**<https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/>**

**LinkedList in Java with Example**

By Chaitanya Singh | Filed Under: [Java Collections](https://beginnersbook.com/category/java-collections/)

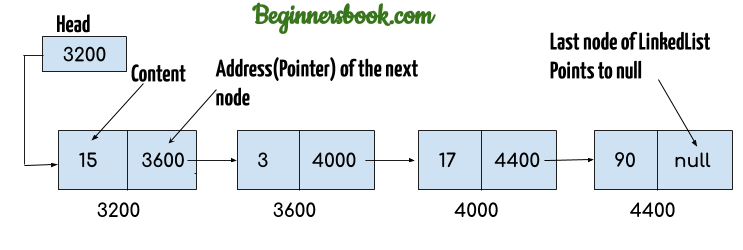
Similar to arrays in Java, **LinkedList is a linear data structure**. However LinkedList elements are not stored in contiguous locations like arrays, they are linked with each other using pointers. Each element of the LinkedList has the reference(address/pointer) to the next element of the LinkedList.

**Table of Contents**

1. [LinkedList Representation](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#linkedlist)  
2. [Why do we need a Linked List?](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#why)  
3. [Hierarchy of LinkedList class in Java](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#hierarchy)  
4. [Java Linked List example of adding elements](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#add)  
5. [Java Linked List example of removing elements](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#remove)  
6. [Example of LinkedList in Java](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#example)  
7. [Methods of LinkedList class](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#methods)  
8. [Tutorials on LinkedList](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/#tutorials) – All the methods of LinkedList class covered in detail in separate tutorials

**LinkedList representation**

Each element in the LinkedList is called the **Node**. Each Node of the LinkedList contains two items: 1) Content of the element 2) Pointer/Address/Reference to the Next Node in the LinkedList.

**This is how a LinkedList looks:**  


**Note:**  
1. **Head** of the LinkedList only contains the Address of the **First element** of the List.  
2. The Last element of the LinkedList contains **null** in the pointer part of the node because it is the end of the List so it doesn’t point to anything as shown in the above diagram.  
3. The diagram which is shown above represents a **singly linked list**. There is another complex type variation of LinkedList which is called **doubly linked list**, node of a doubly linked list contains three parts: 1) Pointer to the previous node of the linked list 2) content of the element 3) pointer to the next node of the linked list.

**Why do we need a Linked List?**

You must be aware of the arrays which is also a linear data structure but **arrays have certain limitations such as:**  
1) **Size of the array is fixed** which is decided when we create an array so it is hard to predict the number of elements in advance, if the declared size fall short then we cannot increase the size of an array and if we declare a large size array and do not need to store that many elements then it is a waste of memory.

2) Array elements **need contiguous memory locations** to store their values.

3) **Inserting an element in an array is performance wise expensive** as we have to shift several elements to make a space for the new element. For example:  
Let’s say we have an array that has following elements: 10, 12, 15, 20, 4, 5, 100, now if we want to insert a new element 99 after the element that has value 12 then we have to shift all the elements after 12 to their right to make space for new element.

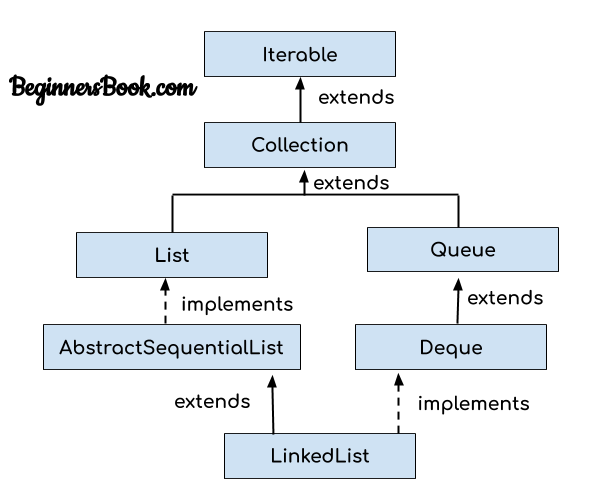
Similarly **deleting an element** from the array is also a performance wise expensive operation because all the elements after the deleted element have to be shifted left.

**These limitations are handled in the Linked List by providing following features:**  
1. Linked list allows **dynamic memory allocation**, which means memory allocation is done at the run time by the compiler and we do not need to mention the size of the list during linked list declaration.

2. Linked list elements **don’t need contiguous memory locations** because elements are linked with each other using the reference part of the node that contains the address of the next node of the list.

3. Insert and delete operations in the Linked list are not performance wise expensive because adding and deleting an element from the linked list does’t require element shifting, only the pointer of the previous and the next node requires change.

**Hierarchy of LinkedList class in Java**



**Java Linked List example of adding elements**

In the following example we are using add(), addFirst() and addLast() methods to add the elements at the desired locations in the LinkedList, there are several such useful methods in the LinkedList class which I have mentioned at the end of this article.

package com.beginnersbook;

import java.util.\*;

public class JavaExample{

public static void main(String args[]){

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

//Adding elements to the Linked list

list.add("Steve");

list.add("Carl");

list.add("Raj");

//Adding an element to the first position

list.addFirst("Negan");

//Adding an element to the last position

list.addLast("Rick");

//Adding an element to the 3rd position

list.add(2, "Glenn");

//Iterating LinkedList

Iterator<String> iterator=list.iterator();

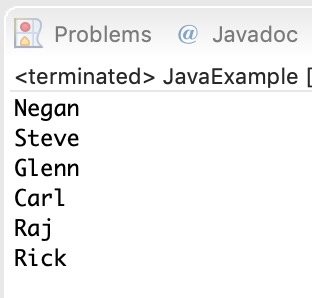
while(iterator.hasNext()){

System.out.println(iterator.next());

}

}

}

**Output:**  


**Java example of removing elements from the LinkedList**

In the following example we are checking out the few popular **remove methods** in the LinkedList that are used to remove elements from certain positions in the LinkedList. Detailed explanation of these methods along with examples are covered in the separate tutorials, links are provided at the end of this article.

package com.beginnersbook;

import java.util.\*;

public class JavaExample{

public static void main(String args[]){

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

//Adding elements to the Linked list

list.add("Steve");

list.add("Carl");

list.add("Raj");

list.add("Negan");

list.add("Rick");

//Removing First element

//Same as list.remove(0);

list.removeFirst();

//Removing Last element

list.removeLast();

//Iterating LinkedList

Iterator<String> iterator=list.iterator();

while(iterator.hasNext()){

System.out.print(iterator.next()+" ");

}

//removing 2nd element, index starts with 0

list.remove(1);

System.out.print("\nAfter removing second element: ");

//Iterating LinkedList again

Iterator<String> iterator2=list.iterator();

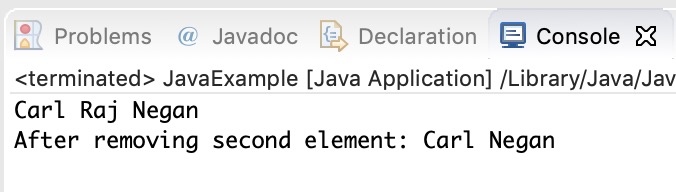
while(iterator2.hasNext()){

System.out.print(iterator2.next()+" ");

}

}

}

**Output:**  


**Example of LinkedList in Java**

import java.util.\*;

public class LinkedListExample {

public static void main(String args[]) {

/\* Linked List Declaration \*/

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

/\*add(String Element) is used for adding

\* the elements to the linked list\*/

linkedlist.add("Item1");

linkedlist.add("Item5");

linkedlist.add("Item3");

linkedlist.add("Item6");

linkedlist.add("Item2");

/\*Display Linked List Content\*/

System.out.println("Linked List Content: " +linkedlist);

/\*Add First and Last Element\*/

linkedlist.addFirst("First Item");

linkedlist.addLast("Last Item");

System.out.println("LinkedList Content after addition: " +linkedlist);

/\*This is how to get and set Values\*/

Object firstvar = linkedlist.get(0);

System.out.println("First element: " +firstvar);

linkedlist.set(0, "Changed first item");

Object firstvar2 = linkedlist.get(0);

System.out.println("First element after update by set method: " +firstvar2);

/\*Remove first and last element\*/

linkedlist.removeFirst();

linkedlist.removeLast();

System.out.println("LinkedList after deletion of first and last element: " +linkedlist);

/\* Add to a Position and remove from a position\*/

linkedlist.add(0, "Newly added item");

linkedlist.remove(2);

System.out.println("Final Content: " +linkedlist);

}

}

Output:

Linked List Content: [Item1, Item5, Item3, Item6, Item2]

LinkedList Content after addition: [First Item, Item1, Item5, Item3, Item6, Item2, Last Item]

First element: First Item

First element after update by set method: Changed first item

LinkedList after deletion of first and last element: [Item1, Item5, Item3, Item6, Item2]

Final Content: [Newly added item, Item1, Item3, Item6, Item2]

**Methods of LinkedList class:**

Here I have mentioned the brief description of the LinkedList methods, I have covered each one of these methods in separate tutorials, links are provided at the end of this article.

For all the examples in the below methods, consider llistobj as a reference for LinkedList<String>.

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

1) **boolean add(Object item)**: It adds the item at the end of the list.

llistobj.add("Hello");

It would add the string “Hello” at the end of the linked list.

2) **void add(int index, Object item)**: It adds an item at the given index of the the list.

llistobj.add(2, "bye");

This will add the string “bye” at the 3rd position( 2 index is 3rd position as index starts with 0).

3) **boolean addAll(Collection c)**: It adds all the elements of the specified collection c to the list. It throws NullPointerException if the specified collection is null. Consider the below example –

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

ArrayList<String> arraylist= new ArrayList<String>();

arraylist.add("String1");

arraylist.add("String2");

llistobj.addAll(arraylist);

This piece of code would add all the elements of ArrayList to the LinkedList.

4) **boolean addAll(int index, Collection c)**: It adds all the elements of collection c to the list starting from a give index in the list. It throws NullPointerException if the collection c is null and IndexOutOfBoundsException when the specified index is out of the range.

llistobj.add(5, arraylist);

It would add all the elements of the ArrayList to the LinkedList starting from position 6 (index 5).

5) **void addFirst(Object item)**: It adds the item (or element) at the first position in the list.

llistobj.addFirst("text");

It would add the string “text” at the beginning of the list.

6) **void addLast(Object item)**: It inserts the specified item at the end of the list.

llistobj.addLast("Chaitanya");

This statement will add a string “Chaitanya” at the end position of the linked list.

7) **void clear()**: It removes all the elements of a list.

llistobj.clear();

8) **Object clone()**: It returns the copy of the list.

For e.g. My linkedList has four items: text1, text2, text3 and text4.

Object str= llistobj.clone();

System.out.println(str);

Output: The output of above code would be:

[text1, text2, text3, text4]

9) **boolean contains(Object item)**: It checks whether the given item is present in the list or not. If the item is present then it returns true else false.

boolean var = llistobj.contains("TestString");

It will check whether the string “TestString” exist in the list or not.

10) **Object get(int index)**: It returns the item of the specified index from the list.

Object var = llistobj.get(2);

It will fetch the 3rd item from the list.

11) **Object getFirst()**: It fetches the first item from the list.

Object var = llistobj.getFirst();

12) **Object getLast()**: It fetches the last item from the list.

Object var= llistobj.getLast();

13) **int indexOf(Object item)**: It returns the index of the specified item.

llistobj.indexOf("bye");

14) **int lastIndexOf(Object item)**: It returns the index of last occurrence of the specified element.

int pos = llistobj.lastIndexOf("hello);

integer variable pos will be having the index of last occurrence of string “hello”.

15) **Object poll()**: It returns and removes the first item of the list.

Object o = llistobj.poll();

16) **Object pollFirst()**: same as poll() method. Removes the first item of the list.

Object o = llistobj.pollFirst();

17) **Object pollLast()**: It returns and removes the last element of the list.

Object o = llistobj.pollLast();

18) **Object remove()**: It removes the first element of the list.

llistobj.remove();

19) **Object remove(int index)**: It removes the item from the list which is present at the specified index.

llistobj.remove(4);

It will remove the 5th element from the list.

20) **Object remove(Object obj)**: It removes the specified object from the list.

llistobj.remove("Test Item");

21) **Object removeFirst()**: It removes the first item from the list.

llistobj.removeFirst();

22) **Object removeLast()**: It removes the last item of the list.

llistobj.removeLast();

23) **Object removeFirstOccurrence(Object item)**: It removes the first occurrence of the specified item.

llistobj.removeFirstOccurrence("text");

It will remove the first occurrence of the string “text” from the list.

24) **Object removeLastOccurrence(Object item)**: It removes the last occurrence of the given element.

llistobj.removeLastOccurrence("String1);

It will remove the last occurrence of string “String1”.

25) **Object set(int index, Object item)**: It updates the item of specified index with the give value.

llistobj.set(2, "Test");

It will update the 3rd element with the string “Test”.

26) **int size()**: It returns the number of elements of the list.

llistobj.size();