CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

LinkedList

Overview

1. Introduction
2. Hierarchy Diagram of LinkedList class
3. LinkedList class declaration
4. LinkedList Constructors
5. Features of LinkedList class
6. How does Insertion work in LinkedList?
7. How does Deletion work in LinkedList?
8. LinkedList Methods
9. LinkedList Deque Methods
10. When is best case to use LinkedList in a Java app?
11. When is worst case to use LinkedList in a Java app?

Introduction

➲ **LinkedList in Java** is a linear data structure that uses a doubly linked list internally to store a group of elements. A doubly linked list consists of a group of nodes that together represents a sequence in the list. It stores the group of elements in the sequence of nodes.

➲ Each node contains three fields: a *`data`* field that contains data stored in the node, left and right fields contain references or pointers that point to the *`previous`* and *`next`* nodes in the list. A pointer indicates the addresses of the next node and the previous node. Elements in the linked list are called **nodes**.

➲ Since the *`previous`* field of the first node and the *`next`* field of the last node do not point to anything, we must set it with the null value.

➲ `LinkedList` in Java is a very convenient way to store elements (data). When we store a new element in the linked list, a new node is automatically created.

Introduction

➲ Its size will grow with the addition of each and every element. Therefore, its initial capacity is zero. When an element is removed, it will automatically shrink.

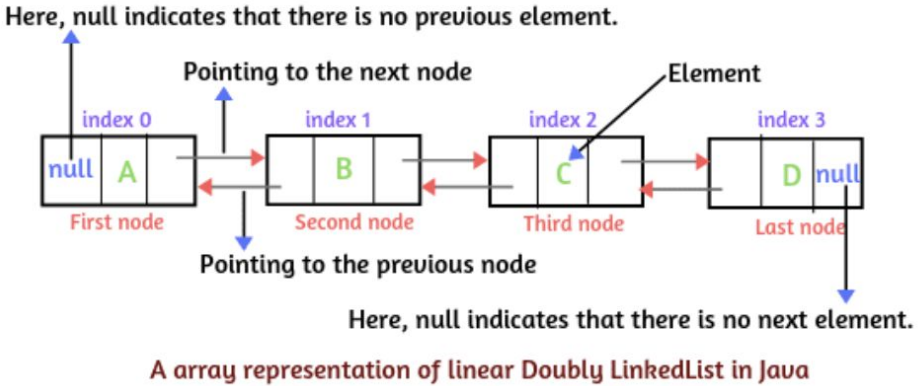
➲ Adding elements into the LinkedList and removing elements from the LinkedList are done quickly and take the same amount of time (i.e. constant time). So, it is especially useful in situations where elements are inserted or removed from the middle of the list.

➲ In the Linked List, the elements are not stored in the consecutive memory location. An element (often called node) can be located anywhere in the free space of the memory by connecting each other using the left and right sides of the node portion.

Introduction

An array representation of linear doubly LinkedList in Java is shown in the below figure.

Representation of Java LinkedList Node



A array representation of linear Doubly LinkedList in Java

Introduction

Thus, it avoids the rearrangement of elements required *(by ArrayList)* but requires that each element is connected to the next and previous by a link.

Therefore, a LinkedList is often a better choice if elements are added or removed from intermediate locations within the list.

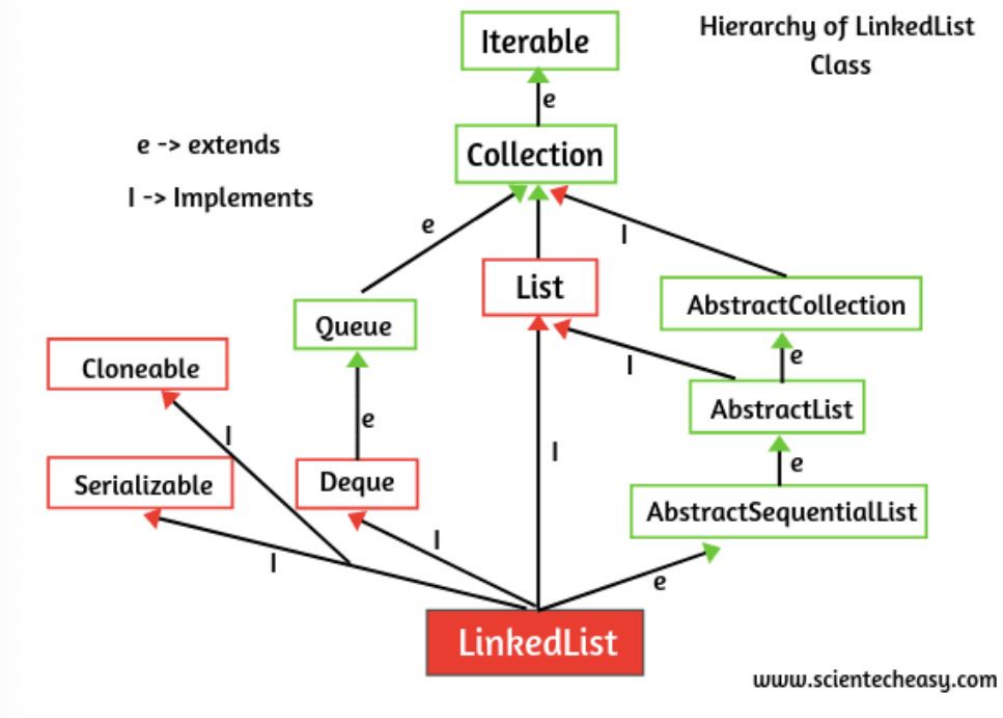
The right side of the node contains the address of the next element and the left side of the node contains the address of the previous element in the list.

Hierarchy Diagram of LinkedList class

The java `LinkedList` class implements `List`, `Deque`, and `Queue` interfaces. It extends `AbstractSequentialList`. It also implements marker interfaces such as `Serializable` and `Cloneable` but does **not** implement `RandomAccess` interface.

Hierarchy Diagram of LinkedList class

The hierarchy diagram of the LinkedList class in java can be shown in the below figure.



LinkedList class declaration

LinkedList is a generic class, just like ArrayList class that can be declared as:

```
class LinkedList<E>
```

Here, E specifies the type of elements (objects) in angle brackets that the linked list will hold.

For example,

```
LinkedList<String> or LinkedList<Employee>
```

LinkedList class was introduced in Java 1.2 version and it is placed in `java.util` package.

LinkedList Constructors

Like `ArrayList` class, `LinkedList` class consists of two constructors. They are listed in below table.

SN	Constructor	Description
1.	LinkedList()	Used to create an empty LinkedList object.
2.	LinkedList(Collection c)	Used to construct a list containing the elements of the given collection.

We can create an empty linked list object for storing String type elements (objects) as:

```
LinkedList<String> llist = new LinkedList<>(); // An empty list.
```

Features of LinkedList class

The main features of the Java LinkedList class are as follows:

1. The underlying data structure of LinkedList is a doubly LinkedList data structure. It is another concrete implementation of the List interface like an ArrayList.

2. LinkedList class allows storing duplicate elements.

3. Null elements can be added to the linked list.

4. Heterogeneous elements are allowed in the linked list.

5. Java LinkedList is not synchronized. So, multiple threads can access the same LinkedList object at the same time. Therefore, It is not thread-safe. Since LinkedList is not synchronized. Hence, its operation is faster.

6. Insertion and removal of elements in the LinkedList are fast because, in the linked list, there is no shifting of elements after each adding and removal. The only reference for next and previous elements changed.

7. LinkedList is the best choice if your frequent operation is insertion or deletion in the middle.

8. Retrieval (getting) of elements is very slow in LinkedList because it traverses from the beginning or ending to reach the element.

9. The LinkedList can be used as a “stack“. It has pop() and push() methods which make it function as a stack.

10. LinkedList does not implement random access interface. So, the element cannot be accessed (getting) randomly. To access the given element, we have to traverse from the beginning or ending to reach elements in the LinkedList.

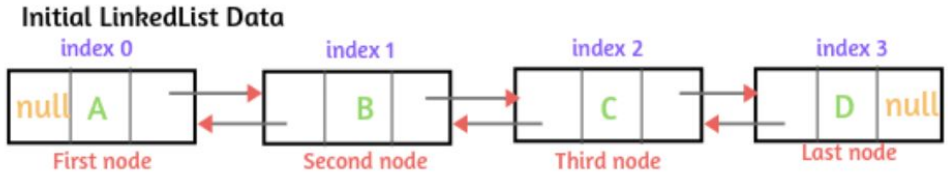
11. We can iterate linked list elements by using ListIterator.

How does Insertion work in LinkedList?

In the java LinkedList, we can perform insertion (addition) operations without affecting any of data items already stored in the list. Let’s take an example to understand this concept.

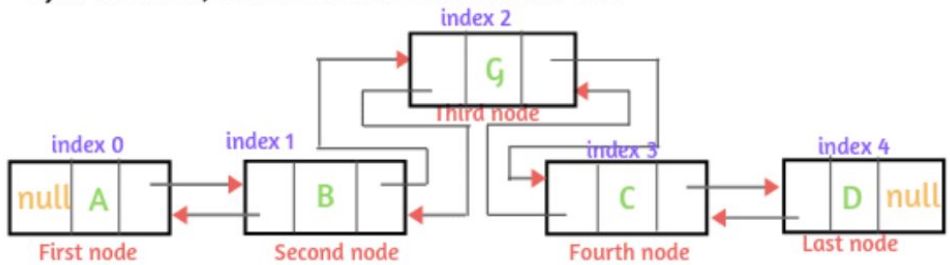
How does Insertion work in LinkedList?

1. Let us assume that our initial LinkedList has the following data as shown in the below figure.



linkedlist.add(2,"G");

After Insertion, LinkedList Data will look like this.



You can see that one node is created with element G and simply changes the next and previous pointer only. No shift of operation has occurred.

How does Insertion work in LinkedList?

2. Now we will perform insertion operation on this linked list. We will add an element G at index position 2 using add() method. The syntax for adding element G is as follow:

```
linkedlist.add(2,"G");
```

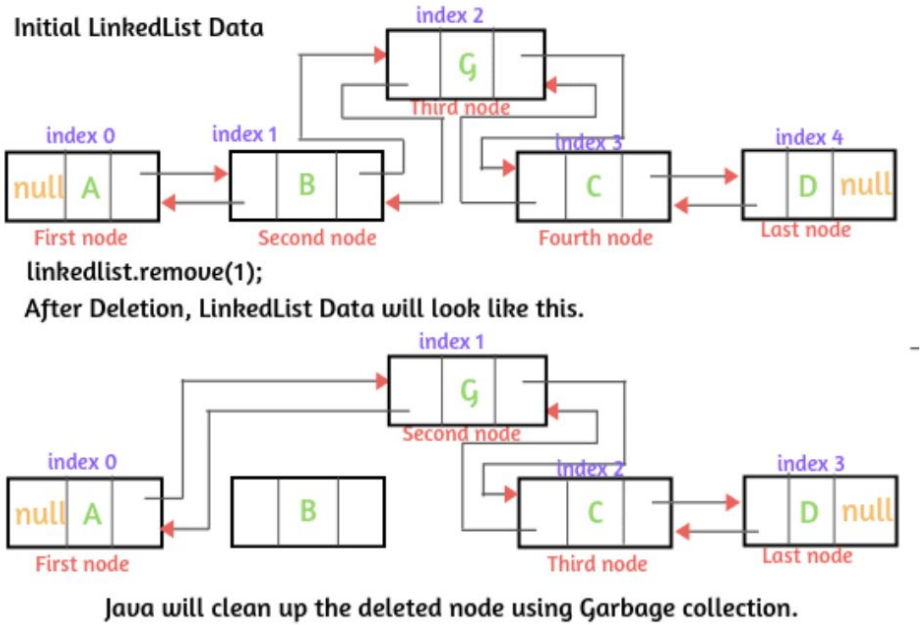
When the insertion operation takes place in the linked list, internally, LinkedList will create one node with G element at anywhere available space in the memory and changes the next and previous pointer only without shifting of any element in the list. Look at the updated LinkedList in the above figure.

How does Deletion work in LinkedList?

In the previous section, we have already seen how linked list performs insertion operations internally. Now we will discuss how Java linked list performs deletion operation internally.

How does Deletion work in LinkedList?

1. Let us assume that our initial LinkedList has the following data.



How does Deletion work in LinkedList?

2. We will perform deletion operations on this LinkedList. Suppose we want to remove or delete element B from index position 1. The syntax to delete element B is as follows:

```
linkedlist.remove(1);
```

When deletion operation takes place, the node with element B is deleted with changing the next and previous pointer. The deleted node becomes unused memory.

Therefore, Java will clean up the unused memory space using the garbage collection.

LinkedList Methods

In addition to implementing the List interface, Java LinkedList class also provides methods for retrieving, inserting, and removing elements from both ends of the list. LinkedList methods are listed in the below table:

LinkedList Methods

SN	Method	Description
1.	boolean add(Object o)	It is used to add the specified element to the end of a list.
2.	void add(int index, Object o)	Used to insert the specified element at the specified position in a list.
3.	boolean addAll(Collection c)	Used to add all of the elements in the specified collection to the end of this list
4.	boolean addAll(int index, Collection c)	It is used to add all the elements of the specified collection at specified index position in the list.
5.	void clear()	It is used to remove or delete all the elements from a list.
6.	boolean contains(Object o)	It returns true if a list contains a specified element.

LinkedList Methods

SN	Method	Description
7.	int size()	It is used to get the number of elements in a list.
8.	Object[] toArray()	It returns an array containing all the elements in a list in proper sequence
9.	Object clone()	This method is used to return a shallow copy of an ArrayList.
10.	boolean remove(Object o)	This method is used to remove the first occurrence of the specified element in the linked list.
11.	Element remove(int index)	This method is used to remove the element at the specified position in the linked list.
12.	Element element()	This method is used to retrieve the first element of the linked list.

LinkedList Methods

SN	Method	Description
13.	E get(int index)	It is used to get the element at the specified position in a list.
14.	Element set(int index, E element)	It is used to replace the element at the specified position in a list with the specified element.
15.	int indexOf(Object o)	It is used to get the index of the first occurrence of the specified element in the list. It returns -1 if the list does not contain any element.
16.	int lastIndexOf(Object o)	It is used to get the index of the last occurrence of the specified element in a list. It returns -1 if the list does not contain any element.
17.	Iterator iterator()	This method returns an iterator over the elements in a proper sequence in the linked list.
18.	ListIterator listIterator(int index)	This method returns a list-iterator of the elements in a proper sequence, starting at the specified position in the list.

LinkedList Deque Methods

Java LinkedList class has also various specific methods that are inherited from Deque interface. They are listed in the below table:

LinkedList Deque Methods

SN	Method	Description
1.	void addFirst(Object o)	It is used to add the specified element in the first position of the LinkedList.
2.	void addLast(Object o)	It is used to add the specified element to the end of the LinkedList.
3.	Object getFirst()	This method is used to return the first element from the list.
4.	Object getLast()	It is used to get the last element from the list.
5.	Object removeFirst()	It removes the first element from the linked list and returns it.
6.	Object removeLast()	It removes the last element from the linked list and returns it.
7.	boolean offerFirst(Object o)	It is used to insert the specified element at the front of a list.
8.	boolean offerLast(Object o)	It is used to insert the specified element at the end of a list.

LinkedList Deque Methods

SN	Method	Description
10.	Object peekFirst()	It is used to retrieve the first element from the linked list. It will return null if the list is empty.
11.	Object peekLast()	It is used to retrieve the last element from the linked list. It will return null if the list is empty.
12.	Object poll()	It retrieves and removes the first element from the linked list.
13.	Object pollFirst()	It retrieves and removes the first element from the linked list. It returns null if a list is empty.
14.	Object pollLast()	It retrieves and removes the last element from the linked list. It returns null if a list is empty.
15.	Object pop()	This method is used to pop an element from the stack represented by a list.
16.	void push(Object o)	This method is used to push an element onto the stack represented by a list.
10.	Object peekFirst()	It is used to retrieve the first element from the linked list. It will return null if the list is empty.

When is best case to use LinkedList in a Java app?

LinkedList is the best choice to use when your frequent operation is adding or removing elements in the middle of the list because the adding and removing of elements in the linked list is faster as compared to ArrayList.

Let’s take a real-time scenario to understand this concept. Suppose there are 100 elements in the ArrayList. If we remove the 50th element from the ArrayList, 51st element will go to 50th position, 52nd element to 51st position, and similarly for other elements that will consume a lot of time for shifting. Due to which the manipulation will be slow in ArrayList.

But in the case of linked list, if we remove 50th element from the linked list, no shifting of elements will take place after removal. Only the reference of the next and previous node will change.

Moreover, LinkedList can be used when we need a stack (LIFO) or queue (FIFO) data structure by allowing duplicates.

When is worst case to use LinkedList in a Java app?

Java LinkedList is the worst choice to use when your frequent operation is retrieval (getting) of elements from the linked list because retrieval of elements is very slow in the LinkedList as compared to ArrayList.

Since LinkedList does not implement Random Access Interface. Therefore, an element cannot be accessed (getting) randomly. We will have to traverse from the beginning or ending to reach elements in the linked list.

Let us consider a real-time scenario to understand this concept. Assume that there are ten elements in the list. Suppose that LinkedList accesses the first element in one second from the list and retrieves the element.

When is worst case to use LinkedList in a Java app?

Since the address of second element is available in the first node. So, it will take two seconds time to access and get the second element.

Similarly, if we want to get 9th element from the list then LinkedList will take 9 sec time to get element. Why? This is because the address of 9th element is available in 8th node. The address of 8th node is available in 7th node and so on.

But if there are one crore elements in the list and we want to get 50th lakh element from the list, may be linked list will take one year time to access and getting 50th lakh element. Therefore, LinkedList is the worst choice for the retrieval or searching of elements from the linked list.

In this case, ArrayList is the best choice to use for getting elements from the list because ArrayList implements randomaccess interface. So, we can get an element from the arraylist very fast from any arbitrary position.

END