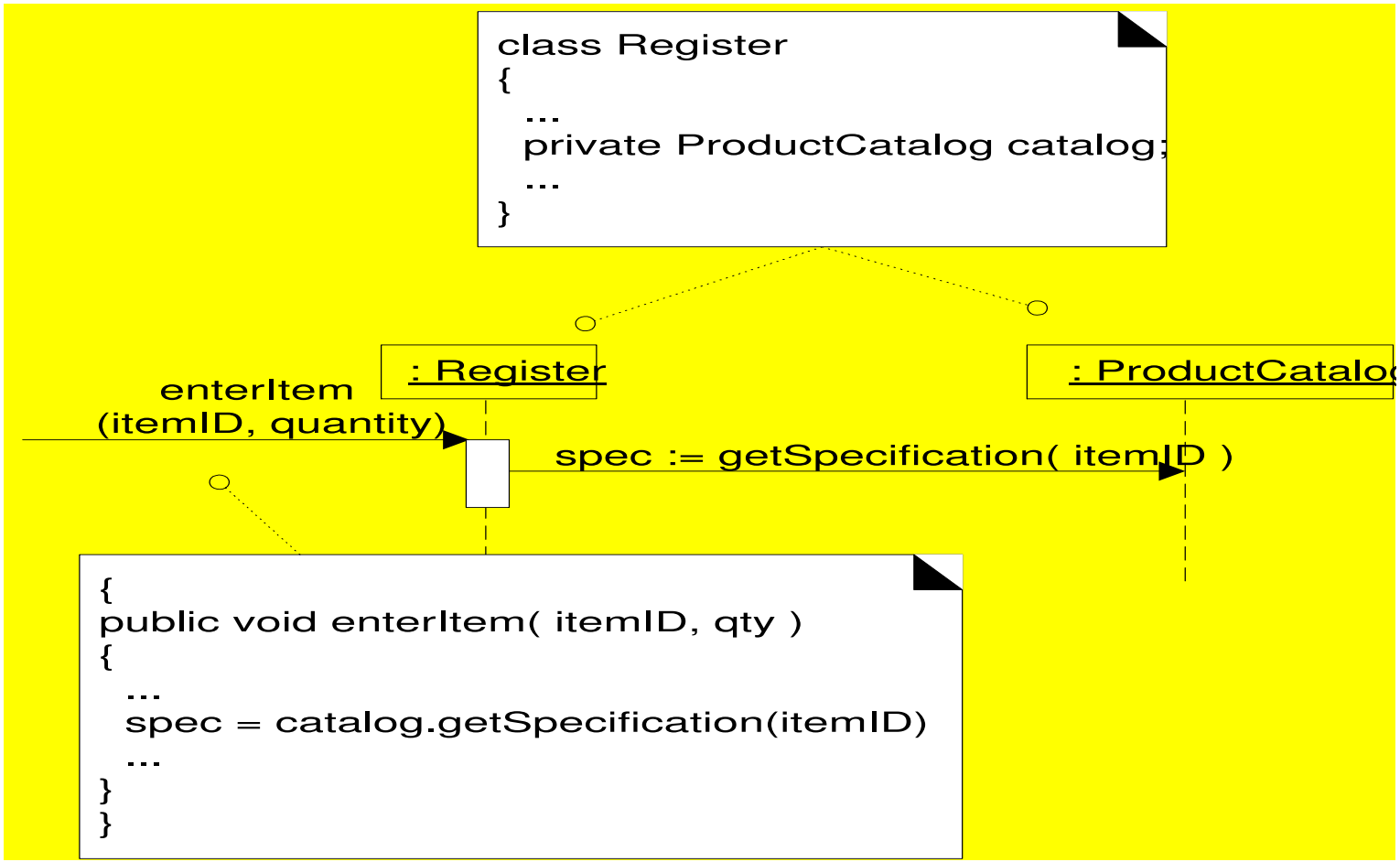
Modelo de Design : Determinando Visibilidade

Visibilidade

- Habilidade de um objeto enxergar ou fazer referência a outro
  - os designs criados para eventos do sistema (*enterItem*, etc.) ilustram mensagens entre objetos
  - Para um objeto *sender* enviar uma mensagem para um objeto *receiver*, o *sender* deve ter algum tipo de referência ou apontador para o objeto *receiver*
    - Ex. a mensagem *getSpecification enviada de* um *Register* para um *ProductCatalog* implica que a instância de *ProductCatalog* é visivel para a instância de *Register*

Visibilidade de Register para ProductCatalog



Based on C. Larman, 2002

Visibilidade

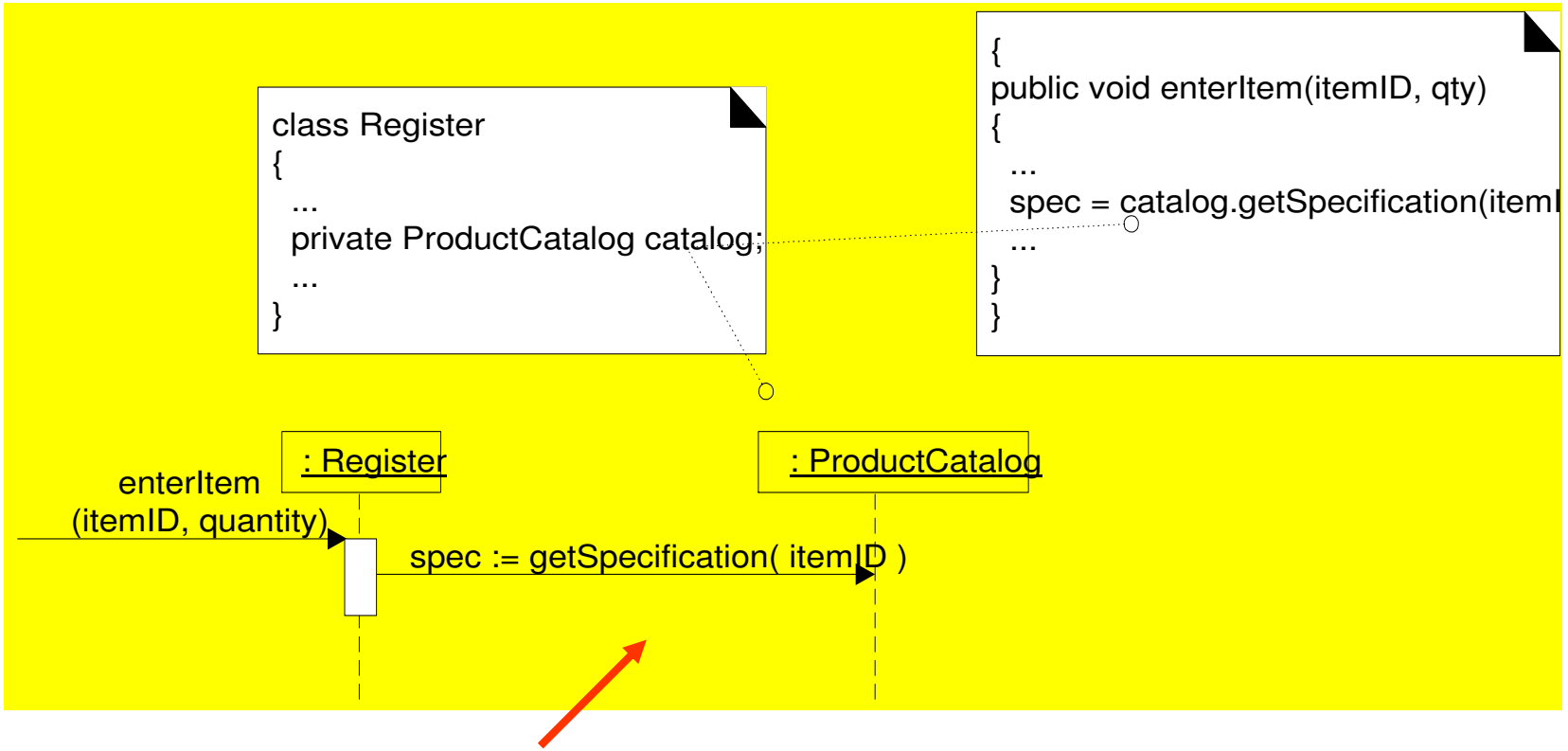
- Relacionada a escopo: Estaria um recurso [ex. uma instância] dentro do escopo de outra?
- 4 maneiras de alcançar visibilidade do objeto A para o objeto B:
  - **Por Atributo** [B é um atributo de A]
  - **Por Parâmetro** [B é um parâmetro de um método de A]
  - **Local** [B é um objeto local em um método de A]
  - **Global** [B é visível globalmente]

Visibilidade por atributo de A para B

- Existe quando B é um atributo de A
- Visibilidade Permanente
  - Persiste enquanto A e B existirem
  - Ex. na definição de uma classe Java para *Register*, uma instância de *Register* pode ter visibilidade por atributo para um *ProductCatalog*, uma vez que é um atributo (Java instance variable) de *Register*

```
Public class Register
{
...
Private ProductCatalog catalog;
...
}
```

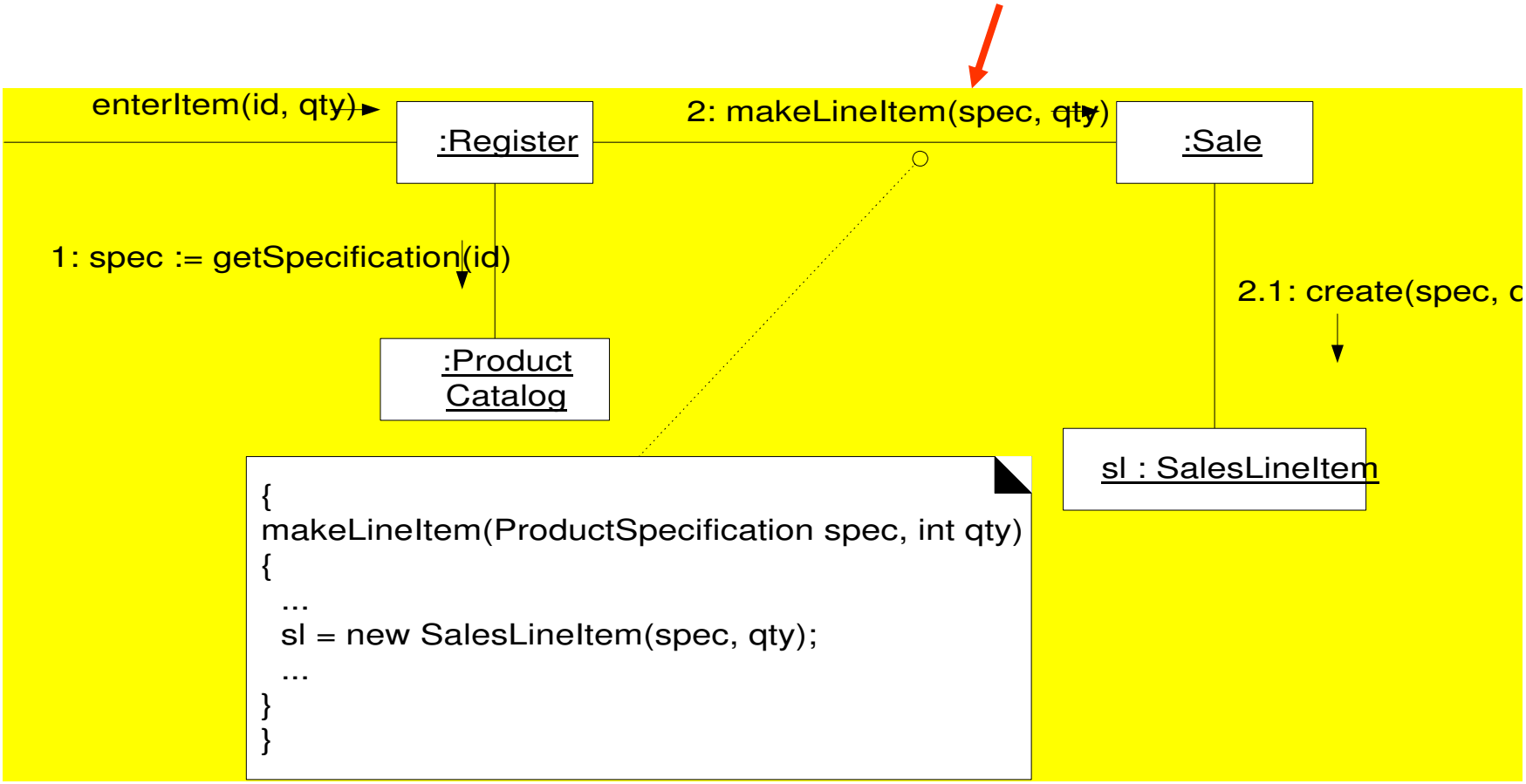
Visibilidade por Atributo



Visibilidade por parâmetro de A para B

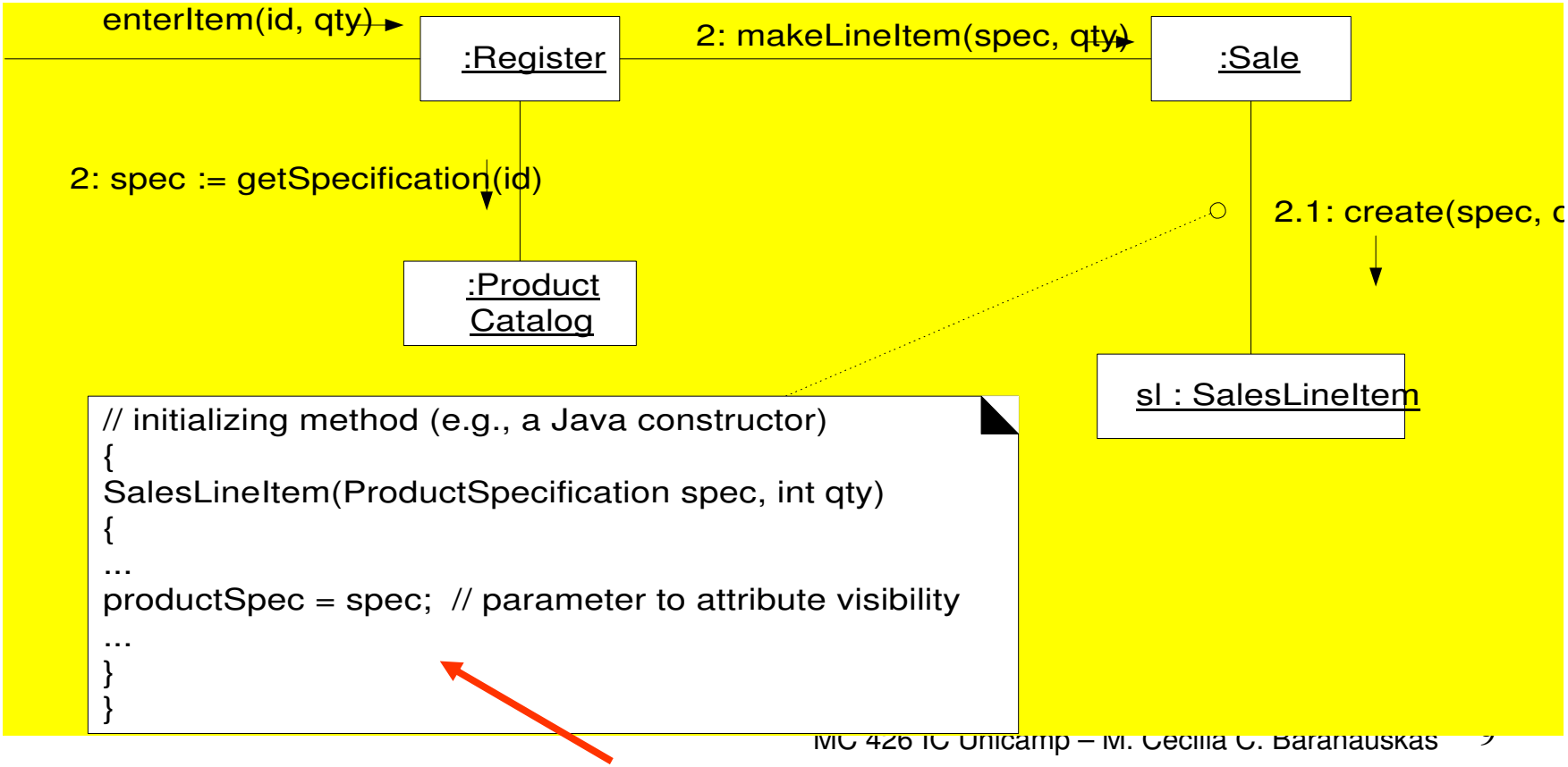
- Existe quando B é passado como um parâmetro para um método de A.
- Visibilidade temporária
  - Persiste dentro do escopo do método
  - Ex. quando *makeLineItem* é enviado para uma instância de Sale, um *ProductSpecification* é passado como parâmetro
    - Dentro do escopo do método *makeLineItem*, o *Sale* tem visibilidade por parâmetro para um *ProductSpecification*

Visibilidade por Parâmetro



Parâmetro atribuído a um atributo

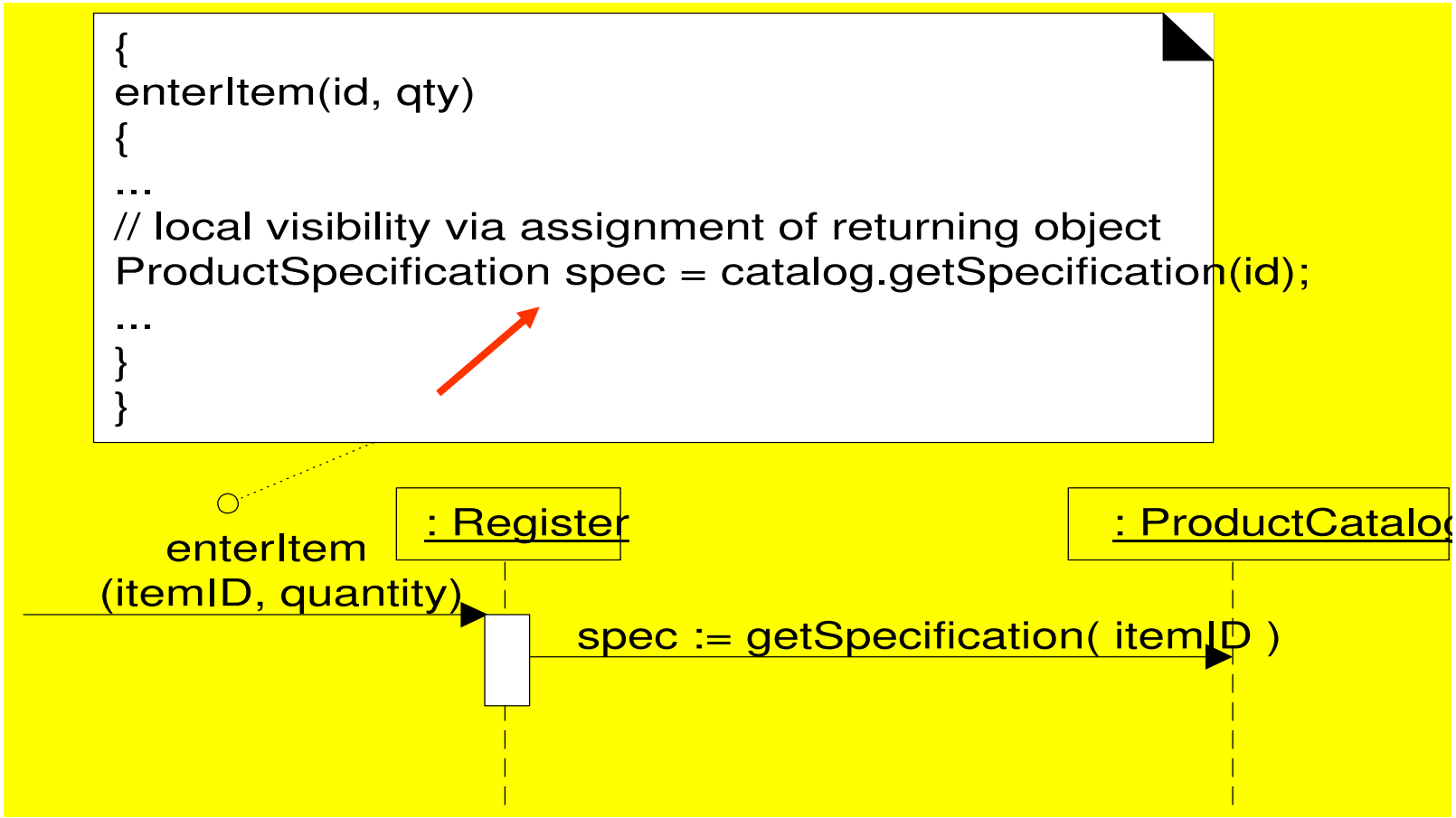
Também estabelece visibilidade



Visibilidade local de A para B

- Existe quando B é declarada como objeto local dentro de um método de A
- Visibilidade Temporária
  - Persiste dentro do escopo do método
- Duas maneiras de alcançar visibilidade local:
  - Criando uma nova instância local e atribuindo-a a uma variável local
  - Atribuindo o objeto que retorna da invocação de um método para uma variável local

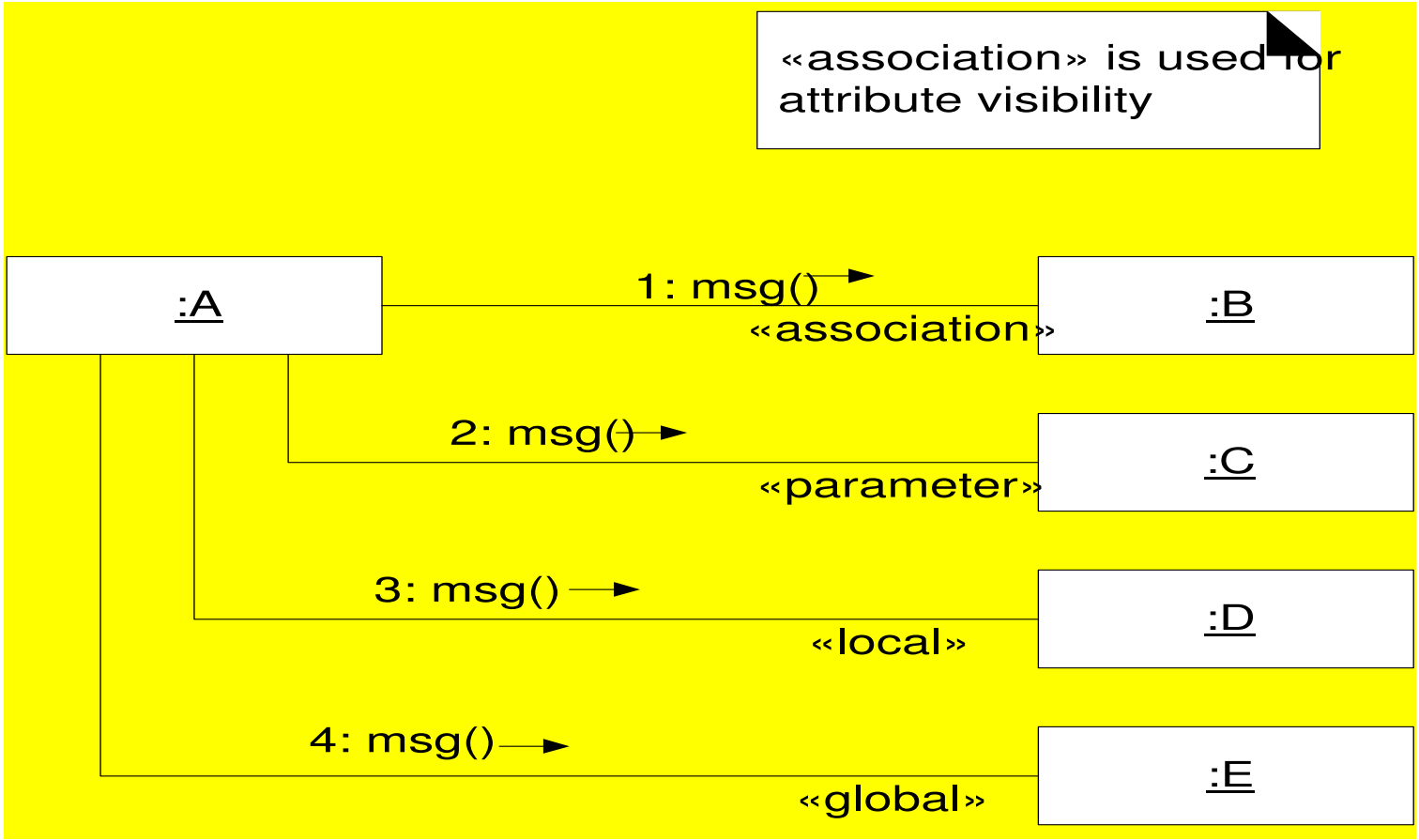
Visibilidade local [2nd caso]



Visibilidade de A para B

- Existe quando B é global a A
- Visibilidade permanente
- Forma menos comum de visibilidade em sistemas oo
- Uma maneira de alcançar visibilidade para uma variável global
  - Isto é possível em algumas linguagens [C++], mas não é possível em outras [Java]

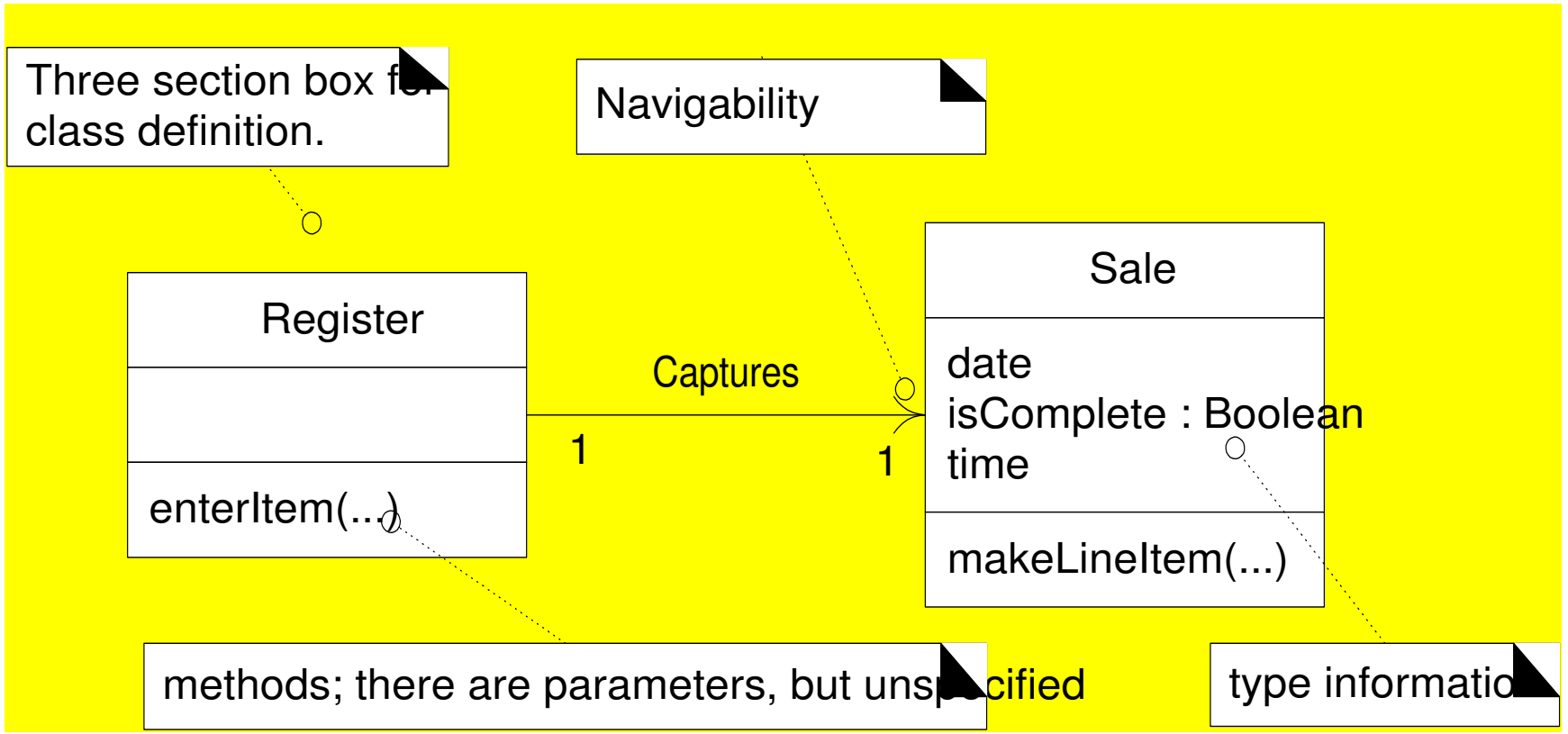
Notação UML para mostrar o tipo de visibilidade



Modelo de Design: Criando Diagramas de Classe de Design

Identificar a especificação para as classes de software que participam na solução do sftw, e anotá-las com detalhes de design, tais como métodos

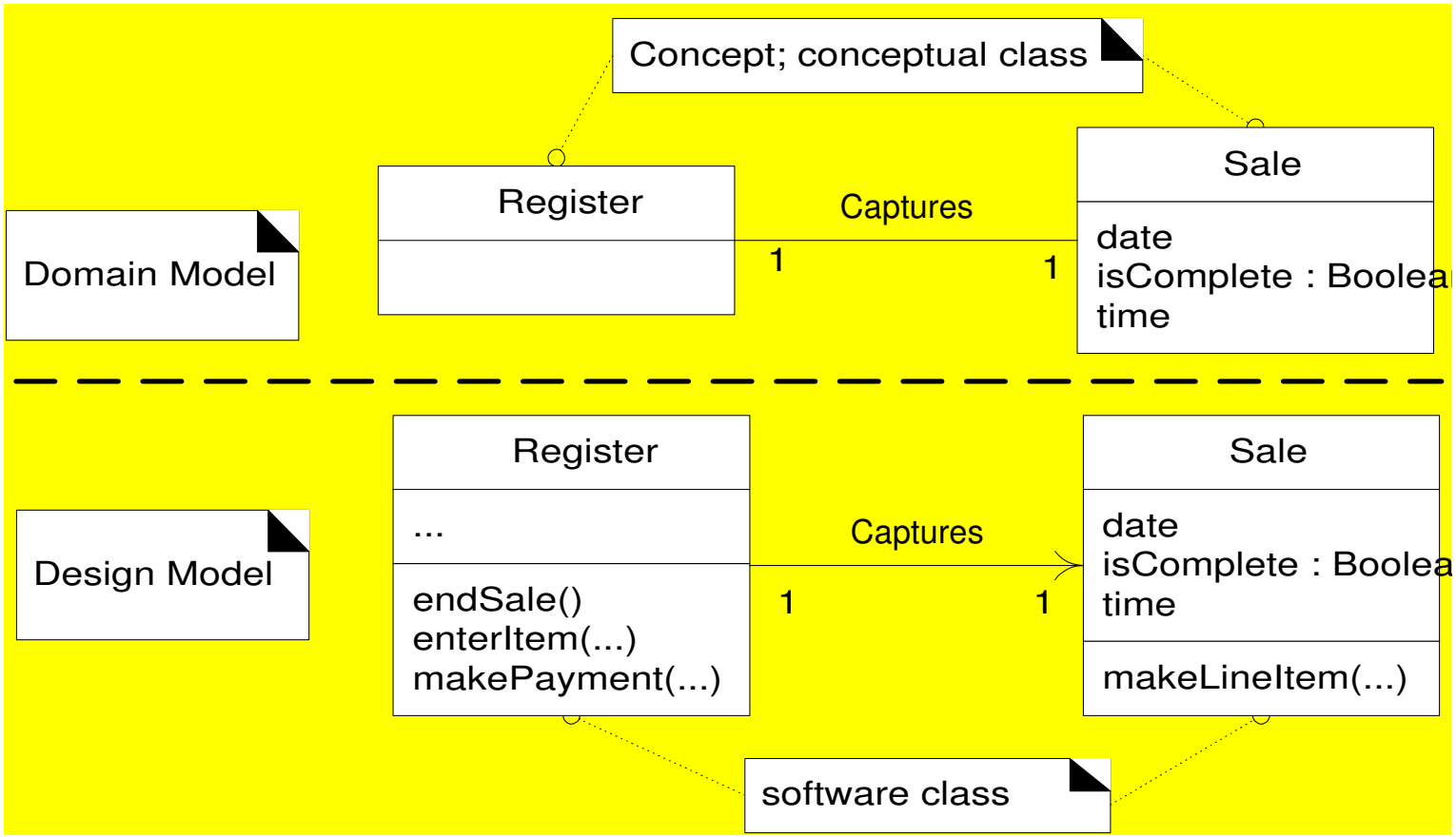
DCD – definição parcial das classes Register e Sale



Um Diagrama de Classe de Design– DCD

- Ilustra as especificações para classes de sftw na aplicação. Informação típica inclui:
  - Classes, associações e atributos
  - Métodos
  - Informação de tipo de atributo
  - Navegabilidade
  - Dependências

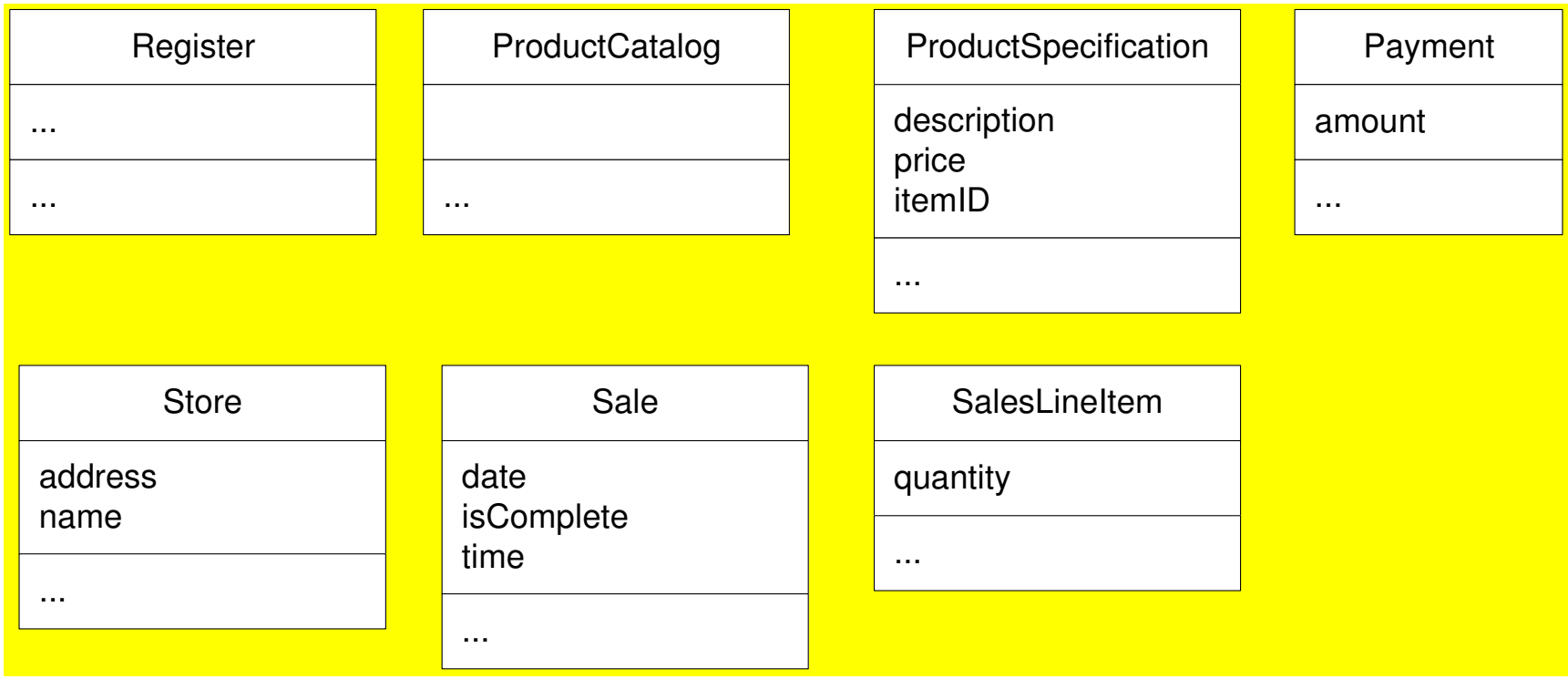
Modelo de Domínio x Modelo de Classes de Design



Criando um DCD para POS

- 1. Identificar as classes que participam na solução do sftw
  - Olhar nos diagramas de interação:
    - *Register, ProductCatalog, Store, Payment, Sale, ProductSpecification, SalesLineItem*
- 2. Desenhar um diagrama de classes para essas classes
  - Incluir os atributos previamente identificados no DM

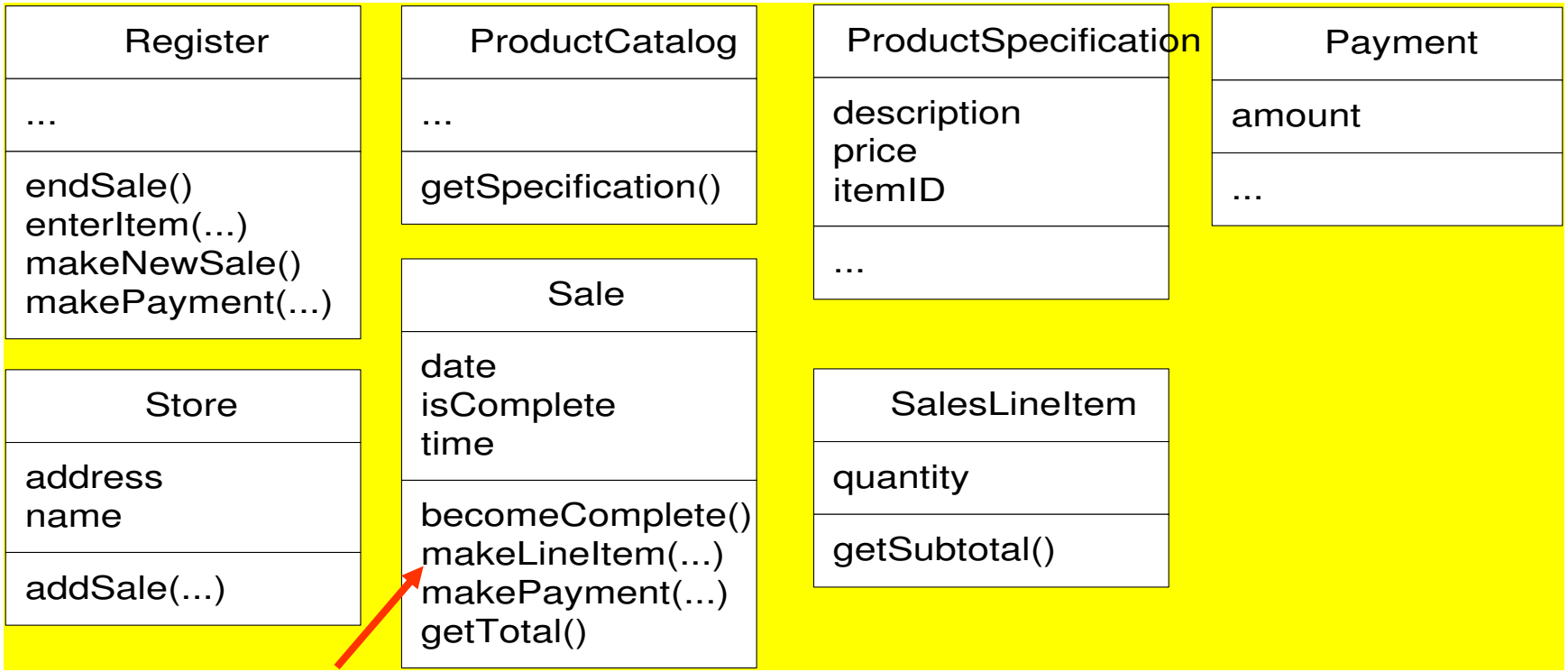
Sftw Classes in POS



Criando um DCD para POS [cont]

- 3. Adicionar nomes de métodos
  - Os métodos de cada classe podem ser identificados analisando o diagrama de interação
  - Ex. se a mensagem *makeLineItem* é enviada para uma instância da classe *Sale*, então a classe *Sale* deve definir um método *makeLineItem*

Métodos em POS



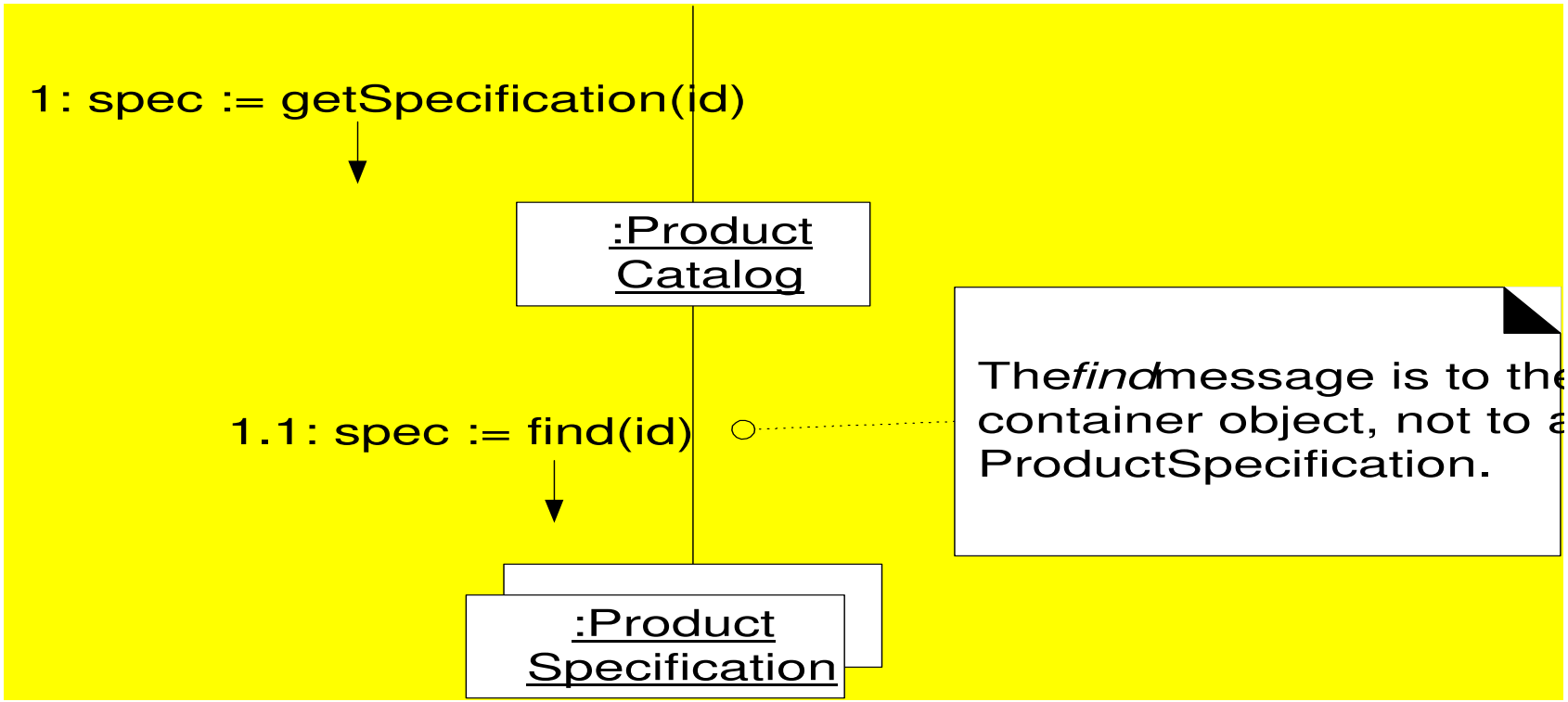
Nomes de métodos- create

- A mensagem *create* é uma forma independente da UML para indicar instanciação e inicialização
  - Ao traduzir o projeto para uma linguagem de programação OO, deve ser expresso em termos de seu idioma para instanciação e inicialização
  - Não há um método real *create* em C++, Java ou Smalltalk
    - Em C++ implica alocação automática de espaço livre com o operador *new*, seguido por uma chamada de construtor
    - Em Java implica a invocação do operador *new*, seguido por uma chamada de construtor

Nomes de Métodos - Multiobjetos

- Uma mensagem para um multiobjeto é interpretada como uma mensagem para o próprio objeto *container/collection*
  - Ex. a mensagem *find* para o multiobjeto é interpretada como uma mensagem para o objeto container/collection tal como para um Java *Map*, a C++ *map* or a Smalltalk *Dictionary*
    - *[é parte da interface do multiobjeto]*

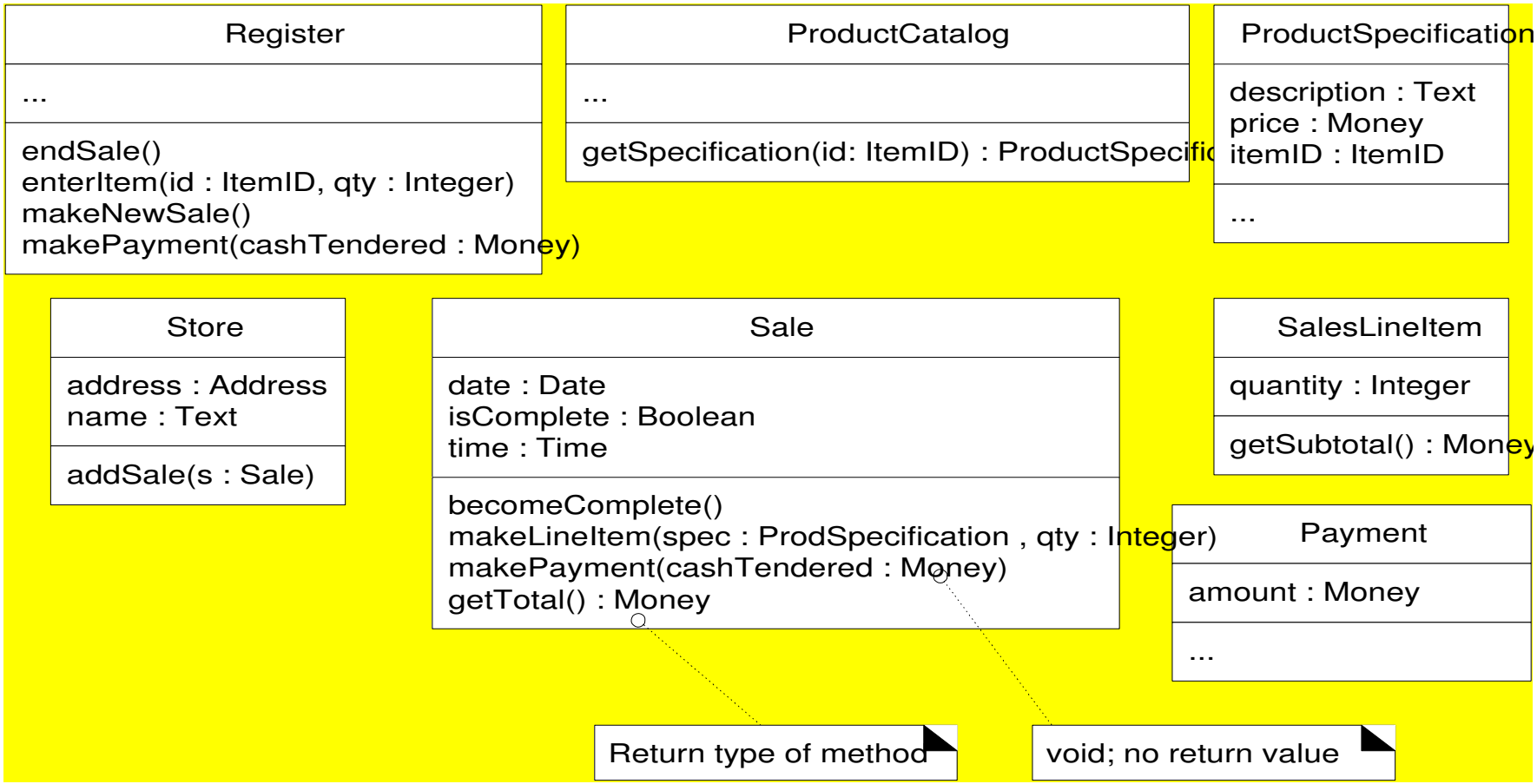
Mensagem para um multiobjeto



Criando um DCD para POS [cont]

- 4. Adicionar mais informação de tipo
  - Dependendo se o DCD está sendo criado com uma ferramenta CASE com geração automática de código
    - Detalhes sobre tipos dos atributos, parâmetros dos métodos e valores de retorno são necessários
  - Se criados para leitura de desenvolvedores de sftw
    - Detalhes de baixo nível podem não ser necessários

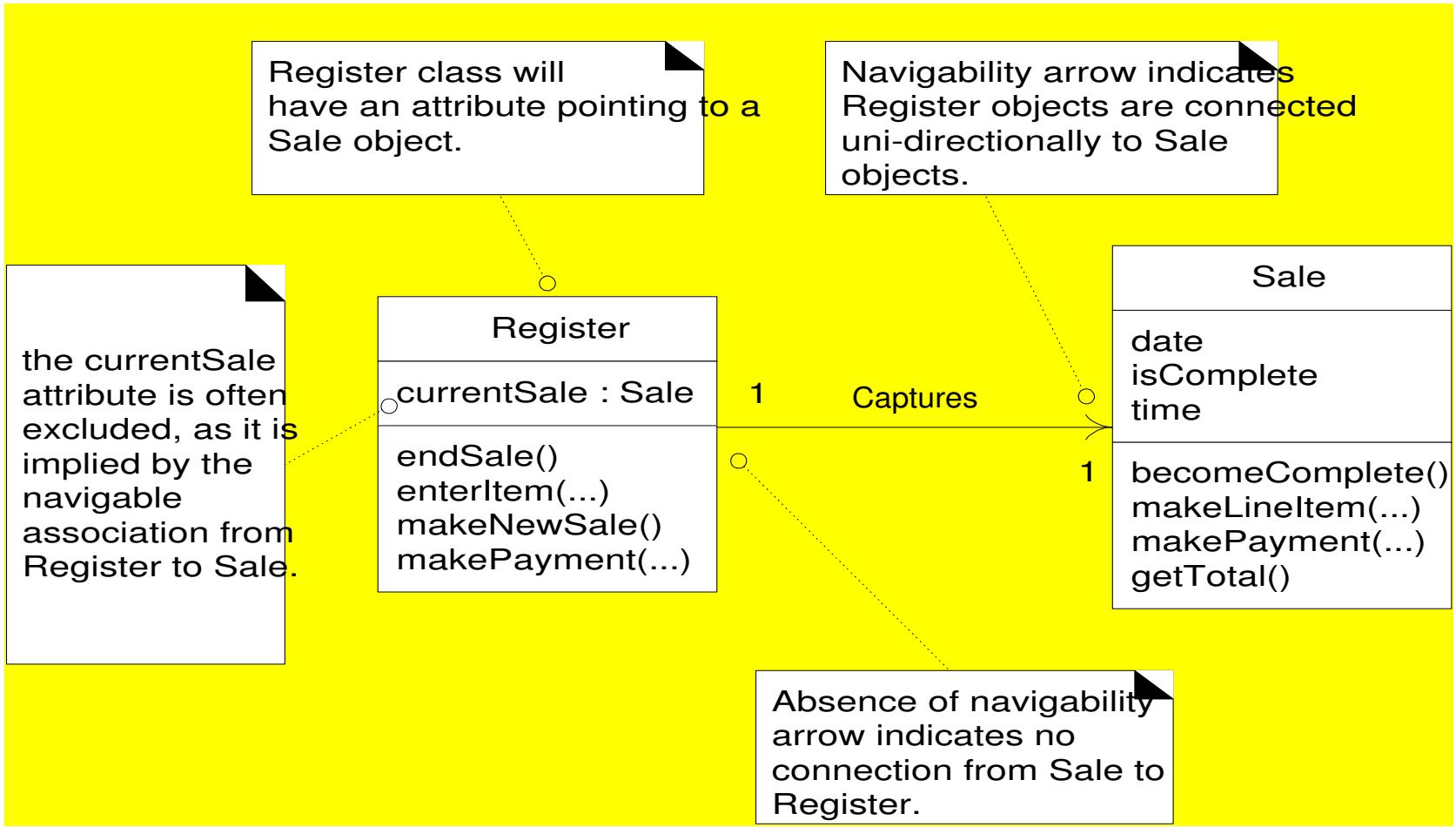
Adicionando informação de Tipo



Criando um DCD para POS [cont]

- 5. Adicionar Associações e Navegabilidade
  - Cada final de uma associação é chamado ***role***
  - No DCD um *role* pode ser decorado com uma seta de navegabilidade
  - Navegabilidade implica visibilidade [usualmente visibilidade de atributo]
    - Pode indicar uni-direcionalidade de navegação

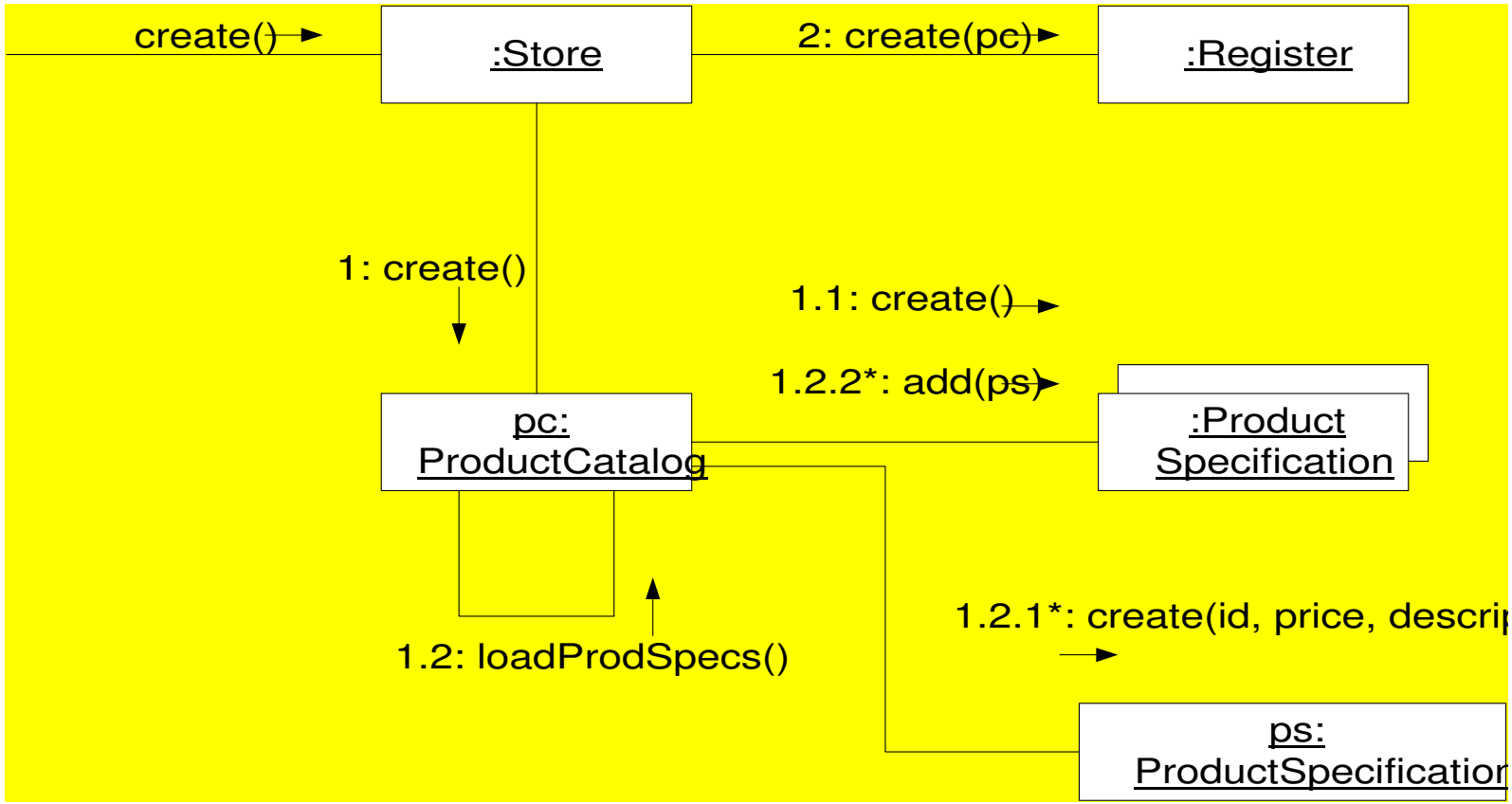
Mostrando navegabilidade ou visibilidade de atributo



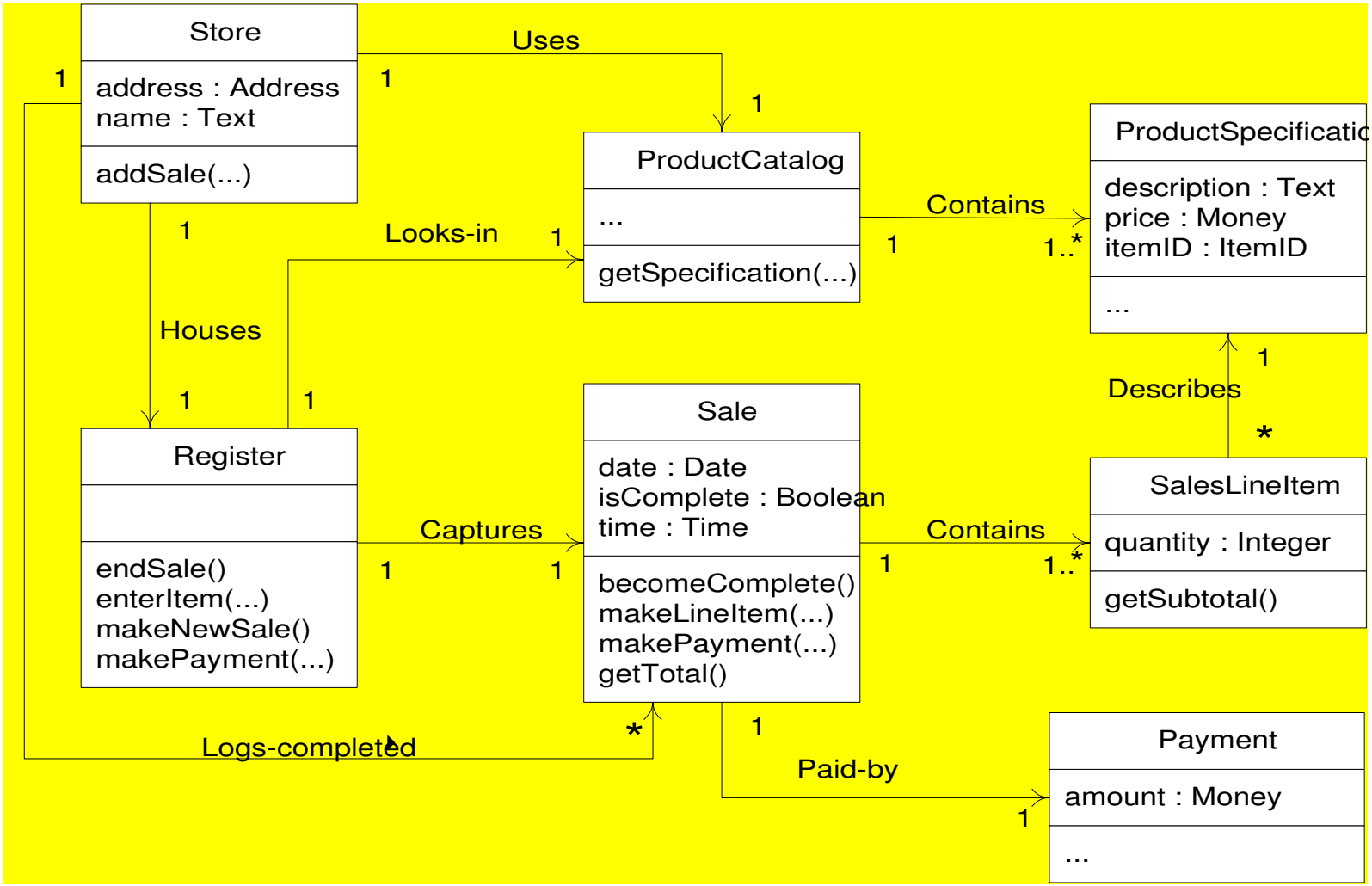
Navegabilidade

- É uma propriedade do role que indica que é possível navegar uni-direcionalmente pelas associações da origem para a classe destino
- Situações comuns sugerindo uma necessidade de definir uma associação com navegabilidade de A para B:
  - A envia uma mensagem para B
  - A cria uma instância de B
  - A necessita manter uma conexão para B

Navegabilidade é identificada de diagramas de interação



DCD para POS *[ver associações com navegabilidade]*



Adicionando Dependências

- A relação UML de **dependência** indica que um elemento (de qualquer tipo) tem conhecimento de outro elemento
  - É ilustrada com uma seta pontilhada
- Útil para mostrar visibilidade não de atributo entre classes
  - [visibilidade de parâmetro, global, ou local]

Relação de Dependência

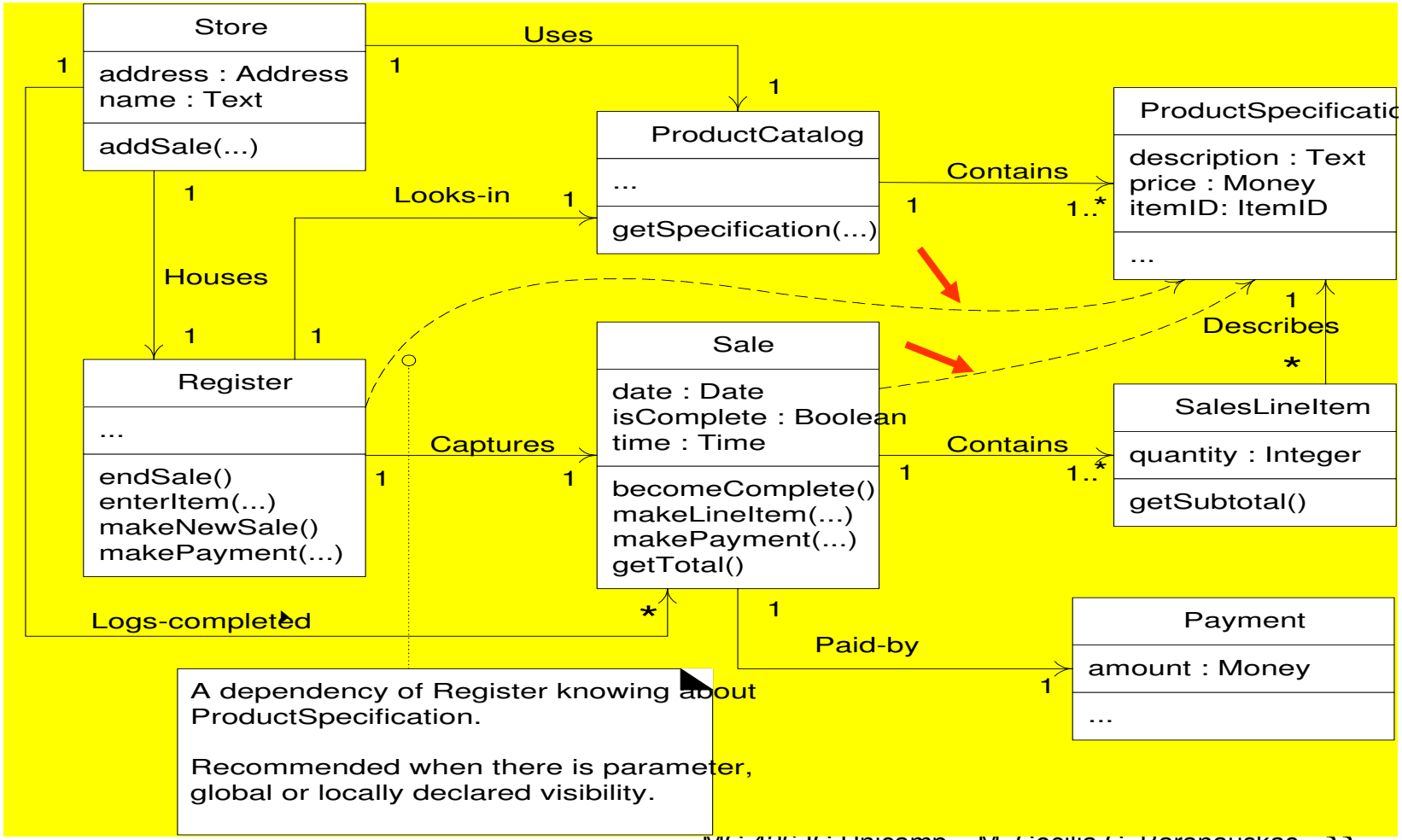
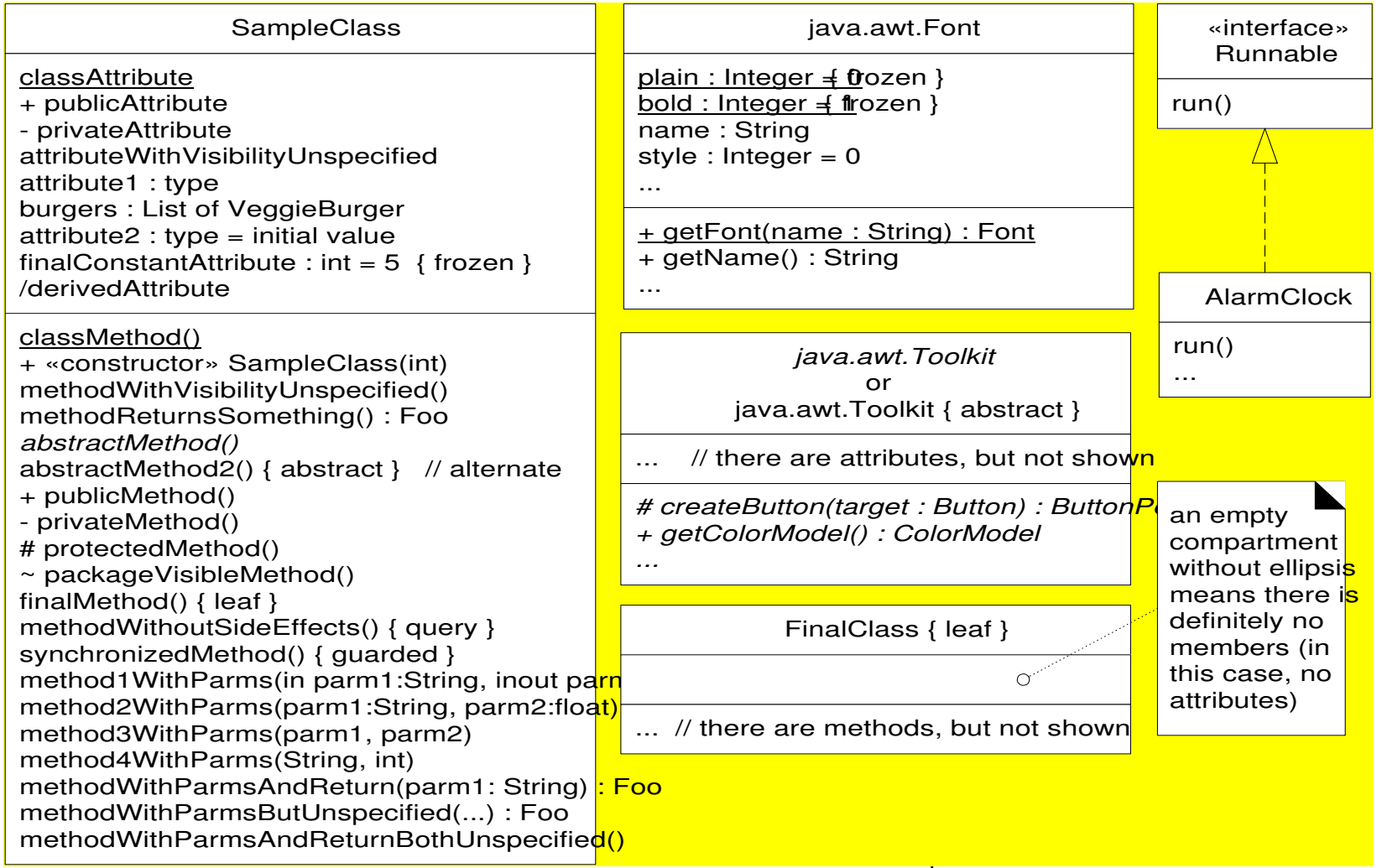
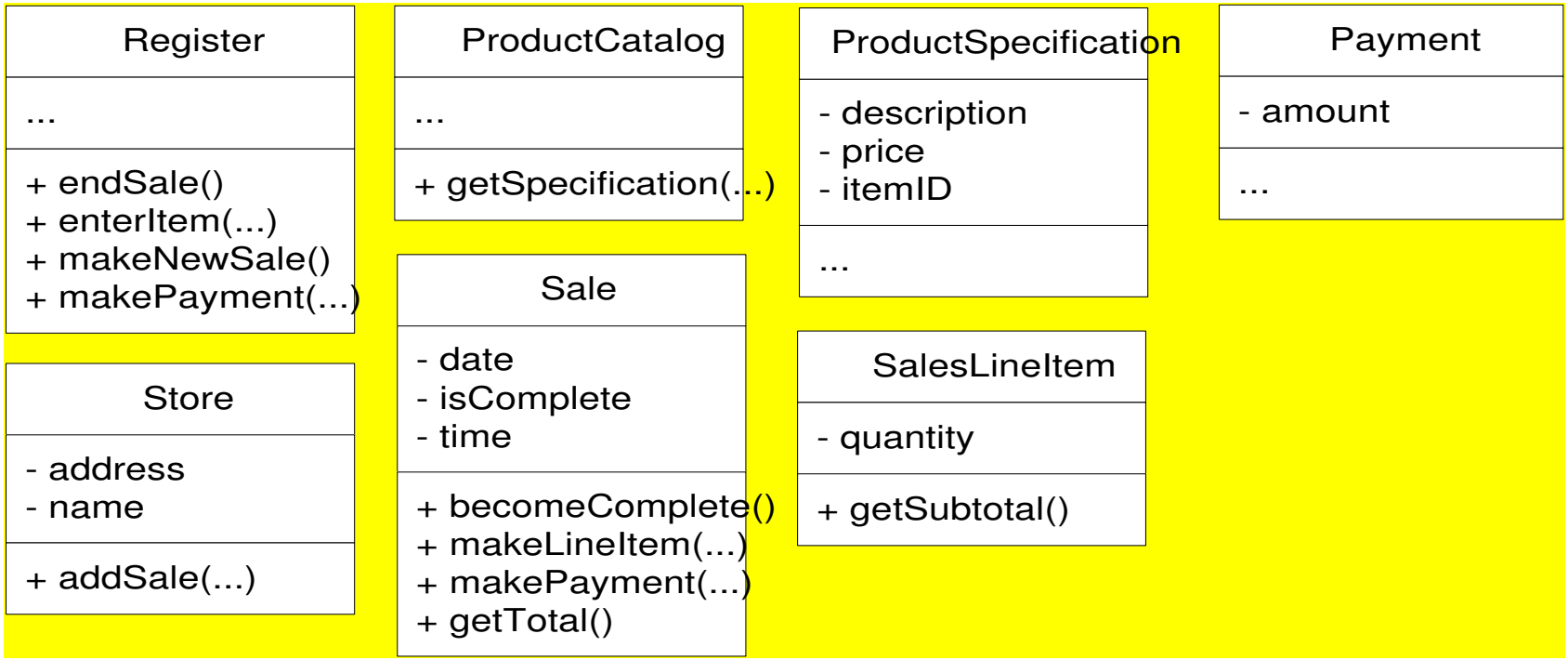


Diagrama de Classe UML
notação para detalhes

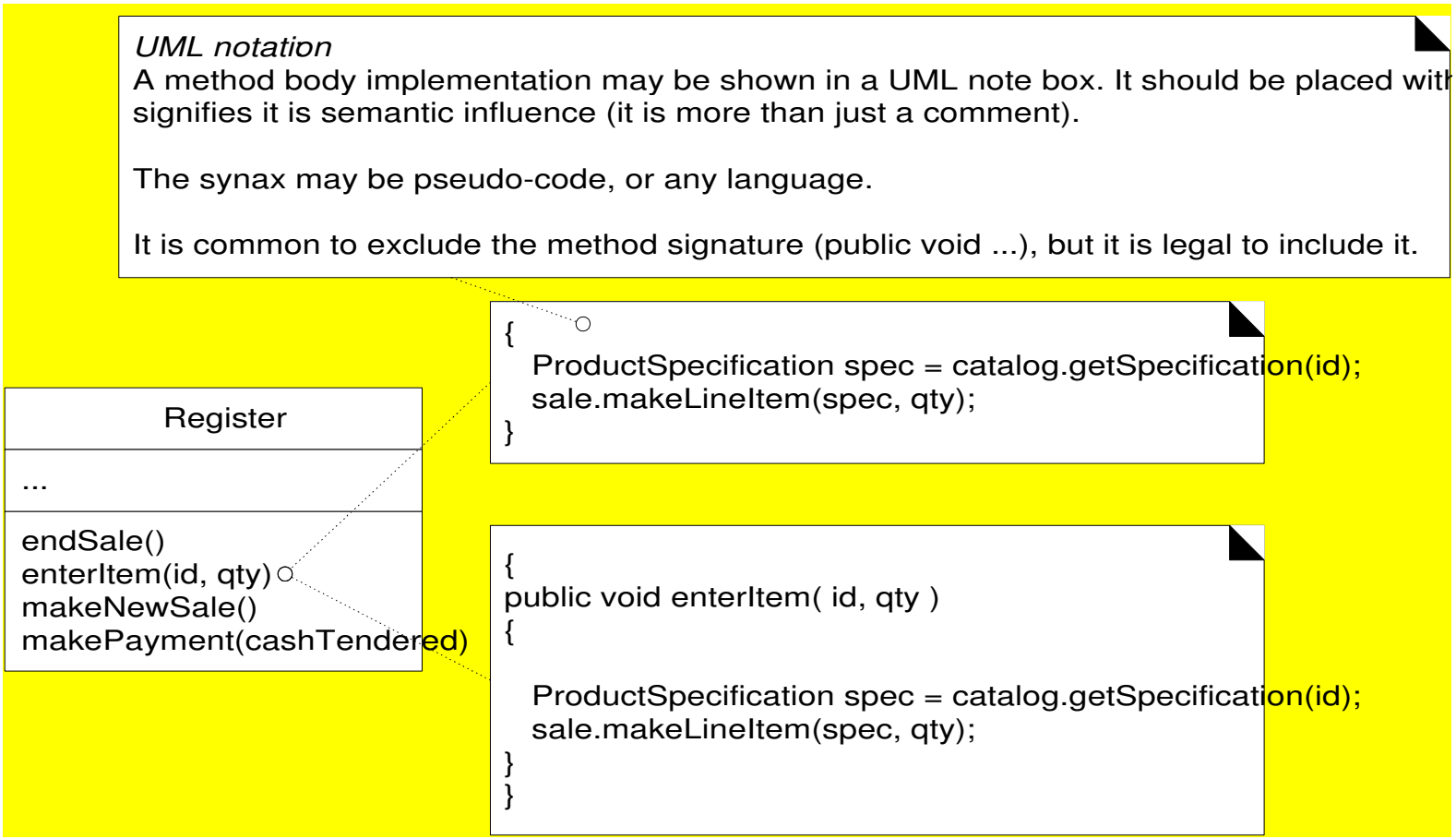


Detalhes no diagrama de classes de POS



Todos os atributos são privados [-]
e todos os métodos são públicos [+]

Notação para corpo de métodos em DCDs [e em diagramas de interação]



Referências

- Larman, C. (2002) *Applying UML and Patterns – An Introduction to Object Oriented Analysis and Design and the Unified Process*, Prentice-Hall Inc.