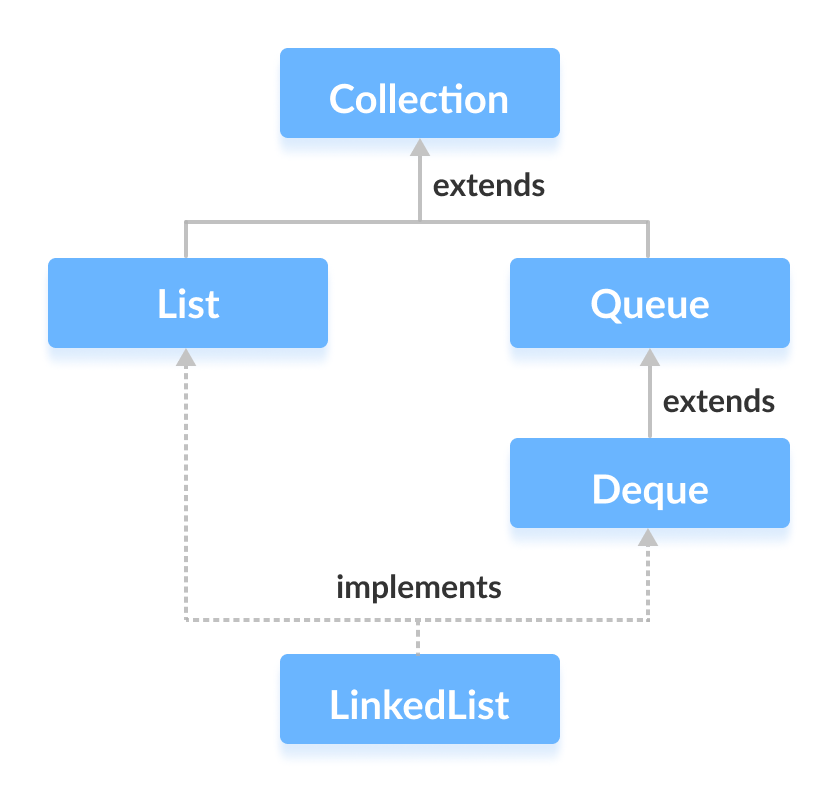
**Java LinkedList**

## The LinkedList class of the Java Collections framework provides the functionality of the linked list data structure.

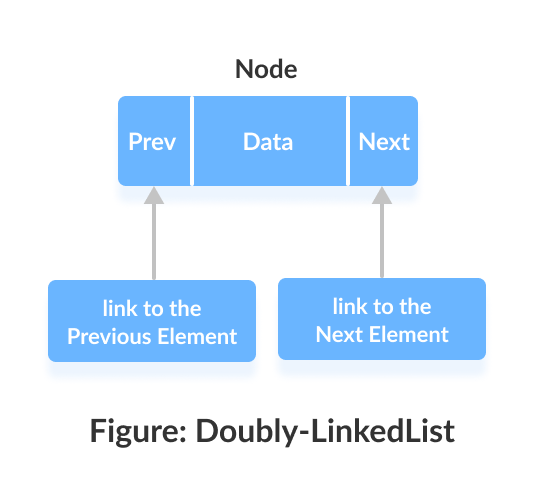
## Interfaces implemented by LinkedList

* Java List Interface
* Java Queue Interface
* Java Deque Interface



## LinkedList Implementation in Java

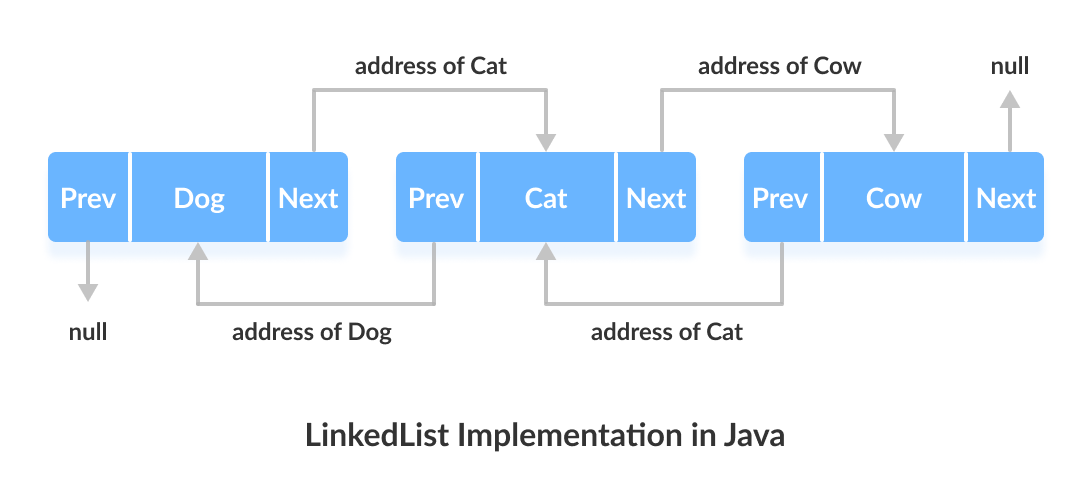
The Java LinkedList class provides a doubly linked list implementation.



Each element in a linked list is known as a **node**. It consists of 3 fields:

* **Prev** - Stores an address of the previous element in the list. It is null for the first element.
* **Next** - Stores an address of the next element in the list. It is null for the last element.
* **Data** - Stores the actual data.

Elements in linked lists are not stored in sequence. Instead, they are scattered and connected through links (Prev and Next).



Here we have 3 elements in a linked list.

* Dog - it is the first element that holds null as previous address and address of Cat as next address
* Cat - it is the second element that holds an address of Dog as previous address and address of Cow as next address
* Cow - it is the last element that holds the address of Cat as the previous address and null as the next element

## Creating a LinkedList

Here is how we can create linked lists in Java:

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

Here, Type indicates the type of a linked list. For example,

// create Integer type linked list

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

// create String type linked list

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

## Creating a LinkedList using Interfaces

Let's take an example.

List<String> animals1 = new LinkedList<>();

Here we have declared a linked list, animals1, using the List interface. The linked list can only access the methods of the List interface.

Let's take one more example.

Queue<String> animals2 = new LinkedList<>();

Deque<String> animals3 = new LinkedList<>();

Here, animals2 can only access the methods of the Queue interface.

However, animals3 can only access the methods of the Deque and Queue interfaces. It's because Deque is a subinterface of Queue.

## Methods of LinkedList

LinkedList provides various methods that allow us to perform different operations in linked lists.

### Add Elements to LinkedList

**1. Add Elements: using the add() method**

To add an element (node) to the end of the linked list, we use the add() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args){

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

// Add elements to LinkedList

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

System.out.println("LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Cat, Horse]

**2. Add Elements: using an index number**

We can also add elements to the linked list using indexes. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args){

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

// Add elements using indexes

animals.add(0,"Dog");

animals.add(1,"Cat");

animals.add(2,"Horse");

System.out.println("LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Cat, Horse]

**3. Add Elements: One linked list to another**

To add all the elements of a linked list to another linked list, we use the addAll() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args){

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

mammals.add("Dog");

mammals.add("Cat");

mammals.add("Horse");

System.out.println("Mammals: " + mammals);

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

animals.add("Crocodile");

// Add all elements of mammals in animals

animals.**addAl**l(mammals);

System.out.println("Animals: " + animals);

}

}

**Output**

Mammals: [Dog, Cat, Horse]

Animals: [Crocodile, Dog, Cat, Horse]

**4. Add Elements: using listIterator() Method**

We can also use the listsIterator() method to add elements to the linked list. To use it, we must import java.util.ListIterator package. For example,

import java.util.ArrayList;

import java.util.ListIterator;

class Main {

public static void main(String[] args) {

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

// Creating an object of ListIterator

ListIterator<String> listIterate = animals.listIterator();

listIterate.add("Dog");

listIterate.add("Cat");

System.out.println("LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Cat]

### Access LinkedList Elements

**1. Access Elements: using get() Method**

To access an element from the linked list, we can use the get() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// Add elements in the linked list

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("LinkedList: " + animals);

// Get the element from the linked list

String str = animals.get(1);

System.out.print("Element at index 1: " + str);

}

}

**Output**

LinkedList: [Dog, Horse, Cat]

Element at index 1: Horse

**2. Access Elements: using iterator() method**

To iterate over the elements of a linked list, we can use the iterator() method. We must import java.util.Iterator package to use this method. For example,

import java.util.LinkedList;

import java.util.Iterator;

class Main {

public static void main(String[] args) {

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

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

// Creating an object of Iterator

Iterator<String> iterate = animals.iterator();

System.out.print("LinkedList: ");

while(iterate.hasNext()) {

System.out.print(iterate.next());

System.out.print(", ");

}

}

}

**Output**

LinkedList: Dog, Cat, Horse,

**Note:**

* hasNext() - returns true if there is a next element
* next() - returns the next element

To learn more about Iterator, visit Java Iterator Interface.

**3. Access Elements: using the listIterator() Method**

We can also use the listIterator() method to iterate over the elements of a linked list. To use this method, we must import java.util.ListIterator package.

The listsIterator() method is more preferred in linked lists. It is because objects of listIterator() can iterate backward as well. For example,

import java.util.LinkedList;

import java.util.ListIterator;

class Main {

public static void main(String[] args) {

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

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

}

}

}

**Output**

LinkedList: Dog, Horse, Cat,

**Note:**

* hasNext() - returns true if there is a next element
* next() - returns the next element

### Search LinkedList Elements

**1. Search element: using the contains() Method**

To check if a linked list contains a particular element or not, we use the contains() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// Add elements in the linked list

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("LinkedList: " + animals);

// Checks if Dog is in the linked list

if(animals.contains("Dog")) {

System.out.println("Dog is in LinkedList.");

}

}

}

**Output**

LinkedList: [Dog, Horse, Cat]

Dog is in LinkedList.

**2. Search element: using the indexOf() Method**

* indexOf() - returns the index of the first occurrence of an element
* lastIndexOf() - returns the index of the last occurrence of an element

For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// Add elements in the linked list

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

animals.add("Dog");

System.out.println("LinkedList: " + animals);

// First Occurrence of Dog

int index1 = animals.indexOf("Dog");

System.out.println("First Occurrence of Dog: " + index1);

// Last Occurrence of Dog

int index2 = animals.lastIndexOf("Dog");

System.out.println("Last Occurrence of Dog: " + index2);

}

}

**Output**

LinkedList: [Dog, Horse, Cat, Dog]

First Occurrence of Dog: 0

Last Occurrence of Dog: 3

**Note:** Both indexOf() and lastIndexOf() returns -1 if the specified element is not found.

### Change LinkedList Elements

**1. Change element: using the set() Method**

To change elements of a linked list, we can use the set() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// Add elements in the linked list

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

animals.add("Dog");

System.out.println("LinkedList: " + animals);

// Change elements at index 3

animals.set(3, "Zebra");

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Horse, Cat, Dog]

New LinkedList: [Dog, Horse, Cat, Zebra]

### Remove LinkedList Elements

**1. Remove element: using remove() Method**

To remove an element from the linked list, we can use the remove() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

animals.add("Zebra");

System.out.println("LinkedList: " + animals);

// Remove elements from index 1

String str = animals.remove(1);

System.out.println("Removed Element: " + str);

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList : [Dog, Horse, Cat, Zebra]

Removed Element: Horse

New LinkedList: [Dog, Cat, Zebra]

**3. Remove elements: using clear() Method**

To remove all elements from a linked list, we use the clear() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// Add elements in LinkedList

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

System.out.println("LinkedList: " + animals);

// Remove all the elements

animals.clear();

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Cat, Horse]

New LinkedList: []

**Note:** We can also use the removeAll() method to remove all the elements. However, the clear() method is considered more efficient than the removeAll() method.

**4. Remove element: using the removeIf() Method**

We can also remove elements from a linked list if they satisfied a certain condition. For this, we use the removeIf() method. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

LinkedList<Integer> animals= new LinkedList<>();

// Add elements in LinkedList

animals.add(2);

animals.add(3);

animals.add(4);

animals.add(5);

System.out.println("LinkedList: " + animals);

// Remove all elements less than 4

animals.removeIf((Integer i)->i < 4);

System.out.println("New LinkedList: " + animals);

/\*\* Here we have used the lambda expression

\* For now just remember

\* parameter inside removeIf() is a condition

\*/

}

}

**Output**

LinkedList: [2, 3, 4, 5]

New LinkedList: [4, 5]

**Note:** (Integer i)->i<4 is a lambda expression. To learn about the lambda expression, visit Java Lambda Expression.

## LinkedList as Deque and Queue

Since the LinkedList class also implements the Queue and the Deque interface, it can implement methods of these interfaces as well. Here are some of the commonly used methods:

### addFirst() and addLast() Method

* addFirst() - adds the specified element at the beginning of the linked list
* addLast() - adds the specified element at the end of the linked list

For example,

import java.util.LinkedList;

import java.util.Deque;

class Main {

public static void main(String[] args){

Deque<String> animals = new LinkedList<>();

// Add element at starting of LinkedList

animals.addFirst("Cow");

animals.addFirst("Dog");

animals.addFirst("Cat");

System.out.println("LinkedList: " + animals);

// Add elements at the end of LinkedList

animals.addLast("Zebra");

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Cat, Dog, Cow]

New LinkedList: [Cat, Dog, Cow, Zebra]

### getFirst() and getLast() Method

* getFirst() - returns the first element
* getLast() - returns the last element

For example,

import java.util.LinkedList;

import java.util.Deque;

class Main {

public static void main(String[] args) {

Deque<String> animals= new LinkedList<>();

// Add elements in the linked list

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("LinkedList: " + animals);

// Get the first element from the linked list

String str1 = animals.getFirst();

System.out.println("First Element: " + str1);

// Get the last element from the linked list

String str2 = animals.getLast();

System.out.println("Last Element: " + str2);

}

}

**Output**

LinkedList: [Dog, Horse, Cat]

First Element: Dog

Last Element: Cat

### removeFirst() and removeLast() Method

* removeFirst() - removes the first element
* removeLast() - removes the last element

For example,

import java.util.LinkedList;

import java.util.Deque;

class Main {

public static void main(String[] args) {

Deque<String> animals= new LinkedList<>();

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("LinkedList: " + animals);

// Remove the first element from LinkedList

String str1 = animals.removeFirst();

System.out.println("Removed Element: " + str1);

// Remove the last element from LinkedList

String str2 = animals.removeLast();

System.out.println("Removed Element: " + str2);

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Horse, Cat]

Removed Element: Dog

Removed Element: Cat

New LinkedList: [Horse]

### peek() Method

The peek() method returns the first element (head) of the linked list. For example,

import java.util.LinkedList;

import java.util.Queue;

class Main {

public static void main(String[] args) {

Queue<String> animals= new LinkedList<>();

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("LinkedList: " + animals);

// Access the first element of LinkedList

String str = animals.peek();

System.out.println("Element Accessed: " + str);

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Horse, Cat]

Element Accessed: Dog

New LinkedList: [Dog, Horse, Cat]

### poll() method

The poll() method returns and removes the first element from the linked list. For example,

import java.util.LinkedList;

import java.util.Queue;

class Main {

public static void main(String[] args) {

Queue<String> animals= new LinkedList<>();

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("LinkedList: " + animals);

// Returns and removes the first element

String str = animals.poll();

System.out.println("Removed Element: " + str);

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Horse, Cat]

Removed Element: Dog

New LinkedList: [Horse, Cat]

### offer() Method

The offer() method adds the specified element at the end of the linked list. For example,

import java.util.LinkedList;

import java.util.Queue;

class Main {

public static void main(String[] args) {

Queue<String> animals= new LinkedList<>();

// Add elements in LinkedList

animals.add("Dog");

animals.add("Horse");

System.out.println("LinkedList: " + animals);

// Adds element at the end of LinkedList

animals.offer("Cat");

System.out.println("New LinkedList: " + animals);

}

}

**Output**

LinkedList: [Dog, Horse]

New LinkedList: [Dog, Horse, Cat]

## Iterating through LinkedList

**1. Using the forEach loop**

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

// Creating a linked list

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

animals.add("Cow");

animals.add("Cat");

animals.add("Dog");

System.out.println("LinkedList: " + animals);

// Using forEach loop

System.out.println("Accessing linked list elements:");

for(String animal: animals) {

System.out.print(animal);

System.out.print(", ");

}

}

}

**Output**

LinkedList: [Cow, Cat, Dog]

Accessing linked list elements:

Cow, Cat, Dog,

**2. Using for loop**

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

// Creating a linked list

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

animals.add("Cow");

animals.add("Cat");

animals.add("Dog");

System.out.println("LinkedList: " + animals);

// Using for loop

System.out.println("Accessing linked list elements:");

for(int i=0; i < animals.size(); i++) {

System.out.print(animals.get(i));

System.out.print(", ");

}

}

}

**Output**

LinkedList: [Cow, Cat, Dog]

Accessing linked list elements:

Cow, Cat, Dog,

In both examples, we have accessed individual elements of a linked list using loops.

**3. Using iterator() Method**

We can use the iterator() method to access elements of a linked list. In order to use this method, we must import the java.util.Iterator package.

import java.util.LinkedList;

import java.util.Iterator;

class Main {

public static void main(String[] args) {

// Creating a linked list

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

animals.add("Cow");

animals.add("Cat");

animals.add("Dog");

System.out.println("LinkedList: " + animals);

// Using the iterator() method

System.out.println("LinkedList using the iterator() method:");

Iterator<String> iterate = animals.iterator();

while(iterate.hasNext()) {

System.out.print(iterate.next());

System.out.print(", ");

}

}

}

**Output**

LinkedList: [Cow, Cat, Dog]

LinkedList using the iterator() method:

Cow, Cat, Dog,

The following code example demonstrates converting an Array of Strings to a LinkedList of Strings.

String[] sa = new String[] { "A", "B", "C" };

        // convert array to LinkedList

        LinkedList ll = new LinkedList(Arrays.asList(sa));

        // iterate over each element in LinkedList and show what is in the list.

        Iterator iterator = ll.iterator();

        while (iterator.hasNext())

{

            // Print element to console

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

         }

## LinkedList vs. ArrayList

Both LinkedList and ArrayList implements the List interface of the Collections framework. However, there exists some difference between them.

|  |  |
| --- | --- |
| Linked list | Array list |
| stores 3 values (**previous address**, **data,** and **next address**) in a single position | stores a single value in a single position |
| provides the doubly-linked list implementation of List | provides a resizable array implementation |
| whenever an element is added, prev and next address are changed | whenever an element is added,  all elements after that position are shifted |
| To access an element, we need to iterate from the beginning to the element | can randomly access elements using indexes. |