

Assignment-15

Part -A

Que-1) Explain fail-fast and fail-safe iterators.

fail-fast iterator throws a `ConcurrentModificationException` if the collection is modified during iteration, as it works directly on the original collection and is not thread-safe.

fail-safe iterator does not throw any exception because it iterates over a copy of the collection, making it thread-safe but slightly slower.

Que-2) What is the `Iterator` interface?

The Java `Iterator` interface allows us to access and move through elements of a collection like `List`, `Set`, or `Map` easily. It is used to retrieve elements one by one and perform operations on them. `Iterator` is called a universal iterator because it works with all `Collection` objects. Using `Iterator`, we can traverse elements only in the forward direction. It supports both reading and removing elements safely. The `Iterator` interface was introduced in JDK 1.2 and is available in the `java.util` package. Before `Iterator`, `Enumeration` was used, which was introduced in JDK 1.0.

Que-3) What is the `ListIterator` interface?

The `ListIterator` interface in Java is used to iterate elements of a `List` (like `ArrayList`, `LinkedList`). It extends the `Iterator` interface and provides more features. Using `ListIterator`, we can traverse the list in both forward and backward directions. It allows reading, removing, adding, and updating elements while iterating. Important methods include `hasNext()`, `next()`, `hasPrevious()`, `previous()`, `add()`, `remove()`, and `set()`. The `ListIterator` interface is present in the `java.util` package and was introduced in JDK 1.2.

Que-4) What are `Comparable` and `Comparator` interfaces? `Comparable` is an interface which is present in `java.lang` package and contains a `compareTo` method. The return type of `compareTo` is `int`. It is used to define default sorting order.

`Comparator` is an interface which is present in `java.util` package and contains a `compare` method. The return type of `compare` is `int`. It is used to customize sorting order.

Que-5) Difference between `Comparable` and `Comparator`.

Comparable - `Comparable` is used to define the natural ordering of objects. The sorting logic is written inside the same class by implementing the `compareTo()` method. It supports only one sorting order and is useful when a single, default sorting sequence is needed.

Comparator - `Comparator` is used to define custom or external sorting logic. The comparison logic is written in a separate class using the `compare()` method. It supports multiple sorting orders and is helpful when different ways of sorting are required without modifying the original class.

Part-B

Que-1) Write a program to count occurrences of words using HashMap.

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

public class OccuranceOfWord {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        HashMap<String, Integer> hm = new HashMap<String, Integer>();
        System.out.println("Enter a String : ");
        String str = sc.nextLine();
        String[] arr = str.split(" ");
        for (String x : arr) {
            if (hm.containsKey(x))
                hm.put(x, hm.get(x) + 1);
            else
                hm.put(x, 1);
        }

        for (Entry<String, Integer> ent : hm.entrySet()) {
            System.out.println(ent);
        }

    }
}

Output - Enter a
String : aa bb
cc aa bb aa aa=3
bb=2 cc=1
```

Que-2) Write a program to find duplicate characters using HashMap.

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

public class DuplicateChar {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        HashMap<Character, Integer> hm = new HashMap<>();
        System.out.println("Enter a String : ");
        String str = sc.nextLine();
        char[] ch = str.toCharArray();
        for (char x : ch) {
            if
(hm.containsKey(x))
                hm.put(x, hm.get(x) + 1);
            else
                hm.put(x, 1);
        }
    }
}
```

```

        for (Entry<Character, Integer> ent : hm.entrySet()) {
            if(ent.getValue() > 1)
                System.out.println(ent);
        }
    }
}

```

Que-3) Write a program to sort a HashMap by keys.

```

import java.util.Collections;
import java.util.HashMap;
import java.util.List; import
java.util.Map.Entry; import
java.util.Scanner; import
java.util.TreeMap;

public class SortKey {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        HashMap<Integer, String> hm = new HashMap<>();
        hm.put(1, "rutuja");
        hm.put(3, "nikita");
        hm.put(2, "sanika");
        hm.put(4, "jiya");
        hm.put(5, "jagruti");

        TreeMap<Integer, String> tm = new TreeMap<>(hm);

        for (Entry<Integer, String> ent : hm.entrySet()) {

            System.out.println(ent);
        }
    }
}

```

Output -
1=rutuja
2=sanika
3=nikita
4=jiya
5=jagruti

Que-4) Write a program to sort a HashMap by values. import java.util.*;

```

public class SortHashMapByValue {      public
static void main(String[] args) {

```

```

        HashMap<Integer, String> map = new HashMap<>();
map.put(1, "rutuja");          map.put(3, "sanika");
map.put(2, "Rani");           map.put(4, "Mayuri");
map.put(5, "Anisha");

        List<Map.Entry<Integer, String>> list =
new ArrayList<>(map.entrySet());
Collections.sort(list, (e1, e2) ->
        e1.getValue().compareTo(e2.getValue()));

        LinkedHashMap<Integer, String> sortedMap = new LinkedHashMap<>();
for (Map.Entry<Integer, String> entry : list) {
sortedMap.put(entry.getKey(), entry.getValue());
}
        for (Map.Entry<Integer, String> entry : sortedMap.entrySet()) {
                System.out.println(entry.getKey() + " = " +
entry.getValue());
        }
}

```

Output -

```

5 = Anisha
4 = Mayuri
3 = Rani
2 = rutuja
1 = sanika

```

Que-5) Write a program to merge two HashMaps. import
java.util.*;

```

public class SortHashMapByValue {
    public static void main(String[] args) {

        HashMap<Integer, String> map1 = new HashMap<>();
map1.put(1, "rutuja");          map1.put(2, "Ovi");
map1.put(3, "Rani");           map1.put(4, "Mayuri");
map1.put(5, "Anisha");

        HashMap<Integer, String> map2 = new HashMap<>();
map2.put(6, "rutuja");          map2.put(7, "Ovi");
map2.put(8, "Rani");           map2.put(9, "Mayuri");
map2.put(10, "Anisha");

        HashMap<Integer, String> map3 = new HashMap<>(map1);

        map3.putAll(map2);

        for (Map.Entry<Integer, String> entry : map3.entrySet()) {
                System.out.println(entry.getKey() + " = " +
entry.getValue());
        }
}

```

```
}
```

OUTPUT -

```
1  = rutuja
2  = Ovi
3  = Rani
4  = Mayuri
5  = Anisha
6  = rutuja
7  = Ovi
8  = Rani
9  = Mayuri
10 = Anisha
```