AKETCH MARY FRANCES

S20B23/215

DESIGN PATTERNS

Types of Design Patterns

1. Creation Patterns: Provide a way to create objects by hiding the creation logic instead of instantiating objects directly using the new operator. This gives the program more flexibility in deciding which objects to create for a specific use case.

2. Structural Patterns: These design patterns relate to the composition of classes and objects. The concept of inheritance is used to create interfaces and define ways to create objects to obtain new functionality

3. Behavioral Patterns: 3. These design patterns relate specifically to the communication between objects, their interaction, and the distribution of responsibilities between them

4. J2EE Patterns: These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

Why Design Patterns are needed

Clients accessing a complex subsystem directly reference or depend on a large number of objects with vastly different interfaces. This makes clients particularly difficult for developers to deploy, customize, test, and reuse. This is where a facade design template can come in handy.

1. Proven Solution
2. Reusable(modified to solve many kinds of problems)
3. Expressive(an elegant solution)
4. Prevent the Need for Refactoring Code
5. Lower the Size of the Code-base

When used well, design patterns can both speed up the development process and generally reduce the chance of errors.

Facade Design Pattern

The facade design pattern is a "structural" design pattern that helps provide a single interface (class) to access many different codes/objects. Facade hides the complexity of various subsystems (often organized into classes) with a simple interface. The facade model defines an exemplary solution for the simple connection of different interfaces in complex systems. The universal facade class, which also acts as an interface, delegates important software functions tosuitable subsystems in order to simplify the functioning of the various program part components as much as possible.For example, an e-commerce customer wants a single point of interaction with the brand instead of individual communication (interface) with each sales support system such as product inventory, authentication, security, payment processing, order fulfillment, etc.In this case, Fasada encompasses all activities and "control" systems to provide a single interface: the customer remains completely in the dark about what is happening behind the scenes. Facade is an important concept that supports a loosely coupled microservices architecture.

Problems that the facade pattern addresses

Clients that access a complex sub-system refer directly to a large number of objects with completely different interfaces or are dependent on these objects. This makes the implementation, adaptation, testing, and reuse of the clients particularly difficult for developers. This is where the facade design pattern can be useful.

The facade design pattern defines a central facade object that:

- implements a universal interface for the various interfaces of the sub-system(s).
- and (if necessary) can perform additional functions before or after forwarding a client request.

As an intermediary, the facade object ensures that access and communication with the individual components of a subsystem are simplified and direct dependence on these components is minimized. It delegates the client calls so clients don’t need to know the classes or their relationships and dependencies.