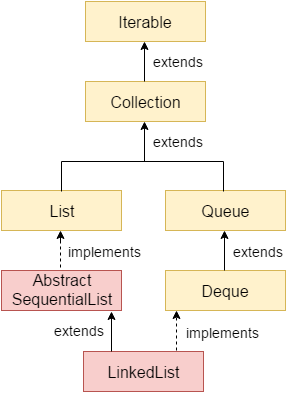
**Java LinkedList class**



Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

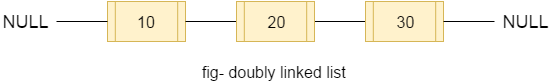
* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to occur.
* Java LinkedList class can be used as a list, stack or queue.

**Hierarchy of LinkedList class**

As shown in the above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

**Doubly Linked List**

In the case of a doubly linked list, we can add or remove elements from both sides.



**LinkedList class declaration**

Let's see the declaration for java.util.LinkedList class.

1. public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable

**Constructors of Java LinkedList**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| LinkedList() | It is used to construct an empty list. |
| LinkedList(Collection<? extends E> c) | It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator. |

**Methods of Java LinkedList**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(E e) | It is used to append the specified element to the end of a list. |
| void add(int index, E element) | It is used to insert the specified element at the specified position index in a list. |
| boolean addAll(Collection<? extends E> c) | It is used to append 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. |
| boolean addAll(Collection<? extends E> c) | It is used to append 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. |
| boolean addAll(int index, Collection<? extends E> c) | It is used to append all the elements in the specified collection, starting at the specified position of the list. |
| void addFirst(E e) | It is used to insert the given element at the beginning of a list. |
| void addLast(E e) | It is used to append the given element to the end of a list. |
| void clear() | It is used to remove all the elements from a list. |
| Object clone() | It is used to return a shallow copy of an ArrayList. |
| boolean contains(Object o) | It is used to return true if a list contains a specified element. |
| Iterator<E> descendingIterator() | It is used to return an iterator over the elements in a deque in reverse sequential order. |
| E element() | It is used to retrieve the first element of a list. |
| E get(int index) | It is used to return the element at the specified position in a list. |
| E getFirst() | It is used to return the first element in a list. |
| E getLast() | It is used to return the last element in a list. |
| int indexOf(Object o) | It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element. |
| int lastIndexOf(Object o) | It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element. |
| ListIterator<E> listIterator(int index) | It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list. |
| boolean offer(E e) | It adds the specified element as the last element of a list. |
| boolean offerFirst(E e) | It inserts the specified element at the front of a list. |
| boolean offerLast(E e) | It inserts the specified element at the end of a list. |
| E peek() | It retrieves the first element of a list |
| E peekFirst() | It retrieves the first element of a list or returns null if a list is empty. |
| E peekLast() | It retrieves the last element of a list or returns null if a list is empty. |
| E poll() | It retrieves and removes the first element of a list. |
| E pollFirst() | It retrieves and removes the first element of a list, or returns null if a list is empty. |
| E pollLast() | It retrieves and removes the last element of a list, or returns null if a list is empty. |
| E pop() | It pops an element from the stack represented by a list. |
| void push(E e) | It pushes an element onto the stack represented by a list. |
| E remove() | It is used to retrieve and removes the first element of a list. |
| E remove(int index) | It is used to remove the element at the specified position in a list. |
| boolean remove(Object o) | It is used to remove the first occurrence of the specified element in a list. |
| E removeFirst() | It removes and returns the first element from a list. |
| boolean removeFirstOccurrence(Object o) | It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail). |
| E removeLast() | It removes and returns the last element from a list. |
| boolean removeLastOccurrence(Object o) | It removes the last occurrence of the specified element in a list (when traversing the list from head to tail). |
| E set(int index, E element) | It replaces the element at the specified position in a list with the specified element. |
| Object[] toArray() | It is used to return an array containing all the elements in a list in proper sequence (from first to the last element). |
| <T> T[] toArray(T[] a) | It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array. |
| int size() | It is used to return the number of elements in a list. |

**Java LinkedList Example**

1. import java.util.\*;
2. public class LinkedList1{
3. public static void main(String args[]){
5. LinkedList<String> al=new LinkedList<String>();
6. al.add("Ravi");
7. al.add("Vijay");
8. al.add("Ravi");
9. al.add("Ajay");
11. Iterator<String> itr=al.iterator();
12. while(itr.hasNext()){
13. System.out.println(itr.next());
14. }
15. }
16. }

Output: Ravi

Vijay

Ravi

Ajay

**Java LinkedList example to add elements**

Here, we see different ways to add elements.

1. import java.util.\*;
2. public class LinkedList2{
3. public static void main(String args[]){
4. LinkedList<String> ll=new LinkedList<String>();
5. System.out.println("Initial list of elements: "+ll);
6. ll.add("Ravi");
7. ll.add("Vijay");
8. ll.add("Ajay");
9. System.out.println("After invoking add(E e) method: "+ll);
10. //Adding an element at the specific position
11. ll.add(1, "Gaurav");
12. System.out.println("After invoking add(int index, E element) method: "+ll);
13. LinkedList<String> ll2=new LinkedList<String>();
14. ll2.add("Sonoo");
15. ll2.add("Hanumat");
16. //Adding second list elements to the first list
17. ll.addAll(ll2);
18. System.out.println("After invoking addAll(Collection<? extends E> c) method: "+ll);
19. LinkedList<String> ll3=new LinkedList<String>();
20. ll3.add("John");
21. ll3.add("Rahul");
22. //Adding second list elements to the first list at specific position
23. ll.addAll(1, ll3);
24. System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+ll);
25. //Adding an element at the first position
26. ll.addFirst("Lokesh");
27. System.out.println("After invoking addFirst(E e) method: "+ll);
28. //Adding an element at the last position
29. ll.addLast("Harsh");
30. System.out.println("After invoking addLast(E e) method: "+ll);
32. }
33. }

Initial list of elements: []

After invoking add(E e) method: [Ravi, Vijay, Ajay]

After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]

After invoking addAll(Collection<? extends E> c) method:

[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]

After invoking addAll(int index, Collection<? extends E> c) method:

[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]

After invoking addFirst(E e) method:

[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]

After invoking addLast(E e) method:

[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat, Harsh]

**Java LinkedList example to remove elements**

Here, we see different ways to remove an element.

import java.util.\*;

public class LinkedList3 {

        public static void main(String [] args)

        {

           LinkedList<String> ll=new LinkedList<String>();

           ll.add("Ravi");

           ll.add("Vijay");

           ll.add("Ajay");

           ll.add("Anuj");

           ll.add("Gaurav");

           ll.add("Harsh");

           ll.add("Virat");

           ll.add("Gaurav");

           ll.add("Harsh");

           ll.add("Amit");

           System.out.println("Initial list of elements: "+ll);

         //Removing specific element from arraylist

              ll.remove("Vijay");

              System.out.println("After invoking remove(object) method: "+ll);

         //Removing element on the basis of specific position

              ll.remove(0);

              System.out.println("After invoking remove(index) method: "+ll);

              LinkedList<String> ll2=new LinkedList<String>();

              ll2.add("Ravi");

              ll2.add("Hanumat");

         // Adding new elements to arraylist

              ll.addAll(ll2);

              System.out.println("Updated list : "+ll);

         //Removing all the new elements from arraylist

              ll.removeAll(ll2);

              System.out.println("After invoking removeAll() method: "+ll);

         //Removing first element from the list

              ll.removeFirst();

              System.out.println("After invoking removeFirst() method: "+ll);

          //Removing first element from the list

              ll.removeLast();

              System.out.println("After invoking removeLast() method: "+ll);

          //Removing first occurrence of element from the list

              ll.removeFirstOccurrence("Gaurav");

              System.out.println("After invoking removeFirstOccurrence() method: "+ll);

          //Removing last occurrence of element from the list

              ll.removeLastOccurrence("Harsh");

              System.out.println("After invoking removeLastOccurrence() method: "+ll);

              //Removing all the elements available in the list

              ll.clear();

              System.out.println("After invoking clear() method: "+ll);

       }

    }

Initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]

After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]

After invoking remove(index) method: [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]

Updated list : [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit, Ravi, Hanumat]

After invoking removeAll() method: [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]

After invoking removeFirst() method: [Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]

After invoking removeLast() method: [Gaurav, Harsh, Virat, Gaurav, Harsh]

After invoking removeFirstOccurrence() method: [Harsh, Virat, Gaurav, Harsh]

After invoking removeLastOccurrence() method: [Harsh, Virat, Gaurav]

After invoking clear() method: []

**Java LinkedList Example to reverse a list of elements**

1. import java.util.\*;
2. public class LinkedList4{
3. public static void main(String args[]){
5. LinkedList<String> ll=new LinkedList<String>();
6. ll.add("Ravi");
7. ll.add("Vijay");
8. ll.add("Ajay");
9. //Traversing the list of elements in reverse order
10. Iterator i=ll.descendingIterator();
11. while(i.hasNext())
12. {
13. System.out.println(i.next());
14. }
16. }
17. }

Output: Ajay

Vijay

Ravi

**Java LinkedList Example: Book**

1. import java.util.\*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7. this.id = id;
8. this.name = name;
9. this.author = author;
10. this.publisher = publisher;
11. this.quantity = quantity;
12. }
13. }
14. public class LinkedListExample {
15. public static void main(String[] args) {
16. //Creating list of Books
17. List<Book> list=new LinkedList<Book>();
18. //Creating Books
19. Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20. Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
21. Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22. //Adding Books to list
23. list.add(b1);
24. list.add(b2);
25. list.add(b3);
26. //Traversing list
27. for(Book b:list){
28. System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
29. }
30. }
31. }

Output:

101 Let us C Yashwant Kanetkar BPB 8

102 Data Communications & Networking Forouzan Mc Graw Hill 4

103 Operating System Galvin Wiley 6

**Difference between ArrayList and LinkedList**

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

However, there are many differences between ArrayList and LinkedList classes that are given below.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |