Coding Assignments: Dependency Injection

1. Dependency Injection & Inversion of Control

Assignment Set

1. Define a Notification Interface and Implementation

- Create a `NotificationService` interface with a `send(to: string, message: string): Promise` method.
- Implement both `SMSService` and `EmailService` classes using `@Service()` and the interface.

2. Use Constructor Injection in a Service

- Write an `AppointmentService` class that receives a `NotificationService` via constructor injection.
- Use the injected service to send a booking confirmation message.

3. Swap Implementations at Runtime

- Using TypeDI's `Container.set`, register `EmailService` as the implementation for `NotificationService`.
- Book an appointment and verify that the email message is logged.

4. Test with a Mock Service

- Create a `MockNotifier` class implementing `NotificationService` that logs messages to an array.
- Write a test that injects this mock and asserts that messages are recorded when booking an appointment.

5. Add and Inject a Billing Service

- Define a `BillingService` interface and a `StripeBillingService` implementation.
- Inject `BillingService` into `AppointmentService` and charge the patient when booking.

---

2. MVC Pattern & Modular Design

Assignment Set

1. Define a Book Model

- Create a `Book` interface with fields: `id`, `title`, `author`, `isBorrowed`.

2. Implement a Repository Interface

- Write an `IBookRepository` interface with methods: `findAll()`, `findById(id)`, `save(book)`.

3. Build an In-Memory Repository

- Implement `InMemoryBookRepository` that stores books in an array and fulfills the interface.

4. Create a Book Service

- Write a `BookService` class that uses `IBookRepository` to implement `borrowBook(bookId: string)` with business rules:
  - Throw an error if the book is not found or already borrowed.

5. Write a Book Controller

- Implement a `BookController` class with a `borrowBook(req, res)` method that calls the service and handles errors.

---

3. Repository Pattern

Assignment Set

1. Define a Course Domain Model

- Create a `Course` interface with `id`, `name`, `capacity`, and `students` (array of IDs).

2. Write a Repository Interface

- Define `ICourseRepository` with methods: `findAll`, `findById`, `save`, `enrollStudent`, `findByStudentId`.

3. Implement an In-Memory Repository

- Implement `InMemoryCourseRepository` with all interface methods, storing courses in an array.

4. Implement a Database Repository (Stub)

- Create a `DatabaseCourseRepository` class with stubbed methods for database operations.
