1. **INTRODUCTION**

A Student Database Management System (SDMS) is a simple console-based application designed to efficiently manage student records. This project is developed using Java, focusing on basic CRUD (Create, Read, Update, Delete) operations. It provides an interactive command-line interface to store, retrieve, and manage student details such as ID, name, age, and course.

The project demonstrates fundamental Object-Oriented Programming (OOP) principles and is an excellent way for beginners to strengthen their skills in Java, data handling, and user interaction.

1. **OBJECTIVES**

The main objectives of this project are:

* To create a user-friendly system for managing student data.
* To implement basic CRUD operations without using an external database.
* To demonstrate OOP concepts such as encapsulation, classes, and objects.
* To improve data handling efficiency using Java collections.

1. **FEATURES**

This system offers the following functionalities:

* Add Student – Users can enter details such as ID, name, age, and course to register a new student.
* Display Students – Lists all stored student records in a structured format.
* Search Student – Allows users to find a specific student by ID.
* Delete Student – Removes a student record based on ID.
* Interactive Console – Simple and efficient user interaction through text-based input.

1. **METHODOLOGY**

The project follows a structured approach:

**4.1 REQUIREMENT ANALYSIS**

Language Used: Java

Development Environment: Any Java compiler (e.g., Eclipse, IntelliJ, or Command Prompt)  
 IDE used here – NetBeans

Data Storage: Handled in memory using Java ArrayLists

**4.2 SYSTEM DESIGN**

The system consists of:

* A Student class to represent individual student records.
* A main program that provides a menu-driven interface for user interaction.
* Methods to add, search, display, and delete student records dynamically.

**4.3 IMPLEMENTATION**

The program is implemented using:

* Classes & Objects for structured data management.
* ArrayList for dynamic storage of student records.
* Loops & Conditional Statements to process user input efficiently.

**4.4 ADVANTAGES**

* Easy to Use – Simple console-based interaction.
* Fast & Lightweight – Runs efficiently without requiring a database.
* Scalable – Can be expanded with features like file storage or GUI.

**4.5 APPLICATIONS**

* Can be integrated into college or school management systems.
* Serves as a foundation for larger database-driven applications.
* Helps students and beginners practice Java programming concepts.

1. **CODE SNIPPET**

import java.util.ArrayList;

import java.util.Scanner;

class Student {

int id;

String name;

int age;

String course;

Student(int id, String name, int age, String course) {

this.id = id;

this.name = name;

this.age = age;

this.course = course;

}

void display() {

System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age + ", Course: " + course);

}

}

class StudentManagement {

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

static Scanner scanner = new Scanner(System.in);

public static void main(String[] args) {

while (true) {

System.out.println("\nStudent Management System");

System.out.println("1. Add Student");

System.out.println("2. Display Students");

System.out.println("3. Search Student");

System.out.println("4. Delete Student");

System.out.println("5. Exit");

System.out.print("Enter choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1: addStudent(); break;

case 2: displayStudents(); break;

case 3: searchStudent(); break;

case 4: deleteStudent(); break;

case 5:

System.out.println("Exiting...");

return;

default: System.out.println("Invalid choice! Try again."); break;

}

}

}

private static void addStudent() {

System.out.print("Enter ID: ");

int id = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Age: ");

int age = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter Course: ");

String course = scanner.nextLine();

students.add(new Student(id, name, age, course));

System.out.println("Student added successfully!");

}

private static void displayStudents() {

if (students.isEmpty()) {

System.out.println("No students found!");

return;

}

System.out.println("\nStudent List:");

for (Student s : students) s.display();

}

private static void searchStudent() {

System.out.print("Enter Student ID to search: ");

int id = scanner.nextInt();

for (Student s : students) {

if (s.id == id) {

System.out.println("Student Found:");

s.display();

return;

}

}

System.out.println("Student not found!");

}

private static void deleteStudent() {

System.out.print("Enter Student ID to delete: ");

int id = scanner.nextInt();

for (Student s : students) {

if (s.id == id) {

students.remove(s);

System.out.println("Student deleted successfully!");

return;

}

}

System.out.println("Student not found!");

}

}

1. **LIMITATIONS & FUTURE ENHANCEMENTS**

**6.1 LIMITATIONS**

* Data is not persistent (lost when the program exits).
* No GUI (Graphical User Interface) – works only through the command line.
* Limited search functionality (only by ID).

**6.2 FUTURE ENHANCEMENTS**

* Implement file handling or database integration for persistent storage.
* Add Graphical User Interface (GUI) using JavaFX or Swing.
* Enhance search functionality with name-based or course-based filtering.

1. **CONCLUSION**

The Student Database Management System is a simple yet effective Java project that demonstrates OOP concepts and data management. It provides essential functionalities for student record handling while allowing room for expansion. This project serves as an excellent starting point for learning Java programming, data structures, and CRUD operations.