**EXERCISE – 1** : Problem Statement on Design patterns

1. Two use cases to demonstrate two behavioural design pattern.

These are about how objects interact and communicate.

1. **Strategy Pattern (Choosing Payment Method)**

* Problem: A shopping app can let users pay by Credit Card, UPI, or Wallet.
* Solution: Define a common PaymentStrategy interface and let user choose which strategy to apply.
* Benefit: Easy to add new payment methods later without changing the main code.

1. **Observer Pattern (Weather Station Notifier)**

* Problem: A weather station updates temperature, and multiple devices (Mobile, TV, Smartwatch) should get notified automatically.
* Solution: WeatherStation is Subject, devices are Observers. Whenever temperature changes, all devices get notified.
* Benefit: Loose coupling, automatic updates.

1. Two use cases to demonstrate two creational design pattern.

These deal with object creation logic.

1. **Singleton Pattern (App Configuration Manager)**

* Problem: A system should have only one instance of configuration (like database URL, API keys).
* Solution: Singleton ensures only one ConfigManager object exists.
* Benefit: Saves memory, provides a global access point.

1. **Factory Pattern (Shape Creator)**

* Problem: A drawing app creates different shapes (Circle, Rectangle, Triangle) based on user input.
* Solution: A ShapeFactory creates objects without exposing creation logic to the user.
* Benefit: Centralized creation, easy to manage.

1. Two use cases to demonstrate two structural design pattern.

These are about how classes and objects are combined.

1. **Adapter Pattern (Old Printer with New USB Port)**

* Problem: A new laptop has only USB-C, but your printer works on old USB.
* Solution: Adapter class converts requests between incompatible interfaces.
* Benefit: Reuse old system without modifying it.

1. **Decorator Pattern (Coffee with Add-ons)**

* Problem: A coffee shop sells plain coffee, but customers may want sugar, milk, or cream.
* Solution: Use Decorator to wrap a Coffee object with extra features dynamically.
* Benefit: Flexible, avoids multiple subclasses for every combination.