

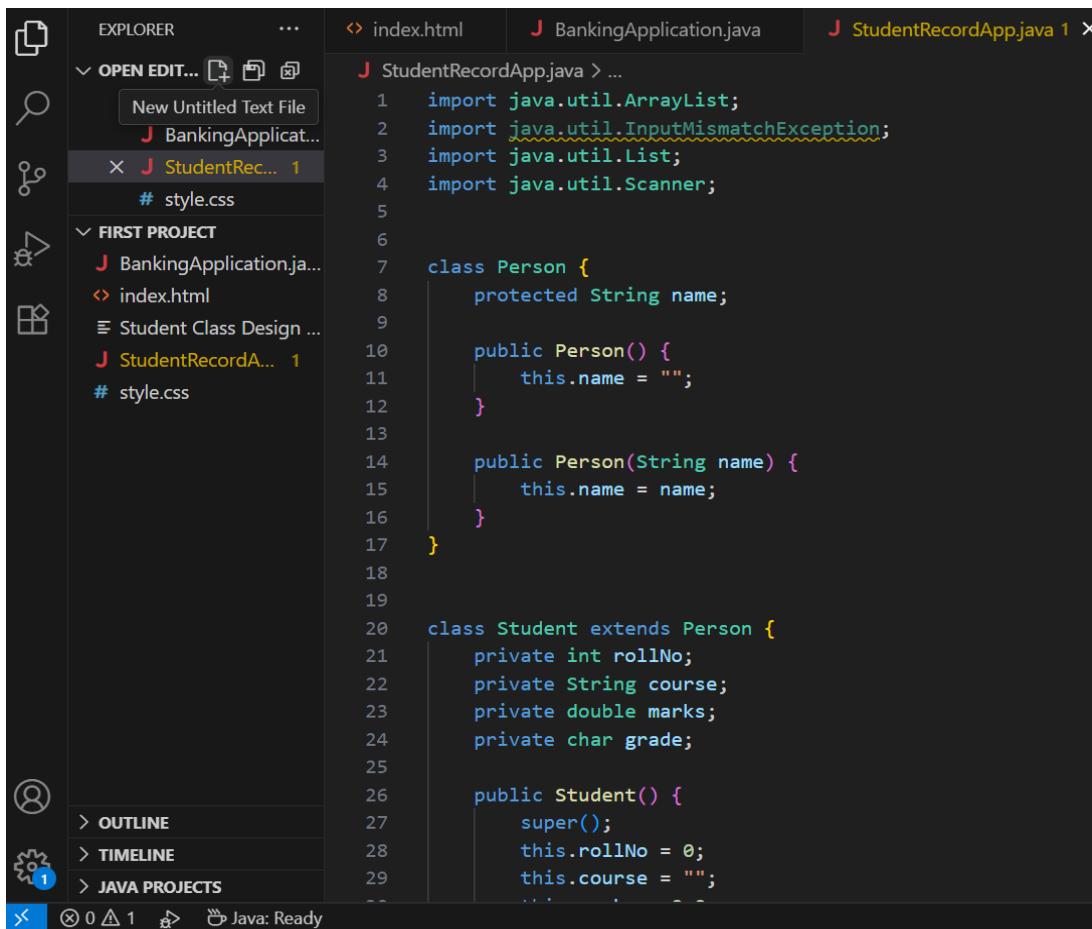
JAVA LAB ASSIGNMENT NUMBER: 01

Name: Aman

Roll no: 2401201115

Course: BCA (AI&DS)

INPUT:



The screenshot shows a Java development environment with the following details:

- EXPLORER View:** Shows a project structure under "FIRST PROJECT". It includes files: index.html, BankingApplication.java, Student Class Design ..., StudentRecordApp.java (selected), and style.css.
- Code Editor:** Displays two Java files:
 - BankingApplication.java:** Contains imports for java.util.ArrayList, java.util.InputMismatchException, java.util.List, and java.util.Scanner. It defines a Person class with a protected String name and a constructor that initializes it to an empty string. It also defines a public Person(String name) constructor.
 - StudentRecordApp.java:** Contains imports for java.util.ArrayList, java.util.InputMismatchException, java.util.List, and java.util.Scanner. It defines a Student class that extends Person. The Student class has private fields for rollNo (int), course (String), marks (double), and grade (char). Its constructor calls super() and initializes rollNo to 0 and course to an empty string.
- Bottom Status Bar:** Shows icons for file operations, a Java status message ("Java: Ready"), and other system indicators.

```
EXPLORER ... index.html J BankingApplication.java J StudentRecordApp.java 1 X # style.css  
OPEN EDITORS index.html J BankingApplicat... J StudentRec... 1 # style.css  
Run and Debug (Ctrl+Shift+D) BankingApplication.java...  
index.html Student Class Design ... J StudentRecordA... 1 # style.css  
StudentRecordApp.java 1 ...  
20  class Student extends Person {  
21      public Student() {  
22          this.marks = 0.0;  
23          this.grade = 'F';  
24      }  
25  
26      public Student(int rollNo, String name, String course, double marks) {  
27          super(name);  
28          this.rollNo = rollNo;  
29          this.course = course;  
30          this.marks = marks;  
31          calculateGrade();  
32      }  
33  
34      public void inputDetails(Scanner sc) {  
35          this.rollNo = readInt(sc, prompt: "Enter Roll No: ");  
36          this.name = readNonEmpty(sc, prompt: "Enter Name: ");  
37          this.course = readNonEmpty(sc, prompt: "Enter Course: ");  
38          this.marks = readMarks(sc, prompt: "Enter Marks (0 - 100): ");  
39          calculateGrade();  
40      }  
41  
42      public void displayDetails() {  
43          System.out.println("Roll No: " + rollNo);  
44          System.out.println("Name: " + name);  
45          System.out.println("Course: " + course);  
46      }  
47  
48      private void calculateGrade() {  
49          if (marks <= 10) {  
50              grade = 'E';  
51          } else if (marks >= 11 & marks <= 33) {  
52              grade = 'D';  
53          } else if (marks >= 34 & marks <= 50) {  
54              grade = 'C';  
55          } else if (marks >= 51 & marks <= 66) {  
56              grade = 'B';  
57          } else {  
58              grade = 'A';  
59          }  
60      }  
61  
62      private static int readInt(Scanner sc, String prompt) {  
63          while (true) {  
64              System.out.print(prompt);  
65              try {  
66                  String line = sc.nextLine().trim();  
67                  int val = Integer.parseInt(line);  
68                  return val;  
69              } catch (NumberFormatException e) {  
70                  System.out.println("Invalid integer. Please try again.");  
71              }  
72          }  
73      }  
74  
75      private static double readMarks(Scanner sc, String prompt) {  
76          while (true) {  
77              System.out.print(prompt);  
78          }  
79      }  
80  
81      public void displayDetails() {  
82          System.out.println("Roll No: " + rollNo);  
83          System.out.println("Name: " + name);  
84          System.out.println("Course: " + course);  
85          System.out.println("Grade: " + grade);  
86      }  
87  
88      public void calculateGrade() {  
89          calculateGrade();  
90      }  
91  
92      public void displayDetails() {  
93          System.out.println("Roll No: " + rollNo);  
94          System.out.println("Name: " + name);  
95          System.out.println("Course: " + course);  
96          System.out.println("Grade: " + grade);  
97      }  
98  }  
99
```

```
EXPLORER ... index.html J BankingApplication.java J StudentRecordApp.java 1 X # style.css  
OPEN EDITORS index.html J BankingApplicat... J StudentRec... 1 # style.css  
Run and Debug (Ctrl+Shift+D) BankingApplication.java...  
index.html Student Class Design ... J StudentRecordA... 1 # style.css  
StudentRecordApp.java 1 ...  
20  class Student extends Person {  
21      public Student() {  
22          this.marks = 0.0;  
23          this.grade = 'F';  
24      }  
25  
26      public Student(int rollNo, String name, String course, double marks) {  
27          super(name);  
28          this.rollNo = rollNo;  
29          this.course = course;  
30          this.marks = marks;  
31          calculateGrade();  
32      }  
33  
34      public void inputDetails(Scanner sc) {  
35          this.rollNo = readInt(sc, prompt: "Enter Roll No: ");  
36          this.name = readNonEmpty(sc, prompt: "Enter Name: ");  
37          this.course = readNonEmpty(sc, prompt: "Enter Course: ");  
38          this.marks = readMarks(sc, prompt: "Enter Marks (0 - 100): ");  
39          calculateGrade();  
40      }  
41  
42      public void displayDetails() {  
43          System.out.println("Roll No: " + rollNo);  
44          System.out.println("Name: " + name);  
45          System.out.println("Course: " + course);  
46      }  
47  
48      private void calculateGrade() {  
49          if (marks <= 10) {  
50              grade = 'E';  
51          } else if (marks >= 11 & marks <= 33) {  
52              grade = 'D';  
53          } else if (marks >= 34 & marks <= 50) {  
54              grade = 'C';  
55          } else if (marks >= 51 & marks <= 66) {  
56              grade = 'B';  
57          } else {  
58              grade = 'A';  
59          }  
60      }  
61  
62      private static int readInt(Scanner sc, String prompt) {  
63          while (true) {  
64              System.out.print(prompt);  
65              try {  
66                  String line = sc.nextLine().trim();  
67                  int val = Integer.parseInt(line);  
68                  return val;  
69              } catch (NumberFormatException e) {  
70                  System.out.println("Invalid integer. Please try again.");  
71              }  
72          }  
73      }  
74  
75      private static double readMarks(Scanner sc, String prompt) {  
76          while (true) {  
77              System.out.print(prompt);  
78          }  
79      }  
80  
81      public void displayDetails() {  
82          System.out.println("Roll No: " + rollNo);  
83          System.out.println("Name: " + name);  
84          System.out.println("Course: " + course);  
85          System.out.println("Grade: " + grade);  
86      }  
87  
88      public void calculateGrade() {  
89          calculateGrade();  
90      }  
91  
92      public void displayDetails() {  
93          System.out.println("Roll No: " + rollNo);  
94          System.out.println("Name: " + name);  
95          System.out.println("Course: " + course);  
96          System.out.println("Grade: " + grade);  
97      }  
98  }  
99
```

The screenshot shows a Java code editor interface with the following details:

- File Explorer:** Shows files: `BankingApplication.java`, `StudentRecordApp.java` (active tab), and `style.css`.
- Code Editor:** Displays `StudentRecordApp.java` with code related to reading student marks and non-empty strings.
- Status Bar:** Shows "Run | Debug" and "Java: Ready".
- Bottom Bar:** Shows line 122, column 1.

```
20  class Student extends Person {
95      private static double readMarks(Scanner sc, String prompt) {
98          try {
99              String line = sc.nextLine().trim();
100             double m = Double.parseDouble(line);
101             if (m < 0 || m > 100) {
102                 System.out.println(x: "Marks must be between 0 and 100. Try again.");
103             } else {
104                 return m;
105             }
106         } catch (NumberFormatException e) {
107             System.out.println(x: "Invalid number. Please enter marks like 87.5 or 90");
108         }
109     }
110 }
111
112     private static String readNonEmpty(Scanner sc, String prompt) {
113         while (true) {
114             System.out.print(prompt);
115             String s = sc.nextLine().trim();
116             if (!s.isEmpty()) return s;
117             System.out.println(x: "Input cannot be empty. Try again.");
118         }
119     }
120
121
122 public class StudentRecordApp {
123 }
```

The screenshot shows a Java code editor interface with the following details:

- File Explorer:** Shows files: `index.html`, `BankingApplication.java`, `StudentRecordApp.java` (active tab), and `style.css`.
- Code Editor:** Displays `StudentRecordApp.java` with a main loop handling student records.
- Status Bar:** Shows "Run | Debug" and "Java: Ready".
- Bottom Bar:** Shows line 122, column 1.

```
123 public class StudentRecordApp {
124     public static void main(String[] args) {
125         Scanner sc = new Scanner(System.in);
126         Student> students = new ArrayList<>();
127         boolean running = true;
128
129         while (running) {
130             printMenu();
131             int choice = readMenuChoice(sc);
132
133             switch (choice) {
134                 case 1:
135                     Student s = new Student();
136                     s.inputDetails(sc);
137                     students.add(s);
138                     System.out.println(x: "Student added successfully!");
139                     break;
140                 case 2:
141                     if (students.isEmpty()) {
142                         System.out.println(x: "No student records to display.");
143                     } else {
144                         System.out.println(x: "\n==== All Student Records ====");
145                         for (Student st : students) {
146                             st.displayDetails();
147                         }
148                     }
149                     break;
150                 case 3:
151                     System.out.println(x: "Exiting the application. Goodbye!");
152             }
153         }
154     }
155 }
```

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "BankingApplication".
- File List:** The files listed are index.html, BankingApplication.java, StudentRecordApp.java, and style.css.
- Code Editor:** The code editor displays the `StudentRecordApp.java` file. The code implements a menu system for managing student records using a `Scanner` object (`sc`) and `System.out.println` statements.

```
1 public class StudentRecordApp {  
2     public static void main(String[] args) {  
3         boolean running = false;  
4         Scanner sc = new Scanner(System.in);  
5         int choice;  
6         String line;  
7         System.out.println("Welcome to the Student Record Management System");  
8         System.out.println("Please select an option from the menu below:");  
9         printMenu();  
10        while (running == false) {  
11            System.out.print("Enter your choice: ");  
12            try {  
13                line = sc.nextLine().trim();  
14                choice = Integer.parseInt(line);  
15                switch (choice) {  
16                    case 1:  
17                        addStudent(sc);  
18                        break;  
19                    case 2:  
20                        displayAllStudents(sc);  
21                        break;  
22                    case 3:  
23                        running = true;  
24                        System.out.println("Thank you for using the system.");  
25                        break;  
26                    default:  
27                        System.out.println("Invalid choice. Please select 1, 2, or 3.");  
28                }  
29            } catch (NumberFormatException e) {  
30                System.out.println("Invalid input. Enter a number (1-3).");  
31            }  
32        }  
33    }  
34  
35    private static void printMenu() {  
36        System.out.println("\n===== Student Record Menu =====");  
37        System.out.println("1. Add Student");  
38        System.out.println("2. Display All Students");  
39        System.out.println("3. Exit");  
40    }  
41  
42    private static int readMenuChoice(Scanner sc) {  
43        int choice;  
44        System.out.print("Enter your choice: ");  
45        try {  
46            choice = Integer.parseInt(sc.nextLine());  
47        } catch (NumberFormatException e) {  
48            System.out.println("Invalid input. Enter a number (1-3).");  
49        }  
50        return choice;  
51    }  
52  
53    private static void addStudent(Scanner sc) {  
54        System.out.print("Enter student name: ");  
55        String name = sc.nextLine();  
56        System.out.print("Enter student age: ");  
57        int age = sc.nextInt();  
58        System.out.print("Enter student grade: ");  
59        int grade = sc.nextInt();  
60        System.out.println("Student added successfully!");  
61    }  
62  
63    private static void displayAllStudents(Scanner sc) {  
64        System.out.println("Displaying all students...");  
65        // Logic to display all students goes here.  
66    }  
67}
```

- Toolbars and Status Bar:** The status bar at the bottom shows "Java: Ready" and other standard IDE status indicators.

OUTPUT

```
123 public class StudentRecordApp {  
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
===== Student Record Menu =====  
1. Add Student  
2. Display All Students  
3. Exit  
Enter your choice: 5  
Invalid choice. Please select 1, 2, or 3.  
  
===== Student Record Menu =====  
1. Add Student  
2. Display All Students  
3. Exit  
Enter your choice: 6  
Invalid choice. Please select 1, 2, or 3.  
  
===== Student Record Menu =====  
1. Add Student  
2. Display All Students  
3. Exit  
Enter your choice: 6  
Invalid choice. Please select 1, 2, or 3.  
  
===== Student Record Menu =====  
1. Add Student  
2. Display All Students  
3. Exit  
Enter your choice: 6  
Invalid choice. Please select 1, 2, or 3.
```

Explanation —

1. Purpose of the Program

The aim of the program is to build a system that can:

- Take student details from the user
- Store multiple student records
- Display all stored records
- Calculate grade based on marks

It also teaches how to structure a program using OOP principles.

◆ 2. Class Design (OOP Concept)

Person Class (Parent Class)

- This is the base class.
- It contains only one field: `name`.
- It is used to show **inheritance**, where student objects inherit the name attribute.

Student Class (Child Class extending Person)

Student inherits `name` from Person and includes:

- `rollNo` (int)
- `course` (String)
- `marks` (double)
- `grade` (char)

This demonstrates the OOP principle “**IS-A relationship**”:

A Student **is a** Person.

◆ 3. Constructors

The Student class has two constructors:

✓ Default Constructor

Initializes the object with empty or zero values.

✓ Parameterized Constructor

Allows direct initialization of:

- roll number
- name
- course
- marks

Constructors help in object creation and initialization.

◆ 4. Methods in Student Class

✓ inputDetails()

This method:

- Takes input from the user using Scanner
- Reads roll number, name, course, and marks
- Validates marks
- Calls `calculateGrade()`

✓ calculateGrade()

This method determines the grade based on marks:

Marks Range Grade

90–100	A
75–89	B
50–74	C
33–49	D
Below 33	F

This demonstrates the use of **if-else conditional statements**.

✓ displayDetails()

This method displays:

- Roll number
- Name
- Course

- Marks
- Grade

◆ 5. Data Storage Using ArrayList

The program uses:

```
ArrayList<Student> students = new ArrayList<>();
```

Reasons:

- It automatically grows as new students are added.
 - It is easier to use compared to arrays.
 - Helps store multiple objects of the Student class.
-

◆ 6. Menu-Driven Program

The program repeatedly shows a menu using a **while loop**:

1. Add Student
2. Display All Students
3. Exit

User chooses an option using **switch-case**:

- **1:** Creates a Student object and calls `inputDetails()`
- **2:** Calls `displayDetails()` for every student in the list
- **3:** Exits the program

This teaches:

- Loops
 - Switch-case
 - User input handling
-

◆ 7. Input Validation

The program ensures proper user input:

- Marks must be between **0 and 100**
- Name and course cannot be empty
- Incorrect input is handled properly

This makes the program robust.

◆ 8. OOP Concepts Demonstrated

✓ Classes and Objects

Student and Person are classes; each student record is an object.

✓ Inheritance

Student extends Person → Student inherits attributes of Person.

✓ Encapsulation

Student fields are private; accessed only through methods.

✓ Constructors

Used for object initialization.

✓ Methods

Used to input, process, and display data.

◆ 9. Control Structures

The program uses:

- **While loop** → for menu repetition
- **If-else** → for grade calculation
- **Switch-case** → for menu operations
- **For-each loop** → to display all students