12 . Esempio di Progettazione con i pattern GRASP

Sviluppo di Applicazioni Software

Matteo Baldoni

a.a. 2023/24

Università degli Studi di Torino - Dipartimento di Informatica

Attenzione!

Table of contents

Esempi di progettazione con GRASP

Progettazione

Importante

L'assegnazione delle responsabilità e la progettazione delle collaborazioni sono passi molto importanti e creativi della progettazione, sia durante la creazione dei diagrammi che durante la codifica.

Però, l'assegnazione delle responsabilità e la scelta delle collaborazioni può e deve essere spiegata in modo razionale.

Realizzazione di un caso d'uso

Realizzazione di un caso d'uso (o di uno scenario)

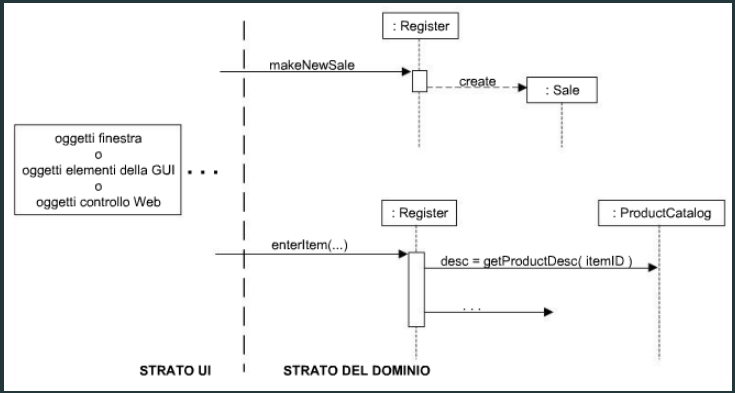
Una realizzazione di un caso d'uso descrive come viene realizzato un caso d'uso all'interno del Modello di Progetto, in termini di oggetti che collaborano.

- Il caso d'uso suggerisce le operazioni di sistema, mostrate negli SSD
- Le operazioni di sistema diventano i messaggi iniziali entranti nei controller per i diagrammi di interazione per lo strato del dominio
- I diagrammi di interazione per lo strato del dominio illustrano come gli oggetti interagiscono per soddisfare i compiti richiesti, ovvero la realizzazione di caso d'uso

Gestione delle operazioni di sistema

Le operazioni di sistema negli SSD sono usate come messaggi iniziali per gli oggetti controller dello strato del dominio.

Gli eventi di sistema sono catturati dallo strato UI che li "trasforma" in operazioni di sistema.



Esempio: makeNewSale

L'operazione di sistema *makeNewSale* avviene quando un cassiere inizia una richiesta per avviare una nuova vendita, dopo che un cliente è giunto alla cassa con degli articoli da acquistare.

Contratto CO1: makeNewSale

Operazione:	makeNewSale()
Riferimenti:	Casi d'Uso: Elabora Vendita
Pre-condizioni:	nessuna.
Post-condizioni:	– è stata creata un'istanza s di Sale. – s è stata associata con il Register. – gli attributi di s sono stati inizializzati.

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: makeNewSale, controller

La prima scelta di progetto riguarda la scelta del controller per il messaggio dell'operazione di sistema (*controller*).

- Un oggetto di sistema complessivo, l'oggetto radice, un dispositivo speciale, un punto d'accesso o un sottosistema principale: *Store*, *Register*, *POS-Terminal* sono possibili scelte
- Un oggetto che rappresenta un ricevitore o un gestore di tutti gli eventi di sistema di uno scenario di caso d'uso: *ProcessSaleHandler*, *ProcessSaleSession* sono possibili scelte

Esempio: makeNewSale, controller

La prima scelta di progetto riguarda la scelta del controller per il messaggio dell'operazione di sistema (*controller*).

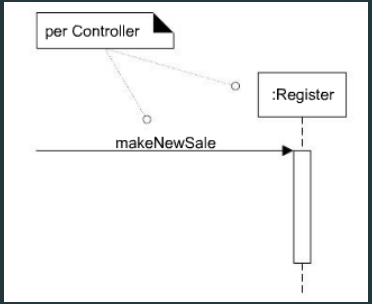
- Un oggetto di sistema complessivo, l'oggetto radice, un dispositivo speciale, un punto d'accesso o un sottosistema principale: *Store*, *Register*, *POS-Terminal* sono possibili scelte
- Un oggetto che rappresenta un ricevitore o un gestore di tutti gli eventi di sistema di uno scenario di caso d'uso: *ProcessSaleHandler*, *ProcessSaleSession* sono possibili scelte

La scelta di *facade controller* relativa a un oggetto-dispositivo come *Register* è soddisfacente se ci sono solo poche operazioni di sistema, e se il facade controller non assume troppe responsabilità (cioè non diventa poco coeso).

Esempio: makeNewSale, controller

Si noti che

Register è un oggetto software nel *Modello di Progetto*, non un registratore di cassa fisico.



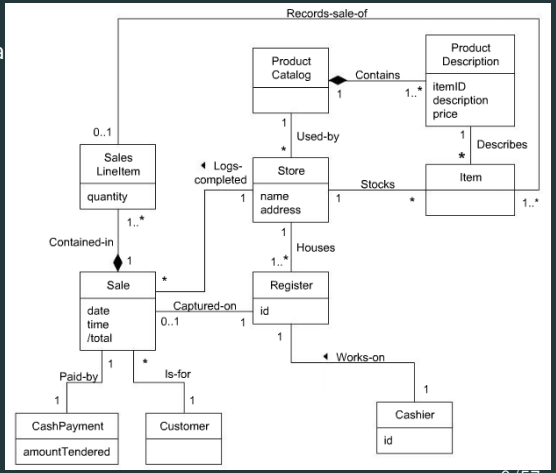
©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: makeNewSale, prima post-condizione "è stata creata un'istanza s di Sale"

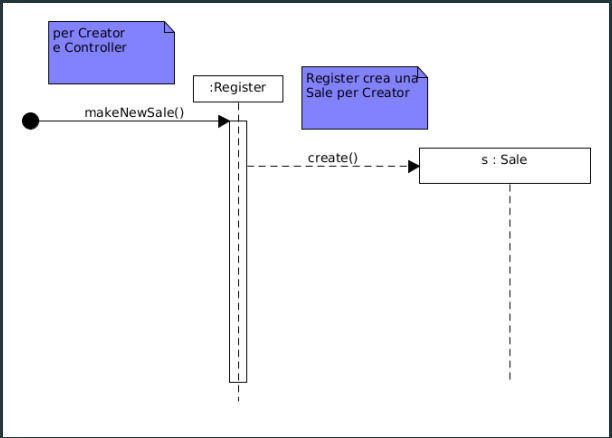
Una post-condizione del contratto indica la creazione di un oggetto *concettuale Sale*.
Il pattern GRASP Creator, suggerisce di assegnare la responsabilità per la creazione a una classe che aggrega, contiene o registra l'oggetto da creare.

- *Register* può essere considerato come il registratore di una *Sale*
- *Store* registra vendite, e pertanto va considerato un altro candidato creatore di una *Sale*

Store registra vendite completate, la *Sale* da creare va considerata, al momento, solo un tentativo di vendita. Pertanto *Register* è un candidato ragionevole per la creazione di una *Sale*.



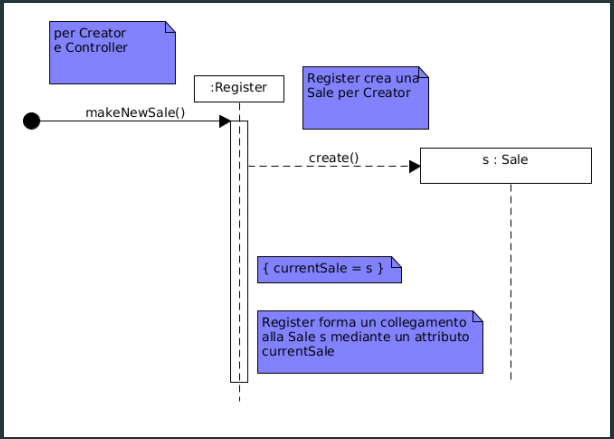
Esempio: makeNewSale, diagramma di interazione



Esempio: makeNewSale, seconda post-condizione “s è stata associata con il Register”

Dopo che il *Register* ha creato la *Sale*, è possibile associare la *Sale* al *Register* per tutta l'esecuzione del caso d'uso, in modo che durante le operazioni successive all'interno della sessione il *Register* mantenga un riferimento all'istanza corrente nell'attributo *currentSale*.

Esempio: makeNewSale, diagramma di interazione



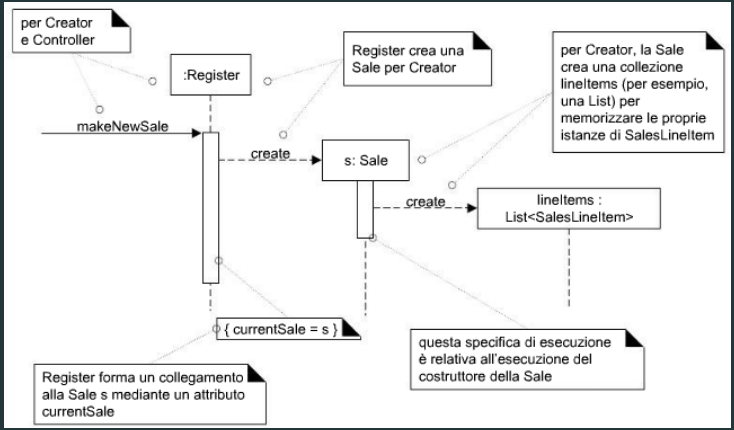
Esempio: makeNewSale, terza post-condizione “gli attributi di s sono stati inizializzati”

La *Sale* deve creare una collezione vuota per memorizzare tutte le future istanze di *SalesLineItem* che verranno aggiunte alla vendita.

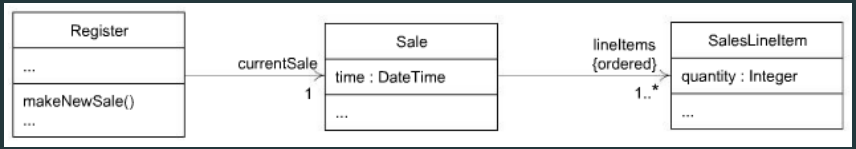
Questa collezione sarà associata all'istanza *Sale* e mantenuta dalla stessa. Quindi *Sale* è un buon candidato anche per la sua creazione

Pertanto il *Register* crea la *Sale*, la *Sale* crea una collezione vuota di *SalesLineItem*, e infine il *Register* memorizza la *Sale* corrente.

Esempio: makeNewSale, diagramma di interazione



Esempio: makeNewSale, diagramma delle classi



Esempio: enterItem

L'operazione di sistema *enterItem* avviene quando un cassiere inserisce l'*itemID* e la quantità di un articolo da acquistare.

Contratto CO2: enterItem

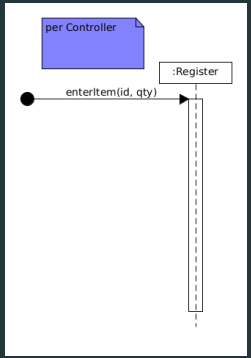
Operazione:	enterItem(itemID: ItemID, quantity: Integer)
Riferimenti:	Casi d'uso: Elabora Vendita
Pre-condizioni:	è in corso una vendita s.
Post-condizioni:	– è stata creata un'istanza sli di SalesLineItem. – sli è stata associata con la Sale corrente s. – sli è stata associata con una ProductDescription, in base alla corrispondenza con itemID. – sli.quantity è diventata quantity.

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: enterItem, controller

Il pattern *Controller* suggerisce di utilizzare la stessa classe controller per tutte le operazioni di sistema di un caso d'uso; pertanto si continuerà a utilizzare *Register* come controller per l'operazione *enterItem* e per le ulteriori operazioni di sistema del caso d'uso *Elabora Vendita*.

Esempio: enterItem, diagramma di interazione

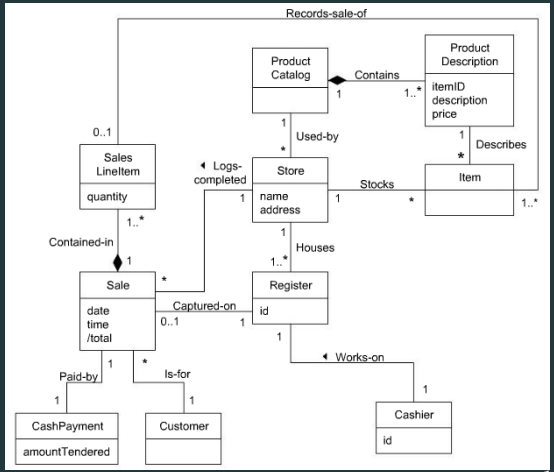


Esempio: enterItem, prima post-condizione “è stata creata un’istanza sli di SalesLineItem”

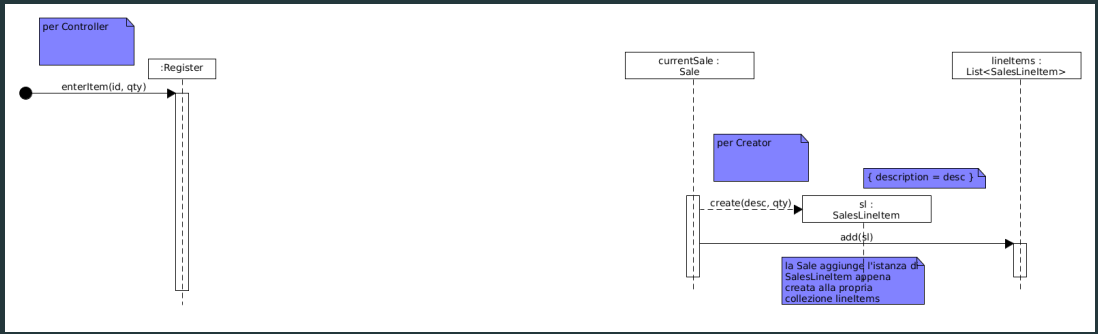
È necessario creare, inizializzare e associare una *SalesLineItem*.

- Una *Sale* software (traendo ispirazione dal Modello di Dominio) contenere degli oggetti software *SalesLineItem*. Per *Creator*, una *Sale* software è un candidato appropriato per la creazione di una *SalesLineItem*
- Un altro candidato creatore di *SalesLineItem* è il controller *Register*, poiché possiede tutti i dati necessari per l’inizializzazione della nuova vendita

Creator suggerisce di preferire la *Sale*, in virtù della composizione tra *Sale* e *SalesLineItem*.



Esempio: enterItem, diagramma di interazione



Esempio: enterItem, seconda, quarta post-condizione “sli è stata associata con la Sale corrente s” e “sli.quantity è diventata quantity”

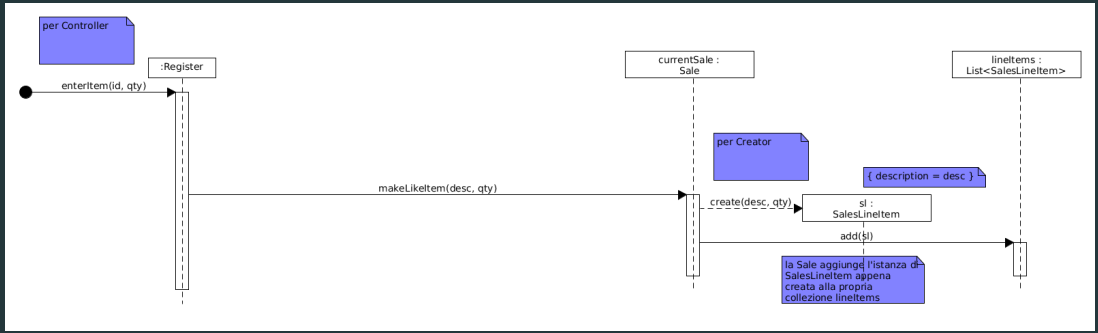
La *Sale* può essere collegata con la *SalesLineItem* appena creata memorizzando la nuova istanza nella sua collezione di righe di articoli.

Pertanto, per *Creator*, viene inviato un messaggio *makeLineItem* a una *Sale* affinché essa crei una *SalesLineItem*. I parametri per il messaggio *makeLineItem* comprendono:

- *ProductDescription* che corrisponde all'*itemID*
- *quantity*

SalesLineItem memorizza tali parametri.

Esempio: enterItem, diagramma di interazione



Esempio: enterItem, terza post-condizione “sli è stata associata con una ProductDescription, in base alla corrispondenza con itemID”

La nuova *SalesLineItem* va collegata a una *ProductDescription* che corrisponde all'*itemID* inserito.

Ciò implica la ricerca di una *ProductDescription* in base a una corrispondenza con *itemID*.

Responsabilità

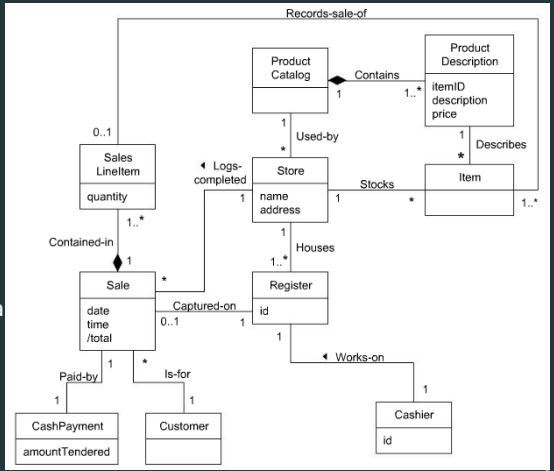
Chi deve essere responsabile della conoscenza di una *ProductDescrition* in base a una corrispondenza con *itemID*?

Esempio: enterItem, terza post-condizione “sli è stata associata con una ProductDescription, in base alla corrispondenza con itemID”

Dal Modello di Dominio, *ProductCatalog* contiene logicamente tutte le *ProductDescription*.

Per *Information Expert*, *ProductCatalog* è un buon candidato per questa responsabilità di ricerca, poiché conosce tutti gli oggetti *ProductDescription*.

La responsabilità può essere rappresentata con un metodo chiamato *getProductDescription*. Il *ProductCatalog* può soddisfare la responsabilità di conoscere gli oggetti *ProductDescription* utilizzando una mappa *descriptions*.



Esempio: enterItem, terza post-condizione “sli è stata associata con una ProductDescription, in base alla corrispondenza con itemID”

Responsabilità

Chi deve inviare il messaggio *getProductDescription* al *ProductCatalog* per cercare una *ProductDescription*?

Esempio: enterItem, terza post-condizione “sli è stata associata con una ProductDescription, in base alla corrispondenza con itemID”

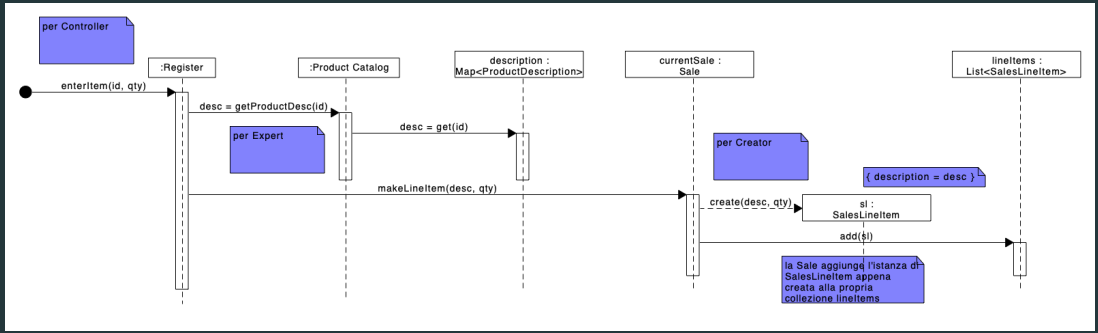
Responsabilità

Chi deve inviare il messaggio *getProductDescription* al *ProductCatalog* per cercare una *ProductDescription*?

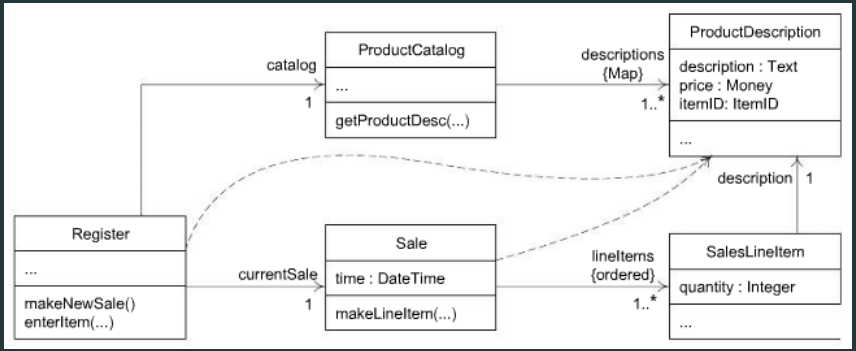
È ragionevole pensare che durante l’avviamento iniziale siano stati creati degli oggetti di lunga vita *Register* e *ProductCatalog* e che l’oggetto *Register* sia stato connesso in modo permanente all’aggetto *ProductCatalog*.

Register può inviare il messaggio *getProductDescription* al *ProductCatalog*.

Esempio: enterItem, diagramma di interazione



Esempio: enterItem, diagramma delle classi



Esempio: endSale

L'operazione di sistema *endSale* avviene quando un cassiere preme un pulsante per indicare la fine dell'inserimento degli articoli in una vendita.

Contratto CO3: endSale

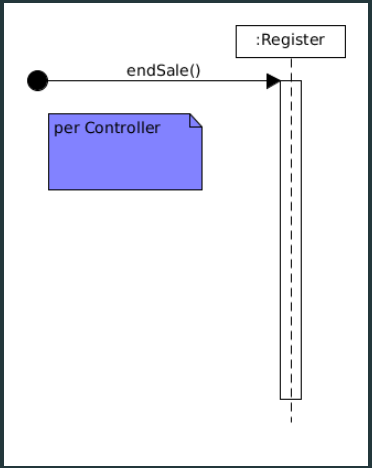
Operazione:	endSale()
Riferimenti:	Casi d'uso: Elabora Vendita
Pre-condizioni:	è in corso una vendita.
Post-condizioni:	– nessuna.

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: endSale, controller

Il pattern *Controller* suggerisce di utilizzare la stessa classe controller per tutte le operazioni di sistema di un caso d'uso; pertanto si continuerà a utilizzare *Register* come controller per l'operazione *endSale* e per le ulteriori operazioni di sistema del caso d'uso *Elabora Vendita*.

Esempio: endSale, diagramma di interazione



Esempio: endSale, post-condizioni

Durante l'analisi a oggetti, per l'operazione di sistema *endSale* non sono state identificate post-condizioni. Dunque questa operazione si configura principalmente come un'interrogazione, per calcolare e visualizzare il *totale di una vendita*, e non anche come una trasformazione.

Esempio: endSale, gestione dello stato del caso d'uso

Attenzione!

A volte si vuole ragionare sullo *stato del caso d'uso*.

Ad esempio, si potrebbe voler impedire l'esecuzione dell'operazione *makeCashPayment* fino a che non è stata eseguita l'operazione *endSale*.

In questo caso, l'operazione *endSale* dovrebbe implicare anche una **trasformazione**, per ricordare che è terminato l'inserimento degli articoli della vendita.

Ad esempio: un attributo *booleano itemEntryComplete* di *Sale*. In questo caso, per *Expert*, il controller Register potrebbe richiedere alla currentSale di impostare questo attributo a *true*.

Ad esempio: si può utilizzare un oggetto "sessione" per tenere traccia dello stato della sessione del caso d'uso.

Nota: questo viene per ora omesso.

Esempio: endSale, mostrare il totale

Scenario principale di successo:

1. Il Cliente arriva...
2. Il Cassiere inizia una nuova vendita.
3. Il Cassiere inserisce il codice identificativo di un articolo.
4. Il Sistema registra la riga di vendita per l'articolo e...
Il Cassiere ripete i passi 3-4 fino a che non indica che ha terminato.
5. Il Sistema mostra il totale con le imposte calcolate.

Nel passo 5, viene mostrato un totale.

In virtù del principio di separazione Modello-Vista, nella progettazione dello strato del dominio non ci si deve preoccupare di progettare come sarà visualizzato il totale della vendita, ma si deve assicurare che il totale sia noto.

Si noti che, al momento, nessuna classi di progetto conosce il totale della vendita, per cui è necessario creare un progetto di interazione di oggetti per soddisfare questo requisito.

Esempio: endSale, mostrare il totale

Responsabilità

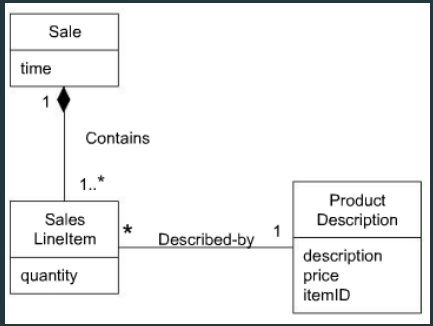
Chi deve essere responsabile di conoscere il totale della vendita?

Esempio: endSale, mostrare il totale

Responsabilità

Chi deve essere responsabile di conoscere il totale della vendita?

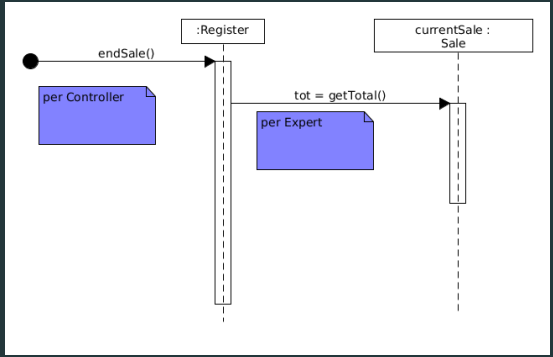
Secondo Expert, *Sale* è la migliore candidata: occorre conoscere tutte le istanze *SalesLineItem* della vendita e la somma dei relativi totali parziali. Un'istanza *Sale* li contiene, è un *esperto delle informazioni* per questo compito. Nota: quello riportato è il Modello di Dominio.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: endSale, mostrare il totale

Si assegna la responsabilità a *Sale* di conoscere il suo totale, esprimendo questa responsabilità con un metodo chiamato *getTotal*.
Dal modello di dominio al modello di progetto: un diagramma delle classi con l'indicazione dei metodi (salto rappresentazionale basso).



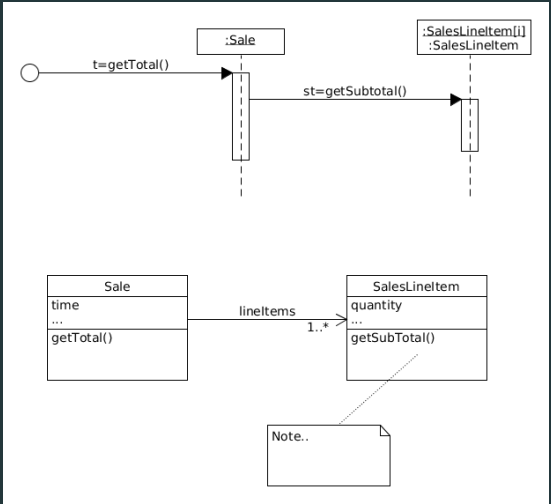
Esempio: endSale, mostrare il totale

Quali informazioni sono necessarie per determinare il totale parziale per una riga di vendita per un articolo? *SalesLineItem.quantity* e *ProductDescription.price*, ovvero un oggetto *SalesLineItem* conosce la sua *quantità* e la *ProductDescription* ad esso associata, pertanto per Expert, *SalesLineItem* deve determinare il totale parziale, è l'*esperto delle informazioni*.

Classe di progetto	Responsabilità
Sale	sa calcolare il totale della vendita; conosce le righe di vendita della vendita
SalesLineItem	sa calcolare il totale parziale della riga di vendita; conosce il prodotto della riga di vendita
ProductDescription	conosce il prezzo del prodotto

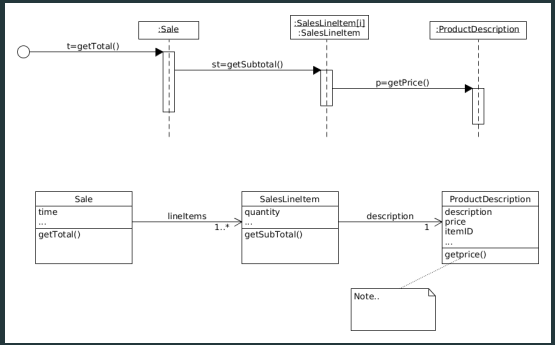
Esempio: endSale, mostrare il totale

Quindi *Sale* deve inviare messaggi *getSubtotal* a ciascuna delle sue *SalesLineItem* e sommare i risultati.



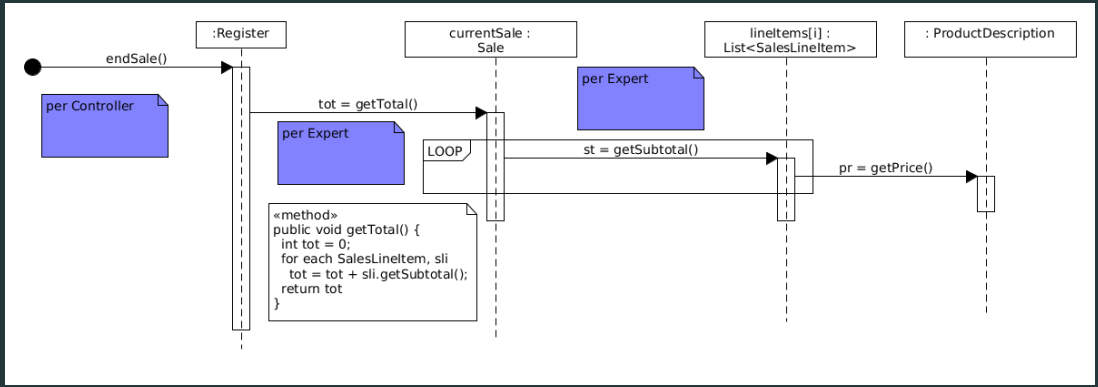
Esempio: endSale, mostrare il totale

Quindi *Sale* deve inviare messaggi *getSubtotal* a ciascuna delle sue *SalesLineItem* e sommare i risultati. Per soddisfare la sua responsabilità, *SalesLineItem* deve conoscere il prezzo del prodotto a cui si riferisce. *SalesLineItem* invia a *ProductDescription* un messaggio *getPrice* chiedendogli il prezzo del prodotto.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: endSale, diagramma di interazione



Esempio: makeCashPayment

L'operazione di sistema *makeCashPayment* avviene quando un cassiere inserisce l'importo in contanti offerto dal cliente per il pagamento.

Contratto CO4: makeCashPayment

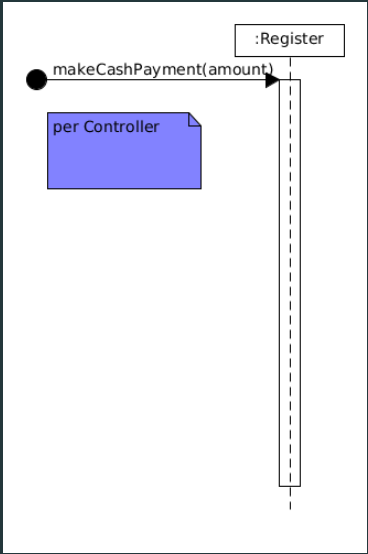
Operazione:	makeCashPayment(amount: Money)
Riferimenti:	Casi d'uso: Elabora Vendita
Pre-condizioni:	è in corso una vendita s.
Post-condizioni:	– è stata creata un'istanza p di CashPayment. – p è stata associata con la Sale corrente s. – p.amountTendered è diventato amount. – la Sale corrente s è stata associata con lo Store (s è stata aggiunta al registro storico delle vendite completate).

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Esempio: makeCashPayment, controller

Il pattern *Controller* suggerisce di utilizzare la stessa classe controller per tutte le operazioni di sistema di un caso d'uso; pertanto si continuerà a utilizzare *Register* come controller per l'operazione *makeCashPayment* e per le ulteriori operazioni di sistema del caso d'uso *Elabora Vendita*.

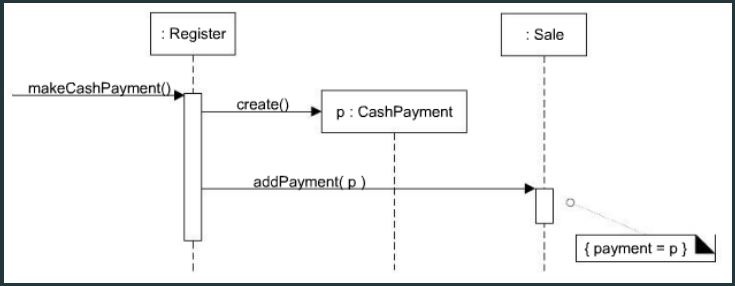
Esempio: makeCashPayment, diagramma di interazione



Esempio: makeCashPayment, prima, seconda e terza post-condizione "è stata creata un'istanza p di CashPayment", "p è stata associata con la Sale corrente s" e "p.amountTendered è diventato amount"

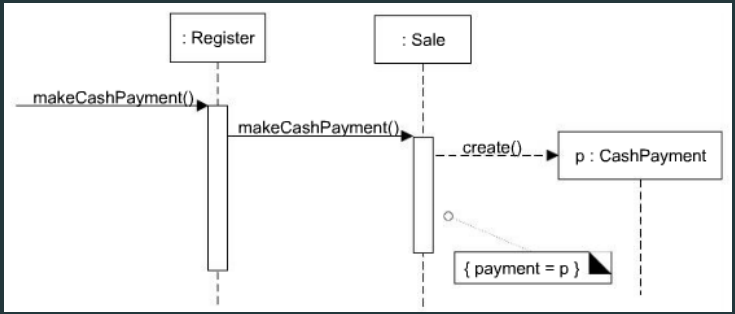
Prima soluzione:

- con il pattern Creator si sceglie *Register* come creatore di *Payment*, suggerito dalle responsabilità nel "mondo reale" (registra i pagamenti)
- dunque uso metodo *addPayment(p)* per comunicare con *Sale*



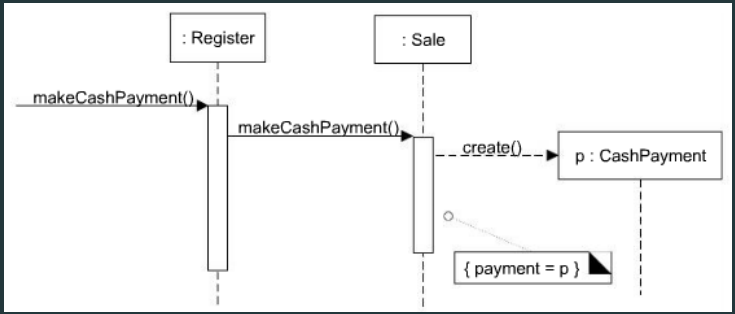
Esempio: makeCashPayment, prima, seconda e terza post-condizione “è stata creata un’istanza p di CashPayment”, “p è stata associata con la Sale corrente s” e “p.amountTendered è diventato amount”

Seconda soluzione: per Expert, *Sale* conosce *Payment*, la creazione di *CashPayment* la può fare la *Sale*.



Esempio: makeCashPayment, prima, seconda e terza post-condizione “è stata creata un’istanza p di CashPayment”, “p è stata associata con la Sale corrente s” e “p.amountTendered è diventato amount”

Seconda soluzione: per Expert, *Sale* conosce *Payment*, la creazione di *CashPayment* la può fare la *Sale*.



Esempio: makeCashPayment, prima, seconda e terza post-condizione “è stata creata un’istanza p di CashPayment”, “p è stata associata con la Sale corrente s” e “p.amountTendered è diventato amount”

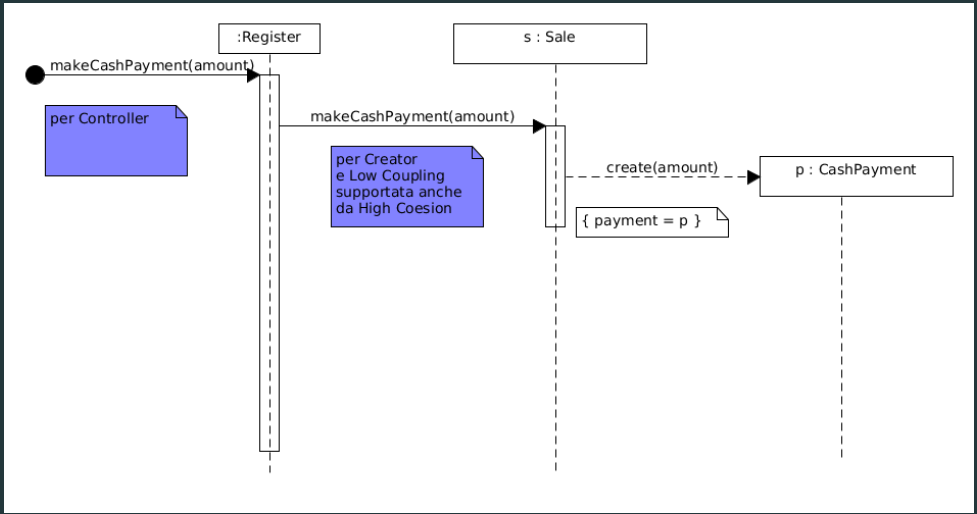
Valutazione con Low Coupling:

- Per Low Coupling la seconda soluzione va preferita poiché mantiene un accoppiamento complessivo più basso (si veda discussione su *Low Coupling*)

Valutazione con High Coesion:

- La prima soluzione significa che il *Register* si assume non solo la responsabilità di ricevere l’operazione di sistema *makeCashPayment*, ma anche parte della responsabilità di soddisfarla
- Se si continua a rendere la classe **Register** responsabile di eseguire una parte del lavoro o l’intero lavoro relativo a sempre più operazione di sistema, essa diventerà sempre più carica di compiti, e diventerà **non coesa**
- Per High Coesion la seconda soluzione va preferita

Esempio: makeCashPayment, diagramma di interazione



Esempio: makeCashPayment, quarta post-condizione “la Sale corrente s è stata associta con lo Store (s è stata aggiunta al registro storico delle vendite completate)”

I requisiti affermano che la vendita deve essere inserita in un registro (log) storico delle vendite completate.

Per Expert e l'analisi del Modello di Dominio suggerisce che lo *Store* conosca e registri le *Sale* completate.

Un'alternativa è l'introduzione di un *SalesLedger*, un libro mastro (nuovo concetto).

Esempio: makeCashPayment, quarta post-condizione “la Sale corrente s è stata associta con lo Store (s è stata aggiunta al registro storico delle vendite completate)”

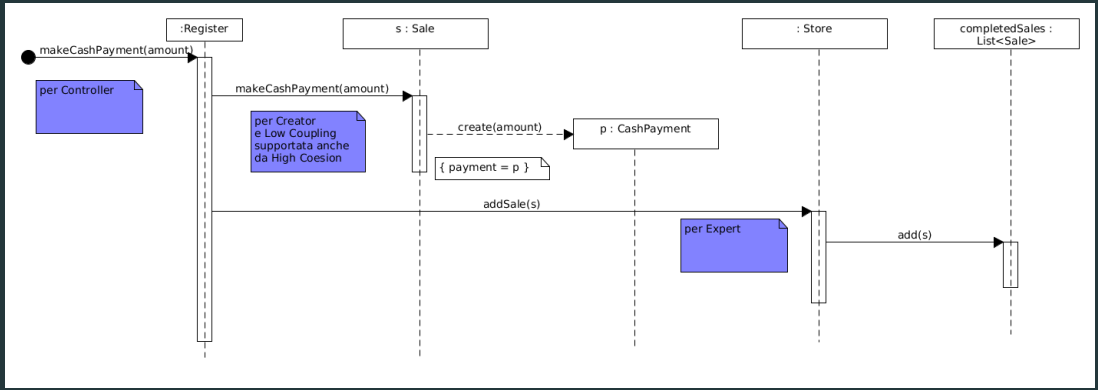
I requisiti affermano che la vendita deve essere inserita in un registro (log) storico delle vendite completate.

Per Expert e l'analisi del Modello di Dominio suggerisce che lo *Store* conosca e registri le *Sale* completate.

Un'alternativa è l'introduzione di un *SalesLedger*, un libro mastro (nuovo concetto).

Valutazione con High Coesion: la scelta di *Store* per questa responsabilità è accettabile se *Store* ha poche responsabilità, mentre l'utilizzo di un oggetto *SalesLedger* ha senso man mano che il progetto cresce e *Store* diventa poco coeso.

Esempio: makeCashPayment, diagramma di interazione



Esempio: makeCashPayment, calcolo del resto

In virtù del principio di Separazione Modello-Vista, non ci si dovrebbe preoccupare del modo in cui il resto sia visualizzato o stampato, ma ci si deve assicurare che sia conosciuto.

Al momento nessuna classe conosce il resto, per cui è necessario creare un progetto di interazioni di oggetti che soddisfi questo requisito.

Responsabilità

Chi è responsabile di conoscere il resto?

Esempio: makeCashPayment, calcolo del resto

In virtù del principio di Separazione Modello-Vista, non ci si dovrebbe preoccupare del modo in cui il resto sia visualizzato o stampato, ma ci si deve assicurare che sia conosciuto.

Al momento nessuna classe conosce il resto, per cui è necessario creare un progetto di interazioni di oggetti che soddisfi questo requisito.

Responsabilità

Chi è responsabile di conoscere il resto?

Per calcolare il resto è necessario conoscere il totale della vendita e l'importo in contanti offerto.

Per Expert, *Sale* e *CashPayment* sono esperti parziali (rispettivamente esperto del totale della vendita ed esperto dell'importo in contanti offerto).

Esempio: makeCashPayment, prima, seconda e terza post-condizione “è stata creata un’istanza p di CashPayment”, “p è stata associata con la Sale corrente s” e “p.amountTendered è diventato amount”

Valutazione con Low Coupling:

- Se il *CashPayment* è il responsabile principale per conoscere il resto, necessita della visibilità nei confronti della *Sale*, per chiedere alla *Sale* il suo totale. Poiché al momento non conosce la *Sale*, aumenterebbe l’accoppiamento complessivo
- Se la *Sale* è il responsabile principale per conoscere il resto, necessita della visibilità nei confronti del *CashPayment*, per chiedergli l’importo in contanti consegnato. Dal momento che la *Sale* ha già la visibilità nei confronti del *CashPayment* (era il suo creatore, si vedano i lucidi precedenti), questo approccio non aumenta l’accoppiamento complessivo.[1]

[1] Per determinare gli accoppiamenti si contano le dipendenze che sono direzionali.

Esempio: makeCashPayment, prima, seconda e terza post-condizione “è stata creata un’istanza p di CashPayment”, “p è stata associata con la Sale corrente s” e “p.amountTendered è diventato amount”

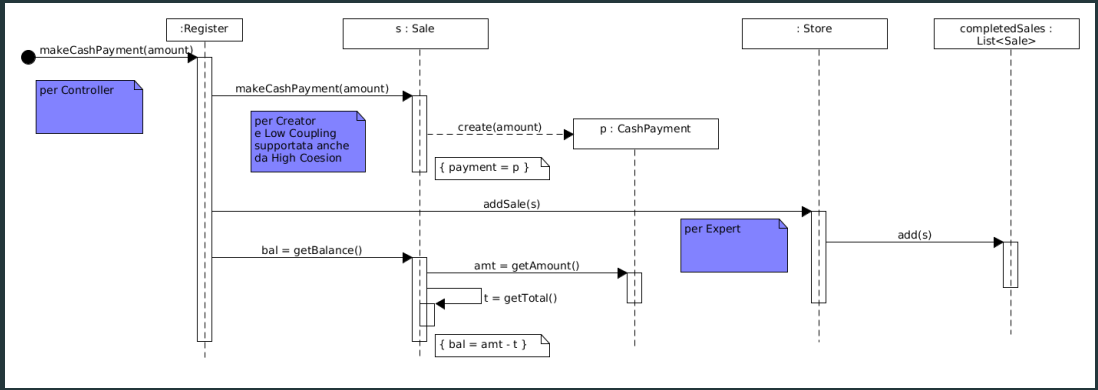
Valutazione con Low Coupling:

- Se il *CashPayment* è il responsabile principale per conoscere il resto, necessita della visibilità nei confronti della *Sale*, per chiedere alla *Sale* il suo totale. Poiché al momento non conosce la *Sale*, aumenterebbe l’accoppiamento complessivo
- Se la *Sale* è il responsabile principale per conoscere il resto, necessita della visibilità nei confronti del *CashPayment*, per chiedergli l’importo in contanti consegnato. Dal momento che la *Sale* ha già la visibilità nei confronti del *CashPayment* (era il suo creatore, si vedano i lucidi precedenti), questo approccio non aumenta l’accoppiamento complessivo.[1]

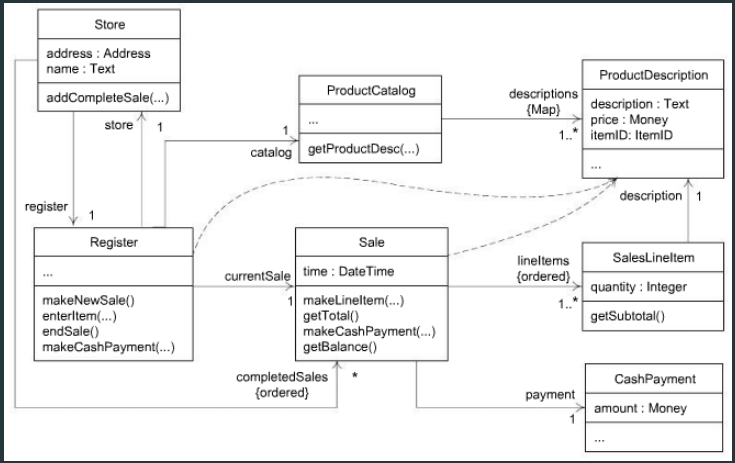
Il responsabile è quindi Sale.

[1] Per determinare gli accoppiamenti si contano le dipendenze che sono direzionali.

Esempio: makeCashPayment, diagramma di interazione



Esempio: DCD finale di NextGen per l'iterazione 1, strato di dominio



Collegare lo strato UI allo strato del dominio e inizializzazione

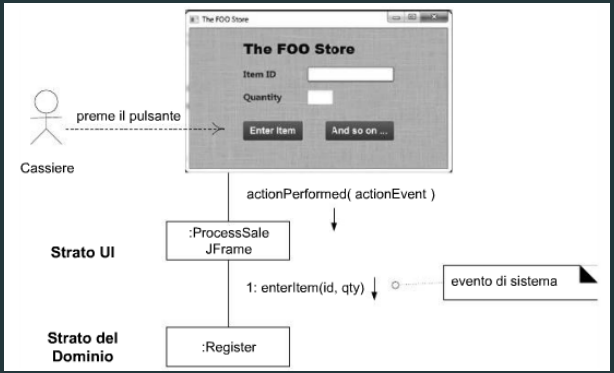
Collegare lo strato UI allo strato di dominio

Scelte comuni per dare agli oggetti dello strato UI la visibilità nei confronti degli oggetti dello strato di dominio sono:

- Un oggetto inizializzatore chiamato dal metodo iniziale dell'applicazione che crea sia un oggetto UI che un oggetto di dominio e passa l'oggetto di dominio all'oggetto UI
- Un oggetto UI che recupera l'oggetto di dominio da una sorgente nota, come un oggetto factory responsabile della creazione di oggetti di dominio

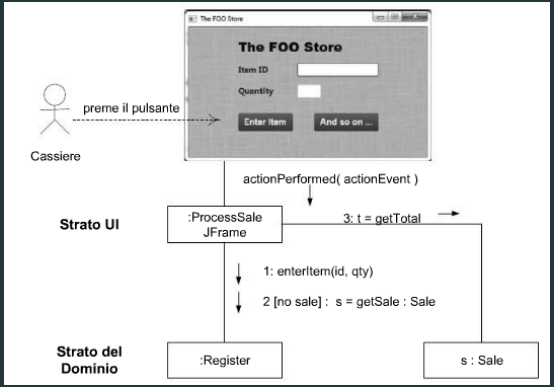
Collegare lo strato UI allo strato di dominio

Nel nostro esempio, una volta che l'oggetto UI ha una connessione all'istanza di Register, può inoltrare ad essa i messaggi per gli eventi di sistema.



Collegare lo strato UI allo strato di dominio

Per visualizzare il totale corrente (o altra informazione relativa alla vendita): un oggetto UI chiede il riferimento all'oggetto Sale corrente, e invia direttamente i messaggi alla Sale. Alternativa, si passa da Register.



Caso d'uso d'avviamento o inizializzazione

La maggior parte dei sistemi hanno un caso d'uso di Avviamento (Start Up), implicito oppure esplicito, e alcune operazioni di sistema iniziali relative all'avvio dell'applicazione.

Importante

La progettazione dell'inizializzazione va eseguita per ultima

Un idioma di progettazione comune è quello di creare un **oggetto di dominio iniziale** o un insieme di oggetti di dominio iniziali di pari grado che sono i primi oggetti software di "dominio" creati.

Caso d'uso d'avviamento o inizializzazione

Questa creazione può avvenire in modo esplicito nel metodo iniziale *main* o in un oggetto *Factory* chiamato dal metodo *main*.

```
public class Main {
   public static void main( String[] args ) {

      /* Store è l'oggetto di dominio iniziale.
       * Lo Store crea degli altri oggetti di dominio. */

      Store store = new Store();

      Register register = store.getRegister();

      ProcessSaleJFrame frame = new ProcessSaleJFrame( register );
      ...
   }
}
```

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Nota: `register` è il controller GRASP.

Caso d'uso d'avviamento o inizializzazione

Esempio per Elabora Vendita. Si è scelto Store come l'oggetto iniziale.