Modelo de Design: Concretizando Use-Cases com Padrões GRASP

Based on C. Larman, 2002

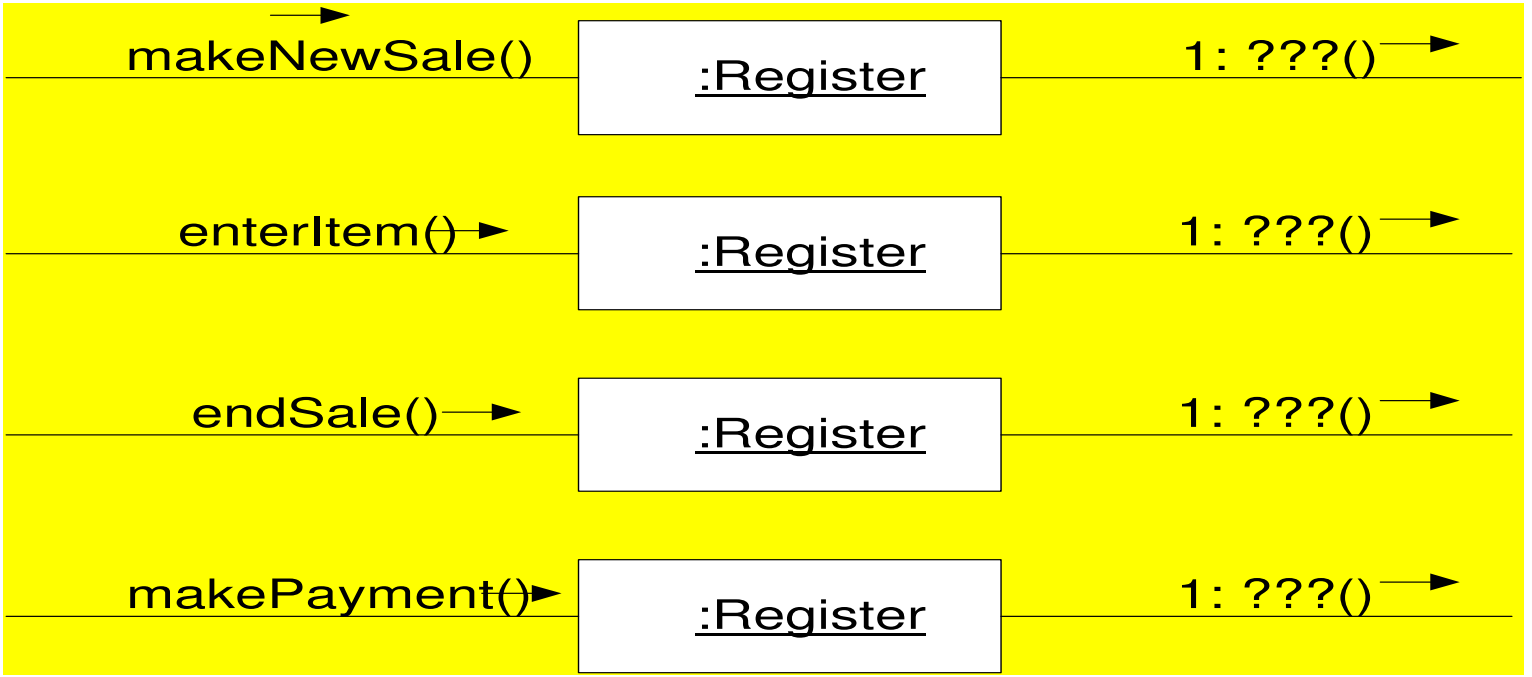
Uma concretização de use-case

- Descreve como um UC particular é tratado dentro do modelo de design, em termos da colaboração entre objetos
  - um designer pode descrever o design de um ou mais cenários de um caso de uso
  - Cada um deles é chamado uma concretização de um use-case
- É um conceito do UP para nos lembrar da conexão entre requisitos expressos como uc e o design de objetos que satisfaz os requisitos.

Based on C. Larman, 2002

Se diagramas de colaboração forem usados para ilustrar a concretização de uc

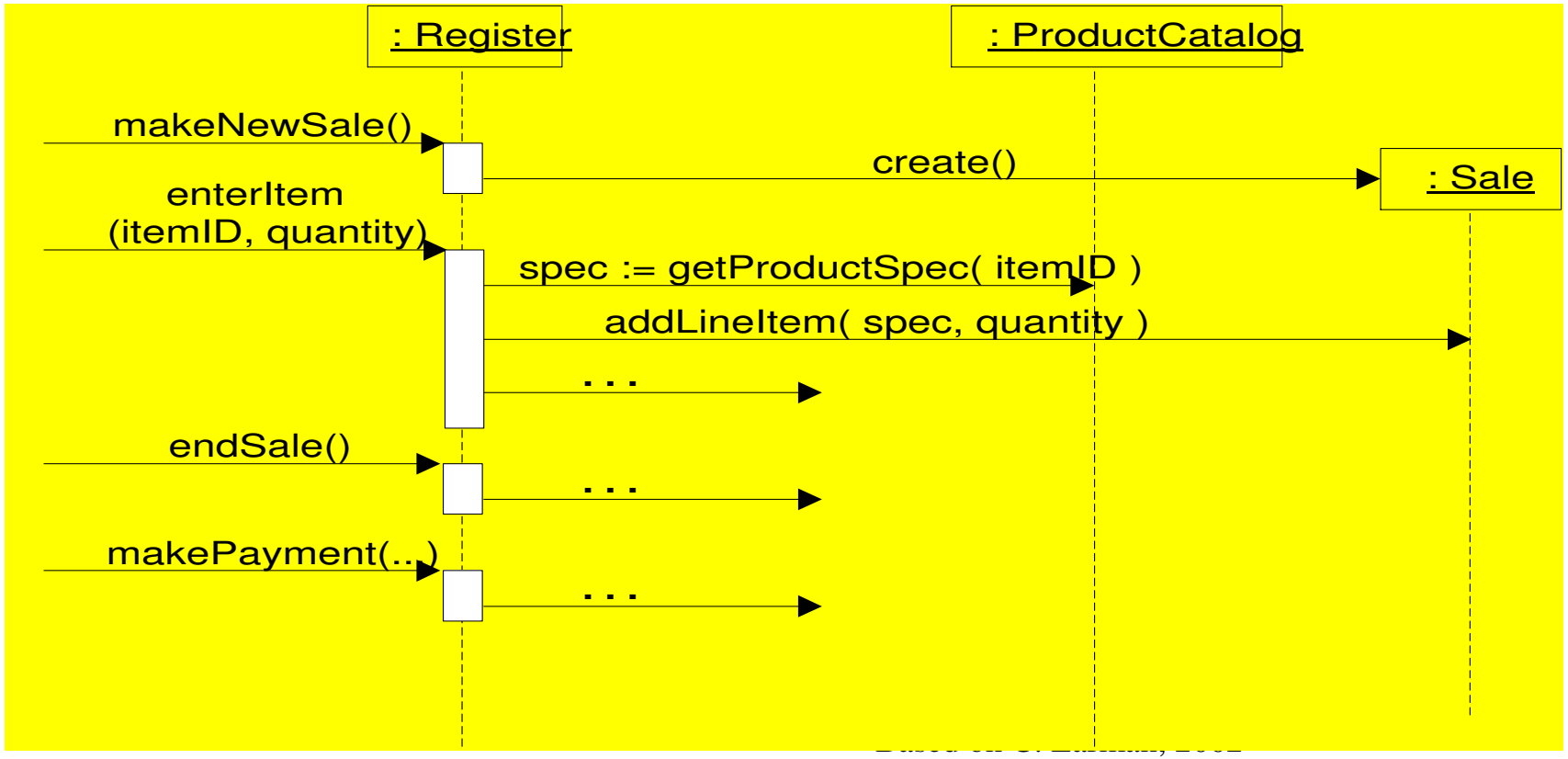
Eles devem mostrar o tratamento de cada mensagem-evento do sistema



Based on C. Larman, 2002

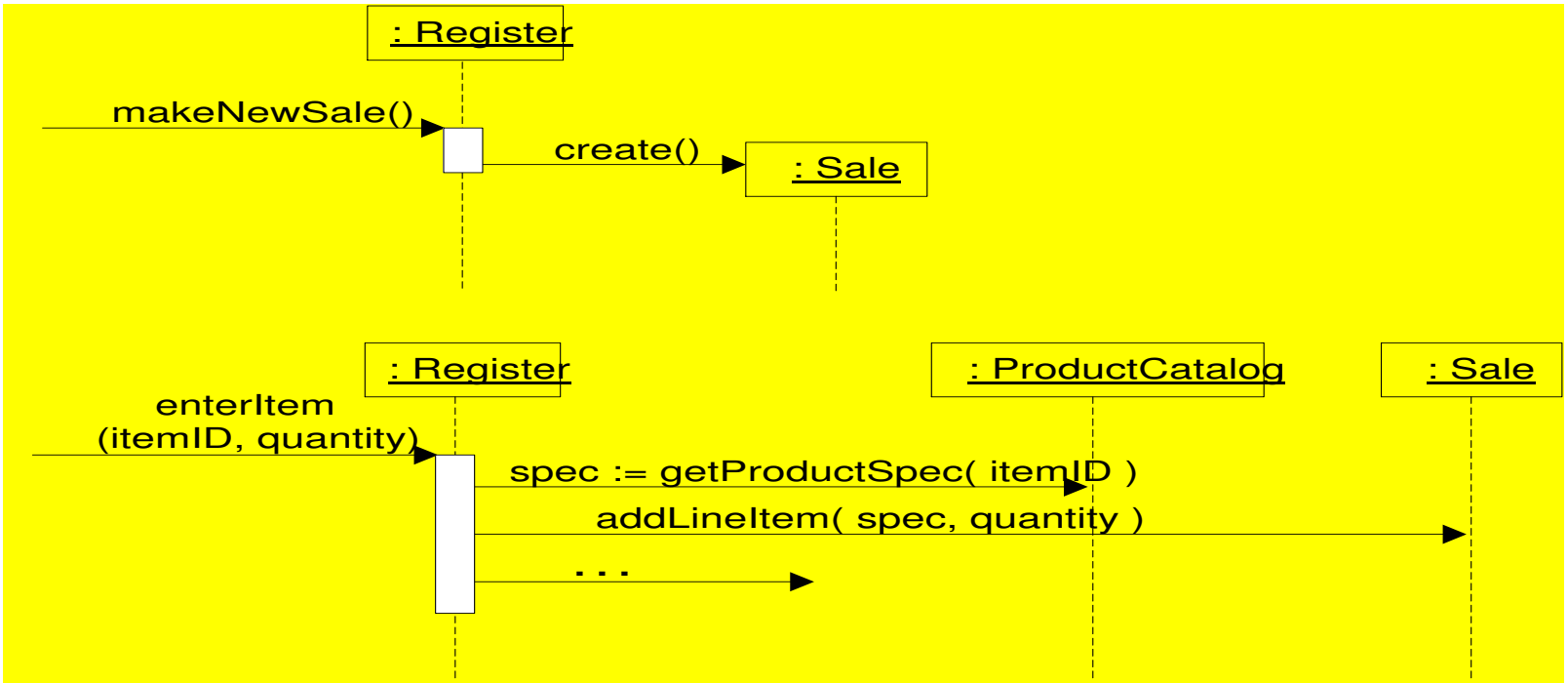
Se diagramas de seqüência forem usados,

Todas as mensagens-evento do sistema devem ser mostradas no mesmo diagrama



Se o diagrama de seqüência é muito complexo…

Pode ser usado um diagrama de seqüência para cada mensagem-evento do sistema



Based on C. Larman, 2002

Contratos e Concretização de Use-Case

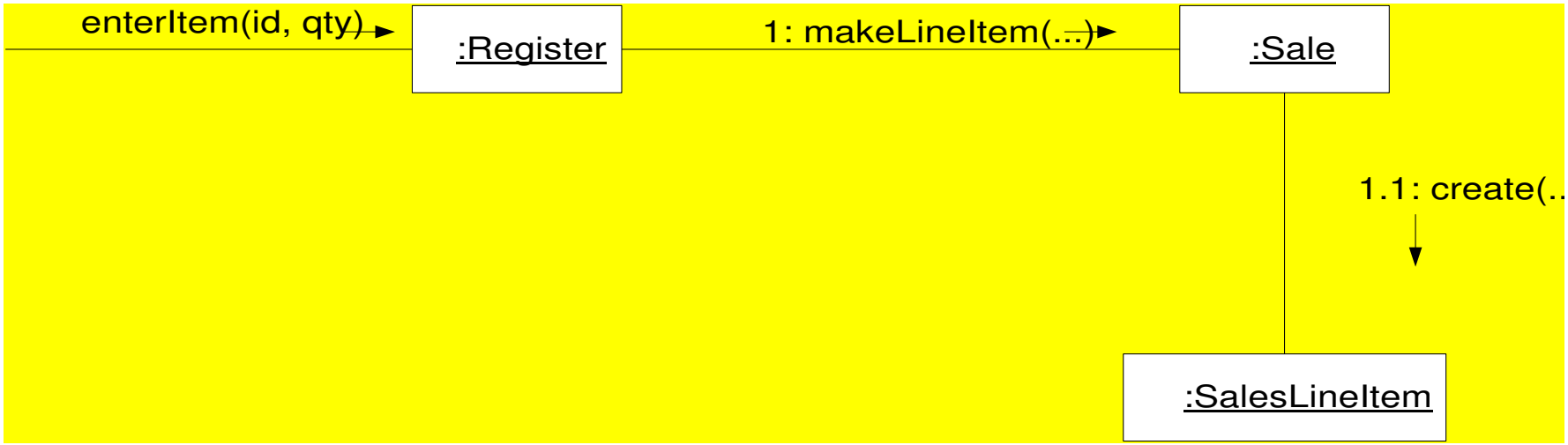
- É possível criar as concretizações do uc diretamente do texto do uc

Contract CO2: enteritem
Operation: enteritem(itemID: ItemID, quantity: integer)
Cross References: Use Cases: Process Sale
Preconditions: There is a sale underway.
Postconditions: A SalesLineItem instance sli was created (instance creation)
...

Based on C. Larman, 2002

Diagrama de Interação Parcial

Satisfazendo os requisitos



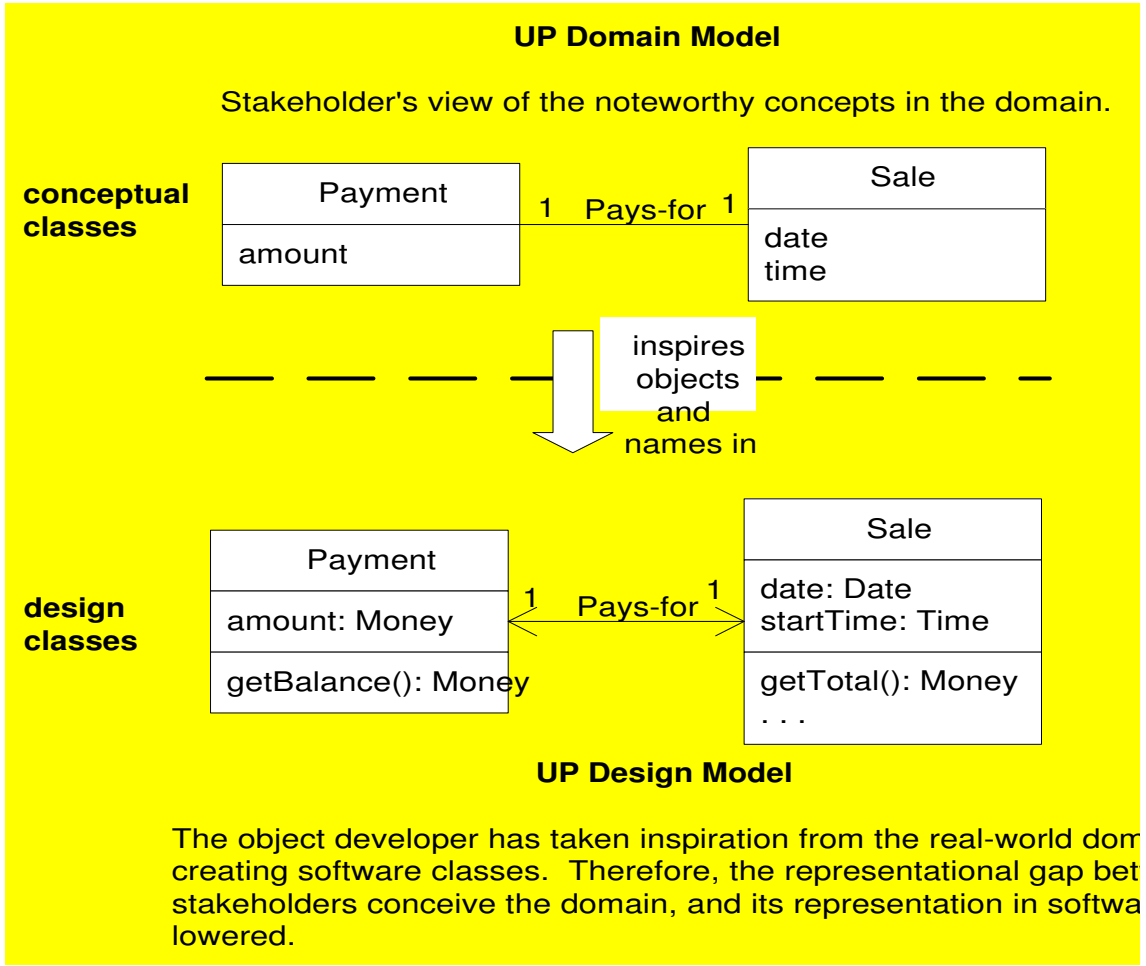
Based on C. Larman, 2002

O Modelo de Domínio e a Concretização do Use-Case

- Alguns objetos do sftw que interagem via mensagens no diagrama de interação são inspirados no DM
  - e.g. *Sale* classe conceitual e *Sale* classe de design
- A atribuição de responsabilidades usando GRASP vem em parte do DM
- Isto cria um design com um gap representacional menor entre o sftw design e os conceitos do domínio real

Based on C. Larman, 2002

Diminuindo o gap representacional



Based on C. Larman, 2002

Concretizações de Use-Case para POS

- Não há magica na criação de diagramas de interação bem desenhados
  - use GRASP e princípios para justificar
- Object Design: makeNewSale
  - Uma operação do sistema ocorre quando um caixa pede para iniciar uma nova venda, depois que um cliente chegou com coisas a comprar.

Based on C. Larman, 2002

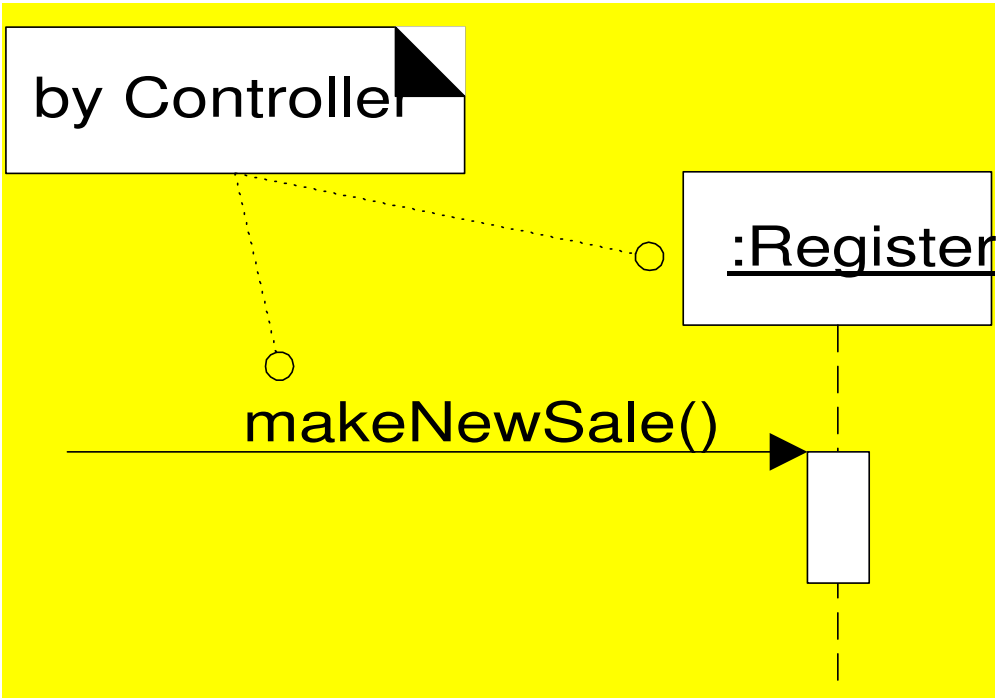
Contract CO1: makeNewSale

- **Operation:** makeNewSale()
- **Cross Reference:** Use Cases: Process Sale
- **Preconditions:** none
- **Postconditions:**
  - A Sale instance s was created (instance creation)
  - s was associated with the Register (association formed)
  - attributes of s were initialised

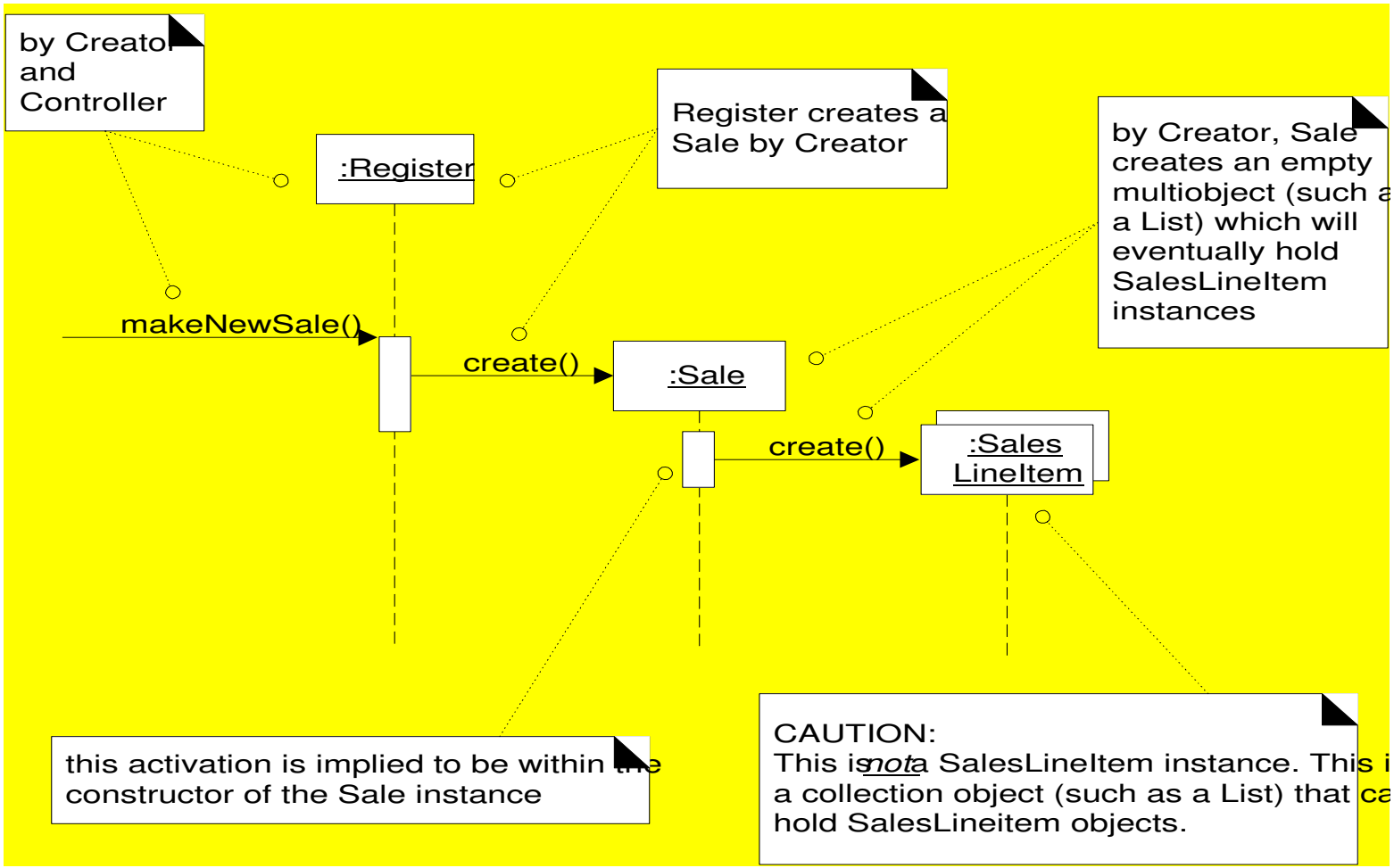
Based on C. Larman, 2002

Primeira escolha de design:

Envolve escolher o *controller* para a mensagem-evento do sistema **makeNewSale**



Criando uma Nova *Sale*



Based on C. Larman, 2002

Object Design: enterItem

- A operação de sistema ***enterItem*** ocorre quando um caixa entra o itemID e (opcionalmente) a quantidade de algo a ser comprado

Contract CO2: enterItem

Operation: enteritem(itemID, quantity: integer)

Cross References: Use Cases: Process Sale

Preconditions: There is an underway sale.

Postconditions:

A SalesLineItem instance sli was created (instance creation)

sli was associated with the current Sale (association formed)

sli.quantity became quantity (attribute modification)

sli was associated with a ProductSpecification, based on itemID match (association formed)

Based on C. Larman, 2002

Escolhas de Design:

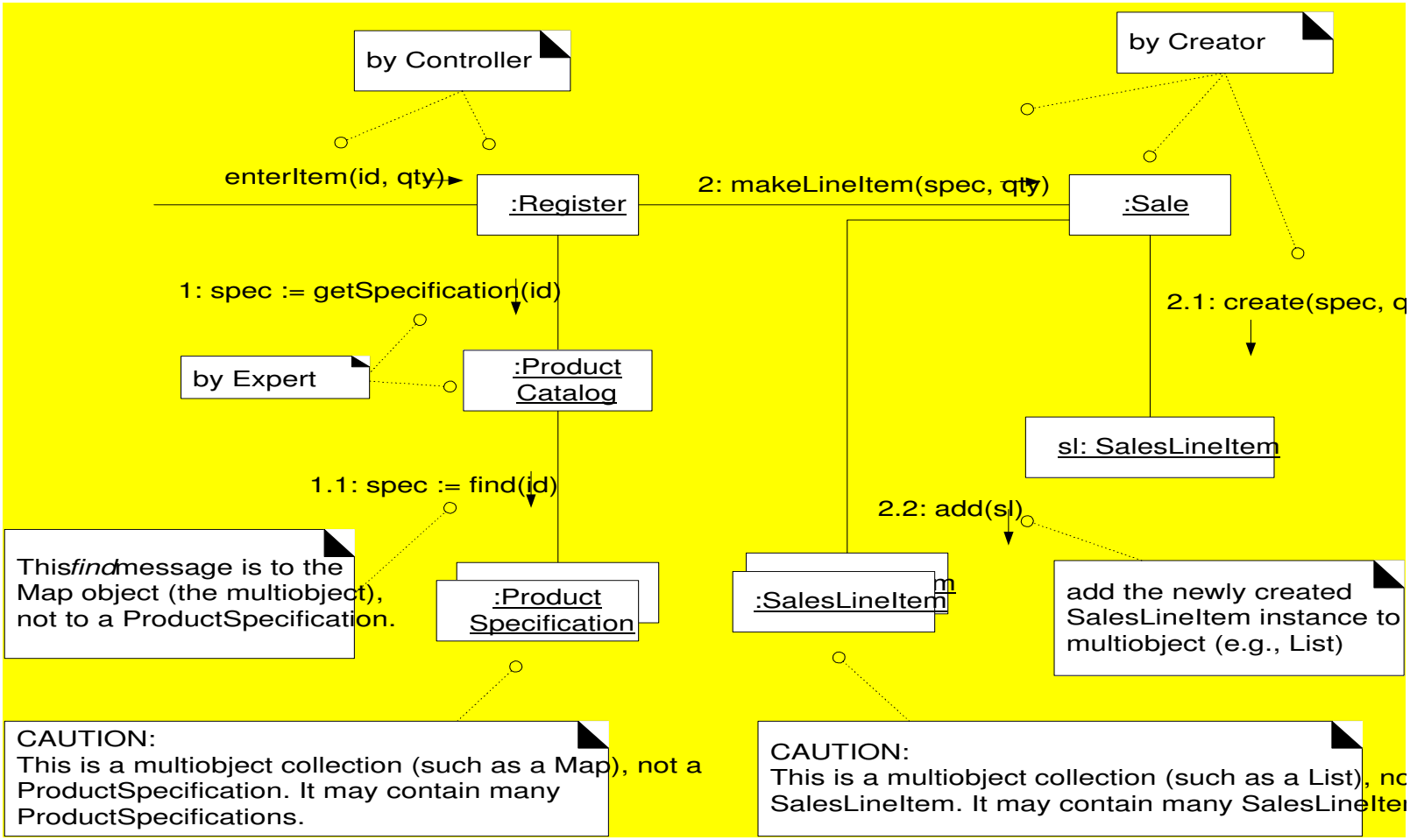
- Escolher a classe controller para lidar com a responsabilidade pela mensagem-operação **enterItem**
  - Com base no padrão Controller use Register como controladora
- Mostrar Descrição de Item e Preço?
  - Com base em umprincípio de design chamado **Model-View Separation**, não é responsabilidade de non-GUI objetos (ex. Register ou Sale) envolver-se em tarefas de output.
    - Tudo que é requerido é a informação a ser conhecida

- Criando um **New SalesLineItem**
  - Inspirando-se no domínio, pelo **Creator**, uma mensagem **makeLineItem** é enviada para uma **Sale** para que crie uma **SalesLineItem**. **Sale** cria **SalesLineItem** e armazena as novas instâncias em sua coleção permanente.
- Encontrando um **ProductSpecification**
  - É necessário buscar a **ProductSpecification**, com base no **itemID** match
  - Quem deveria ser responsável por conhecer a **ProductSpecification**, com base no **itemID** match?
  - Pelo Padrão Information Expert, quem conhece sobre todos os objetos **ProductSpecification**?
  - Analisando o DM, é o **ProductCatalog**
  - Isto pode ser implementado com um método **getSpecification**

- Visibilidade para um **ProductCatalog**
  - Quem deveria enviar a mensagem **getSpecification** para o **ProductCatalog** para perguntar sobre um **ProductSpecification**?
  - Assumimos que instancias de **Register** e **ProductCatalog** foram criadas durante o UC StartUp e que há uma conexão permanente de **Register** para **ProductCatalog**
  - Visibilidade é a habilidade de um objeto "enxergar" ou ter uma referência para outro objeto

Based on C. Larman, 2002

O diagrama de interação **enterItem**



Based on C. Larman, 2002

Object Design: endSale

- A operação de sistema **endSale** ocorre quando um caixa pressiona o botão indicando o fim de uma venda

Contract CO3: endSale

Operation: endSale()

Cross References: Use Cases: ProcessSale

Preconditions: There is an underway sale

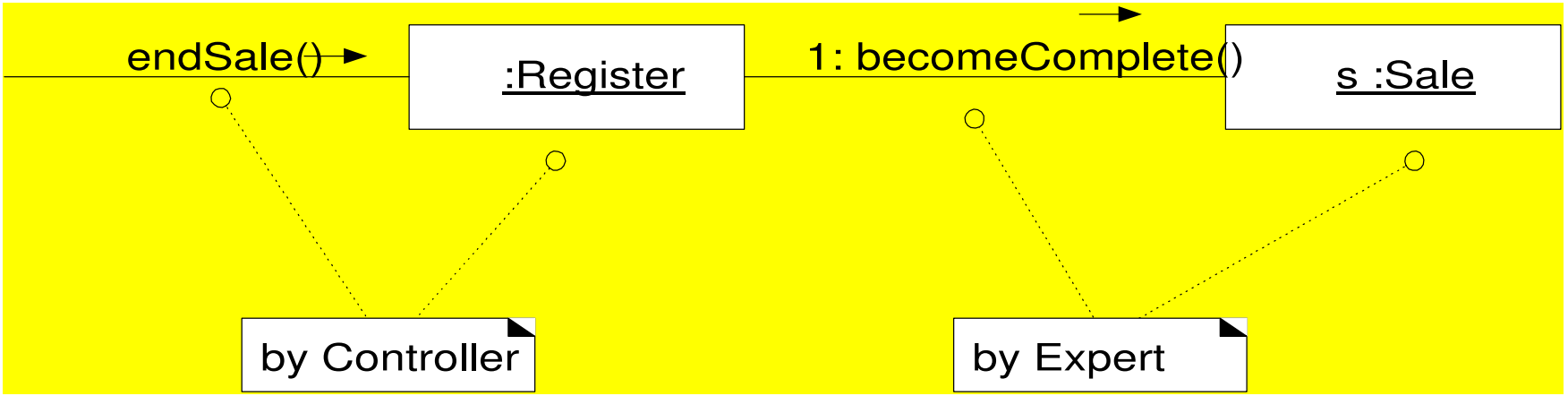
PostConditions: Sale.isComplete became true (attribute modification)

Based on C. Larman, 2002

Escolhendo a Classe *Controller*

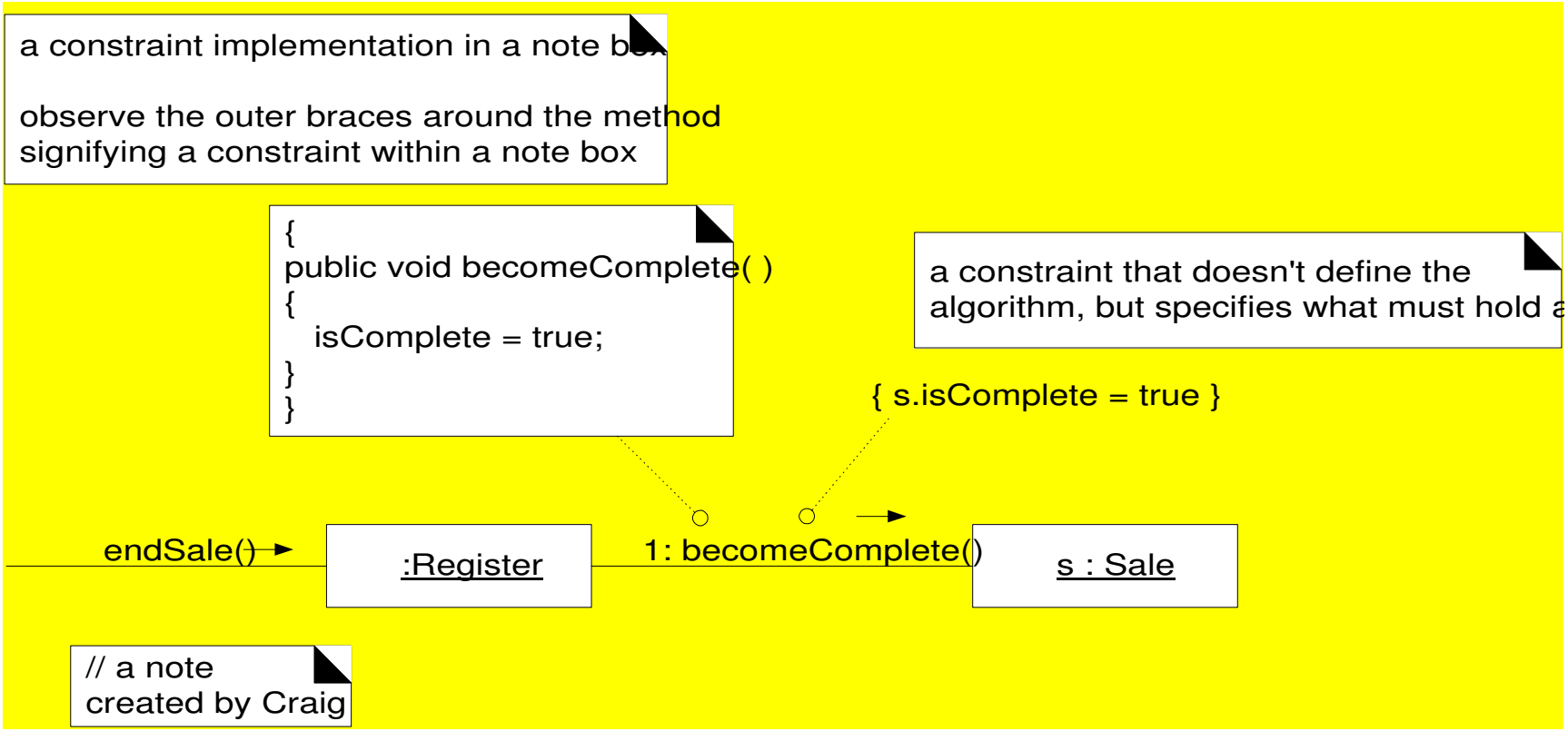
- Responsabilidade pela mensagem operação de sistema **endSale**
  - Baseada no padrão Controller [GRASP], use Register como controller [como para enterItem]
- Mudando o atributo Sale.isComplete
  - Quem deveria ser responsável?
  - Pelo padrão Expert [não é um problema de controle ou criação]
  - Deveria ser **Sale** pq ela mantém o atributo isComplete

Entrada da *completion of item*



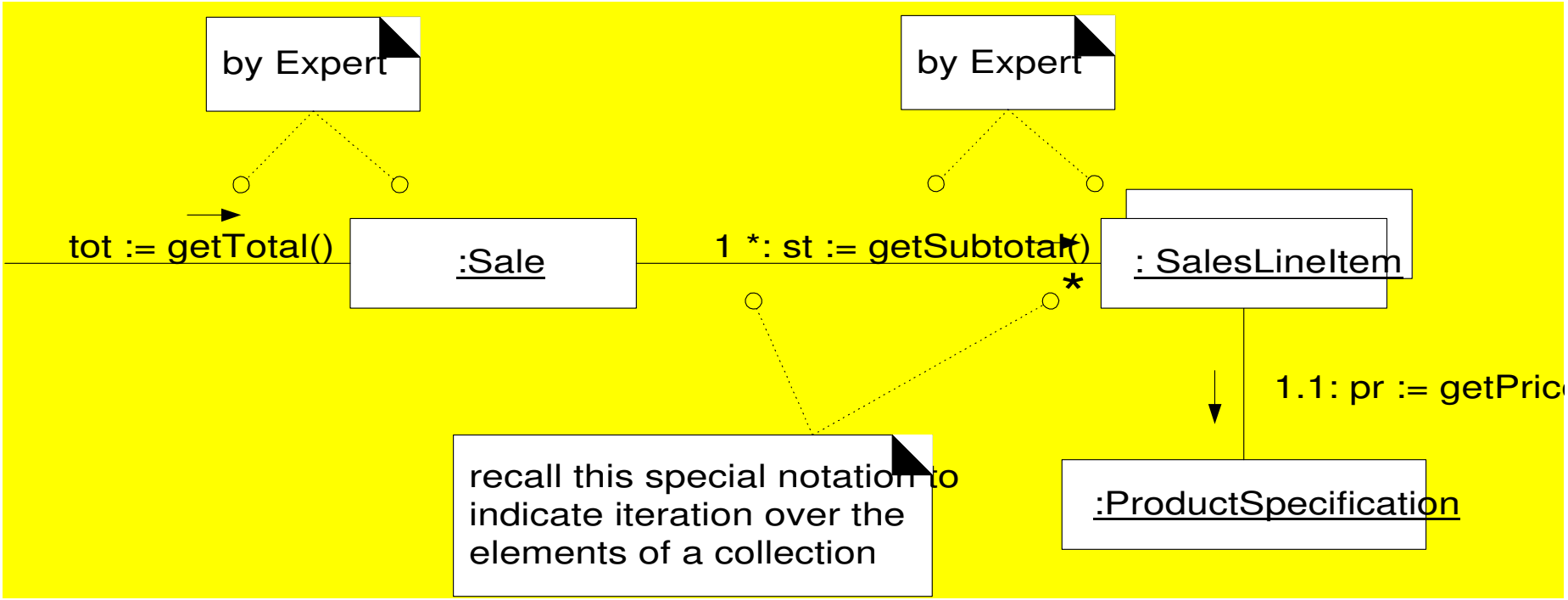
Based on C. Larman, 2002

Notação UML para *Constraints, Notes, and Algorithms*



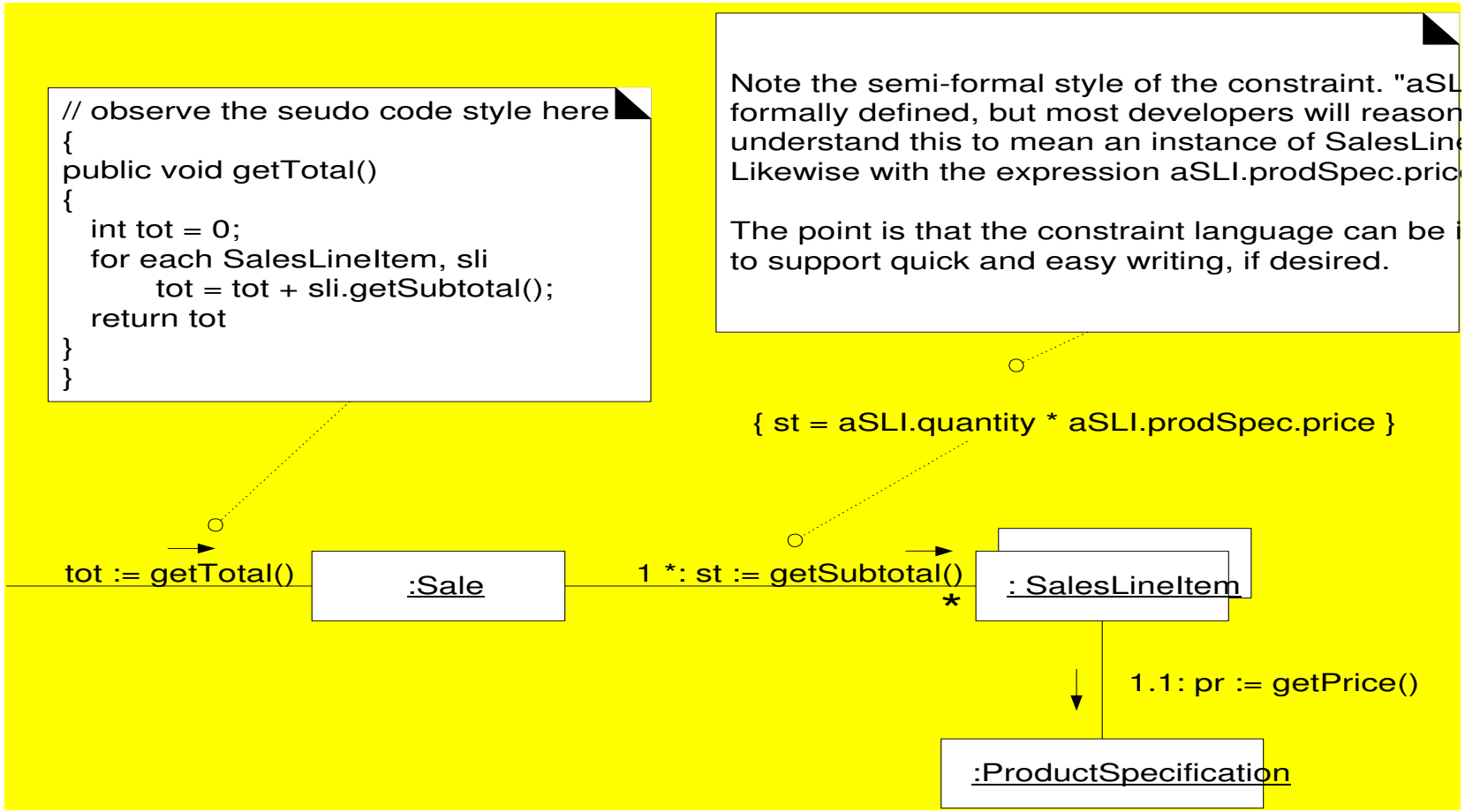
Based on C. Larman, 2002

Diagrama de Interação para Sale--getTotal



Based on C. Larman, 2002

Detalhes de getTotal and getSubtotal através de *algorithm notes and constraints*



Based on C. Larman, 2002

Object Design: makePayment

- A operação de sistema **makePayment** ocorre quando uma caixa entra a quantidade total de cash para pagamento

Contract CO4: makePayment

Operation: makePayment(amount: Money)

Cross References: Use Cases: Process Sale

Preconditions: There is an underway sale

Postconditions: A Payment instance p was created

p.amountTendered became amount

p was associated with the current Sale

the current sale was associated with the Store

Based on C. Larman, 2002

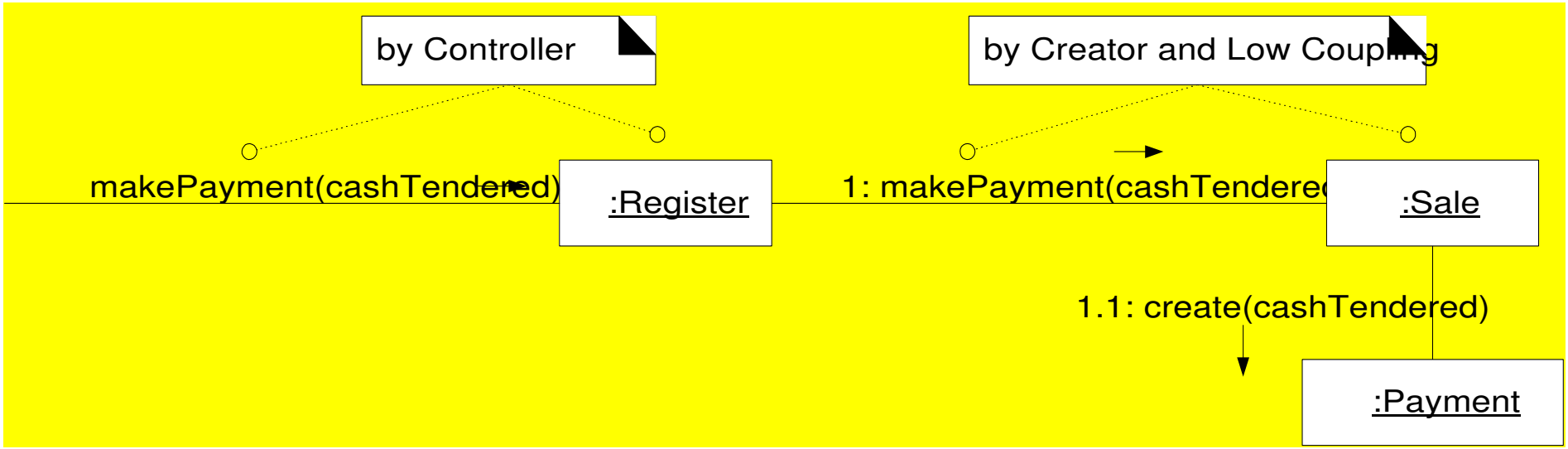
Criando **Payment**

- Quem grava, agrega, usa mais de perto, ou contém um Payment?
  - **Register** registra logicamente um **Payment**
  - Um **Sale** sftw usa mais de perto um **Payment**
- Quem é o Information Expert com respeito a inicialização de dados?
  - **Register** é o controller que recebe a operação do sistema **makePayment**, de forma que ela terá a quantia
    - **Register** é novamente candidata

Based on C. Larman, 2002

Diagrama de Interação Register--makePayment

O **Payment** foi criado associado com **Sale** [lower coupling in Register], e seu amountTendered foi inicializado

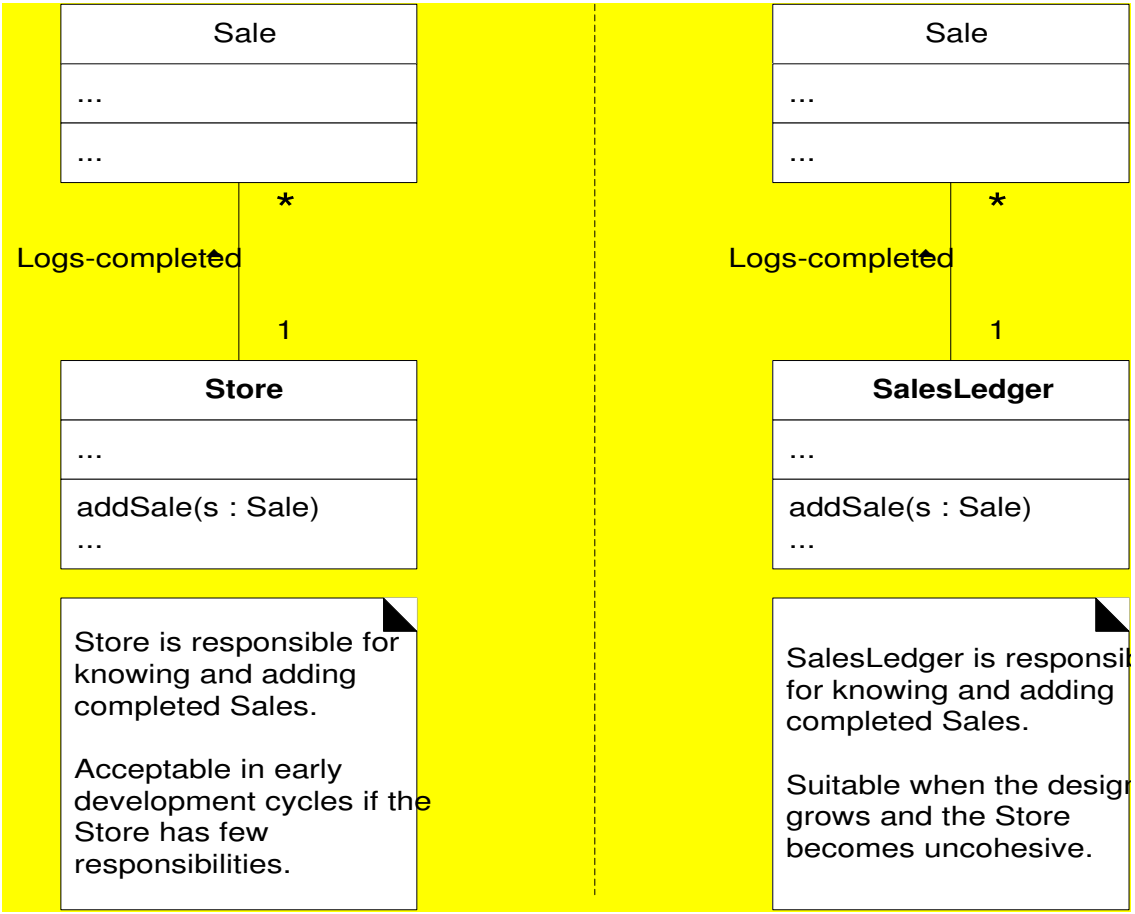


Logging a Sale

- Uma vez completa a venda, os requisitos pedem que a **venda** deveria ser registrada em um log [histórico].
  - Quem é responsável por conhecer todos os logged sales, e fazer o logging?
  - Pela meta do baixo gap representacional...

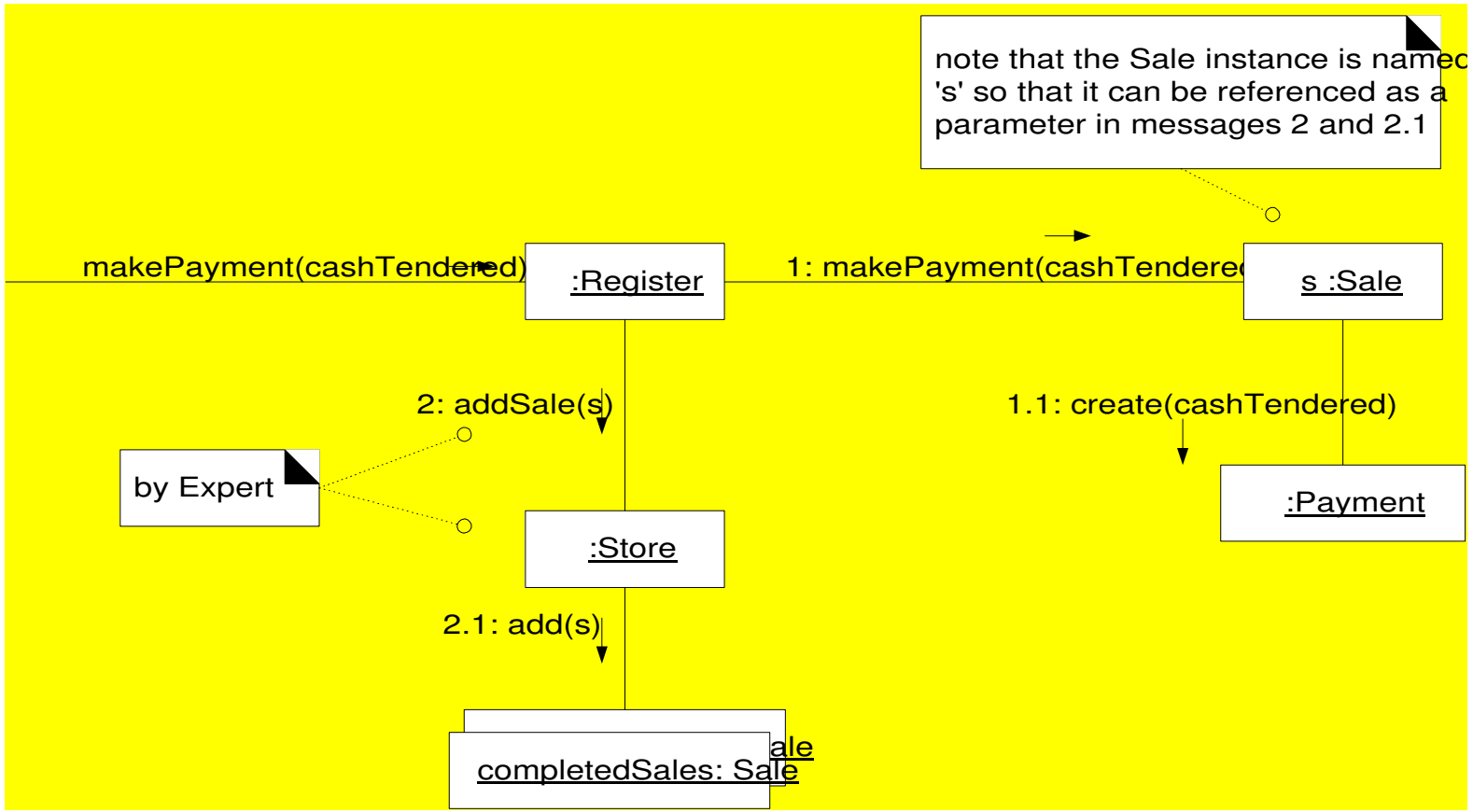
Based on C. Larman, 2002

Quem deveria ser responsável por conhecer as **sales** completadas?



Based on C. Larman, 2002

Logging uma sale completada



Based on C. Larman, 2002

Calculando o balance

- O uc **Process Sale** demanda que o balance devido de um pagamento seja impresso em um recibo e mostrado de alguma forma.
  - Por causa do princípio do Model-View Separation não devemos nos proecupar como ele será mostrado, mas assegurar que seja conhecido
  - Nenhuma classe atual conhece o balance. Precisamos criar o design de um objeto de interação que satisfaça o requisito
- Quem é responsável por conhecer o balance?
  - **Sale** and **Payment** são Experts parciais

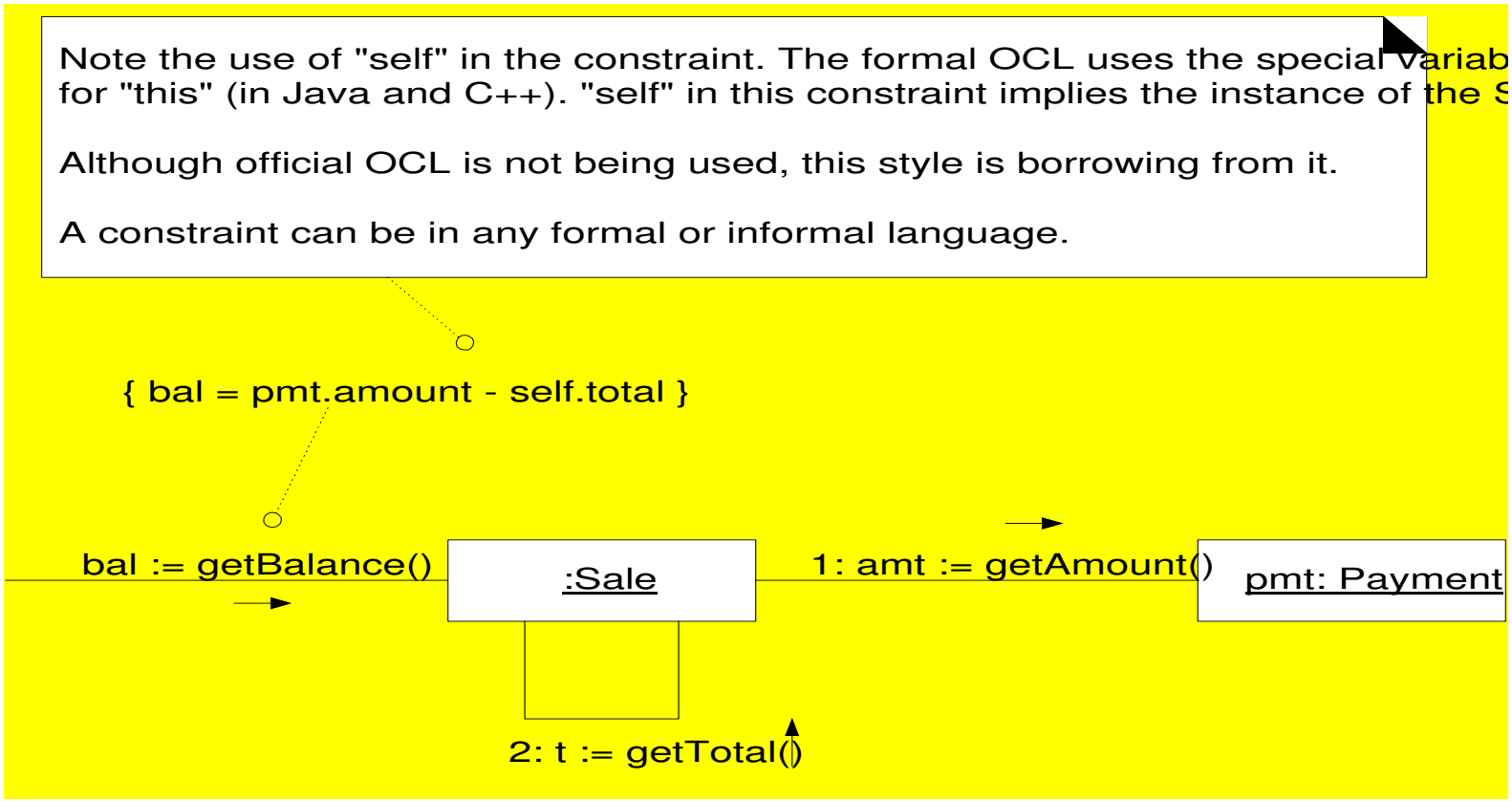
Based on C. Larman, 2002

Calculando o balance

- Para calcular o balance, o **sale total** e **payment cash tendered** são requeridos
- Se escolhermos **Payment**, ela necessitará visibilidade para **Sale** (para seu total)
  - Uma vez que atualmente ela não conhece Sale, esta abordagem aumentaria o coupling…
- A Sale já tem visibilidade para o Payment – como seu creator-
  - Esta abordagem não aumenta o coupling geral

Based on C. Larman, 2002

diagrama de interação para Sale--getBalance



Based on C. Larman, 2002

Object Design: StartUp

- A operação Start Up representa abstratamente a fase de inicialização da execução, quando uma aplicação é lançada
- É dependente da linguagem de programação e sistema operacional
- Um idioma de design comum envolve criar um objeto inicial de dominio.
  - O objeto inicial de domínio é responsável pela criação de seus objetos de domínio filhos direto. [e.g. se store é o objeto inicial de domínio, ele será responsável pela criação do objeto register]

Based on C. Larman, 2002

Como Aplicações *Start Up*

- O lugar onde esse objeto inicial de domínio é criado depende da tecnologia de objetos escolhida. Em uma aplicação Java, o método *main* pode criá-lo.

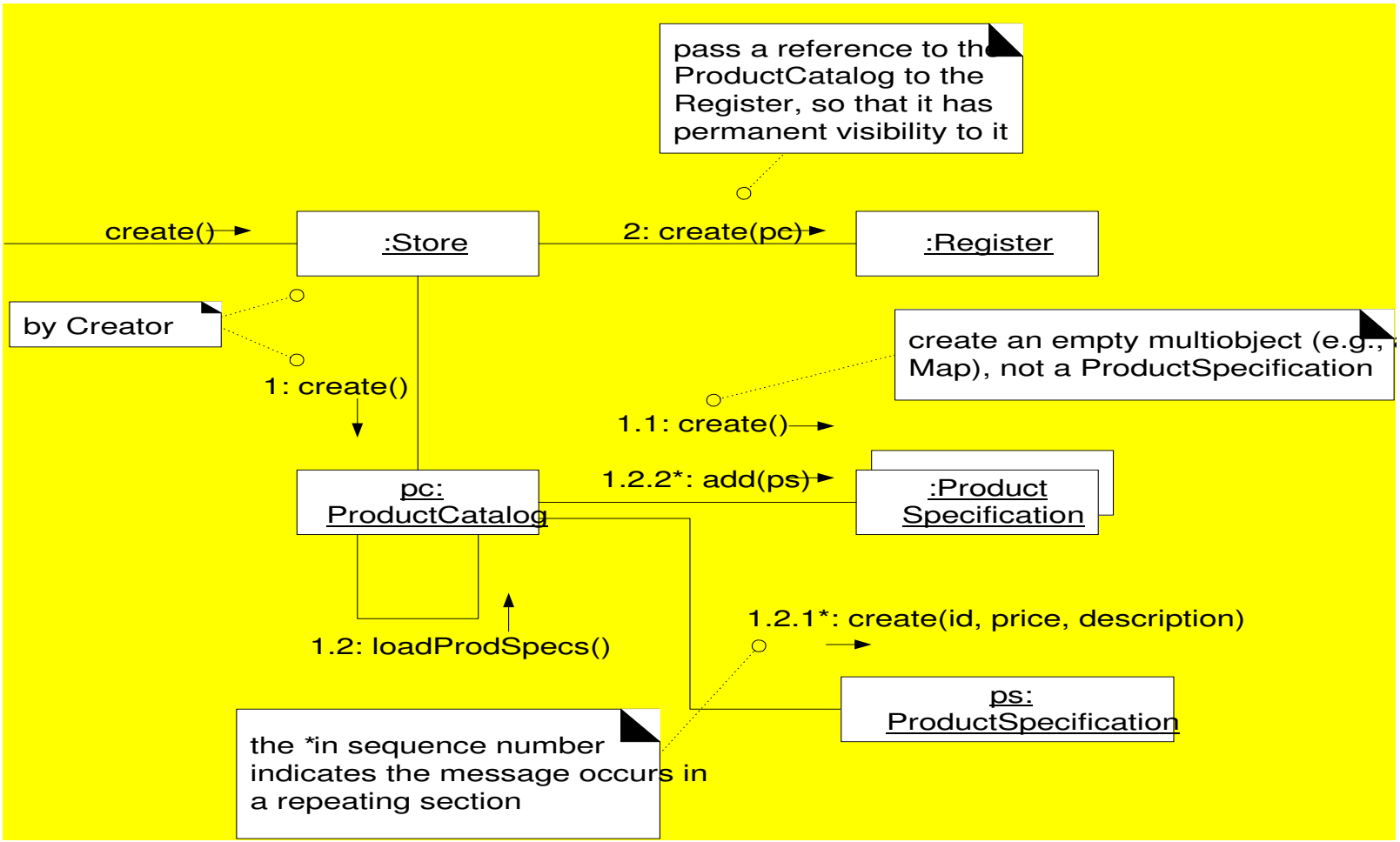
```
public class Main
{
public static void main(String[] args)
{
//store is the initial domain object.
//the store creates some other domain objects.
Store store = new Store ();
Register register = store.getRegister();
ProcessSaleJFrame frame = new
   ProcessSaleJFrame(register);
…
}}
```

Store--create() Design

- Trabalho de inicialização:
  - Store, Register, ProductCatalog, ProductSpecifications devem ser criadas
  - ProductCatalog deve ser associada com ProductSpecifications
  - Store deve ser associada com ProductCatalog
  - Store deve ser associada com Register
  - Register deve ser associada com ProductCatalog

Based on C. Larman, 2002

Criação do objeto inicial de domínio



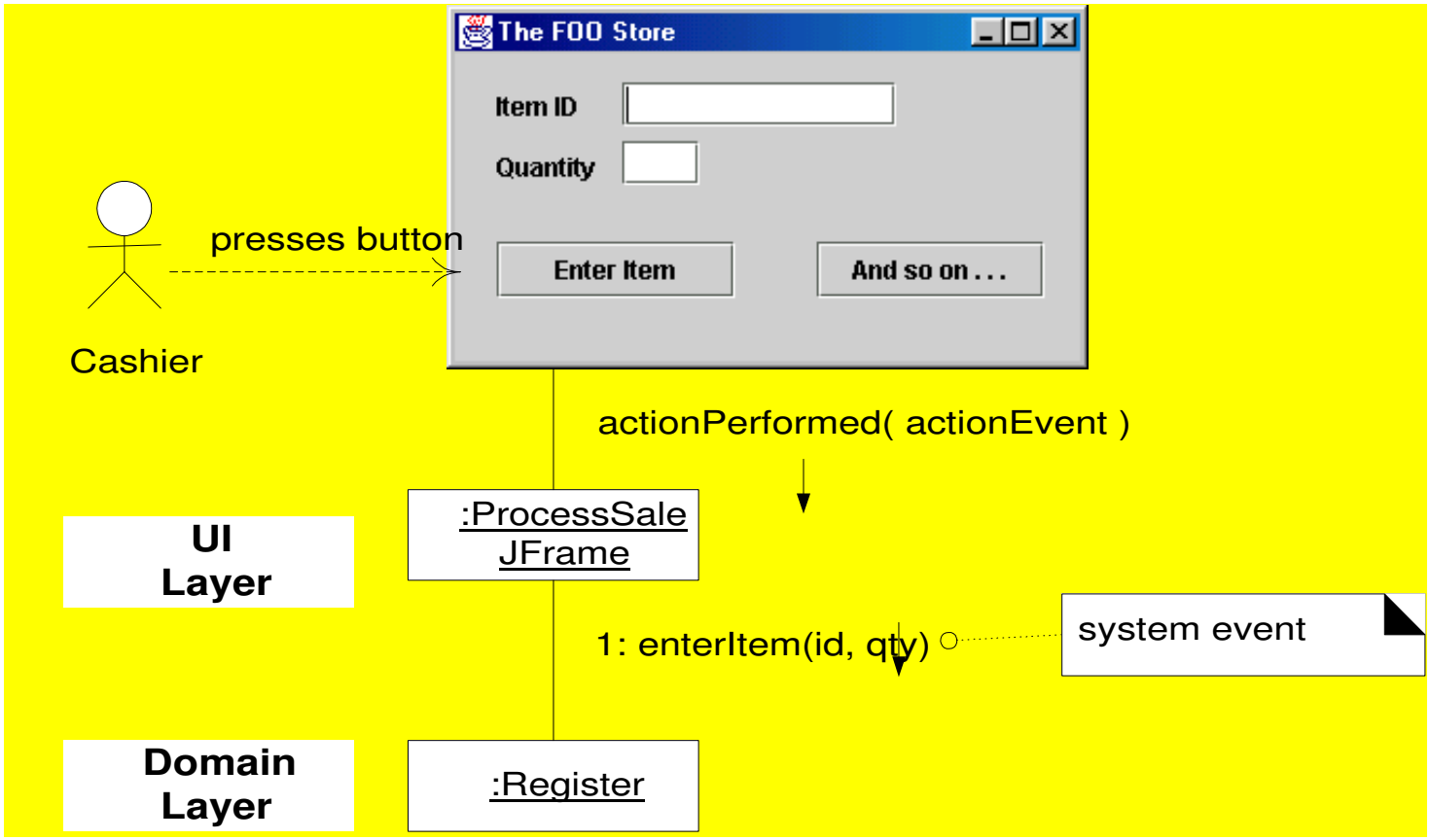
Based on C. Larman, 2002

Conectando a UI Layer à Domain Layer

- Uma vez que um objeto UI tenha uma conexão comum a instancia de Register, ele pode encaminhar *system event messages* para este [tais como **enterItem** e **endSale**]
- No caso da mensagem enterItem a janela deve mostrar o total corrente após cada entrada
  - adicionar um método getTotal ao Register [torna Register menos coeso]
  - Solicitar uma referência para o objeto Sale [aumenta o acoplamento da UI para a camada de domínio, *but with a stable thing*]

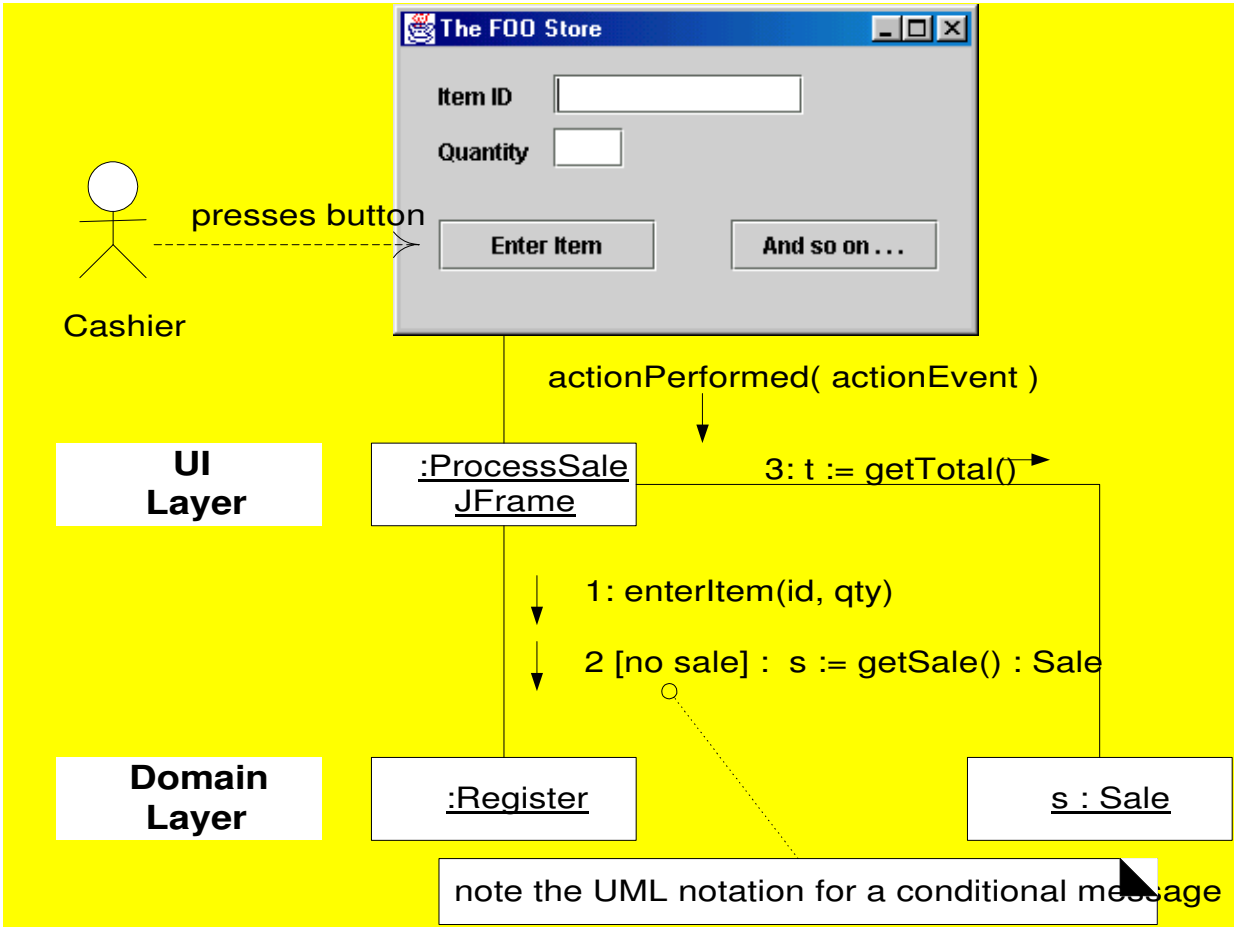
Based on C. Larman, 2002

Conectando as UI and the Domain Layers



Based on C. Larman, 2002

Conectando as UI and the Domain Layers



Based on C. Larman, 2002