2. Develop a Java program for adding elements [Apple, Banana, Orange] into HashSet, TreeSet and LinkedHashSet to perform the following operations directed as follows

HashSet Demonstration : HashSet after adding elements: [Apple, Orange, Banana], Size of HashSet: 3, Is HashSet empty? false, Contains 'Apple'? true, Contains 'Grape'? False, After removing 'Banana': [Apple, Orange]
TreeSet Demonstration : TreeSet (automatically sorted): [Elephant, Lion, Tiger, Zebra]
LinkedHashSet Demonstration: LinkedHashSet (maintains insertion order): [Red, Green, Blue]
Set Operations Demonstration: Union of sets: [1, 2, 3, 4, 5, 6, 7, 8], Intersection of sets: [4, 5]
Difference of sets (set1 - set2): [1, 2, 3]
Iteration Demonstration : Using for-each loop,Using Iterator

Answer:

```
package Collections;


import java.util.*;

public class MapInterfaceDemo {
    public static void main(String[] args) {
        // HashMap Demonstration
        System.out.println("=== HashMap Demonstration ===");
        Map<String, Integer> hashMap = new HashMap<>();

        // 1. Basic Operations
        // Adding elements
        hashMap.put("Apple", 1);
        hashMap.put("Banana", 2);
        hashMap.put("Orange", 3);
        System.out.println("Initial HashMap: " + hashMap);

        // Updating value
        hashMap.put("Apple", 5); // Updates existing value
        System.out.println("After updating Apple's value: " + hashMap);

        // putIfAbsent
        hashMap.putIfAbsent("Apple", 10); // Won't update as key exists
        hashMap.putIfAbsent("Grape", 4);  // Will add new entry
        System.out.println("After putIfAbsent operations: " + hashMap);

        // 2. Accessing Elements
        System.out.println("\nAccessing Elements:");
        System.out.println("Value for Apple: " + hashMap.get("Apple"));
        System.out.println("Value for missing key: " + hashMap.get("Mango"));
        System.out.println("Value for missing key with default: " + hashMap.getOrDefault("Mango", 0));
```

```
        // 3. Removing Elements
        hashMap.remove("Banana");
        System.out.println("After removing Banana: " + hashMap);

        // Conditional remove
        hashMap.remove("Apple", 5); // Removes only if value matches
        System.out.println("After conditional remove: " + hashMap);

        // 4. TreeMap Demonstration (Sorted Map)
        System.out.println("\n=== TreeMap Demonstration ===");


TreeMap<String, Integer> scores = new TreeMap<>();

        // 5 Adding elements (put operation)
        scores.put("Alice", 95);
        scores.put("Bob", 82);
        scores.put("Charlie", 90);
        scores.put("David", 78);
        scores.put("Eva", 88);

        // Display the TreeMap (naturally sorted by keys)
        System.out.println("TreeMap contents: " + scores);

        // 6 Accessing elements (get operation)
        System.out.println("Charlie's score: " + scores.get("Charlie"));

        // 7. removing elements
        scores.remove("David");
        System.out.println("After removing David: " + scores);

        // 8 Navigation operations (TreeMap-specific)
        // First (lowest) and last (highest) entries
        System.out.println("First entry: " + scores.firstEntry());
        System.out.println("Last entry: " + scores.lastEntry());

// 8  naturally ordered by keys

        Map<String, Integer> treeMap = new TreeMap<>();
        treeMap.put("Zebra", 1);
        treeMap.put("Lion", 2);
        treeMap.put("Elephant", 3);
        System.out.println("TreeMap (naturally ordered by keys): " + treeMap);

 // 9.   eldest entry if size exceeds 3

        System.out.println("\n=== LinkedHashMap Demonstration ===");
        Map<String, Integer> linkedHashMap = new LinkedHashMap<>() {
            @Override
```

```
    protected boolean removeEldestEntry(Map.Entry<String, Integer> eldest) {
        return size() > 3; // Remove eldest entry if size exceeds 3
    }
};

linkedHashMap.put("A", 1);
linkedHashMap.put("B", 2);
linkedHashMap.put("C", 3);
System.out.println("Initial LinkedHashMap: " + linkedHashMap);
linkedHashMap.put("D", 4); // Will remove eldest entry
System.out.println("After adding D (notice removal of eldest): " + linkedHashMap);

// 10. Iterating Over Map
System.out.println("\n=== Map Iteration ===");

// Iterating over entries
System.out.println("Iterating over entries:");
for (Map.Entry<String, Integer> entry : hashMap.entrySet()) {
    System.out.println(entry.getKey() + " -> " + entry.getValue());
}

// Iterating over keys
System.out.println("\nIterating over keys:");
for (String key : hashMap.keySet()) {
    System.out.println("Key: " + key);
}

// Iterating over values
System.out.println("\nIterating over values:");
for (Integer value : hashMap.values()) {
    System.out.println("Value: " + value);
}

// 11. Map Operations
System.out.println("\n=== Map Operations ===");
Map<String, Integer> map1 = new HashMap<>();
map1.put("A", 1);
map1.put("B", 2);

// Merging maps
Map<String, Integer> map2 = new HashMap<>();
map2.put("B", 3);
map2.put("C", 4);

// Merge with custom remapping function
map2.forEach((key, value) ->
    map1.merge(key, value, (v1, v2) -> v1 + v2));
System.out.println("After merging maps: " + map1);
```

```
        // 12. Compute Operations
        System.out.println("\n=== Compute Operations ===");
        map1.compute("A", (k, v) -> (v == null) ? 1 : v * 2);
        System.out.println("After computing A: " + map1);

        map1.computeIfPresent("B", (k, v) -> v * 3);
        System.out.println("After computeIfPresent B: " + map1);

        map1.computeIfAbsent("D", k -> 10);
        System.out.println("After computeIfAbsent D: " + map1);

        // 13. Bulk Operations
        System.out.println("\n=== Bulk Operations ===");
        Map<String, Integer> newMap = new HashMap<>();
        newMap.putAll(map1);
        System.out.println("After putAll: " + newMap);

        // Clear the map
        newMap.clear();
        System.out.println("After clearing: " + newMap);

        // 14. Checking Operations
        System.out.println("\n=== Checking Operations ===");
        System.out.println("Is map empty? " + newMap.isEmpty());
        System.out.println("Map size: " + map1.size());
        System.out.println("Contains key 'A'? " + map1.containsKey("A"));
        System.out.println("Contains value 1? " + map1.containsValue(1));
    }
}
```

o/p

```
=== HashMap Demonstration ===
Initial HashMap: {Apple=1, Orange=3, Banana=2}
After updating Apple's value: {Apple=5, Orange=3, Banana=2}
After putIfAbsent operations: {Apple=5, Grape=4, Orange=3, Banana=2}

Accessing Elements:
Value for Apple: 5
Value for missing key: null
Value for missing key with default: 0
After removing Banana: {Apple=5, Grape=4, Orange=3}
After conditional remove: {Grape=4, Orange=3}

=== TreeMap Demonstration ===
TreeMap contents: {Alice=95, Bob=82, Charlie=90, David=78, Eva=88}
Charlie's score: 90
After removing David: {Alice=95, Bob=82, Charlie=90, Eva=88}
First entry: Alice=95
Last entry: Eva=88
```

TreeMap (naturally ordered by keys): {Elephant=3, Lion=2, Zebra=1}

=== LinkedHashMap Demonstration ===
Initial LinkedHashMap: {A=1, B=2, C=3}
After adding D (notice removal of eldest): {B=2, C=3, D=4}

=== Map Iteration ===
Iterating over entries:
Grape -> 4
Orange -> 3

Iterating over keys:
Key: Grape
Key: Orange

Iterating over values:
Value: 4
Value: 3

=== Map Operations ===
After merging maps: {A=1, B=5, C=4}

=== Compute Operations ===
After computing A: {A=2, B=5, C=4}
After computeIfPresent B: {A=2, B=15, C=4}
After computeIfAbsent D: {A=2, B=15, C=4, D=10}

=== Bulk Operations ===
After putAll: {A=2, B=15, C=4, D=10}
After clearing: {}

=== Checking Operations ===
Is map empty? true
Map size: 4
Contains key 'A'? true
Contains value 1? false

Exercises

Execute the following exercise programs in lab session and copy it down in the observation after successful execution.

Hashmap

1. Write a Java program to traverse / iterate all the keys with the specified value in a HashMap
(1,”Apple”), (2,”Strawberry”), (3,”Pear”), (4,”Cucumber”), (5,”Grapes”)
[hint : 6. Iterating Over Map in the worked out example]

2. Remove the following all elements from the HashMap then verify that its size is zero and include once again following elements and check its size
(1,”Banana”), (2,”Orange”), (3,”Guava”), (4,”Pomegranate ”), (5,”Amla”)
[Hint : clear() , size() and put()]

3.Write a Java program to copy all mappings from the specified map to another map as shown below
Hashmap1 => (1, "Red") ,(2, "Green"), (3, "Black")
Hashmap2 => (4, "White"),(5, "Blue"),(6, "Orange")
Add all the values of Hashmap1 intoLiberation Serif Hashmap2 and print the result

4.Write a Java program to check whether a map contains Key-Values mappings (empty) or not after adding all the following elements into Hashmap and after removing all the elements from the Hashmap.
(1, "Red"),(2, "Green"), (3, "Black"), (4, "White"),(5, "Blue")

TreeMap

1. Write a Java program to traverse / iterate all the keys with the specified value in a TreeMap
(1,”Apple”), (2,”Strawberry”), (3,”Pear”), (4,”Cucumber”), (5,”Grapes”)
[hint : put() ; for (Map.Entry<Integer,String> entry : tree_map.entrySet()); getKey() and getValue()]

2. Write a Java program to search for keys C4 and C5 are present or not in the following Tree Map.
("C1", "Red"); ("C2", "Green"); ("C3", "Black");("C4", "White")

3. Remove the following all elements from the TreeMap then verify that its size is zero and include once again following elements and iterate them and check its size
(1,”Banana”), (2,”Orange”), (3,”Guava”), (4,”Pomegranate ”), (5,”Amla”)
[Hint : clear() , size() and put()]

4. Write a Java program to get the first (lowest) key and the last (highest) key currently in the following Treemap.
("C2", "Red"),("C1", "Green"), ("C4", "Black"), ("C3", "White")
[Hint : firstKey() and lastKey()]

LinkedHashMap

1. Write a Java program to traverse / iterate all the keys with the specified value in a linked HashMap (1,”Apple”), (2,”Strawberry”), (3,”Pear”), (4,”Cucumber”), (5,”Grapes”) and display only elderly entered 4 elements only
[hint : Refer to 5. LinkedHashMap Demonstration of worked out example]

2. Include the following elements into the LinkedHashMap called programminglanguages and print all of them.
("Java", 1995);("Python", 1991);(“JavaScript", 1995);("C++", 1985);

3. Include the following elements into the LinkedHashMap called programminglanguages and print as follows.

Java was developed in 1995
Python was developed in 1991.
JavaScript was developed in 1995
C++ was developed in 1985

4. Add elements in the same order in the LinkedHaspMap using for() loop and put()

```
String[] keys = {"C", "A", "B", "E", "D"};
Integer[] values = {3, 1, 2, 5, 4};
```