✅ 1. OOP Concepts Overview

🎯 Direct Challenges:

1. Create a class with properties and a method to display details.
2. Demonstrate class instantiation and method invocation.
3. Use getters and setters to access private data members.

🧩 Scenario-Based Challenges:

1. Build a `Book` class and create objects to store different book information.
2. Create a `BankAccount` class and show deposit and withdrawal actions.

✅ 2. Classes and Objects

🎯 Direct Challenges:

1. Create a `Student` class with name, age, and grade attributes.
2. Define a method inside a class to display details.
3. Instantiate multiple objects and show their data.

🧩 Scenario-Based Challenges:

1. Create a class `Employee` with a method to calculate salary based on hours worked.
2. Write a `Car` class and display its specifications using a method.

✅ 3. Inheritance

🎯 Direct Challenges:

1. Create a base class `Animal` and a derived class `Dog`.
2. Use `super` keyword to access parent class constructor.
3. Override a method from parent class in child class.

🧩 Scenario-Based Challenges:

1. Build a class `Vehicle` and extend it to `Truck`, `Car`, and `Bike`.
2. Create a class `Shape` with area method and extend it for `Circle`, `Rectangle`.

✅ 4. Polymorphism

🎯 Direct Challenges:

1. Demonstrate method overloading with different parameters.
2. Show method overriding in child class.
3. Use `instanceof` to check the type at runtime.

🧩 Scenario-Based Challenges:

1. Create a `Payment` class and override `pay()` in `CreditCard`, `Cash` subclasses.
2. Build a `Notification` class and implement different behaviors for `Email`, `SMS`, `Push`.

✅ 5. Abstraction

🎯 Direct Challenges:

1. Create an abstract class `Shape` with an abstract method `draw()`.
2. Implement the abstract class in `Circle` and `Square`.
3. Show partial abstraction using non-abstract and abstract methods.

🧩 Scenario-Based Challenges:

1. Define a class `Employee` with abstract method `calculateSalary()` and implement in `FullTime` and `PartTime` subclasses.
2. Create an abstract `Appliance` class and implement it for `Fan` and `AC`.

✅ 6. Encapsulation

🎯 Direct Challenges:

1. Create a class with private fields and public getters/setters.
2. Demonstrate how encapsulation protects data.
3. Prevent setting negative values to `age` field using validation in setter.

🧩 Scenario-Based Challenges:

1. Create a `Patient` class that restricts direct access to `medicalHistory`.
2. Build a `LoanAccount` class and encapsulate the balance with setter restrictions.

✅ 7. Interfaces

🎯 Direct Challenges:

1. Create an interface `Drawable` with method `draw()`.
2. Implement the interface in class `Circle`.
3. Create another interface `Colorable` and implement both interfaces in a single class.

🧩 Scenario-Based Challenges:

1. Design an interface `Payable` and implement it in classes `Invoice` and `Employee`.
2. Create an interface `Database` with `connect()` method and implement for `MySQL` and `Oracle`.

✅ 8. Composition

🎯 Direct Challenges:

1. Create a class `Engine` and use it in class `Car`.
2. Use composition to build a `Computer` with `Processor`, `RAM`, and `HardDrive` objects.
3. Demonstrate "has-a" relationship using class `Library` with `Book` objects.

🧩 Scenario-Based Challenges:

1. Model a `Student` having an `Address` and `IDCard` as composed objects.

2. Create a `House` class that has `Room` and `Kitchen` as components.

✅ 9. Overloading and Overriding

🎯 Direct Challenges:

1. Overload a method `add()` with two and three parameters.
2. Override `toString()` method of a custom class.
3. Override a `display()` method from base class in child class.

🧩 Scenario-Based Challenges:

1. Create a `Logger` class with overloaded `log()` methods for different data types.
2. Build a `Vehicle` class with overridden `move()` in `Car` and `Bike` subclasses.

✅ 10. Aggregation

🎯 Direct Challenges:

1. Create `Department` and `Student` classes showing aggregation.
2. Model a `Team` that contains players as aggregated objects.
3. Illustrate aggregation with `Teacher` and `Subject`.

🧩 Scenario-Based Challenges:

1. Build a `University` class that aggregates multiple `College` objects.
2. Create a `Hospital` with a list of `Doctor` objects (not tightly bound).