**STRATEGY PATTERN**

**Types of Design Patterns**  
There are mainly three types of design patterns:

1. **Creational**  
   These design patterns are all about class instantiation or object creation. These patterns can be further categorized into Class-creational patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.
2. **Structural**  
   These design patterns are about organizing different classes and objects to form larger structures and provide new functionality.

Structural design patterns are Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Private Class Data, and Proxy.

1. **Behavioral**  
   Behavioral patterns are about identifying common communication patterns between objects and realize these patterns.

Behavioral patterns are Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Null Object, Observer, State, Strategy, Template method, Visitor

Use Case of Behavioral Design Pattern-

Template pattern defines the skeleton of an algorithm in an operation deferring some steps to sub-classes, Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm structure.

*(say for an example in your project you want the behavior of the module can be extended, such that we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications. However, No one is allowed to make source code changes to it. it means you can add but can’t modify the structure in those scenarios a developer can approach template design pattern.)*

*Trong Design Pattern có 3 nhóm bao gồm:*

*Creational Pattern (nhóm khởi tạo) gồm: Abstract Factory, Factory Method, Singleton, Builder, Prototype. Nó sẽ giúp bạn trong việt khởi tạo đối tượng, như bạn biết để khởi tạo bạn phải sử dụng từ khóa new, nhóm Creational Pattern sẽ sử dụng một số thủ thuật để khởi tạo đối tượng mà bạn sẽ không nhìn thấy từ khóa này.*

*Structural Pattern (nhóm cấu trúc) gồm: Adapter, Bridge, Composite, Decorator, Facade, Proxy và Flyweight… Nó dùng để thiết lập, định nghĩa quan hệ giữa các đối tượng.*

*Behavioral Pattern gồm: Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy và Visitor. Nhóm này dùng trong thực hiện các hành vi của đối tượng.*

**Intent**

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

**Applicability**

Use the Strategy pattern when

• many related classes differ only in their behavior. Strategies provide a way to configure a class with one of many behaviors.

• you need different variants of an algorithm. For example, you might define algorithms reflecting different space/time trade-offs. Strategies can be used when these variants are implemented as a class hierarchy of algorithms [HO87].

• an algorithm uses data that clients shouldn't know about. Use the Strategy pattern to avoid exposing complex, algorithm-specific data structures.

• a class defines many behaviors, and these appear as multiple conditional statements in its operations. Instead of many conditionals, move related conditional branches into their own Strategy class.

**The Strategy pattern has the following benefits and drawbacks:**

1. Families of related algorithms. Hierarchies of Strategy classes define a family of algorithms or behaviors for contexts to reuse. Inheritance can help factor out common functionality of the algorithms.

2. An alternative to subclassing. Inheritance offers another way to support a variety of algorithms or behaviors. You can subclass a Context class directly to give it different behaviors. But this hard-wires the behavior into Context. It mixes the algorithm implementation with Context's, making Context harder to understand, maintain, and extend. And you can't vary the algorithm dynamically. You wind up with many related classes whose only difference is the algorithm or behavior they employ. Encapsulating the algorithm in separate Strategy classes lets you vary the algorithm independently of its context, making it easier to switch, understand, and extend.

3. Strategies eliminate conditional statements. The Strategy pattern offers an alter- native to conditional statements for selecting desired behavior. When different behaviors are lumped into one class, it's hard to avoid using conditional statements to select the right behavior. Encapsulating the behavior in separate Strategy classes eliminates these conditional statements.

4. A choice of implementations.

5. Clients must be aware of different Strategies.

6. Communication overhead between Strategy and Context.

7. Increased number of objects.

Tham khảo:

<https://viblo.asia/p/tim-hieu-strategy-pattern-znmMdy7YGr69>

<https://www.bogotobogo.com/DesignPatterns/strategy.php>

<https://sourcemaking.com/design_patterns/strategy/cpp/1>