Design Patterns

Arce Llamas Irvin de Jesus

Multiplatform Software Development
Universidad Tecnológica de Tijuana
4A 06/02/24

Design Patterns

Design patterns are essential tools in software engineering. They provide reusable solutions to common problems encountered during application or system design. Rather than prescribing finished designs, they offer templates for solving specific issues within a given context. Let's explore some key points about design patterns:

Types of Design Patterns

- Creational Patterns: These focus on object creation mechanisms. Examples include the Singleton, Factory Method, and Builder patterns.
- Structural Patterns: These deal with the composition of classes and objects. Notable examples are the Adapter, Decorator, and Facade patterns.
- Behavioral Patterns: These address how objects interact and communicate. Some well-known behavioral patterns include the Observer, Strategy, and Command patterns.

Origin and Popularity

- Design patterns originated as an architectural concept by Christopher Alexander in the late 1970s.
- In the field of programming, Kent Beck and Ward Cunningham experimented with applying patterns to code during the late 1980s.
- The influential book "Design Patterns: Elements of Reusable Object-Oriented Software" (often referred to as the "Gang of Four" book) was published in 1994. It introduced design patterns to a wider audience and remains a classic reference.

Application and Usage

- Design patterns are best practices that guide programmers in solving recurring problems.
- They are particularly useful in object-oriented programming, where they illustrate relationships and interactions between classes or objects.
- However, some patterns may not be suitable for functional programming languages or non-object-oriented contexts.

Creational Patterns

- Singleton: Ensures a class has only one instance and provides a global point of access to it.
- Factory Method: Defines an interface for creating objects but allows subclasses to decide which class to instantiate.
- Builder: Separates the construction of a complex object from its representation, allowing step-by-step creation.

Structural Patterns

- Adapter: Converts the interface of one class into another interface clients expect.
- Decorator: Dynamically adds responsibilities to objects without altering their structure.
- Facade: Provides a unified interface to a set of interfaces in a subsystem.

Behavioral Patterns

- Observer: Defines a dependency between objects so that when one changes state, all its dependents are notified.
- Strategy: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
- Command: Turns a request into a stand-alone object containing all information about the request.