

Unit 2: Collection Framework

2

Unit Structure

- 2.1 Learning Objectives
- 2.2 Introduction
- 2.3 Why Collection Framework?
- 2.4 Hierarchy in collection framework
- 2.5 Collection interface
- 2.6 Set interface
- 2.7 List interface
- 2.8 Queue interface
- 2.9 Deque interface
- 2.10 Iterator interface
- 2.11 Implementation of List
- 2.12 Implementation of Queue
- 2.13 Implementation of Set
- 2.14 Let us sum up
- 2.15 Check your Progress
- 2.16 Check your Progress: Possible Answers
- 2.17 Further Reading
- 2.18 Assignments

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand need of Collection framework in java.
- Understand the hierarchy of classes and interfaces of Collection framework.
- Study and understand the functionalities provided by various class and interfaces of collection framework.
- Study the example of utilization of various classes of collection framework like ArrayList, LinkedList, Stack, Vector, PriorityQueue, HashSet, TreeSet etc.

2.2 INTRODUCTION

Java collection is a collection of various classes which can handle data and perform various operations like searching, sorting, accessing and deleting data. Java has a list of classes and interfaces which handles group of objects as a single unit. This is also called collection framework. The collection framework has a Collection interface and Map interface as a root of all collection classes and interfaces. They are available in java.util package. To use classes of collection framework we need to import java.util package in our program.

As discussed earlier a java collection framework provides an architecture using which we can store and manipulate a group of object as a single unit. In collection framework java has implementation of classes, interface and algorithms. Interfaces are the abstract data types which allows collection to operate independently. Classes are the implementation of collection interfaces. They represents the data structures which a programmer can be use to improve performance. Algorithms are the ways using which the data can be search, sort or manipulated efficiently.

The java collection framework provides a ready made implementation of various data structures as well as the algorithm implementation for those data structure using which we can manipulate the data.

2.3 WHY COLLECTION FRAMEWORK?

In earlier days when collection framework was not introduced the standard way of managing group of java objects as a single unit were Arrays, Vectors and HashTable. These classes have no common interface. Using these data structure it will be difficult for programmers to implement programs, as each data structure has different method or syntax for manipulating member objects. Hence it will be difficult for programmers to write a common algorithm for all this classes. The other limitation was about the Vector class. As the methods of Vector class are declared final we can not extend Vector class to implement other similar data structure.

Hence java developers decided to deal with this problem and introduced the Collection Framework in JDK 1.2.

The Vector and HashTable classes were modified in collection frame work as per the new requirements.

The followings are the advantages of using collection framework.

- Set of common functions implemented for all classes like ArrayList, Vector, LinkedList etc.
- Programmer is free from implementation of data structure. The efficient implementation of data structure is readily available which programmer can directly use.
- Hence the data structures and algorithm are efficiently implemented the performance of program can be improved.

2.4 Hierarchy in collection framework

The Below igure 58 shows the various interfaces and classes of collection framework and relationship among them. In this diagram the grey colour boxes are the interfaces and white colour boxes represents classes.

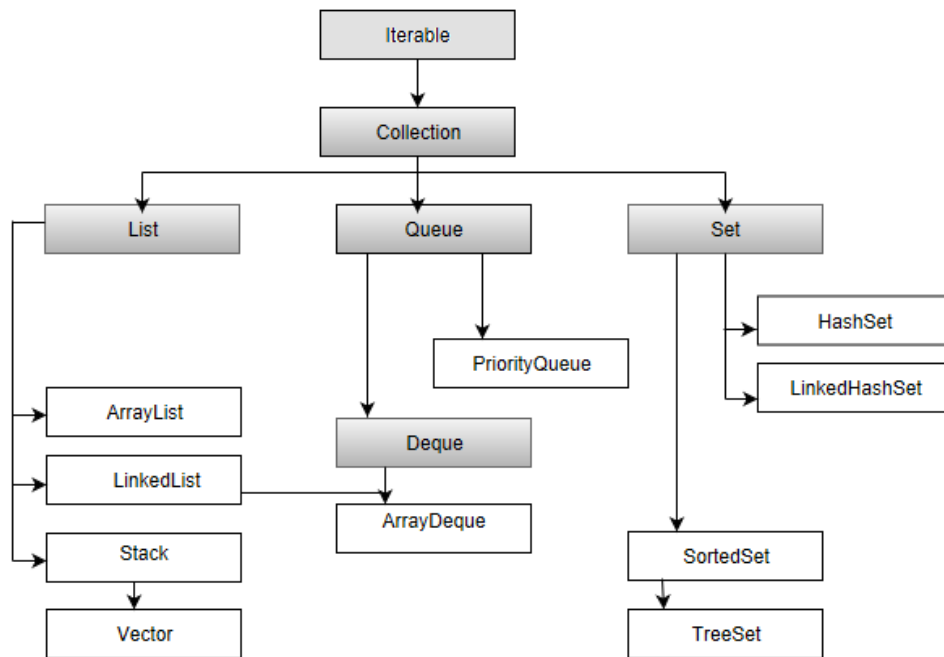


Figure-58 Java Collection Framework

2.5 COLLECTION INTERFACE

The collection interface is the top of the java collection hierarchy. The various method declared in this interface are,

- 1). **boolean add (Object obj)** : Ensures that this Collection contains the specified element
- 2). **boolean addAll(Collection c)** : Adds all of the elements in the specified Collection to this Collection.
- 3). **void clear ()** : Removes all of the elements from this Collection (optional operation).
- 4). **boolean contains (Object o)** : Returns true if this Collection contains the specified element.
- 5). **boolean containsAll (Collection c)** : Returns true if this Collection contains all of the elements in the specified Collection.
- 6). **boolean equals (Object o)** : Compares the specified Object with this Collection for equality.
- 7). **int hashCode ()** : Returns the hash code value for this Collection.
- 8). **boolean isEmpty ()** : Returns true if this Collection contains no elements.

- 9). **Iterator iterator()** : Returns an Iterator over the elements in this Collection.
- 10). **boolean remove (Object o)** : Removes a single instance of the specified element from this Collection, if it is present (optional operation).
- 11). **boolean removeAll (Collection c)** : Removes from this Collection all of its elements that are contained in the specified Collection (optional operation).
- 12). **boolean retainAll (Collection c)** : Retains only the elements in this Collection that are contained in the specified Collection (optional operation).
- 13). **int size ()** : Returns the number of elements in this Collection.
- 14). **Object [] toArray ()** : Returns an array containing all of the elements in this Collection.
- 15). **Object [] toArray (Object[] a)** : Returns an array containing all of the elements in this Collection, whose runtime type is that of the specified array.

2.6 SET INTERFACE

It extends the Collection interface and contains a unique elements i.e. it can not store duplicate objects. It is an unordered collection of the objects. Set is implemented by HashSet, LinkedHashSet or TreeSet classes. The methods declared in Set interface are,

- 1). **int size()**: to get the number of elements in the Set.
- 2). **boolean isEmpty()**: to check if Set is empty or not.
- 3). **boolean contains(Object o)**: Returns true if this Set contains the specified element.
- 4). **Iterator iterator()**: Returns an iterator over the elements in this set. The elements are returned in no particular order.
- 5). **Object[] toArray()**: Returns an array containing all of the elements in this set. If this set makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.
- 6). **boolean add(E e)**: Adds the specified element to this set if it is not already present (optional operation).
- 7). **boolean remove(Object o)**: Removes the specified element from this set if it is present (optional operation).

- 8). **boolean removeAll(Collection c)**: Removes from this set all of its elements that are contained in the specified collection (optional operation).
- 9). **boolean retainAll(Collection c)**: Retains only the elements in this set that are contained in the specified collection (optional operation).
- 10). **void clear()**: Removes all the elements from the set.
- 11). **Iterator iterator()**: Returns an iterator over the elements in this set.

2.7 LIST INTERFACE

The `Java.util.List` extends the `Collection` interface. It stores ordered collection of objects. The duplicate values can be stored in list. The List preserves the insertion order and hence allows positional access and insertion of elements. List Interface is implemented as `ArrayList`, `LinkedList`, `Vector` and `Stack` classes.

The followings are the methods declared in List interface.

- 1). **void add(int index, Object O)**: This method adds given element at specified index.
- 2). **boolean addAll(int index, Collection c)**: This method adds all elements from specified collection to list. First element gets inserted at given index. If there is already an element at that position, that element and other subsequent elements(if any) are shifted to the right by increasing their index.
- 3). **Object remove(int index)**: This method removes an element from the specified index. It shifts subsequent elements(if any) to left and decreases their indexes by 1.
- 4). **Object get(int index)**: This method returns element at the specified index.
- 5). **Object set(int index, Object new)**: This method replaces element at given index with new element. This function returns the element which was just replaced by new element.
- 6). **int indexOf(Object o)**: This method returns first occurrence of given element or -1 if element is not present in list.
- 7). **int lastIndexOf(Object o)**: This method returns the last occurrence of given element or -1 if element is not present in list.

- 8). **List subList(int fromIndex,int toIndex)**:This method returns List view of specified List between fromIndex(inclusive) and toIndex(exclusive).

2.8 QUEUE INTERFACE

The Queue interface is available in java.util package and extends the Collection interface. The queue collection is used to hold the elements about to be processed and provides various operations like the insertion, removal etc. It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of list i.e. it follows the FIFO or the First-In-First-Out principle. Being an interface the queue needs a concrete class for the declaration and the most common classes are the PriorityQueue and LinkedList in Java.It is to be noted that both the implementations are not thread safe. PriorityBlockingQueue is one alternative implementation if thread safe implementation is needed.

The function of Queue interface are,

- 1). **add()**: This method is used to add elements at the tail of queue. More specifically, at the last of linkedlist if it is used, or according to the priority in case of priority queue implementation.
- 2). **peek()** : This method is used to view the head of queue without removing it. It returns Null if the queue is empty.
- 3). **element()**:This method is similar to peek(). It throws NoSuchElementException when the queue is empty.
- 4). **remove()**: This method removes and returns the head of the queue. It throws NoSuchElementException when the queue is impty.
- 5). **poll()**: This method removes and returns the head of the queue. It returns null if the queue is empty.
- 6). **size()**: This method return the no. of elements in the queue.

2.9 DEQUE INTERFACE

The java.util.Deque interface is a subtype of the java.util.Queue interface. The Deque is related to the double-ended queue that supports addition or removal of elements from either end of the data structure, it can be used as a queue (first-in-

first-out/FIFO) or as a stack (last-in-first-out/LIFO). These are faster than Stack and LinkedList.

The following are the methods of Queue interface,

- 1). **add(element)**: Adds an element to the tail.
- 2). **addFirst(element)**: Adds an element to the head.
- 3). **addLast(element)**: Adds an element to the tail.
- 4). **offer(element)**: Adds an element to the tail and returns a boolean to explain if the insertion was successful.
- 5). **offerFirst(element)**: Adds an element to the head and returns a boolean to explain if the insertion was successful.
- 6). **offerLast(element)**: Adds an element to the tail and returns a boolean to explain if the insertion was successful.
- 7). **iterator()**: Returns an iterator for this deque.
- 8). **descendingIterator()**: Returns an iterator that has the reverse order for this deque.
- 9). **push(element)**: Adds an element to the head.
- 10). **pop(element)**: Removes an element from the head and returns it.
- 11). **removeFirst()**: Removes the element at the head.
- 12). **removeLast()**: Removes the element at the tail.
- 13). **poll()**: Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty.
- 14). **pollFirst()**: Retrieves and removes the first element of this deque, or returns null if this deque is empty.
- 15). **pollLast()**: Retrieves and removes the last element of this deque, or returns null if this deque is empty.
- 16). **peek()**: Retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty.
- 17). **peekFirst()**: Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty.
- 18). **peekLast()**: Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty.

2.10 ITERATOR INTERFACE

Iterator is an interface that iterates the elements. It is used to traverse the list and modify the elements. Iterator interface has three methods which are mentioned below:

- 1). **public boolean hasNext():** This method returns true if the iterator has more elements.
- 2). **public object next() :** It returns the element and moves the cursor pointer to the next element.
- 3). **public void remove():** This method removes the last elements returned by the iterator.

2.11 IMPLEMENTATION OF LIST

The List interface is further implemented into the following classes:

1. ArrayList
2. LinkedList
3. Vectors

➤ ArrayList

ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. The size of the list can be increased dynamically if the elements are added more than the initial size.

```
ArrayList obj = new ArrayList ();
```

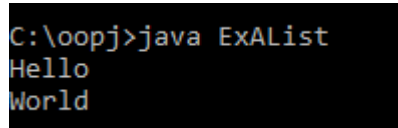
Some of the methods in ArrayList are listed below:

- 1). **boolean add(Collection c) :** Appends the specified element to the end of a list.
- 2). **void add(int index, Object element):** Inserts the specified element at the specified position.
- 3). **void clear() :** Removes all the elements from this list.
- 4). **int lastIndexOf(Object o) :** Return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
- 5). **Object clone() :** Return a shallow copy of an ArrayList.
- 6). **Object[] toArray() :** Returns an array containing all the elements in the list.

7). **void trimToSize()** : Trims the capacity of this ArrayList instance to be the list's current size.

Example,

```
import java.util.*;
class ExAList
{
    public static void main(String args[])
    {
        ArrayList al = new ArrayList();
        al.add("Hello");
        al.add("World");
        Iterator itr = al.iterator();
        while(itr.hasNext())
        { System.out.println (itr.next()); }
    }
}
```



```
C:\oopj>java ExAList
Hello
World
```

Figure-59 Output of program

➤ **LinkedList**

LinkedList is a sequence of links which contains items. Each link contains a connection to another link.

Syntax: `LinkedList object = new LinkedList();`

Java LinkedList class uses two types of Linked list to store the elements:

- Singly Linked List
- Doubly Linked List

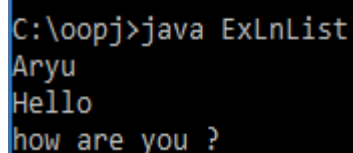
Some of the methods in the LinkedList are listed below:

- 1). **boolean add(Object o)** : It is used to append the specified element to the end of the vector.
- 2). **boolean contains(Object o)** : Returns true if this list contains the specified element.
- 3). **void add (int index, Object element)** : Inserts the element at the specified element in the vector.

- 4). **void addFirst(Object o)** : It is used to insert the given element at the beginning.
- 5). **void addLast(Object o)** : It is used to append the given element to the end.
- 6). **int size()** : It is used to return the number of elements in a list
- 7). **boolean remove(Object o)** : Removes the first occurrence of the specified element from this list.
- 8). **int indexOf(Object element)** : Returns the index of the first occurrence of the specified element in this list, or -1.
- 9). **int lastIndexOf(Object element)** : Returns the index of the last occurrence of the specified element in this list, or -1.

Example,

```
import java.util.*;
public class ExLnList
{
    public static void main(String args[])
    {
        LinkedList<String> al = new LinkedList<String>();
        al.add("Aryu");
        al.add("Hello");
        al.add("how are you ?");
        Iterator<String> itr = al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```



```
C:\oopj>java ExLnList
Aryu
Hello
how are you ?
```

Figure-60 Output of program

➤ Vectors

Vectors are similar to arrays, where the elements of the vector object can be accessed via an index into the vector. Vector implements a dynamic array. Also, the

vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList, but with two differences : Vector is synchronized and Vector contains many legacy methods that are not part of the collections framework.

We can create a Vector using following constructor.

- 1). **Vector ()**: Creates a default vector of initial capacity is 10.
- 2). **Vector (int size)**: Creates a vector whose initial capacity is specified by size.
- 3). **Vector (int size, int incr)**: Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time that a vector is resized upward.
- 4). **Vector (Collection c)**: Creates a vector that contains the elements of collection c.

The followings are some of the methods of the Vector class,

- 1). **boolean add(Object o)** : Appends the specified element to the end of the list.
- 2). **void clear()** : Removes all of the elements from this list.
- 3). **void add(int index, Object element)** : Inserts the specified element at the specified position.
- 4). **boolean remove(Object o)** : Removes the first occurrence of the specified element from this list.
- 5). **boolean contains(Object element)** : Returns true if this list contains the specified element.
- 6). **int indexOfObject (Object element)** : Returns the index of the first occurrence of the specified element in the list, or -1.
- 7). **int size()** : Returns the number of elements in this list.
- 8). **int lastIndexOf (Object o)** : Return the index of the last occurrence of the specified element in the list, or -1 if the list does not contain any element.

```
import java.util.*;
class ExVector {
    public static void main(String[] arg)
    {
```

```

Vector v = new Vector();

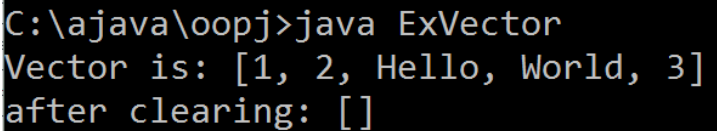
v.add(0, 1);
v.add(1, 2);
v.add(2, "Hello");
v.add(3, "World");
v.add(4, 3);

System. out. println ("Vector is: " + v);

v.clear();

System. out. println ("after clearing: " + v);
}
}

```



```

C:\ajava\oopj>java ExVector
Vector is: [1, 2, Hello, World, 3]
after clearing: []

```

Figure-61 Output of program

➤ Stack

A Stack class models and implements Stack data structure. The class is based on the principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek. The Stack class extends Vector class with the five stack functions. The Stack class can also be referred to as the subclass of Vector.

The followings are the methods in Stack class.

- 1). **Object push(Object element)** : Pushes an element on the top of the stack.
- 2). **Object pop()** : Removes and returns the top element of the stack. An 'EmptyStackException' exception is thrown if we call pop() when the invoking stack is empty.
- 3). **Object peek()** : Returns the element on the top of the stack, but does not remove it.
- 4). **boolean empty()** : It returns true if nothing is on the top of the stack. Else, returns false.
- 5). **int search(Object element)** : It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1.

Example,

```
import java.util.Stack;
public class ExStack {
    public static void main(String[] args) {
        Stack<String> stk = new Stack<>();
        stk.push("one");
        stk.push("two");
        stk.push("three");
        stk.push("four");
        System.out.println ("Stack => " + stk);
        System.out.println ();
        String tp = stk.pop();
        System.out.println ("Stack.pop() => " + tp);
        System.out.println ("Current Stack => " + stk);
        System.out.println ();
        tp = stk.peek();
        System.out.println ("Stack.peek() => " + tp);
        System.out.println ("Current Stack => " + stk);
    }
}
```

```
C:\oopj>java ExStack
Stack => [one, two, three, four]

Stack.pop() => four
Current Stack => [one, two, three]

Stack.peek() => three
Current Stack => [one, two, three]
```

Figure-62 Output of program

2.12 IMPLEMENTATION OF QUEUE

A PriorityQueue implements Queue interface. A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a queue follows First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play. The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor are used.

Followings are some of the methods of PriorityQueue class,

- 1). **boolean add(E element)**: This method inserts the specified element into this priority queue.
- 2). **public remove()**: This method removes a single instance of the specified element from this queue, if it is present
- 3). **public poll()**: This method retrieves and removes the head of this queue, or returns null if this queue is empty.
- 4). **public peek()**: This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- 5). **Iterator iterator()**: Returns an iterator over the elements in this queue.
- 6). **boolean contains(Object o)**: This method returns true if this queue contains the specified element
- 7). **void clear()**: This method is used to remove all of the contents of the priority queue.
- 8). **boolean offer(E e)**: This method is used to insert a specific element into the priority queue.
- 9). **int size()**: The method is used to return the number of elements present in the set.
- 10). **toArray()**: This method is used to return an array containing all of the elements in this queue.
- 11). **Comparator comparator()**: The method is used to return the comparator that can be used to order the elements of the queue.

Example,

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

```

class ExQueue {
    public static void main(String args[]){
        PriorityQueue<String> queue=new PriorityQueue<String>();
        queue.add("Hello");
        queue.add("World");
        queue.add("Aryu");
        System.out.println ("head:"+queue.element());
        System.out.println ("head:"+queue.peek());
        System.out.println ("iterating the queue elements:");
        Iterator itr=queue.iterator();
        while(itr.hasNext()){
            System.out.println (itr.next());
        }
        queue.remove();
        queue.poll();
        System.out.println ("after removing two elements:");
        Iterator<String> itr2=queue.iterator();
        while(itr2.hasNext()){
            System.out.println (itr2.next());
        }
    }
}

```

```

C:\ajava\oopj>java ExQueue
head:Aryu
head:Aryu
iterating the queue elements:
Aryu
World
Hello
after removing two elements:
World

```

Figure-63 Output of program

2.13 IMPLEMENTATION OF SET

Set has its implementation in various classes such as HashSet, TreeSet and LinkedHashSet. HashSet stores elements in random order whereas LinkedHashSet stores elements according to insertion order and TreeSet stores according to natural ordering.

➤ **HashSet**

Java HashSet class creates a collection that uses a hash table for storage. HashSet only contains unique elements and it inherits the AbstractSet class and implements Set interface. Also, it uses a mechanism of hashing to store the elements.

Following are some of the methods of HashSet class:

- 1). **boolean add (Object o)** : Adds the specified element to this set if it is not already present.
- 2). **boolean contains (Object o)** : Returns true if the set contains the specified element.
- 3). **void clear ()** : Removes all the elements from the set.
- 4). **boolean isEmpty()** : Returns true if the set contains no elements.
- 5). **boolean remove(Object o)** : Remove the specified element from the set.
- 6). **Object clone()** : Returns a shallow copy of the HashSet instance: the elements themselves are not cloned.
- 7). **Iterator iterator()** : Returns an iterator over the elements in this set.
- 8). **int size()** : Return the number of elements in the set.

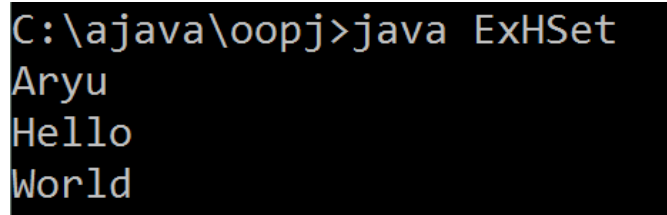
Example,

```
import java.util.*;
class ExHashSet
{
    public static void main ( String args [] )
    {
        HashSet <String> al = new HashSet ();
        al.add( "Hello" );
        al.add( "World" );
        al.add( "Aryu" );
        Iterator <String> itr = al.iterator ();
        while ( itr.hasNext () )
```

```

    {
    System. out. println ( itr.next() );
    }
    }
}

```



```

C:\ajava\oopj>java ExHSet
Aryu
Hello
World

```

Figure-64 Output of program

➤ **LinkedHashset**

Java LinkedHashSet class is a Hash table and Linked list implementation of the set interface. It contains only unique elements like HashSet. Linked HashSet also provides all optional set operations and maintains insertion order.

- 1). **public boolean add(Object o)** : Adds an object to a LinkedHashSet if already not present in HashSet.
- 2). **public boolean remove(Object o)** : Removes an object from LinkedHashSet if found in HashSet.
- 3). **public boolean contains(Object o)** : Returns true if object found else return false
- 4). **public boolean isEmpty()** : Returns true if LinkedHashSet is empty else return false
- 5). **public int size()** : Returns number of elements in the LinkedHashSet

Example,

```

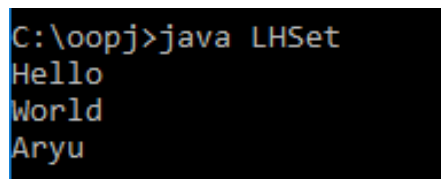
import java.util.*;
public class LHSet
{
public static void main ( String args[] )
{
LinkedHashSet <String> al = new LinkedHashSet();
al.add ( "Hello" );

```

```

al.add ( "World" );
al.add ( "Aryu" );
Iterator <String> itr = al.iterator ();
While ( itr.hasNext() )
{
System. out. println ( itr.next() );
}
}
}
}

```



```

C:\oopj>java LHSet
Hello
World
Aryu

```

Figure-65 Output of program

➤ TreeSet

TreeSet class implements the Set interface that uses a tree for storage. The objects of this class are stored in the ascending order. Also, it inherits AbstractSet class and implements NavigableSet interface. It contains only unique elements like HashSet. In TreeSet class, access and retrieval time are faster.

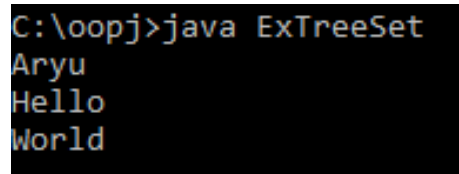
The followings are some of the methods of TreeSet class.

- 1). **boolean addAll(Collection c)** : Add all the elements in the specified collection to this set.
- 2). **boolean contains(Object o)** : Returns true if the set contains the specified element.
- 3). **boolean isEmpty()** : Returns true if this set contains no elements.
- 4). **boolean remove(Object o)** : Remove the specified element from the set.
- 5). **void add(Object o)** : Add the specified element to the set.
- 6). **void clear()** : Removes all the elements from the set.
- 7). **Object clone()** : Return a shallow copy of this TreeSet instance.
- 8). **Object first()** : Return the first element currently in the sorted set.
- 9). **Object last()** : Return the last element currently in the sorted set.

10). **int size()** : Return the number of elements in the set. Let us understand these.

Example,

```
import java.util.*;
class ExTreeSet
{
    public static void main( String args[] )
    {
        TreeSet <String> al = new TreeSet <String> ();
        al.add ( "Hello" );
        al.add ( "World" );
        al.add ( "Aryu" );
        Iterator <String> itr = al.iterator();
        While ( itr.hasNext() )
        {
            System. out. println ( itr.next ( ) );
        }
    }
}
```



```
C:\oopj>java ExTreeSet
Aryu
Hello
World
```

Figure-66 Output of program

2.14 LET US SUM UP

Collection: It is a parent interface of all classes of collection framework. It declares the methods that every collection will have.

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

List: List interface is the child interface of Collection interface. It inherits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

ArrayList: The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

LinkedList: LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized.

Vector: Vector uses a dynamic array to store the data elements. It is similar to ArrayList. It is synchronized and contains many methods that are not the part of Collection framework.

Stack: The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

Queue: Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed.

PriorityQueue: The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue does not allow null values to be stored in the queue

Set: Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items.

HashSet: HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

LinkedHashSet: It represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

TreeSet: Java TreeSet class implements the Set interface that uses a tree for storage. TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order

2.15 CHECK YOUR PROGRESS

➤ MCQ

1) Which of these interface handle sequences?

- | | |
|---------|---------------|
| a) Set | c) Comparator |
| b) List | d) Collection |

2) Which of these interface declares core method that all collections will have?

- | | |
|-----------------|---------------|
| a) set | c) Comparator |
| b) EventListner | d) Collection |

3) Which of this interface must contain a unique element?

- | | |
|---------|---------------|
| a) Set | c) Array |
| b) List | d) Collection |

4) What is the output of this program?

```
import java.util.*;
class Collection_Algos
{
    public static void main(String args[])
    {
        LinkedList list = new LinkedList();
        list.add(new Integer(2));
        list.add(new Integer(8));
        list.add(new Integer(5));
        list.add(new Integer(1));
        Iterator i = list.iterator();
```


- a)pop()
- b)top()
- c)peek()
- d)search()

8) To delete the top element of the stackmethod is used?

- a)delete()
- b)pop()
- c)remove()
- d)peep()

9) To insert a key-value pair in a hashtablemethod is used?

- a)insert()
- b) put()
- c) add()
- d) push()

10) To find the value of an element by it's key.....method is used?

- a) get()
- b) getvalue()
- c) pop()
- d) value()

2.16 CHECK YOUR PROGRESS:POSSIBLE ANSWERS

➤ MCQ

- 1) b
- 2) d
- 3) a
- 4) c
- 5) d
- 6) b
- 7) b
- 8) b
- 9) c
- 10) a

2.17 FURTHER READING

- 1) Java Collections Framework Tutorials - BeginnersBook.com
<https://beginnersbook.com/java-collections-tutorials/>
- 2) Java Collections Framework | Collections in Java With Examples
<https://www.edureka.co/blog/java-collections/>
- 3) “Java 2: The Complete Reference” by Herbert Schildt, McGraw Hill Publications.
- 4) “Effective Java” by Joshua Bloch, Pearson Education.

2.18 ASSIGNMENTS

- 1) Using list perform following operation on it in java program. (use ArrayList and LinkedList)

1. Creating a new list
2. Basic operations
3. Iterating over a list
4. Searching for an element in a list
5. Sorting a list
6. Copying one list into another
7. Shuffling elements in a list
8. Reversing elements in a list
9. Extracting a portion of a list
10. Converting between Lists and arrays
11. List to Stream
12. Concurrent lists.

- 2) Write a java program to evaluate arithmetic operation using stack.
- 3) Implement a java program to show various operation of queue.
- 4) Implement singly linked list and its operations in java program.