JAVA CHAPTER 5:- COLLECTION FRAMEWORK AND STREAM API

PROGRAM 1:-ARRY_LIST

```
import java.util.*;
 public class Arry_list{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
       list.add("Mango");
       list.add("Apple");
       list.add("Banana");
       list.add("Grapes");
       System.out.println(list);
 }
}
```

PROGRAM 2- COLECT STREAM

```
import java.util.List;
public class Colect_stream {
    public static void main(String[] args) {
        List<String> vowels = List.of("a", "e", "i", "o", "u");
     StringBuilder result = vowels.stream()
                .collect(StringBuilder::new, (x, y) -> x.append(y),
                (a, b) -> a.append(",").append(b));


     System.out.println(result.toString());
     StringBuilder result1 = vowels.parallelStream()
                .collect(StringBuilder::new, (x, y) -> x.append(y),
```

```
                (a, b) -> a.append(",").append(b));
        System.out.println(result1.toString());

    }


}
```

PROGRAM 3 -

```
//package JAVA_Theory.Chapter_5;


public class collection_framework{
    public static void main(String[] args) {
        /* what is collection = any group of individual object which
are represented as a
           single unit is known as the collectiom of the object
        */
        /* what is framework = A framework is a set of classes and
interfaces which provide
           a ready made architecture
        */
        /* what is collection framework = collection framework is java
API which provides
           architecture to store and manipulate group of object
        */
    }
}
```

PROGRAM 4- COMPRABLE

```
public class Comparable {
    public static void main(String args[]) {
        String str1 = "SoftwareTestingHelp";
        String str2 = "Java Series tutorial";
        String str3 = "Comparable Interface";
        System.out.println("str1:" + str1);
        System.out.println("str2:" + str2);
        System.out.println("str3:" + str3);
 int retval1 = str1.compareTo( str2 );
 System.out.println("\nstr1 & str2 comparison: "+retval1);

 int retval2 = str1.compareTo( str3 );
 System.out.println("str1 & str3 comparison: "+retval2);

 int retval3 = str2.compareTo("compareTo method");
 System.out.println("str2 & string argument comparison: "+retval3);

    }
 }
```

PROGRAM 5 - *EQUALS*

```java
import java.util.Scanner;


public class Equals {
    public static void main(String[] args) {
        String username="Noob"; String password="noob45";
        String use,pass;
        System.out.println("Enter your username & password");
        Scanner sc = new Scanner(System.in);
         use =sc.nextLine();
         pass= sc.nextLine();
         if(use.equals(username)){
            if(pass.equals(password)){
                System.out.println("welcome NOOB");
            }
         }
         else{
            System.out.println("please enter correct username &
password");
         }
         sc.close();
    }
}
```

PROGRAM 6 –FILTER_STREAM

```java
import java.util.Arrays;
import java.util.List;
```

```java
import java.util.stream.Collectors;

public class Filter_Stream
{
    public static void main(String[] args)
    {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,
10);

        List<Integer> evenNumbers = list.stream()
                    .filter(n -> n % 2 == 0)
                    .map(n -> n * n)
                    .collect(Collectors.toList());

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

PROGRAM 7 – HASH_CODE

```java
import java.util.Scanner;


class Hash_code{
    public static void main(String[] args){
            String password1,password2;
            System.out.println("Enter the password");
```

```
            Scanner sc = new Scanner(System.in);
            password1=sc.nextLine();
            password2=sc.nextLine();
            password1.hashCode();
            password2.hashCode();

System.out.println(password1.hashCode()+"\n"+password2.hashCode());
            if(password1.equals(password2)){
                System.out.println("same");
            }
            else{
                System.out.println("w");
            }
            sc.close();
    }
}
```

PROGRAM 8 – LINKED_LIST

```
import java.util.*;


public class Linked_list {


    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();
```

```
        ll.add("NOOB");
        ll.add("FF");
        ll.add(1, "In");


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

PROGRAM 9-MAP

```
import java.util.HashMap;
import java.util.Map;
public class Map_java {
    public static void main(String[] args) {
        Map map = new HashMap<String ,String >();
        map.put("A", "B");
        map.put("C", "B");
        map.put("D", "B");
        System.out.println(map);


    }
}
```

PROGRAM 10 – MAP_STREAM

```
import java.util.*;
class Map_stream {
    public static void main(String[] args)
    {
        System.out.println("The stream after applying " + "the
function is : ");
        List<Integer> list = Arrays.asList(3, 6, 9, 12, 15);
        list.stream().map(number -> number *
3).forEach(System.out::println);
    }
}
```

PROGRAM 11-STREAM

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;
public class Stream_a {
   public static void main(String[] args) {
      List<Integer> list = Arrays.asList(25, 50, 100, 200, 250, 300,
400, 500);
      System.out.println("List elements...");
      for (int res : list)
      {
         System.out.println(res);
      }
      Stream<Integer> stream = list.stream();
```

```
      System.out.println("Stream =
"+Arrays.toString(stream.toArray()));
   }
}
```

PROGRAM 12- TO_STRING METHOD

```
class To_string{
    int rollno;
    String name;
    String city;

    To_string(int rollno, String name, String city){
    this.rollno=rollno;
    this.name=name;
    this.city=city;
    }
    public String toString(){
     return rollno+" "+name+" "+city;
    }
    public static void main(String args[]){
      To_string s1=new To_string(101,"Raj","lucknow");
      To_string s2=new To_string(102,"Vijay","ghaziabad");

      System.out.println(s1);
      System.out.println(s2);
    }
   }
```

PROGRAM 13 – TREE MAP

```java
import java.util.TreeMap;
public class Tree_hash_map {
    public static void main(String[] args) {
        TreeMap <String,String> vi = new TreeMap<String,String>();
        vi.put("J", "JAVA");
        vi.put("P", "PYTHON");
        vi.put("C", "C_SHARPE");
        System.out.println(vi);


    }
}
```

POINT 1-

5.1 Implementing equals, hashcode and toString methods

1|Java equals()[

1.The java equals() is a method of lang.Object class, and it is used to compare two objects.

2.To compare two objects that whether they are the same, it compares the values of both the object's attributes.

3.By default, two objects will be the same only if stored in the same memory location.

4.syntax - public boolean equals(Object obj)

]

2|hashcode()[

1.A hashcode is an integer value associated with every object in Java, facilitating the hashing in hash tables.

2.To get this hashcode value for an object, we can use the hashcode() method in Java. It is the means hashcode()

method that returns the integer hashcode value of the given object.

3.Since this method is defined in the Object class, hence it is inherited by user-defined classes also.

4.The hashcode() method returns the same hash value when called on two objects, which are equal according to the

equals() method. And if the objects are unequal, it usually returns different hash values.

5.syntax - public int hashCode()

]

4|toString[

1.If you want to represent any object as a string, toString() method comes into existence.

2.The toString() method returns the String representation of the object.

3.If you print any object, Java compiler internally invokes the toString() method on the object. So overriding the

toString() method, returns the desired output, it can be the state of an object etc. depending on your implementation.

]

POIN-2

5.2 The Comparable interface and Comparator interface

1| Comparable interface[

1.Comparable interface sorts the list structures like Arrays and ArrayLists containing custom objects.

2.Once the list objects implement Comparable interface, we can then use the Collections.sort () method

or Arrays.sort () in case of the arrays to sort the contents.

3.The method ‘compareTo’ of the Comparable interface is used to compare the current object to the given object.

4.We can use the compareTo () method to sort:

1|String type objects

2|Wrapper class objects

3|User-defined or custom objects

5.syntax- public int compareTo(Object obj)

6.program:- see Comparable.java file

]

2|Comparator interface[

1.Using the Comparator interface, we can create more than one comparator depending on how many fields we want to

use to sort the custom objects.

2.Using the comparator interface, supposing we want to sort the custom object on two member fields name and age,

then we need to have two comparators, one for name and one for age.

3.The comparator interface contains a ‘compare’ object that is used to compare objects of two different classes.

4.The Comparator interface is a part of the java.util package and apart from the compare method; it also contains

another method named equals

5.If you want to sort the elements of a collection, you need to implement Comparator interface.

6. syntax :- public int compare (Object obj1, Object obj2);

7. program - optional

]

3|Difference Between Comparable Vs Comparator[

*Comparable:-1.The comparable interface provides single field sorting.

2.Part of the java.lang package.

3.Provides compareTo () method to sort elements.

4.Uses Collections.sort (List) to sort elements.

5.Comparable interface sorts object as per natural ordering.

*Comparator:-1.Comparator interface provides multiple fields sorting.

2.Part of java.util package.

3.Provides compare () method to sort elements

4.Uses Collections.sort (List, Comparator) to sort the elements.

5.Comparator interface sorts various attributes of different objects.

]

POINT 3-

5.3 The Collection interface, List interface, Map interface

1|Collections in Java[

1.The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects

2.Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion,

manipulation, and deletion

3.A Collection represents a single unit of objects, i.e., a group.

]

2|What is a framework in Java[

1.It provides readymade architecture.

2.It represents a set of classes and interfaces.

3.It is optiona

]

3|Collection Interface[

1|The Collection interface is the interface which is implemented by all the classes in the collection framework. It

declares the methods that every collection will have. In other words, we can say that the Collection interface builds

the foundation on which the collection framework depends

2|Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(),

etc. which are implemented by all the subclasses of Collection interface

]

4| List interface[

1|List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can

store the ordered collection of objects. It can have duplicate values

2|List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack

3|To instantiate the List interface, we must use :

1. List <data-type> list1= new ArrayList();
2. List <data-type> list2 = new LinkedList();
3. List <data-type> list3 = new Vector();
4. List <data-type> list4 = new Stack();

]

5| Map Interface[

1|The map interface is present in java.util package represents a mapping between a key and a value.

2|The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit differently

from the rest of the collection types. A map contains unique keys

3|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.

4|duplicate keys are not allowed but value can be duplicate

4|type of maps :-

1|HashMap (IMP)
2|IdentityHashMap
3|WeakHashMap
4|SortedMap
5|Tree_Map (IMP)

]

POINT 4-

5.4 Using Lists: ArrayList and LinkedList classes

1| Array_list class[

1|Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit.

2|We can add or remove elements anytime. So, it is much more flexible than the traditional array.

3|it is found in the java.util package. It is like the Vector in C++.

4|syntax-public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable

5|view the program- Arry_list.java

]

2|LinkedList class[

1|Like An ArrayList in java, LinkedList class in java is also a famous data structure. But Java LinkedList doesn't store

elements in contiguous locations like ArrayList

2|In Java LinkedList, each element linked with each other using pointers.

3|The LinkedList class in java is a part of the Java Collection Framework which implements the List interface, Deque interface,

and extends the AbstractList class

4|Declaration :-LinkedList<E> NameOfLinkedList = new LinkedList <E>(collectionName c);

5|view the program- Linked_list.java

]

POINT 5-

5.5 Using Maps: HashMap and TreeMap classes

1| HashMap class[

1|The java.util.HashMap class is the Hash table based implementation of the Map interface.Following are the important

points about HashMap –

2|This class makes no guarantees as to the iteration order of the map; in particular, it does not guarantee that the

order will remain constant over time.

3|This class permits null values and the null key.

4|Class declaration:-

```
public class HashMap<K,V>
    extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable
```

5|Parameters:-

K – This is the type of keys maintained by this map.

V – This is the type of mapped values.

]

2|TreeMap class[

1|The java.util.TreeMap class is the Red-Black tree based implementation of the Map interface.Following are the important points about TreeMap

2|The TreeMap class guarantees that the Map will be in ascending key order.

3|The Map is sorted according to the natural sort method for the key Class, or by the Comparator provided at map creation time, that will

depend on which constructor used.

4|Class declaration:-

```
public class TreeMap<K,V>
    extends AbstractMap<K,V>
    implements NavigableMap<K,V>, Cloneable, Serializable
```

5|Parameters:-

K – This is the type of keys maintained by this map.

V – This is the type of mapped values.

]

POINT 6-

5.6 Stream API

|5.6.1--> Retrieving a Stream from a Collection

|5.6.2--> Filtering Streams using filter method

|5.6.3--> Mapping Streams using map method

|5.6.4--> Collecting Streams into Collections using collect method

|5.6.5--> Reducing Streams to values using reduce method

|5.6.6--> Using forEach method

1|what is Stream:-Java provides a new additional package in Java 8 called java.util.stream. This package consists of classes,

interfaces and enum to allows functional-style operations on the elements. You can use stream by importing java.util.stream

package.

2|stream provide following features[

1|Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O

channel, through a pipeline of computational operations.

2|Stream is lazy and evaluates code only when required.

3|The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be

generated to revisit the same elements of the source.

4|You can use stream to filter, collect, print, and convert from one data structure to other etc

5|to create the strem :-

```
Stream<Integer> stream = list.stream();

Arrays.toString(stream.toArray()));
```

6| view the program :- Stream_a.java

]

2|Filtering Streams using filter method[

1|filter() is a intermediate Stream operation.

2|It returns a Stream consisting of the elements of the given stream that match the given predicate.

3|The filter() argument should be stateless predicate which is applied to each element in the stream to determine if

it should be included or not.

4|Predicate is a functional interface. So, we can also pass lambda expression also.

5|It returns a new Stream so we can use other operations applicable to any stream.

6|syntax :- Stream<T> filter(Predicate<? super T> condition)

7| view program Filter_Stream.java

]

3|Mapping Streams using map method[

*What is mapping with Streams:-1|Mapping in the context of Java 8 Streams refers to converting or transforming a Stream

carrying one type of data to another type.

2| The map() method helps us transform one type of stream to another type of stream.

*Defining map(); method:-<R> Stream<R> map(Function<? super T,? extends R> mapper)

*program:- see Map_stream.java

]

4|Collecting Streams into Collections using collect method[

1|This Stream method is a terminal operation which reads given stream and returns a collection like List or Set (or Map)

2|There are 2 variants of collect() method and we are going discuss only one as mentioned below

3|Method signature :- <R, A> R collect(Collector<? super T, A, R> collector)

4|collect() method accepts Collectors which is a final class with various utility methods to perform reduction on the

given Stream

5|Collectors.toCollection() to convert any Collection class like ArrayList/HashSet

6|program Colect_stream.java

]