Encapsulation:

1. Implement a **Circle** class with private attributes for the radius. Include methods to set and get the radius, and a method to calculate the area. Ensure that the radius is always positive.

2. Create a **Student** class with private attributes for name, age, and grade. Include methods to set and get these attributes, and ensure that the age and grade are within valid ranges.

3. Design a **TemperatureConverter** class with private attributes for Celsius and Fahrenheit temperatures. Include methods to set and get these attributes, and ensure that both temperatures are in sync based on the conversion formula.

Abstraction:

1. Create an abstract class **Shape** with an abstract method **calculateArea()**. Implement two concrete subclasses, **Circle** and **Rectangle**, providing the area calculation for each.

2. Create an abstract class **Bank** with an abstract method **calculateInterest()**. Implement two subclasses, **SavingsAccount** and **FixedDeposit**, providing their interest calculation formulas.

3. Create an abstract class **Person** with abstract methods **getFullName()** and **displayInfo()**. Implement a concrete subclass **Employee** with fields **firstName, lastName**, and **employeeId**, providing implementations for the abstract methods.

Inheritance:

1. Create a base class **Shape** with methods like **area()** and **perimeter()**. Then, create subclasses like **Circle**, **Rectangle, and Triangle** that inherit from the Shape class and provide their own implementations of the **area() and perimeter()** methods.

2. Model a hierarchy of bank accounts with a base class **BankAccount** and derive classes like **SavingsAccount** and **CheckingAccount**. Each derived class can have specific methods like **calculateInterest** for SavingsAccount and **deductMonthlyFee** for **CheckingAccount**.

Polymorphism:

1. Create a base class **Animal** with a method **makeSound()**. Create two subclasses **Dog and Cat** that override the **makeSound()** method.

2. Design a class **MathOperations** with overloaded methods for basic mathematical operations. Implement methods for **addition, subtraction, multiplication, and division for different data types (int, double)**.

3. Create a class **TimeConverter** with overloaded methods for converting time between different units. Implement methods for converting **seconds to minutes, minutes to hours, and hours to days**.

Interface:

1. Create an interface **Movable** with a method **move()**. Create another interface **Resizable** with a method **resize()**. Create a class **Shape** that implements both **Movable and Resizable**. Write a program to demonstrate how an instance of the Shape class can be moved and resized.

2. Create an interface **MathOperations** with static methods **add(int a, int b)** and **subtract(int a, int b)**. Implement this interface in a class **Calculator**. Write a program to demonstrate the use of the static methods.

3. Create an interface **Constants** with constant variables **PI and SPEED_OF_LIGHT**. Implement this interface in a class **PhysicsConstants**. Write a program to access and print these constant values.

OOPs Concept:

1. Create a class **StringManipulator** with overloaded methods to concatenate two strings. Overload the method for different types of inputs, such as two strings, a string and an integer, and an integer and a string.

2. Create a class **SortingHelper** with overloaded methods to sort arrays. Include methods to sort arrays of integers and strings.

3. Create a base class **Person** with properties such as firstName, lastName, and age. Include a method getFullName that returns the full name of the person. Then, create two derived classes Student and Teacher that inherit from the Person class. Add specific properties and methods to each derived class related to their roles. Finally, create instances of both Student and Teacher and demonstrate the use of inheritance by invoking methods.

4. Create a base class **Shape** with properties such as **color and a method calculateArea** that returns 0. Then, create two derived classes **Circle and Rectangle** that inherit from the Shape class. Add specific properties and methods to each derived class to calculate the area. Finally, create **an array of Shape objects** containing both circles and rectangles, and demonstrate the use of inheritance by invoking methods.

5. Create a base class **Person** with a method **displayInfo**. Then, create two derived classes **Student and Teacher** that inherit from the **Person** class. Each derived class should provide its own implementation of the **displayInfo** method. Finally, create an array of **Person objects**, including instances of both **Student and Teacher**, and demonstrate polymorphism by invoking the **displayInfo** method for each person.

6. Create a class **Library** with encapsulated properties books (an array of Book objects) and methods to add a book to the library and display information about all books. The Book class should have properties like title, author, and isbn. Use encapsulation to ensure proper access control for these properties.