Design Patterns

Design patterns are specific solutions to common problems in software design. Each pattern is like a blueprint the developer can customize to solve a particular design problem in their code.

Benefits of patterns

Patterns are a toolkit of solutions to common problems in software design. They define a common language that helps their team communicate more efficiently.

Categories of Design Patterns:

1. **Creational**

Creational Design Patterns assist the creation of objects.

2. **Structural**

Structural Design Patterns assist relationships between objects and classes to

form a larger structure.

3. **Behavioural**

Behavioral Design Patterns assist patterns of communications between objects.

Creational Design Patterns

Singleton

- Easy understandable
- Type of object that can only be instantiated once

A Singleton class creates a static creation method that acts as a constructor and makes its constructor private so it cannot be instantiated with a new keyword. Furthermore, it provides a global access point to this instance.

Prototype

- Another word for clone

With Prototype patterns, existing objects can be copied without making the code dependent on their classes. In addition, it makes it much easier to share functionality between objects.

Builder

With Builder patterns, objects will create step by step using a method rather than the constructor and building logic can be delegated to an entirely different class.

Factory

Factory patterns use a method instead keywords to instantiate an object. They allow subclasses to change the type of objects that will be instantiated. It is excellent for a cross-platform app or logistic management application.

Structural Design Patterns

Facade

A facade is the face of a building, whereby the inside is hidden, and the user does not need to know what is inside. It is a simplified API to hide other low-level details in the code base.

Proxy

- Another word for substitute

The Proxy design pattern controls access to the original object. It allows one to perform something before or after the request gets to the original object. So, for example, the object will only be created if the client needs it. Excellent for large objects.

Behavioural Design Patterns

Iterator

The Iterator pattern allows the developer to traverse through a collection of objects. Modern languages already provide abstractions for the iterator pattern like the for loop.

Observer

- A push-based system

The Observer is a behavioural pattern that allows the developer to specify a subscription structure to notify multiple objects of events that happen to the object they are observing. In the real world, an observer can be a tower that sends live updates or notifications to all

receivers simultaneously. It is used all over the place in app development. For example, when the data is changed in the server, all clients will get an update simultaneously.

Mediator

The Mediator design pattern creates a reliable mediator for object coordination and communication. For example, in the real world, it can be a traffic control tower at an airport, whereby the tower is responsible for coordinating the runway and communicating with the aeroplanes.

State

A state design pattern uses conditionals to handle a bunch of different possibilities based on the state or data in the application. It allows the developers to start with one base class and then provides it with different functionality based on its internal state. Every condition has its class, and the object will behave entirely differently when the state changes. At the same time, the API does not have to change.

References

Refactoring (2022) Design Patterns. Available from: https://refactoring.guru/design-patterns [Accessed 02 September 2022].