

Scuola Politecnica e delle Scienze di Base

Corso di *Laurea Magistrale* in  
Ingegneria Informatica

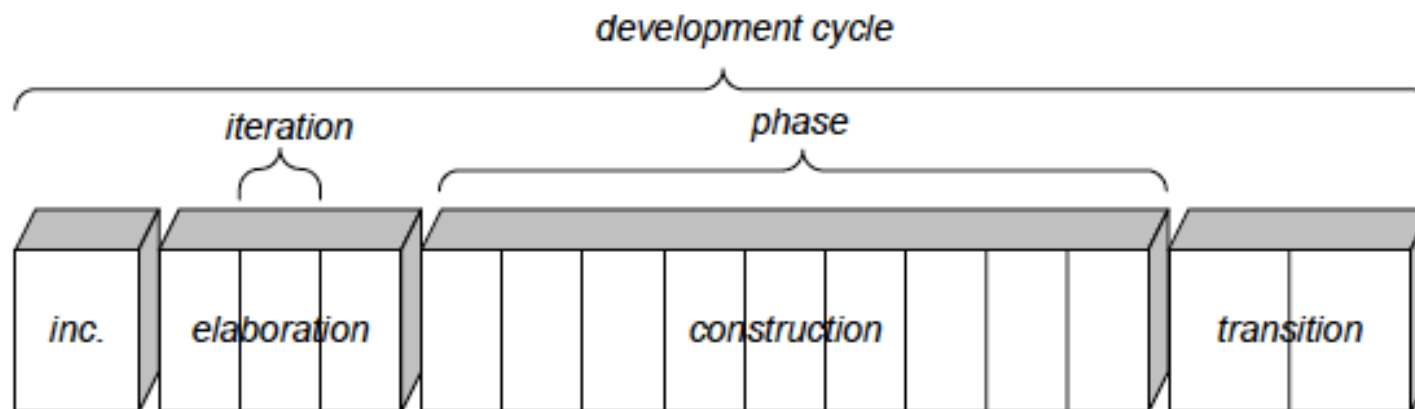


**Elaborazione in UP:  
Iterazione 1**


# Sommario

- Analisi OO: Modello di Dominio (Cap. 12 di Larman)
- Operazioni di Sistema e Diagrammi di Sequenza di Sistema (Cap. 13)
- Contratti delle Operazioni di Sistema con pre e post-condizioni (Cap. 14)
- Dai requisiti alla progettazione (cap. 15)
- Architettura Logica con strati (Cap. 16)
- Verso la progettazione ad oggetti attraverso i Diagrammi di Interazione (Cap. 17)

# Fasi del Processo UP



# Dall'Ideazione verso la Prima Iterazione

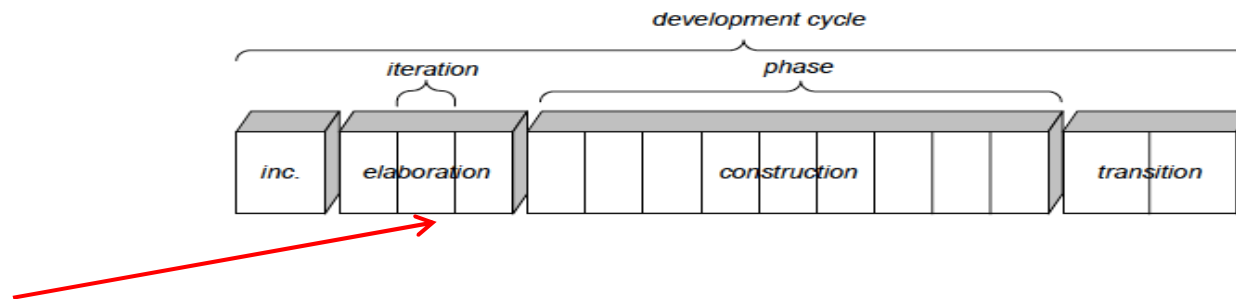
- In fase di **Ideazione** si è determinata la fattibilità.  
Cosa si è fatto?
- Un breve Workshop dei requisiti
- Assegnazione dei nomi ad attori, obiettivi, casi d'uso 
- Scrittura in formato breve della maggior parte dei casi d'uso
- Scrittura dettagliata di un primo 10% dei requisiti
- Requisiti di qualità più importanti
- Lista dei Rischi
- Prototipi (tecnici proof-of-concept, prototipi di UI)
- Proposta dell'architettura di Alto livello
- Piano per la prima Iterazione

# I requisiti di PosNextGen scelti per la prima iterazione



- Implementare uno scenario base del caso d'uso ElaboraVendita: inserire articoli e ricevere un pagamento in contanti.
- Implementare un caso di Avviamento (StartUp) per gestire l'inizializzazione.
- Considerare solo i casi base
- Non c'è collaborazione con servizi esterni (come la base di dati dei prodotti o sistema esterno di contabilità)

# Verso l'Elaborazione


- Durante le iterazioni di elaborazione verrà:
  - Programmato e verificato il nucleo, rischioso, dell'architettura software
  - Viene scoperta e stabilizzata la maggior parte dei requisiti
  - I rischi maggiori vengono attenuati



# Elaborati dell'Elaborazione

Elaborato	
Modello di Dominio	È una visualizzazione degli oggetti di dominio 
Modello di Progetto	Diagrammi della Progettazione Logica: Class Diagrams, Diagrammi di interazione fra oggetti, diagramma dei Package 
Documento dell'Architettura Software	Descrizione degli aspetti principali dell'architettura. È una sintesi delle scelte di progetto fatte e loro motivazione
Modello dei Dati	Schemi della Base di Dati e strategie di Mapping fra la rappresentazione ad oggetti e la base di dati
Storyboard dei Casi d'Uso, Prototipi UI	Descrizione della UI, della navigazione

## Modello di Dominio


- È il modello object-oriented più importante dell'analisi OO. Esso illustra i concetti fondamentali del dominio.
- In UP è chiamato *Modello degli Oggetti di Business*
- Può contenere solo:
  - Classi concettuali o oggetti di dominio (oggetti presenti nel dominio del problema) 
  - Associazioni fra classi concettuali
  - Attributi di Classi concettuali
  - NON sono definite operazioni
- Un Modello di Dominio non coincide con il Modello dei Dati (non conterrà solo classi con attributi da memorizzare in modo persistente)



# Linee Guida per costruire Modelli di Dominio

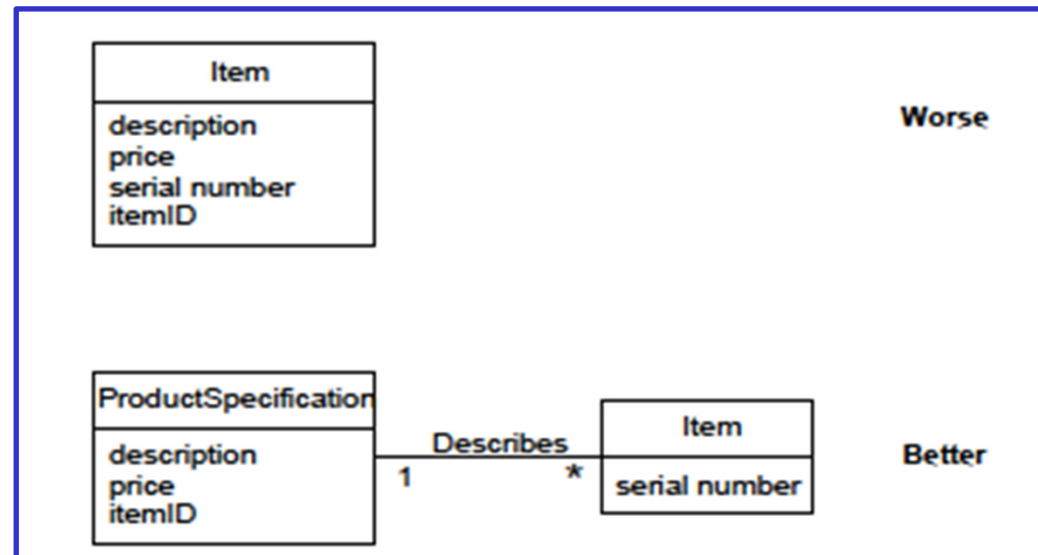
- Trovare le classi concettuali candidate.
  1. Usare Modelli di Dominio già esistenti
    - Alcuni domini (finanza, inventario, salute...) hanno già modelli di dominio definiti
    - Analysis Patterns di Martin Fowler
    - Data Model Patterns di David Hay,..
  2. Usare Categorie di Classi concettuali comuni
  3. Usare l'analisi linguistica (analisi di nomi e locuzioni nominali) di specifiche informali, o anche scenari di casi d'uso

# Esempi di Categorie di classi concettuali

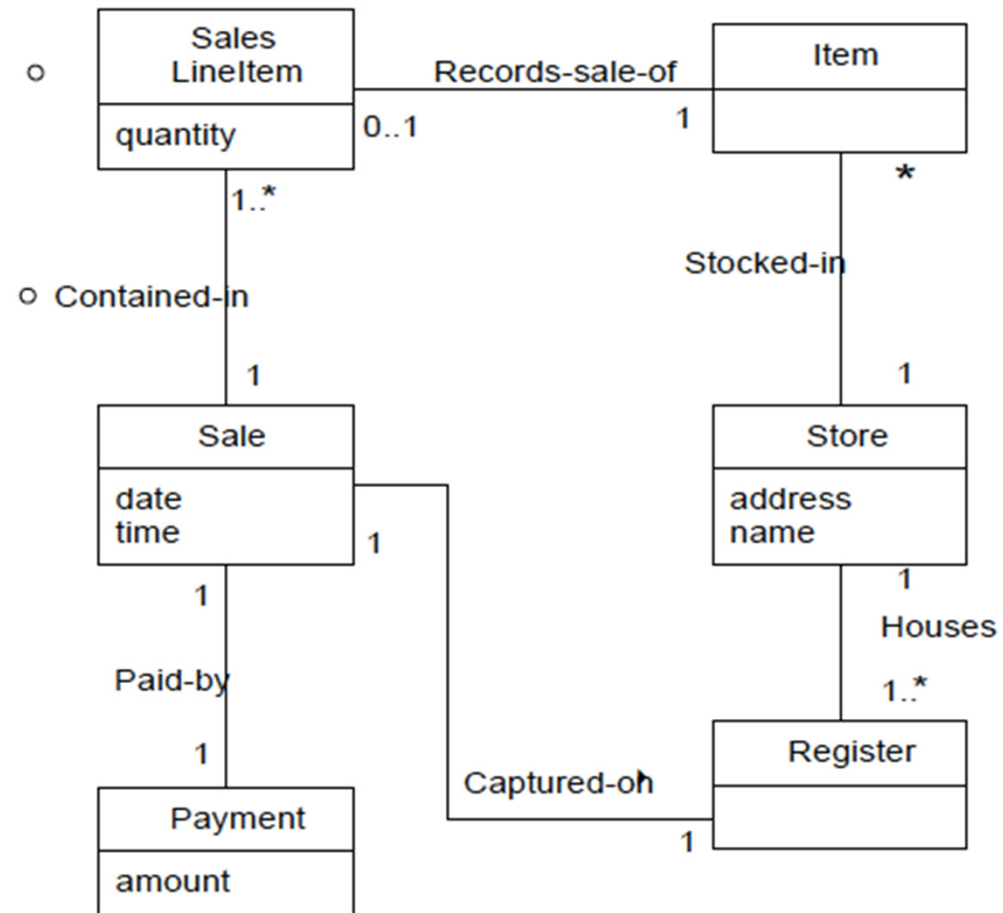
- Transazioni commerciali (es. Sale)
- Righe di transazioni (SalesLineItem)
- Prodotto/articolo collegato ad una transazione (Item/ Flight/ Seat/..)
- Dove viene registrata la transazione? (Register, LibroMastro)
- Ruoli di persone nelle transazioni (Cassiere, Cliente, Passeggero...)
- Luogo della transazione (Negozio, Aeroporto, Aereo, Posto)
- Eventi significativi (Vendita, Pagamento, Turno di Gioco, )
- Oggetti Fisici (Articolo, Registro, Scacchiera, Dado..)
- Descrizioni di Oggetti (ProductDescription, FlightDescription) 
- Cataloghi
- Altri sistemi che collaborano (sistema AutorizzazionePagamenti)

## Quando usare le classi descrizione?

- Quando è necessaria una descrizione di un articolo, o servizio, indipendentemente dall'effettiva esistenza di istanze di articoli/ servizi
- Quando l'eliminazione di istanze di oggetti comporterebbe perdita di informazioni da conservare
- Quando si vogliono evitare informazioni ripetute o ridondanti.




# Esempio: Modello di Dominio del Sistema POS (versione iniziale)



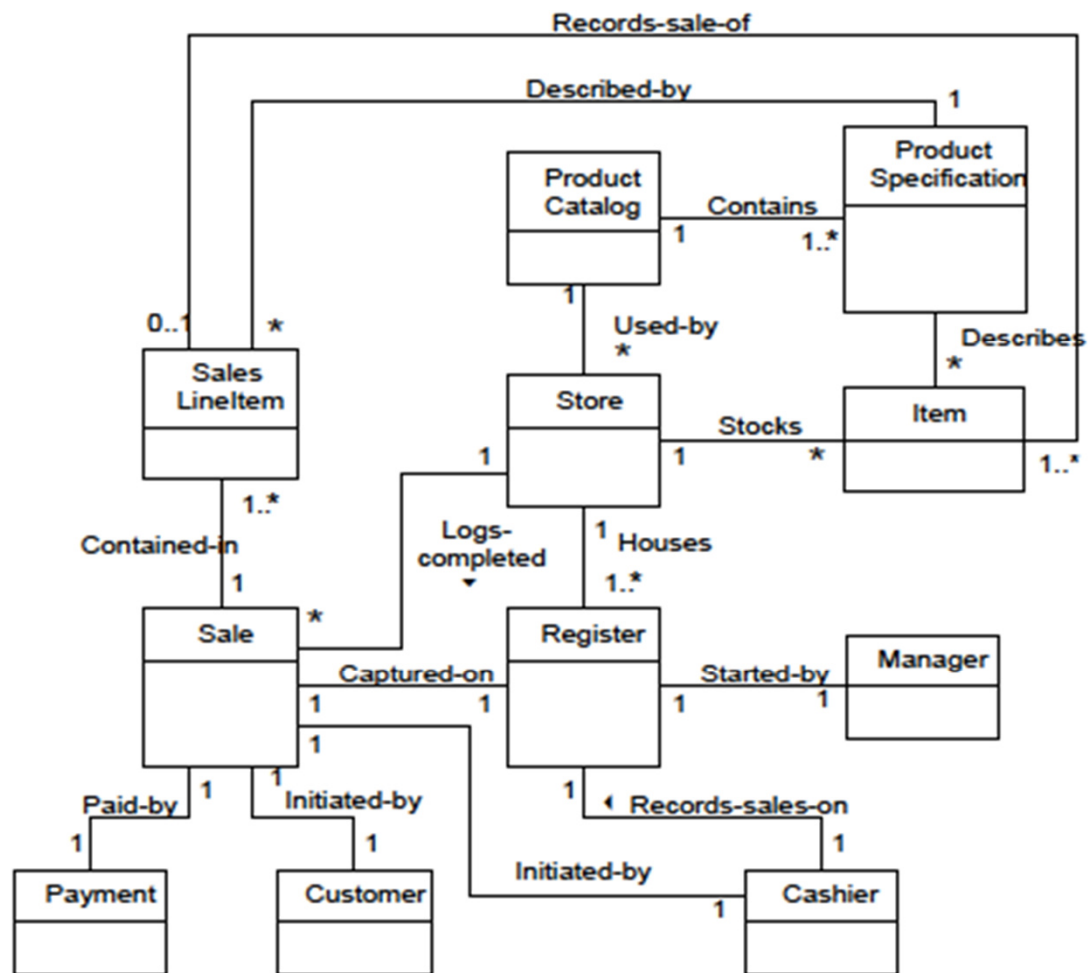
## Linea Guida: modellare il mondo non reale

- Alcuni sistemi software riguardano domini che hanno poche analogie con i domini aziendali (es. Un sistema per le telecomunicazioni)
- È comunque possibile creare un modello di dominio anche per essi.
- Prestare attenzione alla terminologia e concetti usati dagli esperti:
- Es. Classi concettuali del dominio di un Commutatore telefonico:
  - Messaggio, Connessione, Porto, Dialogo, Route, Protocollo

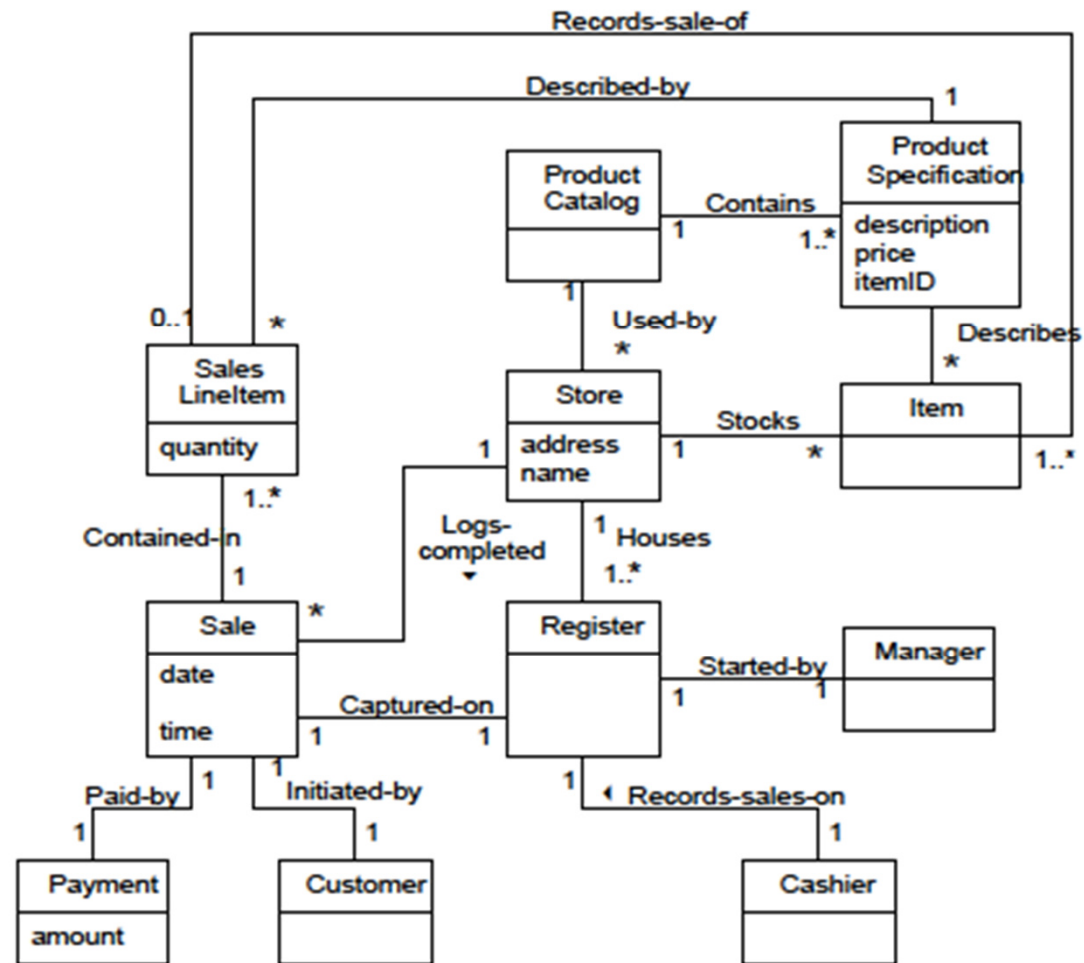
# Definire le Associazioni nei modelli di dominio

- Proprietà delle associazioni:
  - Significato dell'associazione (nome significativo)
  - Molteplicità 
  - Ruolo
  - Associazioni multiple
  - Aggregazione e Composizione
    - La composizione richiede che siano soddisfatti tre vincoli:
      - Ciascuna istanza della parte può appartenere ad una sola istanza del composto
      - Ciascuna parte deve appartenere sempre ad un composto (non esiste al di fuori di esso)
      - La vita delle parti è limitata da quella del composto (la parte non esiste prima del composto e non sopravvive al composto)

# Modello di Dominio del Sis. POS (versione raffinata)

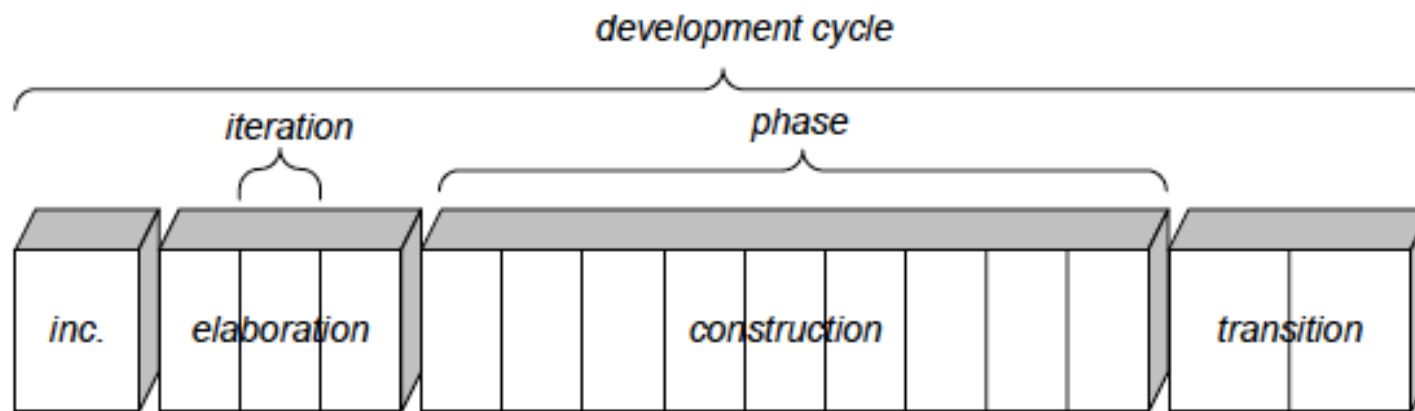


# Modello di Dominio con attributi





# Modello di Dominio in UP



## Modelli di Dominio in UP

- Di solito in UP il modello di dominio è iniziato e completato in fase di Elaborazione.

Discipline	Artifact Iteration→	Incep. I1	Elab. E1..En	Const. C1..Cn	Trans. T1..T2
Business Modeling Requirements	<i>Domain Model</i>		s		
	Use-Case Model (SSDs)	s	r		
	Vision	s	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	
Implementation	Implementation Model		s	r	r
Project Management	SW Development Plan	s	r	r	r
Testing Environment	Test Model		s	r	
	Development Case	s	r		

- Non è necessario in fase di Ideazione

## Verso il modello di Progetto

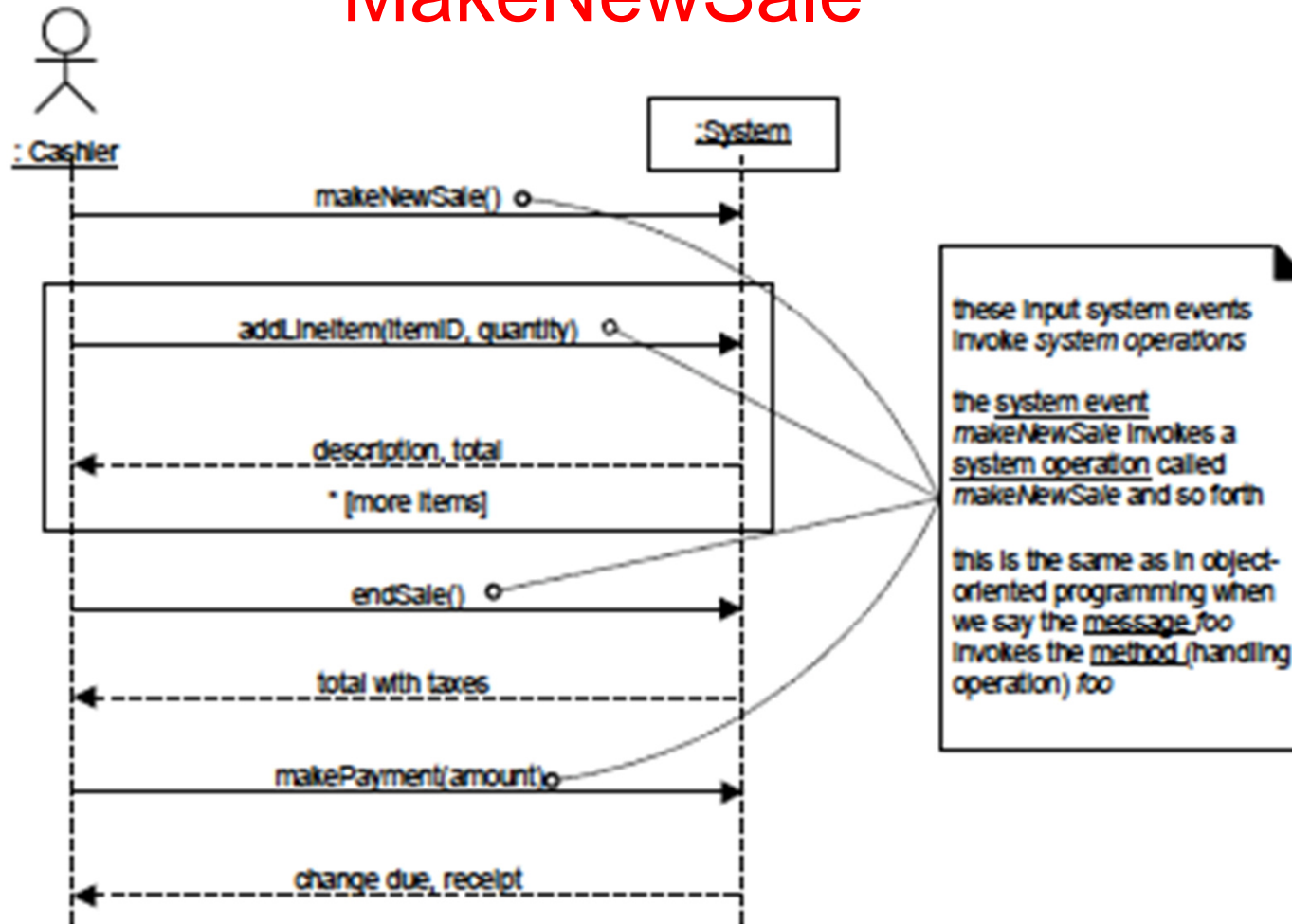
Occorre identificare le operazioni offerte dal sistema.

Ci arriveremo attraverso i Diagrammi di Sequenza di Sistema (SSD)

# Definizione di Operazioni di Sistema usando gli SSD


- Un Diagramma di Sequenza di Sistema (SSD) può essere usato per definire gli input ed output dei sistemi da realizzare.
- Un SSD rappresenta, per un particolare scenario di caso d'uso:
  - *Gli attori che interagiscono col sistema*
  - *Il Sistema a scatola nera*
  - *Eventi di sistema* (ossia gli input di un attore che richiedono l'esecuzione di *operazioni di sistema*)
    - Es. Cassiere inserisce codice prodotto (richiede la registrazione prodotto)
- Analizzando l'SSD si potranno dedurre alcune operazioni che il sistema dovrà offrire

# Esempio: SSD per il Caso d'uso MakeNewSale



Operazioni richieste: makeNewSale, addLineItem, endSale, makePayment (con i rispettivi valori di ritorno).

## SSD all'interno di UP

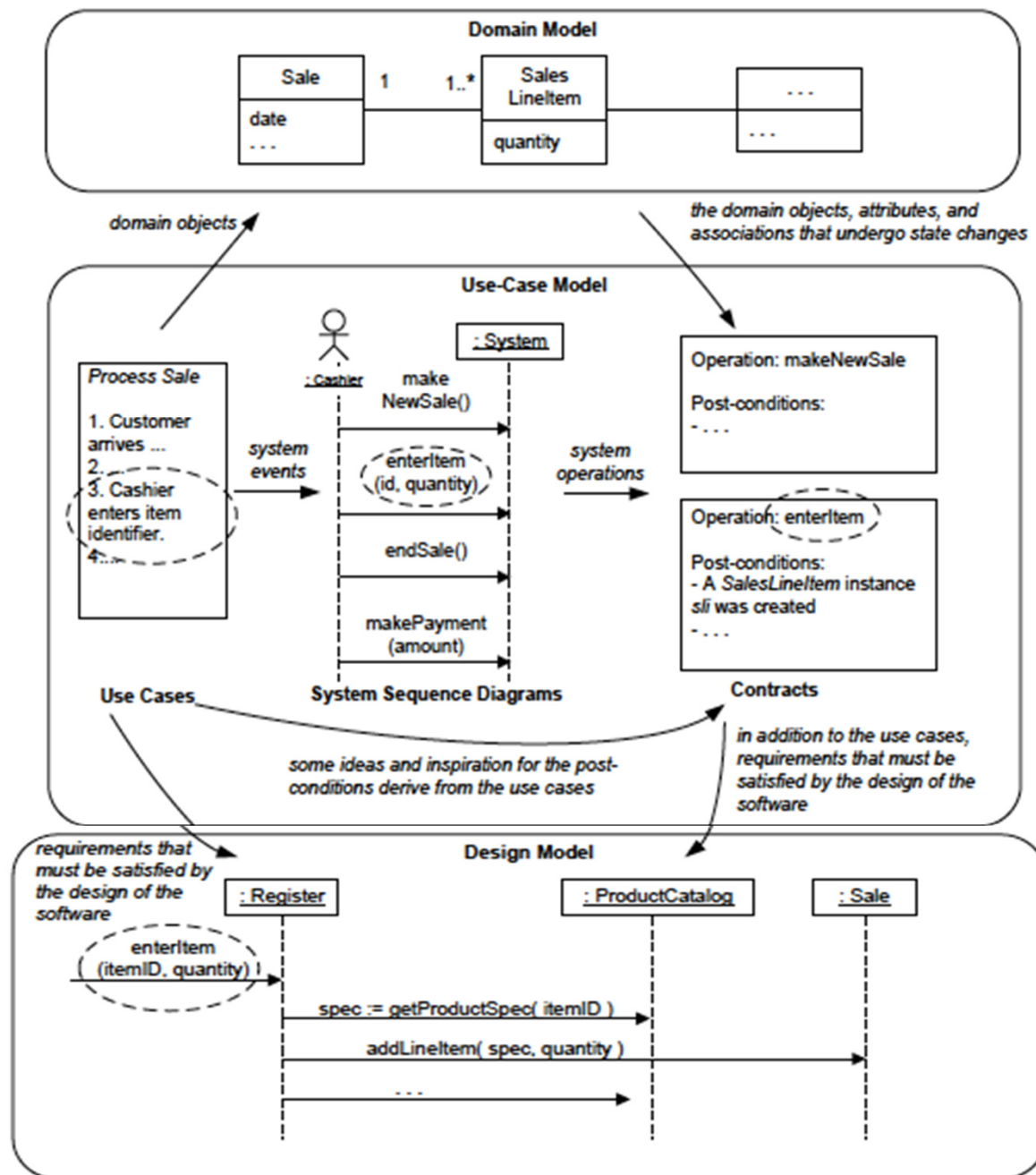
- Gli SSD vanno abbinati ai casi d'uso, che descrivono graficamente.
- Non sono necessari in fase di Ideazione.
- Vanno costruiti in fase di Elaborazione, per definire gli eventi e le operazioni del sistema e poi scrivere i contratti delle operazioni di sistema. 
- Ovviamente, vanno sviluppati solo per i casi d'uso e gli scenari che saranno oggetto della specifica Iterazione.

# Contratti delle Operazioni di sistema

- I contratti usano pre-condizioni e post-condizioni per specificare i cambiamenti agli oggetti di un dominio, come risultato dell'esecuzione di un'operazione di sistema.
- I contratti possono far parte del Modello dei Casi d'uso.
- Gli input per i contratti delle operazioni sono:
  - SSD,
  - Modello di Dominio
- I contratti saranno gli input per la progettazione ad oggetti

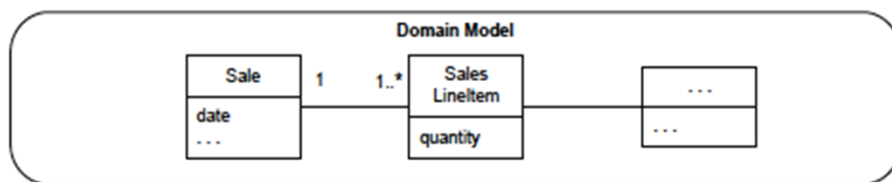
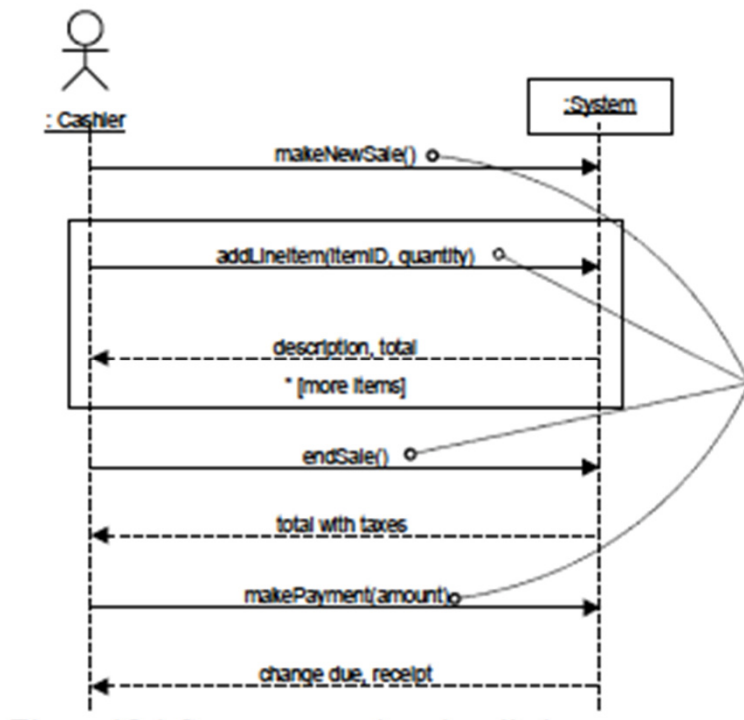
# Influenza fra elaborati in UP

## Modello di Business





# Esempio di Contratto dell' operazione *addLineItem*



**Operazione:** *addLineItem* (itemID:  
itemID, quantity:integer)

**Riferimenti:** caso d'uso *Elabora  
Vendita*

**Pre-condizioni:** è in corso una  
vendita *s*

**Post-condizioni:**

- è stata creata un'istanza *sli* di *SalesLineItem*
- *sli* è stata associata con la *Sale s*
- *Sli* è stata associata con una *ProductDescription*
- *sli.quantity* è diventata *quantity*

## Le sezioni di un Contratto

- **Operazione:** nome e parametri (firma) dell'operazione
- **Riferimenti:** Casi d'uso in cui compare
- **Pre-condizioni:** ipotesi su stato degli oggetti nel Modello di Dominio prima di eseguire l'operazione
- **Post-condizioni:** i cambiamenti di stato degli oggetti, dopo il completamento dell'operazione
  - Es.: Creazione/ Cancellazione di un oggetto
  - Formazione/ Rottura di un collegamento
  - Cambiamento del valore di un attributo

# Esempi

- **Contratto CO1: makeNewSale...**
  - Post-condizioni:
    - è stata creata una istanza s di Sale
    - s è stata associata con il Register
    - Gli attributi di s sono stati inizializzati
- **Contratto CO4: makePayment**
  - Precondizioni: è in corso una vendita s
  - Post-condizioni:
    - È stata creata una istanza p di Payment
    - p è stata associata alla Sale s
    - p.amount è diventato amount
    - La sale s è stata collegata allo Store (registro vendite completate)

## Contratti in UP

- Ideazione: i contratti non sono giustificati in fase di ideazione
- Elaborazione: I contratti vanno definiti in questa fase, solo per le operazioni di sistema più complesse
- Ovviamente, attraverso i contratti emergerà la necessità di registrare nuove classi concettuali, attributi o associazioni → il Modello di Dominio sarà ampliato iterativamente


Dai requisiti alla Progettazione, iterativamente.  
Architettura Logica.

Cap. 15 e 16(Larman)

# Processo UP: prime iterazioni

- Durante l'ideazione, viene esaminato un 10% dei requisiti
- Durante la prima Iterazione di Elaborazione, si saranno analizzati i requisiti con la produzione delle prime versioni dei **Requisiti**, **Casi d'uso**, *Modello di Dominio*, *Sequence Diagram di Sistema*, e eventuali *Contratti di Operazioni*.
- In questa iterazione, si avvia anche la *Progettazione* di una soluzione, in termini di oggetti software che collaborano.

# Architettura Logica

- È l'organizzazione delle classi software in Package, sottosistemi e strati.
- È logica perché non specifica come questi componenti siano distribuiti in processi o diversi computer di una rete (vedi architettura di deployment). 
- Architettura Logica a strati:
  - Ogni strato è un insieme di classi, package, sottosistemi che ha delle responsabilità coese rispetto agli obiettivi del sistema.
  - Strati più alti usano i servizi degli strati inferiori


# Architettura a Strati

- User Interface (Interfaccia Utente o Presentazione)
  - Oggetti software per gestire interazione utente (finestre, menu, dialog, pulsanti, oggetti per gestire eventi)
- Application Logic e Domain Objects
  - Oggetti che rappresentano concetti di dominio che soddisfano i requisiti della applicazione (es. calcolare totale di una vendita)
- Servizi tecnici
  - Oggetti che forniscono servizi di supporto, come accesso alla base di dati o logging degli errori.
  - I servizi possono essere indipendenti dall'applicazione e riusabili in altri contesti

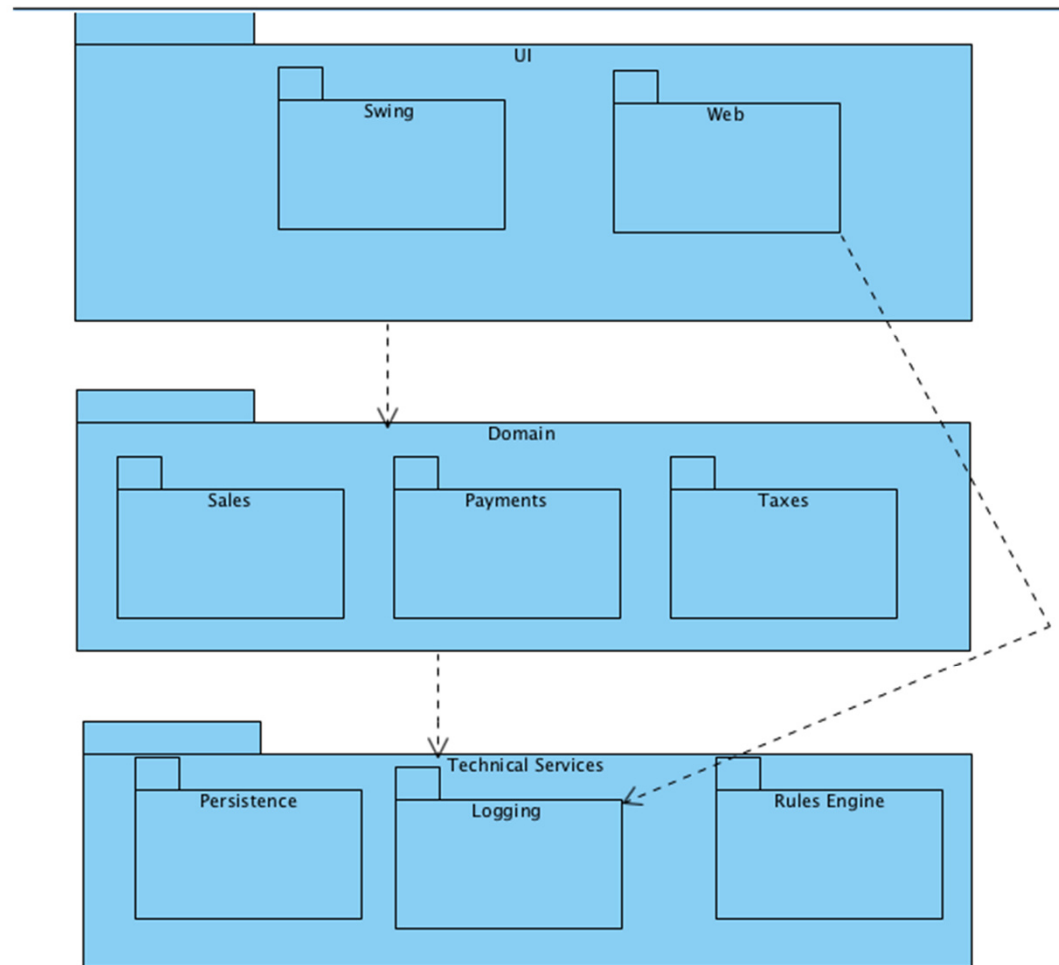




# Architettura a Strati con Package Diagram UML

- Ogni strato può essere modellato da un Package UML (contenitore di classi, package, casi d'uso, etc.).
- Un Package UML fornisce un namespace. 
- Es. per la classe Date, il nome completamente qualificato è : *java::util::Date*
- Tra i package si possono rappresentare relazioni di dipendenza (linea tratteggiata)
- Oppure relazioni di Contenimento, rappresentate in vari modi.




# Esempio



## Linea Guida: progettazione con gli strati

- L'uso degli strati risolve vari problemi, quali:
- Le modifiche si propagano in tutto il sistema, con parti molto accoppiate
- La logica applicativa è intrecciata con la UI, per cui la logica non è riusabile con interfacce diverse, e non è distribuibile su nodi diversi
- I servizi o la logica più generali si intrecciano con logica specifica di una applicazione, per cui non sono riusabili nè ridistribuibili
- Le parti sono accoppiate, per cui è difficile separare il lavoro fra diversi sviluppatori

# Linea Guida: Mantenere la Separazione di Interessi

- Una possibile Architettura a strati di un sistema informatico:
- Strato UI
- Strato Application 
- Strato Domain 
- Low-Level Business Services (es. Convertitore di Valuta)
- Technical Services (es. Persistenza, Sicurezza)
- Foundation (Core Services, utilità e framework, thread, matematica, file, DB, I/O da rete) 

## Vantaggi dell'uso degli strati

- C'è separazione di interessi, tra servizi di alto e basso livello, tra servizi generali e specifici
  - Minori accoppiamenti e dipendenze, maggiore coesione, maggiore riuso e comprensibilità
- La complessità è decomponibile
- Alcuni strati sono sostituibili da altri
  - In genere quelli di più alto livello sono sostituibili meglio (UI, Application e Dominio) di quelli dei Servizi Tecnici
- Gli strati più bassi contengono funzioni riusabili
- Alcuni strati sono distribuibili (Es. Domain e Technical Services)
- Sviluppo separabile fra più team

# Corrispondenza fra Strati e package UML e organizzazione del codice

**/\*\* Strato UI\*\*/**

com.mycompany.nextgen.ui.swing

com.mycompany.nextgen.ui.web

**/\*\* Strato Domain \*\*/**

com.mycompany.nextgen.domain.sales

com.mycompany.nextgen.domain.payments

**/\*\* Strato Technical Services\*\*/**

org.apache.log4j

org.apache.soap.rpc

**/\*\* Strato Util \*\*/**

com.mycompany.util


Usare in fase di programmazione le stesse convenzioni sui nomi dei package dei vari strati, per ricostruire il package diagram anche per reverse engineering.

# Linee Guida per progettare gli Strati

## -Strato di Dominio-



- Evitare di porre tutta la logica applicativa in una sola classe.
- È preferibile progettare più classi che abbiano nomi ed informazioni simili al dominio del mondo reale e assegnare ad esse responsabilità della logica.
- Es. le Classi *Sale* e *CashPayment* sono oggetti di dominio, ove *Sale* è responsabile di calcolare il suo totale.
- Strato di Dominio: contiene oggetti di dominio per gestire la logica applicativa
- Lo strato di Dominio (è software) non è il Modello di Dominio (analisi), ma è ispirato ad esso.
  - Salto rappresentazionale basso

# Confronto fra Strato di Dominio e Strato Application

- In alcuni casi può essere utile interporre uno strato Application fra strati UI e Dominio. Quando?? 
- L'Application contiene oggetti responsabili di mediare, ricevono dalla UI le richieste di operazioni di sistema e le inoltrano agli oggetti di Dominio.
- Gli oggetti dell'Application possono essere responsabili di conoscere lo stato delle sessioni con i client o di controllare il flusso di lavoro (es. flusso schermate)
- Lo strato Application è opzionale e serve proprio in questi casi.




# Linea Guida: Principio di Separazione Modello-Vista

- Gli oggetti non-UI non devono essere direttamente accoppiati agli oggetti UI. 
  - È errato che un oggetto Sale abbia direttamente un riferimento all'oggetto JFrame di Java Swing.
    - Gli oggetti dominio dovrebbero poter essere riutilizzati in altre applicazioni o con altra UI
- Non mettere logica applicativa nei metodi di un oggetto della UI 
  - Questi oggetti dovrebbero solo inizializzare gli elementi della UI, ricevere gli eventi UI (es. click) e delegare le richieste agli oggetti non UI (es. oggetti di dominio)
- In pratica: separazione Modello dalla Vista (View).

## Separazione Model-View

- Gli oggetti del modello non devono conoscere gli oggetti della View.
- Un rilassamento di questo principio è il Pattern Observer, in cui oggetti del Dominio mandano messaggi a oggetti della UI, ma indirettamente, usando i metodi di una interfaccia *PropertyListener*, che non è di tipo UI.
- Vantaggi:
  - si possono aggiungere nuove viste senza impatti sul dominio
  - Più viste simultanee sugli stessi oggetti di modello
  - Porting semplificato dello strato modello

# Verso la progettazione ad oggetti

- Alcuni approcci per progettare:
  - Progettare mentre si codifica (in Java..) ,magari con strumenti come TDD e Refactoring. Dal modello mentale al codice.
  - Disegno in UML, poi codifica.
  - Solo Disegno: lo strumento genera il codice dai diagrammi.
  - Un possibile approccio è: Disegno leggero in UML e poi codifica. 
  - Nella progettazione ad oggetti, si usano due tipi di modelli (statici e dinamici).

# Modellazione a oggetti dinamica e statica in UML

- Modellazione a oggetti statica:
  - Basata su diagrammi delle classi UML,
  - Anche su Package diagrams e Deployment diagrams
- Modellazione a oggetti dinamica:
  - Basata su Diagrammi di Interazione UML:
  - Sequence e Collaboration diagrams aiutano a progettare la logica, il comportamento del codice e il corpo dei metodi.
  - Altra modalità: Progettazione Guidata dalle Responsabilità con i Principi GRASP (General Responsibility Assignment Software Patterns) 