

Assignment 1: Object-Oriented Student Management System

Learning Goals

- Apply Java fundamentals: data types, variables, conditions, loops, and arrays
- Implement classes, objects, constructors, methods, and data fields
- Demonstrate encapsulation, abstraction, and data hiding using access modifiers
- Use getters and setters to enforce proper state management
- Work with arrays of objects and implement composition between classes
- Produce clean, well-structured, maintainable Java code
- Communicate findings in a structured GitHub README report with supporting screenshots
- Maintain a consistent and organized Git workflow

1) Core Class Implementations

Each student must implement the following classes based on learned topics:

Class 1: Student

Private Fields

- name — full name
- id — student ID
- major — academic program
- gpa — current GPA
- credits — total earned credits

Constructor

- Accepts name, id, major
- Initializes gpa = 0.0 and credits = 0

Required Methods

- Getters and setters for all fields
- addCredits(int c) — increases credits
- updateGPA(double newGPA) — sets GPA
- isHonors() — true if GPA ≥ 3.5
- toString() — textual representation of the student

Class 2: Course

Private Fields

- courseName — name of the course
- instructor — instructor name
- students[] — array of Student objects

Constructor

- Accepts courseName, instructor, array size

Required Methods

- addStudent(Student s, int index) — inserts into array
- courseAverageGPA() — compute average GPA
- highestCreditStudent() — return student with max credits
- toString() — return course summary

2) Implementation Requirements (Part 1)

Code Quality Standards

- Clean, readable Java code
- Proper OOP structure using encapsulation
- Meaningful variable and method names
- No direct access to private fields outside the class
- Error-handling for index issues and invalid data
- Consistent formatting and spacing

Functionality Requirements

- Correct implementation of all Student and Course methods
- Use loops and conditional logic appropriately
- Use arrays of objects in at least one scenario
- Driver program (Main.java) must:
 - Instantiate multiple Student objects
 - Modify their GPAs and credits
 - Print details via toString()
 - Create a Course object and insert students
 - Print highest-credit student and average GPA

3) Data Processing Tasks (Part 2 – Arrays & Calculations)

Student Array Processing (inside your driver program):

Using an array of at least **five Students**, compute:

- Highest GPA student
- Number of honors students
- Total credits earned

Implement these tasks as separate static methods:

```
Student getTopStudent(Student[] arr);
int countHonors(Student[] arr);
int totalCredits(Student[] arr);
```

4) Report Requirements (README.md in GitHub Repository)

Your README.md must serve as your written report and include the following:

A. Project Overview

- Summary of all classes, goals, and program purpose
- Explanation of how the project integrates learned topics

B. Class Descriptions

- Explanation of Student class fields, constructor, methods
- Explanation of Course class and its composition relationship

C. Instructions to Compile and Run

Include example commands:

```
javac *.java  
java Main
```

D. Screenshots

- Program output screenshots
- Clear demonstration of functionality

E. Reflection Section

1–2 paragraphs discussing:

- What you learned
- Challenges faced
- Benefits of encapsulation and OOP principles

5) GitHub Workflow

Your submission must be a **GitHub repository**.

Repository Structure

```
assignment1-student-management/  
└── src/  
    ├── Student.java  
    ├── Course.java  
    └── Main.java  
└── docs/  
    └── screenshots/  
        └── uml-diagram (optional)  
└── README.md  
└── .gitignore
```

Commit Storyline Example

- init: project structure and base files
- feat(student): implemented Student class
- feat(course): added Course class with composition
- feat(driver): created Main program and array processing
- docs(readme): added project overview and instructions
- perf(cleanup): code formatting and minor fixes
- release: v1.0

6) Grading Criteria

Total: 100 points

1. Student Class – 30 pts
 - Encapsulation, fields, getters/setters
 - Constructor correctness
 - Utility methods
 - `toString` implementation
2. Course Class – 20 pts
 - Composition logic
 - Required calculations
3. Main Program – 15 pts
 - Student array processing
 - Course interaction
 - Correct outputs and flow
4. Code Quality – 10 pts
 - Formatting, readability, naming, comments
 - No compilation errors
5. README Report – 20 pts
 - Overview & class descriptions
 - Instructions & screenshots
 - Reflection & structure
 - Formatting & clarity
6. GitHub Structure – 5 pts
 - Correct project organization
 - Meaningful commits