Topics

<https://www.geeksforgeeks.org/data-structures/>

Array

List

Stack

Queue

Set

Map

Tables

Binary Tree

Binary Search Tree

Heap

Hashing

Graph

Matrix

Misc

Advanced Data Structure

**Arrays**

Array: Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. Stores a fixed-size sequential collection of elements of the same type.

String[ ] acrs = {"Volvo", "BMW", "Ford", "Mazda"};

Access the Elements of an Array

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

Change an Array Element

To change the value of a specific element, refer to the index number:

cars[0] = "Opel";

Array Length

To find out how many elements an array has, use the length property:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);

Loop Through an Array

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (int i = 0; i < cars.length; i++) {

System.out.println(cars[i]);

}

Loop Through an Array with For-Each

There is also a "for-each" loop, which is used exclusively to loop through elements in arrays:

Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

To create a two-dimensional array, add each array within its own set of curly braces:

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

int x = myNumbers[1][2];

System.out.println(x); // Outputs 7

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes):

public class MyClass {

public static void main(String[] args) {

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

for (int i = 0; i < myNumbers.length; ++i) {

for(int j = 0; j < myNumbers[i].length; ++j) {

System.out.println(myNumbers[i][j]);

}

}

}

}

## **Processing Arrays**

When processing array elements, we often use either **for** loop or **foreach** loop because all of the elements in an array are of the same type and the size of the array is known.

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements

for (int i = 0; i < myList.length; i++) {

System.out.println(myList[i] + " ");

}

// Summing all elements

double total = 0;

for (int i = 0; i < myList.length; i++) {

total += myList[i];

}

System.out.println("Total is " + total);

// Finding the largest element

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) max = myList[i];

}

System.out.println("Max is " + max);

}

}

### **Output**

1.9

2.9

3.4

3.5

Total is 11.7

Max is 3.5

## **Passing Arrays to Methods**

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an **int** array −

### **Example**

public static void printArray(int[] array) {

for (int i = 0; i < array.length; i++) {

System.out.print(array[i] + " ");

}

}

## **Returning an Array from a Method**

A method may also return an array. For example, the following method returns an array that is the reversal of another array −

### **Example**

public static int[] reverse(int[] list) {

int[] result = new int[list.length];

for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {

result[j] = list[i];

}

return result;

}

Top 10 Methods for Java Arrays

0. Declare an array

String[] aArray = new String[5];

String[] bArray = {"a","b","c", "d", "e"};

String[] cArray = new String[]{"a","b","c","d","e"};

1. Print an array in Java

int[] intArray = { 1, 2, 3, 4, 5 };

String intArrayString = Arrays.toString(intArray);

// print directly will print reference value

System.out.println(intArray);

// [I@7150bd4d

System.out.println(intArrayString);

// [1, 2, 3, 4, 5]

2. Create an ArrayList from an array

String[] stringArray = { "a", "b", "c", "d", "e" };

ArrayList<String> arrayList = new ArrayList<String>(Arrays.asList(stringArray));

System.out.println(arrayList);

// [a, b, c, d, e]

3. Check if an array contains a certain value

String[] stringArray = { "a", "b", "c", "d", "e" };

boolean b = Arrays.asList(stringArray).contains("a");

System.out.println(b);

// true

4. Concatenate two arrays

int[] intArray = { 1, 2, 3, 4, 5 };

int[] intArray2 = { 6, 7, 8, 9, 10 };

// Apache Commons Lang library

int[] combinedIntArray = ArrayUtils.addAll(intArray, intArray2);

5. Declare an array inline

method(new String[]{"a", "b", "c", "d", "e"});

6. Joins the elements of the provided array into a single String

// containing the provided list of elements

// Apache common lang

String j = StringUtils.join(new String[] { "a", "b", "c" }, ", ");

System.out.println(j);

// a, b, c

7. Convert an ArrayList to an array

String[] stringArray = { "a", "b", "c", "d", "e" };

ArrayList<String> arrayList = new ArrayList<String>(Arrays.asList(stringArray));

String[] stringArr = new String[arrayList.size()];

arrayList.toArray(stringArr);

for (String s : stringArr)

System.out.println(s);

8. Convert an array to a set

Set<String> set = new HashSet<String>(Arrays.asList(stringArray));

System.out.println(set);

//[d, e, b, c, a]

9. Reverse an array

int[] intArray = { 1, 2, 3, 4, 5 };

ArrayUtils.reverse(intArray);

System.out.println(Arrays.toString(intArray));

//[5, 4, 3, 2, 1]

10. Remove element of an array

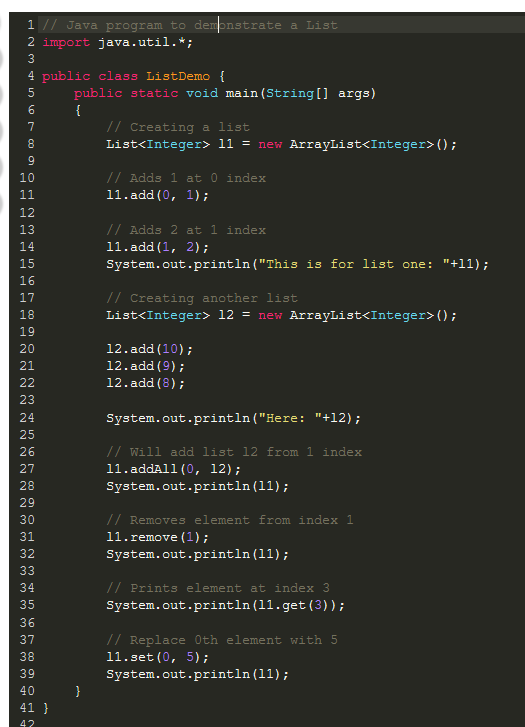
int[] intArray = { 1, 2, 3, 4, 5 };

int[] removed = ArrayUtils.removeElement(intArray, 3);//create a new array

System.out.println(Arrays.toString(removed));

List

The List interface provides a way to store the ordered collection. It is a child interface of [Collection](https://www.geeksforgeeks.org/collections-in-java-2/). It is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order, it allows positional access and insertion of elements.



Output:

This is for list one: [1, 2]

Here: [10, 9, 8]

[10, 9, 8, 1, 2]

[10, 8, 1, 2]

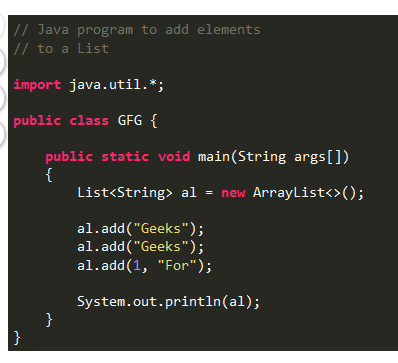
2

[5, 8, 1, 2]

## **Performing various operations using *List Interface* and** [***ArrayList class***](https://www.geeksforgeeks.org/arraylist-in-java/)

**1. Adding Elements:** In order to add an element to the list, we can use the [add() method](https://www.geeksforgeeks.org/java-util-list-add-method-java/?ref=rp). This method is overloaded to perform multiple operations based on different parameters. They are:

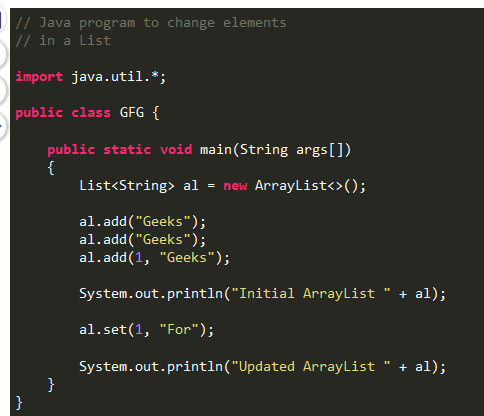
* **add(Object):** This method is used to add an element at the end of the List.
* **add(int index, Object):** This method is used to add an element at a specific index in the List.



Output;

[Geeks, For, Geeks]

**2. Changing Elements:** After adding the elements, if we wish to change the element, it can be done using the [set()](https://www.geeksforgeeks.org/abstractlist-set-method-in-java-with-examples/) method. Since List is indexed, the element which we wish to change is referenced by the index of the element. Therefore, this method takes an index and the updated element which needs to be inserted at that index.



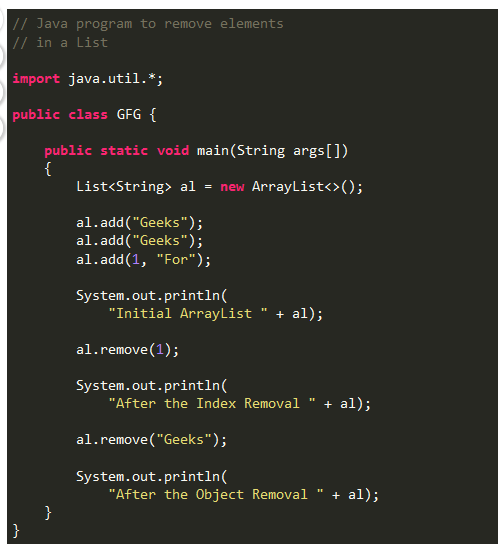
**Output:**

Initial ArrayList [Geeks, Geeks, Geeks]

Updated ArrayList [Geeks, For, Geeks]

**3. Removing Elements:** In order to remove an element from an List, we can use the [remove() method](https://www.geeksforgeeks.org/list-removeobject-obj-method-in-java-with-examples/). This method is overloaded to perform multiple operations based on different parameters. They are:

* **remove(Object):** This method is used to simply remove an object from the List. If there are multiple such objects, then the first occurrence of the object is removed.
* **remove(int index):** Since a List is indexed, this method takes an integer value which simply removes the element present at that specific index in the List. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.



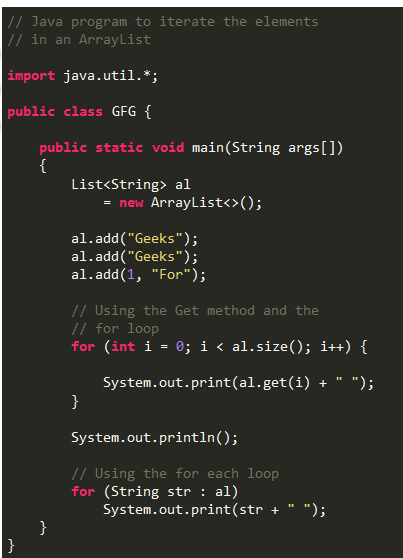
**Output:**

Initial ArrayList [Geeks, For, Geeks]

After the Index Removal [Geeks, Geeks]

After the Object Removal [Geeks]

**4. Iterating the List:** There are multiple ways to iterate through the List. The most famous ways are by using the basic [for loop](https://www.geeksforgeeks.org/loop-java-important-points/) in combination with a [get() method](https://www.geeksforgeeks.org/list-get-method-java-examples/) to get the element at a specific index and the [advanced for loop](https://www.geeksforgeeks.org/for-each-loop-in-java/).



**Output:**

Geeks For Geeks

Geeks For Geeks

**Creating List Objects**

Since **List** is an [interface](https://www.geeksforgeeks.org/interfaces-in-java/), objects cannot be created of the type list. We always need a class which extends this list in order to create an object. And also, after the introduction of [Generics](https://www.geeksforgeeks.org/generics-in-java/) in Java 1.5, it is possible to restrict the type of object that can be stored in the List. This type safe list can be defined as:

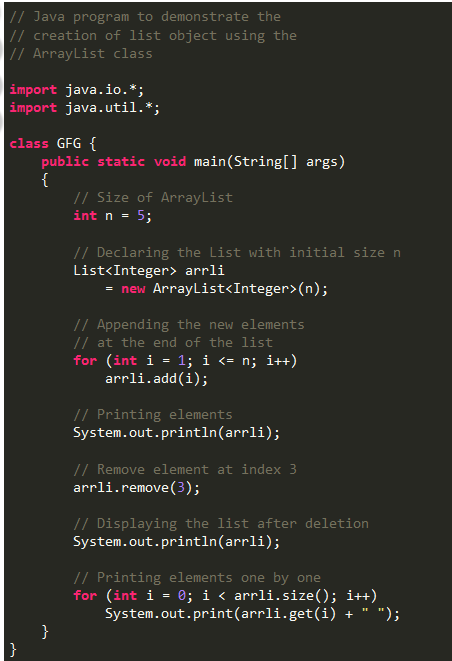
// Obj is the type of the object to be stored in List

List<Obj> list = new ArrayList<Obj> ();

## **Classes which implement the List Interface**

**1.** [**ArrayList:**](https://www.geeksforgeeks.org/arraylist-in-java/) ArrayList class which is implemented in the collection framework provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. Let’s see how to create a list object using this class.

List<Integer> arrli = new ArrayList<Integer>(n);



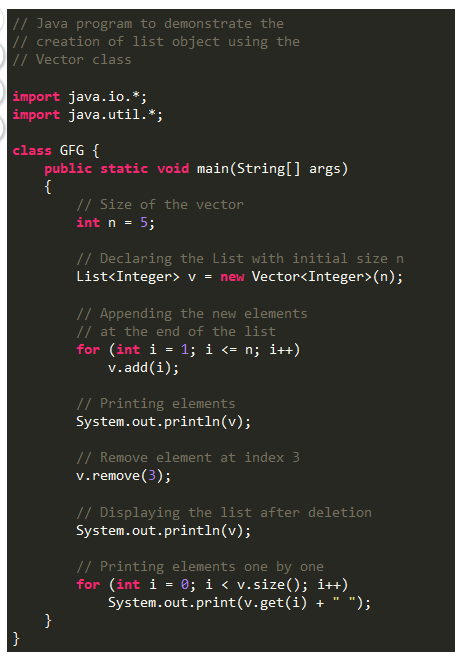
**Output:**

[1, 2, 3, 4, 5]

[1, 2, 3, 5]

1 2 3 5

[**Vector:**](https://www.geeksforgeeks.org/java-util-vector-class-java/) Vector is a class which is implemented in the collection framework implements a growable array of objects. Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index. Vectors basically fall in legacy classes but now it is fully compatible with collections. Let’s see how to create a list object using this class.



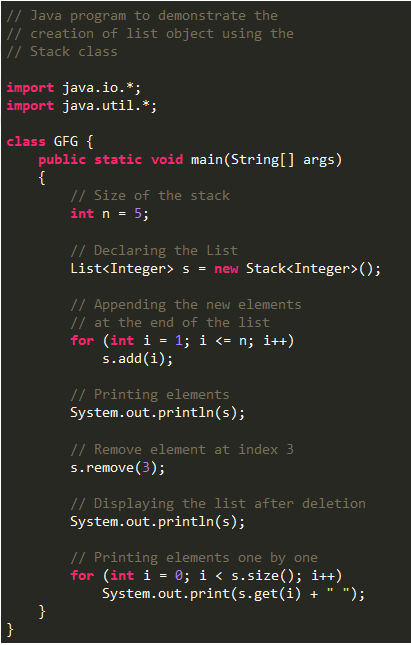
**Output:**

[1, 2, 3, 4, 5]

[1, 2, 3, 5]

1 2 3 5

[**Stack:**](https://www.geeksforgeeks.org/stack-class-in-java/) Stack is a class which is implemented in the collection framework and extends the vector class models and implements the [Stack data structure](http://www.geeksforgeeks.org/stack-data-structure/). The class is based on the basic 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. Let’s see how to create a list object using this class.



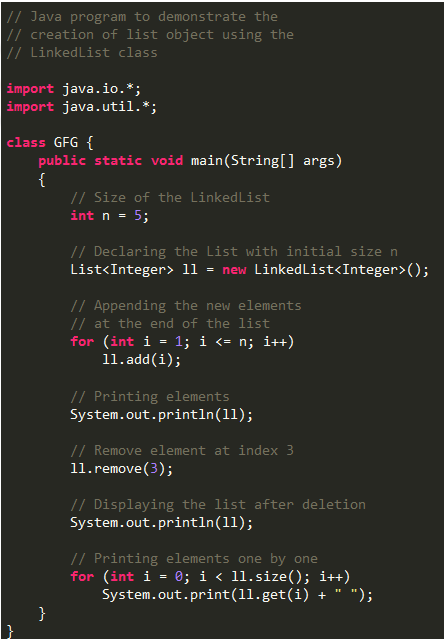
**Output:**

[1, 2, 3, 4, 5]

[1, 2, 3, 5]

1 2 3 5

[**LinkedList:**](https://www.geeksforgeeks.org/linked-list-in-java/) LinkedList is a class which is implemented in the collection framework which inherently implements the [linked list data structure](https://www.geeksforgeeks.org/data-structures/linked-list/). It is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. Let’s see how to create a list object using this class.

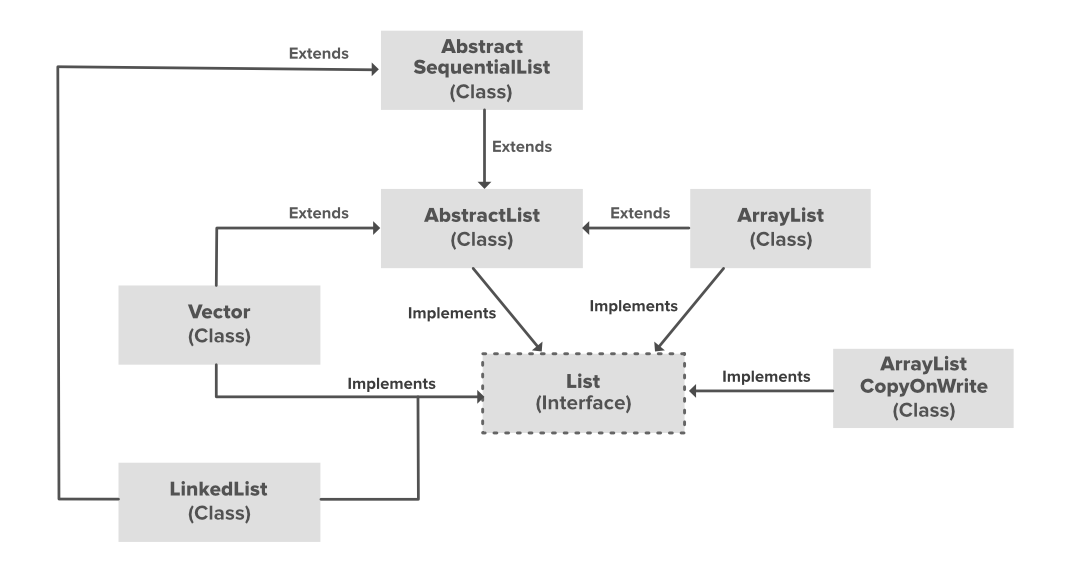


**Output:**

[1, 2, 3, 4, 5]

[1, 2, 3, 5]

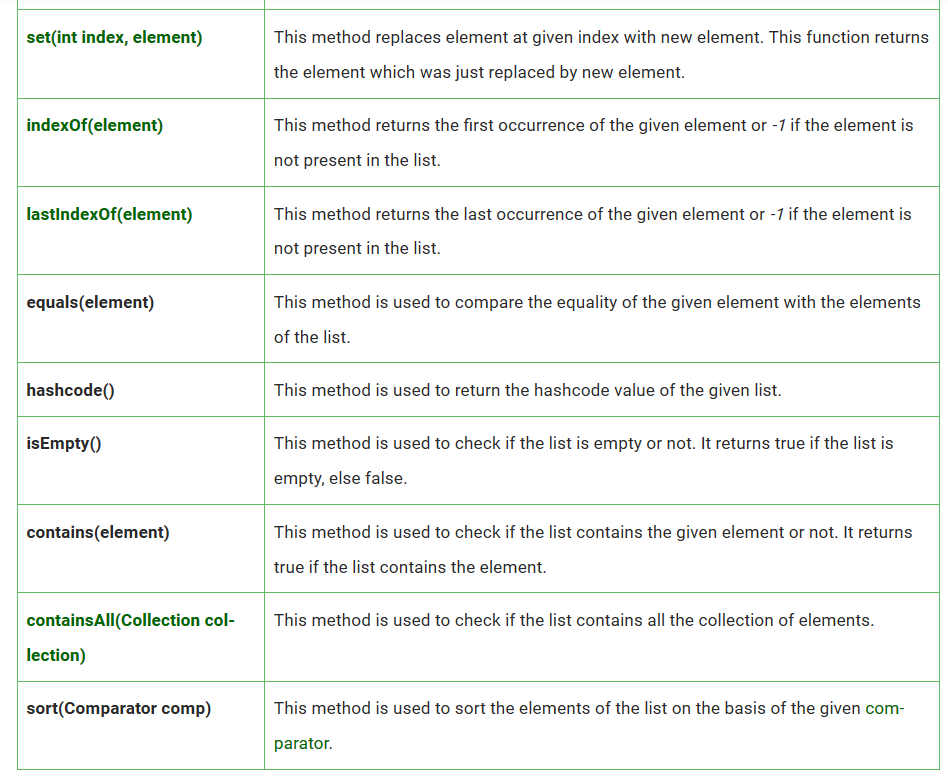
1 2 3 5



In the above illustration, [AbstractList](https://www.geeksforgeeks.org/abstractlist-in-java-with-examples/), [CopyOnWriteArrayList](https://www.geeksforgeeks.org/copyonwritearraylist-in-java/) and the [AbstractSequentialList](https://www.geeksforgeeks.org/abstractsequentiallist-in-java-with-examples/) are the classes which implement the list interface. A separate functionality is implemented in each of the mentioned classes. They are:

1. **AbstractList:** This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the *get()* and the *size()* methods.
2. **CopyOnWriteArrayList:** This class implements the list interface. It is an enhanced version of [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) in which all the modifications(add, set, remove, etc.) are implemented by making a fresh copy of the list.
3. **AbstractSequentialList:** This class implements the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) and the AbstractCollection class. This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the *get()* and the *size()* methods.



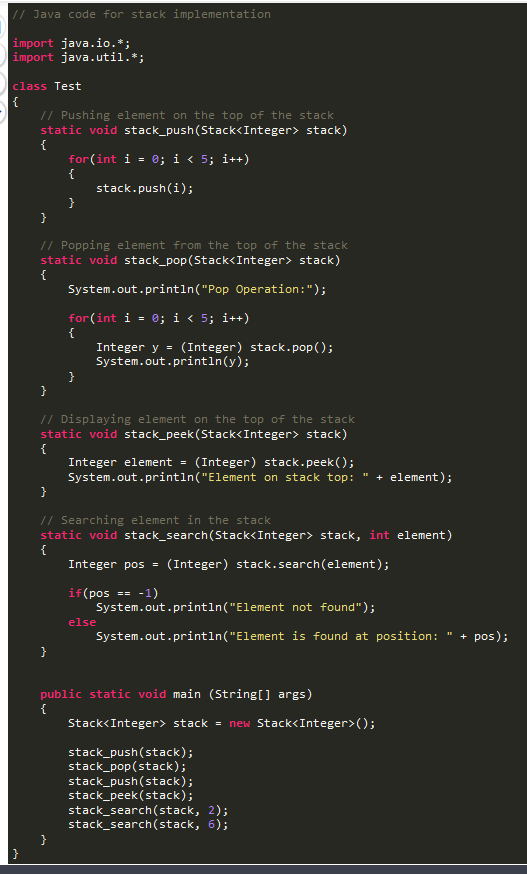


Stacks

Java [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) provides a Stack class that models and implements a [**Stack data structure**](http://www.geeksforgeeks.org/stack-data-structure/). The class is based on the basic 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 class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector. The below diagram shows the **hierarchy of the Stack class**:

### **How to Create a Stack?**

Stack<E> stack = new Stack<E>();



**Output:**

Pop Operation:

4

3

2

1

0

Element on stack top: 4

Element is found at position: 3

Element not found

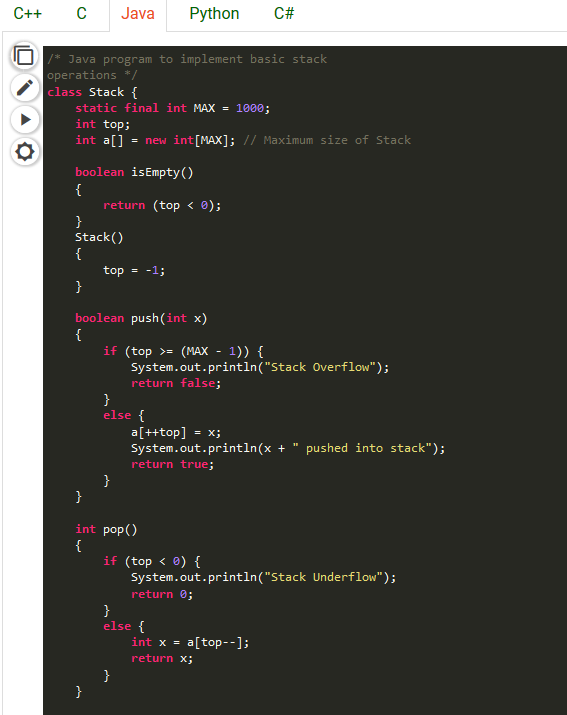
**Time Complexities of operations on stack:**

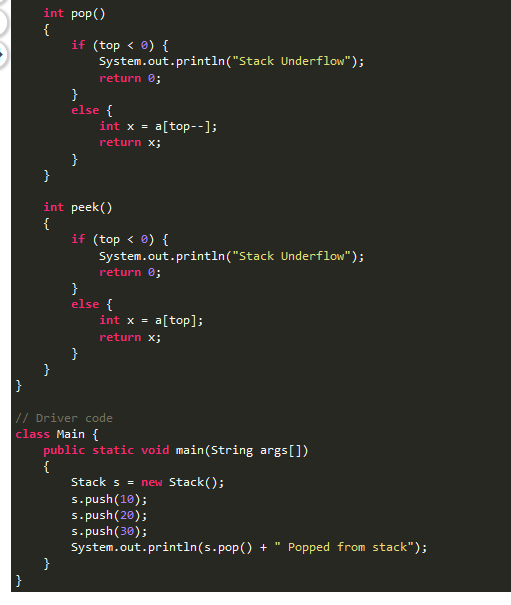
push(), pop(), isEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

**Implementation:**

There are two ways to implement a stack:

* Using array
* Using linked list





**Output :**

10 pushed into stack

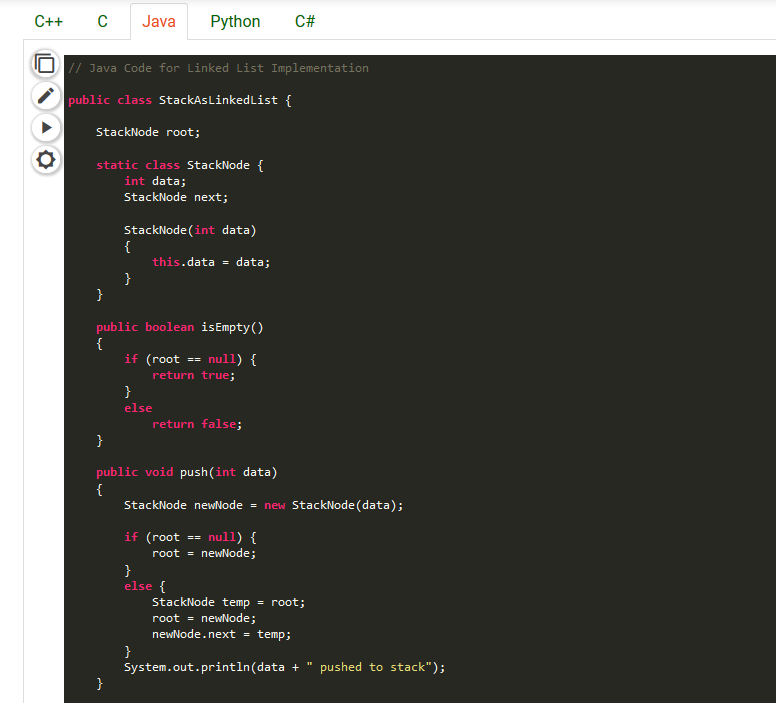
20 pushed into stack

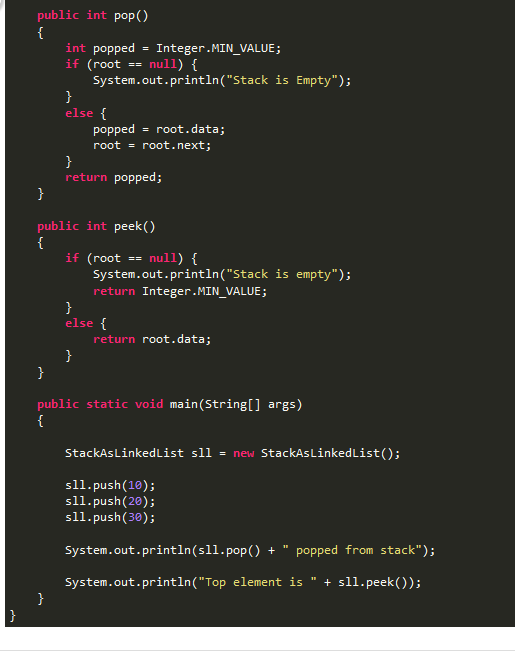
30 pushed into stack

30 popped from stack

**Pros:** Easy to implement. Memory is saved as pointers are not involved.

**Cons:** It is not dynamic. It doesn’t grow and shrink depending on needs at runtime.





Output:

10 pushed to stack

20 pushed to stack

30 pushed to stack

30 popped from stack

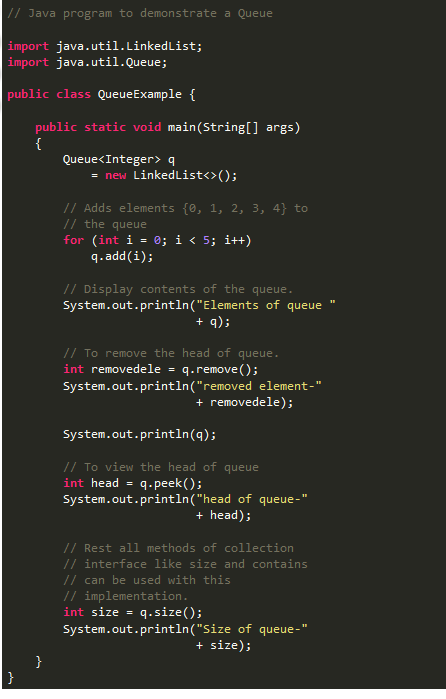
Top element is 20

**Pros:** The linked list implementation of stack can grow and shrink according to the needs at runtime.

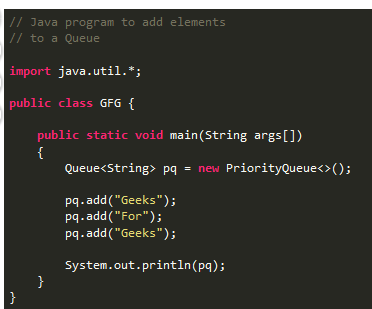
**Cons:** Requires extra memory due to involvement of pointers.

Queue

The Queue interface present in the [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) is used to hold the elements about to be processed in FIFO(First In First Out) order. 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 the list, (i.e.), it follows the FIFO or the First-In-First-Out principle.



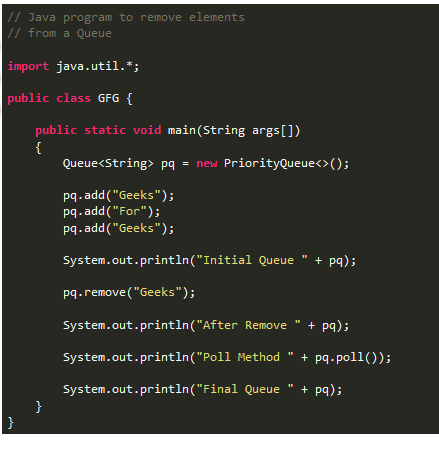
**1. Adding Elements:** In order to add an element in a queue, we can use the [add() method](https://www.geeksforgeeks.org/queue-add-method-in-java/). The insertion order is not retained in the PriorityQueue. The elements are stored based on the priority order which is ascending by default.



Output:

[For, Geeks, Geeks]

**2. Removing Elements:** In order to remove an element from a queue, we can use the [remove() method.](https://www.geeksforgeeks.org/queue-remove-method-in-java/) If there are multiple such objects, then the first occurrence of the object is removed. Apart from that, poll() method is also used to remove the head and return it.



**Output:**

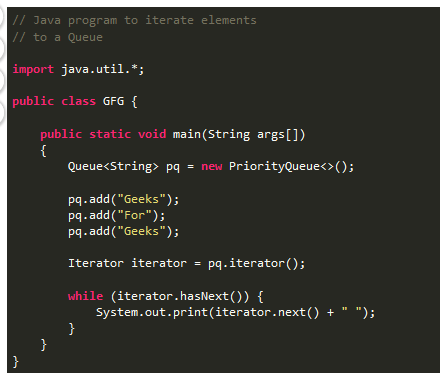
Initial Queue [For, Geeks, Geeks]

After Remove [For, Geeks]

Poll Method For

Final Queue [Geeks]

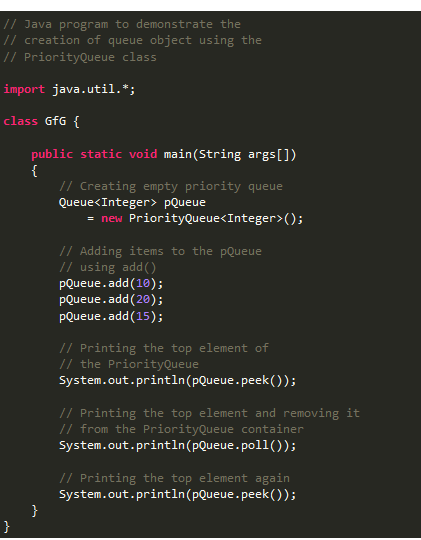
**3. Iterating the Queue:** There are multiple ways to iterate through the Queue. The most famous way is converting the queue to the array and traversing using the for loop. However, the queue also has an inbuilt iterator which can be used to iterate through the queue.



**Output:**

For Geeks Geeks

[**1. PriorityQueue:**](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/) PriorityQueue class which is implemented in the collection framework provides us a way to process the objects 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. Let’s see how to create a queue object using this class.



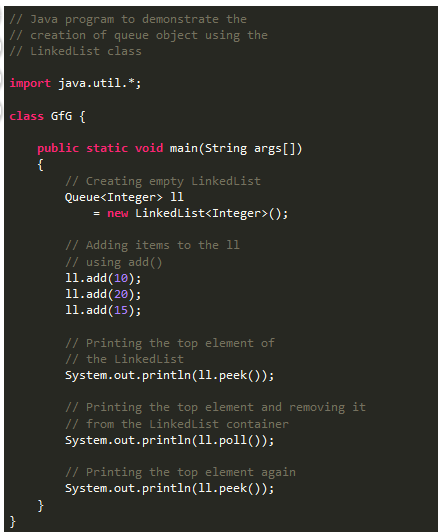
**Output:**

10

10

15

**2.** [**LinkedList:**](https://www.geeksforgeeks.org/linked-list-in-java/) LinkedList is a class which is implemented in the collection framework which inherently implements the [linked list data structure](https://www.geeksforgeeks.org/data-structures/linked-list/). It is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays or queues. Let’s see how to create a queue object using this class.

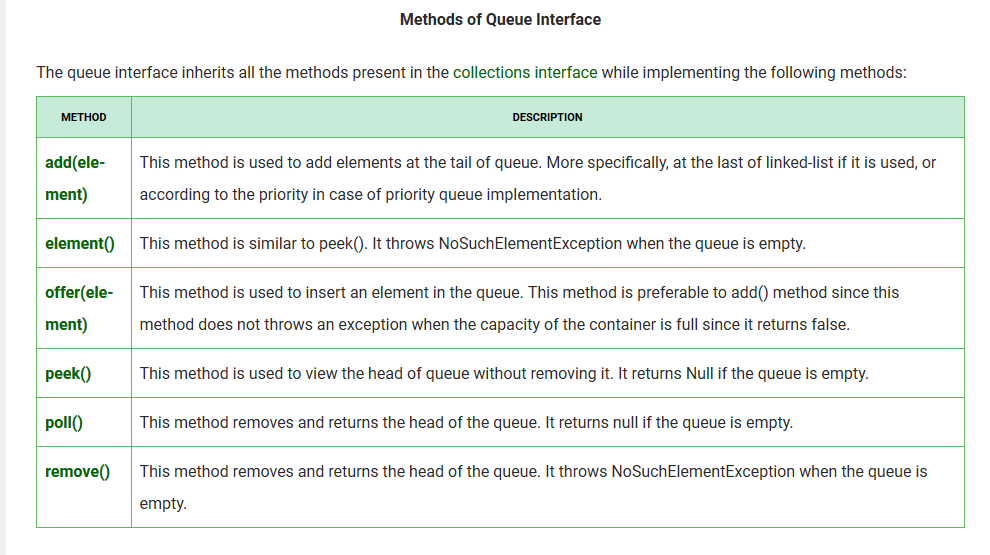


**Output:**

10

10

20



# **Set**

Which duplicate values cannot be stored



**Output:**

[Set, Example, Geeks, For]

**Creating Set Objects**

Since Set is an [interface](https://www.geeksforgeeks.org/interfaces-in-java/), objects cannot be created of the type set. We always need a class which extends this list in order to create an object. And also, after the introduction of [Generics](https://www.geeksforgeeks.org/generics-in-java/) in Java 1.5, it is possible to restrict the type of object that can be stored in the Set. This type-safe set can be defined as:

// Obj is the type of the object to be stored in Set

Set<Obj> set = new HashSet<Obj> ();