MAP

(1) KEY CHARACTERISTICS OF MAP INTERFACE?

- Map is an interface.
- Map is not a part of collection framework.
- Key-Value Pair: Each Entry in map consist of key and value.
- Unique Keys : No two entry can have the same key.
- One Value per key: Each Key map to a single value.
- Order : Some Implementations maintain insertion order like (LinkedHashMap) , natural Order (TreeMap) , or no order (HashMap).

HASHMAP

(1) HASHMAP KEY CHARACTERISTICS ?

- UnOrdered : doesn't maintain any order of it's elements.
- Allow null key & null values: can have one null key and multiple null values.
- Not synchronized : not thread safe , required external synchronization.
- performance : constant-time performance O(1) for basic operations like get and put.

(2) INTERNAL WORKING OF HASHMAP?

there are 4 component of hashmap:

(a) Key

(b) Value

(c) Bucket

(d) Hash function

HOW DATA STORE IN HASHMAP ?

STEPS 1: Hashing the key ?
first, key is passed through a hash function to generate a unique hash code (integer) this hash code helps to determine where key value pair will be stored in the array (called "bucket").

STEPS 2 : calculating the index?
hash code is used to calculate an index in array using
int index = hashCode % arraySize;
the index decides which bucket will hold this key-value pair
for Eg:- if array is 16 , the key's hash code will be divided by 16 and the remainder will be the index.

STEPS 3 : Storing in the bucket?
the key-value pair is stored in the bucket at the calculated index.each bucket can hold multiple key-value pairs.
(this is called a collision handling mechanism).

HOW HASHMAP RETRIEVES DATA?

when we call get(key) , the hashmap follows these below steps:

(1) Hashing key: called hash function to calculate hash code.
(2) Finding the index: the hashcode is used to find the index of the bucket where the key-value pair is stored.
(3) searching in the bucket: Once the correct bucket is found,it checks for the key in the bucket ,if it finds the key its returns the associated value.

COLLISION :
to handle linked list (threshold=8), when exceed threshold uses Balanced Binary Search Tree (Red Black tree).

HASHMAP RESIZING (REHASHING) ?
hash map has an internal array size, which by default is 16, when the number of elements (key-value) pairs grows and exceeds a certain load factor (0.75) , hash map automatically resizes the array to hold more data, this process called rehashing. the default size of array is 16 , so when more than 12 elements (16 * 0.75) are inserted , the hash map will resize during rehashing the array size will be double, and then all existing

entries will be rehashed if no collusion time complexity O(1) otherwise log(n)

Ex-

```
public static void main(String[] args) {
    HashMap<Integer,String> map = new HashMap<>(17,0.5f);

    // put ()
    map.put(32,"sahil");
    map.put(12,"rahul");
    map.put(22,"meraz");

    System.out.println(map);

    // put () with duplicate key will replace old one with new value
    map.put(12,"rani");

    // get () with key
    String s = map.get(32);

    // containsKey () with key for key checking
    boolean b = map.containsKey(12);

    //  containsValue () with value for value checking
    boolean sahil = map.containsValue("sahil");

    // keySet () for finding keySet
    Set<Integer> keys = map.keySet();

    // iteration over map way 1 using foreach loop:
    for (Integer k : keys) {
        System.out.println(map.get(k));
    }

    // entrySet () for finding entries
    Set<Map.Entry<Integer, String>> entries = map.entrySet();

    // iteration over map way 2 using forEach loop
    for (Map.Entry<Integer, String> entry : entries) {
        System.out.println(entry.getKey() +" : "+ entry.getValue());
    }

    // setValue ()
    for (Map.Entry<Integer, String> entry : entries) {
        entry.setValue(entry.getValue().toUpperCase());
    }

    // remove () with key
    String remove = map.remove(12);

    // getOrDefault ()
    map.getOrDefault(12,"value nhi hai");

    // putIfAbsent ()
    map.putIfAbsent(12,"salam");}
```

(3) HASHCODE & EQUALS METHOD?

```
(4) package org.example.MAP_2.HASHMAP;

    import java.util.HashMap;
    import java.util.Objects;

    class Student {
        String name;
        int id;

        public Student(String name, int id) {
            this.name = name;
            this.id = id;
        }

        public String getName() {
            return name;
        }

        public int getId() {
            return id;
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj) {
                return true;
            }
            if (obj == null || getClass() != obj.getClass()) {
                return false;
            }
            Student student = (Student) obj;
            return id == student.id && Objects.equals(name,
    student.name);
        }
        @Override
        public int hashCode() {
            return Objects.hash(name, id);
        }

        @Override
        public String toString() {
            return "id : "+id +", "+ "name "+" : "+name;
        }
    }
    public class HashCodeAndEquals_02 {
        public static void main(String[] args) {
            HashMap<Student ,String> map = new HashMap<>();

            Student s1 = new Student("sahil",1);
            Student s2 = new Student("rahul",1);
            Student s3 = new Student("sahil",1);

            // if hash code and equals method will not implemented in
```

```
the student
        // class then because of new keyword 3 index will be there.
        map.put(s1,"Engineer"); // hashCode1 --> index1
        map.put(s2,"Designer"); // hashCode2 --> index2
        map.put(s3,"Manager"); // hashCode1 --> index1--> equals
--> replace
    }
}
```