Java Collection Framework Linked list

Java LinkedList class provides implementation of linked-list data structure. It used doubly linked list to store the elements. It implements List, Deque and Queue interface and extends AbstractSequentialList class. below is the declaration of LinkedList class.

LinkedList class Declaration

```
public class LinkedList<E> extends AbstractSequentialList<E> implements
List<E>, Deque<E>, Cloneable, Serializable
```

Copy

1. LinkedList class extends AbstractSequentialList and implements List,Deque and Queue inteface.
2. It can be used as List, stack or Queue as it implements all the related interfaces.
3. It allows null entry.
4. It is dynamic in nature i.e it allocates memory when required. Therefore insertion and deletion operations can be easily implemented.
5. It can contain duplicate elements and it is not synchronized.
6. Reverse Traversing is difficult in linked list.
7. In LinkedList, manipulation is fast because no shifting needs to be occurred.

LinkedList Constructors

LinkedList class has two constructors. First is used to create empty LinkedList and second for creating from existing collection.

```
LinkedList() // It creates an empty LinkedList

LinkedList( Collection c) // It creates a LinkedList that is
initialized with elements of the Collection c
```

Copy

LinkedList class Example

Lets take an example to create a linked-list, no element is inserted so it creates an empty LinkedList. See the below example.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)
  {

    // Creating LinkedList

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

    // Displaying LinkedLIst

    System.out.println(linkedList);

  }

}
```

Copy

```
[]
```

LinkedList Methods

The below table contains methods of LinkedList. We can use them to manipulate its elements.

Method	**Description**
boolean add(E e)	It appends the specified element to the end of a list.
void add(int index, E element)	It inserts the specified element at the specified position index in a list.
boolean addAll(Collection<? extends E> c)	It appends all of the elements in the specified collection to the end of this list.
boolean addAll(Collection<? extends E> c)	It appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It appends all the elements in the specified collection, starting at the specified position of the list.

void addFirst(E e)	It inserts the given element at the beginning of a list.
void addLast(E e)	It appends the given element to the end of a list.
void clear()	It removes all the elements from a list.
Object clone()	It returns a shallow copy of an ArrayList.
boolean contains(Object o)	It returns true if a list contains a specified element.
Iterator<E> descendingIterator()	It returns an iterator over the elements in a deque in reverse sequential order.
E element()	It retrieves the first element of a list.
E get(int index)	It returns the element at the specified position in a list.
E getFirst()	It returns the first element in a list.
E getLast()	It returns the last element in a list.
int indexOf(Object o)	It returns the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the

	list does not contain any element.
ListIterator<E> listIterator(int index)	It returns a list-iterator of the elements in proper sequence, starting at the specified position in the list.
boolean offer(E e)	It adds the specified element as the last element of a list.
boolean offerFirst(E e)	It inserts the specified element at the front of a list.
boolean offerLast(E e)	It inserts the specified element at the end of a list.
E peek()	It retrieves the first element of a list
E peekFirst()	It retrieves the first element of a list or returns null if a list is empty.
E peekLast()	It retrieves the last element of a list or returns null if a list is empty.
E poll()	It retrieves and removes the first element of a list.
E pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
E pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.
E pop()	It pops an element from the stack represented by a list.

void push(E e)	It pushes an element onto the stack represented by a list.
E remove()	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst()	It removes and returns the first element from a list.
boolean removeFirstOccurrence(Object o)	It removes the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast()	It removes and returns the last element from a list.
boolean removeLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.
int size()	It returns the number of elements in a list.

Add Elements to LinkedList

To add elements into LinkedList, we are using `add()` method. It adds elements into the list in the insertion order.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating LinkedList

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

    linkedList.add("Delhi");

    linkedList.add("NewYork");

    linkedList.add("Moscow");

    linkedList.add("Dubai");

    // Displaying LinkedList

    System.out.println(linkedList);

  }

}
```

Copy

```
[Delhi, NewYork, Moscow, Dubai]
```

Removing Elements

To remove elements from the list, we are using remove method that remove the specified elements. We can also pass index value to remove the elements of it.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating LinkedList

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

    linkedList.add("Delhi");
    linkedList.add("NewYork");

    linkedList.add("Moscow");

    linkedList.add("Dubai");

    // Displaying LinkedList

    System.out.println(linkedList);

    // Removing Elements

    linkedList.remove("Moscow");

    System.out.println("After Deleting Elements: \n"+linkedList);

    // Removing Second Element

    linkedList.remove(1);

    System.out.println("After Deleting Elements: \n"+linkedList);

  }

}
```

Copy

```
[Delhi, NewYork, Moscow, Dubai]

After Deleting Elements:

[Delhi, NewYork, Dubai]

After Deleting Elements:

[Delhi, Dubai]
```

Traversing Elements of LinkedList

Since LinkedList is a collection then we can use loop to iterate its elements. In this example we are traversing elements. See the below example.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating LinkedList

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

    linkedList.add("Delhi");
    linkedList.add("NewYork");

    linkedList.add("Moscow");

    linkedList.add("Dubai");

    // Traversing ArrayList

    for(String element : linkedList) {

      System.out.println(element);

    }

  }

}
```

Copy

```
Delhi

NewYork

Moscow

Dubai
```

Get size of LinkedList

Sometimes we want to know number of elements an LinkedList holds. In that case we use `size()` then returns size of LinkedList which is equal to number of elements present in the list.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating LinkedList

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

    linkedList.add("Delhi");

    linkedList.add("NewYork");

    linkedList.add("Moscow");

    linkedList.add("Dubai");

    // Traversing ArrayList
    for(String element : linkedList) {

      System.out.println(element);

    }

System.out.println("Total Elements: "+linkedList.size());

  }

}
```

Copy

```
Delhi

NewYork

Moscow

Dubai

Total Elements: 4
```

Sorting LinkedList Elements

To sort elements of an LinkedList, Java provides a class Collections that includes a static method sort(). In this example, we are using sort method to sort the elements.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating LinkedList

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

    linkedList.add("Delhi");

    linkedList.add("NewYork");

    linkedList.add("Moscow");

    linkedList.add("Dubai");

    // Sorting elements

    Collections.sort(linkedList);

    // Traversing ArrayList

    for(String element : linkedList) {
```

```java
        System.out.println(element);

      }

   }

}
```

Copy

```
OUTPUT-

Delhi

Dubai

Moscow

NewYork
```