

NAME : NANDAWULA MARIA

REG NO : S20B23/207

ACCESS NO : A90651

COURSE : BSCS

COURSE UNIT : SOFTWARE CONSTRUCTION

LECTURER : MR. SIMON FRED LUBAMBO

Write short notes about design patterns

Design patterns are solutions to general problems that software developers faced during software development.

**Types of Design Patterns**

1. **Creational Patterns**: Provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case
2. **Structural Patterns:** These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities
3. **Behavioral Patterns**: These design patterns are specifically concerned with communication between objects, how they interact and how responsibilities are allocated 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**

Design patterns offer a best practice approach to support object-oriented software design, which is easier to design, implement, change, test and reuse. These design patterns provide best practices and structures.

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 Codebase

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 one interface (class) for access to a large body of code / various objects. A facade hides complexities of various sub-systems (often organized into a class) with a simple interface. The facade pattern defines a sample solution for the simple merging of different interfaces in complex systems. A universal facade class, which also functions as an interface, delegates important functionalities of the software to the respective sub-systems in order to make handling the various sub-components of a program as simple as possible.

For example, an eCommerce customer only wants one point of interacting with a brand, rather than individually communicating (interfacing) with each system to support the sale such as product inventory, authentication, security, payment processing, order fulfillment, etc. In this case, the Facade has encapsulated all the “order” activities and systems to provide a single interface – the customer remains completely unaware of what’s going on behind the scenes. Facade is an important concept to support the 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.