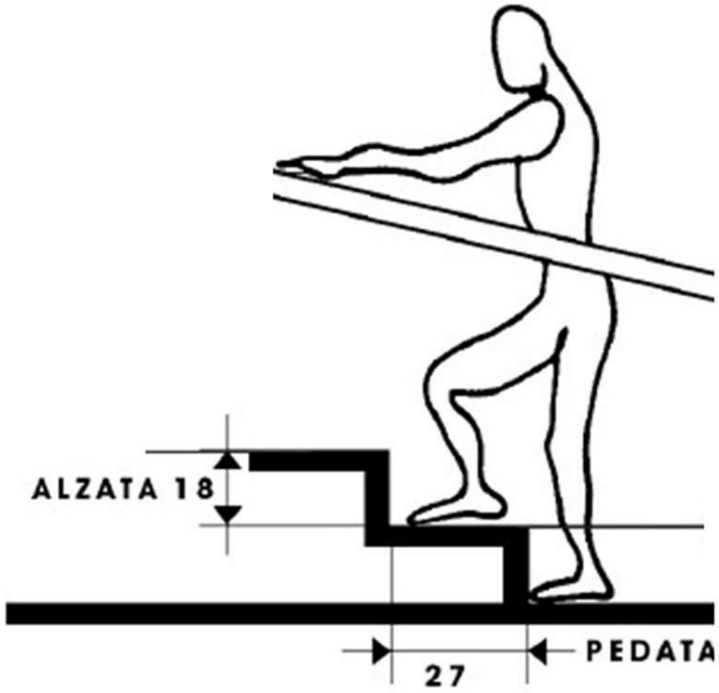
TECNICHE DI PROGETTAZIONE: DESIGN PATTERNS

Prof. Mariacarla Staffa


ATTENTION
BAD DESIGN
“CALATRAVA BRIDGE”

PATTERN PER UN GRADINO



- **Formula di Blondel:**
- **2Alzata + Pedata = 62 ÷ 65 cm**
- Questa formula ha dei limiti sulle piccole altezze di alzata:
  - quando i gradini sono bassi conduce a dei **risultati insoddisfacenti.**

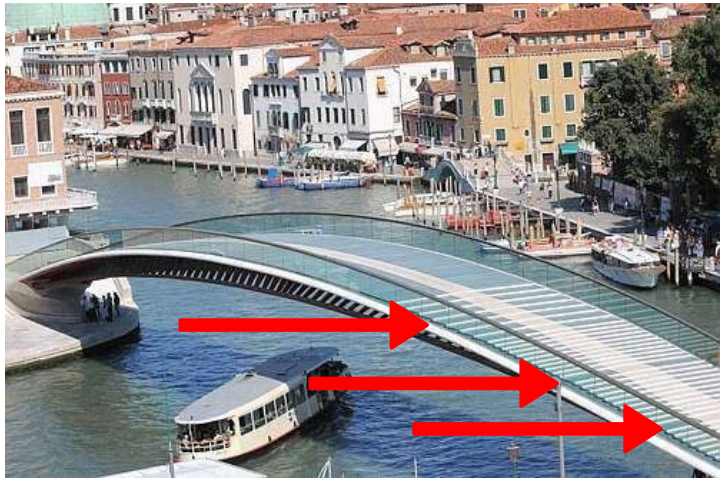
- **Regola di Hermant:** proporzione costante fra alzata e le variabili del passo, espresso dalla formula

"H x P =600"

dove H è l'altezza e P è il passo Alzata * Pedata = 600

PONTE DI CALATRAVA

- Pedata= 50, Altezza = 8
  - Blondel:50+2x8=66 ≈ 62 ÷ 65 cm
  - Regola di Hermant: 50*8=400 ≠ 600



DOV'È IL GRADINO?

MATERIALI

- Pietra d'Istria alternata alla trachite scura
- Soluzione individuata dai veneziani già nel XV secolo.

SO WHAT?

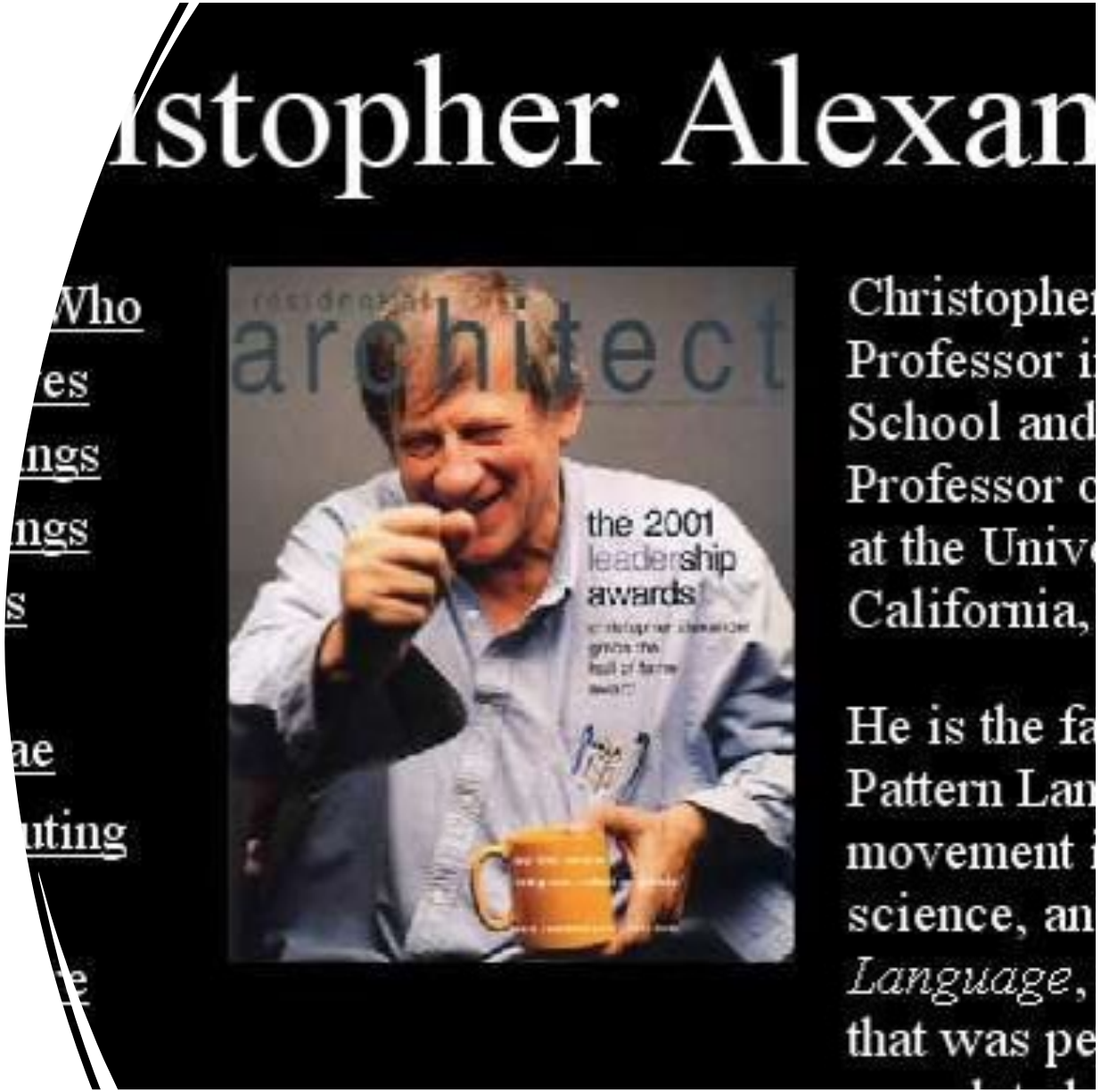
- Esistono una serie di *regole pratiche* che il progettista può seguire per costruire una scala:
  - Rapporto salita/corsa (Blondel e Hermant)
  - Materiali...
- Queste regole pratiche sono i design patterns.
- Sono state definite grazie a secoli di esperienza.

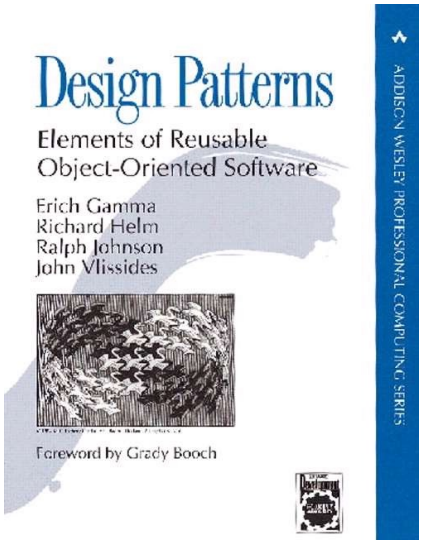

IL DESIGN NON È SOLTANTO UN PROCESSO CREATIVO

COSA È UN (DESIGN) PATTERN?

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

- Christopher Alexander
- A Pattern Language, 1977



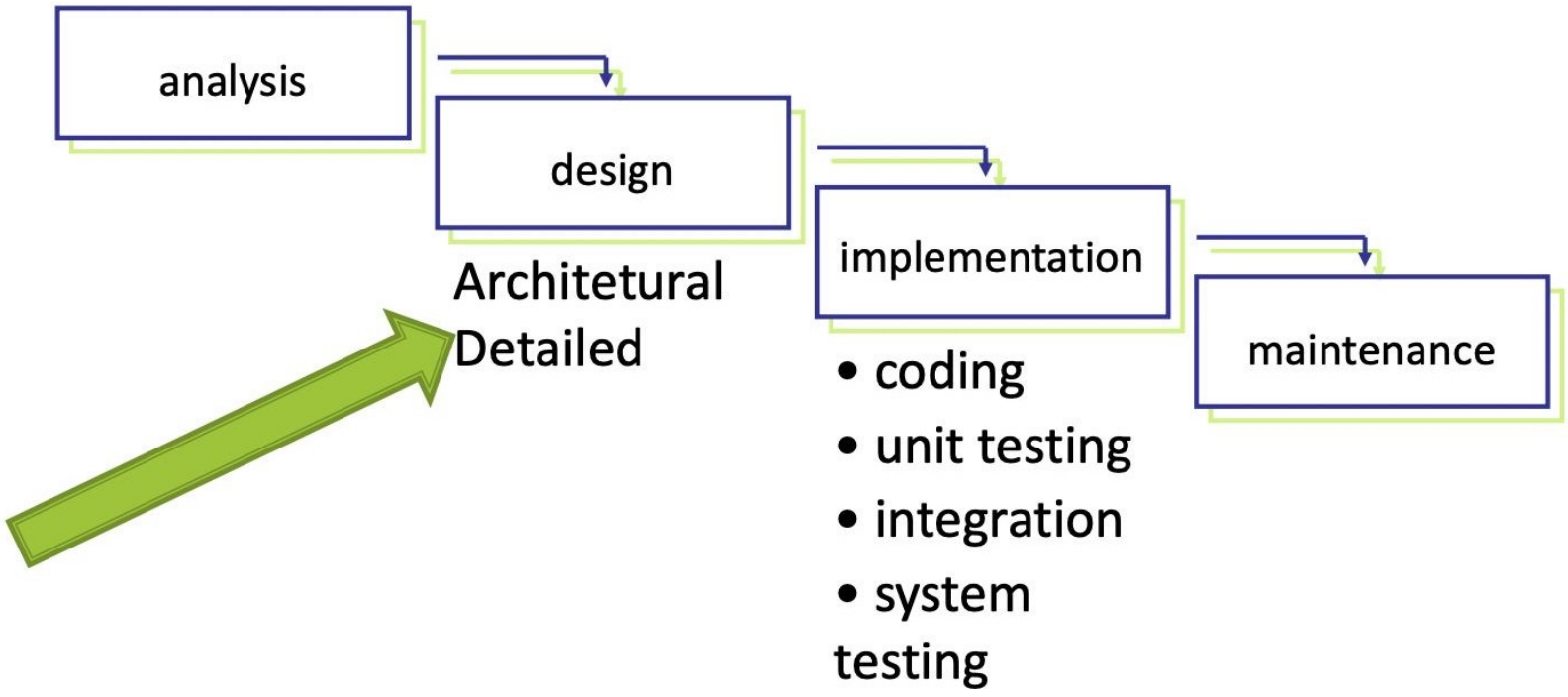


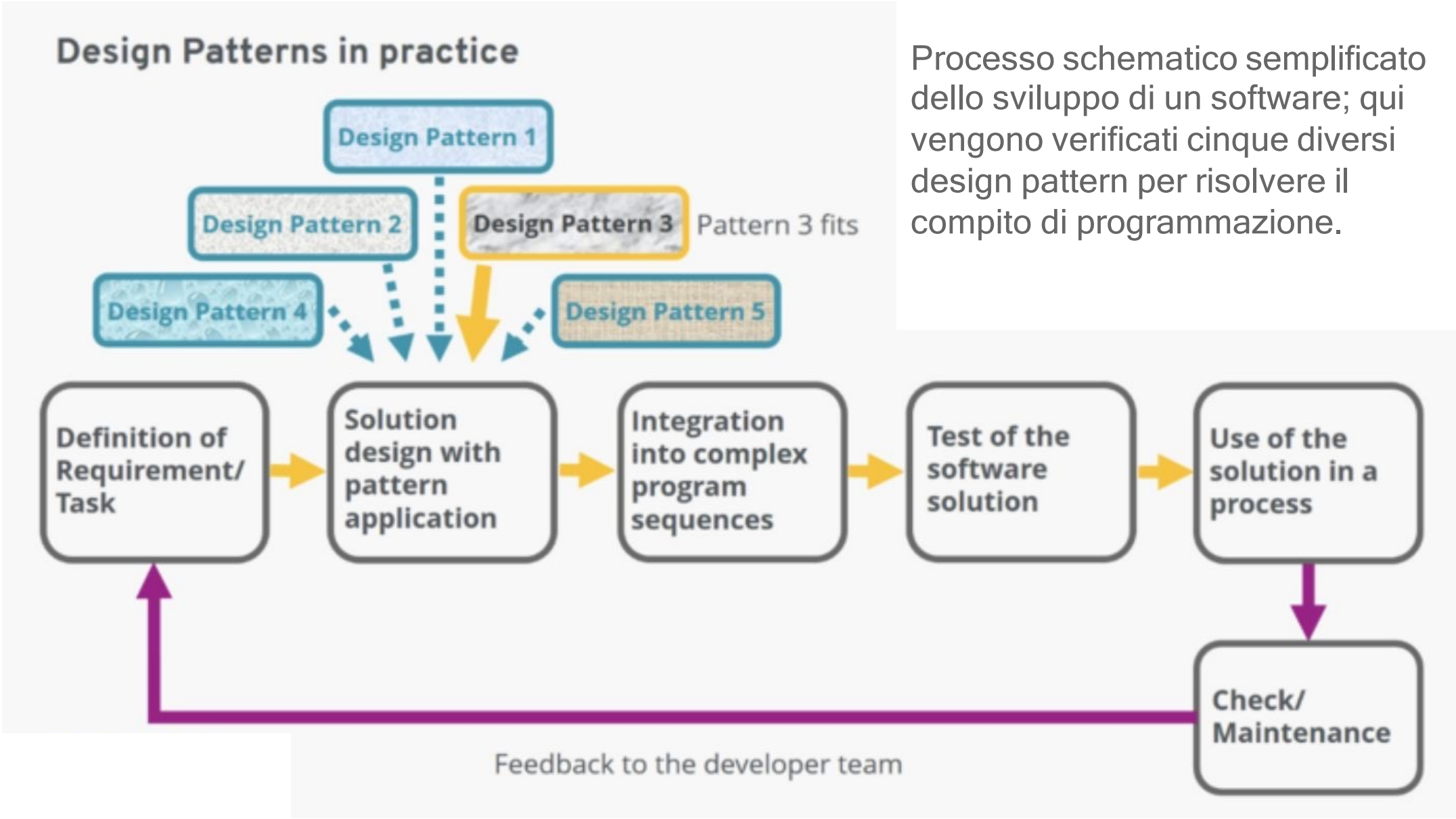
LIBRI SUGGERITI

DEFINIZIONE

- C. Alexander ha definito i design patterns studiando tecniche per migliorare il processo di progettazione di edifici e aree urbane
- Ogni pattern è una regola in tre parti, che esprime una relazione tra
  - Un contesto
  - Un problema
  - Una soluzione
- DEF: "una soluzione a un problema in un contesto"
- I pattern possono essere applicati a diverse aree, compreso lo sviluppo software

IN CHE FASE SI APPLICANO





Processo schematico semplificato dello sviluppo di un software; qui vengono verificati cinque diversi design pattern per risolvere il compito di programmazione.

PERCHÉ I PATTERN NEL SOFTWARE?

"Progettare software OO è difficile e progettare software OO riutilizzabile è ancora più difficile". – Erich Gamma

Iprogettisti esperti riutilizzano le soluzioni che hanno funzionato in passato.

I sistemi OO ben strutturati hanno modelli ricorrenti di classi e oggetti.

La conoscenza degli schemi che hanno funzionato in passato consente al progettista di essere più produttivo e ai progetti risultanti di essere più flessibili e riutilizzabili.

DESIGN PATTERN LEVELS OF ABSTRACTION

Complex design for an entire application or subsystem

Solution to a general design problem in a particular context

Simple reusable design class such as a linked list, hash table, etc.

More Abstract

More Concrete

ARCHITETTURA-PROGETTAZIONE DI DETTAGLIO-CODICE

Patterns o stili architecturali

Pipes and Filters, Publish-Subsribe, Model-View-Controller, ...

Design Patterns

Progettazione e raffinamento dei componenti.

E.g. abstract factory, decorator, ...

Idioms o Coding Design Patterns

Pattern di basso livello specifici di un linguaggio di programmazione.

Un idioma è più limitato di un modello di progettazione, ma descrive comunque un problema ricorrente

Ad esempio, in C, allocazione e deallocazione della memoria, convenzioni sui nomi delle variabili ...

DESIGN PATTERN

Con i programmi informatici determinati processi tendono sempre a ripetersi, per questo è nata l'idea di creare dei modelli.

Questi schemi progettuali (chiamati design pattern) possono semplificare il lavoro di programmazione

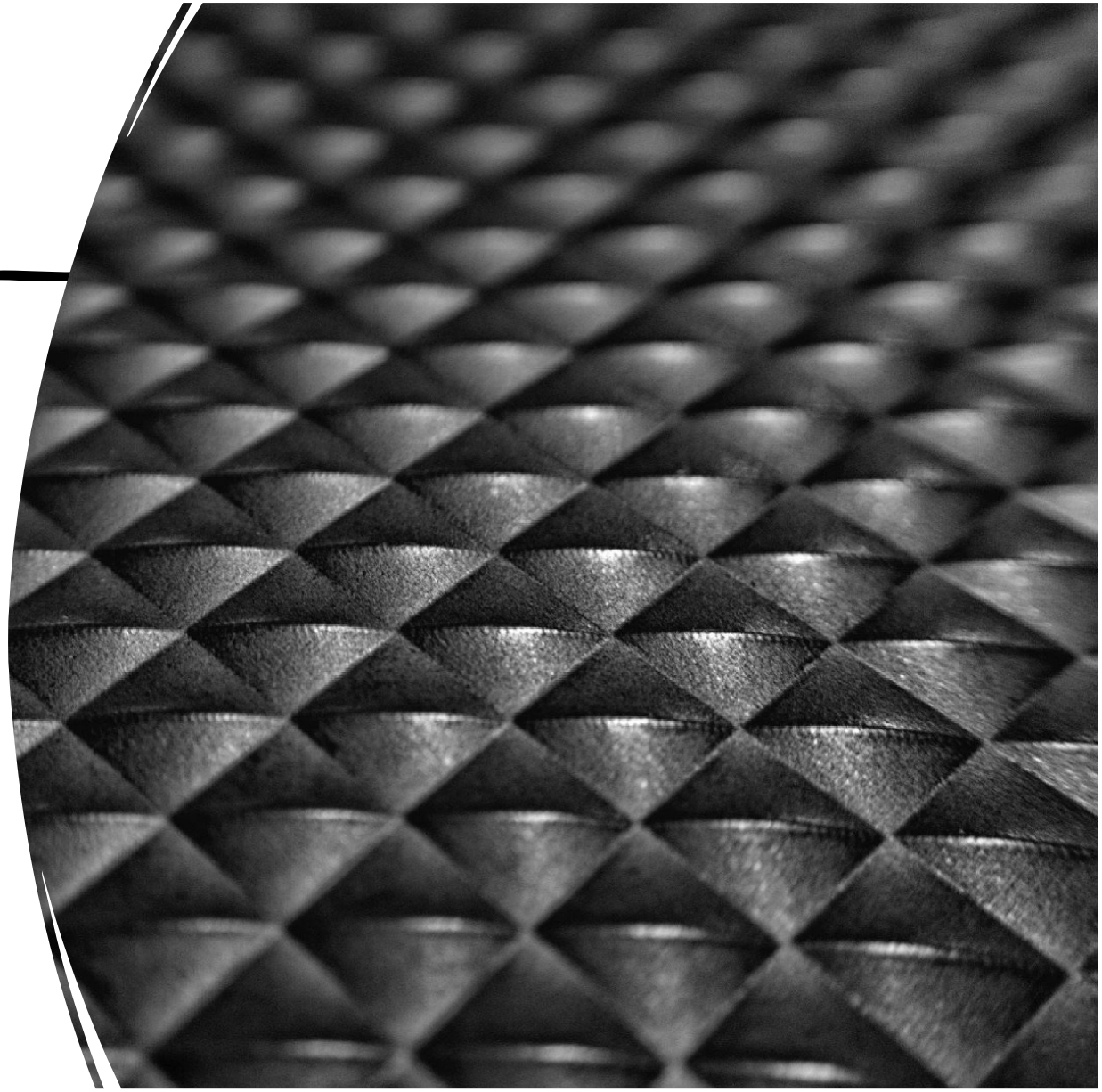
PRO E CONTRO NELL'UTILIZZO DI DESIGN PATTERN

- VANTAGGI:

- La possibilità di attingere a **soluzioni valide** va di pari passo con un **risparmio di tempo e costi**. I team di sviluppatori non devono costantemente ricominciare da zero per trovare una soluzione per un nuovo programma in caso di problemi già parzialmente risolti.

- Di solito, i singoli schemi vengono denominati in base a un **vocabolario comune di termini tecnici,** semplificando in questo modo sia la discussione tra sviluppatori, sia la comunicazione con l'utente destinatario della soluzione finale. Si **semplifica** anche la **documentazione** di un software, considerato che possono essere utilizzate componenti già documentate in precedenza. Tutti questi vantaggi aiutano nella fase di manutenzione e ulteriore sviluppo di un programma.

PRO E CONTRO NELL'UTILIZZO DI DESIGN PATTERN



SVANTAGGI:

- L'utilizzo degli schemi progettuali richiede un **ampio bagaglio di conoscenze** pregresse. Inoltre,la disponibilità di design pattern può anche far credere che gli schemi progettuali presenti possano risolvere quasi tutti i problemi. Detto in parole povere: **questo può limitare la creatività** e la curiosità di trovare soluzioni nuove (e migliori).

SOFTWARE PATTERNS HISTORY

- 1987 – Cunningham and Beck used Alexander's ideas to develop a small pattern language for Smalltalk
- 1990 – The Gang of Four (Gamma, Helm, Johnson & Vlissides) begin compiling a catalog of design patterns
- 1991 – First Patterns Workshop at OOPSLA
- 1993 – Kent Beck and Grady Booch sponsor the first meeting of what is now known as the Hillside Group
- 1994 - 1st Pattern Languages of Programs (PLoP) conf.
- 1995 – The Gang of Four (GoF) *Design Patterns book*

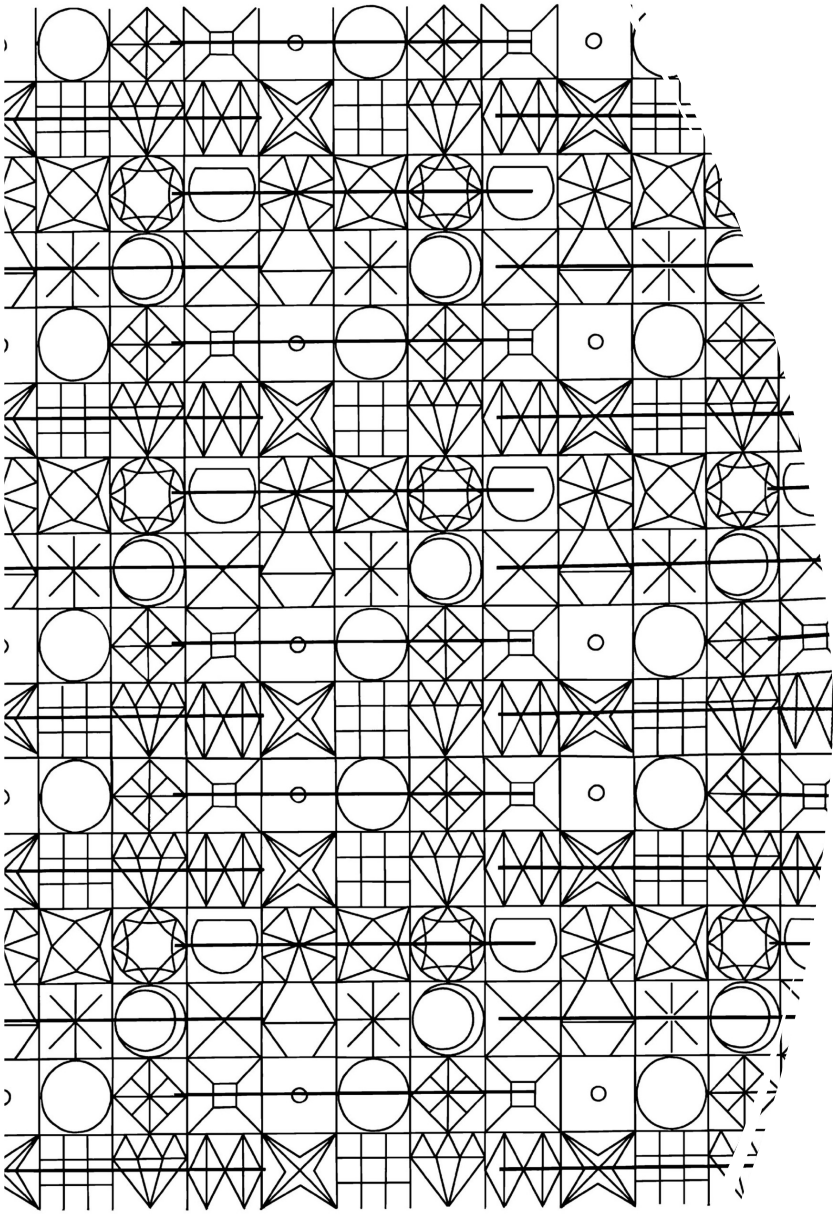
GOF DESIGN PATTERNS

Sono 23 design pattern suddivisi in base al loro scopo

Creazionali:	• propongono soluzioni per creare oggetti
Comportamentali:	• propongono soluzioni per gestire il modo in cui vengono suddivise le responsabilità delle classi e degli oggetti
Strutturali:	• propongono soluzioni per la composizione strutturale di classi e oggetti

MODELLI CREAZIONALI

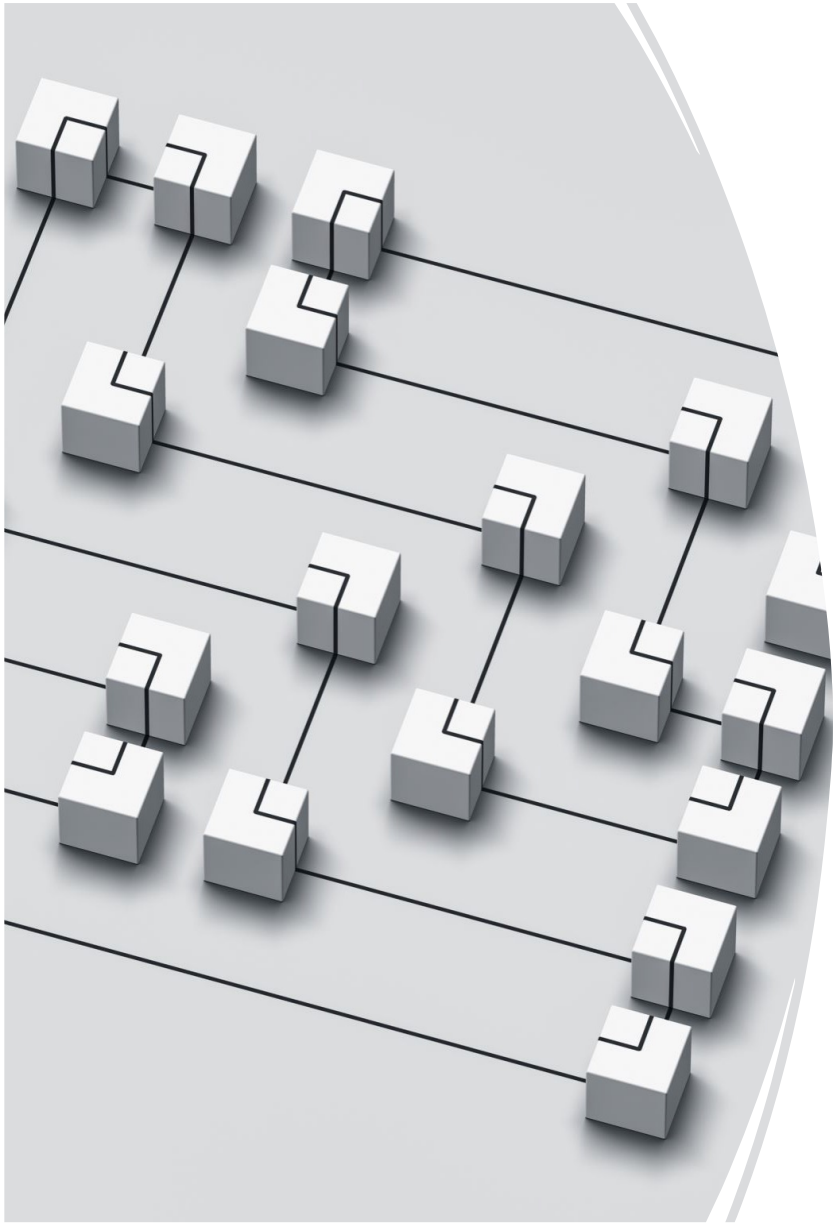
- I **creational patterns** permettono di creare oggetti che rappresentano in modo semplificato istanze precise, indipendentemente dal modo in cui i singoli oggetti sono creati e rappresentati in un software.
- Propongono soluzioni per creare oggetti.
  - **Builder Pattern**: il **costruttore** nella categoria dei pattern creazionali separa lo sviluppo di oggetti (complessi) dalle loro rappresentazioni.
  - **Factory Pattern**: come pattern, il **factory method** crea un oggetto attraverso il richiamo ad un metodo invece che a un costruttore.
  - **Singleton Pattern**: come pattern, il **singleton** fa in modo che per ogni classe esista solo un'unica istanza. Per di più un singleton è disponibile a livello globale.



MODELLI STRUTTURALI

I cosiddetti **structural patterns** sono degli schemi preimpostati per i legami tra le classi. Con essi si mira a un'astrazione che possa comunicare anche con altre soluzioni; (programmazione di interface)

- Propongono soluzioni per la composizione strutturale di classi e oggetti
  - Composite Pattern: un pattern strutturale composto, chiamato in inglese **composite**, rivolto principalmente alle strutture dinamiche, per es. per l'organizzazione o la compressione di dati.
  - Decorator Pattern: il cosiddetto **decorator** integra le classi esistenti di ulteriori funzionalità o responsabilità.
  - Facade Pattern: il modello **facciata** rappresenta un'interfaccia verso altri sistemi, sottosistemi o subsistemi.



MODELLI COMPORTAMENTALI

- Con i **behavioral pattern** si modella il comportamento del software. Questi pattern semplificano processi complessi di comando e controllo. Si può scegliere tra algoritmi e responsabilità di oggetti.
- Propongono soluzioni per gestire il modo in cui vengono suddivise le responsabilità delle classi e degli oggetti.
    - **Observer Pattern**: l'**osservatore** trasmette alle strutture le modifiche apportate a un oggetto, che dipendono dall'oggetto iniziale.
    - **Strategy Pattern**: la **strategia** definisce una famiglia di algoritmi intercambiabili.
    - **Visitor Pattern**: il **visitatore** permette di isolare le operazioni eseguibili, in modo da compiere nuove operazioni senza modificare le classi interessate.

GOF CLASSIFICATION OF DESIGN PATTERNS

Purpose - what a pattern does

- Creational Patterns
  - Concern the process of object creation
  - *Abstract Factory, Builder, Factory Method, Prototype, Singleton.*
- Structural Patterns
  - Deal with the composition of classes and objects
  - *Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy.*
- Behavioral Patterns
  - Deal with the interaction of classes and objects
  - *Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template, Visitor.*

GOF PATTERN TEMPLATE

Pattern Name and Classification

- A good , concise name for the pattern and the pattern's type

Intent

- Short statement about what the pattern does

Also Known As

- Other names for the pattern

Motivation

- A scenario that illustrates where the pattern would be useful

Applicability

- Situations where the pattern can be used

GOF PATTERN TEMPLATE (CONTINUED)

Structure

- A graphical representation of the pattern

Participants

- The classes and objects participating in the pattern

Collaborations

- How to do the participants interact to carry out their responsibilities?

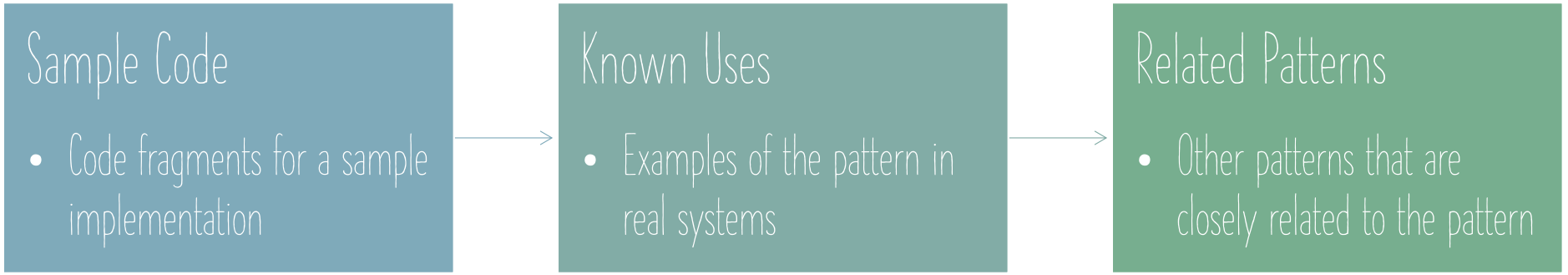
Consequences

- What are the pros and cons of using the pattern?

Implementation

- Hints and techniques for implementing the pattern

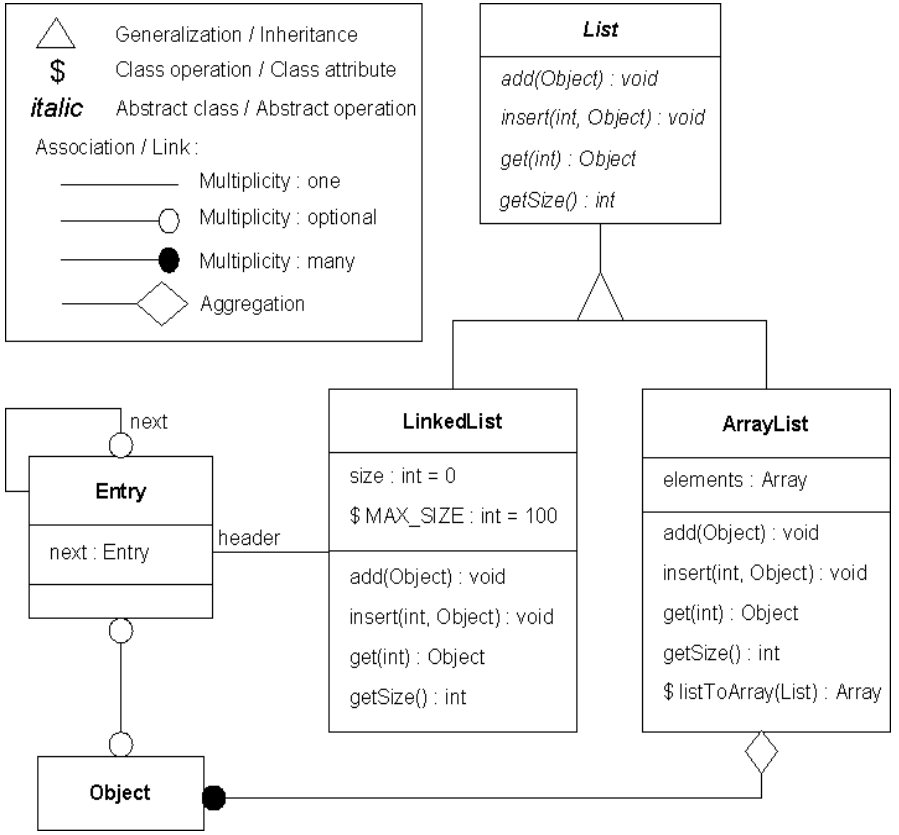
GOF PATTERN TEMPLATE (CONTINUED)



GOF NOTATION

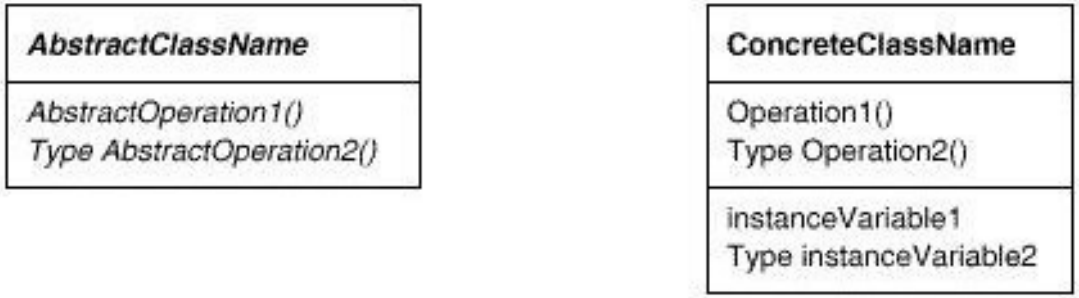
The GoF book uses the Object Modeling Technique (OMT) notation for class and object diagrams

Head first uses UML



Object Modeling Technique (OMT) object model

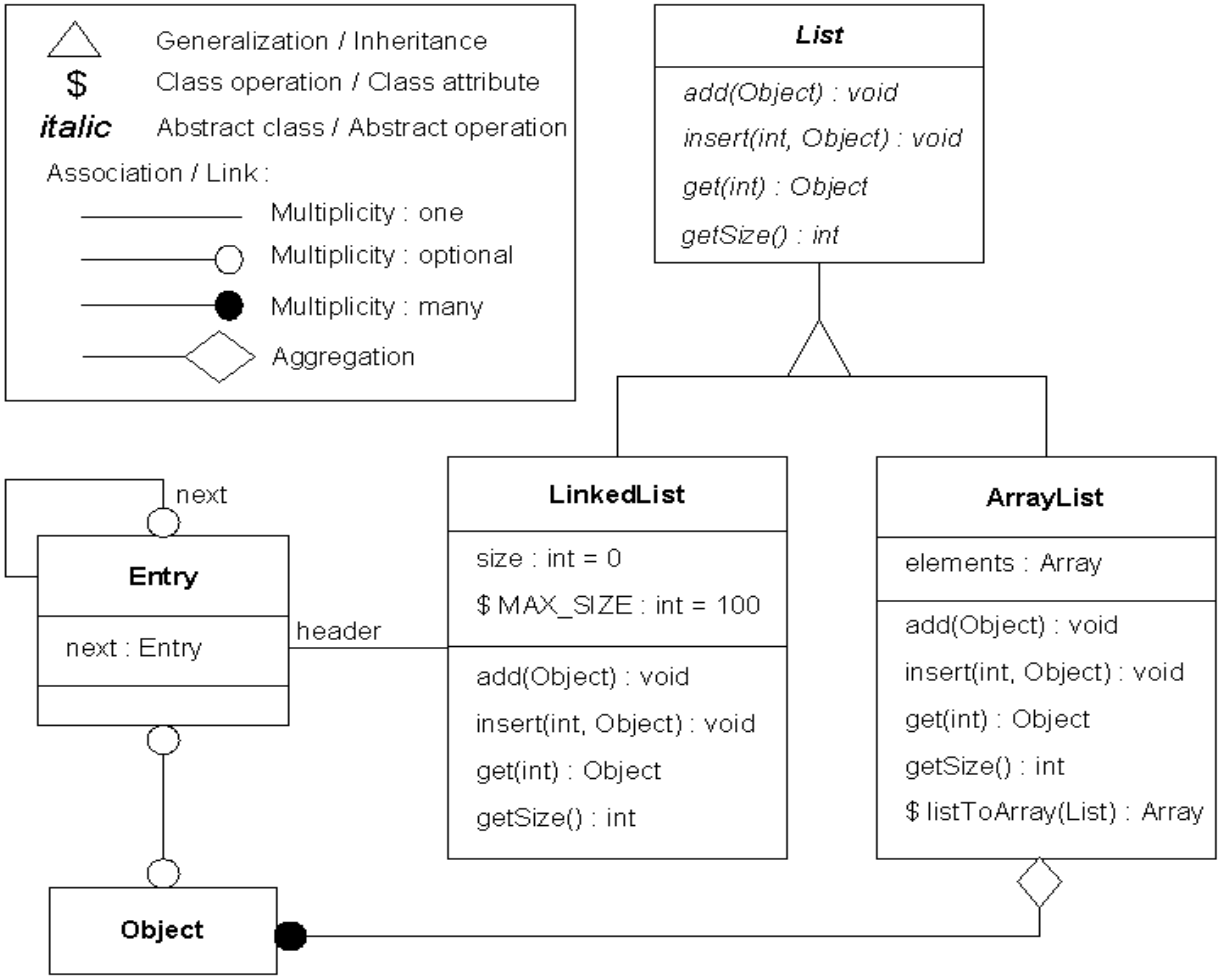
Appendix B of the GoF book.



(a) Abstract and concrete classes

(b) Participant Client class (left) and implicit Client class (right)

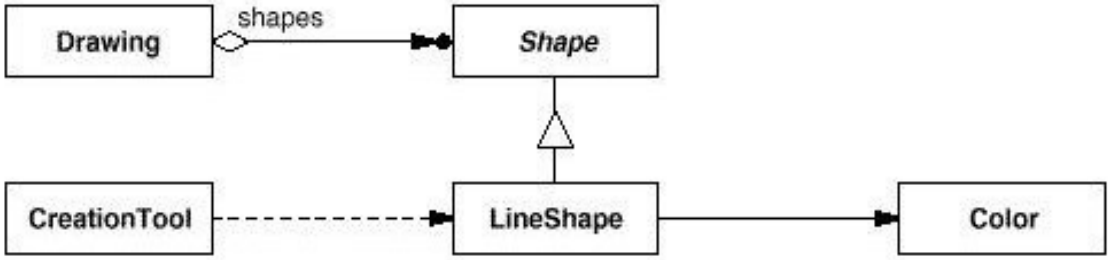
OMT object model (continued)

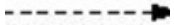


OMT OBJECT MODEL (CONTINUED)

reference (do not use associations) when describing DP

instantiation relatio

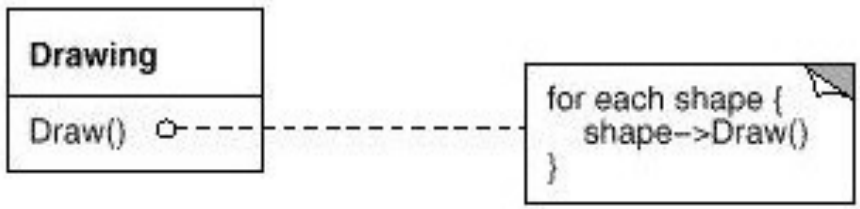




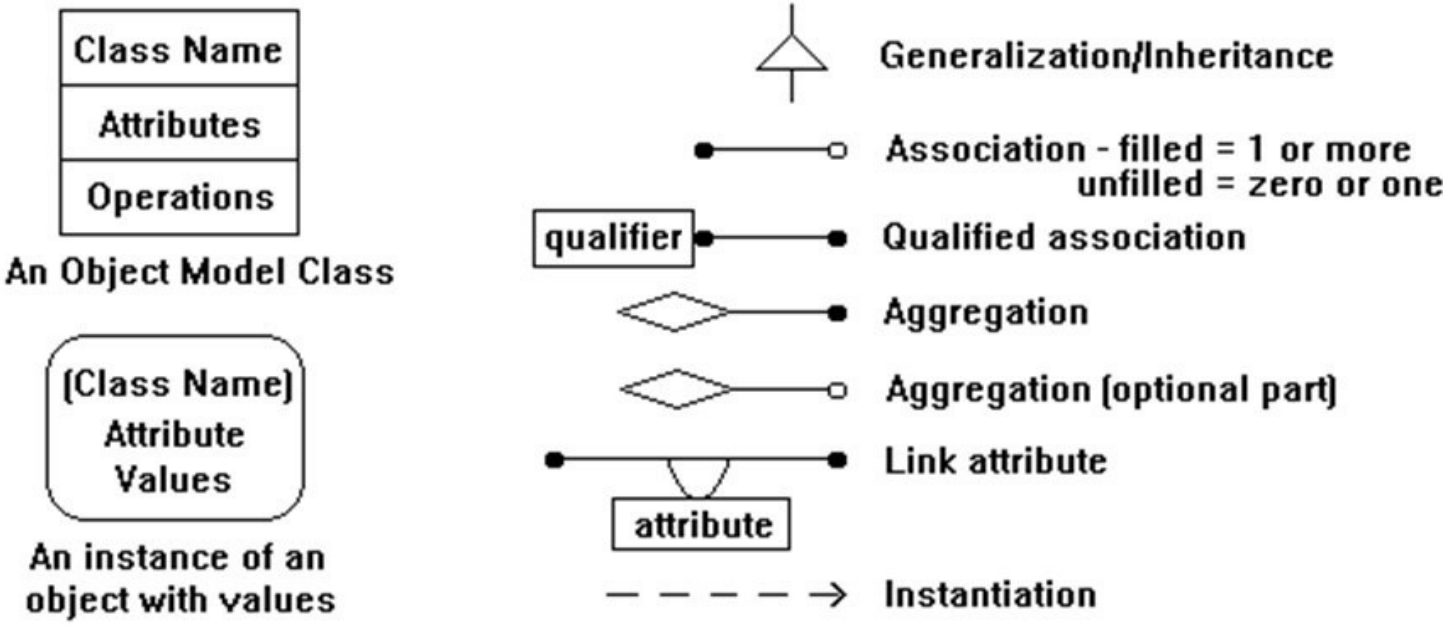
(c) Class relationships

OMT OBJECT MODEL (CONTINUED)

anchor a note



(d) Pseudocode annotation


Class Name
Attributes
Operations
An Object Model Class
(Class Name)
Attribute
Values
An instance of an
object with values
Generalization/Inheritance
Association - filled = 1 or more
unfilled = zero or one
qualifier
Qualified association
Aggregation
Aggregation (optional part)
Link attribute
attribute
Instantiation

CLASSES & INSTANCES (METADATA & DATA)