Develop a project management system inspired by Coursera, with modifications that utilize advanced Object-Oriented Programming (OOP) principles. The system should handle various users (students, instructors, and admins), manage course offerings, and provide tools for assignments, assessments, notifications, and collaborative group work. Some of the features of the project are provided below, rest you can add any other classes, member functions, operators, or relationships to improve the quality of the project. File handling is required for the data classes e.g. student, instructor, course etc.

1. Class Structure and Relationships

- **Person (Abstract Base Class)**
  - Attributes: Include at least three essential attributes (e.g., name, ID, email).
  - Virtual Functions: `displayInfo()`, `getRole()`, `sendNotification()`.
- **Student (Derived from Person)**
  - Attributes: Include at least three additional attributes specific to students (e.g., studentID, enrollmentDate, completedCourses).
  - Inheritance: Public inheritance from `Person`.
  - Functions: Methods to enroll in courses, submit assignments, check progress, and view certifications.
  - Associations: `Group` (for peer collaboration).
- **Instructor (Derived from Person)**
  - Attributes: Include at least three additional attributes specific to instructors (e.g., employeeID, department, assignedCourses).
  - Inheritance: Public inheritance from `Person`.
  - Functions: Create assignments, grade submissions, monitor student progress, and issue certifications.
  - Associations: `Course` (aggregation to link instructors with courses).
- **Admin (Derived from Person)**
  - Attributes: Include at least three additional attributes specific to admins (e.g., adminID, privileges, department).
  - Inheritance: Private inheritance from `Person` to restrict inherited member access.
  - Functions: Add/remove courses, assign instructors, generate reports, manage certificates, and send system notifications.
  - Associations: `Notification` (to facilitate system-wide communication).
- **Course**
  - Attributes: Includes essential details for each course (e.g., courseTitle, courseCode, description).
  - Relationships: Aggregates with `Instructor` and associates with `Student`. Composed with `Module`.
- **Module (Composition with Course)**
  - Attributes: Contains module-specific information (e.g., moduleTitle, moduleID, contentSections).

- Integral to `Course` and cannot exist independently.

- **Assignment**
  - Attributes: Holds assignment details (e.g., assignmentID, description, maxScore, dueDate).
  - Associations: With both `Student` and `Instructor` to manage submissions and grading.
- **Assessment**
  - Attributes: Manages assessment information (e.g., assessmentID, assessmentType, totalMarks).
  - Functions: Methods for students to take assessments and view scores.
- **Certificate**
  - Attributes: Certificate details (e.g., certificateID, issueDate, courseTitle, recipient).
  - Associations: Tied to `Course` and `Student` to represent course completion.
- **Notification**
  - Attributes: Notification-specific details (e.g., notificationID, message, dateSent).
  - Aggregation: With both `Student` and `Instructor` for broadcasting updates.
- **Group (Static Class)**
  - Attributes: Static group information for collaborative purposes (e.g., groupID, members).
  - Static Functions: Methods to manage study groups, handle peer discussions, and facilitate student collaboration.

2. Operator Overloading

- **Standard Operators**
  - == for equality comparison across all classes.
  - = for deep copy functionality in all classes.
  - << and >> for streamlined input and output for all classes.
  - `[]`: Allows indexed access to data members (e.g., `student[1]` for `gpa`).
  - `()`: Used to access a specific data member by name (e.g., `student(gpa)`).
- **Custom Operators**
  - `+` and –: Used for adding/removing `Student` objects from `Course` rosters, and managing instructor-course assignments.
  - Union and Intersection (`|` and `&`): `Student1 | Student2` returns all courses of both students; `Student1 & Student2` finds courses both students share.
  - Additional operators as necessary.

3. Project Deliverables

- **Class Diagrams and UML**
  - Provide a UML diagram with detailed class hierarchies, inheritance types, and clear visualizations of associations, aggregations, and compositions.
- **Code Implementation**

  - Develop fully functional classes based on the specifications above, with implemented methods and overloaded operators.
- **Documentation**
  - Include comprehensive comments within the code, particularly for overloaded operators, static members, and inheritance types.

Evaluation Rubrics/ Self-Assessment Report

Rubrics	Max Marks	Student 1			Student 2		
		Self-Assessment	Marks by Evaluator	Comments	Self-Assessment	Marks by Evaluator	Comments
Classes (Constructors, accessors, mutators)	12						
Const	3						
Static	3						
Copy Constructor	6						
Aggregations	3						
Compositions	3						
Associations	4						
Inheritance	6						
Polymorphism	5						
Overloaded operators (12*6)+20	10						
Destructors	5						
Pointers	10						
File Handling	5						
Bonus	5						

The decision for the bonus marks will be made by the evaluator at time of evaluation.

Student 1

Name: _________________

Roll Number: _______________

Student 2

Name: _________________

Roll Number: _______________