**1. Write a java program to demonstrate the working of Singly Linked List.**

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class SinglyLinkedList {
    Node head;
    Node tail;
    public SinglyLinkedList() {
        head = null;
        tail = null;
    }
    public boolean isEmpty() {
        return (head == null);
    }
    public void insert(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = tail = newNode;
            tail.next = head; // Circular link
        } else {
            tail.next = newNode;
            tail = newNode;
            tail.next = head; // Circular link
        }
    }
    public void insertAtHead(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = tail = newNode;
            tail.next = head; // Circular link
        } else {
            newNode.next = head;
            head = newNode;
            tail.next = head; // Update tail's link
        }
    }
    public void insertAtTail(int data) {
        insert(data); // Reuse the `insert` logic
    }
```

```
public void insertAtPosition(int pos, int data) {
    Node newNode = new Node(data);
    int totalNodes = countNodes();

    if (isEmpty()) {
        head = tail = newNode;
        tail.next = head; // Circular link
    } else if (pos == 1) {
        insertAtHead(data);
    } else if (pos > 1 && pos <= totalNodes + 1) {
        Node current = head, prev = null;
        for (int i = 1; i < pos; i++) {
            prev = current;
            current = current.next;
        }
        newNode.next = current;
        if (prev != null) {
            prev.next = newNode;
        }
        if (pos == totalNodes + 1) { // If inserted at the tail
            tail = newNode;
        }
        tail.next = head; // Ensure circular link
    } else {
        System.out.println("Invalid Node Position!!!");
    }
}
public void deleteAtHead() {
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!!!");
    } else if (head == tail) { // Single node case
        head = tail = null;
    } else {
        head = head.next;
        tail.next = head; // Maintain circular link
    }
}
public void deleteAtTail() {
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!!!");
    } else if (head == tail) { // Single node case
        head = tail = null;
    } else {
        Node current = head;
        while (current.next != tail) {
            current = current.next;
        }
        current.next = head; // Circular link
```

```
            tail = current;
        }
    }

    public void deleteAtPosition(int pos) {
        int totalNodes = countNodes();
        if (isEmpty()) {
            System.out.println("Circular Linked List is empty!");
        } else if (pos == 1) {
            deleteAtHead();
        } else if (pos > 1 && pos <= totalNodes) {
            Node current = head, prev = null;
            for (int i = 1; i < pos; i++) {
                prev = current;
                current = current.next;
            }
            prev.next = current.next;
            if (current == tail) { // If deleting the tail node
                tail = prev;
                tail.next = head; // Maintain circular link
            }
        } else {
            System.out.println("Invalid Node Position!!!");
        }
    }
    public void displayList() {
        if (isEmpty()) {
            System.out.println("The Circular Linked List is Empty");
            return;
        }
        Node current = head;
        do {
            System.out.print(current.data + " --> ");
            current = current.next;
        } while (current != head);
        System.out.println("(Back to Head)");
    }
    public int countNodes() {
        if (isEmpty()) {
            return 0;
        }
        int count = 0;
        Node current = head;
        do {
            count++;
            current = current.next;
        } while (current != head);
        return count;
```

```
}
public void reverseList() {
    if (isEmpty()) {
        System.out.println("List is Empty");
        return;
    }
    Node prev = null, current = head, next;
    tail = head;
    do {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    } while (current != head);
    head = prev;
    tail.next = head; // Maintain circular link
}
public void search(int key) {
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!");
        return;
    }
    Node current = head;
    int pos = 1, flag = 0;
    do {
        if (current.data == key) {
            System.out.println(key + " found at position " + pos);
            flag = 1;
        }
        current = current.next;
        pos++;
    } while (current != head);
    if (flag == 0) {
        System.out.println(key + " not found!!");
    }
}
public static void main(String[] args) {
    SinglyLinkedList cll = new SinglyLinkedList();
    Scanner scanner = new Scanner(System.in);
    int choice, data, pos;
    do {
        System.out.println("\n---- Circular Linked List Operations ----");
        System.out.println("1. Insert");
        System.out.println("2. Insert At Head");
        System.out.println("3. Insert At Tail");
        System.out.println("4. Insert At Position");
        System.out.println("5. Delete At Head");
        System.out.println("6. Delete At Tail");
```

```
System.out.println("7. Delete At Position");
System.out.println("8. Reverse List");
System.out.println("9. Search for Key");
System.out.println("10. Total Nodes");
System.out.println("11. Display List");
System.out.println("12. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();

switch (choice) {
   case 1:
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insert(data);
      cll.displayList();
      break;
   case 2:
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insertAtHead(data);
      cll.displayList();
      break;
   case 3:
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insertAtTail(data);
      cll.displayList();
      break;
   case 4:
      System.out.print("Enter the position: ");
      pos = scanner.nextInt();
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insertAtPosition(pos, data);
      cll.displayList();
      break;
   case 5:
      cll.deleteAtHead();
      cll.displayList();
      break;
   case 6:
      cll.deleteAtTail();
      cll.displayList();
      break;
   case 7:
      System.out.print("Enter the position to delete: ");
      pos = scanner.nextInt();
      cll.deleteAtPosition(pos);
```

```
                cll.displayList();
                break;
            case 8:
                cll.reverseList();
                cll.displayList();
                break;
            case 9:
                System.out.print("Enter the key to search: ");
                data = scanner.nextInt();
                cll.search(data);
                break;
            case 10:
                System.out.println("Total Nodes: " + cll.countNodes());
                break;
            case 11:
                cll.displayList();
                break;
            case 12:
                System.out.println("Exiting program...");
                break;
            default:
                System.out.println("Invalid choice! Try again.");
        }
    } while (choice != 12);
    scanner.close();
  }
}
```

**Output:**

```
1. Insert
2. Insert At Head
3. Insert At Tail
4. Insert At Position
5. Delete At Head
6. Delete At Tail
7. Delete At Position
8. Reverse List
9. Search for Key
10. Total Nodes
11. Display List
12. Exit
Enter your choice: 1
Enter a data: 10
10 --> (Back to Head)
```

```
Enter your choice: 1
Enter a data: 15
10 --> 15 --> (Back to Head)
```

```
Enter your choice: 9
Enter the key to search: 5
5 found at position 4
```

```
Enter your choice: 10
Total Nodes: 4
```

```
Enter your choice: 11
20 --> 15 --> 10 --> 5 --> (Back to Head)
```

```
Enter your choice: 5
15 --> 10 --> 5 --> (Back to Head)
```

```
Enter your choice: 6
15 --> 10 --> (Back to Head)
```

```
Enter your choice: 2
Enter a data: 5
5 --> 10 --> 15 --> (Back to Head)
```

```
Enter your choice: 3
Enter a data: 20
5 --> 10 --> 15 --> 20 --> (Back to Head)
```

```
Enter your choice: 7
Enter the position to delete: 2
15 --> (Back to Head)
```

```
Enter your choice: 12
Exiting program...
```

```
Enter your choice: 8
20 --> 15 --> 10 --> 5 --> (Back to Head)
```

**2. Write a java program to demonstrate the working of Circular Linked List.**

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class CircularLinkedList {
    Node head;
    Node tail;

    public CircularLinkedList() {
        head = null;
        tail = null;
    }
    public boolean isEmpty() {
        return (head == null);
    }
    public void insert(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
              newNode.next=newNode;
            head = tail = newNode;
        } else {
              newNode.next=head;
            tail.next = newNode;
            tail=newNode;
        }
    }
    public void insertAtHead(int data) {
        Node newNode = new Node(data);
```

```
    if (isEmpty()) {
          newNode.next=newNode;
        head = tail = newNode;
    } else {
        newNode.next = head;
        head = newNode;
        tail.next = head;
    }
}
public void insertAtTail(int data) {
    insert(data);
}
public void insertAtPosition(int pos, int data)
{
    Node newNode = new Node(data);
    int totalNodes = countNodes();

    if (isEmpty()) {
        head = tail = newNode;
        tail.next = head;
    } else if (pos == 1) {
        insertAtHead(data);
    } else if (pos > 1 && pos <= totalNodes + 1) {
        Node current = head, prev = null;
        for (int i = 1; i < pos; i++) {
            prev = current;
            current = current.next;
        }
        newNode.next = current;
        if (prev != null) {
            prev.next = newNode;
        }
        if (pos == totalNodes + 1) {
            tail = newNode;
        }
        tail.next = head;
    } else {
        System.out.println("Invalid Node Position!!!");
    }
}
public void deleteAtHead() {
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!!!");
    } else if (head == tail) {
        head = tail = null;
    } else {
        head = head.next;
        tail.next = head;
```

```
    }
}
public void deleteAtTail() {
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!!!");
    } else if (head == tail) {
        head = tail = null;
    } else {
        Node current = head;
        while (current.next != tail) {
            current = current.next;
        }
        current.next = head;
        tail = current;
    }
}
public void deleteAtPosition(int pos) {
    int totalNodes = countNodes();
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!");
    } else if (pos == 1) {
        deleteAtHead();
    } else if (pos > 1 && pos <= totalNodes) {
        Node current = head, prev = null;
        for (int i = 1; i < pos; i++) {
            prev = current;
            current = current.next;
        }
        prev.next = current.next;
        if (current == tail) {
            tail = prev;
            tail.next = head;
        }
    } else {
        System.out.println("Invalid Node Position!!!");
    }
}
public void displayList() {
    if (isEmpty()) {
        System.out.println("The Circular Linked List is Empty");
        return;
    }
    Node current = head;
    do {
        System.out.print(current.data + " --> ");
        current = current.next;
    } while (current != head);
    System.out.println("(Back to Head)");
```

```
}
public int countNodes() {
    if (isEmpty()) {
        return 0;
    }
    int count = 0;
    Node current = head;
    do {
        count++;
        current = current.next;
    } while (current != head);
    return count;
}
public void search(int key) {
    if (isEmpty()) {
        System.out.println("Circular Linked List is empty!");
        return;
    }
    Node current = head;
    int pos = 1; boolean found=false;
    do {
        if (current.data == key) {
            System.out.println(key + " found at position " + pos);
            found=true;
        }
        current = current.next;
        pos++;
    } while (current != head);
    if (!found) {
        System.out.println(key + " not found!!");
    }
}
public static void main(String[] args) {
    CircularLinkedList cll = new CircularLinkedList();
    Scanner scanner = new Scanner(System.in);
    int choice, data, pos;
    do {
        System.out.println("\n---- Circular Linked List Operations ----");
        System.out.println("1. Insert");
        System.out.println("2. Insert At Head");
        System.out.println("3. Insert At Tail");
        System.out.println("4. Insert At Position");
        System.out.println("5. Delete At Head");
        System.out.println("6. Delete At Tail");
        System.out.println("7. Delete At Position");
        System.out.println("8. Search for Key");
        System.out.println("9. Total Nodes");
        System.out.println("10. Display List");
```

```
System.out.println("11. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();
switch (choice) {
   case 1:
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insert(data);
      cll.displayList();
      break;
   case 2:
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insertAtHead(data);
      cll.displayList();
      break;
   case 3:
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insertAtTail(data);
      cll.displayList();
      break;
   case 4:
      System.out.print("Enter the position: ");
      pos = scanner.nextInt();
      System.out.print("Enter a data: ");
      data = scanner.nextInt();
      cll.insertAtPosition(pos, data);
      cll.displayList();
      break;
   case 5:
      cll.deleteAtHead();
      cll.displayList();
      break;
   case 6:
      cll.deleteAtTail();
      cll.displayList();
      break;
   case 7:
      System.out.print("Enter the position to delete: ");
      pos = scanner.nextInt();
      cll.deleteAtPosition(pos);
      cll.displayList();
      break;
   case 8:
      System.out.print("Enter the key to search: ");
      data = scanner.nextInt();
      cll.search(data);
```

```
                break;
            case 9:
                System.out.println("Total Nodes: " + cll.countNodes());
                break;
            case 10:
                cll.displayList();
                break;
            case 11:
                System.out.println("Exiting program...");
                break;
            default:
                System.out.println("Invalid choice! Try again.");
        }
    } while (choice != 11);
    scanner.close();
  }
}
```

**Output:**

```
---- Circular Linked List Operations ----
1. Insert
2. Insert At Head
3. Insert At Tail
4. Insert At Position
5. Delete At Head
6. Delete At Tail
7. Delete At Position
8. Search for Key
9. Total Nodes
10. Display List
11. Exit
Enter your choice: 1
Enter a data: 10
10 --> (Back to Head)
```

```
Enter your choice: 2
Enter a data: 5
5 --> 10 --> (Back to Head)
```

```
Enter your choice: 3
Enter a data: 20
5 --> 10 --> 20 --> (Back to Head)
```

```
Enter your choice: 4
Enter the position: 3
Enter a data: 15
5 --> 10 --> 15 --> 20 --> (Back to Head)
```

```
Enter your choice: 5
10 --> 15 --> 20 --> (Back to Head)
```

```
Enter your choice: 6
10 --> 15 --> (Back to Head)
```

```
Enter your choice: 7
Enter the position to delete: 2
10 --> (Back to Head)
```

```
Enter your choice: 8
Enter the key to search: 10
10 found at position 2
```

```
Enter your choice: 9
Total Nodes: 4
```

```
Enter your choice: 10
5 --> 10 --> 15 --> 20 --> (Back to Head)
```

```
Enter your choice: 11
Exiting program...
```

**3. Write a java program to demonstrate the working of Doubly Linked List.**

```
import java.util.Scanner;
class Node {
    int data;
    Node prev;
    Node next;

    public Node(int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}
public class DoublyLinkedList {
    Node head;
    Node tail;

    public DoublyLinkedList() {
        head = null;
        tail = null;
    }
    public boolean isEmpty() {
        return head == null;
    }
    public void insertAtHead(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
    }
    public void insertAtTail(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
    }
    public void insertAtPosition(int pos, int data) {
        if (pos == 1) {
```

```
            insertAtHead(data);
            return;
        }
        int totalNodes = countNodes();
        if (pos > totalNodes + 1 || pos < 1) {
            System.out.println("Invalid Node Position!!!");
            return;
        }
        Node newNode = new Node(data);
        Node current = head;
        for (int i = 1; i < pos - 1; i++) {
            current = current.next;
        }
        newNode.next = current.next;
        newNode.prev = current;

        if (current.next != null) {
            current.next.prev = newNode;
        }
        current.next = newNode;
        if (newNode.next == null) { // If inserted at the tail
            tail = newNode;
        }
    }
    public void deleteAtHead() {
        if (isEmpty()) {
            System.out.println("Doubly Linked List is empty!");
        } else if (head == tail) { // Single node case
            head = tail = null;
        } else {
            head = head.next;
            head.prev = null;
        }
    }
    public void deleteAtTail() {
        if (isEmpty()) {
            System.out.println("Doubly Linked List is empty!");
        } else if (head == tail) { // Single node case
            head = tail = null;
        } else {
            tail = tail.prev;
            tail.next = null;
        }
    }
    public void deleteAtPosition(int pos) {
        if (pos == 1) {
            deleteAtHead();
            return;
```

```
        }
        int totalNodes = countNodes();
        if (pos > totalNodes || pos < 1) {
            System.out.println("Invalid Node Position!!!");
            return;
        }
        Node current = head;
        for (int i = 1; i < pos; i++) {
            current = current.next;
        }
        if (current.next != null) {
            current.next.prev = current.prev;
        }
        if (current.prev != null) {
            current.prev.next = current.next;
        }
        if (current == tail) { // If deleting the tail
            tail = current.prev;
        }
    }
    public void displayList() {
        if (isEmpty()) {
            System.out.println("Doubly Linked List is empty!");
            return;
        }

        Node current = head;
        System.out.print("Forward: ");
        while (current != null) {
            System.out.print(current.data + " <-> ");
            current = current.next;
        }
        System.out.println("null");
    }
    public void displayListReverse() {
        if (isEmpty()) {
            System.out.println("Doubly Linked List is empty!");
            return;
        }
        Node current = tail;
        System.out.print("Reverse: ");
        while (current != null) {
            System.out.print(current.data + " <-> ");
            current = current.prev;
        }
        System.out.println("null");
    }
    public int countNodes() {
```

```
    int count = 0;
    Node current = head;
    while (current != null) {
        count++;
        current = current.next;
    }
    return count;
}
public void search(int key) {
    if (isEmpty()) {
        System.out.println("Doubly Linked List is empty!");
        return;
    }
    Node current = head;
    int pos = 1;
    while (current != null) {
        if (current.data == key) {
            System.out.println(key + " found at position " + pos);
            return;
        }
        current = current.next;
        pos++;
    }
    System.out.println(key + " not found!");
}
public static void main(String[] args) {
    DoublyLinkedList dll = new DoublyLinkedList();
    Scanner scanner = new Scanner(System.in);
    int choice, data, pos;

    do {
        System.out.println("\n---- Doubly Linked List Operations ----");
        System.out.println("1. Insert At Head");
        System.out.println("2. Insert At Tail");
        System.out.println("3. Insert At Position");
        System.out.println("4. Delete At Head");
        System.out.println("5. Delete At Tail");
        System.out.println("6. Delete At Position");
        System.out.println("7. Display List");
        System.out.println("8. Display List Reverse");
        System.out.println("9. Search for Key");
        System.out.println("10. Total Nodes");
        System.out.println("11. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter data: ");
```

```
            data = scanner.nextInt();
            dll.insertAtHead(data);
            dll.displayList();
            break;
        case 2:
            System.out.print("Enter data: ");
            data = scanner.nextInt();
            dll.insertAtTail(data);
            dll.displayList();
            break;
        case 3:
            System.out.print("Enter position: ");
            pos = scanner.nextInt();
            System.out.print("Enter data: ");
            data = scanner.nextInt();
            dll.insertAtPosition(pos, data);
            dll.displayList();
            break;
        case 4:
            dll.deleteAtHead();
            dll.displayList();
            break;
        case 5:
            dll.deleteAtTail();
            dll.displayList();
            break;
        case 6:
            System.out.print("Enter position to delete: ");
            pos = scanner.nextInt();
            dll.deleteAtPosition(pos);
            dll.displayList();
            break;
        case 7:
            dll.displayList();
            break;
        case 8:
            dll.displayListReverse();
            break;
        case 9:
            System.out.print("Enter key to search: ");
            data = scanner.nextInt();
            dll.search(data);
            break;
        case 10:
            System.out.println("Total Nodes: " + dll.countNodes());
            break;
        case 11:
            System.out.println("Exiting program...");
```

```
                break;
            default:
                System.out.println("Invalid choice! Try again.");
        }
    } while (choice != 11);
    scanner.close();
  }
}
```

**Output:**

```
---- Doubly Linked List Operations
1. Insert At Head
2. Insert At Tail
3. Insert At Position
4. Delete At Head
5. Delete At Tail
6. Delete At Position
7. Display List
8. Display List Reverse
9. Search for Key
10. Total Nodes
11. Exit
Enter your choice: 1
Enter data: 5
Forward: 5 <-> null
```

```
Enter your choice: 2
Enter data: 15
Forward: 5 <-> 15 <-> null
```

```
Enter your choice: 3
Enter position: 2
Enter data: 10
Forward: 5 <-> 10 <-> 15 <-> null
```

```
Enter your choice: 8
Reverse: 15 <-> 10 <-> 5 <-> null
```

```
Enter your choice: 9
Enter key to search: 10
10 found at position 2
```

```
Enter your choice: 10
Total Nodes: 3
```

```
Enter your choice: 4
Forward: 10 <-> 15 <-> null
```

```
Enter your choice: 5
Forward: 10 <-> null
```

```
Enter your choice: 11
Exiting program...
```

```
Enter your choice: 7
Forward: 5 <-> 10 <-> 15 <-> null
```

**4. Write a java program to perform addition of two polynomials using Linked List.**

```
class Node{
        int coeff,pow;
        Node nextNode;
        public Node(int c, int p) {
                coeff = c;
                pow = p;
                nextNode=null;
        }
}
public class PolyAdd {
        static Node addPolynomial(Node head1,Node head2)
```

```
        {
                if (head1==null) return head2;
                if(head2==null) return head1;

                if(head1.pow>head2.pow)
                {
                        Node nextPtrNode =addPolynomial(head1.nextNode, head2);
                        head1.nextNode=nextPtrNode;
                        return head1;
                }
                else if (head1.pow<head2.pow) {
                        Node nextPtrNode=addPolynomial(head1, head2.nextNode);
                        head2.nextNode=nextPtrNode;
                        return head2;
                }
                Node nextPtrNode=addPolynomial(head1.nextNode, head2.nextNode);
                head1.coeff=head1.coeff+head2.coeff;
                return head1;
        }
        static void printList(Node head)
        {
                Node currNode=head;
                while (currNode!=null)
                {
                        System.out.print(currNode.coeff+"x^"+currNode.pow+"-->");
                        currNode=currNode.nextNode;
                }
                System.out.println();
        }
        public static void main(String[] args) {
                Node head1=new Node(4,3);
                head1.nextNode=new Node(3,2);
                head1.nextNode.nextNode=new Node(3, 0);

                Node head2=new Node(2 ,3);
                head2.nextNode=new Node(-7,0);

                Node head=addPolynomial(head1, head2);
                printList(head);
        }
}
```

**Output:**

```
6x^3-->3x^2-->-4x^0-->
```

**5. Write a java program to demonstrate the working of Stack using Linked List.**

```
class Node {
    int data;
    Node next;

    public Node(int d) {
        data =  d;
        this.next = null;
    }
}
class Stack {
    Node head;

    public Stack() {
        this.head = null;
    }
    Boolean isEmpty() {
        return head == null;
    }
    void push(int d) {
        Node newNode = new Node(d);
        if (newNode == null) {
            System.out.println("\nStack Overflow!");
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    void pop() {
        if (isEmpty())  {
            System.out.println("\nStack underflow");
            return;
        } else {
            Node temp = head;
            head = head.next; // Move the head to the next node
            temp = null;  // Free the old head
        }
    }
    int peek() {
        if (!isEmpty()) {
            return head.data;
```

```
        } else {
            System.out.println("\nStack is Empty!!");
            return Integer.MIN_VALUE;
        }
    }
    public void displayStack() {
        Node currentNode = head;
        if (head == null) {
            System.out.println("\nStack is empty");
            return;
        }
        System.out.println("\nThe data in Stack: ");
        while (currentNode != null) {
            System.out.print(currentNode.data + "--->");
            currentNode = currentNode.next;
        }
        System.out.print("null");
        System.out.println();
    }
}
public class StackSLL {
    public static void main(String[] args) {
        Stack st = new Stack();
        st.push(10);
        st.push(20);
        st.push(30);
        st.push(40);
        st.displayStack();
        System.out.println("Top element is " + st.peek());
        st.pop();
        st.displayStack();
        System.out.println("Top element is " + st.peek());
    }
}
```

**Output:**

```
The data in Stack:
40--->30--->20--->10--->null
Top element is 40

The data in Stack:
30--->20--->10--->null
Top element is 30
```

**6. Write a java program to demonstrate the working of Queue using Linked List.**

```
class Node {
    int data;
    Node next;

    public Node(int d) {
        data =  d;
        this.next = null;
    }
}
class Queue {
    Node frontNode, rearNode;

    public Queue() {
        frontNode = rearNode = null;
    }
    Boolean isEmpty() {
        return frontNode == null && rearNode == null;
    }
    void enqueue(int d) {
        Node newNode = new Node(d);
        if (rearNode == null) {
            frontNode = rearNode = newNode;
            return;
        }
        rearNode.next = newNode;
        rearNode = newNode;
    }
    void dequeue() {
        if (isEmpty()) {
            System.out.println("\nQueue underflow");
            return;
        }
        frontNode = frontNode.next;
        if (frontNode == null)
            rearNode = null;
    }
    int getFrontNode() {
        if (!isEmpty()) {
            return frontNode.data;
        } else {
```

```
            System.out.println("\nQueue is empty!");
            return Integer.MIN_VALUE;
        }
    }
    int getRearNode() {
        if (!isEmpty()) {
            return rearNode.data;
        } else {
            System.out.println("\nQueue is empty!");
            return Integer.MIN_VALUE;
        }
    }
    public void displayQueue() {
        if (isEmpty()) {
            System.out.println("\nQueue is empty!");
            return;
        }
        Node currentNode = frontNode;
        System.out.println("\nThe data in Queue: ");
        while (currentNode != null) {
            System.out.print(currentNode.data + " ---> ");
            currentNode = currentNode.next;
        }
        System.out.print("null\n");
    }
}
public class QueueSLL {
    public static void main(String[] args) {
        Queue queue = new Queue();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);

        queue.displayQueue();
        System.out.println("Front element is: " + queue.getFrontNode());
        System.out.println("Rear element is: " + queue.getRearNode());

        queue.dequeue();
        System.out.println("\nAfter dequeue:");
        queue.displayQueue();
        System.out.println("Front element is: " + queue.getFrontNode());
```

```
        System.out.println("Rear element is: " + queue.getRearNode());
    }
}
```

**Output:**

```
The data in Queue:
10 ---> 20 ---> 30 ---> 40 ---> null
Front element is: 10
Rear element is: 40

After dequeue:

The data in Queue:
20 ---> 30 ---> 40 ---> null
Front element is: 20
Rear element is: 40
```

**7. Write a java program to demonstrate the working of Priority Queue using Linked List.**

```
import java.security.PublicKey;

class Node{
        int data,priority;
        Node nextNode;
        public Node(int d, int p) {
                data = d;
                priority = p;
                nextNode=null;
        }
}
public class PriorityQueue {
        Node headNode;
        public PriorityQueue() {
                headNode=null;
        }
        public void add(int d,int p) {
                Node startNode=headNode;
                Node newNode=new Node(d, p);
                if (headNode==null) {
                        headNode=newNode;
                        return;
                }
```

```
                if (headNode.priority>p)
                {
                        newNode.nextNode=headNode;
                        headNode=newNode;
                }
                else {
                        while (startNode.nextNode!=null &&
startNode.nextNode.priority<p)
                        {
                                startNode=startNode.nextNode;
                        }
                        newNode.nextNode=startNode.nextNode;
                        startNode.nextNode=newNode;
                }
        }
        public Node removeNode()
        {
                Node tempNode=headNode;
                headNode=headNode.nextNode;
                tempNode=null;
                return headNode;
        }
        int getHeadData()
        {
                return headNode.data;
        }
        boolean isEmpty()
        {
                return headNode==null;

        }
        public void displayStack() {
      Node currentNode = headNode;
      if (headNode == null) {
         System.out.println("\nPriority Queue is empty");
         return;
      }
      System.out.println("\nThe data in Priority Queue: ");
      while (currentNode != null) {
         System.out.print(currentNode.data + "--->");
         currentNode = currentNode.nextNode;
      }
```

```
        System.out.print("null");
        System.out.println();
    }

        public static void main(String[] args)
        {
                PriorityQueue pQueue=new PriorityQueue();
                pQueue.add(10, 1);
                pQueue.add(5, 2);
                pQueue.add(6, 0);
                pQueue.add(8,-1);
                pQueue.displayStack();
                System.out.println("The deleted data is "+ pQueue.removeNode().data);
                System.out.println("Head node data "+pQueue.getHeadData());
                while (!pQueue.isEmpty()) {
                        System.out.print("remvoing nodes is
="+pQueue.getHeadData()+",");
                        pQueue.removeNode();
                }
                pQueue.displayStack();
        }

}
```

**Output:**

```
The data in Priority Queue:
8--->6--->10--->5--->null
The deleted data is 6
Head node data 6
remvoing nodes is =6,remvoing nodes is =10,remvoing nodes is =5,
Priority Queue  is empty
```

**8. Write a java program to demonstrate the working of Double Ended Queue using Linked List.**

**Code:**

```
import java.util.Scanner;
class Node {
    int data;
    Node next,prev;
```

```
  public Node(int data) {
     this.data = data;
     this.next = null;
     this.prev=null;
   }
}
public class DEQueue
{
        Node front;
        Node rear;
        int Size;
        public DEQueue()
        {
                front=rear=null;
                Size=0;
        }
        public boolean isEmpty() {
                return (front==null);
        }
        public int size() {return Size;}
        public void insertFront(int data)
        {
                Node newNode=new Node(data);
                if (newNode==null)
                        System.out.print("Overflow!\n");
                else {
                        if (front==null) {
                                rear=front=newNode;
                        }
                        else {
                                newNode.next=front;
                                front.prev=newNode;
                                front=newNode;
                        }
                        Size++;
                }  }
        public void insertRear(int data)
        {
                Node newNode=new Node(data);
                if (newNode==null)
                {
                        System.out.print("OverFlow!\n");
```

```
		}
		else {
			if (rear==null)
			{
				front=rear=newNode;
			}
			else {
				newNode.prev=rear;
				rear.next=newNode;
				rear=newNode;
			}
			Size++;
		}
	}
	public void deleteFront()
	{
		if (isEmpty()) {
			System.out.println("UnderFlow - Deque is Empty!\n");
		}
		else {
			Node tempNode = front;
			front=front.next;
			if (front==null)
			{
				rear=null;
			}
			else {
				front.prev=null;
			}
			Size--;
		}
	}
	void deleteRear()
	{
		if (isEmpty())
		{
			System.out.print("quot;UnderFlow - Deque is empty!\n");
		}
		else {
			Node tempNode=rear;
			rear=rear.prev;
```

```
                if (rear==null)
                {
                        front=null;
                }
                else {
                        rear.next=null;
                }
                Size--;
        }
}
public int getFront()
{
        if (isEmpty())
        {
                return -1;

        }
        return front.data;
}
public int getRear() {
        if (isEmpty())
        {
                return -1;
        }
        return rear.data;
}
public void display()
{
        Node currentNode=front;
        if (front==null)
        {
                System.out.println("The double ended queue is empty!");
                return;
        }
        System.out.println("The data are Double Ended Queue are: ");
        System.out.println("NULL<--");
        while (currentNode!=null)
        {
                System.out.println(currentNode.data+"<->");
                currentNode=currentNode.next;
        }
        System.out.println("Null-->");
```

```
            System.out.println();
    }
    public void deleteAllNodes()
    {
            rear=null;
            while (front!=null)
            {
                    Node tempNode=front;
                    front=front.next;
            }
            Size=0;
    }
    public static void main(String[] args) {
            DEQueue dq = new DEQueue();
            Scanner sc = new Scanner(System.in);
            int data;
            int choice;
            do
            {
            System.out.print("\n1.Insert At Front\n");
            System.out.print("2.Insert At Rear\n");
            System.out.print("3.Delete At Front\n");
            System.out.print("4.Delete At Rear\n");
            System.out.print("5.Delete all nodes\n");
            System.out.print("6.Display List\n");
            System.out.print("7.Exit\n");
            System.out.print("Enter your choice :" );
            choice = sc.nextInt();
            switch (choice)
            {
            case 1: System.out.print("\nInsert at Front - Enter data :");
                    data= sc.nextInt();
                    dq.insertFront(data);
                    dq.display();
                    break;
            case 2: System.out.print("\nInsert at Rear - Enter data: ");
                    data= sc.nextInt();
                    dq.insertRear(data);
                    dq.display();

                    break;
            case 3: System.out.print("\nDelete at Front: ");
```

```
            dq.deleteFront();
            dq.display();
            break;
        case 4: System.out.print("\nDelete at Rear: ");
            dq.deleteRear();
            dq.display();
            break;
        case 5: System.out.print("\nDelete all nodes : ");
            dq.deleteAllNodes();
            dq.display();
            break;
        case 6: System.out.print("\nDisplay List : ");
            dq.display();
            break;
        case 7: System.out.println("Exiting the program.");
            break;
        default:
            System.out.print("\nWrong choice! \n");
        }while(choice!=7); /*End of while*/
        sc.close();
        }}
```

**Output**

```
1.Insert At Front
2.Insert At Rear
3.Delete At Front
4.Delete At Rear
5.Delete all nodes
6.Display List
7.Exit
Enter your choice :1

Insert at Front - Enter data :5
The data are Double Ended Queue are:
NULL<--
5<->
Null-->
```

```
Enter your choice :2

Insert at Rear - Enter data: 10
The data are Double Ended Queue are:
NULL<--
5<->
10<->
Null-->
```

```
Enter your choice :4

Delete at Rear: The data are Double Ended Queue are:
NULL<--
5<->
Null-->
```

```
Enter your choice :5

Delete all nodes : The double ended queue is empty!
```

```
Enter your choice :6

Display List : The data are Double Ended Queue are:
NULL<--
5<->
10<->
Null-->
```

```
Enter your choice :3

Delete at Front: The data are Double Ended Queue are:
NULL<--
5<->
10<->
Null-->
```

**Conclusion:**

The document contains a comprehensive set of Java programs to demonstrate various operations using linked lists, such as singly linked lists, circular linked lists, doubly linked lists, stacks, queues, and more. Each implementation showcases key operations like insertion, deletion, traversal, and search in their respective contexts. The practical examples aim to provide a hands-on understanding of data structures through Java programming.

In conclusion, mastering these programs equips you with foundational knowledge of linked list operations, an essential component of computer science and software development, ensuring a strong grasp of both theory and practical application.