K.R. Mangalam University

School of Engineering & Technology

Lab Assignment 1

Java Programming

Submitted by:

Name: Rakesh G

Roll No: 2401201064

Class: BCA (AI & DS)

Submitted to:

Dr. Manish Kumar

GitHub Repository:

https://github.com/rakesh4407/Java_Assignment_Rakesh

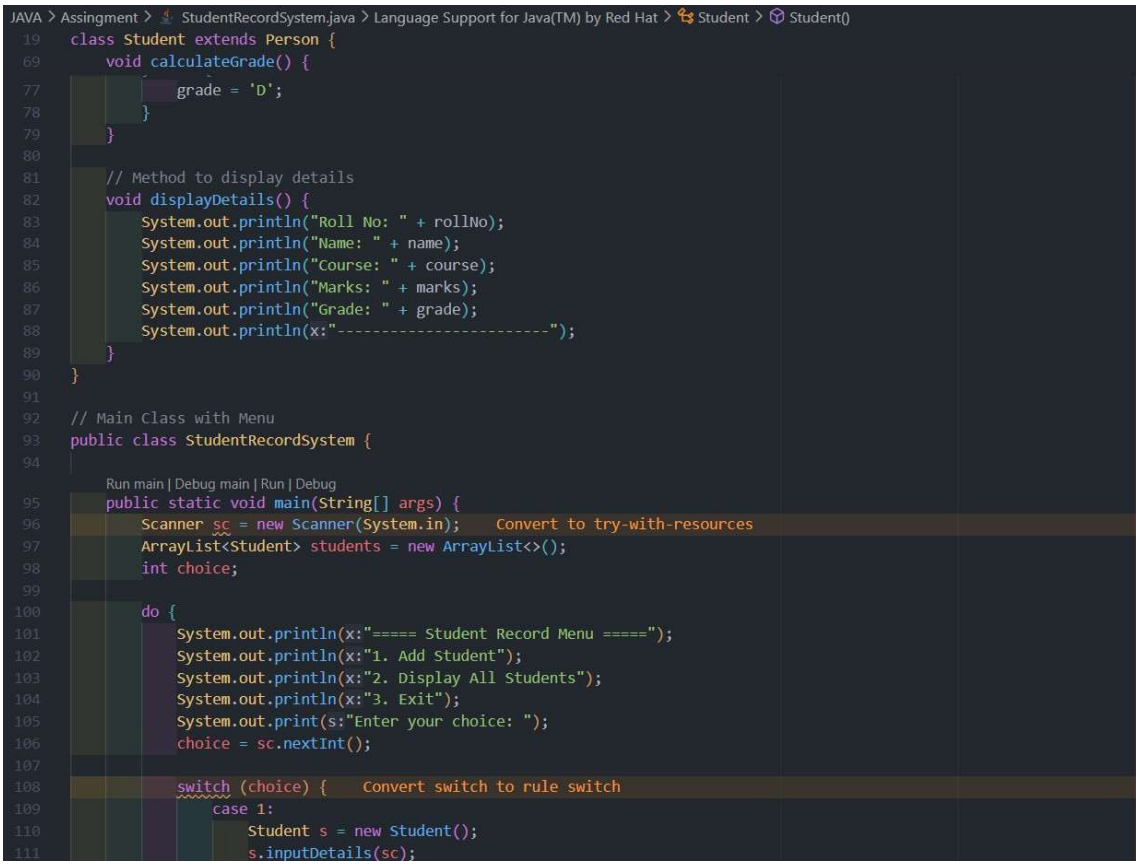
CODE:

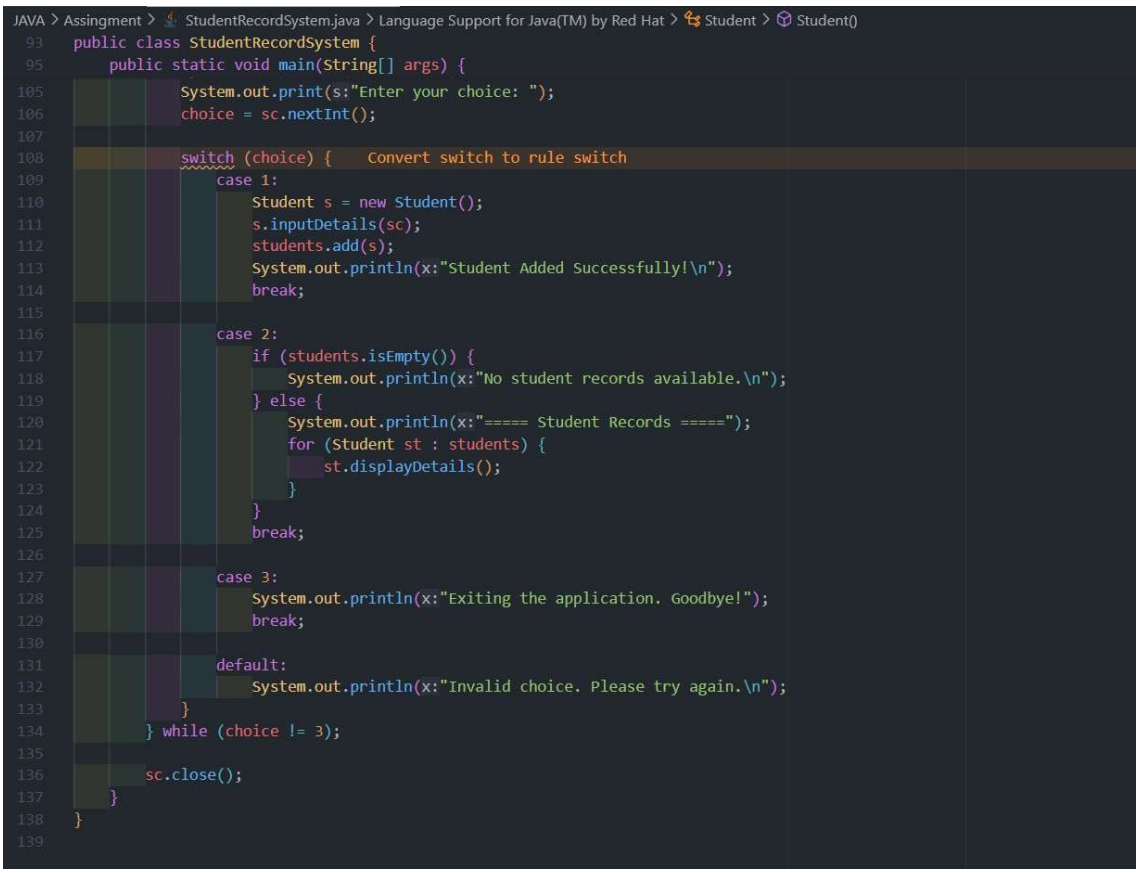
D:\RAKESH\VSC\JAVA\practice.java ecordSystem.java > Language Support for Java(TM) by Red Hat > Student > Student()

```java
import java.util.ArrayList;
import java.util.Scanner;

// Base Class
class Person {

    String name;

    Person() {
        this.name = "";
    }

    Person(String name) {
        this.name = name;
    }
}

// Derived Class: Student
class Student extends Person {

    int rollNo;
    String course;
    double marks;
    char grade;

    // Default Constructor
    Student() {
        super();
        this.rollNo = 0;
        this.course = "";
        this.marks = 0.0;
        this.grade = 'F';
    }

    // Parameterized Constructor
    Student(int rollNo, String name, String course, double marks) {    Constructor is never used
        super(name);
        this.rollNo = rollNo;
```

JAVA > Assingment > StudentRecordSystem.java > Language Support for Java(TM) by Red Hat > Student > Student()

```java
class Student extends Person {
    Student(int rollNo, String name, String course, double marks) {    Constructor is never used
        this.marks = marks;
        calculateGrade();    Overridable method call in constructor
    }

    // Method to take input from user
    void inputDetails(Scanner sc) {
        System.out.print(s:"Enter Roll No: ");
        rollNo = sc.nextInt();
        sc.nextLine(); // consume newline

        System.out.print(s:"Enter Name: ");
        name = sc.nextLine();

        System.out.print(s:"Enter Course: ");
        course = sc.nextLine();

        System.out.print(s:"Enter Marks (0-100): ");
        marks = sc.nextDouble();

        // Validate marks
        while (marks < 0 || marks > 100) {
            System.out.print(s:"Invalid Marks! Enter again (0-100): ");
            marks = sc.nextDouble();
        }

        calculateGrade();
    }

    // Method to calculate grade
    void calculateGrade() {
        if (marks >= 85) {
            grade = 'A';
        } else if (marks >= 70) {
            grade = 'B';
        } else if (marks >= 50) {
            grade = 'C';
```


JAVA > Assingment > StudentRecordSystem.java > Language Support for Java(TM) by Red Hat > Student > Student()
class Student extends Person {
void calculateGrade() {
grade = 'D';
}
}
// Method to display details
void displayDetails() {
System.out.println("Roll No: " + rollNo);
System.out.println("Name: " + name);
System.out.println("Course: " + course);
System.out.println("Marks: " + marks);
System.out.println("Grade: " + grade);
System.out.println(x:"------------------------");
}
}
// Main Class with Menu
public class StudentRecordSystem {
Run main | Debug main | Run | Debug
public static void main(String[] args) {
Scanner sc = new Scanner(System.in); Convert to try-with-resources
ArrayList<Student> students = new ArrayList<>();
int choice;
do {
System.out.println(x:"===== Student Record Menu =====");
System.out.println(x:"1. Add Student");
System.out.println(x:"2. Display All Students");
System.out.println(x:"3. Exit");
System.out.print(s:"Enter your choice: ");
choice = sc.nextInt();
switch (choice) { Convert switch to rule switch
case 1:
Student s = new Student();
s.inputDetails(sc);


JAVA > Assingment > StudentRecordSystem.java > Language Support for Java(TM) by Red Hat > Student > Student()
public class StudentRecordSystem {
public static void main(String[] args) {
System.out.print(s:"Enter your choice: ");
choice = sc.nextInt();
switch (choice) { Convert switch to rule switch
case 1:
Student s = new Student();
s.inputDetails(sc);
students.add(s);
System.out.println(x:"Student Added Successfully!\n");
break;
case 2:
if (students.isEmpty()) {
System.out.println(x:"No student records available.\n");
} else {
System.out.println(x:"===== Student Records =====");
for (Student st : students) {
st.displayDetails();
}
}
break;
case 3:
System.out.println(x:"Exiting the application. Goodbye!");
break;
default:
System.out.println(x:"Invalid choice. Please try again.\n");
}
} while (choice != 3);
sc.close();
}
}

OUTPUT:

```
===== Student Record Menu =====
1. Add Student
2. Display All Students
3. Exit
Enter your choice: 1
Enter Roll No: 64
Enter Name: Rakesh
Enter Course: BCA
Enter Marks (0-100): 99
Student Added Successfully!

===== Student Record Menu =====
1. Add Student
2. Display All Students
3. Exit
Enter your choice: 2
===== Student Records =====
Roll No: 64
Name: Rakesh
Course: BCA
Marks: 99.0
Grade: A
-----------------------
===== Student Record Menu =====
1. Add Student
2. Display All Students
3. Exit
Enter your choice: 3
Exiting the application. Goodbye!

D:\RAKESH\VSC>
```

Explanation of the Code

This program implements a simple Student Record Management System using Object-Oriented Programming (OOP) concepts in Java.
It allows the user to add student details, calculate grades, and display all student records using a structured, class-based approach.

1. Class Design
The program defines a class named Student that models a real-world student entity.
It contains the following fields (attributes):

- **int rollNo – to store the student's roll number.**
- **String name – to store the student's name.**
- **String course – to store the course name.**
- **double marks – to store marks obtained.**
- **char grade – to store the calculated grade.**

This class also includes a default constructor (for initializing empty values) and a parameterized constructor (to initialize data when creating objects directly).

2. Methods in the Student Class
The Student class defines three important methods:

- **inputDetails() → Takes user input for roll number, name, course, and marks using the Scanner class.**
- **calculateGrade() → Calculates the grade based on marks using conditional statements (if-else).**
  For example:
  - **Marks ≥ 90 → Grade A**
  - **Marks ≥ 75 → Grade B**
  - **Marks ≥ 50 → Grade C**
  - **Else → Grade D**
- **displayDetails() → Displays all the student's details including calculated grade.**

This structure shows how methods encapsulate behavior within a class — a key OOP principle.

3. Managing Multiple Students
In the main program (inside main()), an ArrayList or 1D array is used to store multiple student objects.
The program uses a menu-driven system:
1. Add Student
2. Display All Students
3. Exit

- **Option 1 → Calls inputDetails() and adds the new student to the list.**
- **Option 2 → Iterates through the array or list and displays details of each student.**
- **Option 3 → Exits the application gracefully.**

4. Control Structures
The menu uses a while loop to continuously display options until the user selects Exit.
A switch-case statement handles menu choices clearly and avoids long chains of if-else statements.

5. Data Validation

The code ensures valid input for marks (between 0 and 100).

If invalid marks are entered, it displays an error message and prevents the record from being added.

This shows the use of conditional control statements for validation.

6. Core Java Concepts Used

Concept	Usage in the Program
Class & Object	To model a real-world student.
Constructors	To initialize student data.
Encapsulation	Data is stored and accessed through methods.
Array / ArrayList	To store multiple student records.
Conditional Statements	To calculate grades and validate marks.
Loops	To create a menu system.
Scanner Class	For user input.

Conclusion

This assignment effectively demonstrates the core principles of OOP such as class design, encapsulation, and method implementation.

It also integrates control structures, input/output handling, and arrays to build a practical Student Record Management System.

By completing this program, the student gains hands-on experience in object-oriented programming, data validation, and user interaction using Java.