DataStructures and algorithms

Lab 3

Name:Pranjal bajaj

Cms:023-21-0022

Task 1

**addToFront: Adds a new node with the given data to the front of the linked list**.

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

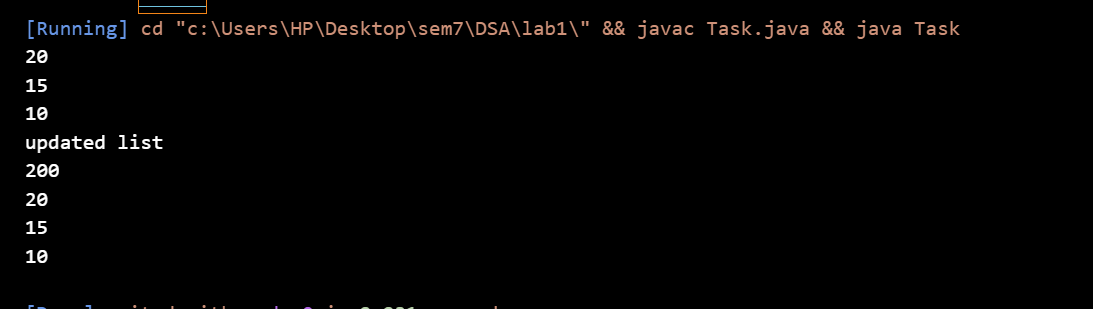
            ll.creation(200);

            System.out.println("updated list");

             ll.traverser();

    }

}



**getFrontItem: Returns the data stored in the first node (front) of the linked list.**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        int getFrontItem(){

            if(head == null){

                return 0;

            }

            else

            {return head.data;}

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

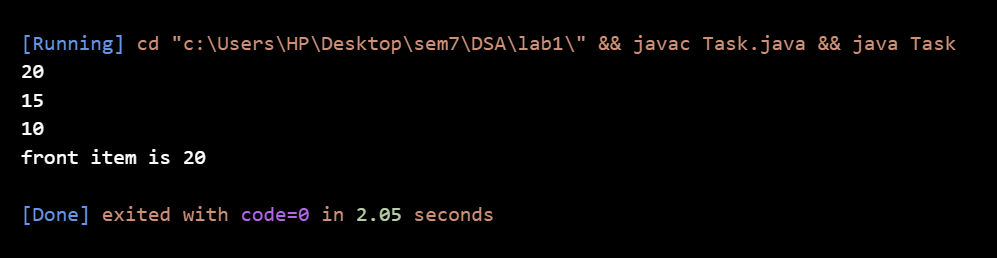
            ll.traverser();

            int item = ll.getFrontItem();

            System.out.println("front item is "+item);

    }

}



**removeFrontItem: Removes the first node (front) from the linked list.**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void removeFrontItem(){

                head = head.next;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

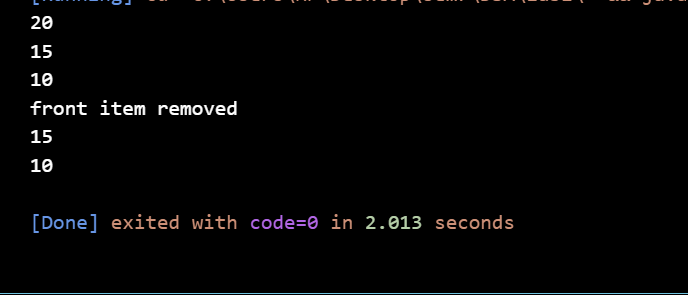
            System.out.println("front item removed ");

            ll.removeFrontItem();

            ll.traverser();

    }

}



**addToBack: Adds a new node with the given data to the end (back) of the linked list.**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void addToBack(int data){

            Node nn = new Node(data);

            tail.next = nn;

            nn.prev = tail;

            tail = nn;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

            System.out.println("back item added ");

            ll.addToBack(200);

            ll.traverser();

    }

}



**getBackItem: Returns the data stored in the last node (back) of the linked list**.

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        int getBackItem(){

            return tail.data;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

            int item = ll.getBackItem();

             System.out.println("back item is "+item);

    }

}



**removeBackItem: Removes the last node (back) from the linked list.**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void  removeBackItem(){

            tail = tail.prev;

            tail.next = null;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

            System.out.println("last item removed");

            ll.removeBackItem();

            ll.traverser();

    }

}



find: Checks if the given key is present in the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

                void find(int data){

            Node temp = head;

            while(temp.next != null){

                if(temp.data == data){

                    System.out.println(temp.data +" founded ");

                }

             temp = temp.next;

            }

                }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

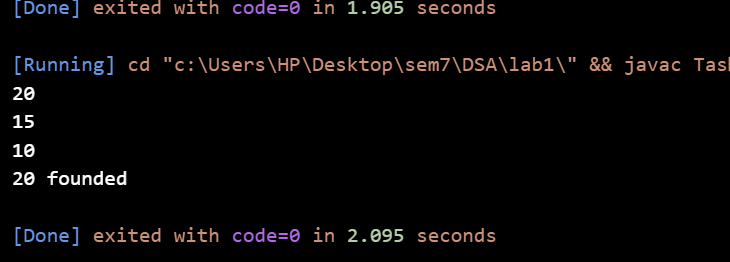
            ll.traverser();

            //System.out.println("last item removed");

            ll.find(20);

    }

}



**Remove: Removes the node with the given key from the linked list, if present.**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

                void Remove(int data){

            Node temp = head;

            while(temp.next != null){

                if(temp.next.data == data){

                    System.out.println(temp.next.data +" removed ");

                    temp.next = temp.next.next;

                    temp.next.prev = temp;

                }

             temp = temp.next;

            }

                }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

            //System.out.println("last item removed");

            ll.Remove(15);

            ll.traverser();

    }

}



**isListEmpty: Checks if the linked list is empty**.

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

                void Remove(int data){

            Node temp = head;

            while(temp.next != null){

                if(temp.next.data == data){

                    System.out.println(temp.next.data +" removed ");

                    temp.next = temp.next.next;

                    temp.next.prev = temp;

                }

             temp = temp.next;

            }

                }

                void isempty(){

                    if(head == null){

                        System.out.println("list is empty");

                    }

                    else

                    System.out.println("list is not empty");

                }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

            //System.out.println("last item removed");

            ll.isempty();

    }

}



**addKeyBeforeNode: Adds a new node with the given key before the node containing the specified data in the linked list.**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void keybeforenode(int data , int add){

            Node temp = head;

            Node newNode = new Node(add);

            if (head.data == data) {

                newNode.next = head;

                head = newNode;

                return;

            }

            Node current = head;

            while (current.next != null && current.next.data != data) {

                current = current.next;

            }

            if (current.next == null) {

                System.out.println("Target key not found.");

                return;

            }

                newNode.next = current.next;

                current.next.prev = newNode;

                current.next = newNode;

                newNode.prev = current;

            }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

            ll.keybeforenode(15 , 100);

            System.out.println(" item added ");

            ll.traverser();

    }

}



addKeyAfterNode: Adds a new node with the given key after the node containing the specified data in the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void keyafternode(int data , int add){

            Node temp = head;

            Node newNode = new Node(add);

            if (head.data == data) {

                newNode.next = head;

                head = newNode;

                return;

            }

            Node current = head;

            while (current.next != null && current.data != data) {

                current = current.next;

            }

            if (current.next == null) {

                System.out.println("Target key not found.");

                return;

            }

                newNode.next = current.next;

                current.next.prev = newNode;

                current.next = newNode;

                newNode.prev = current;

            }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.traverser();

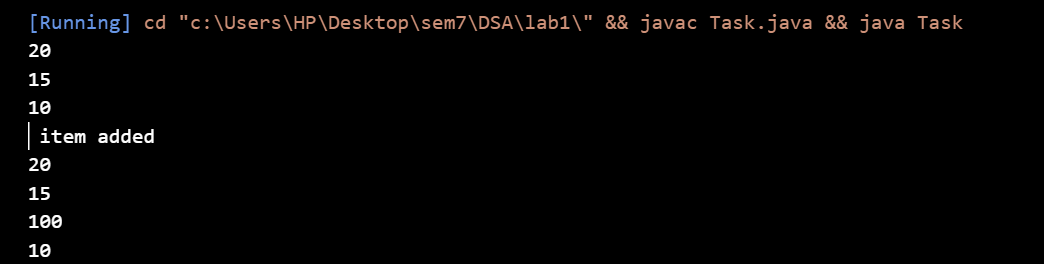
            ll.keyafternode(15 , 100);

            System.out.println(" item added ");

            ll.traverser();

    }

}



**printAll: Prints all the values in the linked list**

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(34);

            ll.creation(36);

            ll.creation(278);

            ll.creation(76867);

            ll.traverser();

    }

}



Task 2

class Task{

    static class Node{

    int data;

    Node next;

    Node prev;

    Node(int data){

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

                tail = newNode;

            }

            else

            {

            head.prev = newNode;

            newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void addToBack(int data){

            Node nn = new Node(data);

            tail.next = nn;

            nn.prev = tail;

            tail = nn;

        }

        void  removeBackItem(){

            tail = tail.prev;

            tail.next = null;

        }

        void printreverse(){

            Node current = tail;

            while( current.prev!=null){

            System.out.println(current.data);

            current = current.prev;

        }}

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(34);

            ll.creation(36);

            ll.creation(278);

            ll.creation(76867);

            System.out.println("item added at last");

            ll.addToBack(50);

            ll.traverser();

            System.out.println("item reemoved at last");

            ll.removeBackItem();

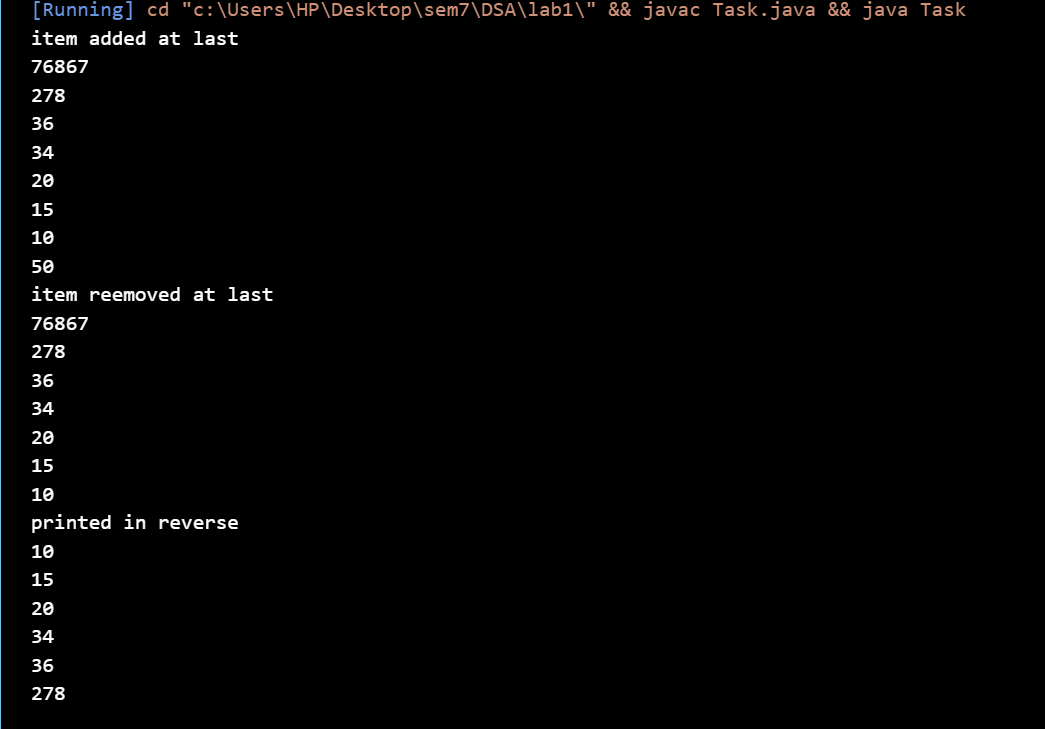
            ll.traverser();

            System.out.println("printed in reverse");

            ll.printreverse();

    }

}



Task 3

class Task {

    class Node {

    int data;

    Node next;

    public Node(int data) {

        this.data = data;

        this.next = null;

    }

}

     Node last;

    public Task() {

        last = null;

    }

    public void insertAtBeginning(int data) {

        Node newNode = new Node(data);

        if (last == null) {

            last = newNode;

            last.next = newNode;

        } else {

            newNode.next = last.next;

            last.next = newNode;

        }

    }

    public void insertAtEnd(int data) {

        Node newNode = new Node(data);

        if (last == null) {

            last = newNode;

            last.next = newNode;

        } else {

            newNode.next = last.next;

            last.next = newNode;

            last = newNode;

        }

    }

    public void deleteFromBeginning() {

        if (last == null) {

            System.out.println("The list is empty!");

        } else if (last.next == last) {

            last = null;

        } else {

            last.next = last.next.next;

        }

    }

    public void deleteFromEnd() {

        if (last == null) {

            System.out.println("The list is empty!");

        } else if (last.next == last) {

            last = null;

        } else {

            Node temp = last.next;

            while (temp.next != last) {

                temp = temp.next;

            }

            temp.next = last.next;

            last = temp;

        }

    }

    public void display() {

        if (last == null) {

            System.out.println("The list is empty!");

            return;

        }

        Node temp = last.next;

        do {

            System.out.print(temp.data + " -> ");

            temp = temp.next;

        } while (temp != last.next);

        System.out.println();

    }

    public static void main(String[] args) {

        Task cll = new Task();

        // Insert elements

        cll.insertAtEnd(10);

        cll.insertAtEnd(20);

        cll.insertAtEnd(30);

        cll.insertAtBeginning(5);

        System.out.println("List after insertions:");

        cll.display();

        // Delete from the beginning

        cll.deleteFromBeginning();

        System.out.println("List after deleting from beginning:");

        cll.display();

        // Delete from the end

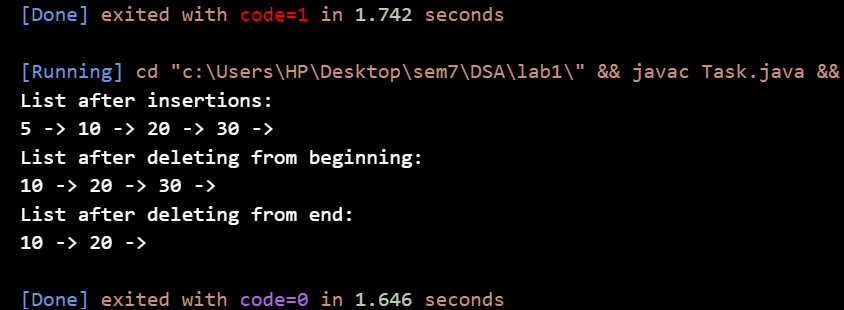
        cll.deleteFromEnd();

        System.out.println("List after deleting from end:");

        cll.display();

    }

}



Task 4

class Node {

    int data;

    Node next;

    public Node(int data) {

        this.data = data;

        this.next = null;

    }

}

class Task {

    private Node last;

    public Task() {

        last = null;

    }

    // Insert at the beginning

    public void insertAtBeginning(int data) {

        Node newNode = new Node(data);

        if (last == null) {

            last = newNode;

            last.next = newNode; // Point to itself

        } else {

            newNode.next = last.next;

            last.next = newNode;

        }

    }

    public void display() {

        if (last == null) {

            System.out.println("The list is empty!");

            return;

        }

        Node temp = last.next;

        do {

            System.out.print(temp.data + " -> ");

            temp = temp.next;

        } while (temp != last.next);

        System.out.println();

    }

    // Check for cycle in the list

    public boolean hasCycle() {

        if (last == null) return false;

        Node slow = last.next; // Tortoise

        Node fast = last.next; // Hare

        while (fast != null && fast.next != null) {

            slow = slow.next;           // Move slow by 1 step

            fast = fast.next.next;     // Move fast by 2 steps

            if (slow == fast) {         // Cycle detected

                return true;

            }

        }

        return false; // No cycle found

    }

    // Main method to test the operations

    public static void main(String[] args) {

        Task cll = new Task();

        // Insert elements

        cll.insertAtBeginning(5);

        cll.insertAtBeginning(10);

        cll.insertAtBeginning(20);

        cll.insertAtBeginning(50);

        System.out.println("List after insertions:");

        cll.display();

        // Check for cycle

        System.out.println("Does the list have a cycle? " + cll.hasCycle());

        // Create a cycle for testing

        Node cycleNode = new Node(40);

        cll.last.next = cycleNode; // Create a cycle: last node points to 40

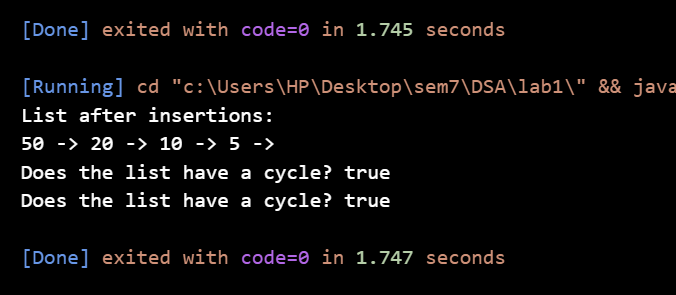
        cycleNode.next = cll.last.next; // Cycle points back to node with value 10

        // Check for cycle again

        System.out.println("Does the list have a cycle? " + cll.hasCycle());

    }

}



Tasks 5

class DoublyNode {

    int data;

    DoublyNode next;

    DoublyNode prev;

    public DoublyNode(int data) {

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

class Task {

    private DoublyNode head;

    public void insert(int data) {

        DoublyNode newNode = new DoublyNode(data);

        if (head == null) {

            head = newNode;

        } else {

            DoublyNode temp = head;

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = newNode;

            newNode.prev = temp; // Set previous pointer

        }

    }

    // Method to find the middle element

    public Integer findMiddle() {

        if (head == null) return null; // List is empty

        DoublyNode slow = head; // Slow pointer

        DoublyNode fast = head; // Fast pointer

        while (fast != null && fast.next != null) {

            slow = slow.next;           // Move slow by 1

            fast = fast.next.next;     // Move fast by 2

        }

        return slow.data; // Slow pointer is at the middle

    }

    // Method to display the list

    public void display() {

        DoublyNode temp = head;

        while (temp != null) {

            System.out.print(temp.data + " <-> ");

            temp = temp.next;

        }

        System.out.println("NULL");

    }

    // Main method for testing

    public static void main(String[] args) {

        Task dll = new Task();

        dll.insert(1);

        dll.insert(2);

        dll.insert(3);

        dll.insert(4);

        dll.insert(5);

        dll.display();

        System.out.println("Middle Element: " + dll.findMiddle());

    }

}

