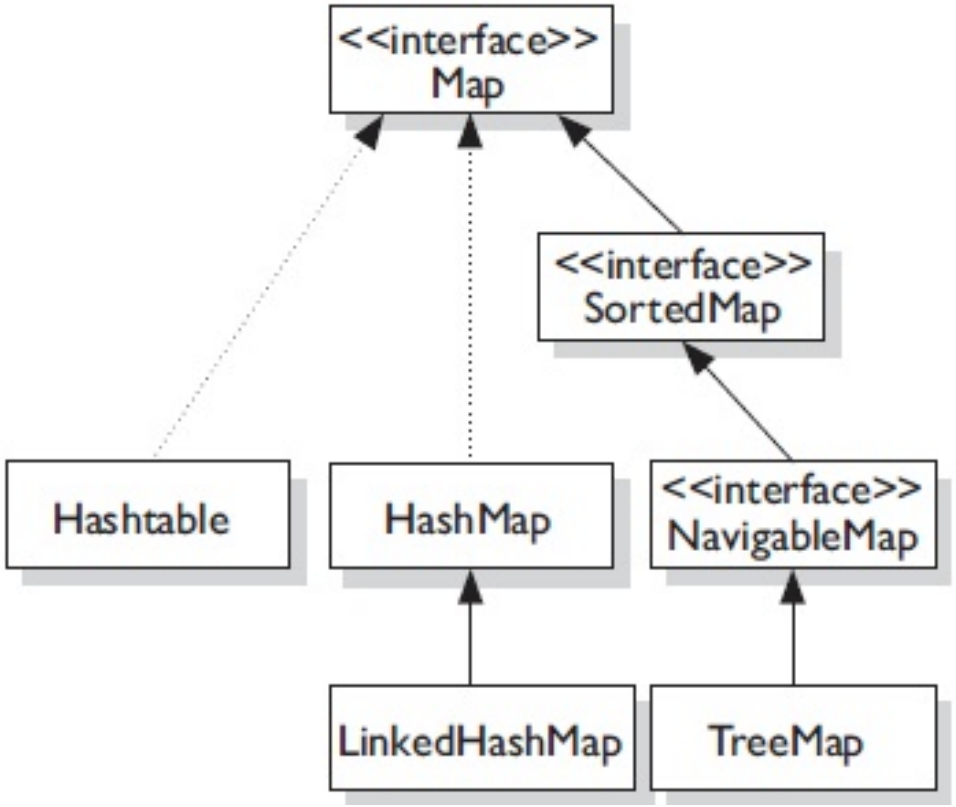
FUNDAMENTOS DE PROGRAMACIÓN

5 – Interfaz Map

Introducción



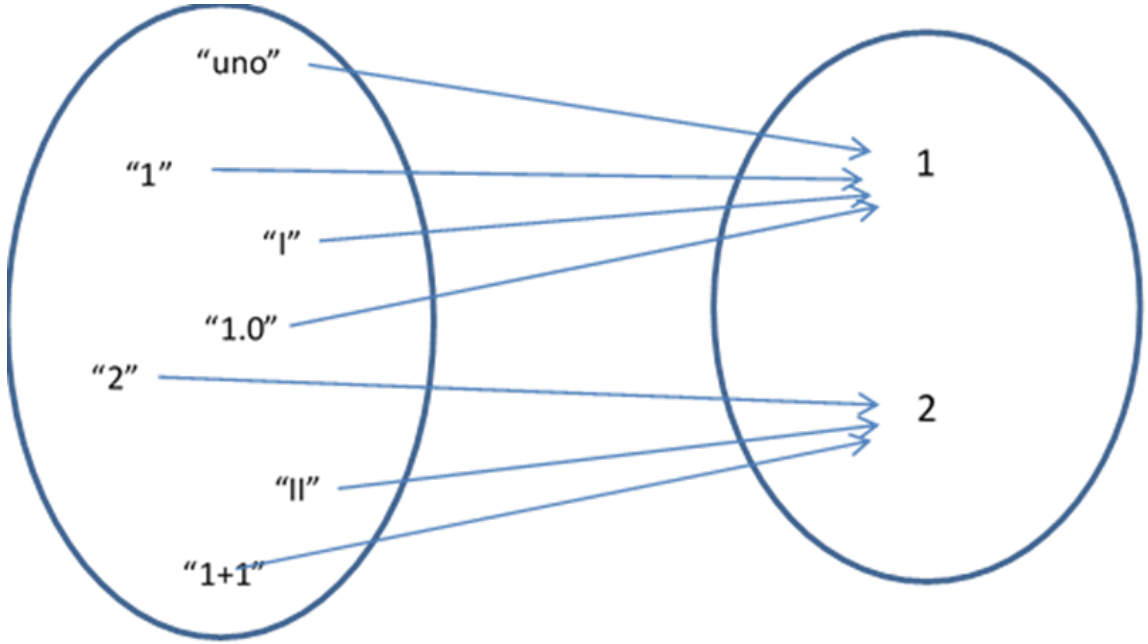
ÍNDICE

ÍNDICE

1. El tipo Map

- Incluido en el paquete java.util.
- Modelar el concepto de aplicación:
  - Una relación entre los elementos de dos conjuntos de modo que a cada elemento del conjunto inicial le corresponde uno y solo un elemento del conjunto final.
  - Los elementos del conjunto inicial se denominan claves (*keys*) y los del conjunto final valores (*values*).
- Entre las claves no puede haber elementos duplicados.
- Una clave no puede estar asociada con más de un valor.
- Sí que puede haber valores duplicados.

1. El tipo Map



Clave	Valor
"uno"	1
"1"	1
"I"	1
"1.0"	1
"2"	2
"II"	2
"1 + 1"	2

1. El tipo Map

- La interfaz *Map* tiene dos tipos genéricos, que suelen denominarse K (de Key) y V (de Value): Map<K, V>.
- HashMap es una clase que implementa la interfaz *Map* y obliga a definir de forma correcta el método hashCode del tipo de las claves para evitar comportamientos incorrectos, como claves repetidas. Asimismo, al igual que pasa en el tipo *Set*, los objetos que se introducen en un Map son "vistas" o referencias a objetos y, por tanto, si estos cambian, el Map puede dejar de ser coherente. En general, deben usarse claves que sean tipos inmutables.
- Un Map cuyas claves sean String y cuyos valores sean Integer se definiría e inicializaría de la siguiente forma:

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

1. El tipo Map

- Constructores de HashMap:
  - HashMap()
    - Construye un HashMap vacío
  - HashMap(Map<? extends K,? extends V> m)
    - Construye un Map con los mismos pares que el Map que se le pasa como argumento (equivale a crear un Map vacío y aplicar putAll)
- Existe una clase LinkedHashMap, que se comporta igual que HashMap excepto en que los iteradores sobre las claves, los valores o los pares, los devuelven en el orden en que se insertaron.
- La operación *equals* entre dos Map devuelve true si y solo si los conjuntos devueltos por *entrySet* en ambos conjuntos (sus pares) son iguales.

ÍNDICE

2. Métodos del tipo Map

- `void clear()`
  - Elimina todos los elementos (pares o entradas) del Map.
- `boolean containsKey(Object key)`
  - Devuelve true si el Map contiene la clave especificada
- `boolean containsValue(Object value)`
  - Devuelve true si una o más claves del Map tienen asociadas el valor especificado.
- `boolean isEmpty()`
  - Devuelve true si el Map no contiene ningún par.

2. Métodos del tipo Map

- `V get(Object key)`
  - Devuelve el valor asociado con la clave especificada o null si esa clave no está asociada con ningún valor (la clave no está en el conjunto de claves).
- `Set<K> keySet()`
  - Devuelve un Set que es una vista de las claves que contiene el Map.
- `V put(K key, V value)`
  - Asocia el valor con la clave especificada. Devuelve el valor previamente asociado con la clave si esta ya estaba en el Map o null, en caso contrario
- `V remove(Object key)`
  - Elimina el par que tiene como clave el parámetro especificado. Devuelve el valor previamente asociado con la clave, o null si la clave no existía.

2. Métodos del tipo Map

- `int size()`
  - Devuelve el número de pares del Map.
- `Collection<V> values()`
  - Devuelve una Collection que es una vista de los valores del Map.
- `Set<Map.Entry<K,V>> entrySet()`
  - Devuelve una vista del conjunto de todos los pares del Map (el tipo Map.Entry se verá más adelante).
- `void putAll(Map<? extends K, ? extends V> m)`
  - Añade al Map todos los pares contenidos en m. Tiene el mismo efecto que hacer put de todos los elementos de m.

ÍNDICE

3. El tipo Map.Entry

- Los pares de elementos (también llamados entradas) de los que está compuesto un Map<K, V> son de un tipo que viene implementado por la interfaz Map.Entry<K, V>. Esta interfaz, aparte de la operación `equals` que permite comparar pares para comprobar su igualdad (y el correspondiente `hashCode`), tiene tres operaciones:
  - `K` **`getKey`**`()`
    - Devuelve la clave del par.
  - `V` **`getValue`**`()`
    - Devuelve el valor del par.
  - `V` **`setValue`**`(V value)`
    - Modifica el valor del par, dándole como nuevo valor value. Devuelve el valor (segunda componente) previo del par.

ÍNDICE

4. Inicialización de un objeto de tipo Map

1. Creación del objeto de tipo Map
2. Para cada *clave*
    1. Si la *clave* ya está en el Map (*containsKey*)
        1. Obtener el *valor* asociado a esa clave (*get*)
        2. Actualizar el *valor* u obtener *nuevovalor* a partir de *valor*
        3. Si es necesario, añadir al Map el par *clave*, *nuevovalor* (*put*)
    2. Si la *clave* no está en el Map
        1. Inicializar *valor*
        2. Añadir al Map el par *clave*, *valor* (*put*)

4. Inicialización de un objeto de tipo Map

Calcular la frecuencia absoluta de aparición o número de veces que aparecen los caracteres en un String. Para ello se define un Map cuyas claves son de tipo Character y cuyos valores son de tipo Integer. El método recibirá un String y devolverá un objeto de tipo Map<Character, Integer>. El código del método sería:

```
public static Map<Character, Integer> contadorCarac(String frase) {
    Map<Character, Integer> contador = new HashMap<Character, Integer>();
    frase = frase.toUpperCase(); // todos los caracteres en mayúsculas
    for (int i = 0; i < frase.length(); i++) {
       Character caracter = frase.charAt(i);
       if (!contador.containsKey(caracter)) {
                        contador.put(caracter, 1);
       } else {
                        Integer valor = contador.get(caracter);
                        valor++;
                        contador.put(caracter, valor);
             }
    }
    return contador;
}
```

4. Inicialización de un objeto de tipo Map

Obtener un índice de las posiciones que ocupan las cadenas de la lista palabras mediante una lista de enteros.
"la palabra que más aparece en este texto es la palabra palabra" => {"que"=[2], "aparece"=[4], "este"=[6], "texto"=[7], "palabra"=[1, 10, 11], "la"=[0, 9], "en"=[5], "más"=[3], "es"=[8]}

```
public static Map<String, List<Integer>> indicePalabras(List<String> palabras) {
   Map<String, List<Integer>> indice = new HashMap<String, List<Integer>>();
   int pos = 0;
   for (String palabra : palabras) {
      if (indice.containsKey(palabra)) {
                     indice.get(palabra).add(pos);
      } else {
         List<Integer> lista = new ArrayList<Integer>();
         lista.add(pos);
         indice.put(palabra, lista);
      }
      pos++;
   }
   return indice;
}
```

ÍNDICE

5. El tipo SortedMap

- El tipo SortedMap es un subtipo de Map en el que el conjunto de las claves está ordenado.
- La clase que implementa SortedMap es TreeMap.
- Es necesario que el tipo de las claves tenga un orden natural (es decir, que implemente Comparable), o bien que se indique el orden mediante un comparador.

```
SortedMap<String, List<Integer>> indice = new TreeMap<String, List<Integer>>();
```

5. El tipo SortedMap

- La clase *TreeMap* tiene varios constructores:
  - Sin argumentos: construye un *SortedMap* vacío que utilizará el orden natural de las claves.
  - Con un argumento de tipo *Comparator*: construye un *SortedMap* vacío que utilizará el orden inducido por el comparador.
  - Con un argumento de tipo *Map*: construye un *SortedMap* con los mismos pares que el *Map* que recibe como argumento, pero donde las claves estarán ordenadas según el orden natural de estas.
  - Con un argumento de tipo *SortedMap*: construye un *SortedMap* con los mismos elementos que el que recibe como argumento y ordenado según el mismo criterio (sea el natural o uno inducido). Es el constructor copia.