```
class ListNode {

      int data;
      ListNode next;

      public ListNode(int data){
            this.data = data;
            this.next = null;
      }

      public int getData() {
            return data;
      }

      public ListNode getNext() {
            return next;
      }
}

public class SingleLinkedList {

      ListNode head;

      public int getLength(){
            int count = 0;
            if(head == null)
                  count = 0;
            else {
                  ListNode current = head;
                  while(current!=null){
                        count++;
                        current = current.next;
                  }
            }
            return count;
      }

      public void insertAtBeg(int data){

            ListNode temp = new ListNode(data);

            if(head == null){
                  head = temp;
            }
            else {
                  temp.next = head;
                  head = temp;
            }
      }

      public void insertAtEnd(int data){

            ListNode temp = new ListNode(data);
            if(head == null){
                  head = temp;
```

```
        }
        else {
            ListNode current = head;
            while (current.next != null) {
                current = current.next;
            }

            current.next = temp;
        }
    }

    public void insertAtPos(int data, int position){

        if(position == 1){
            insertAtBeg(data);
        }
        else {
            int index = 1;
            ListNode current = head;
            while (index < position - 1) {
                current = current.next;
                index++;
            }
            ListNode temp = new ListNode(data);
            temp.next = current.next;
            current.next = temp;
        }
    }

    public int deleteFromBeg(){
        if(head == null){
            System.out.println("List is empty");
            return -1;
        }
        else {
            ListNode temp;
            temp = head;
            head = head.next;
            temp.next = null;
            return temp.data;
        }
    }

    public int deleteFromEnd(){
        if(head == null){
            System.out.println("List is empty");
            return -1;
        }
        else {
            ListNode current = head;
            ListNode temp;
            while(current.next.next != null){
                current = current.next;
            }
            temp = current.next;
```

```
			current.next = null;
			return temp.data;
		}
	}

	public int deleteAtPos(int position){

		if(head == null){
			System.out.println("List is Empty");
			return -1;
		}
		else if(position > getLength()){
			System.out.println("Position > length of the list");
			return -1;
		}
		else if(position == 1){
			return deleteFromBeg();
		}
		else if(position == getLength()){
			return deleteFromEnd();
		}
		else{
			ListNode current = head;
			int index = 1;
			while(index < position-1){
				current = current.next;
				index++;
			}
			ListNode temp = current.next;
			current.next = current.next.next;
			temp.next = null;
			return temp.data;
		}
	}

	public void printList(){
		if(head == null){
			System.out.println("List is empty");
		}
		else{
			ListNode current = head;
			while(current != null){
				System.out.print(current.data + " ");
				current = current.next;
			}
			System.out.println();
		}
	}

	public static void main(String args[]){
		SingleLinkedList sll = new SingleLinkedList();
		sll.insertAtBeg(1);
		sll.insertAtEnd(2);
		sll.insertAtEnd(3);
		sll.insertAtEnd(5);
```

```
            sll.insertAtPos(4,4);
            sll.printList();
            sll.deleteAtPos(3);
            sll.printList();
      }
}
```

```
package linkedList;

public class ReverseLL {

    Node head;

    void push(int data) {

        Node newNode = new Node(data);

        newNode.next = head;

        head = newNode;
    }

    Node reverseIterative(Node head){
        Node prev = null;
        Node curr = head;
        Node next = null;

        while(curr != null){
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }

        head = prev;
        return head;
    }

    public Node reverseInPairs(Node p){
        if(p==null || p.next==null){
            System.out.println("Either list is empty or there is just one
element, so reverse in pairs not possible...");
            return null;
        }

        Node curr = p;
        Node next = curr.next;
        Node temp = curr.next;
        while(curr != null){
            curr.next = next.next;
            next.next = curr;
```

```java
            curr = curr.next;
        }
        return temp;
    }

    public Node partitionLL(Node head, int K){

        Node rootHead = new Node(0);
        Node root = rootHead;
        Node prev = null;
        Node temp;
        if(head.data >= K){
            temp = head;
            root.next = temp;
            head = head.next;
            root.next.next = null;
            root = root.next;
        }
        Node curr = head;
        Node list = head;

        while(curr != null){
            if(curr.data >= K) {
                temp = curr;
                if (prev != null) {
                    prev.next = curr.next;
                }
                root.next = temp;
                temp.next = null;
                root = root.next;
                curr = prev.next;
            }
            else{
                prev = curr;
                curr = curr.next;
            }
        }

        prev.next = rootHead.next;
        return list;
    }

    public Node rotateLLByK(Node head,int K){
        if(head == null || head.next == null){
            return null;
        }
        Node rotateEnd = head;
        Node rotateStart = head;
        while (K > 0){
            rotateEnd = rotateEnd.next;
            if (rotateEnd == null){
                rotateEnd = head;
            }
            K--;
        }
        if(rotateEnd == rotateStart)
            return head;
        while(rotateEnd.next != null){
            rotateEnd = rotateEnd.next;
            rotateStart = rotateStart.next;
        }

        Node temp = rotateStart.next;
        rotateEnd.next = head;
        rotateStart.next = null;
        head = temp;
        return head;
    }

    public Node reverseInKGroups(Node head, int k){
        int count = k;
        Node prev = null;
        Node next = null;
        Node current = head;

        while(current != null && count>0){
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
            count--;
        }

        if(next != null)
            head.next = reverseInKGroups(next,k);

        return prev;
    }


    void printList(Node head){
        while (head != null){
            System.out.print(head.data + " ---> ");
            head = head.next;
        }
        System.out.print("NULL");
        System.out.print('\n');
    }

    public static void main(String[] args) {
        ReverseLL linkedList = new ReverseLL();
        linkedList.push(1);
        linkedList.push(2);
        linkedList.push(3);
        linkedList.push(4);
        linkedList.push(5);
        linkedList.push(6);
        linkedList.push(7);
        linkedList.push(8);
        linkedList.push(9);
        linkedList.push(10);

        linkedList.printList(linkedList.head);
        System.out.println("K groups reverse: ");
        Node newHead = linkedList.reverseInKGroups(linkedList.head,3);
        linkedList.printList(newHead);

        //Node newHead = linkedList.reverseIterative(linkedList.head);
        //Node newHead = linkedList.reverseInPairs(linkedList.head);
        //Node newHead = linkedList.partitionLL(linkedList.head,10);
        //linkedList.printList(newHead);
        //Node rotated = linkedList.rotateLLByK(newHead,2);
        //linkedList.printList(rotated);


    }

}



class ListNode{
      int data;
      ListNode next;

      public ListNode(int data){
            this.data = data;
            this.next = null;
      }
}

class ReverseSLL {

      ListNode head;

      public void insertAtEnd(int data){

            ListNode temp = new ListNode(data);
            if(head == null){
                  head = temp;
            }
            else {
                  ListNode current = head;
                  while (current.next != null) {
                        current = current.next;
                  }

                  current.next = temp;
            }
      }

      public ListNode nthNodeFromEnd(int n){
            if(head != null){
                  ListNode mainNode = head;
                  ListNode traverseNode = head;
                  int count = n;
                  while(count > 0){
                        if(traverseNode == null){
                              System.out.println(n + "is greater than length
of linked list");
                        }
                        traverseNode = traverseNode.next;
                        count--;
                  }

                  while(traverseNode != null){
                        mainNode = mainNode.next;
                        traverseNode = traverseNode.next;
                  }

                  return mainNode;
            }
            return null;
      }

      public ListNode middleNode(ListNode p){
            if(p != null){
                  ListNode slowPtr = p;
                  ListNode fastPtr = p;
                  while(fastPtr.next != null && fastPtr.next.next != null){
                        fastPtr = fastPtr.next.next;
                        slowPtr = slowPtr.next;
                  }
                  return slowPtr;
            }
            return null;
      }

      public void printLLRecur(ListNode p){
            if(p == null){
                  return;
            }

            System.out.print(p.data + " ");
            printLLRecur(p.next);
      }

      public void printLLReverse(ListNode p){
            if(p == null){
                  return;
            }

            printLLReverse(p.next);
            System.out.print(p.data + " ");
      }

      public void reverseIterative(){
            if(head == null)
                  return;
            ListNode prev = null;
            ListNode current = head;
            ListNode next = null;

            while(current!=null){
                  next = current.next;
                  current.next = prev;
                  prev = current;
                  current = next;
            }

            head = prev;

      }

      public void reverseRecursive(ListNode p){
            if(p.next == null){
                  head = p;
                  return;
            }
            reverseRecursive(p.next);
            ListNode q = p.next;
            q.next = p;
            p.next = null;
      }

      public ListNode reverseInPairs(ListNode p){
            if(p==null || p.next==null){
                  System.out.println("Either list is empty or there is just
one element, so reverse in pairs not possible...");
                  return null;
            }

            ListNode curr = p;
            ListNode next = curr.next;
            ListNode temp = curr.next;
            while(curr != null){
                  curr.next = next.next;
                  next.next = curr;
                  curr = curr.next;
            }
            return temp;
      }

      public static void main(String args[]){
            ReverseSLL rSll = new ReverseSLL();
            rSll.insertAtEnd(1);
        rSll.insertAtEnd(2);
        rSll.insertAtEnd(3);
        rSll.insertAtEnd(4);
        rSll.insertAtEnd(5);
        rSll.insertAtEnd(6);
        rSll.insertAtEnd(7);

        rSll.printLLRecur(rSll.head);
        System.out.println();
        ListNode newHead = rSll.reverseInPairs(rSll.head);
        rSll.printLLRecur(newHead);
        System.out.println();

        /*rSll.printLLReverse(rSll.head);
        System.out.println();

        ListNode middle = rSll.middleNode(rSll.head);
        if(middle != null){
            System.out.println("middle node of the linked list: " +
middle.data);
        }

        rSll.reverseIterative();
        rSll.printLLRecur(rSll.head);
        System.out.println();

        rSll.reverseRecursive(rSll.head);
        rSll.printLLRecur(rSll.head);
        System.out.println();

        ListNode nthNode = rSll.nthNodeFromEnd(3);
        if(nthNode != null){
            System.out.println("3rd node from end: " + nthNode.data);
        }*/


      }

}



package linkedList;

public class mergeTwoSLL {

    public ListNode mergeTwoSLLInOneLL(ListNode head1, ListNode head2){
        ListNode head = new ListNode(0);
        ListNode curr = head;
        ListNode next1,next2;
        while(head1!=null && head2!=null){
            if(head1.data <= head2.data){
                curr.next = head1;
                next1 = head1.next;
                head1.next = null;
                head1 = next1;
                curr = curr.next;
            }
            else {
                curr.next = head2;
                next2 = head2.next;
                head2.next = null;
                head2 = next2;
                curr = curr.next;
            }
        }
```

```java
        while(head1!=null){
            curr.next = head1;
            next1 = head1.next;
            head1.next = null;
            head1 = next1;
            curr = curr.next;
        }
        while(head2!=null){
            curr.next = head2;
            next2 = head2.next;
            head2.next = null;
            head2 = next2;
            curr = curr.next;
        }

        return head.next;
    }

    public void printList(ListNode p){
        if(p == null){
            System.out.println("List is empty");
        }
        else{
            ListNode current = p;
            while(current != null){
                System.out.print(current.data + " ");
                current = current.next;
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        SingleLinkedList sll1 = new SingleLinkedList();
        sll1.insertAtBeg(1);
        sll1.insertAtEnd(3);
        sll1.insertAtEnd(5);
        sll1.insertAtEnd(7);
        sll1.printList();
        SingleLinkedList sll2 = new SingleLinkedList();
        sll2.insertAtBeg(2);
        sll2.insertAtEnd(4);
        sll2.insertAtEnd(6);
        sll2.insertAtEnd(8);
        sll2.printList();

        mergeTwoSLL mergeTwoSLL = new mergeTwoSLL();
        ListNode p = mergeTwoSLL.mergeTwoSLLInOneLL(sll1.head,sll2.head);
        mergeTwoSLL.printList(p);

    }
}
```

```
class ListNode {
      int data;
      ListNode next;

      public ListNode(int data){
            this.data = data;
            this.next = null;
      }
}

class InsertionSortLL {

      public void printLL(ListNode head){
            ListNode curr = head;
            while(curr != null){
                  System.out.print(curr.data + " ");
                  curr = curr.next;
            }
            System.out.println();
      }

      public ListNode insertionSortLL(ListNode head){
            if(head == null || head.next == null)
                  return head;

            ListNode newHead = new ListNode(head.data);
            ListNode pointer = head.next;
            while(pointer != null){
                  ListNode innerPointer = newHead;
                  ListNode next = pointer.next;

                  if(pointer.data <= newHead.data){
                        ListNode oldHead = newHead;
                        newHead = pointer;
                        newHead.next = oldHead;
                  }
                  else {
                        while(innerPointer.next != null){
                              if(pointer.data > innerPointer.data &&
pointer.data <= innerPointer.next.data){
                                    ListNode oldNode = innerPointer.next;
                                    innerPointer.next = pointer;
                                    pointer.next = oldNode;
                              }
                              innerPointer = innerPointer.next;
                        }

                        if(innerPointer.next == null && pointer.data >
innerPointer.data){
                              innerPointer.next = pointer;
                              pointer.next = null;
                        }
                  }

                  pointer = next;
            }

            return newHead;
      }

      public static void main(String args[]){
            ListNode head = new ListNode(4);
            head.next = new ListNode(3);
            head.next.next = new ListNode(1);
            head.next.next.next = new ListNode(7);
            head.next.next.next.next = new ListNode(2);

            InsertionSortLL insertionSortLL = new InsertionSortLL();
            System.out.println("Original List: ");
            insertionSortLL.printLL(head);

            ListNode sortedHead = insertionSortLL.insertionSortLL(head);
            System.out.println("Sorted List: ");
            insertionSortLL.printLL(sortedHead);

      }
}



class ListNode{
      int data;
      ListNode next;
      ListNode random;

      public ListNode(int data){
            this.data = data;
            this.next = null;
            this.random = null;
      }
}

class CloneClass {

      public ListNode cloneLLWithRandomPointer(ListNode head){
            ListNode curr = head;
            ListNode next;
            while(curr != null){
                  next = curr.next;
                  ListNode newNode = new ListNode(curr.data);
                  newNode.next = next;
                  curr.next = newNode;
                  curr = next;
            }

            curr = head;
            ListNode cloneHead = curr.next;

            while(curr != null){
                  curr.next.random = curr.random.next;
```

```
			if(curr.next != null){
				curr = curr.next.next;
			}
			else {
				curr = curr.next;
			}
		}

		curr = head;
		ListNode cloneNode = cloneHead;

		while(curr != null && cloneNode != null){
			if(curr.next != null){
				curr.next = curr.next.next;
			}
			else{
				curr.next = curr.next;
			}
			if(cloneNode.next != null){
				cloneNode.next = cloneNode.next.next;
			}
			else {
				cloneNode.next = cloneNode.next;
			}
			curr = curr.next;
			cloneNode = cloneNode.next;
		}

		return cloneHead;
	}

	public void printRandomList(ListNode head){
		ListNode curr = head;
		while(curr != null){
			System.out.println("Node: " + "curr.data-> " + curr.data +
" and random.data->" + curr.random.data);
			curr = curr.next;
		}
	}

	public static void main(String args[]){
		ListNode head = new ListNode(1);
		head.next = new ListNode(2);
		head.next.next = new ListNode(3);
		head.next.next.next = new ListNode(4);
		head.next.next.next.next = new ListNode(5);

		head.random = head.next.next;
		head.next.random = head.next.next.next.next;
		head.next.next.random = head;
		head.next.next.next.random = head.next.next.next;
		head.next.next.next.next.random = head.next.next;

		CloneClass cc = new CloneClass();
```

```
            System.out.println("Original List: ");
            cc.printRandomList(head);

            ListNode cloneHead = cc.cloneLLWithRandomPointer(head);
            System.out.println("Cloned List: ");
            cc.printRandomList(cloneHead);

      }
}
```