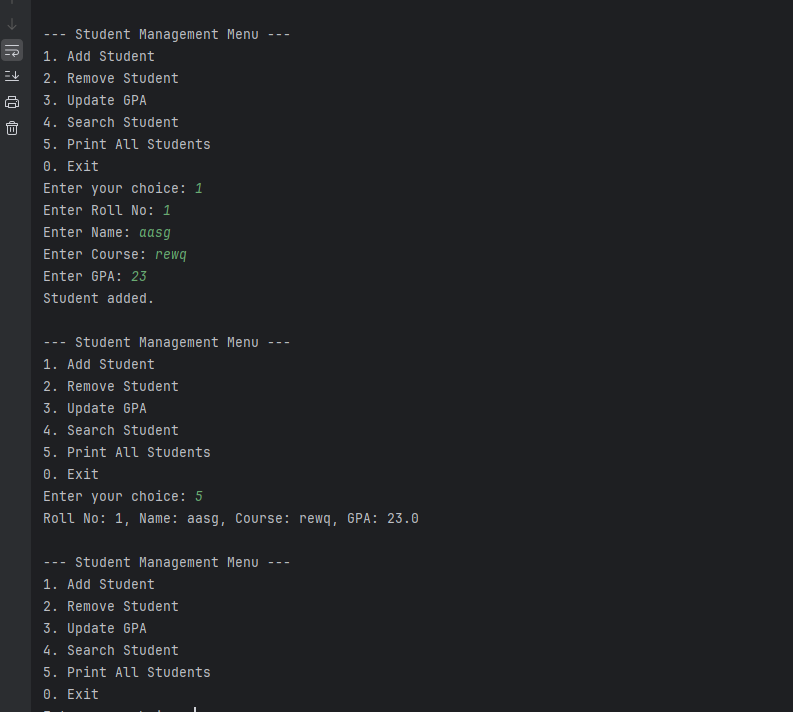
**Graded Lab 1 DSA (0395)**

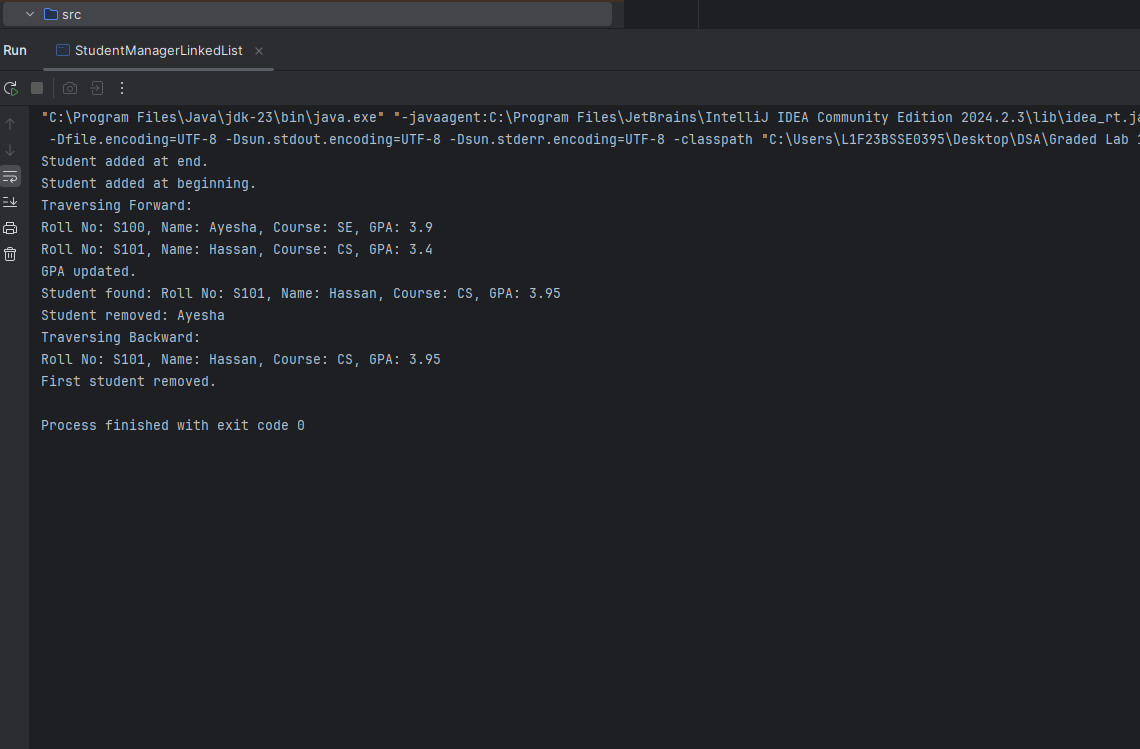
**TASK1**

import java.util.ArrayList;  
import java.util.Scanner;  
class Student {  
 String rollNo;  
 String name;  
 String course;  
 double GPA;  
  
 public Student(String rollNo, String name, String course, double GPA) {  
 this.rollNo = rollNo;  
 this.name = name;  
 this.course = course;  
 this.GPA = GPA;  
 }  
  
 @Override  
 public String toString() {  
 return "Roll No: " + rollNo + ", Name: " + name + ", Course: " + course + ", GPA: " + GPA;  
 }  
}  
  
  
  
  
public class StudentManagerArrayList{  
 private ArrayList<Student> students = new ArrayList<>();  
  
 public void addStudent(Student s) {  
 students.add(s);  
 System.*out*.println("Student added.");  
 }  
  
 public void removeStudent(String rollNo) {  
 for (Student s : students) {  
 if (s.rollNo.equals(rollNo)) {  
 students.remove(s);  
 System.*out*.println("Student removed: " + s.name);  
 return;  
 }  
 }  
 System.*out*.println("Student not found.");  
 }  
  
  
 public void updateGPA(String rollNo, double newGPA) {  
 for (Student s : students) {  
 if (s.rollNo.equals(rollNo)) {  
 s.GPA = newGPA;  
 System.*out*.println("GPA updated.");  
 return;  
 }  
 }  
 System.*out*.println("Student not found.");  
 }  
  
 public void searchStudent(String rollNo) {  
 for (Student s : students) {  
 if (s.rollNo.equals(rollNo)) {  
 System.*out*.println("Student found: " + s);  
 return;  
 }  
 }  
 System.*out*.println("Student not found.");  
 }  
  
 public void printAllStudents() {  
 if (students.isEmpty()) {  
 System.*out*.println("No students to display.");  
 } else {  
 for (Student s : students) {  
 System.*out*.println(s);  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 StudentManagerArrayList manager = new StudentManagerArrayList();  
 Scanner scanner = new Scanner(System.*in*);  
 int choice;  
  
 do {  
 System.*out*.println("\n--- Student Management Menu ---");  
 System.*out*.println("1. Add Student");  
 System.*out*.println("2. Remove Student");  
 System.*out*.println("3. Update GPA");  
 System.*out*.println("4. Search Student");  
 System.*out*.println("5. Print All Students");  
 System.*out*.println("0. Exit");  
 System.*out*.print("Enter your choice: ");  
 choice = scanner.nextInt();  
 scanner.nextLine(); // clear buffer  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter Roll No: ");  
 String rollNo = scanner.nextLine();  
 System.*out*.print("Enter Name: ");  
 String name = scanner.nextLine();  
 System.*out*.print("Enter Course: ");  
 String course = scanner.nextLine();  
 System.*out*.print("Enter GPA: ");  
 double gpa = scanner.nextDouble();  
 manager.addStudent(new Student(rollNo, name, course, gpa));  
 break;  
 case 2:  
 System.*out*.print("Enter Roll No to remove: ");  
 rollNo = scanner.nextLine();  
 manager.removeStudent(rollNo);  
 break;  
 case 3:  
 System.*out*.print("Enter Roll No to update GPA: ");  
 rollNo = scanner.nextLine();  
 System.*out*.print("Enter new GPA: ");  
 gpa = scanner.nextDouble();  
 manager.updateGPA(rollNo, gpa);  
 break;  
 case 4:  
 System.*out*.print("Enter Roll No to search: ");  
 rollNo = scanner.nextLine();  
 manager.searchStudent(rollNo);  
 break;  
 case 5:  
 manager.printAllStudents();  
 break;  
 case 0:  
 System.*out*.println("Exiting...");  
 break;  
 default:  
 System.*out*.println("Invalid choice. Try again.");  
 }  
 } while (choice != 0);  
  
 scanner.close();  
 }  
  
}



**TASK2**

import java.util.LinkedList;  
import java.util.ListIterator;  
import java.util.Scanner;  
  
public class StudentManagerLinkedList {  
 private LinkedList<Student> students = new LinkedList<>();  
  
 public void addFirst(Student s) {  
 students.addFirst(s);  
 System.*out*.println("Student added at beginning.");  
 }  
  
 public void addLast(Student s) {  
 students.addLast(s);  
 System.*out*.println("Student added at end.");  
 }  
  
 public void removeFirst() {  
 if (!students.isEmpty()) {  
 students.removeFirst();  
 System.*out*.println("First student removed.");  
 } else {  
 System.*out*.println("List is empty.");  
 }  
 }  
  
 public void removeLast() {  
 if (!students.isEmpty()) {  
 students.removeLast();  
 System.*out*.println("Last student removed.");  
 } else {  
 System.*out*.println("List is empty.");  
 }  
 }  
  
  
 public void removeByRollNo(String rollNo) {  
 for (Student s : students) {  
 if (s.rollNo.equals(rollNo)) {  
 students.remove(s);  
 System.*out*.println("Student removed: " + s.name);  
 return;  
 }  
 }  
 System.*out*.println("Student not found.");  
 }  
  
  
 public void updateGPA(String rollNo, double newGPA) {  
 for (Student s : students) {  
 if (s.rollNo.equals(rollNo)) {  
 s.GPA = newGPA;  
 System.*out*.println("GPA updated.");  
 return;  
 }  
 }  
 System.*out*.println("Student not found.");  
 }  
  
 public void searchStudent(String rollNo) {  
 for (Student s : students) {  
 if (s.rollNo.equals(rollNo)) {  
 System.*out*.println("Student found: " + s);  
 return;  
 }  
 }  
 System.*out*.println("Student not found.");  
 }  
  
 public void traverseForward() {  
 System.*out*.println("Traversing Forward:");  
 ListIterator<Student> iterator = students.listIterator();  
 while (iterator.hasNext()) {  
 System.*out*.println(iterator.next());  
 }  
 }  
  
 public void traverseBackward() {  
 System.*out*.println("Traversing Backward:");  
 ListIterator<Student> iterator = students.listIterator(students.size());  
 while (iterator.hasPrevious()) {  
 System.*out*.println(iterator.previous());  
 }  
 }  
  
 public static void main(String[] args) {  
 StudentManagerLinkedList manager = new StudentManagerLinkedList();  
  
 manager.addLast(new Student("S101", "Hassan", "CS", 3.4));  
 manager.addFirst(new Student("S100", "Ayesha", "SE", 3.9));  
 manager.traverseForward();  
  
 manager.updateGPA("S101", 3.95);  
 manager.searchStudent("S101");  
  
 manager.removeByRollNo("S100");  
 manager.traverseBackward();  
  
 manager.removeFirst();  
  
 }  
}



**Task 3**

**1. When to Use ArrayList vs LinkedList:**

ArrayList is generally preferred when you need fast random access to elements by index and are not frequently modifying the list (e.g., adding or removing elements). It is best used when the size of the data is mostly static or when you are mostly iterating over the list or accessing individual elements.

LinkedList, on the other hand, is ideal when your list will undergo a lot of insertions or deletions, particularly in the middle or at the ends of the list

**2. Pros and Cons in Terms of Insertion, Deletion, and Traversal:**

Insertion:

**ArrayList**: Inserting an element in the middle of the list or at the beginning requires shifting all the subsequent elements, which results in **O(n)** time complexity.

**LinkedList**: Inserting at the beginning or end of the list is **O(1)**. Inserting at the middle can be O(n), but this is still faster than the ArrayList,

**Deletion:**

**ArrayList**: Deletion from the middle or beginning requires shifting the subsequent elements to fill the gap, resulting in **O(n)** time complexity.

**LinkedList**: Deletion from the beginning or end is **O(1)**, but deleting an element from the middle requires O(n) to traverse the list first, and then O(1) for the deletion itself.

**Traversal:**

**ArrayList: O(n)** for traversing the list. Since the elements are stored in a contiguous block of memory, iteration is faster compared to LinkedList due to better cache locality.

**LinkedList**: **O(n)** for traversal, but since elements are scattered in memory, the performance might be slower due to the higher cost of memory access and pointer dereferencing.

**3. Examples from the Tasks Above:**

**ArrayList** might be better suited for the StudentManagerLinkedList class if you are frequently accessing elements by index, as in situations where you need fast random access to students, for example, when displaying specific details based on their position in the list.

**LinkedList** was more appropriate for the StudentManagerLinkedList class in the task above since the code involves frequent insertions and deletions at both ends (beginning and end), which is more efficient with a LinkedList (O(1) for adding/removing students at both ends). This makes LinkedList a good choice for tasks where elements are added or removed dynamically from both ends.

**4. Time Complexity Comparisons:**

| **Operation** | **ArrayList** | **LinkedList** |
| --- | --- | --- |
| Access by Index | **O(1)** | **O(n)** |
| Search | **O(n)** | **O(n)** |
| Insertion (Beginning) | **O(n)** | **O(1)** |
| Insertion (End) | **O(1)** | **O(1)** |
| Deletion (Beginning) | **O(n)** | **O(1)** |
| Deletion (End) | **O(1)** | **O(1)** |
| Traversal | **O(n)** | **O(n)** |