Overview of Design Patterns

By: Wei Zheng

Creational Design Patterns

- Factory Pattern
- Builder Pattern
- Singleton Pattern

Factory Pattern

Factory pattern is used to create concrete classes without exposing the implementation of the creation code to the client.

It is particularly useful when you have multiple subclasses that can be built with similar parameters. For example, a shape factory could built shapes that are rectangles, triangles, etc.

Limitations occurs when the constructor requires too many pieces of information.

Builder Pattern

Builder pattern is used to construct a concrete class. It builds complex objects using simpler methods and a step by step approach.

It is useful when the class constructor have a large amount of parameters. This allows the user to be able to accurately create classes without having the remember the order of the parameters in the constructor.

For example, if you are building a person class that holds addresses, age, name, ethnicity, etc., a builder class can break down the creation into steps such as a step for address line 1, a step for city, a step for state, etc.

Singleton Pattern

A singleton pattern is when only a single instance of the class exist.

For example, an account manager that would load information after creation. It would be helpful to just load the information once if it is used across multiple classes.

Structural Pattern

- Facade Pattern
- Decorator Pattern
- Adapter Pattern

Facade Pattern

Facade pattern hides the complexity of the implementation by only exposing certain methods to access its data.

It can be used to limit access to data structures. For example, creating a shelter class that holds a list of objects, but only giving methods to some part of the collections structure such as add and remove.

Decorator Pattern

Decorator pattern is used to attach functionality to classes.

For example, when you have a person class and you want to attach some methods to modify its wallet by gambling, you would wrap it in a player class to give it additional methods.

Adapter Pattern

Adapter pattern allows for uses of incompatible interfaces/class pairings by wrapping them in another class.

This is advantageous when wanting to introduce interfaces to an existing class without rewriting the codes for the existing class. For example, if you have a class that have a method add(), but it has a different method signature to another class's add. However, you want to use the latter class so you can wrap the latter class in an adapter class that allows to modify the method signature. The intention is not to add functionality, but to bridge the difference.

Behavioral Pattern

- Observer Pattern
- Strategy Pattern
- Template Pattern
- Command Pattern

Observer Pattern

Observer pattern is when you have an one to many relationship when one class notifies every class when there is a state change in itself.

It is useful when you have multiple class that depends on the state of a single class. For example, this type of structure would be useful in creating a chat log where anytime the log is changed, the observers (users) should be notified of the new changes.

Strategy Pattern

Strategy pattern allows for the runtime choice of different algorithms independently from the client by encapsulating the algorithm.

For example, you have a list class and 2 sorting algorithms. If you have implement the sort to the list, you would likely have to chose one of the two. However by using the strategy method, you can independently allow the subclasses of the list class to choose the type of sorting algorithm.

Template Pattern

Template pattern allows for a defined skeleton of an algorithm of which parts can be overridden to change its behavior but without changing the structure.

This is useful when you have classes that shares the algorithm but with slight variations in some steps. For example, a method for parsing a int or a string from user input is similar but with key differences in type. By using the template pattern, you can write one algorithm for both and have key methods overridden in their respective classes.

Command Pattern

Command pattern decouples the request and the execution into different classes.

This allows for better adherence to SRP. You would have one class responsible to inoke the operation and one that does the operation.