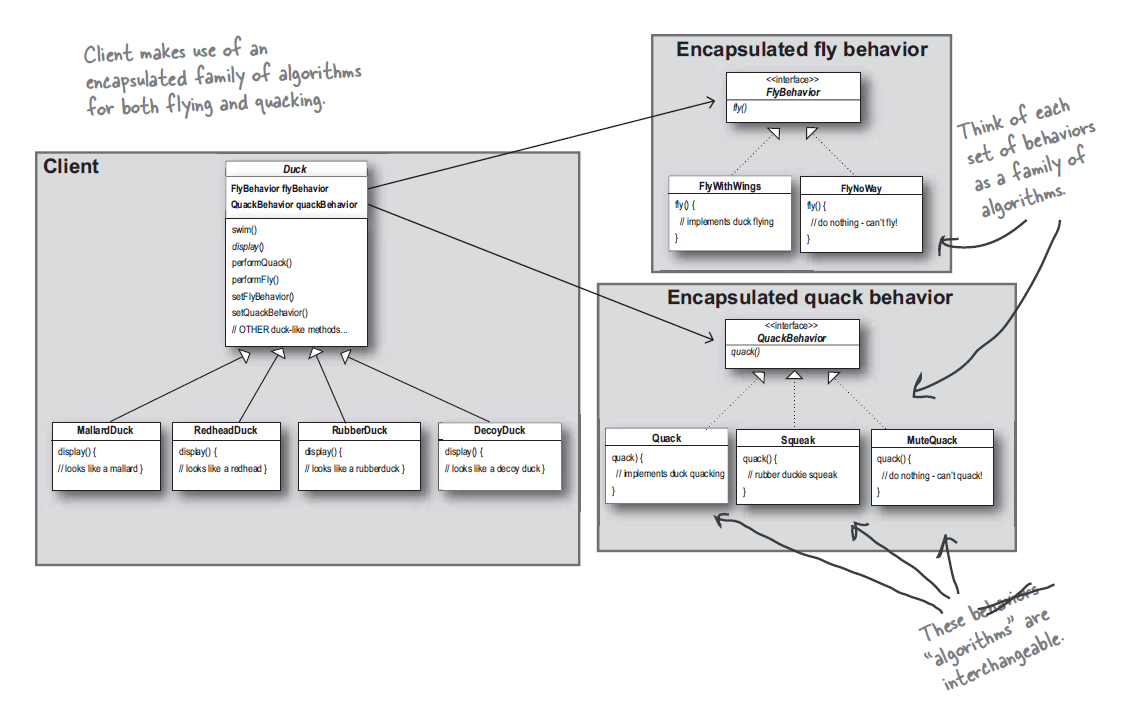
Design Principles:

1. Identify the aspects of your application that vary and separate them from what stays the same – take the parts that vary and encapsulate them, so that later they can be altered or extended without affecting those that don’t.
2. Program to an interface, not an implementation.
3. Favor composition over inheritance (HAS-A can be better than IS-A).
4. Strive for loosely coupled designs between objects that interact – when two objects are loosely coupled, they can interact, but have very little knowledge of each other.
5. Open-close: classes should be open for extension but closed for modification.
6. Dependency Inversion Principle: Depend upon abstractions? Do not depend upon concrete classes. High level components should not depend on low level components, they should both depend on abstractions
7. SRP -

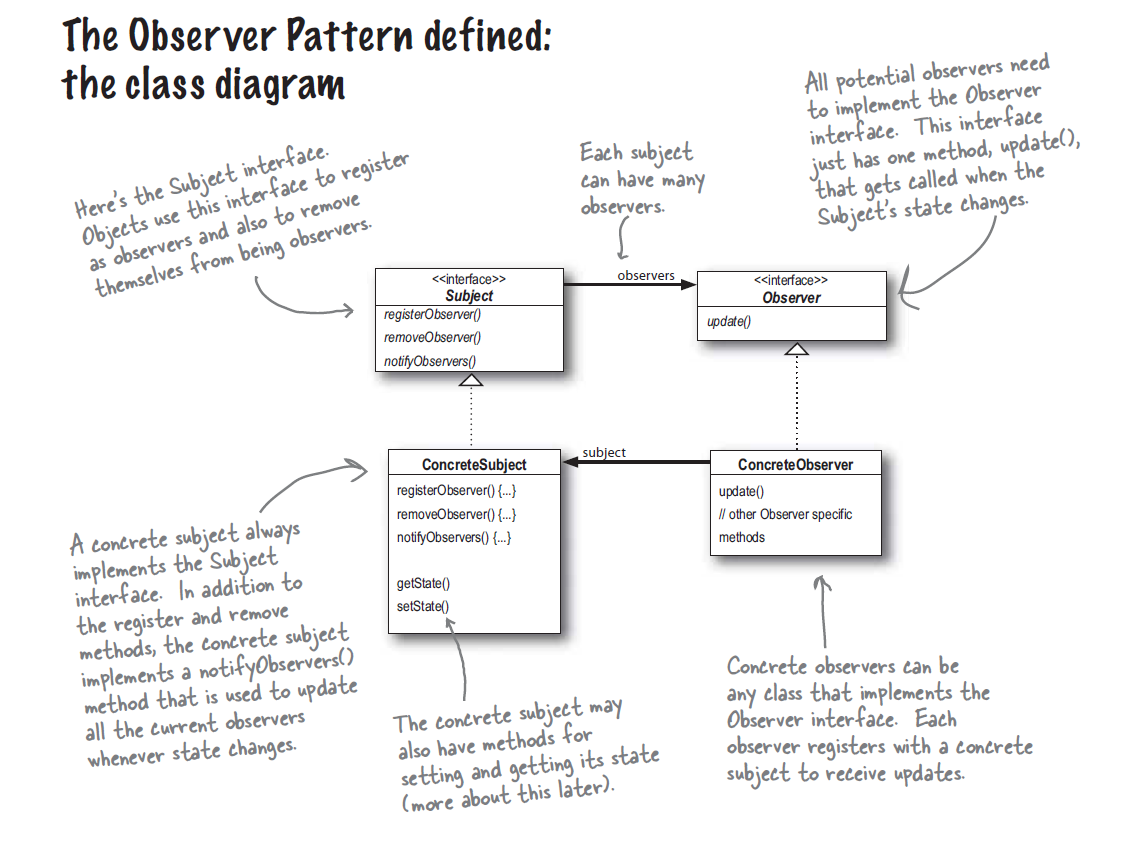
Design Patterns:

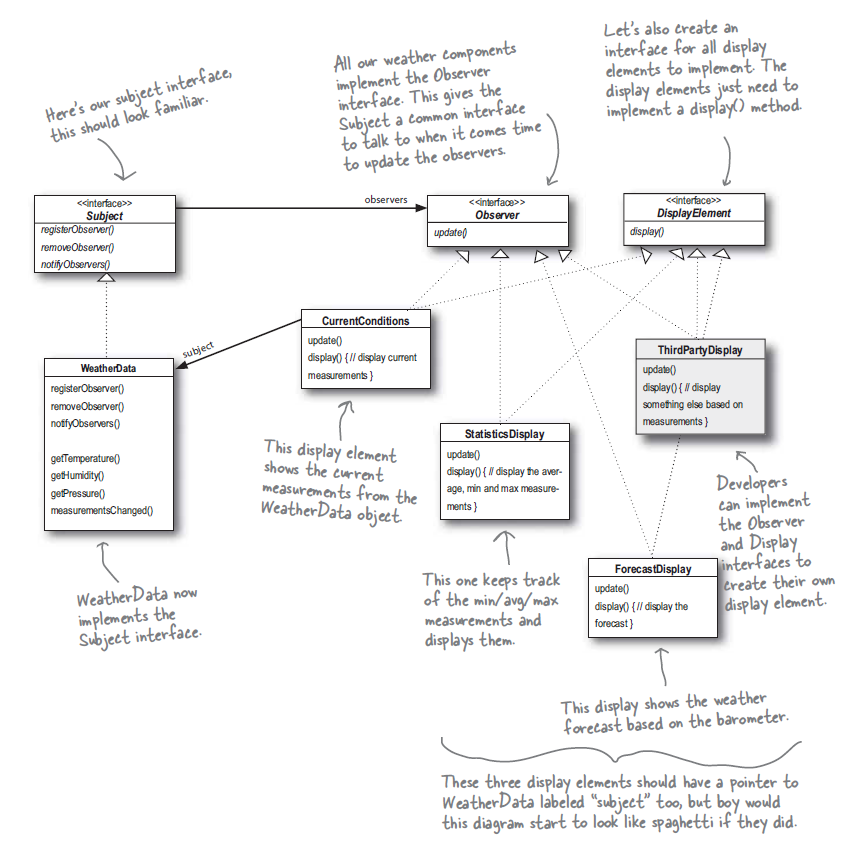
1. Strategy – defines a family of algorithms, encapsulates each one and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

* Identified and extracted what varies (FlyBehavior and QuackBehavior)
* Programmed to an interface (the behaviors are defined as interfaces with concrete implementations like FlyWithWings or Squeak)
* Used composition to add dependencies to Duck class.

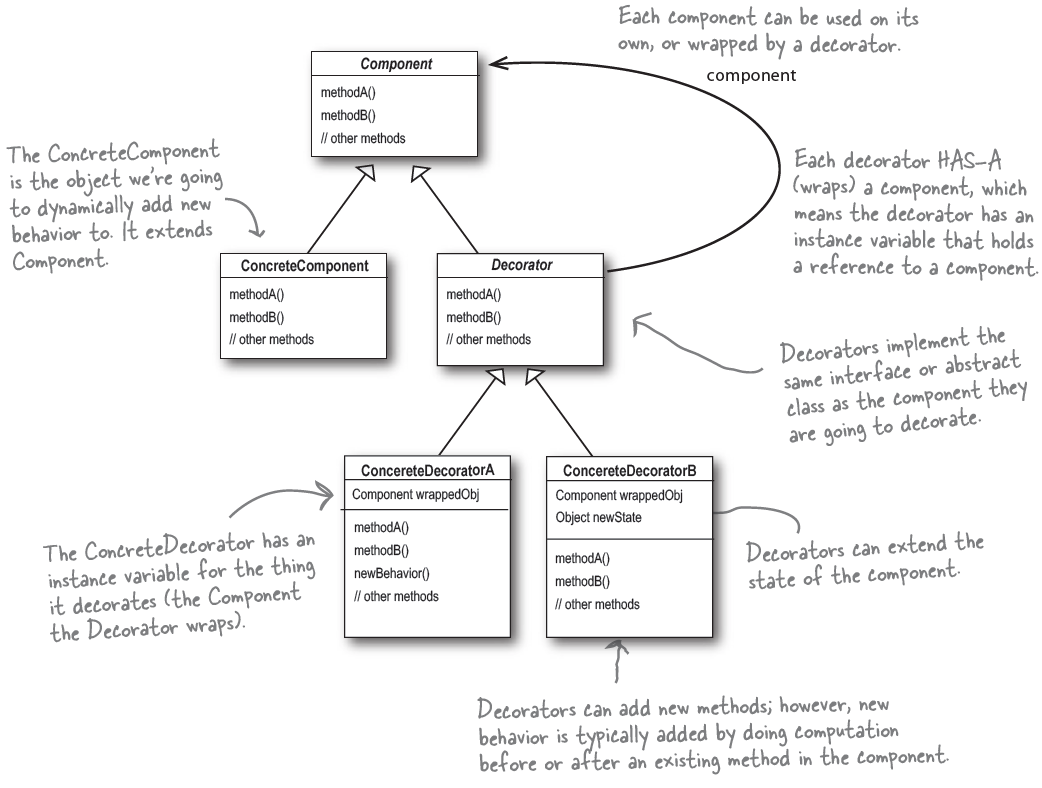


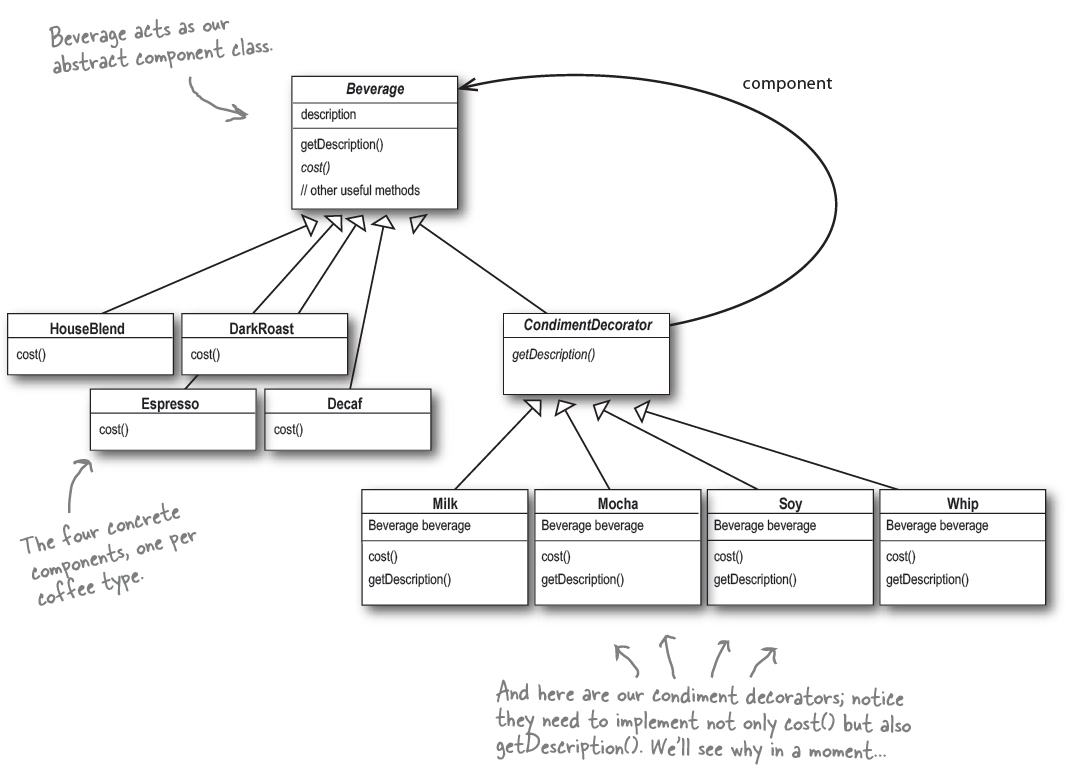
1. Observer – defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.



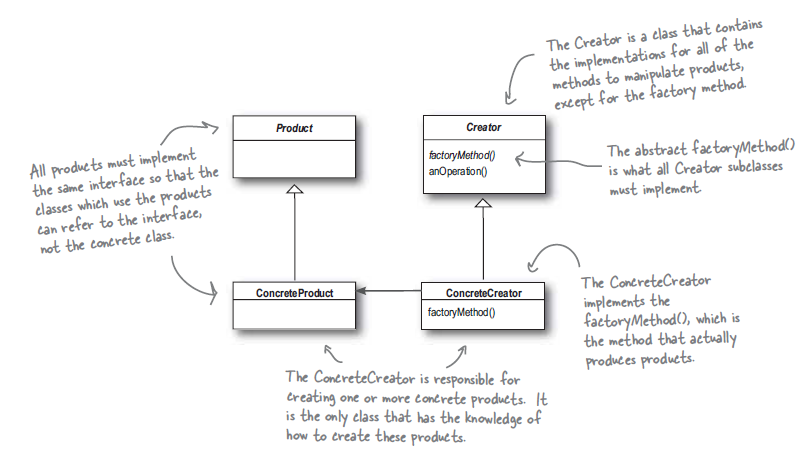


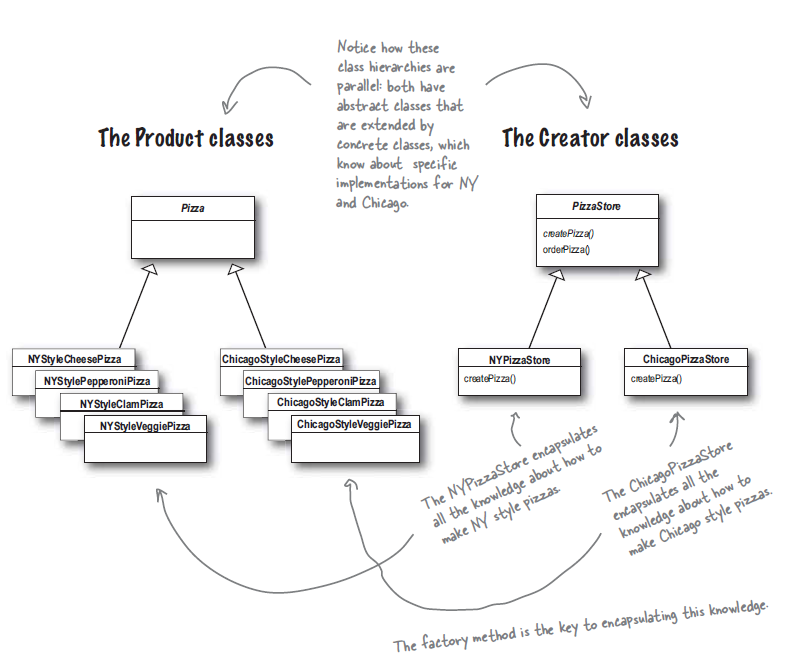
1. Decorator Pattern – attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.



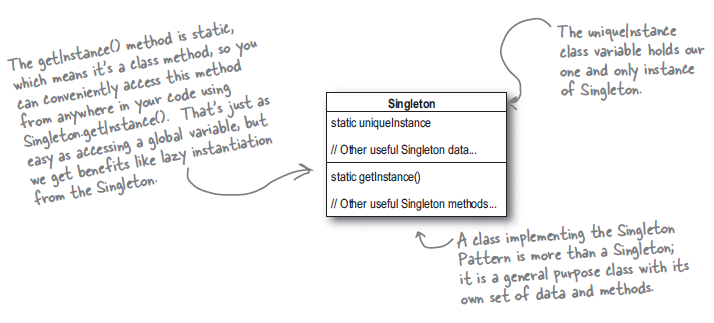


1. Factory Method Pattern – the factory method pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate. It lets a class defer instantiation to subclasses.

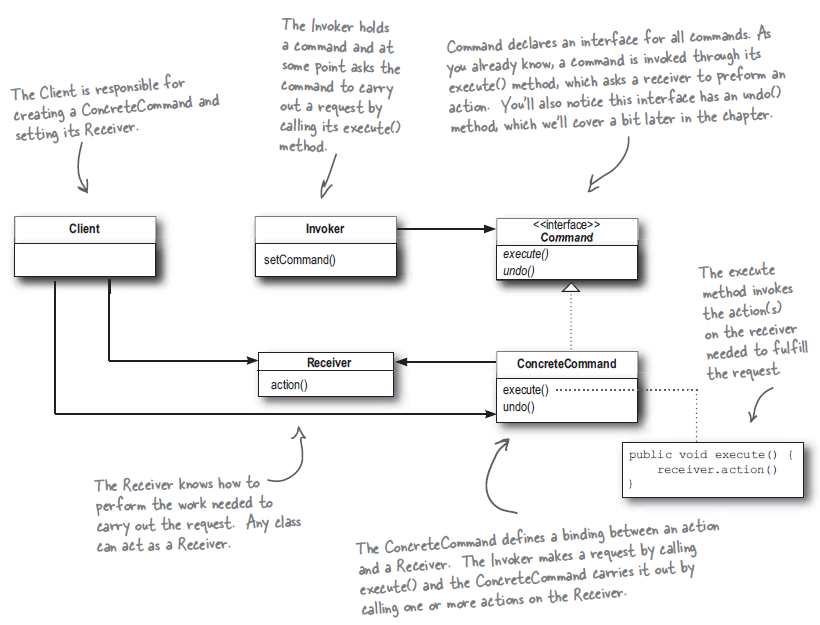


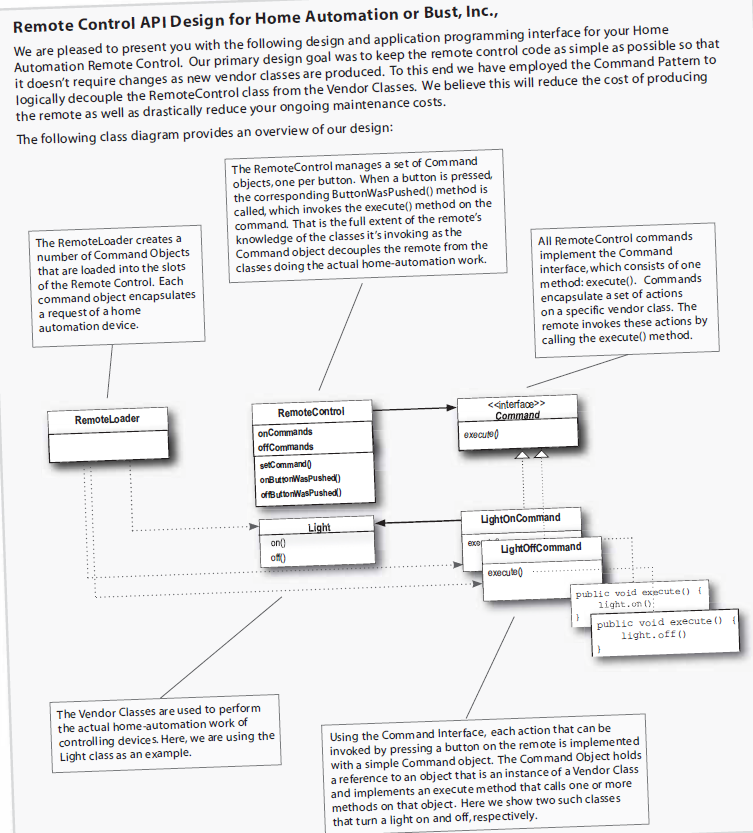


1. Singleton Pattern – ensures that a class has only one instance and provides a global point of access to it.

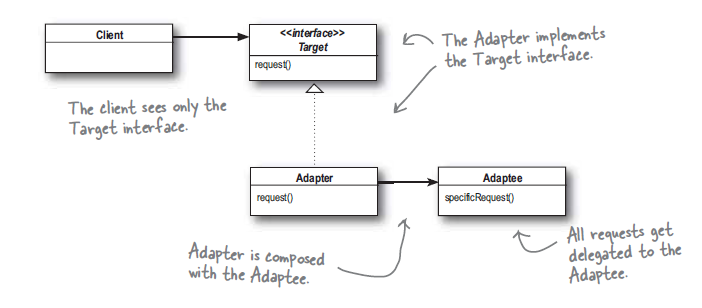


1. Command Pattern – encapsulates a request as an object, thereby letting you parameterize other objects with different request, queue or log reports and support undoable operations.

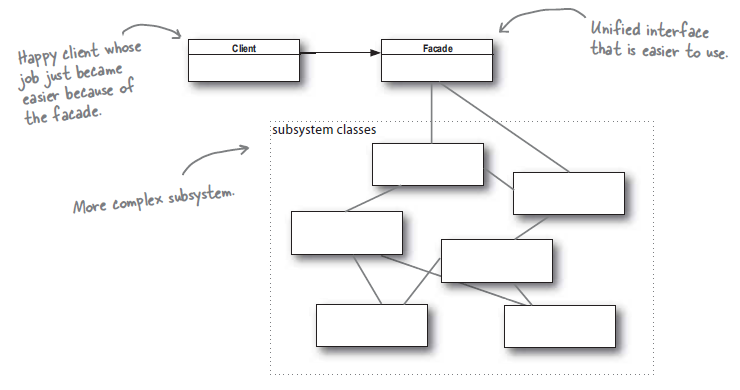


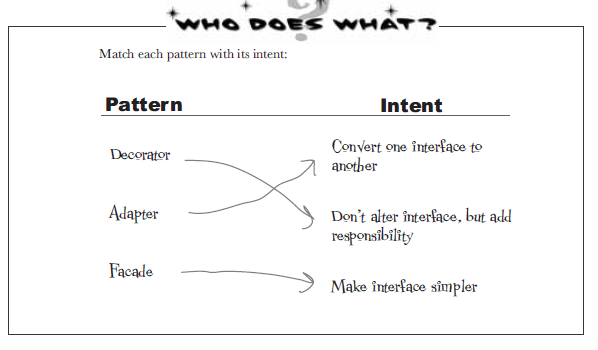


1. Adapter Pattern – converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

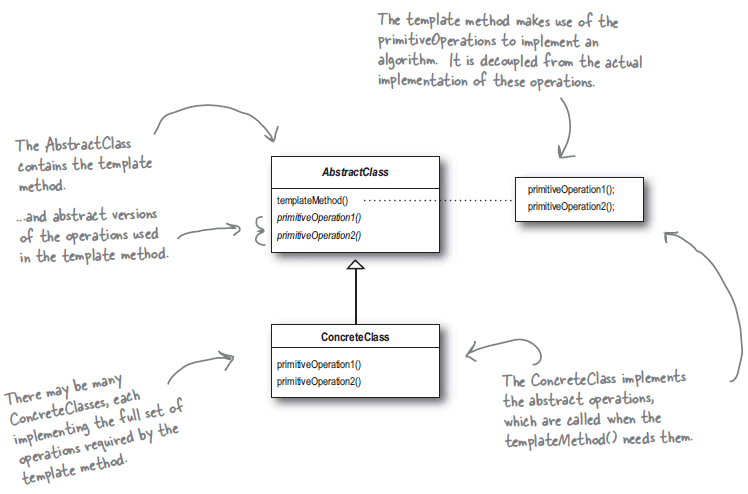


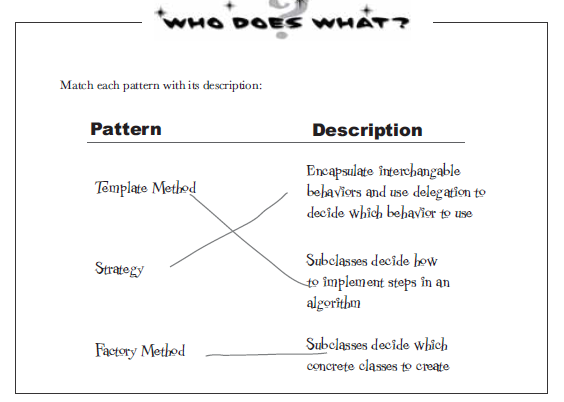
1. Façade Pattern – provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher level interface that makes the subsystem easier to use.



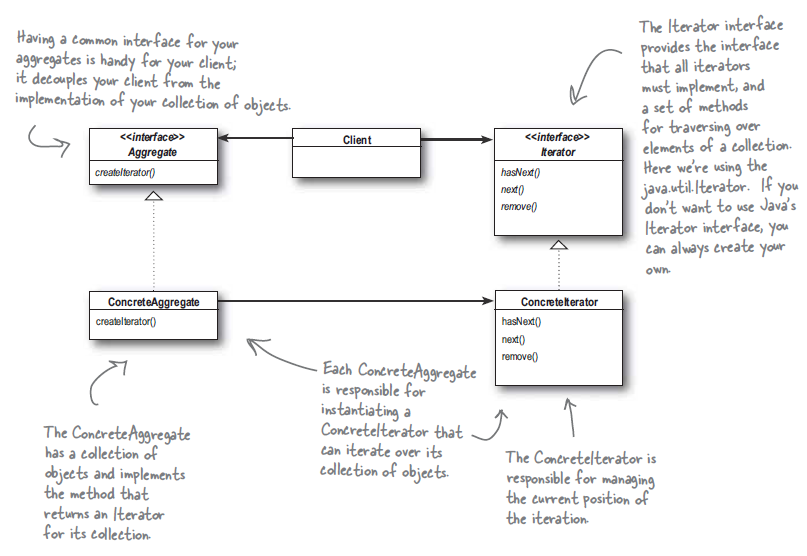


1. Template method pattern – defines a skeleton of an algorithm in a method, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm’s structure.

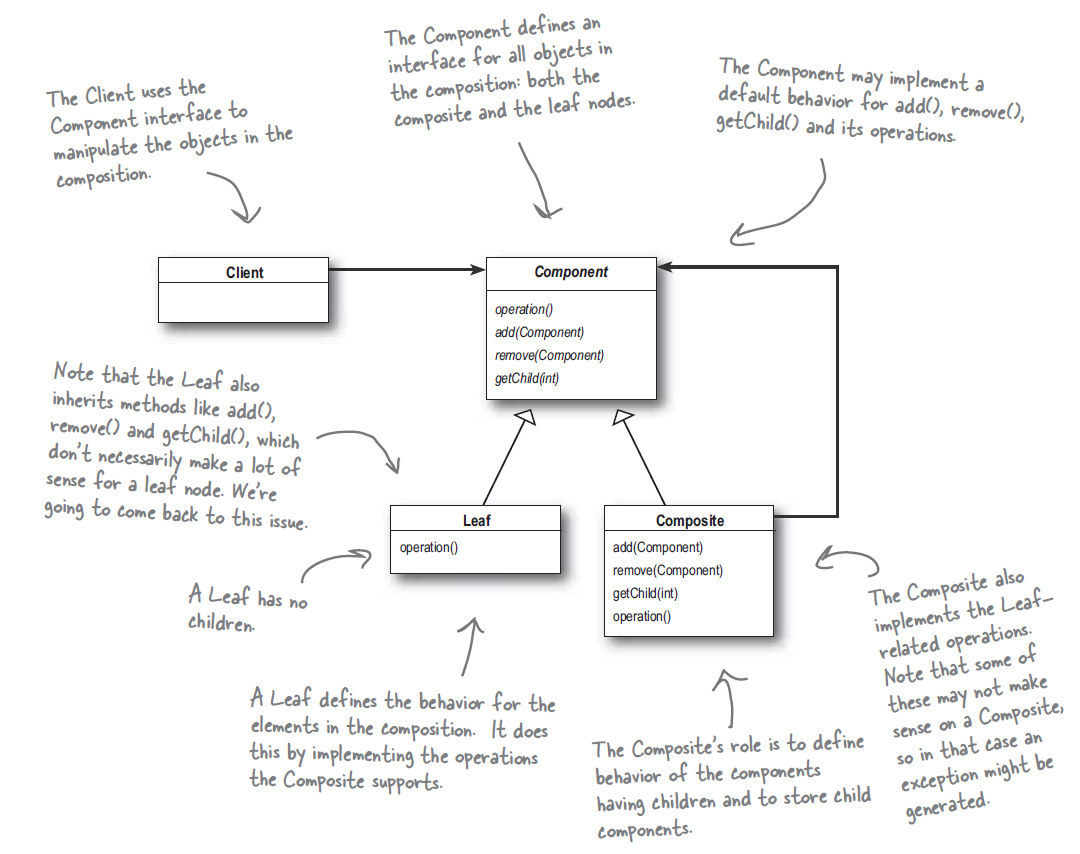


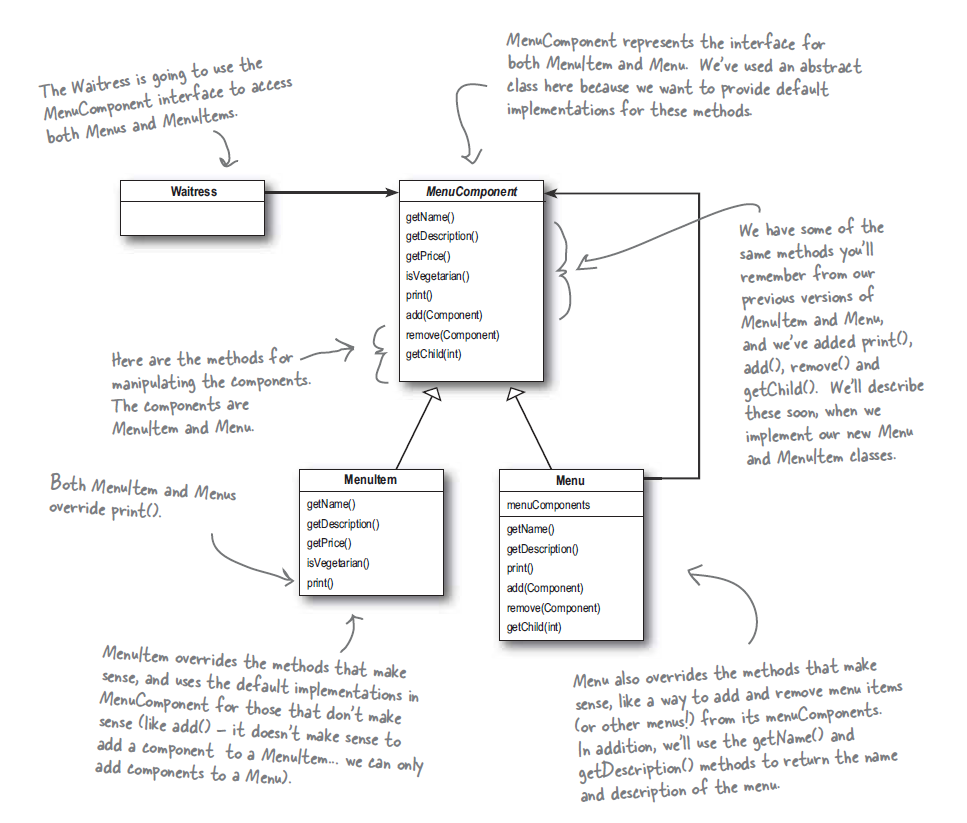


1. Iterator Pattern – provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation. It also places the task of traversal on the iterator object, not on the aggregate, which simplifies the aggregate interface and implementation and places responsibility where is should be.



1. Composite Pattern – allows to compose objects into tree structures to represent part whole hierarches. Composite lets clients treat individual objects and compositions of objects uniformly.





1. State Pattern – allows an object to alter its behavior when its internal state changes. The object will appear to change its class.

