Collection(Interface) & their types with description in java.

****What is a Java Collection Framework?****

A Java collection framework provides an architecture to store and manipulate a group of objects. A Java collection framework includes the following:

* Interfaces
* Classes
* Algorithm

****Interfaces****: Interface in Java refers to the abstract data types. They allow Java collections to be manipulated independently from the details of their representation. Also, they form a hierarchy in object-oriented programming languages.

****Classes:****Classes in Java are the implementation of the collection interface. It basically refers to the data structures that are used again and again.

****Algorithm:**** Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces. Algorithms are polymorphic in nature as the same method can be used to take many forms or you can say perform different implementations of the Java collection interface.

So why do you think we need Java collections? The Java collection framework provides the developers to access prepackaged data structures as well as algorithms to manipulate data. Next, let us move to the Java collections framework hierarchy and see where these interfaces and classes resides.

## ****Why use Java collection?****

There are several benefits of using Java collections such as:

* Reducing the effort required to write the code by providing useful data structures and algorithms.
* Java collections provide high-performance and high-quality data structures and algorithms thereby increasing the speed and quality.
* Unrelated APIs can pass collection interfaces back and forth.
* Decreases extra effort required to learn, use, and design new API’s.
* Supports reusability of standard data structures and algorithms.

## ****Java Collection Framework Hierarchy****

As we have learned Java collection framework includes interfaces and classes. Now, let us see the Java collections framework hierarchy.



In the above image, blue part refers to the different interfaces and the yellow part defines the class. Now, let us understand these components in detail.

## ****Java Collections : Interface****

****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.

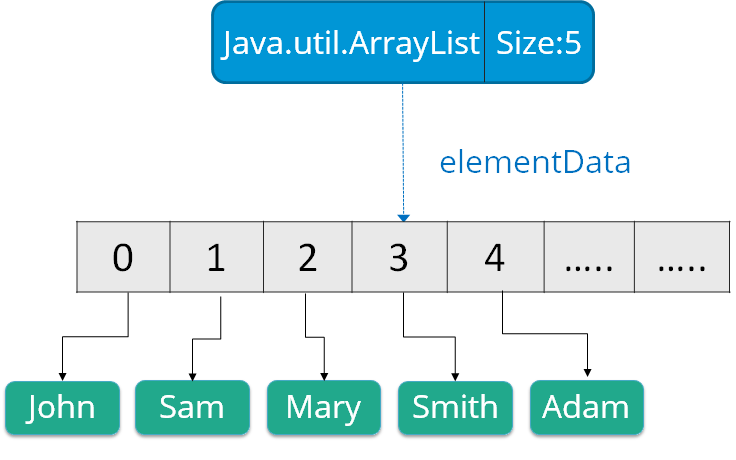
There are three components that extend the collection interface i.e List, Queue and Sets. Let’s learn about them in detail:

## ****Java collections: List****

A List is an ordered Collection of elements which may contain duplicates. It is an interface that extends the Collection interface. Lists are further classified into the following:

* ArrayList
* LinkedList
* Vectors

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



****Syntax****: ArrayList object = new ArrayList ();

Some of the methods in array list are listed below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(Collection c) | Appends the specified element to the end of a list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| void clear() | Removes all the elements from this list. |
| 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. |
| Object clone() | Return a shallow copy of an ArrayList. |
| Object[] toArray() | Returns an array containing all the elements in the list. |
| void trimToSize() | Trims the capacity of this ArrayList instance to be the list’s current size. |

Let us understand Array list with a programmatic example:

**import** java.util.\*;

**class** ArrayListExample{

**public** **static** **void** main(String args[]){

 ArrayList al=**new** ArrayList();  // creating array list

 al.add("Rohit");                // adding elements

 al.add("Krishna");

 Iterator itr=al.iterator();

**while**(itr.hasNext()){

 System.out.println(itr.next());

 }

 }

 }

In the above code, it will return the names that we have added using add() method i.e:

Rohit

Krishna

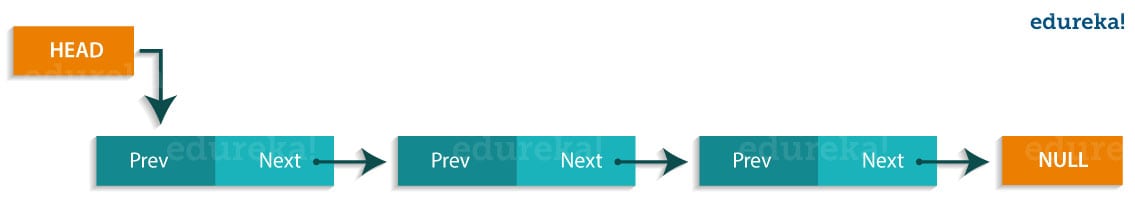
****Linked List****: Linked List is a sequence of links which contains items. Each link contains a connection to another link.

Syntax: Linkedlist object = new Linkedlist();

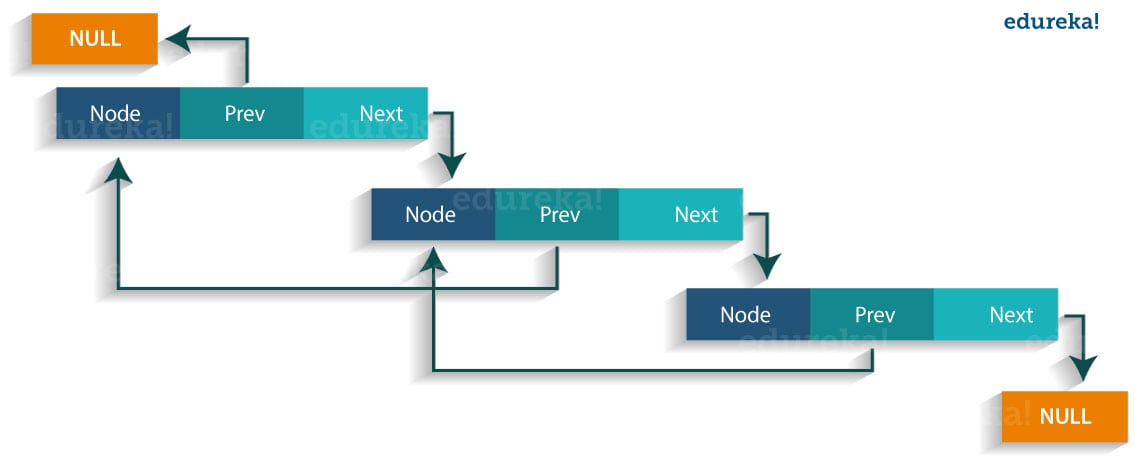
Java Linked List class uses two types of Linked list to store the elements:

* Singly Linked List
* Doubly Linked List

****Singly Linked List****: In a singly Linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list. Refer to the below image to get a better understanding of single Linked list.



****Doubly Linked List****: In a doubly Linked list, it has two references, one to the next node and another to previous node. You can refer to the below image to get a better understanding of doubly linked list.

  
Some of the methods in the linked list are listed below:

|  |  |
| --- | --- |
| ****Method**** | ****Description**** |
| boolean add( Object o) | It is used to append the specified element to the end of the vector. |
| boolean contains(Object o) | Returns true if this list contains the specified element. |
| void add (int index, Object element) | Inserts the element at the specified element in the vector. |
| void addFirst(Object o) | It is used to insert the given element at the beginning. |
| void addLast(Object o) | It is used to append the given element to the end. |
| int size() | It is used to return the number of elements in a list |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| int indexOf(Object element) | Returns the index of the first occurrence of the specified element in this list, or -1. |
| int lastIndexOf(Object element) | Returns the index of the last occurrence of the specified element in this list, or -1. |

Let us understand linked list with a programmatic example:

**import** java.util.\*;

**public** **class** LinkedlistExample{

**public** **static** **void** main(String args[]){

LinkedList<String> al=**new** LinkedList<String>();// creating linked list

al.add("Sai"); // adding elements

al.add("Krishna");

al.add("Rohit");

Iterator<String> itr = al.iterator();

**while**(itr.hasNext()){

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

   }

  }

}

The output of the above program would be:

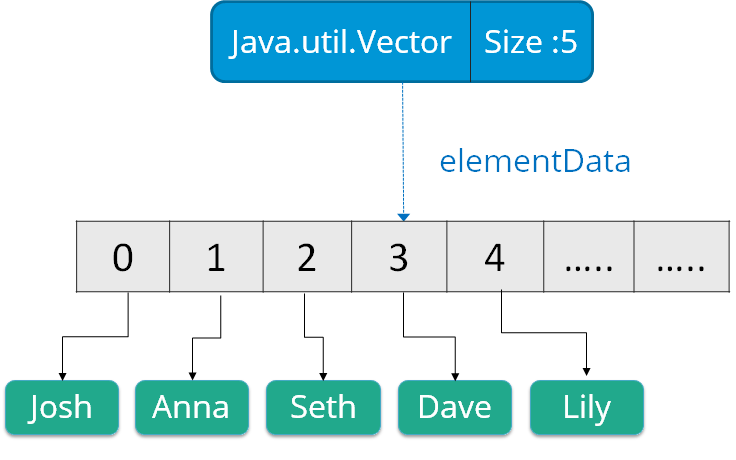
Sai

Krishna

Rohit

****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.
* Vector contains many legacy methods that are not part of the collections framework.



****Syntax****:

Vector object = new Vector(size,increment);

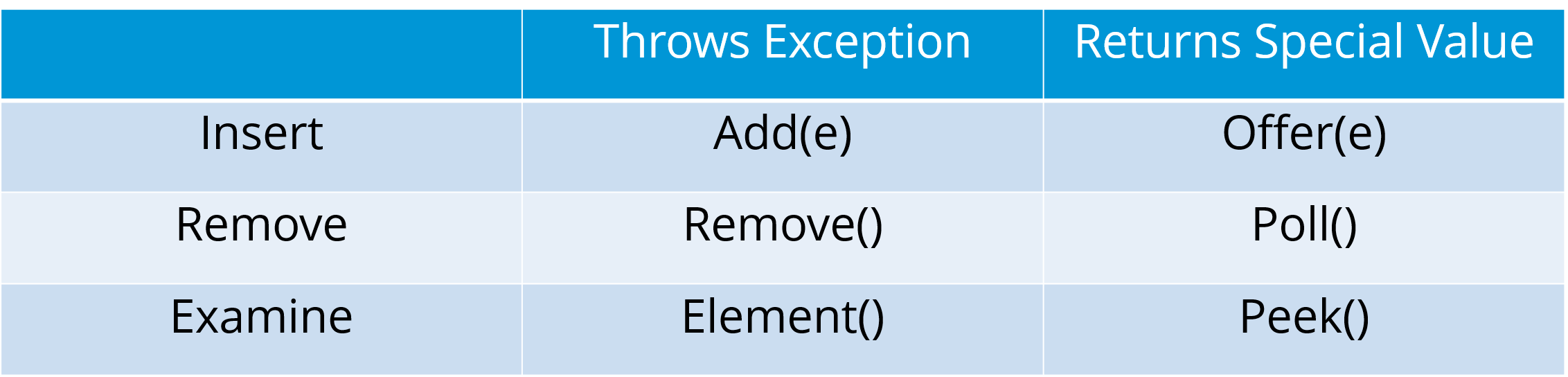
Below are some of the methods of the Vector class:

|  |  |
| --- | --- |
| ****Method**** | ****Description**** |
| boolean add(Object o) | Appends the specified element to the end of the list. |
| void clear() | Removes all of the elements from this list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| boolean contains(Object element) | Returns true if this list contains the specified element. |
| int indexOfObject (Object element) | Returns the index of the first occurrence of the specified element in the list, or -1. |
| int size() | Returns the number of elements in this list. |
| 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. |

Now, let us  move to the next sub type of the Java Collections interface i.e Queue.

## ****Java collections: Queue****

Queue in Java follows a FIFO approach i.e. it orders the elements in First In First Out manner. In a queue, the first element is removed first and last element is removed in the end. Each basic method exists in two forms: one throws an exception if the operation fails, the other returns a special value.



Also, priority queue implements Queue interface. The elements of the ****priority queue**** are ordered according to their natural ordering, or by a Comparator provided at the queue construction time. The head of this queue is the least element with respect to the specified ordering.

Below are some of the methods of Java Queue interface:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(object) | Inserts the specified element into the queue and returns true if it is a success. |
| boolean offer(object) | Inserts the specified element into this queue. |
| Object remove() | Retrieves and removes the head of the queue. |
| Object poll() | Retrieves and removes the head of the queue, or returns null if the queue is empty. |
| Object element() | Retrieves, but does not remove the head of the queue. |
| Object peek() | Retrieves, but does not remove the head of this queue, or returns null if the queue is empty. |

Let us understand these priority queues with a programmatic example:

**import** java.util.\*;

**class** QueueExample {

**public** **static** **void** main(String args[]){

PriorityQueue<String> queue=**new** PriorityQueue<String>();

        // creating priority queue

        queue.add("Sai");

        // adding elements

        queue.add("Krishna");

        queue.add("Rohit");

        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());

        }

    }

}

In the above code, the output would be :

head:Sai

head:Sai

iterating the queue elements:

Sai

Krishna

Rohit

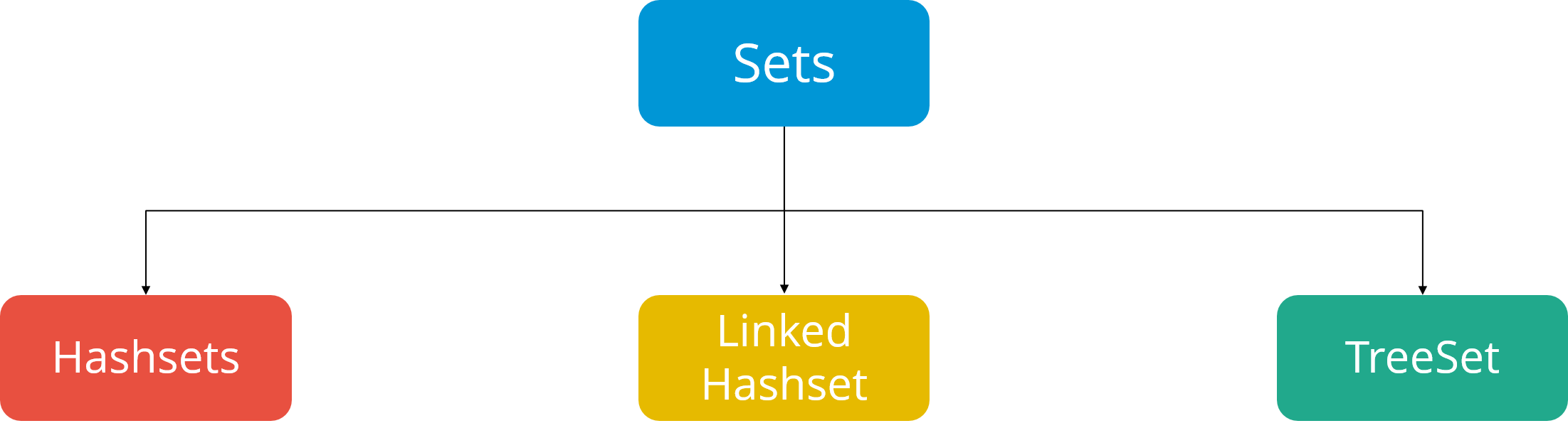
after removing two elements:

Rohit

Next, let us move forward to our next topic in “Java Collections” blog, i.e. Sets.

## ****Java Collections: Sets****

A Set refers to a collection that cannot contain duplicate elements. It is mainly used to model the mathematical set abstraction. Set has its implementation in various classes such as HashSet, TreeSetand LinkedHashSet.



Let’s go into detail on each one of them:

****HashSet****: Java HashSet class creates a collection that use a hash table for storage. Hashset only contain unique elements and it inherits the AbstractSet class and implements Set interface. Also, it uses a mechanism hashing to store the elements.    
Below are some of the methods of Java HashSet class:

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

Let us understand these Hashset with a programmatic example:

**import** java.util.\*;

**class** HashsetExample{

**public** **static** **void** main(String args[]){

HashSet&amp;amp;lt;String&amp;amp;gt; al=**new** HashSet(); // creating hashSet

 al.add("Sai");                 // adding elements

 al.add("Krishna");

 al.add("Rohit");

 Iterator&amp;amp;lt;String&amp;amp;gt; itr=al.iterator();

**while**(itr.hasNext()){

 System.out.println(itr.next());

 }

 }

 }

The output of the above code would be:

Krishna

Sai

Rohit

****Linked Hashset**** : 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. Let us understand these linked Hashset with a programmatic example:

**import** java.util.\*;

**class** LinkedHashsetExample{

**public** **static** **void** main(String args[]){

LinkedHashSet&amp;amp;lt;String&amp;amp;gt; al=**new** LinkedHashSet(); // creating linkedhashset

al.add("Rohit");                            // adding elements

al.add("Sai");

al.add("Krishna");

Iterator&amp;amp;lt;String&amp;amp;gt; itr=al.iterator();

**while**(itr.hasNext()){

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

}

}

}

}

The output of the above code would be:

Rohit

Sai

Krishna

****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.  
Below are some of the methods of Java TreeSet class:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean addAll(Collection c) | Add all the elements in the specified collection to this set. |
| boolean contains(Object o) | Returns true if the set contains the specified element. |
| boolean isEmpty() | Returns true if this set contains no elements. |
| boolean remove(Object o) | Remove the specified element from the set. |
| void add(Object o) | Add the specified element to the set. |
| void clear() | Removes all the elements from the set. |
| Object clone() | Return a shallow copy of this TreeSet instance. |
| Object first() | Return the first element currently in the sorted set. |
| Object last() | Return the last element currently in the sorted set. |
| int size() | Return the number of elements in the set. |

Let us understand these TreeSet with a programmatic example:

**import** java.util.\*;

**class** TreeSetExample{

**public** **static** **void** main(String args[]){

TreeSet&amp;amp;lt;String&amp;amp;gt; al=**new** TreeSet&amp;amp;lt;String&amp;amp;gt;();  // creating treeSet

al.add("Sai");                            // adding elements

al.add("Rohit");

al.add("Krishna");

Iterator&amp;amp;lt;String&amp;amp;gt; itr=al.iterator();

**while**(itr.hasNext()){

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

}

}

}

The output of the above program would be:

Sai

Krishna

Rohit