Java Map Interface

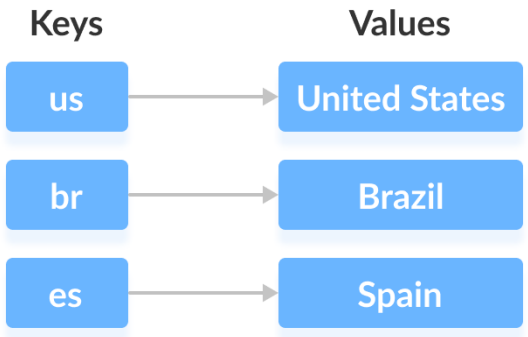
In this tutorial, we will learn about the Java Map interface and its methods.

The `Map` interface of the Java collections framework provides the functionality of the map data structure.

Working of Map

In Java, elements of `Map` are stored in **key/value** pairs. **Keys** are unique values associated with individual **Values**.

A map cannot contain duplicate keys. And, each key is associated with a single value.



We can access and modify values using the keys associated with them.

In the above diagram, we have values: `United States`, `Brazil`, and `Spain`. And we have corresponding keys: `us`, `br`, and `es`.

Now, we can access those values using their corresponding keys.

Note: The `Map` interface maintains 3 different sets:

- the set of keys
- the set of values
- the set of key/value associations (mapping).

Hence we can access keys, values, and associations individually.

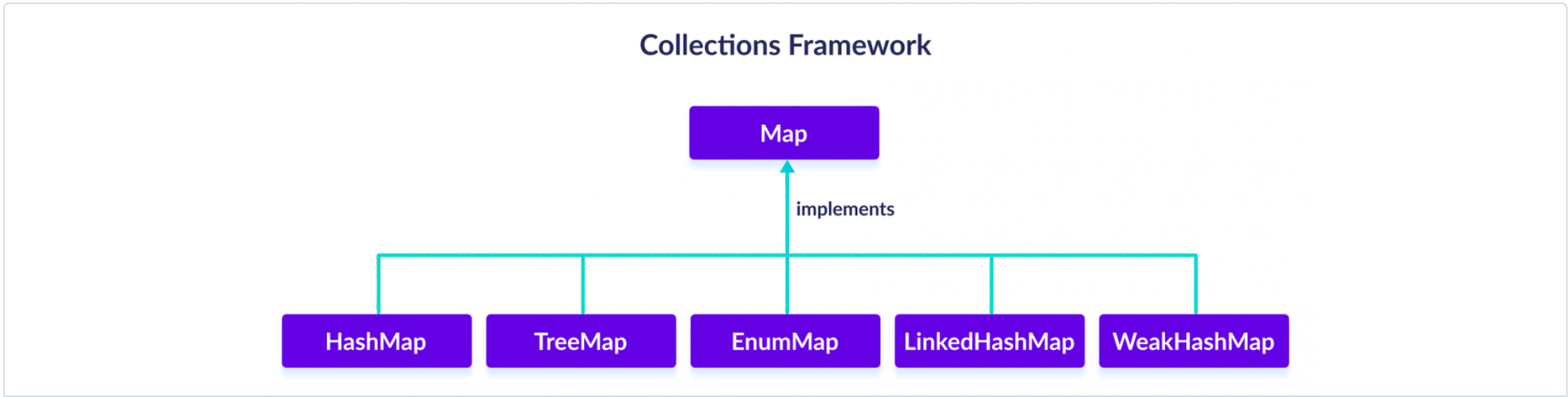
Classes that implement Map

Since `Map` is an interface, we cannot create objects from it.

In order to use functionalities of the `Map` interface, we can use these classes:

- HashMap
- EnumMap
- LinkedHashMap
- WeakHashMap
- TreeMap

These classes are defined in the collections framework and implement the `Map` interface.

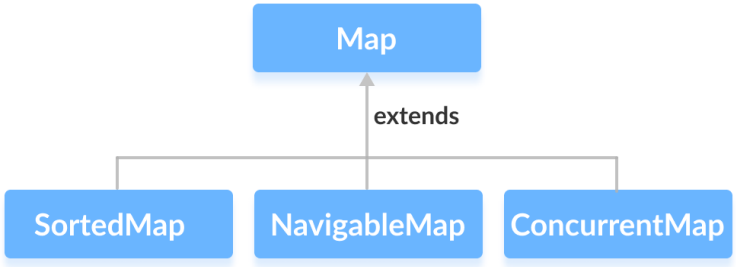


Java Map Subclasses

Interfaces that extend Map

The `Map` interface is also extended by these subinterfaces:

- SortedMap
- NavigableMap
- ConcurrentMap



Java Map Subinterfaces

How to use Map?

In Java, we must import the `java.util.Map` package in order to use `Map`. Once we import the package, here's how we can create a map.

```
// Map implementation using HashMap
Map<Key, Value> numbers = new HashMap<>();
```

In the above code, we have created a `Map` named `numbers`. We have used the `HashMap` class to implement the `Map` interface.

Here,

- `Key` - a unique identifier used to associate each element (value) in a map
- `Value` - elements associated by keys in a map

Methods of Map

The `Map` interface includes all the methods of the `Collection` interface. It is because `Collection` is a super interface of `Map`.

Besides methods available in the `Collection` interface, the `Map` interface also includes the following 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.
- **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.

Implementation of the Map Interface

1. Implementing HashMap Class

```java
import java.util.Map;
import java.util.HashMap;

class Main {

    public static void main(String[] args) {
        // Creating a map using the HashMap
        Map<String, Integer> numbers = new HashMap<>();

        // Insert elements to the map
        numbers.put("One", 1);
        numbers.put("Two", 2);
        System.out.println("Map: " + numbers);

        // Access keys of the map
        System.out.println("Keys: " + numbers.keySet());

        // Access values of the map
        System.out.println("Values: " + numbers.values());

        // Access entries of the map
        System.out.println("Entries: " + numbers.entrySet());

        // Remove Elements from the map
        int value = numbers.remove("Two");
        System.out.println("Removed Value: " + value);
    }
}
```

Output

```
Map: {One=1, Two=2}
Keys: [One, Two]
Values: [1, 2]
Entries: [One=1, Two=2]
Removed Value: 2
```

To learn more about `HashMap`, visit Java HashMap.

---

2. Implementing TreeMap Class

```java
import java.util.Map;
import java.util.TreeMap;

class Main {

    public static void main(String[] args) {
        // Creating Map using TreeMap
        Map<String, Integer> values = new TreeMap<>();

        // Insert elements to map
        values.put("Second", 2);
        values.put("First", 1);
        System.out.println("Map using TreeMap: " + values);

        // Replacing the values
        values.replace("First", 11);
        values.replace("Second", 22);
        System.out.println("New Map: " + values);

        // Remove elements from the map
        int removedValue = values.remove("First");
        System.out.println("Removed Value: " + removedValue);
    }
}
```

Output

```
Map using TreeMap: {First=1, Second=2}
New Map: {First=11, Second=22}
Removed Value: 11
```