Object Oriented Programming (OOP) – IT2030

Interactively Perform Collection Framework Lecture in class room

Objective

Students should understand the behavior and uniqueness of each collection type (List, Set, Map, Queue, Stack and their related sub classes.

Introduction

This document contains set of **exercises you can share this document with students** to perform Lecture session interactively. You can **group student's row wise** or **according to their preference** and ask them to sit accordingly. Volunteer student from each group can write the outputs of given exercise on the board.

You can switch with Lecture slide for **theory sessions** and **exercises in this document** as per the instruction below. This document is aligned with the slides number in your provided slides. A separate **project with source code** will be shared **only for Lecturers** that would be instrumental to compile and show the output pertaining to each exercise.

There are some exercises in the Lecture slides as well. Please follow them as it is and this document would give more additional exercises.

- As usual, start the Lecture as per the slides and continue up to slide no 03 and **explain the Hierarchy of collection framework.** Then proceed up to **slide 06 explain List Interface and Queue interface** and then continue below **exercises related to the List, Stack and Queue Interfaces**.

List Interface

1. Write the output of following program. You are going to **ArrayList without having generic types** to store any object.

```
public static void main(String[] args) {

    listTotal(addElements());

}
```

Now write following methods that add elements to **ArrayList** and using the **listToTotal() method** and write the outputs of them. Explain the reason for the output and remodify the **program using generics**. (**Hint:- ArrayList<Integer>())**

```java
public static List addElements() {

    List list = new ArrayList();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add("abc");
    return list;
}

/**
 * Calculate Total
 * @param list
 */
public static void listTotal(List list) {

    int total = 0;
    for (Object object : list) {

        total += (Integer) object;
    }
    System.out.println("Total is = " + total);
}
```

Stack Interface

Write the output of following program. Once you perform pop() and add() functions how would be the output? Remodify adding “11111111” and “222222” for the stack and write output as well.

```java
Stack<String> stack = new Stack<String>();
stack.push("aaaaaaa");
stack.push("bbbbbbb");
stack.push("ccccccc");
stack.push("ddddddd");
stack.push("eeeeeee");
stack.add("ffffffff");

System.out.println(stack);
//Remove last Item
System.out.println("Stack pop = " + stack.pop() + "\n");

System.out.println(stack);
//Add elements
System.out.println("Stack add = " + stack.add("1111111") + "\n");
System.out.println("Stack add = " + stack.add("2222222") + "\n");

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

Queue Interface

Write the output of the following queue. Remodify the program by adding 99 add(99) and rewrite the output

```
public static void showPriorityQueue(){

    PriorityQueue<Integer> priorityQueue = new PriorityQueue<Integer>();
    priorityQueue.add(11);
    priorityQueue.add(22);
    priorityQueue.add(22);
    priorityQueue.add(44);
    priorityQueue.add(33);
    priorityQueue.add(55);
    priorityQueue.add(66);
    priorityQueue.add(77);
    priorityQueue.add(88);

    System.out.println("=======Before=======================");
    System.out.println(priorityQueue);

    //Show first element
    System.out.println("Queue Peek = " + priorityQueue.peek());
    //Poll delete the first element
    System.out.println("Queue Poll = " + priorityQueue.poll());
    //Poll delete the second element
    System.out.println("Queue Poll = " + priorityQueue.poll());

    System.out.println("=======After=======================");
    System.out.println(priorityQueue);
}
```

- Explain the slide 07 in the Lecture and continue following exercises related to the **Set Interface**.

Set Interface

1. Write the output for the following program and specify the reason?

2. Then modify the same program (**instead of HashSet() use LinkedHashSet())** and rewrite the program for the new modification. Explain the reason for the output difference.

```
public static void showHashSet() {

    Set<Integer> set = new HashSet<Integer>();
    set.add(10);
    set.add(20);
    set.add(50);
    set.add(50);
    set.add(20);
    set.add(10);

    for (Integer value : set) {
        System.out.println(value);
    }
}
```

Output

```
50
20
10
```

3. Write the output for the following program and specify the reason

```
public static void showTreeSet() {

    SortedSet<Integer> set = new TreeSet<Integer>();
    set.add(10);
    set.add(20);
    set.add(50);
    set.add(50);
    set.add(20);
    set.add(10);
    set.add(30);
    set.add(40);
    set.add(60);
    set.add(10);

    for (Integer value : set) {
        System.out.println(value);
    }
}
```

4. Create a class called **Person** and override the default constructor to set **Age** as follows.

```
public class Person {

    private int age;

    public Person(int age) {
        this.age = age;
```

```
    }

    public int getAge() {
          return age;
    }
}
```

5. Then use Person with **HashSet** as follows and add Person objects and check the output? **Mention the reason that display duplicates?**

```
public static void showPersons() {

          Set<Person> hashSet = new HashSet<Person>();
          hashSet.add(new Person(10));
          hashSet.add(new Person(10));
          hashSet.add(new Person(20));
          hashSet.add(new Person(30));
          hashSet.add(new Person(30));
          hashSet.add(new Person(20));

          for (Person person : hashSet) {
                System.out.println(person.getAge());
          }

    }
```

6. **Modify the Person class** to display the Output uniquely as follows
   20
   10
   30

➢ Now Continue **Lecture slide from 08 to 15** and explain the **Working with Map slide.**

Map Interface

➢ Implement the following programs and write the outputs **for HashMap(), displayLinkedHashMap(), displayTreeMap(), and showHashTable() examples**. Once student wrote the answer compile and show the output of provided source code.

1. Exercise for **HashMap** write the output of below method

```java
public static Map<Integer, String> displayHashMap(){

    Map<Integer, String> hashMap = new HashMap<Integer, String>();
    hashMap.put(10, "abc");
    hashMap.put(20, null);
    hashMap.put(30, null);
    hashMap.put(40, "abc");
    hashMap.put(40, "cde");
    hashMap.put(50, "efg");

    for (Integer key : hashMap.keySet()) {
        System.out.println(key + ", " + hashMap.get(key));
    }
    return hashMap;
}
```

2. Exercise **for LinkedHashMap** write the output of below method

```java
public static Map<Integer, String> displayLinkedHashMap(){

    Map<Integer, String> linkedHashMap = new LinkedHashMap<Integer, String>();
    linkedHashMap.put(10, "abc");
    linkedHashMap.put(20, null);
    linkedHashMap.put(30, null);
    linkedHashMap.put(40, "abc");
    linkedHashMap.put(40, "cde");
    linkedHashMap.put(50, "efg");

    for (Integer key : linkedHashMap.keySet()) {
        System.out.println(key + ", " + linkedHashMap.get(key));
    }
    return linkedHashMap;
}
```

3. Exercise **for the TreeMap** write the output of below method

```java
public static void displayTreeMap(){

    SortedMap<Integer, String> treeMap = new TreeMap<Integer, String>();
    treeMap.put(10, "abc");
    treeMap.put(20, null);
    treeMap.put(30, null);
    treeMap.put(40, "abc");
    treeMap.put(40, "cde");
    treeMap.put(50, "efg");
    treeMap.put(50, "fgh");

    for (Map.Entry<Integer, String> mapEntry : treeMap.entrySet()) {
        System.out.println(mapEntry.getKey() + ", " + mapEntry.getValue());
    }
}
```

4. Exercise **for the HashTable** write the output of below method

5. What is the advantage of having **HashTable** compared to the other maps in **Map Interface**?

```java
public static void showHashTable(){

    Map<String, String> hashTable = new Hashtable<String, String>();
    hashTable.put("key1", "value1");
    hashTable.put("key1", "value2");
    hashTable.put("key2", "value3");
    hashTable.put("key2", "value3");
    hashTable.put("key4", "");
    hashTable.put("key5",  "");
    //hashTable.put(null, null); either key or value can not be null

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

    System.out.println("=========Map.keySet()======================");
    for (String key : hashTable.keySet()) {
        System.out.println(key + ", " + hashTable.get(key));
    }

    System.out.println("=========Map.values========================");
    for (String value : hashTable.values()) {
        System.out.println(value);
    }
}
```

Collection Framework

1. Sort the following ArrayList() using Collections class

```
ArrayList<String> arrayList = new ArrayList<String>();
arrayList.add("ZZZZZ");
arrayList.add("YYYYY");
arrayList.add("BBBBB");
arrayList.add("XXXXX");
arrayList.add("QQQQQ");
arrayList.add("PPPPP");
```

Output should be as follows after sorting

```
<terminated> Fifth [Java Application] C:\
/////////After Sort///////////
BBBBB
PPPPP
QQQQQ
XXXXX
YYYYY
ZZZZZ
```

2. Convert following String Array into ArrayList through Collection Framework. The Sort it into Ascending order and Descending order. [Hint :- Use the **sort()** and **reverse()** methods in the Collection Framework]

   String [] array = new String[]{"Sam", "Udara", "Ann", "Mike", "Upul"};

   Display the below output

```
[Sam, Udara, Ann, Mike, Upul]
[Ann, Mike, Sam, Udara, Upul]
[Upul, Udara, Sam, Mike, Ann]
```

You can consider the Lecture & Tutorial both as a 3 hour Tutorial session and conduct the session more interactive way.

THE END