Linked list implementation

```
/*boolean removeFirst()
  Removes  the first element from the list.*/
 public boolean removeFirst(){
    if (Size==0)
        return false; //empty list
    else if (Size==1) //one element inside list
        Front=Back=null;
    else
        Front=Front.next;

    Size--; //update size
    return true;
 }
```

Linked list implementation

```
 /*boolean removeLast()
   Removes  the last element from this list.*/
  public boolean removeLast(){
    if (Size==0)
         return false; //empty list
    else if (Size==1) // one element inside the list
         Front=Back=null;
    else{
        Node current= Front;
        for (int i=0;i<Size-2;i++)
           current=current.next;

       current.next=null;
       Back=current;
  }
    Size--; //update size
    return true;
}
```

Linked list implementation

```
 /*boolean remove(int index)
   * Removes the element at the specified position in the list*/
public boolean remove(int index){
   if (Size==0)return false;//empty linked list
   else if (index==0)return removeFirst(); //remove first element
   else if (index==size-1)return removeLast();//remove last element
   else if (index >0 && index<Size-1){
        Node current=Front;
        for (int i=0;i<index-1;i++)
            current=current.next;
        current.next= current.next.next;
        Size--;
        return true;
   }
    else return false; // out of boundary(invalid index)

}
```

Linked list implementation

```
 /*object remove(int index)
   * Removes the element at the specified position in the list*/
public object remove(int index){



                   Write the code here





}
```

Linked list implementation

```
 /*Print linked list == Traversing linked list  recursively*/
public void traverse (Node current){

    if (current!=null){
     System.out.println(current.element);
     traverse (current.next)


}
```

Linked list implementation

```
/** Remove the first node and
   *  return the object that is contained in the removed node. */

public Object removeFirst() {




          // Implementation left as an exercise



}
```

Linked list implementation

```
/** Remove the last node and
  * return the object that is contained in the removed node. */
 public Object removeLast() {

     // Implementation left as an exercise

     return null;
 }
```

Linked list implementation: H.W

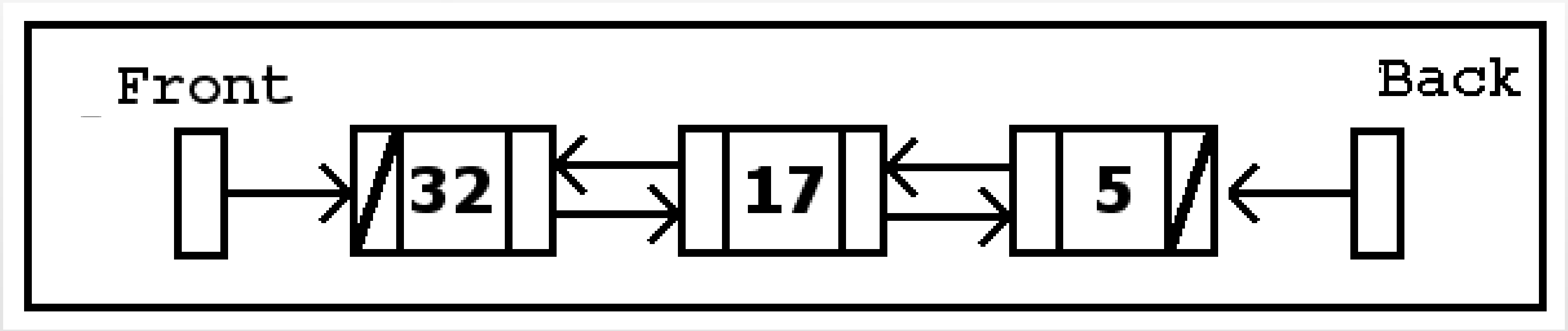
You have one week to do the following:

- /* void clear() Removes all of the elements from the list.*/
- int find (Object o); /* return the first index for the specified element in the list*/
- /*boolean remove(Object o) Removes the first occurrence of the specified element in the list*/.

Hint: use the implemented `int find (Object o);`
use the implemented `boolean remove(int index);` Removes the element at the specified position in the list

Double-linked list

- Add a **`prev`** pointer to our **`Node` class**
- Allows backward iteration
- some methods need to be modified
  - when adding or removing a node, we must fix the **`prev`** and **`next`** pointers to have the correct value!
  - can make it easier to implement some methods such as `remove`



Double-linked list

Double-linked list

```
/* Stores one element of a linked list. */
Public  class Node
{



}
```

Double-linked list

```
/* Stores one element of a linked list. */
Public  class Node
{  public Object element;
   public Node prev, next;

   public Node(Object element) {
         this(element, null, null);
     }

   public Node(Object element, Node prev, Node next){
            this.element = element;
            this.prev = prev;
            this.next = next;
  }
}
```

Double-linked list

```
/* Models an entire linked list. */

 public class DoubleLinkedList {
   private Node Front, Back;
   private int Size;
 public DoublyLinkedList() {

     Front = null;
     Back =  null;
     Size = 0;
  }
}
```

Double-linked list

```
/* void addFirst(Object o)
 * Inserts the given element at the beginning of the list. */
public void addFirst(Object element){
    Node newNode;
    newNode= new Node (element);

}
```

Double-linked list

```
/* void addFirst(Object o)
 * Inserts the given element at the beginning of the list. */
public void addFirst(Object element){
    Node newNode;
    newNode= new Node (element);
    if (Size==0)
      Front=Back=newNode;
    else {
       newNode.next=Front;
       Front.prev=newNode;
       Front=newNode;
   }
   Size++;

}
```

Double-linked list : H.W

You have one week to do the following:

- ❑ **/*boolean removeLast()**
  Removes the last element from the list.*/

- ❑ **/* void add(int index, Object element) Inserts the specified element at the**
  *** specified position index in this list.**
  ***/**

Double-linked list : H.W

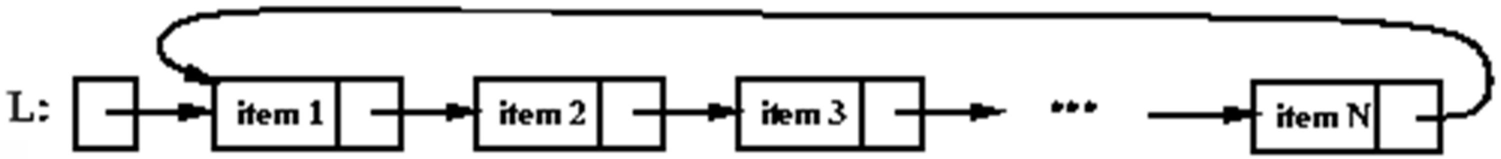
You have one week to do the following:

- **/*boolean removeLast()**
  Removes the last element from the list.*/

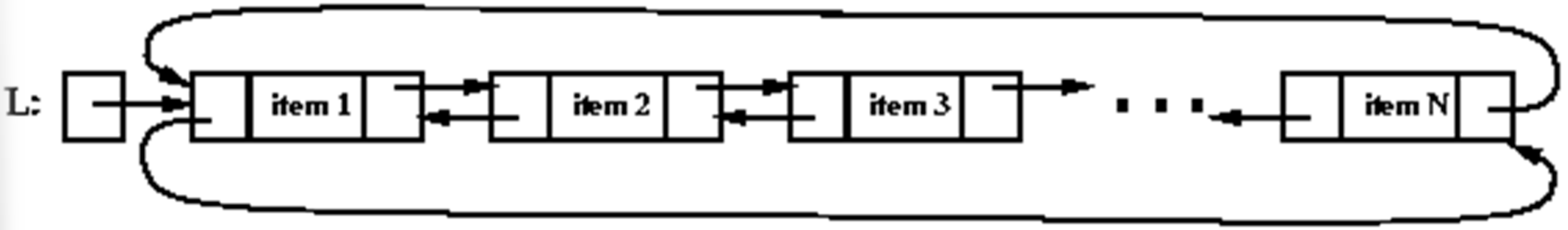
- **/* void add(int index, Object element) Inserts the specified element at the**
  *** specified position index in this list.**
  ***/**

Circular linked lists

Circular, singly linked list:



Circular, doubly linked list:



Extra Exercises

Implement all the non-implemented methods for :

- **Linked List**
- **Double Linked List**

Examples:

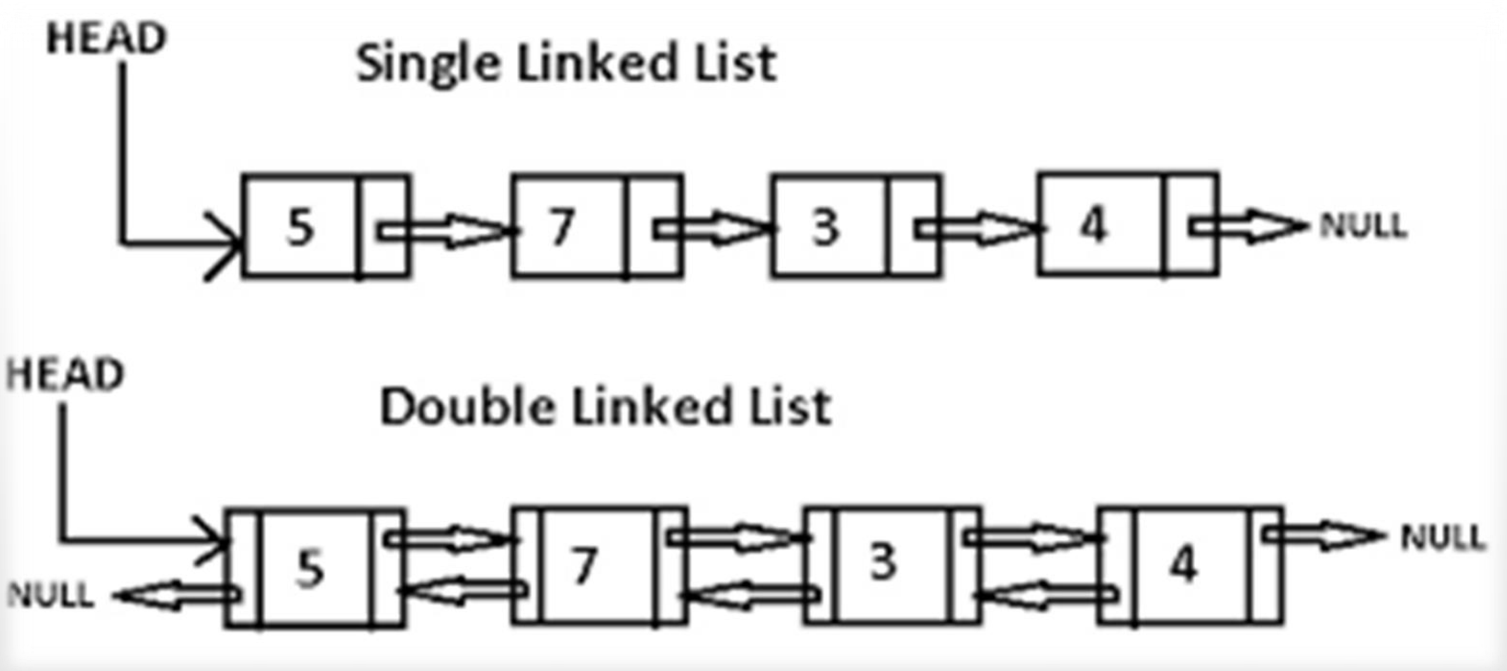
boolean contains(Object o) //Returns true if the list contains the specified element.

int lastIndexOf(Object o) //Returns the index in the list of the last occurrence of the specified element, or -1 if the list does not contain this element.

void printList (); // print all the list element
void printList in revers(); // print the list elements in reverse order
……

Extra Exercises

Write the Node Class for the Circular Linked-List (single and double list)


HEAD
Single Linked List
5
7
3
4
NULL
HEAD
Double Linked List
NULL
5
7
3
4
NULL