DSA Lab 3

Muhammad Qasim

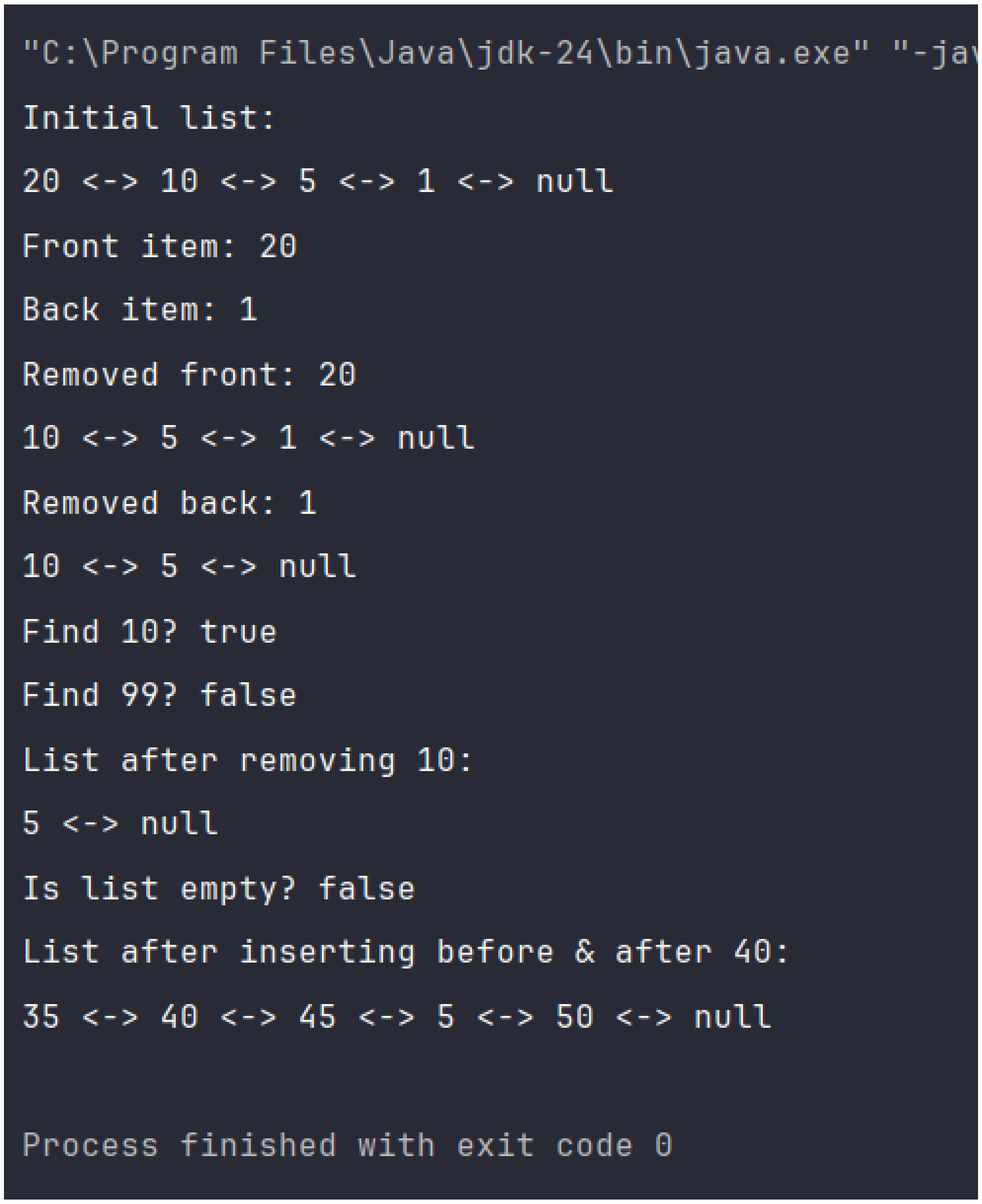
Muhammad Qasim
[COMPANY NAME] [Company address]

TASK 1:

```
public class DoublyList {
  Node Head;

  public DoublyList() {
    Head = null;
  }

  class Node {
    int data;
    Node prev;
    Node next;

    public Node(int d) {
      this.data = d;
      this.prev = null;
      this.next = null;
    }
  }

  void addToFront(int data) {
    Node newNode = new Node(data);
    if (Head == null) {
      Head = newNode;
    } else {
        newNode.next = Head;

        Head.prev = newNode;

        Head = newNode;

    }

}


void addToBack(int data) {

    Node newNode = new Node(data);

    if (Head == null) {

        Head = newNode;

        return;

    }

    Node temp = Head;

    while (temp.next != null) {

        temp = temp.next;

    }

    temp.next = newNode;

    newNode.prev = temp;

}



int getFrontItem() {

    if (Head == null) {

        throw new IllegalStateException("List is empty");

    }

    return Head.data;
```

```
}

int getBackItem() {
    if (Head == null) {
        throw new IllegalStateException("List is empty");
    }
    Node temp = Head;
    while (temp.next != null) {
        temp = temp.next;
    }
    return temp.data;
}

int removeFrontItem() {
    if (Head == null) {
        throw new IllegalStateException("List is empty");
    }
    int removedData = Head.data;
    Head = Head.next;
    if (Head != null) {
        Head.prev = null;
    }
    return removedData;
}

int removeBackItem() {
    if (Head == null) {
        throw new IllegalStateException("List is empty");
    }
    if (Head.next == null) {
        int val = Head.data;
        Head = null;
        return val;
    }
    Node temp = Head;
    while (temp.next != null) {
        temp = temp.next;
    }
    int val = temp.data;
    temp.prev.next = null;
    return val;
}
boolean find(int key) {
    Node temp = Head;
    while (temp != null) {
        if (temp.data == key) return true;
        temp = temp.next;
    }
    return false;
}

void remove(int key) {
    if (Head == null) return;
    if (Head.data == key) {
        Head = Head.next;
        if (Head != null) Head.prev = null;
        return;
    }
    Node temp = Head;
    while (temp != null && temp.data != key) {
        temp = temp.next;
    }
    if (temp != null) {
        if (temp.next != null) {
            temp.next.prev = temp.prev;
        }
        if (temp.prev != null) {
            temp.prev.next = temp.next;
        }
    }
}

boolean isListEmpty() {
    return Head == null;
}

void addKeyBeforeNode(int key, int target) {
    if (Head == null) return;
    Node temp = Head;
    while (temp != null && temp.data != target) {
        temp = temp.next;
    }
    if (temp != null) {
        Node newNode = new Node(key);
        newNode.next = temp;
        newNode.prev = temp.prev;
        if (temp.prev != null) {
            temp.prev.next = newNode;
        } else {
            Head = newNode;
        }
        temp.prev = newNode;
    }
}

void addKeyAfterNode(int key, int target) {
    Node temp = Head;
    while (temp != null && temp.data != target) {
        temp = temp.next;
    }
    if (temp != null) {
        Node newNode = new Node(key);
        newNode.next = temp.next;
        newNode.prev = temp;
        if (temp.next != null) {
            temp.next.prev = newNode;
        }
        temp.next = newNode;
    }
}

void printList() {
    Node temp = Head;
    while (temp != null) {
        System.out.print(temp.data + " <-> ");
        temp = temp.next;
    }
    System.out.println("null");
}

void printAll() {
    printList();
}

public static void main(String[] args) {
    DoublyList list = new DoublyList();

    list.addToFront(10);
    list.addToFront(20);
    list.addToBack(5);
    list.addToBack(1);


    System.out.println("Initial list:");

    list.printAll();


    System.out.println("Front item: " + list.getFrontItem());

    System.out.println("Back item: " + list.getBackItem());


    System.out.println("Removed front: " + list.removeFrontItem());

    list.printAll();


    System.out.println("Removed back: " + list.removeBackItem());

    list.printAll();


    System.out.println("Find 10? " + list.find(10));

    System.out.println("Find 99? " + list.find(99));


    list.remove(10);

    System.out.println("List after removing 10:");

    list.printAll();


    System.out.println("Is list empty? " + list.isListEmpty());


    list.addToFront(40);

    list.addToBack(50);

    list.addKeyBeforeNode(35, 40);
```

```
        list.addKeyAfterNode(45, 40);


        System.out.println("List after inserting before & after 40:");
        list.printAll();
    }
}
```



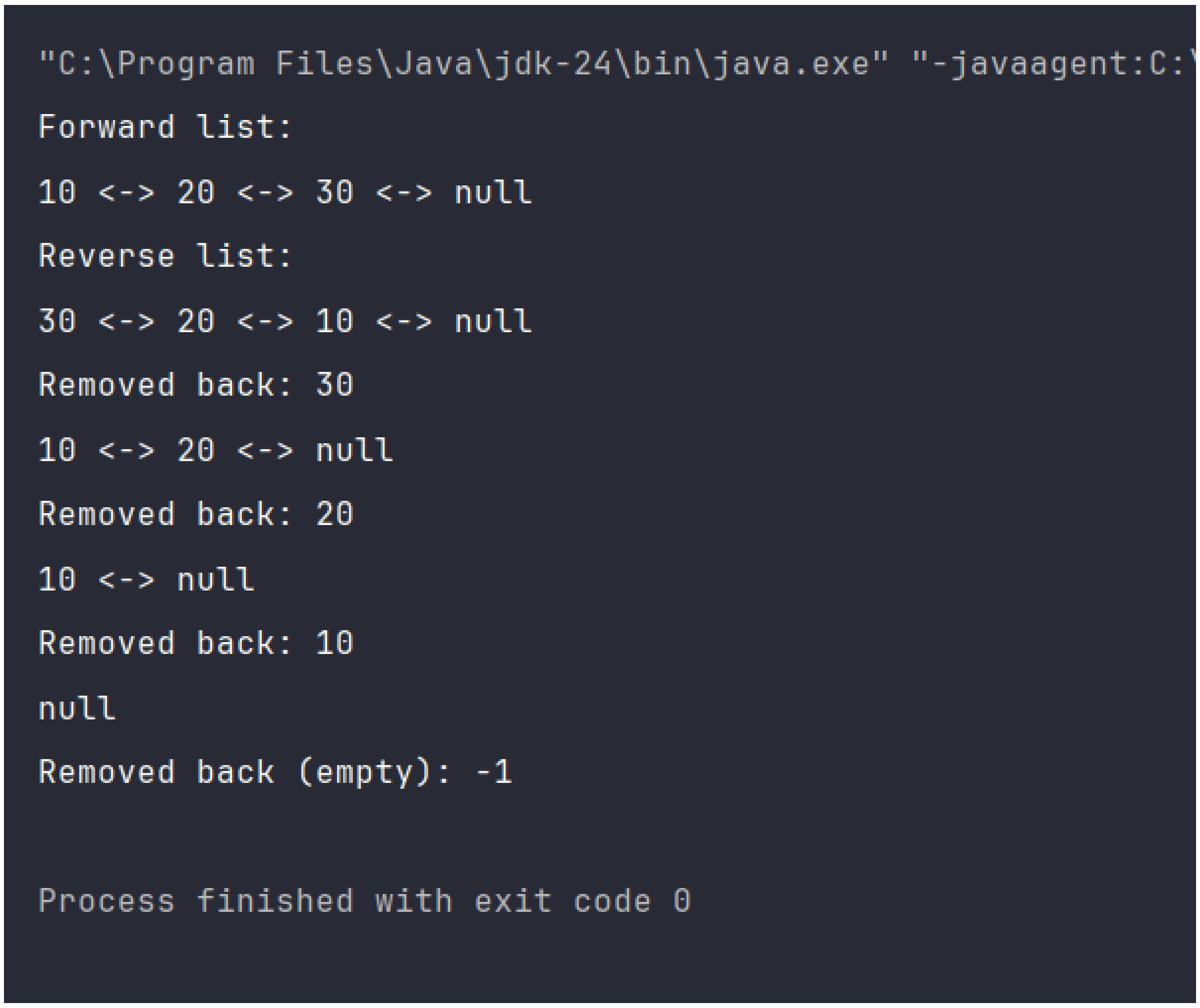
```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-ja
Initial list:
20 <-> 10 <-> 5 <-> 1 <-> null
Front item: 20
Back item: 1
Removed front: 20
10 <-> 5 <-> 1 <-> null
Removed back: 1
10 <-> 5 <-> null
Find 10? true
Find 99? false
List after removing 10:
5 <-> null
Is list empty? false
List after inserting before & after 40:
35 <-> 40 <-> 45 <-> 5 <-> 50 <-> null


Process finished with exit code 0
```

TASK 2:

```
public class DoublyList2 {

  Node head;

  Node tail;


  class Node {

    int data;

    Node prev;

    Node next;


    Node(int d) {

      this.data = d;

      this.prev = null;

      this.next = null;

    }

  }


  void addToBack(int data) {

    Node newNode = new Node(data);

    if (tail == null) {

      head = tail = newNode;

    } else {

      tail.next = newNode;

      newNode.prev = tail;

      tail = newNode;
    }
  }

  int removeBackItem() {
    if (tail == null) {
      return -1; // list is empty
    }
    int val = tail.data;
    tail = tail.prev;
    if (tail != null) {
      tail.next = null;
    } else {
      head = null;
    }
    return val;
  }

  void printInReverseOrder() {
    Node temp = tail;
    while (temp != null) {
      System.out.print(temp.data + " <-> ");
      temp = temp.prev;
    }
    System.out.println("null");
  }
```

```
void printList() {
   Node temp = head;
   while (temp != null) {
      System.out.print(temp.data + " <-> ");
      temp = temp.next;
   }
   System.out.println("null");
}

public static void main(String[] args) {
   DoublyList2 list = new DoublyList2();

   list.addToBack(10);
   list.addToBack(20);
   list.addToBack(30);

   System.out.println("Forward list:");
   list.printList();

   System.out.println("Reverse list:");
   list.printInReverseOrder();

   System.out.println("Removed back: " + list.removeBackItem());
   list.printList();

   System.out.println("Removed back: " + list.removeBackItem());
```

```
        list.printList();

        System.out.println("Removed back: " + list.removeBackItem());
        list.printList();

        System.out.println("Removed back (empty): " + list.removeBackItem());
    }
}
```

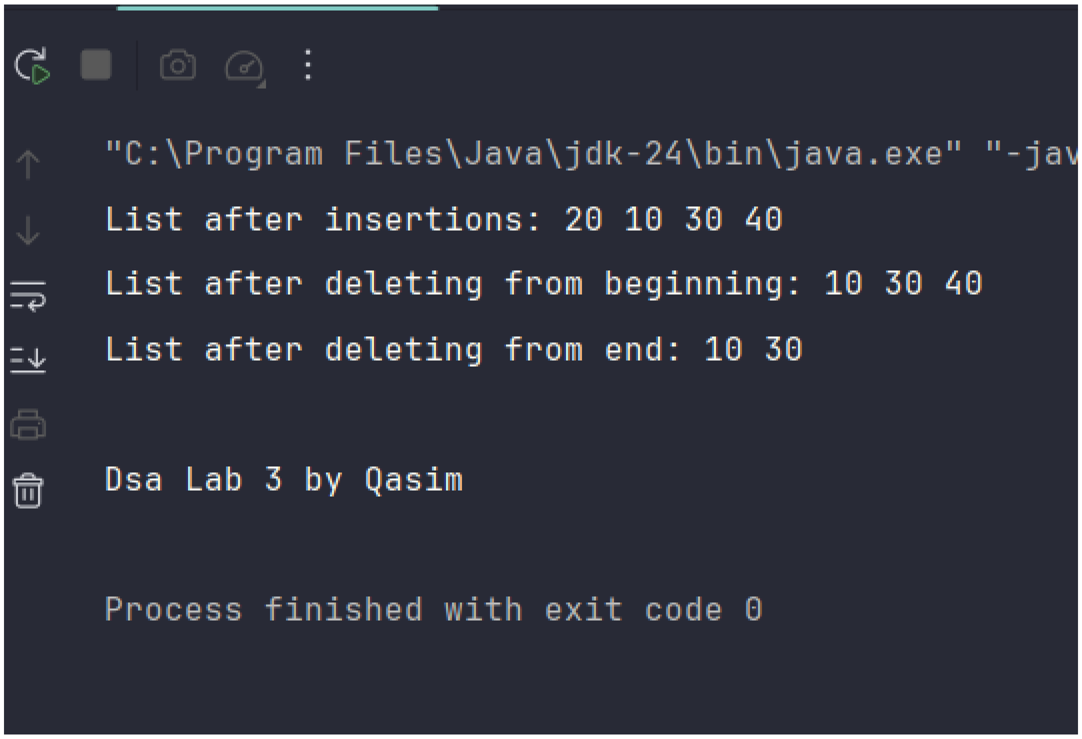


```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\
Forward list:
10 <-> 20 <-> 30 <-> null
Reverse list:
30 <-> 20 <-> 10 <-> null
Removed back: 30
10 <-> 20 <-> null
Removed back: 20
10 <-> null
Removed back: 10
null
Removed back (empty): -1

Process finished with exit code 0
```

TASK 3:

```
public class DoublyList3 {
  Node head, tail;

  class Node {
    int data;
    Node next;
    Node(int data) {
      this.data = data;
    }
  }

  void insertAtBeginning(int data) {
    Node NN = new Node(data);
    if (head == null) {
      head = tail = NN;
      tail.next = head;
    } else {
      NN.next = head;
      head = NN;
      tail.next = head;
    }
  }

  void insertAtEnd(int data) {
    Node NN = new Node(data);
    if (head == null) {
        head = tail = NN;
        tail.next = head;
    } else {
        tail.next = NN;
        tail = NN;
        tail.next = head;
    }
}

void deleteFromBeginning() {
    if (head == null) return;
    if (head == tail) {
        head = tail = null;
    } else {
        head = head.next;
        tail.next = head;
    }
}

void deleteFromEnd() {
    if (head == null) return;
    if (head == tail) {
        head = tail = null;
    } else {
        Node temp = head;
        while (temp.next != tail) {
            temp = temp.next;
        }
        temp.next = head;
        tail = temp;
    }
}

void display() {
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    Node temp = head;
    do {
        System.out.print(temp.data + " ");
        temp = temp.next;
    } while (temp != head);
    System.out.println();
}

public static void main(String[] args) {
    DoublyList3 list = new DoublyList3();

    list.insertAtBeginning(10);
```

```
        list.insertAtBeginning(20);

        list.insertAtEnd(30);

        list.insertAtEnd(40);

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

        list.display();


        list.deleteFromBeginning();

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

        list.display();


        list.deleteFromEnd();

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

        list.display();

        System.out.println("\nDsa Lab 3 by Qasim");

    }

}
```

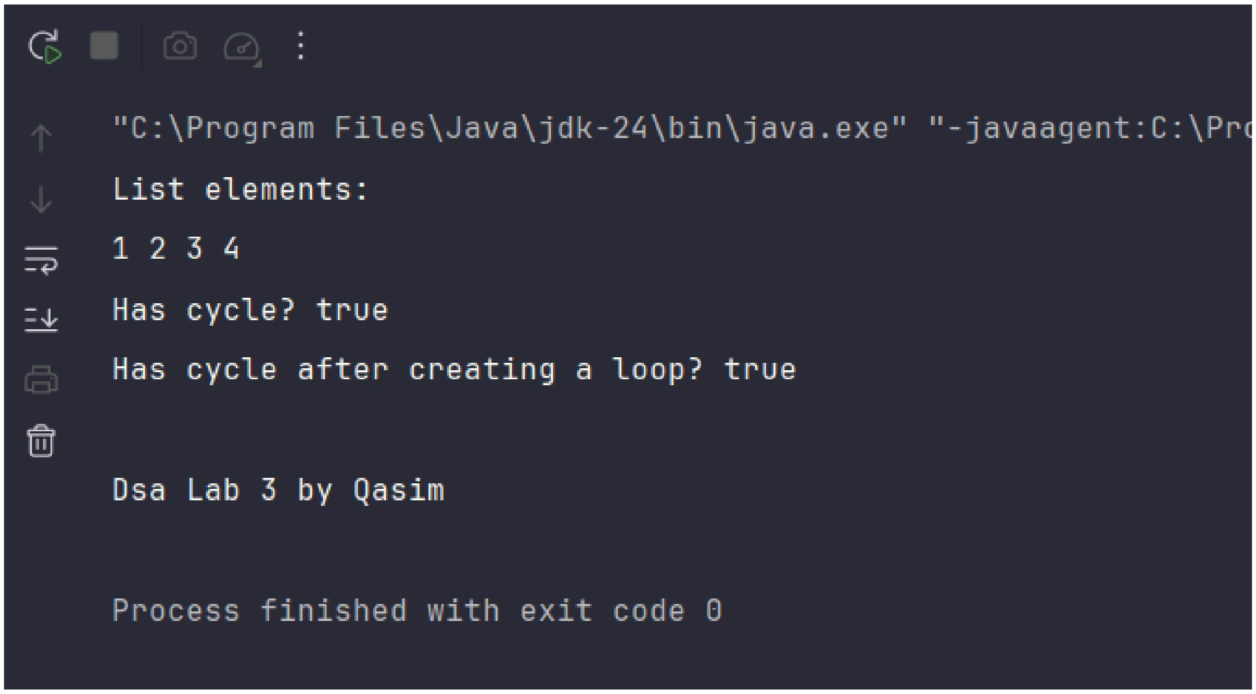

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-jav
List after insertions: 20 10 30 40
List after deleting from beginning: 10 30 40
List after deleting from end: 10 30

Dsa Lab 3 by Qasim

Process finished with exit code 0
```

TASK 4:

```
public class DoublyList4 {
  Node head, tail;


  class Node {
    int data;
    Node next;
    Node(int data) {
      this.data = data;
    }
}

void insertAtBeginning(int data) {
   Node NN = new Node(data);
   if (head == null) {
      head = tail = NN;
      tail.next = head;
   } else {
      NN.next = head;
      head = NN;
      tail.next = head;
   }
}

void insertAtEnd(int data) {
   Node NN = new Node(data);
   if (head == null) {
      head = tail = NN;
      tail.next = head;
   } else {
      tail.next = NN;
      tail = NN;
      tail.next = head;
   }
}
```

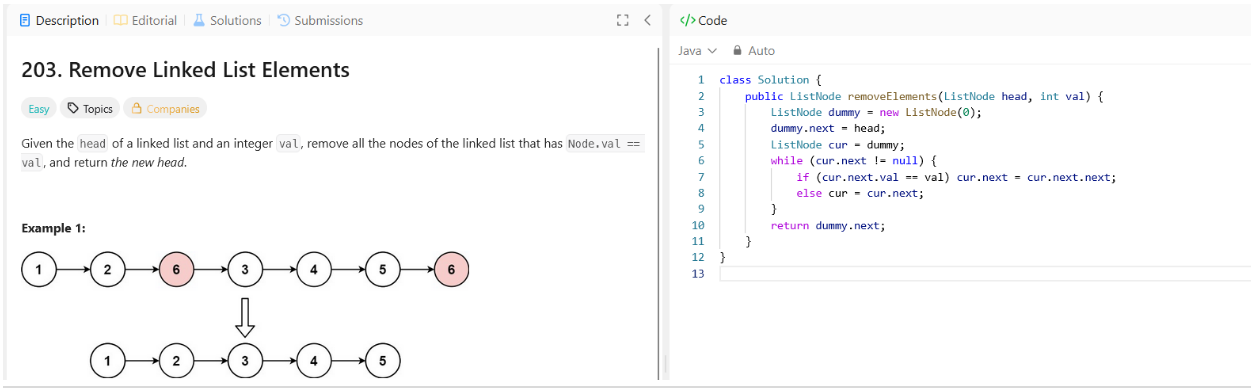
```
void deleteFromBeginning() {
    if (head == null) return;
    if (head == tail) {
        head = tail = null;
    } else {
        head = head.next;
        tail.next = head;
    }
}

void deleteFromEnd() {
    if (head == null) return;
    if (head == tail) {
        head = tail = null;
    } else {
        Node temp = head;
        while (temp.next != tail) {
            temp = temp.next;
        }
        temp.next = head;
        tail = temp;
    }
}

void display() {
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    Node temp = head;
    do {
        System.out.print(temp.data + " ");
        temp = temp.next;
    } while (temp != head);
    System.out.println();
}

boolean hasCycle() {
    if (head == null) return false;
    Node slow = head, fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow == fast) return true;
    }
    return false;
}

public static void main(String[] args) {
    DoublyList4 list = new DoublyList4();

    list.insertAtEnd(1);
```

```
        list.insertAtEnd(2);
        list.insertAtEnd(3);
        list.insertAtEnd(4);

        System.out.println("List elements: ");
        list.display();

        System.out.println("Has cycle? " + list.hasCycle());

        list.tail.next = list.head.next;

        System.out.println("Has cycle after creating a loop? " + list.hasCycle());



        System.out.println("\nDsa Lab 3 by Qasim");
    }
}
```



```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Pro
List elements:
1 2 3 4
Has cycle? true
Has cycle after creating a loop? true

Dsa Lab 3 by Qasim

Process finished with exit code 0
```

LEETCODE QUESTION 1:

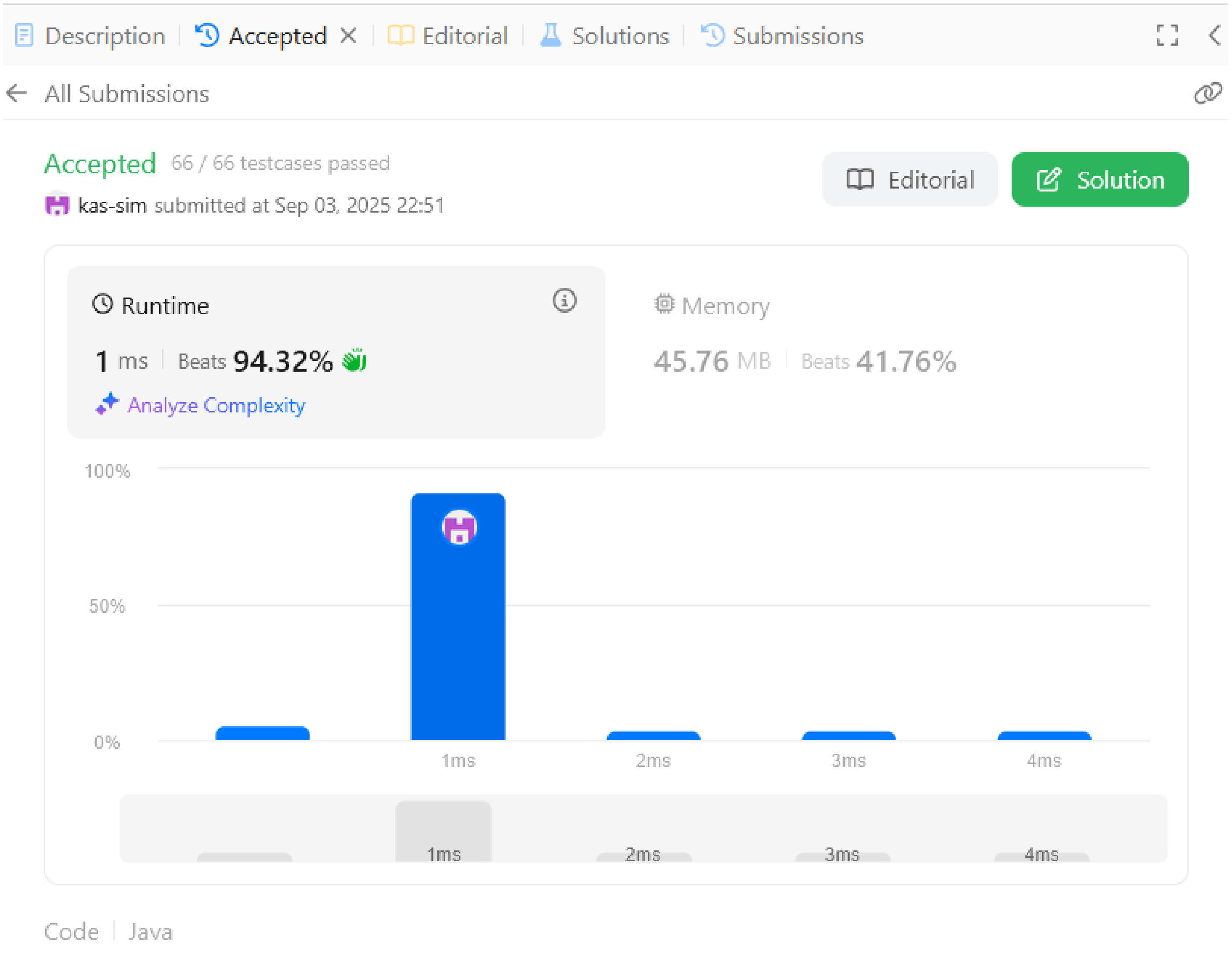


203. Remove Linked List Elements

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head*.

Example 1:

```java
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode cur = dummy;
        while (cur.next != null) {
            if (cur.next.val == val) cur.next = cur.next.next;
            else cur = cur.next;
        }
        return dummy.next;
    }
}
```



Accepted 66 / 66 testcases passed

kas-sim submitted at Sep 03, 2025 22:51

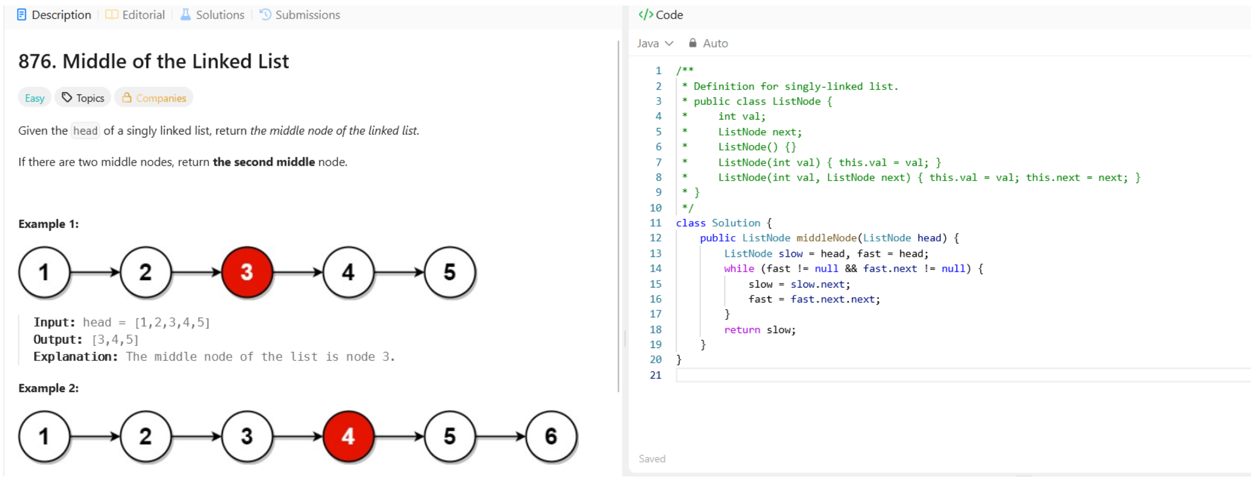
Runtime

1 ms | Beats 94.32%

Memory

45.76 MB | Beats 41.76%

Leetcode question 2:



876. Middle of the Linked List

Easy Topics Companies

Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

Example 1:

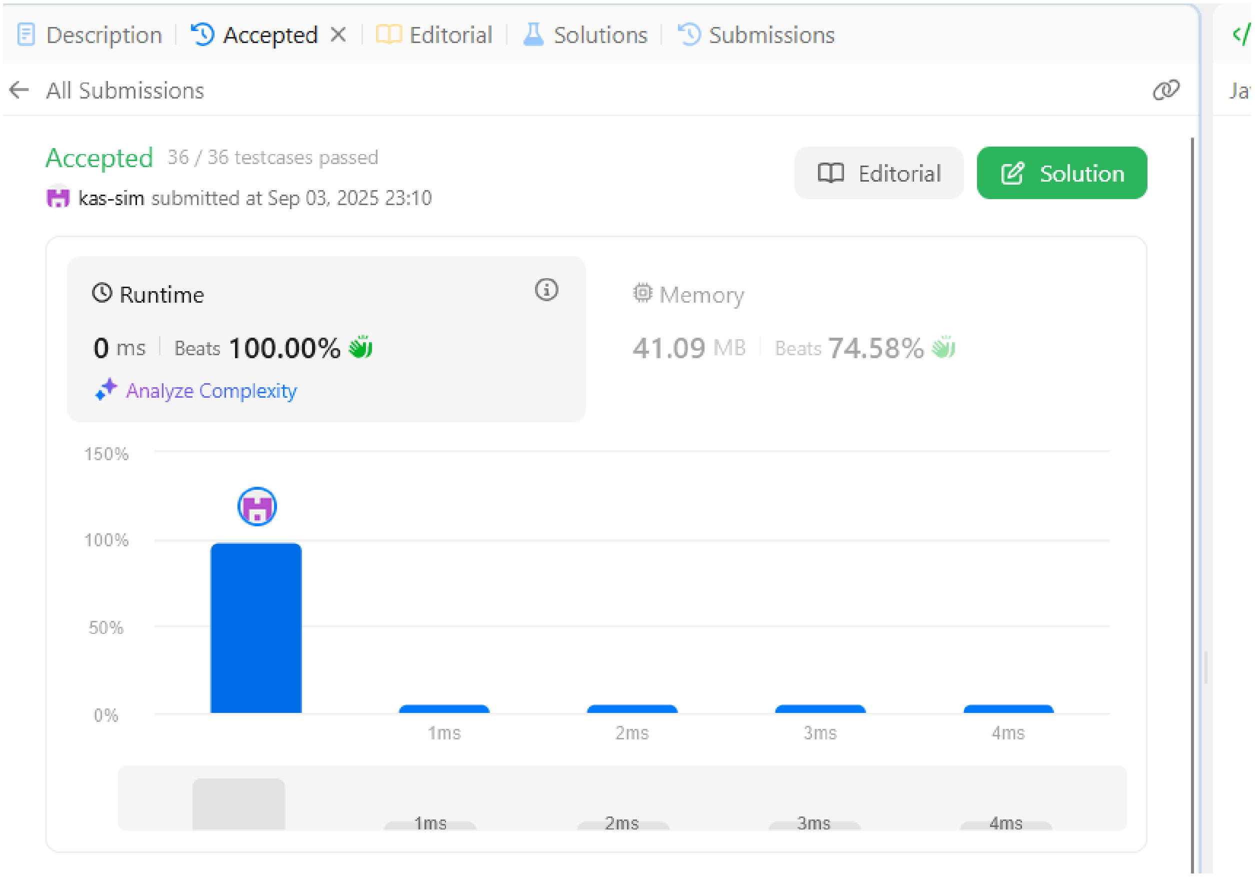
$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \end{tikzcd}$$

```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:

$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \end{tikzcd}$$

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```



Accepted 36 / 36 testcases passed

kas-sim submitted at Sep 03, 2025 23:10

Runtime

0 ms | Beats 100.00%

Memory

41.09 MB | Beats 74.58%