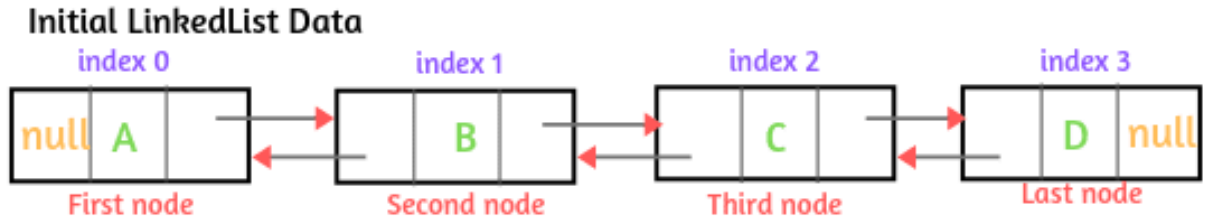
LINKEDLIST

LinkedList internally makes use of doubly linked list data structure to store data where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node.

How LinkedList is better than ArrayList?

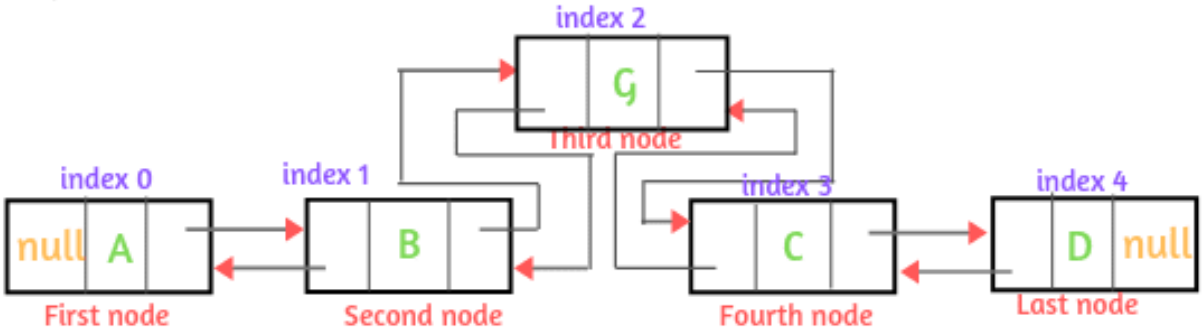
Just like arrays, ArrayList also expects contiguous Memory locations in RAM. But since LinkedList Makes use of doubly LinkedList, it can make use Of dispersed memory location thus overcoming all the disadvantages of arrays.

Internal Implementation of LinkedList



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.

Built-in Methods in LinkedList:

1. **add(int index, E element):** This method Inserts the specified element at the specified position in this list.
2. **add(E e):** This method Appends the specified element to the end of this list.
3. **addAll(int index, Collection c):** This method Inserts all of the elements in the specified collection into this list, starting at the specified position.
4. **addAll(Collection c):** This method 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.
5. **addFirst(E e):** This method Inserts the specified element at the beginning of this list.
6. **addLast(E e):** This method Appends the specified element to the end of this list.
7. **clear():** This method removes all of the elements from this list.
8. **getFirst():** This method returns the first element in this list.
9. **getLast():** This method returns the last element in this list.
10. **indexOf(Object o):** This method returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
11. **lastIndexOf(Object o):** This method returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
12. **peek():** This method retrieves, but does not remove, the head (first element) of this list.
13. **peekFirst():** This method retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
14. **peekLast():** This method retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
15. **poll():** This method retrieves and removes the head (first element) of this list.
16. **pollFirst():** This method retrieves and removes the first element of this list, or returns null if this list is empty.
17. **pollLast():** This method retrieves and removes the last element of this list, or returns null if this list is empty.
18. **pop():** This method Pops an element from the stack represented by this list.
19. **push(E e):** This method Pushes an element onto the stack represented by this list.
20. **remove():** This method retrieves and removes the head (first element) of this list.
21. **remove(int index):** This method removes the element at the specified position in this list.

22. **removeFirst():** This method removes and returns the first element from this list.
23. **removeLast():** This method removes and returns the last element from this list.
24. **set(int index, E element):** This method replaces the element at the specified position in this list with the specified element.
25. **size():** This method returns the number of elements in this list.

Let us understand these built-in methods with the help of code

```
import java.util.LinkedList;
class Demo
{
    public static void main(String[] args)
    {
        LinkedList ll = new LinkedList();
        ll.add(10);
        ll.add(20);
        ll.add(30);
        ll.addFirst(40);
        ll.addFirst(50);
        ll.addLast(60);
        ll.addLast(70);
        System.out.println(ll);
        System.out.println(ll.getFirst());
        System.out.println(ll.getLast());
        System.out.println(ll.peekFirst());
        System.out.println(ll);
        System.out.println(ll.pollFirst());
        System.out.println(ll);
        System.out.println(ll.peekLast());
        System.out.println(ll);
        System.out.println(ll.pollLast());
        System.out.println(ll);
    }
}
```

OUTPUT

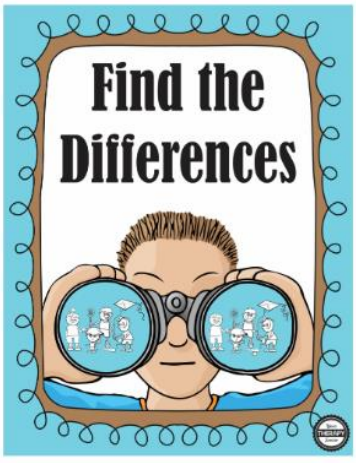
```
[50, 40, 10, 20, 30, 60, 70]
50
70
50
[50, 40, 10, 20, 30, 60, 70]
50
[40, 10, 20, 30, 60, 70]
70
[40, 10, 20, 30, 60, 70]
70
```

Difference between peekFirst() and getFirst()

If the List is empty and **getFirst()** is called, it **would result** in **NoSuchElementException.**
If the List is empty and **peekFirst()** is called, **it would not result** in any **Exception.**

Let us understand the difference with the help of code

```
import java.util.LinkedList;
class Demo
{
    public static void main(String[] args)
    {
        LinkedList ll = new LinkedList();
        System.out.println(ll);
        //System.out.println(ll.getFirst()); ----> NoSuchElementException
        System.out.println(ll.peekFirst());
        System.out.println(ll);
    }
}
```



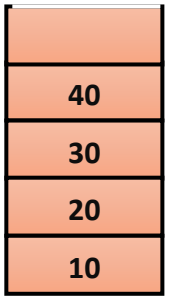
OUTPUT

```
[ ]
null
[ ]
Press any key to continue . . .
```

Implementation of Stack using LinkedList

Stack can be implemented by using push() and pop() in LinkedList as shown below:

```
import java.util.LinkedList;
class Demo
{
    public static void main(String[] args)
    {
        LinkedList ll = new LinkedList();
        ll.push(10);
        ll.push(20);
        ll.push(30);
        ll.push(40);
        System.out.println(ll);
        System.out.println(ll.pop());
        System.out.println(ll.pop());
        System.out.println(ll.pop());
        System.out.println(ll.pop());
    }
}
```



STACK

OUTPUT

```
[40, 30, 20, 10]
40
30
20
10
```

ARRAYDEQUE

ArrayDeque in Java provides a way to apply resizable-array in addition to the implementation of the Deque interface. It is also known as **Array Double Ended Queue or Array Deck.** This is a special kind of array that grows and allows users to add or remove an element from both the sides of the queue.

Few important features of ArrayDeque are as follows:

- Array deques have no capacity restrictions and they grow as necessary to support usage.
- They are not thread-safe which means that in the absence of external synchronization, ArrayDeque does not support concurrent access by multiple threads.
- Null elements are prohibited in the ArrayDeque.
- ArrayDeque class is likely to be faster than Stack when used as a stack.
- ArrayDeque class is likely to be faster than LinkedList when used as a queue.

Built-in Methods in ArrayDeque:

1. add(Element e) : The method inserts particular element at the end of the deque.
2. addFirst(Element e) : The method inserts particular element at the start of the deque.
3. addLast(Element e) : The method inserts particular element at the end of the deque. It is similar to add () method
4. clear() : The method removes all deque elements.
5. size() : The method returns the no. of elements in deque.
6. clone() : The method copies the deque.
7. contains(Obj) : The method checks whether a deque contains the element or not.
8. element() : The method returns element at the head of the deque
9. getFirst(): The method returns first element of the deque
10. getLast(): The method returns last element of the deque
11. isEmpty(): The method checks whether the deque is empty or not.
12. toArray(): The method returns array having the elements of deque.
13. offer(Element e) : The method inserts element at the end of deque.
14. offerFirst(Element e) : The method inserts element at the front of deque.
15. offerLast(Element e) : The method inserts element at the end of deque.
16. peek() : The method returns head element without removing it.
17. peekFirst() : The method returns first element without removing it.
18. peekLast() : The method returns last element without removing it.
19. poll() : The method returns head element and also removes it
20. pollFirst() : The method returns first element and also removes it
21. pollLast() : The method returns last element and also removes it
22. pop() : The method pops out an element for stack represented by deque
23. push(Element e) : The method pushes an element onto stack represented by deque
24. remove() : The method returns head element and also removes it
25. removeFirst() : The method returns first element and also removes it
26. removeLast() : The method returns last element and also removes it
27. removeFirstOccurrence(Obj) : The method removes the element where it first occur in the deque.
28. removeLastOccurrence(Obj) : The method removes the element where it last occur in the deque.

Accessing ArrayDeque from front-end.

```
import java.util.ArrayDeque;
class Demo
{
    public static void main(String[] args)
    {
        ArrayDeque dq = new ArrayDeque();
        dq.add(10);
        dq.add(20);
        dq.add(30);
        dq.add(40);
        System.out.println(dq);
        System.out.println(dq.pollFirst());
        System.out.println(dq.pollFirst());
        System.out.println(dq.pollFirst());
        System.out.println(dq.pollFirst());
    }
}
```

OUTPUT

```
[10, 20, 30, 40]
10
20
30
40
Press any key to continue . . .
```

Accessing ArrayDeque from rear-end.

```
import java.util.ArrayDeque;
class Demo55
{
    public static void main(String[] args)
    {
        ArrayDeque dq = new ArrayDeque();
        dq.add(10);
        dq.add(20);
        dq.add(30);
        dq.add(40);
        System.out.println(dq);
        System.out.println(dq.pollLast());
        System.out.println(dq.pollLast());
        System.out.println(dq.pollLast());
        System.out.println(dq.pollLast());
    }
}
```

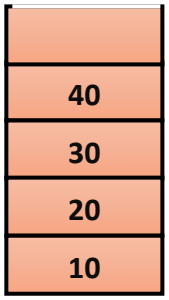
OUTPUT

```
[10, 20, 30, 40]
40
30
20
10
Press any key to continue . . .
```

Implementation of Stack using ArrayDeque

Stack can be implemented by using push() and pop() in ArrayDeque as shown below:

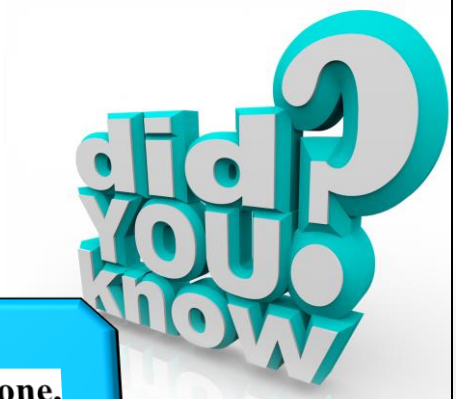
```
import java.util.ArrayDeque;
class Demo55
{
    public static void main(String[] args)
    {
        ArrayDeque dq = new ArrayDeque();
        dq.push(10);
        dq.push(20);
        dq.push(30);
        dq.push(40);
        System.out.println(dq);
        System.out.println(dq.pop());
        System.out.println(dq.pop());
        System.out.println(dq.pop());
        System.out.println(dq.pop());
    }
}
```



STACK

OUTPUT

```
[40, 30, 20, 10]
40
30
20
10
```



Technophobia is the fear of technology,
Nomophobia is the fear of being without a mobile phone,
Cyberphobia is the fear of computers.