Processo di Sviluppo

- ✓ Requisiti del cliente
- ✓ Analisi del Dominio
- ✓ Analisi dei Rischi (Parziale)
- ✓ Analisi dei Requisiti
- ✓ Casi d'uso
- ✓ Scenari
- ✓ Diagramma statico delle classi
- Design

Design

- La fase di design prevede la specifica completa del sistema
- Si deve tenere conto delle tecnologie utilizzate
  - Framework e Linguaggio di programmazione
    - Nel nostro caso .Net e C#
  - Tipo di Applicazione:
    - *Standalone*
    - *Web*
    - ...
  - Strategie di persistenza
    - Database relazionale
    - Xml
    - Serializzazione binaria
    - …

Design – Da Fare…

Rendere il diagramma di analisi "implementabile"

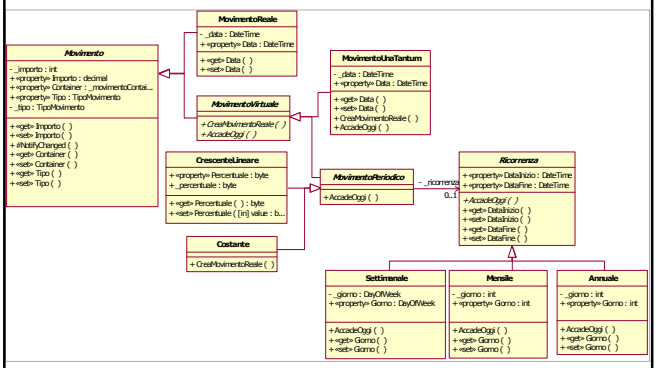
- Aggiungere le classi collezione
- Aggiungere le operazioni necessarie
- Reificare associazioni (in attributi) e classi associazione (in classi)

Per le classi collezione, generalmente si sottintendono le operazioni Add, Remove, Clear, una o più proprietà Indexer, Enumeratori, ecc.. È bene che altre operazioni "non ovvie" siano elencate.

Design – Movimenti$_1$

- In generale: dettagliati metodi e proprietà
- Specificati due tipi di movimenti periodici (*CrescenteLineare*, *Costante*)
- Da notare: l'implementazione della ricorrenza nei movimenti virtuali periodici

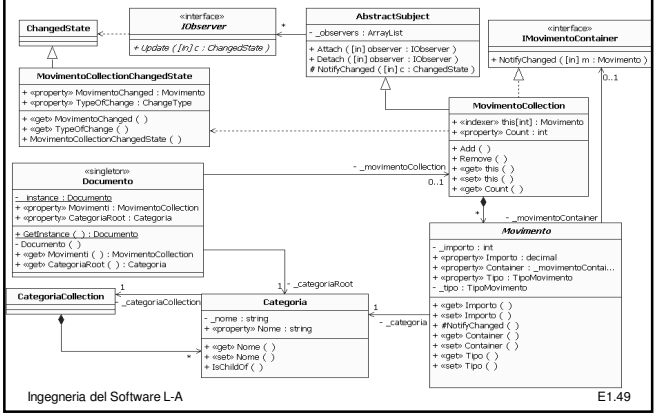
Design – Movimenti$_2$



Movimenti & Categorie$_1$

- Per una questione pratica sarebbe bene che fossero facilmente raggiungibili → tipicamente si realizza un *singleton* (Documento) contenente i dati "interessanti"
- Nel caso specifico, il Documento contiene la collezione di tutti i Movimenti e la Categoria radice dell'albero delle categorie
  - Altro?
- Poiché è necessario che le selezioni si aggiornino automaticamente, occorre che queste siano notificate di ogni cambiamento → pattern *observer!*

Movimenti & Categorie$_2$

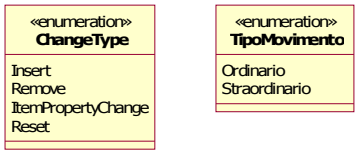


Movimenti & Categorie$_3$

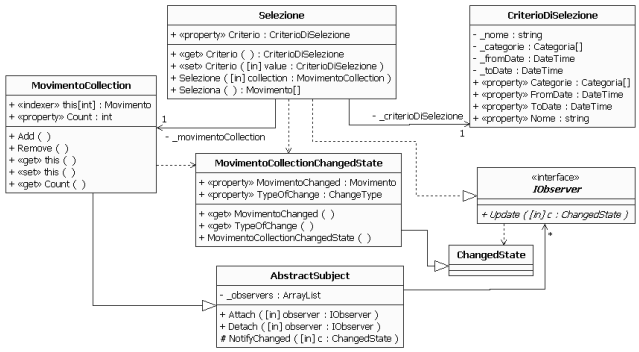
- Pattern Observer, in generale: **AbstractSubject** (l'osservato), **IObserver** (l'osservatore), **ChangeState** (il cambiamento avvenuto)
  - La collezione dei movimenti (**MovimentiCollection**) deriva da **AbstractSubject**
  - Chi vuole osservare implementa **IObserver**
  - Un contratto osservatore-osservato può prevedere la costruzione di una classe derivata da **ChangeState**
- **MovimentoCollectionChangeState** dà un'indicazione di come sia cambiata la collezione dei movimenti
- Ovviamente chi si registra per osservare DEVE sapere cosa sta osservando
- Da notare: il **Movimento** contiene un riferimento alla relativa **Categoria** e NON la **Categoria** contiene una collezione di movimenti!
  - Perché!?

Gli enumerativi…

- Non stavano (in termini di spazio) negli altri diagrammi…

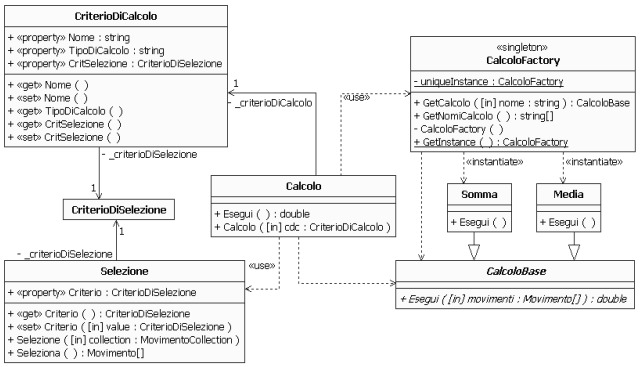


Design – Selezioni



Nota: tutte le proprietà di **CriterioDiSelezione** sono get, set

Design – Calcoli$_1$



Design – Calcoli$_2$

- Pattern: Factory (adattato) + Strategy (= Flyweight?), Singleton
- Factory: si occupa della creazione delle strategie di calcolo
- Strategy: fornisce un'interfaccia comune alle strategie di calcolo in modo che possano essere intercambiabili
  → il principio di sostituibilità di Liskov è fondamentale!

- Funzionamento:
  1. Creare un'istanza di **Calcolo** specificando il **CriterioDiCalcolo**
  2. Il metodo di esecuzione del calcolo **Calcolo**:
     1. Crea (e memorizza) una **Selezione** basata sul **CriterioDiSelezione** contenuta nel **CriterioDiCalcolo**
     2. Esegue la **Selezione**
     3. Ottiene dalla **CalcoloFactory** il **CalcoloBase** specifico (il nome è contenuto nel CriterioDiCalcolo
     4. Esegue il calcolo passando il risultato della selezione e restituisce il risultato del calcolo

Design – Factory?

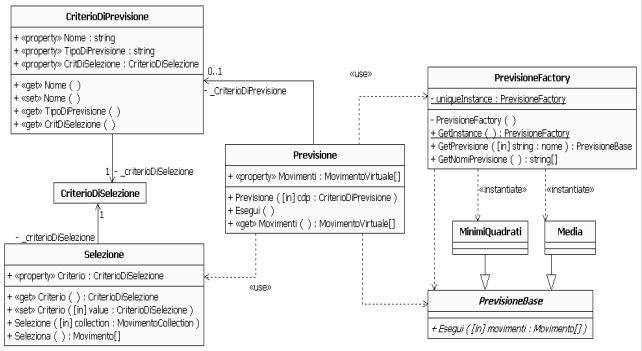
In .Net varie possibilità per costruire una "semplice" factory:

1. La factory conosce le classi concrete da creare e contiene le informazioni di associazione fra nome comune e classi di cui creare le istanze (*switch…case*)
   - Vantaggio: semplicità
   - Svantaggio: factory cablata → aggiungere una nuova classe da costruire significa cambiare la factory → la factory ha una dipendenza verso le classi da creare

Design – Factory?

2. Nel costruttore statico della Factory o nella sezione di inizializzazione dell'applicazione (da file di configurazione?) è contenuto il codice per registrare le classi di cui creare istanze presso la factory; possibili due declinazioni:
   a. Si registrano "Creatori di Istanze" → Pattern Abstract Factory
      - Svantaggio: molte classi in gioco
   b. Si registrano i tipi di cui creare le istanze
      - Occorre conoscere la *reflection...*
3. Le classi di cui creare istanze vengono marcate con un apposito attributo; la factory, all'attivazione dell'applicazione, cerca tutte le classi che derivano da Calcolatore e che possiedono l'attributo
   - Vantaggio: meravigliosamente generale
   - Svantaggio: occorre conoscere la *reflection*…

Design – Previsioni$_1$



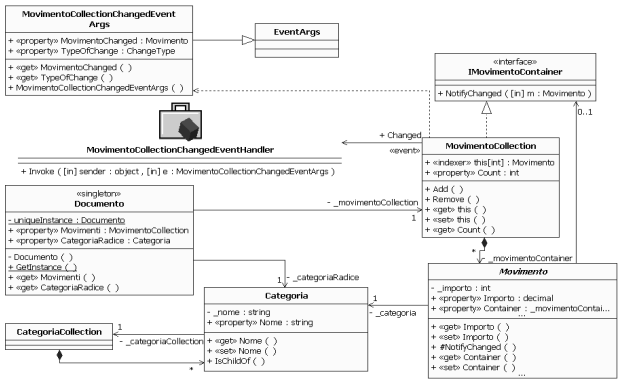
Design – Previsioni$_2$

- Forti analogie con il "problema" dei Calcoli
- Si ha ancora a che fare con Factory e Strategy
- …niente di nuovo…

Sulle selezioni...

- Una **Selezione** fornisce una vista sui dati: consente la selezione dei movimenti partendo da alcune loro caratteristiche
  - Attualmente è implementato in modo "statico" → il **CriterioDiSelezione** è un'entità piuttosto rigida: si stabilisce un intervallo di date e una lista di **Categorie**...
  - L'ideale sarebbe avere la possibilità di esprimere la selezione tramite una *query*
    - Necessità di creare un linguaggio apposito per le *query* (tipo clausola *where* di SQL)
    - Necessità di realizzare un parser ed un interprete…
    - E' un po' complicato… per ora!

Con gli Eventi .Net$_1$



Con gli Eventi .Net$_2$

- Non esiste più l'interfaccia **IObserver**
- Tutti gli eventi derivano da **Delegate** (v. Laboratorio di Ing. Del Sw.)
- Tutti gli argomenti di eventi derivano da **EventArgs**
- Può registrarsi presso un evento qualunque classe abbia un metodo avente parametri e tipi di ritorno identici a quelli esposti dall'evento (stessa signature del tipo del delegato dell'evento)
- Sostanzialmente è un pattern observer... di sistema!

Processo di Sviluppo

- ✓ Requisiti del cliente
- ✓ Analisi del Dominio
- ✓ Analisi dei Rischi (Parziale)
- ✓ Analisi dei Requisiti
- ✓ Casi d'uso
- ✓ Scenari
- ✓ Diagramma statico delle classi
- Analisi dinamica

Analisi Dinamica

- Mostra come si evolve il sistema nel tempo
- Fornisce un'idea (solo un'idea…) di come il sistema potrà funzionare
- Ad ogni scenario di funzionamento può essere fatto corrispondere un diagramma di sequenza
- L'Analisi Dinamica consente di individuare altre operazioni e classi non precedentemente evidenziate

Diagramma di Sequenza

Esecuzione di un Calcolo

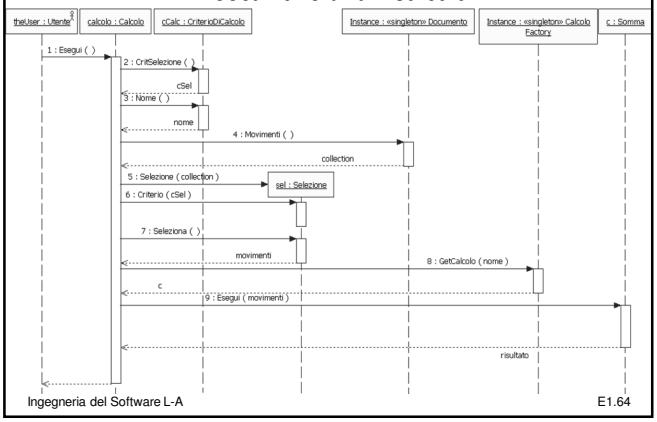


Diagramma di sequenza… …note

- Il numero prima di ogni messaggio evidenzia l'ordine dei messaggi
- Usare una proprietà C# è come usare un metodo → in UML non esistono le "proprietà"
- Non si sa cosa avvenga dentro **CalcoloFactory** → non è stato "esploso" il metodo GetCalcolo...

Diagramma di sequenza

Crea Movimento Reale da Movimento Virtuale$_1$

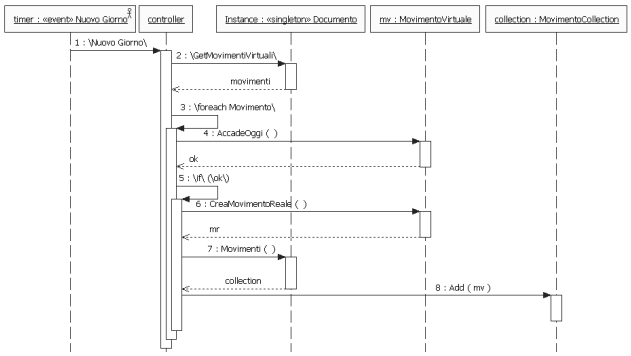


Diagramma di sequenza
Crea Movimento Reale da Movimento Virtuale$_2$

- Chi è il "controller"?
  - Qualcuno che si prende la responsabilità di compiere l'azione sollecitata dall'evento
  - In realtà "quel controller" dovrebbe essere connotato un po' meglio... Meglio come!?
- Chi decide quando è il momento di chiedere ad un movimento virtuale la generazione di un movimento reale?
  - Come funziona l'evento?
- Il metodo GetMovimentiVirtuali va aggiunto a MovimentoCollection...
  - Alternative?

Considerazioni
Crea Movimento Reale da Movimento Virtuale1

- L'evento *Nuovo Giorno* è un attore
- Fa partire la scansione dei movimenti virtuali
- Durante la scansione viene "chiesto" ad ogni movimento virtuale se deve concretizzarsi in un movimento reale (metodo *AccadeOggi*)
- Se il movimento virtuale *AccadeOggi* allora viene invocata l'operazione CreaMovimentoReale()

I passi sopra specificati danno luogo ad un comportamento corretto del sistema solamente se l'evento "Nuovo Giorno" si verifica solamente una volta al giorno…

Considerazioni
Crea Movimento Reale da Movimento Virtuale$_2$

- E se "per sbaglio" l'evento "Nuovo Giorno" scatta più di una volta al giorno?
- Quali sono le precondizioni dell'operazione *CreaMovimentoReale()*?

Precondizioni
CreaMovimentoReale()

1. *AccadeOggi*
2. Non è ancora stato prodotto alcun movimento reale per la data odierna

- La precondizione 2. è indispensabile
- La precondizione 1. non è una vera e propria precondizione →
  - Se il movimento non *AccadeOggi* si può restituire *null*

Design – Cosa manca?

- Persistenza dei dati
- Interfaccia utente
- …