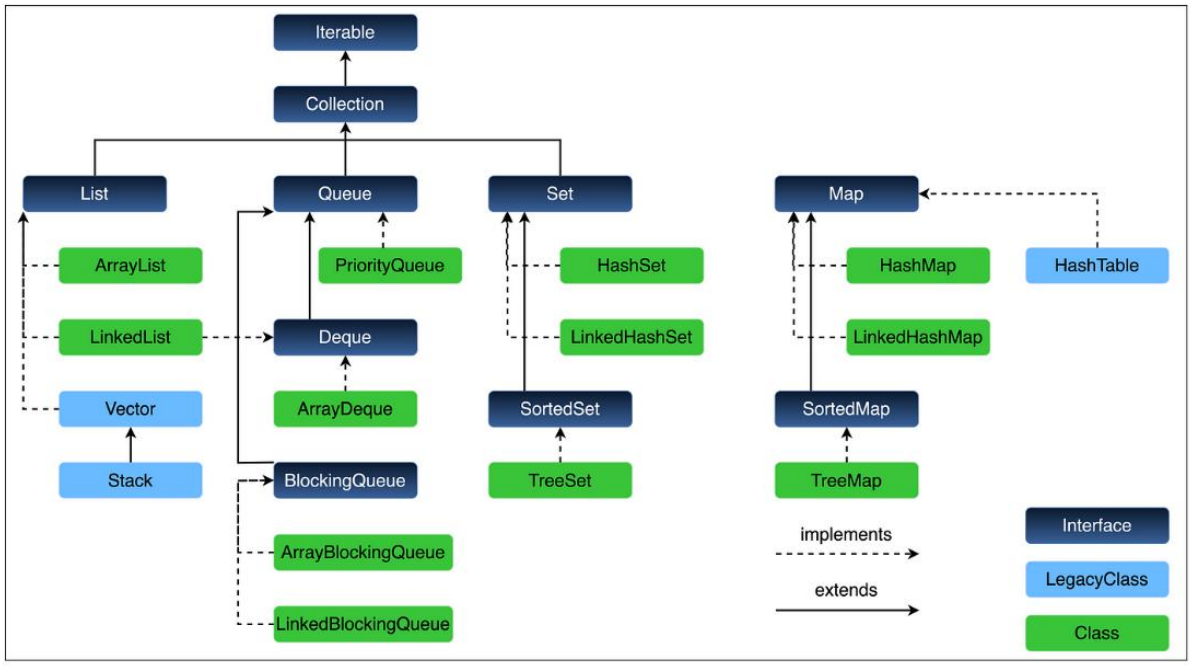
13.4-Map

The Map interface is part of the Java Collection Framework, although it does not extend the Collection interface. In Java, a Map represents a collection of key-value pairs, where each unique key maps to a specific value. This concept is similar to a telephone directory, where a person's name (key) is associated with a phone number (value). When you provide a key, the corresponding value can be retrieved.



👉Key Features of Map Interface

- **Mapping between Keys and Values**: The Map interface allows for the association of unique keys with specific values.
- **Unique Keys**: Each key in a Map must be unique, while values can be duplicated.
- **Not a Subtype of Collection Interface**: Unlike other collection types, Map is not a subtype of the Collection interface, so it behaves differently.

👉Creating Map Objects

Since Map is an interface, you cannot create objects of the Map type directly. You must use a class that implements the Map interface to create an object, such as HashMap, LinkedHashMap, or TreeMap.

👉Example Syntax for Creating a Map

```
Map<ObjectType1, ObjectType2> mapName = new HashMap<>();
```

- The ObjectType1 represents the type of the keys, while ObjectType2 represents the type of the values.

👉Example: Storing Student Names and Marks

Here's how you can create a Map to store student names (as keys) and their marks (as values):

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

public class Demo {
    public static void main(String[] args) {
        Map<String, Integer> students = new HashMap<>();
        students.put("Navin", 56);
        students.put("Harsh", 65);
        students.put("Sushil", 73);
        students.put("Kiran", 96);

        System.out.println(students);
    }
}
```

👉Output:

```
{Navin=56, Harsh=65, Sushil=73, Kiran=96}
```

👉Explanation

- We created a Map object named students using HashMap, specifying the key type as String and the value type as Integer.
- The put() method is used to add key-value pairs to the map.
- When we print the students map, it displays the names and their corresponding marks.

👉Working with Map Elements

- **Accessing Values**: You can retrieve a value from the map using the get() method by providing the corresponding key. For example:

```
System.out.println(students.get("Harsh"));
```

👉**Output**: 65

- **Handling Duplicate Keys**: If you add a key that already exists in the map, the new value will replace the old value. For instance:

```
students.put("Harsh", 45);

System.out.println(students);
```

👉**Output**:

```
{Navin=56, Harsh=45, Sushil=73, Kiran=96}
```

As you can see, the value associated with the key "Harsh" was updated from 65 to 45.

👉Iterating Over a Map

To separate keys and values or iterate through the map, you can use a for-each loop with methods like keySet() to get all keys.

👉Example: Iterating Over Keys and Values

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

public class Demo {
    public static void main(String[] args) {
        Map<String, Integer> students = new HashMap<>();
        students.put("Navin", 56);
        students.put("Harsh", 45);
        students.put("Sushil", 73);
        students.put("Kiran", 96);

        for (String name : students.keySet()) {
            System.out.println(name + ": " +
students.get(name));
        }
    }
}
```

👉Output:

```
Navin: 56

Harsh: 45

Sushil: 73

Kiran: 96
```

- The **keySet()** method retrieves only the keys from the map, and we use the get() method to get corresponding values.

👉Important Methods of the Map Interface

Below is a list of commonly used methods in the Map interface along with their descriptions:

Method	**Description**	**Time Complexity**
put(Key, Value)	Associates the specified key with the specified value in the map.	O(1)
get(Key)	Returns the value associated with the specified key.	O(1)
containsKey(Key)	Checks if the map contains the specified key. Returns true if it does, otherwise false.	O(1)
containsValue(Value)	Checks if the map contains the specified value. Returns true if it does, otherwise false.	O(n)
isEmpty()	Returns true if the map contains no key-value pairs.	O(1)
clear()	Removes all key-value pairs from the map.	O(n)
remove(Key)	Removes the mapping for a key from this map if it is present.	O(1)
size()	Returns the number of key-value mappings in the map.	O(1)

Method	Description	Time Complexity
entrySet()	Returns a Set view of the mappings contained in this map.	O(n)
keySet()	Returns a Set view of the keys contained in this map.	O(n)
values()	Returns a collection view of the values contained in this map.	O(n)
putIfAbsent(Key, Value)	Associates the specified value with the specified key if the key is not already associated.	O(1)
replace(Key, Value)	Replaces the value for the specified key with the new value if the key exists.	O(1)
replace(Key, OldValue, NewValue)	Replaces the value for the specified key only if it is currently mapped to the old value.	O(1)

👉Hashtable vs. HashMap

- **HashMap**: Allows null keys and values, and is not synchronized (not thread-safe).
- **Hashtable**: Does not allow null keys or values, and is synchronized (thread-safe).

👉Frequently Asked Questions (FAQs)

Q1. What is a Map Interface in Java?

- A Map interface is a collection of key-value pairs where each key is unique, and you can retrieve values using their corresponding keys.

Q2. What are the types of Map Interface implementations in Java?

- The main implementations of the Map interface are HashMap, LinkedHashMap, and TreeMap.