Experiment-7

Student Name: Gagnesh Kakkar
UID: 23BCS11196
Branch: B.E-C.S.E
Section/Group: 23KRG-2B
Semester: 5th
Date of Performance: 06/10/2025
Subject Name: PBLJ
Subject Code: 23CSH-304

Easy Level

1. **Aim:** Create a Java program to connect to a MySQL database and fetch data from a single table.
   The program should:
   Use DriverManager and Connection objects.
   Retrieve and display all records from a table named Employee with columns EmpID, Name, and Salary.

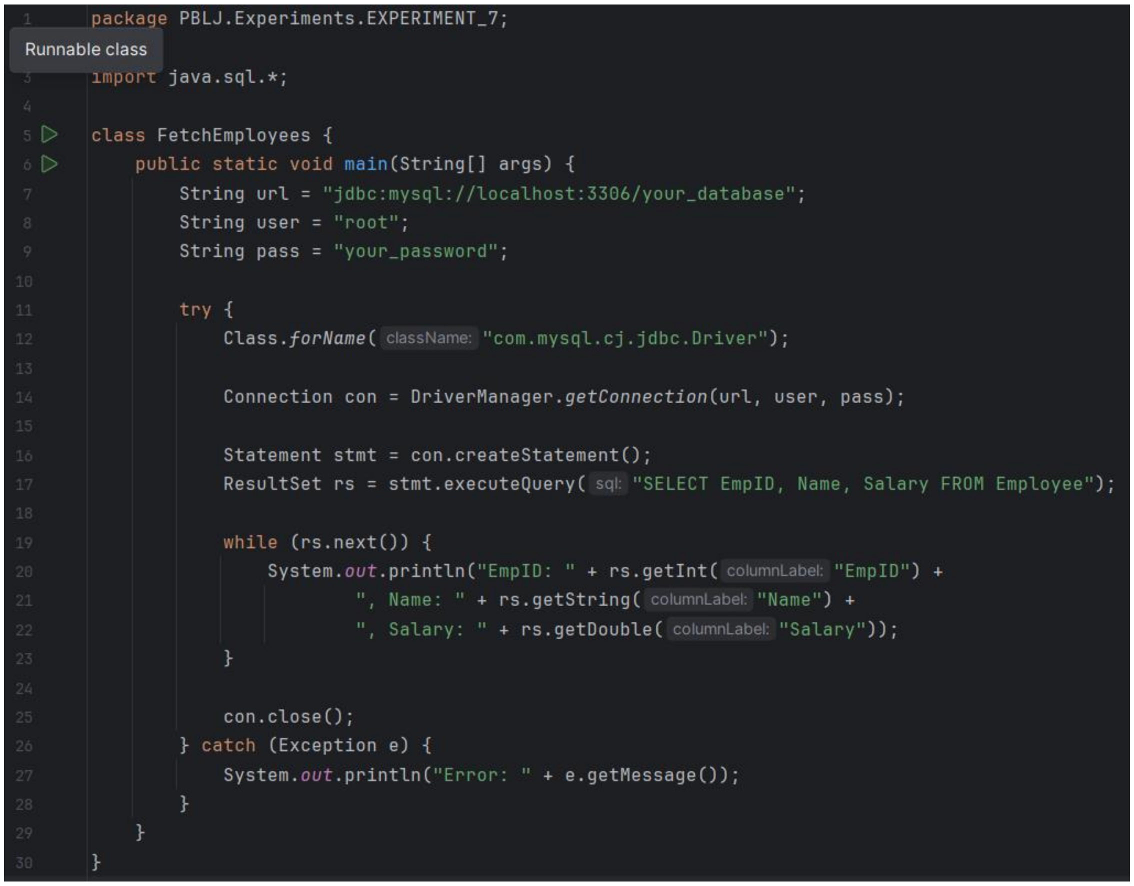
2. **Objective:** Understand JDBC connectivity using DriverManager and Connection classes to retrieve data.

3. **Input/Apparatus Used:** MySQL, JDBC API, Employee Table, Java Application.

4. **Procedure:**
   1. Set up a MySQL database with a table named 'Employee' having columns: EmpID, Name, and Salary.
   2. Load the JDBC driver class using Class.forName().
   3. Establish a connection using DriverManager.getConnection().
   4. Create a Statement object and execute a SELECT query.
   5. Process the ResultSet to retrieve and display all employee records.
   6. Close the connection and handle SQL exceptions using try-catch blocks.

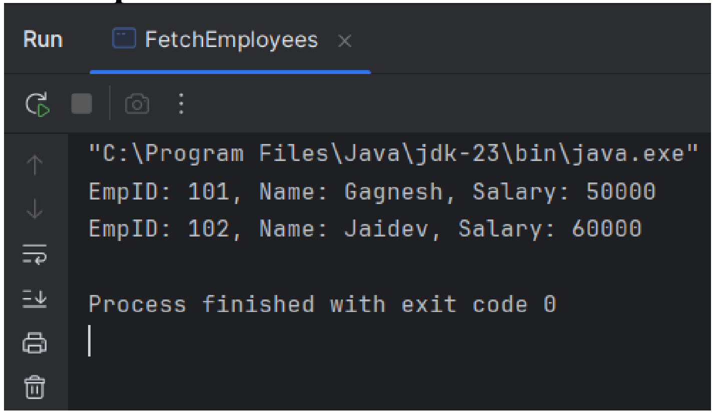
5.
Sample Output:
```
EmpID: 101, Name: John, Salary: 50000
EmpID: 102, Name: Alice, Salary: 60000
```

6. Code:



```
package PBLJ.Experiments.EXPERIMENT_7;

import java.sql.*;

class FetchEmployees {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/your_database";
        String user = "root";
        String pass = "your_password";

        try {
            Class.forName( className: "com.mysql.cj.jdbc.Driver");

            Connection con = DriverManager.getConnection(url, user, pass);

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery( sql: "SELECT EmpID, Name, Salary FROM Employee");

            while (rs.next()) {
                System.out.println("EmpID: " + rs.getInt( columnLabel: "EmpID") +
                        ", Name: " + rs.getString( columnLabel: "Name") +
                        ", Salary: " + rs.getDouble( columnLabel: "Salary"));
            }

            con.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

7. Output:



```
"C:\Program Files\Java\jdk-23\bin\java.exe"
EmpID: 101, Name: Gagnesh, Salary: 50000
EmpID: 102, Name: Jaidev, Salary: 60000

Process finished with exit code 0
```

Medium Level

1. **Aim:** Build a program to perform CRUD operations (Create, Read, Update, Delete) on a database table Product with columns:
   ProductID, ProductName, Price, and Quantity.
   The program should include:
   Menu-driven options for each operation.
   Transaction handling to ensure data integrity.

2. **Objective:** Learn how to manage data integrity through JDBC transaction handling in a menu-driven program.

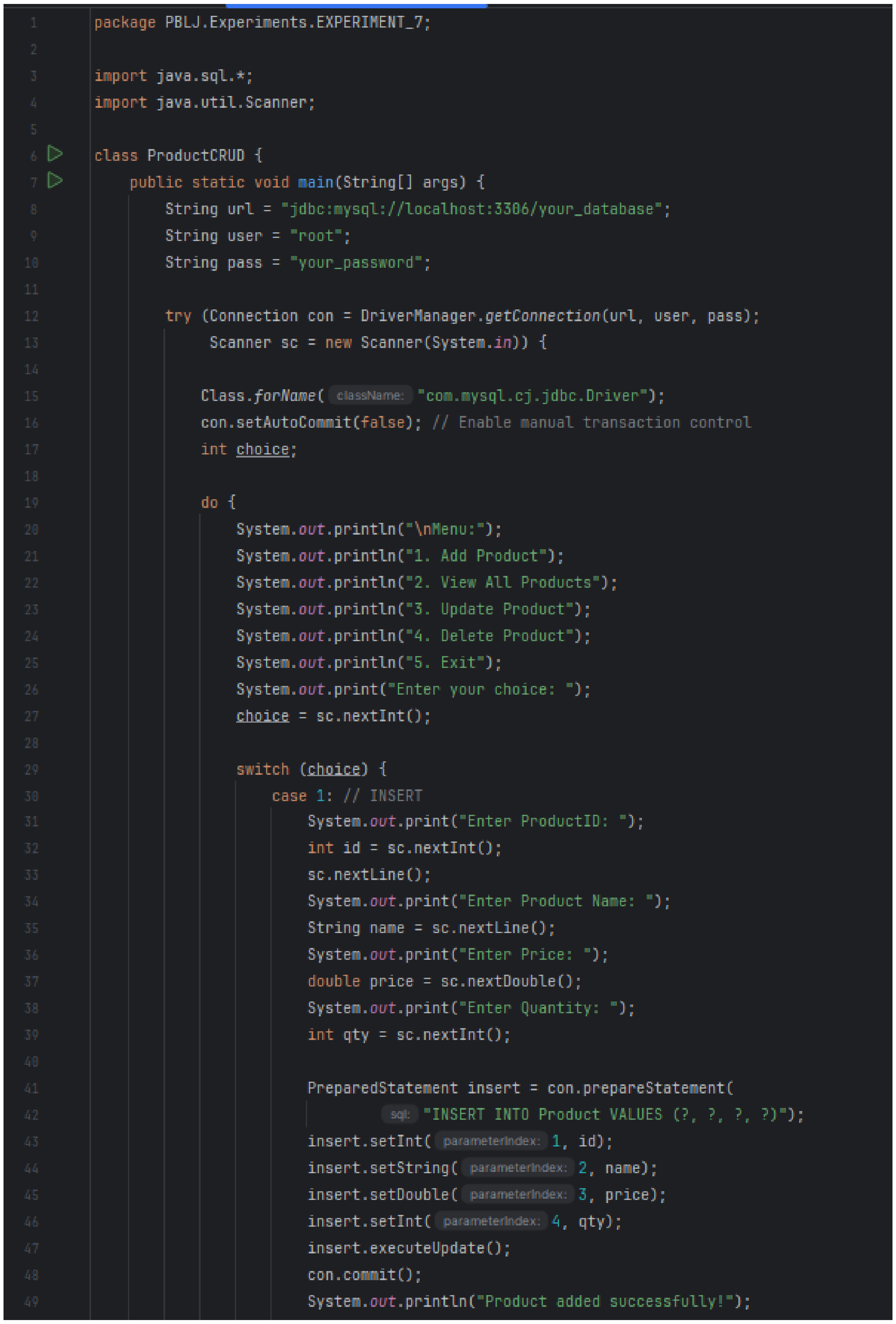
3. **Input/Apparatus Used:** MySQL Database, Product Table, JDBC API, Java Code

4. **Procedure:**
   1. Create a MySQL table named 'Product' with columns: ProductID, ProductName, Price, Quantity.
   2. Establish a database connection using JDBC.
   3. Create menu options to perform Create, Read, Update, and Delete operations.
   4. Use PreparedStatement to prevent SQL injection.
   5. Implement transaction handling: commit for success, rollback on exceptions.
   6. Close all JDBC resources in finally blocks.

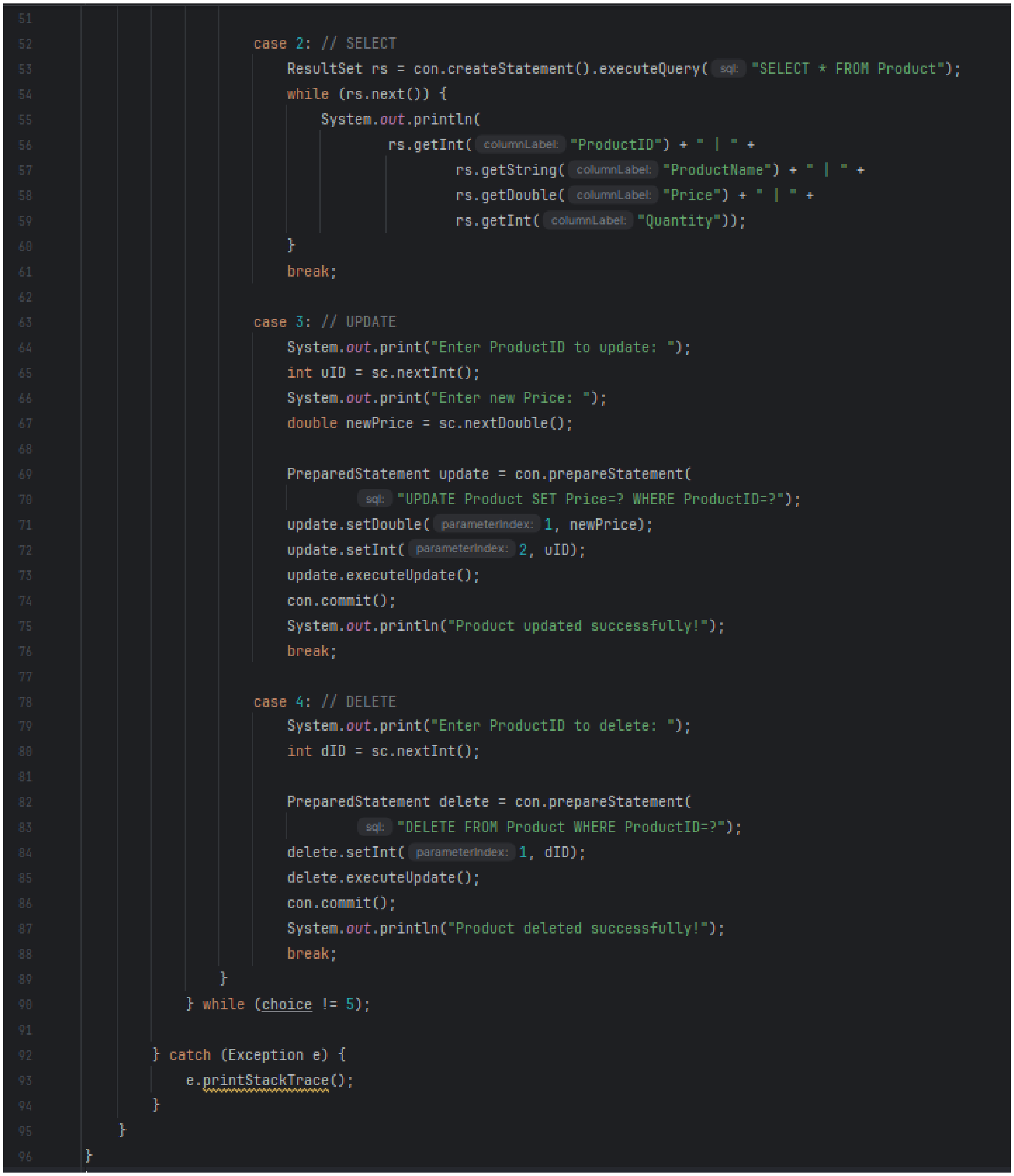
5. **Sample Output :**

```
Menu:
1. Add Product
2. View All Products
3. Update Product
4. Delete Product
5. Exit
Enter your choice: 1
Product added successfully!
```

6. Code:



```java
package PBLJ.Experiments.EXPERIMENT_7;

import java.sql.*;
import java.util.Scanner;

class ProductCRUD {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/your_database";
        String user = "root";
        String pass = "your_password";

        try (Connection con = DriverManager.getConnection(url, user, pass);
             Scanner sc = new Scanner(System.in)) {

            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            con.setAutoCommit(false); // Enable manual transaction control
            int choice;

            do {
                System.out.println("\nMenu:");
                System.out.println("1. Add Product");
                System.out.println("2. View All Products");
                System.out.println("3. Update Product");
                System.out.println("4. Delete Product");
                System.out.println("5. Exit");
                System.out.print("Enter your choice: ");
                choice = sc.nextInt();

                switch (choice) {
                    case 1: // INSERT
                        System.out.print("Enter ProductID: ");
                        int id = sc.nextInt();
                        sc.nextLine();
                        System.out.print("Enter Product Name: ");
                        String name = sc.nextLine();
                        System.out.print("Enter Price: ");
                        double price = sc.nextDouble();
                        System.out.print("Enter Quantity: ");
                        int qty = sc.nextInt();

                        PreparedStatement insert = con.prepareStatement(
                                 sql: "INSERT INTO Product VALUES (?, ?, ?, ?)");
                        insert.setInt( parameterIndex: 1, id);
                        insert.setString( parameterIndex: 2, name);
                        insert.setDouble( parameterIndex: 3, price);
                        insert.setInt( parameterIndex: 4, qty);
                        insert.executeUpdate();
                        con.commit();
                        System.out.println("Product added successfully!");
```



```java

                    case 2: // SELECT
                        ResultSet rs = con.createStatement().executeQuery("SELECT * FROM Product");
                        while (rs.next()) {
                            System.out.println(
                                    rs.getInt("ProductID") + " | " +
                                            rs.getString("ProductName") + " | " +
                                            rs.getDouble("Price") + " | " +
                                            rs.getInt("Quantity"));
                        }
                        break;

                    case 3: // UPDATE
                        System.out.print("Enter ProductID to update: ");
                        int uID = sc.nextInt();
                        System.out.print("Enter new Price: ");
                        double newPrice = sc.nextDouble();

                        PreparedStatement update = con.prepareStatement(
                                "UPDATE Product SET Price=? WHERE ProductID=?");
                        update.setDouble(1, newPrice);
                        update.setInt(2, uID);
                        update.executeUpdate();
                        con.commit();
                        System.out.println("Product updated successfully!");
                        break;

                    case 4: // DELETE
                        System.out.print("Enter ProductID to delete: ");
                        int dID = sc.nextInt();

                        PreparedStatement delete = con.prepareStatement(
                                "DELETE FROM Product WHERE ProductID=?");
                        delete.setInt(1, dID);
                        delete.executeUpdate();
                        con.commit();
                        System.out.println("Product deleted successfully!");
                        break;
                }
            } while (choice != 5);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Hard Level

1. Aim:

Develop a Java application using JDBC and MVC architecture to manage student data. The application should:

1. Use a Student class as the model with fields like StudentID, Name, Department, and Marks.
2. Include a database table to store student data.
3. Allow the user to perform CRUD operations through a simple menu-driven view.
4. Implement database operations in a separate controller class.

2. Objective:

Demonstrate advanced stream operations including groupingBy, maxBy, and averagingDouble.

3. Input/Apparatus Used:

Apply MVC design pattern for separation of concern and perform CRUD operations effectively.

4. Procedure:

1. Create a MySQL table 'Student' with columns: StudentID, Name, Department, Marks.
2. Create a Student class as the model with fields and constructor.
3. Create a View class to display a menu for user input and show results.
4. Develop a Controller class with methods to perform CRUD using JDBC.
5. Use main() method to connect view actions to controller methods.
6. Handle all exceptions and maintain clean separation between layers.

Sample Output:

```
--- Student Management System ---
1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2
StudentID: 1001, Name: Ravi, Department: CSE, Marks: 85
```

5. Code:

Model Class (Student.java)

```
public class Student {
    int studentID;
    String name;
    String department;
    double marks;

    public Student(int studentID, String name, String department, double
marks) {
        this.studentID = studentID;
        this.name = name;
        this.department = department;
        this.marks = marks;
    }
}
```

Controller Class (StudentController.java)

```
import java.sql.*;


public class StudentController {

    Connection con;


    public StudentController(String url, String user, String pass) throws
Exception {

        Class.forName("com.mysql.cj.jdbc.Driver");

        con = DriverManager.getConnection(url, user, pass);

    }


    public void addStudent(Student s) throws Exception {

        PreparedStatement ps = con.prepareStatement("INSERT INTO Student VALUES
(?, ?, ?, ?)");

        ps.setInt(1, s.studentID);

        ps.setString(2, s.name);

        ps.setString(3, s.department);
        ps.setDouble(4, s.marks);
        ps.executeUpdate();
        System.out.println("Student added successfully!");
    }

    public void viewStudents() throws Exception {
        ResultSet rs = con.createStatement().executeQuery("SELECT * FROM Student");
        while (rs.next()) {
            System.out.println(rs.getInt(1) + " | " + rs.getString(2) + " | " + rs.getString(3) + " | " + rs.getDouble(4));
        }
    }
}
```

View / Main (StudentApp.java)

```
import java.util.*;

public class StudentApp {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);

        StudentController controller =
                new StudentController("jdbc:mysql://localhost:3306/your_database", "root", "your_password");

        int choice;
        do {
            System.out.println("\n--- Student Management System ---");
            System.out.println("1. Add Student");
```

```
                System.out.println("2. View All Students");
                System.out.println("3. Exit");
                System.out.print("Enter choice: ");
                choice = sc.nextInt();

                switch (choice) {
                    case 1:
                        System.out.print("Student ID: ");
                        int id = sc.nextInt();
                        sc.nextLine();
                        System.out.print("Name: ");
                        String name = sc.nextLine();
                        System.out.print("Department: ");
                        String dept = sc.nextLine();
                        System.out.print("Marks: ");
                        double marks = sc.nextDouble();
                        controller.addStudent(new Student(id, name, dept, marks));
                        break;

                    case 2:
                        controller.viewStudents();
                        break;
                }
            } while (choice != 3);

            sc.close();
        }
    }
```

6. Output:

```
 --- Student Management System ---
1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2
StudentID: 1001, Name: Gagnesh, Department: CSE, Marks: 85
```