WEEK-11

ROLL NO:230701236

1) **Java HashSet** class implements the Set interface, backed by a hash table which is actually a HashMap instance.

No guarantee is made as to the iteration order of the hash sets which means that the class does not guarantee the constant order of elements over time.

This class permits the null element.

The class also offers constant time performance for the basic operations like add, remove, contains, and size assuming the hash function disperses the elements properly among the buckets.

Java HashSet Features

A few important features of HashSet are mentioned below:

- Implements Set Interface.
- The underlying data structure for HashSet is Hashtable.
- As it implements the Set Interface, duplicate values are not allowed.
- Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code.
- NULL elements are allowed in HashSet.
- HashSet also implements **Serializable** and **Cloneable** interfaces.
- public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable Sample
  Input and Output:
  5
  90
  56 45
  78 25
  78
  Sample Output:
  78 was found in the set.
  Sample Input and output:
  3 2
  7
  9
  5

Sample Input and output:
5 was not found in the set.

CODE:

```
import java.util.HashSet; import
java.util.Scanner;


public class HashSetExample {      public
static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
HashSet<Integer> hashSet = new HashSet<>();
int n = scanner.nextInt();        for (int i = 0; i < n; i++) {
int element = scanner.nextInt();
hashSet.add(element);
      }
      int searchElement = scanner.nextInt();
if (hashSet.contains(searchElement)) {
         System.out.println(searchElement + " was found in the set.");
      } else {
         System.out.println(searchElement + " was not found in the set.");
      }
      scanner.close();
   }
}
```

OUTPUT:

	Test	Input	Expected	Got	
✓	1	5 90 56 45 78 25 78	78 was found in the set.	78 was found in the set.	✓
✓	2	3 -1 2 4 5	5 was not found in the set.	5 was not found in the set.	✓

Passed all tests! ✓

2) Write a Java program to compare two sets and retain elements that are the same.

Sample Input and Output:

5

Football

Hockey

Cricket

Volleyball

Basketball

7 // **HashSet 2:**

Golf

Cricket

Badminton

Football

Hockey

Volleyball

Handball

SAMPLE OUTPUT:

Football

Hockey

Cricket

Volleyball

Basketball

CODE:

```
import java.util.HashSet; import
java.util.Scanner;

public class SetComparison {     public static void
main(String[] args) {        Scanner scanner = new
Scanner(System.in);        int n1 = scanner.nextInt();
scanner.nextLine();
      HashSet<String> set1 = new HashSet<>();
      for (int i = 0; i < n1; i++) {
         String element = scanner.nextLine();
set1.add(element);
      }
      int n2 = scanner.nextInt();        scanner.nextLine(); //
Consume the newline character

      HashSet<String> set2 = new HashSet<>();
      for (int i = 0; i < n2; i++) {
         String element = scanner.nextLine();
set2.add(element);
```

```
        }
        set1.retainAll(set2);
for (String element : set1) {
            System.out.println(element);
        }
        scanner.close();
    }
}
```

OUTPUT:

	Test	Input	Expected	Got	
✓	1	5 Football Hockey Cricket Volleyball Basketball 7 Golf Cricket Badminton Football Hockey Volleyball Throwball	Cricket Hockey Volleyball Football	Cricket Hockey Volleyball Football	✓
✓	2	4 Toy Bus Car Auto 3 Car Bus Lorry	Bus Car	Bus Car	✓

Passed all tests! ✓

3) Java HashMap Methods

containsKey() Indicate if an entry with the specified key exists in the map containsValue() Indicate if an entry with the specified value exists in the map putIfAbsent() Write an entry into the map but only if an entry with the same key does not already exist remove() Remove an entry from the map replace() Write to an entry in the map only if it exists size() Return the number of entries in the map

Your task is to fill the incomplete code to get desired output

CODE:

```
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Set; import
java.util.Scanner;

class prog {    public static void
main(String[] args) {
      // Creating HashMap with default initial capacity and load factor
      HashMap<String, Integer> map = new HashMap<String, Integer>();

      String name;
int num;
      Scanner sc = new Scanner(System.in);
int n = sc.nextInt();       for (int i = 0; i < n;
i++) {          name = sc.next();          num =
sc.nextInt();          map.put(name, num);
      }

      // Printing key-value pairs
```

```
        Set<Entry<String, Integer>> entrySet = map.entrySet();
for (Entry<String, Integer> entry : entrySet) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
        System.out.println("----------");

        // Creating another HashMap
        HashMap<String, Integer> anotherMap = new HashMap<String, Integer>();

        // Inserting key-value pairs to anotherMap using put() method
anotherMap.put("SIX", 6);        anotherMap.put("SEVEN", 7);

        // Inserting key-value pairs of map to anotherMap using putAll() method
anotherMap.putAll(map);  // Filling in the missing code here        entrySet =
anotherMap.entrySet();        for (Entry<String, Integer> entry : entrySet) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
        map.putIfAbsent("FIVE", 5);

        // Retrieving a value associated with key 'TWO'
        Integer value = map.get("TWO"); // Using Integer instead of int to handle null case
System.out.println( (value != null ? value : "Key not found"));

        // Checking whether key 'ONE' exists in map
        System.out.println( map.containsKey("ONE"));

        // Checking whether value '3' exists in map
        System.out.println( map.containsValue(3));
```

```
        // Retrieving the number of key-value pairs present in map
        System.out.println( map.size());


        // Close the scanner
sc.close();
    }
}
```

OUTPUT:

	Test	Input	Expected	Got	
✓	1	3 ONE 1 TWO 2 THREE 3	ONE : 1 TWO : 2 THREE : 3 ---------- SIX : 6 ONE : 1 TWO : 2 SEVEN : 7 THREE : 3 2 true true 4	ONE : 1 TWO : 2 THREE : 3 ---------- SIX : 6 ONE : 1 TWO : 2 SEVEN : 7 THREE : 3 2 true true 4	✓

Passed all tests! ✓