1. **Conceptual Understanding**:
    - Explain the difference between procedural programming and object-oriented programming.
    - Define the four pillars of OOP: Abstraction, Encapsulation, Inheritance, and Polymorphism.
    - Give an example of where abstraction is applied in real-life scenarios.
2. **Abstraction**:
    - Create an abstract class called `Animal` with methods `makeSound()` and `move()`. Implement subclasses `Bird` and `Fish` to define these methods.
3. **Encapsulation**:
    - Write a `BankAccount` class with private fields for `accountNumber`, `holderName`, and `balance`. Add getter and setter methods to access and modify these fields while validating that the balance cannot be negative.
4. **Inheritance**:
    - Create a class `Vehicle` with methods `start()` and `stop()`. Derive two classes `Car` and `Motorcycle` with an additional method specific to each class.
5. **Polymorphism**:
    - Implement method overloading in a `Calculator` class for `add()` that works with two integers, two doubles, and a single number that's being used twice.
    - Demonstrate method overriding by creating a parent class `Shape` with a method `draw()` and overriding it in subclasses `Circle` and `Square`.
6. **Constructors**:
    - Define a `Student` class with a default constructor that sets default values for `name` and `rollNumber`, and a parameterized constructor to initialize them with user-provided values.

7. **`this` and `super` Keywords**:
    - Write a `Person` class with a constructor that initializes the `name` field. Use the `this` keyword to differentiate between the constructor parameter and the class field.
    - Extend the `Person` class into a `Teacher` class and use the `super` keyword to call the parent class constructor.
8. **Getters and Setters**:
    - Add getter and setter methods to the `Student` class from the Constructors task and demonstrate their usage.
9. **Access Modifiers**:
    - Create a `Library` class to demonstrate the use of `public`, `protected`, `private`, and default access modifiers for fields and methods.

Hands-On Practice:

Project 1: Vehicles

Objective: Develop a simple vehicle management program using object-oriented programming. This project focuses on inheritance, method overriding, and class hierarchy.

Requirements:

1. **Base Class: `Vehicle`**
    - Add attributes like `brand` and implement methods `start()` and `stop()`.
2. **Derived Classes:**
    - `Car`: Extend the `Vehicle` class. Add attributes like `model` and a method `play_music()`. Implement a `details()` method to display car-specific information.
    - `Motorcycle`: Extend the `Vehicle` class. Add attributes like `cc` (engine capacity) and implement a `details()` method.
    - `Truck`: Extend the `Vehicle` class. Add attributes like `capacity` (in tons) and implement a `details()` method.
3. **Demonstration:**
    - Create objects of `Car`, `Motorcycle`, and `Truck`.
    - Call methods to start and stop each vehicle and display specific details.

In the end:

- A program that demonstrates the functionality of all vehicle types.
- Output showing interactions with each type of vehicle.

Project 2: Shape Hierarchy and Composite Shapes

Objective: Create a program that models different geometric shapes using object-oriented principles. This project emphasizes inheritance, abstraction, and composite structures to handle complex systems.

Requirements:

1. **Abstract Base Class: `Shape`**
    - Define two abstract methods: `area()` and `perimeter()`.
2. **Second Layer: `TwoDimensionalShape`**
    - Extend the `Shape` class and add an abstract method `dimensions()` to describe the shape's specific properties (e.g., radius, length, width).
3. **Concrete Classes:**
    - `Circle`: A class to represent a circle. Accept radius as an argument and implement `area()` and `perimeter()`.
    - `Rectangle`: A class to represent a rectangle. Accept length and width as arguments and implement `area()` and `perimeter()`.
4. **Composite Shape:**
    - Create a `CompoundShape` class that can store multiple shapes (e.g., circles and rectangles) and calculate the total area and perimeter of all stored shapes.
    - Use an inner class to handle individual shapes within the composite shape.
5. **`Three Dimensional Shapes`……**

In the end:

- A program that demonstrates the functionality of all shapes.
- Examples of adding shapes to a `CompoundShape` and calculating the total area and perimeter.