Clase 13: Map

Programación y Laboratorio III

Agenda

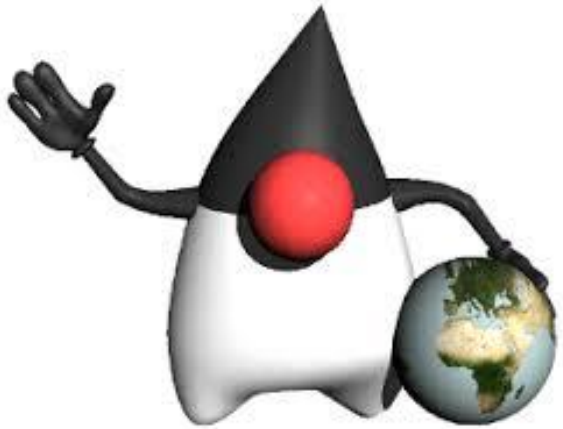
- **Interfaz Map**
- HashMap
- Principio de Hashing
- equals() & hashCode()
- HashMap vs. Hashtable
- LinkedHashMap
- TreeMap

Interfaz Map

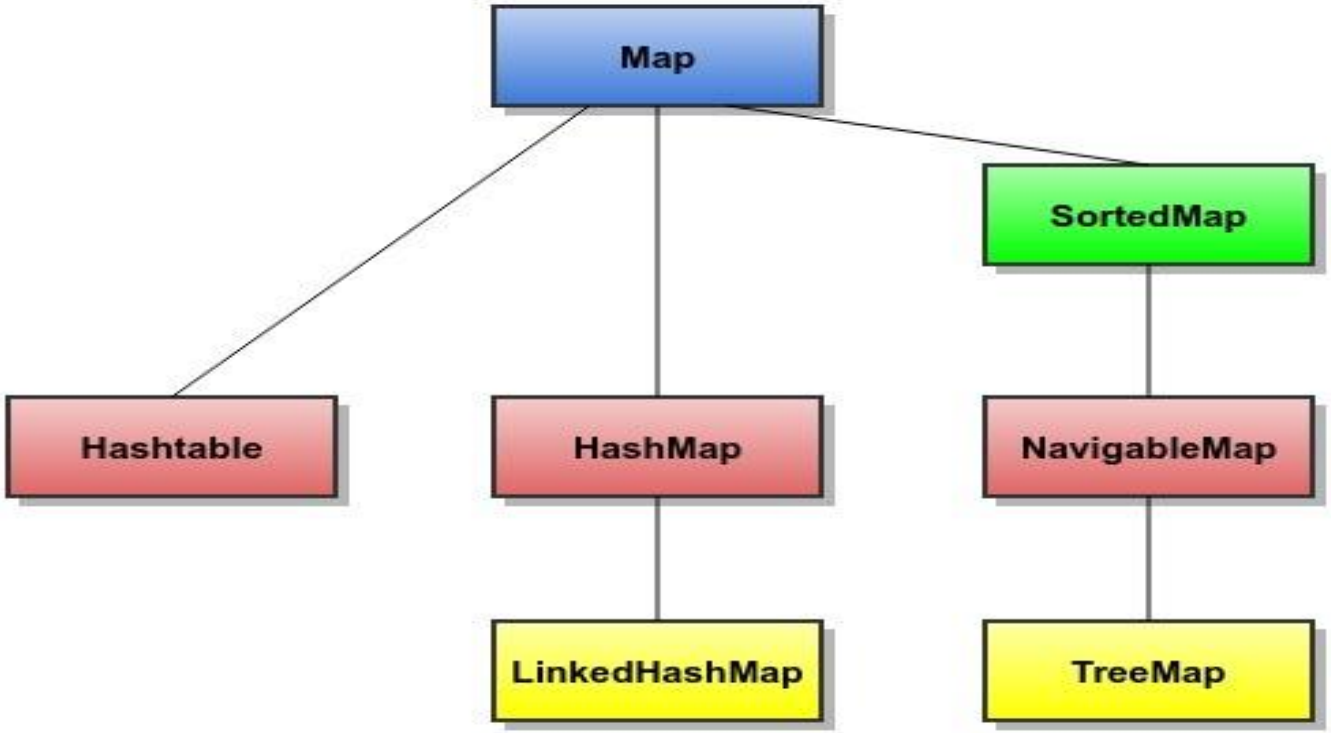
- Nos permite representar una estructura de datos para almacenar pares “clave-valor”.
- Para una clave sólo tenemos un valor.
- Map tienen implementada por debajo toda la teoría de las estructuras de datos de árboles, por lo tanto permiten añadir, eliminar y modificar elementos de forma transparente.
- La clave funciona como un identificador único y no se admiten claves duplicadas.

Map - Operaciones

- boolean containsKey(Object key)
- boolean containsValue(Object var1);
- V get(Object var1);
- V put(K var1, V var2);
- V remove(Object var1);
- int size();
- boolean isEmpty();



Interfaz Map



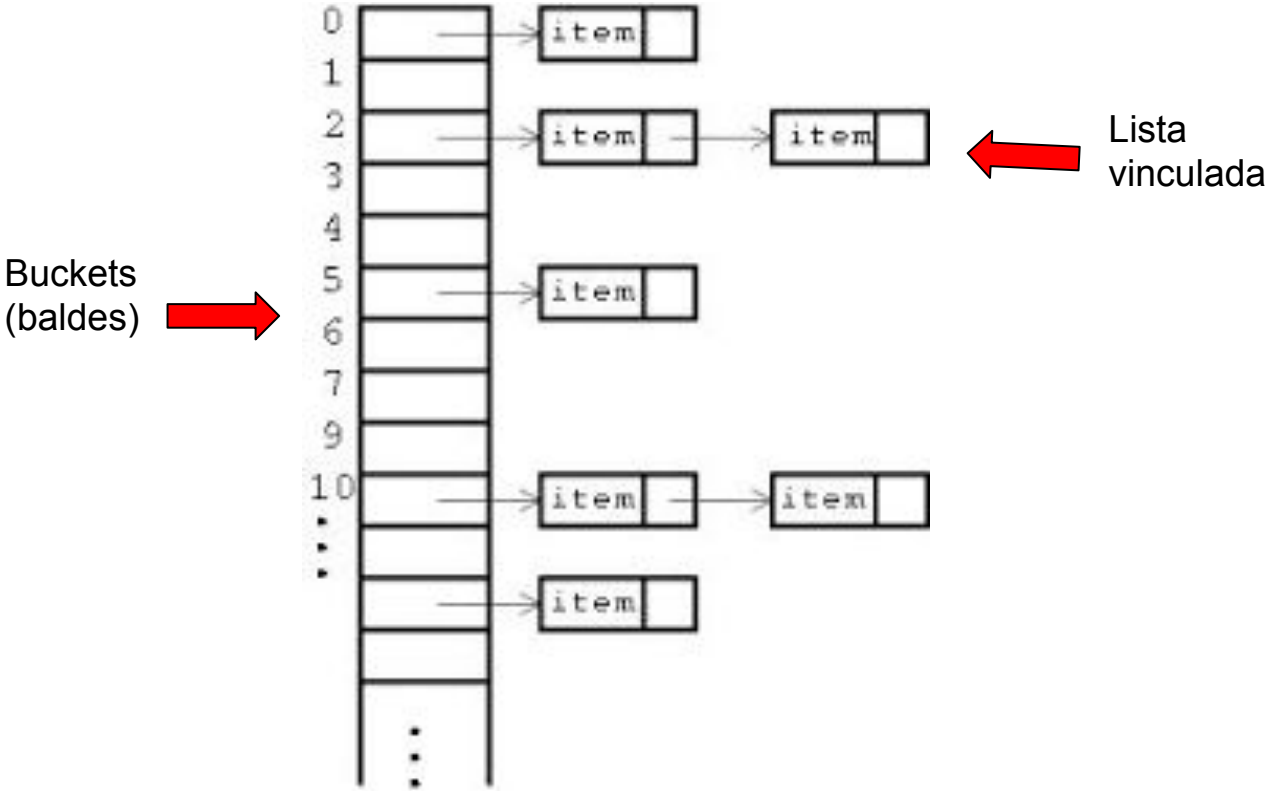
Agenda

- ~~Interfaz Map~~
- **HashMap**
- Principio de Hashing
- equals() & hashCode()
- HashMap vs. Hashtable
- LinkedHashMap
- TreeMap

HashMap

- Un HashMap es una colección de objetos, como los arrays, pero estos no tienen orden.
- Cada objeto se identifica mediante algún identificador y conviene que sea inmutable (`final`), de modo que no cambie en tiempo de ejecución.
- El nombre Hash hace referencia a una técnica de organización de archivos llamada hashing o “dispersión” en el cual se almacenan registros en una dirección del archivo que es generada por una función que se aplica a la clave del mismo.
- Los elementos que se insertan en un HashMap no tendrán un orden específico.
- Permite una clave null.

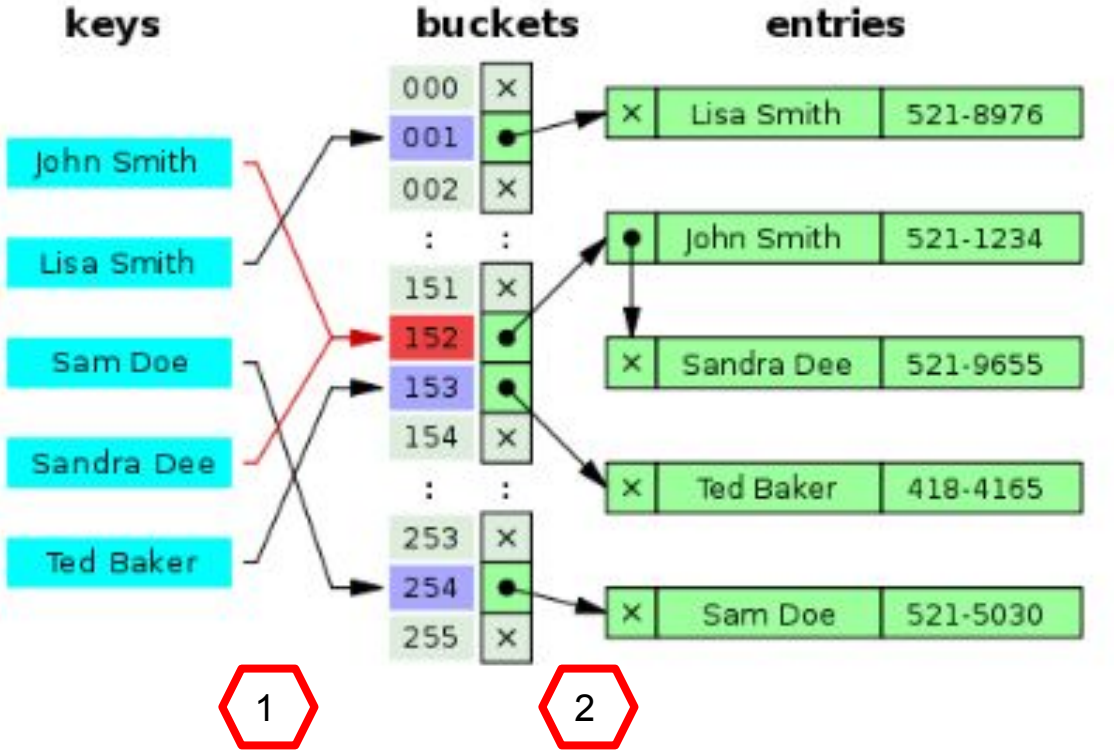
HashMap - Estructura



Agenda

- ~~Interfaz Map~~
- ~~HashMap~~
- **Principio de Hashing**
- equals() & hashCode()
- HashMap vs. Hashtable
- LinkedHashMap
- TreeMap

Principio de Hashing



Principio de Hashing (2)

1. Se utiliza el método `hashCode()` para encontrar el bucket correspondiente.

2. Se utiliza el método equals() para buscar el valor correspondiente a la clave dada.

- Dos objetos idénticos tendrán el mismo identificador retornado por el método hashCode().

Agenda

- ~~Interfaz Map~~
- ~~HashMap~~
- ~~Principio de Hashing~~
- **equals() & hashCode()**
- HashMap vs. Hashtable
- LinkedHashMap
- TreeMap

`equals()` & `hashCode()`

NO OLVIDAR:

“Si dos objetos son iguales usando equals(), entonces la invocación a hashCode() de ambos objetos debe retornar el mismo valor.”

equals() & hashCode() (2)

1) **Objetos que son iguales pero retornan diferentes hashCodes.**

Cada uno de estos baldes tiene asignado un número que lo identifica. Cuando agregamos un valor al HashMap, almacena el dato en uno de esos baldes. El balde que se usa depende del hashCode que devuelva el objeto a ser almacenado. Por ejemplo, si el método hashCode() del objeto retorna 49, entonces se almacena en el balde 49 dentro del HashMap.

Más tarde, cuando verifiquemos si la colección contiene al elemento invocando el método contains(elemento), el HashMap primero obtiene el hashCode de ese "elemento". Luego buscará el balde que corresponde a ese hashCode. Si el balde está vacío, significa que el HashMap no contiene al elemento y devuelve false.

equals() & hashCode() (3)

2) **Objetos que no son iguales pero retornan el mismo hashCode.**

Se utiliza este mecanismo de "baldes" por un tema de eficiencia. Si todos los objetos que se agregan a un HashMap se almacenaran en una única lista grande, entonces tendríamos que comparar la entrada con todos los objetos de la lista para determinar si un elemento en particular está contenido en el Map. Como se usan baldes, sólo se comparan los elementos del balde específico, y en general cada balde sólo almacena una pequeña cantidad de elementos en el HashMap.

Agenda

- ~~Interfaz Map~~
- ~~HashMap~~
- ~~Principio de Hashing~~
- ~~equals() & hashCode()~~
- **HashMap vs. Hashtable**
- LinkedHashMap
- TreeMap

HashMap vs. Hashtable

- La principal diferencia entre uno y otro es la sincronización interna del objeto. Para aplicaciones multihilos es preferible elegir Hashtable sobre HashMap, que no tiene sincronización.
- HashMap es mejor en cuanto a performance.
- Hashtable no admite valores nulos en ninguna de sus partes, mientras que HashMap sí.

Agenda

- ~~Interfaz Map~~
- ~~HashMap~~
- ~~Principio de Hashing~~
- ~~equals() & hashCode()~~
- ~~HashMap vs. Hashtable~~
- **LinkedHashMap**
- TreeMap

LinkedHashMap

- Similar a HashMap pero con la diferencia que mantiene una lista doblemente vinculada, además del array de baldes.
- La lista doblemente vinculada define el orden de iteración de los elementos

Agenda

- ~~Interfaz Map~~
- ~~HashMap~~
- ~~Principio de Hashing~~
- ~~equals() & hashCode()~~
- ~~HashMap vs. Hashtable~~
- ~~LinkedHashMap~~
- **TreeMap**

TreeMap

- Implementado mediante un árbol binario y permite tener un mapa ordenado.
- Cuando iteramos un objeto TreeMap los objetos son extraídos en forma ascendente según sus keys.
- TreeMap no sabe cómo ordenar la colección cuando se utiliza una key creada por el programador, sólo lo hace si trabajamos con objetos tipo String, Integer, etc.

Bibliografía oficial

- https://docs.oracle.com/javase/8/docs/api/java/util/Map.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html
-