Test your Knowledge on OOP

Complete the following exercises individually and put into practice the concepts you have learned in the support classes. Remember, the only difference between a master and a novice is that the master has failed more times than the novice has even tried.

Question 1

Task: Create a class hierarchy where a `Person` class is the superclass, and `Student` and `Teacher` are subclasses.

1. The `Person` class should have common fields like `name` and `age`, and a constructor to initialize them.
2. The `Student` class should have an additional field `grade` and a constructor to initialize all fields, calling the superclass constructor where needed.
3. The `Teacher` class should have an additional field `subject` and a constructor that also calls the superclass constructor.

Requirement: Write a main method to create instances of `Student` and `Teacher` and display their details using a method called `displayInfo()` in each subclass.

Question 2

Task: Create a `Vehicle` class with a default constructor that prints "Vehicle is created". Then, create a `Car` class that extends `Vehicle`.

- In the `Car` class, define a constructor that calls the `Vehicle` constructor using `super` and then prints "Car is created".

Requirement: Instantiate the `Car` class in the main method and observe the output.

Question 3

Task: Implement an interface `Playable` with a method `play()`. Create two classes, `Football` and `Basketball`, that implement the `Playable` interface.

- In the `Football` class, implement the `play()` method to print "Playing Football".
- In the `Basketball` class, implement the `play()` method to print "Playing Basketball".

Requirement: Write a main method that creates an array of `Playable` type and populates it with both `Football` and `Basketball` objects. Use a loop to call the `play()` method on each object.

Question 4

Task: Create an abstract class `Animal` with an abstract method `makeSound()` and a concrete method `sleep()` that prints "Animal is sleeping".

- Create two subclasses `Dog` and `Cat` that extend `Animal` and provide implementations for the `makeSound()` method.

Requirement: Write a main method that demonstrates creating instances of `Dog` and `Cat`, calling both the `makeSound()` and `sleep()` methods on each object

Question 5

Task: Create an abstract class `Shape` with an abstract method `draw()` and a concrete method `getType()` that returns the type of shape.

- Define an interface `ThreeDimensional` with a method `calculateVolume()`.
- Create two classes `Cube` and `Sphere` that inherit from `Shape` and implement the `ThreeDimensional` interface. Provide implementations for the `draw()` method and `calculateVolume()` method.

Requirement: Write a main method to create instances of `Cube` and `Sphere`, call all their methods, and print the results

Question 6

Task: Design a small library system where:

- Create an abstract class `LibraryItem` with fields like `title`, `author`, and an abstract method `displayDetails()`.
- Create subclasses `Book`, `Magazine`, and `DVD` that inherit from `LibraryItem` and provide implementations for the `displayDetails()` method.
- Use polymorphism to create an array or list of `LibraryItem` type and populate it with instances of `Book`, `Magazine`, and `DVD`.

Requirement: Write a method to display details of all items in the list, demonstrating the use of polymorphism

Question 7

Task: Create a class `Employee` with fields `name` and `salary` and a constructor to initialize them. Then, create a class `Manager` that extends `Employee` with an additional field `department`.

- Use the `super` keyword in the `Manager` constructor to initialize the inherited fields.

Requirement: Write a main method to create an instance of `Manager` and display all its details, demonstrating the use of `super`.

Question Bonus Challenge

Task: Create an abstract class `Appliance` with an abstract method `turnOn()` and a concrete method `turnOff()`.

- Define an interface `Rechargeable` with a method `charge()`.
- Create classes `WashingMachine` and `ElectricCar` that inherit from `Appliance` and implement `Rechargeable`.

Requirement: Write code to demonstrate the use of polymorphism, calling methods from both the abstract class and the interface.