Python Design Patterns

Design Patterns is the most essential part of Software Engineering, as they provide the general repeatable solution to a commonly occurring problem in software design. They usually represent some of the best practices adopted by experienced object-oriented software developers.

Classification of Design Patterns

Creational Design Pattern:

Creational patterns provides essential information regarding the Class instantiation or the object instantiation. Class Creational Pattern and the Object Creational pattern is the major categorization of the Creational Design Patterns.

Classification of Creational Design Patterns –

- Factory Method
- Abstract Factory Method
- Builder Method
- Prototype Method
- Singleton Method

Structural Design Patterns:

Structural design patterns are about organizing different classes and objects to form larger structures and provide new functionality while keeping these structures flexible and efficient. Mostly they use Inheritance to compose all the interfaces. It also identifies the relationships which led to the simplification of the structure.

Classification of Structural Design Patterns –

- Adapter Method
- Bridge Method
- Composite Method
- Decorator Method
- Facade Method
- Proxy Method
- FlyWeight Method

Behavioral Design Pattern:

Behavioral patterns are all about identifying the common communication patterns between objects and realize these patterns. These patterns are concerned with algorithms and the assignment of responsibilities between objects.

Classification of Behavioral Design Patterns –

- Chain of Responsibility Method
- Command Method
- Iterator Method

- Mediator Method
- Memento Method
- Observer Method
- State Method
- Strategy Method
- Template Method
- Visitor Method

Advantages of using Design Patterns

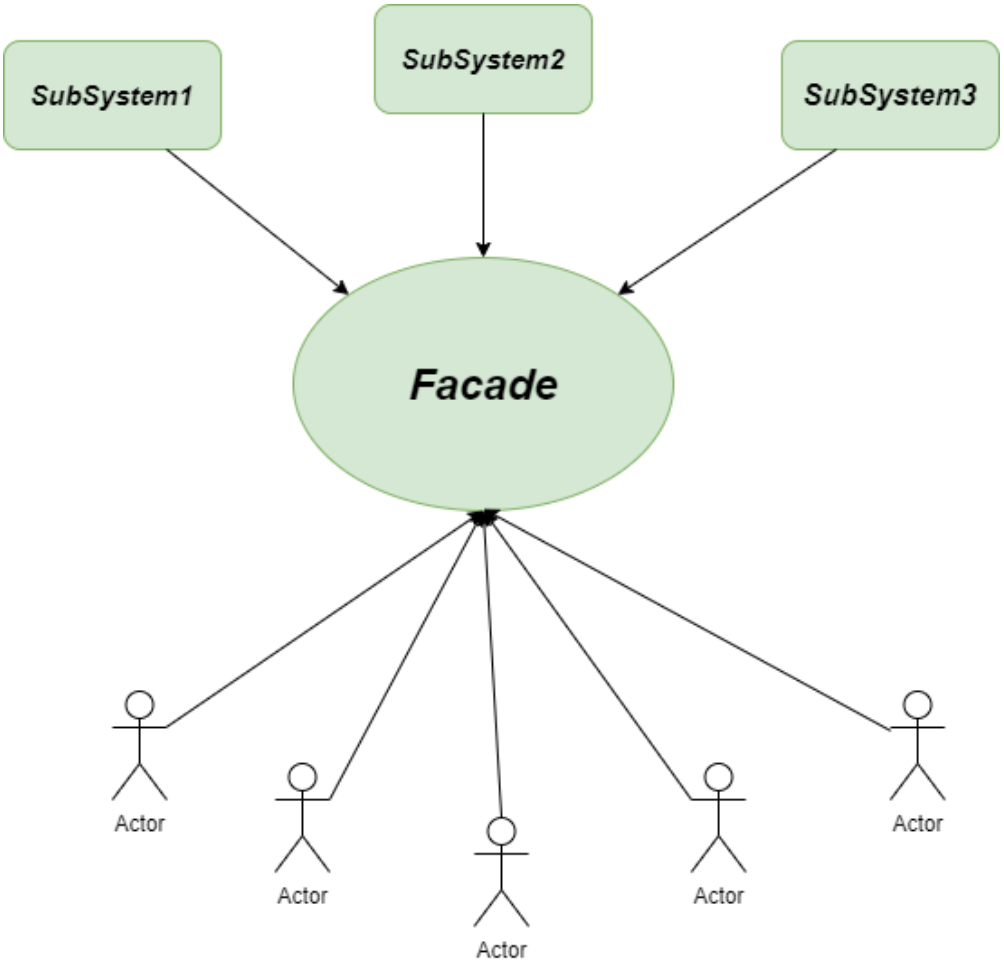
- **Reusable Solutions:** Pattern designing emphasizes creating solutions that are not specific to a particular project but can be reused across various projects. This promotes code reusability and consistency.
- **Scalability:** Patterns provide a structured approach to design, making it easier to scale a system by applying well-defined solutions to new components or modules. This promotes a consistent architecture across the entire application.
- **Abstraction and Communication:** Patterns use a common vocabulary and abstraction level, making it easier for developers to communicate and understand design decisions. This facilitates collaboration within a development team.

- **Maintainability:** Design patterns contribute to a modular and organized codebase, making it easier to maintain and update. Changes to specific patterns can be localized without affecting the entire system.
- **Speeds Up Development:** Patterns offer ready-made solutions to common problems, speeding up the development process. Developers can focus on application-specific logic rather than spending time on low-level design challenges.
- **Tool for Problem Solving:** Pattern designing serves as a tool for problem-solving in software development. It provides a set of proven strategies that developers can apply to overcome challenges encountered in various stages of the development lifecycle.

Facade Design Pattern :

Facade is structural design pattern provides a unified interface to a set of interfaces in a subsystem. It defines a higher-level interface that any subsystem can use.

we create a Facade layer that helps in communicating with subsystems easily to the clients.



Facade-Method

Problem without using Facade Method

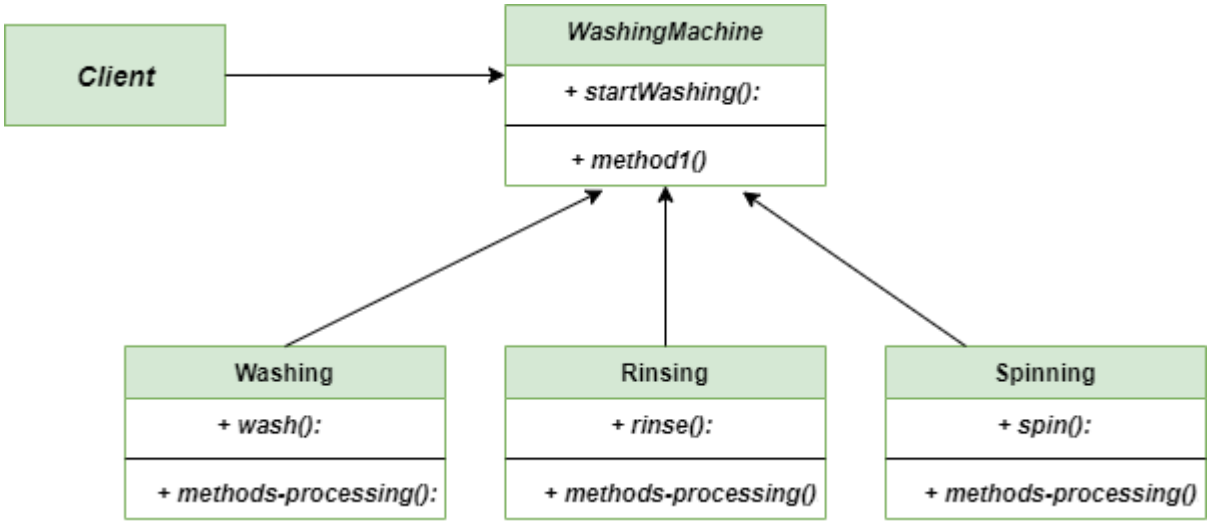
Imagine we have a washing machine which can wash the clothes, rinse the clothes and spin the clothes but all the tasks separately. As the whole system is quite complex, we need to abstract the complexities of the subsystems. We need a system that can automate the whole task without the disturbance or interference of us.

Solution using Facade Method

To solve the above-described problem, we would like to hire the Facade Method. It will help us to hide or abstract the complexities of the subsystems as follows.

Example: Washing Machine*

Class Diagram for Facade Method:



Facade-method-Class-Diagram

Flyweight Design Pattern :

Flyweight method is a Structural Design Pattern that focus on minimizing the number of objects that are required by the program at the run-time. Basically, it creates a Flyweight object which is shared by multiple contexts. It is created in such a fashion that you can not distinguish between an object and a Flyweight Object. One important feature of flyweight objects is that they are **immutable**. This means that they cannot be modified once they have been constructed.

To implement the Flyweight method in **Python**, we use **Dictionary** that stores reference to the object which have already been created, every object is associated with a key.

Example : Car Family