Collection Framework and Map Assignment

Q1. What is the collection framework in Java?

Ans. Any group of individual objects that are represented as a single unit is known as a Java Collection of Objects. In Java, a separate framework named the "*Collection Framework*" has been defined in JDK 1.2 which holds all the Java Collection Classes and Interface in it.
In Java, the Collection interface (java.util.Collection) and Map interface (java.util.Map) are the two main "root" interfaces of Java collection classes.

Q2. What is the difference between ArrayList and Linkedlist?

Ans.

	ArrayList	LinkedList
1.	This class uses a dynamic array to store the elements in it. With the introduction of generics, this class supports the storage of all types of objects.	This class uses a doubly linked list to store the elements in it. Similar to the ArrayList, this class also supports the storage of all types of objects.
2.	Manipulating ArrayList takes more time due to the internal implementation. Whenever we remove an element, internally, the array	Manipulating LinkedList takes less time compared to ArrayList because, in a doubly-linked list, there is no concept of shifting the memory bits. The list is

	is traversed and the memory bits are shifted.	traversed and the reference link is changed.
3.	Inefficient memory utilization.	Good memory utilization.
4.	It can be one, two or multi-dimensional.	It can either be single, double or circular LinkedList.
5.	Insertion operation is slow.	Insertion operation is fast.
6.	This class implements a List interface. Therefore, this acts as a list.	This class implements both the List interface and the Deque interface. Therefore, it can act as a list and a deque.
7.	This class works better when the application demands storing the data and accessing it.	This class works better when the application demands manipulation of the stored data.

8.	Data access and storage is very efficient as it stores the elements according to the indexes.	Data access and storage is slow in LinkedList.
9.	Deletion operation is not very efficient.	Deletion operation is very efficient.
10.	It is used to store only similar types of data.	It is used to store any types of data.
11.	Less memory is used.	More memory is used.
12.	This is known as static memory allocation.	This is known as dynamic memory allocation.

Example-:
ArrayList-:

```
import java.io.*;
import java.util.*;

class Bcd{

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

```
    {        ArrayList<Integer> arrli
            = new ArrayList<Integer>();

                 for (int i = 1; i <= 5; i++)
            arrli.add(i);

                  System.out.println(arrli);

                  arrli.remove(3);

                  System.out.println(arrli);
    }
}
```

LinkedList-:

```
import java.util.*;
class A {

        public static void main(String args[])
    {
         LinkedList<String> object
            = new LinkedList<String>();
         object.add("A");
        object.add("B");
        object.addLast("C");

                  System.out.println(object);
         object.remove("B");
        object.removeFirst();

          System.out.println("Linked list after "
                           + "deletion: " + object);
    }
}
```

Q3. What is the difference between iterator and listiterator?

Ans

Iterator	ListIterator
Can traverse elements present in Collection only in the forward direction.	Can traverse elements present in Collection both in forward and backward directions.
Helps to traverse Map, List and Set.	Can only traverse List and not the other two.
Indexes cannot be obtained by using Iterator.	It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing List.
Cannot modify or replace elements present in Collection	We can modify or replace elements with the help of set(E e)

Cannot add elements and it throws ConcurrentModificationException.	Can easily add elements to a collection at any time.
Certain methods of Iterator are next(), remove() and hasNext().	Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e).

Example -: Iterator

```
import java.io.*;
import java.util.*;

class IteratorDemo1 {
    public static void main(String[] args)
    {
        ArrayList<Integer> list
            = new ArrayList<Integer>();

          list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);

        Iterator itr = list.iterator();

          System.out.println("Iterator:");
        System.out.println("Forward traversal: ");
```

```
          while (itr.hasNext())
            System.out.print(itr.next() + " ");

          System.out.println();

                  ListIterator i = list.listIterator();

          System.out.println("ListIterator:");
        System.out.println("Forward Traversal : ");

          while (i.hasNext())
            System.out.print(i.next() + " ");

          System.out.println();

          System.out.println("Backward Traversal : ");

          while (i.hasPrevious())
            System.out.print(i.previous() + " ");

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

ListIterator-:

```
import java.util.ArrayList;
import java.util.ListIterator;

public class ListIteratorDemo2 {
    public static void main(String[] args)
    {

        ArrayList<Integer> aList
            = new ArrayList<Integer>();
        aList.add(1);
        aList.add(2);
        aList.add(3);
        aList.add(4);
        aList.add(5);
```

```
         System.out.println("Elements of ArrayList: ");
        for (Integer i : aList) {
            System.out.println(i);
        }
        ListIterator<Integer> l
            = aList.listIterator();
        l.next();
        l.set(80000);

         System.out.println("\nNow the ArrayList"
                          + " elements are: ");
        for (Integer i : aList) {
            System.out.println(i);
        }
    }
}
```

Q4. What is the difference between iterator and Enumeration?
Ans-:

Iterator	Enumeration
Iterator is a universal cursor as it is applicable for all the collection classes.	Enumeration is not a universal cursor as it applies only to legacy classes.
Iterator has the remove() method.	Enumeration does not have the remove() method.

Iterator can do modifications (e.g using remove() method it removes the element from the Collection during traversal).	Enumeration interface acts as a read only interface, one can not do any modifications to Collection while traversing the elements of the Collection.
Iterator is not a legacy interface. Iterator can be used for the traversal of HashMap, LinkedList, ArrayList, HashSet, TreeMap, TreeSet .	Enumeration is a legacy interface which is used for traversing Vector, Hashtable.

Syntax -: Iterator-:

```
Iterator itr = c.iterator();

Enumeration -:

enum Colour

{

    RED, GREEN, BLUE;

}
```

Q5. What is the difference between List and Set?
Ans.

List	Set
1. The List is an indexed sequence.	1. The Set is an non-indexed sequence.
2. List allows duplicate elements	2. Set doesn’t allow duplicate elements.
3. Elements by their position can be accessed.	3. Position access to elements is not allowed.
4. Multiple null elements can be stored.	4. Null elements can store only once.
5. List implementations are ArrayList, LinkedList, Vector, Stack	5. Set implementations are HashSet, LinkedHashSet.

Example-:

```
import java.io.*;
import java.util.*;
```

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

        List<Integer> l = new ArrayList<>();
        l.add(5);
        l.add(6);
        l.add(3);
        l.add(5);
        l.add(4);

        Set<Integer> s = new HashSet<>();
        s.add(5);
        s.add(6);
        s.add(3);
        s.add(5);
        s.add(4);

        System.out.println("List = " + l);
        System.out.println("Set = " + s);
    }
}
```

Q6. What is the difference between HashSet and TreeSet?
Ans.

Parameters	HashSet	TreeSet
Ordering or Sorting	It does not provide a guarantee to sort the data.	It provides a guarantee to sort the data. The sorting depends on the supplied Comparator.
Null Objects	In HashSet, only an element can be null.	It does not allow null elements.

Parameters	HashSet	TreeSet
Comparison	It uses hashCode() or equals() method for comparison.	It uses compare() or compareTo() method for comparison.
Performance	It is faster than TreeSet.	It is slower in comparison to HashSet.
Implementation	Internally it uses HashMap to store its elements.	Internally it uses TreeMap to store its elements.
Data Structure	HashSet is backed up by a hash table.	TreeSet is backed up by a Red-black Tree.
Values Stored	It allows only heterogeneous value.	It allows only homogeneous value.

Example-: TreeSet

```
import java.util.TreeSet;
public class TreeSetExample
{
public static void main(String[] args)
{
TreeSet<String> ts= new TreeSet<String>();
ts.add("Dell");
ts.add("HP");
ts.add("Apple");
ts.add("Acer");
ts.add("Asus");
ts.add("Lenovo");
ts.add("Samsung");
ts.add("Asus");
System.out.println("TreeSet elements are: ");
for (String temp : ts)
{
System.out.println(temp);
}
}
}
```

Example -: HashSet

```
import java.util.HashSet;
public class HashSetExample
{
public static void main(String[] args)
{
HashSet<String> hs = new HashSet<String>();
hs.add("New York City");
hs.add("Houston");
hs.add("Tucson");
hs.add("Los Angeles");
hs.add("Chicago");
hs.add("Boston");
hs.add("Denver");
hs.add("Chicago");
System.out.println("HashSet elements are: ");
for (String temp : hs)
{
System.out.println(temp);
}
}
21.}
```

Q7. What is the difference between Array and ArrayList?
Ans.

Basis	Array	ArrayList
Definition	An array is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location.	The ArrayList is a class of Java Collections framework. It contains popular classes like Vector, HashTable, and HashMap.
Static/ Dynamic	Array is static in size.	ArrayList is dynamic in size.
Resizable	An array is a fixed-length data structure.	ArrayList is a variable-length data structure. It can be resized itself when needed.
Initializatio n	It is mandatory to provide the size of an array while initializing it directly or indirectly.	We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.
Performan ce	It performs fast in comparison to ArrayList because of fixed size.	ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance.

Primitive/ Generic type	An array can store both objects and primitives type.	We cannot store primitive type in ArrayList. It automatically converts primitive type to object.
Iterating Values	We use for loop or for each loop to iterate over an array.	We use an iterator to iterate over ArrayList.
Type-Safety	We cannot use generics along with array because it is not a convertible type of array.	ArrayList allows us to store only generic/ type, that's why it is type-safe.
Length	Array provides a length variable which denotes the length of an array.	ArrayList provides the size() method to determine the size of ArrayList.
Adding Elements	We can add elements in an array by using the assignment operator.	Java provides the add() method to add elements in the ArrayList.
Single/ Multi-Dimensional	Array can be multi-dimensional.	ArrayList is always single-dimensional

Example -:
Array-:

```
public class ArrayExample
{
public static void main(String args[])
{
int arr[]=new int[4];
arr[0]=12;
arr[1]=2;
arr[2]=15;
arr[3]=67;
for(int i=0;i<arr.length;i++)
{
System.out.println(arr[i]);
}
}
}
```

Example -: ArrayyList -:

```
public class ArrayListExample
{
public static void main(String args[])
{
List<Float> list = new ArrayList<Float>();
list.add(12.4f);
list.add(34.6f);
list.add(56.8f);
list.add(78.9f);
for(Float f:list)
{
System.out.println(f);
}
}
}
```

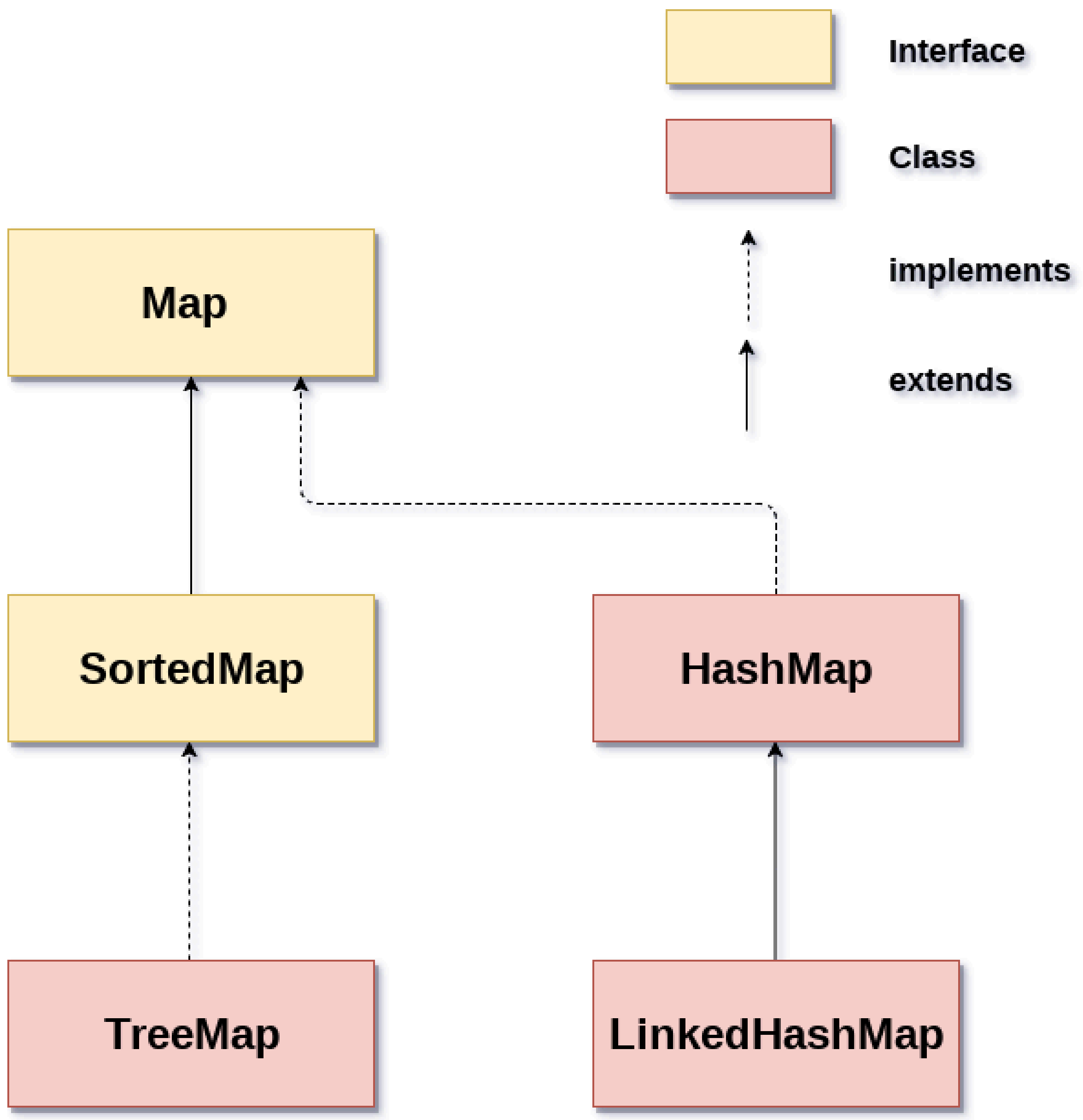
Q8. What is a Map in Java?

Ans. A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:



A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

Q9. What are the commonly used implementation of Map in Java?

Ans. Creating Map Objects

Since Map is an interface, objects cannot be created of the type map. We always need a class that extends this map in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the Map.

Syntax: Defining Type-safe Map

```
Map hm = new HashMap();
```

```
Example-:
import java.util.*;


class B {


        public static void main(String args[])
    {
          Map<String, Integer> hm = new HashMap<String, Integer>();
        hm.put("a", new Integer(100));
        hm.put("b", new Integer(200));
        hm.put("c", new Integer(300));
        hm.put("d", new Integer(400));
        for (Map.Entry<String, Integer> me :
             hm.entrySet()) {
             System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

Q10.What is the difference between HashMap and TreeMap?
Ans.

	HashMap	TreeMap
1.	It does not provide any order for elements.	It provides orders for elements.
2.	It’s speed is fast.	It’s speed is slow.
3.	It allows one key as null and also allows multiple values.	It does not allow keys as null but it allows multiple null values.
4.	It consumes more memory space.	It consumes less memory space.
5.	It has only basic features.	It has advanced features.

6.	For comparing keys, equals() is used.	For comparing keys, compare or compareTo() is used.
7.	It’s complexity is O(1).	Its complexity is O(log n).

Example-:
HashMap-:

```
import java.util.*;

class Main
{
        static void printFreq(int arr[])
    {
         HashMap<Integer, Integer> hmap =new HashMap<Integer,
Integer>();

             for (int i = 0; i < arr.length; i++)
        {
            Integer c = hmap.get(arr[i]);

           if (hmap.get(arr[i]) == null)
               hmap.put(arr[i], 1);

             else
              hmap.put(arr[i], ++c);
        }

              for (Map.Entry m:hmap.entrySet())
          System.out.println("Frequency of " + m.getKey() +
                             " is " + m.getValue());
    }
     public static void main (String[] args)
    {
        int arr[] = {10, 34, 5, 10, 3, 5, 10};
        printFreq(arr);
    }
}
```
TreeMap-:
```
import java.util.*;

class Main
{
    static void printFreq(int arr[])
    {
        TreeMap<Integer, Integer> tmap =
                     new TreeMap<Integer, Integer>();

                for (int i = 0; i < arr.length; i++)
        {
            Integer c = tmap.get(arr[i]);

            if (tmap.get(arr[i]) == null)
               tmap.put(arr[i], 1);

            else
              tmap.put(arr[i], ++c);
        }
         for (Map.Entry m:tmap.entrySet())
          System.out.println("Frequency of " + m.getKey() +
                                " is " + m.getValue());
    }

    public static void main (String[] args)
    {
        int arr[] = {10, 34, 5, 10, 3, 5, 10};
        printFreq(arr);
    }
}
```

Q11. How do you check if a key exists in a Map in Java?

Ans. Java's HashMap uses the containsKey(Object key) method to check if a given key is present in the map. This method checks if the HashMap has a mapping for the supplied key when it is called. The function returns true if the key is present in the HashMap and returns false otherwise
Example-:
```
import java.util.HashMap;
```

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

   HashMap<String, Integer> basket = new HashMap<String, Integer>();



      basket.put("Apples", 50);
   basket.put("Oranges", 30);

   System.out.println("Basket contains Apples: " +
basket.containsKey("Apples"));



      System.out.println("Basket contains Bananas: " +
basket.containsKey("Bananas"));
 }
}
```