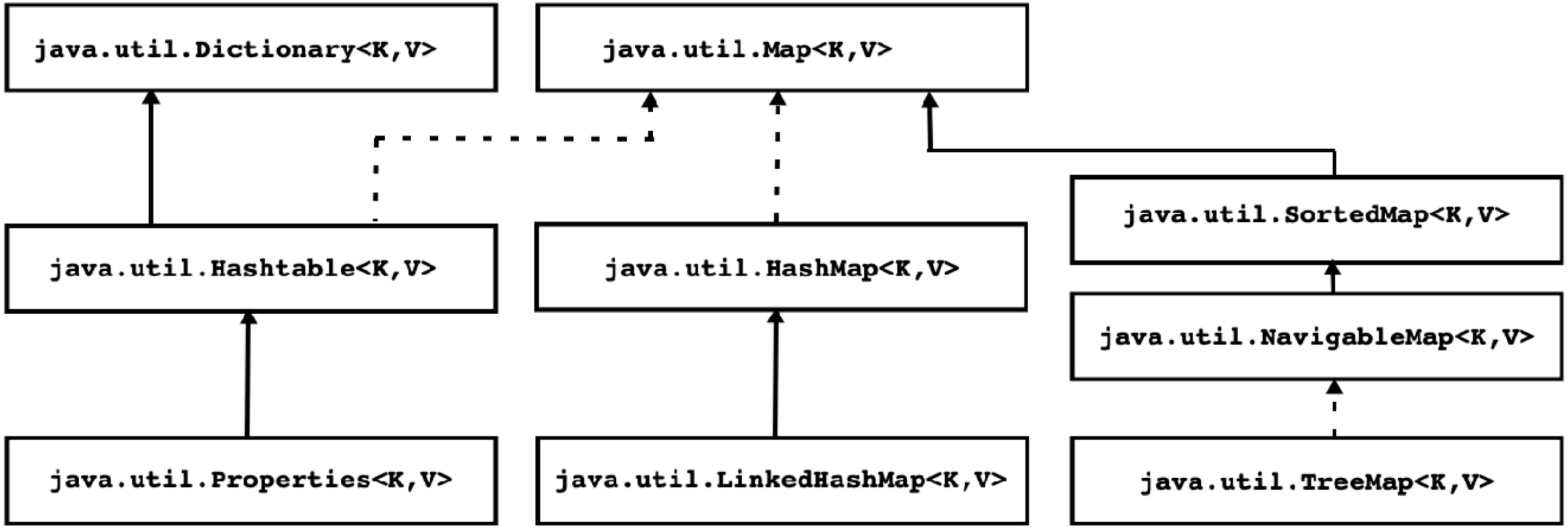
CORE JAVA

Akshita Chanchlani

Map Interface Hierarchy



- Map is not considered to be a true collection, as the Map interface does not extend the *Collection* interface.
- *Map* is not a true collection, its characteristics and behaviors are different than the other collections like *List* or *Set*.

Dictionary<K,V>

- It is abstract class declared in java.util package.
- It is super class of Hashtable.
- It is used to store data in key/value pair format.
- It is not a part of collection framework
- It cannot contain duplicate keys and each key can map to at most one value
- Methods:

    1. public abstract boolean isEmpty()
    2. public abstract V put(K key, V value)
    3. public abstract int size()
    4. public abstract V get(Object key)
    5. public abstract V remove(Object key)
    6. public abstract Enumeration<K> keys()
    7. public abstract Enumeration<V> elements()

- Implementation of Dictionary is Obsolete.

Map<K,V>

- It is part of collection framework but it doesn't extend Collection interface.
- This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.
- HashMap, Hashtable, TreeMap etc are Map collection’s.
- Map collection stores data in key/value pair format.
- In map we can not insert duplicate keys but we can insert duplicate values.
- It maps keys to values, or is a collection of Key-Value pairs
- Map.Entry<K,V> is nested interface of Map<K,V>.
- Following are abstract methods of Map.Entry interface.

  1. K getKey()
  2. V getValue()
  3. V setValue(V value)

Map<K,V>

- Abstract Methods of Map<K,V>

  1. boolean isEmpty()
  2. V put(K key, V value)
  3. void putAll(Map<? extends K,? extends V> m)
  4. int size()
  5. boolean containsKey(Object key)
  6. boolean containsValue(Object value)
  7. V get(Object key)
  8. V remove(Object key)
  9. void clear()
  10. Set<K> keySet()
  11. Collection<V> values()
  12. Set<Map.Entry<K,V>> entrySet()

- An instance, whose type implements Map.Entry<K,V> interface is called enrty instance.

When to use Map<K,V>

Some implementations allow null key and null value (*HashMap* and *LinkedHashMap*) but some does not (*TreeMap*).

The order of a map depends on specific implementations,
e.g TreeMap and *LinkedHashMap* have predictable order, while *HashMap* does not.

Maps are perfect for key-value association mapping such as dictionaries. Use Maps when you want to retrieve and update elements by keys, or perform look-ups by keys.

- A map of zip codes and cities.
- A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
- A map of classes and students. Each class (key) is associated with a list of students (value).

```
Map<Integer, String> hashMap = new HashMap<>();
Map<Integer, String> linkedHashMap = new LinkedHashMap<>();
Map<Integer, String> treeMap = new TreeMap<>();
```

Characterstics Map<K,V>

- HashMap: this implementation uses a hash table as the underlying data structure. It implements all of the Map operations and allows null values and one null key. This class is roughly equivalent to Hashtable - a legacy data structure before Java Collections Framework, but it is not synchronized and permits nulls. HashMap does not guarantee the order of its key-value elements. Therefore, consider to use a HashMap when order does not matter and nulls are acceptable.
- LinkedHashMap: this implementation uses a hash table and a linked list as the underlying data structures, thus the order of a LinkedHashMap is predictable, with insertion-order as the default order. This implementation also allows nulls like HashMap. So consider using a LinkedHashMap when you want a Map with its key-value pairs are sorted by their insertion order.
- TreeMap: this implementation uses a red-black tree as the underlying data structure. A TreeMap is sorted according to the natural ordering of its keys, or by a Comparator provided at creation time. This implementation does not allow nulls. So consider using a TreeMap when you want a Map sorts its key-value pairs by the natural order of the keys (e.g. alphabetic order or numeric order), or by a custom order you specify.So far you have understood the key differences of the 3 major Map’s implementations. And the code examples in this tutorial are around them.

Hashtable<K,V>

- It is Map<K,V> collection which extends Dictionary class.
- It can not contain duplicate keys but it can contain duplicate values.
- In Hashtable, Key and value can not be null.
- It is synchronized collection.
- It is introduced in jdk 1.0
- In Hashtable, if we want to use instance non final type as key then it should override equals and hashCode method.

HashMap<K,V>

- It is map collection
- It's implementation is based on Hashtable.
- It can not contain duplicate keys but it can contain duplicate values.
- In HashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method, we can make it synchronized.
  - ➢ Map m = Collections.synchronizedMap(new HashMap(...));
- It is introduced in jdk 1.2.
- Instantiation
  - ➢ Map<Integer, String> map = new HashMap<>( );
- **Note : In HashMap, if we want to use element of non final type as a key then it should override equals() and hashCode() method.**

LinkedHashMap<K,V>

- It is sub class of HashMap<K,V> class
- Its implementation is based on LinkedList and Hashtable.
- It is Map collection hence it can not contain duplicate keys but it can contain duplicate values.
- In LinkedHashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method we can make it synchronized.
  - ➢ **Map m = Collections.synchronizedMap(new LinkedHashMap(...));**
- LinkedHashMap maintains order of entries according to the key.
- Instantiation:
  - ➢ **Map<Integer, String> map = new LinkedHashMap<>();**
- It is introduced in jdk 1.4

TreeMap<K,V>

- It is map collection.
- It can not contain duplicate keys but it can contain duplicate values.
- It TreeMap, key not be null but value can be null.
- Implementation of TreeMap is based on Red-Black Tree.
- It maintains entries in sorted form according to the key.
- It is unsynchronized collection. Using Collections.synchronizedSortedMap() method, we can make it synchronized.
  - **SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));**
- Instantiation:
  - **Map<Integer, String> map = new TreeMap<>();**
- It is introduced in jdk 1.2
- Note : In TreeMap, if we want to use element of non final type as a key then it should implement Comparable interface.

SUNBEAM

Thank You.