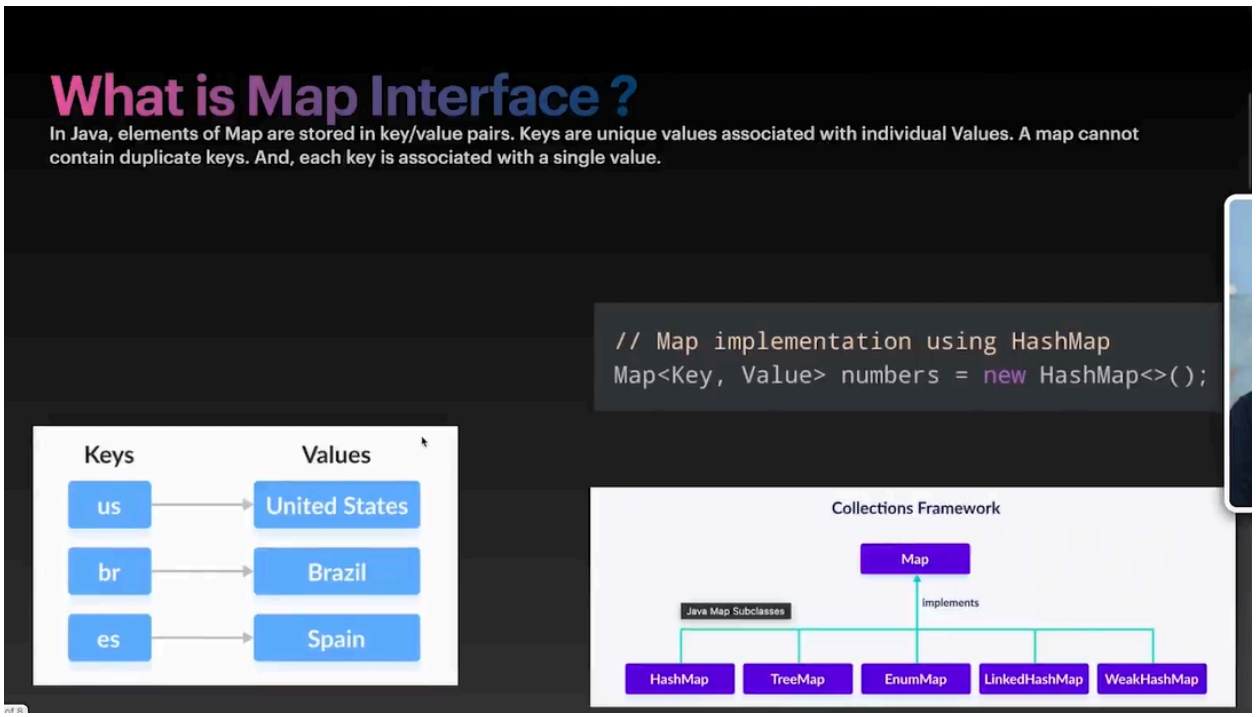
Java Collection Framework-bonus

INSTRUCTOR: Love Babbar

Module 2

27 November 2024



Key should be unique

It maintains 3 states

1. Key
2. Value
3. Key value pair

Map Characteristics:

Here are some key characteristics of the Map interface:

- No Duplicate Keys: Each key can map to at most one value. However, different keys can map to the same value.
- Key-Value Association: It maintains an association of keys to values.
- Implementations: Some of the well-known classes that implement the Map interface are HashMap, TreeMap, LinkedHashMap, and Hashtable.
- Order: The Map interface itself doesn't guarantee any specific order of its elements. However, some specific implementations like TreeMap maintain a sorted order, and LinkedHashMap maintains the insertion order.
- Null Values: Maps allow null values and, depending on the implementation, null keys. For example, HashMap allows one null key and multiple null values, but Hashtable does not allow null keys or values.

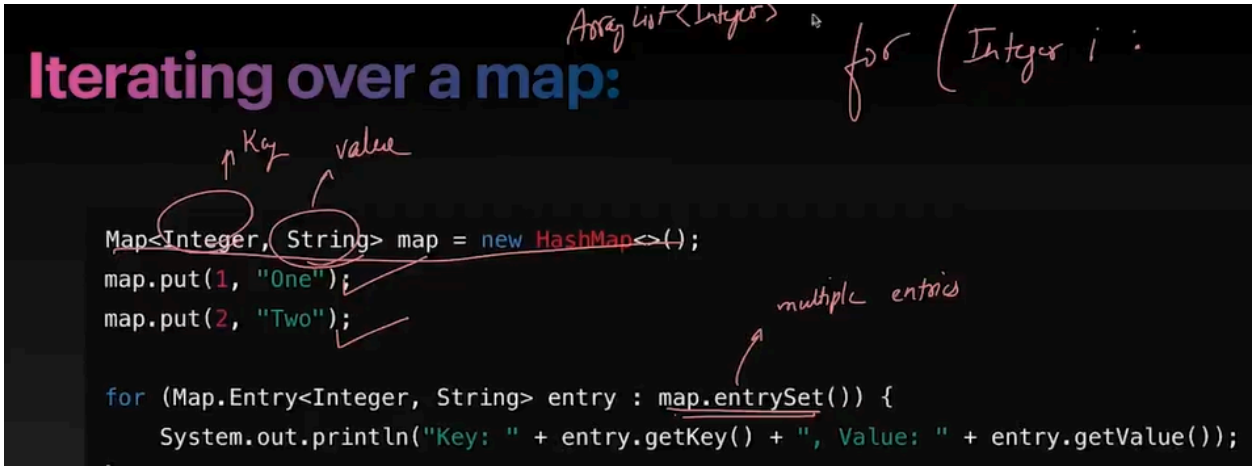
Map Methods:

- put(K, V) - Inserts the association of a key K and a value V into the map. If the key is already present, the new value replaces the old value.
- putAll() - Inserts all the entries from the specified map to this map.
- putIfAbsent(K, V) - Inserts the association if the key K is not already associated with the value V.
- get(K) - Returns the value associated with the specified key K. If the key is not found, it returns null.
- getOrDefault(K, defaultValue) - Returns the value associated with the specified key K. If the key is not found, it returns the defaultValue.
- containsKey(K) - Checks if the specified key K is present in the map or not.
- containsValue(V) - Checks if the specified value V is present in the map or not.

Map Methods:

- replace(K, V) - Replace the value of the key K with the new specified value V.
- replace(K, oldValue, newValue) - Replaces the value of the key K with the new value newValue only if the key K is associated with the value oldValue.
- remove(K) - Removes the entry from the map represented by the key K.
- remove(K, V) - Removes the entry from the map that has key K associated with value V.
- keySet() - Returns a set of all the keys present in a map.
- values() - Returns a set of all the values present in a map.
- entrySet() - Returns a set of all the key/value mapping present in a map.

Iterating over a map:



Comparable Interface:

- Purpose: Defines a natural ordering for the objects of the classes that implement it.
- Method to Implement: compareTo(T o)
  - Functionality: This method compares the current object with the specified object to determine their order.
  - Return Value: Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object, respectively.
- Usage Context: Useful when there is a single, natural ordering of the objects (e.g., alphabetical order for strings, numerical order for numbers).
- Integration: Automatically used by sorting methods in collections that do not specify a custom comparator (e.g., Collections.sort(list) when sorting a list of objects that implement Comparable).

Comparator Interface:

Priority Queue → (a, b) → b − a

- Purpose: Provides a way to define a custom ordering of objects, separate from their natural ordering.
- Method to Implement: compare(T o1, T o2)
  - Functionality: Compares its two arguments for order.
  - Return Value: Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
- Usage Context: Ideal when you need multiple different ways of ordering objects, or when objects do not have a natural ordering.
- Flexibility: Allows specifying the order externally, which is useful for sorting methods when you want to sort based on attributes that are not considered in natural ordering.
- Integration: Used by providing an instance of Comparator to sorting methods, such as Collections.sort(list, comparator) or Arrays.sort(array, comparator).