LAB # 08

Singly Linked List Implementation And Doubly Linked List implementation of the list ADT

OBJECTIVE: Implementing singly linked list, associated operations and Runner technique and Implementing doubly linked list, associated operations and LRU technique.

Lab Task

1. Write a program that can store 10 records of students in a link list manner and apply the following operations on it.

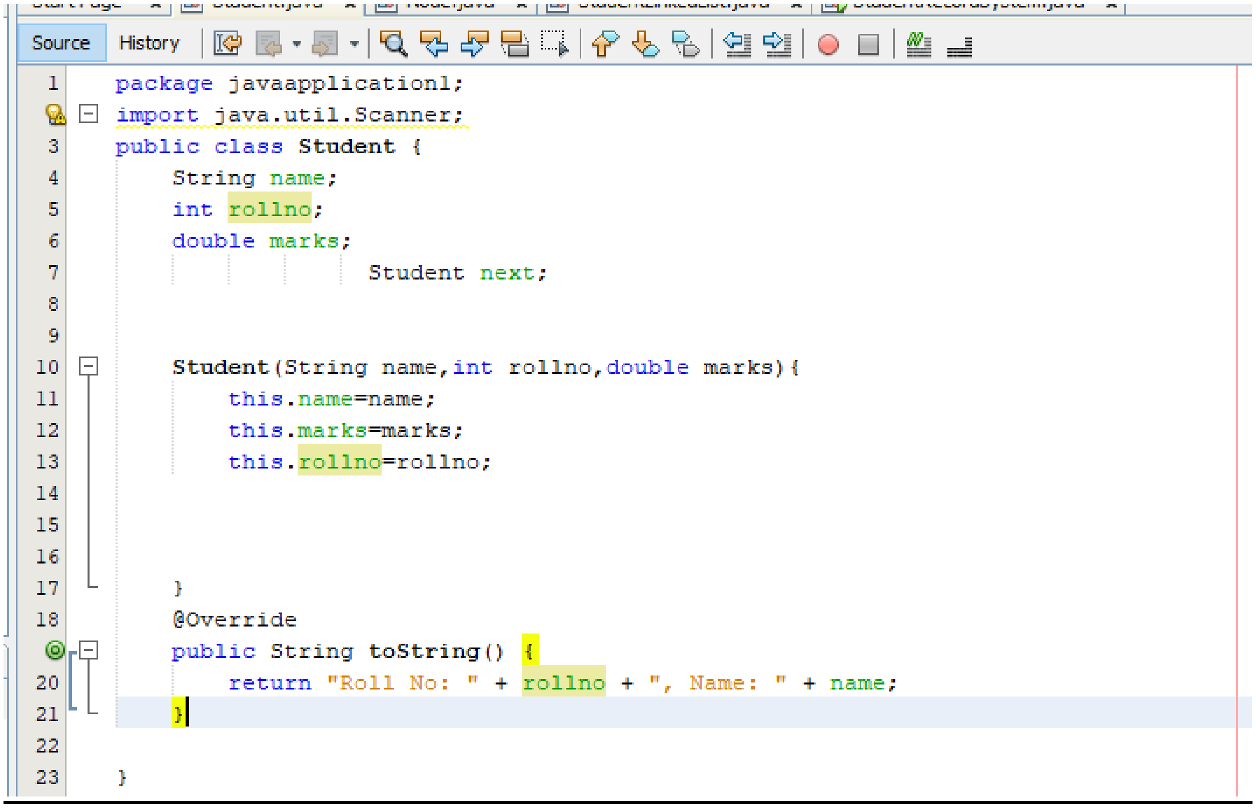
   a. View the list

   b. Insert the elements in different locations of linked list and view it.

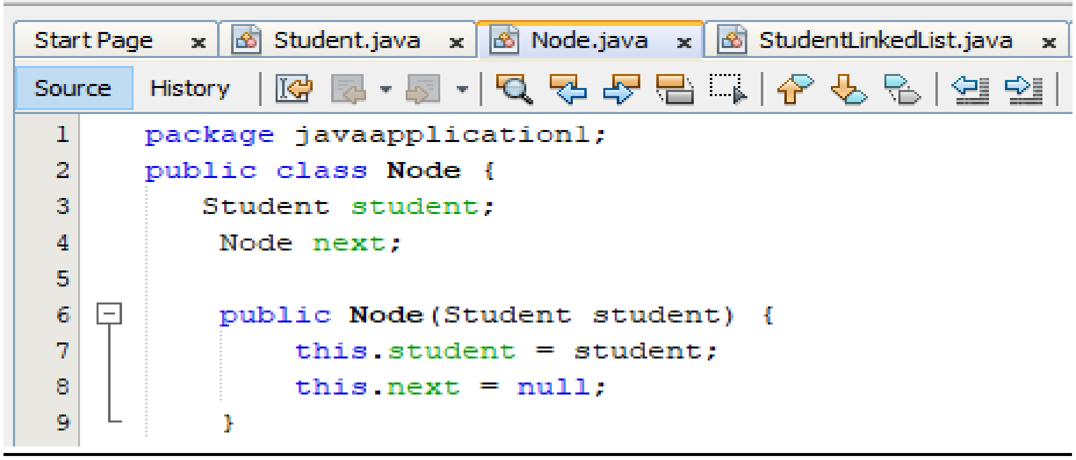
   c. Search any element from the linked list

   d. Delete record again view the list after deletion.

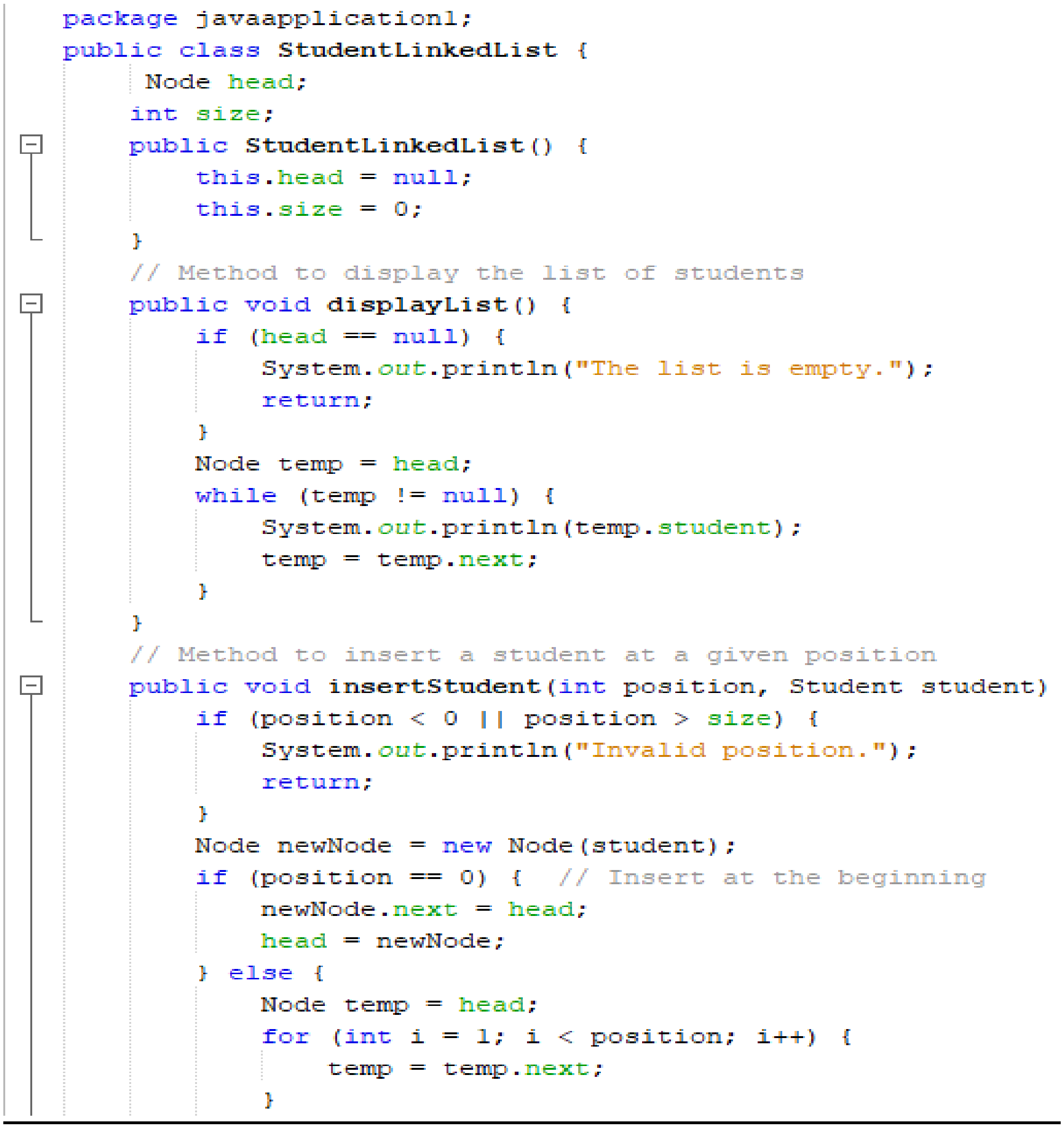
   CODE:



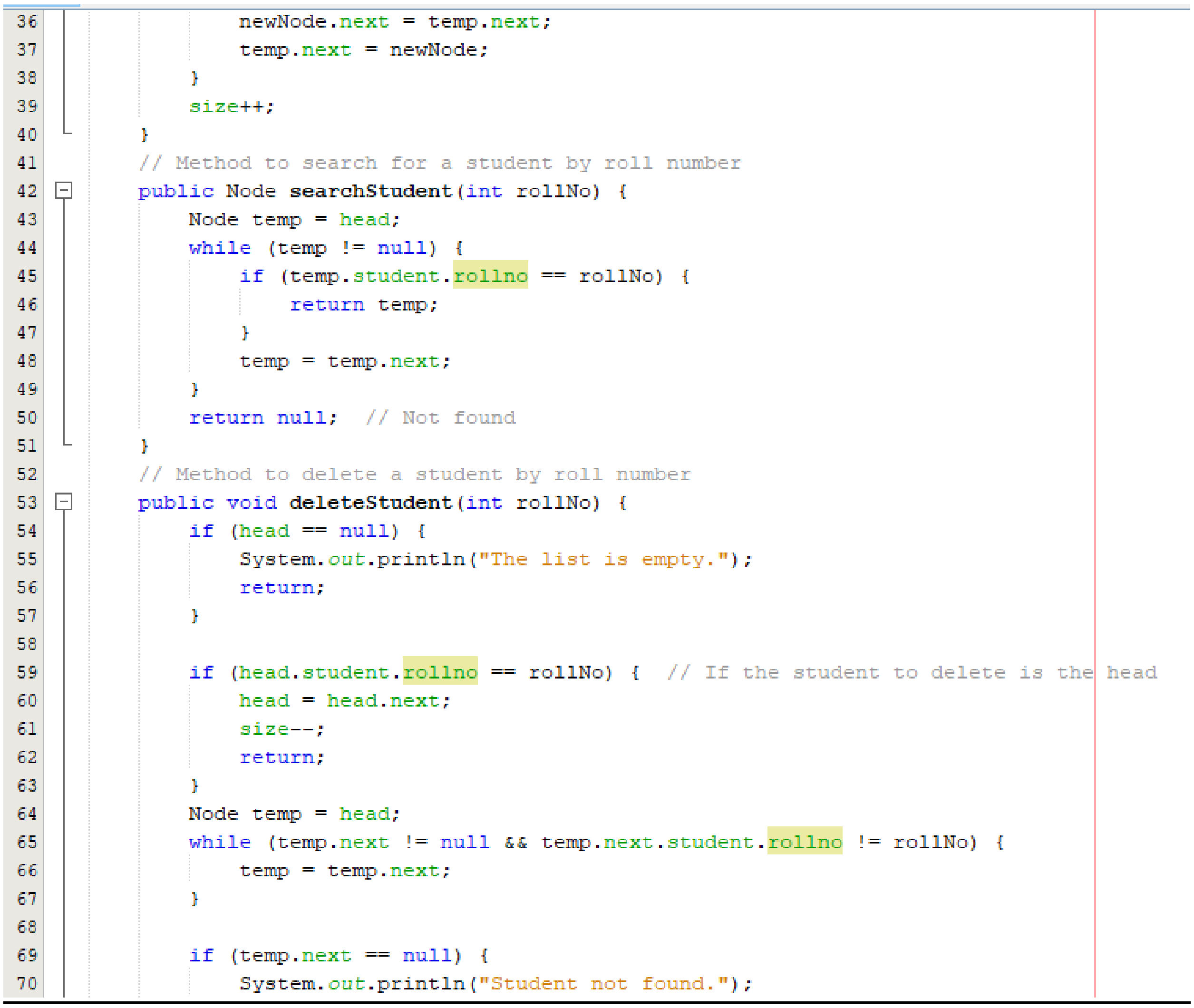
```java
package javaapplication1;
import java.util.Scanner;
public class Student {
    String name;
    int rollno;
    double marks;
                  Student next;


    Student(String name,int rollno,double marks){
        this.name=name;
        this.marks=marks;
        this.rollno=rollno;



    }
    @Override
    public String toString() {
        return "Roll No: " + rollno + ", Name: " + name;
    }

}
```



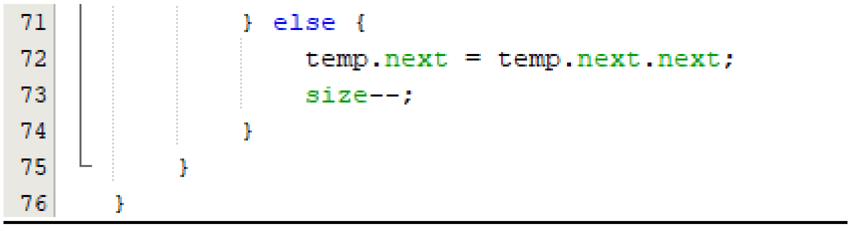
```java
package javaapplication1;
public class Node {
   Student student;
    Node next;

    public Node(Student student) {
        this.student = student;
        this.next = null;
    }
```



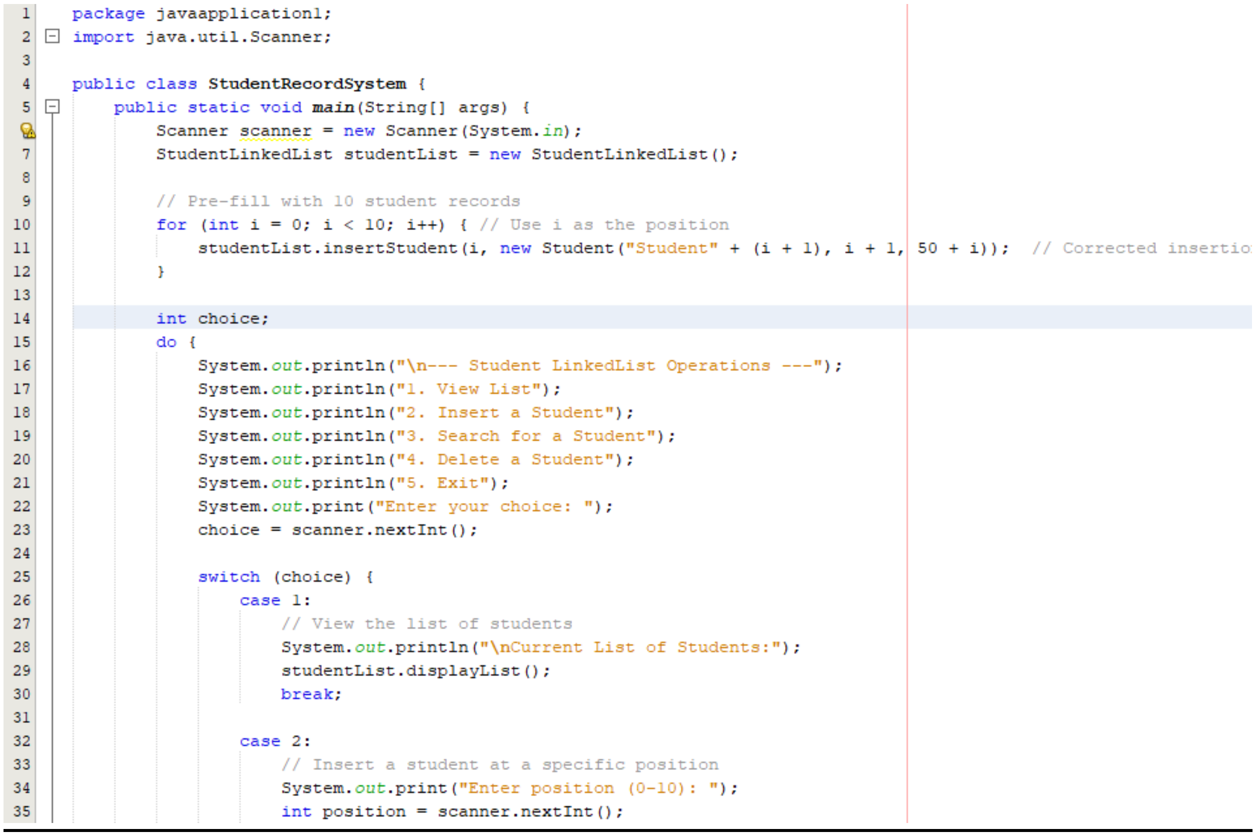
```java
package javaapplication1;
public class StudentLinkedList {
     Node head;
    int size;
    public StudentLinkedList() {
        this.head = null;
        this.size = 0;
    }
    // Method to display the list of students
    public void displayList() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }
        Node temp = head;
        while (temp != null) {
            System.out.println(temp.student);
            temp = temp.next;
        }
    }
    // Method to insert a student at a given position
    public void insertStudent(int position, Student student)
        if (position < 0 || position > size) {
            System.out.println("Invalid position.");
            return;
        }
        Node newNode = new Node(student);
        if (position == 0) {  // Insert at the beginning
            newNode.next = head;
            head = newNode;
        } else {
            Node temp = head;
            for (int i = 1; i < position; i++) {
                temp = temp.next;
            }
```



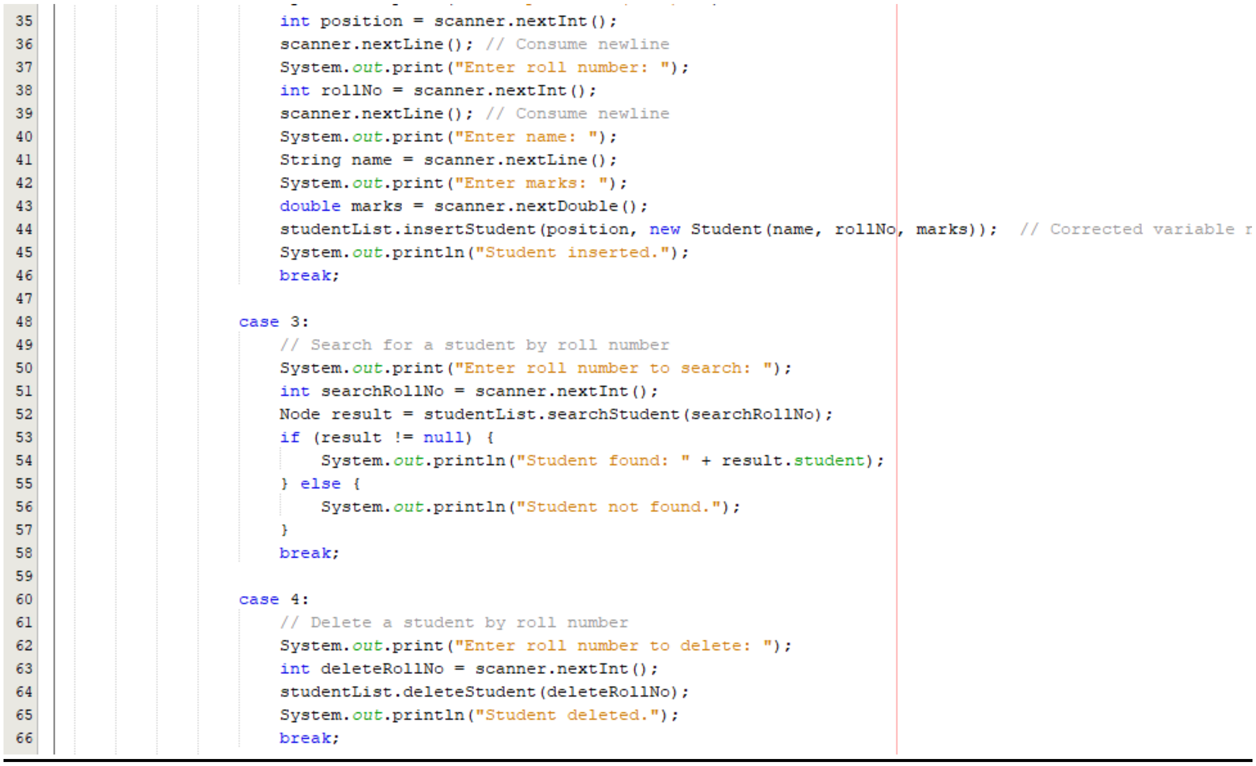
```java
            newNode.next = temp.next;
            temp.next = newNode;
        }
        size++;
    }
    // Method to search for a student by roll number
    public Node searchStudent(int rollNo) {
        Node temp = head;
        while (temp != null) {
            if (temp.student.rollno == rollNo) {
                return temp;
            }
            temp = temp.next;
        }
        return null;  // Not found
    }
    // Method to delete a student by roll number
    public void deleteStudent(int rollNo) {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }

        if (head.student.rollno == rollNo) {  // If the student to delete is the head
            head = head.next;
            size--;
            return;
        }
        Node temp = head;
        while (temp.next != null && temp.next.student.rollno != rollNo) {
            temp = temp.next;
        }

        if (temp.next == null) {
            System.out.println("Student not found.");
```



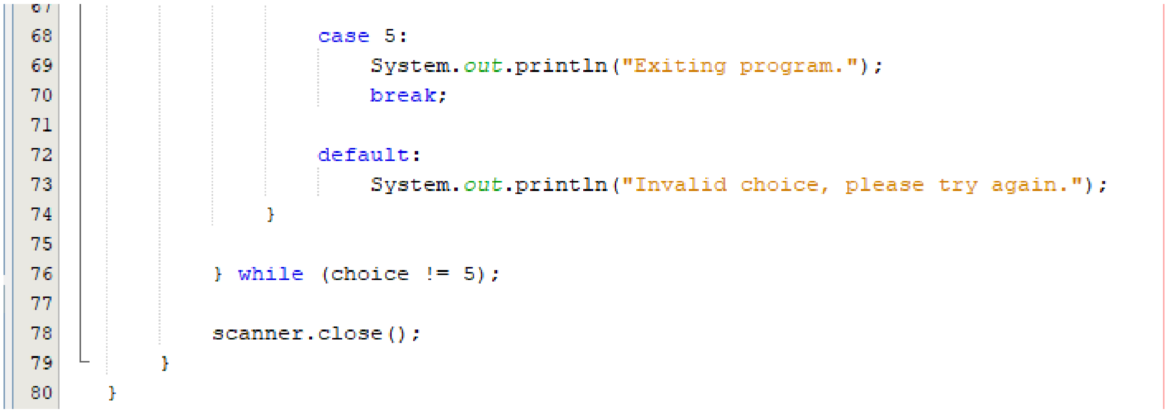
```java
        } else {
            temp.next = temp.next.next;
            size--;
        }
    }
}
```



```
package javaapplication1;
import java.util.Scanner;

public class StudentRecordSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StudentLinkedList studentList = new StudentLinkedList();

        // Pre-fill with 10 student records
        for (int i = 0; i < 10; i++) { // Use i as the position
            studentList.insertStudent(i, new Student("Student" + (i + 1), i + 1, 50 + i));  // Corrected insertio
        }

        int choice;
        do {
            System.out.println("\n--- Student LinkedList Operations ---");
            System.out.println("1. View List");
            System.out.println("2. Insert a Student");
            System.out.println("3. Search for a Student");
            System.out.println("4. Delete a Student");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    // View the list of students
                    System.out.println("\nCurrent List of Students:");
                    studentList.displayList();
                    break;

                case 2:
                    // Insert a student at a specific position
                    System.out.print("Enter position (0-10): ");
                    int position = scanner.nextInt();
```

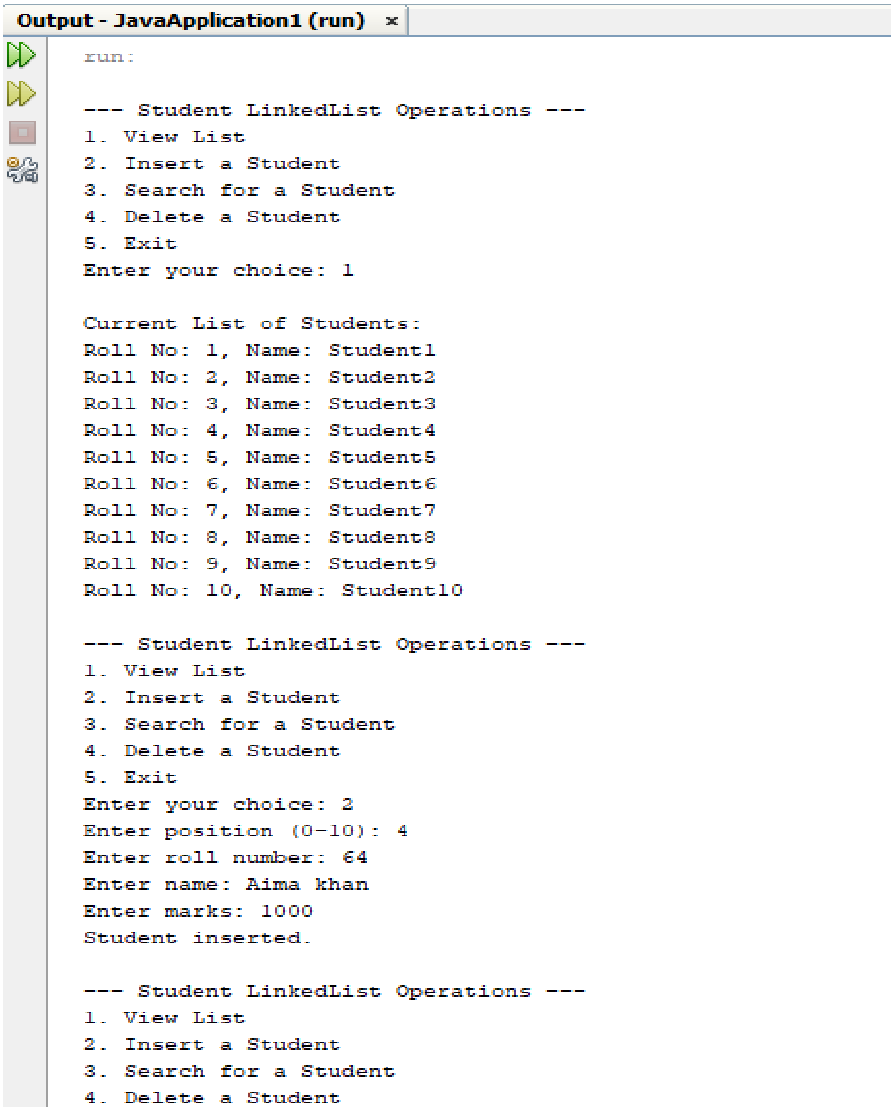


```
                    int position = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
                    System.out.print("Enter roll number: ");
                    int rollNo = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
                    System.out.print("Enter name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter marks: ");
                    double marks = scanner.nextDouble();
                    studentList.insertStudent(position, new Student(name, rollNo, marks));  // Corrected variable r
                    System.out.println("Student inserted.");
                    break;

                case 3:
                    // Search for a student by roll number
                    System.out.print("Enter roll number to search: ");
                    int searchRollNo = scanner.nextInt();
                    Node result = studentList.searchStudent(searchRollNo);
                    if (result != null) {
                        System.out.println("Student found: " + result.student);
                    } else {
                        System.out.println("Student not found.");
                    }
                    break;

                case 4:
                    // Delete a student by roll number
                    System.out.print("Enter roll number to delete: ");
                    int deleteRollNo = scanner.nextInt();
                    studentList.deleteStudent(deleteRollNo);
                    System.out.println("Student deleted.");
                    break;
```

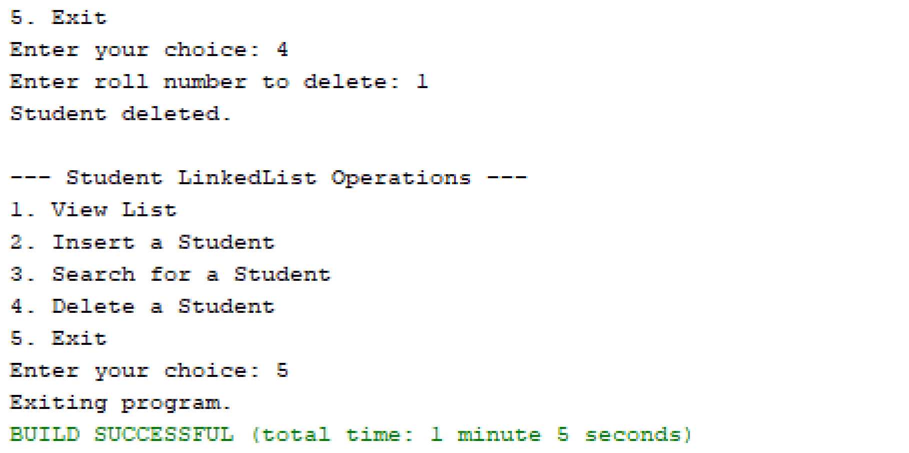


```
                    case 5:
                        System.out.println("Exiting program.");
                        break;

                    default:
                        System.out.println("Invalid choice, please try again.");
                }

            } while (choice != 5);

            scanner.close();
        }
    }
```

OUTPUT:



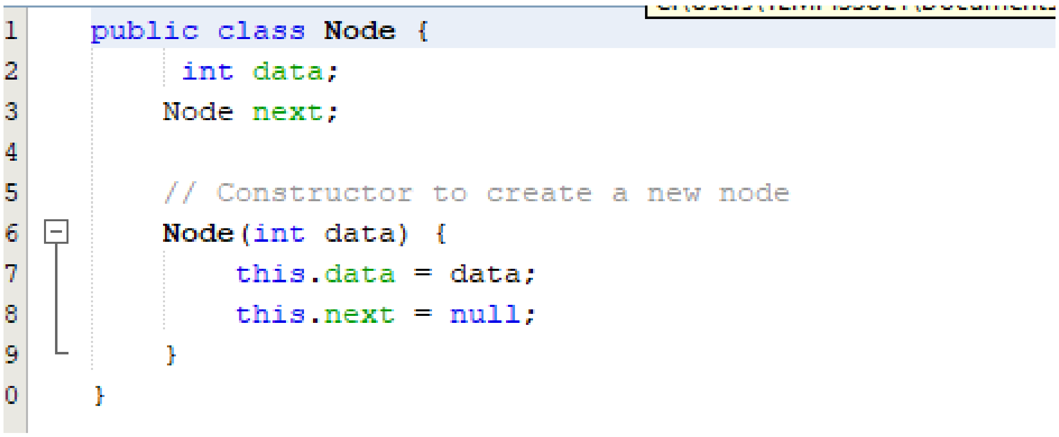
```
run:

--- Student LinkedList Operations ---
1. View List
2. Insert a Student
3. Search for a Student
4. Delete a Student
5. Exit
Enter your choice: 1

Current List of Students:
Roll No: 1, Name: Student1
Roll No: 2, Name: Student2
Roll No: 3, Name: Student3
Roll No: 4, Name: Student4
Roll No: 5, Name: Student5
Roll No: 6, Name: Student6
Roll No: 7, Name: Student7
Roll No: 8, Name: Student8
Roll No: 9, Name: Student9
Roll No: 10, Name: Student10

--- Student LinkedList Operations ---
1. View List
2. Insert a Student
3. Search for a Student
4. Delete a Student
5. Exit
Enter your choice: 2
Enter position (0-10): 4
Enter roll number: 64
Enter name: Aima khan
Enter marks: 1000
Student inserted.

--- Student LinkedList Operations ---
1. View List
2. Insert a Student
3. Search for a Student
4. Delete a Student
```



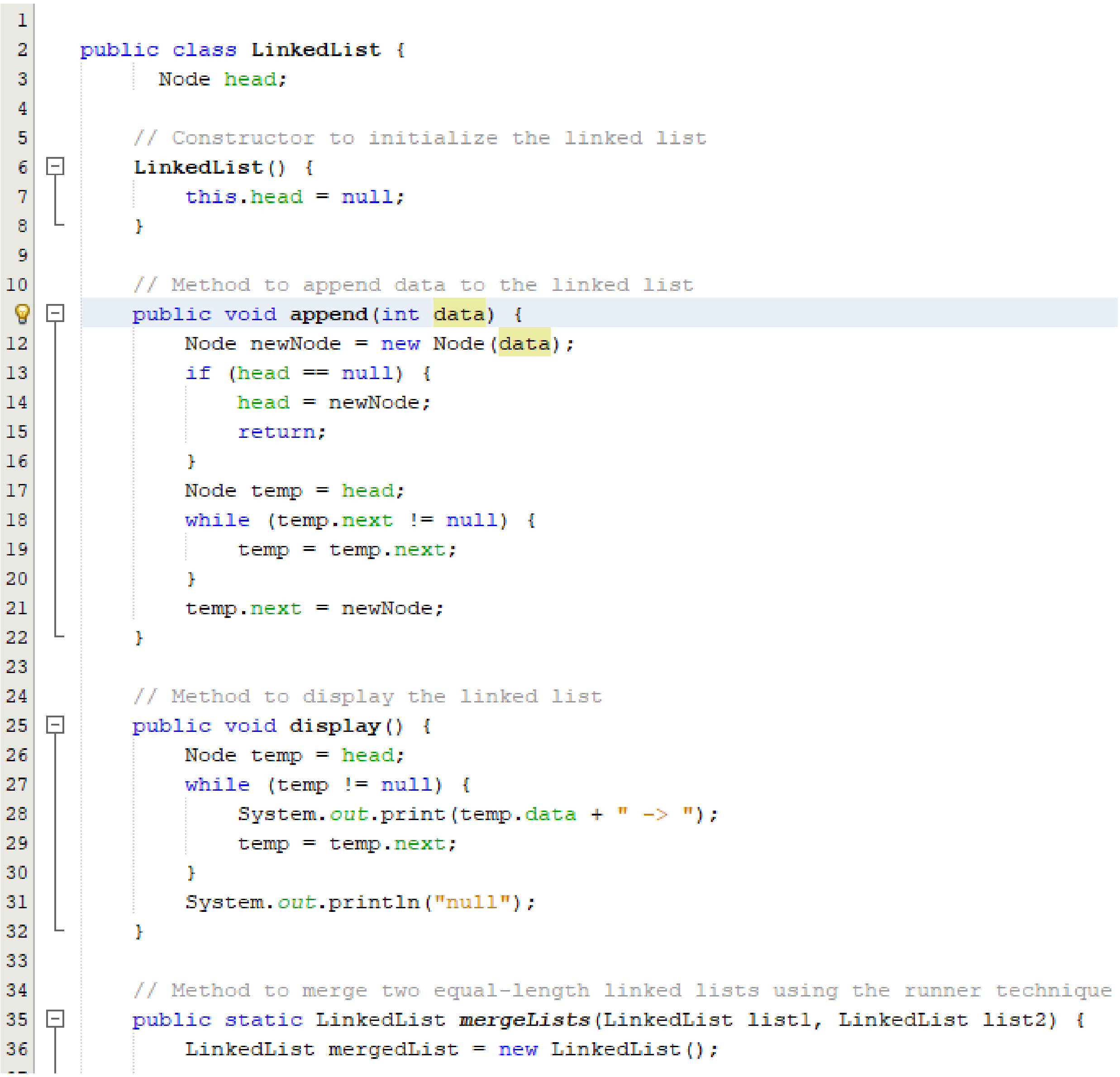
```
5. Exit
Enter your choice: 4
Enter roll number to delete: 1
Student deleted.

--- Student LinkedList Operations ---
1. View List
2. Insert a Student
3. Search for a Student
4. Delete a Student
5. Exit
Enter your choice: 5
Exiting program.
BUILD SUCCESSFUL (total time: 1 minute 5 seconds)
```

2. Write a java program to merge two equal linkedlists using runner technique.

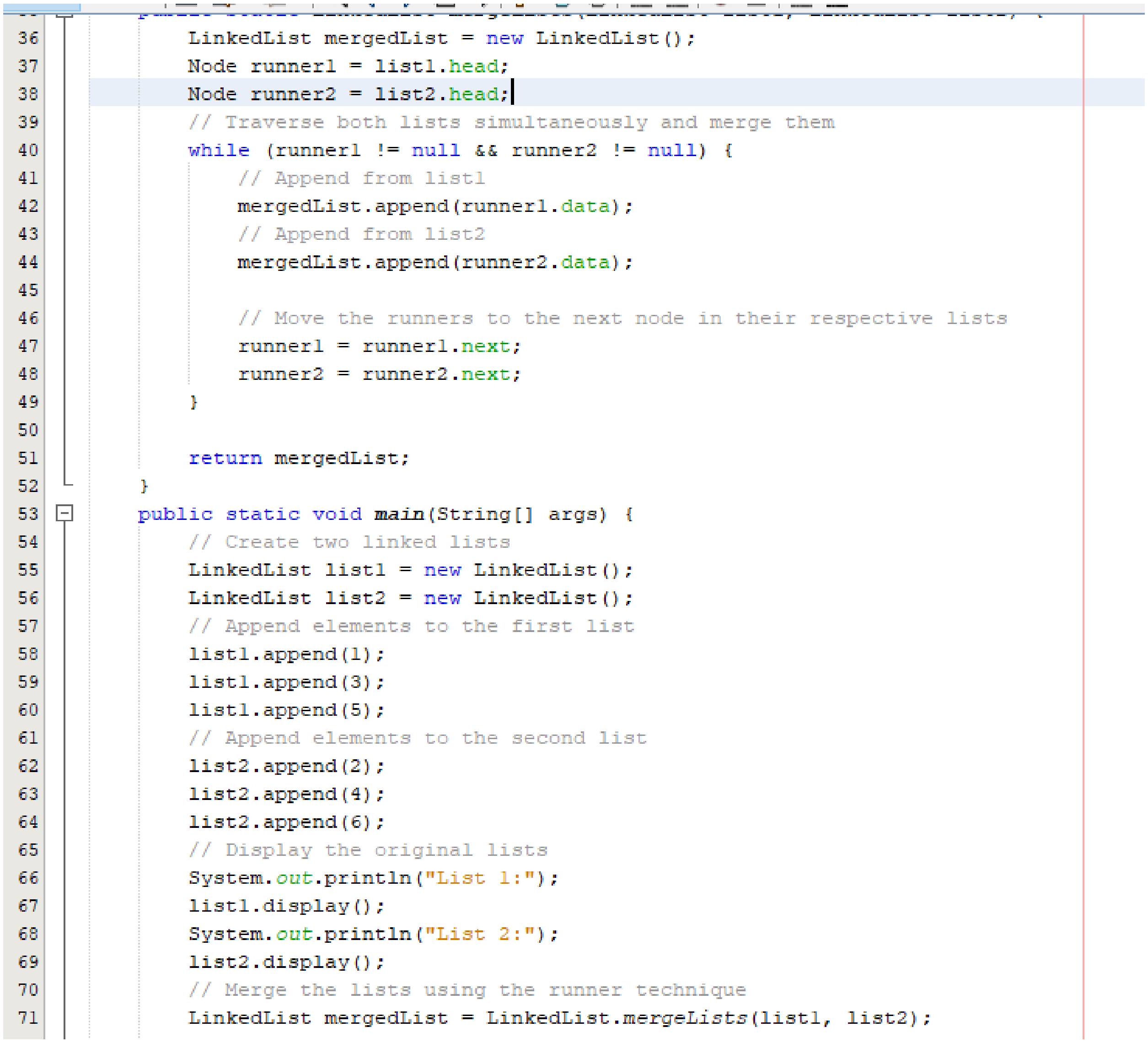
CODE:



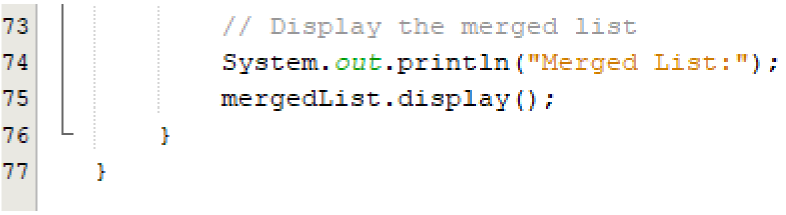
```java
public class Node {
    int data;
    Node next;

    // Constructor to create a new node
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```



```java

public class LinkedList {
    Node head;

    // Constructor to initialize the linked list
    LinkedList() {
        this.head = null;
    }

    // Method to append data to the linked list
    public void append(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;
    }

    // Method to display the linked list
    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }

    // Method to merge two equal-length linked lists using the runner technique
    public static LinkedList mergeLists(LinkedList list1, LinkedList list2) {
        LinkedList mergedList = new LinkedList();
```

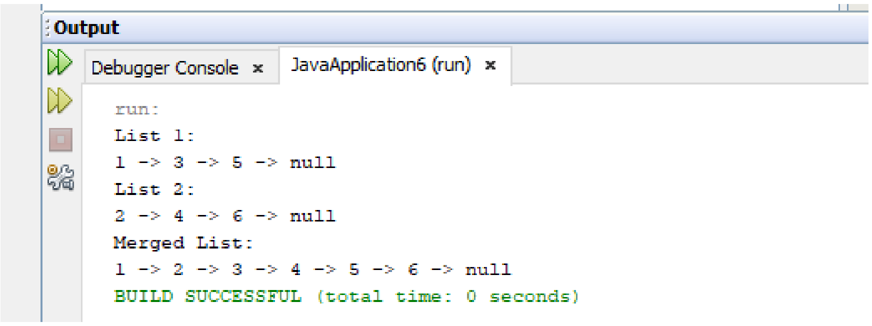


```
        LinkedList mergedList = new LinkedList();
        Node runner1 = list1.head;
        Node runner2 = list2.head;
        // Traverse both lists simultaneously and merge them
        while (runner1 != null && runner2 != null) {
            // Append from list1
            mergedList.append(runner1.data);
            // Append from list2
            mergedList.append(runner2.data);

            // Move the runners to the next node in their respective lists
            runner1 = runner1.next;
            runner2 = runner2.next;
        }

        return mergedList;
    }
    public static void main(String[] args) {
        // Create two linked lists
        LinkedList list1 = new LinkedList();
        LinkedList list2 = new LinkedList();
        // Append elements to the first list
        list1.append(1);
        list1.append(3);
        list1.append(5);
        // Append elements to the second list
        list2.append(2);
        list2.append(4);
        list2.append(6);
        // Display the original lists
        System.out.println("List 1:");
        list1.display();
        System.out.println("List 2:");
        list2.display();
        // Merge the lists using the runner technique
        LinkedList mergedList = LinkedList.mergeLists(list1, list2);
```



```
        // Display the merged list
        System.out.println("Merged List:");
        mergedList.display();
    }
}
```

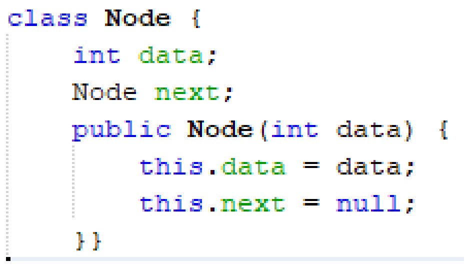
Output:



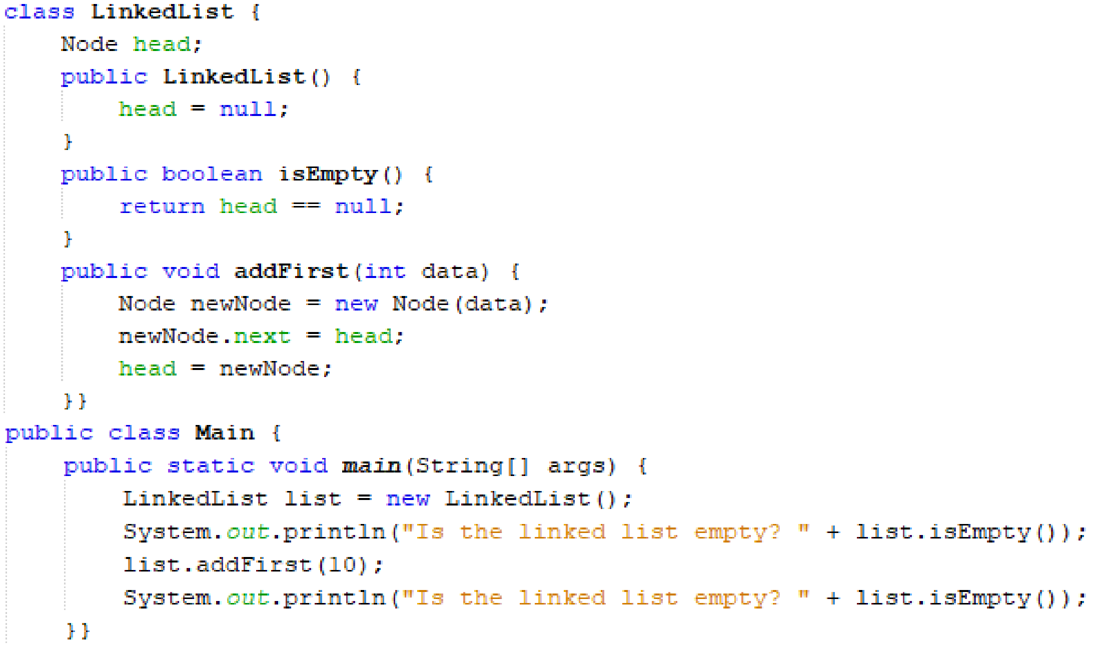
```
run:
List 1:
1 -> 3 -> 5 -> null
List 2:
2 -> 4 -> 6 -> null
Merged List:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Write a program to check whether the linkedlist is empty or not.

CODE:

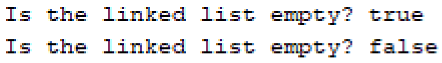


```
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }}
```



```
class LinkedList {
    Node head;
    public LinkedList() {
        head = null;
    }
    public boolean isEmpty() {
        return head == null;
    }
    public void addFirst(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }}
public class Main {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        System.out.println("Is the linked list empty? " + list.isEmpty());
        list.addFirst(10);
        System.out.println("Is the linked list empty? " + list.isEmpty());
    }}
```

Output:

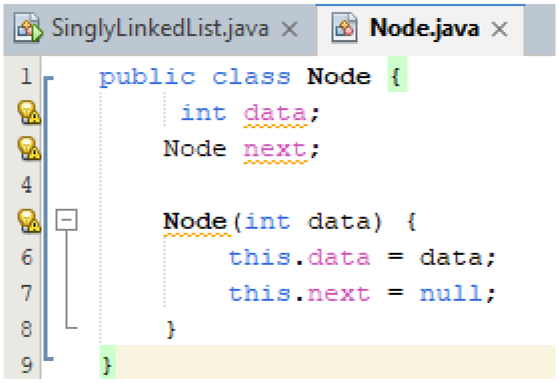


```
Is the linked list empty? true
Is the linked list empty? false
```

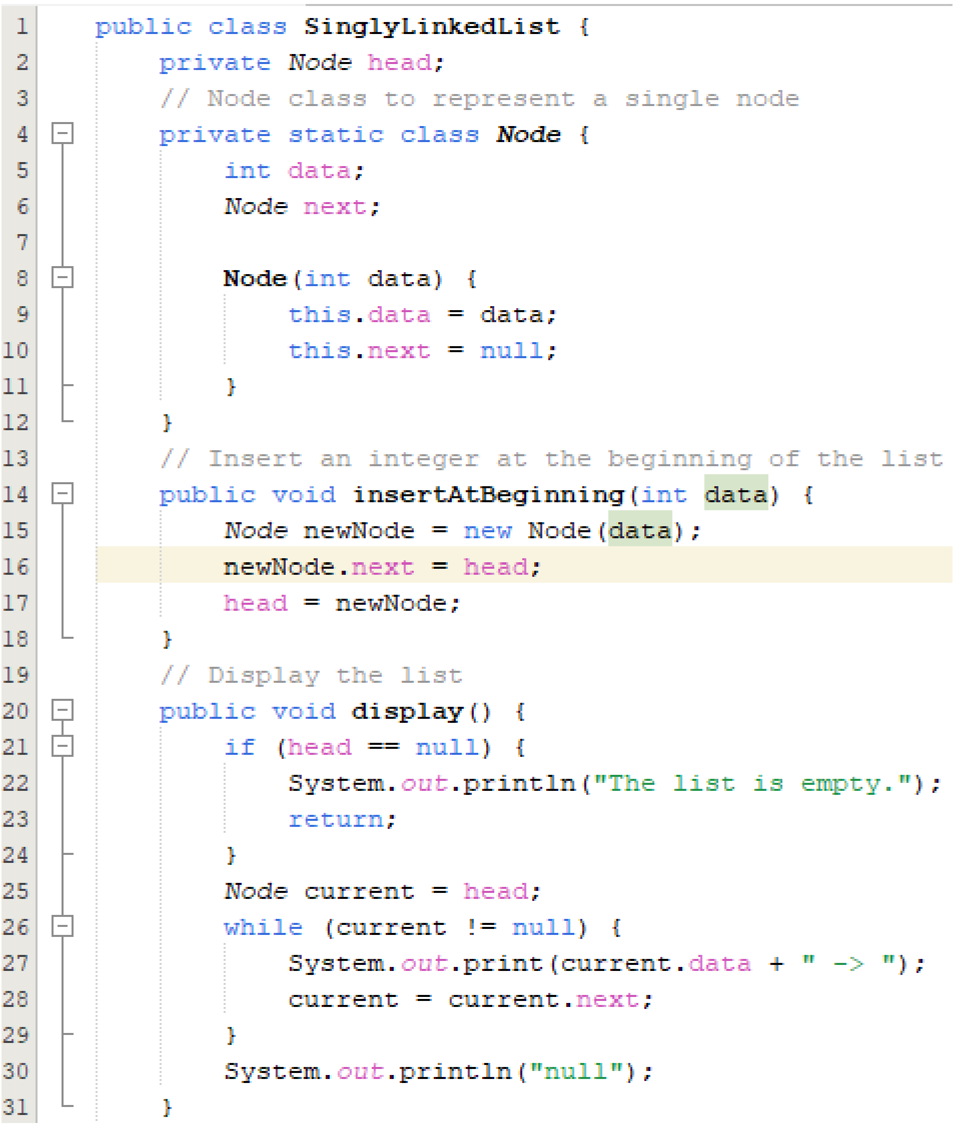
4. You are managing a list of integers in a class, and you need to implement a **Singly Linked List** with the following operations:

a) **Insert** an integer at the **beginning** of the list.
b) **Display** the list.
c) Find the **middle element** of the list. If the list has an even number of elements, return the **first middle element**.

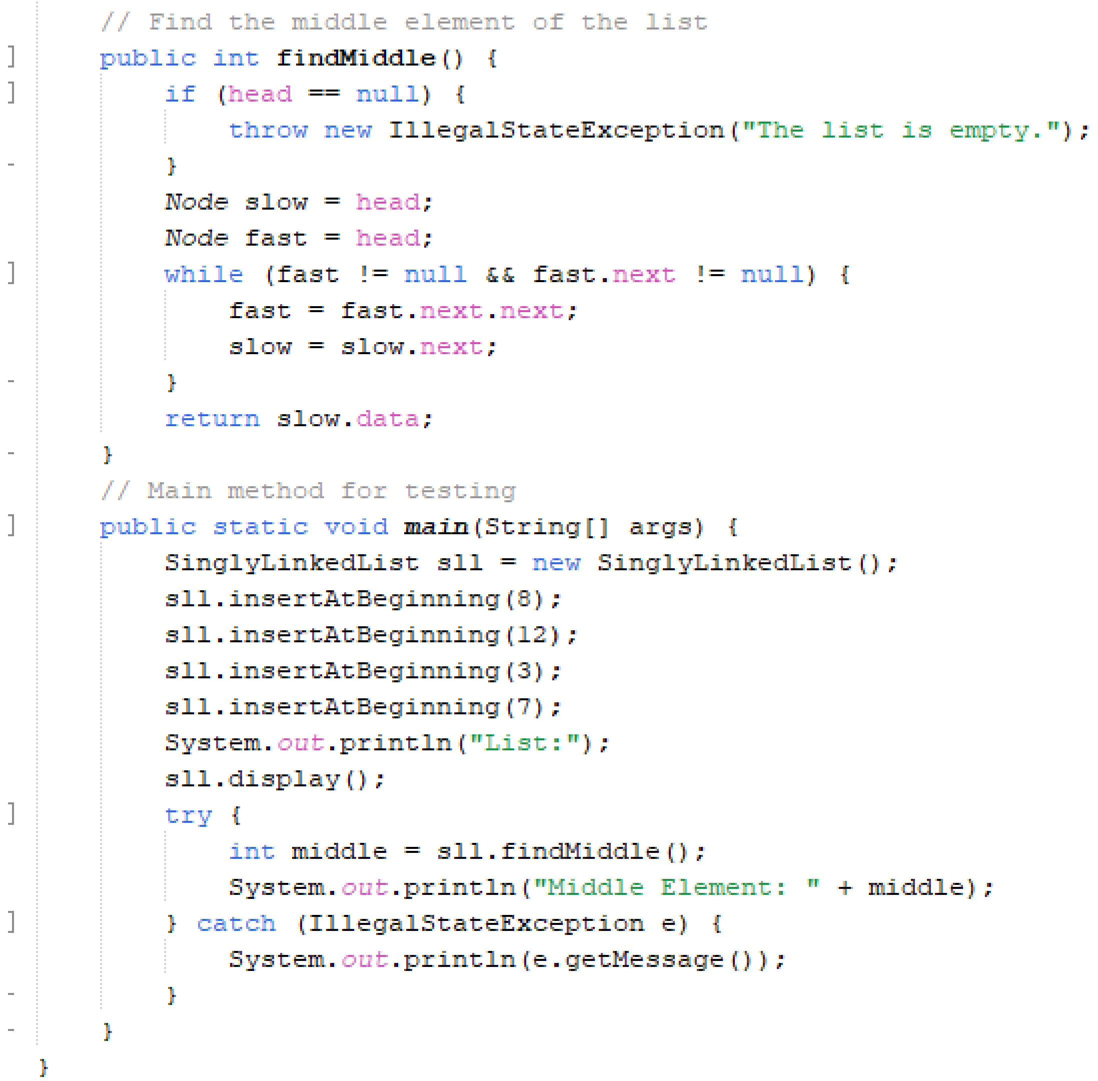
CODE:



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

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

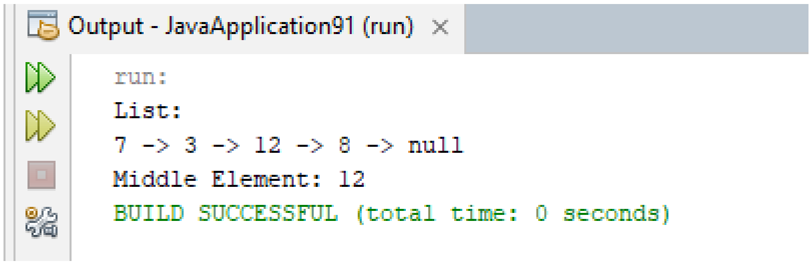


```java
public class SinglyLinkedList {
    private Node head;
    // Node class to represent a single node
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    // Insert an integer at the beginning of the list
    public void insertAtBeginning(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }
    // Display the list
    public void display() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }
```



```
    // Find the middle element of the list
    public int findMiddle() {
        if (head == null) {
            throw new IllegalStateException("The list is empty.");
        }
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow.data;
    }
    // Main method for testing
    public static void main(String[] args) {
        SinglyLinkedList sll = new SinglyLinkedList();
        sll.insertAtBeginning(8);
        sll.insertAtBeginning(12);
        sll.insertAtBeginning(3);
        sll.insertAtBeginning(7);
        System.out.println("List:");
        sll.display();
        try {
            int middle = sll.findMiddle();
            System.out.println("Middle Element: " + middle);
        } catch (IllegalStateException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

OUTPUT:



```
run:
List:
7 -> 3 -> 12 -> 8 -> null
Middle Element: 12
BUILD SUCCESSFUL (total time: 0 seconds)
```

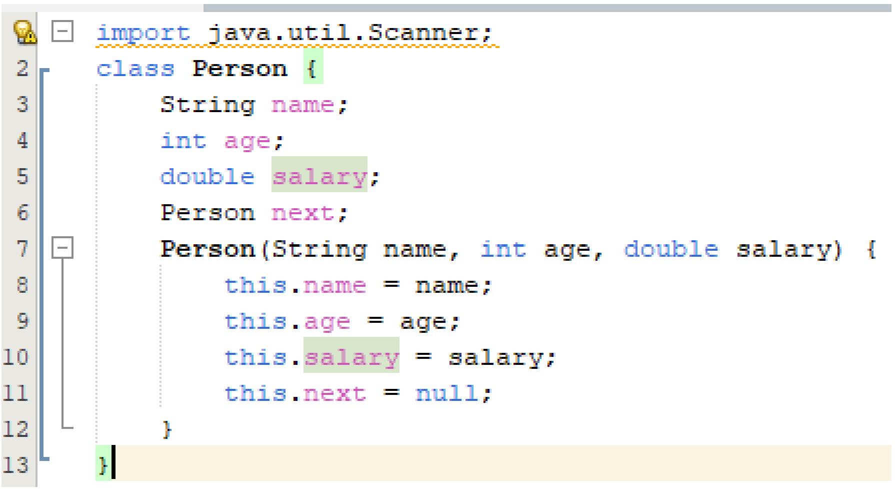
Home Task for Singly linked list

1. Write a program that reads the name, age and salary of 10 persons and perform the following operations on it.

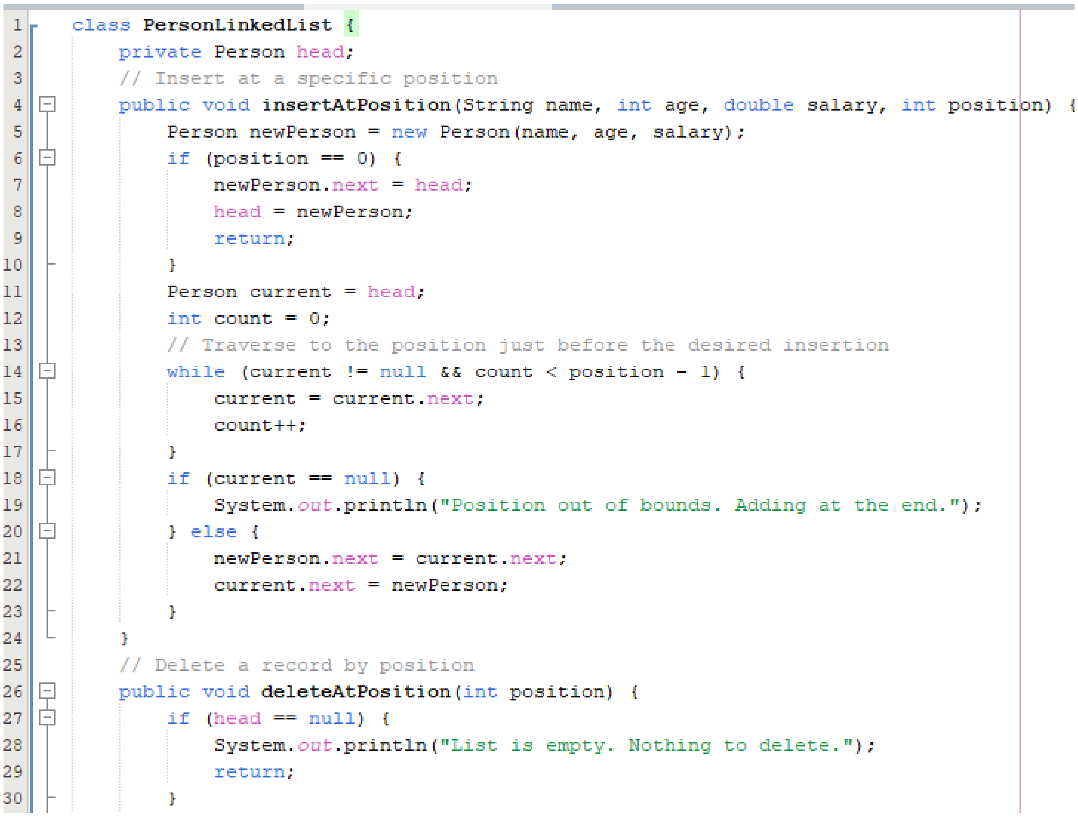
   a. Insert the elements in different locations of linked list and view it.

   b. Delete record and again view the list after deletion.

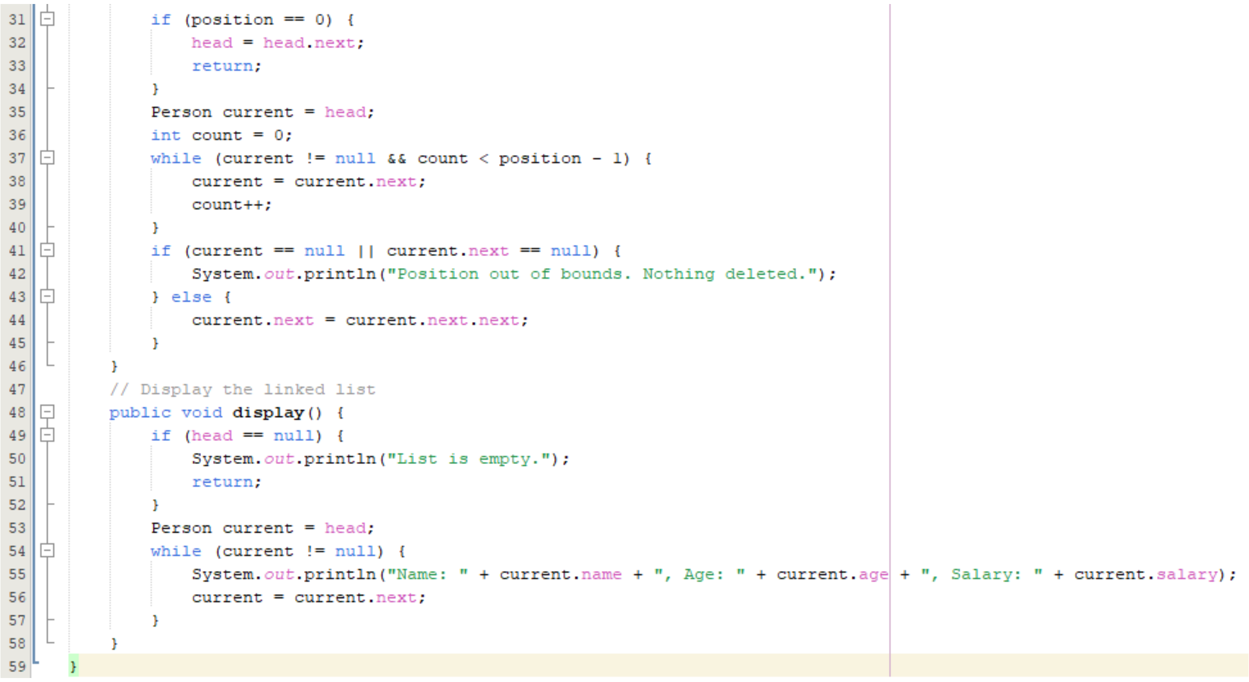
CODE:



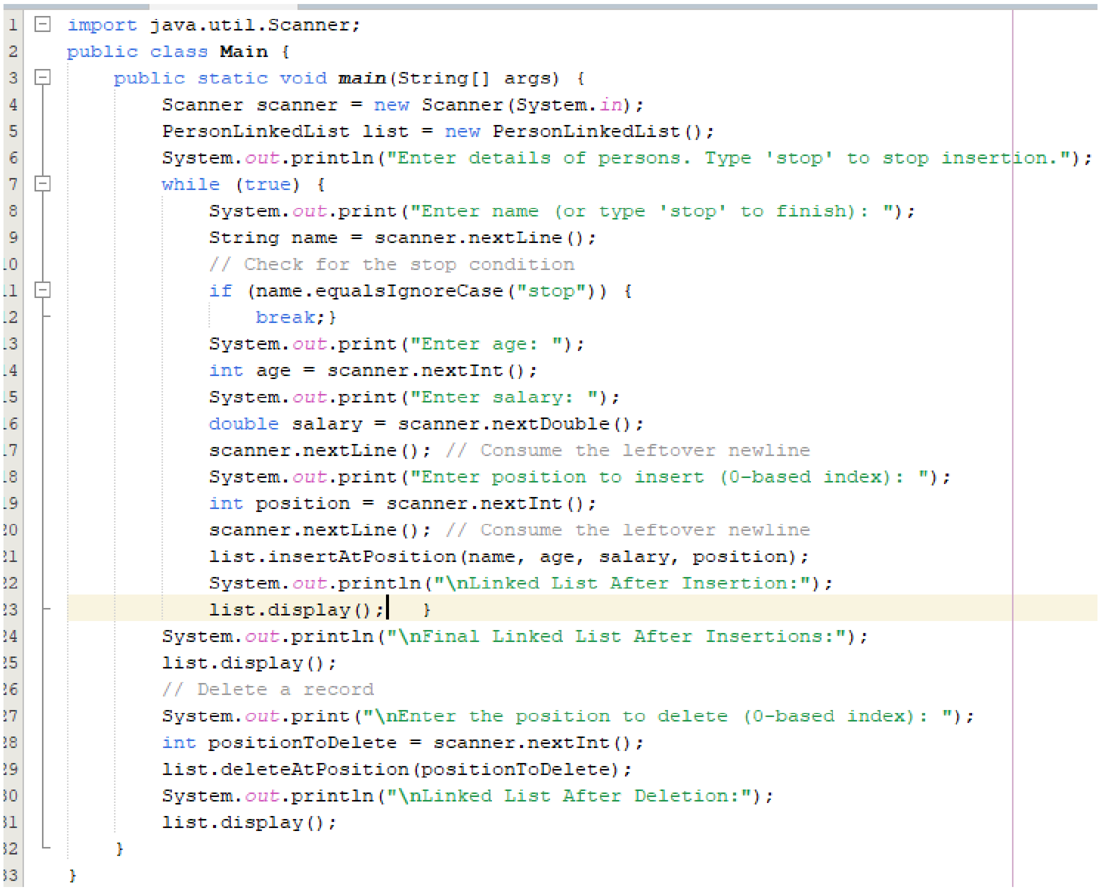
```java
import java.util.Scanner;
class Person {
    String name;
    int age;
    double salary;
    Person next;
    Person(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
        this.next = null;
    }
}
```



```java
class PersonLinkedList {
    private Person head;
    // Insert at a specific position
    public void insertAtPosition(String name, int age, double salary, int position) {
        Person newPerson = new Person(name, age, salary);
        if (position == 0) {
            newPerson.next = head;
            head = newPerson;
            return;
        }
        Person current = head;
        int count = 0;
        // Traverse to the position just before the desired insertion
        while (current != null && count < position - 1) {
            current = current.next;
            count++;
        }
        if (current == null) {
            System.out.println("Position out of bounds. Adding at the end.");
        } else {
            newPerson.next = current.next;
            current.next = newPerson;
        }
    }
    // Delete a record by position
    public void deleteAtPosition(int position) {
        if (head == null) {
            System.out.println("List is empty. Nothing to delete.");
            return;
        }
```

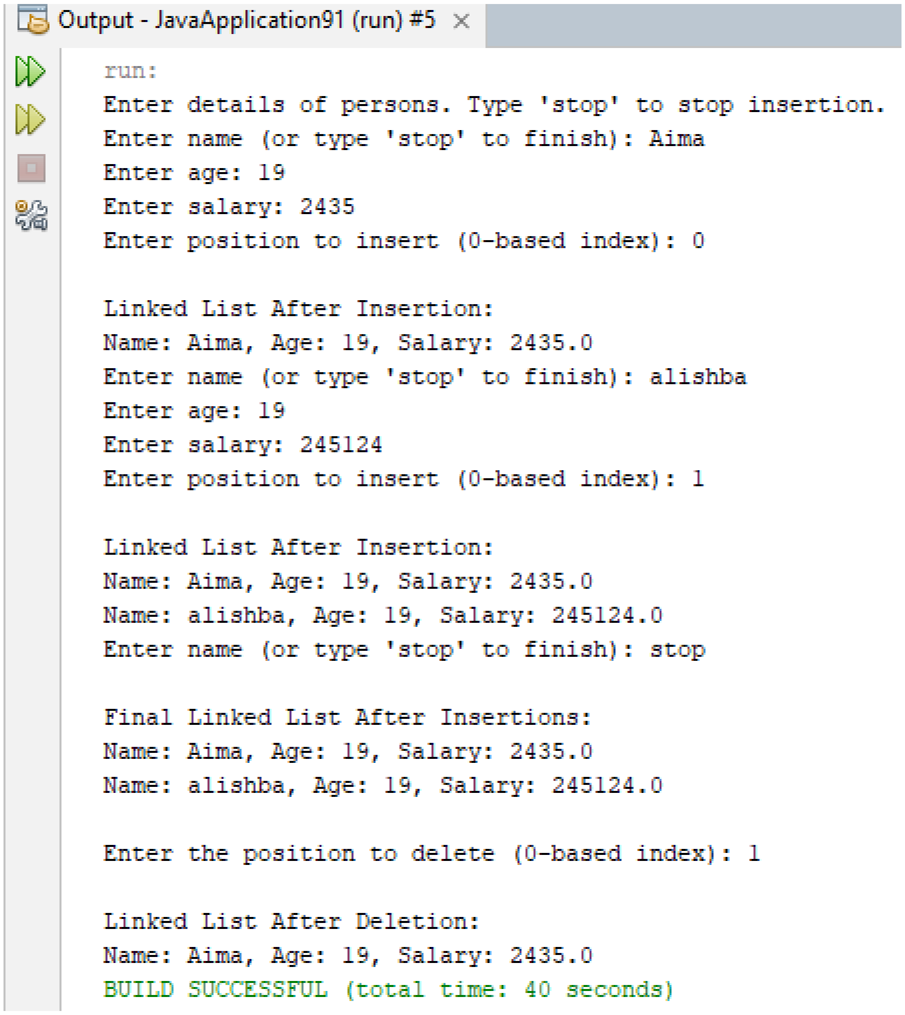


```java
        if (position == 0) {
            head = head.next;
            return;
        }
        Person current = head;
        int count = 0;
        while (current != null && count < position - 1) {
            current = current.next;
            count++;
        }
        if (current == null || current.next == null) {
            System.out.println("Position out of bounds. Nothing deleted.");
        } else {
            current.next = current.next.next;
        }
    }
    // Display the linked list
    public void display() {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
        Person current = head;
        while (current != null) {
            System.out.println("Name: " + current.name + ", Age: " + current.age + ", Salary: " + current.salary);
            current = current.next;
        }
    }
}
```



```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PersonLinkedList list = new PersonLinkedList();
        System.out.println("Enter details of persons. Type 'stop' to stop insertion.");
        while (true) {
            System.out.print("Enter name (or type 'stop' to finish): ");
            String name = scanner.nextLine();
            // Check for the stop condition
            if (name.equalsIgnoreCase("stop")) {
                break;}
            System.out.print("Enter age: ");
            int age = scanner.nextInt();
            System.out.print("Enter salary: ");
            double salary = scanner.nextDouble();
            scanner.nextLine(); // Consume the leftover newline
            System.out.print("Enter position to insert (0-based index): ");
            int position = scanner.nextInt();
            scanner.nextLine(); // Consume the leftover newline
            list.insertAtPosition(name, age, salary, position);
            System.out.println("\nLinked List After Insertion:");
            list.display();    }
        System.out.println("\nFinal Linked List After Insertions:");
        list.display();
        // Delete a record
        System.out.print("\nEnter the position to delete (0-based index): ");
        int positionToDelete = scanner.nextInt();
        list.deleteAtPosition(positionToDelete);
        System.out.println("\nLinked List After Deletion:");
        list.display();
    }
}
```

OUTPUT:

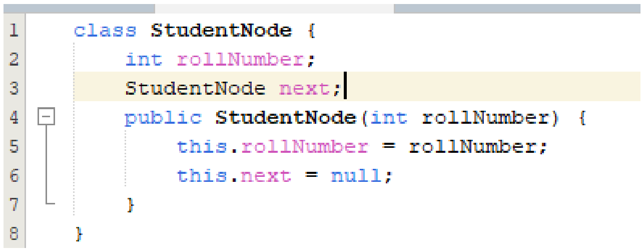


```
run:
Enter details of persons. Type 'stop' to stop insertion.
Enter name (or type 'stop' to finish): Aima
Enter age: 19
Enter salary: 2435
Enter position to insert (0-based index): 0

Linked List After Insertion:
Name: Aima, Age: 19, Salary: 2435.0
Enter name (or type 'stop' to finish): alishba
Enter age: 19
Enter salary: 245124
Enter position to insert (0-based index): 1

Linked List After Insertion:
Name: Aima, Age: 19, Salary: 2435.0
Name: alishba, Age: 19, Salary: 245124.0
Enter name (or type 'stop' to finish): stop

Final Linked List After Insertions:
Name: Aima, Age: 19, Salary: 2435.0
Name: alishba, Age: 19, Salary: 245124.0

Enter the position to delete (0-based index): 1

Linked List After Deletion:
Name: Aima, Age: 19, Salary: 2435.0
BUILD SUCCESSFUL (total time: 40 seconds)
```

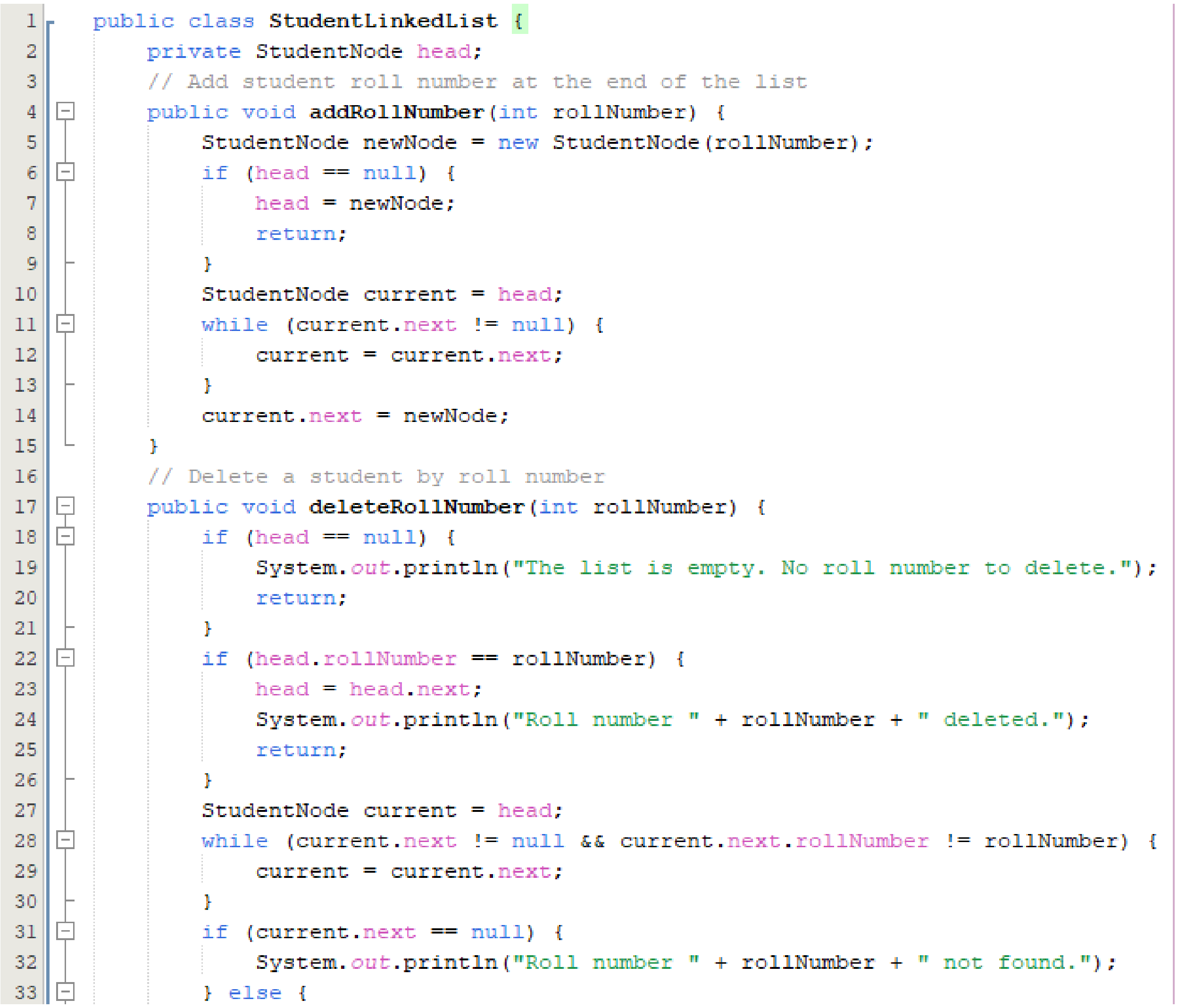
2. You are tasked with managing a list of **students' roll numbers** in a class. Initially, the list is empty. You have to implement a **Singly Linked List** with the following operations:

a) **Add student roll number** at the **end** of the list.
b) **Delete a student by roll number**.
c) **Display the roll numbers** of all students in the class

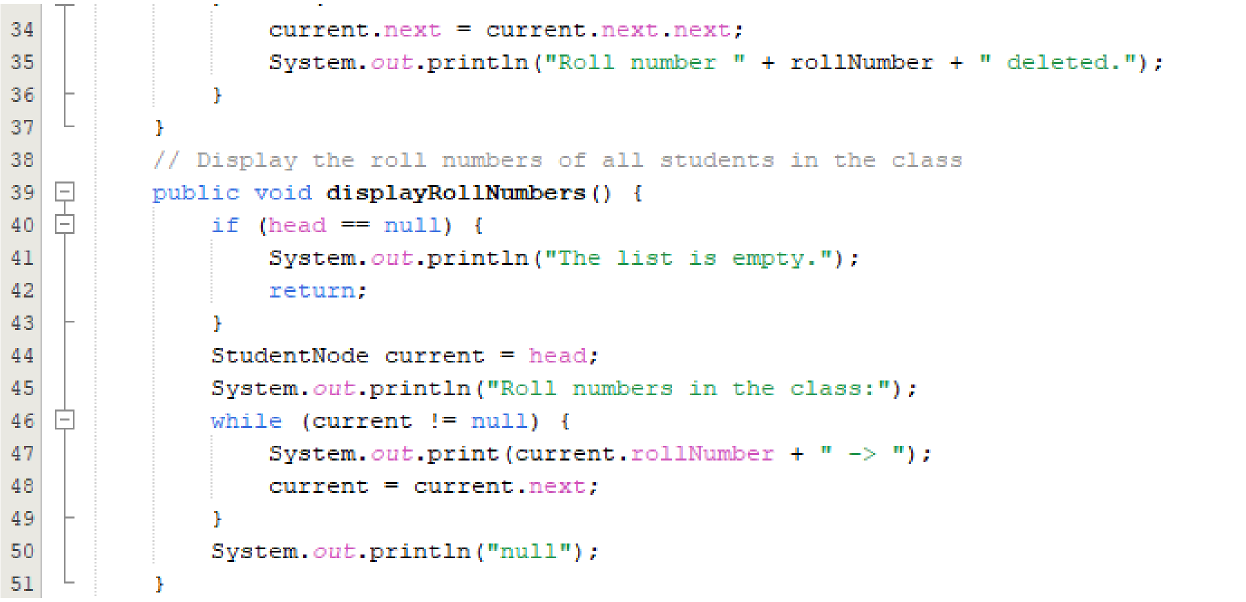
CODE:



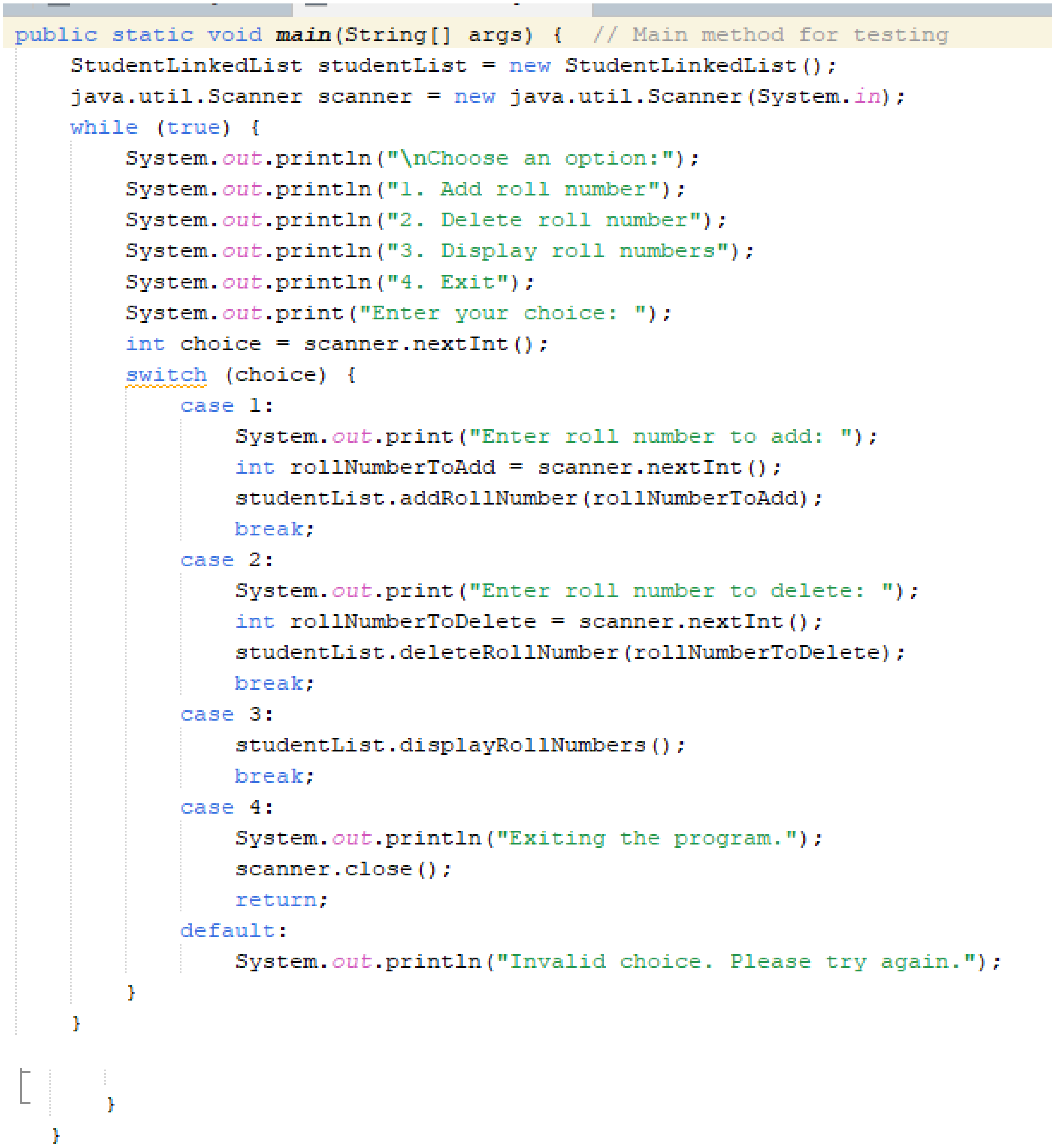
```
class StudentNode {
    int rollNumber;
    StudentNode next;
    public StudentNode(int rollNumber) {
        this.rollNumber = rollNumber;
        this.next = null;
    }
}
```



```java
public class StudentLinkedList {
    private StudentNode head;
    // Add student roll number at the end of the list
    public void addRollNumber(int rollNumber) {
        StudentNode newNode = new StudentNode(rollNumber);
        if (head == null) {
            head = newNode;
            return;
        }
        StudentNode current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }
    // Delete a student by roll number
    public void deleteRollNumber(int rollNumber) {
        if (head == null) {
            System.out.println("The list is empty. No roll number to delete.");
            return;
        }
        if (head.rollNumber == rollNumber) {
            head = head.next;
            System.out.println("Roll number " + rollNumber + " deleted.");
            return;
        }
        StudentNode current = head;
        while (current.next != null && current.next.rollNumber != rollNumber) {
            current = current.next;
        }
        if (current.next == null) {
            System.out.println("Roll number " + rollNumber + " not found.");
        } else {
```

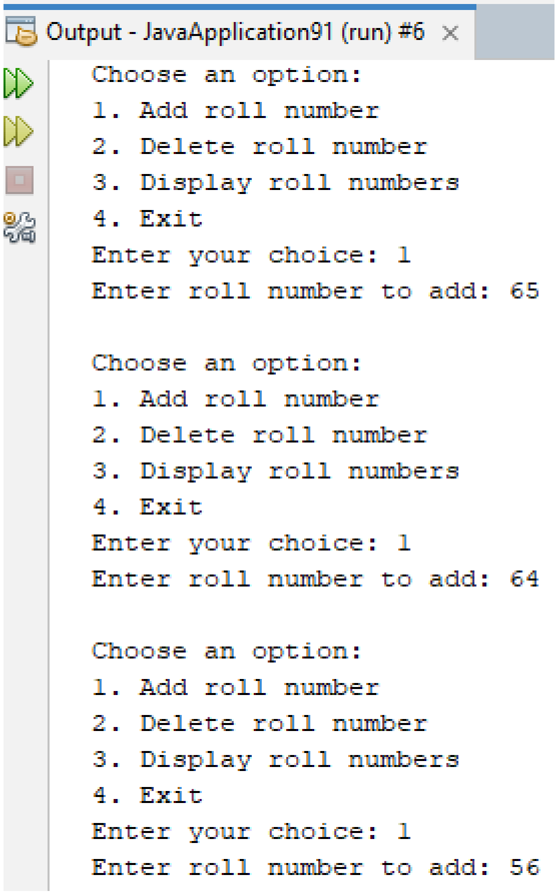


```java
            current.next = current.next.next;
            System.out.println("Roll number " + rollNumber + " deleted.");
        }
    }
    // Display the roll numbers of all students in the class
    public void displayRollNumbers() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }
        StudentNode current = head;
        System.out.println("Roll numbers in the class:");
        while (current != null) {
            System.out.print(current.rollNumber + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }
```

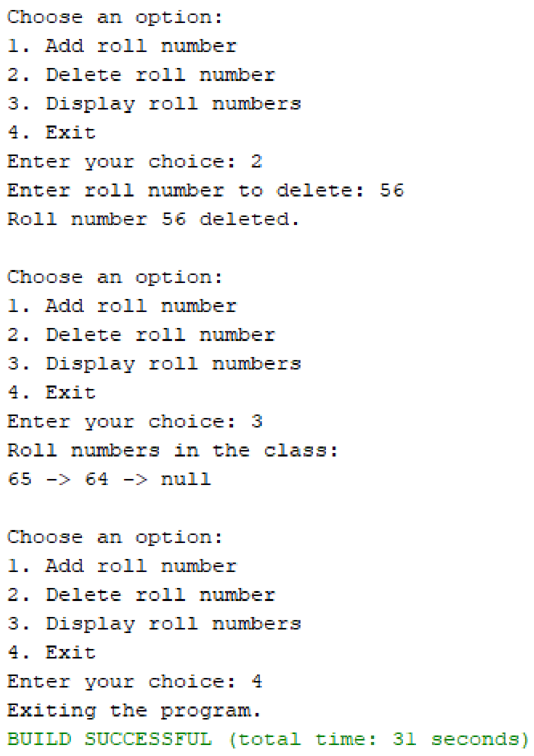


```
public static void main(String[] args) {  // Main method for testing
    StudentLinkedList studentList = new StudentLinkedList();
    java.util.Scanner scanner = new java.util.Scanner(System.in);
    while (true) {
        System.out.println("\nChoose an option:");
        System.out.println("1. Add roll number");
        System.out.println("2. Delete roll number");
        System.out.println("3. Display roll numbers");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter roll number to add: ");
                int rollNumberToAdd = scanner.nextInt();
                studentList.addRollNumber(rollNumberToAdd);
                break;
            case 2:
                System.out.print("Enter roll number to delete: ");
                int rollNumberToDelete = scanner.nextInt();
                studentList.deleteRollNumber(rollNumberToDelete);
                break;
            case 3:
                studentList.displayRollNumbers();
                break;
            case 4:
                System.out.println("Exiting the program.");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }

      }
  }
```

OUTPUT:



```
Choose an option:
1. Add roll number
2. Delete roll number
3. Display roll numbers
4. Exit
Enter your choice: 1
Enter roll number to add: 65

Choose an option:
1. Add roll number
2. Delete roll number
3. Display roll numbers
4. Exit
Enter your choice: 1
Enter roll number to add: 64

Choose an option:
1. Add roll number
2. Delete roll number
3. Display roll numbers
4. Exit
Enter your choice: 1
Enter roll number to add: 56
```

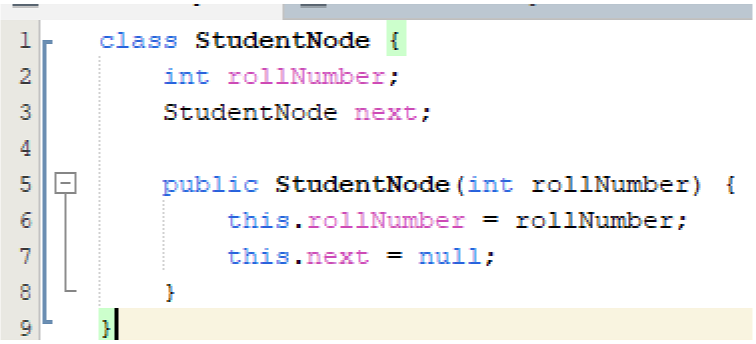


```
Choose an option:
1. Add roll number
2. Delete roll number
3. Display roll numbers
4. Exit
Enter your choice: 2
Enter roll number to delete: 56
Roll number 56 deleted.

Choose an option:
1. Add roll number
2. Delete roll number
3. Display roll numbers
4. Exit
Enter your choice: 3
Roll numbers in the class:
65 -> 64 -> null

Choose an option:
1. Add roll number
2. Delete roll number
3. Display roll numbers
4. Exit
Enter your choice: 4
Exiting the program.
BUILD SUCCESSFUL (total time: 31 seconds)
```

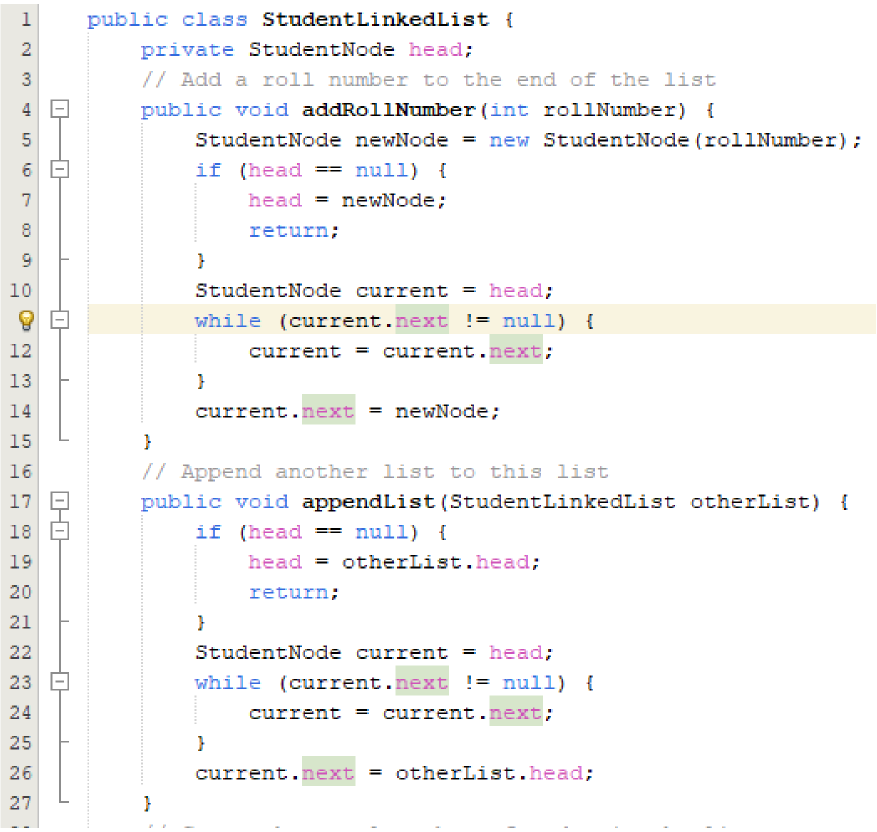
3. You are managing two **singly linked lists** representing **two groups of students**. Your task is to:

a) **Append** the second list to the first list (i.e., add all elements of the second list to the end of the first list).
b) **Count the number of students** in the final list (i.e., the total number of nodes in the list).
c) **Display the final list** after the append operation.

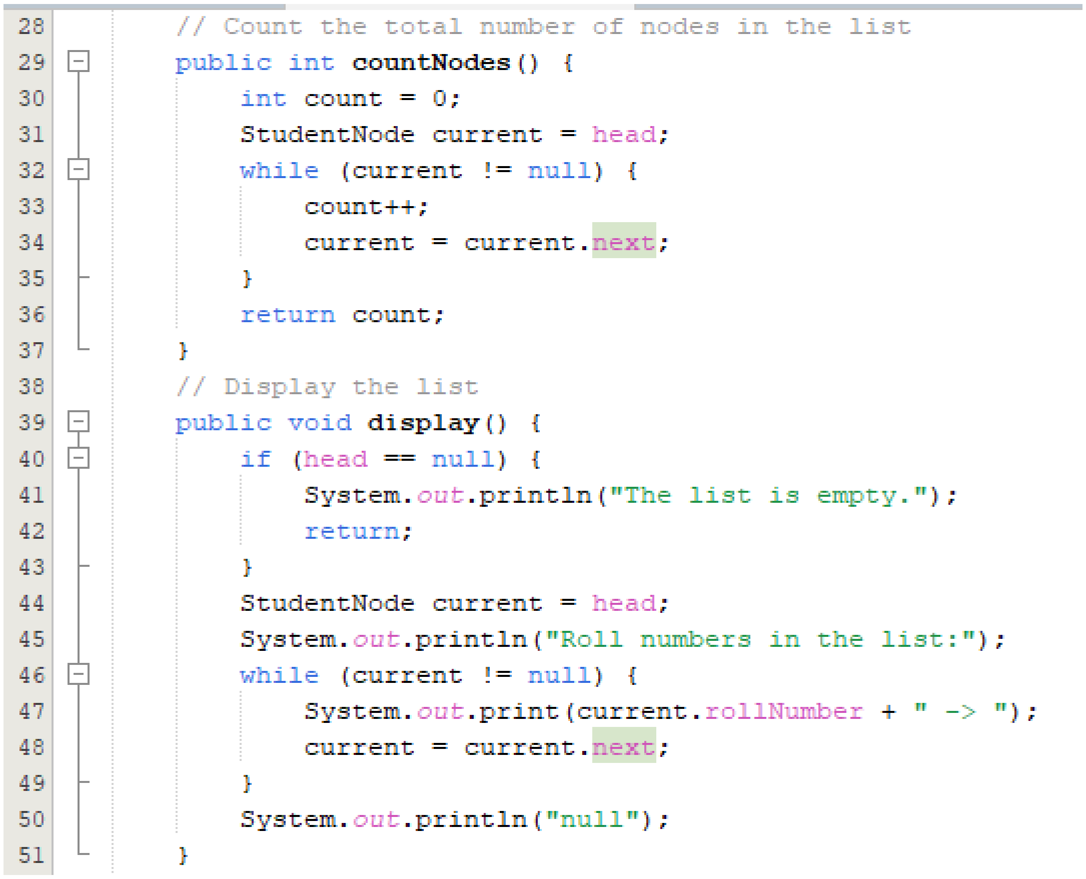
CODE



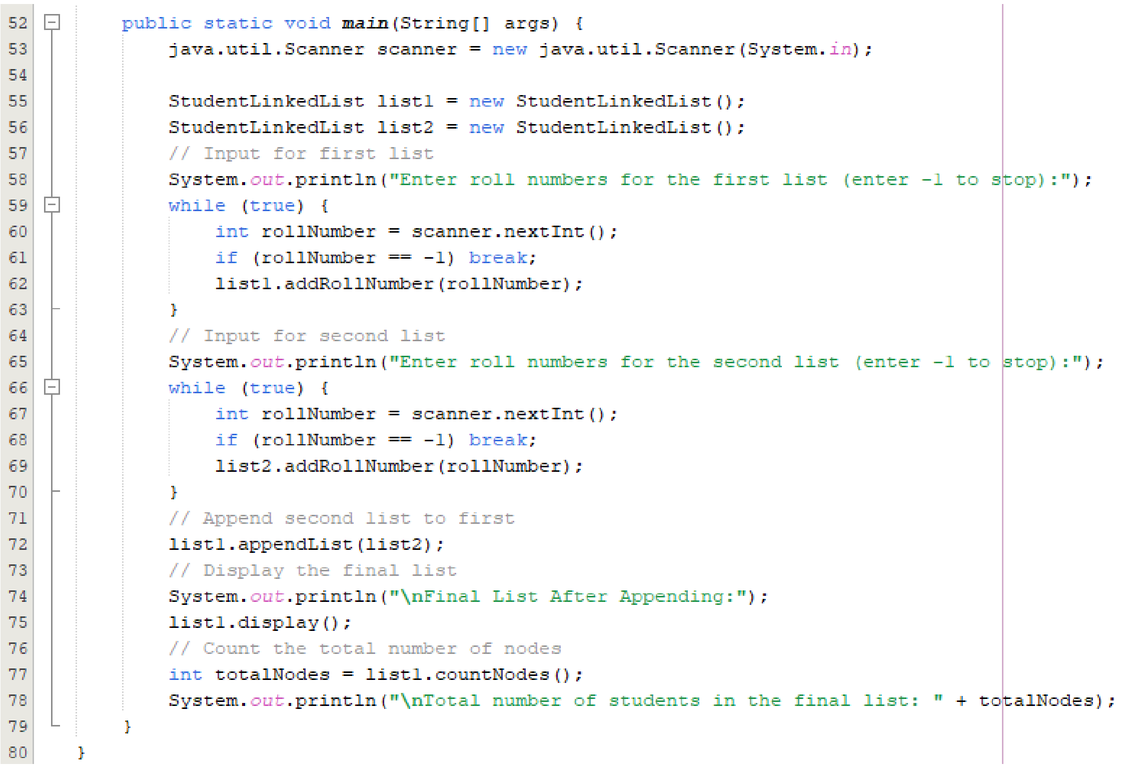
```
class StudentNode {
    int rollNumber;
    StudentNode next;

    public StudentNode(int rollNumber) {
        this.rollNumber = rollNumber;
        this.next = null;
    }
}
```



```
public class StudentLinkedList {
    private StudentNode head;
    // Add a roll number to the end of the list
    public void addRollNumber(int rollNumber) {
        StudentNode newNode = new StudentNode(rollNumber);
        if (head == null) {
            head = newNode;
            return;
        }
        StudentNode current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }
    // Append another list to this list
    public void appendList(StudentLinkedList otherList) {
        if (head == null) {
            head = otherList.head;
            return;
        }
        StudentNode current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = otherList.head;
    }
```

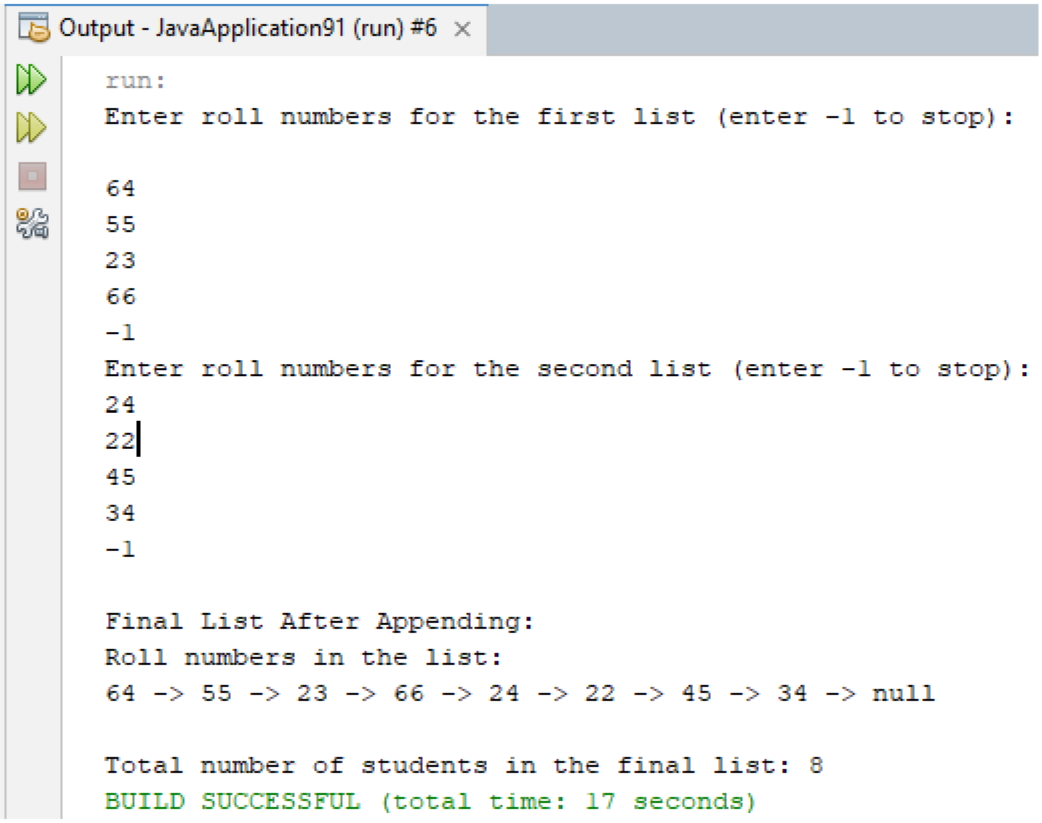


```java
    // Count the total number of nodes in the list
    public int countNodes() {
        int count = 0;
        StudentNode current = head;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }
    // Display the list
    public void display() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }
        StudentNode current = head;
        System.out.println("Roll numbers in the list:");
        while (current != null) {
            System.out.print(current.rollNumber + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }
```



```java
    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        StudentLinkedList list1 = new StudentLinkedList();
        StudentLinkedList list2 = new StudentLinkedList();
        // Input for first list
        System.out.println("Enter roll numbers for the first list (enter -1 to stop):");
        while (true) {
            int rollNumber = scanner.nextInt();
            if (rollNumber == -1) break;
            list1.addRollNumber(rollNumber);
        }
        // Input for second list
        System.out.println("Enter roll numbers for the second list (enter -1 to stop):");
        while (true) {
            int rollNumber = scanner.nextInt();
            if (rollNumber == -1) break;
            list2.addRollNumber(rollNumber);
        }
        // Append second list to first
        list1.appendList(list2);
        // Display the final list
        System.out.println("\nFinal List After Appending:");
        list1.display();
        // Count the total number of nodes
        int totalNodes = list1.countNodes();
        System.out.println("\nTotal number of students in the final list: " + totalNodes);
    }
}
```

OUTPUT:



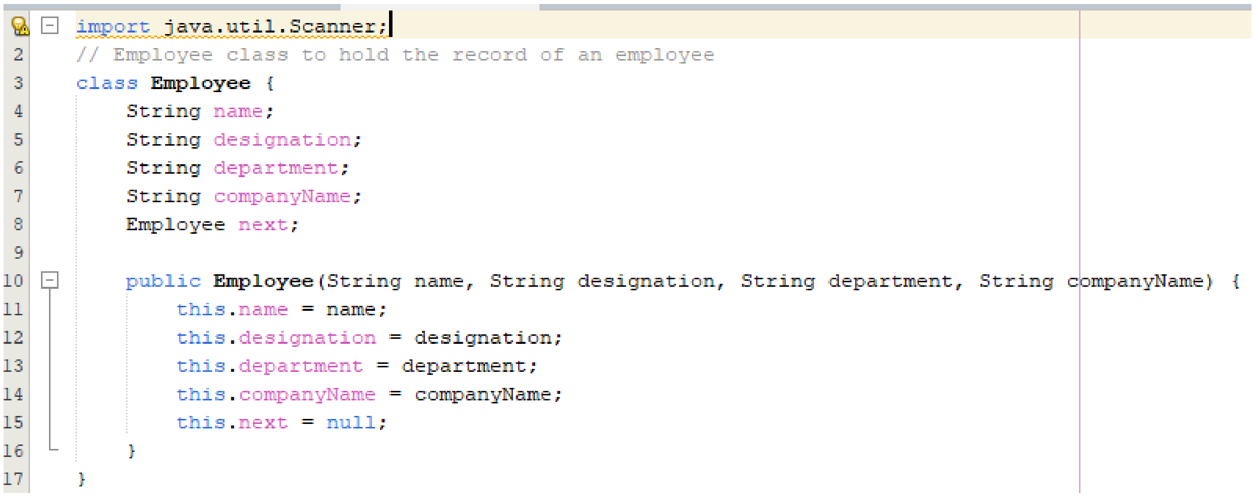
```
run:
Enter roll numbers for the first list (enter -1 to stop):

64
55
23
66
-1
Enter roll numbers for the second list (enter -1 to stop):
24
22
45
34
-1

Final List After Appending:
Roll numbers in the list:
64 -> 55 -> 23 -> 66 -> 24 -> 22 -> 45 -> 34 -> null

Total number of students in the final list: 8
BUILD SUCCESSFUL (total time: 17 seconds)
```

Doubly Linked List implementation of the list ADT

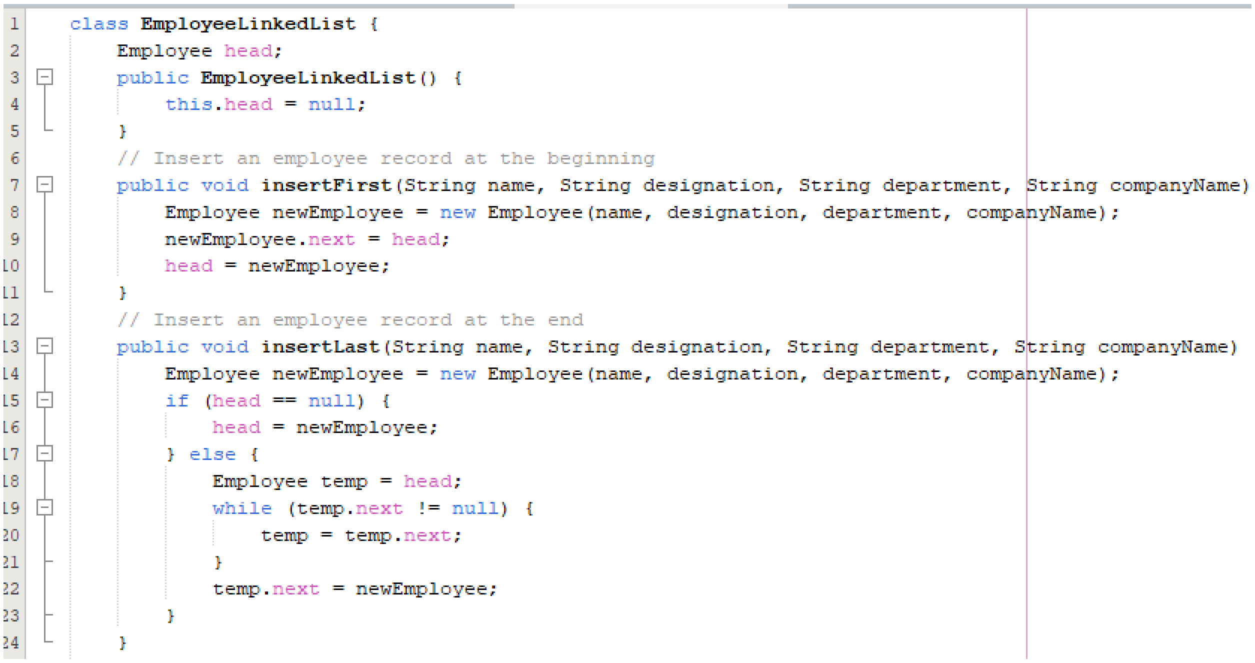
Lab Task

1. Write a program that can insert the records of employees in a link list. The record includes employee's name, designation, department and company name. The program should be able to insert the record as first, last and as middle node in the list and search any record.

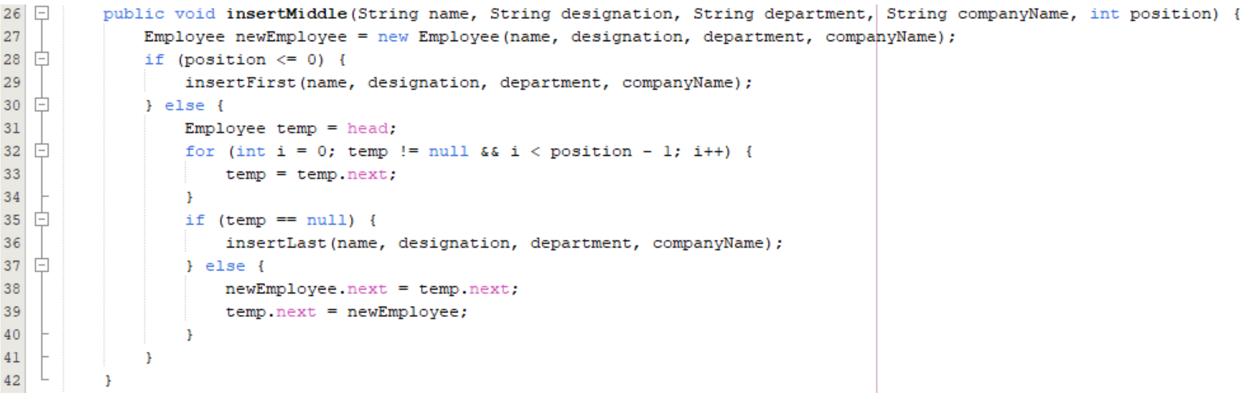
CODE



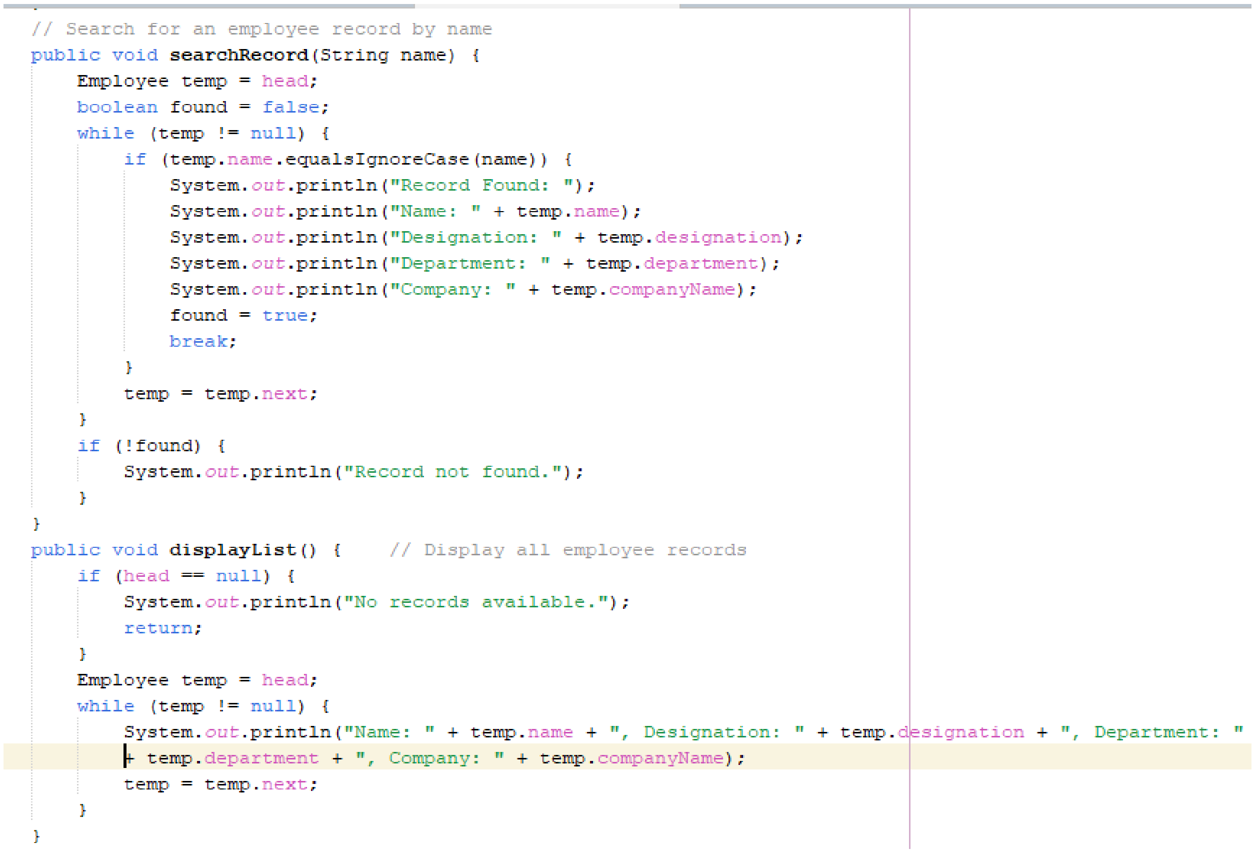
```java
import java.util.Scanner;
// Employee class to hold the record of an employee
class Employee {
    String name;
    String designation;
    String department;
    String companyName;
    Employee next;

    public Employee(String name, String designation, String department, String companyName) {
        this.name = name;
        this.designation = designation;
        this.department = department;
        this.companyName = companyName;
        this.next = null;
    }
}
```



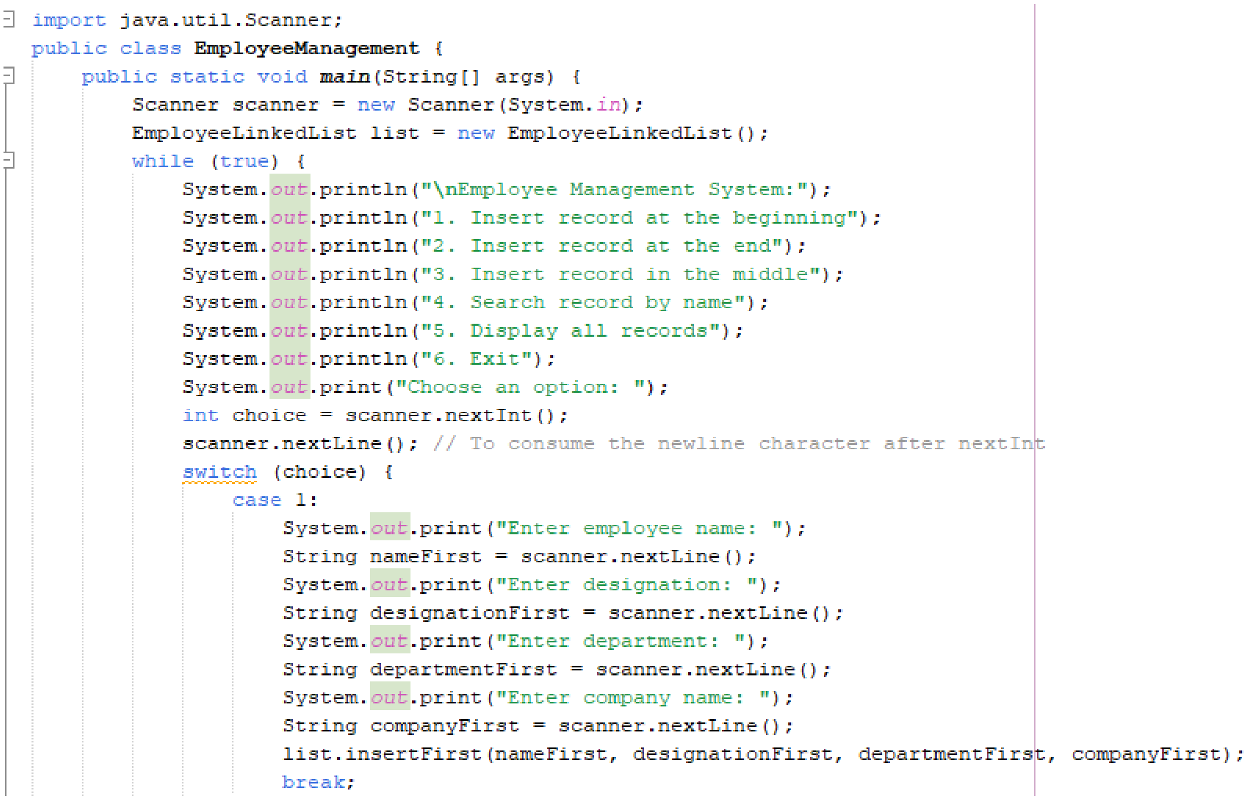
```java
class EmployeeLinkedList {
    Employee head;
    public EmployeeLinkedList() {
        this.head = null;
    }
    // Insert an employee record at the beginning
    public void insertFirst(String name, String designation, String department, String companyName)
        Employee newEmployee = new Employee(name, designation, department, companyName);
        newEmployee.next = head;
        head = newEmployee;
    }
    // Insert an employee record at the end
    public void insertLast(String name, String designation, String department, String companyName)
        Employee newEmployee = new Employee(name, designation, department, companyName);
        if (head == null) {
            head = newEmployee;
        } else {
            Employee temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newEmployee;
        }
    }
```



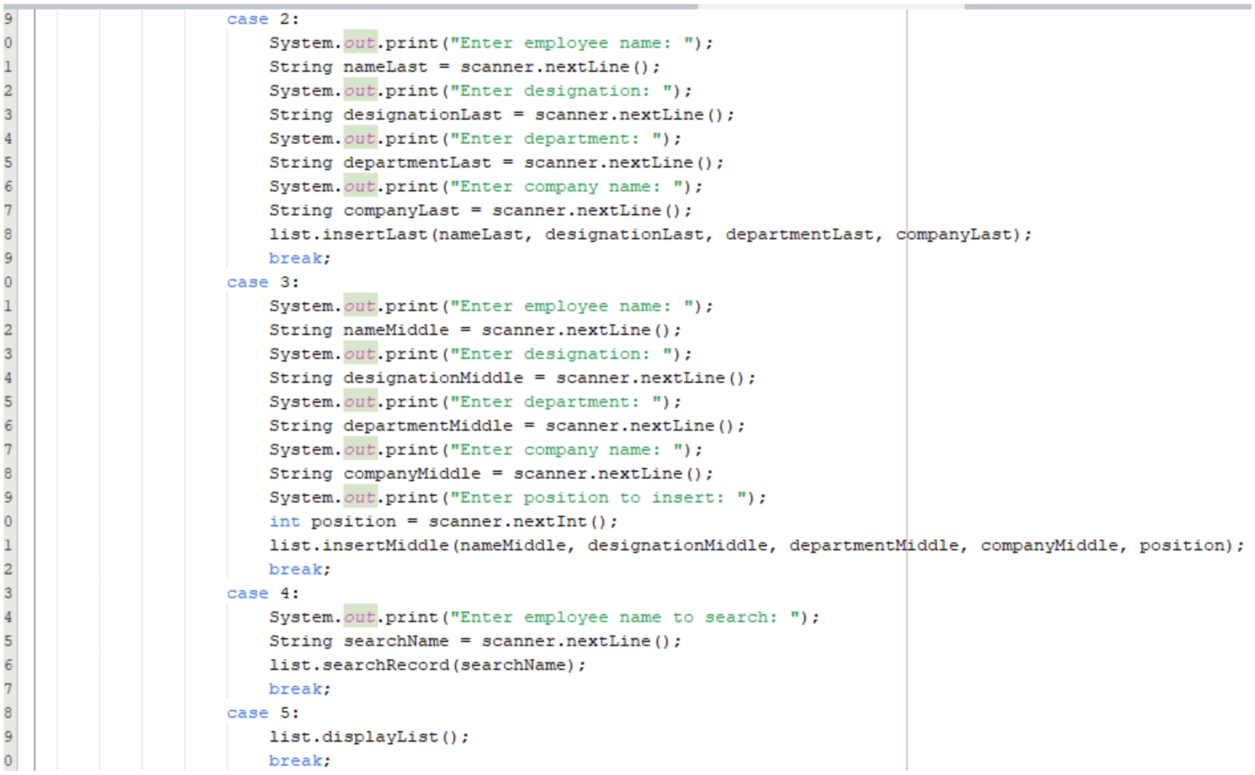
```java
    public void insertMiddle(String name, String designation, String department, String companyName, int position) {
        Employee newEmployee = new Employee(name, designation, department, companyName);
        if (position <= 0) {
            insertFirst(name, designation, department, companyName);
        } else {
            Employee temp = head;
            for (int i = 0; temp != null && i < position - 1; i++) {
                temp = temp.next;
            }
            if (temp == null) {
                insertLast(name, designation, department, companyName);
            } else {
                newEmployee.next = temp.next;
                temp.next = newEmployee;
            }
        }
    }
```



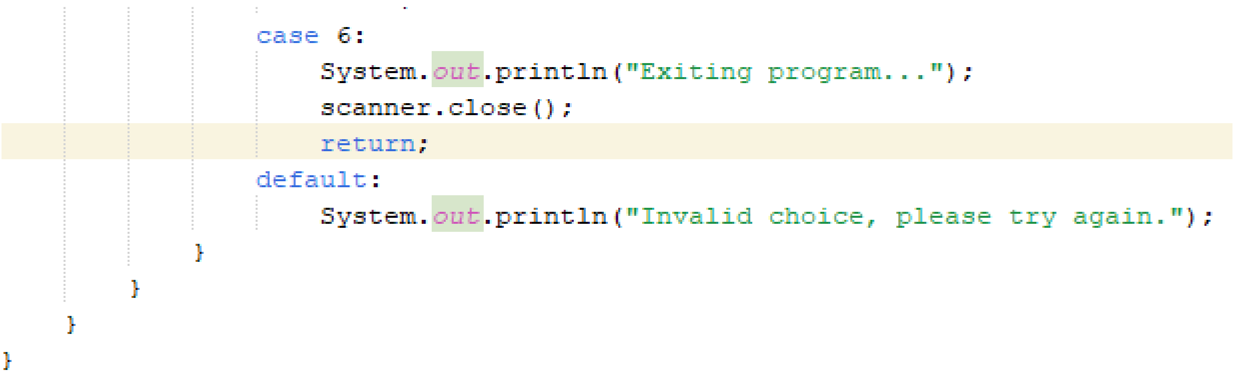
```
// Search for an employee record by name
public void searchRecord(String name) {
    Employee temp = head;
    boolean found = false;
    while (temp != null) {
        if (temp.name.equalsIgnoreCase(name)) {
            System.out.println("Record Found: ");
            System.out.println("Name: " + temp.name);
            System.out.println("Designation: " + temp.designation);
            System.out.println("Department: " + temp.department);
            System.out.println("Company: " + temp.companyName);
            found = true;
            break;
        }
        temp = temp.next;
    }
    if (!found) {
        System.out.println("Record not found.");
    }
}
public void displayList() {    // Display all employee records
    if (head == null) {
        System.out.println("No records available.");
        return;
    }
    Employee temp = head;
    while (temp != null) {
        System.out.println("Name: " + temp.name + ", Designation: " + temp.designation + ", Department: "
        + temp.department + ", Company: " + temp.companyName);
        temp = temp.next;
    }
}
```



```
import java.util.Scanner;
public class EmployeeManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        EmployeeLinkedList list = new EmployeeLinkedList();
        while (true) {
            System.out.println("\nEmployee Management System:");
            System.out.println("1. Insert record at the beginning");
            System.out.println("2. Insert record at the end");
            System.out.println("3. Insert record in the middle");
            System.out.println("4. Search record by name");
            System.out.println("5. Display all records");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // To consume the newline character after nextInt
            switch (choice) {
                case 1:
                    System.out.print("Enter employee name: ");
                    String nameFirst = scanner.nextLine();
                    System.out.print("Enter designation: ");
                    String designationFirst = scanner.nextLine();
                    System.out.print("Enter department: ");
                    String departmentFirst = scanner.nextLine();
                    System.out.print("Enter company name: ");
                    String companyFirst = scanner.nextLine();
                    list.insertFirst(nameFirst, designationFirst, departmentFirst, companyFirst);
                    break;
```

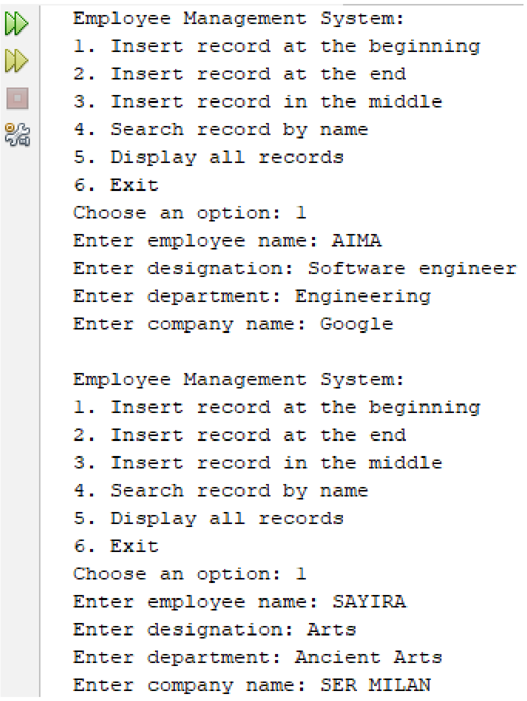


```
                case 2:
                    System.out.print("Enter employee name: ");
                    String nameLast = scanner.nextLine();
                    System.out.print("Enter designation: ");
                    String designationLast = scanner.nextLine();
                    System.out.print("Enter department: ");
                    String departmentLast = scanner.nextLine();
                    System.out.print("Enter company name: ");
                    String companyLast = scanner.nextLine();
                    list.insertLast(nameLast, designationLast, departmentLast, companyLast);
                    break;
                case 3:
                    System.out.print("Enter employee name: ");
                    String nameMiddle = scanner.nextLine();
                    System.out.print("Enter designation: ");
                    String designationMiddle = scanner.nextLine();
                    System.out.print("Enter department: ");
                    String departmentMiddle = scanner.nextLine();
                    System.out.print("Enter company name: ");
                    String companyMiddle = scanner.nextLine();
                    System.out.print("Enter position to insert: ");
                    int position = scanner.nextInt();
                    list.insertMiddle(nameMiddle, designationMiddle, departmentMiddle, companyMiddle, position);
                    break;
                case 4:
                    System.out.print("Enter employee name to search: ");
                    String searchName = scanner.nextLine();
                    list.searchRecord(searchName);
                    break;
                case 5:
                    list.displayList();
                    break;
```

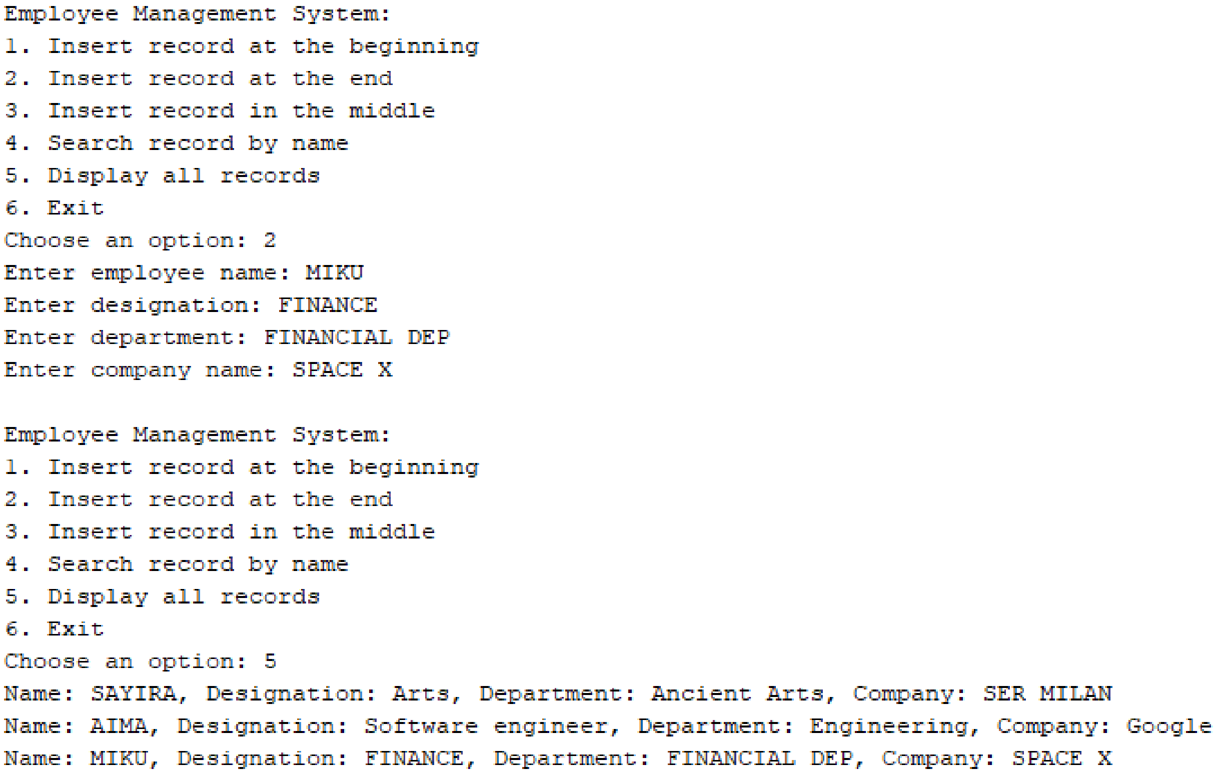


```
                case 6:
                    System.out.println("Exiting program...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice, please try again.");
            }
        }
    }
}
```

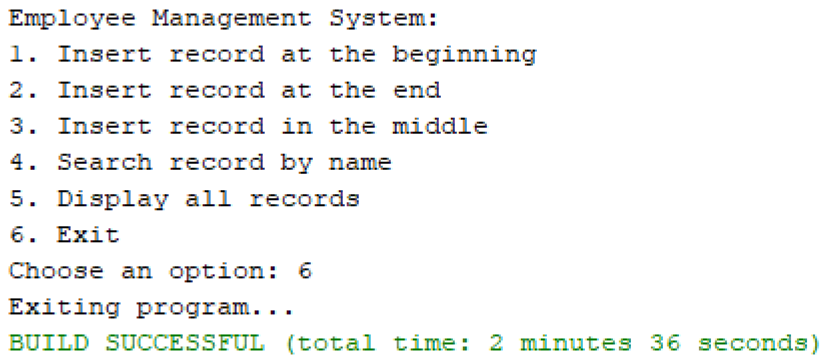
OUTPUT



```
Employee Management System:
1. Insert record at the beginning
2. Insert record at the end
3. Insert record in the middle
4. Search record by name
5. Display all records
6. Exit
Choose an option: 1
Enter employee name: AIMA
Enter designation: Software engineer
Enter department: Engineering
Enter company name: Google

Employee Management System:
1. Insert record at the beginning
2. Insert record at the end
3. Insert record in the middle
4. Search record by name
5. Display all records
6. Exit
Choose an option: 1
Enter employee name: SAYIRA
Enter designation: Arts
Enter department: Ancient Arts
Enter company name: SER MILAN
```



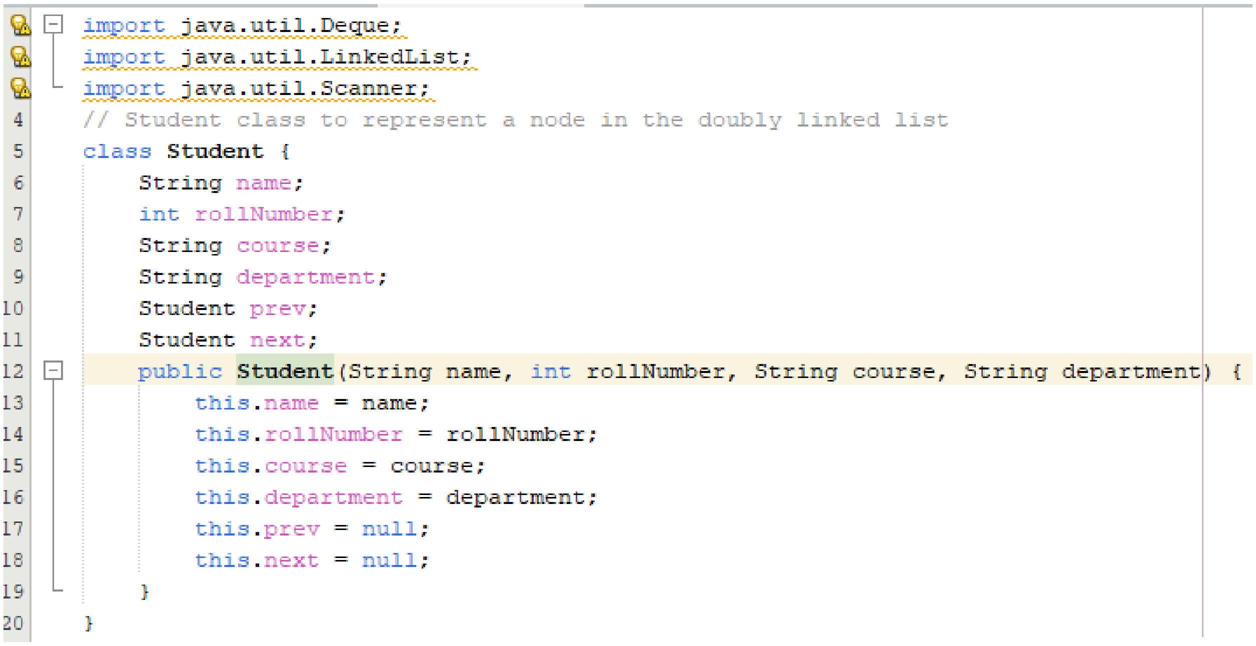
```
Employee Management System:
1. Insert record at the beginning
2. Insert record at the end
3. Insert record in the middle
4. Search record by name
5. Display all records
6. Exit
Choose an option: 2
Enter employee name: MIKU
Enter designation: FINANCE
Enter department: FINANCIAL DEP
Enter company name: SPACE X

Employee Management System:
1. Insert record at the beginning
2. Insert record at the end
3. Insert record in the middle
4. Search record by name
5. Display all records
6. Exit
Choose an option: 5
Name: SAYIRA, Designation: Arts, Department: Ancient Arts, Company: SER MILAN
Name: AIMA, Designation: Software engineer, Department: Engineering, Company: Google
Name: MIKU, Designation: FINANCE, Department: FINANCIAL DEP, Company: SPACE X
```



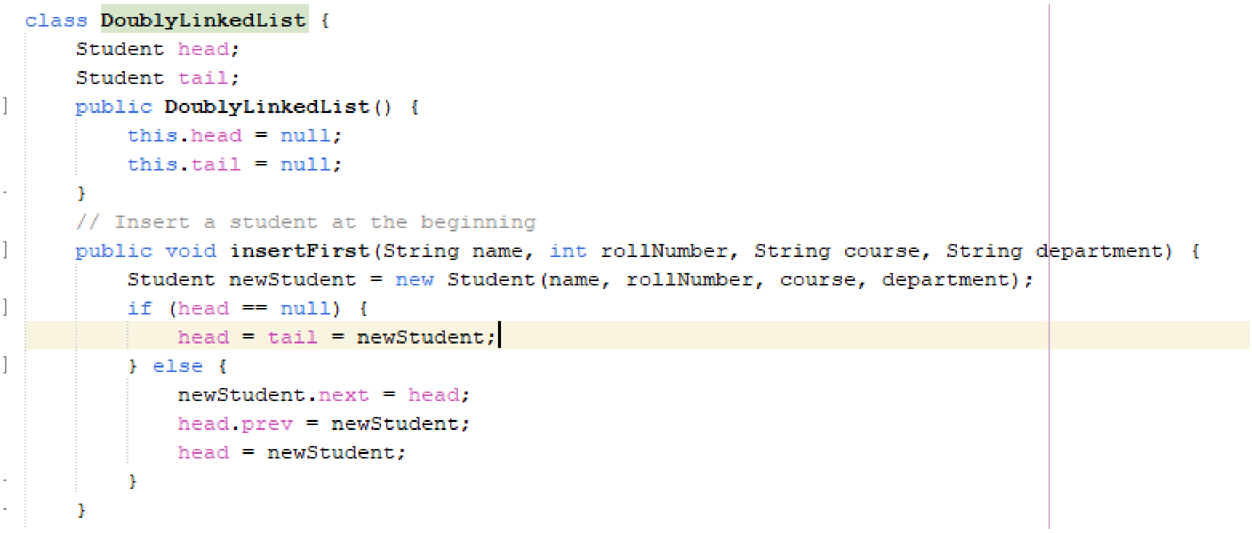
```
Employee Management System:
1. Insert record at the beginning
2. Insert record at the end
3. Insert record in the middle
4. Search record by name
5. Display all records
6. Exit
Choose an option: 6
Exiting program...
BUILD SUCCESSFUL (total time: 2 minutes 36 seconds)
```

2. Write a program to insert the records of students in a Doubly linked list and insert elements at first and last node using Deque.

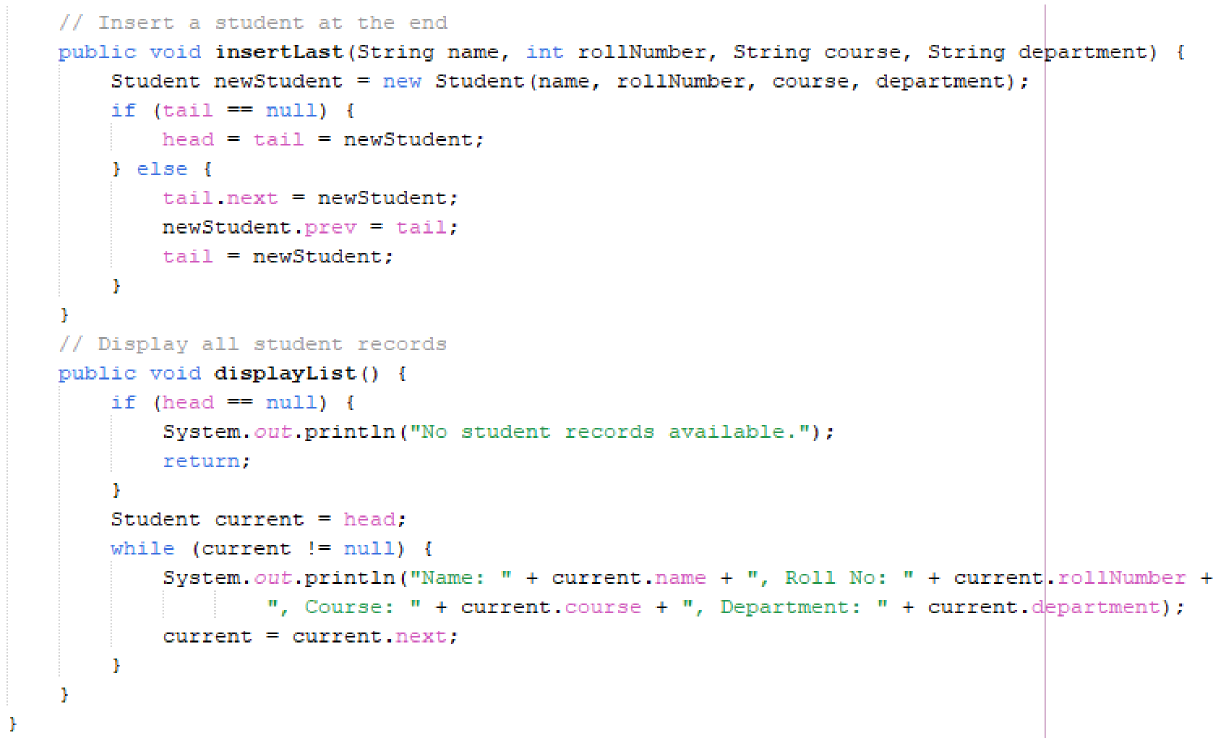
CODE:



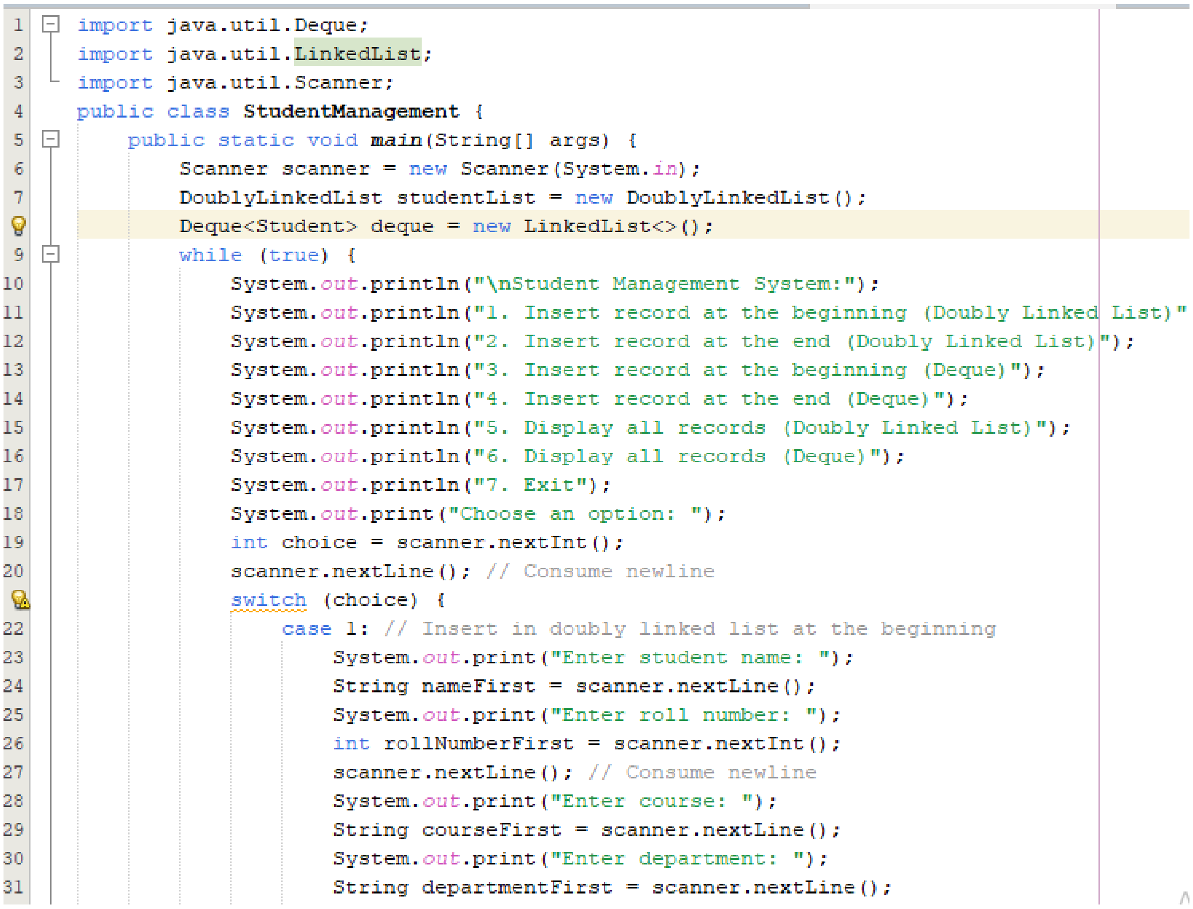
```java
import java.util.Deque;
import java.util.LinkedList;
import java.util.Scanner;
// Student class to represent a node in the doubly linked list
class Student {
    String name;
    int rollNumber;
    String course;
    String department;
    Student prev;
    Student next;
    public Student(String name, int rollNumber, String course, String department) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.course = course;
        this.department = department;
        this.prev = null;
        this.next = null;
    }
}
```



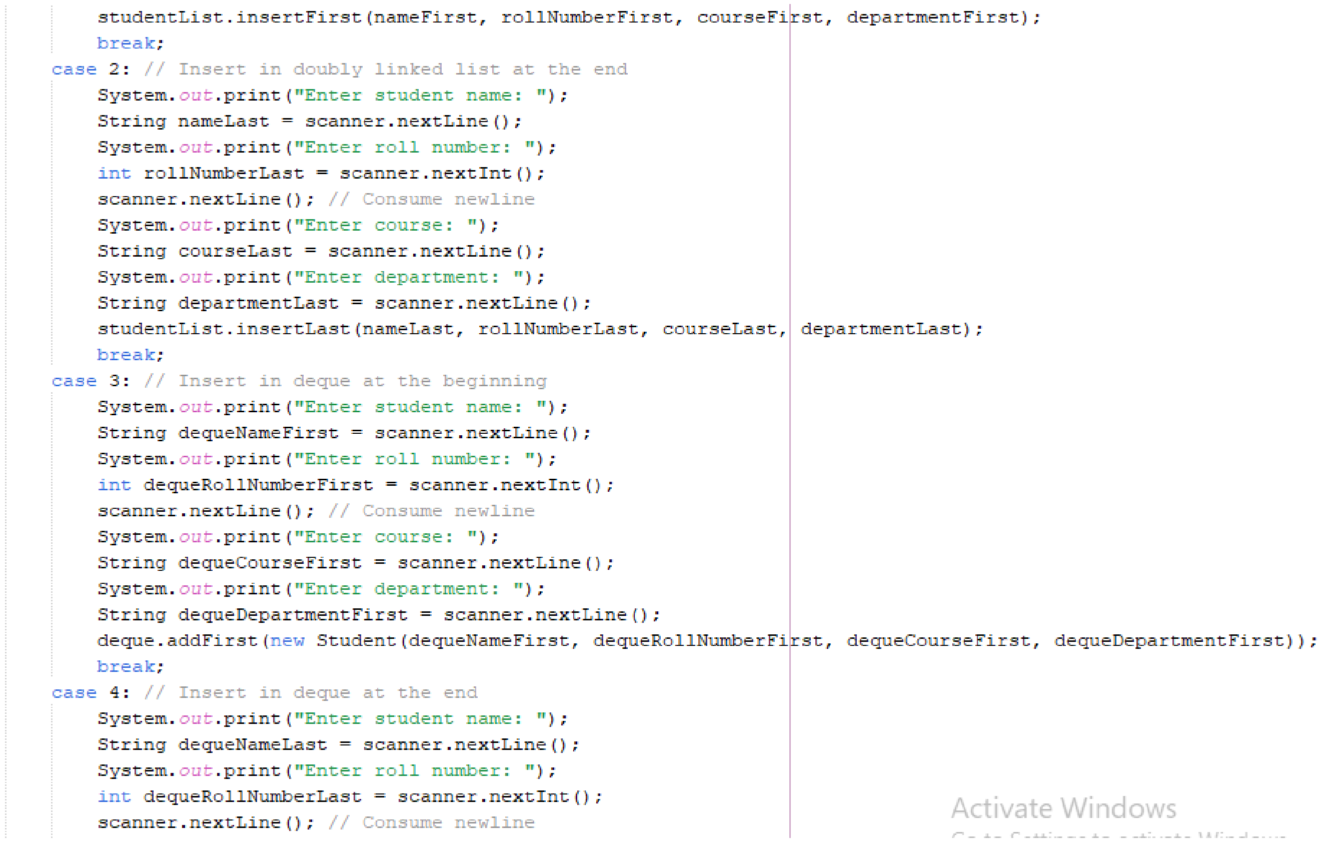
```java
class DoublyLinkedList {
    Student head;
    Student tail;
    public DoublyLinkedList() {
        this.head = null;
        this.tail = null;
    }
    // Insert a student at the beginning
    public void insertFirst(String name, int rollNumber, String course, String department) {
        Student newStudent = new Student(name, rollNumber, course, department);
        if (head == null) {
            head = tail = newStudent;
        } else {
            newStudent.next = head;
            head.prev = newStudent;
            head = newStudent;
        }
    }
```



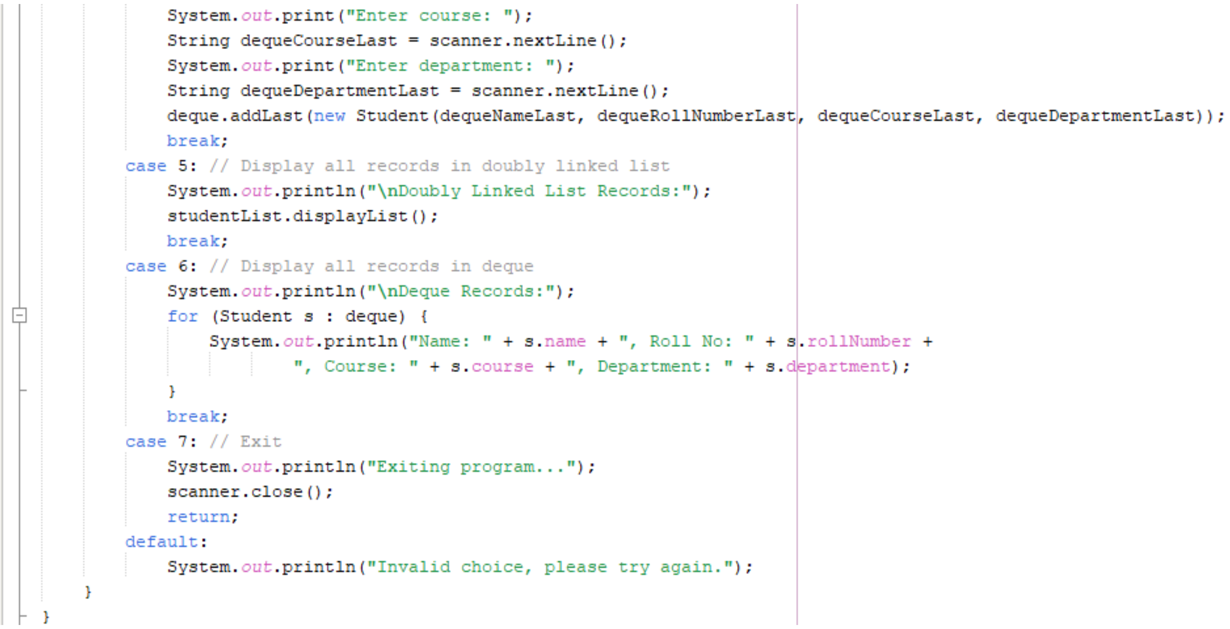
```
    // Insert a student at the end
    public void insertLast(String name, int rollNumber, String course, String department) {
        Student newStudent = new Student(name, rollNumber, course, department);
        if (tail == null) {
            head = tail = newStudent;
        } else {
            tail.next = newStudent;
            newStudent.prev = tail;
            tail = newStudent;
        }
    }
    // Display all student records
    public void displayList() {
        if (head == null) {
            System.out.println("No student records available.");
            return;
        }
        Student current = head;
        while (current != null) {
            System.out.println("Name: " + current.name + ", Roll No: " + current.rollNumber +
                    ", Course: " + current.course + ", Department: " + current.department);
            current = current.next;
        }
    }
}
```



```
import java.util.Deque;
import java.util.LinkedList;
import java.util.Scanner;
public class StudentManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DoublyLinkedList studentList = new DoublyLinkedList();
        Deque<Student> deque = new LinkedList<>();
        while (true) {
            System.out.println("\nStudent Management System:");
            System.out.println("1. Insert record at the beginning (Doubly Linked List)"
            System.out.println("2. Insert record at the end (Doubly Linked List)");
            System.out.println("3. Insert record at the beginning (Deque)");
            System.out.println("4. Insert record at the end (Deque)");
            System.out.println("5. Display all records (Doubly Linked List)");
            System.out.println("6. Display all records (Deque)");
            System.out.println("7. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            switch (choice) {
                case 1: // Insert in doubly linked list at the beginning
                    System.out.print("Enter student name: ");
                    String nameFirst = scanner.nextLine();
                    System.out.print("Enter roll number: ");
                    int rollNumberFirst = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
                    System.out.print("Enter course: ");
                    String courseFirst = scanner.nextLine();
                    System.out.print("Enter department: ");
                    String departmentFirst = scanner.nextLine();
```

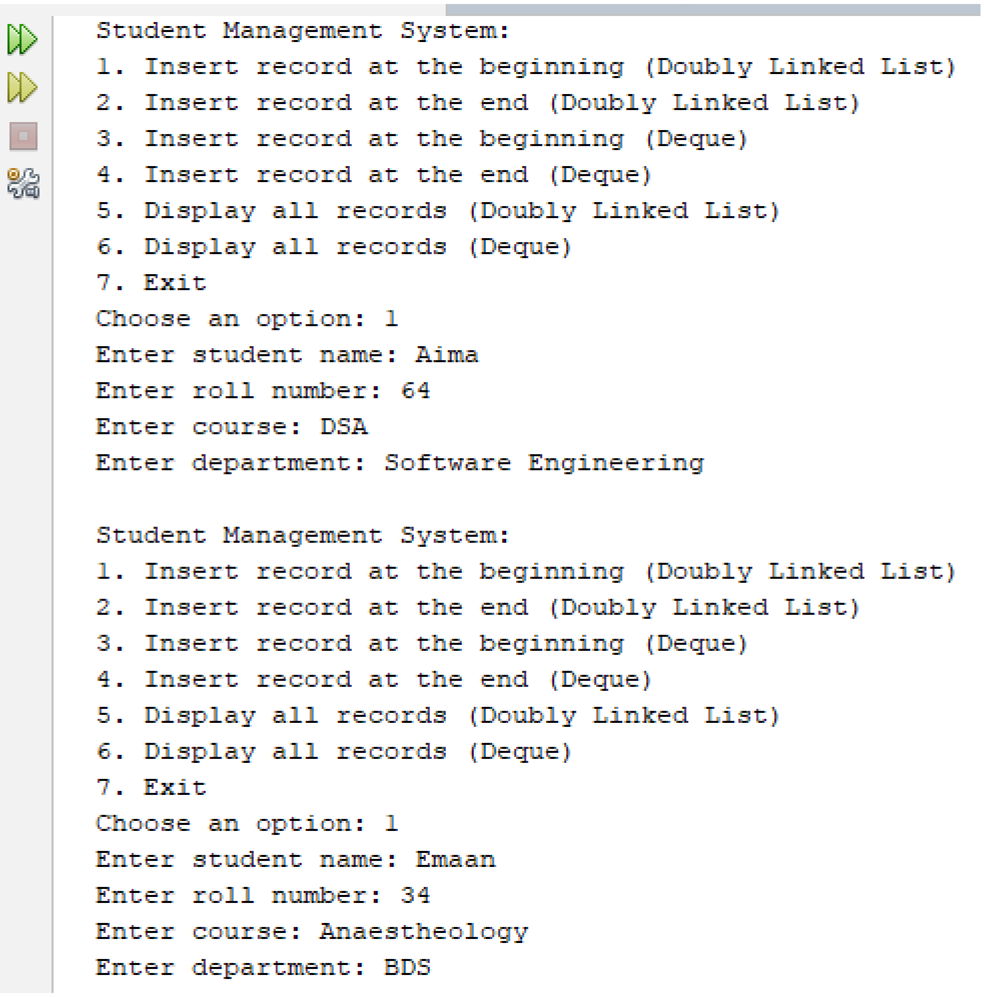


```
                studentList.insertFirst(nameFirst, rollNumberFirst, courseFirst, departmentFirst);
                break;
            case 2: // Insert in doubly linked list at the end
                System.out.print("Enter student name: ");
                String nameLast = scanner.nextLine();
                System.out.print("Enter roll number: ");
                int rollNumberLast = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                System.out.print("Enter course: ");
                String courseLast = scanner.nextLine();
                System.out.print("Enter department: ");
                String departmentLast = scanner.nextLine();
                studentList.insertLast(nameLast, rollNumberLast, courseLast, departmentLast);
                break;
            case 3: // Insert in deque at the beginning
                System.out.print("Enter student name: ");
                String dequeNameFirst = scanner.nextLine();
                System.out.print("Enter roll number: ");
                int dequeRollNumberFirst = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                System.out.print("Enter course: ");
                String dequeCourseFirst = scanner.nextLine();
                System.out.print("Enter department: ");
                String dequeDepartmentFirst = scanner.nextLine();
                deque.addFirst(new Student(dequeNameFirst, dequeRollNumberFirst, dequeCourseFirst, dequeDepartmentFirst));
                break;
            case 4: // Insert in deque at the end
                System.out.print("Enter student name: ");
                String dequeNameLast = scanner.nextLine();
                System.out.print("Enter roll number: ");
                int dequeRollNumberLast = scanner.nextInt();
                scanner.nextLine(); // Consume newline
```

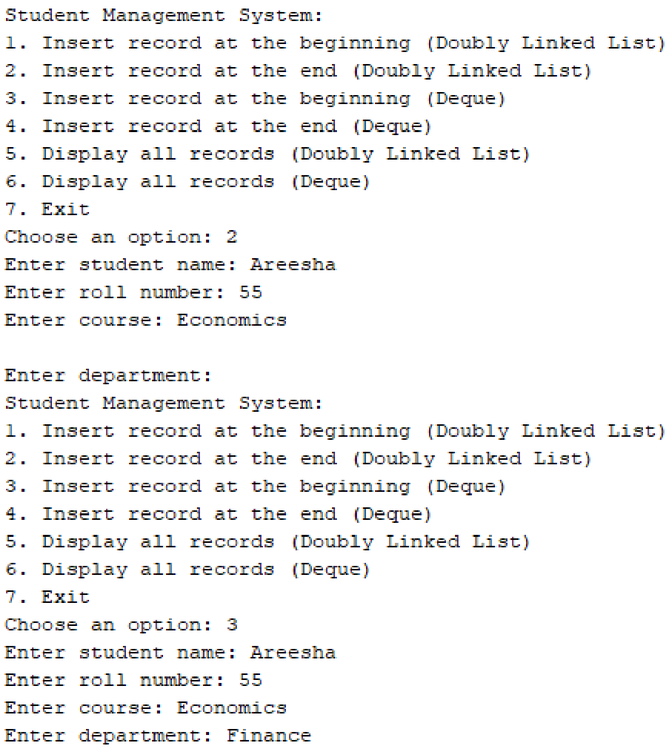


```
                System.out.print("Enter course: ");
                String dequeCourseLast = scanner.nextLine();
                System.out.print("Enter department: ");
                String dequeDepartmentLast = scanner.nextLine();
                deque.addLast(new Student(dequeNameLast, dequeRollNumberLast, dequeCourseLast, dequeDepartmentLast));
                break;
            case 5: // Display all records in doubly linked list
                System.out.println("\nDoubly Linked List Records:");
                studentList.displayList();
                break;
            case 6: // Display all records in deque
                System.out.println("\nDeque Records:");
                for (Student s : deque) {
                    System.out.println("Name: " + s.name + ", Roll No: " + s.rollNumber +
                            ", Course: " + s.course + ", Department: " + s.department);
                }
                break;
            case 7: // Exit
                System.out.println("Exiting program...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice, please try again.");
        }
    }
```

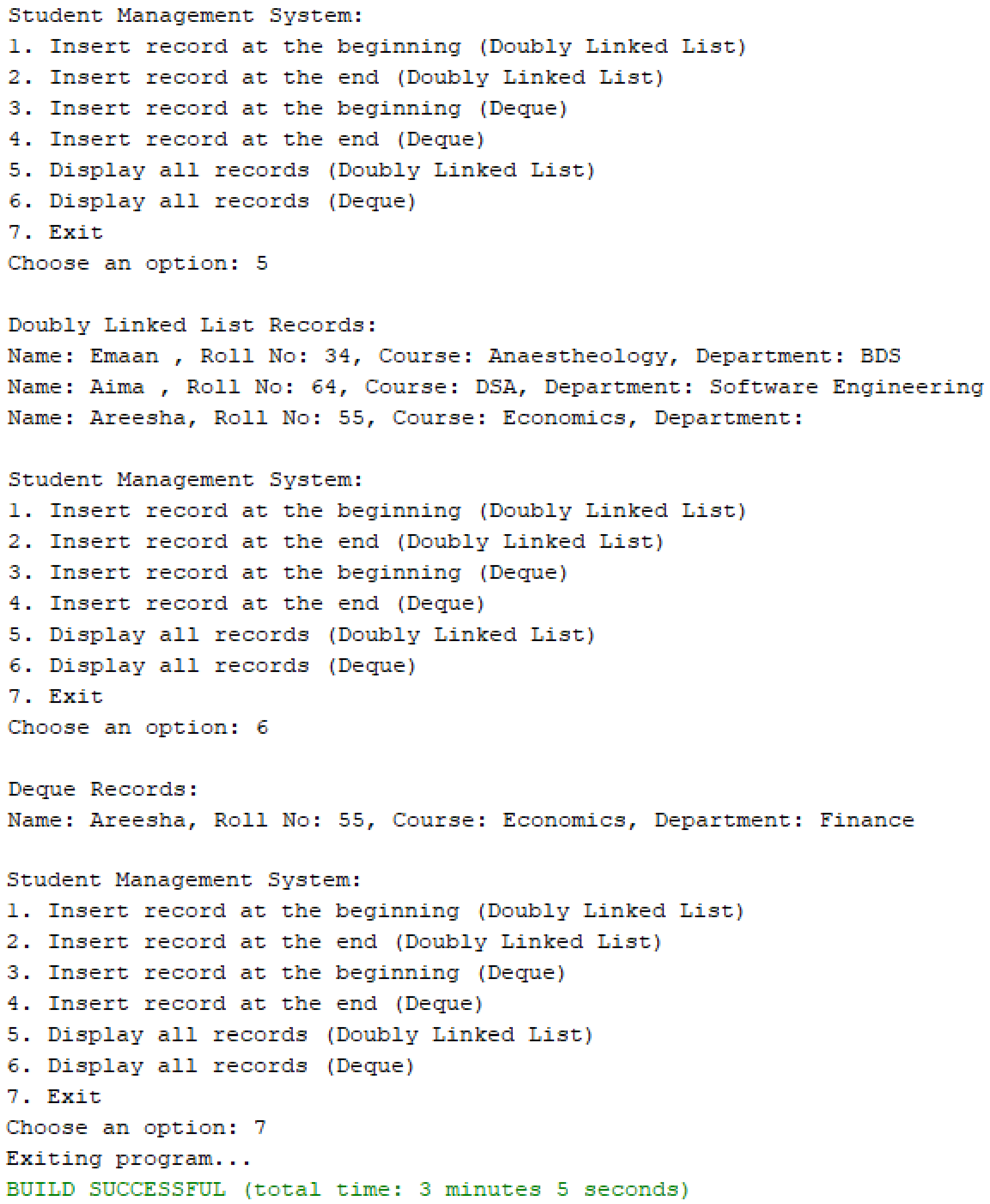
OUTPUT:



```
Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 1
Enter student name: Aima
Enter roll number: 64
Enter course: DSA
Enter department: Software Engineering

Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 1
Enter student name: Emaan
Enter roll number: 34
Enter course: Anaestheology
Enter department: BDS
```



```
Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 2
Enter student name: Areesha
Enter roll number: 55
Enter course: Economics

Enter department:
Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 3
Enter student name: Areesha
Enter roll number: 55
Enter course: Economics
Enter department: Finance
```



```
Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 5

Doubly Linked List Records:
Name: Emaan , Roll No: 34, Course: Anaestheology, Department: BDS
Name: Aima , Roll No: 64, Course: DSA, Department: Software Engineering
Name: Areesha, Roll No: 55, Course: Economics, Department:

Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 6

Deque Records:
Name: Areesha, Roll No: 55, Course: Economics, Department: Finance

Student Management System:
1. Insert record at the beginning (Doubly Linked List)
2. Insert record at the end (Doubly Linked List)
3. Insert record at the beginning (Deque)
4. Insert record at the end (Deque)
5. Display all records (Doubly Linked List)
6. Display all records (Deque)
7. Exit
Choose an option: 7
Exiting program...
BUILD SUCCESSFUL (total time: 3 minutes 5 seconds)
```

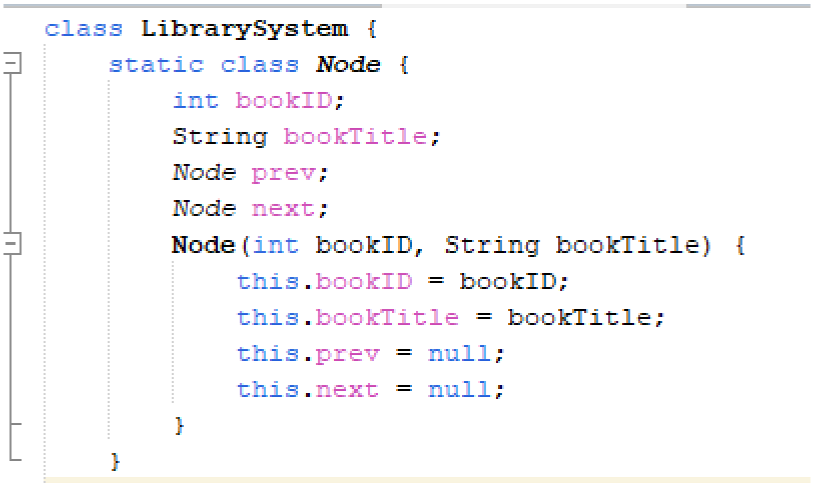
3. You are managing a **library system** where each book is represented by a node in a **doubly linked list**. Each node contains:

- **Book ID** (integer)
- **Book Title** (string)

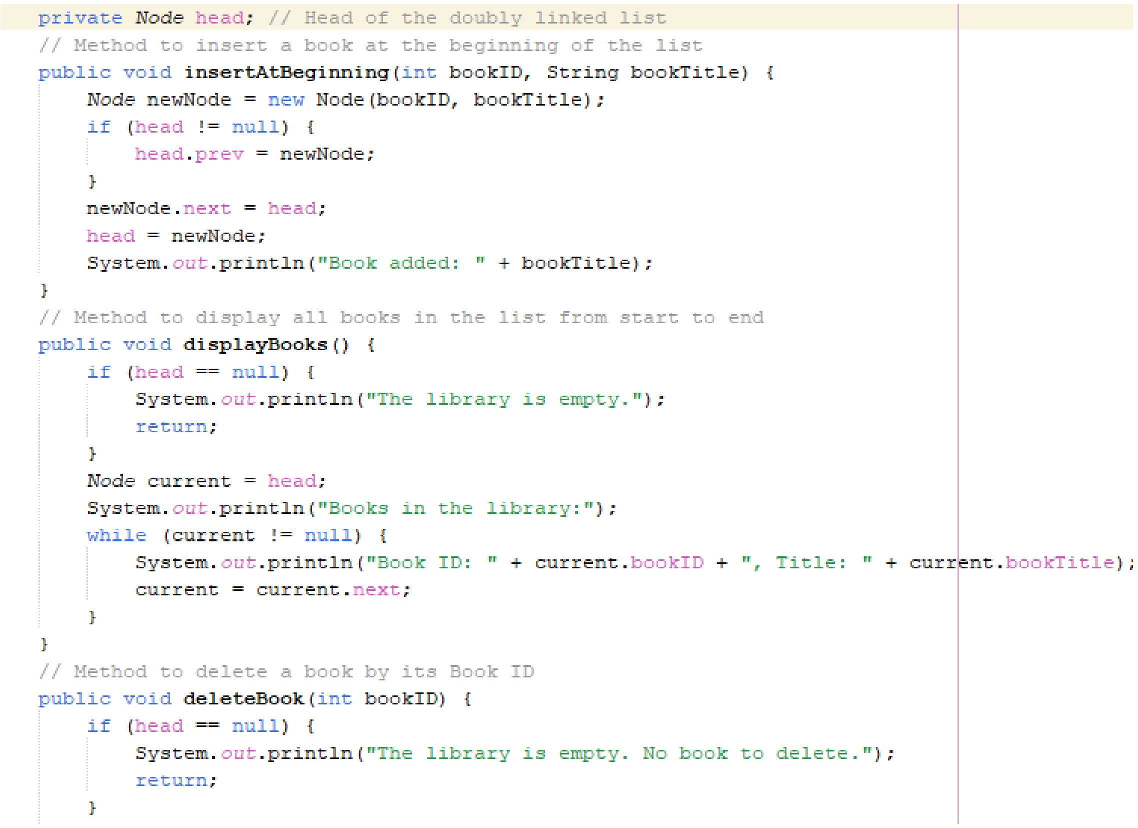
Your task is to:

a) **Insert** a book at the **beginning** of the list.
b) **Display** all books in the list from **start to end**.
c) **Delete** a book by its **Book ID**.
d) **Display** the list after deletion

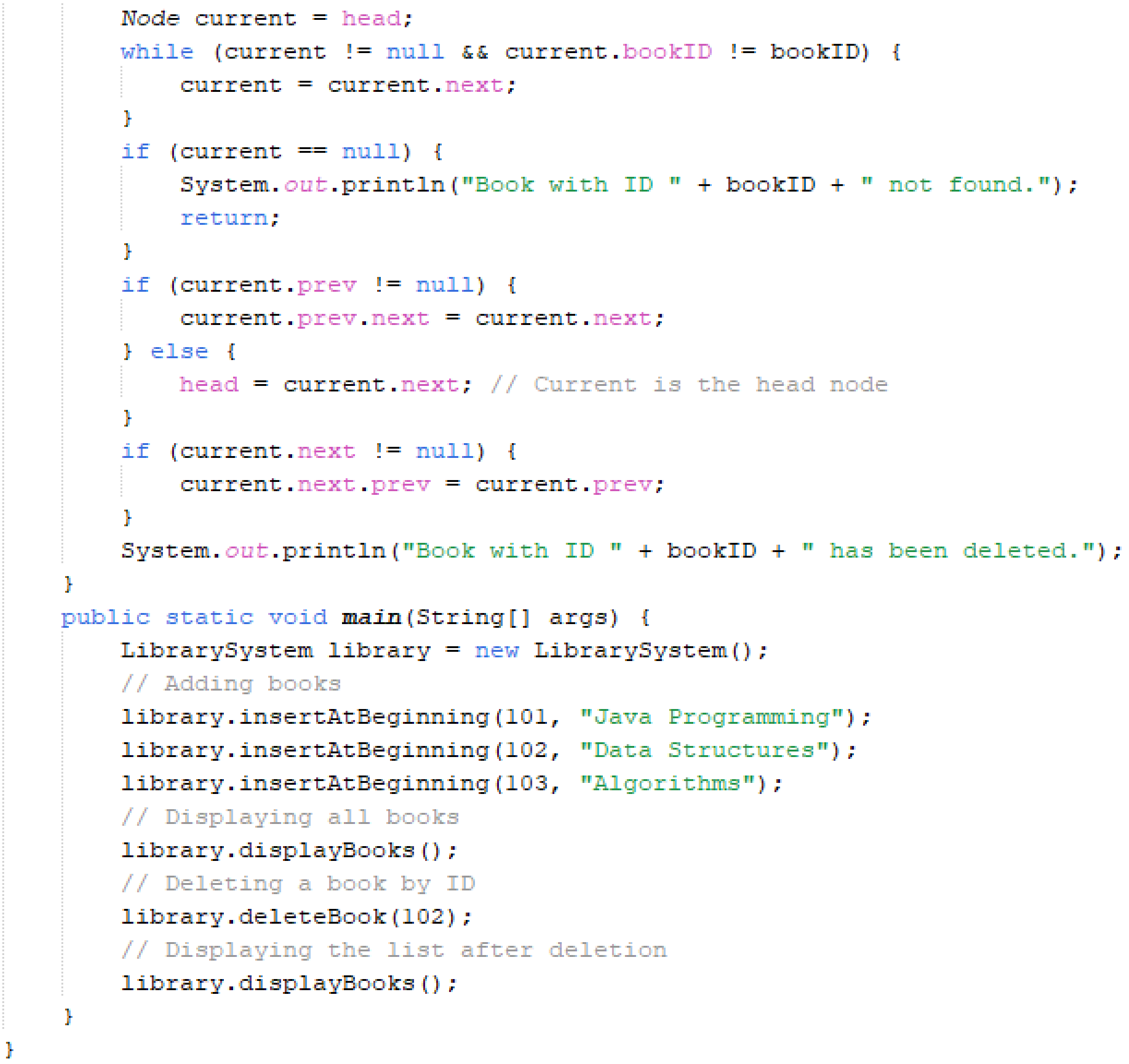
CODE:



```
class LibrarySystem {
    static class Node {
        int bookID;
        String bookTitle;
        Node prev;
        Node next;
        Node(int bookID, String bookTitle) {
            this.bookID = bookID;
            this.bookTitle = bookTitle;
            this.prev = null;
            this.next = null;
        }
    }
```

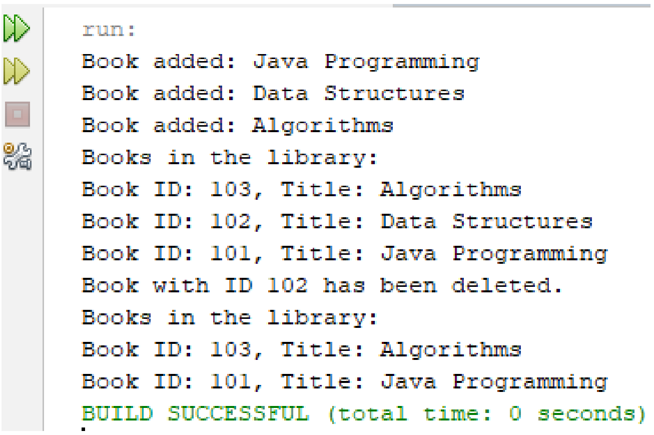


```
private Node head; // Head of the doubly linked list
// Method to insert a book at the beginning of the list
public void insertAtBeginning(int bookID, String bookTitle) {
    Node newNode = new Node(bookID, bookTitle);
    if (head != null) {
        head.prev = newNode;
    }
    newNode.next = head;
    head = newNode;
    System.out.println("Book added: " + bookTitle);
}
// Method to display all books in the list from start to end
public void displayBooks() {
    if (head == null) {
        System.out.println("The library is empty.");
        return;
    }
    Node current = head;
    System.out.println("Books in the library:");
    while (current != null) {
        System.out.println("Book ID: " + current.bookID + ", Title: " + current.bookTitle);
        current = current.next;
    }
}
// Method to delete a book by its Book ID
public void deleteBook(int bookID) {
    if (head == null) {
        System.out.println("The library is empty. No book to delete.");
        return;
    }
```



```
        Node current = head;
        while (current != null && current.bookID != bookID) {
            current = current.next;
        }
        if (current == null) {
            System.out.println("Book with ID " + bookID + " not found.");
            return;
        }
        if (current.prev != null) {
            current.prev.next = current.next;
        } else {
            head = current.next; // Current is the head node
        }
        if (current.next != null) {
            current.next.prev = current.prev;
        }
        System.out.println("Book with ID " + bookID + " has been deleted.");
    }
    public static void main(String[] args) {
        LibrarySystem library = new LibrarySystem();
        // Adding books
        library.insertAtBeginning(101, "Java Programming");
        library.insertAtBeginning(102, "Data Structures");
        library.insertAtBeginning(103, "Algorithms");
        // Displaying all books
        library.displayBooks();
        // Deleting a book by ID
        library.deleteBook(102);
        // Displaying the list after deletion
        library.displayBooks();
    }
}
```

OUTPUT

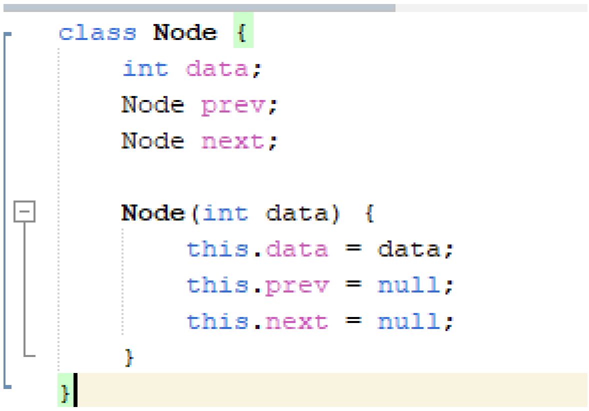


```
run:
Book added: Java Programming
Book added: Data Structures
Book added: Algorithms
Books in the library:
Book ID: 103, Title: Algorithms
Book ID: 102, Title: Data Structures
Book ID: 101, Title: Java Programming
Book with ID 102 has been deleted.
Books in the library:
Book ID: 103, Title: Algorithms
Book ID: 101, Title: Java Programming
BUILD SUCCESSFUL (total time: 0 seconds)
```

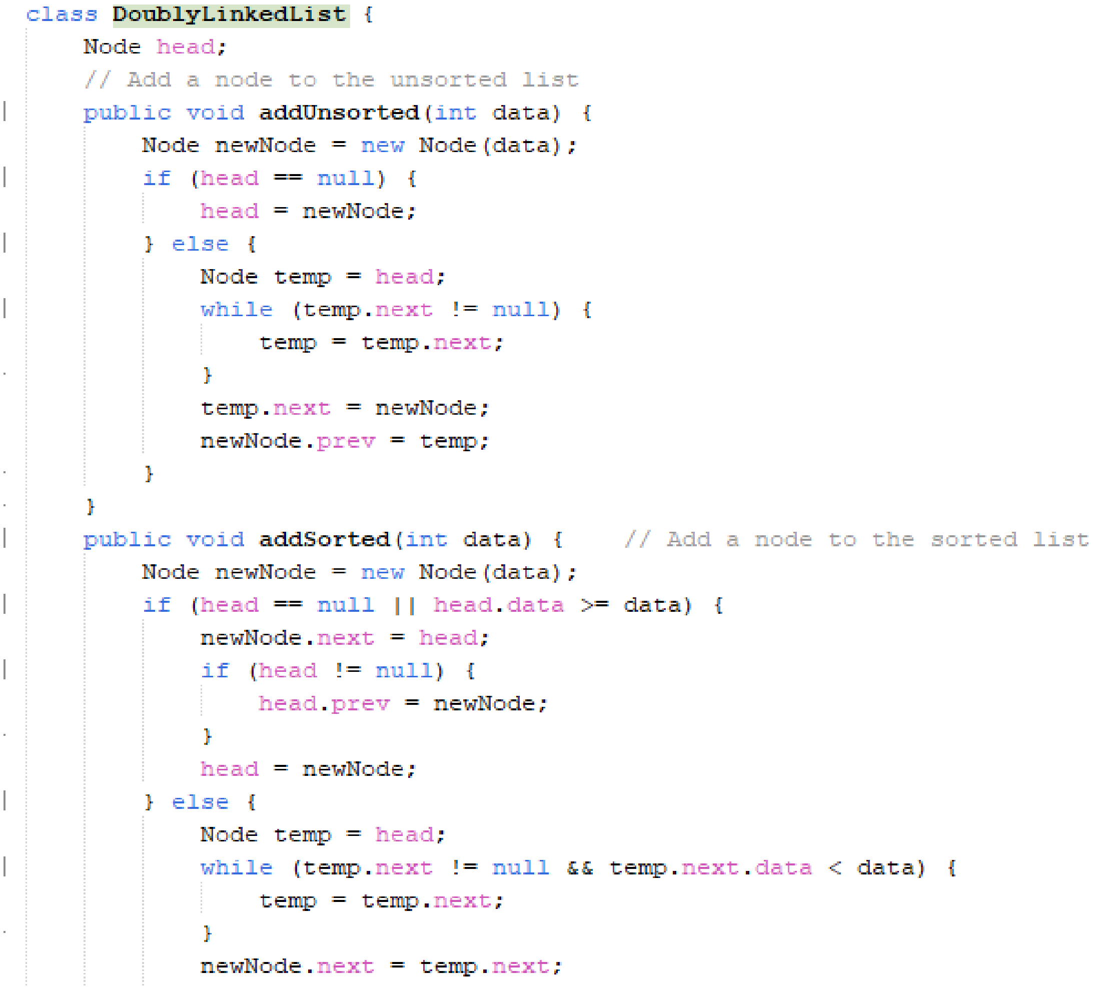
Home Task

1. Write a program to create Unsorted LinkedList as well as Sorted LinkedList.

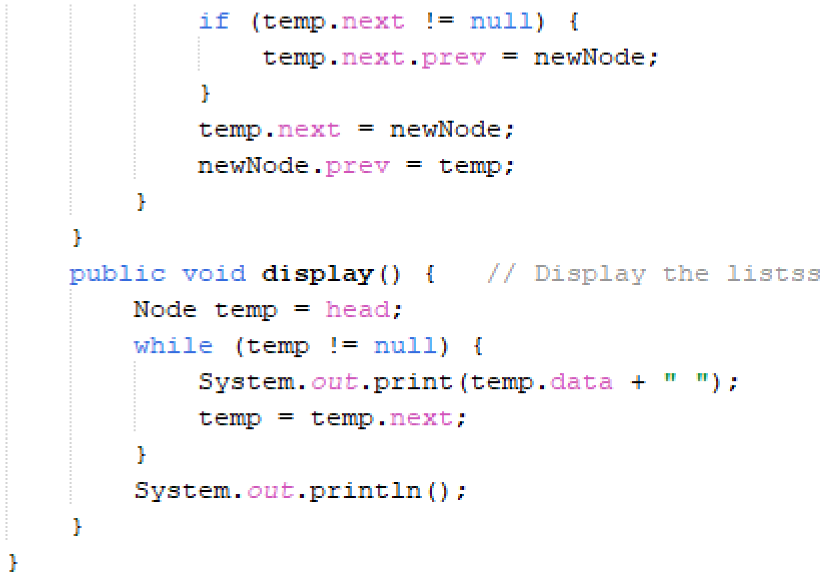
CODE:



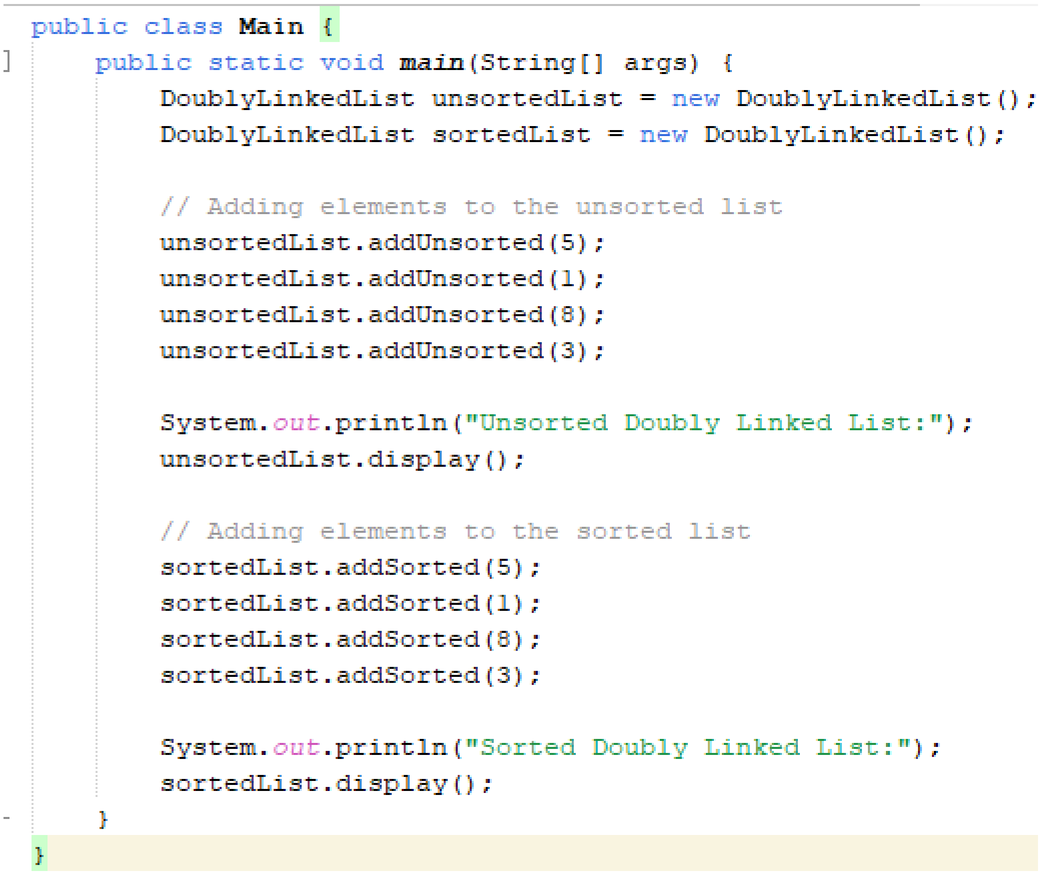
```
class Node {
    int data;
    Node prev;
    Node next;

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



```
class DoublyLinkedList {
    Node head;
    // Add a node to the unsorted list
    public void addUnsorted(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.prev = temp;
        }
    }
    public void addSorted(int data) {    // Add a node to the sorted list
        Node newNode = new Node(data);
        if (head == null || head.data >= data) {
            newNode.next = head;
            if (head != null) {
                head.prev = newNode;
            }
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null && temp.next.data < data) {
                temp = temp.next;
            }
            newNode.next = temp.next;
```

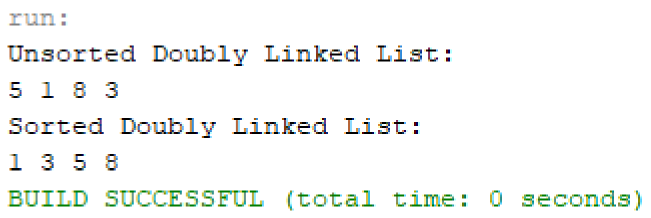


```
            if (temp.next != null) {
                temp.next.prev = newNode;
            }
            temp.next = newNode;
            newNode.prev = temp;
        }
    }
    public void display() {   // Display the listss
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}
```



```
public class Main {
    public static void main(String[] args) {
        DoublyLinkedList unsortedList = new DoublyLinkedList();
        DoublyLinkedList sortedList = new DoublyLinkedList();

        // Adding elements to the unsorted list
        unsortedList.addUnsorted(5);
        unsortedList.addUnsorted(1);
        unsortedList.addUnsorted(8);
        unsortedList.addUnsorted(3);

        System.out.println("Unsorted Doubly Linked List:");
        unsortedList.display();

        // Adding elements to the sorted list
        sortedList.addSorted(5);
        sortedList.addSorted(1);
        sortedList.addSorted(8);
        sortedList.addSorted(3);

        System.out.println("Sorted Doubly Linked List:");
        sortedList.display();
    }
}
```

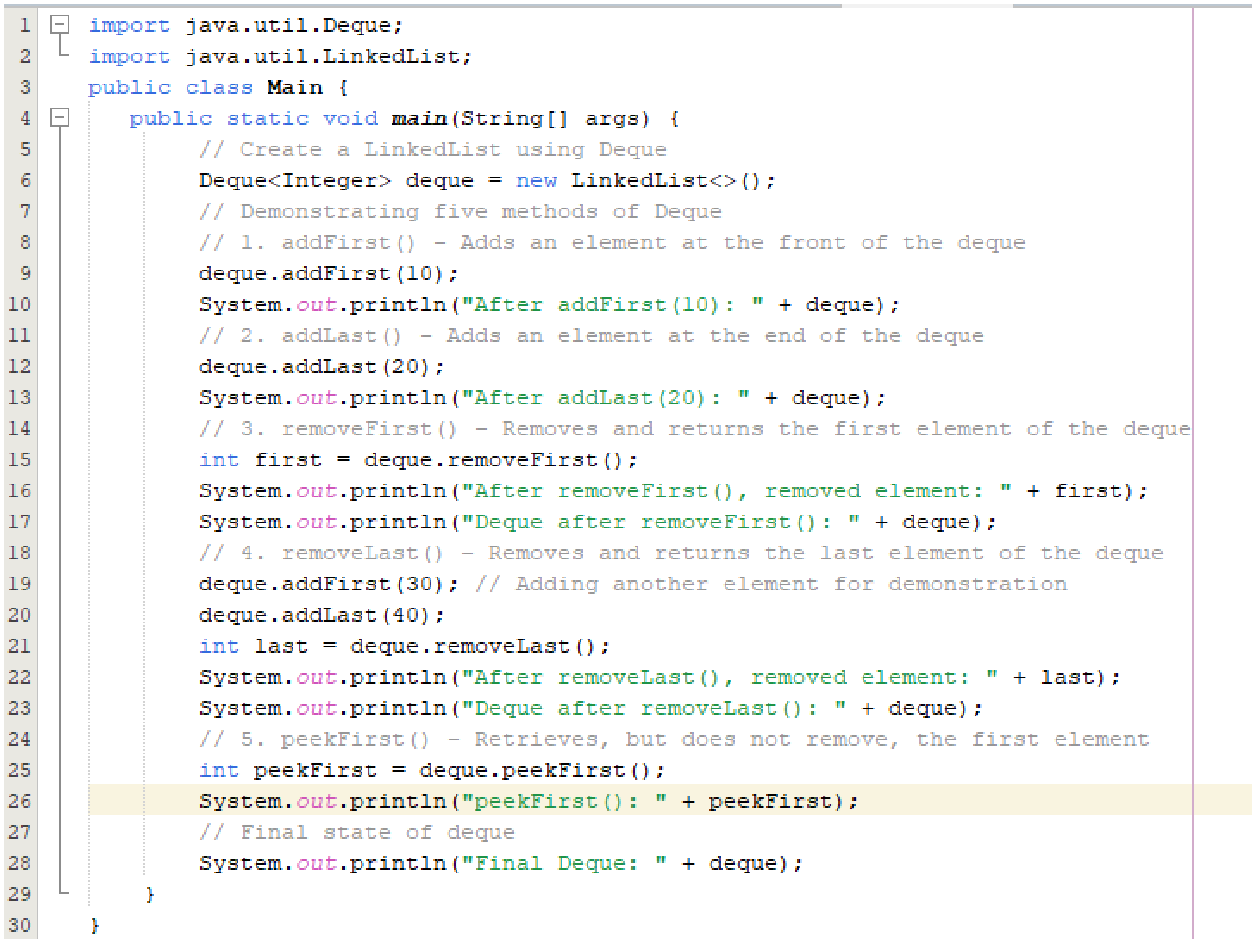
OUTPUT:



```
run:
Unsorted Doubly Linked List:
5 1 8 3
Sorted Doubly Linked List:
1 3 5 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

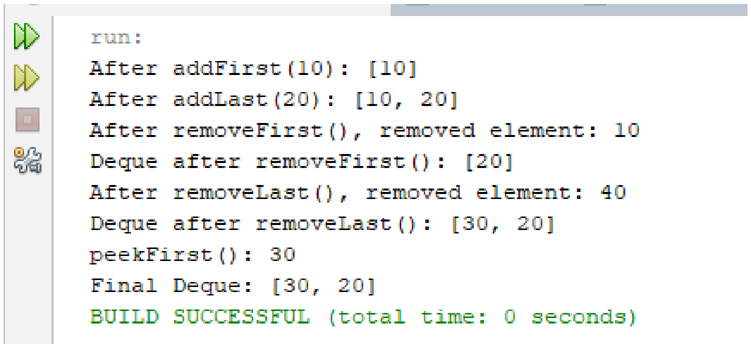
2. Write a program to create LinkedList using Deque and apply any five methods of Deque interface.

CODE



```java
import java.util.Deque;
import java.util.LinkedList;
public class Main {
    public static void main(String[] args) {
        // Create a LinkedList using Deque
        Deque<Integer> deque = new LinkedList<>();
        // Demonstrating five methods of Deque
        // 1. addFirst() - Adds an element at the front of the deque
        deque.addFirst(10);
        System.out.println("After addFirst(10): " + deque);
        // 2. addLast() - Adds an element at the end of the deque
        deque.addLast(20);
        System.out.println("After addLast(20): " + deque);
        // 3. removeFirst() - Removes and returns the first element of the deque
        int first = deque.removeFirst();
        System.out.println("After removeFirst(), removed element: " + first);
        System.out.println("Deque after removeFirst(): " + deque);
        // 4. removeLast() - Removes and returns the last element of the deque
        deque.addFirst(30); // Adding another element for demonstration
        deque.addLast(40);
        int last = deque.removeLast();
        System.out.println("After removeLast(), removed element: " + last);
        System.out.println("Deque after removeLast(): " + deque);
        // 5. peekFirst() - Retrieves, but does not remove, the first element
        int peekFirst = deque.peekFirst();
        System.out.println("peekFirst(): " + peekFirst);
        // Final state of deque
        System.out.println("Final Deque: " + deque);
    }
}
```

OUTPUT



```
run:
After addFirst(10): [10]
After addLast(20): [10, 20]
After removeFirst(), removed element: 10
Deque after removeFirst(): [20]
After removeLast(), removed element: 40
Deque after removeLast(): [30, 20]
peekFirst(): 30
Final Deque: [30, 20]
BUILD SUCCESSFUL (total time: 0 seconds)
```

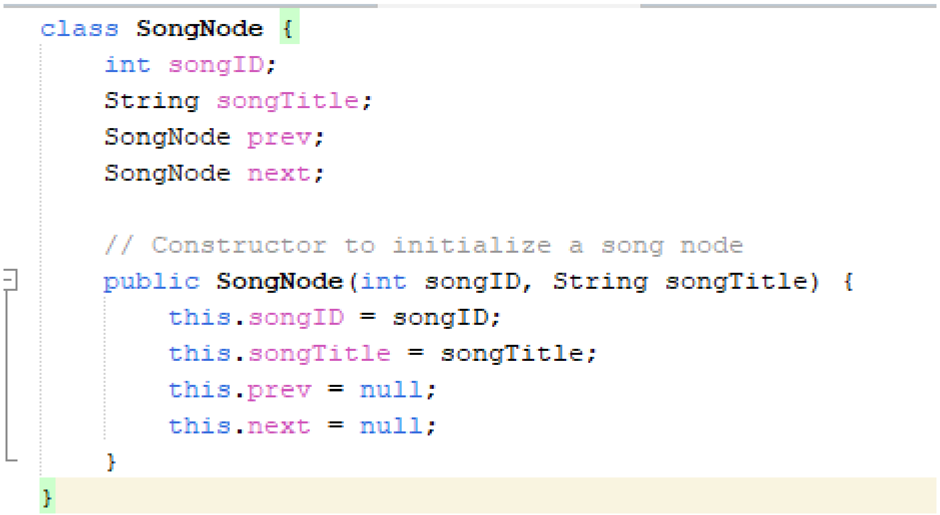
3. You are managing a **playlist system** where each song is represented by a node in a **doubly linked list**. Each node contains:

- **Song ID** (integer)
- **Song Title** (string)

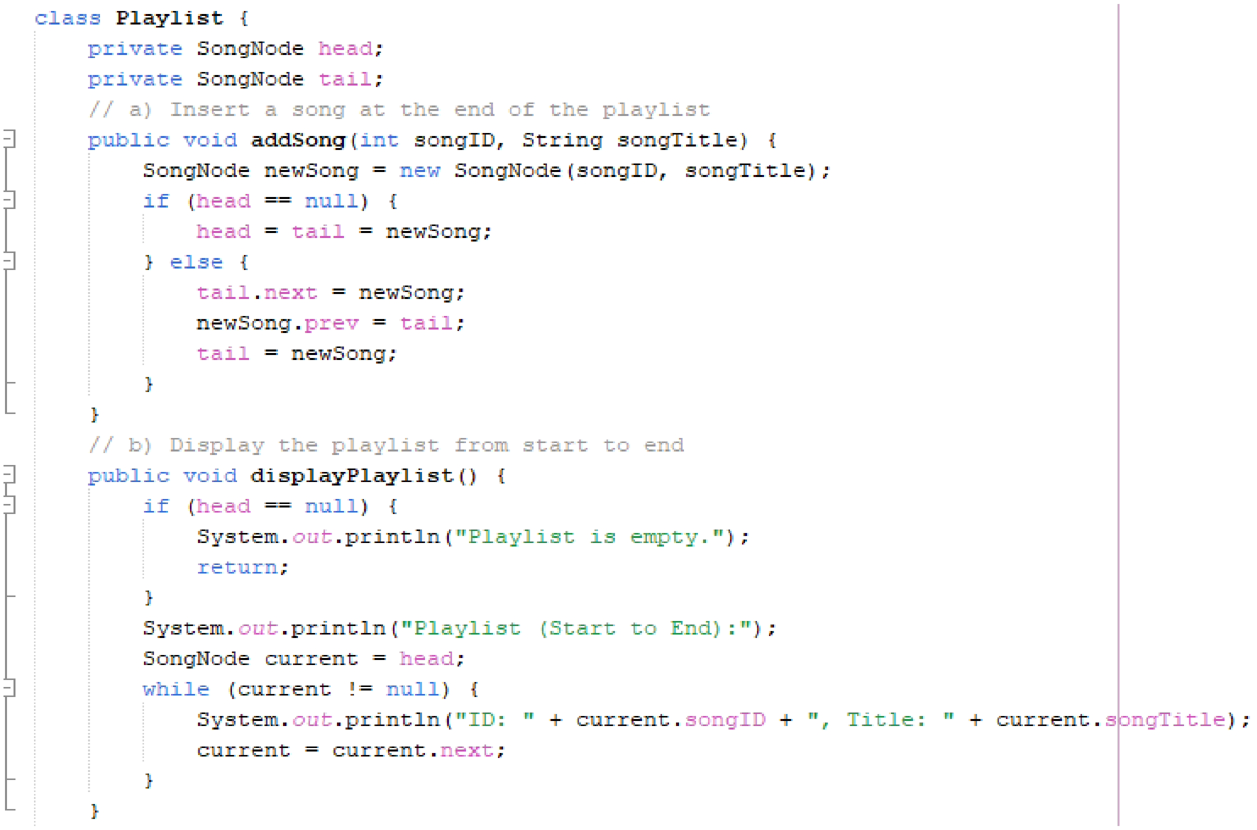
Your task is to:

a) **Insert** a song at the **end** of the playlist.
b) **Display** the playlist from **start to end**.
c) **Reverse** the playlist and display it again.

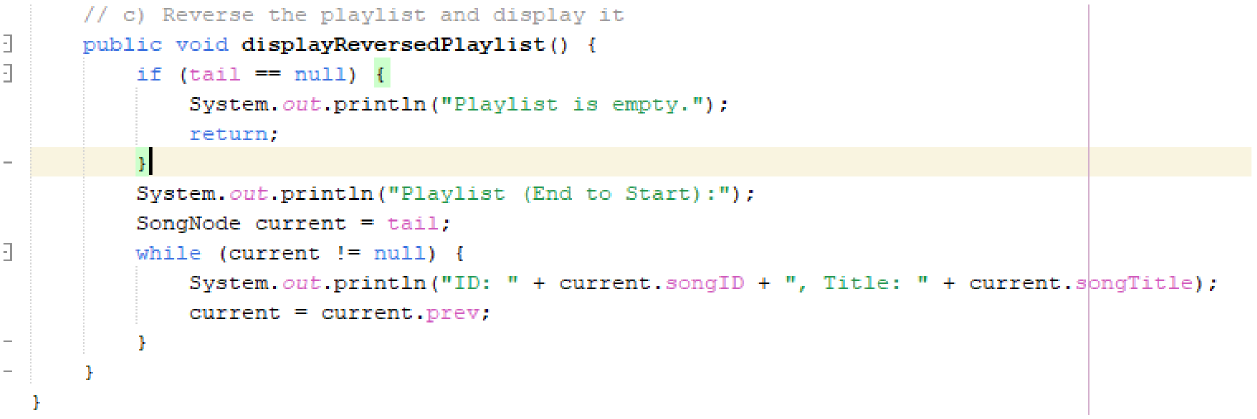
CODE



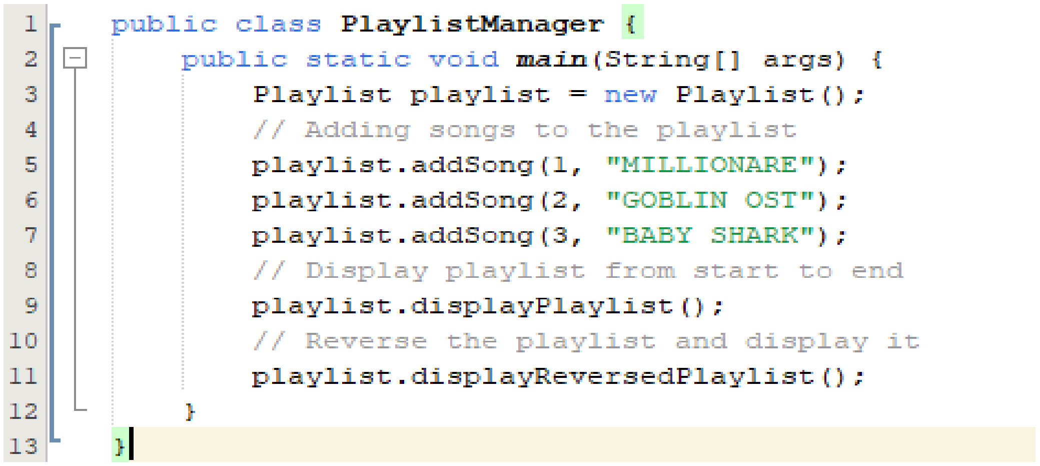
```java
class SongNode {
    int songID;
    String songTitle;
    SongNode prev;
    SongNode next;

    // Constructor to initialize a song node
    public SongNode(int songID, String songTitle) {
        this.songID = songID;
        this.songTitle = songTitle;
        this.prev = null;
        this.next = null;
    }
}
```



```java
class Playlist {
    private SongNode head;
    private SongNode tail;
    // a) Insert a song at the end of the playlist
    public void addSong(int songID, String songTitle) {
        SongNode newSong = new SongNode(songID, songTitle);
        if (head == null) {
            head = tail = newSong;
        } else {
            tail.next = newSong;
            newSong.prev = tail;
            tail = newSong;
        }
    }
    // b) Display the playlist from start to end
    public void displayPlaylist() {
        if (head == null) {
            System.out.println("Playlist is empty.");
            return;
        }
        System.out.println("Playlist (Start to End):");
        SongNode current = head;
        while (current != null) {
            System.out.println("ID: " + current.songID + ", Title: " + current.songTitle);
            current = current.next;
        }
    }
```

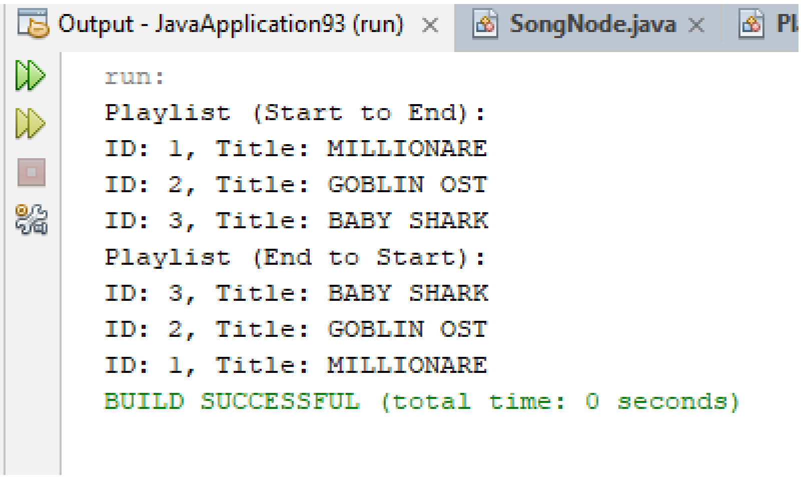


```java
    // c) Reverse the playlist and display it
    public void displayReversedPlaylist() {
        if (tail == null) {
            System.out.println("Playlist is empty.");
            return;
        }
        System.out.println("Playlist (End to Start):");
        SongNode current = tail;
        while (current != null) {
            System.out.println("ID: " + current.songID + ", Title: " + current.songTitle);
            current = current.prev;
        }
    }
}
```



```java
public class PlaylistManager {
    public static void main(String[] args) {
        Playlist playlist = new Playlist();
        // Adding songs to the playlist
        playlist.addSong(1, "MILLIONARE");
        playlist.addSong(2, "GOBLIN OST");
        playlist.addSong(3, "BABY SHARK");
        // Display playlist from start to end
        playlist.displayPlaylist();
        // Reverse the playlist and display it
        playlist.displayReversedPlaylist();
    }
}
```

OUTPUT



```
run:
Playlist (Start to End):
ID: 1, Title: MILLIONARE
ID: 2, Title: GOBLIN OST
ID: 3, Title: BABY SHARK
Playlist (End to Start):
ID: 3, Title: BABY SHARK
ID: 2, Title: GOBLIN OST
ID: 1, Title: MILLIONARE
BUILD SUCCESSFUL (total time: 0 seconds)
```