Introduction to Design Patterns

This article introduces the concept of design patterns, their main categories, and provides brief examples of how they can be used within a development effort.

At the beginning of our project, we could add new classes and features without much hassle. But as our project grew bigger, it kept getting harder to make updates. We forgot to think through the design of our system. We needed some help getting back on track.

This article is here to help with *design patterns*, which are reusable solutions to problems that commonly occur during software development having to do with the way we organize our *components*, the pieces of code that make up our application. Design patterns can be thought of as a set of ways to structure system components and how they communicate with each other.

Why Do We Need Design Patterns?

Software continues to be worked on far after it is released. Most of the development time is spent modifying or maintaining a released software system. A software system change can result from:

- Adding new code
- Changing existing code to work with the new code
- Deleting code

Over time, these changes complicate our system, making it difficult to make more changes. Design patterns allow us to engineer our systems to better accommodate change. We often will pick a pattern to implement based on the problem that we are trying to solve. As we practice with design patterns, we develop our ability to identify the pattern that should be applied to the problem we are seeing in our system, as well as anticipating where these problems will occur.

Types of Design Patterns

There were three major types of object-oriented design patterns first proposed. Over time, more types of design patterns have been introduced, but here we will focus on those most applicable to any object-oriented system.

- *Creational patterns* hide the logic needed to create an object from clients that need to use the object.
- *Structural patterns* provide ways to combine objects into larger structures.
- *Behavioral patterns* define clear ways for objects to interact with each other.

Creational Patterns

Creational patterns solve problems that involve hiding the logic required to make objects from components that use the objects.

Let’s say that we have a class that interacts with a database. When we connect to the database, we need to pass in credentials and configurations. Should every class that needs to access the database need to pass all of this information in? Does the setup process need to happen each time the database is first accessed by an object?

The *Singleton pattern* is an example of a Creation pattern that solves this creation conundrum. This pattern is often used as the way to access an application’s database or anything else that might not need to be instantiated more than once. The Singleton pattern gives a codebase widespread access to the same singular instance of an object. This pattern enforces that only one instance exists and prevents additional instances from being created.

Structural Patterns

Structural patterns are all about composing classes and objects into larger structures.

Imagine we are creating a hierarchy of objects in which there are many variants. Our `Rectangle` might have `RedRectangles`, `BlueRectangles`, `RoundedRectangles`, `RoundedBlueRectangles`, and all sorts of other types. Each time we add a new feature of a `Rectangle` the number of classes needed to represent every variant explodes combinatorially. Instead of representing each variant as a class, we need to create classes out of types of these variants and have our base object use these as instance variables.

One specific example of a Structural pattern is the *Bridge pattern*. This pattern seeks to replace some of the responsibility of inheritance (many subclasses) with object composition (making objects out of other types of objects). We seek to take a dimension of an object that is causing the number of subclasses to grow, turn that dimension into a class, and make an instance of that class an attribute of the base object.

Behavioral Patterns

Behavioral patterns are those that don't fit into the Creational or Structural pattern categories. These patterns are described as defining ways for objects to interact with each other in an extensible way.

For example, imagine we want to be notified when certain events happen to one of our objects, such as when a field is changed. The *Observer pattern*, a specific Behavioral pattern, allows us to add new entities listening in on these events and create new events that these objects can listen to.

Multiple Choice Question
Which type of design pattern is concerned with hiding the instantiation logic of a class?
Structural Patterns
Creational Patterns
Organizational Patterns
Behavioral Patterns

Correct! Creational patterns are concerned with hiding the logic of object creation or instantiation.

When Should Design Patterns Be Used?

When first learning design patterns, it might be tempting to apply them throughout our systems. However, too many design patterns are likely to overcomplicate things. Design patterns are best applied in two scenarios:

- When we are encountering a problem that a design pattern is meant to solve
- When we are introducing a part of our system that is likely to change repeatedly

For example, if we are struggling to come up with a design for how two pieces of our system might communicate with each other, we may want to look at the problem descriptions for behavioral patterns. You might find one that applies directly to the situation we are in. This would be the perfect time to apply that pattern.

Review

Multiple Choice Question

Which statement is **NOT** true?

Design patterns are best used when we encounter a problem that a design pattern solves.

Design patterns are best chosen before development begins.

Design patterns are solutions to problems that regularly occur in software development.

Design patterns in object oriented programming can be separated into three categories: creational, structural, and behavioral patterns.

Correct! Design patterns are best chosen during development in response to a problem that is occurring.

Throughout this article, we have introduced design patterns, the categories in which they are organized, and described the scenarios in which several important design patterns are used. Design patterns give us solutions to common object-oriented problematic situations, as well as a shared vocabulary for discussing design decisions.