Introduzione ai Design Pattern

Sandro Morasca

Università degli Studi dell'Insubria

Dipartimento di Scienze Teoriche e Applicate

Via Carloni 78

22100 Como

sandro.morasca@uninsubria.it

Design pattern

- Introduzione
- Definizioni
- Categorie
- Factory
- Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Observer
- Pattern architetturali
- MVC
- Esempio

- I problemi incontrati nello sviluppo di grossi progetti software sono spesso ricorrenti e prevedibili
- I **design pattern** sono "schemi di soluzioni" riutilizzabili
- Introdotti dall'architetto (di edifici, non di software) Christopher Alexander.

  "Ogni pattern descrive un problema che si presenta frequentemente nel nostro ambiente, e quindi descrive il nucleo della soluzione così che si possa impiegare tale soluzione milioni di volte, senza peraltro produrre due volte la stessa realizzazione."
- Per noi il principio è ugualmente valido, anche se riferito ad oggetti, classi ed interfacce piuttosto che ad elementi architettonici come muri, archi, pilastri, ecc.

Vantaggi

- Introduzione
- Definizioni
- Categorie
- Factory
- Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Observer
- Pattern architetturali
- MVC
- Esempio

- Che cosa fornisce un design pattern al progettista software?
  - una soluzione consolidata per un problema ricorrente
  - un'astrazione di granularità e livello di astrazione più elevati di una classe
  - un modo per progettare software con caratteristiche predefinite
  - un supporto alla progettazione di sistemi complessi
  - un modo per gestire la complessità del software

Vantaggi

- Introduzione
- Definizioni
- Categorie
- Factory
- Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Observer
- Pattern architetturali
- MVC
- Esempio

- Permettono di non inventare da capo soluzioni ai problemi già risolti, ma di utilizzare dei “mattoni” di provata efficacia
  - un bravo progettista sa riconoscerli, nella documentazione o direttamente nel codice, e utilizzarli per comprendere i programmi scritti da altri
- Catturano l’esperienza e la “saggezza” degli esperti
- Forniscono quindi un vocabolario comune che facilita la comunicazione tra progettisti

Svantaggi

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Observer
- Pattern architetturali
- MVC
- Esempio

- Possono rendere la struttura del progetto/codice più complessa del necessario
  - di volta in volta bisogna decidere se adottare semplici soluzioni ad-hoc o riutilizzare pattern noti
  - ricordare i seguenti motti
    - **when in doubt, leave it out**
    - **keep it simple**

Definizione

Introduzione
➢ Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Definizione:

  descrizione di oggetti e classi comunicanti adattabili per risolvere un problema ricorrente di progettazione in un contesto specifico

- Abbastanza astratti
  - in modo da poter essere condivisi da progettisti con punti di vista diversi
- Non complessi e non domain-specific
  - non rivolti alla specifica applicazione ma riusabili in parti di applicazioni diverse

Caratteristiche

Introduzione
➢ Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Un Design Pattern
  - nomina
  - astrae
  - identifica

  gli aspetti chiave di una struttura comune di design che la rendono utile nel contesto del riuso in ambito object-oriented

- Un Design Pattern identifica
  - le classi (e le istanze) partecipanti
  - le associazioni ed i ruoli
  - le modalità di collaborazione tra le classi coinvolte
  - la distribuzione delle responsabilità nella soluzione del particolare problema di design considerato

Nome

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Identifica il pattern
- Un buon nome cattura l'essenza del pattern
- Fornisce un mezzo di espressione comune
- Costituisce un riferimento nel catalogo dei pattern

Problema

Introduzione
➢ Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- Include il contesto
- Descrive quando e perché è utile applicare un pattern
- Può contenere una lista di condizioni di applicabilità

Soluzione

Introduzione
➢ Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- Descrizione astratta della soluzione
- Elementi di design, relazioni, collaborazioni

Conseguenze

Introduzione
➢ Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Risultati dell'applicazione del pattern
- Utili per valutare i pro e contro della soluzione proposta
- Sempre ad un livello abbastanza astratto, ad esempio conseguenze e limiti rispetto a
  - flessibilità del sistema complessivo
  - estensibilità
  - portabilità

Esempi

Introduzione
➢ Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

Contesto	Problema	Soluzione	Conseguenze
• sviluppare un'interfaccia utente	• variabilità interfaccia utente	• separare la GUI nei componenti X, Y e Z	• > modificabilità • > efficienza • > flessibilità • > estensibilità
• accedere ai servizi di un server	• efficienza e trasparenza dell'accesso	• realizzare un intermediario	
• mostrare diverse viste dei dati	• flessibilità nel meccanismo delle viste	• separare le funzionalità di gestione delle viste da quelle di presentazione dei dati	

Il catalogo dei design pattern

Introduzione
Definizioni
➢ Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Il testo fondamentale sui design pattern è il libro “Design Patterns - Elements of Reusable Object-Oriented Software” by E. Gamma, R. Helm, R. Johnson, J. Vlissides, noti come “the gang of four”
- In questo libro è riportato il catalogo dei pattern, cioè un elenco di pattern con la loro descrizione
  - il catalogo consta di 23 pattern
  - altri pattern sono stati descritti più recentemente dalla comunità dei ricercatori e dei progettisti che fanno uso di pattern

Organizzazione del catalogo

Introduzione
Definizioni
➢ Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Esistono diverse categorie di pattern, che descrivono la funzione (purpose) e il dominio (scope) del pattern
- Funzione (purpose), ovvero cosa fa il pattern
  - Creazionali (creational): forniscono meccanismi sofisticati per la creazione di oggetti
  - Strutturali (structural): gestiscono la separazione tra interfaccia e implementazione e le modalità di composizione tra oggetti
  - Comportamentali (behavioral): consentono la modifica del comportamento degli oggetti minimizzando le necessità di cambiare il codice

Organizzazione del catalogo

Introduzione
Definizioni
➢ Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Dominio (scope), indica se il pattern si applica a classi o oggetti
  - i class pattern contemplano classi e loro relazioni
    - tipicamente si riferiscono a situazioni statiche, ovvero riguardano il compile-time.
  - gli object pattern riguardano oggetti (istanze) e loro relazioni
    - tipicamente si riferiscono a situazioni dinamiche (le relazioni tra oggetti possono ovviamente cambiare a run-time)

Categorie di design pattern

Introduzione
Definizioni
➢ Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Natura delle diverse categorie

Introduzione
Definizioni
➢ Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- *Creational class pattern* rimandano parte della creazione di oggetti ad altre classi (sottoclassi)
- *Creational object pattern* rimandano parte della creazione di oggetti ad altri oggetti
- *Structural class pattern* usano l'eredità per comporre classi
- *Structural object pattern* descrivono modalità di composizione per assemblare oggetti
- *Behavioral class pattern* usano l'eredità per descrivere algoritmi e flussi di controllo
- *Behavioral object pattern* descrivono la cooperazione di oggetti diversi per la realizzazione di un compito che nessun oggetto potrebbe svolgere da solo

Design pattern nei package Java

Introduzione
Definizioni
➢ Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer Pattern
architetturali
MVC
Esempio

- I pattern sono utilizzati largamente dalle classi standard di Java, e sono alla base della progettazione ad oggetti
- Pattern presentati:
  - Factory, Singleton, Flyweight, State, Strategy, Proxy, Adaptor e Decorator

Creazione di oggetti

Introduzione
Definizioni
Categorie
➢ Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

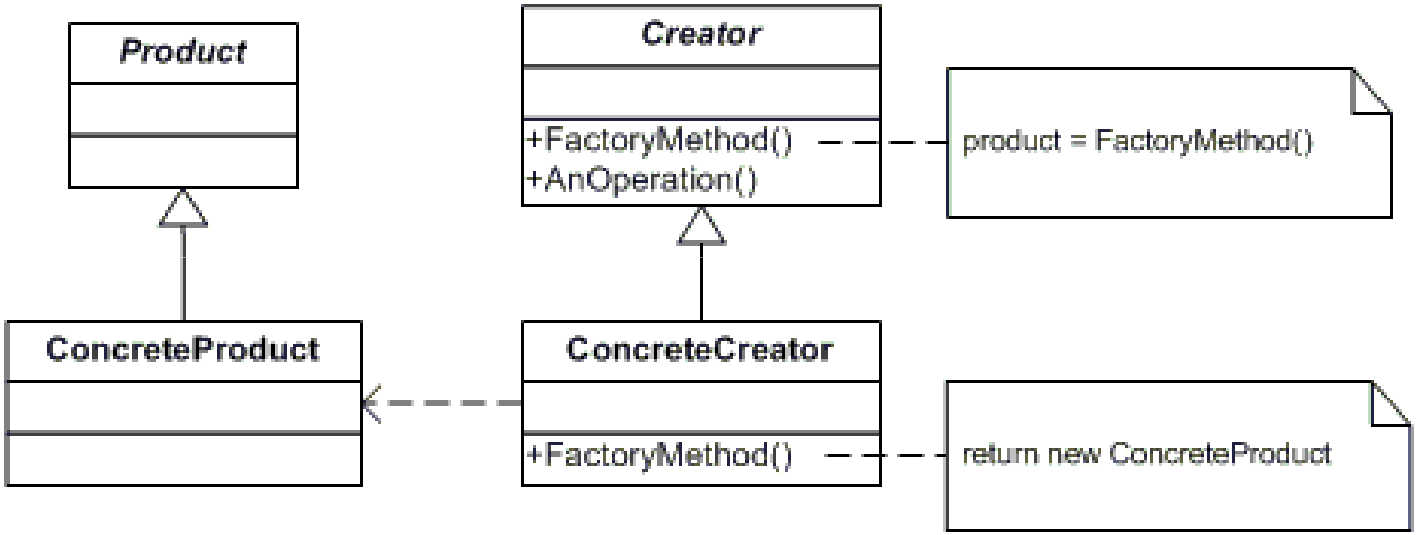
- Il codice di un programma ad oggetti per lo più non dipende dalla precisa classe cui appartiene un certo oggetto
- I programmi richiedono a un oggetto solo il rispetto del "contratto" corrispondente alla sua specifica (il suo tipo)
  - limitare le dipendenze dalle classi è desiderabile perché permette di sostituire un'implementazione con un altra
- **Eccezione: le chiamate ai costruttori**
  - il codice utente che chiama il costruttore di una determinata classe rimane vincolato a quella classe
- Per questo esistono pattern per un'operazione semplice come la creazione di un oggetto
  - disaccoppiano (separandolo) il codice che fa uso di un tipo da quello che sceglie quale implementazione del tipo (classe) utilizzare

Factory

Introduzione
Definizioni
Categorie
➢ Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- La soluzione è nascondere la creazione in un metodo detto factory: un metodo che restituisce un oggetto di una classe senza essere costruttore di quella classe
- In Java le chiamate ai costruttori non sono personalizzabili
- Una factory può invece scegliere la strategia di allocazione

Introduzione
Definizioni
Categorie
➢ Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio



Factory

Introduzione
Definizioni
Categorie
➢ Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Il metodo può creare oggetti di classi diverse a seconda dei parametri, ma tutti questi oggetti avranno lo stesso tipo
  - Esempio: un polinomio del tipo ax^n+b viene implementato da una classe SparsePoly, mentre il polinomio generico è un esemplare di DensePoly

```
public static Poly createPoly (int[ ] a) {
  int degree = -1, numCoeffs = 0;
  for (int n = 0; n < a.length; n++)
          if (a[n] != 0) numCoeffs++; degree = n;
  if ((numCoeffs == 2 && a[0] != 0) || numCoeffs == 1 )
          return new SparsePoly (degree, a[degree], a[0]);
  return new DensePoly (degree, a);
}
```

Ottimizzazioni comuni

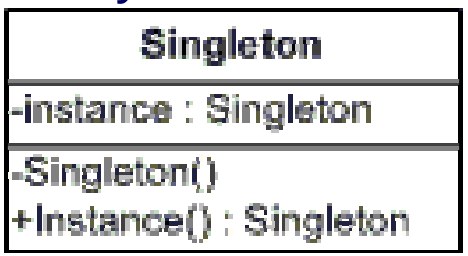
Introduzione
Definizioni
Categorie
➢ Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Alcuni pattern forniscono “trucchi” semplici e funzionali per velocizzare un programma o ridurne i requisiti di memoria
- A volte l’utilizzo di questi pattern non fa parte del progetto vero e proprio del sistema, ma un programmatore competente sa riconoscere le occasioni in cui usarli efficacemente

Singleton

Introduzione
Definizioni
Categorie
Factory Method
➢ Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- A volte una classe è istanziata per definizione da un solo oggetto
  - ad esempio, un archivio in cui ogni elemento è individuato univocamente dal suo identificatore (quindi se ci fossero più tabelle non si avrebbe questa garanzia di unicità)
- Una normale classe con soli metodi statici non assicura che esista un solo esemplare della classe, se viene reso visibile il costruttore
- In una classe Singleton il costruttore è protetto o privato
- Un metodo statico, o una factory, forniscono l'accesso alla sola copia dell'oggetto



Singleton pattern: il codice

Introduzione
Definizioni
Categorie
Factory Method
➢ Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

```
public class SingletonClass {
   private static SingletonClass s; // istanza singola

   private SingletonClass() { … } // costruttore
   public static SingletonClass getObject(){
        // crea l'oggetto solo se non esiste
        if (s == null) s = new SingletonClass();
        return s;
   }
   // altri metodi
}
```

Flyweight

Introduzione
Definizioni
Categorie
Factory Method
Singleton
➢ Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Quando molti oggetti identici (e **immutabili**) vengono utilizzati contemporaneamente, è utile costruire solo un oggetto per ogni “classe di equivalenza di oggetti identici”
  - gli oggetti condivisi vengono chiamati flyweight (pesi mosca) perchè spesso sono molto piccoli
- Questo pattern va ovviamente usato solo se il numero di oggetti condivisi è molto elevato
- Gli oggetti flyweight devono essere immutabili per evitare problemi di aliasing

Implementazione del pattern

Introduzione
Definizioni
Categorie
Factory Method
Singleton
➢ Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Occorre una tabella per memorizzare gli oggetti flyweight quando vengono creati
- Non si possono usare i costruttori
  - un costruttore costruisce sempre una nuova istanza
  - naturale usare una factory per creare gli oggetti
    - la factory deve controllare se l'oggetto richiesto esiste già nella tabella prima di crearlo
      - se non esiste, chiama un costruttore (privato!), altrimenti restituisce un riferimento all'oggetto esistente

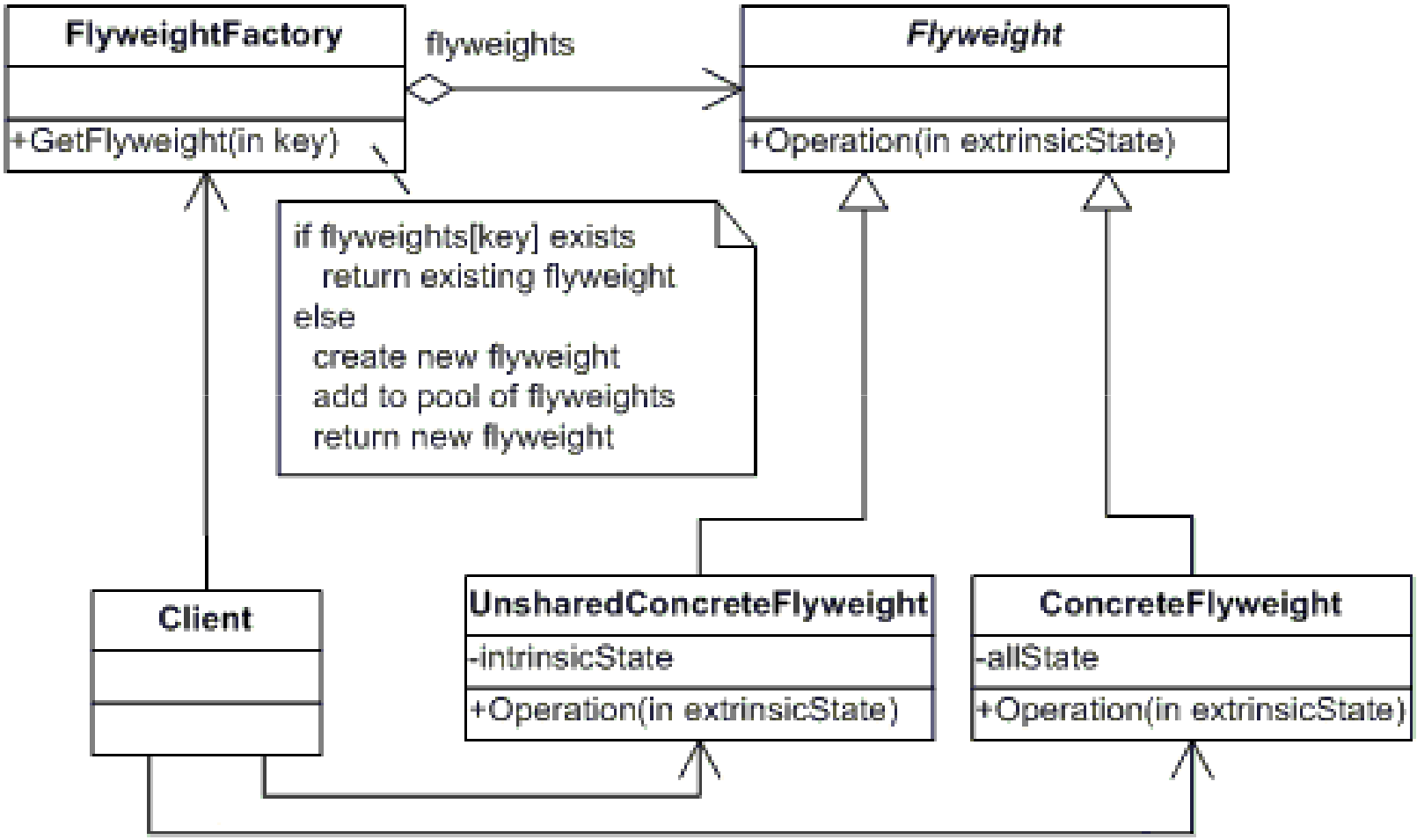
Implementazione del pattern

Introduzione
Definizioni
Categorie
Factory Method
Singleton
➢ Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Se necessario, occorre rimuovere gli oggetti dalla tabella quando non sono più utilizzati
- È consigliabile usare questo pattern se c'è un alto grado di condivisione degli oggetti
  - si risparmia memoria
  - non si perde tempo a inizializzare oggetti duplicati
  - si può usare == per il confronto al posto di equals

Implementazione del pattern

Introduzione
Definizioni
Categorie
Factory Method
Singleton
➢ Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio



State

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
➢ State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- A volte la variabilità degli oggetti mutabili può essere molto elevata
  - per esempio, una classe vettore può usare una rappresentazione diversa a seconda del numero degli elementi
  - anche il comportamento può cambiare grandemente: per esempio, in un sistema che gestisce l'accesso di utenti, molti metodi, prima che l'utente si sia autenticato, potrebbero semplicemente lanciare sempre un'eccezione
    - **il codice degli oggetti mutabili può diventare assai complicato e pieno di condizioni**

Soluzione

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
➢ State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

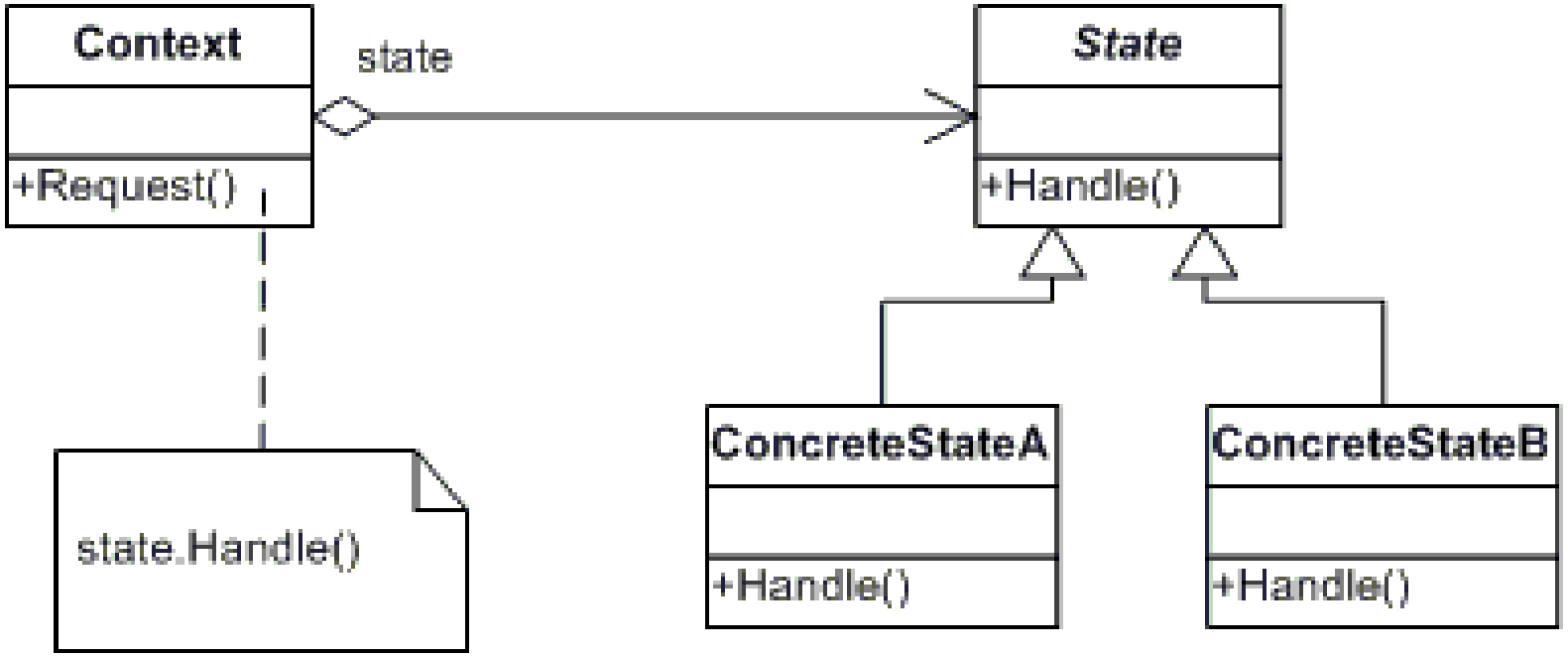
- Gli oggetti cambiano configurazione a seconda dello stato in cui si trovano
- Il pattern State introduce un ulteriore strato tra il tipo implementato e l'implementazione
  - a un unico tipo si fanno corrispondere più classi che lo implementano, e che corrispondono a diversi stati in cui possono trovarsi gli esemplari del tipo
  - nel corso della vita dell'oggetto, è possibile utilizzare diverse implementazioni senza che l'utente se ne accorga

Implementazione del pattern

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Si crea un'interfaccia o una classe astratta che rappresenta le parti dell'oggetto che possono essere sostituite nel corso della vita dell'oggetto
  - ciascuna delle possibili rappresentazioni (stati) diventa un'implementazione dell'interfaccia o un erede della classe astratta
  - la classe principale conterrà il codice per scegliere la rappresentazione più adatta e per delegare l'implementazione alla sottoclasse più appropriata per lo stato dell'oggetto

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
➢ State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



Procedure come oggetti

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

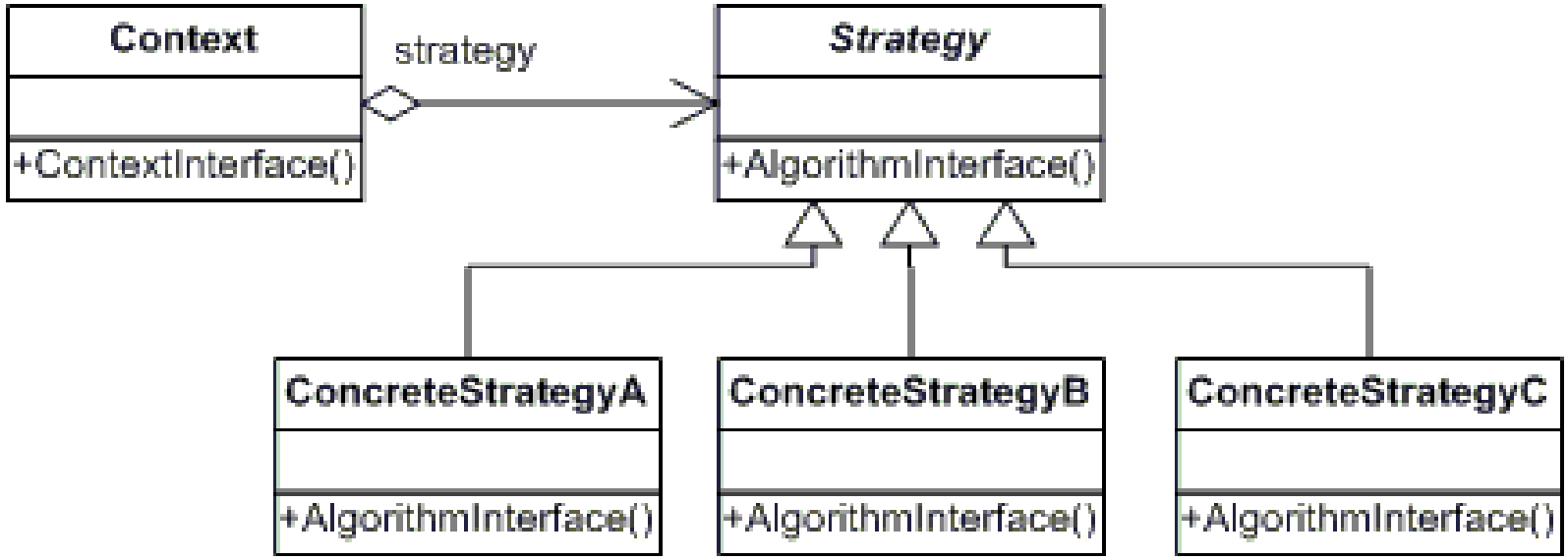
- Java non permette di utilizzare come oggetti le chiamate a un metodo
- Questo, tuttavia, può essere utile per definire astrazioni altamente generiche ed estendibili (pluggable)
- L'unico modo di ottenere questo risultato è definire classi o interfacce molto piccole
  - esempi nella libreria di classi di Java
    - Comparable, Runnable, ActionListener

Strategy

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Il pattern Strategy fornisce un oggetto che compie un'operazione precisa, richiesta dall'esterno
  - Per esempio, stabilire un ordinamento tra oggetti
- L'operazione è esprimibile con clausole **requires** e **ensures**
- Un esempio di questo pattern nell'interfaccia Comparator di JDK 1.4

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



Ordinamento di oggetti

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- Vogliamo ordinare un contenitore di oggetti (p.es. un array)
- La procedura di ordinamento è sempre la stessa per tutti i tipi di oggetti possibili…
  - vorremmo quindi fare un unico metodo per tutti i tipi
  - qualcosa come

```
//@ensures (* s è ordinato *)
public static void sort(Object [] s){…}
```

- Serve un modo per confrontare gli elementi in s.
  - Object non ha un metodo per il confronto e quindi occorre definirlo da qualche altra parte

Ordinamento di oggetti

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- **Idea**: aggiungo come argomento al metodo un “oggettino” incaricato del confronto
- Per potere rendere il metodo sort applicabile a ogni tipo, l’oggetto sarà di tipo interfaccia. Quindi:
  - definisco l'interfaccia Comparator (esiste peraltro in java.util), che definisce sintatticamente il confronto di due oggetti
  - fornisco una implementazione di Comparator per il tipo che voglio ordinare (es. IntegerComparator)
  - passo anche un Comparator quando chiamo la procedura per confrontare gli elementi

Interface Comparator

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

```
interface Comparator {
    //@ se o1 e o2 non sono di tipi confrontabili
    //@ lancia ClassCastException
    //@ altrimenti: o1<o2 → ret –1
    //@             o1==o2 → ret 0
    //@             o1>o2 → ret 1

    public int compare (Object o1, Object o2)
        throws ClassCastException, NullPointerException;
    }
```

- Questa interfaccia **non** è supertipo dei tipi i cui elementi vanno comparati!

Metodo sort

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
➢ Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Un oggetto di tipo Comparator (uno solo per tutti gli elementi!)
- Esempio da java.util.Arrays:

```
public static void sort (Object[] a, Comparator c) {
   …
     if ( c.compare(a.[i], a.[j]) )
   …
}
```

- Esempio d'uso

```
public class AlphabeticComparator implements Comparator{
   public int compare(Object o1, Object o2) {
     String s1 = (String)o1; String s2 = (String)o2;
     return s1.toLowerCase().compareTo( s2.toLowerCase());
   }
}
String[] s = new String[30];
Java.util.Arrays.sort(s, new AlphabeticComparator());
```

Proxy

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
➢ Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

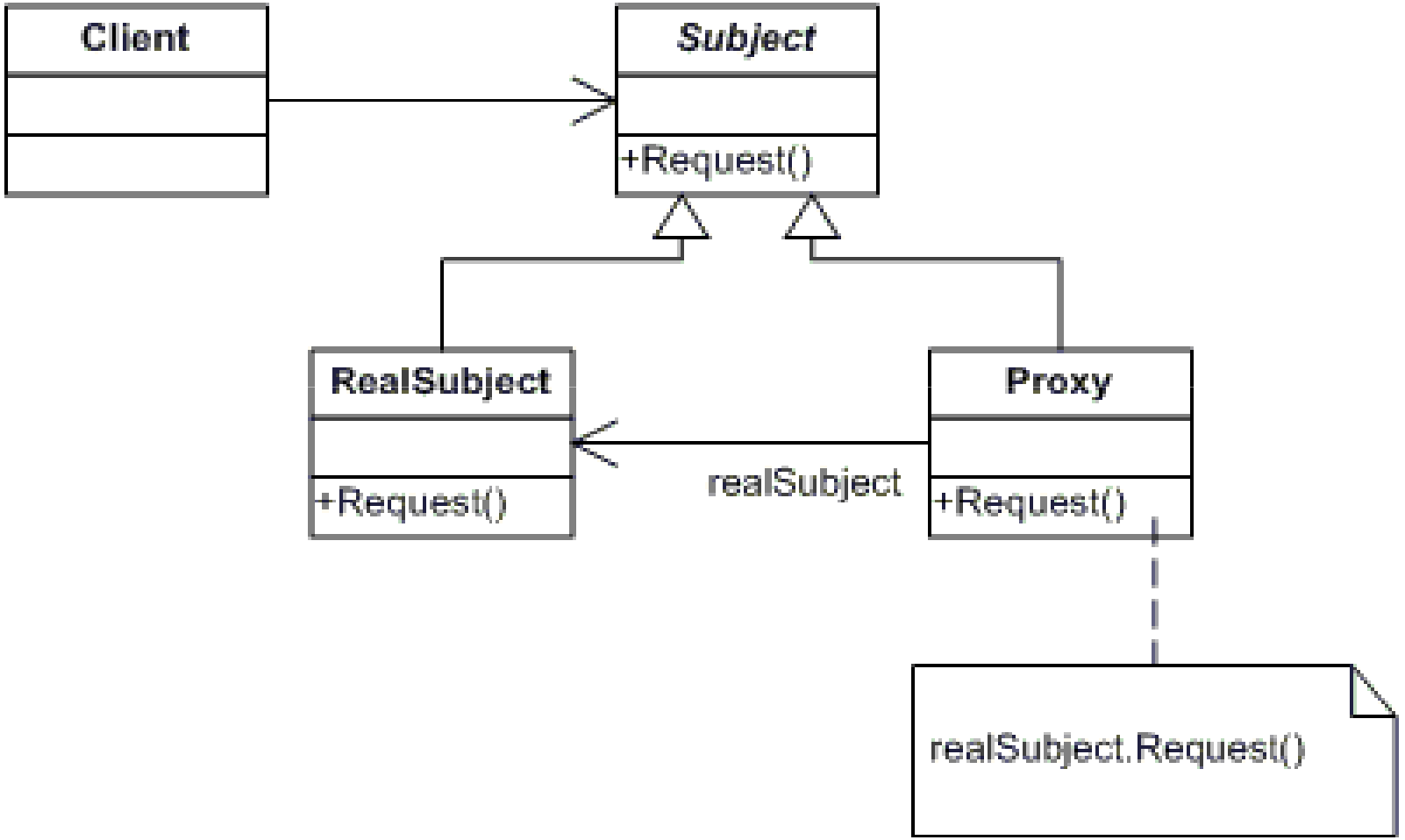
- Quando l'oggetto interposto espone esattamente la stessa interfaccia dell'oggetto separato, di cui fa le veci, esso prende il nome di Proxy
- Contesto: fornire un riferimento a un oggetto surrogato che sostituisce un altro oggetto, allo scopo di controllare meglio l'accesso a quell'oggetto
  - per esempio quando molteplici client devono accedere ai servizi offerti da un server
  - controllo dei permessi dei client sui servizi offerti dal server

Pattern Strutturali: Proxy

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
➢ Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

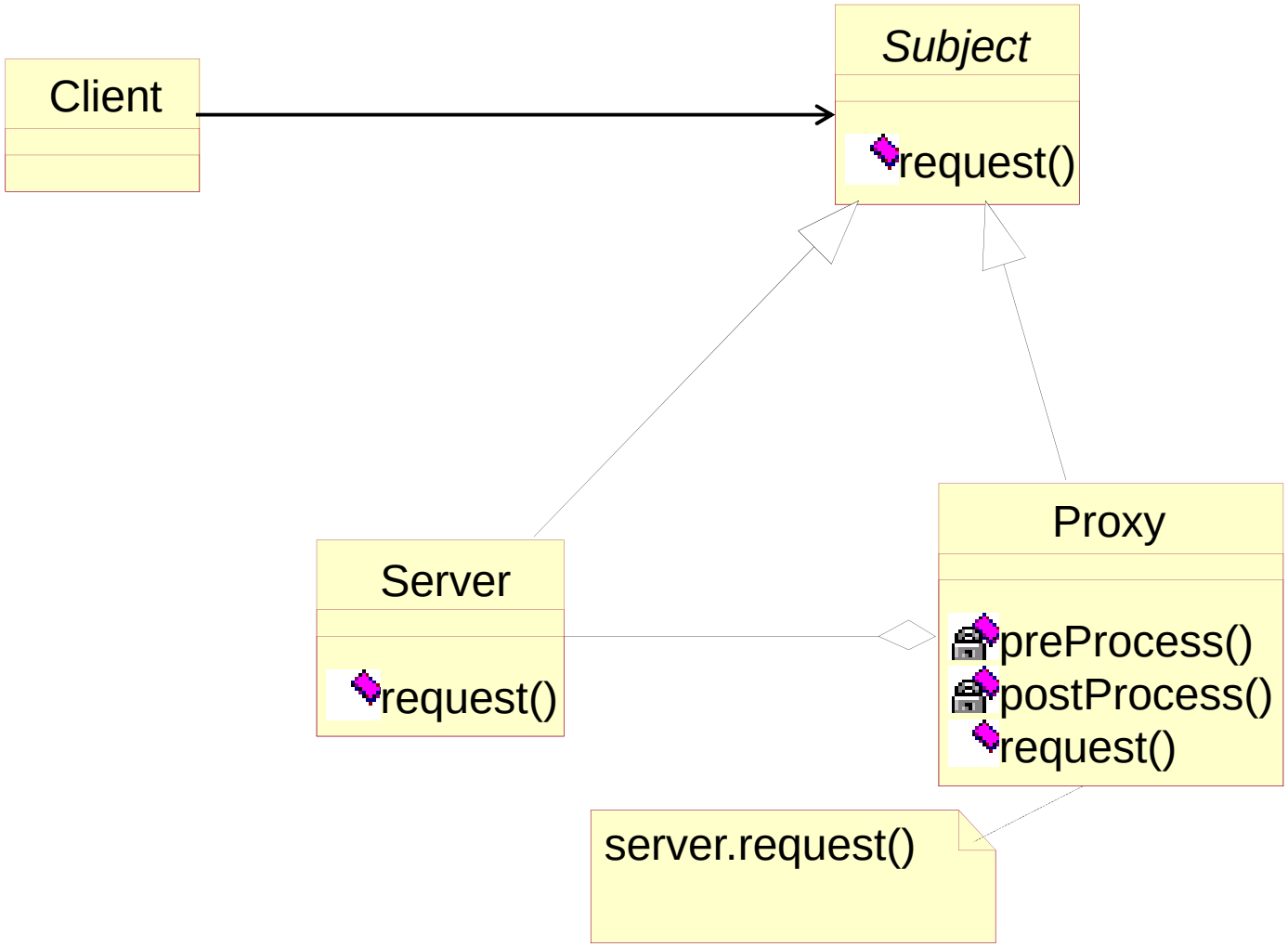
- Problema: evitare di accedere direttamente (location transparency), garantire sicurezza ed efficienza
  - esempi
    - i DBMS
    - il Proxy WWW
- Uno scopo del Proxy è di posporre o addirittura evitare l'istanziazione di oggetti "pesanti", se non necessaria
  - es. gli stub di RMI "sembrano" oggetti locali, ma si occupano di serializzare i parametri, inviarli in rete, attendere il risultato, ecc., senza però essere i "veri" oggetti
- Soluzione: istituire un intermediario, che effettua pre- e post-processing (controllo di accessi, caching, …)

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
➢ Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



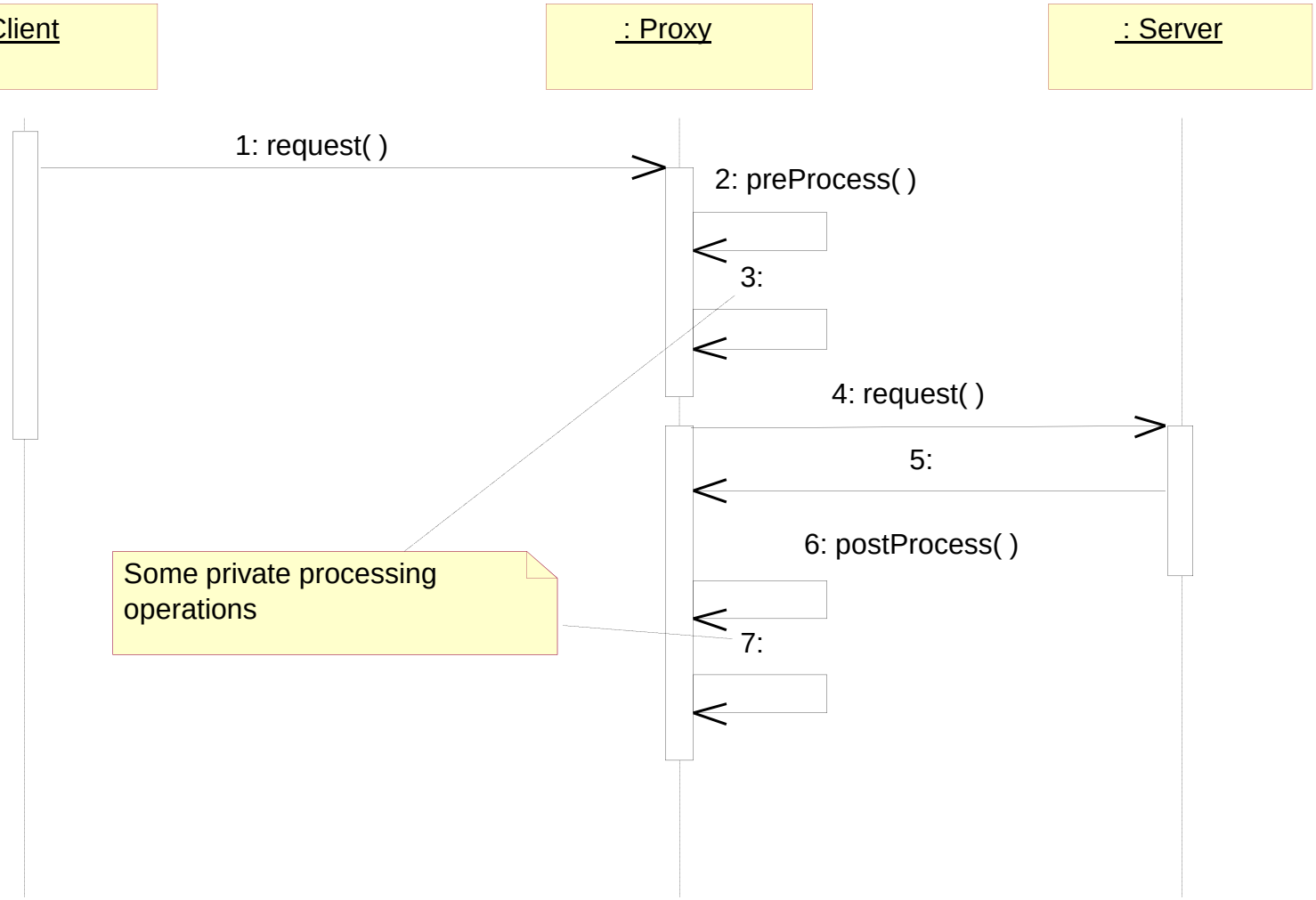
Client-Server

- Introduzione
- Definizioni
- Categorie
- Factory
- Method
- Singleton
- Flyweight
- State
- Strategy
- ➢ Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Observer
- Pattern architetturali
- MVC
- Esempio



Proxy: comportamento dinamico

- Introduzione
- Definizioni
- Categorie
- Factory
- Method
- Singleton
- Flyweight
- State
- Strategy
- ➢ Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Observer
- Pattern architetturali
- MVC
- Esempio



Applicazioni

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
➢ Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Remote Proxy
  - fornisce un'interfaccia locale che maschera la comunicazione con un oggetto localizzato in un altro spazio di indirizzamento
- Virtual Proxy
  - crea oggetti "costosi" (che occupano per esempio molta memoria) solo quando è necessario
- Protection Proxy
  - gestisce livelli di accesso diversi ad un oggetto da parte di client diversi
- Smart Reference
  - viene usato in sostituzione ad un semplice riferimento. Effettua azioni aggiuntive quali, per esempio, il lock/unlock dei dati che vengono acceduti

Proxy: esempi d'uso

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
➢ Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- OMG-CORBA, Java RMI
  - meccanismo di comunicazione remota fra oggetti basata su "proxy"
  - per ogni oggetto remoto esiste un proxy locale
- HTTP Proxy
  - intermediario fra web browser e http server
  - caching di documenti a cui accedono frequentemente diversi browser
- Firewall di rete
  - controllo degli accessi

“Adattare” interfacce diverse

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
➢ Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

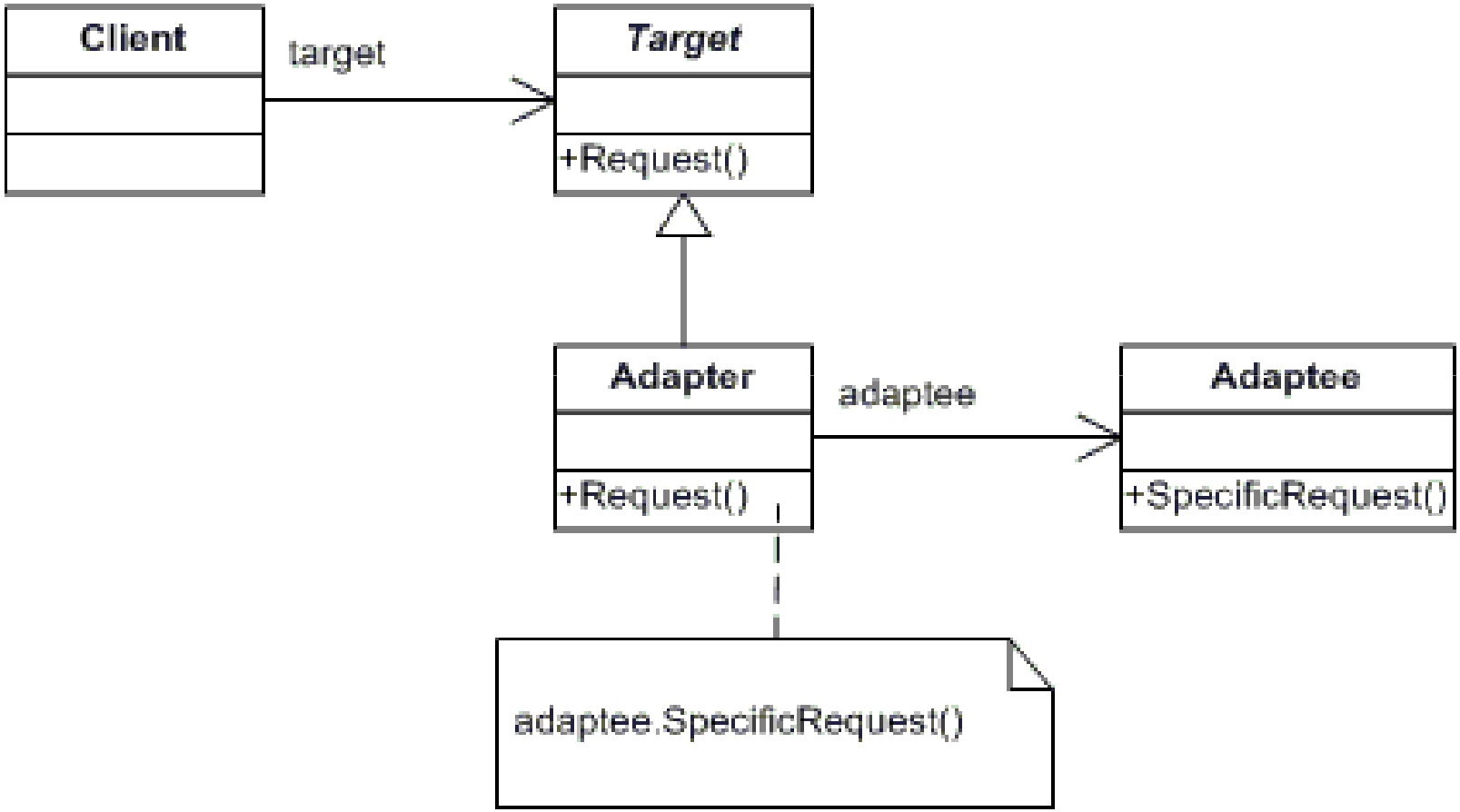
- Molto spesso librerie diverse espongono interfacce diverse per fare la stessa cosa
  - Windows e MacOS sono ambienti grafici incompatibili tra loro
- Una stessa soluzione si adatta a svariati problemi
  - si scrivono nuove classi che impongano una stessa interfaccia e uno stesso insieme di precondizioni e postcondizioni
- Gli esemplari delle nuove classi usano un oggetto interno che contiene la vera implementazione
  - esempio del motto “Every problem in computer science can be solved by adding another level of indirection”
  - l’oggetto visibile all’esterno chiama oggetto esterno

Adapter

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
➢ Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Si applica quando l'interfaccia dell'oggetto interno è diversa da quella dell'oggetto esterno
- L'oggetto esterno è l'Adapter, quello interno l'Adaptee
  - le librerie di classi per l'interfaccia grafica, come AWT o Swing, non sono altro che enormi raccolte di oggetti Adapter
  - in Java, java.io.OutputStreamWriter permette di scrivere caratteri a 16-bit (Unicode) su di un OutputStream che lavora per byte
  - gli skeleton di RMI mappano su di un protocollo binario i metodi di un'interfaccia Java

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
➢ Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



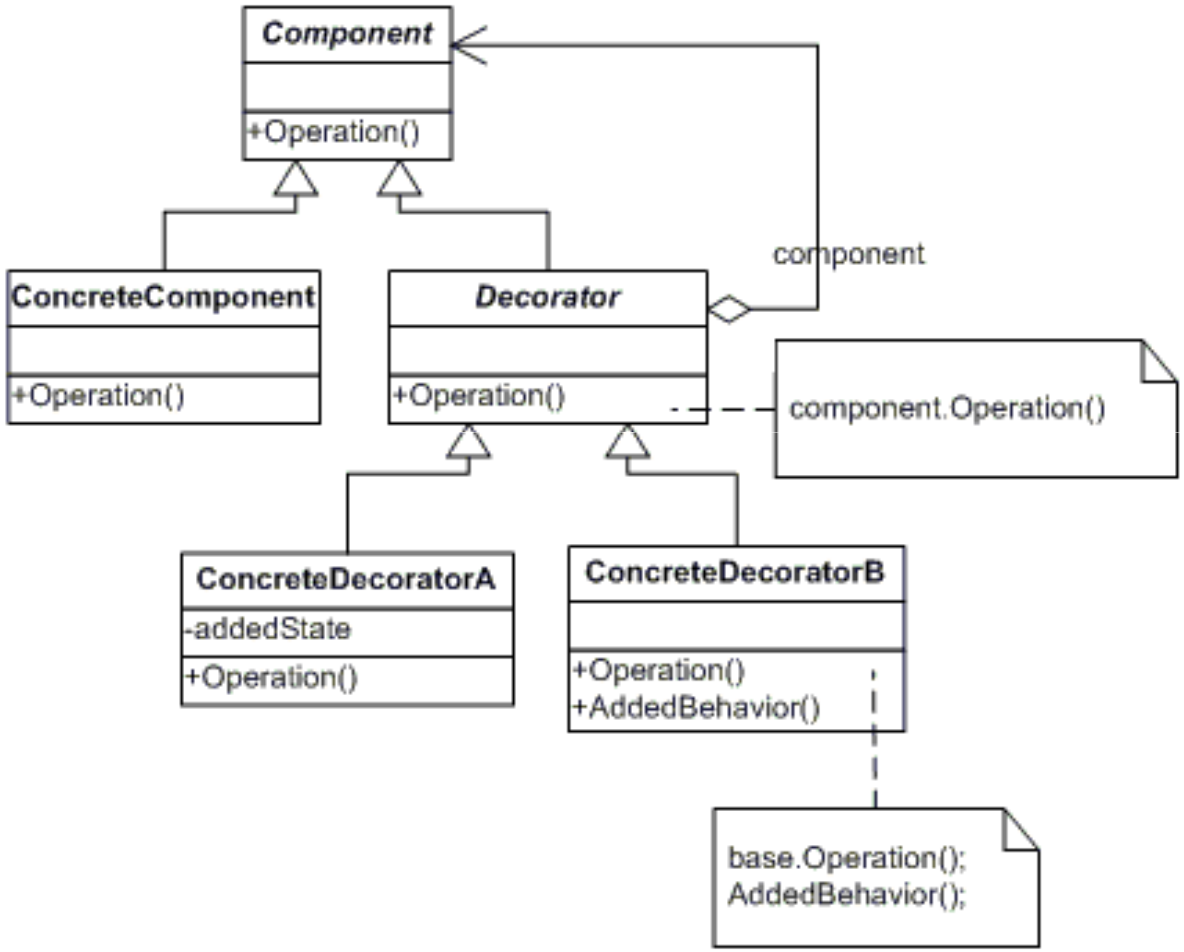
Decorator

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
➢ Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- Altre volte, invece, l'oggetto fornisce funzionalità aggiuntive:prende allora il nome di Decorator
  - java.util.zip.CheckedOutputStream calcola un checksum al volo e possiede un metodo aggiuntivo per restituirlo
- La libreria di classi di Java (Stream, RMI, interfaccia grafica) utilizza pesantemente Adaptor, Proxy e Decorator

Decorator

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
➢ Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



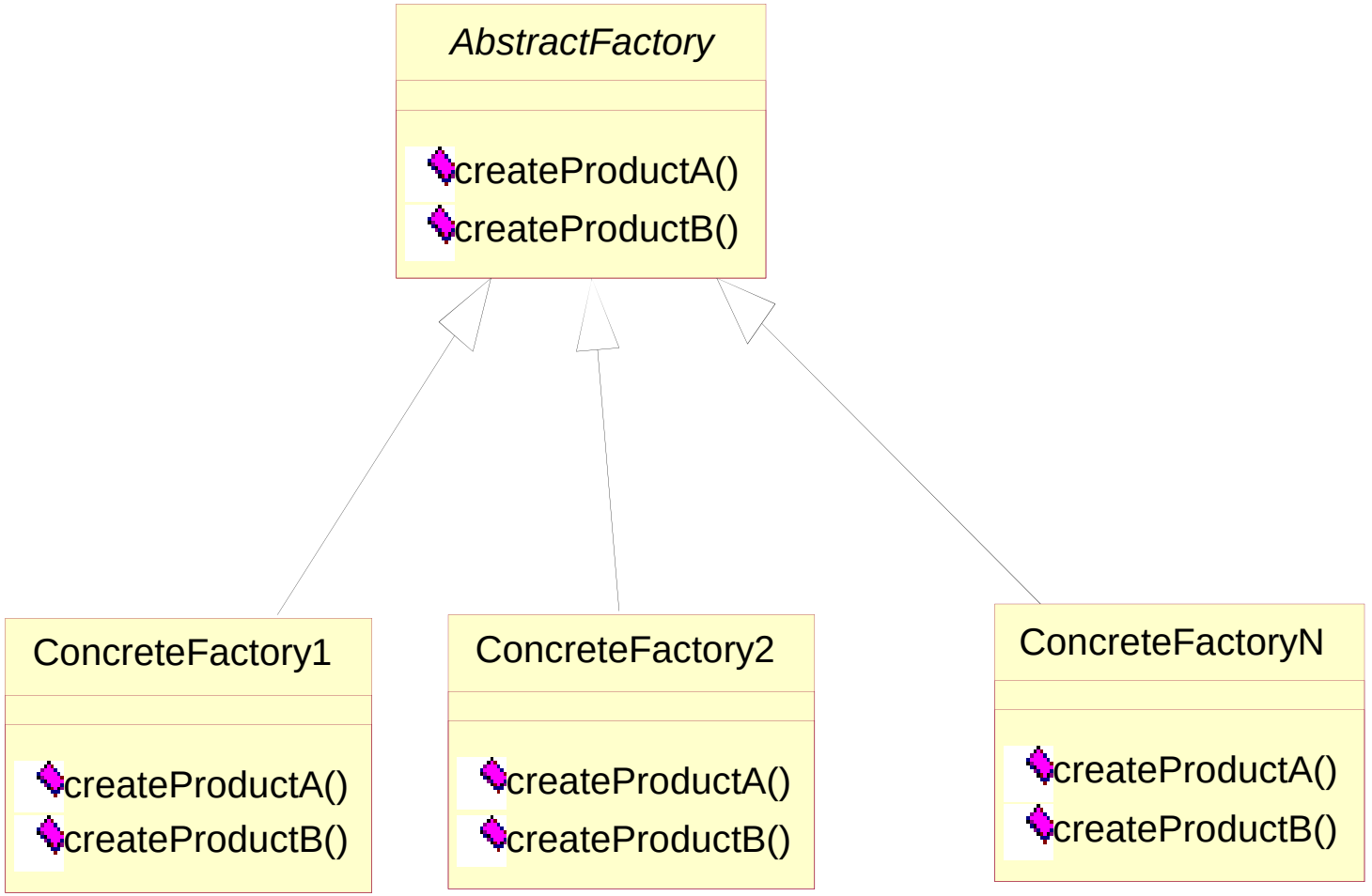
Il pattern Abstract Factory

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- È un pattern **creazionale**
- **Contesto**: creazione di famiglie di oggetti correlati senza specificare la loro classe concreta
- **Problema**: Dare un'interfaccia programmatica unica per la creazione delle varie famiglie e minimizzare le modifiche del codice della logica dell'applicazione
- **Soluzione**: fornire l'interfaccia e i meccanismi di creazione degli oggetti attraverso super-classi astratte (una per famiglia di prodotti) e demandare l'istanziazione degli oggetti-prodotto a loro sotto-classi
- Il caso della GUI è solo uno degli scenari esemplificativi di motivazione

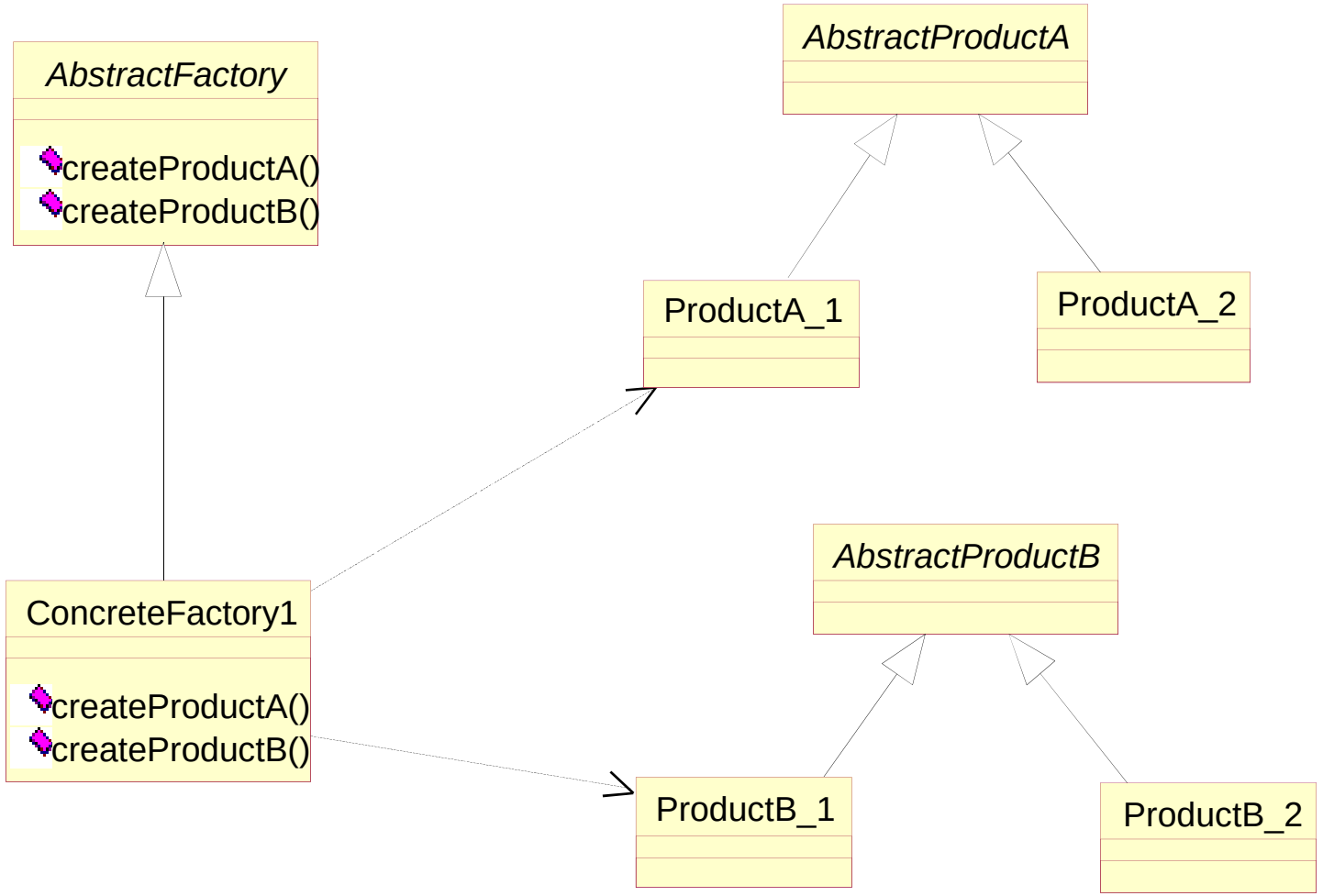
Schema

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio



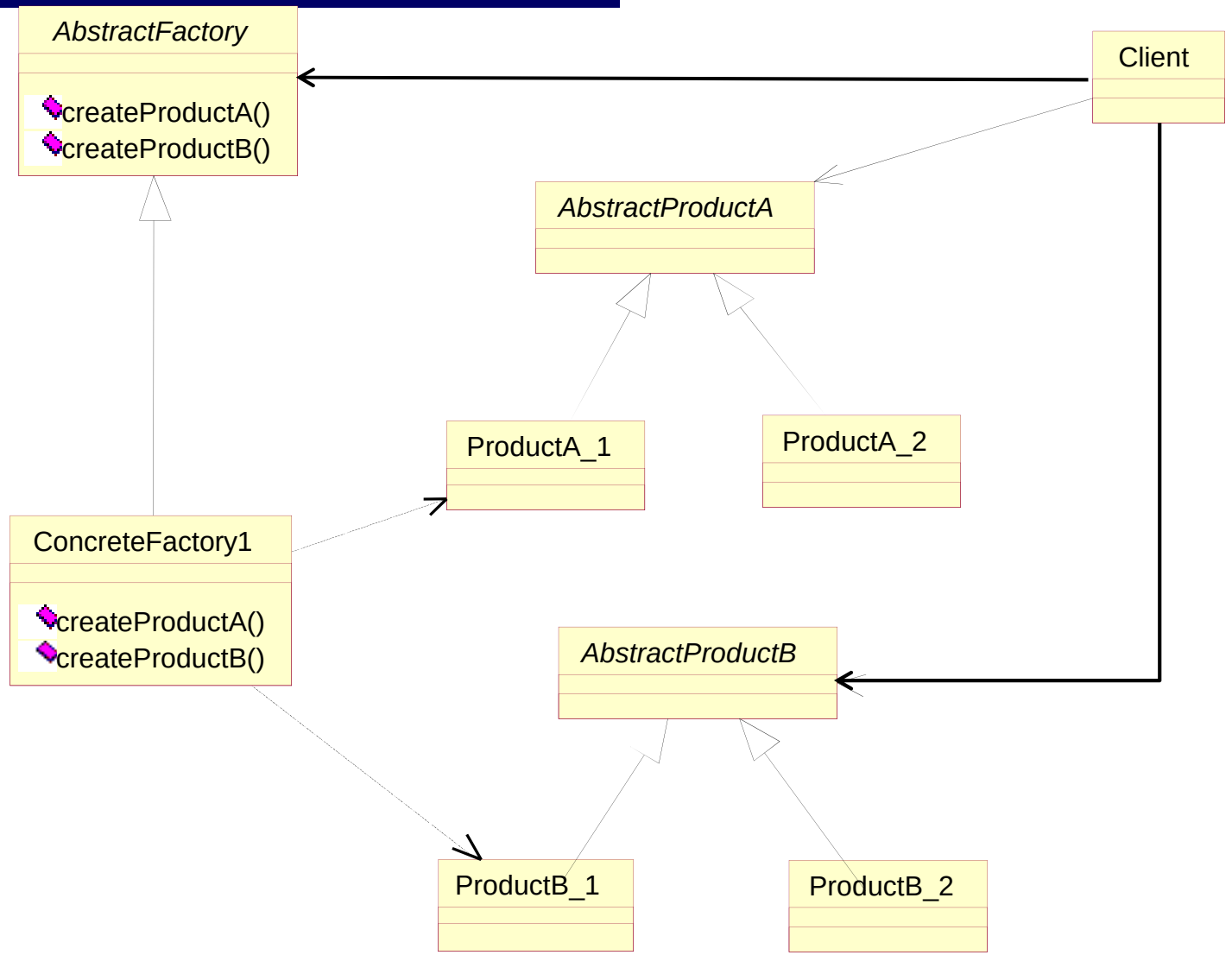
Esempio

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



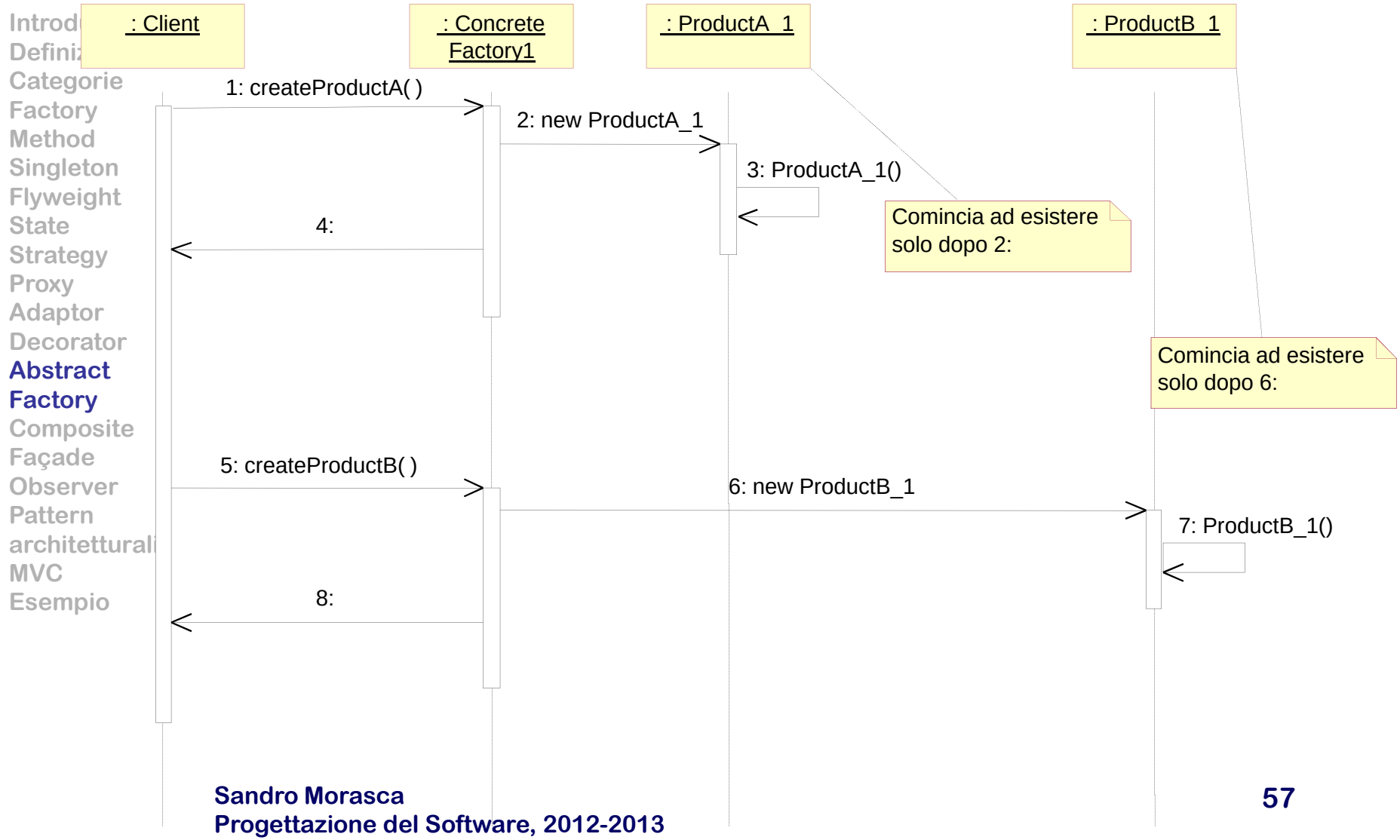
Esempio

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio



Sequence diagram

Introd
Defini
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



Partecipanti al pattern Abstract Factory

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- AbstractFactory
  - dichiara un'interfaccia per creare i prodotti di una famiglia
- ConcreteFactory
  - implementa le operazioni di creazione dei prodotti concreti
- AbstractProduct
  - dichiara l'interfaccia per un tipo di prodotto
- ConcreteProduct
  - fonisce un'implementazione per il prodotto
  - è creato dalla concrete factory corrispondente
- Client
  - usa l'interfaccia AbstractProduct
  - crea prodotti chiamando la ConcreteFactory attraverso l'interfaccia AbstractFactory

Abstract Factory: sintesi (1)

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Isola le classi concrete
  - i clienti non devono sapere niente delle classi concrete che useranno, neanche al momento dell'instanziazione degli oggetti
- È facile cambiare famiglia di prodotto
  - basta cambiare 1 linea di codice che riguarda la creazione della factory
- Promuove la consistenza tra i prodotti
  - i prodotti sono organizzati in famiglie
  - i prodotti di una famiglia sono coordinati per lavorare insieme

Abstract Factory: sintesi (2)

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
➢ Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Supportare l'inserimento di un nuovo prodotto in una famiglia è difficile
  - puo' richiedere cambiamenti all'interfaccia dell'Abstract Factory e alle sue sottoclassi
  - richiede l'aggiunta di una nuova classe ConcreteFactory e di nuovi AbstractProducts e Products
- La creazione di oggetti non avviene nel modo standard
  - i clienti devono sapere che devono usare la factory invece del costruttore per istanziare nuovi oggetti
  - altrimenti la correttezza dell'implementazione non è garantita

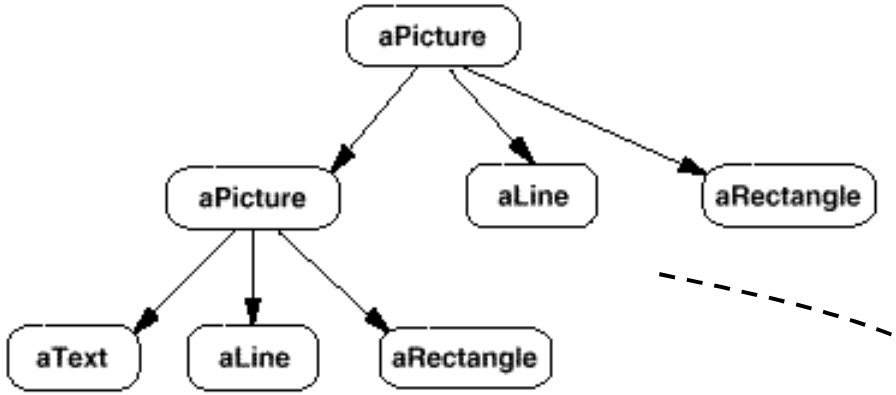
Pattern Strutturali: Composite

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
➢ Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio

- Obiettivi
  - fornire la possibilità di comporre oggetti in strutture ad albero che rappresentano gerarchie intero-parte
  - consentire ai client di trattare oggetti singoli e compositi in modo uniforme
  - minimizzare il più possibile la complessità di una gerarchia intero-parte
    - riducendo il numero di tipologie di oggetti che possono trovarsi nei diversi nodi dell'albero

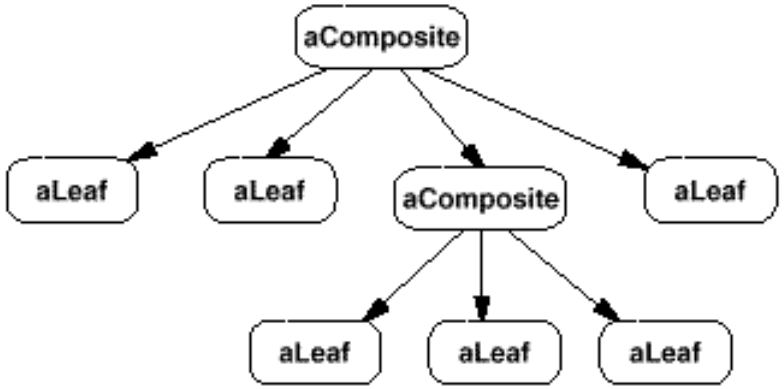
Composite: Esempio

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
➢ Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



Un esempio di struttura con Oggetti compositi

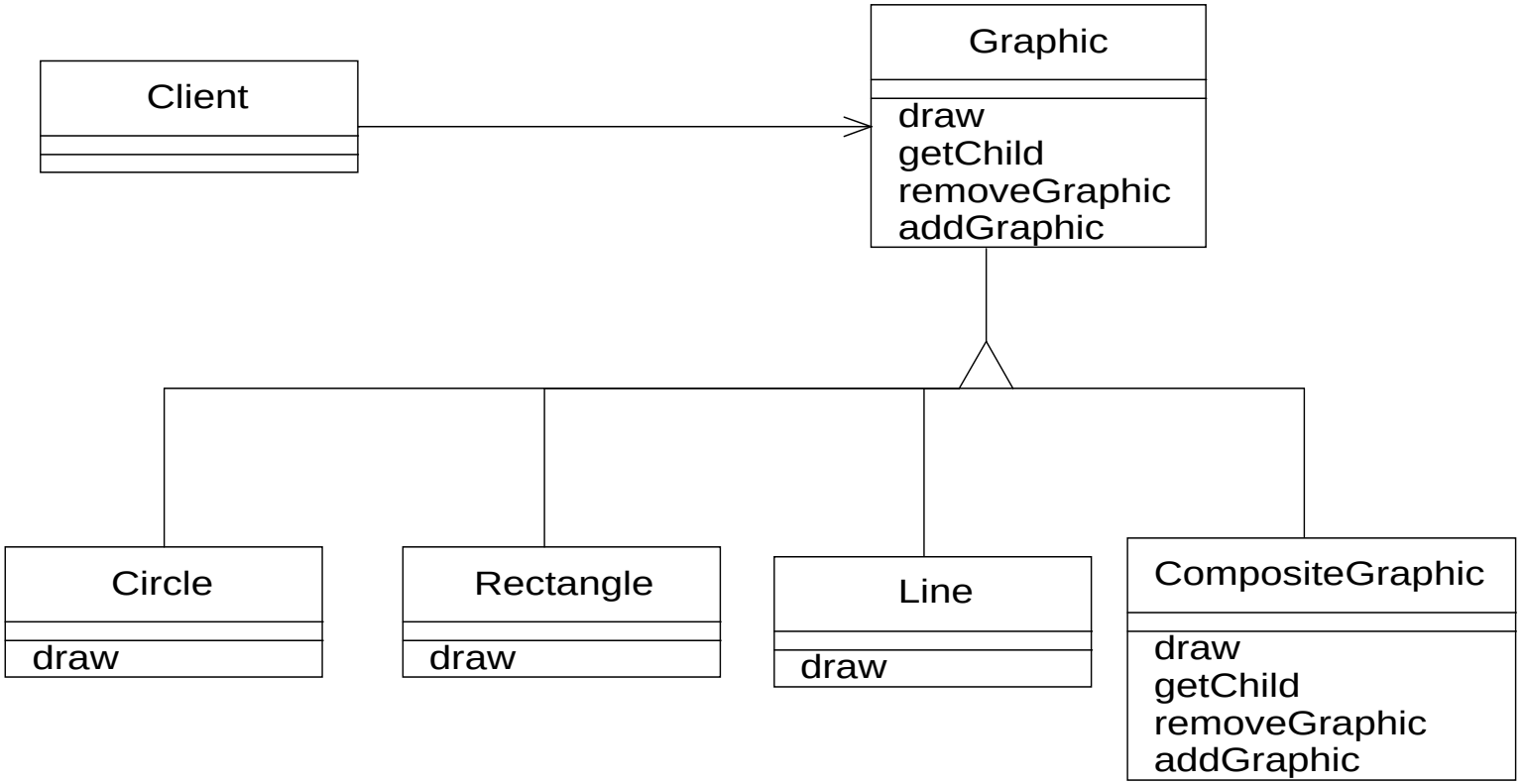
Generalizzazione della struttura ad albero



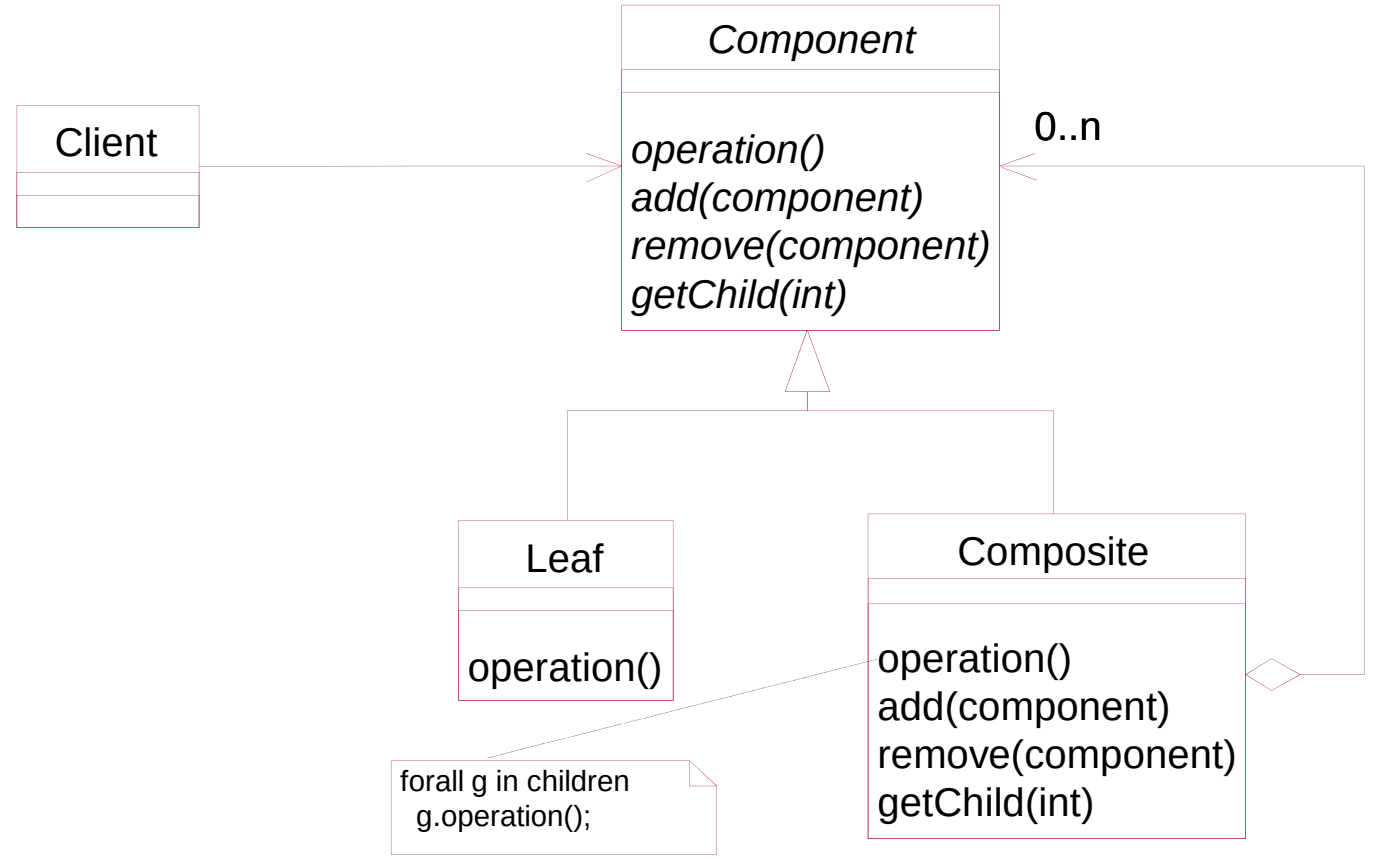
Esempio tratto da: E. Gamma, R. Helm, R. Johnson, J. Vlissides.
Design patterns - Elements of reusable object oriented software.

Composite: Esempio

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
➢ Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
➢ Composite
Façade
Observer
Pattern
architetturali
MVC
Esempio



Composite: Considerazioni

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
➢ Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Il pattern Composite
  - Consente di definire gerarchie di classi costituite da oggetti primitivi e compositi.
    - Gli oggetti primitivi possono essere composti per formare oggetti più complessi, che a loro volta potranno essere composti ricorsivamente.
    - In tutti i punti in cui il client si aspetta di utilizzare un oggetto primitivo, potrà essere indifferentemente utilizzato un oggetto composito.
  - Semplifica il client.
    - I client possono trattare strutture composite e singoli oggetti in modo uniforme.
    - I client solitamente non sanno (e non dovrebbero neanche preoccuparsene) se stanno operando con una foglia o con un componente composito.

Composite: Considerazioni (2)

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
➢ Composite
Façade
Observer
Pattern architetturali
MVC
Esempio

- Il pattern Composite
  - Rende più semplice l'aggiunta di nuove tipologie di componenti
    - nuove sottoclassi Leaf o Composite potranno automaticamente essere utilizzate nelle strutture esistenti
    - il codice dei client, non deve essere modificati quando viene aggiunto una nuova sottoclasse di Component
  - Può rendere il progetto troppo generico.
    - La flessibilità nell'aggiunta di nuove tipologie di componenti rende difficile porre delle restrizioni ai componenti che fanno parte di una struttura composita.
    - Se una struttura composita deve poter contenere soltanto determinate tipologie di componenti occorre inserire nel codice dei controlli specifici che saranno effettuati a run-time.

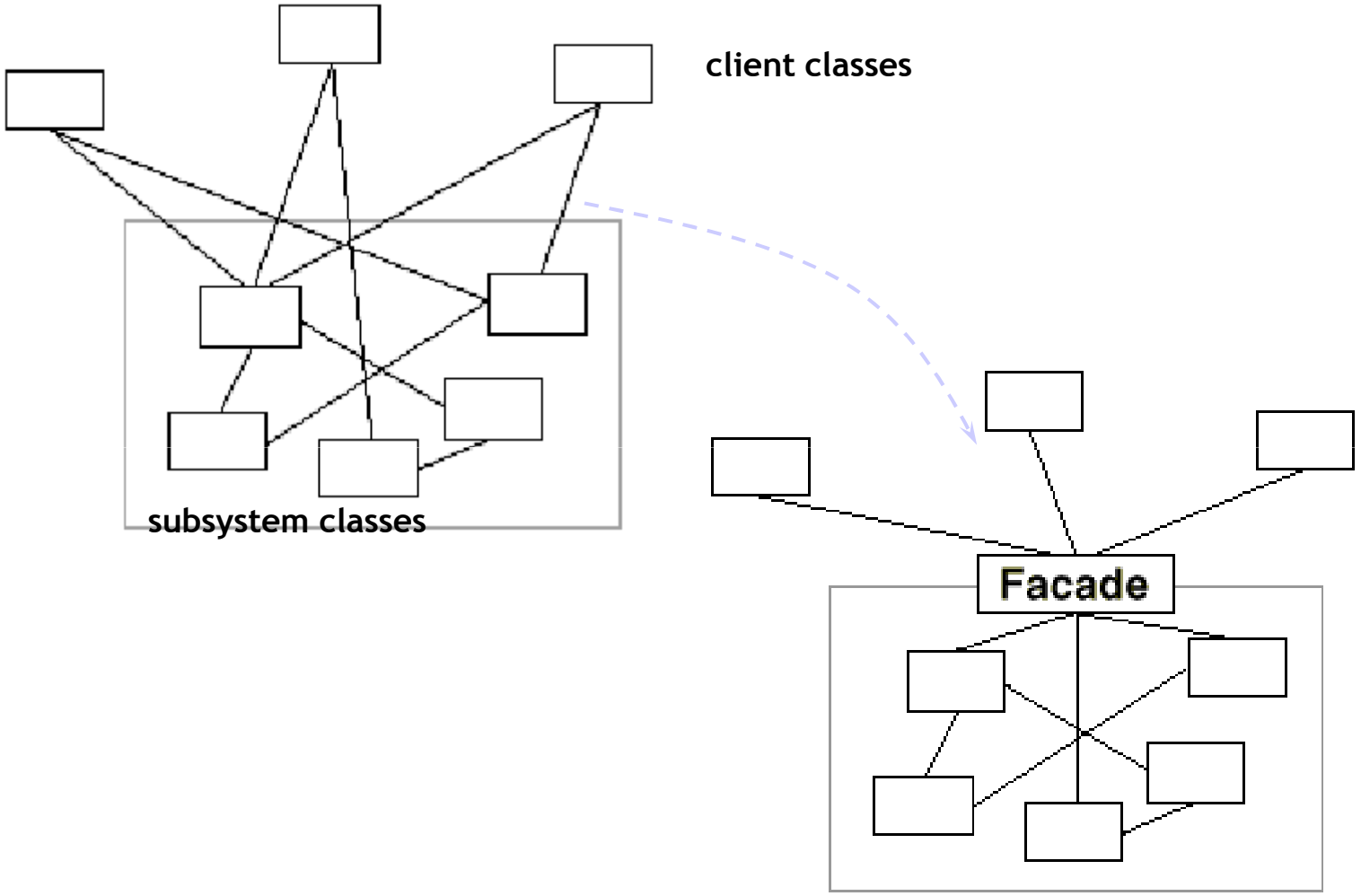
Pattern Strutturali: Façade

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
➢ Façade
Observer Pattern
architetturali
MVC
Esempio

- Obiettivi
  - fornire un'interfaccia unificata e di alto livello verso un sottosistema complesso, rendendolo più semplice da utilizzare
  - nascondere ai client i dettagli sulla struttura interna del sottosistema
  - ottenere un accoppiamento lasco tra client e sottosistema

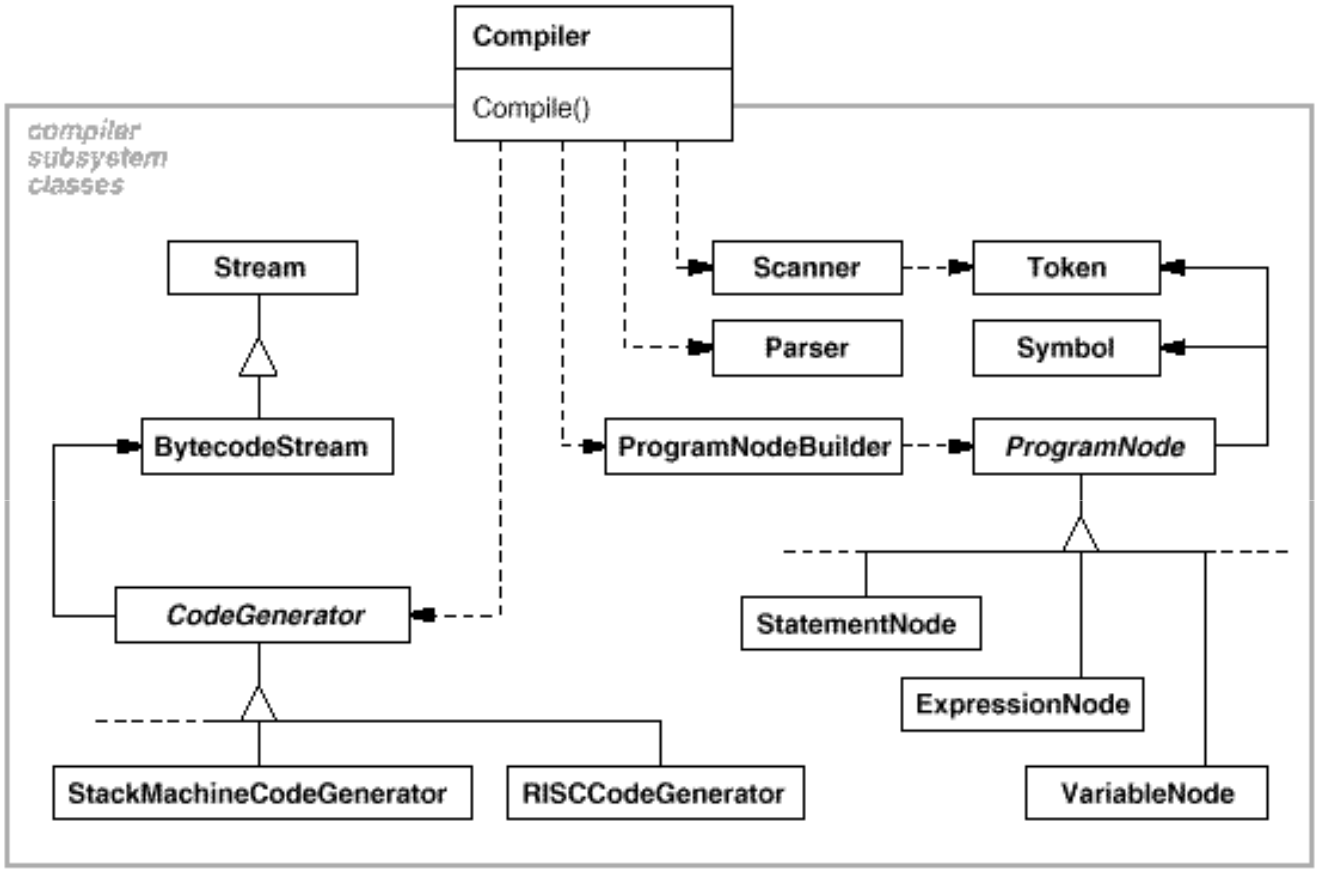
Façade: Esempio

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- **Façade**
- Observer Pattern
- architetturali
- MVC
- Esempio



Façade : Esempio di applicazione

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
➢ Façade
Observer
Pattern
architetturali
MVC
Esempio



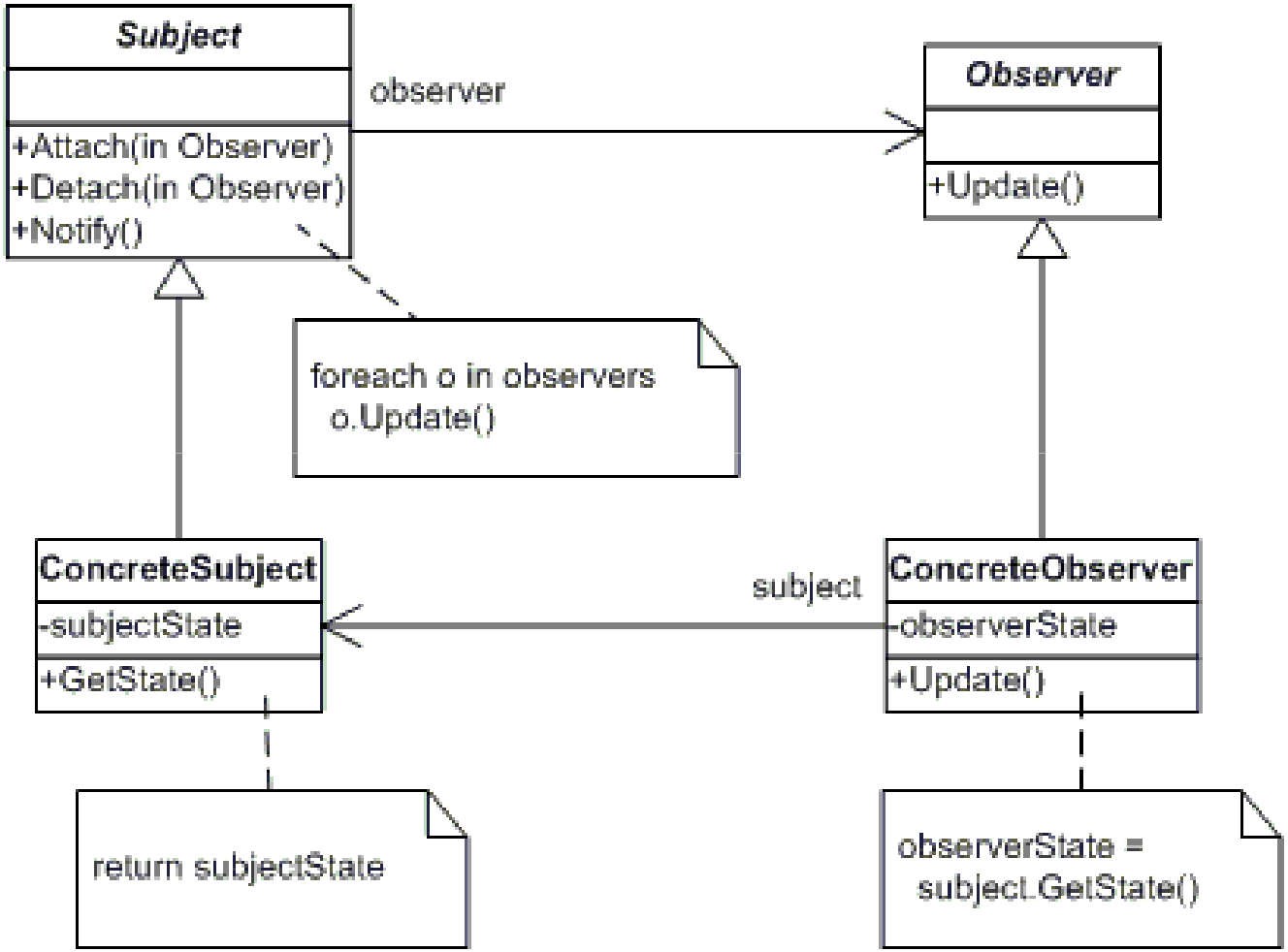
Esempio tratto da: E. Gamma, R. Helm, R. Johnson, J. Vlissides.
Design patterns - Elements of reusable object oriented software.

Un pattern comportamentale: l'Observer

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
➢ Observer
Pattern architetturali
MVC
Esempio

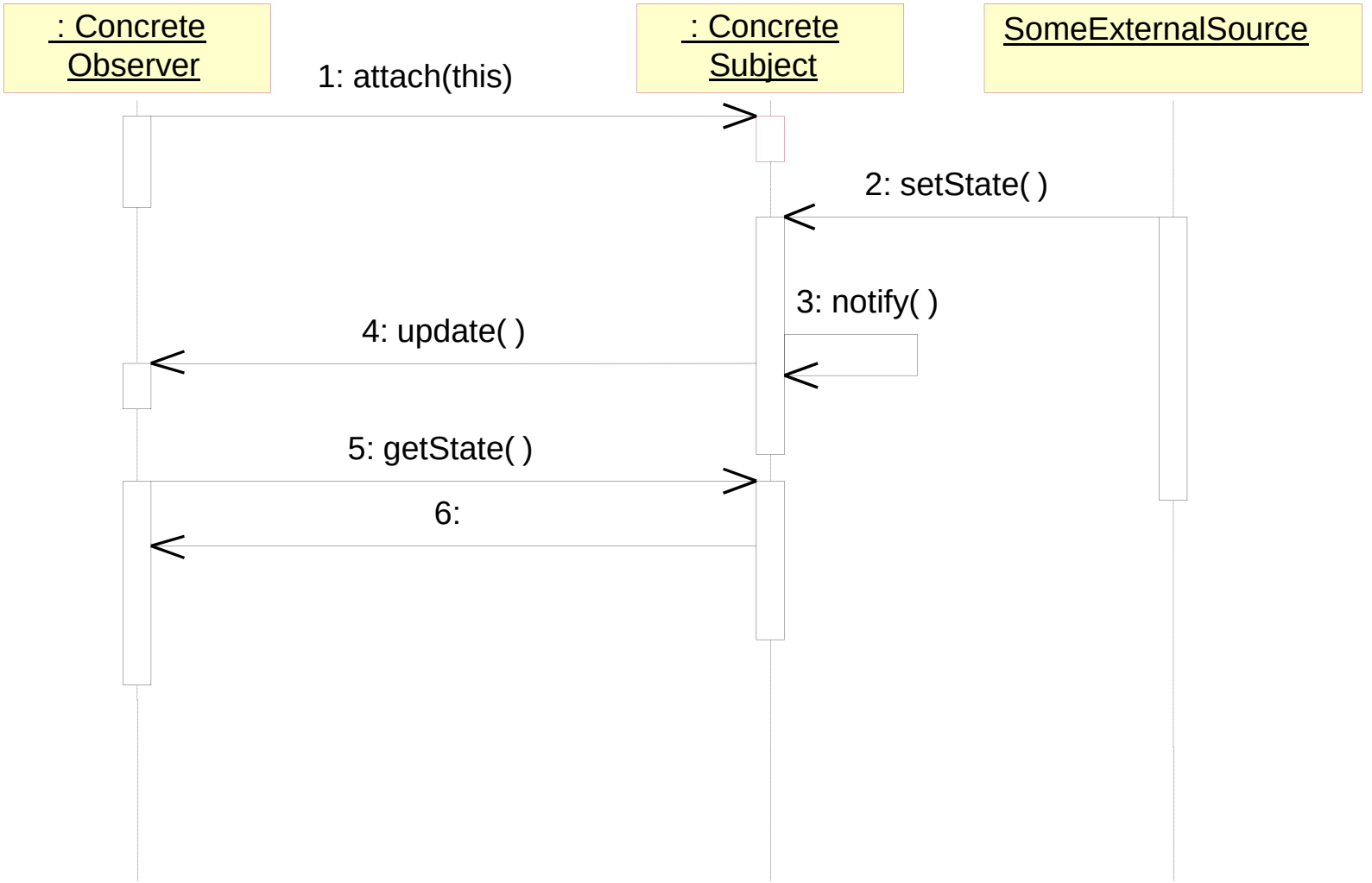
- **Contesto**
  - uno o più oggetti sono interessati ad osservare i cambiamenti di stato di un soggetto
- **Problema**
  - il soggetto deve essere indipendente dal numero e dal tipo degli osservatori
  - deve essere possibile aggiungere nuovi osservatori durante l'esecuzione dell'applicazione
- **Soluzione**
  - inserire nel soggetto delle operazioni che consentono all'osservatore di dichiarare il proprio interesse per un cambiamento di stato

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
➢ Observer
Pattern
architetturali
MVC
Esempio



L'Observer: comportamento dinamico

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
➢ Observer
Pattern architetturali
MVC
Esempio



Quando usare il pattern Observer

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
➢ Observer
Pattern architetturali
MVC
Esempio

- Quando i cambiamenti su un soggetto devono essere inviati in "broadcast" a tanti oggetti
- Quando oggetti indipendenti dall'applicazione (i soggetti) devono richiamare oggetti implementati specificamente per l'applicazione per notificare un cambiamento nel loro stato interno (i soggetti non devono conoscere niente riguardo ai loro osservatori ne' come funzionano)
- I JavaBeans sono componenti riusabili implementati sulla base di questo pattern

Non esistono solo i Design Pattern

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
➢ Pattern architetturali
MVC
Esempio

- Esistono diverse tipologie di pattern, che si differenziano principalmente per la scala ed il livello di astrazione
- **Architectural Pattern**
  - utili per strutturare un sistema in sottosistemi
- **Design Pattern**
  - operano a livello di un sottosistema evidenziando le caratteristiche delle classi coinvolte e delle associazioni tra le classi
- **Idiom**
  - utili per l'implementazione di specifici aspetti di design in un particolare linguaggio di programmazione

Architectural Pattern

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
➢ Pattern architetturali
MVC
Esempio

- Architectural Pattern
  - definiscono uno schema di organizzazione della struttura di un sistema software
  - individuano un insieme di sottosistemi, specificano le diverse responsabilità ed includono regole e linee guida per organizzare le relazioni tra i diversi sottosistemi
  - Esempi: Layers, Pipes&Filters, Broker, Model-View-Controller

Idiom

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
➢ Pattern architetturali
MVC
Esempio

- Idioms
  - sono pattern che operano a basso livello
  - sono specifici per un determinato linguaggio di implementazione
  - descrivono come implementare particolari aspetti di singoli componenti utilizzando le features di un particolare linguaggio
  - esempi: Implementazione del pattern Singleton in Java o C++

Pattern mining

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
➢ Pattern architetturali
MVC
Esempio

- I design pattern non si inventano, ma si scoprono
- Si estraggono dal codice di programmi particolarmente “buoni”
- *Buoni*, ma in che senso?
  - “qualità senza nome” (Alexander): si sente ma non si può definire
  - eleganti
  - programmi che sono belli da manutenere
- Un pattern
  - *estrae e astrae* l’esperienza del bravo progettista software dall’applicazione in questione ...
  - … e la mette a disposizione di tutti i progettisti software per applicazioni future

Un esempio di pattern mining: MVC

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
➢ MVC
Esempio

- Per capire in pratica di che si tratta, consideriamo le classi Model, View e Controller (MVC) introdotte nel 1988 per costruire interfacce in Smalltalk-80
- Model
  - rappresenta l'oggetto (il dato) vero e proprio, funzionale all'applicazione
  - è indipendente dalle specifiche rappresentazioni su schermo
  - è indipendente dalle modalità di input dei dati da parte dell'utente
- View
  - la rappresentazione dell'oggetto sullo schermo dell'utente
  - una view ottiene i dati per la presentazione dal Model. Possono esserci (e tipicamente esistono) viste multiple dello stesso modello

MVC

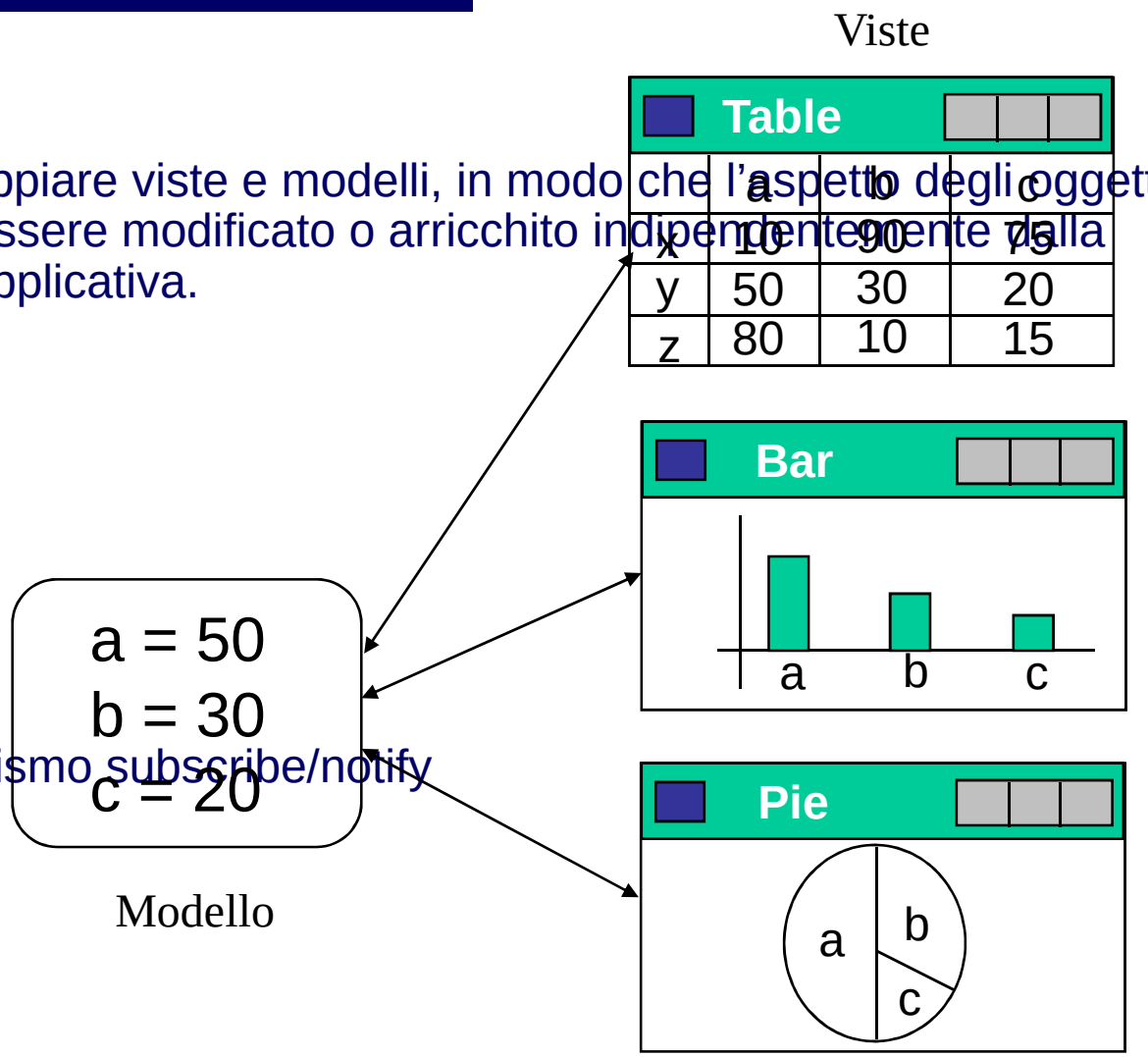
Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
➢ MVC
Esempio

- Controller
  - insieme di regole che stabiliscono le reazioni della presentazione sullo schermo in relazione all'input dei dati da parte dell'utente
  - ciascuna view ha associato un componente controller, il cui compito è
    - ricevere gli input dell'utente (tipicamente come eventi a seguito di operazioni con mouse o tastiera - Es. pressione di un pulsante)
    - traduce gli eventi in richieste di servizio per il modello o la vista cui è associato
  - l'utente interagisce con il sistema solo ed esclusivamente attraverso il controller

MVC - esempio

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
➢ MVC
Esempio

- Strategia:
  - disaccoppiare viste e modelli, in modo che l'aspetto degli oggetti possa essere modificato o arricchito indipendentemente dalla logica applicativa.
- Come:
  - meccanismo subscribe/notify



Dalla soluzione al pattern (1)

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
➢ MVC
Esempio

- MVC costituisce un esempio di progetto che prevede di disaccoppiare viste e modelli per una particolare applicazione
- In realtà la strategia adottata da MVC può essere impiegata in una classe di problemi più ampia
  - quando è necessario disaccoppiare oggetti tali che cambiamenti in uno influenzino gli altri, senza che l'oggetto modificato debba conoscere dettagli sulla natura degli altri
- Questo principio di progettazione estratto da MVC è descritto dal design pattern Observer

Dalla soluzione al pattern (2)

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
➢ MVC
Esempio

- MVC fornisce anche viste annidate, realizzate mediante la classe CompositeView, derivata da View
  - le componenti di CompositeView sono a loro volta delle View
- Si potrebbe ritenere che questa sia una soluzione specifica
- Di fatto è ri-applicabile tutte le volte che vogliamo trattare un gruppo di oggetti come un oggetto unico, avente la stessa natura dei suoi componenti
- Il pattern **Composite** permette di creare una gerarchia di classi in cui alcune classi sono primitive (ad es. il bottone in una vista) mentre altre (come CompositeView) definiscono oggetti composti che raggruppano oggetti primitivi

Dalla soluzione al pattern (3)

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
➢ MVC
Esempio

- MVC incapsula i meccanismi di controllo degli eventi generati dall'interfaccia utente in un oggetto Controller
  - c'è una gerarchia di classi di controllo, che permette di aggiungere e modificare controllori
- Ciascuna vista usa un'istanza di controllore per implementare una strategia di risposta all'utente. L'associazione vista-controllore è modificabile a run-time
- Questo è un esempio di pattern **Strategy**
  - Strategy è un'oggetto che incapsula un algoritmo, che può essere rimpiazzato sia staticamente sia dinamicamente all'interno di un altro oggetto

Problema: “Look-and-Feel” multipli

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
➢ Esempio

- Vogliamo essere in grado di configurare un word processor per supportare diversi stili di presentazione (look-and-feel dell’interfaccia utente)
- La compresenza di diversi stili di presentazione nell’implementazione è dovuta a differenze nelle piattaforme su cui viene eseguita l’applicazione (per esempio, Mac, Windows, o UNIX)
- In alcuni casi la presentazione può variare anche sulla stessa piattaforma (per esempio, Athena, Motif, Open Look su UNIX)

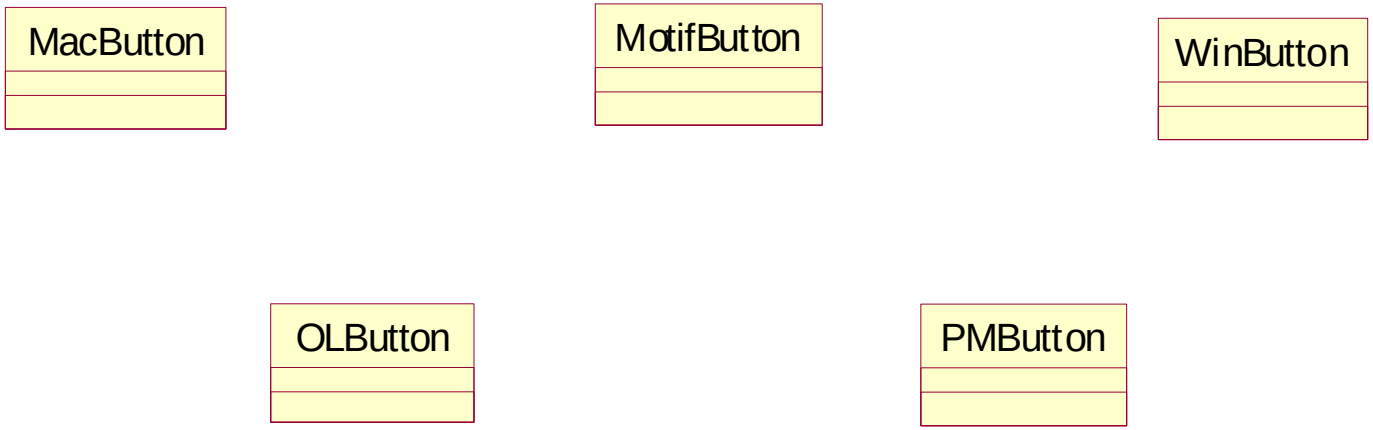
Requisiti

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
➢ Esempio

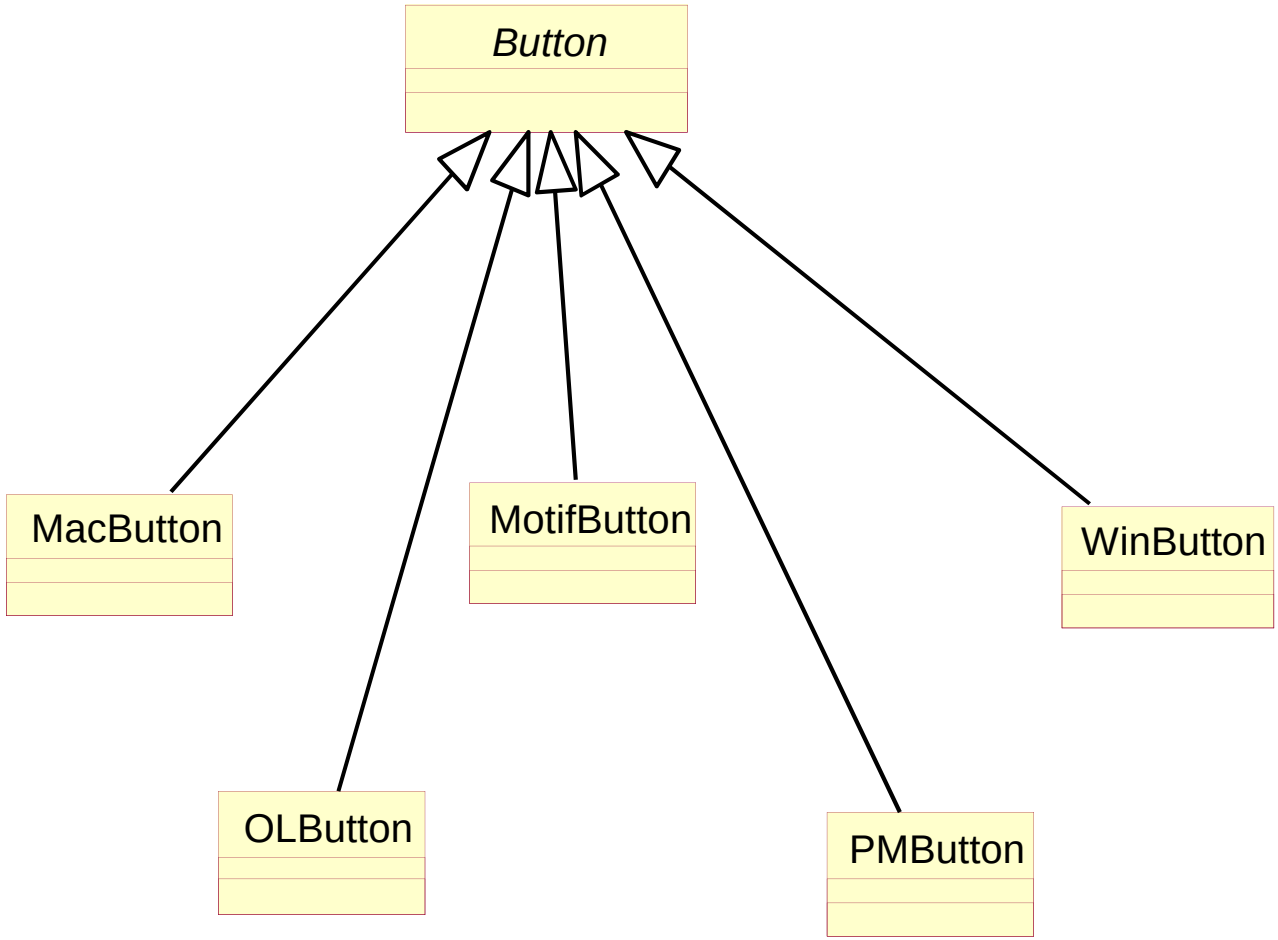
- Dobbiamo essere in grado di cambiare standard di presentazione quando l'applicazione viene mandata in esecuzione su una diversa piattaforma
- Dobbiamo essere in grado di aggiungere nuovi standard facilmente
- Tutti gli elementi dell'interfaccia utente visualizzati in un certo istante dell'esecuzione devono appartenere allo stesso standard (non si può avere un bottone Motif con una barra di scorrimento Windows)
- Vogliamo rendere la logica dell'applicazione indipendente dagli standard supportati
- Il codice che incapsula la logica dell'applicazione dovrebbe usare la stessa interfaccia astratta per interagire con gli elementi dei diversi standard di presentazione

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio

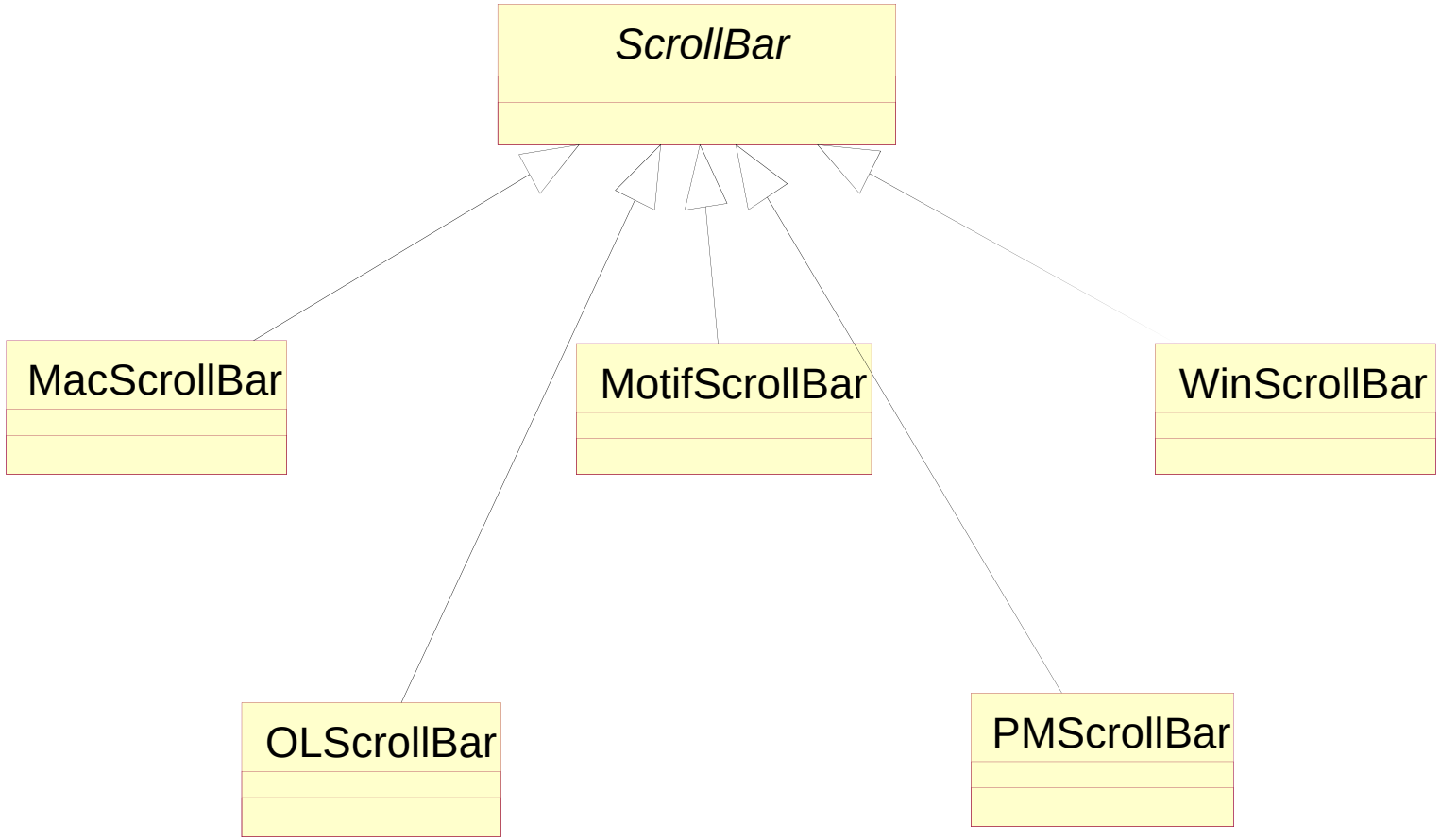
- Come possiamo fornire la stessa interfaccia astratta a questi oggetti di classi diverse?



Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio



Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio



E poi...

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
➢ Esempio

- Per creare un'astrazione completa dell'interfaccia utente dobbiamo creare le gerarchie per ogni elemento componente
  - Menus
  - Check Boxes
  - Radio Buttons
  - ecc...
- Si ha una proliferazione di classi ma si riesce a rimuovere la dipendenza della logica dell'applicazione dalla presentazione

Uso degli elementi dell'interfaccia utente astratta

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
➢ Esempio

- Il codice che utilizza gli elementi dell'interfaccia utente può ora interagire con gli elementi di alto livello dell'interfaccia utente
  - ad esempio, la logica dell'applicazione userà le operazioni offerte dalla classe bottone senza far riferimento direttamente alla classe MotifButton
- Abbiamo ancora un problema
  - dobbiamo creare gli elementi che costituiscono in pratica l'interfaccia utente
  - al momento della creazione dobbiamo conoscere l'implementazione delle sottoclassi per ogni piattaforma

Il problema della creazione (1)

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio

- Abbiamo bisogno di un modo per proteggere la logica dell'applicazione dal dover conoscere le sottoclassi dell'interfaccia utente anche al momento della creazione
  - ScrollBar sb = new MotifScrollBar;
- Non vogliamo cambiare il codice in tutti i punti in cui una barra di scorrimento è creata quando passiamo da uno standard di presentazione ad un altro

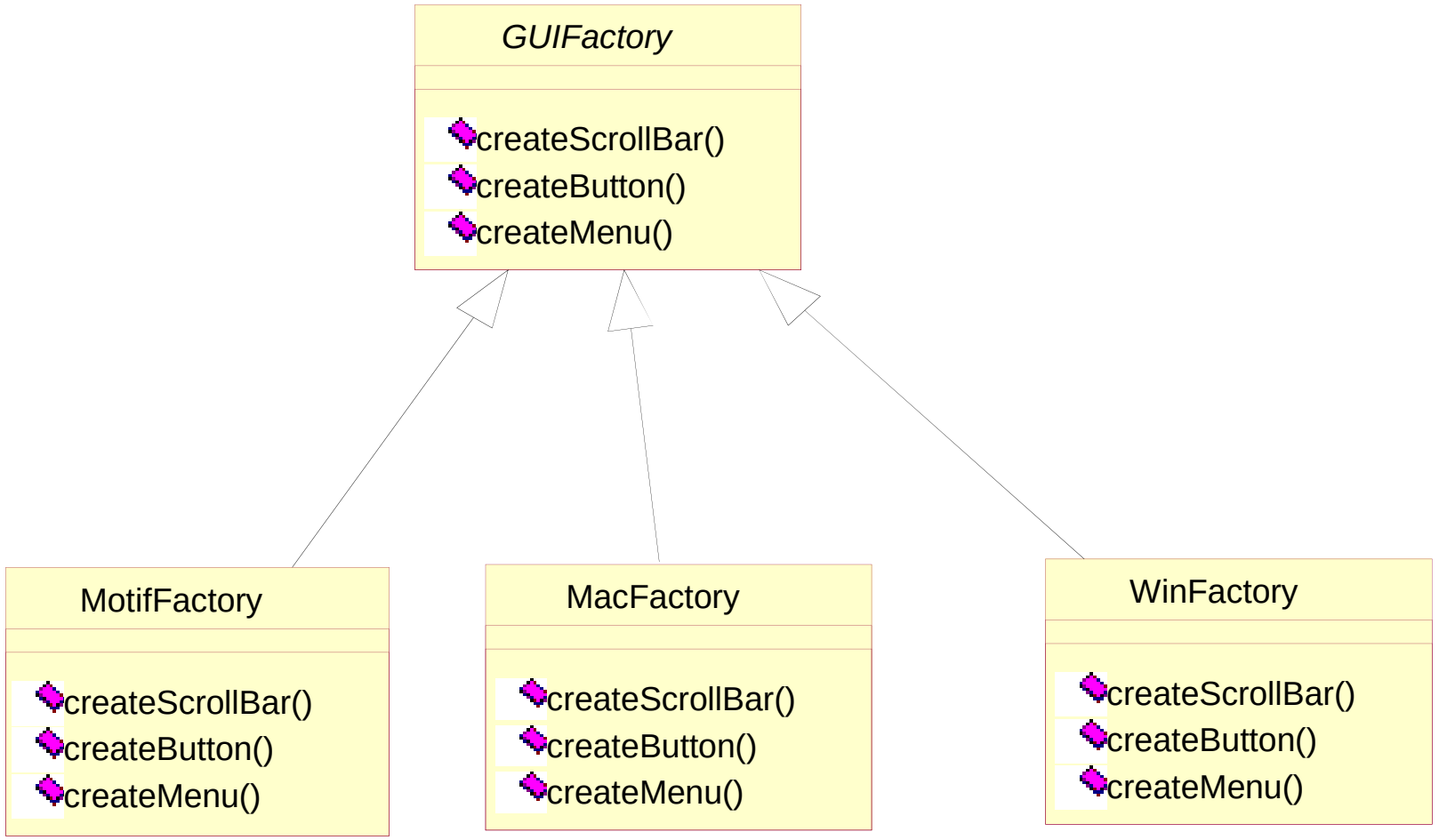
Il problema della creazione (2)

Introduzione
Definizioni
Categorie
Factory Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
➢ Esempio

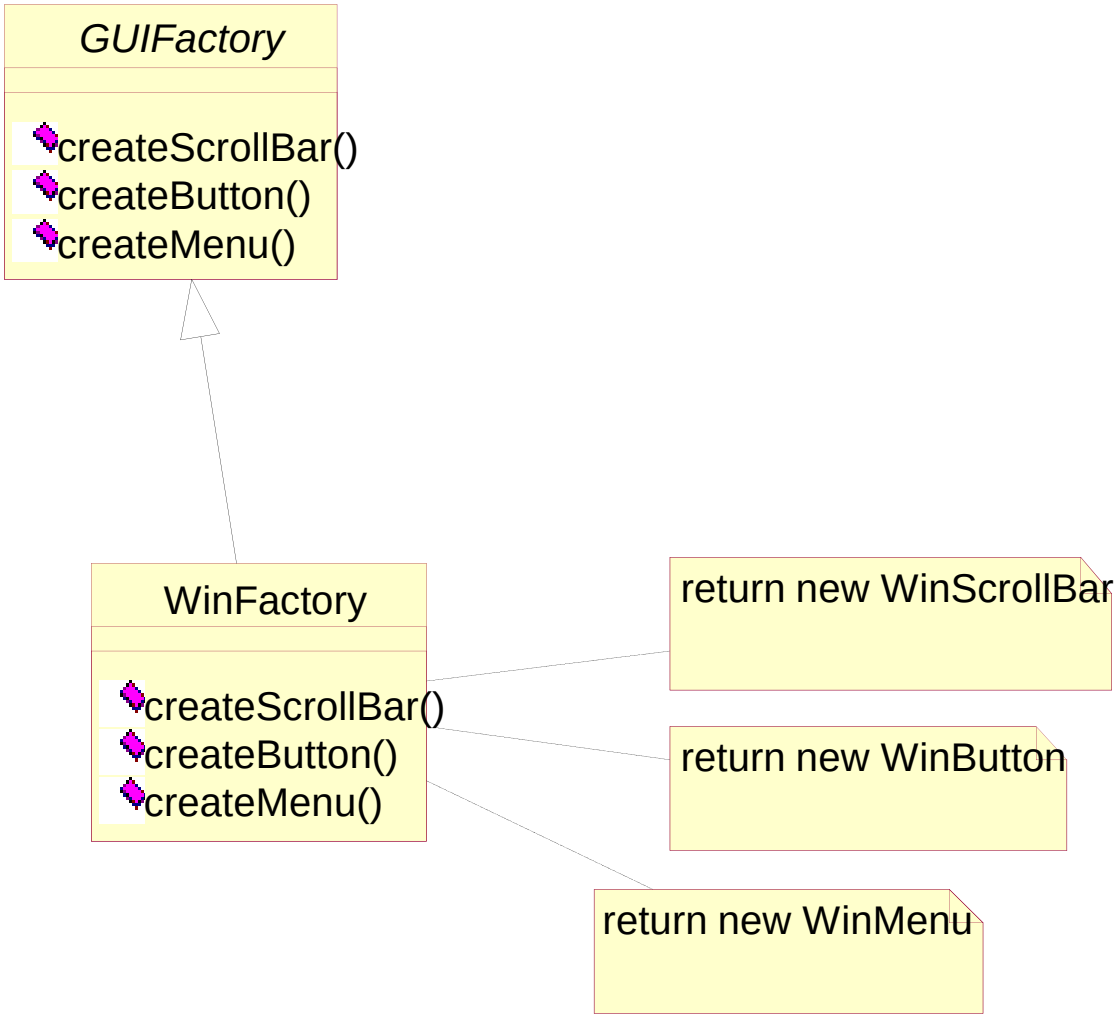
- La soluzione:
  - disporre di particolari funzioni di creazione generalizzate
  - raggruppare tali funzioni di creazione in una classe “GUIFactory” (fabbrica) responsabile di creare su richiesta gli elementi dell’interfaccia utente
  - diverse sottoclassi della classe factory forniranno le implementazioni specializzate delle funzioni di creazione
  - una sottoclasse per ognuno dei diversi standard di presentazione

  - ScrollBar sb = guiFactory.createScrollBar();

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio



Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio



Uso della GUIFactory

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract
Factory
Composite
Façade
Observer
Pattern
architetturali
MVC
➢ Esempio

- La creazione degli oggetti dell'interfaccia utente è effettuata accedendo ad una variabile di tipo GUIFactory
  Ad esempio:
  - GUIFactory guiFactory;
  - ScrollBar sb = guiFactory.createScrollBar();
- Per supportare diversi standard di presentazione la variabile guiFactory assume valori iniziali diversi
  Ad esempio: guiFactory = new MotifFactory;

Valutazione della soluzione GUIFactory

Introduzione
Definizioni
Categorie
Factory
Method
Singleton
Flyweight
State
Strategy
Proxy
Adaptor
Decorator
Abstract Factory
Composite
Façade
Observer
Pattern architetturali
MVC
➢ Esempio

- Il codice con la logica dell'applicazione non deve essere modificato per gestire la creazione di elementi di interfaccia utente che rispettano standard di presentazione diversi
- Uno fra i diversi standard di presentazione puo' essere scelto allo start-up dell'applicazione
- Solo una linea di codice (quella in cui la variabile guifactory viene inizializzata) deve essere modificata per avere presentazioni diverse
- La GUIFactory garantisce da errori quali il tentativo di creazione di un'interfaccia utente composta da elementi appartenenti a standard di presentazione diversi
- Il codice deve seguire una nuova convenzione per la creazione degli oggetti dell'interfaccia utente (non vengono usati i costruttori standard)

Formato di un design pattern SW

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Proxy
- Observer
- Pattern architetturali
- MVC
- Esempio

- Tipicamente (vedi il libro della “Gang of Four”) è suddiviso in (differisce dal formato “classico” di Alexander)
  - Name & classification
  - Intent
  - AKA
  - Motivation (scenario)
  - Applicability
  - Structure
  - Participants
  - Collaborations
  - Consequences
  - Implementation
  - Sample code
  - Known uses
  - Related patterns

Come scegliere un pattern

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Proxy
- Observer
- Pattern architetturali
- MVC
- Esempio

- Valutare come i DP risolvono i problemi di progetto
- Considerare l'intento di ciascun DP
- Studiare le interrelazioni tra DP
- Studiare i pattern con funzioni simili
- Design for change!
  - esaminare le cause che possono implicare la modifica del progetto
  - considerare le caratteristiche di flessibilità e adattabilità desiderate

Come usare un pattern

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Proxy
- Observer
- Pattern architetturali
- MVC
- Esempio

- Leggerne l'intera descrizione
  - soffermandosi con attenzione su Applicability e Consequences
- Studiare Structure, Participants e Collaborations
  - per capire le relazioni
- Leggere il sample code
  - per imparare come si implementa in pratica
- Scegliere nomi significativi nel dominio applicativo per gli elementi del DP
- Definire le classi
  - interfacce, eredità, attributi, … modifica di classi esistenti per armonizzarle col DP
- Dare ai metodi nomi significativi nel dominio applicativo
- Implementare le operazioni che realizzano responsabilità e collaborazioni nel DP

Come NON usare i pattern

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Proxy
- Observer
- Pattern architetturali
- MVC
- Esempio

- I pattern spesso aggiungono livelli di indirezione, che possono appesantire il progetto in termini di complessità e prestazioni
- Questo perché per essere ri-usabili devono avere un certo grado di astrazione
- Quindi dovrebbero essere usati quando la flessibilità che forniscono è realmente necessaria
- La descrizione delle Conseguenze dell'adozione del pattern fornisce elementi decisionali spesso utili

Conclusioni

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Proxy
- Observer
- Pattern architetturali
- MVC
- Esempio

- I design pattern offrono un importante contributo alla catalogazione delle strutture tipiche che si trovano in maniera ricorrente nel progetto di un sistema software
- Che cosa dobbiamo aspettarci dallo studio e dall'applicazione dei pattern?
  - Un vocabolario comune di progetto
  - Un aiuto alla documentazione
  - Un aiuto all'apprendimento delle tecniche object-oriented
  - Un supporto alla reingegnerizzazione del software

Conclusioni

- Introduzione
- Definizioni
- Categorie
- Factory Method
- Singleton
- Flyweight
- State
- Strategy
- Proxy
- Adaptor
- Decorator
- Abstract Factory
- Composite
- Façade
- Proxy
- Observer
- Pattern architetturali
- MVC
- Esempio

- I pattern forniscono un vocabolario comune tra i progettisti, che facilita la comprensione di un progetto esistente o lo sviluppo di uno nuovo
  - Abbiamo visto solo un piccolo insieme di pattern:
- I pattern migliorano le prestazioni del codice e/o lo rendono più flessibile
- Tuttavia, il codice che utilizza i pattern potrebbe risultare più complesso del necessario: occorre quindi valutare e confrontare costi e benefici
- **http://www.dofactory.com/Patterns/Patterns.aspx**