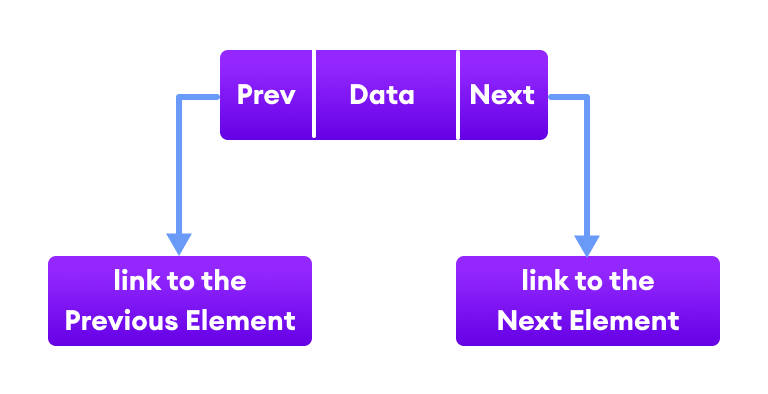
**Java LinkedList**

The LinkedList class of the Java collections framework provides the functionality of the linked list data structure



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

**Creating a Java 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<>();

**Example: Create LinkedList in Java**

import java.util.LinkedList;

class Main {

public static void main(String[] args){

// create linkedlist

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

// Add elements to LinkedList

animals.add("Dog");

animals.add("Cat");

animals.add("Cow");

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

}

}

Run Code

**Output**

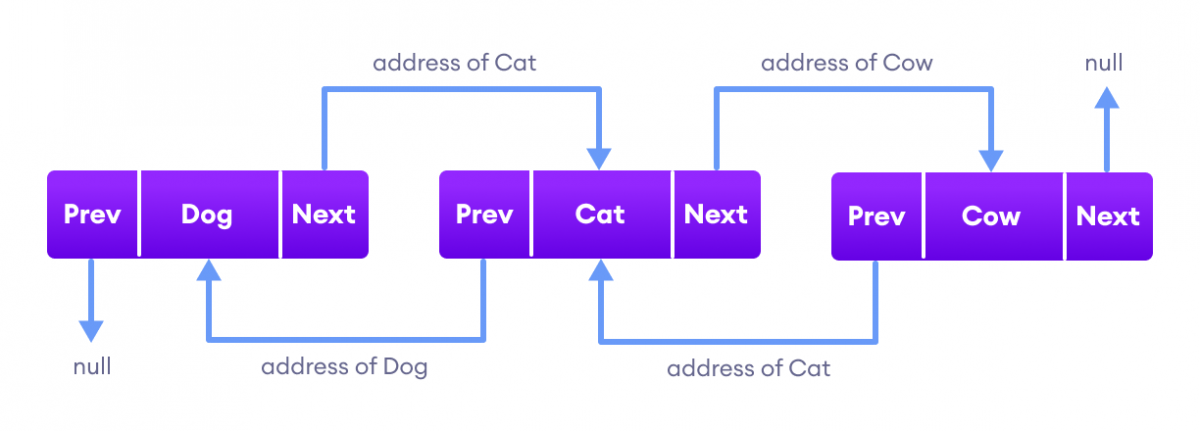
LinkedList: [Dog, Cat, Cow]

In the above example, we have created a LinkedList named animals.

Here, we have used the add() method to add elements to the LinkedList. We will learn more about the add() method later in this tutorial.

**Working of a Java LinkedList**

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

Java LinkedList Implementation

Here we have 3 elements in a linked list.

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

**Methods of Java LinkedList**

LinkedList provides various methods that allow us to perform different operations in linked lists. We will look at four commonly used LinkedList Operators in this tutorial:

* Add elements
* Access elements
* Change elements
* Remove elements

**1. Add elements to a LinkedList**

We can use the add() method to add an element (node) at the end of the LinkedList. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args){

// create linkedlist

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

// add() method without the index parameter

animals.add("Dog");

animals.add("Cat");

animals.add("Cow");

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

// add() method with the index parameter

animals.add(1, "Horse");

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

}

}

Run Code

**Output**

LinkedList: [Dog, Cat, Cow]

Updated LinkedList: [Dog, Horse, Cat, Cow]

In the above example, we have created a LinkedList named animals. Here, we have used the add() method to add elements to animals.

Notice the statement,

animals.add(1, "Horse");

Here, we have used the **index number** parameter. It is an optional parameter that specifies the position where the new element is added.

.

**2. Access LinkedList elements**

The get() method of the LinkedList class is used to access an element from the LinkedList. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// add elements in the linked list

languages.add("Python");

languages.add("Java");

languages.add("JavaScript");

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

// get the element from the linked list

String str = languages.get(1);

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

}

}

Run Code

**Output**

LinkedList: [Python, Java, JavaScript]

Element at index 1: Java

In the above example, we have used the get() method with parameter **1**. Here, the method returns the element at index **1**.

We can also access elements of the LinkedList using the iterator() and the listIterator() method. To learn more,

**3. Change Elements of a LinkedList**

The set() method of LinkedList class is used to change elements of the LinkedList. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// add elements in the linked list

languages.add("Java");

languages.add("Python");

languages.add("JavaScript");

languages.add("Java");

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

// change elements at index 3

languages.set(3, "Kotlin");

System.out.println("Updated LinkedList: " + languages);

}

}

Run Code

**Output**

LinkedList: [Java, Python, JavaScript, Java]

Updated LinkedList: [Java, Python, JavaScript, Kotlin]

In the above example, we have created a LinkedList named languages. Notice the line,

languages.set(3, "Kotlin");

Here, the set() method changes the element at index **3** to Kotlin.

**4. Remove element from a LinkedList**

The remove() method of the LinkedList class is used to remove an element from the LinkedList. For example,

import java.util.LinkedList;

class Main {

public static void main(String[] args) {

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

// add elements in LinkedList

languages.add("Java");

languages.add("Python");

languages.add("JavaScript");

languages.add("Kotlin");

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

// remove elements from index 1

String str = languages.remove(1);

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

System.out.println("Updated LinkedList: " + languages);

}

}

Run Code

**Output**

LinkedList: [Java, Python, JavaScript, Kotlin]

Removed Element: Python

New LinkedList: [Java, JavaScript, Kotlin]

Here, the remove() method takes the index number as the parameter. And, removes the element specified by the index number.

**Other Methods**

|  |  |
| --- | --- |
| Methods | Description |
| contains() | checks if the LinkedList contains the element |
| indexOf() | returns the index of the first occurrence of the element |
| lastIndexOf() | returns the index of the last occurrence of the element |
| clear() | removes all the elements of the LinkedList |
| iterator() | returns an iterator to iterate over LinkedList |

**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:

|  |  |
| --- | --- |
| Methods | Descriptions |
| addFirst() | adds the specified element at the beginning of the linked list |
| addLast() | adds the specified element at the end of the linked list |
| getFirst() | returns the first element |
| getLast() | returns the last element |
| removeFirst() | removes the first element |
| removeLast() | removes the last element |
| peek() | returns the first element (head) of the linked list |
| poll() | returns and removes the first element from the linked list |
| offer() | adds the specified element at the end of the linked list |

**Example: Java LinkedList as Queue**

import java.util.LinkedList;

import java.util.Queue;

class Main {

public static void main(String[] args) {

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

// add elements

languages.add("Python");

languages.add("Java");

languages.add("C");

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

// access the first element

String str1 = languages.peek();

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

// access and remove the first element

String str2 = languages.poll();

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

System.out.println("LinkedList after poll(): " + languages);

// add element at the end

languages.offer("Swift");

System.out.println("LinkedList after offer(): " + languages);

}

}

Run Code

**Output**

LinkedList: [Python, Java, C]

Accessed Element: Python

Removed Element: Python

LinkedList after poll(): [Java, C]

LinkedList after offer(): [Java, C, Swift]

**Example: LinkedList as Deque**

import java.util.LinkedList;

import java.util.Deque;

class Main {

public static void main(String[] args){

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

// add element at the beginning

animals.add("Cow");

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

animals.addFirst("Dog");

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

// add elements at the end

animals.addLast("Zebra");

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

// remove the first element

animals.removeFirst();

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

// remove the last element

animals.removeLast();

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

}

}

Run Code

**Output**

LinkedList: [Cow]

LinkedList after addFirst(): [Dog, Cow]

LinkedList after addLast(): [Dog, Cow, Zebra]

LinkedList after removeFirst(): [Cow, Zebra]

LinkedList after removeLast(): [Cow]

**Iterating through LinkedList**

We can use the Java for-each loop to iterate through LinkedList. For example,

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(", ");

}

}

}

Run Code

**Output**

LinkedList: [Cow, Cat, Dog]

Accessing linked list elements:

Cow, Cat, Dog,

**LinkedList Vs. ArrayList**

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

|  |  |
| --- | --- |
| LinkedList | ArrayList |
| Implements List, Queue, and Deque interfaces. | Implements List interface. |
| 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. | 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. |

**Note**: We can also create a LinkedList using interfaces in Java. For example,

// create linkedlist using List

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

// creating linkedlist using Queue

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

// creating linkedlist using Deque

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

Here, if the LinkedList is created using one interface, then we cannot use methods provided by other interfaces. That is, animals1 cannot use methods specific to Queue and Deque interfaces.