Polymorphic Academic Grading System Using Inheritance and Abstraction

Problem Context

A university evaluates students differently depending on their program type. You are required to design a system that can compute final grades polymorphically, using an abstract superclass Student and its specialized subclasses.

In addition to computing grades, the system should determine pass/fail status, apply bonus marks based on academic level, and print out formatted grade reports.

Program Requirements

1. Abstract Base Class: Student

Define an abstract class Student that includes:

- • Instance variables: String name, String studentId, String department
- • Abstract methods: double calculateFinalGrade(), String getGradeLetter()
- • Concrete methods: boolean isPassed(), String toString()

The isPassed() method returns true if final grade ≥ 60.

2. Subclasses of Student

Implement the following subclasses, each overriding the abstract methods according to their grading scheme.

1. a) UndergraduateStudent

Attributes: assignmentScore, midtermScore, finalExamScore
Formula: finalGrade = assignmentScore * 0.3 + midtermScore * 0.3 + finalExamScore * 0.4
Bonus: If the department is "CSE", add a 2% departmental merit bonus.

2. b) GraduateStudent

Attributes: projectScore, examScore, participationScore
Formula: finalGrade = projectScore * 0.5 + examScore * 0.4 + participationScore * 0.1
Bonus: Add 5% extra if projectScore ≥ 90.

3. c) PhDStudent

Attributes: publicationScore, defenseScore
Formula: finalGrade = publicationScore * 0.7 + defenseScore * 0.3
Bonus: If publicationScore ≥ 95, add 5 bonus marks (not percent).

3. Supporting Class: Course

Define a separate Course class with attributes: courseCode, title, and Student[] enrolledStudents.
Include a method printGradeReport() that calls each student's calculateFinalGrade(), getGradeLetter(), and isPassed() polymorphically, printing results in tabular format.

4. Main Program

In the main() method:
1. Create objects of UndergraduateStudent, GraduateStudent, and PhDStudent with sample data.
2. Store them in a Student[] array or ArrayList<Student>.
3. Create a Course object and pass the array.
4. Call course.printGradeReport().

5. Expected Output Example

Course: CSE 505 - Advanced Programming Paradigms

```
--------------------------------------------------
ID      Name          Type         Final Grade   Grade   Status
--------------------------------------------------
U101    Alice Rahman  Undergraduate  78.2        B       Pass
G302    Rahim Karim   Graduate       88.5        A       Pass
P501    Dr. Nabila Amin  PhD         92.0        A+      Pass
--------------------------------------------------
```

Additional Challenge Requirements

4. 1. Use polymorphism to invoke calculateFinalGrade() and getGradeLetter() on Student references.
5. 2. Use abstract class and method overriding correctly — no if/else type-checking allowed.
6. 3. Add a method to compute the class average within Course.
7. 4. Demonstrate at least one example of dynamic binding in comments or test cases.
8. 5. Implement validation so grades do not exceed 100 after applying bonuses.

Optional Extension (Extra Credit)

Add a new subclass ExchangeStudent that inherits from UndergraduateStudent and applies grading on a 4.0 GPA scale instead of 100-point scale. Override getGradeLetter() to map GPA to letter grades (A, B, C, etc.).