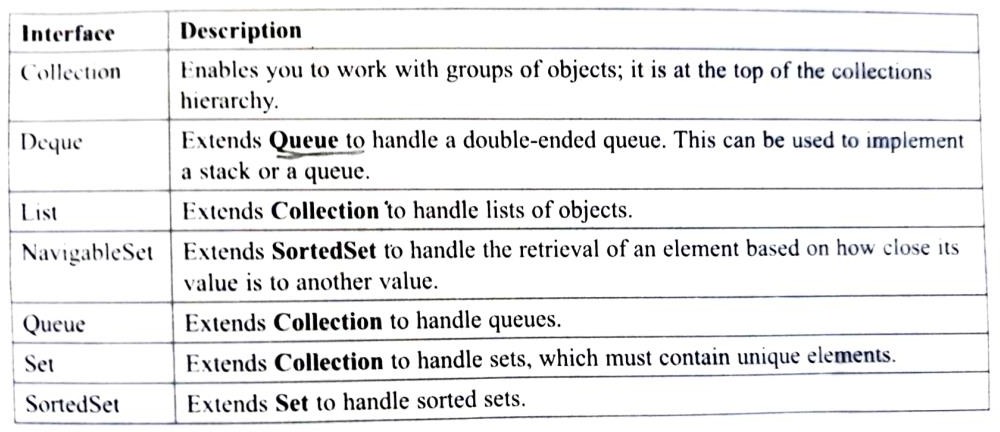
# The Java Collections Framework

* Java Collection Framework enables the user to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements.
* It provides the implementations for the data structures like dynamic array, stack, queue, linked list, tree, and has table
* Collection frameworks have Collection classes, interfaces, and algorithms
* One more important aspect of Collections Framework is the iterator. It offers a way of accessing the elements within a collection, one by one.

## The Collection Interfaces

* The collection interfaces are available in the package **java.util**

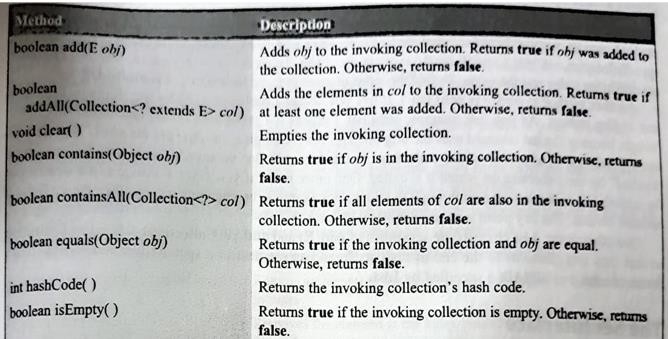


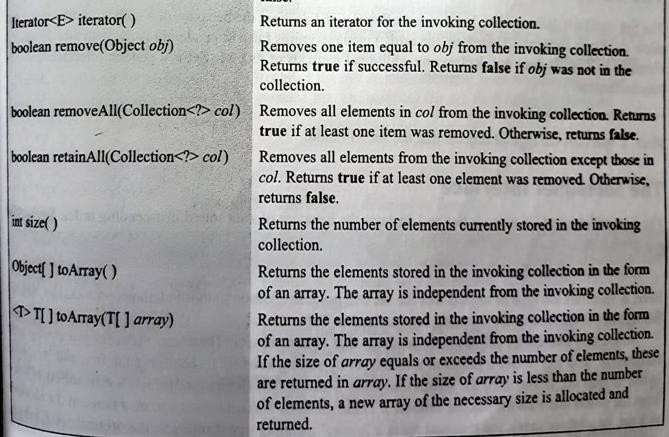
* Collection is a generic interface and the syntax is

## interface Collection<E>

**E** specifies the type of objects that the list will hold

## Methods of Collection interface

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

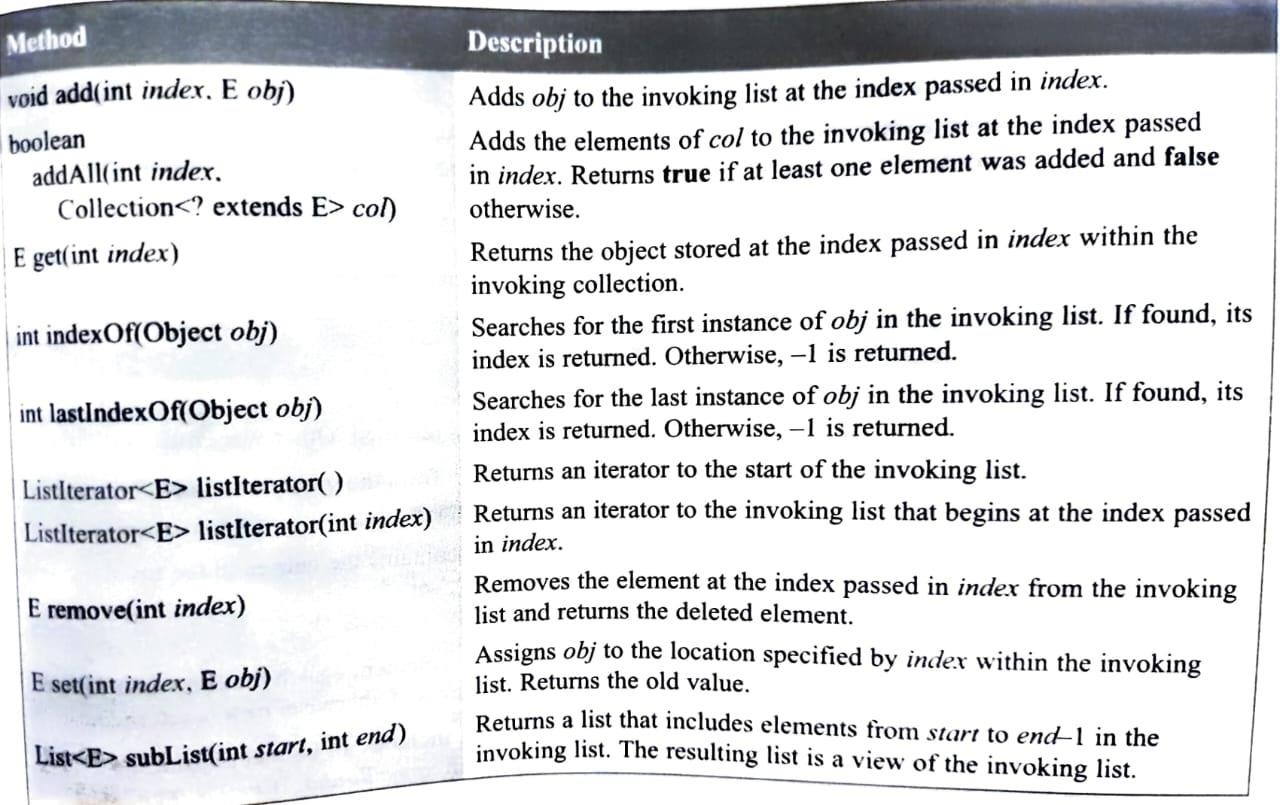


## The List Interface

* The List interface extends Collection and declares the behaviour of a collection that stores a sequence of elements
* The syntax of List interface is

## interface List<E>

**E** specifies the type of objects that the list will hold



## The Set Interface

* + The Set interface defines a set.
  + The syntax to declare one is

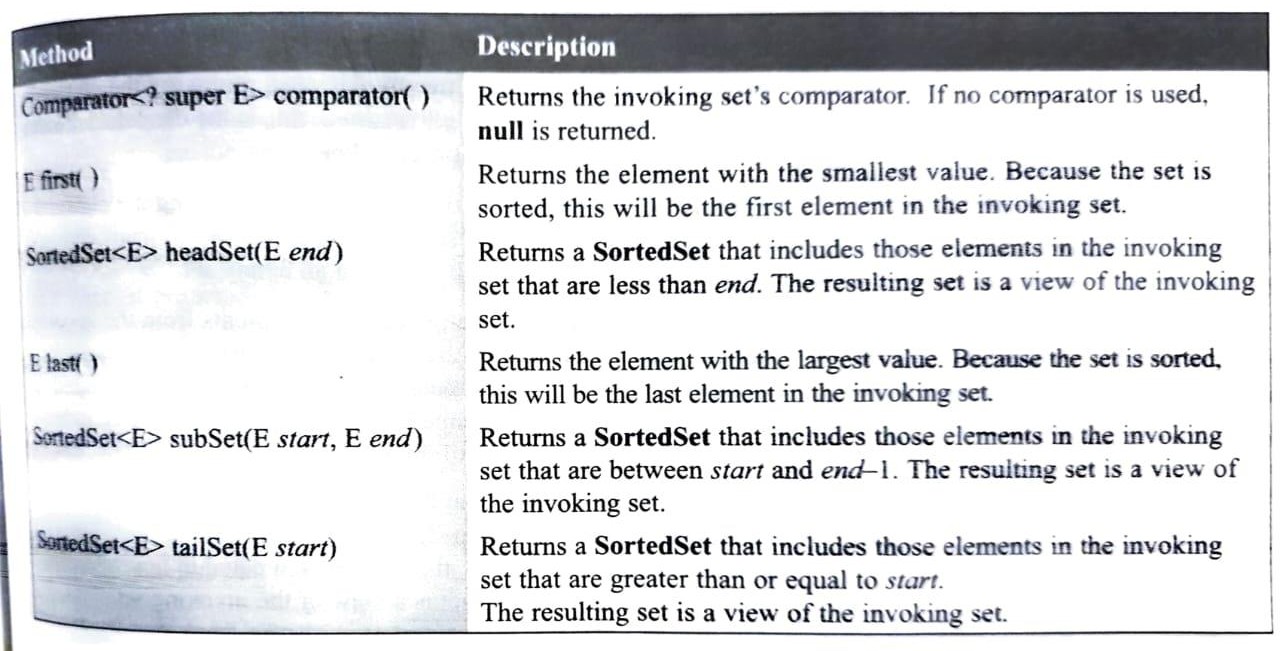
## interface Set<E>

**E** specifies the type of objects that the set will hold.

## The SortedSet Interface

* It extends Set and declares the behaviour of a set sorted in ascending order
* The syntax to declare is

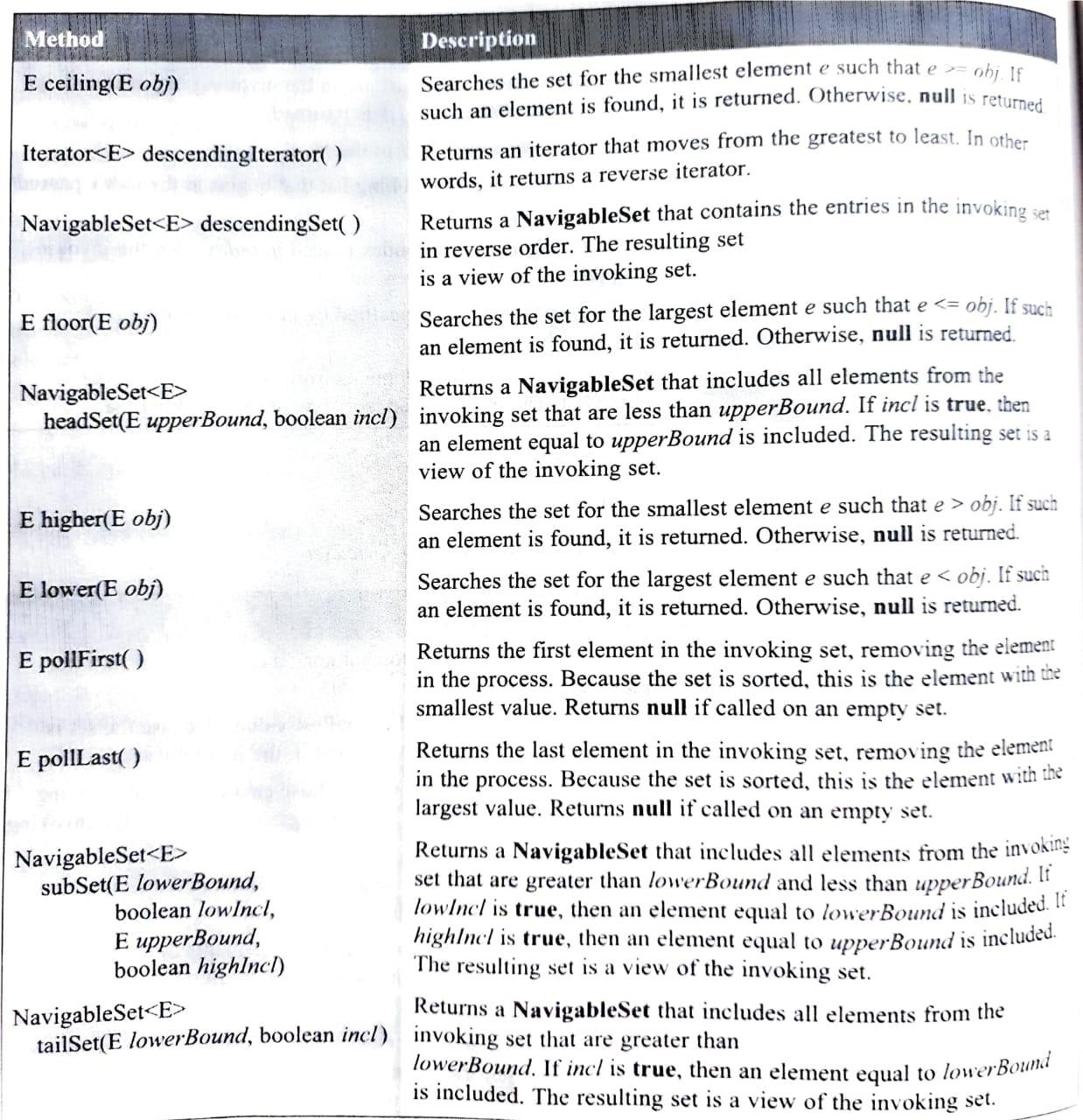
## interface SortedSet<E>



**The NavigableSet Interface**

* It extends SortedSet and declares the behaviour of a collection that supports the retrieval of an element
* The syntax to declare one is

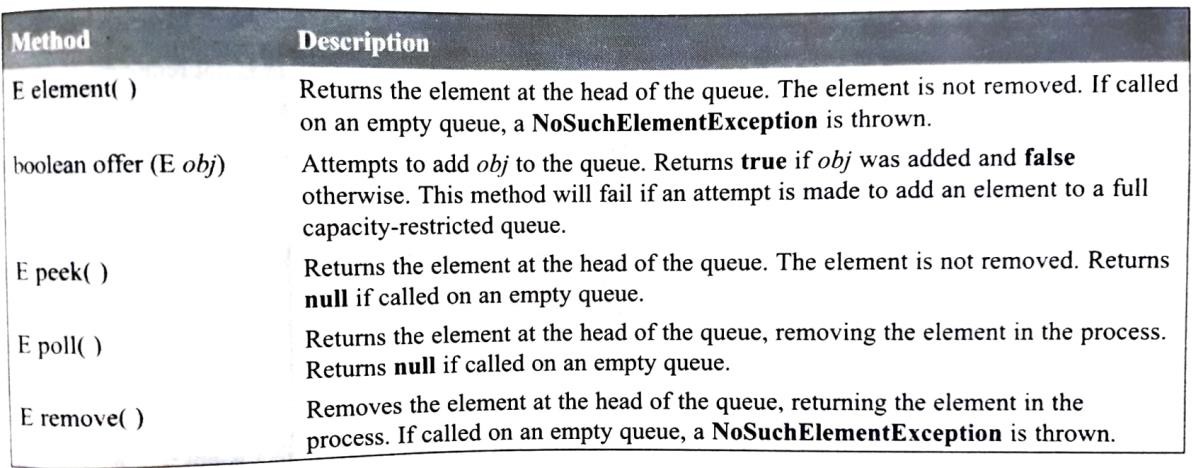
**interface NavigableSet<E> E** specifies the type of objects that the set will hold.



## The Queue Interface

* It extends Collection and declares the behaviour of a queue. Its syntax is

**interface Queue<E> E** specifies the type of objects that the queue will hold.

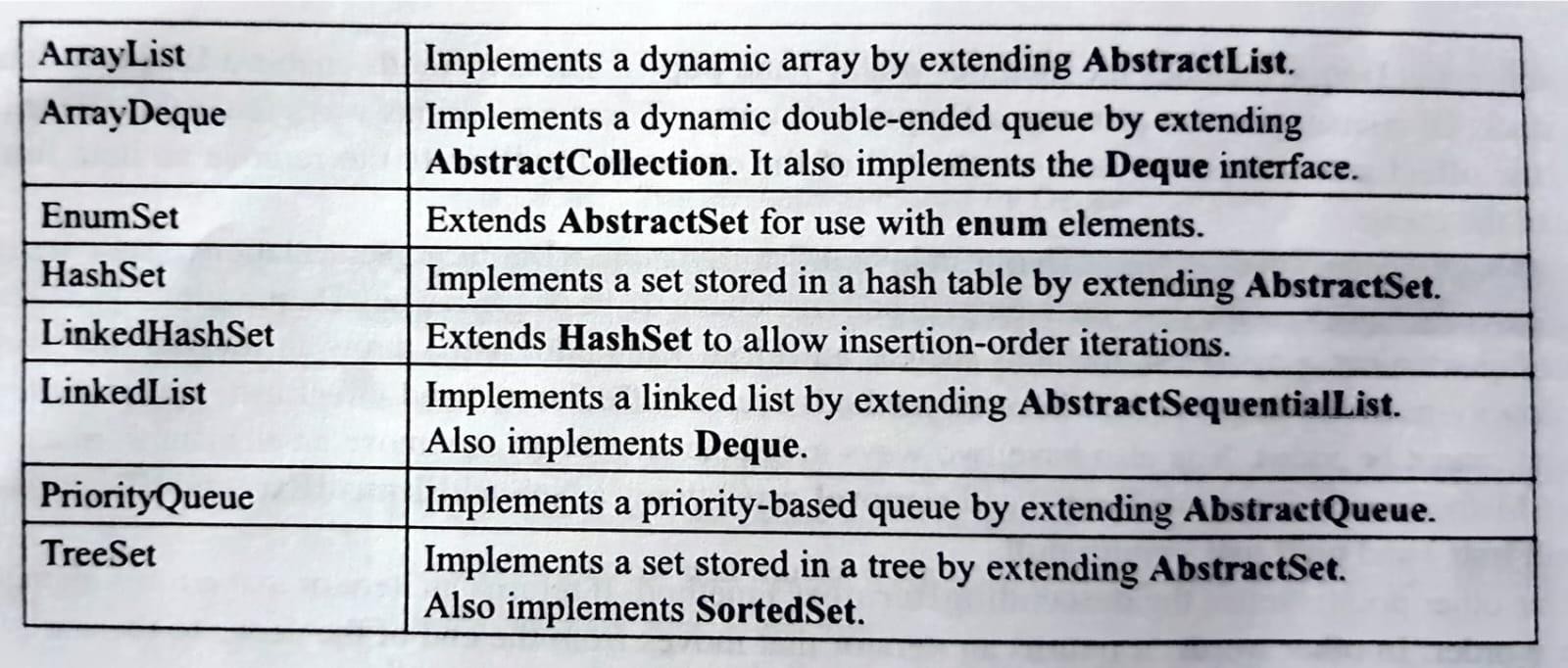
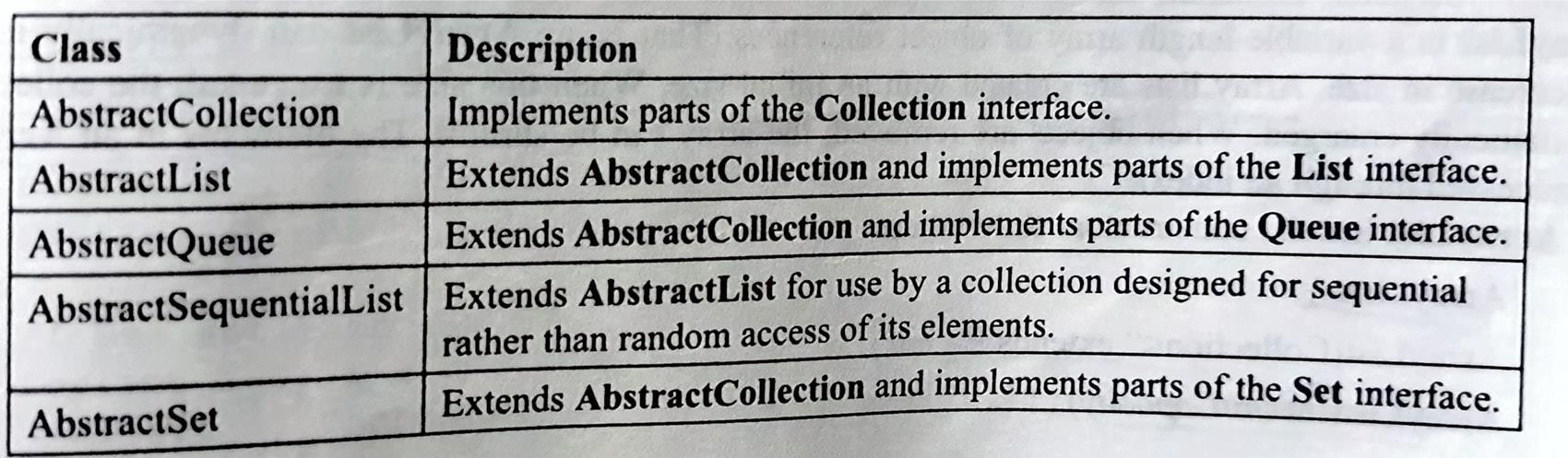


## The Deque Interface

* + It extends Queue and declares the behaviour of a double-ended queue.
  + Its syntax is

## interface Deque<E>

**The Collection Classes**

* The collection interfaces are implemented by different classes

## The ArrayList Class

* + It extends AbstractList and implements the List interface
  + It has the following declaration

## class ArrayList<E>

* + ArrayList supports dynamic arrays that can grow as needed whereas standard arrays are of fixed length
  + ArrayList has the following constructors
    1. ArrayList( )
       - It creates an empty array list
    2. ArrayList(Collection<? extends E> col)
       - It builds an array list that is initialized with the elements of the collection col
    3. ArrayList(int initCapacity)
       - It builds an array list that has the initial capacity passed to initCapacity

## The LinkedList Class

* It extends AbstractSequentialList and implements the List and Deque interfaces
* It provides a doubly linked list data structure
* The syntax goes like this

class LinkedList<E>

* It has two constructors

1. LinkedList( )
   * It creates an empty list
2. LinkedList(Collection<? extends E> col)
   * It builds a linked list that is initialized with the elements of the collection col

* Since it implements Deque interface, all the methods defined by Deque can be accessed by linked list class

## The HashSet Class

* It extends AbstractSet and implements the Set interface
* It creates a collection that uses a hash table for storage
* Its syntax is
* It has the following constructors

1. HashSet( )
   * Creates an empty hash set
2. HashSet(Collection<? extends E> col)
   * Creates a hash set that is initialized with the elements of the collection col
3. HashSet(int initCapacity)
   * It builds a has set that has the initial capacity passed to initCapacity
4. HashSet(int initCapacity, float fillRatio)
   * It initializes both initial capacity and fill ratio of the hash set from its arguments

## The TreeSet Class

* It extends AbstractSet and implements the NavigableSet interface
* Creates a collection that uses form of balanced, sorted binary tree for storage
* Objects are stored in sorted, ascending order
* Its syntax is

class TreeSet<E>

* It has the following constructors

1. TreeSet( )
   * Creates an empty tree set that will be sorted in ascending order
2. TreeSet(Collection<? extends E> col)
   * Creates a tree set that is initialized with the elements of the collection col
3. TreeSet(Comparator<? super E> comp)
   * Creates an empty tree set that will be sorted according to the comparator specified by comp
4. TreeSet(SortedSet<E> ss)
   * Builds a tree set that contains the elements of ss

## The LinkedHashSet Class

* It extends HashSet and does not add any members on its own
* The syntax is

class LinkedHashSet<E>

* It has the same constructors as HashSet
* It also uses hash table to store elements

## The ArrayDeque Class

* It extends AbstractCollection and implements the Deque interface
* Its syntax is

class ArrayDeque<E>

* Its constructors are

1. ArrayDeque( )
   * Creates an empty deque
2. ArrayDeque(Collection<? extends E> col)
   * It builds a deque that is initialized with the elements of the collection col
3. ArrayDeque(int initCapacity)
   * It builds a deque that has the initial capacity passed to initCapacity

## The PriorityQueue Class

* It extends AbstractQueue and implements the Queue interface
* It creates a sorted queue, with the sort order indicating the priority
* Its syntax is

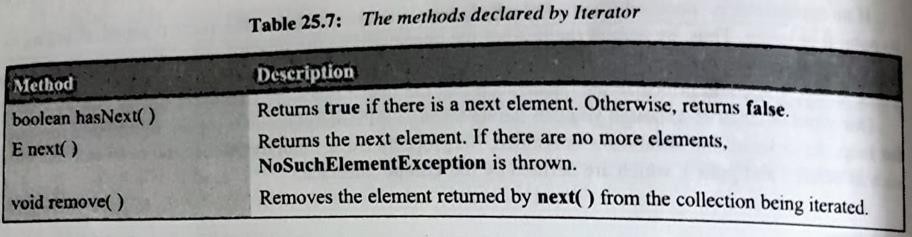
class PriorityQueue<E>

* It has the following constructors

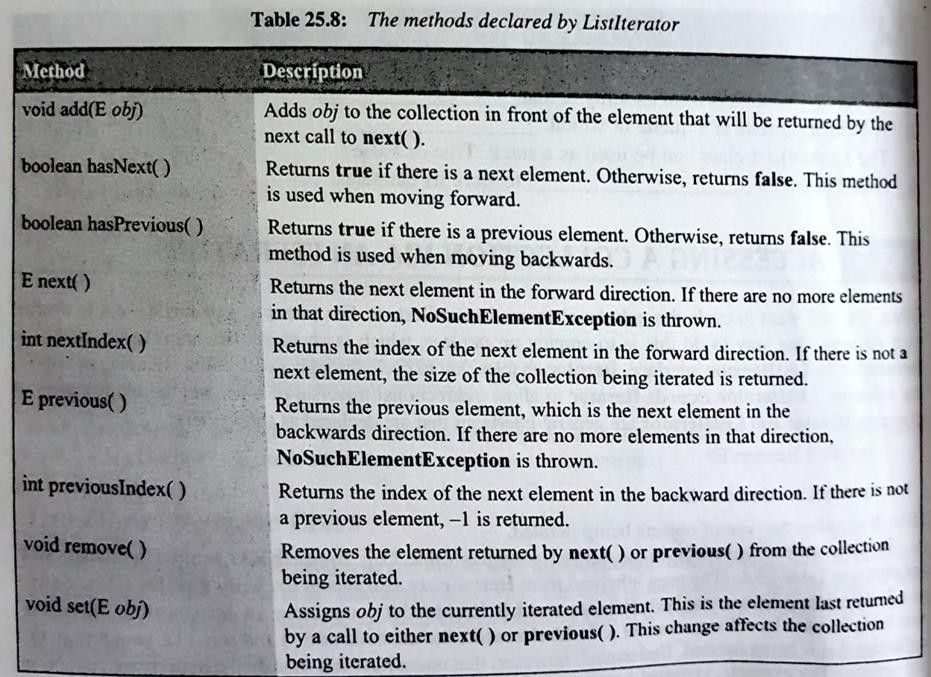
1. PriorityQueue( )
   * Creates an empty queue
2. PriorityDeque(int initCapacity)
   * It builds a queue that has the initial capacity passed to initCapacity
3. PriorityQueue(int initCapacity, Comparator<? super E> comp)
   * It creates a queue with the specified initial capacity and comparator

## Accessing A Collection via an Iterator

* + Iterator enables to cycle through a collection, obtaining or removing elements
  + Its syntax is

interface Iterator<E>

* + ListIterator extends Iterator to allow bidirectional traversal of a list and the modification of elements

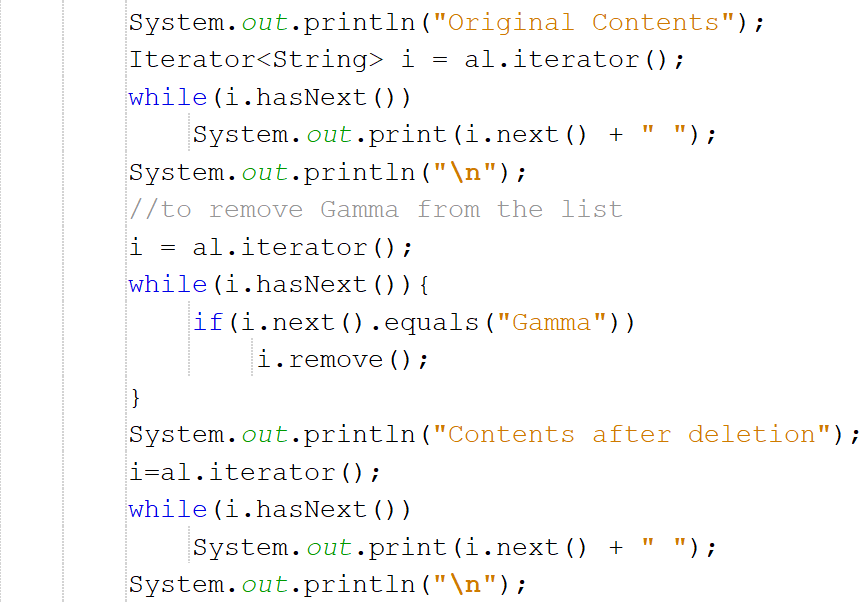
interface ListIterator<E>

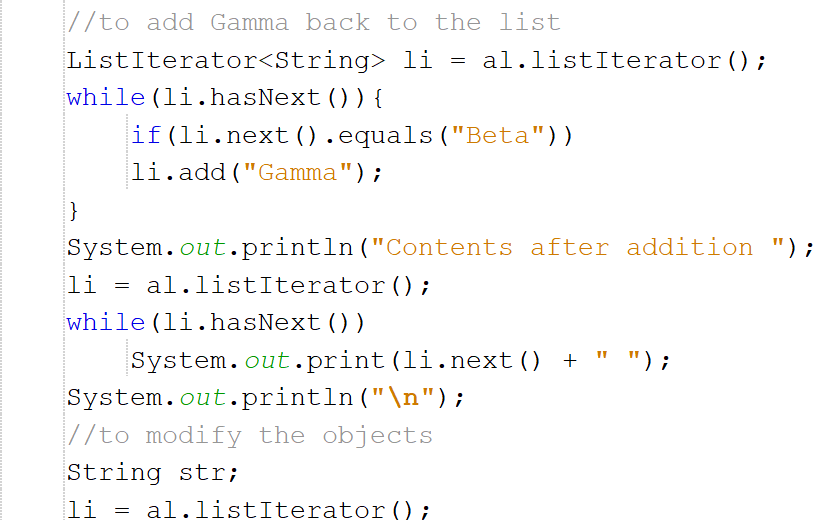
## Using and Iterator

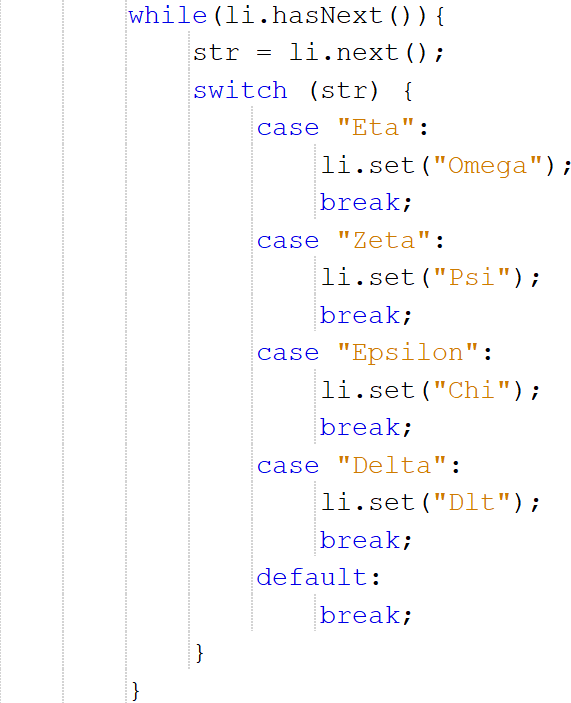
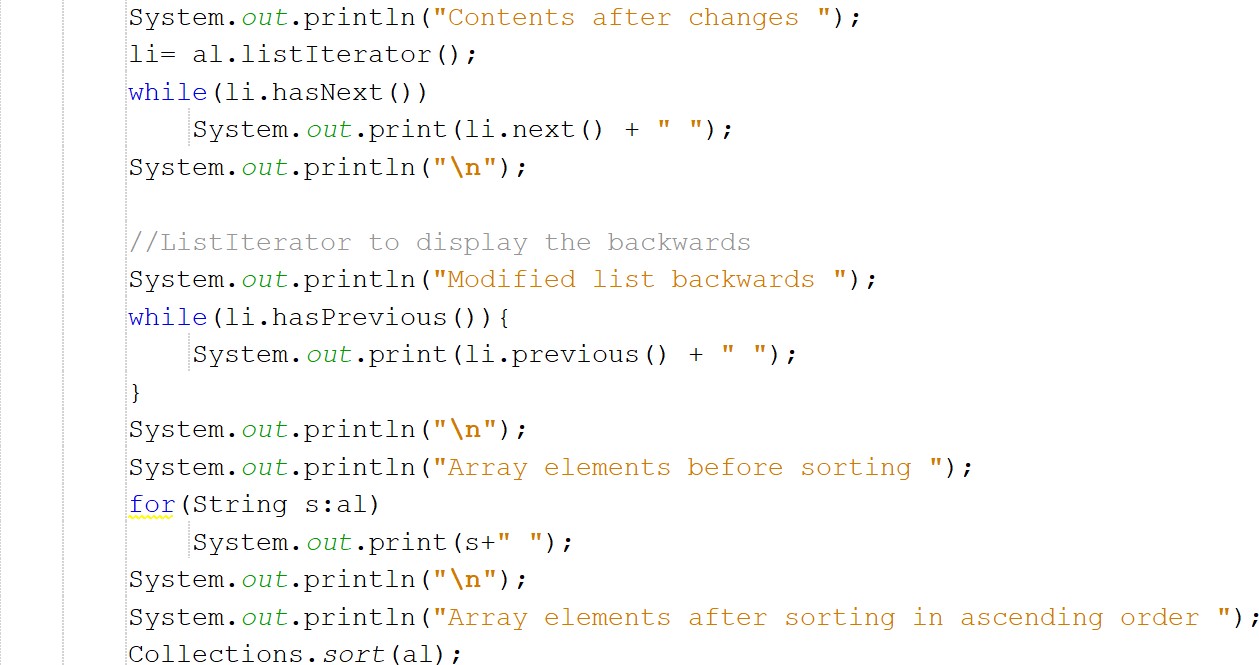
* + - Each one of the collection classes provides an iterator( ) method that returns an iterator to the start of the collection
    - Steps to use an iterator to cycle through the contents of a collection,

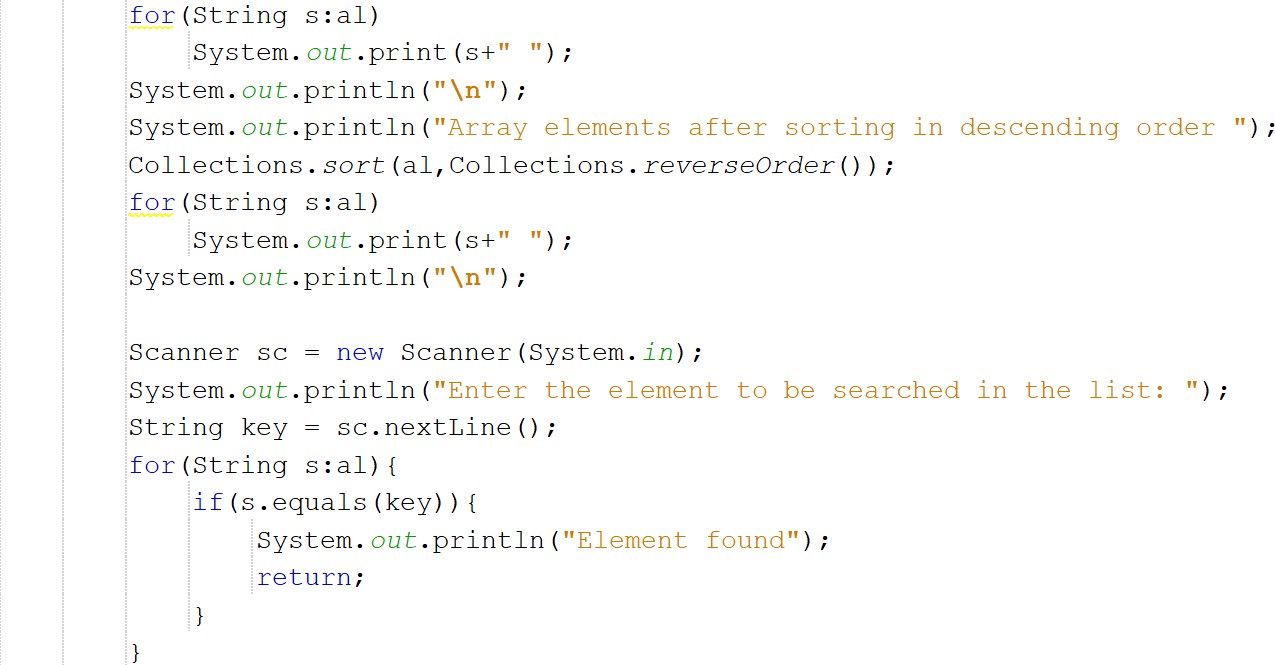
1. Obtain an itearator to the start of the collection by calling the collection’s iterator() method.
2. Set up a loop that makes a call to hasNext( ). Iterate as long as hasNext( ) returns true
3. Within the loop, obtain each element by calling next( ) Example of Iterator and ListIterator

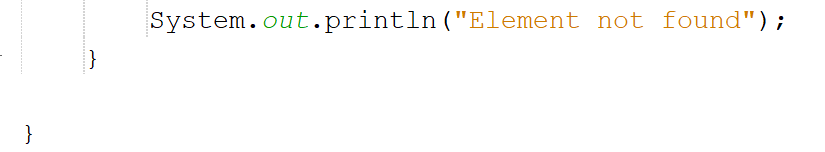




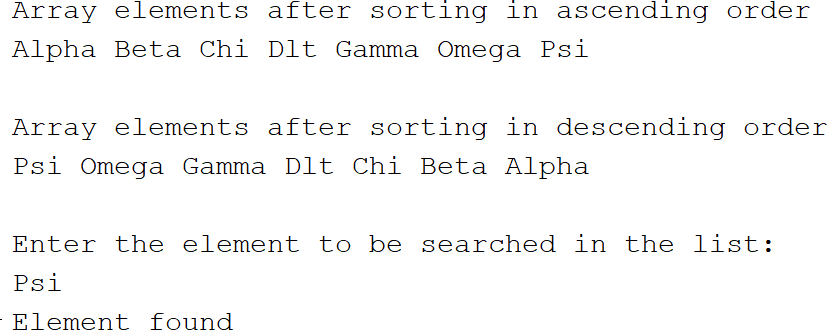
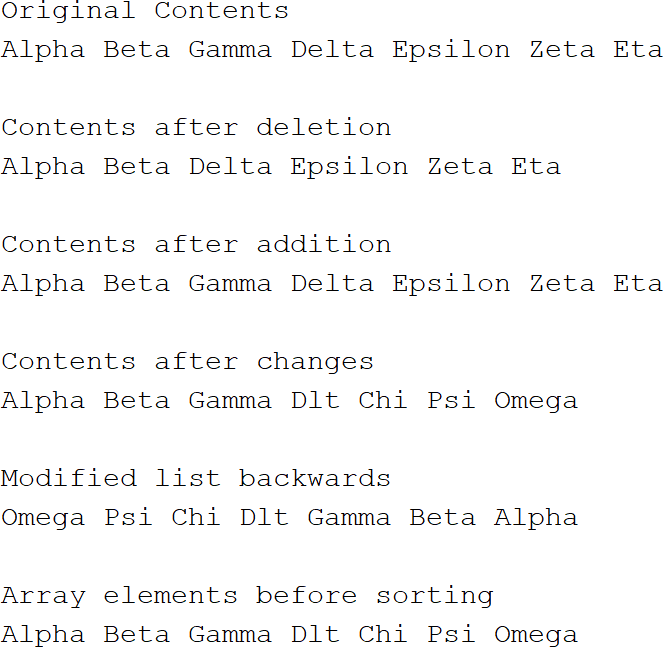








**OUTPUT:**



# Java Lambda Expressions

* + Lambda expression provides a clear and concise way to represent one method interface using an expression.
  + It is very useful in collection library. It helps to iterate, filter and extract data from collection.
  + It is used to provide the implementation of an interface which has functional interface

## Functional Interface

* + - Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface.
    - Java provides an annotation @FunctionalInterface, which is used to declare an interface as functional interface.

## Java Lambda Expression Syntax

(argument-list) -> {body}

## Types of Lambda Expression:

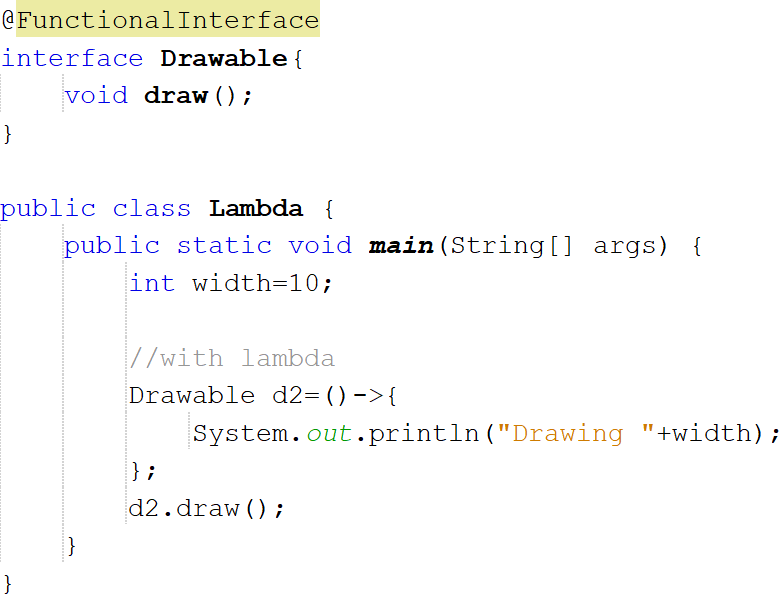
* + - There are 3 types of lambda expressions

## Zero or No Parameter

() -> {

//Body of no parameter lambda

}



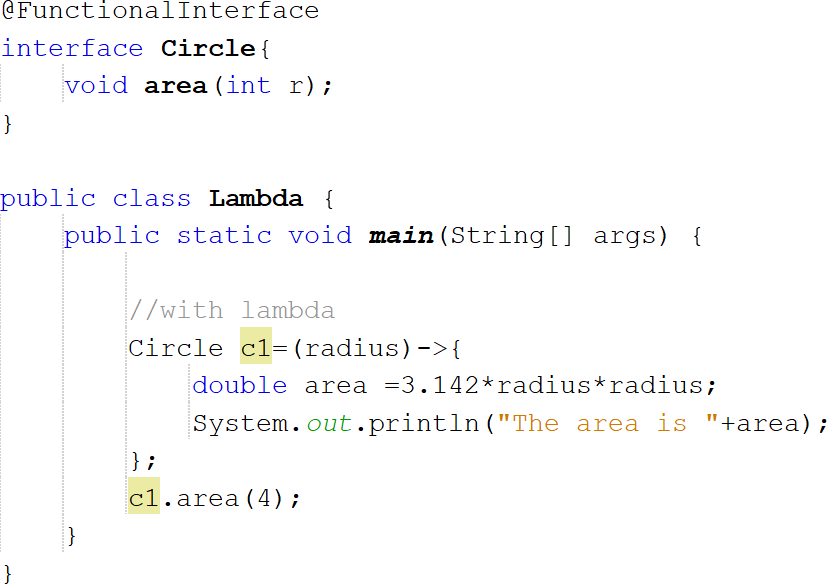
## OUTPUT:



1. **One Parameter**

(p1) -> {

//Body of single parameter lambda

}

## OUTPUT:



1. **Multiple Parameters**

(p1, p2) -> {

//Body of multiple parameter lambda

}

## Example:

**OUTPUT:**

