ASSIGNMENT

1)How do arrayList, set and hashmap work internally ?

A) ARRAY LIST:

- The list interface allows duplicate elements
- The list maintains insertion order.
- We can add any number of null values.
- List implementation classes are Array List, LinkedList
- The list provides get() method to get the element at a specified index.
- If you need to access the elements frequently by using the index then we can use the list
- To traverse the list elements by using Listlterator.

EXAMPLE:

```
import java.util.*;

public class CollectionsInJava1

{

      public static void main(String[] args)

         {

         List<String> l = new ArrayList<>(); //List Implementation

         l.add("Sam");  //adding objects to list

         l.add("Sandy");

         l.add("Joe");

         l.add("Arya");

         l.add("Nik");

         System.out.println("List objects are:  " +l); // printing the list

         l.remove("Nik"); //removing objects from list

         System.out.println("After Removing Nik, List Objects are" +l);

      }

 }
```

SET:

- Set does not allow duplicate elements.
- Set do not maintain any insertion order.
- But in set almost only one null value.
- Set implementation classes are HashSet, LinkedHashSet, and TreeSet.

- Set does not provide get method to get the elements at a specified index
- If you want to create a collection of unique elements then we can use set
- Iterator can be used traverse the set elements

EXAMPLE:

```
import java.util.*;
 public class CollectinosInJava2
{
   public static void main(String[] args)
    {
       // Demonstrating Set using HashSet
       // Declaring object of type String
       Set<String> hash_Set = new HashSet<String>();
            // Adding elements to the Set
            // using add() method
       hash_Set.add("appi");
       hash_Set.add("appu");
       hash_Set.add("gopi");
       hash_Set.add("tofil");
       hash_Set.add("select");
             // Printing elements of HashSet object
       System.out.println(hash_Set);
    }
}
```

MAP:

- The map does not allow duplicate elements
- The map also does not maintain any insertion order.
- The map allows a single null key at most and any number of null values.
- Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap.
- The map does not provide get method to get the elements at a specified index
- If you want to store the data in the form of key/value pair then we can use the map.
- Through keyset, value, and entry set.

EXAMPLE:

```
import java.util.*;
class Collections4
{
int id;
String name,author,publisher;
int quantity;
public Collections4(int id, String name, String author, String publisher, int quantity)
{
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
   this.quantity = quantity;
}
}
public class MapExample
{
public static void main(String[] args)
{
   //Creating map of Books
   Map<Integer,Collections4> map=new LinkedHashMap<Integer,Collections4>();
   //Creating Books
   Collections4 b1=new Collections4(101,"Let us C","Yashwant Kanetkar","BPB",8);
   Collections4 b2=new Collections4(102,"Data Communications &
Networking","Forouzan","Mc Graw Hill",4);
   Collections4 b3=new Collections4(103,"Operating System","Galvin","Wiley",6);
   //Adding Books to map
   map.put(2,b2);
   map.put(1,b1);
   map.put(3,b3);
```

```
    //Traversing map
    for(Map.Entry<Integer, Collections4> entry:map.entrySet()){
       int key=entry.getKey();
       Collections4 b=entry.getValue();
       System.out.println(key+" Details:");
       System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

2) How to maintain order in set ?

A) Maintaining the order in set:

- Set is just an interface. In order to retain order, you have to use a specific implementation of that interface and the sub-interface SortedSet,
- For example TreeSet or LinkedHashSet.
- You can wrap your Set this way:

```
Set myOrderedSet = new LinkedHashSet
```

- The Set interface does not provide any ordering guarantees.Its sub-interface SortedSet represents a set that is sorted according to some criterion.
- In Java 6, there are two standard containers that implement SortedSet. They are TreeSet and ConcurrentSkipListSet.
- In addition to the SortedSet interface, there is also the LinkedHashSet class.
- It remembers the order in which the elements were inserted into the set, and returns its elements in that order.

❖ There are 2 different things:Sort the elements in a set.

- For which we have SortedSet and similar implementations.Maintain insertion order in a set.
- For which LinkedHashSet and CopyOnWriteArraySet (thread-safe) can be used.

3)What is the differences between ArrayList and LinkedList ?

A)

ArrayList	LinkedList
1) ArrayList internally uses a dynamic array to store the elements.	**1)** LinkedList internally uses a doubly linked list to store the elements.
2) Manipulation with ArrayList is slow because it internally uses an array.if any elements is removed from the array, all the bits are shifted in memory.	**2)** Manipulation with ArrayList is faster than ArrayList because it uses a doubly linked list,so no bit shifting is required in memory.
3) An ArrayList class can act as a list only because it implements List only.	**3)** An LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
4) Arraylist is better for storing and accessing tha data.	**4**) Linkedlist is better for manipulating tha data.
5) ArrayList is an inefficient memory utilization	**5)** LinkedList is an good memory utilization
6) The size of ArrayList needs to be declared before execution.	**6)** The size of LinkedList is variable thus need not be declared.
7) Memory is allocated at stack memory location.	**7)** Memory is allocated at heap memory location.
8) It contains less memory.	**8)** It contains more memory**.**
9) Insertion operation performed is slow, thus have time complexity O(n).	**9)** Insertion operation performed is fast, thus have time complexity O(1).
10) ArrayList can be used to store similar types of data	**10)** any type of data can be stored using LinkedList.

4)What is Fail Fast and Fast Safe ?

A) Fail Fast:

- This iterator directly works on the collection object itself.
- It doesn't allow to modify the collection while iterating, will throw ConcurrentModificationException.
- This iterator doesn't require any extra memory.
- Example of this iterator are ArrayList,HashMap,Vector,HashSet and these classes are in java.util package.

Example Program:

```
import java.util.HashMap;
 import java.util.Iterator;
  import java.util.Map;
public class FailFastDemo

{

   public static void main(String[] args)

      {
         Map<String, String> empName = new HashMap<String, String>();
         empName.put("Sam Hanks", "New york");
         empName.put("Will Smith", "LA");
         empName.put("Scarlett", "Chicago");
         Iterator iterator = empName.keySet().iterator();
         while (iterator.hasNext()) {
            System.out.println(empName.get(iterator.next()));
            // adding an element to Map
            // exception will be thrown on next call
            // of next() method.
            empName.put("Istanbul", "Turkey");
         }
      }
}
```

Fail Safe:

- This iterator works on the clone or copy of the collection object.
- It will allow to modify the collection while iterating , it doesn't throw any exception.
- This one requires extra memory, consumes heap.
- Example of this iterator are ConcurrentHashmap,CopyOnWriteArrayList and these classes are in java.util.concurrent package.

Example program:

```
import java.util.concurrent.CopyOnWriteArrayList;

import java.util.Iterator;

class FailSafeDemo {

   public static void main(String args[])

   {

      CopyOnWriteArrayList<Integer> list

        = new CopyOnWriteArrayList<Integer>(new Integer[] { 1, 7, 9, 11 });
```

```
    Iterator itr = list.iterator();
    while (itr.hasNext()) {
       Integer i = (Integer)itr.next();
       System.out.println(i);
       if (i == 7)
          list.add(15); // It will not be printed
       //This means it has created a separate copy of the collection
    }
}
```

5)What is HashCode , Equals and why to use it for collections?

A) HashCode():-

- The hashCode() method of objects is used when you insert them into a HashTable, HashMap or HashSet.
- When inserting an object into a hastable you use a key. The hash code of this key is calculated, and used to determine where to store the object internally.
- When you need to lookup an object in a hashtable you also use a key. The hash code of this key is calculated and used to determine where to search for the object.

Here are two rules that are good to know about implementing the hashCode() method in your own classes,

- if the hashtables in the Java Collections API are to work correctly:
- If object1 and object2 are equal according to their equals() method, they must also have the same hash code.
- If object1 and object2 have the same hash code, they do NOT have to be equal too.

In shorter words:

If equal, then same hash codes too.

Same hash codes no guarantee of being equal.

Here are two example implementation of the hashCode() method matching the equals() methods shown earlier:

```
public class Employee {
  protected long   employeeId;
  protected String firstName;
  protected String lastName;

  public int hashCode(){
    return (int) employeeId;
  }
}
public class Employee {
    protected long   employeeId;
    protected String firstName;
    protected String lastName;

  public int hashCode(){
    return (int) employeeId *
                firstName.hashCode() *
                lastName.hashCode();
  }
}
```

equals() method:

- When comparing two objects together, Java calls their equals() method which returns true if the two objects are equal,or false otherwise.
- Note that this comparison using equals()method is very different than using the == operator.
- The equals()method is designed to compare two objects semantically (by comparing the data members of the class), whereas the == operator compares two objects technically (by comparing their references i.e. memory addresses).

NOTE:

- The implementation of equals()method in the Object class compares references of two objects.

- That means we should override it in our classes for semantic comparison. Almost classes in the JDK override their own version of equals()method, such as String, Date, Integer, Double, etc.
- A typical example is String comparison in Java. Let’s see the following code:

```
String s1 = new String("This is a string");

String s2 = new String("This is a string");

 boolean refEqual = (s1 == s2);

boolean secEqual = (s1.equals(s2));

 System.out.println("s1 == s2: " + refEqual);

System.out.println("s1.equals(s2): " + secEqual);
```

output:

```
s1 == s2: false

s1.equals(s2): true
```