UNIT 4 JAVA API

Structure **Page No.**

4.0 INTRODUCTION

In this unit, you are going to learn about Java Application Programming Interface (API) for Date, Time and the Collections. The Java API is a group of pre-written packages, classes, interfaces along with their methods and attributes. You can import a particular package or a class as per your requirement and use its pre-written code in your program to perform basic and advanced programming tasks. This, in turn, reduces the lines of code of your program. Java API is a component of Java. Development Kit. Java 8 introduced a completely new API in the java.time package that comprises a large number of classes to perform various Date and Time operations.

Collections API in Java is used for storing and manipulating a group of objects. The Collection framework is contained in java.util package. Students are advised to refer to Figure 1 in order to understand this section on the Introduction of Java API. The Iterable interface is the root interface of all collection classes and is extended by the Collection interface. All derived classes of collection interface implement the Iterable interface.

The Collection framework comprises interfaces like Collection (Figure 1) and Map (Figure 2). List, Set and Queue are the subinterfaces of the Collection interface, as shown in Figure 1. For the implementation of the List subinterface, there are Standard classes such as ArrayList, LinkedList and Vector. Further, the Vector class is inherited by Stack class. Next, for implementation of the Set subinterface, there are Standard classes such as, HashSet, LinkedHashSet and TreeSet. For the implementation of Queue subinterface, PriorityQueue class and ArrayDeque class are used.

There are few classes that provide an implementation of the Map interface. Student should refer Figure 2, which shows hierarchy of Map interface. AbstractMap class provides an implementation of the Map interface, and further HashMap class extends AbstractMap class in order to use a hash table. As shown in Figure 2, the SortedMap interface extends the Map interface and is inherited by the TreeMap class.

Finally, as discussed earlier Collection interface extends the Iterable interface, which is the root interface of all collection classes. The iterator() method of the Iterable interface facilitates the traversal of the elements in a collection. You can perform searching, sorting, insertion, and deletion easily and efficiently with Java Collections.

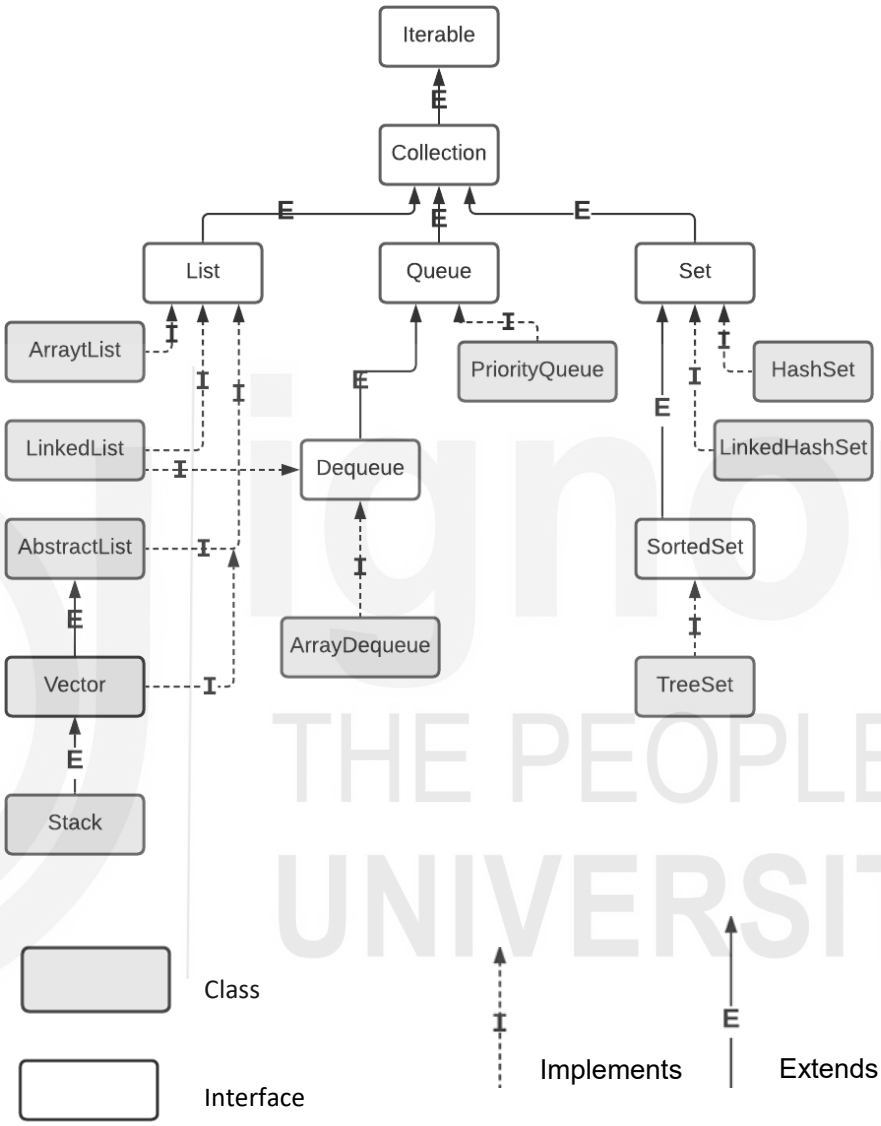

Figure 1. Collection Hierarchy in Java

4.1 OBJECTIVES

After going through this unit, you will be able to:

- explain the benefits of Java APIs,
- write programs that compare different date and time values,
- describe the use of various interfaces and classes of collection framework,
- use various interfaces, namely Set, Map and List, in order to write programs in a simpler way,

- use Traversal of Set using Iterator, and
- write programs by using the methods to perform various operations on maps.

4.2 DATE AND TIME

The older Date class before Java 8, was provided by java.util package, and it offered various functions related to date and time, as described in Table 1. There are two constructors for the Date class as given below: -

- Date( ) : It constructs Date object and initializes it with the current date and time.
- Date(long millisec) : It receives an argument corresponding to the quantity of milliseconds that have passed after twelve midnight, 1st January, 1970.

Table 1: Date class and its methods

S.No.	Method	Description
1	boolean after(Date dt)	It gives true when the Date object that invoked **after** the method has a value of date which is later than the one stated by object dt, else gives false.
2	boolean before(Date dt)	Gives true when the Date object that invoked **before** the method has a value of date which is before the one stated by object dt, else gives false.
3	Object clone( )	Returns duplicate object of the Date object that invoked clone method.
4	int compareTo (Date dt)	Compares the value date of invoking object to the value of date of dt object. If both are equal, this method returns 0. It returns a negative value if the value of date of invoking object is earlier than that of dt object. It returns a positive value if the value of date of invoking object is after the date of the dt object.
5	int compareTo (Object ob)	This function is identical to compareTo(Date) provided ob is an object of class Date. If not, an exception named ClassCastException is thrown.
6	boolean equals (Object dt)	Gives true when the Date object that invoked **equals** method has the same time and date value as mentioned in dt object, else the method gives false.
7	long getTime( )	It gives value of time in terms of number of milliseconds passed since 1st January 1970 (midnight).

8	int hashCode( )	Gives the hash code of the object that invoked this method
9	void setTime(long t)	Sets the current time and date as stated by t. The time in long corresponds to the time in milliseconds that have passed since midnight, 1st January, 1970.
10	String toString( )	Changes the Date object that invoked this method into a string, and the same is returned.

```
import java.util.Date;
class DateEx
{
   public static void main(String args[])
   {
    // Create an object of Date class
    Date date = new Date();
    // Print time and date
    System.out.println(date);
    // Retrieve the  number of milliseconds since midnight, January 1, 1970 GMT
    long msec = date.getTime();
    System.out.println("Milliseconds since Jan. 1, 1970 GMT = " + msec);
    // Create a Date object with msec value (as long datatype)as passed to constructor
    Date date1 = new Date(15991178175891);
    // Retrieve the number of milliseconds in date1 object
    long msec1 = date1.getTime();
    // Compare current date with date1 object and print message accordingly
    int result = date.compareTo (date1);
    if (result==0)
    System.out.println("Both dates are same");
    else if (result==1)
    System.out.println(msec+" is later than "+msec1);
    else
    System.out.println(msec+" is earlierr than "+msec1);
   }
}
```

Output

```
Tue Nov 03 12:04:32 IST 2020
Milliseconds since Jan. 1, 1970 GMT = 1604385272113
1604385272113 is later than 1599117817589
```

Date Formatting Using SimpleDateFormat

In order to give an understandable format for date, there is a class called SimpleDateFormat. This class allows us to choose any pattern for displaying date-time. You can also change a string to date by means of an appropriate method of this class.

```
import java.util.*;
import java.text.*;
public class Date_form
{
  public static void main(String args[])
  {
    Date date = new Date( );
    SimpleDateFormat date_format = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
```

```
    System.out.println("Current Date: " + date_format.format(date));
  }
}
```

Output

```
Current Date: Tue 2020.11.03 at 12:16:55 PM IST
```

There are various codes for SimpleDateFormat. These are given below in Table 2

Table 2: SimpleDateFormat codes

Character	Description	Example
G	Era designator	AD
y	Year in four digits	2001
M	Month in year	July or 07
d	Day in month	10
h	Hour in A.M./P.M. (1~12)	12
H	Hour in day (0~23)	22
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	234
E	Day in week	Tuesday
D	Day in year	360
F	Day of week in month	2 (second Wed. in July)
w	Week in year	40
W	Week in month	1
a	A.M./P.M. marker	PM
k	Hour in day (1~24)	24
K	Hour in A.M./P.M. (0~11)	10
z	Time zone	Eastern Standard Time
'	Escape for text	Delimiter
"	Single quote	`

This Date-Time API was not thread-safe and offered very few operations related to date. With the advent of Java 8, these drawbacks were removed as a new Date-Time API was introduced, which is immutable, doesn't have setter methods and includes many date operations. Two important classes present in this API are mentioned below: -

- LocalDate: It is a simplified date-time API, and there is not any complexity of timezone handling.
- ZonedDateTime: It is a special date-time API that makes you deal with various timezones in an easy manner.

☞ Check Your Progress-1

1) What do you understand by the Collection framework in Java?

……

……

……

……

2) Which of these classes is not part of Java's collection framework?

a) Array

b) Maps

c) Stack

d) Queue

3) How do you know if two date objects are equal?

……

……

……

……

4.3 SET INTERFACE

We can define a set as a group of dissimilar objects. We all know that sets are one of the most fundamental concepts in mathematics. Java provides Set as an interface that extends the Collection interface, and hence it has access to all the methods inherited from the Collection interface, as explained in Table 3.

Table 3: Description of methods of Collection interface

No.	Method	Description
1	public boolean add(object o)	It is written to insert an object in the collection. Returns true if object is added.
2	public boolean addAll(Collection c)	The purpose is to insert the elements of the mentioned collection c, in the collection that invoked the addAll method. The addAll() method returns true if the collection got changed because of the method call otherwise, false is returned.
3	public boolean remove(Object o)	The purpose of this method is to remove an object from a particular collection that invoked this method. It returns true if object is removed.
4	public boolean removeAll (Collection c)	The purpose of this method is to remove all the elements of the mentioned collection c, from the collection that invoked this method. The removeAll() method returns true only if the invoking collection was changed otherwise, false is returned.
5	default boolean removeIf (Predicate filter)	The purpose of this method is to remove all the elements of the collection that satisfy the given predicate. The parameter 'filter' represents a predicate which returns true for the elements to be removed. This method returns true if we are able to remove elements.
6	public boolean retainAll (Collection c)	The purpose of this method is to remove all the elements of invoking collection except the mentioned collection c. It returns true if it is successful.
7	public int size()	The purpose of this method is to tell the count of elements of the invoking collection.
8	public void clear()	This method deletes all elements of the mentioned collection.
9	public boolean contains(Object o)	With this method, we can search an object in the collection. The method will give true when the object is found.
10	public boolean containsAll(Collection c)	With this method, we can search the stated collection in the invoking collection. Again, the method will give true when the collection c is found.
11	public Iterator iterator()	It gives an iterator over elements of the invoking collection.
12	public Object[] toArray()	It changes the invoking collection to an array and returns the same.

13	public <T> T[] toArray(T[] a)	It changes the invoking collection to an array and returns the same. In this method, the returned array's runtime type is same as that of the mentioned array.
14	public boolean isEmpty()	This method would test whether the invoking collection is empty and returns true if it is so.
15	public boolean equals(Object element)	This method tests the two collections for similarity and returns true if they match.
16	public int hashCode()	This method gives the hash code value of the invoking collection.

Set represents an unordered group of members, and the members must be unique. So, a false is returned by the add() method when you try to add duplicate elements in it. You are allowed to have just one null value in a Set. There are no extra methods defined by Set; only the methods of collection interface are available for use with Set.

4.3.1 Classes that implement Set interface

The Set interface is implemented by HashSet and LinkedHashSet class. There is an interface named SortedSet that extends Set interface, and the class TreeSet implements the Sortedset interface.

HashSet class: Its purpose is to produce a collection and keep it in a hash table. Each element in the Set has a unique value of its own, called hash code which is utilized as an index at which element related to that hash code is present. Hashing ensures that the execution time of operations like add(), remove(), remain constant irrespective of the size of the HashSet. The members are inserted in HashSet irrespective of the order of insertion, and the order depends on the hashcode of elements. Constructors for HashSet class are given below: -

- HashSet( ) : Constructs a default HashSet which is empty.
- HashSet(Collection coll) : Initializes the HashSet with the members of collection coll.
- HashSet(int cap) : Initializes the HashSet with the size of the integer value of cap.
- HashSet(int cap, float fill_Ratio) : This constructor is used to initialize the data members - capacity and the fill ratio, of HashSet. The valid value of fill ratio has to be between 0.0 and 1.0, and it governs how full the HashSet is allowed to be before it is resized upwards. If the number of elements in HashSet is more than the capacity (cap) of the Hashset multiplied by its fill ratio, then the hash set is lengthened.

Corresponding to the first three constructors, the value of fill ratio is 0.75.

For instantiating a object of the HashSet class, we have to mention the data type of elements of set. For example:-

```
HashSet<datatype> hset = new HashSet<datatype>();
```

Here, datatype is the type of elements that can be stored in hset.

```
import java.util.*;
class HashSetProgram
{
   public static void main(String args[])
   {
      // create a HashSet
      HashSet<String> hset1 = new HashSet<String>();
      // add elements to the hash set
      hset1.add("Java");
      hset1.add("C++");
      hset1.add("Python");
      hset1.add("R");
      hset1.add("Matlab");
      hset1.add("Scala");
      System.out.println(hset1);
   }
}
```

Output

```
[Java, C++, R, Scala, Python, Matlab]
```

We can see that the set elements are not printed as per the order of insertion of elements.

Traversal of Set using Iterator

If we want to pass over the elements of any collection, we can make use of Iterable interface, which is the most important interface of the collection framework and is at the root of the hierarchy. As we know that the Iterable interface is extended by collection interface; hence all the interfaces and classes of collection framework implement this interface. The most important feature of the Iterable interface is to deliver an iterator for the collections. Iterable interface has just one method which is abstract and whose name is iterator(). It returns an object of type Iterator.

Iterator iterator();

The methods declared by Iterator are described in Table 4.

Table 4: Description of methods of Iterator interface

S.No.	Method	Description
1.	boolean hasNext( )	Gives true only if there are more elements in the collection. Else, this method returns false.
2.	Object next( )	Gives the next element. This method throws NoSuchElementException if there does not exist any next element.
3.	void remove( )	Eliminates the current element. It throws IllegalStateException if an attempt is made to invoke remove( ) method that is not preceded by a call to next( ) method.

In order to use an iterator to pass over the elements of a collection, you should follow the following steps.

1. Get an iterator to the start of the collection by invoking the iterator( ) method of the collection.

2. Write a while loop that invokes hasNext( ). The loop would iterate till the time hasNext( ) returns true.

3. Inside the loop, get and display each element by invoking next( ) method.

Same program is written here again by using iterator.

```
import java.util.*;
class HashSetwithIterator
{
   public static void main(String args[])
   {
      // create a HashSet
      HashSet<String> hset1 = new HashSet<String>();
      // add elements to the hash set
      hset1.add("Java");
      hset1.add("C++");
      hset1.add("Python");
      hset1.add("R");
      hset1.add("Matlab");
      hset1.add("Scala");
      // traverse the set using Iterator and print each element one by one
      Iterator<String> itr = hset1.iterator();
      while(itr.hasNext())
      {
         System.out.println(itr.next());
      }
   }
}
```

Output

```
Java
C++
R
Scala
Python
Matlab
```

LinkedHashSet class

A LinkedHashSet is an ordered form of HashSet that keeps a doubly-linked list of the elements in the Set. The sequence of insertion is preserved in LinkedHashSet and while iterating through LinkedHashSet, the elements are returned in the order of insertion. This is an advantage over HashSet. The hash code is used for indexing in the same way as in HashSet.
Constructors for LinkedHashSet class are given below,

Given below are the list of constructors supported by the LinkedHashSet:

- LinkedHashSet(): Constructs a default LinkedHashSet which is empty.
- LinkedHashSet(Collection Coll): This constructor would Initialize the LinkedHashSet with the elements of collection Coll.
- LinkedHashSet(int cap): This constructor would initialize the HashSet with capacity of the integer value of cap parameter.

- LinkedHashSet(int capacity, float fillRatio): This particular constructor would initialize both the capacity and the fill ratio of the LinkedHashSet. The permissible value of fill ratio is between 0.0 and 1.0, and it governs how full the HashSet can be before it is resized upwards. If the number of elements in LinkedHashSet is greater than the capacity of the LinkedHashset multiplied by its fill ratio, then it is expanded.

For the first three constructor functions, the value of the fill ratio is 0.75.

All methods of Collection interface and Iterable interface are available to LinkedHashSet class.

In order to instantiate an object of LinkedHashSet class, we have to mention the data type of its elements. For example:-

```
LinkedHashSet<datatype> Lhset = new LinkedHashSet<datatype>();
```

Here, datatype is the type of elements that can be stored in Lhset.

Let's see a program based on LinkedHashSet class.

```
import java.util.*;
class LinkedHSet
{
   public static void main(String args[])
   {
      // create a LinkedHashSet
      LinkedHashSet<String> Lhset = new LinkedHashSet<String>();
      // add elements to the LinkedHashSet
      Lhset.add("Operating System");
      Lhset.add("Java Programming");
      Lhset.add("Data Structures");
      Lhset.add("Database Management System");
      Lhset.add("Computer Networks");
      Lhset.add("Computer Architecture");
      // traverse the set using Iterator and print each element one by one
      Iterator<String> itr = Lhset.iterator();
      System.out.println("Original contents of LinkedHashset");
      while(itr.hasNext())
      {
            System.out.println(itr.next());
      }
      //Remove an element from the LinkedHashSet
      Lhset.remove("Data Structures");
      // traverse the set again
      System.out.println("Elements of LinkedHashset after removing 'Data Structures' ");
      System.out.println(Lhset);
   }
}
```

Output

```
Original contents of LinkedHashset
Operating System
Java Programming
Data Structures
Database Management System
Computer Networks
Computer Architecture
Elements of LinkedHashset after removing 'Data Structures'
[Operating System, Java Programming, Database Management System, Computer Net
Computer Architecture]
```

Students can notice here that the elements iterate in the order of their insertion.

4.3.2 Interface that extends Set interface

Sorted Set Interface implemented by TreeSet class

This interface extends Set interface and affirms that the elements of set are sorted in increasing order. It defines first() and last() methods to quickly return the first and last element respectively. This interface is implemented by **TreeSet class**. The important features of TreeSet class are given below: -

- It uses a tree for storage of the elements.
- Elements can be accessed and retrieved very fast.
- TreeSet class does not allow null element to be stored.
- As TreeSet class extends Sorted Set interface, automatically it maintains ascending order.
- Although elements are stored in increasing order, we can pass through in decreasing sequence also by using method TreeSet.descendingIterator().

The constructors supported by the TreeSet class are given below: -

- TreeSet(): Constructs a default TreeSet which is empty.
- TreeSet(Collection C): Initializes the TreeSet with the elements of collection c.
- TreeSet(Comparator cmp): Creates an empty TreeSet that will be sorted as per the value of comparator specified by cmp..
- TreeSet(SortedSet sset): Creates a treeset that comprises of members of SortedSet sset.

You can instantiate an object of TreeSet class in one of the following ways.

```
TreeSet<datatype> tset = new TreeSet<datatype>();
```

Here, datatype is the type of elements that are allowed to be stored in tset.
Another way to instantiate an object of TreeSet is by using SortedSet interface, as given below: -

```
SortedSet<datatype> tset = new TreeSet<datatype>();
```

Let's see a program based on TreeSet class.

```
import java.util.*;
class TreeSetProg
{
  public static void main(String args[])
  {
    // create a TreeSet
    TreeSet<String> tset1 = new TreeSet<String>();
    // add elements to the TreeSet
    tset1.add("India");
```

```
        tset1.add("Australia");
        tset1.add("USA");
        tset1.add("Canada");
        tset1.add("Japan");
        // traverse the set using Iterator and print each element one by one
        Iterator<String> itr = tset1.iterator();
        System.out.println("Original contents of TreeSet, by default in ascending order");
        while(itr.hasNext())
        {
           System.out.println(itr.next());
        }
        //Print the last element
        System.out.print("The last element is ");
        System.out.println(tset1.last());
        // traverse the treeset again but now in descending order
        System.out.println("Contents of TreeSet in descending order are...");
        Iterator<String> desc_itr= tset1.descendingIterator();
        while(desc_itr.hasNext())
        {
           System.out.println(desc_itr.next());
        }
    }
}
```

Output

```
Original contents of TreeSet, by default in ascending order
Australia
Canada
India
Japan
USA
The last element is USA
Contents of TreeSet in descending order are...
USA
Japan
India
Canada
Australia
```

☞ Check Your Progress-2

1) What do you understand by Iterator in the Java Collection Framework?

……

……

……

……

2) What is the HashSet class in Java, and how does it store elements?

……

……

……

……

3) Differentiate between TreeSet and HashSet?

……

……

……

……

4) What is the initial fill ratio of HashSet?

a) 1.5
b) 0.5
c) 0.75
d) 1.0

5) What is the output of the following program

```
import java.util.*;
class test1
{
   public static void main(String[] args)
   {
      HashSet<String> set=new HashSet<String>();
      set.add(null);
      set.add("One");
      for (String s: set)
      System.out.println(s);
   }
}
```

4.4 MAP INTERFACE

Map interface is a part of java.util package but map is not a collection as it does not implement the collection interface. Map interface represents a mapping between a key and a value. A map contains unique keys though the values for two or more keys can be same. Each key can map to at the most one value only. A Map is suitable whenever we want to search, update or delete items on the basis of a key. An example of map is zip code (key) and city (value). The hierarchy of classes and interfaces of Map interface is shown in Figure 2. The methods to perform various operations on maps are given in Table 5.

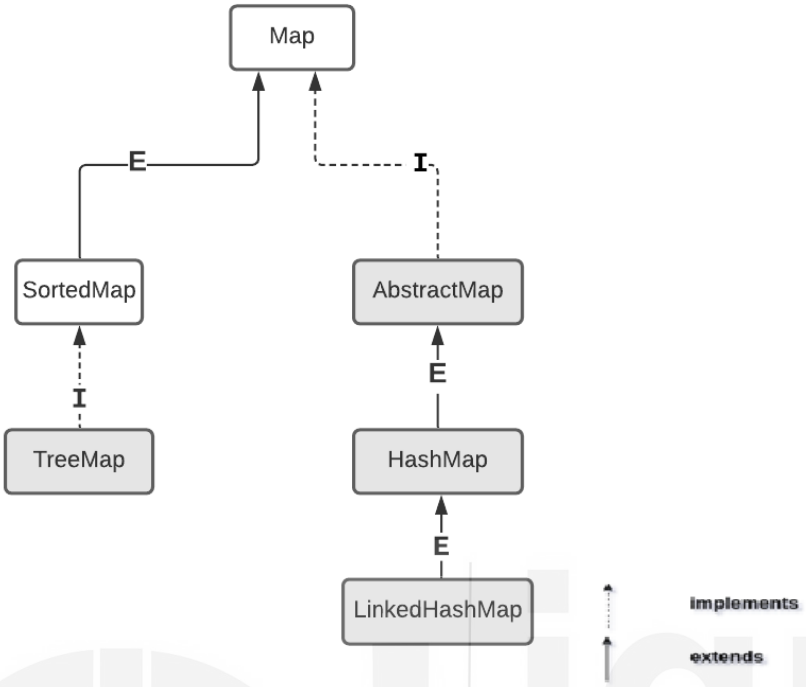

Figure 2 Map interface of Collections Framework

Table 5: Description of methods of Map interface

S.No.	Method	Description
1	void clear()	Deletes all key/value pairs from the invoking map.
2	boolean containsKey(Object k)	Gives true if the map object that invoked this method has k as one of the keys, else it gives false.
3	Boolean containsValue(Object obj)	It gives true if the map object that invoked this method has obj as one of the values, else its gives false.
4	Set entrySet( )	This method gives a Set that comprises of all the elements of the map. The set has in it all objects of type Map.Entry. We can use this method to provide a Set-view of the invoking map.
5	boolean equals(Object ob)	It gives true if ob is a Map and has the same entries as invoking Map. Else, it gives false.
6	Object get(Object obj)	It gives the value corresponding to the key obj.

7	int hashCode( )	It gives the hash code of the map that invoked this method.
8	boolean isEmpty( )	If the object that invoked this method does not have any element in it, then it gives true, else false.
9	Set keySet( )	This method produces a Set that has all the keys of the invoking map. It is used to provide a set-view of only the keys present in the invoking map.
10	Object put(Object k, Object v)	It adds an element in the invoking map. If there is already an element in the map having key same as k, then the value of that key is overwritten. Here, k corresponds to the key and v corresponds to the value. This method gives null if the key was not present already in the map. If a key was present, then the previous value linked to that key is returned.
11	void putAll(Map mp)	Adds all the elements of map mp into the invoking map.
12	Object remove(Object k)	Deletes the entry of map whose key is equal to k.
13	int size( )	It gives the number of key/value pairs that are present in the invoking map.
14	Collection values( )	This method gives a collection-view of only the values of the map.

4.5 HASHMAP CLASS

HashMap class implements the Map interface. A hash table is used by HashMap in order to implement Map interface. The advantage of hash table is that the time for basic operations like get() and put() remains constant irrespective of size of Map. Given below are the constructors for instantiating an object of HashMap.

- HashMap( ) : It instantiates a default hash map.
- HashMap(Map mp) : It initializes the hash map with the elements of m.
- HashMap(int cap) : It constructs a hash map with a capacity equal to argument cap.
- HashMap(int cap, float fillR): This constructor uses the arguments cap and fillR to initialize both the capacity and fill ratio. The gist of capacity and fill ratio remains the same as for HashSet, discussed earlier.

You should remember that there is no guarantee of the order of elements in the case of HashMap. So, the order in which elements are inserted into a hash map may not match the order of their retrieval by the iterator.

You can instantiate an object of HashMap class by using one of the given below techniques.

1. Non-Generic way (Old Style)

   Map mp=new HashMap(); or HashMap mp=new HashMap();

   Anything involving HashMap or Map without a datatype argument (the angle brackets < and > and the part between them) is a raw type that is an older approach and shouldn't be used. A raw type is not generic and lets you do hazardous things.

2. Generic way (New Style)

Map<T1,T2> mp1=new HashMap<T1,T2>();

T1, T2 − The generic type parameter passed to the generic interface Map and its implementation class HashMap, during map declaration.

Generics declaration allows compiler to check the 'mp1' usages during compile-time and it is very safe.

We will be using the Generic way in the example given below: -

```
import java.util.*;
class HashMapEx
{
   public static void main(String args[])
      {
        Map<Integer,String> map1=new HashMap<Integer,String>();
        map1.put(1,"Anita");
        map1.put(2,"Charu");
        map1.put(3,"Geeta");
        System.out.println();
        System.out.print("Map Elements :-" +map1);
        System.out.println();
        //For traversing Map, convert it into Set
        Set set=map1.entrySet();//Converting to Set so that we can traverse
        Iterator itr1=set.iterator();
        System.out.println("Map elements by traversing the map after being converted to set");
        while(itr1.hasNext())
        {
          //Converting to Map.Entry so that we can get key and value separately
          Map.Entry entry=(Map.Entry)itr1.next();
          System.out.println(entry.getKey()+" "+entry.getValue());
        }
        System.out.println("Number of elements in map is "+map1.size()+" elements.");
        boolean b=map1.containsKey(1);
        System.out.println("It is "+b+" that the map conatins an element with key 1");
        //Remove the entry whose key is 2
        map1.remove(2);
        System.out.println("Contents of map after deleting element with key 2 are "+map1);
        if (map1.isEmpty())
        System.out.println("Map is empty");
        else
        System.out.println("Map is not empty");
```

```
        //Delete all elements of map
        map1.clear();
        System.out.println("All elements cleared");
        if (map1.isEmpty())
        System.out.println("Now Map is empty");
        else
        System.out.println("Now Map is not empty");
    }
}
```

Output

```
Map Elements :-{1=Anita, 2=Charu, 3=Geeta}
Map elements by traversing the map after being converted to set
1 Anita
2 Charu
3 Geeta
Number of elements in map is 3 elements.
It is true that the map conatins an element with key 1
Contents of map after deleting element with key 2 are {1=Anita, 3=Geeta}
Map is not empty
All elements cleared
Now Map is empty
```

Given below is a program that uses HashMap for storing employee ids with their names. This program is menu-driven and takes input from the user, and accordingly uses an appropriate method to display the result.

```
import java.util.*;
class HashMapEx1
{
   public static void main(String args[])
    {
      Scanner sc = new Scanner(System.in);
      Map<Integer,String> map1=new HashMap<Integer,String>();
      map1.put(401,"Ramesh");
      map1.put(205,"Balram");
      map1.put(156,"Vinod");
      map1.put(555,"Geeta");
      System.out.println();
      int choice=1;
      do
      {
        System.out.println("Main Menu");
        System.out.println("1: Size of Map");
        System.out.println("2: Remove all elements of Map");
        System.out.println("3: Check if hashmap is empty?");
        System.out.println("4: Search for an employee");
        System.out.println("5: Input Employee ID and display name of that employee");
        System.out.println("6: Display names of all employees");
        System.out.println("7: Display employee id with their names");
        System.out.println("8: Exit");
        System.out.println("Enter choice: ");
        choice = sc.nextInt();
        switch (choice)
        {
          case 1:   System.out.println("Size of the hash map: "+map1.size()+" records");
                    break;
```

```
            case 2:   map1.clear();
                      System.out.println("The New map: " + map1);
                      break;
            case 3:   boolean result = map1.isEmpty();
                      System.out.println("It is " + result + " that hash map is empty");
                      break;
            case 4:   System.out.print("Enter employee id to be searched.. ");
                      int eid=sc.nextInt();
                      if (map1.containsKey(eid))
                              System.out.println("yes! " + eid + " is present.");
                      else
                              System.out.println("no! " + eid + " is not there.");
                      break;
            case 5:   System.out.print("Enter employee id whose name is required.. ");
                      eid=sc.nextInt();
                      if (map1.containsKey(eid))
                      {
                              System.out.println("yes! " + eid + " is present.");
                              String val=(String)map1.get(eid);
                              System.out.println("Name for " + eid + " is " + val);
                      }
                      else
                      System.out.println("no! " + eid + " is not there.");
                      break;
            case 6:   System.out.println("Names of all employees: "+ map1.values());
                      break;
            case 7:   Set set = map1.entrySet();
                      System.out.println("Employee ID with names: " + set);
                      break;
             case 8:  System.out.println("Byeeee....");
                      break;
            default:  System.out.println("Invalid choice");
                      break;
          }
        }
        while(choice!=8);
    }
}
```

Output

```
Main Menu
1: Size of Map
2: Remove all elements of Map
3: Check if hashmap is empty?
4: Search for an employee
5: Input Employee ID and display name of that employee
6: Display names of all employees
7: Display employee id with their names
8: Exit
Enter choice:
1
Size of the hash map: 4 records
Main Menu
1: Size of Map
2: Remove all elements of Map
3: Check if hashmap is empty?
4: Search for an employee
5: Input Employee ID and display name of that employee
6: Display names of all employees
7: Display employee id with their names
8: Exit
```

```
Enter choice:
3
It is false that hash map is empty
Main Menu
1: Size of Map
2: Remove all elements of Map
3: Check if hashmap is empty?
4: Search for an employee
5: Input Employee ID and display name of that employee
6: Display names of all employees
7: Display employee id with their names
8: Exit
Enter choice:
5
Enter employee id whose name is required.. 6
no! 6 is not there.
Main Menu
1: Size of Map
2: Remove all elements of Map
3: Check if hashmap is empty?
4: Search for an employee
5: Input Employee ID and display name of that employee
```

```
Enter choice:
6
Names of all employees: [Ramesh, Geeta, Vinod, Balram]
Main Menu
1: Size of Map
2: Remove all elements of Map
3: Check if hashmap is empty?
4: Search for an employee
5: Input Employee ID and display name of that employee
6: Display names of all employees
7: Display employee id with their names
8: Exit
Enter choice:
8
Byeeee....
```

☞ Check Your Progress-3

1) Differentiate between Set and Map.

……

……

……

……

2) Why do we use Map interface? What are the main classes implementing Map interface?

……

……

……

……

3) What is the difference between HashSet and HashMap?

……

……

……

……

4) Is hashmap an ordered collection.

a) True

b) False

5) Write down the output produced by the following program

```
import java.util.*;
public class test2
{
   public static void main(String[] args)
   {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("Ram", 20);
        map.put("Shyam", 25);
        map.put("Harish", 22);
        System.out.println("Size of map is:- "+ map.size());
        System.out.println(map);
        if (map.containsKey("Shyam"))
        {
           Integer a = map.get("Shyam");
           System.out.println("value for key"+ " \"Shyam\" is:- "+ a);
        }
   }
}
```

4.6 LIST INTERFACE

List interface extends the Collection interface, and hence it comprises all the methods available in Collection interface. It is used to manage the ordered collection of objects. Duplicate data is also allowed in list. Elements can be accessed by referring to their position in the list, with zero-based index. Various methods defined by list interface are mentioned in Table 6. The list interface is implemented by Vector, ArrayList and LinkedList classes. The object of list can be instantiated by any of the following approaches: -

List <T> aList = new ArrayList<> ();
List <T> lList = new LinkedList<> ();
List <T> v = new Vector<> ();
Where T is a generic type parameter passed to the generic interface List.

Table 6: Description of methods of List Interface

Sr.No.	Method	Description
1	void add (int idx, Object ob)	Adds ob into the invoking list at the index mentioned in the idx. None of the elements is lost by overwriting because any pre-existing element at or after the position of insertion are shifted up.
2	boolean addAll (int idx, Collection col)	Adds all elements of col in the list at the index passed in the idx. None of the elements is lost by overwriting because any pre-existing element at or after the point of insertion are moved up. This method gives true if the invoking list modifies otherwise, it gives false.
3	Object get(int idx)	It gives the object available at the mentioned idx of the invoking list.
4	int indexOf(Object ob)	It gives the index of the first occurrence of ob in the invoking list. If ob is not present in the list, it returns -1.
5	int lastIndexOf(Object ob)	It gives the index of the last occurrence of ob in the invoking list. If ob is not found in the list, then it returns -1
6	ListIterator lIterator( )	It gives an iterator to the beginning of the invoking list.
7	ListIterator Iterator (int index)	It gives an iterator that begins at the specified index of the invoking list.
8	Object remove(int idx)	It deletes the element at position idx from the invoking list and gives that deleted element. This makes the resulting list compact, and hence the total number of elements is decreased by one.
9	Object set (int idx, Object ob)	Allocates ob to the location specified by idx within the list that invoked this method.
10	List subList (int begin, int end)	It gives a list that includes elements from begin to (end-1) from the invoking list.

4.7 VECTOR CLASS

The Vector class extends AbstractList class. The List interface is implemented by this class. Vector implements a dynamic array that can grow or shrink as per the requirement. The elements of vector can be retrieved with an integer index. Vector is quite alike ArrayList, but it is synchronized i.e. it is thread-safe. The Vector class should be used in the thread-safe implementation only, and if you don't require synchronous access then ArrayList should be used as it will perform better.

Given below are the Constructors for Vector class:
Vector(): This constructs a default vector with the original capacity to store ten elements.
Vector(int s): A vector is constructed that has initial capacity equal to argument s.
Vector(int s, int increm): This constructor initializes both the size and increment of vector by using its arguments. The meaning of increm is the number of elements to assign each time when a vector is resized upward. (If the argument increm is not mentioned then vector's capacity will be doubled in each allocation round).
Vector(Collection col): This particular constructor initializes the vector with the elements of collection col.
Vector defines the following data members that have protected visibility mode:
int capacityIncrement: Stores the value of the increment.
int elementCount: Contains the count of elements presently in vector.
Object elementData[]: An array that stores the elements of the vector.

Vector class has all the methods inherited from its super classes, and it also defines additional methods explained in Table 7.

Table 7: Description of methods of Vector class

S.No.	Method	Description
1	void add (int idx, Object obj)	Adds the specified object at the mentioned idx in the Vector. This method is not synchronous.
2	boolean add (Object obj)	Returns true after appending the mentioned object to the end of the Vector.
3	boolean addAll(Collection col)	Appends all elements of the mentioned Collection to the end of the Vector, in the same sequence as they are returned by the iterator of the mentioned Collection. Returns true after appending.
4	boolean addAll (int idx, Collection c)	Inserts each and every element of the mentioned Collection into the Vector at the mentioned index. Returns true after inserting element.
5	void addElement (Object ob)	Adds the mentioned object to the end of the invoking vector, and increases its size by one.

		addElement() is synchronized.
6	int capacity()	The current capacity of the invoking vector is returned.
7	void clear()	All elements from the vector are removed.
8	Object clone()	A clone of the vector is returned.
9	boolean contains(Object obj)	Returns true if the mentioned object is a member in the invoking vector.
10	boolean containsAll(Collection c)	If the invoking vector contains all the elements of the mentioned Collection, then this method returns true.
11	void copyInto (Object[] anArray)	All the components of the invoking vector are copied into the mentioned array.
12	Object elementAt(int idx)	The component at the mentioned index is returned.
13	Enumeration elements()	An enumeration of all components of the invoking vector is returned.
14	void ensureCapacity (int minCapacity)	This method increases the capacity of the invoking vector, if required, so that it can store no less than the number of components mentioned by minCapacity.
15	boolean equals (Object ob)	It the invoking vector is equal to Object, then true is returned.
16	Object firstElement()	It gives the first element (the object at index 0) of the invoking vector.
17	Object get(int idx)	It gives the element at the mentioned index of the vector.
18	int hashCode()	The hash code for the invoking vector is returned.
19	int indexOf(Object obj)	The position of first occurrence of passed object is searched and returned
20	int indexOf (Object obj, int idx)	The first occurrence of passed object is searched but the search begins at idx.
21	void insertElementAt (Object ob, int idx)	The passed object is inserted (as a component) in the vector at the mentioned index.
22	boolean isEmpty()	Returns true if the vector does not have any components.
23	Object lastElement()	The last element of the invoking vector is

		returned.
24	int lastIndexOf (Object obj)	The index of the last occurrence of object obj, in the vector is returned.
25	int lastIndexOf (Object obj, int index)	Searches towards the back for passed object, beginning at the mentioned index, and gives the position of it.
26	Object remove(int idx)	Deletes and returns the element at the mentioned index of the vector.
27	boolean remove (Object obj)	Deletes the first occurrence of the mentioned object in the invoking vector and returns true. If the element is not found in the vector, it returns false.
28	boolean removeAll (Collection c)	Eliminates all elements of the specified collection from the vector and returns true. The method returns false, if the collection is not found.
29	void removeAllElements()	All elements of the vector are removed and its size is set to zero.
30	boolean removeElement (Object obj)	The first occurrence of the passed object is removed from the vector.
31	void removeElementAt (int idx)	The element at the mentioned index is removed from the vector.
32	protected void removeRange (int fmIndex, int toIndex)	All elements whose index lies between fmIndex, inclusive and toIndex, exclusive, are removed from the vector.
33	boolean retainAll(Collection c)	Only the elements that are contained in the mentioned Collection are retained in the invoking vector.
34	Object set(int index, Object element)	Element at the specified index in the invoking vector is replaced with the mentioned element.
35	void setElementAt(Object obj, int idx)	The object at the specified index (idx) of the vector is set to the object obj.
36	void setSize(int newSize)	The size of the vector is set to the newSize.
37	int size()	It gives the number of elements in the invoking vector.
38	List subList (int fromIndex, int toIndex)	A view of the slice of the List between fromIndex, inclusive, and toIndex, exclusive is returned.

39	Object[] toArray()	An array containing all the components of the vector in the exact order is returned.
40	Object[] toArray(Object[] a)	An array comprising all of the elements in this vector in the correct order is returned. The runtime type of the returned array is same as that of the specified array.
41	String toString()	A string representation of the vector is returned.
42	void trimToSize()	The capacity of the vector is trimmed to its current size.

You can instantiate an object of HashMap class in one of the following ways.

1. Non-Generic way (Old Style)
   Vector v=new Vector(); or List v=new Vector();
   Anything involving HashMap or Map without a datatype argument (the angle brackets < and > and the part between them) is a raw type that is an older approach and shouldn't be used. A raw type is not generic and lets you do hazardous things.

2. Generic way (New Style)
   Vector<T> v=new Vector<T>();
   T − The generic type parameter passed to the generic interface List and its implementation class Vector.
   Generic declaration allows compiler to check the 'v' usages during compile-time and it is very safe.
   We will be using the Generic way in the example given below: -

We will be using the Generic way in the example given below: -

```
import java.util.*;
public class VectorEx
{
   public static void main(String args[])
   {
      //Create an empty vector
      Vector<String> vec1 = new Vector<String>();
      //Add elements to the vector
      vec1.add("Television");
      vec1.add("Refrigerator");
      vec1.add("Washing Machine");
      //display size and capacity
      System.out.println("Size is: "+vec1.size());
      System.out.println("Default capacity is: "+vec1.capacity());
      //Display Vector elements
      System.out.println("Vector element is: "+vec1);
      //Add two more elements
      vec1.add("Microwave Oven");
      vec1.add("Mobile Phone");
      //Again display size and capacity after two insertions
      System.out.println("Size after addition: "+vec1.size());
      System.out.println("Capacity after addition is: "+vec1.capacity());
      //Display Vector elements again
      System.out.println("Elements are: "+vec1);
      //Checking if Television is present or not in this vector
```

```
    if(vec1.contains("Television"))
    {
      System.out.println("Television is present at the index " +vec1.indexOf("Television"));
    }
    else
    {
      System.out.println("Television is not present in the list.");
    }
    //Print the first element
    System.out.println("The first element of the vector is = "+vec1.firstElement());
    //Print the last element
    System.out.println("The last element of the vector is = "+vec1.lastElement());
    //Add Refrigerator again
    vec1.add("Refrigerator");
    //Display Vector elements again
    System.out.println("Elements are: "+vec1);
    //use remove() method to delete the first occurence of an element
    System.out.println("Remove first occurrence of element Refrigerator:
    "+vec1.remove("Refrigerator"));
    //Display the vector elements after remove() method
    System.out.println("Values in vector: " +vec1);
    //Remove the element at index 4
    System.out.println("Remove element at index 3: " +vec1.remove(3));
    System.out.println("New Values in vector: " +vec1);
    //Get the hashcode for this vector
    System.out.println("Hash code of this vector = "+vec1.hashCode());
    //Get the element at specified index
    System.out.println("Element at index 1 is = "+vec1.get(1));
  }
}
```

Output

```
Size is: 3
Default capacity is: 10
Vector element is: [Television, Refrigerator, Washing Machine]
Size after addition: 5
Capacity after addition is: 10
Elements are: [Television, Refrigerator, Washing Machine, Microwave Oven, Mobile Phone]
Television is present at the index 0
The first element of the vector is = Television
The last element of the vector is = Mobile Phone
Elements are: [Television, Refrigerator, Washing Machine, Microwave Oven, Mobile Phone, Refrigerator]

Remove first occurrence of element Refrigerator: true
Values in vector: [Television, Washing Machine, Microwave Oven, Mobile Phone, Refrigerator]
Remove element at index 3: Mobile Phone
New Values in vector: [Television, Washing Machine, Microwave Oven, Refrigerator]
Hash code of this vector = 1730521510
Element at index 1 is = Washing Machine
```

4.8 STACK CLASS

The stack is a linear data structure that extends the Vector class. It is based on Last-In-First-Out (LIFO). Only one constructor is defined by the Stack class, which creates an empty stack.

Stack<T> stk = new Stack<T>();

T − The generic type parameter passed to the Stack.

All methods defined by Vector class are available in Stack in addition to few of its own methods. These added methods are given in Table 8.

Table 8: Description of methods of Stack class

S.No.	Method	Description
1	boolean empty( )	If stack is empty, true is returned otherwise false is returned.
2	Object peek( )	The element at the top of the stack is returned without removing it from stack.
3	Object pop( )	The element at the top of the stack is returned as well as removed from stack.
4	Object push(Object element)	The element is inserted onto the stack.
5	int search(Object element)	The mentioned element is searched in the stack. If found, its offset from the top of the stack is returned otherwise, -1 is returned.

Given program demonstrates some of the operations on stack.

```
import java.util.Stack;
import java.util.Scanner;
public class StackExample
{
   public static void main(String[] args)
   {
      int j;
      Scanner in = new Scanner(System.in);
      //creating an instance of Stack class
      Stack<Integer> stk= new Stack<>();
      // checking stack is empty or not
      boolean result = stk.empty();
      System.out.println("It is "+result+" stack is empty");
      // pushing five elements into stack
      for (int i=1;i<6;i++)
      {
         System.out.println("Enter element number "+i+" to be pushed");
         j = in.nextInt();
         stk.push(j);
      }
      System.out.println("Elements in Stack: " + stk);
      System.out.println("Enter the element to be searched...");
      j = in.nextInt();
      Integer pos = (Integer) stk.search(j);
      if(pos == -1)
      System.out.println("Element not found");
      else
      System.out.println("Element is found at position: " + pos);
   }
}
```

Output

```
It is true stack is empty
Enter element number 1 to be pushed
56
Enter element number 2 to be pushed
43
Enter element number 3 to be pushed
98
Enter element number 4 to be pushed
73
Enter element number 5 to be pushed
122
Elements in Stack: [56, 43, 98, 73, 122]
Enter the element to be searched...
73
Element is found at position: 2
```

☞ Check Your Progress-4

1) Write down the differences and similarities between List and Set?

……

……

……

……

2) Why do we use the List interface? Name the classes that implement the List interface.

……

……

……

……

3) What is the output of the following program?

```
import java.util.*;
public class test3
{
   public static void main(String[] args)
   {
      List<Integer> list1 = new ArrayList<Integer>();
      list1.add(0, 1);
      list1.add(1, 2);
      System.out.println(list1);
      List<Integer> list2 = new ArrayList<Integer>();
      list2.add(1);
      list2.add(2);
      list2.add(3);
      list1.addAll(1, list2);
      System.out.println(list1);
      list1.remove(1);
      System.out.println(list1);
      System.out.println(list1.get(3));
      list1.set(0, 5);
```

```
        System.out.println(list1);
    }
}
```

4) Explain the Vector class and its importance.

……

……

……

……

5) What do you mean by the Stack class, and what are the various methods provided by it?

……

……

……

……

4.9 SUMMARY

This unit explains various classes and interfaces that are part of java.util package. We have deliberated the functions of date and time classes along with suitable program codes. This unit deals with List and Set that are important interfaces of collection framework. Different classes that implement these interfaces have been elaborated with various programs. We have discussed Map interface along with HashMap class that implements it. Overall, the facilities offered by Collection framework have been explained in detail.

4.10 SOLUTIONS/ANSWERS

☞ Check Your Progress-1

1) Collection Framework is a group of classes and interfaces to store and manipulate the group of objects. Various operations like searching, sorting, insertion and deletion can be performed on these objects. Various classes such as ArrayList, Vector, Stack, and HashSet, etc. are provided by the collection framework. Also, there are many interfaces such as List, Queue, Set, etc. available in collection framework.

2) Maps is not a part of the collection framework.

3) In order to check whether two date objects are equal, the following method is used.

int compareTo (Date date) : The value of the invoking object is compared with that of date object. If both are equal, the function returns 0. A negative

value is returned, if the invoking object is earlier than date. Or a positive value is returned, if the invoking object is later than the date.

☞ Check Your Progress-2

1) In order to pass over the elements of any collection, we can make use of Iterable interface, which is the root interface of the whole collection framework. The Iterable interface is extended by the collection interface. Hence, all the interfaces and classes of collection framework implement this interface. The most important feature of Iterable interface is to offer an iterator for the collections. Only one method, which is abstract and whose name is iterator(), is contained in this interface. It returns an object of type Iterator.

Iterator iterator();

2) java.util.HashSet class is a member of the Java collections framework, which implements the Set interface. Its purpose is to create a collection and store it in a hash table. Each element in Set has a unique value of its own, called hash code which is used as an index at which element associated with that hash code is stored. Hash code is obtained by converting the informational content into a unique value (known as hash code). Hashing ensures that the execution time of operations like add(), remove(), remain constant even for set of bigger size. The elements are inserted in HashSet irrespective of the order of insertion, and the order depends on the hash code of elements.

3) The HashSet and TreeSet, both classes, implement Set interface. The differences between the both are listed below: -

HashSet	TreeSet
It does not maintain any sequence of elements added to it.	It maintains ascending order of elements.
It is implemented by hash table.	It is implemented by a Tree structure.
Performance is better than TreeSet.	TreeSet performance is not as good as HashSet.
HashSet is backed by HashMap.	TreeSet is backed by TreeMap.

4) 0.75
5) null
One

☞ Check Your Progress-3

1) The differences between Set and Map are given below: -

- Set contains only the values, whereas Map contains key and values both.
- Set has unique values, whereas Map can contain unique Keys with duplicate values (Keys are unique but two or more keys can have the same value.)

... Set can include just one null, whereas Map can include a single null key with n number of null values.

2) Map interface is a part of java.util package, but map is not a collection as it does not implement the collection interface. Map interface represents a mapping between a key and a value. A map contains unique keys though the values for two or more keys can be the same. At the most, one value can be mapped by each key. A Map is suitable if we have to search, update or delete elements on the basis of a key. The classes that implement Map interface are: HashMap, Hashtable, LinkedHashMap.

3) The differences between the HashSet and HashMap are listed below.

HashSet	HashMap
Set interface is implemented by Hashset.	Map interface is implemented by the HashMap.
It contains only values.	It contains a mapping between a key and its value.
It can be traversed.	HashMap needs to be converted into Set to be iterated.
It cannot have any duplicate value.	It can have duplicate values with unique keys.
It can hold just one null value.	It can contain a single null key with n number of null values.

4) False. Hashmap outputs in the order of hashcode of the keys. Though actually it is unordered but will always have the same result for the same set of keys.

5)

```
Size of map is:- 3
{Shyam=25, Harish=22, Ram=20}
value for key "Shyam" is:- 25
```

☞ Check Your Progress-4

1) The List and Set both extend the collection interface. However, there are some differences between the both, as mentioned below :-

List	Set
Duplicate elements are allowed in List.	Only unique elements are allowed in Set.
It is an ordered collection that maintains the order of insertion of elements.	It is an unordered collection that does not preserve the order of insertion.
The List interface can allow any number of null values.	The Set interface allows just a single null value.

2) The List interface is an ordered collection of elements. List preserves the insertion order and allows duplicate values to be stored. This interface comprises different methods that enable easy manipulation of elements based on the element index. The main classes that implement List interface are ArrayList, LinkedList, Stack, and Vector.

3)

```
[1, 2]
[1, 1, 2, 3, 2]
[1, 2, 3, 2]
2
[5, 2, 3, 2]
```

3) An index is used to access the elements of a vector. Vector is just like a dynamic array. There is no specific size of a vector; it can shrink or grow automatically whenever required. It is quite similar to ArrayList, but there are two differences as given below: -

- Vector is synchronized.
- Many legacy methods are available in Vector class that are not part of the collections framework.

5) Last-in-first-out is the basic principle of Stack class. The elements are added as well as removed from the rear end (that is where the latest item was inserted or removed from). The action of adding an element to a stack is called push, while removing an element is referred to as pop. In order to retrieve or fetch the top element of the Stack, we can use peek() method. The element retrieved is only accessed, not deleted or removed from the Stack.

4.11 REFERENCE/FURTHER READING

- Herbert Schildt "Java The Complete Reference", McGraw-Hill,2017
- Horstmann, Cay S., and Gary Cornell, " *Core Java: Advanced Features*" ,Vol.2, Pearson Education, 2013.
- E Balagurusamy , "Programming with Java:, McGraw-Hill Education,2019.
- Benjamin J. Evans, David Flanagan , "Java in a Nutshell: A Desktop Quick Reference", O'Reilly,2019.
- https://www.javatpoint.com/collections-in-java
- https://www.tutorialspoint.com/java/java_collections.htm
- https://www.geeksforgeeks.org/collections-in-java-2/
- https://www.programiz.com/java-programming/collections
- https://en.wikipedia.org/wiki/Java_collections_framework
- https://dzone.com/articles/an-introduction-to-the-java-collections-framework