Design Patterns

CREATIONAL, STRUCTURAL, BEHAVIORAL PATTERNS

Introduction to Design Patterns

- Design patterns are proven solutions to common software design problems.
- They are essential tools in object-oriented design and provide a standard vocabulary.

Overview of Design Patterns Categories

- Design Patterns can be categorized into three main types:
  - 1. Creational Patterns
  - 2. Structural Patterns
  - 3. Behavioral Patterns

- We'll explore examples of each pattern and their use cases.

Creational Design Patterns

- Creational patterns deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
- Examples:
  - - Singleton
  - - Factory Method
  - - Abstract Factory
  - - Builder

Singleton Pattern (Creational)

- Ensures a class has only one instance and provides a global point of access.
- Use Case: Database connections, logging services, configuration management.

Code Example:

```
class Singleton {
    private static Singleton instance;
    private Singleton() {}
    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Structural Design Patterns

- Structural patterns deal with object composition, ensuring that complex structures can be built using simple components.
- Examples:
  - - Adapter
  - - Decorator
  - - Proxy
  - - Composite

Adapter Pattern (Structural)

- Allows incompatible interfaces to work together by creating an adapter that converts one interface to another.
- Use Case: Integrating legacy systems or third-party libraries.

Code Example:

```
class PayPalAdapter implements PaymentProcessor {
    private PayPalAPI payPalAPI;
    public PayPalAdapter(PayPalAPI payPalAPI) {
        this.payPalAPI = payPalAPI;
    }
    @Override
    public void processPayment(double amount) {
        payPalAPI.makePayment(amount);
    }
}
```

Behavioral Design Patterns

- Behavioral patterns focus on communication between objects, helping to define how they interact with one another.
- Examples:
  - - Observer
  - - Strategy
  - - Command
  - - Chain of Responsibility

Observer Pattern (Behavioral)

- Defines a one-to-many dependency where one object (subject) notifies other objects (observers) about state changes.
- Use Case: Real-time notifications (e.g., stock price updates, user activity).

Code Example:

```
class StockPriceSubject {
   private List<StockPriceObserver> observers = new ArrayList<>();
   public void addObserver(StockPriceObserver observer) { observers.add(observer); }
   public void setStockPrice(double price) { notifyObservers(price); }
   public void notifyObservers(double price) { observers.forEach(o -> o.update(price)); }
}
```

Adapter + Observer in Action (Real-World Use Case)

- Combining Adapter and Observer patterns:
- Use Adapter to integrate different systems (payment gateways), and Observer to notify users about the status of their transactions.
- Real-world Example: Payment system with notifications (PayPal, Stripe).

Benefits of Design Patterns

- - Promotes code reuse, maintainability, and flexibility.
- - Helps create scalable and extensible systems.
- - Provides a common vocabulary for developers.

When to Use Design Patterns

- - When you face a common recurring problem.
- - To promote best practices and design principles.
- - To make code more adaptable and easier to maintain.

Summary and Conclusion

- Design patterns are essential tools for building efficient, scalable systems.
- The Singleton, Adapter, and Observer patterns are just a few of the key patterns in software design.
- Understanding these patterns helps developers build **maintainable** and **extensible** applications.

Thank You!

- Thank you for your valuable time!