Diagrammi UML di qualità: alcune linee guida

Dr. Andrea Baruzzo
andrea.baruzzo@dimi.uniud.it
www.dimi.uniud.it/~baruzzo

Sommario

- Cosa significa fare diagrammi UML di qualità?
- Leggibilità e semplicità dei diagrammi
- Diagrammi e layout: pattern di stile
- Diagrammi e colore: esplicitare le intenzioni del progettista
- Per saperne di più…

Cosa significa fare diagrammi UML di qualità?

- Modellare per **documentare**
- Modellare per **progettare**
- **Esigenze diverse**: documentare un prodotto finito, progettare il prodotto da costruire, da modificare, da estendere
- Noi ci concentreremo sulle esigenze di progetto…

Modelli per ragionare

- UML è uno strumento per **ragionare**, non solo per disegnare diagrammi
- Ragionare su cosa? su problema e soluzione
- diagrammi di qualità si costruiscono, non si "ottengono" (*reverse engineering*? Mah …)
- UML dovrebbe servire per esplicitare le **intenzioni del progettista** che giustificano il codice

Utilità di un modello

- Non esistono modelli “giusti” o “sbagliati”, esistono modelli più o meno utili (*Martin Fowler*)
- L’utilità di un modello non è legata solo alla sua forma sintattica
- Lo **scopo** e il **livello di dettaglio** sono due elementi chiave della modellazione
- Un modello è utile se comunica **informazione** (le macchine processano dati, gli uomini informazione)

Questo diagramma comunica informazione?

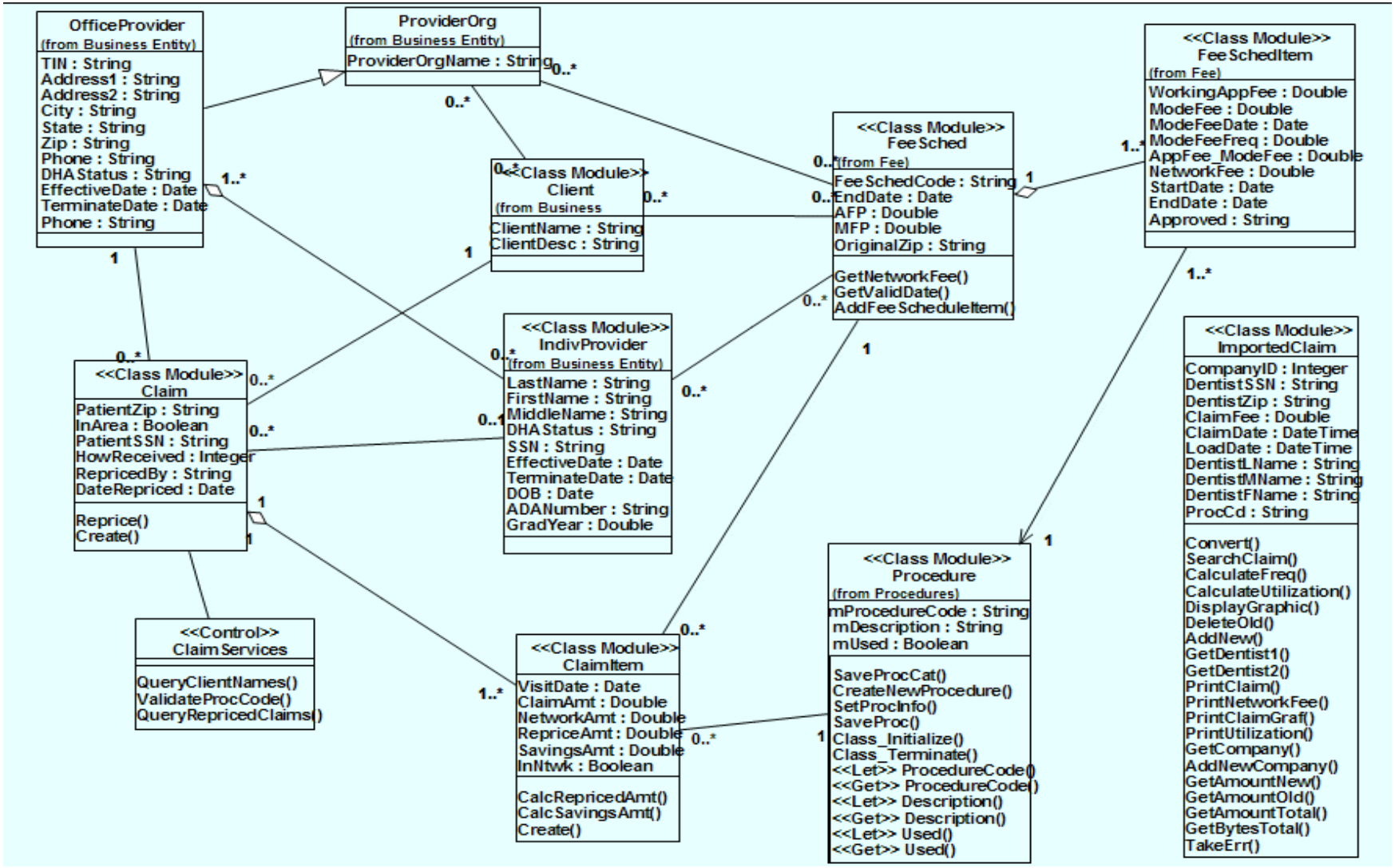


Figura tratta da [Pescio, 2001] “Modelli che parlano: realizzare diagrammi espressivi”

Diagrammi e caos

- Troppo dettaglio!
- Layout che non facilita la lettura (linee incrociate, linee verso l'alto e linee verso il basso), classi "isola"
- Tutte le classi sembrano importanti allo stesso modo, non emergono ruoli
- Ma cosa vuol dire allora "comunicare informazione"?

“Modelli che parlano” (Carlo Pescio)

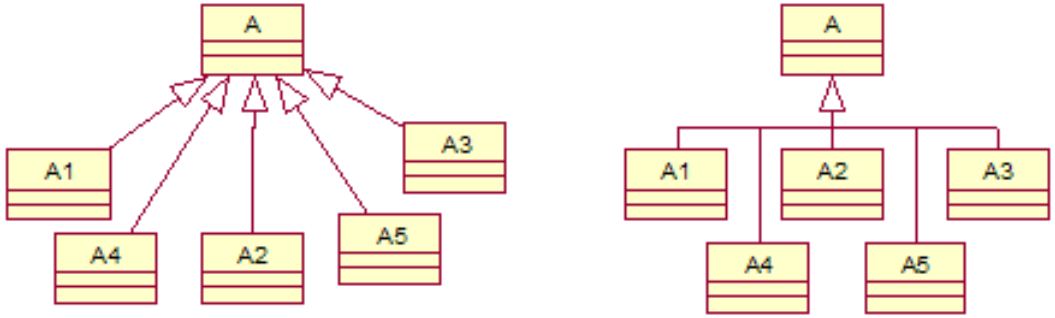
- Quali informazioni interessano ad un progettista?
- Conoscere i punti di **estensione**
- Percepire il grado di **riuso** delle classi di una libreria di terze parti
- Percepire il grado di riuso delle classi di un sottosistema nell’intero progetto
- Ragionare sulle **dipendenze** di un componente
- Identificare i **pattern di comunicazione** che hanno ispirato la gestione del controllo e delle collaborazioni

Sommario

- Cosa significa fare diagrammi UML di qualità?
- Leggibilità e semplicità dei diagrammi
- Diagrammi e layout: pattern di stile
- Diagrammi e colore: esplicitare le intenzioni del progettista
- Per saperne di più…

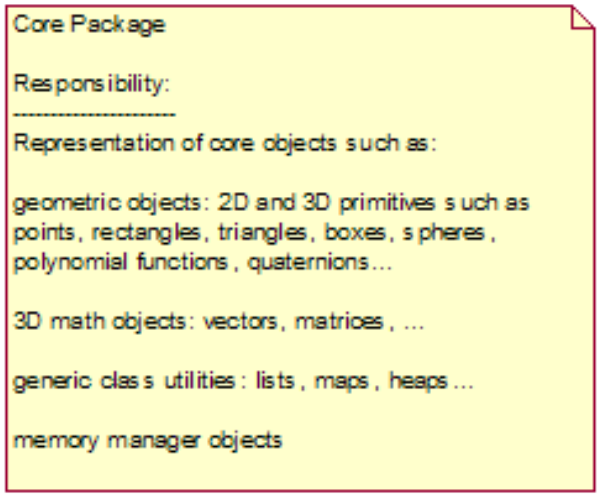
Linee guida di leggibilità 1/2

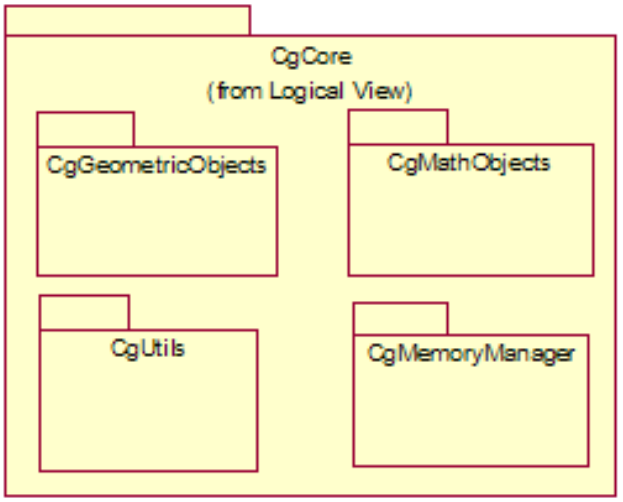
- Organizzare i diagrammi da sinistra a destra, dall'alto verso il basso (nelle culture occidentali …)
- Evitare linee incrociate o sovrapposte
- Dimensionare i simboli in base alla loro importanza
- Preferire le linee rette a quelle curve o inclinate



Linee guida di leggibilità 2/2

- Corredare ogni diagramma con una legenda (mission, autore, progetto, data ultima modifica, …)
- Utilizzare font leggibili sia a video, sia sulle versioni stampate dei diagrammi
- Corredare ogni classe e ogni package con una nota descrittiva o una descrizione di documentazione





Linee guida di semplicità dei modelli

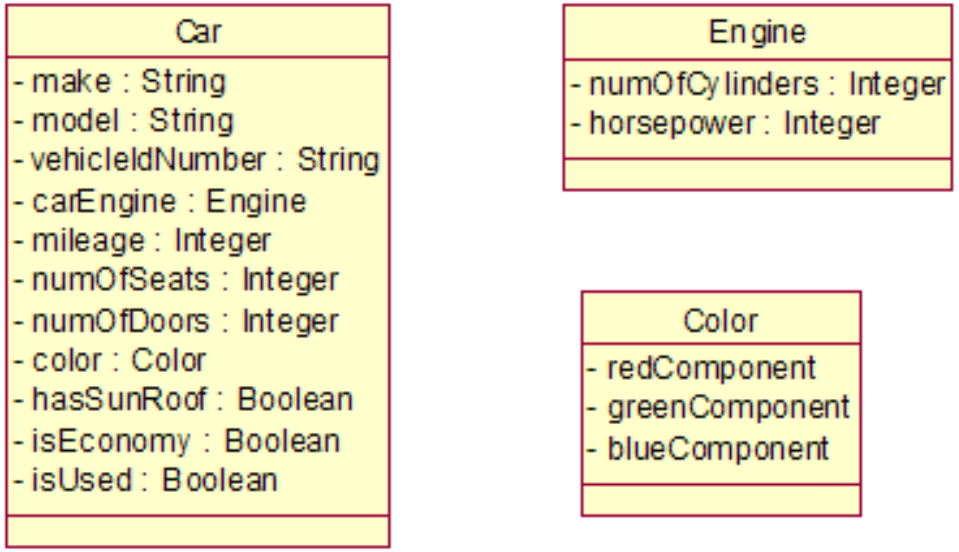
- Rappresentare solo i **concetti essenziali** in base alla mission del diagramma
- Non abusare di simboli esoterici oppure obsoleti che possono generare confusione (stereotipi non standard, uses e includes…)
- Scomporre i diagrammi troppo grandi per **ridurre la complessità**
- Preferire **diagrammi stampabili** in un foglio A4
- Focalizzare **l'attenzione sul contenuto** prima che sull'estetica

Sommario

- Cosa significa fare diagrammi UML di qualità?
- Leggibilità e semplicità dei diagrammi
- Diagrammi e layout: pattern di stile
- Diagrammi e colore: esplicitare le intenzioni del progettista
- Per saperne di più …

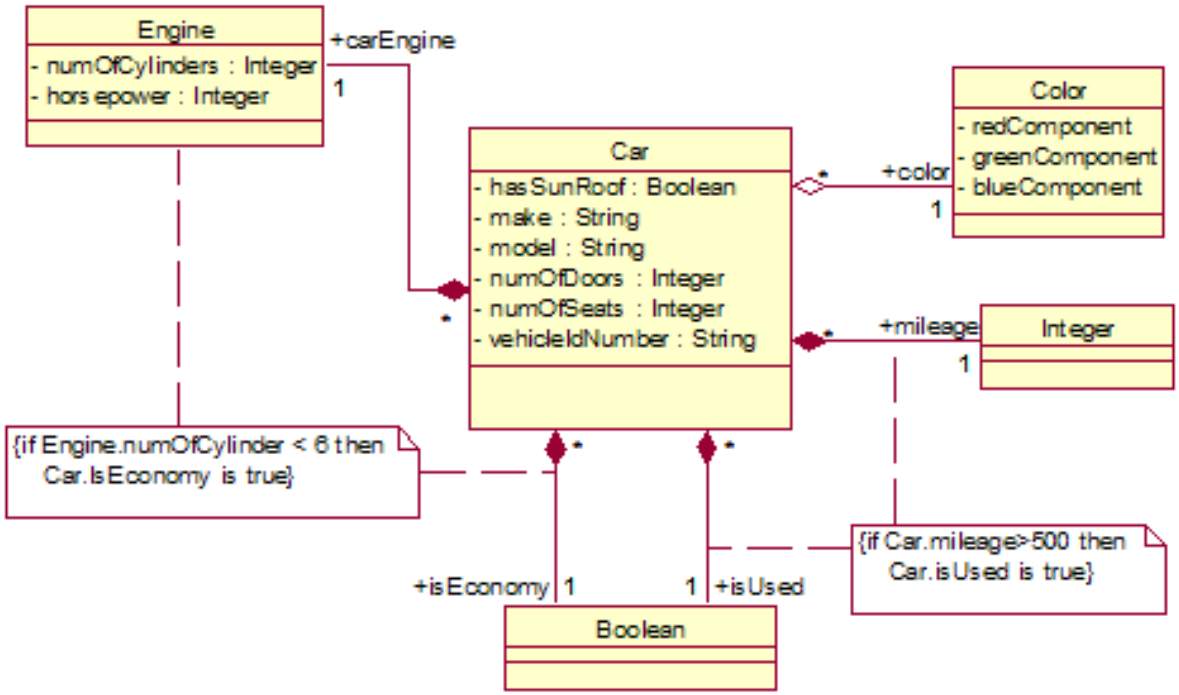
“Attributes As Composition To Types”

- Esempio di classe (*Car*) con un eccessivo dettaglio dovuto alla presenza di molti attributi



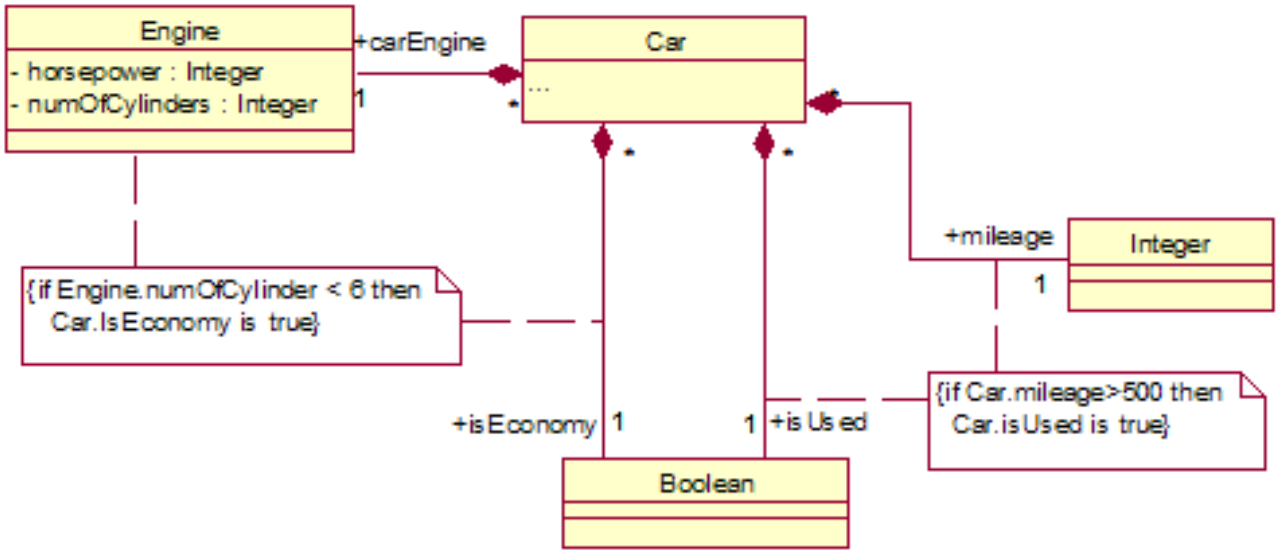
“Attributes As Composition To Types”

- spostare al di fuori di una classe gli attributi che rappresentano relazioni di contenimento con altre classi del sistema (o di libreria), ad esempio per esplicitarne il ruolo oppure un vincolo logico.



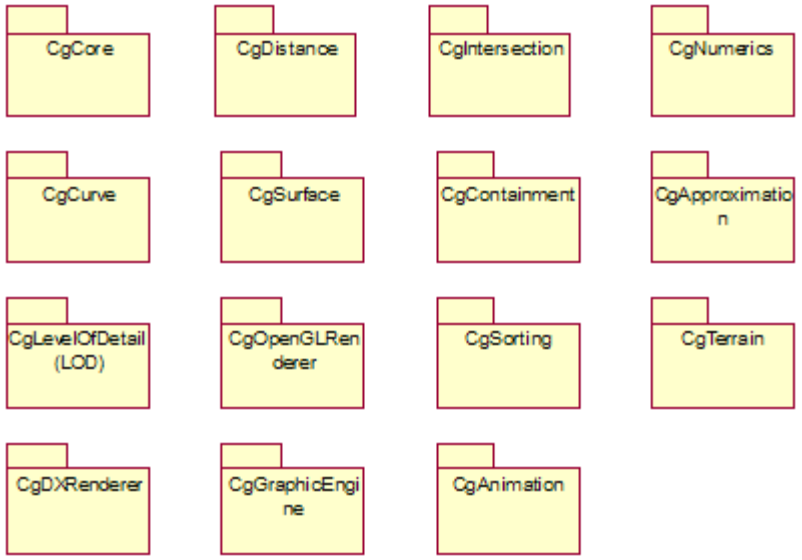
“Explicit Elision”

- omettere deliberatamente ogni elemento irrilevante rispetto alla missione e alla prospettiva del diagramma (analisi vs. design vs. implementazione)



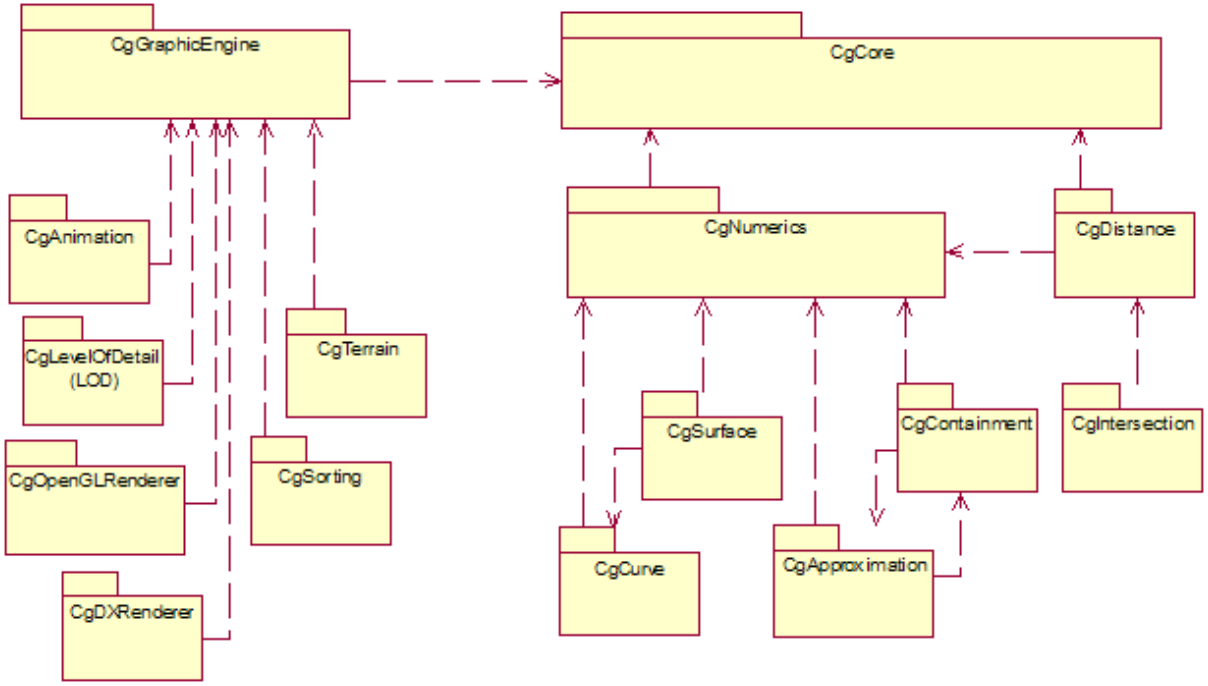
“Tombstone Packages”

- Diagramma di package poco informativo a causa dell'elisione delle relazioni tra i package
- Tale esempio rappresenta fedelmente la complessità del software di produzione in ambiti industriali e la necessità di diagrammi più espressivi.



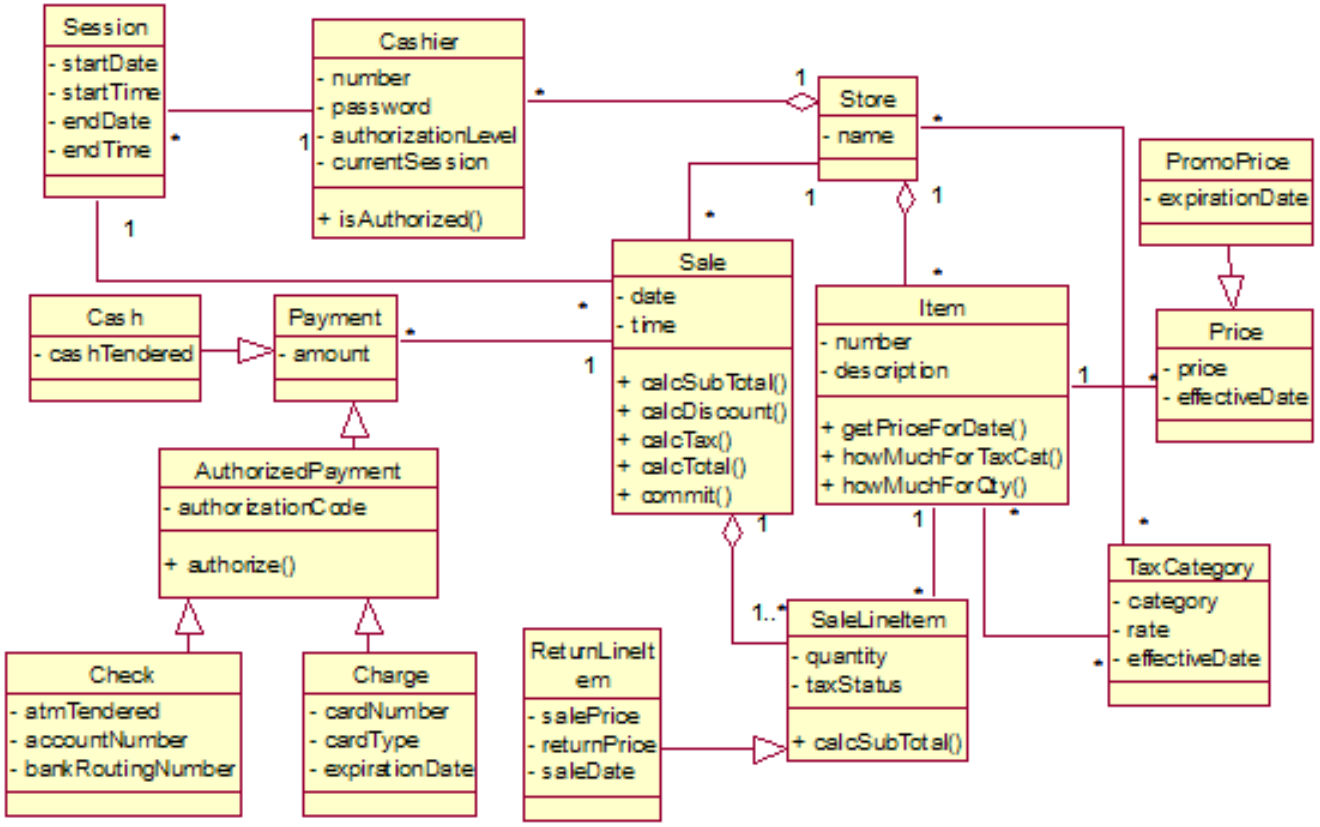
“Tombstone Packages”

- esplicitare le relazioni tra i package, in particolare **dipendenze** e i **livelli d’astrazione**



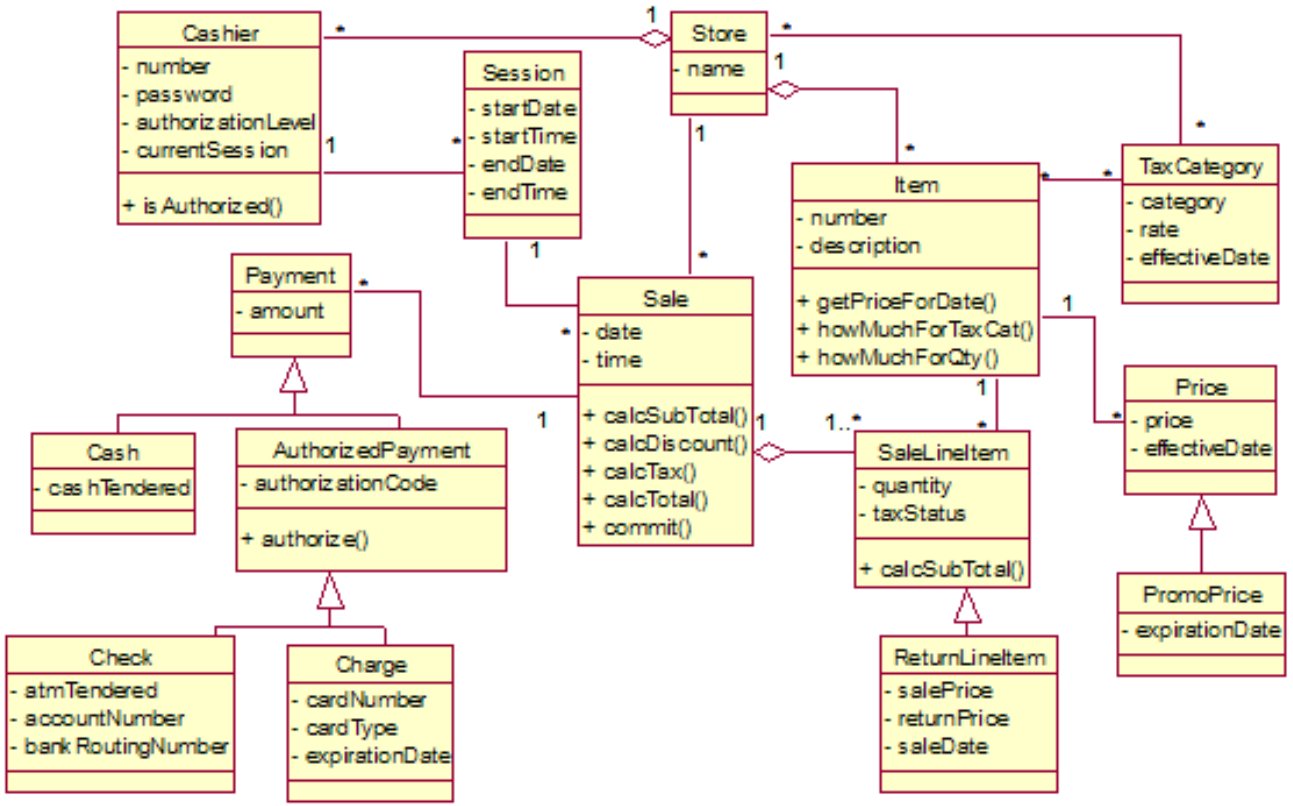
“Inheritance Goes Up”

- Il diagramma non rispetta lo standard (“verso l’alto”) di rappresentazione per le gerarchie IS-A



“Inheritance Goes Up”

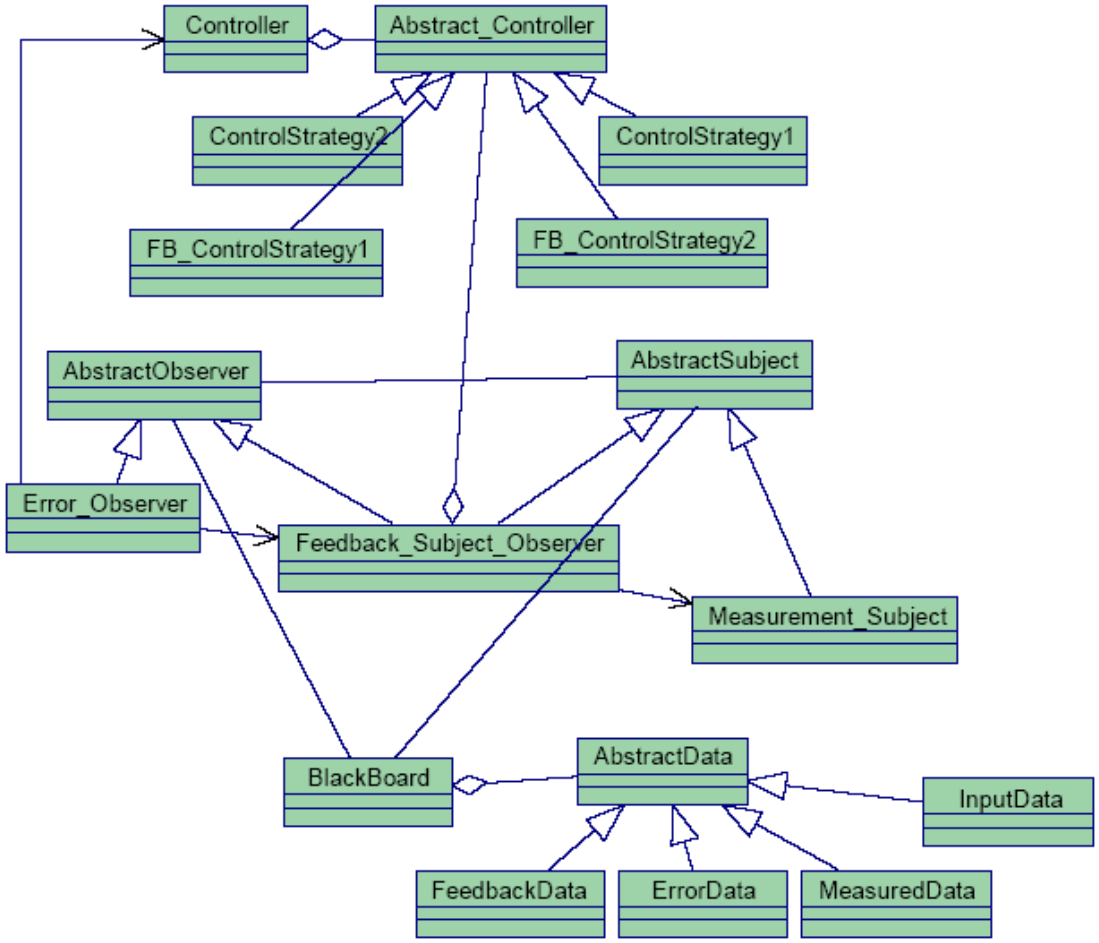
- Se lo riportiamo allo standard (concetti astratti più in alto di quelli concreti), è più facile localizzare le gerarchie d’ereditarietà



“Dependency Goes Up” (Carlo Pescio)

- Osservazione: gli **elementi concreti** sono **meno riusabili**
- Idea: posizionare **elementi generali in alto**, quelli più concreti in basso (le dipendenze vanno sempre dal concreto al generale)
- Obiettivi:
  - Facilitare il **ragionamento sulle dipendenze**
  - **Rendere immediato ciò che è importante**
  - **Rendere esplicito ciò che è essenziale**
- Non sempre i diagrammi cui siamo abituati rispettano queste importanti proprietà
- Layout e complessità: tra ordine e caos
- Vediamo un diagramma che nasconde aspetti importanti legati alle dipendenze

“Dependency Goes Up”(Carlo Pescio)

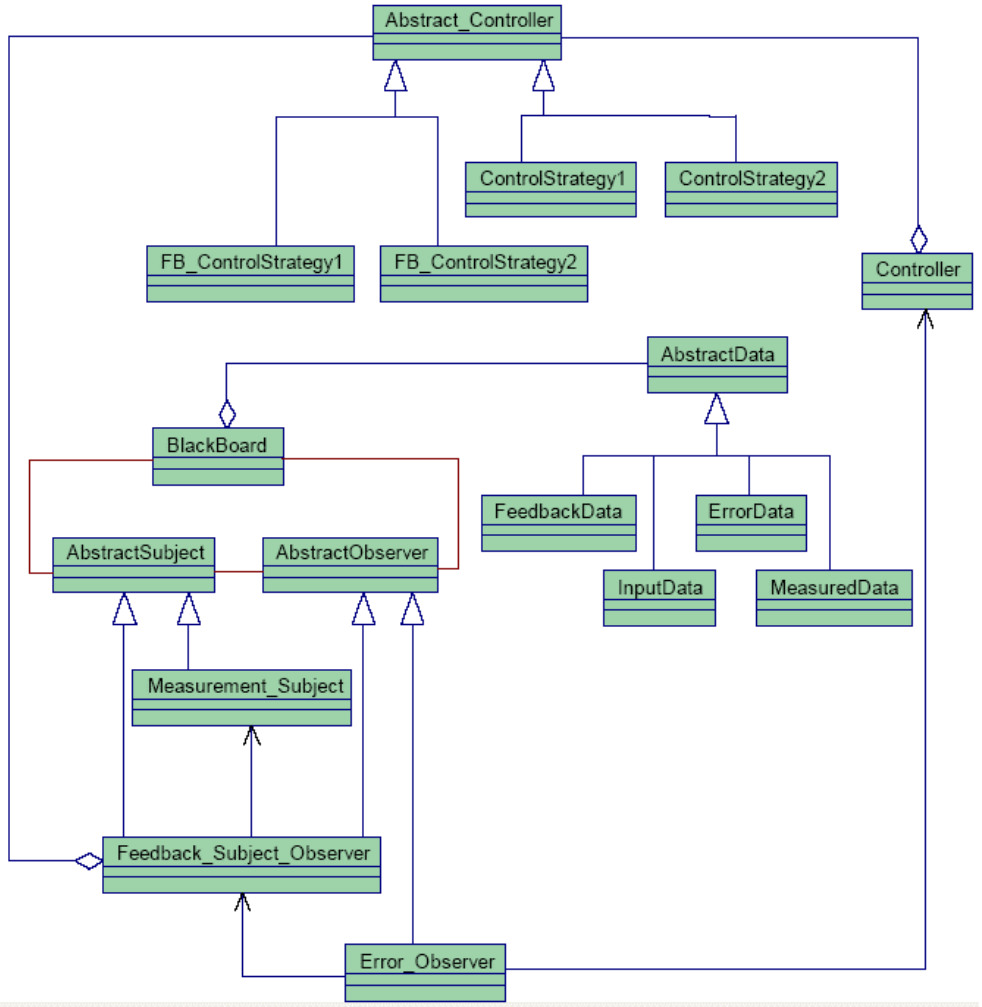


“Dependency Goes Up”(Carlo Pescio)

- Un aspetto importante del design sono le **dipendenze circolari** (limitano il riuso, aumentano l’accoppiamento, rendono difficile il test …)
- So individuare a colpo d’occhio se ci sono dipendenze circolari nel diagramma precedente?
- So dire quante ce ne sono?
- Niente da fare, c’è troppo caos, **devo proprio studiare il diagramma** nei dettagli …
- Proviamo con la linea guida “tutte verso l’alto”

“Dependency Goes Up”(Carlo Pescio)

- Diagramma più leggibile
- Una sola dipendenza circolare, in rosso
- Riesco a ragionare più velocemente sulle dipendenze
- E’ stratificato (layering)
- Supporta meglio l’attività di design



Sommario

- Cosa significa fare diagrammi UML di qualità?
- Leggibilità e semplicità dei diagrammi
- Diagrammi e layout: pattern di stile
- Diagrammi e colore: esplicitare le intenzioni del progettista
- Per saperne di più …

Modellazione e colore: gli esordi

- Diagrammi e colore: **modelli più espressivi** con uno sforzo quasi trascurabile
- Primi tentativi da parte di Peter Coad
- Obiettivo: capire a colpo d'occhio il **ruolo** di una classe all'interno di un diagramma
- … bisogna allora pensare a quali ruoli possono risultare interessanti per un progettista. Secondo voi?

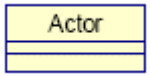
Ruoli delle classi nei modelli di analisi

- Analisi e design esaltano ruoli diversi
- Coad si concentra sui modelli di analisi
- Quattro ruoli principali:
  - **Actor**, classe principale, ruolo attivo (giallo)
  - **Moment**, classe tempo-evento (rosa)
  - **Thing**, classe secondaria (verde)
  - **Description**, classe descrittiva, passiva (blu)
- Colori tenui!

Un set di colori esteso per il design

- Carlo Pescio parte dai colori di Coad e li estende per il design
- Aggiunge altri tre ruoli:
  - **HotSpot**, punto di estensione (arancione)
  - **Library**, classe di libreria di terze parti (grigio)
  - **Elsewhere**, classe definita “altrove” (bronzo)
- Strana classe l’ultima … secondo voi a che cosa serve?

Classi attore (giallo)

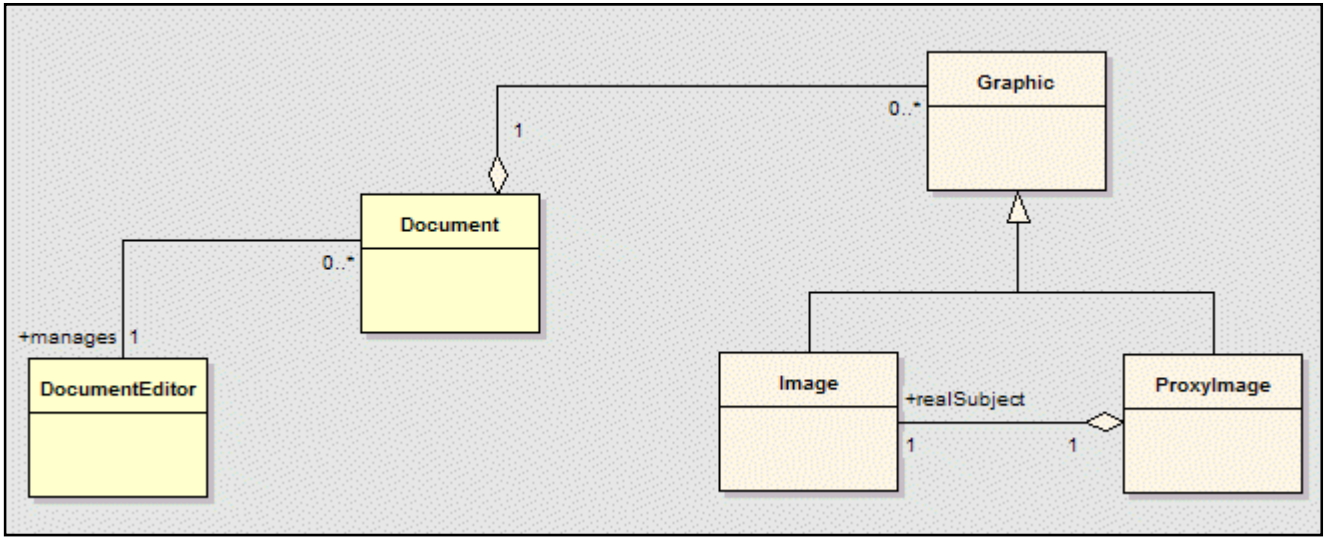


- Classe attore (o ***classi attive***, secondo una prospettiva più orientata verso il design)

- Esistono diversi tipi di classi attive in un diagramma. Le più comuni distinguono:
  - **elementi "intelligenti"** del sistema
  - **elementi reattivi** (elementi che reagiscono, rispondono a degli stimoli provenienti dall'esterno del sistema o del diagramma in questione)

Esempio di classi attive

- Assumo che le classi *Document* e *DocumentEditor* assumano un ruolo centrale in questo design.

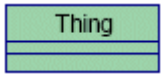


Classi "tempo-evento" (rosa) e secondarie (verdi)

- Rosa, classe Temporale

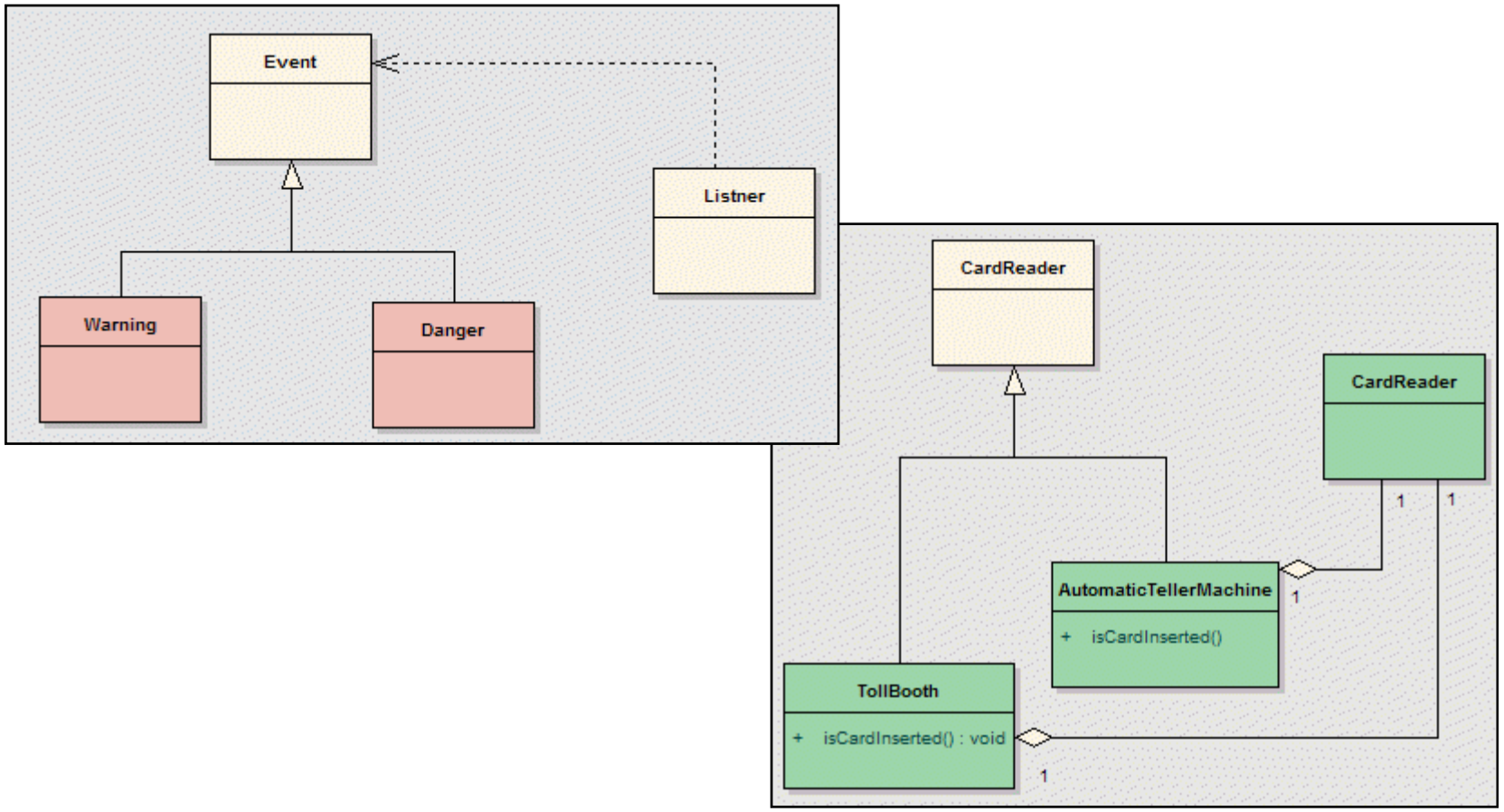
  - Identifica le ***classi "tempo-evento"*: ** classi caratterizzate da messaggi asincroni, eccezioni o gestione di eventi di sistema

- Verde, classe secondaria



  - Identifica ***classi di infrastruttura*** o ***classi secondarie*** (contenitori come liste, array, hash table; stream, file, …), purché non si tratti di classi di libreria (c'è un altro colore per questo)

Esempio di classi Tempo-Evento e classi secondarie



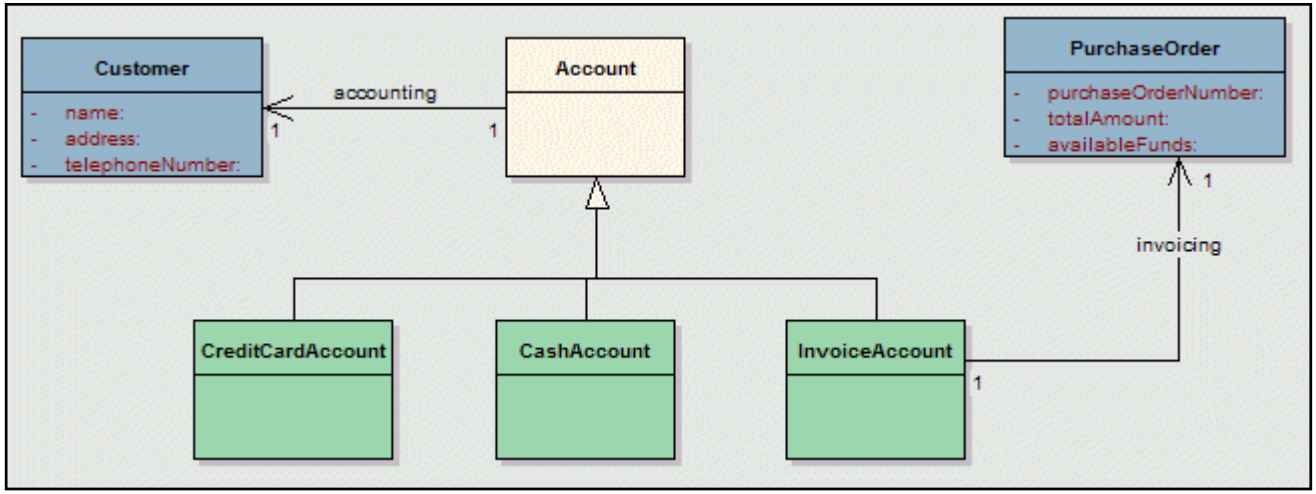
Classi passive (blu)

- Indica ***classi di descrizione*** o *classi passive*.
  - Esempi di tali classi sono aggregati tipi di dati semplici come date, profili utenti

- Si tratta di strutture che sono **prive di "intelligenza"**, ossia non hanno comportamento "interessante"
  - Spesso forniscono solo metodi accessori (get/set): poca sostanza, poca intelligenza: questo è in generale deprecabile, ma è un altro discorso!

Esempio di classi passive

- Se *Customer* assumerà altre responsabilità che lo renderanno più attivo, passerò al verde o al giallo.



Classi punti di estensione (arancione) e di libreria (grigio)

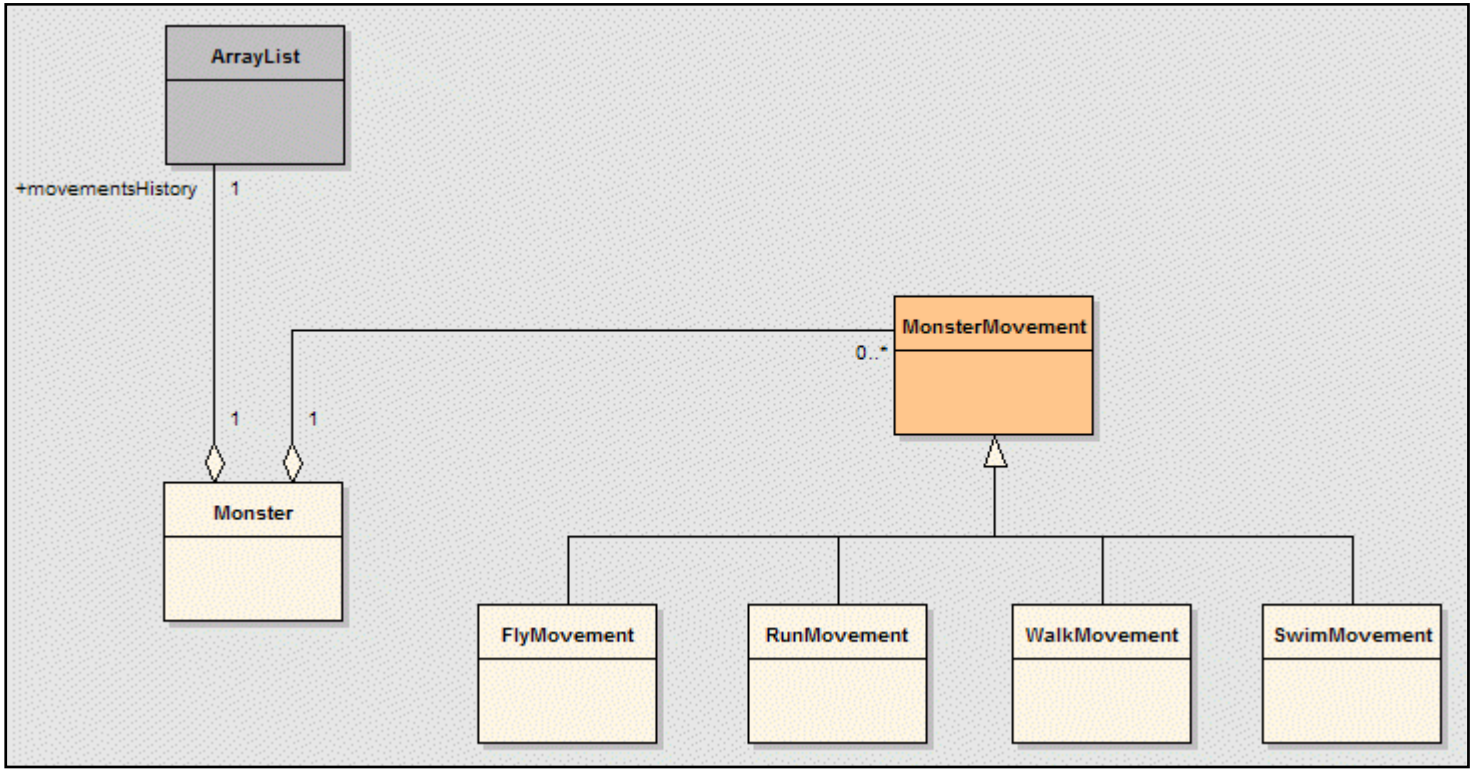
- Arancione, classi punti di estensione

  - Identifica *classi "Hot-Point"*, punti di estendibilità.
  - Esempi sono le classi base astratte (interfacce), il cui ruolo consiste nel permettere l'estensione del sistema mediante ereditarietà

- Grigio, classi di libreria

  - Identifica classi di terze parti, non sottoposte a manutenzione interna, fornendo quindi una percezione di riuso (esterno).

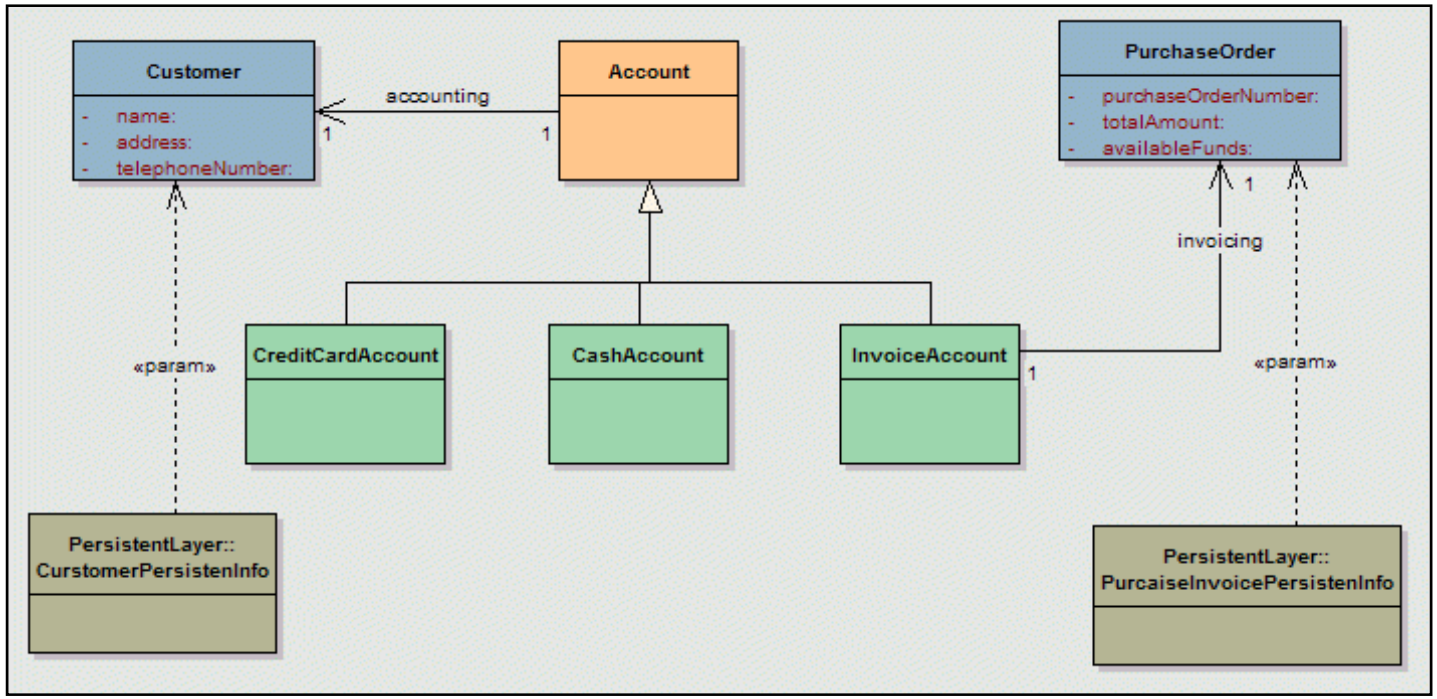
Esempio di estensione e riuso esterno



Classi “Elsewhere” (bronzo)

- Identifica ***classi “elsewhere”***, classi che appartengono al sistema (cioè classi sono soggette a manutenzione), ma la cui definizione si trova in un altro diagramma (package)
- Le classi hanno una rilevanza diversa a seconda del diagramma in cui compaiono perché svolgono ruoli diversi
- Qualora una classe definita in un altro diagramma svolge un ruolo secondario nel diagramma corrente, essa può essere colorata di bronzo, fornendo così una percezione del riuso (interno)

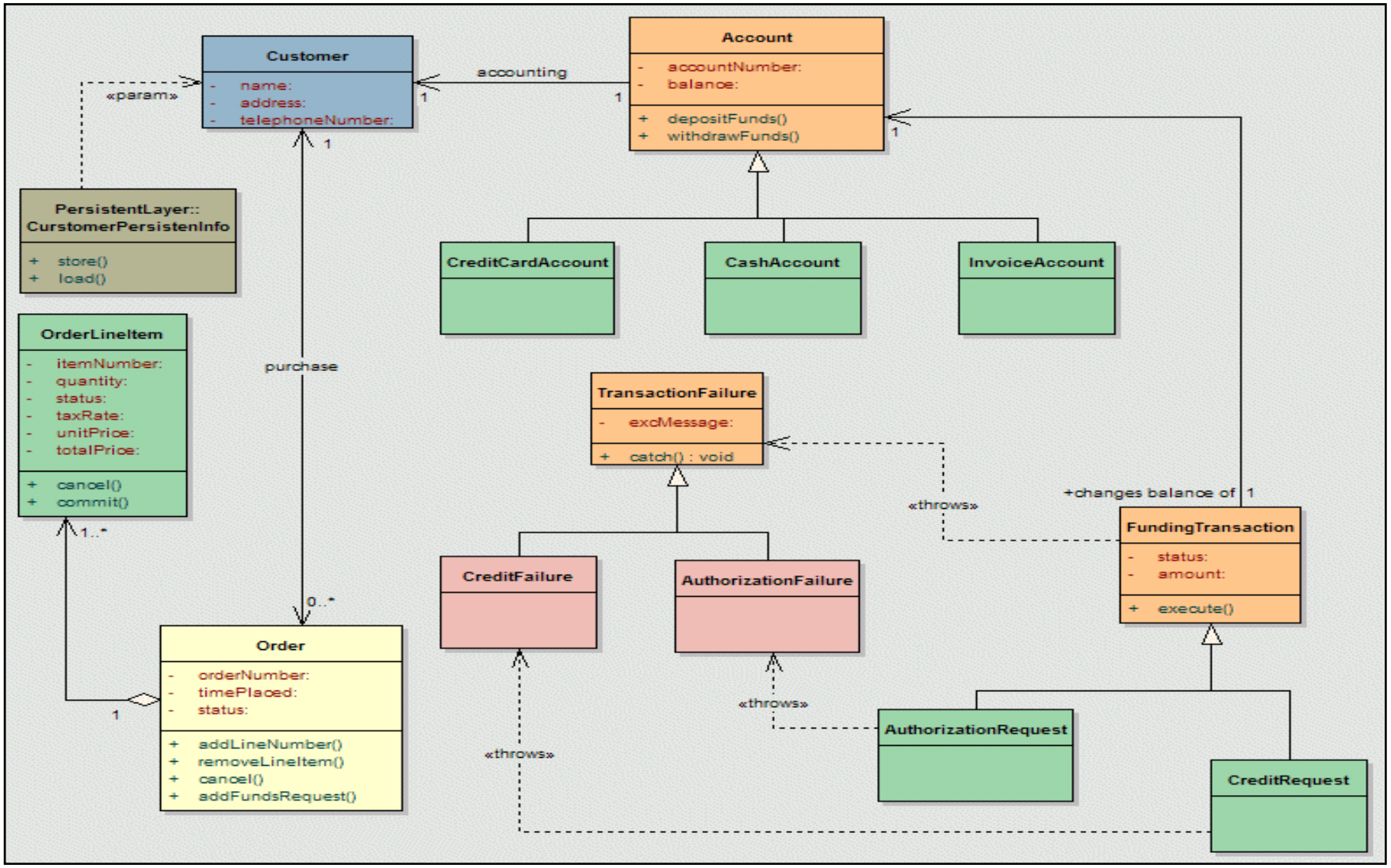
Esempio di classi “Elsewhere”



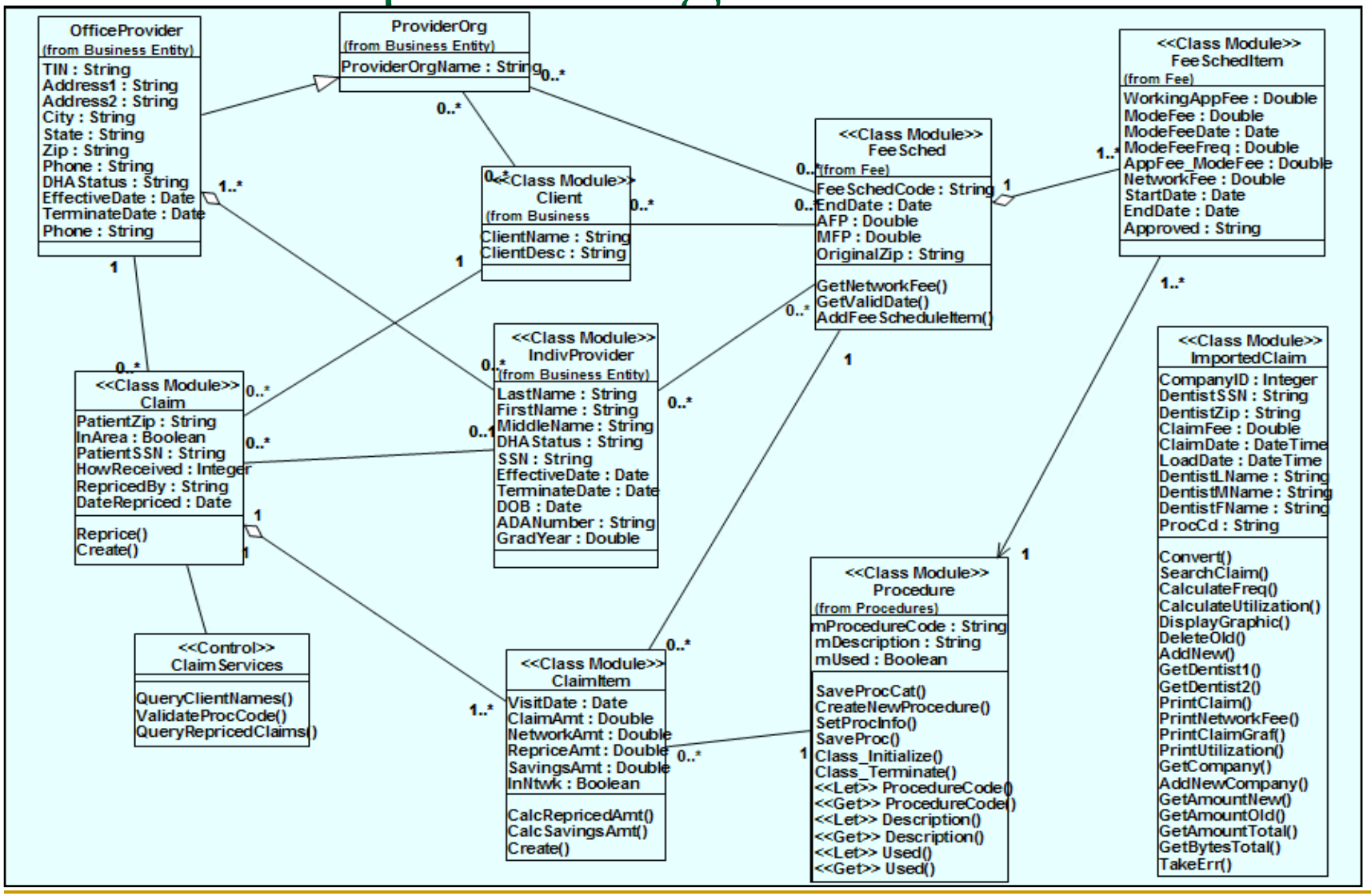
Riassunto dei colori di Pescio

- I colori ci danno una percezione di:
  - **Riuso esterno** (libreria)
  - **Riuso interno** (“elsewhere”)
  - **Estendibilità** (hot point)
  - **Distribuzione dell’intelligenza** (ruoli attivi, secondari e passivi)
  - **Layering** (livelli di astrazione)
  - **Dinamica**, anche se un po’ implicitamente (eccezioni, eventi, relativa gestione)

Un diagramma espressivo



Un altro tipo di diagramma …



In conclusione …

- Confrontando questi due diagrammi, quale secondo voi:
  - È più espressivo?
  - Descrive le cose essenziali?
  - Non nasconde gli aspetti importanti?
  - Fornisce un'idea dei ruoli delle classi?
  - E' più utile per ragionare?
  - E' più resiliente alle modifiche del codice (problema dell'allineamento tra modello e codice)?

Ci sono metriche sui colori?

- Beh, in generale no (modellazione vs. aspetti percettivi, soggettività, …)
- Però qualche linea guida esiste
  - Giallo come punto di partenza per leggere il diagramma
  - Tanto verde, più o meno grigio, poco bronzo
  - Poco blu (se il design è object-oriented)
  - Arancione qua e là (se il design è flessibile)
  - Rosa qua e là (se il design è reattivo)

Sommario

- Cosa significa fare diagrammi UML di qualità?
- Leggibilità e semplicità dei diagrammi
- Diagrammi e layout: pattern di stile
- Diagrammi e colore: esplicitare le intenzioni del progettista
- Per saperne di più …

Per saperne di più …

- Il miglior modo di imparare è fare!
- La tesi può essere un'opportunità
- Alcuni argomenti sui quali posso seguirvi assieme al prof. Mizzaro e/o al prof Tasso:
  - Pattern di modellazione in UML
  - Modellazione di sistemi embedded – real-time
  - Modelli UML e testing object-oriented
  - Modellazione di applicazioni "AI-based"

Per saperne di più …

- Linee guida di stile
  - Ambler, Scott W. – "*The Elements of UML Style*", Cambridge University Press, 2003
  - Evitts, Paul – "*A UML Pattern Language*", Macmillan Technical Publishing, 2000
  - Pescio, Carlo – "*UML: Manuale di Stile*", Eptacom, 2000
  - Baruzzo, Andrea; Carlo, Pescio – "*Diagrammi, Layout e Gestione della Complessità*", Computer Programming No 136, giugno 2004, Edizioni Infomedia

Per saperne di più …

- Linee guida di stile (continua …)
  - Baruzzo, Andrea – "*Modelli, Macchine e Progettazione*", Computer Programming No 132, Febbraio 2004, Edizioni Infomedia
  - Baruzzo, Andrea – "*Modelli UML: Pattern di stile parte I*", Computer Programming No 133, Marzo 2004, Edizioni Infomedia
  - Baruzzo, Andrea – "*Modelli UML: Pattern di stile parte II*", Computer Programming No 134, Aprile 2004, Edizioni Infomedia
  - Pescio, Carlo – "*Modelli che Parlano: realizzare diagrammi espressivi*", Eptacom, 2001

Per saperne di più …

- Modelli e colore
  - Coad, Peter – "*Show Your Colors*", The Coad Letter No 44, Sept. 1997
  - Coad, Peter – "*Component Models in Color*", The Coad Letter No 51, July 1998
  - Coad, Peter; Lefebvre, Eric; De Luca, Jeff – "*Java Modeling in Color with UML: Enterprise Components and Process*", Prentice Hall, 1999
  - Pescio, Carlo – "*UML: Manuale di Stile*", Eptacom, 2000
  - Baruzzo, Andrea; Pescio, Carlo – "*Progettare con UML e il colore: facciamo parlare la struttura*", Computer Programming No 145, Aprile 2005