COLLECTION FREAMEWORK

Collections in Java is a framework that stores and manipulates a group of objects. It is a hierarchy of interfaces and classes that provides easy management of a group of objects. Java Collection framework provides many interfaces (**List, Queue, Deque, Set**) and classes (**ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet**).

❖ framework in Java

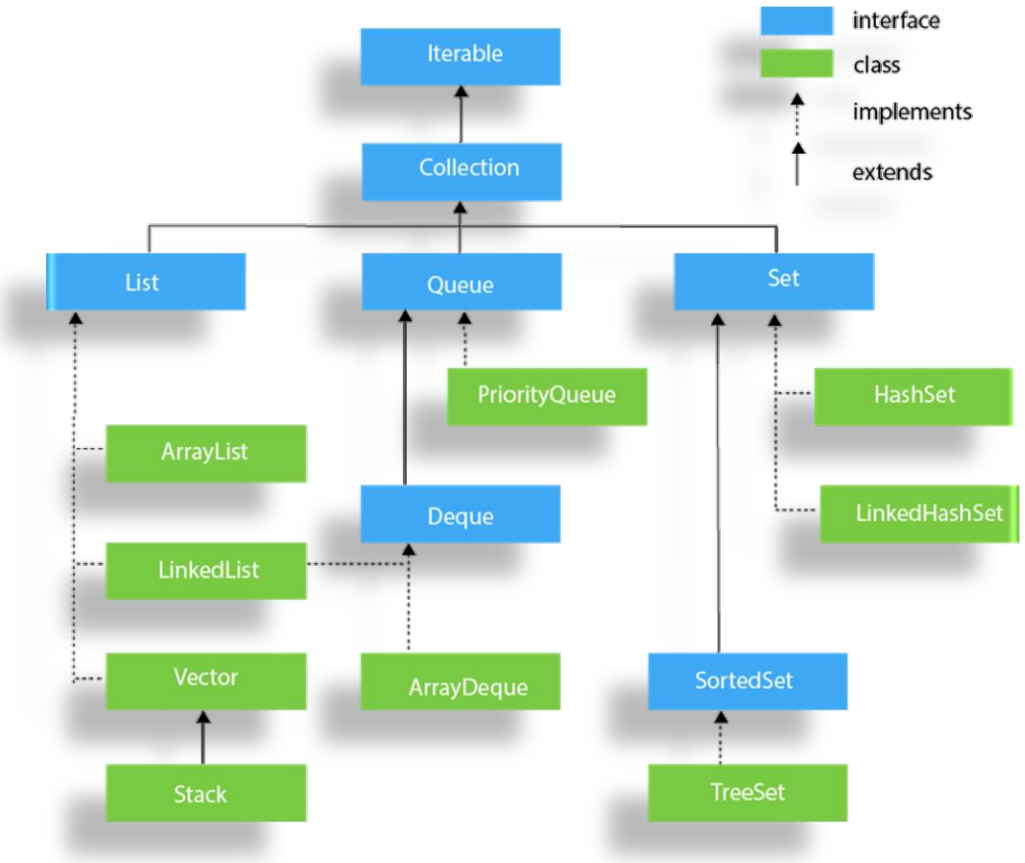
- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

❖ Collections Framework

- The **Collections Framework** is defined as a unified architecture for representing and manipulating collections.
- In Java, the Collections Framework is a hierarchy of interfaces and classes that provides easy management of a group of objects.
- The **java.util** package contains the powerful tool of Collections Framework.
- It was defined in **JDK 1.2** version which is one of the most used frameworks to date.
- It provides a ready-made architecture for interfaces and classes and is used for storing and manipulating a group of objects.
- All collections frameworks contain interfaces, classes, and algorithms.

❖ Hierarchy of Collection Framework:

*Let us see the **hierarchy of Collection framework**. The java.util package contains all the classes and interfaces for the Collection framework.*



The ***above figure*** illustrates the hierarchy of the Collections Framework. It consists of four core interfaces such as ***Collection, List, Set, Queue***, and ***various classes*** which get implemented through them.

Methods of Collection interface:

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	Public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	Default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T> T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.
15	default Stream<E> parallelStream()	It returns a possibly parallel Stream with the collection as its source.
16	default Stream<E> stream()	It returns a sequential Stream with the collection as its source.

17	default Spliterator<E> spliterator()	It generates a Spliterator over the specified elements in the collection.
18	public boolean equals(Object element)	It matches two collections.
19	public int hashCode()	It returns the hash code number of the collection.

❖ Iterator interface

- Iterator interface provides the facility of iterating the elements in a forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

❖ Iterable Interface

- The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.
- It contains only one abstract method. i.e.,

Iterator<T> iterator()

- It returns the iterator over the elements of type **T.**

❖ Collection Interface

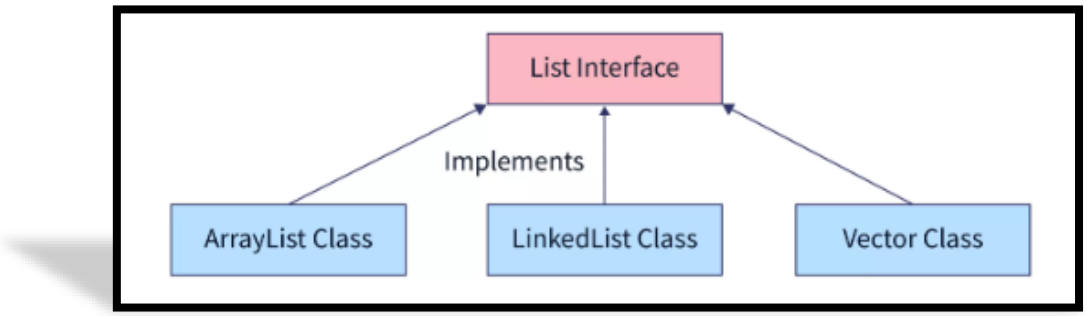
- The Collection Interface is the root or the foundation on which the Collections Framework is built. It is a general interface that has the declaration:

interface Collection<E>

Here, **E** is the type of object that the collection will hold.

❖ List Interface

- The ***list interface*** extends the collection interface.
- A list is used to store ordered collection of data and it may contain duplicates. Ordered collection means the order in which the elements are being inserted and they contain a specific value.
- The elements present, can be accessed or inserted by their position in the list using zero-based indexing.
- The list interface is implemented by LinkedList, ArrayList, Vectors and Stack classes. They are an important part of collections in Java.



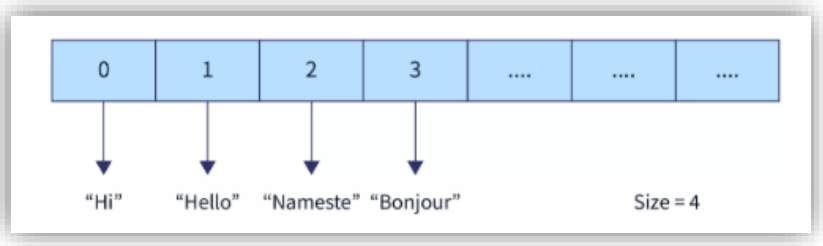
There are **three classes** implemented by the list interface and they are given below.

Syntax:

```
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

1.ArrayList

- So, the ArrayList is similar to Arrays.
- They are also called dynamic arrays.
- That means it does not have a fixed size.
- ts size can be increased or decreased if elements are added or removed.
- It implements the List Interface.
- It is similar to Vectors in C++.
- Since the ArrayList cannot be used for primitive data types like int, char, etc. , we need to use a wrapper class.



In order to initialize an ArrayList, there are 3 ways.

1. **Using the add Keyword:**

```
ArrayList<Data-Type> str = new ArrayList<Data-Type>();
   str.add(<content>);
   str.add(<content>);
   str.add(<content>);
```

Example:

```
//initialising ArrayList using add keyword
import java.util.*;
public class TopperWorld {
   public static void main(String args[])  {
      // create an ArrayList of String type
      ArrayList<String> str = new ArrayList<String>();
      // Initialize an ArrayList with add()
      str.add("Topper");
      str.add("World");
      str.add("Rocks");
  // to print the ArrayList
      System.out.println("ArrayList is" + str);
   }
}
```

Output:

```
ArrayList is [Topper , World, Rocks]
```

2. Using **asList() AsList()** method in Java is used to return a fixed-size list backed by the given array.

```
ArrayList<Data-Type> obj =
    new ArrayList<Data-Type(Arrays.asList(Obj A, Obj B, Obj C, ....));
```

Example:

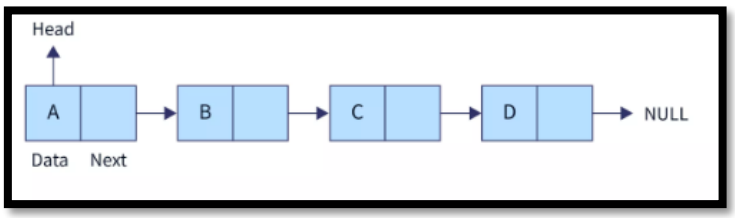
```
// initialise ArrayList using asList method
import java.util.*;
public class TW {
   public static void main(String args[])
   {
      // create an ArrayList of String type
      // and Initialize an ArrayList with asList()
      ArrayList<String> scaler = new ArrayList<String>(
         Arrays.asList("I", "love", "TopperWorld"))
      // to print the ArrayList
      System.out.println("ArrayList is " + Topperworld);
   }
}
```

Output:

```
ArrayList is [I, Love, Java]
```

2. LinkedList

- The **LinkedList** class extends the AbstractSequentialList and it also extends the List, Deque and Queue interface.
- By this, we get a linked-list data structure.
- Linked List is a linear data structure where the elements are called as nodes.
- Here, each node has two fields- data and next. Data stores the actual piece of information and next points to the next node. 'Next' field is actually the address of the next node.
- Elements are not stored in a contiguous memory, so direct access to that element is not possible.
- LinkedList uses Doubly Linked List to store its elements while ArrayList internally uses a dynamic array to store its elements.
- LinkedList is faster in the manipulation of data as it is node-based which makes it unique.



How to create a Linked List?

There are **two ways** in which we can create a linked list using constructors.

1. **Creating an empty Linked List**

```
LinkedList list=new LinkedList();
```

2. **Creating a Linked List from Collection**

It will create a Linked List with all the elements of Collection C. We will use the LinkedList class and new keyword to create a constructor which contains the elements of collection.

```
LinkedList list=new LinkedList(C);
```

Let us take a look at an example of Linked List.

```
public class TopperWorld{
   public static void main(String args[]){
   //creating a LinkedList
   LinkedList<String> list= new LinkedList<String>();
   //displaying the initial size
   System.out.println("Size at the beginning "+list.size());
   //add elements
   list.add("Java");
   list.add("C++");
   list.add("JavaScript");
   list.addFirst("C#");
   list.addLast("Kotlin");
   list.add(2,"Python");
   //displaying the LinkedList
   System.out.println("Original Linked List " + list);
```

```
//displaying the size
  System.out.println("Size after addition "+list.size());
  //remove element at index 5
  list.remove(5);
  list.remove("C#");
  //display the new LinkedList
  System.out.println("New Linked List "+ list)
  //display the new size
  System.out.println("Size after removal "+list.size());
  }
  }
```

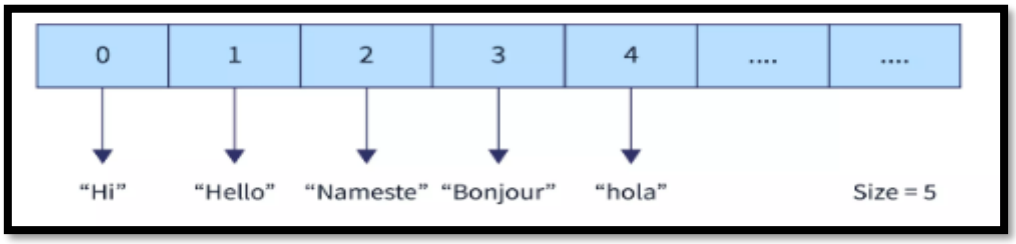
Output:

```
Size at the beginning 0
Original Linked List [C#, Java, Python, C++, JavaScript, Kotlin]
Size after addition 6
New Linked List [Java, Python, C++, JavaScript]
Size after removal 4
```

Vector

- Like ArrayList, Vectors in Java are used for **dynamic arrays.**
- It extends the AbstractList and implements the List interface.
- Vector is **synchronised**. Synchronised means only one thread at a time can access the code.
- This means if one thread is working on Vector, no other thread can get a hold of it.

- Only one operation on vector can be performed at a time. **For example**, if addition is being performed by one thread, other operation cannot be performed until the first one is over.
- Before the introduction of the Collection Framework, Vectors were categorised as **Legacy Classes**(Classes which were a part of the earlier release of Java but now they are re-constructed).



Methods to create the constructor of Vectors

1. **Using the default method**

In this method, the size is not specified so the default size of Vectors is 10.

```
Vector<Data-type> v = new Vector<Data-Type>();
```

2. **By specifying the desired size**

In this method, we specify the size.

```
Vector<Data-type> v = new Vector<Data-Type>(int size);
```

3. **Using size and increment attributes**

This method is used to create a vector whose initial capacity is declared by size and the increment is declared by incr. Here, the increment is the value that specifies the number of elements to allocate each time that vector gets resized upward.

Vector<Data-type> v = new Vector<Data-Type>(int size,int incr);

4. **Creating a Vector from Collection**

It is used to create a vector which contains all the elements of a collection.

Vector<Data-type> v = new Vector<Data-Type>(Collection C);

Example:

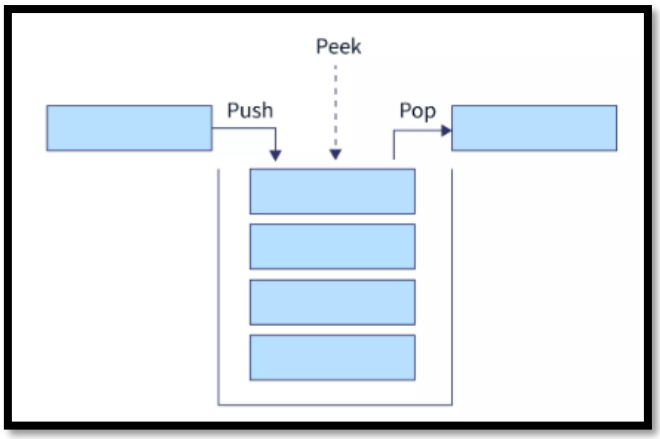
```
import java.util.*;
   public class TopperWorld{
   public static void main(String args[]){
   //creating a Vector
   Vector<Integer> v= new Vector<Integer>();
   //displaying the size
   System.out.println("Size at the beginning "+v.size());
    //add elements
   v.add(19);
   v.add(88);
   v.add(1);
   v.add(39);
    //displaying the Vector
   System.out.println(v);
//displaying the size
   System.out.println("Size after addition "+v.size());
   //remove element at index 3
   v.remove(3);
   //display the new Vector
   System.out.println(v);
//display the new size
   System.out.println("Size after removal "+v.size());
   }
   }
```

Output:

```
Size at the beginning 0
[19, 88, 1, 39]
Size after addition 4
[19, 88, 1]
```

4. Stack

- **Stack** class extends the Vector class and it is its subclass.
- It works on the principle of Last-In, First-Out.
- In order to put an object on the top of the stack, we call the **push() method.**
- To remove and return the top element in the stack, we call **pop() method.**
- There are other methods like **peek(), search() and empty()** which are used to perform operations on the stack.



Methods to create the constructor of Stack:

1. **Creating a default stack**

This creates an empty stack.

```
Stack<Data-Type> stack = new Stack<Data-Type>();
```

Example:

```
import java.util.*;
public class TopperWorld{
   public static void main(String args[]){
   //creating a Stack
   Stack<Integer> s= new Stack<Integer>();
   //displaying the initial size
   System.out.println("Size at the beginning "+s.size());
   //push elements
   s.push(99);
   s.push(28);
   s.push(17);
   s.push(74);
   s.push(1);
   //displaying the Stack
   System.out.println("New Stack" + s);
   //displaying the size
   System.out.println("Size after addition "+s.size());
   //pop the element and display it
   System.out.println("Popped element " + s.pop());
   //display the new Stack
   System.out.println("New Stack after popping"+ s);
   //display the new size
   System.out.println("Size after removal "+s.size());
```

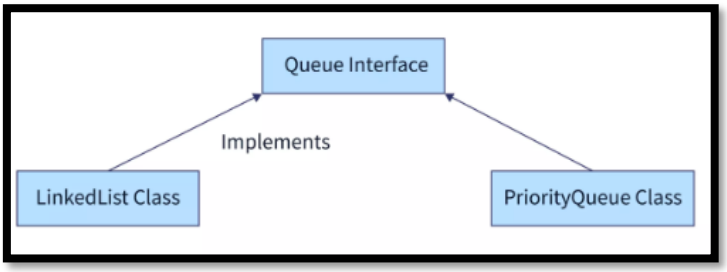
```
//peek method to find the top-most element and display it
   System.out.println("Top-most element " + s.peek());
   //the size remains the same as peek does not remove the element
   System.out.println("Size after Peek "+s.size());
   }
   }
```

Output:

```
Size at the beginning 0
New Stack[99, 28, 17, 74, 1]
Size after addition 5
Popped element 1
New Stack after popping[99, 28, 17, 74]
Size after removal 4
Top-most element 74
Size after Peek 4
```

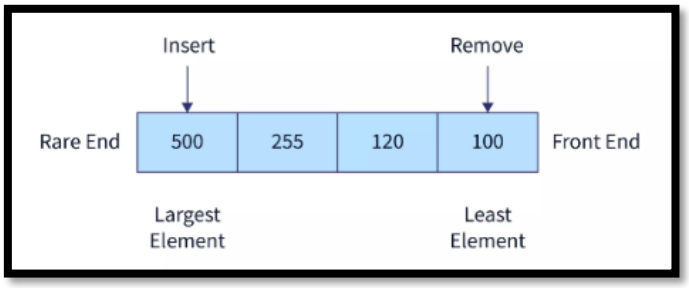
Queue Interface

- The **Queue Interface** extends the Collection interface.
- It uses the principle of **First-In, First-Out (FIFO).**
- A Queue is an ordered list where there is a need to maintain the order of the elements.
- It has classes like PriorityQueue and ArrayDeque.
- The most famous implementation is that of PriorityQueue.



PriorityQueue:

- The **PriorityQueue** class extends AbstractQueue and implements the Queue Interface.
- As the name suggests, they follow the principle of priority of the elements.
- We know that we follow First-In, First-Out for queues, but at times, the elements need to be processed in terms of their priority. This is where the PriorityQueue comes into play.
- It does not allow null values to be stored inside it.
- The add() method is used to add an element while the poll() method is used to remove the top-most element. While, peek() is used to display the top-most element.



Methods to initialise a PriorityQueue using constructors

1. **Creating an empty PriorityQueue**

```
PriorityQueue<Data-Type> pq = new PriorityQueue<Data-Type>();
```

2. **Creating a PriorityQueue with the specified size**

```
PriorityQueue<Data-Type> pq = new PriorityQueue<Data-Type>(int size);
```

3. **Creating a PriorityQueue from Collection**

It will create a PriorityQueue with all the elements of Collection C.

PriorityQueue<Data-Type> pq = new PriorityQueue<Data-Type>(Collection<Data-Type> C);

Example:

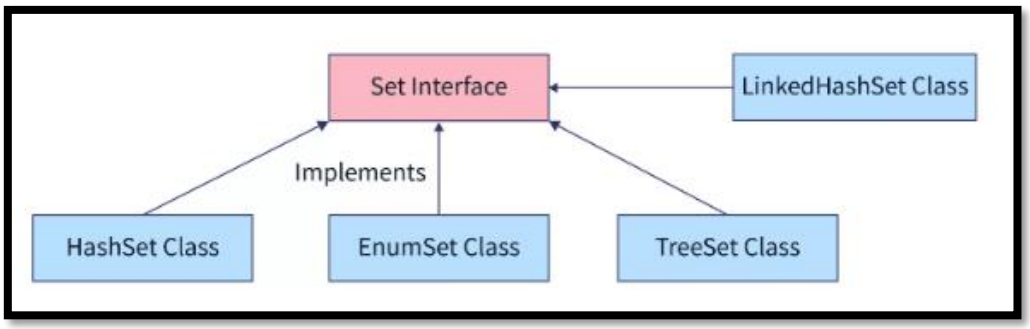
```
import java.util.*;
public class TopperWorld{
   public static void main(String args[])
   {
   // Creating a priority queue
   PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    //displaying the initial size
    System.out.println("Size at the beginning "+pq.size());
      // Adding elements using add()
      pq.add(99);
      pq.add(18);
      pq.add(27);
      pq.add(34);
      //displaying the PriorityQueue
      System.out.println("New PriorityQueue" + pq);
     //displaying the size
      System.out.println("Size after addition "+pq.size());
       // Printing the top element of the PriorityQueue
      System.out.println("Top-most element " +pq.peek());
      // Printing the top element and removing it
      System.out.println("Removing " +pq.poll());
//displaying the PriorityQueue
      System.out.println("New PriorityQueue after removal" + pq);
      //display the new size
      System.out.println("Size after removal "+pq.size()); }}
```

Output:

```
Size at the beginning 0
New PriorityQueue[18, 34, 27, 99]
Size after addition 4
Top-most element 18
Removing 18
New PriorityQueue after removal[27, 34, 99]
Size after removal 3
```

❖ Set Interface

- The Set interface defines an unordered collection.
- It extends the Collection Interface.
- We cannot store duplicate values in this.
- The Set Interface is implemented by popular classes like HashedSet, LinkedHashSet, and TreeSet.



Method to instantiate the Set Interface:

```
Set<Data-Type> hs = new HashSet<Data-Type>();
Set<Data-Type> lhs = new LinkedHashSet<Data-Type>();
Set<Data-Type> ts = new TreeSet<Data-Type>();
```

1.HashSet

- ✓ The **HashSet** class implements the Set Interface.
- ✓ It uses a hash table for storage which uses a mechanism called Hashing.
- ✓ In hashing, the informational content of a key determines a unique value, called its hash code.
- ✓ The hash code is then used as an index, at which the data associated with the key is stored.
- ✓ When we insert elements into the HashSet, it is not guaranteed that it gets stored in the same order.
- ✓ We can store Null values in this.
- ✓ HashSet is non-synchronized means multiple threads at a time can access the code. This means if one thread is working on HashSet, other threads can also get a hold of it. Multiple operations on HashSet can be performed at a time.

For example, if addition is being performed by one thread, other operation can be performed by some other thread too.

Methods to create the constructors of **HashSet**

1. **Creating an empty HashSet It is used to create an empty HashSet object in which the default initial capacity is 16.**

```
HashSet<Data-type> hs = new HashSet<Data-type>();
```

2. **Creating a HashSet with a specified size It is used to create a HashSet with the given size.**

```
HashSet<Data-type> hs = new HashSet<Data-type>(int size);
```

3. **Creating a HashSet with a specified size and fill ratio It is used to create a HashSet with a given size and fill ratio.**

```
HashSet<Data-type> hs = new HashSet<Data-type>(int size,float fillRatio);
```

4. **Creating a HashSet from Collection It is used to create a HashSet which contains all the elements from the collection.**

```
HashSet<Data-type> hs = new HashSet<Data-type>(Collection C);
```

Example:

```
import java.util.*;
public class TopperWorld{
   public static void main(String args[]){
   //creating a HashSet
   HashSet<String> str= new HashSet<String>();
   //displaying the initial size
   System.out.println("Size at the beginning "+str.size());
   //add elements
   str.add("Hello");
   str.add("Hi");
   str.add("Namaste");
   str.add("Bonjour");
   //displaying the HashSet
   System.out.println(str);
   //displaying the size
   System.out.println("Size after addition "+str.size());
   //remove element using value
   str.remove("Bonjour");
```

```
//display the new HashSet
  System.out.println(str);
  //display the new size
  System.out.println("Size after removal "+str.size());
  } }
```

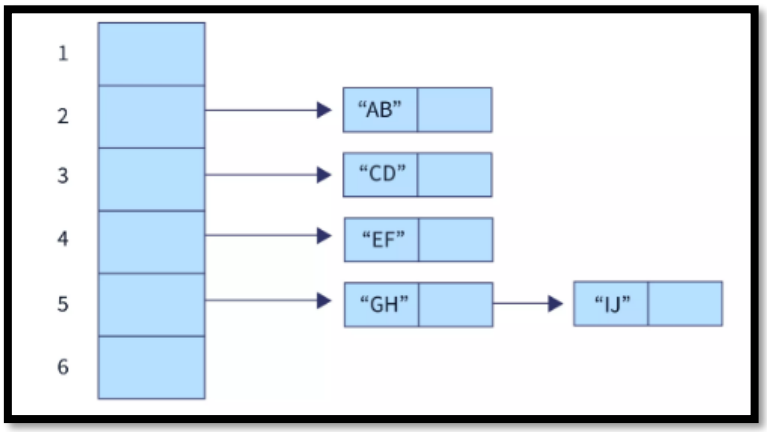
Output:

```
Size at the beginning 0
[Hi, Hello, Namaste, Bonjour]
Size after addition 4
[Hi, Hello, Namaste]
Size after removal 3
```

2.LinkedHashSet

- The LinkedHashSet class extends the HashSet class.
- It maintains a linked list of entries in the set and hence maintains the order in which they were inserted.
- LinkedHashSet is non-synchronized means multiple threads at a time can access the code.
- This means if one thread is working on LinkedHashSet, other threads can also get a hold of it.
- Multiple operations on LinkedHashSet can be performed at a time. For example, if addition is being performed by one thread, other operation can be performed by some other thread too.

The figure below shows the illustration of a **LinkedHashSet**



Methods to create the constructors of **HashSet**

1. **Creating an empty LinkedHashSet It is used to create an empty LinkedHashSet object.**

```
LinkedHashSet<Data-type> lhs = new LinkedHashSet<Data-type>();
```

2. **Creating a LinkedHashSet with a specified size It is used to create a LinkedHashSet with the given size.**

```
LinkedHashSet<Data-type> lhs = new LinkedHashSet<Data-type>(int size);
```

3. **Creating a LinkedHashSet with a specified size and fill ratio It is used to create a LinkedHashSet with a given size and fill ratio. Fill ratio determines how full the hash set can be before it is resized.**

```
LinkedHashSet<Data-type> lhs = new LinkedHashSet<Data-type>(int size,float fillRatio);
```

4. **Creating a LinkedHashSet from Collection It is used to create a LinkedHashSet which contains all the elements from the collection**.

LinkedHashSet<Data-type> lhs = new LinkedHashSet<Data-type>(Collection C);

Example:

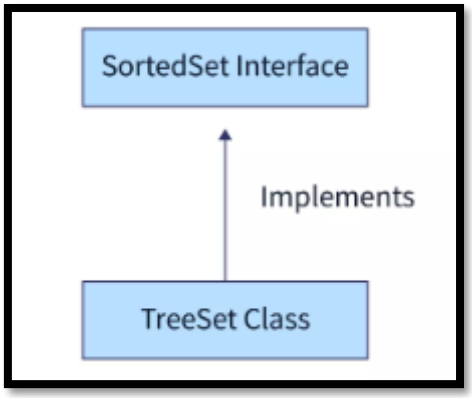
```
import java.util.*;
   public class TopperWorld{
   public static void main(String args[]){
   //creating a HashSet
   LinkedHashSet<String> str= new LinkedHashSet<String>();
   //displaying the initial size
   System.out.println("Size at the beginning "+str.size());
   //add elements
   str.add("Hello");
   str.add("Hi");
   str.add("Namaste");
   str.add("Bonjour");
   //displaying the LinkedHashSet
   System.out.println(str);
   //displaying the size
   System.out.println("Size after addition "+str.size());
   //remove element using value
   str.remove("Bonjour");
   //display the new LinkedHashSet
   System.out.println(str);
   //display the new size
   System.out.println("Size after removal "+str.size());
   }
   }
```

Output:

```
Size at the beginning 0
[Hello, Hi, Namaste, Bonjour]
Size after addition 4
[Hello, Hi, Namaste]
Size after removal 3
```

3.SortedSet Interface

- The SortedSet Interface extends the Set Interface.
- It is similar to the Set Interface but plays a vital role to arrange data in ascending or sorted manner.
- The TreeSet class implements this interface.
- The figure below illustrates the SortedSet Interface.



Method to instantiate the **SortedSet Interface**

```
SortedSet<Data-Type> ts = new TreeSet<Data-type>();
```

4.TreeSet

- The TreeSet class implements the Set Interface.
- It uses a tree to store the elements.
- TreeSet contains unique elements.

- The access and retrieval time is very fast.
- TreeSet is non-synchronized means multiple threads at a time can access the code.
- This means if one thread is working on TreeSet, other threads can also get a hold of it.
- Multiple operations on TreeSet can be performed at a time.
- For example, if addition is being performed by one thread, other operations can be performed by some other thread too.

Example:

```
import java.util.*;
 class TopperWorld {
 public static void main(String[] args)
 {
    // Creating a TreeSet
    Set<String> ts = new TreeSet<>();
   //displaying the initial size
   System.out.println("Size at the beginning "+ts.size());
    // Elements are added using add() method
    ts.add("India");
    ts.add("USA");
    ts.add("Britain");
    //displaying the TreeSet
    System.out.println(ts);
    //displaying the size
    System.out.println("Size after addition "+ts.size());
```

```
// Duplicates will not get inserted into the TreeSet
        ts.add("Britain");
        // Elements get stored in Ascending order
        System.out.println(ts);
    }}
```

Output:

```
Size at the beginning 0
[Britain, India, USA]
Size after addition 3
[Britain, India, USA]
```

❖ Map Interface

- A map is an object that stores key and value pairs.
- It contains unique keys as the same key cannot have multiple mappings.
- Although a part of the Collections Framework, maps are not themselves collections because they do not implement the Collection Interface.

Method to instantiate the Map Interface

```
Map<Data-Type> hm = new HashMap<> ();
```

❖ HashMap

- The HashMap class extends AbstractMap and implements the Map interface.
- It uses a hash table for storing key-value pairs.
- If we want to access a value in a** hash map**, we must know its key.

Example:

```
import java.util.*;
public class TopperWorld{
   public static void main(String args[]) {
      // Creating a HashMap
      HashMap<Integer, Double> hm = new HashMap<Integer, Double>();
  //adding key value pairs using put()
      hm.put(1, 1.9);
      hm.put(2, 2.8);
      hm.put(3, 3.7);
      // Finding the value for a key using get()
      System.out.println("The Value for 1 is " + hm.get(1)); }}
```

Output:

```
The Value for 1 is 1.9
```