COSC 113: Computer Science II
Final Group Project Requirements
3/31/2025 to 5/12/2025 [6 weeks]

Objective:
To demonstrate proficiency in Java programming principles and collaborative software development practices through the creation of a substantial application.

Requirements:

- Teams shall be comprised of two students each.
- The project must be developed utilizing Java 8 or a more recent version.
- GitHub shall be employed for version control, with strict adherence to established branching best practices, including the utilization of feature branches and pull requests.
- The project's design and implementation must comprehensively demonstrate the application of all four core principles of object-oriented programming: **encapsulation, inheritance, polymorphism, and abstraction**.
- The implementation must clearly show the utilization of either the **Heap, Stack or Queue** and provide an explanation of how it is being used.
- A comprehensive code report, detailing the project's design, implementation, and functionality, is required. This report must include neat and concise **UML class diagrams** to visually represent the project's structure. The code report must explicitly delineate the specific **contributions of each team member** to the project's development.
- Thorough **test cases** must be implemented to ensure the robustness and correctness of the developed code.
- A professional **presentation** showcasing the project's features, functionality, and implementation details is required.
- The project should demonstrate mastery of advanced Java concepts such as file input/output and exception handling.

Sample Project Idea: Library Management System

Description: Develop a Java application to manage a library's resources, including books, members, loans, and reservations.

Demonstration of Requirements:

- **Team Formation:** Teams of 2 students collaborate.
- **Java Version:** The project utilizes Java 17.
- **Version Control:** A GitHub repository is established, with feature branches for tasks like "add-book," "loan-processing," "member-registration," and "reservation-system," and pull requests for code review.
- **Object-Oriented Programming (OOP):**
    - **Encapsulation:** The Book class encapsulates book details (title, author, ISBN).
    - **Inheritance:** ReferenceBook and FictionBook inherit from Book, adding specific attributes.
    - **Polymorphism:** A displayDetails() method is overridden in subclasses for different book information displays.
    - **Abstraction:** An LibraryItem interface defines common methods for library items (books, DVDs).
- **Memory Management:** The Heap is used for dynamic object creation (e.g., Book, Member, Loan). The code report will explain how objects are being stored in the heap, and how the garbage collector will remove objects when they are no longer needed.
- **Documentation:** The code report includes UML class diagrams, design explanations, and implementation details.
- **Testing:** JUnit tests are implemented for critical functionalities like book search, loan processing, member registration, and reservation handling.
- **Individual Contribution:** The code report details each member's contributions (e.g., database integration, GUI development, testing, and report writing).
- **Presentation:** A live demonstration of the library management system is presented.
- **Advanced Java Concepts:** Use of collections, file I/O, exception handling, and potentially database interaction is expected.

Other Project Ideas:

The following are provided as exemplary project proposals. Participants may select one of these, or alternatively, propose an original project for consideration.

- **Library Management System**
  → Use classes like Book, Member, Librarian, with interfaces like Borrowable, and abstract classes like Person.

- **Online Course Enrollment Platform**
  → Involves Course, Student, Instructor, and behaviors through interfaces like Payable, and abstract class User.

- **Banking System Simulation**
  → Abstract class Account, subclasses like SavingsAccount, CheckingAccount, interface Transaction, and composition with Customer.

- **Vehicle Rental System**
  → Class hierarchy: Vehicle → Car, Bike, Truck, with aggregation between Customer and Rental.

- **E-Commerce Shopping Cart**
  → Composition between Cart and Product, interfaces like Discountable, and abstract User class with Admin and Customer.

- **Employee Payroll System**
  → Abstract Employee class, subclasses like FullTimeEmployee, PartTimeEmployee, interface Taxable.

- **Hospital Management System**
  → Classes like Doctor, Patient, Appointment, interface Schedulable, aggregation between Doctor and Specialization.

- **Smart Home Device Manager**
  → Interface Controllable, abstract class Device, subclasses Light, Thermostat, composition with Room.

- **University Course Management System**
  → Classes: Student, Professor, Course; interface Gradable, abstract class Person.

- **Zoo Management System**
  → Abstract class Animal, subclasses Mammal, Bird; interface Feedable, aggregation between Zookeeper and Animal.

- **Online Food Ordering App**

→ Restaurant, MenuItem, Order, Customer, interface Deliverable, abstract User class.

- **Flight Reservation System**
  → Classes like Flight, Passenger, Reservation, interface Bookable, composition with FlightSeat.

- **Hotel Room Booking System**
  → Abstract class Room, subclasses like Suite, SingleRoom, interface Bookable, aggregation with Customer.

- **Social Media Platform Prototype**
  → Classes: User, Post, Comment; interface Shareable, Reportable, with nested composition for comments.

- **Online Quiz/Test System**
  → Abstract class Question, subclasses MCQ, TrueFalse, interface Scorable, aggregation with Test.