Design Patterns

Classic Design Patterns

- Practical recognition of utility
- Avoid anti-patterns

Python is a Dynamic Language

- Types or classes are objects at runtime
- Variables can have type as a value and can be modified at runtime
  - For example, a = 5 and a = "John", the a variable is assigned at runtime and type also gets changed
- Polymorphism is built into the language
  - No keywords such as private and protected and everything is public
- Design patterns can be easily implemented

The Simple Factory pattern

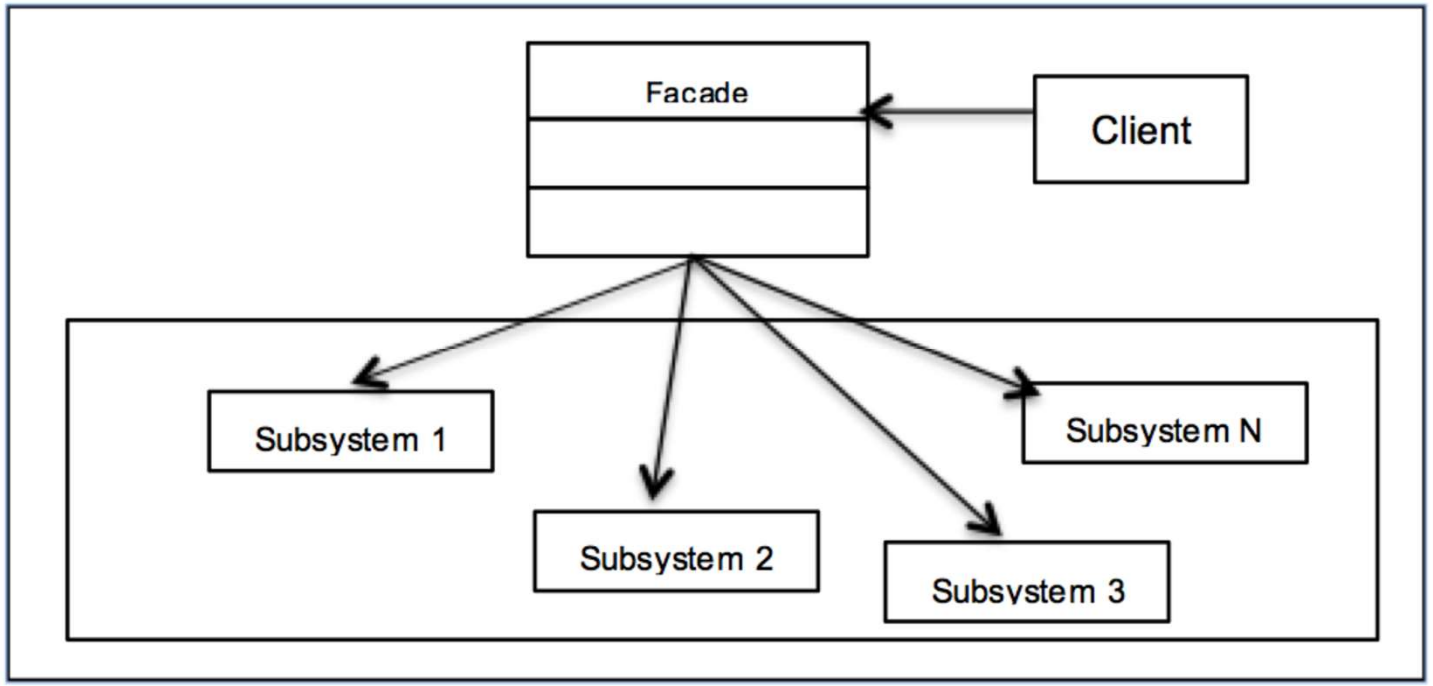
- The Factory creates objects of different types rather than direct object instantiation

The Factory Method pattern

- Responsibility for object creation is deferred to a subclass that decides the class to be instantiated
- Creation is through inheritance and not through instantiation
- Makes it more customizable
  - It can return the same instance or a subclass

The Façade Pattern

- Hides the complexities of an internal system
- Provides an interface to the client to access the system in a simplified way



The principle of least knowledge

- For every object look at the number of classes it interacts with and the way in which the interaction happens
- Avoid situations where there are many classes tightly coupled to each other
- If there are a lot of dependencies between classes, the system becomes hard to maintain
  - Any changes in one part of the system can lead to unintentional changes to other parts of the system
- The Facade pattern provides a system that makes subsystems easy to use

The Proxy Pattern

- Proxy is a class that acts as an interface to real objects
- Proxies can shield against malicious intentions and protect the real object
- Can provide a local interface for remote objects on distributed systems
- Proxies can improve performance by caching heavy or frequently accessed objects

Facade or Proxy

Proxy Pattern

- provides a surrogate for another object to control access to it
- has the same interface as the target object and holds references to target objects
- acts as an intermediary between the client and object that is wrapped

Facade Pattern

- provides an interface to large subsystems of classes
- minimizes the communication and dependencies between subsystems
- provides a single simplified interface

The Observer Pattern

- An object (the Subject) maintains a list of dependents (the Observers)
- The Subject can notify all the Observers about the changes it undergoes using any of the methods defined by the Observer
- Defines a one-to-many dependency between objects so that any change in the Subject will be notified to the Observers automatically

Observer Pull model

- Subject broadcasts to all registered Observers when there is any change
- Observer is responsible for pulling data from the subscriber when there is an amendment
- Can be inefficient
  - Subject must notify the Observer
  - Observer must pull required data from the Subject

Observer Push model

- Changes are pushed by the Subject to the Observer
- Only the required data is sent from the Subject so performance is better

RxPY

- RxPY is increasingly popular
  - https://rxpy.readthedocs.io/en/latest/index.html
  - pip3 install rx
  - For Python 2.x you need version 1.6: pip install rx==1.6.1
- Observer/Subscribe pattern with many operators

The Command Pattern

- A Command object knows about Receiver objects
  - Can invoke a method of the Receiver object
  - Parameters for the receiver method are stored in the Command object
- The invoker knows how to execute a command
- The client creates a Command object and sets its receiver
- The intentions of the Command pattern are
  - Encapsulating a request as an object
  - Allows parameterized clients with different requests
  - Allows requests to be queued
  - Provides an object-oriented callback

Command Pattern Considerations

- Command Pattern advantages
  - Decouples classes that invoke an operation from objects that know how to execute the operation
  - Creates a sequence of commands in a queue system
  - Adding a new command is easy without changing the existing code
- But
  - Can lead to a high number of classes and objects working together to achieve a goal, so need to be careful developing these classes correctly
  - Every individual command is a ConcreteCommand class that increases the volume of classes for implementation and maintenance

Stateful Design Pattern

- State
  - An interface that encapsulates the object's behaviour
- ConcreteState
  - A subclass that implements the State interface
  - Implements the behaviour associated with the current state
- Context
  - Defines the interface of interest to clients
  - Maintains an instance of the ConcreteState subclass that internally defines the implementation of the objects state