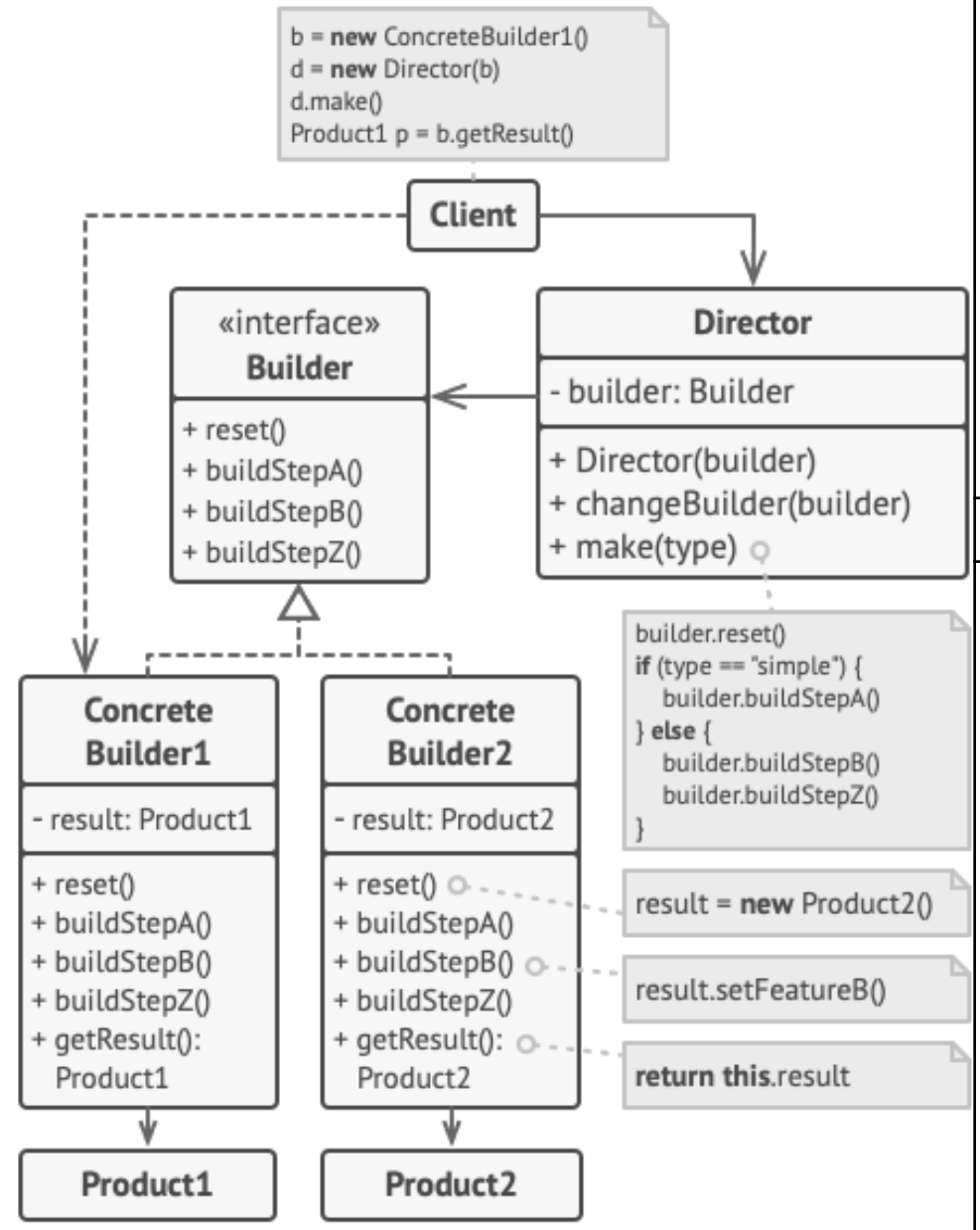
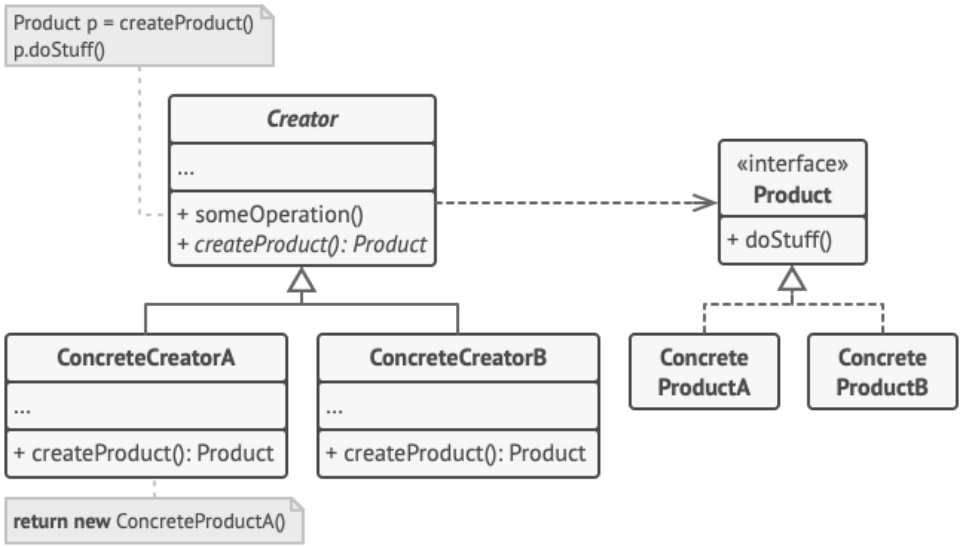
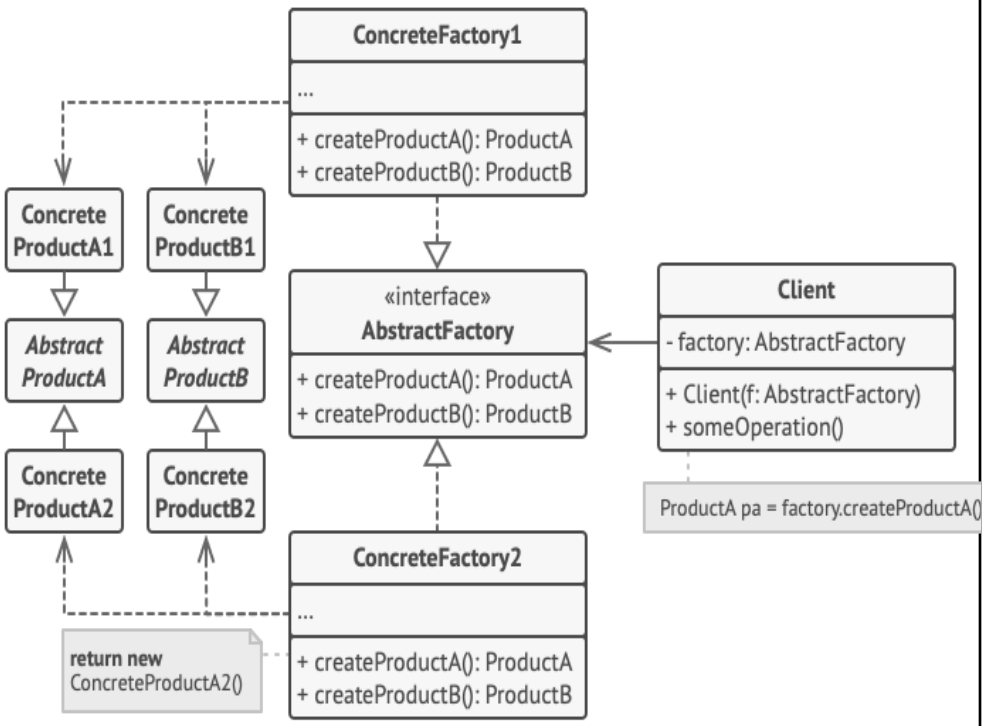
## Builder Method (Creational)



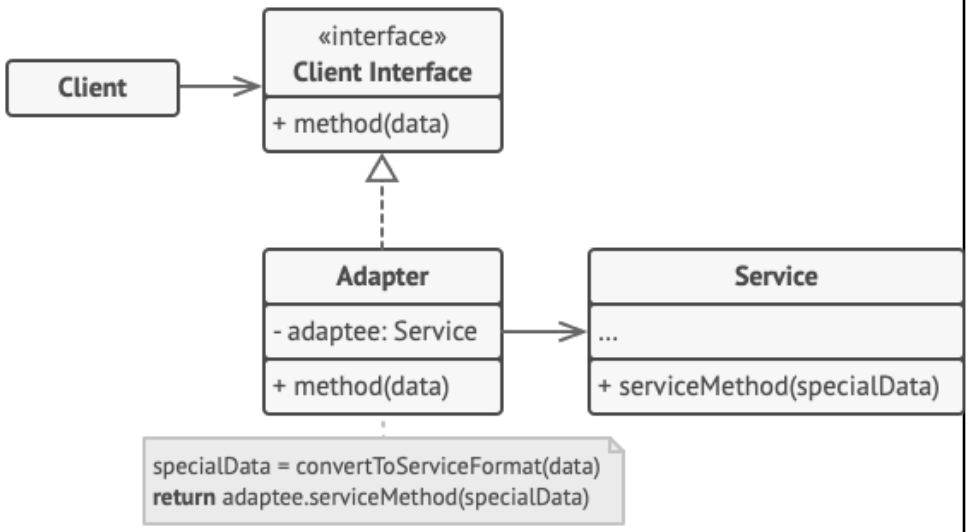

## Factory Method (Creational)



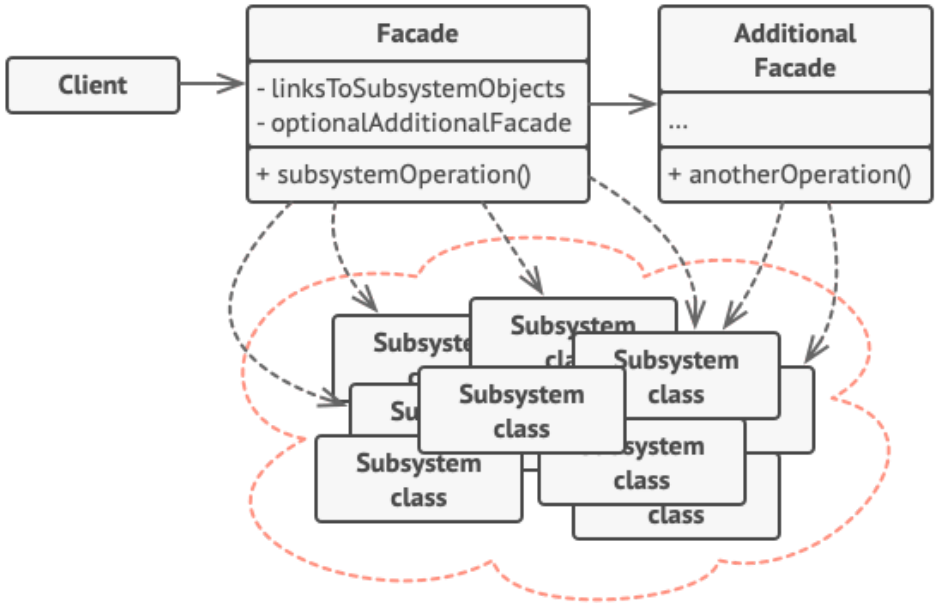

## Abstract Factory Method (Creational)



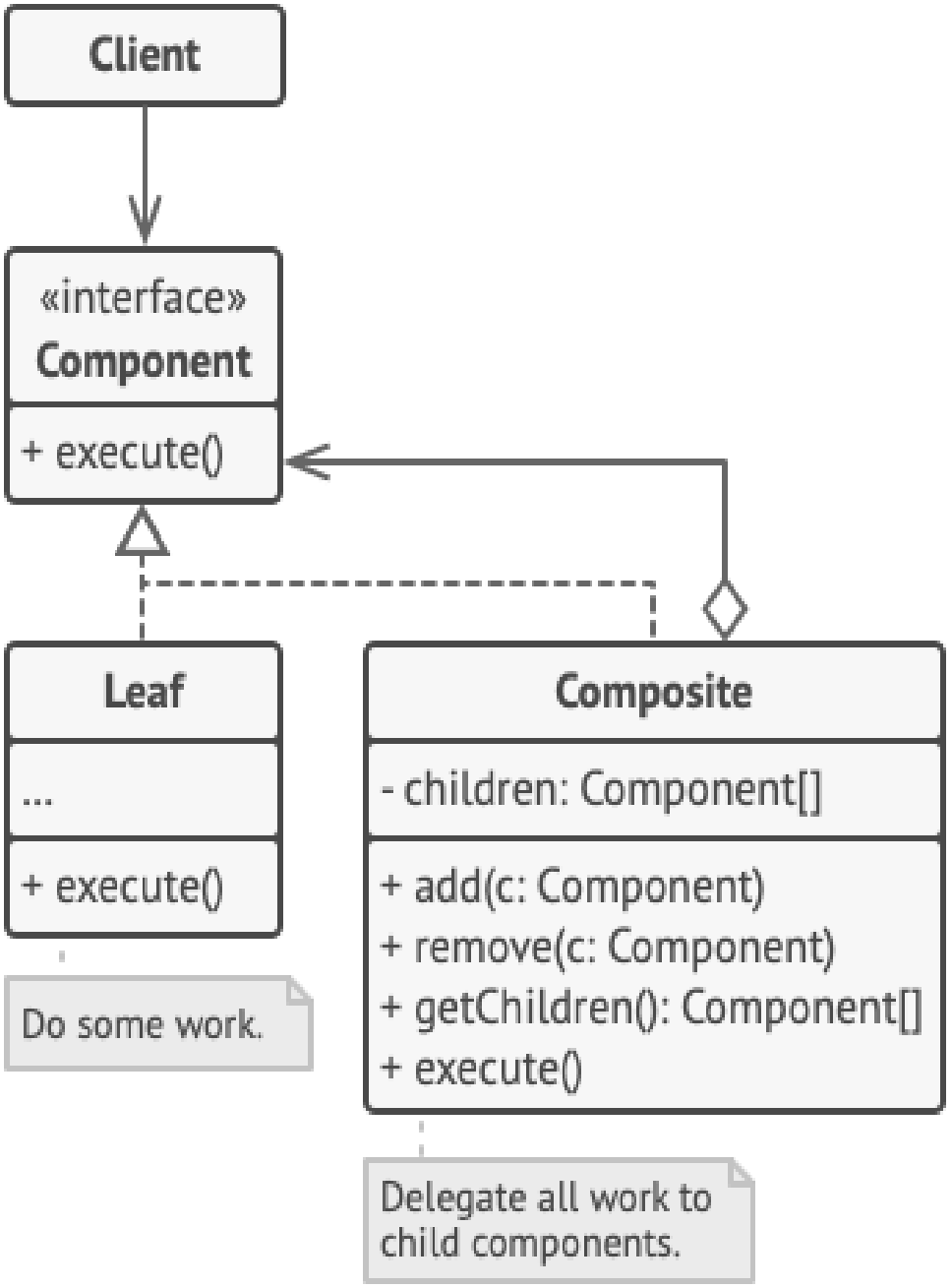
## Adapter (Structural)



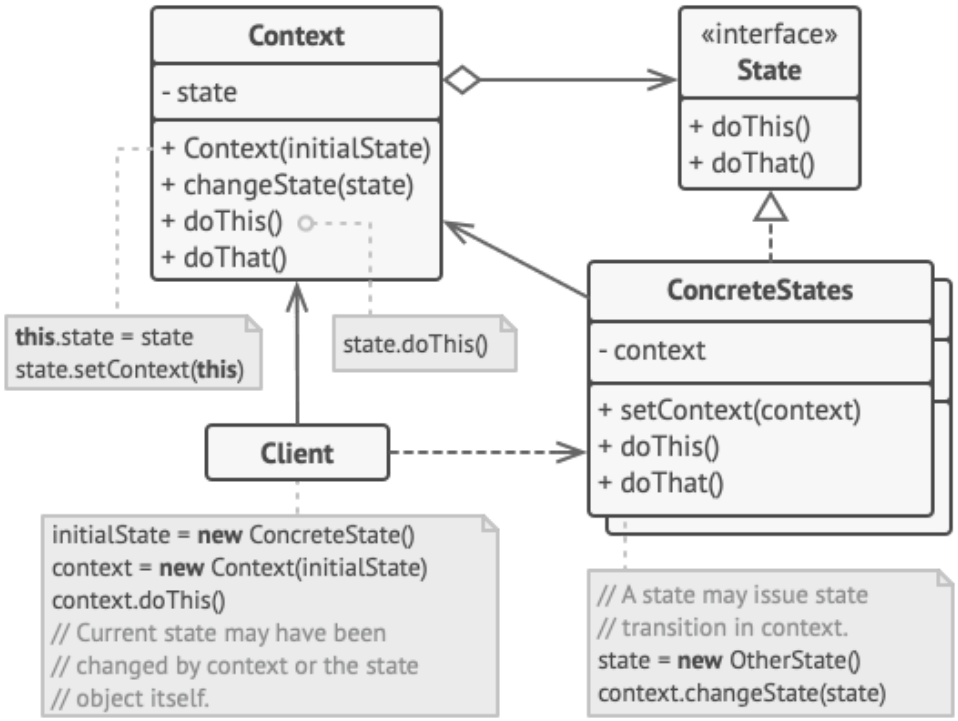

## Facade (Structural)



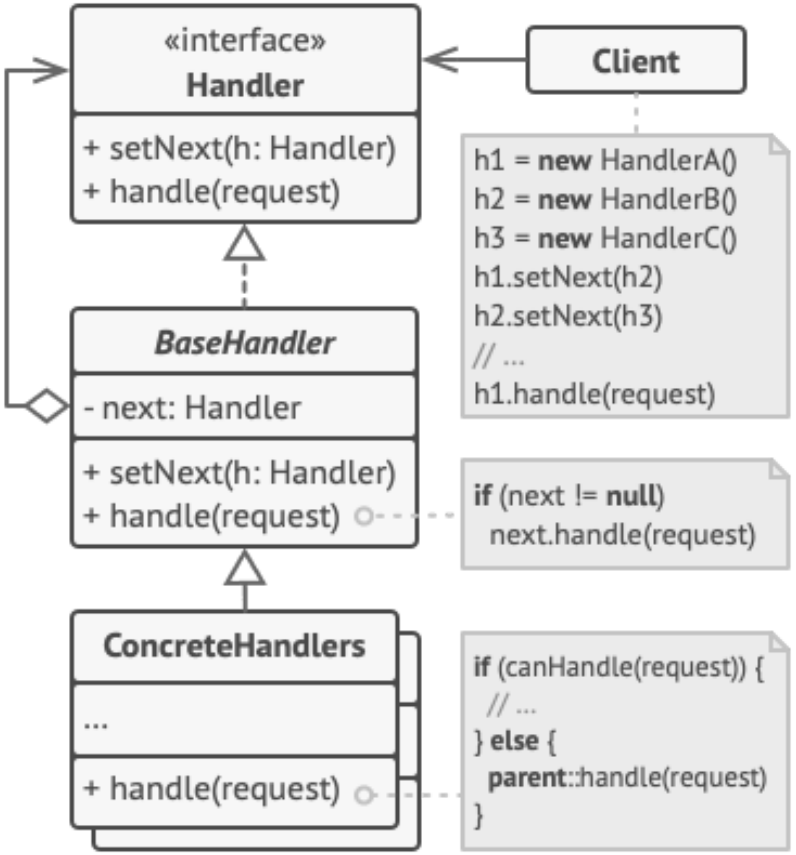

## Composite (Structural)



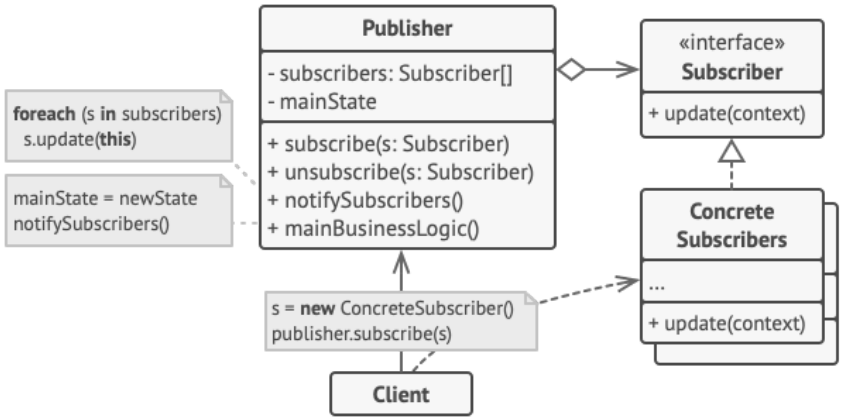
## State (Behavioral)



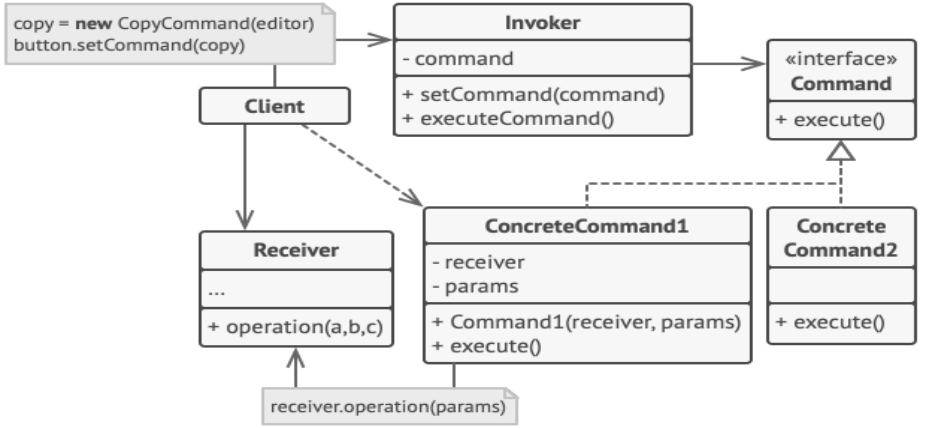

## Chain Of Responsibility (Behavioral)



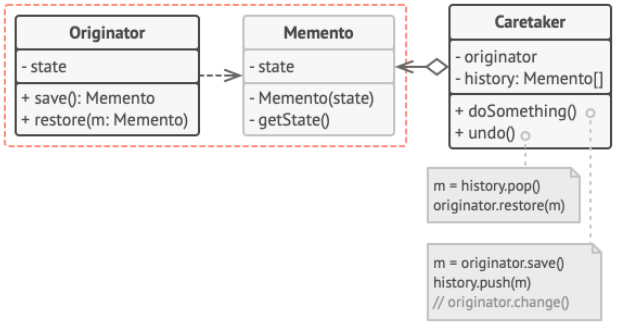

## Observer (Behavioral)

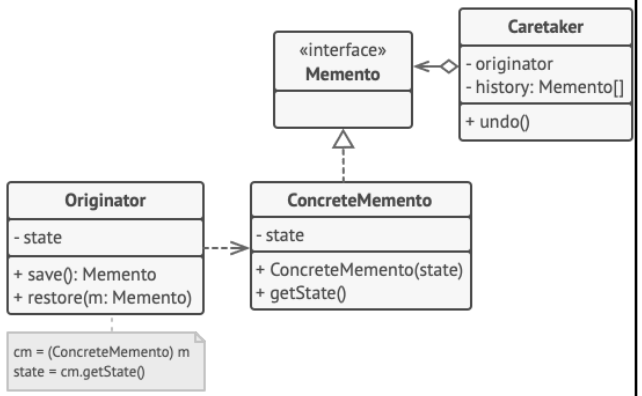


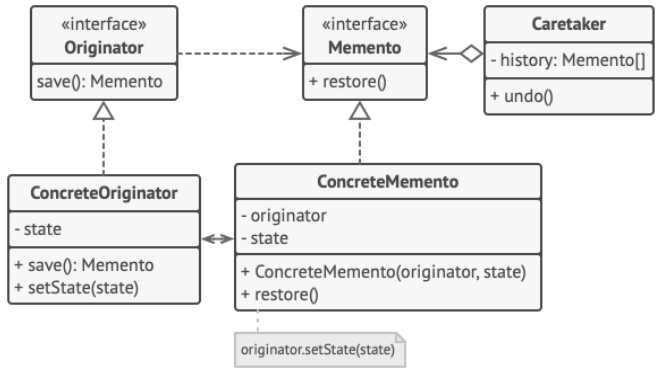

## Command (Behavioral)



## Memento (Behavioral)




### ***Creational Design Patterns***

1. **Factory Method**: provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created
2. **Abstract Factory**: lets you produce families of related objects without specifying their concrete classes
3. **Builder**: lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code

### ***Structural Design Patterns***

1. **Adapter**: allows objects with incompatible interfaces to collaborate
2. **Composite**: lets you compose objects into tree structures and then work with these structures as if they were individual objects
3. **Facade**: provides a simplified interface to a library, a framework, or any other complex set of classes

### ***Behavioral Design Patterns***

1. **Command**: turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request's execution, and support undoable operations
2. **Chain of Responsibility**: lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain
3. **Observer**: lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing
4. **State**: lets an object alter its behavior when its internal state changes. It appears as if the object changed its class
5. **Memento**: lets you save and restore the previous state of an object without revealing the details of its implementation