que ocupa
- Si es un recinto en cerrado o abierto

Tendrá un constructor con parámetros, get y set para sus atributos

INTRODUCCIÓN

Ejercicio 2:

Método `toString` que mostrará la información del habitat en el formato:
“NOMBRE HABITAT: kmqueocupa

- Características: `caracteristicas`
- Espacio abierto/cerrado
- Cantidad de animales: `Cantidad`
- Animales para visitar:
 - `Animal1`
 - `Animal2...`

En main tendréis que crear un zoo con 5 habitats diferentes ordenados por clave y mostrar la información de cada uno de ellos en orden (mostrando el número que los identifica y después su información)