

BBM 486 – DESIGN PATTERNS

1. INTRODUCTION TO DESIGN PATTERNS

In this class, we will learn why (and how) you can exploit the wisdom and lessons learned by other developers who have been confronted with the same design problem and developed a satisfactory solution. We will look at some key OO design principles, and walk-through examples of how patterns work. The best way to use patterns is to load your brain with them and then recognize places in your designs and existing applications where you can apply them. Instead of code reuse, with patterns you get experience reuse. *I will try to make this class as practical as possible.*

A Pattern is a solution to a problem in a context.

- The **context** is the situation in which the pattern applies. This should be a recurring situation.
- The **problem** refers to the goal you are trying to achieve in this context, but it also refers to any constraints that occur in the context.
- The **solution** is what you are after: a general design that anyone can apply which resolves the goal and set of constraints.

An example context is that you have a collection of objects. The problem is that you need to step through the objects without exposing the collection's implementation. And the solution is to encapsulate the iteration into a separate class.

Patterns are general solutions to recurring problems. They also have the benefit of being well tested by lots of developers. So, when you see a need for one, you can sleep well knowing many developers have been there before and solved the problem using similar techniques.

Design patterns had originally been categorized into the following 3 sub-classifications based on kind of problem they solve.

1.1. Creational Patterns

Creational patterns provide the capability to create objects based on a required criterion and in a controlled way. They deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or in added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation. Creational design patterns are composed of two dominant ideas. One is encapsulating knowledge about which concrete classes the system uses. Another is hiding how instances of these concrete classes are created and combined.

Creational design patterns are further categorized into object-creational patterns and Class-creational patterns. In greater details, Object-creational patterns defer part of its object creation to another object,

while Class-creational patterns defer its object creation to subclasses. Well-known design patterns that are parts of creational patterns are:

- Factory Method: Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Singleton Pattern: Ensure that a class has only one instance, and provide a global point of access to it.
- Abstract factory pattern, which provides an interface for creating related or dependent objects without specifying the objects' concrete classes.

1.2. Structural Patterns

Structural patterns are about organizing different classes and objects to form larger structures and provide new functionality. They ease the design by identifying a simple way to realize relationships among entities.

- Decorator Pattern: Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.
- Adapter Pattern: Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces.
- Façade Patterns: Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- Composite Pattern: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Proxy Pattern: Provide a surrogate or placeholder for another object to control access to it.

1.3. Behavioral Patterns

Behavioral patterns are about identifying common communication patterns between objects and realize these patterns. They identify common communication patterns among objects and distribute responsibility. By doing so, these patterns increase flexibility in carrying out communication.

- Strategy Pattern: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
- Observer Pattern: Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.
- Command Pattern: Encapsulate a request as an object, thereby allowing for the parameterization of clients with different requests, and the queuing or logging of requests. It also allows for the support of undoable operations.
- Template Method Pattern: Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
- Iterator Pattern: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- State Pattern: Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

1.4. Class Patterns versus Object Patterns

Patterns are often classified by a second attribute of whether or not the pattern deals with classes or objects.

Class patterns describe how relationships between classes are defined via inheritance. Relationships at class patterns are established at compile time. Template Method, Adapter, and Factory Patterns fall into this category.

Object patterns describe relationships between objects and are primarily defined by composition. Relationships in object patterns are typically created at runtime and are more dynamic and flexible. Other patterns that we review are part of this category.