HowToDoInJava

Java LinkedList class

Last Updated: August 30, 2020 By: Lokesh Gupta Java Collections Java LinkedList

Java LinkedList class is doubly-linked list implementation of the `List` and `Deque` interfaces. It implements all optional list operations, and permits all elements (including null).

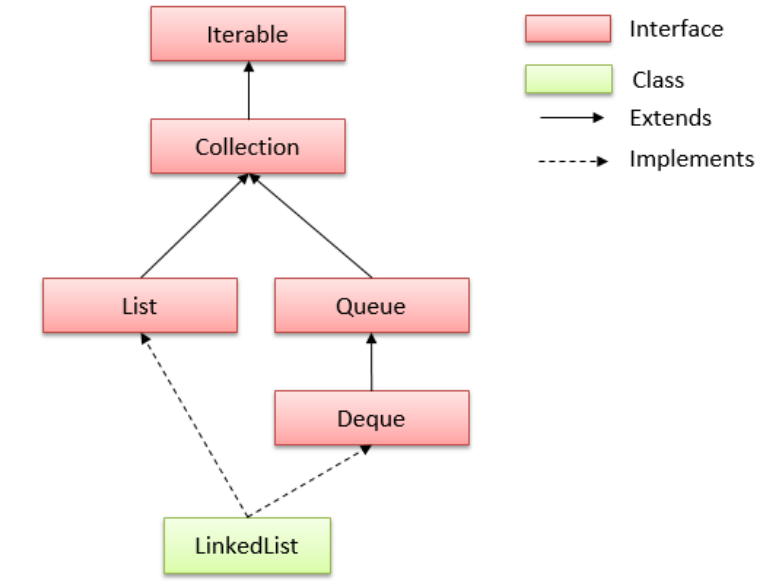
Table of Contents

1. LinkedList Hierarchy

The LinkedList class **extends AbstractSequentialList** class and **implements List and Deque** interfaces. Here `'E'` is the type of values linkedlist stores.

```
LinkedList.java

public class LinkedList<E>
    extends AbstractSequentialList<E>
    implements List<E>, Deque<E>, Cloneable, java.io.Serializable
{
  //implementation
}
```



LinkedList Hierarchy

2. LinkedList Features

- **Doubly linked list** implementation which implements List and Deque interfaces. Therefore, It can also be used as a Queue, Deque or Stack.
- Permits all elements including duplicates and NULL.
- LinkedList maintains the **insertion order** of the elements.
- It is **not synchronized**. If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally.
- Use Collections.synchronizedList(new LinkedList()) to get synchronized linkedlist.
- The iterators returned by this class are fail-fast and may throw `ConcurrentModificationException`.
- It does not implement RandomAccess interface. So we can access elements in sequential order only. It does not support accessing elements randomly.
- We can use ListIterator to iterate LinkedList elements.

3. LinkedList Constructors

1. **LinkedList()** : initializes an empty LinkedList implementation.
2. **LinkedListExample(Collection c)** : initializes a LinkedList containing the elements of the specified collection, in the order they are returned by the collection's iterator.

4. LinkedList Methods

1. **boolean add(Object o)** : appends the specified element to the end of a list.
2. **void add(int index, Object element)** : inserts the specified element at the specified position index in a list.
3. **void addFirst(Object o)** : inserts the given element at the beginning of a list.

4. **void addLast(Object o)** : appends the given element to the end of a list.
5. **int size()** : returns the number of elements in a list
6. **boolean contains(Object o)** : return `true` if the list contains a specified element, else `false`.
7. **boolean remove(Object o)** : removes the first occurence of the specified element in a list.
8. **Object getFirst()** : returns the first element in a list.
9. **Object getLast()** : returns the last element in a list.
10. **int indexOf(Object o)** : returns the index in a list of the first occurrence of the specified element, or -1 if the list does not contain specified element.
11. **lastIndexOf(Object o)** : returns the index in a list of the last occurrence of the specified element, or -1 if the list does not contain specified element.
12. **Iterator iterator()** : returns an iterator over the elements in this list in proper sequence.
13. **Object[] toArray()** : returns an array containing all of the elements in this list in proper sequence.
14. **List subList(int fromIndex, int toIndex)** : returns a view of the portion of this list between the specified fromIndex (inclusive) and toIndex (exclusive).

5. Java LinkedList Example

5.1. Add, remove, iterate

Java program to demo the usage of basic methods in linkedlist class.

```
LinkedListExample examples

import java.util.LinkedList;
import java.util.ListIterator;

public class LinkedListExample
{
    public static void main(String[] args)
    {
        //Create linked list
        LinkedList<String> linkedList = new LinkedList<>();

        //Add elements
        linkedList.add("A");
        linkedList.add("B");
        linkedList.add("C");
        linkedList.add("D");

        System.out.println(linkedList);

        //Add elements at specified position
        linkedList.add(4, "A");
        linkedList.add(5, "A");

        System.out.println(linkedList);

        //Remove element
        linkedList.remove("A");   //removes A
        linkedList.remove(0);   //removes B

        System.out.println(linkedList);
```

```
        //Iterate
        ListIterator<String> itrator = linkedList.listIterator();

        while (itrator.hasNext()) {
            System.out.println(itrator.next());
        }
    }
}
```

Program Output.

```
Console

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

5.2. Convert between Array and LinkedList

Java program to **convert LinkedList to array** and **convert array to linkedlist**.

```
LinkedListExample examples

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

linkedList.add("A");
linkedList.add("B");
linkedList.add("C");
linkedList.add("D");

//1. LinkedList to Array
String array[] = new String[linkedList.size()];
linkedList.toArray(array);

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

//2. Array to LinkedList
LinkedList<String> linkedListNew = new LinkedList<>(Arrays.asList(array));

System.out.println(linkedListNew);
```

Program Output.

```
Console

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

5.3. How to sort LinkedList

Java example to sort a LinkedList using **Collections.sort()** method. Please note that for custom sorting of objects, we can use **Collections.sort(linkedList, comparator)** method.

```
LinkedListExample examples

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

linkedList.add("A");
linkedList.add("C");
linkedList.add("B");
linkedList.add("D");

//Unsorted
System.out.println(linkedList);

//1. Sort the list
Collections.sort(linkedList);

//Sorted
System.out.println(linkedList);

//2. Custom sorting
Collections.sort(linkedList, Collections.reverseOrder());

//Custom sorted
System.out.println(linkedList);
```

Program Output.

```
Console

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

6. LinkedList Usecases

In any desktop application, actions can be recorded in linked list and implement Undo and Redo function iterating from last.

Browser's Next and Previous buttons can be programmed using linkedlist.

Linked Lists (paired with a hashtable) are really useful for LRU Caches.

7. LinkedList Performance

In Java LinkedList class, manipulation is fast because no shifting needs to be occurred. So essentially, all add and remove method provide very good performance **O(1)**.

- add(E element) method is of O(1).
- get(int index) and add(int index, E element) methods are of O(n).
- remove(int index) method is of O(n).
- Iterator.remove() is O(1).
- ListIterator.add(E element) is O(1).

LinkedList should be preferred there are no large number of random access of element while there are a large number of add/remove operations.

8. ArrayList vs LinkedList

Let's list down few notiable **differences between arraylist and linkedlist**.

- ArrayList is implemented with the concept of dynamic resizable array. While LinkedList is a doubly linked list implementation.
- ArrayList allows random access to it's elements while LinkedList does not.
- LinkedList, also implements `Queue` interface which adds more methods than ArrayList, such as offer(), peek(), poll(), etc.
- While comparing to LinkedList, ArrayList is slower in add and remove, but faster in get, because there is no need of resizing array and copying content to new array if array gets full in LinkedList.
- LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object but in case of LinkedList each node holds both data and address of next and previous node.

9. Conclusion

In this **Java LinkedList tutorial**, we learned what is a LinkedList, what are the differences between a LinkedList and an ArrayList, how to create a LinkedList, how to add, remove and search for elements in a LinkedList, and how to iterate over a LinkedList.

Let me know your questions if any.

Happy Learning !!

Reference:

LinkedList Java Docs

Recommended Reading:

1. How to convert LinkedList to ArrayList in Java

2. Difference between LinkedList vs ArrayList in Java
3. How to Detect infinite loop in LinkedList with Example
4. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
5. Java TransferQueue – Java LinkedTransferQueue class
6. Java LinkedHashMap class
7. Java TreeMap class
8. Java TreeSet class
9. Java PriorityBlockingQueue class
0. Java ArrayBlockingQueue class

1 thought on “Java LinkedList class”

Azhwani

January 24, 2020 at 9:33 pm

I was googling around for content about Linkedlist in java this morning when I came across your excellent article! Thanks a lot Lokesh Gupta!

Reply

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial