Q.1 :

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

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

class LinkedList {
   Node head;

   public LinkedList() {
      this.head = null;
   }

   // Function to find the middle of the linked list
   public void findMiddle() {
      if (head == null) {
         System.out.println("The list is empty.");
         return;
      }

      Node slowPtr = head;
      Node fastPtr = head;
```

```
        // Traverse the list with two pointers, one moving one step at a time and the other two steps at a
time
        while (fastPtr != null && fastPtr.next != null) {
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next.next;
        }

        System.out.println("The middle element is: " + slowPtr.data);
    }

    // Function to insert a new node at the end of the linked list
    public void insert(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            return;
        }

        Node last = head;
        while (last.next != null) {
            last = last.next;
        }

        last.next = newNode;
    }

    // Function to display the linked list
    public void display() {
        Node current = head;

        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }

        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();

        // Inserting elements into the linked list
        list.insert(1);
        list.insert(2);
        list.insert(3);
        list.insert(4);
        list.insert(5);

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

        // Finding the middle element
        list.findMiddle();
    }
}
```

Q.2 : class Node {

```
    int data;
    Node next;

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

class LinkedList {
    Node head;

    public LinkedList() {
        this.head = null;
    }

    // Function to insert a new node at the end of the linked list
    public void insert(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            return;
        }

        Node last = head;
        while (last.next != null) {
```

```
        last = last.next;
    }

    last.next = newNode;
}

// Function to create a loop in the linked list (for testing purposes)
public void createLoop(int position) {
    if (position <= 0) {
        return;
    }

    Node current = head;
    Node loopNode = null;

    int count = 1;
    while (current.next != null) {
        if (count == position) {
            loopNode = current;
        }

        current = current.next;
        count++;
    }

    current.next = loopNode; // Creating the loop
}

// Function to detect a loop in the linked list
```

```
    public boolean hasLoop() {
        Node slowPtr = head;
        Node fastPtr = head;

        while (fastPtr != null && fastPtr.next != null) {
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next.next;

            if (slowPtr == fastPtr) {
                // Loop detected
                return true;
            }
        }

        // No loop detected
        return false;
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();

        // Inserting elements into the linked list
        list.insert(1);
        list.insert(2);
        list.insert(3);
        list.insert(4);
        list.insert(5);
        // Creating a loop for testing (connecting the last node to the second node)
        list.createLoop(2);


        // Checking if the linked list has a loop
        boolean hasLoop = list.hasLoop();
        System.out.println("Linked List has a loop: " + hasLoop);
    }
}
```