The single most important activity in pattern writing is reflection.
- To create patterns, developers must reflect on, and document, their successes (and failures), when designing and implementing software systems. Developers use design patterns to capture and use this collective experience, which ultimately helps them share similar successes with other developers.
- Design patterns are grouped into three categories — creational, structural and behavioral
- Creational design patterns describe techniques for instantiating objects (or groups of objects). These design patterns address issues related to the creation of objects, such as preventing a system from creating more than one object of a class (e.g., Singleton) or deferring until execution time the decision as to what types of objects are created (e.g., Factory Method).
- Structural design patterns describe common ways to organize classes and objects in a system. Developers often find two problems with poor organization. The first is that classes are assigned too many responsibilities. Such classes may damage information hiding and violate encapsulation, because each class may have access to information that belongs in a separate class. The second

problem is that classes can overlap responsibilities. Burdening a design with unnecessary classes wastes time for designers because they will spend hours trying to extend or modify classes that should not even exist in the system.

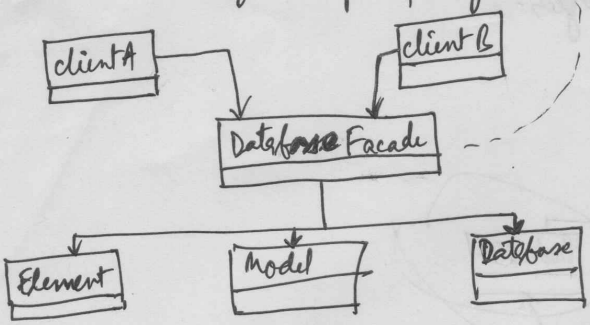
- Behavioral design patterns assign responsibilities to objects. These patterns also provide proven strategies to model how objects collaborate with one another and offer special behaviors appropriate for a wide variety of applications. The Observer pattern is a classic example of collaborations between objects and of assigning responsibilities to objects. For example, GUI components use this pattern to communicate with their listeners, which respond to user interactions. A listener observes state changes in a particular component by registering to handle that component's events. When the user interacts with the component, that component notifies its listeners (also known as its observers) that the component's state has changed (e.g., a button has been pressed).

[behavior/method/operation]

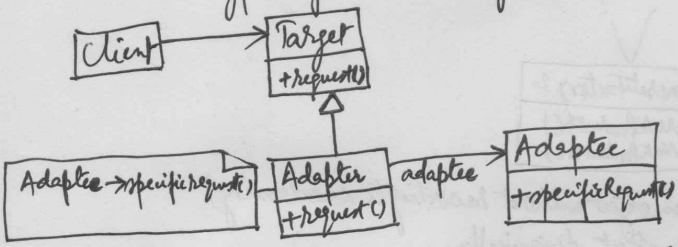
design patterns — class, object → programmer's perspective

structural, Creational, Behavioral,...
(Designer's / Architect's perspective)

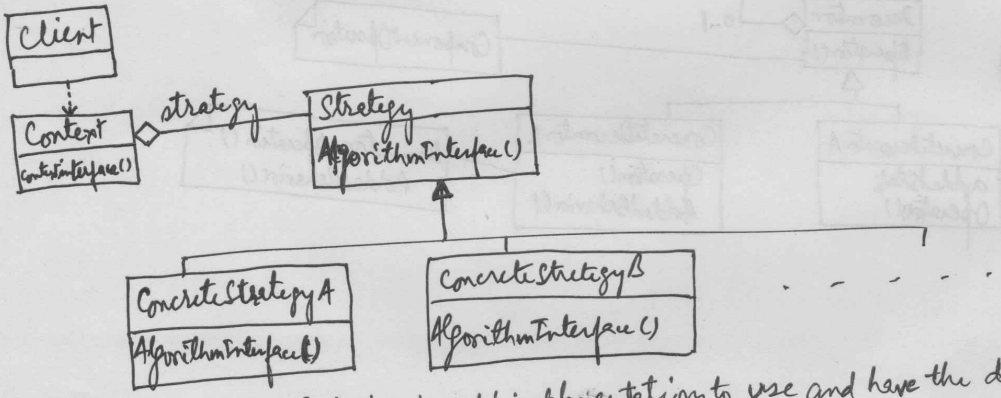
✓ Facade (presents a new interface for the client of the existing system to use)
- get rid of complexity



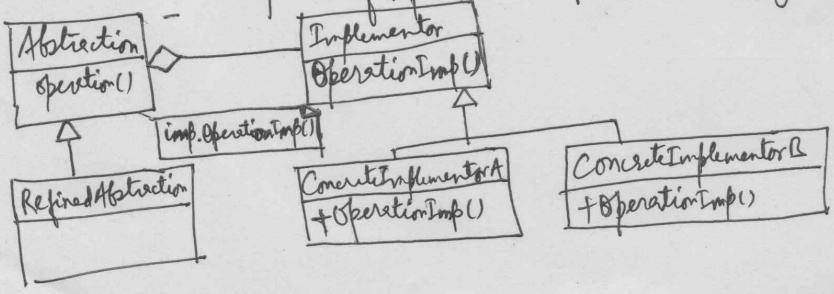
✓ Adapter (provides a wrapper with the desired interface)
- typically used when you have to make something a derivative of an abstract class.



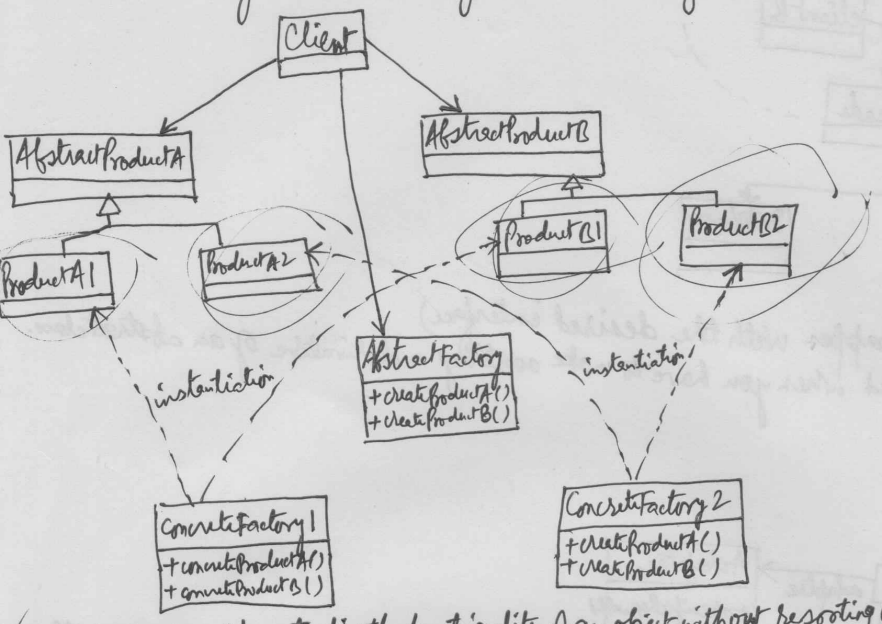
✓ Strategy pattern (- separates the selection of algorithm from the implementation of the algorithm.
- allows for the selection to be made based upon context (in which different business rules or algorithms occur).



✓ Bridge pattern (define an interface for all implementations to use and have the derivations of the abstract class use that)
- decouple a set of implementations from the set of client objects using them (increases extensibility)



Abstract Factory – (coordinates the creation of families of objects)
– Gives a way to take the rules of how to perform the instantiation out of the client object that is using these created objects.



Decorator – (allows for extending the functionality of an object without resorting to subclassing)
– attach additional responsibilities to an object dynamically.