LEZIONE 29
DESIGN PATTERNS 1:
Introduzione ad alcuni GoF

Ingegneria del Software e Progettazione Web

Università degli Studi di Tor Vergata - Roma

Guglielmo De Angelis
guglielmo.deangelis@isti.cnr.it

cosa è UML

"*In short, the Unified Modeling Language (UML) provides industry standard mechanisms for visualizing, specifying, constructing, and documenting software systems.*"

- UML è un linguaggio
- UML non un impone l'adesione ad uno specifico processo di sviluppo del software
- la conoscenza "sintattica" di UML non è funzione di un'appropriata applicazione dei principi O.O.

ipotesi

- supponiamo di disporre di un metodo che ci consenta di:
  - identificare in modo preciso un problema ricorrente
  - proporre (almeno) un suggerimento che definisca uno schema generale di soluzione
  - avere un modo univoco per identificare la coppia: problema/soluzione

tesi

- l’ipotetico metodo contribuirebbe:
  - alla comunicazione tra progettisti
  - alla definizione di tecniche sistematiche a supporto della progettazione
  - all'istruzione ed all'apprendimento di tali tecniche
  - alla discussione ed alla valutazione di tecniche di progettazione similari o alternative

soluzione: prima approssimazione

- in generale definiamo
  - pattern = problema ricorrente + schema di soluzione
- nelle precedenti lezioni, abbiamo mai provato a ragionare in termini di “pattern”?

soluzione: prima approssimazione

- in generale definiamo
  - pattern = problema ricorrente + schema di soluzione
- nelle precedenti lezioni, abbiamo mai provato a ragionare in termini di “pattern”?
- **SI**; qualche pattern lo abbiamo già incontrato:
  - vedere lezioni sulle classi e sulle interfacce
    - polimorfismo
    - metamorfosi
  - vedere lezioni su GRASP
    - legge di Demetra

un po' di storia sui pattern

- 1987 – Cunningham e Beck utilizzarono le idee di Alexander per sviluppare un piccolo linguaggio di pattern per Smalltalk
- 1990 – Gang of Four (Gamma, Helm, Johnson e Vlissides) iniziano a realizzare un catalogo di design pattern
- 1991 – Bruce Anderson a OOPSLA mostra i primi patterns
- 1993 – Kent Beck e Grady Booch sponsorizzano il primo meeting che è conosciuto come Hillside Group
- 1994 – Conferenze First Pattern Languages of Programs (PLoP)
- 1995 – Gang of Four (GoF) pubblicano il libro Design Patterns

GoF : the Gang of Four

- nel 1995 Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides pubblicano :
  - Design Patterns: Elements of Reusable Object-Oriented Software
- “…It's a book of design patterns that describes simple and elegant solutions to specific problems in object-oriented software design.”
  - viene definito un template di riferimento per specificare un pattern
  - vengono identificati e discussi un insieme di design patterns

design patterns

- definizione: è una descrizione di un problema ricorrente nella progettazione. Inoltre ad ogni problema viene associato:
  - un nome
  - una soluzione che può essere “istanziata” in differenti circostanze anche eterogenee tra loro
  - una discussione sulle relative conseguenze e variazioni che conseguono l'applicazione della soluzione
- generalmente i pattern sono definiti/raggruppati per categorie di problemi che intendono risolvere o per domini applicativi
- in pratica un pattern definisce una regola che codifica un'appropriata applicazione dei principi O.O. su problemi ben conosciuti e ricorrenti
- l'uso e la composizione di pattern supportano i modellisti/architetti software verso la definizione di soluzioni dove sia mitigata l'influenza di fattori umani legati ad esperienze personali
  - le scelte di design sono prese in base a soluzioni consolidate

“nuovo” VS design patterns

- il punto focale dei design pattern è quello di *formalizzare* e *strutturare* idiomi/problemi ricorrenti ed esistenti
  - il loro scopo:
    - è supportare l'applicazione di tecniche consolidate
    - non è fornire nuovi spunti alla progettazione
- il termine “nuovo pattern” dovrebbe essere considerato un ossimoro, se esso è inteso per descrivere una nuova idea di progettazione

GoF – struttura

GoF – struttura – 1

- Nome pattern e Classificazione
  - dovrebbero essere altamente significativi
- Intento
  - breve descrizione per mostrare ciò che fa il pattern
  - qual è il suo fondamento/intento?
- Sinonimi
  - altre nomenclature non ufficiali usate per referenziare il pattern
- Motivazione
  - scenario che illustra un problema di progettazione e il modo in cui la struttura di classi e oggetti definita nel pattern lo risolve
- Applicabilità
  - situazioni dove il pattern può essere utilizzato

GoF – struttura – 2

- Struttura
  - rappresentazione grafica del pattern
- Partecipanti
  - classi e oggetti che fanno parte del design e le loro responsabilità
- Collaborazioni
  - come collaborano i partecipanti per potersi assumere le loro responsabilità
- Conseguenze
  - come fa il pattern a raggiungere i propri obiettivi
  - pro e contro nell'applicazione del pattern

GoF – struttura – 3

- Implementazione
  - suggerimenti e tecniche per implementare il pattern
  - problemi specifici connessi a un particolare linguaggio di programmazione
- Codice di esempio
  - frammenti di codice (le descrizioni originali sono per C++ o Smalltalk)
- Usi conosciuti
  - utilizzo di pattern in sistemi reali
- Pattern correlati
  - altri pattern che sono in relazione

catalogo GoF – 1

- i pattern GoF sono organizzati in un catalogo secondo due criteri di classificazione:
  - **scopo**: definisce il dominio di applicazione del pattern
    - Creational: riguarda il processo di creazione di oggetti
    - Structural: riguarda aspetti di composizione per classi ed oggetti
    - Behavioral: riguarda come classi o oggetti interagiscono nel raggiungimento di obiettivi assegnati
  - **contesto**: definisce la tipologia di elementi cui il pattern può essere applicato
    - Class: considera relazioni tra classi e loro sottoclassi (struttura statica)
    - Object: considera relazioni tra oggetti modificabili a run-time (struttura dinamica)

catalogo GoF – 2

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

catalogo GoF – 3

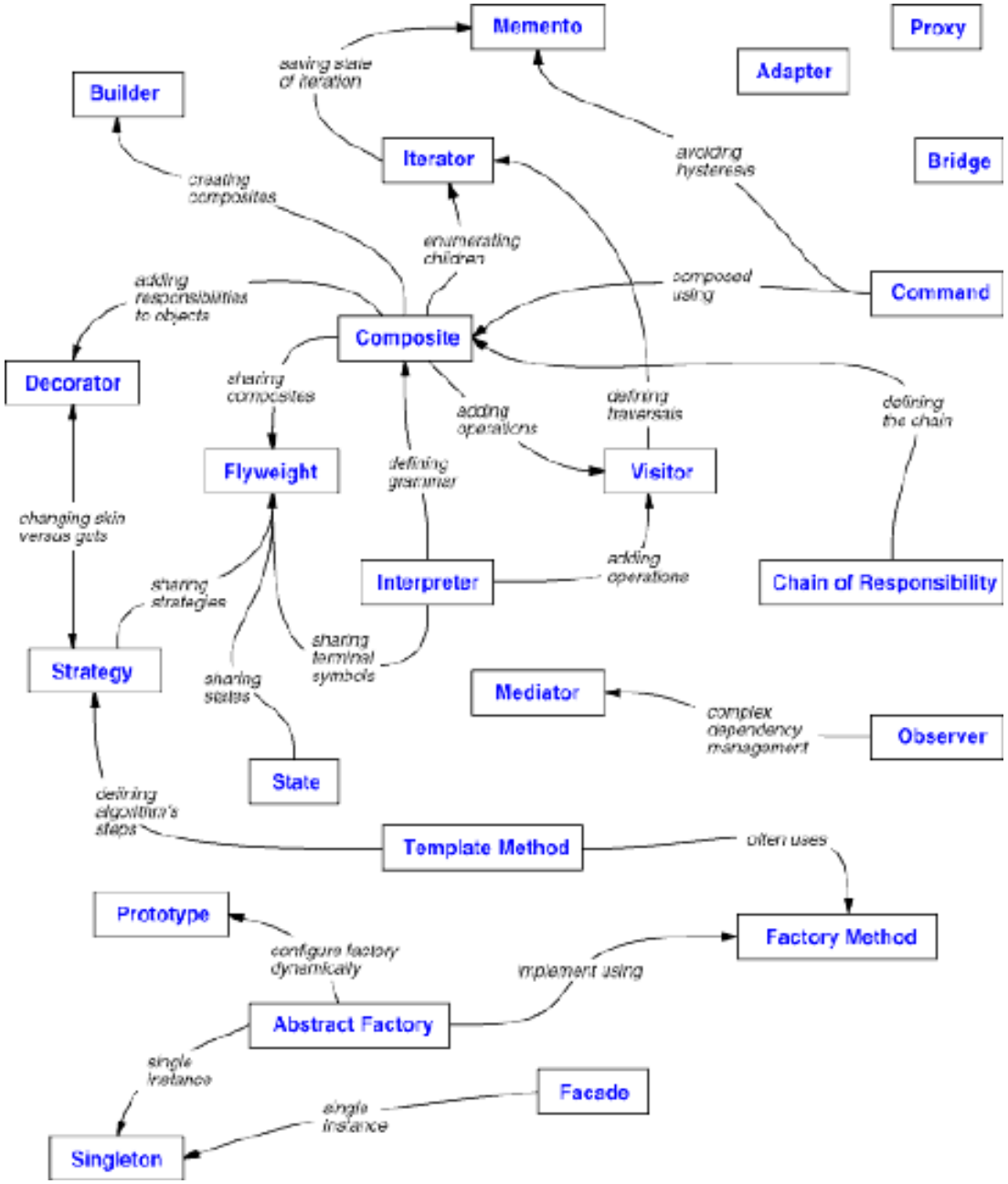
- **Creational && Class**: delegano parte del processo di creazione di un oggetto a sottoclassi
- **Creational && Object**: delegano parte del processo di creazione di un oggetto ad altri oggetti
- **Structural && Class**: utilizzano l'ereditarietà per comporre classi,
- **Structural && Object**: descrivono modi per raggruppare oggetti
- **Behavioral && Class**: utilizzano ereditarietà per descrivere algoritmi e flusso di controllo
- **Behavioral && Object**: descrivono come gruppi di oggetti cooperano per eseguire un compito che un singolo oggetto non potrebbe portare a termine da solo

GoF – struttura – 3

- Implementazione
  - suggerimenti e tecniche per implementare il pattern
  - problemi specifici connessi a un particolare linguaggio di programmazione
- Codice di esempio
  - frammenti di codice (le descrizioni originali sono per C++ o Smalltalk)
- Usi conosciuti
  - utilizzo di pattern in sistemi reali
- Pattern correlati
  - altri pattern che sono in relazione

ABBIAMO ACCENNATO AL FATTO CHE I DESIGN PATTERNS NON SONO ENTITÀ INDIPENDENTI TRA LORO

relazioni tra GoF



pattern creazionali

- forniscono un'astrazione del processo di istanziazione degli oggetti e rendono il sistema indipendente da tale modalità
- basati su classi utilizzano ereditarietà per scegliere la particolare classe da istanziare
- basati su oggetti delegano l'istanziazione ad un altro oggetto
- rendono il sistema maggiormente flessibile poiché conosce soltanto le interfacce degli oggetti definite mediante classi astratte

factory method – 1

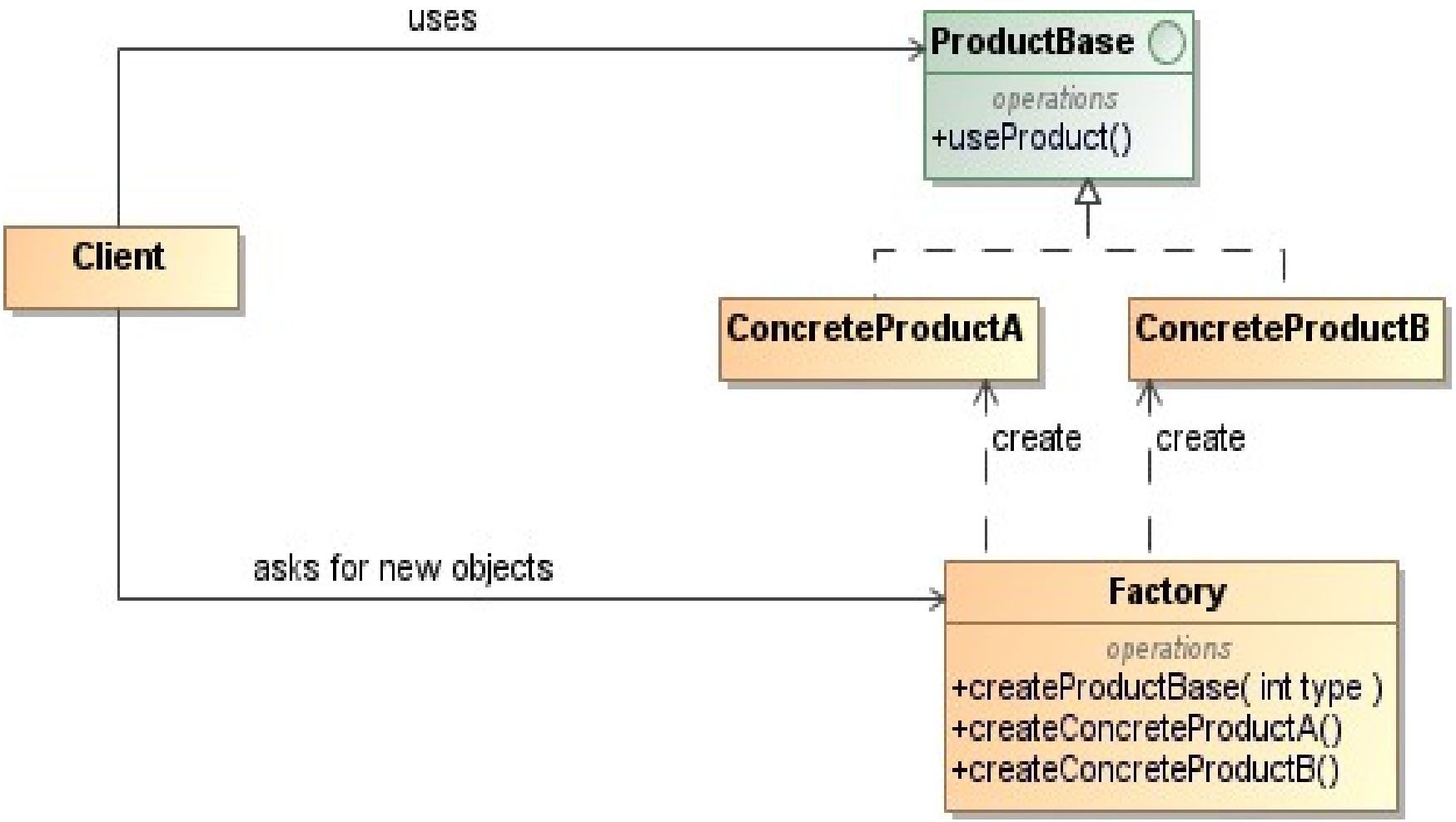
- Scopo
  - fornire un meccanismo per la creazione di oggetti simili (e.g. che implementano la stessa interfaccia) senza specificare quali siano le loro classi concrete
- Sinonimi
  - Virtual Constructor
- Motivazione
  - si ha bisogno di determinare l'esatto tipo dell'oggetto da instanziare solo a run-time

factory method – 2

- Applicabilità
  - sistema deve essere indipendente dalle modalità di creazione, composizione e rappresentazione dei suoi prodotti
  - si vuole una libreria (i.e. insieme di classi) che esponga soltanto l'interfaccia e non la sua l'implementazione

factory method – 3

- Struttura



factory method – 4

- Partecipanti
  - Factory (`Factory`)
    - dichiara ed implementa i meccanismi di creazione per un insieme oggetti simili (e.g. che condividono la stessa interfaccia)
  - AbstractProduct (`ProductBase`)
    - dichiara un'interfaccia (i.e. classe astratta o interface) per una tipologia di oggetti prodotto
  - ConcreteProduct (`ConcreteProductA, ConcreteProductB ConcreteProductC`)
    - definisce un oggetto prodotto che dovrà essere creato dalla corrispondente factory
    - implementa l'interfaccia definita da AbstractProduct
  - Client (`Client`)
    - crea istanze di ConcreteProduct attraveso la Factory
    - utilizza soltanto l'interfaccia dichiarate in AbstractProduct

factory method – 5

- Collaborazioni
  - durante l'esecuzione è preferibile riferire "istanze uniche" per le classi classe `Factory`
  - la `Factory` gestisce la creazione di oggetti simili con un'implementazione specifica
- Conseguenze
  - isola classi concrete
  - consente di cambiare in modo semplice l'implementazione ed in comportamenti esposti da un insieme di prodotti
  - promuove il riuso di prodotti/artefatti/codice
  - aggiunta e supporto di nuovi di prodotti semplice

factory method – 6

- Implementazione
  - creazioni dei prodotti
    - ogni prodotto viene creato mediante un factory method
  - nella formulazione originale si propone un unico metodo che restituisce un solo tipo di prodotto
    - un unico metodo con un parametro in ingresso che stabilisce il tipo del prodotto
    - questo aspetto è sconsigliato in quanto tende ad essere poco sicuro

factory method – 7

- Codice di esempio :

VEDERE MATERIALE ALLEGATO ALLA LEZIONE

abstract factory – 1

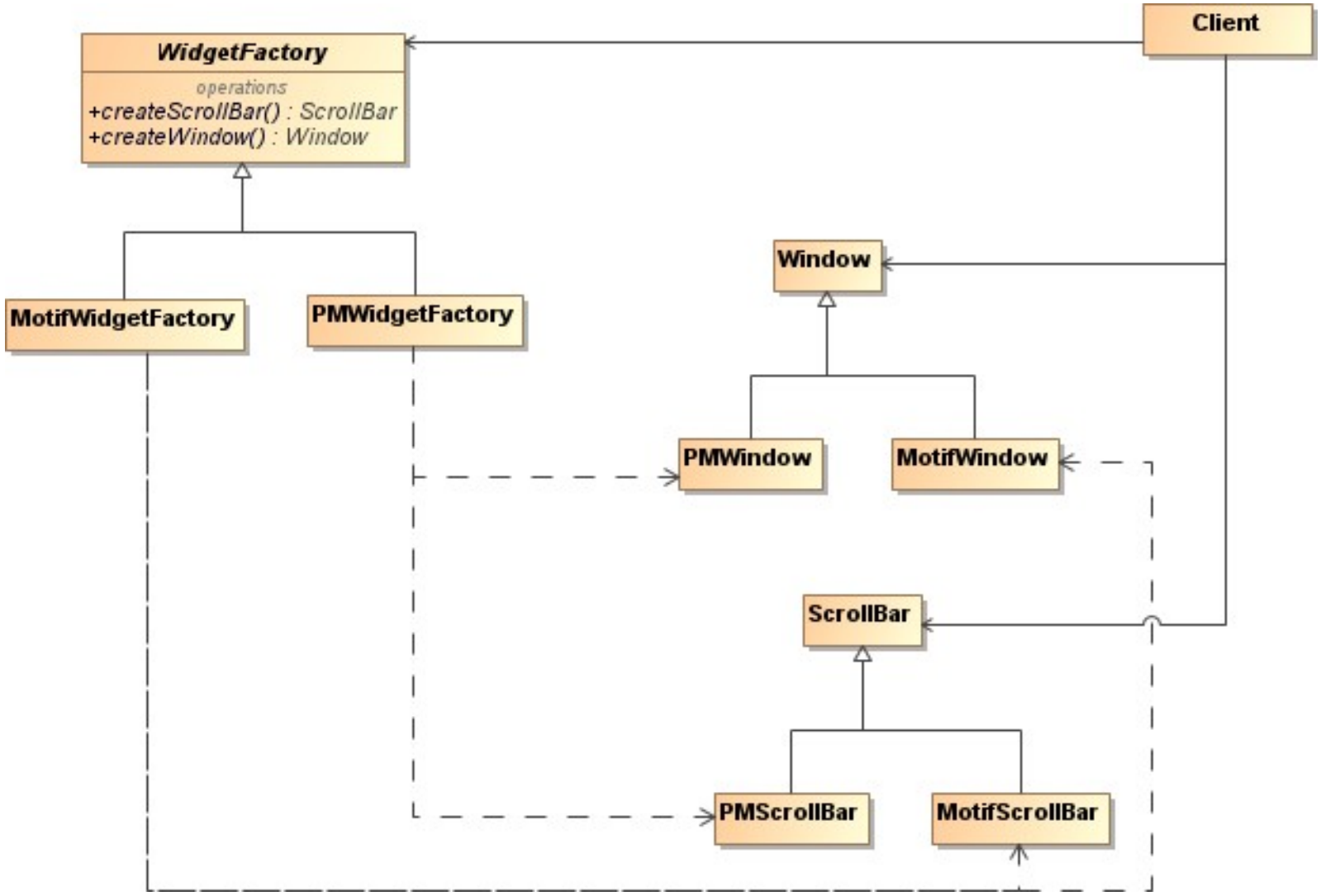
- Scopo
  - fornire un'interfaccia per la creazione di **famiglie di oggetti** correlati o dipendenti senza specificare quali siano le loro classi concrete
- Sinonimi
  - Kit
- Motivazione
  - GUI toolkit che supporta diversi standard di look-and-feel (i.e. motif, presentation manager)
  - diverse modalità di presentazione e comportamento per gli elementi (widget)
  - per garantire portabilità gli elementi grafici di un look-and-feel non devono essere cablati nel codice

abstract factory – 2

- Applicabilità
  - sistema deve essere indipendente dalle modalità di creazione, composizione e rappresentazione dei suoi prodotti
  - esistono diverse famiglie di prodotti alternative tra loro e che devono essere gestite in modo omogeneo
  - ogni prodotto espone un insieme condiviso di operazioni indipendentemente dalla famiglia
  - sistema deve poter essere configurato scegliendo una tra più famiglie di prodotti
  - si vuole una libreria (i.e. insieme di classi) che esponga soltanto l'interfaccia e non la sua l'implementazione

abstract factory – 3

- Struttura



abstract factory – 4

- Partecipanti
  - AbstractFactory (`WidgetFactory`)
    - dichiara un'interfaccia (i.e. classe astratta o interface) per le operazioni di creazione di oggetti prodotto astratti
  - ConcreteFactory (`MotifWidgetFactory`, `PMWidgetFactory`)
    - implementa le creazioni degli oggetti prodotto concreti
  - AbstractProduct (`Window`, `ScrollBar`)
    - dichiara un'interfaccia (i.e. classe astratta o interface) per una tipologia di oggetti prodotto
  - ConcreteProduct (`MotifWindow`, `MotifScrollBar`)
    - definisce un oggetto prodotto che dovrà essere creato dalla corrispondente factory concreta
    - implementa l'interfaccia AbstractProduct
  - Client (`Client`)
    - utilizza soltanto le interfacce dichiarate dalle classi AbstractFactory e AbstractProduct

abstract factory – 5

- Collaborazioni
  - durante l'esecuzione è preferibile riferire "istanze uniche" per le classi classe `ConcreteFactory`
  - la factory concreta gestisce la creazione di una famiglia di oggetti con un'implementazione specifica
  - `AbstractFactory` delega la creazione di oggetti prodotto alle sue sottoclassi `ConcreteFactory`
- Conseguenze
  - isola classi concrete
  - consente di cambiare in modo semplice famiglia di prodotti utilizzata
  - promuove coerenza utilizzo di prodotti
  - aggiunta del supporto di nuovi tipologie di prodotti difficile

abstract factory – 6

- Implementazione
  - factory più generale come classe astratta
    - eventualmente anche come interfaccia
    - la versione che stiamo discutendo differisce da quella originariamente proposta da GoF. Nelle slide che seguono analizziamo quella originale.
  - creazioni dei prodotti
    - ogni prodotto viene creato mediante un factory method
    - ogni famiglia di prodotti è istanziata attraverso una specifica sottoclasse
  - nella formulazione originale si propone un unico metodo che restituisce un solo tipo di prodotto
    - questa versione è fortemente sconsigliata da un punto di vista logico e non è adatto a Java . Il suo uso resta confinato a per particolari linguaggi.
  - definire factory estendibili ovvero un unico metodo con un parametro in ingresso che mi stabilisce il tipo del prodotto
    - questo aspetto è sconsigliato in quanto poco sicuro

abstract factory – 7

- Codice di esempio :

VEDERE MATERIALE ALLEGATO ALLA LEZIONE

singleton – 1

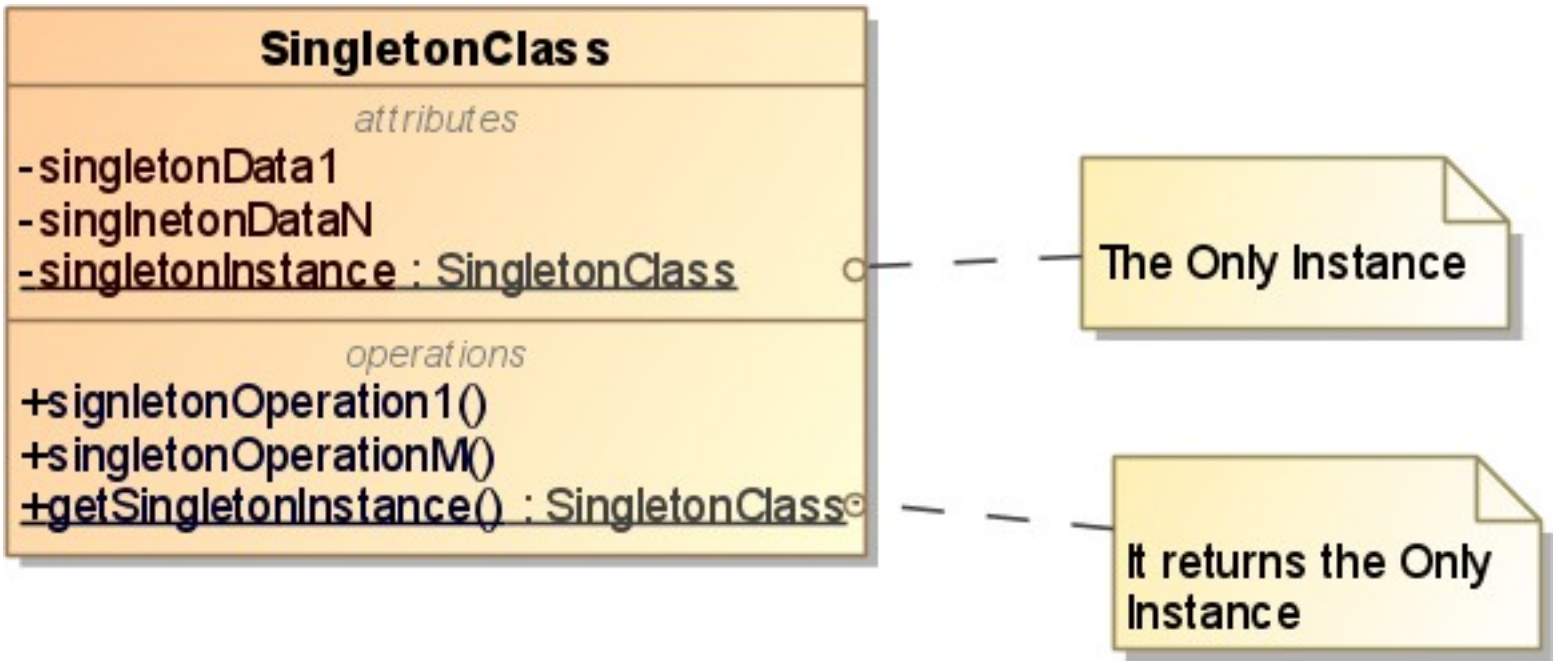
- Scopo
  - assicurare che una classe abbia una sola istanza nell'applicazione
  - fornire un punto d’accesso globale a tale istanza
- Motivazione
  - esempi
    - diverse stampanti ma una sola coda di stampa
    - unico window manager
  - un'unica classe con la responsabilità di creare le altre istanze

singleton – 2

- Applicabilità
  - deve esistere esattamente un'istanza di una classe e tale istanza deve essere accessibile ai client attraverso un punto di accesso noto a tutti gli utilizzatori
  - l'unica istanza deve poter essere estesa attraverso la definizione di sottoclassi e i client devono essere in grado di utilizzare le istanze estese senza dover modificare il proprio codice

singleton – 3

- Struttura



singleton – 4

- Partecipanti
  - Singleton (`SingletonClass, BetterSingletonClass, LazySingletonClass`)
    - definisce un'operazione `getSingletonInstance` che consente ai Client di accedere all'unica istanza esistente della classe
  - `getSingletonInstance` deve essere un'operazione di classe
  - può essere responsabile della creazione della sua unica istanza
- Collaborazioni
  - i Client possono accedere a un'istanza di un singleton soltanto attraverso l'operazione `getSingletonInstance`

singleton – 5

- Conseguenze
  - accesso controllato a un’unica istanza
  - riduzione dello spazio dei nomi ovvero non è necessario definire variabili globali
  - permette il raffinamento di operazioni e rappresentazione ovvero è possibile definire delle sottoclassi che costituiscono l’unica istanza
  - permette di gestire un numero variabili di istanze
  - maggiore flessibilità rispetto a operazioni di classe

singleton – 6

- Implementazione
  - assicurare l'esistenza di un'unica istanza
    - costruttore privato
    - variabile di classe privata
    - metodo di classe che restituisce la variabile di classe
  - definizione di sottoclassi di Singleton
    - costruttore protetto
    - metodo set oppure utilizzo di meccanismi globali per ottenere l'istanza della sottoclasse

singleton – 7

- Codice di esempio :

VEDERE MATERIALE ALLEGATO ALLA LEZIONE

esercizio – 1.1

- come si comportano le classi SINGLETON in sistemi concorrenti?

- quali sono i possibili accorgimenti, che diventa necessario considerare?

esercizio – 1.2

- Codice di esempio :

VEDERE MATERIALE ALLEGATO ALLA LEZIONE

esercizio – 2.1

- idee su come gestire in modo “più elegante” soluzioni basate sul design pattern Abstract Factory ?!!?!

esercizio – 2.2

- idee su come gestire in modo “più elegante” soluzioni basate sul design pattern Abstract Factory ?!!?!

- proviamo a combinare l'uso di Abstract Factory con Singleton

singleton && abstract factory

- Codice di esempio :

VEDERE MATERIALE ALLEGATO ALLA LEZIONE

bibliografia di riferimento

- “Design Patterns – Elementi per il riuso di software a oggetti”, Gamma, Helm, Johnson, Vlissides (GoF). Addison-Wesley (1995).
  - Design Patterns: Elements of Reusable Object-Oriented Software
- “Applicare UML e i Pattern – Analisi e progettazione orientata agli oggetti”, Larman. 3za Edizione. Pearson (2005).
  - “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development”