LinkedList in Java

Linked List is a part of the Collection framework present in java.util package. This class is an implementation of the LinkedList data structure which 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. It also has a few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach a node we wish to access.

How Does LinkedList work Internally?

Since a LinkedList acts as a dynamic array and we do not have to specify the size while creating it, the size of the list automatically increases when we dynamically add and remove items. And also, the elements are not stored in a continuous fashion. Therefore, there is no need to increase the size. Internally, the LinkedList is implemented using the doubly linked list data structure.
The main difference between a normal linked list and a doubly LinkedList is that a doubly linked list contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.

Constructors in the LinkedList:

In order to create a LinkedList, we need to create an object of the LinkedList class. The LinkedList class consists of various constructors that allow the possible creation of the list. The following are the constructors available in this class:

1. **LinkedList():** This constructor is used to create an empty linked list. If we wish to create an empty LinkedList with the name ll, then, it can be created as:
LinkedList ll = new LinkedList();
2. **LinkedList(Collection C):** This constructor is used to create an ordered list that contains all the elements of a specified collection, as returned by the collection's iterator. If we wish to create a LinkedList with the name ll, then, it can be created as:
LinkedList ll = new LinkedList(C);

Methods for Java LinkedList:

Method	Description
add(int index, E element)	This method Inserts the specified element at the specified position in this list.
add(E e)	This method Appends the specified element to the end of this list.
addAll(int index, Collection<E> c)	This method Inserts all of the elements in the specified collection into this list, starting at the specified position.
addAll(Collection<E> c)	This method Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
addFirst(E e)	This method Inserts the specified element at the beginning of this list.
addLast(E e)	This method Appends the specified element to the end of this list.
clear()	This method removes all of the elements from this list.
clone()	This method returns a shallow copy of this LinkedList.

contains(Object o)	This method returns true if this list contains the specified element.
descendingIterator()	This method returns an iterator over the elements in this deque in reverse sequential order.
element()	This method retrieves but does not remove, the head (first element) of this list.
get(int index)	This method returns the element at the specified position in this list.
getFirst()	This method returns the first element in this list.
getLast()	This method returns the last element in this list.
indexOf(Object o)	This method returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
lastIndexOf(Object o)	This method returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
listIterator(int index)	This method returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
offer(E e)	This method Adds the specified element as the tail (last element) of this list.
offerFirst(E e)	This method Inserts the specified element at the front of this list.
offerLast(E e)	This method Inserts the specified element at the end of this list.
peek()	This method retrieves but does not remove, the head (first element) of this list.
peekFirst()	This method retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
peekLast()	This method retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
poll()	This method retrieves and removes the head (first element) of this list.
pollFirst()	This method retrieves and removes the first element of this list, or returns null if this list is empty.
pollLast()	This method retrieves and removes the last element of this list, or returns null if this list is empty.
pop()	This method Pops an element from the stack represented by this list.
push(E e)	This method pushes an element onto the stack represented by this list.
remove()	This method retrieves and removes the head (first element) of this list.
remove(int index)	This method removes the element at the specified position in this list.
remove(Object o)	This method removes the first occurrence of the specified element from this list if it is present.
removeFirst()	This method removes and returns the first element from this list.
removeFirstOccurrence(Object o)	This method removes the first occurrence of the specified element in this list (when traversing the list from head to tail).
removeLast()	This method removes and returns the last element from this list.
removeLastOccurrence(Object o)	This method removes the last occurrence of the specified element in this list (when traversing the list from head to tail).
set(int index, E element)	This method replaces the element at the specified position in this list with the specified element.
size()	This method returns the number of elements in this list.
spliterator()	This method creates a late-binding and fail-fast Spliterator over the elements in this list.
toArray()	This method returns an array containing all of the elements in this list in proper sequence (from first to last element).

toArray(T[] a)	This method returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
toString()	This method returns a string containing all of the elements in this list in proper sequence (from first to the last element), each element is separated by commas and the String is enclosed in square brackets.

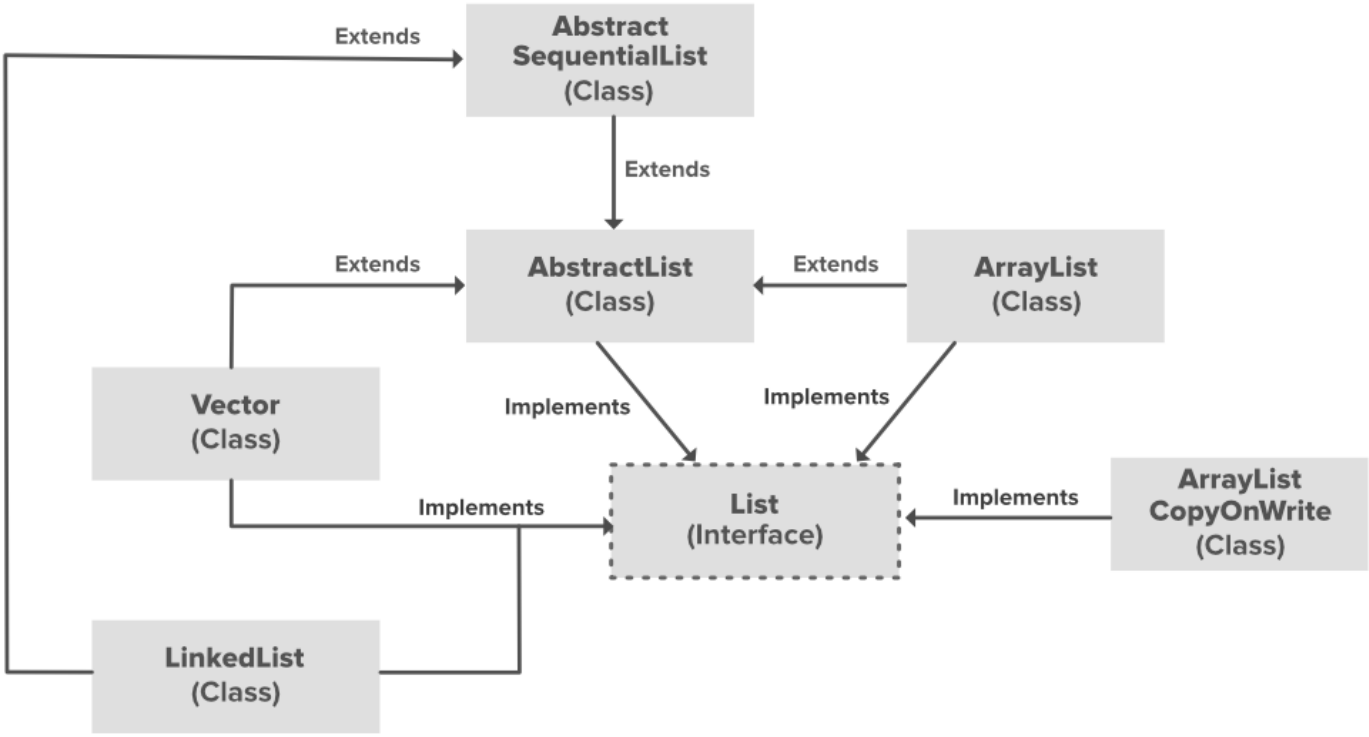
Below is the implementation of the above operations:

- Java

```
// Java Program to Demonstrate
// Implementation of LinkedList
// class

// Importing required classes
import java.util.*;

// Main class
public class GFG {

    // Driver code
    public static void main(String args[])
    {
        // Creating object of the
        // class linked list
        LinkedList<String> ll = new LinkedList<String>();

        // Adding elements to the linked list
        ll.add("A");
        ll.add("B");
        ll.addLast("C");
        ll.addFirst("D");
        ll.add(2, "E");

        System.out.println(ll);

        ll.remove("B");
        ll.remove(3);
        ll.removeFirst();
        ll.removeLast();

        System.out.println(ll);
    }
}
```

Output:

```
[D, A, E, B, C]
[A]
```



In the above illustration, AbstractList, CopyOnWriteArrayList, and AbstractSequentialList are the classes that implement the list interface. A separate functionality is implemented in each of the mentioned classes. They are:

- **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.
- **CopyOnWriteArrayList:** This class implements the list interface. It is an enhanced version of ArrayList in which all the modifications(add, set, remove, etc.) are implemented by making a fresh copy of the list.

Performing Various Operations on LinkedList:

- Adding elements
- Updating elements
- Removing elements
- Iterating over elements

Operation 1: Adding Elements

In order to add an element to an ArrayList, we can use the add() method. 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 LinkedList.
- **add(int index, Object):** This method is used to add an element at a specific index in the LinkedList.

Below is the implementation of the above operation:

- Java

```
// Java program to add elements
```

```
// to a LinkedList

import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");
        ll.add("Geeks");
        ll.add(1, "For");

        System.out.println(ll);
    }
}
```

Output

```
[Geeks, For, Geeks]
```

Operation 2: Changing Elements

After adding the elements, if we wish to change the element, it can be done using the set() method. Since a LinkedList 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.
Below is the implementation of the above operation:

- Java

```
// Java program to change elements
// in a LinkedList

import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");
        ll.add("Geeks");
        ll.add(1, "Geeks");

        System.out.println("Initial LinkedList " + ll);

        ll.set(1, "For");

        System.out.println("Updated LinkedList " + ll);
    }
}
```

Output

```
Initial LinkedList [Geeks, Geeks, Geeks]
Updated LinkedList [Geeks, For, Geeks]
```

Operation 3: Removing Elements

In order to remove an element from a LinkedList, we can use the remove() method. 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 LinkedList. If there are multiple such objects, then the first occurrence of the object is removed.
- **remove(int index):** Since a LinkedList is indexed, this method takes an integer value which simply removes the element present at that specific index in the LinkedList. After removing the element and the indices of elements are updated so do the object of LinkedList is updated giving a new List after the deletion of element/s.

Below is the implementation of the above operation:

- Java

```
// Java program to remove elements
// in a LinkedList

import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");
        ll.add("Geeks");
        ll.add(1, "For");

        System.out.println("Initial LinkedList " + ll);

          // Function call
        ll.remove(1);

        System.out.println("After the Index Removal " + ll);

        ll.remove("Geeks");

        System.out.println("After the Object Removal "
                           + ll);
    }
}
```

Output

```
Initial LinkedList [Geeks, For, Geeks]
After the Index Removal [Geeks, Geeks]
After the Object Removal [Geeks]
```

Operation 4: Iterating the LinkedList

There are multiple ways to iterate through LinkedList. The most famous ways are by using the basic for loop in combination with a get() method to get the element at a specific index and the advanced for-loop.

Below is the implementation of the above operation:

- Java

```
// Java program to iterate the elements
// in an LinkedList

import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll
            = new LinkedList<>();

        ll.add("Geeks");
        ll.add("Geeks");
        ll.add(1, "For");

        // Using the Get method and the
        // for loop
        for (int i = 0; i < ll.size(); i++) {

            System.out.print(ll.get(i) + " ");
        }

        System.out.println();

        // Using the for each loop
        for (String str : ll)
            System.out.print(str + " ");
    }
}
```

Output

```
Geeks For Geeks
Geeks For Geeks
```