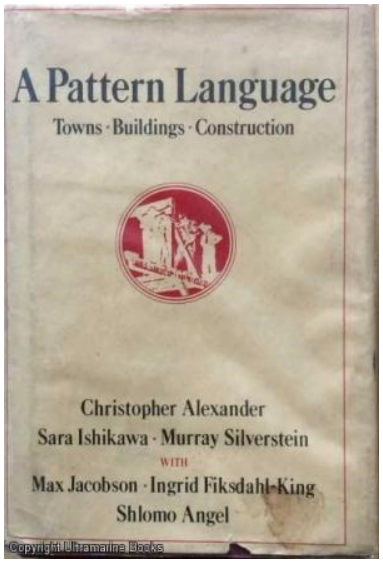




# Design Patterns

Prof. Filippo Lanubile

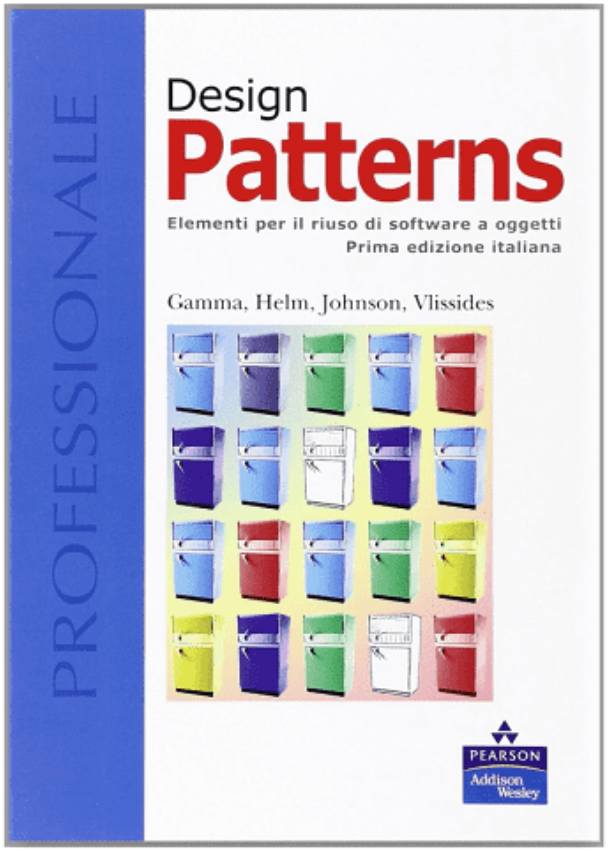
# Design Pattern

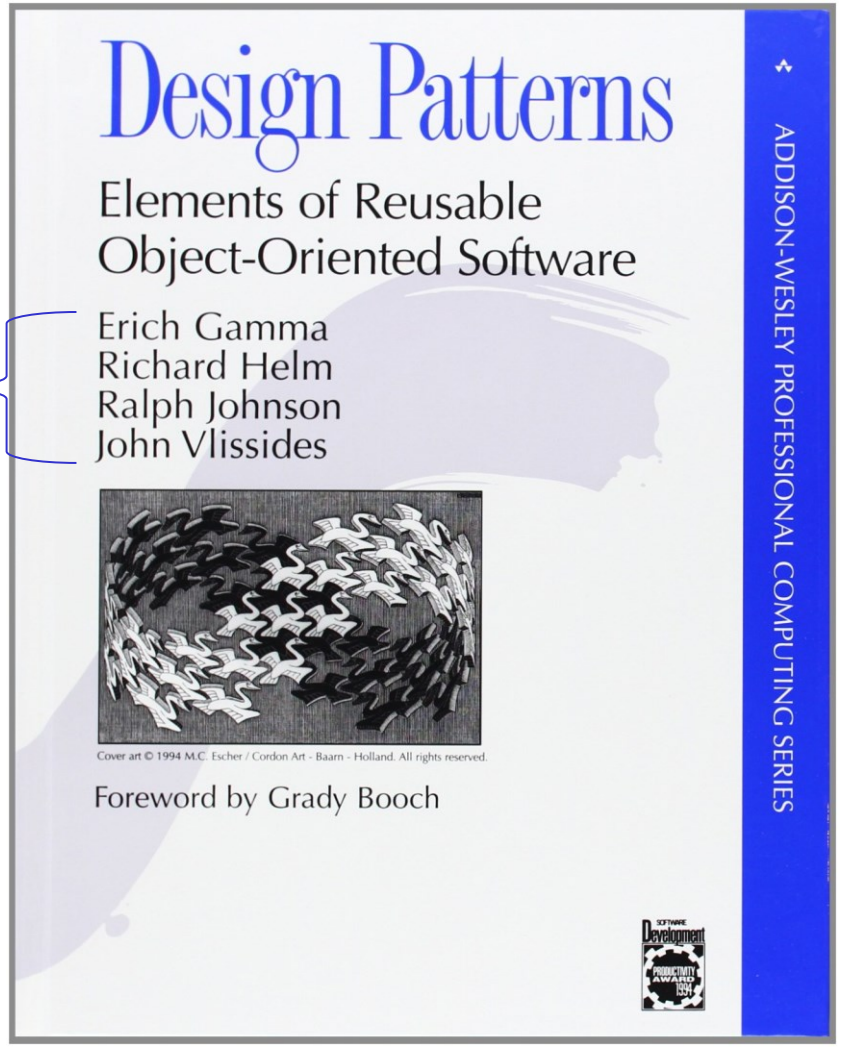



- Generalizzazioni di buone soluzioni progettuali applicate a problemi ricorrenti
  - Ispirati al libro «A Pattern Language» dell'architetto Christopher Alexander (1977)
- Forniscono un **vocabolario comune ai progettisti**

# GoF Design Patterns

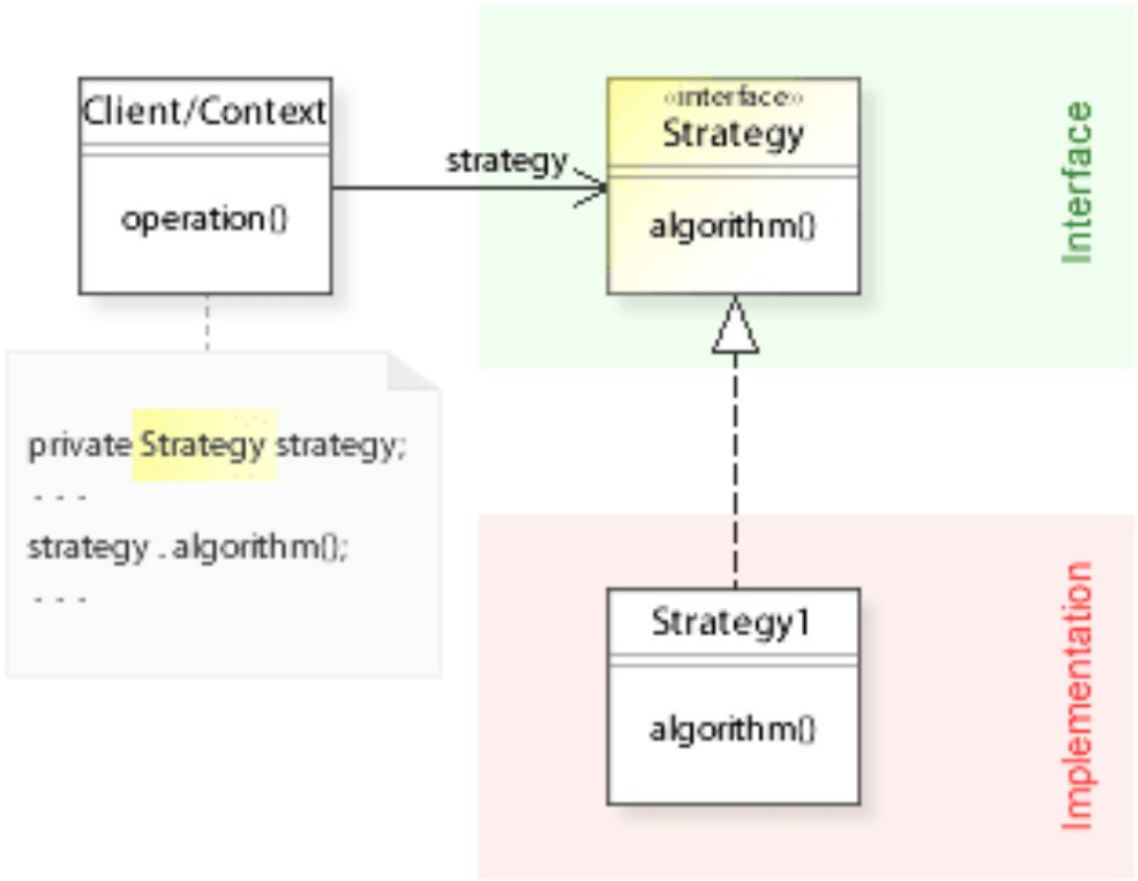
GoF ⇔ «Gang of Four»



# Primo principio dei GoF design pattern

Programma rispetto a un'interfaccia anziché rispetto a un'implementazione



# Secondo principio dei GoF design pattern

Favorisci la composizione di oggetti rispetto all'ereditarietà tra classi

- Con la composizione di oggetti è possibile cambiare il comportamento dinamicamente (run-time)
- Usa l'ereditarietà tra classi solo se c'è una relazione di generalizzazione

# I 23 GoF design patterns

Come istanziare oggetti

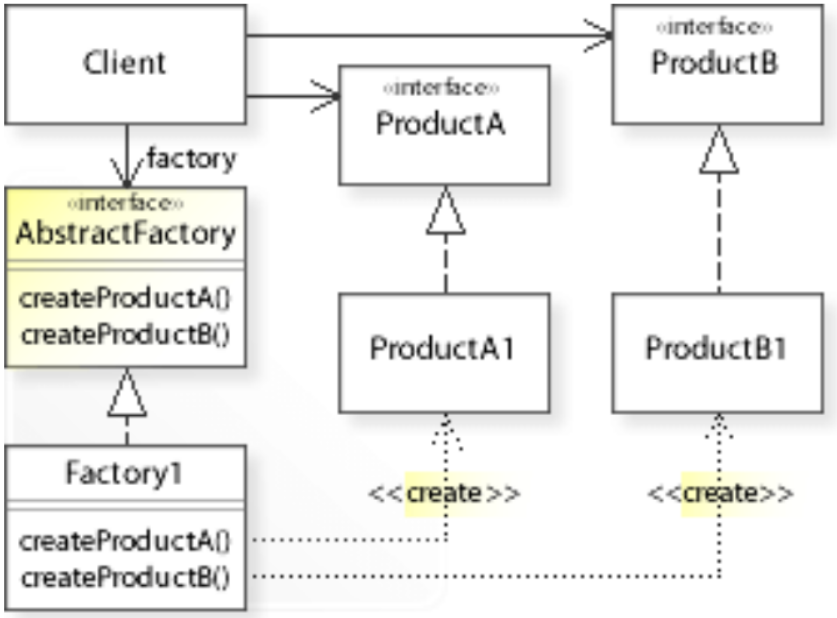
Come comporre oggetti per formare strutture complesse

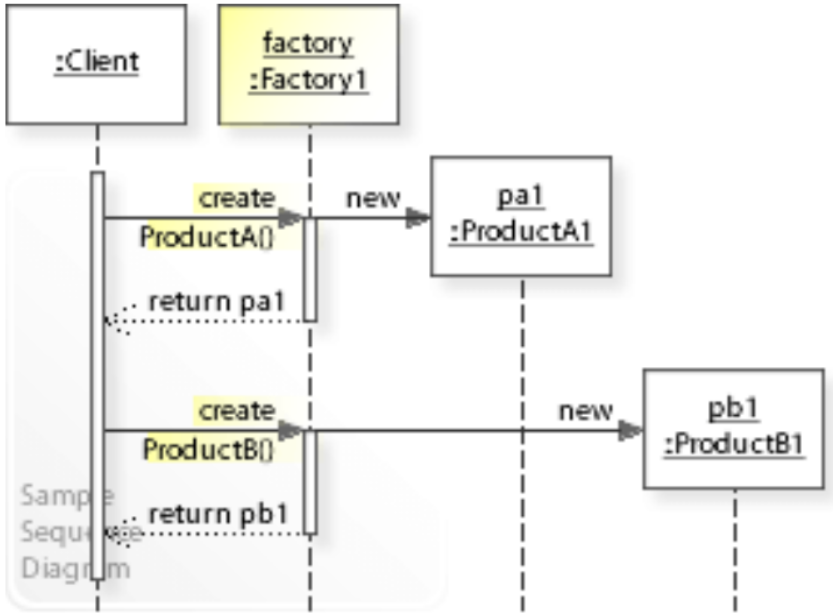
Come assegnare responsabilità a oggetti comunicanti

| Creational Patterns | Structural Patterns | Behavioral Patterns |
|---|---|---|
| Abstract Factory | Adapter | Chain of Responsibility |
| Builder | Bridge | Command |
| Factory Method | Composite | Interpreter |
| Prototype | Decorator | Iterator |
| Singleton | Facade | Mediator |
| | Flyweight | Memento |
| | Proxy | Observer |
| | | State |
| | | Strategy |
| | | Template Method |
| | | Visitor |

# Abstract Factory

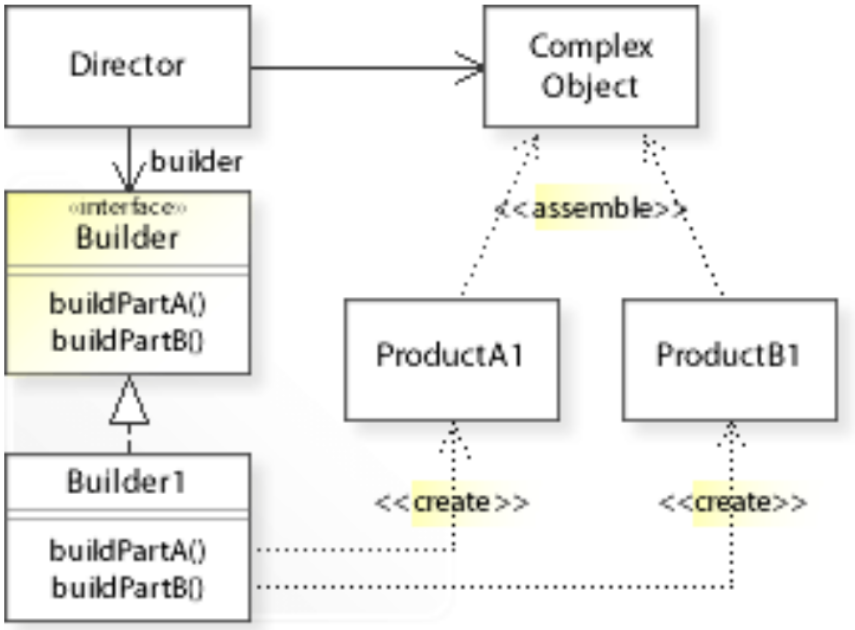
- **Problema**: come può una una classe essere indipendente da come gli oggetti che richiede sono creati?
- **Soluzione**: fornire una interfaccia per la creazione di  famiglie di oggetti correlati

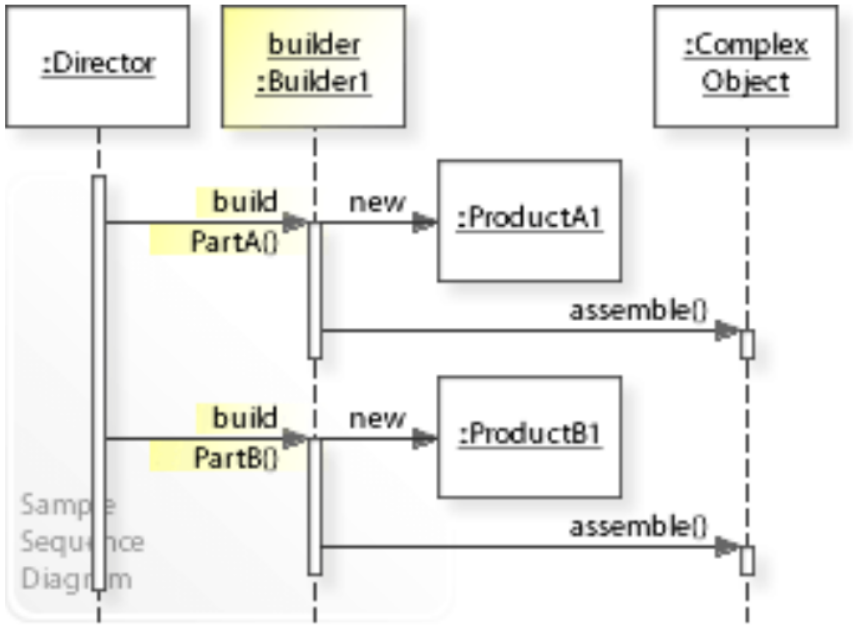


# Builder

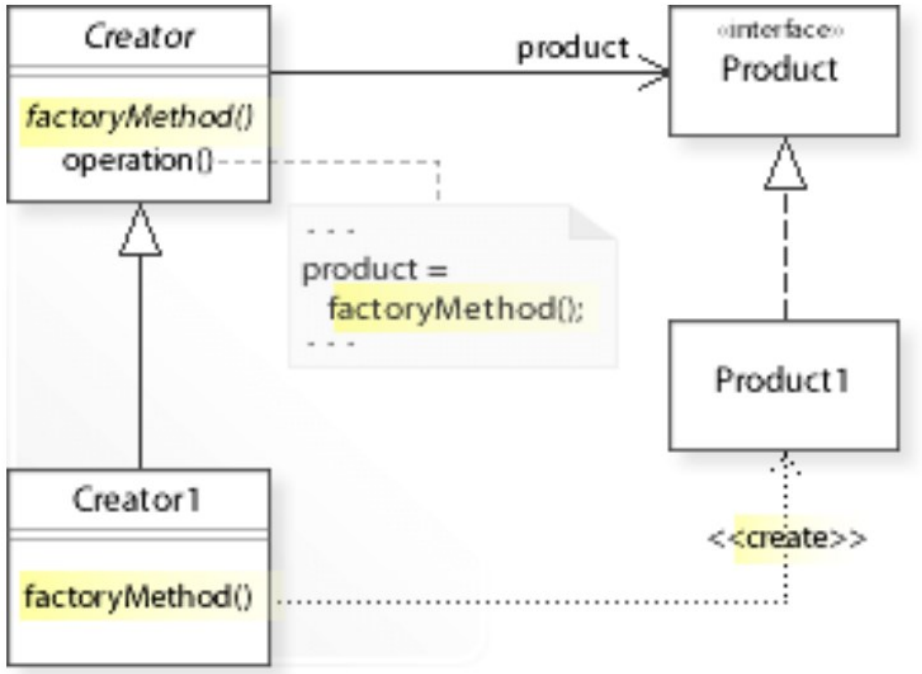
- **Problema**: come può una classe creare rappresentazioni diverse di un oggetto complesso?
- **Soluzione**: separare la costruzione di un oggetto complesso dalla sua rappresentazione in modo tale che lo stesso processo di costruzione possa creare rappresentazioni diverse



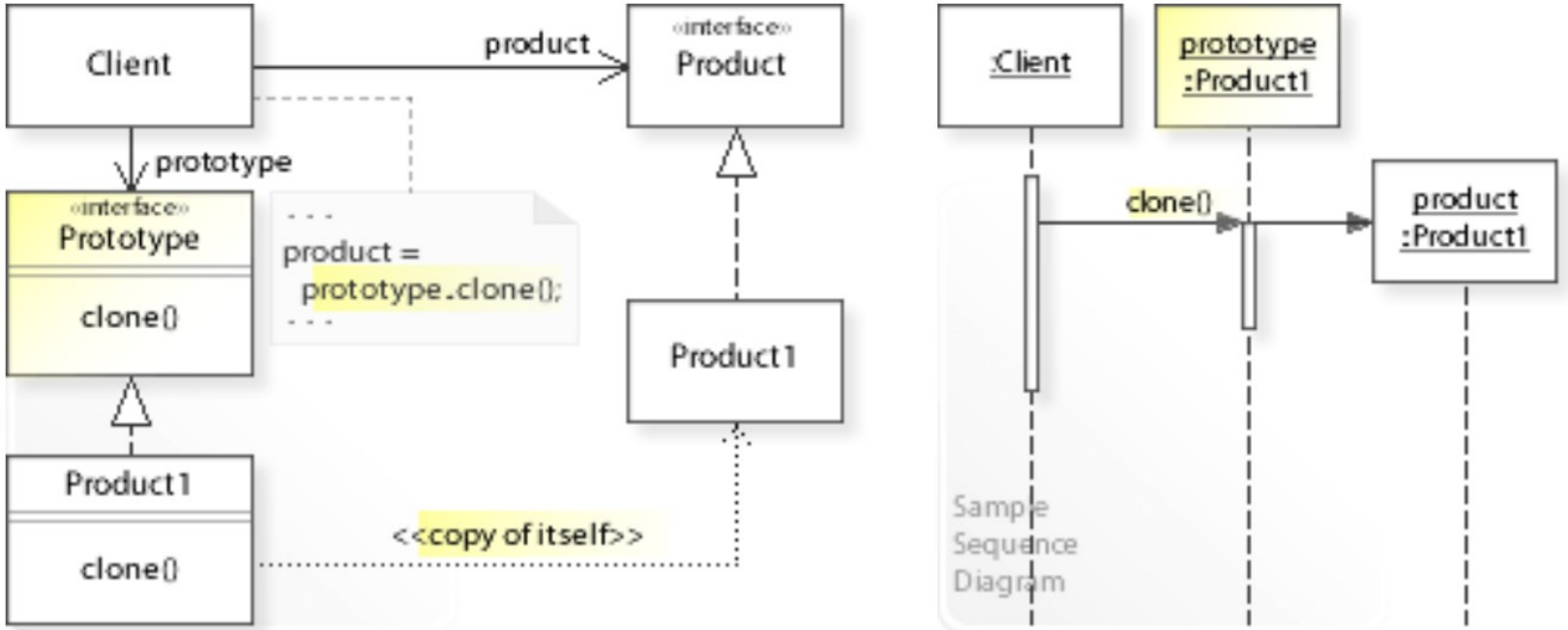
# Factory Method

- **Problema**: come può un oggetto essere creato così che le sottoclassi possano ridefinire quale classe istanziare?
- **Soluzione**: definire un'interfaccia per creare un oggetto ma lasciare che le sottoclassi decidano quale classe istanziare



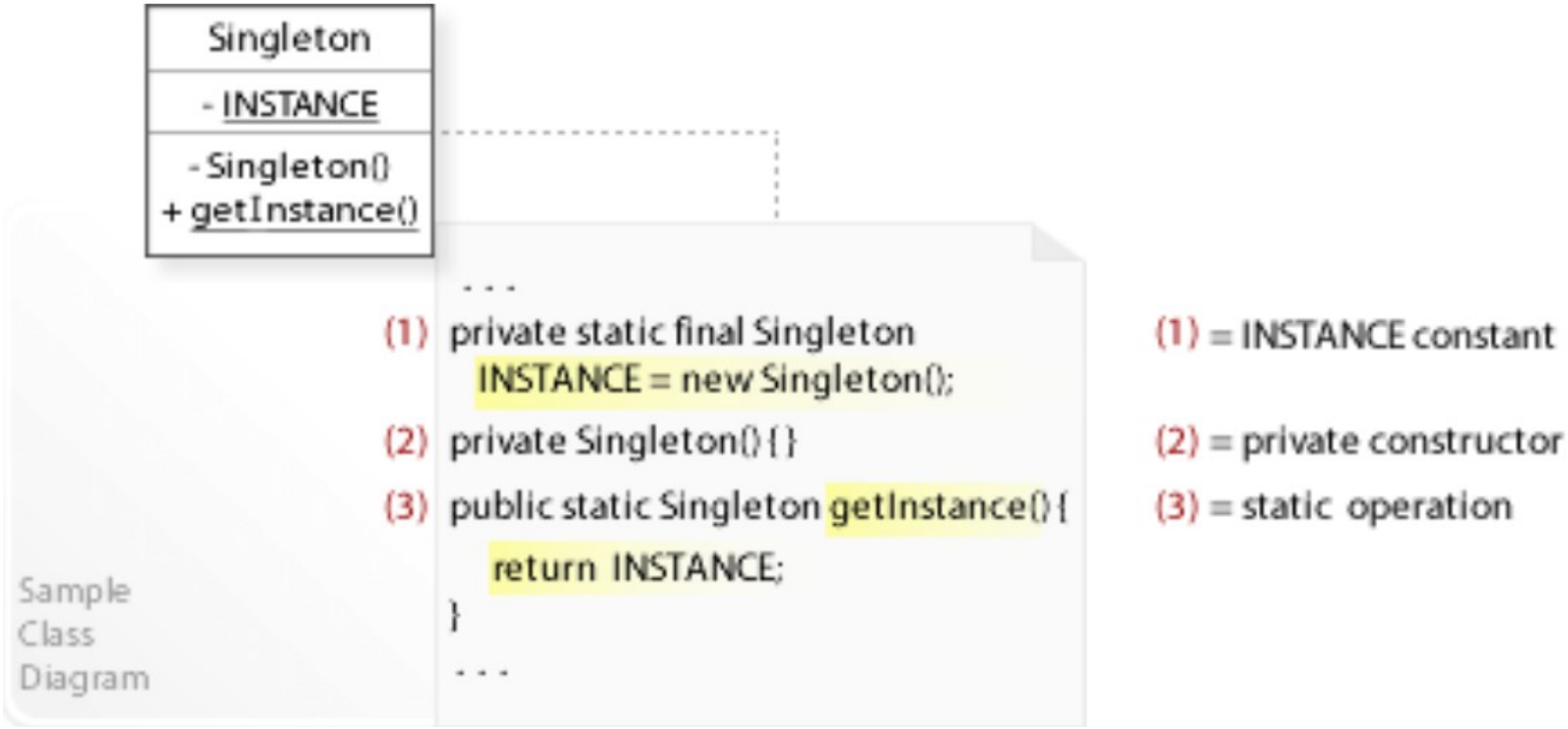
# Prototype

- **Problema**: come può un oggetto essere creato così che gli oggetti da creare possano essere specificati a run-time?
- **Soluzione**: specificare il tipo di oggetti da creare usando un'istanza prototipale e creare nuovi oggetti copiando questo prototipo



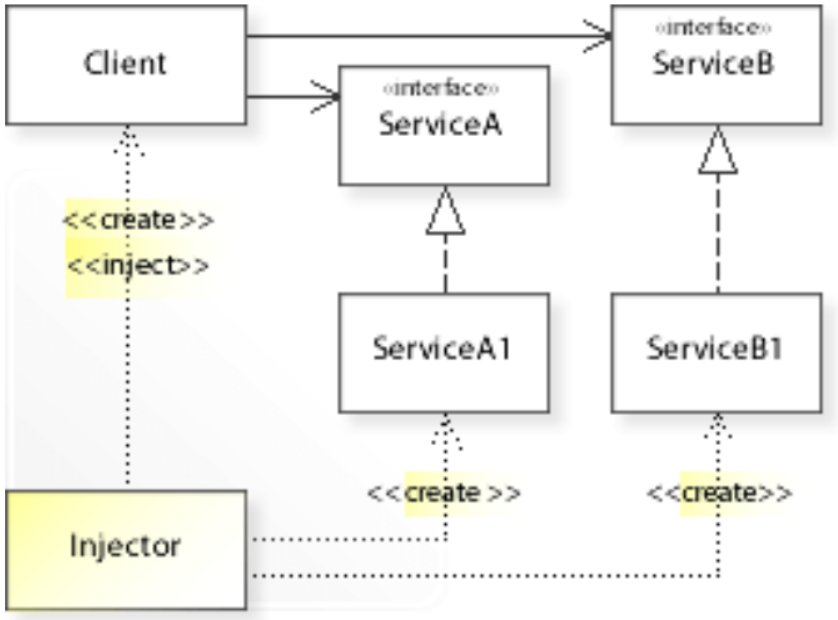
# Singleton

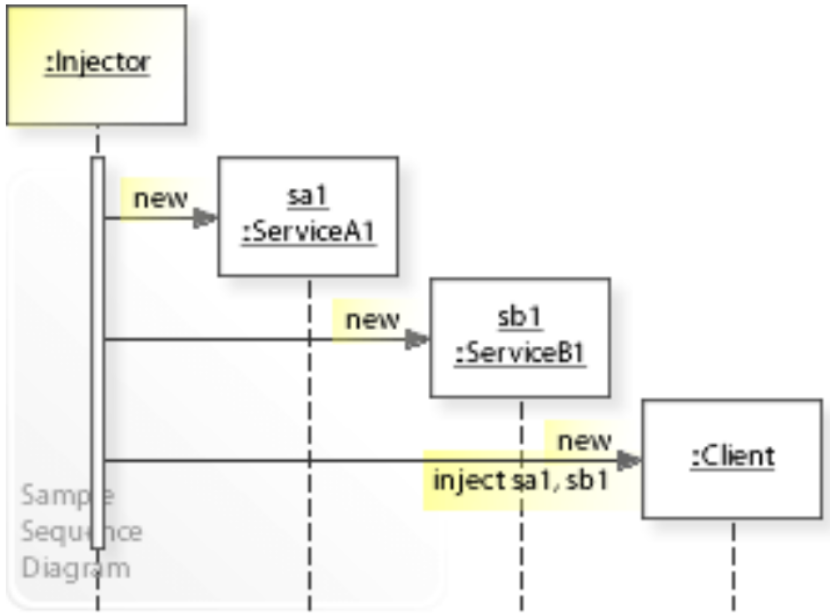
- **Problema**: come posso garantire l'univocità di un'istanza e fornire un punto d'accesso globale a tale istanza?
- **Soluzione**: nascondi il costruttore della classe e definisci un'operazione *static* che restituisce l'unica istanza della classe



# Dependency Injection (non-GoF)

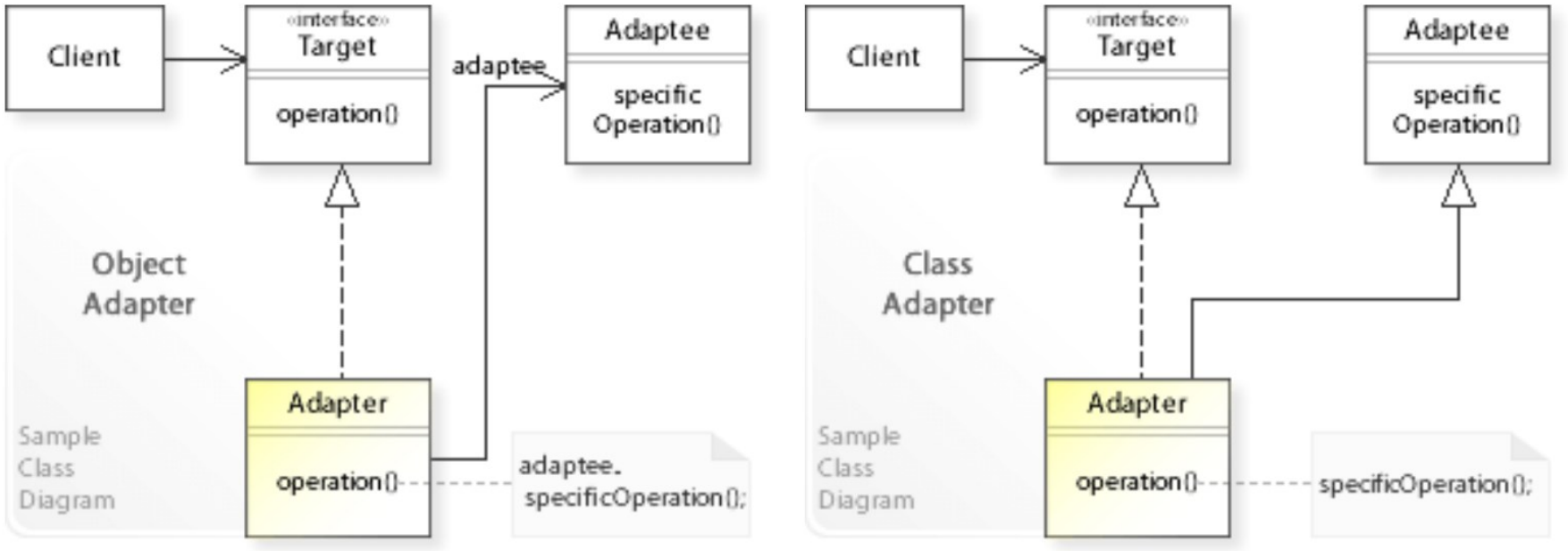
- **Problema**: come può una classe essere indipendente da come sono creati gli oggetti che richiede?
- **Soluzione**: definire un oggetto separato che crea e inietta gli oggetti che una classe richiede, usando un file di configurazione



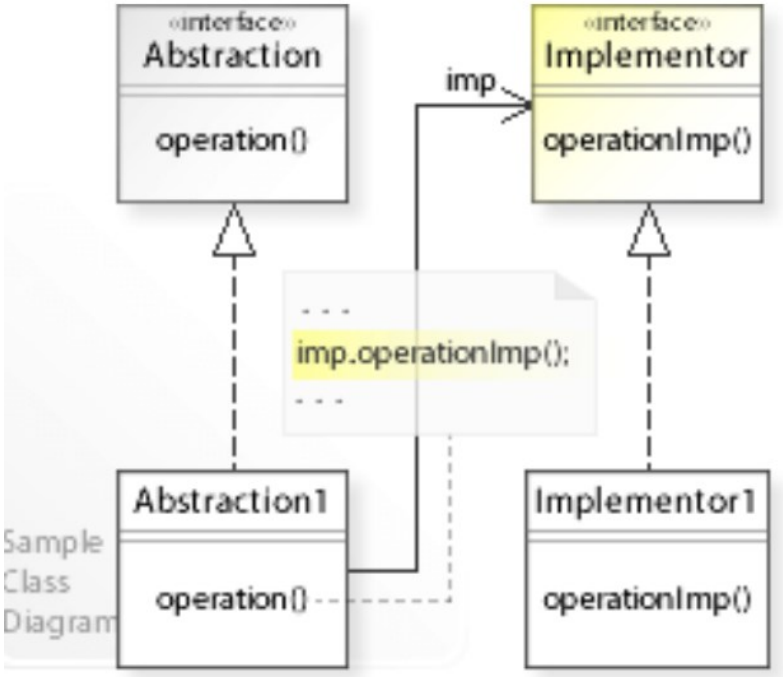
# Adapter

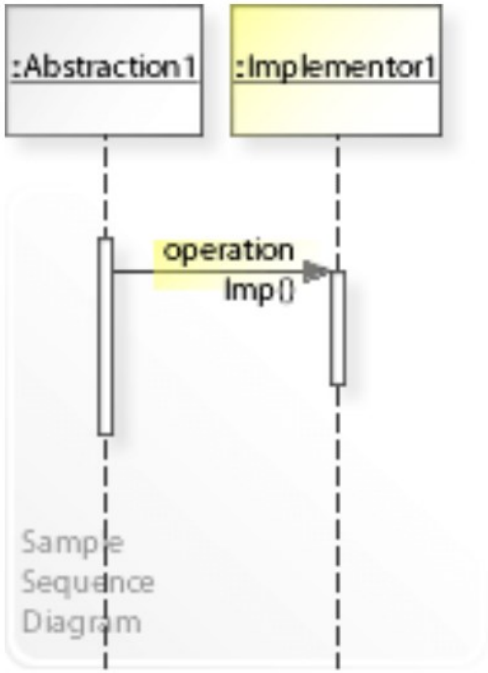
- **Problema**: come può una classe essere riusata sebbene abbia un'interfaccia incompatibile?
- **Soluzione**: converti l'interfaccia di una classe in un'altra interfaccia che il client si aspetta



# Bridge

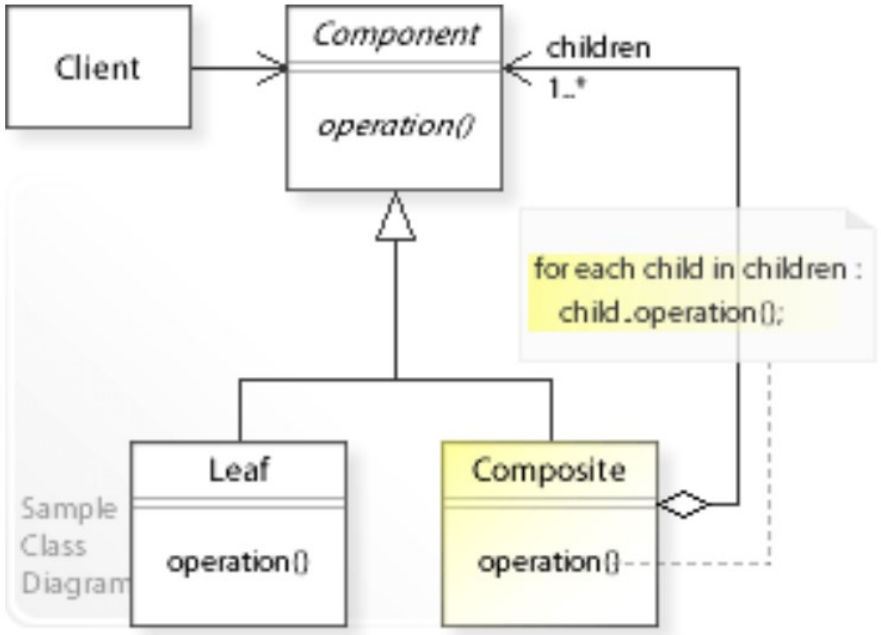
- **Problema**: come può un'implementazione essere selezionata e scambiata a run-time?
- **Soluzione**: disaccoppia un'astrazione dalla sua implementazione così che possano variare indipendentemente

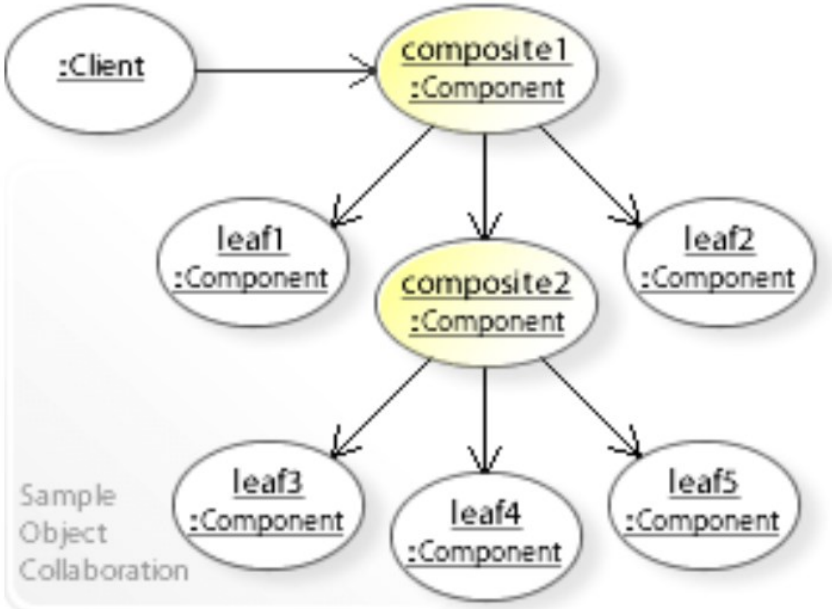


# Composite

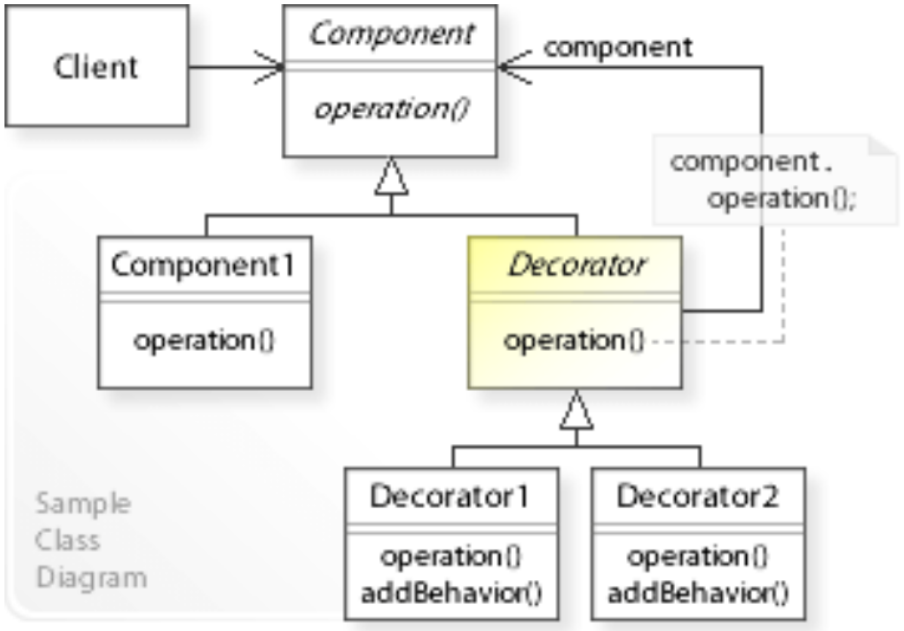
- **Problema**: come può una gerarchia di composizione essere rappresentata in modo che i client trattino gli oggetti individuali e le composizioni in modo uniforme?
- **Soluzione**: componi gli oggetti in strutture ad albero per rappresentare le gerarchie di composizione

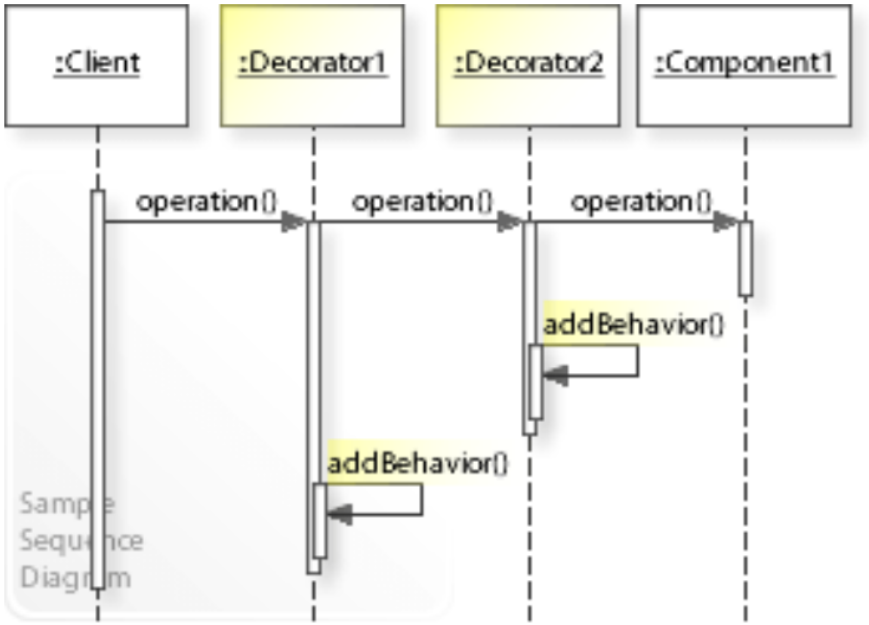


# Decorator

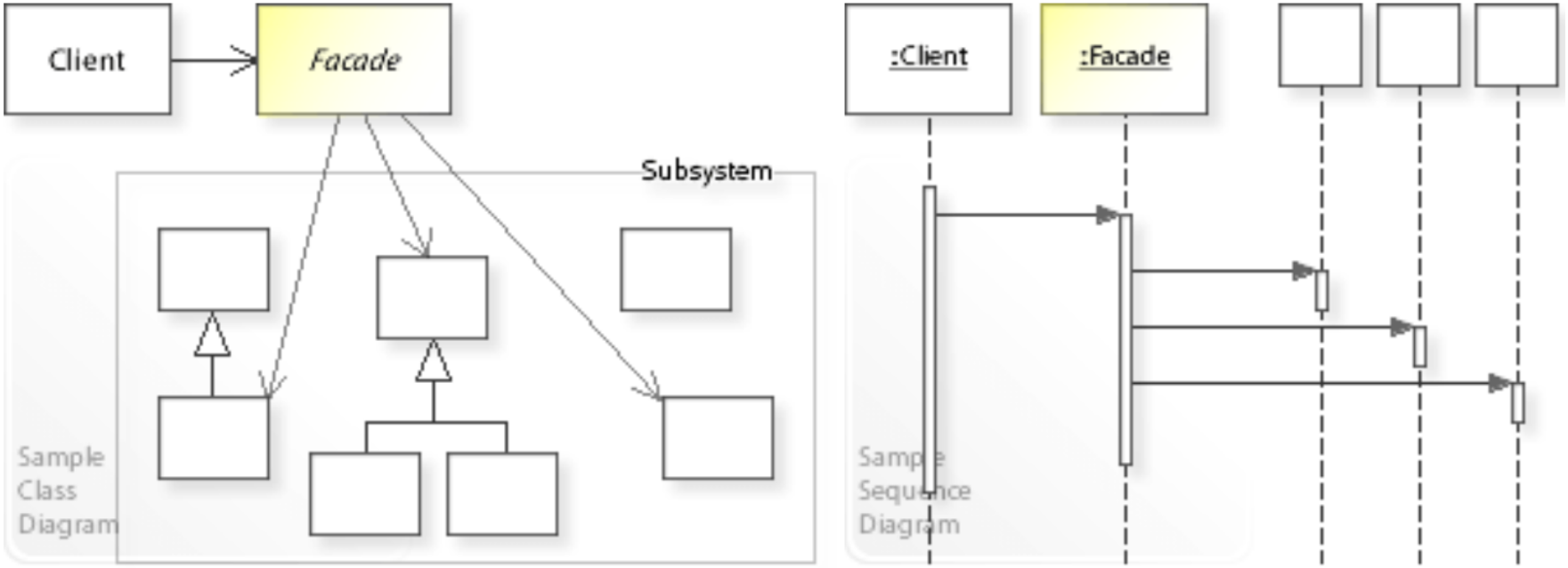
- **Problema**: come possono essere aggiunte responsabilità a un oggetto dinamicamente (senza sottoclassi)?
- **Soluzione**:  definisci oggetti Decorator che estendono le funzionalità di un oggetto a run-time



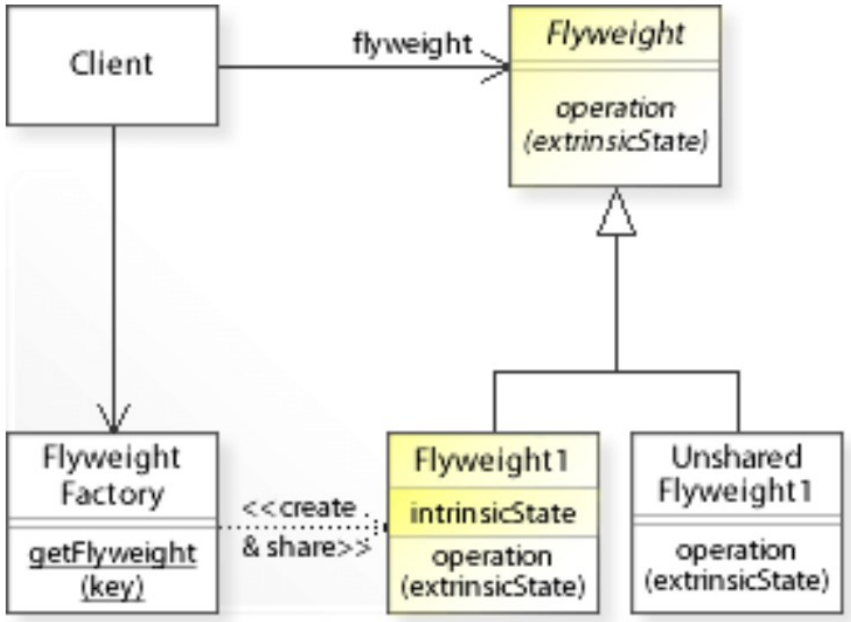
# Facade

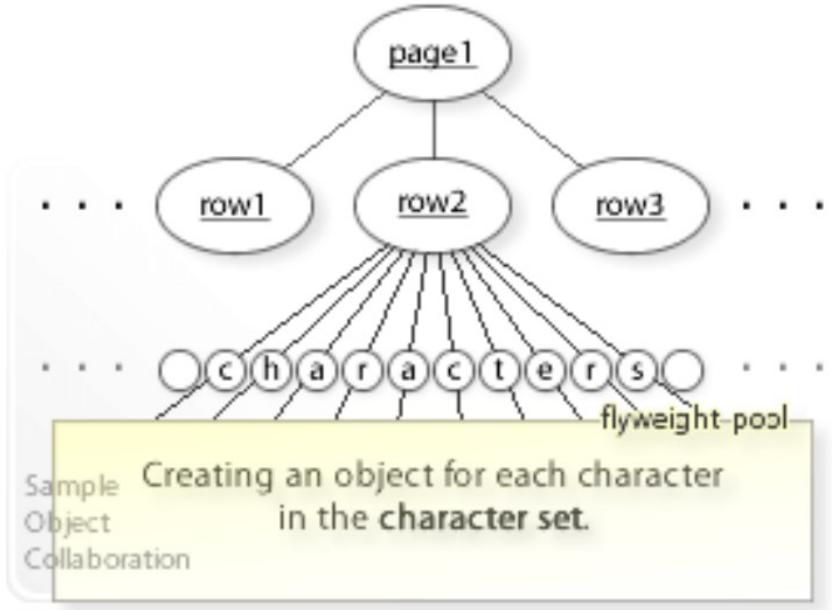
- **Problema**: come è possibile fornire un'interfaccia semplice per un sottosistema complesso?
- **Soluzione**: fornire un'interfaccia unificata di più alto livello rispetto alle interfacce del sottosistema



# Flyweight

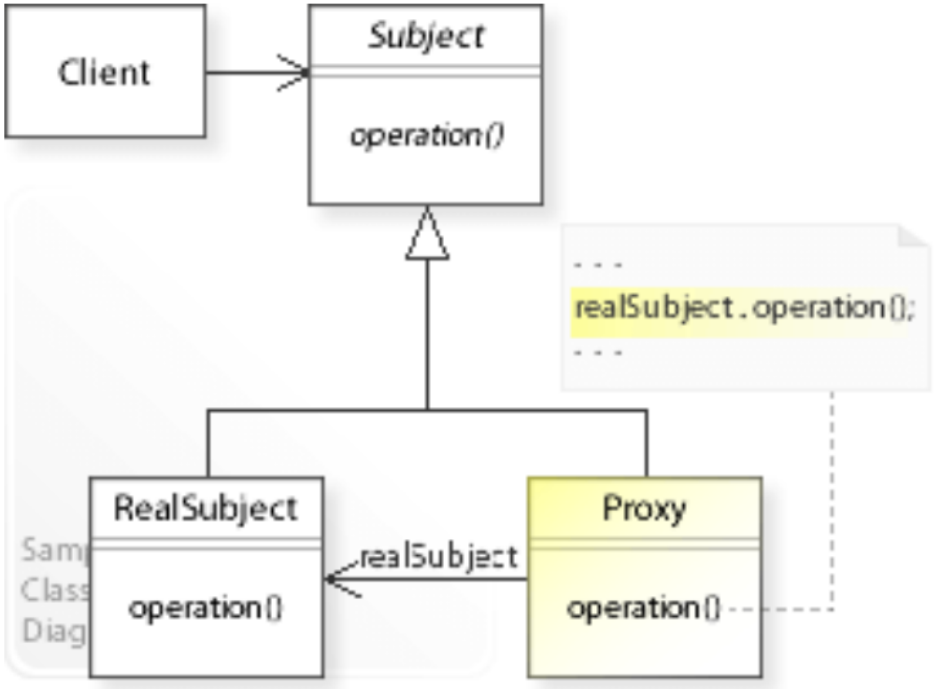
- **Problema**: come può un grande numero di oggetti a grana fine essere supportato in modo efficiente?
- **Soluzione**: usa la condivisione per memorizzare lo stato intrinseco (invariante) degli oggetti

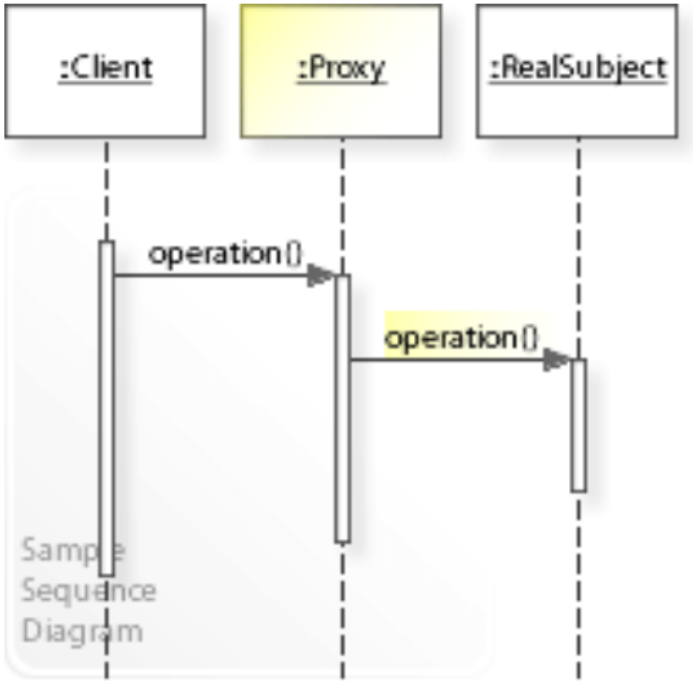


# Proxy

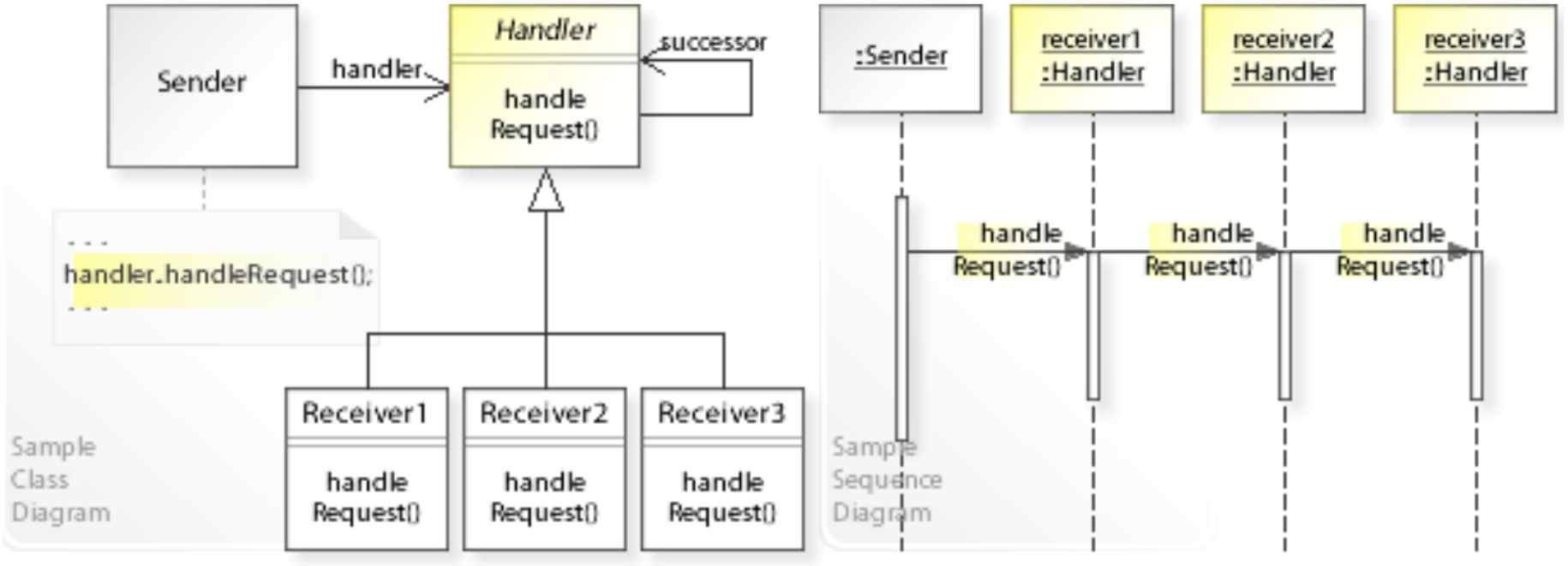
- **Problema**: come può essere controllato l'accesso a un oggetto?
- **Soluzione**: fornire un surrogato per un altro oggetto in modo da poterlo controllare



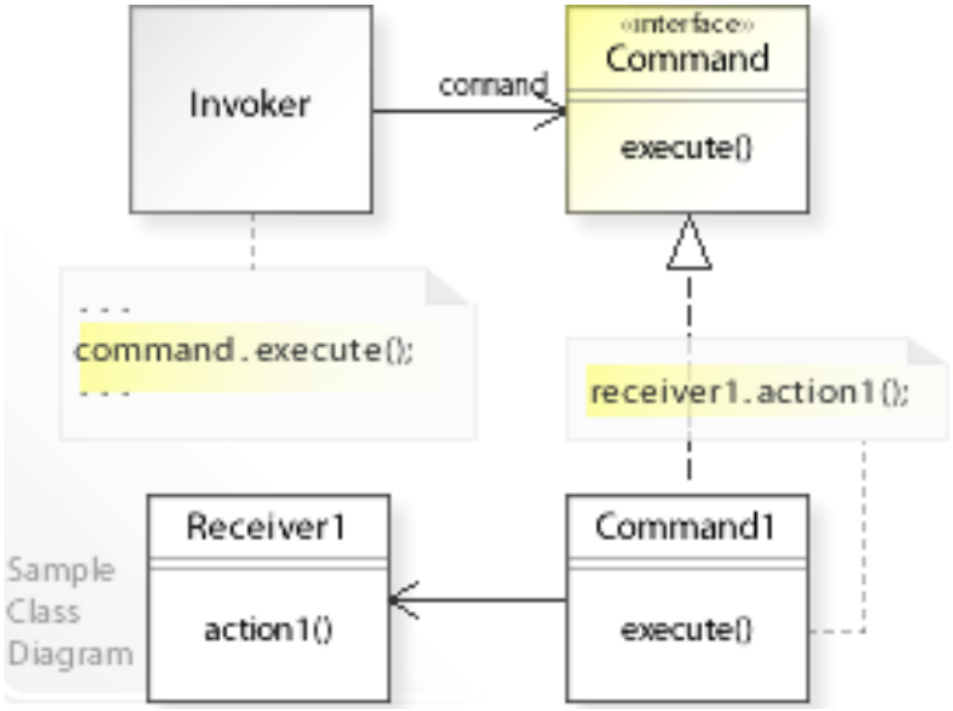
# Chain of Responsibility

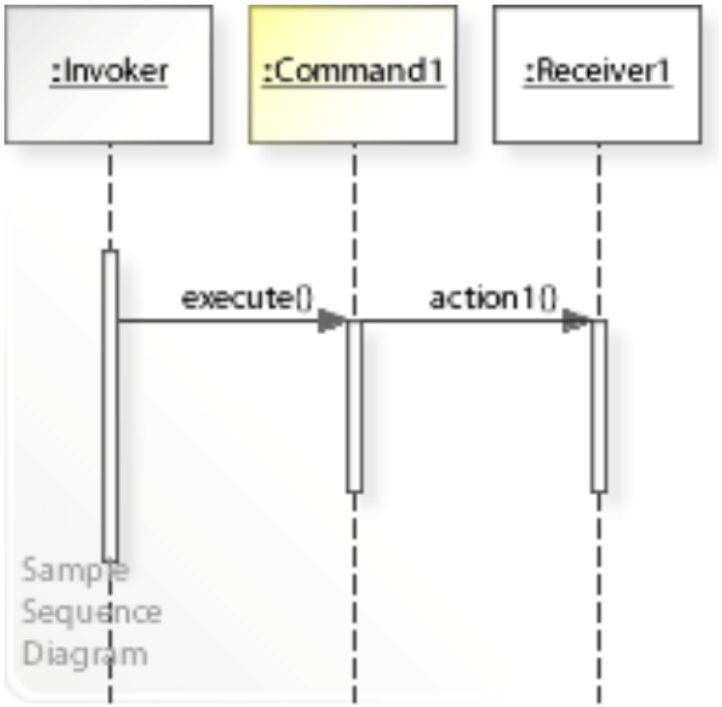
- **Problema**: come può essere evitato l'accoppiamento tra il mittente di una richiesta e il ricevente?
- **Soluzione**: dare a più di un oggetto la possibilità di gestire la richiesta, concatenando gli oggetti riceventi e passando la richiesta lungo la catena fino a che un oggetto la prende in carico



# Command

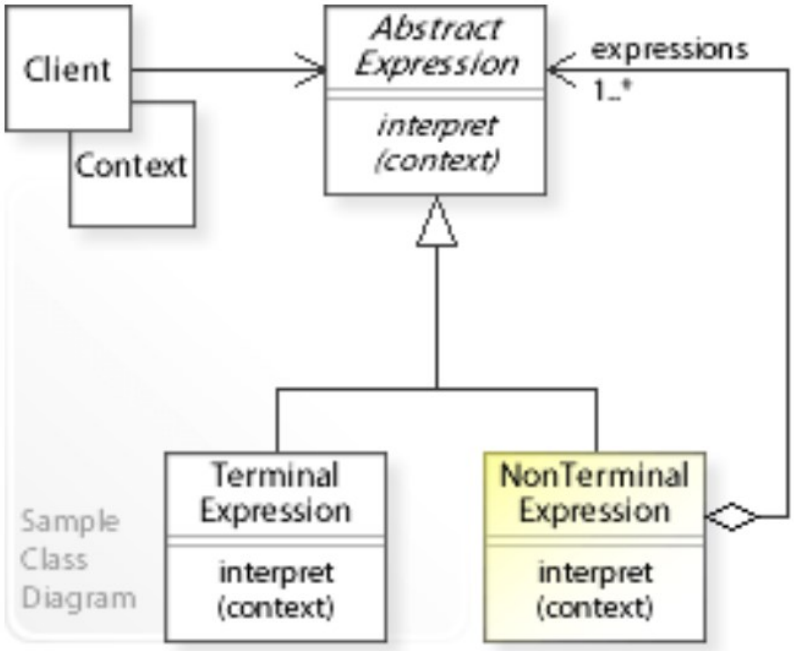
- **Problema**: come può essere evitato l'accoppiamento tra chi invoca una richiesta e chi deve soddisfare la richiesta?
- **Soluzione**: incapsulare la richiesta in un oggetto che accoda, registra le richieste e supporta l'*undo*

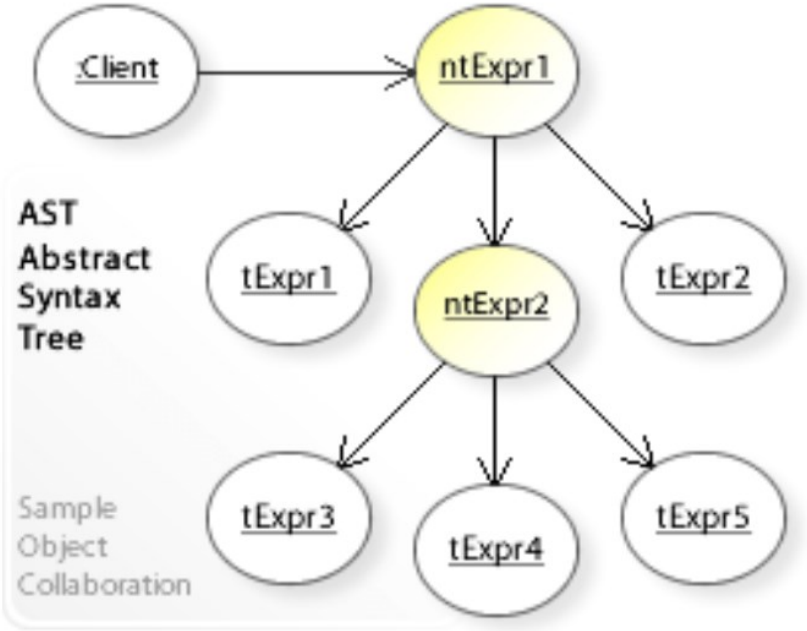


# Interpreter

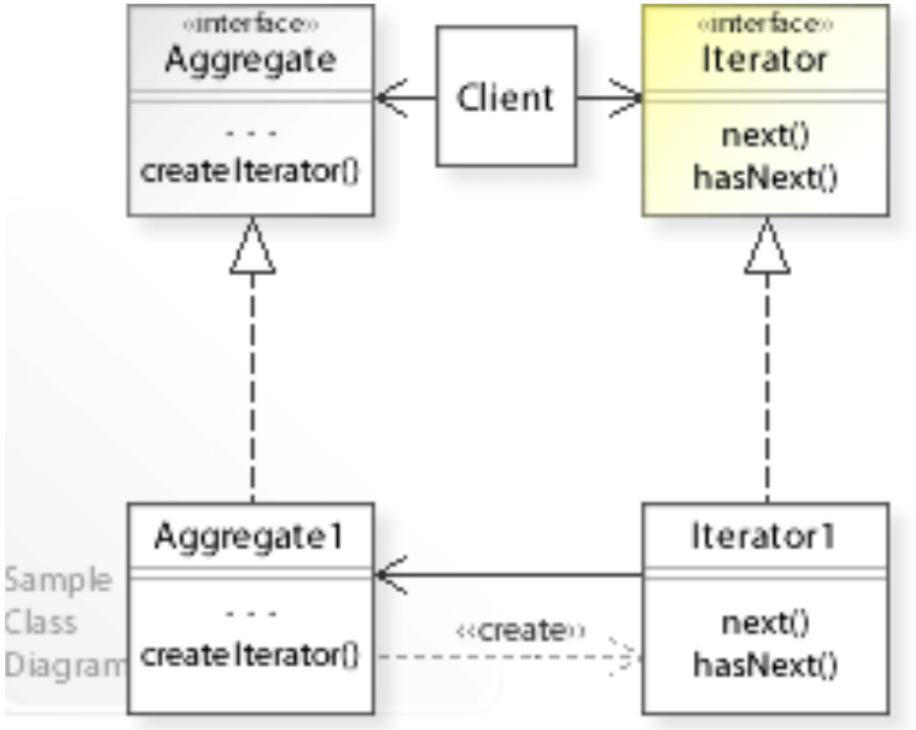
- **Problema**: come può essere definita una grammatica per un semplice linguaggio così che le frasi possano essere interpretate?
- **Soluzione**: data una grammatica del linguaggio, rappresentare una frase mediante AST (abstract sintax tree) per interpretarla

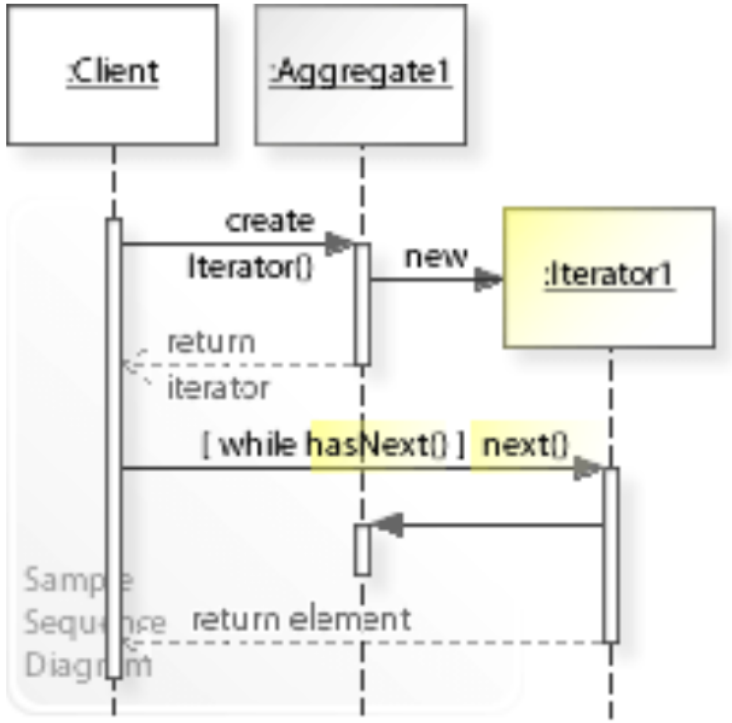


# Iterator

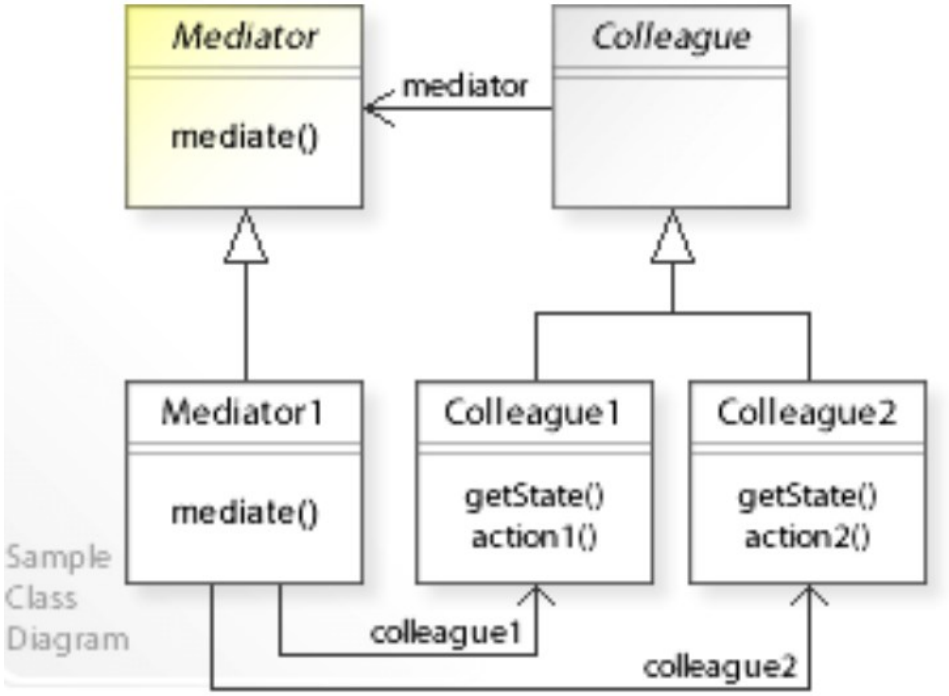
- **Problema**: come è possibile accedere e attraversare gli elementi di un oggetto aggregato senza esporre la sua rappresentazione?
- **Soluzione**: incapsulare in un oggetto separato l'accesso e l'attraversamento di un oggetto aggregato

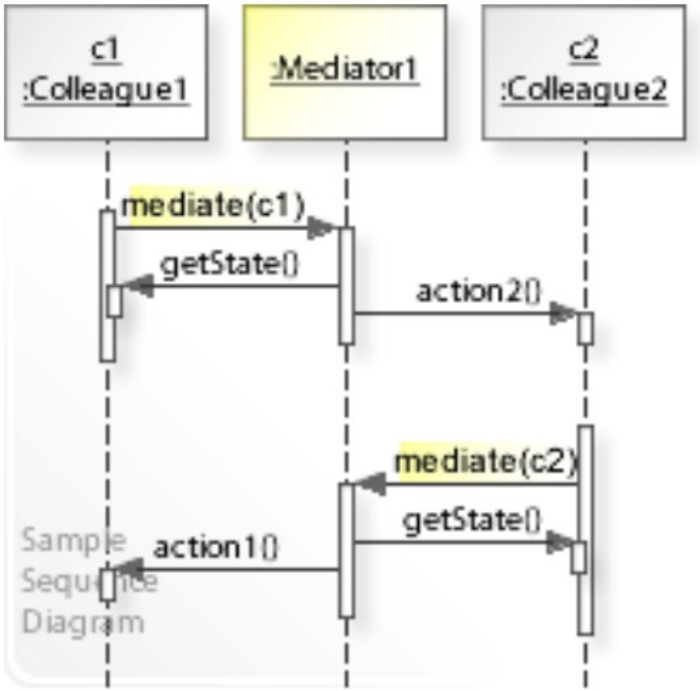


# Mediator

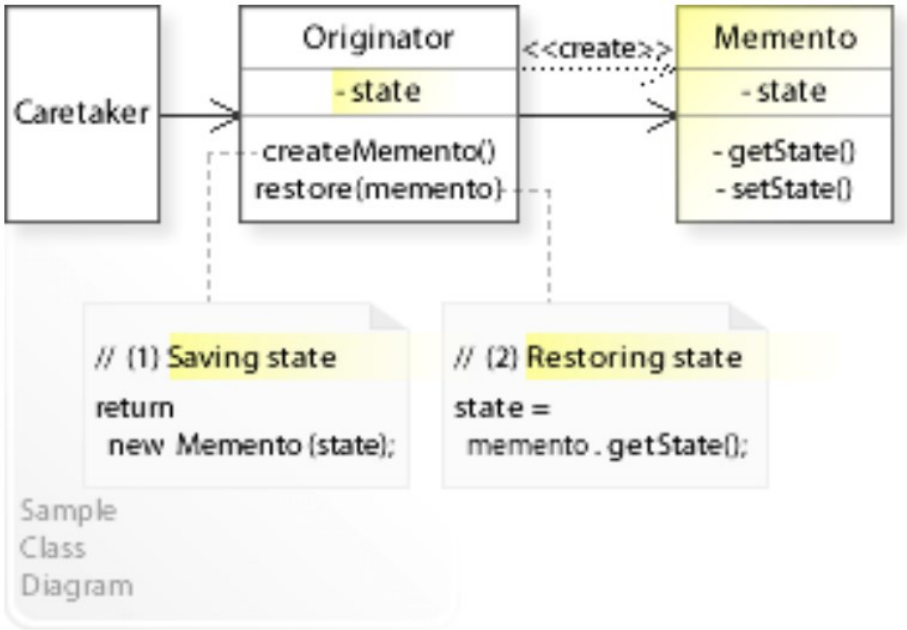
- **Problema**: come è possibile evitare uno stretto accoppiamento tra un insieme di oggetti che interagiscono tra loro?
- **Soluzione**: definire un oggetto che incapsula come interagiscono gli oggetti

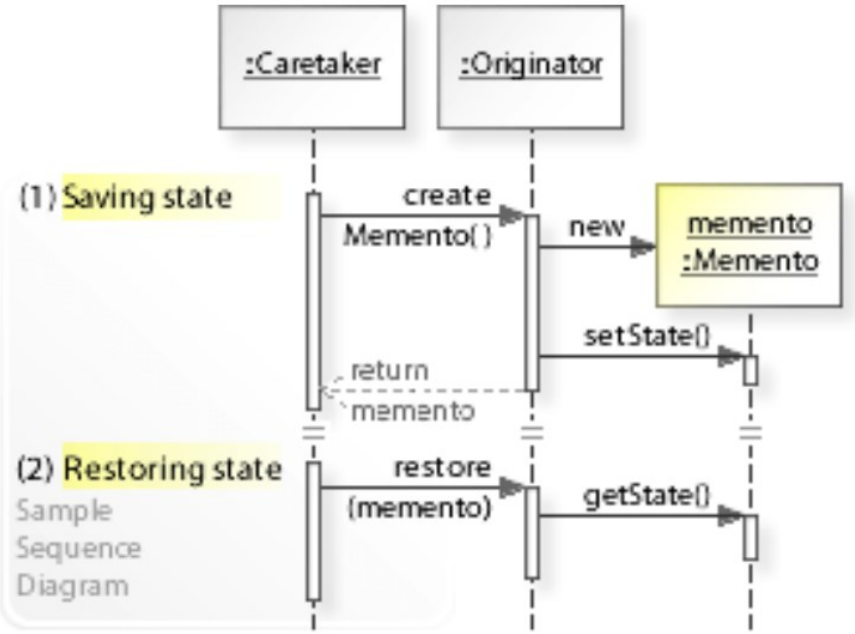


# Memento

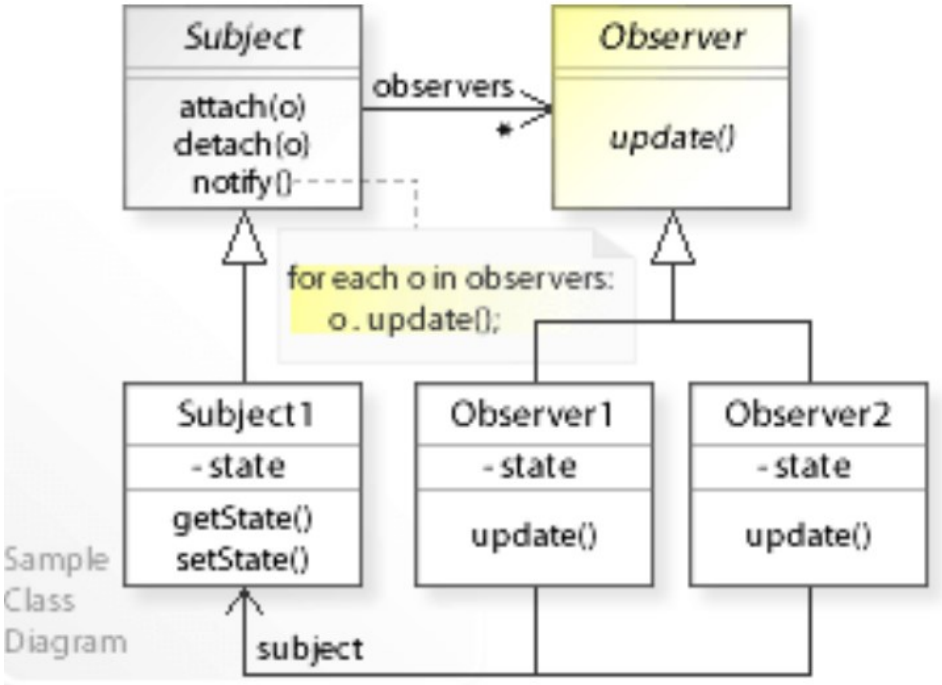
- **Problema**: senza violare l'incapsulamento, come può lo stato interno di un oggetto essere catturato ed esternalizzato in modo tale che l'oggetto possa essere successivamente ripristinato a quello stato
- **Soluzione**: l'oggetto crea un altro oggetto per salvare il suo stato e accede successivamente per ripristinare lo stato

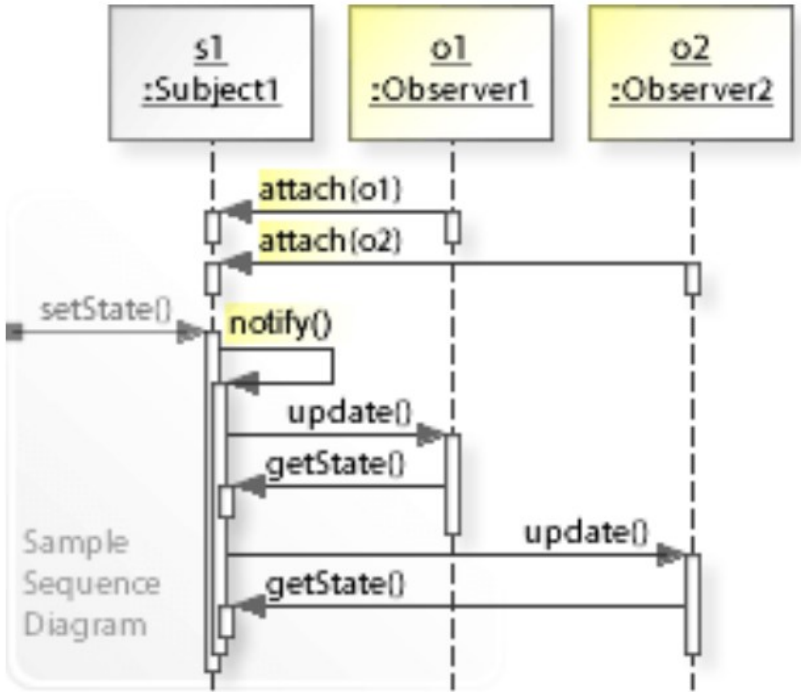


# Observer

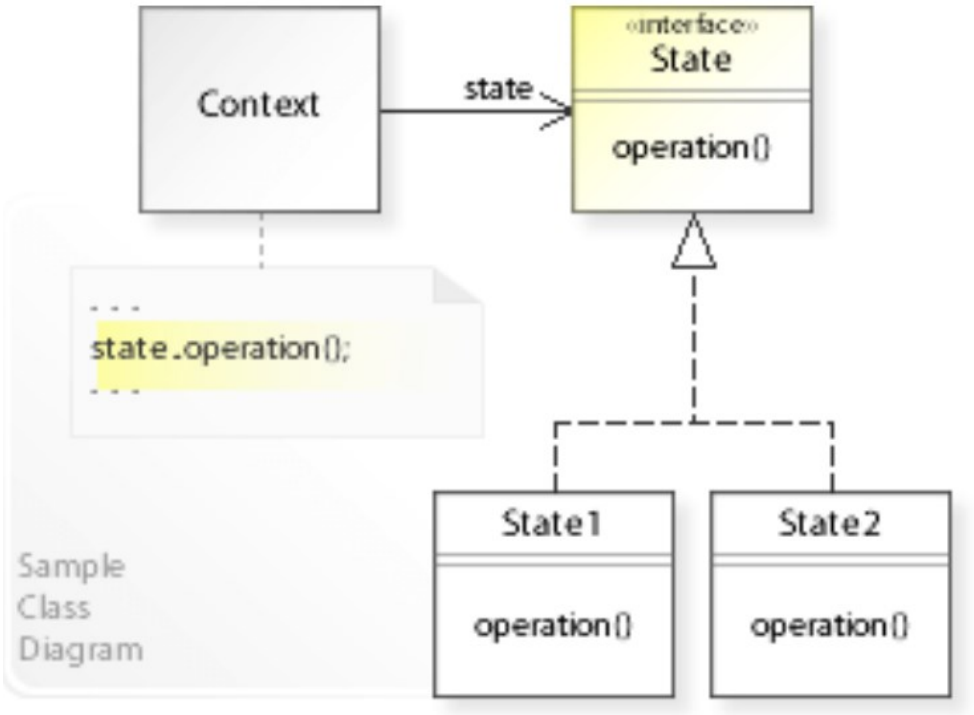
- **Problema**: come può un oggetto notificare un numero aperto di altri oggetti?
- **Soluzione**: definire un meccanismo di *publish-subscribe* in modo tale quando un oggetto cambia stato tutti gli oggetti che dipendono da esso sono notificati e aggiornati automaticamente

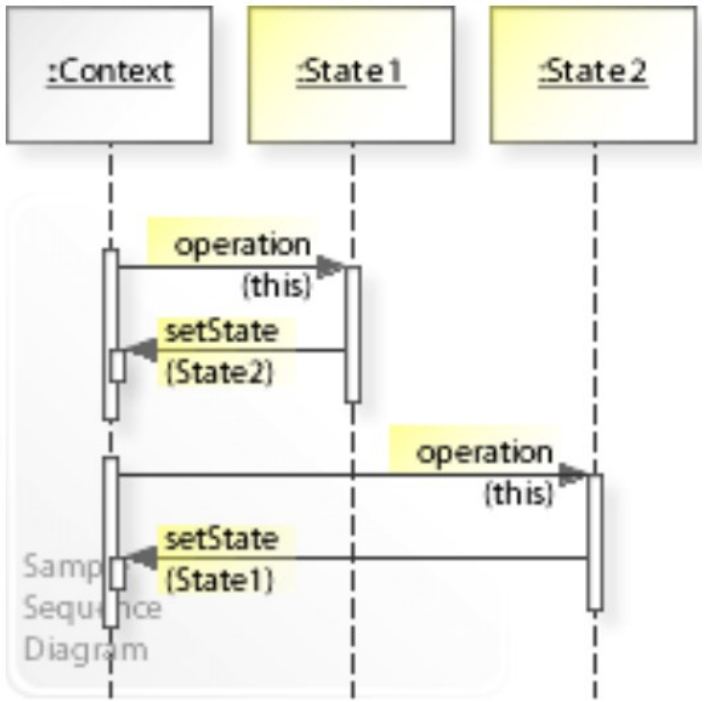


# State

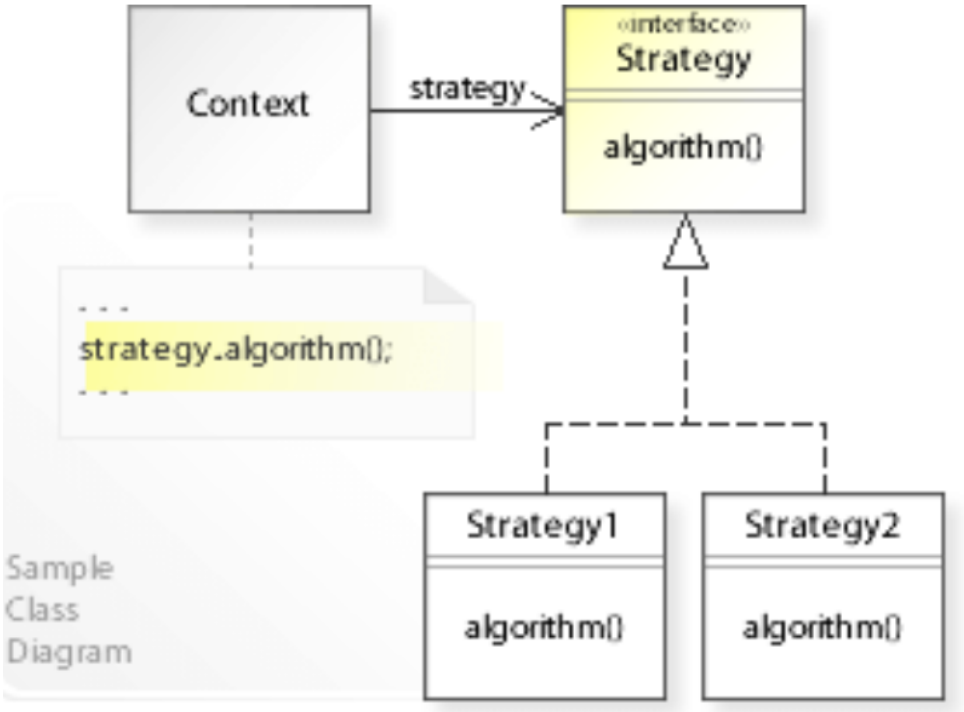
- **Problema**: come può un oggetto cambiare il suo comportamento quando cambia il suo stato interno?
- **Soluzione**: incapsulare il comportamento che dipende dallo stato in un oggetto separato

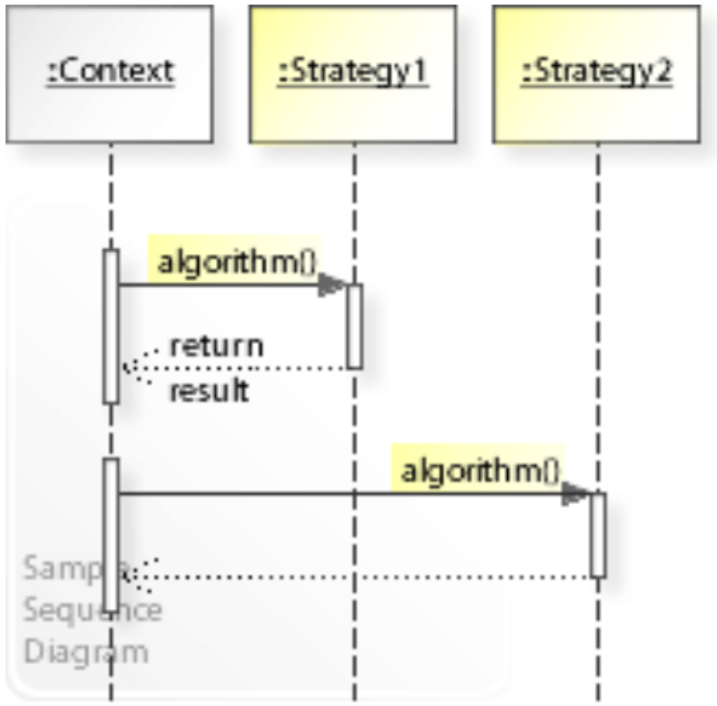


# Strategy

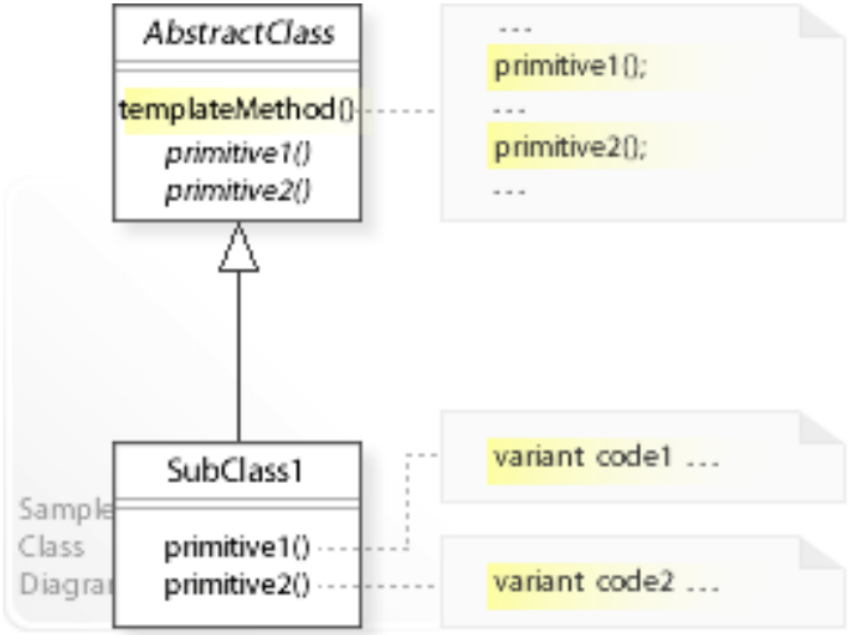
- **Problema**: come può un algoritmo essere selezionato e scambiato a run-time?
- **Soluzione**: definire una famiglia di algoritmi, incapsularli in una gerarchia di classi in modo tale da renderli intercambiabili



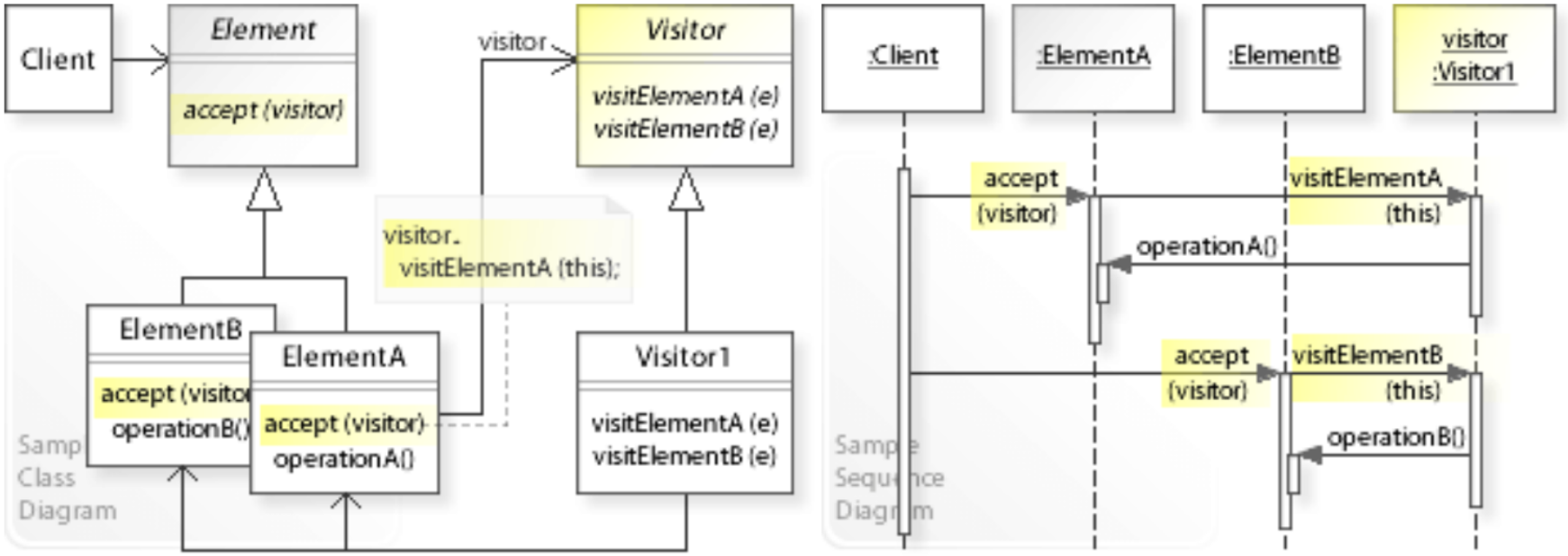
# Template Method

- **Problema**: come possono le sottoclassi ridefinire le parti varianti di un comportamento senza modificare la struttura del comportamento ereditato?
- **Soluzione**: definire in un'operazione la struttura dell'algoritmo rinviando alcuni passi a operazioni delle sottoclassi (*inversion of control*)

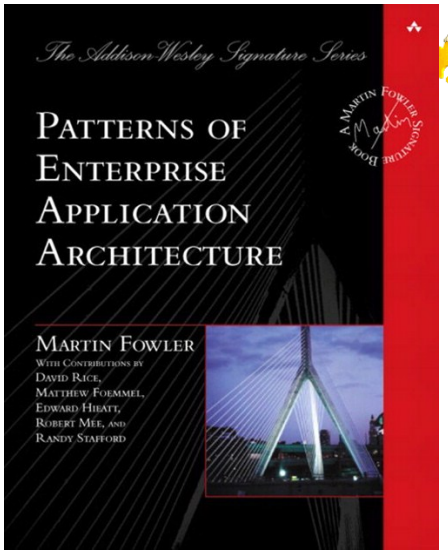


# Visitor

- **Problema**: come possono essere definite nuove operazioni sugli elementi di una struttura a oggetti senza cambiare le classi?
- **Soluzione**: definire un oggetto separato che implementa le operazioni che devono essere eseguite sugli elementi di una struttura a oggetti



# P of EAA




- Domain Logic Patterns
  - Transaction Script, Domain Model, Table Module, Service Layer
- Data Source Architectural Patterns
  - Table Data Gateway, Row Data Gateway, Active Record, Data Mapper
- Object-Relational Behavioral Patterns
  - Unit of Work, Identity Map, Lazy Load
- Object-Relational Structural Patterns
  - Identity Field, Foreign Key Mapping, Association Table Mapping, Dependent Mapping, Embedded Value, Serialized LOB, Single Table Inheritance, Class Table Inheritance, Concrete Table Inheritance, Inheritance Mappers
- Object-Relational Metadata Mapping Patterns
  - Metadata Mapping, Query Objec, Repository
- Web Presentation Patterns
  - Model View Controller, Page Controller, Front Controller, Template View, Transform View, Two-Step View, Application Controller
- Distribution Patterns
  - Remote Facade, Data Transfer Object
- Offline Concurrency Patterns
  - Optimistic Offline Lock, Pessimistic Offline Lock, Coarse Grained Lock, Implicit Lock
- Session State Patterns
  - Client Session State, Server Session State, Database Session State
- Base Patterns
  - Gateway, Mapper, Layer Supertype, Separated Interface, Registry, Value Object, Money, Special Case, Plugin, Service Stub, Record Set