Adaptor

Problem: How to resolve incompatible, or provide a stable interface to similar components with different interfaces?

Solution: Convert the original interface of a component into another interface, through an intermediate ADAPTER object

Issues: Who creates the adapters? Which class of adapter should be created?

Solution: Seperate the details of where adapter is chosen from where adapter is created. This leads to factory pattern.

* Adapter and Facade are both wrappers—the intent of Facade is to produce a simpler interface, and the intent of Adapter is to design to an existing interface

Factory

Problem: Who should be responsible for creating objects when there are special considerations (e.g. complex creationlogic, etc.)

Solution: Create a PURE FABRICATION object called a FACTORY that handles the creation

Issues: Who creates the factory? How is it accessed?

Solution: Don't want to pass through to methods or initialise objects with a ref, so we acn use a single access point through global visibility (only one instance of the factory is needed)

Composite

Problem: How to treat a group or composition structure of objects the same way (polymorphically) as a non-composite (atomic) object?

Solution: Define classes for the composite and atomic objects so that they implement the same interface

* Composite pattern can represent an application of recursion or iteration

Facade

Problem: We require a common, unified interface to disparate set of implementations or interfaces. There may be undersirable coupling to many things in subsystem, what should we do?

Solution: Define a single point of contact for the subsystem — façade object that wraps the subsystem. This object presents a single unified interface and is responsible for collaborating with the subsystem components

Strategy

Problem: How to design for varying, but related, algorithms or policies? How to design for the ability to change these algorithms or policies — we are looking for behavioural element

Solution: Define algorithm/policy/strategy in a separate class with a common interface. This looks like the adapter pattern, but a focus on behaviour. Each class implements different behaviour, it's not just an interface for translating.

Singleton

Problem: How do we create exactly one instance of a class with objects that need a global and single point of access?

Solution: Define a static method of the class that returns the singleton

* Has to be Public, Static, Synchronised

* Don't want to make all Singleton methods static, as static methods are not polymorphic, so can't override

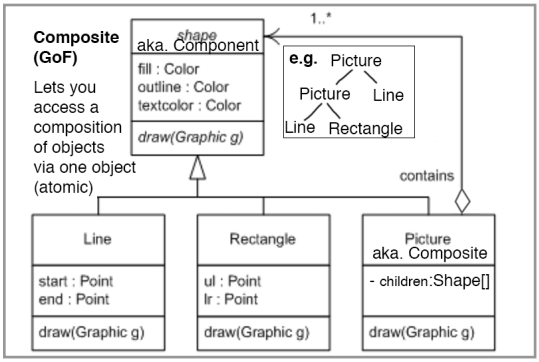
Use criteria:

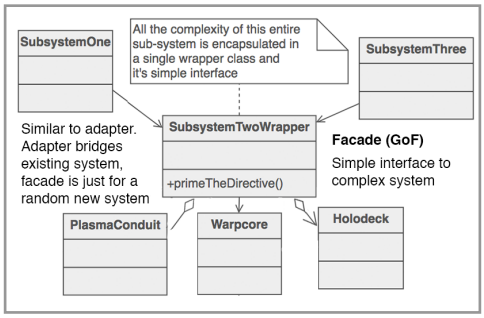
* Ownership of the single instance cannot be reasonably assigned
* Lazy initialization is desirable
* Gloabl access is not otherwise provided for

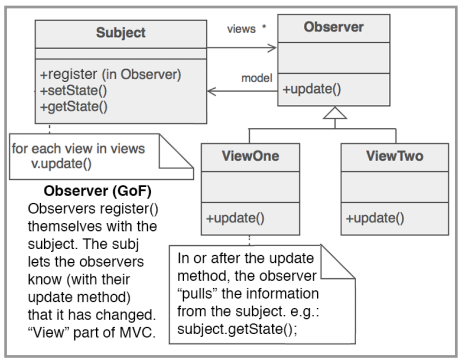
Observer

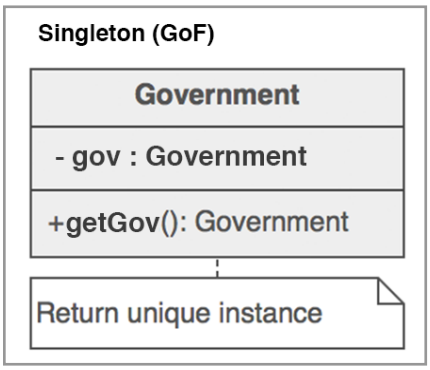
Problem: Different subscriber objects interested in state changes or events of a publisher object. Publisher wants to maintain low coupling to subscribers

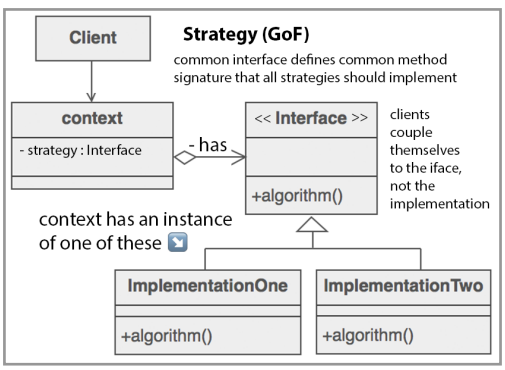
Solution: Define a subscriber or listener interface. Subscribers implement this. Publisher can dynamically register subscribers and notify them when an event occurs.

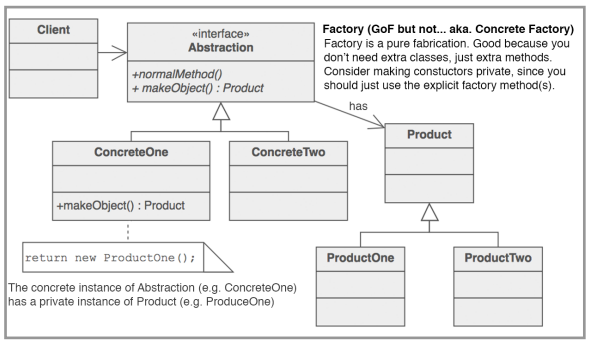












Adapter makes things work after they're designed. Adapter is retrofitted to make unrelated classes work together.Adapter provides a different interface to its subject.Facade defines a new interface, whereas Adapter reuses an old interface.