COMP 3711

OOD

Visibility
And
Mapping Design To Code

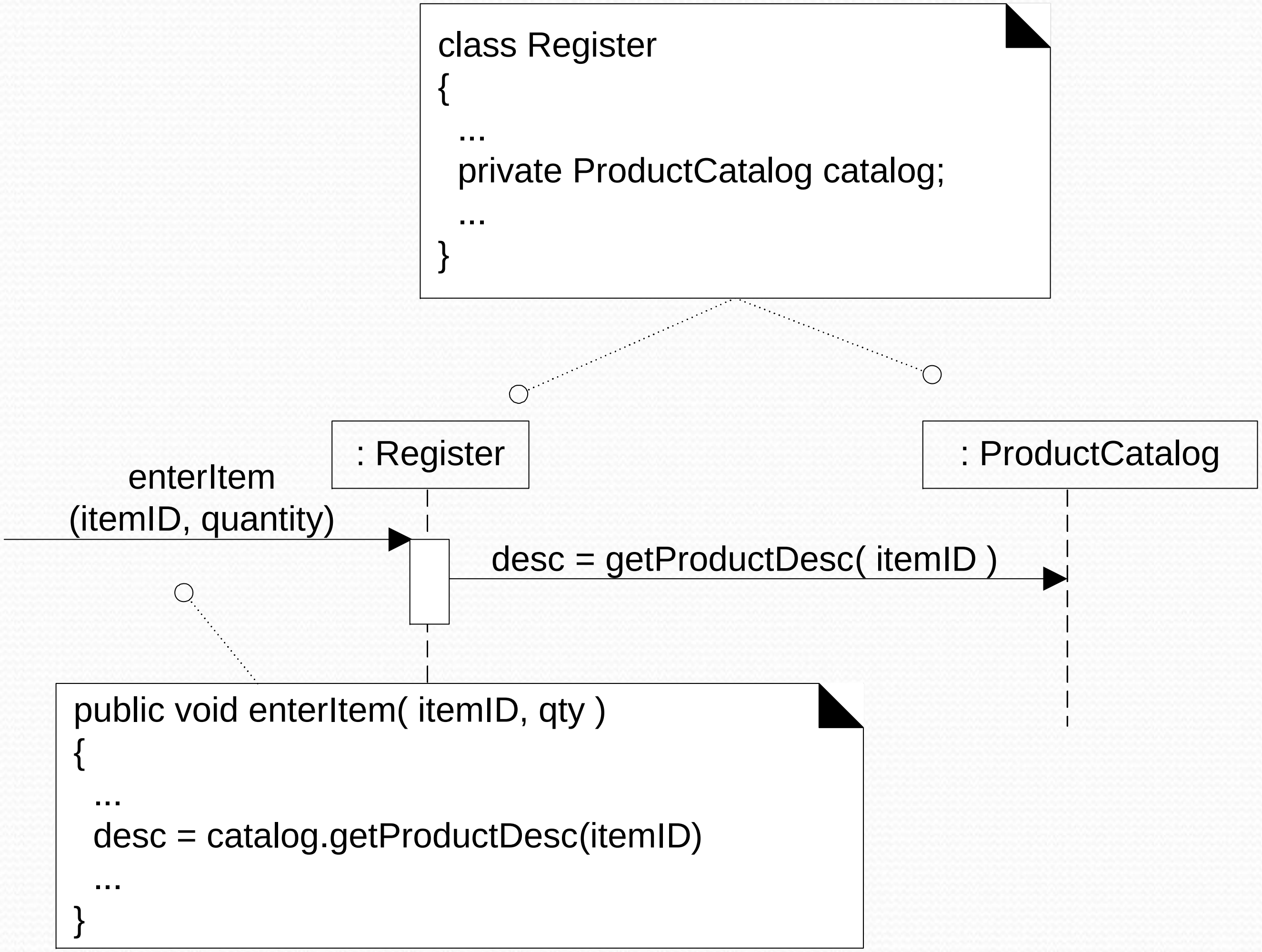
Larman Chapters 19, 20

What is Visibility

- *“Ability of an object to see or have a reference to another object”*
- *So … in order for object A to send a message to object B, object B must be visible to object A*

Example of Visibility

getProductDesc message here implies ProductCatalog instance is visible to the Register instance



```
class Register
{
  ...
  private ProductCatalog catalog;
  ...
}
```

: Register

: ProductCatalog

enterItem
(itemID, quantity)

desc = getProductDesc(itemID)

```
public void enterItem( itemID, qty )
{
  ...
  desc = catalog.getProductDesc(itemID)
  ...
}
```

Larman fig. 19.1

Types Of Visibility

- Example of visibility from object A to object B:
  - Attribute visibility
    - B is an attribute of A
  - Parameter visibility
    - B is a parameter of a method of A
  - Local visibility
    - B is a local object in a method of A
  - Global visibility
    - B is in some way globally visible

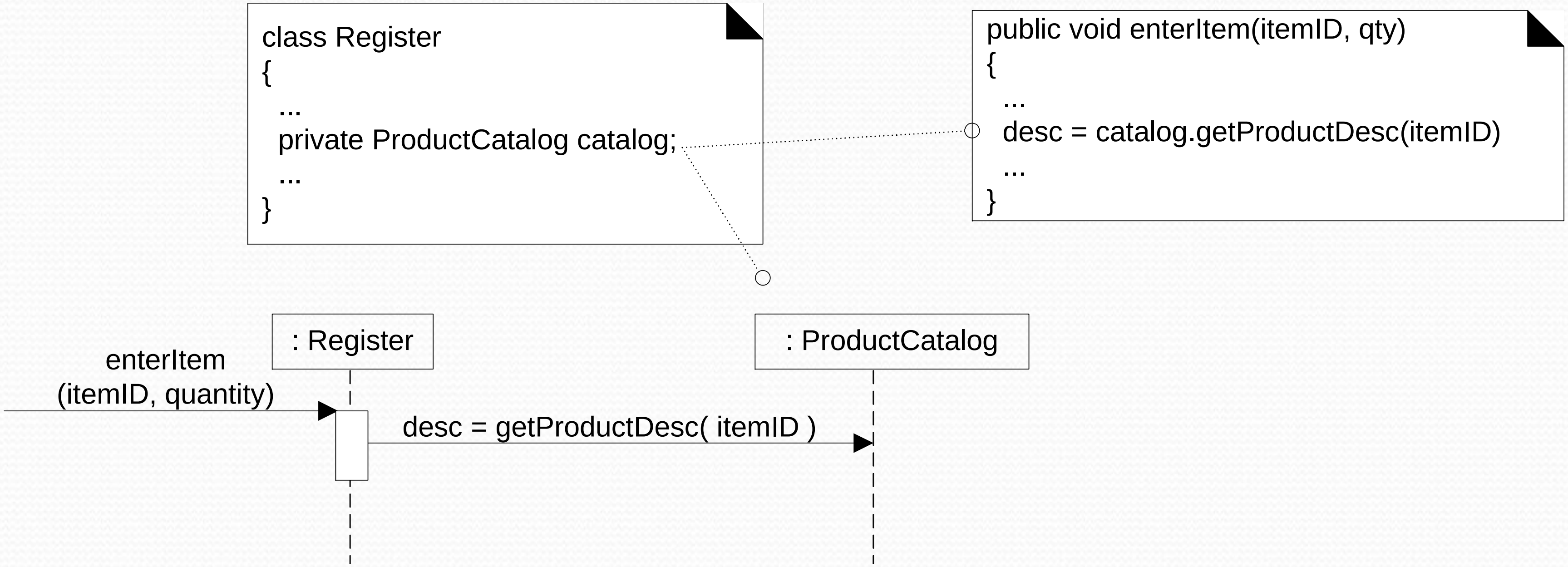
Attribute Visibility

- Attribute visibility exists when B is an attribute of A

- Permanent visibility as it persists as long as A and B exist

Attribute Visibility

Example of Register instance may have attribute visibility to ProductCatalog, since it is an attribute of Register



```
class Register
{
  ...
  private ProductCatalog catalog;
  ...
}
```

```
public void enterItem(itemID, qty)
{
  ...
  desc = catalog.getProductDesc(itemID)
  ...
}
```

Larman fig. 19.2

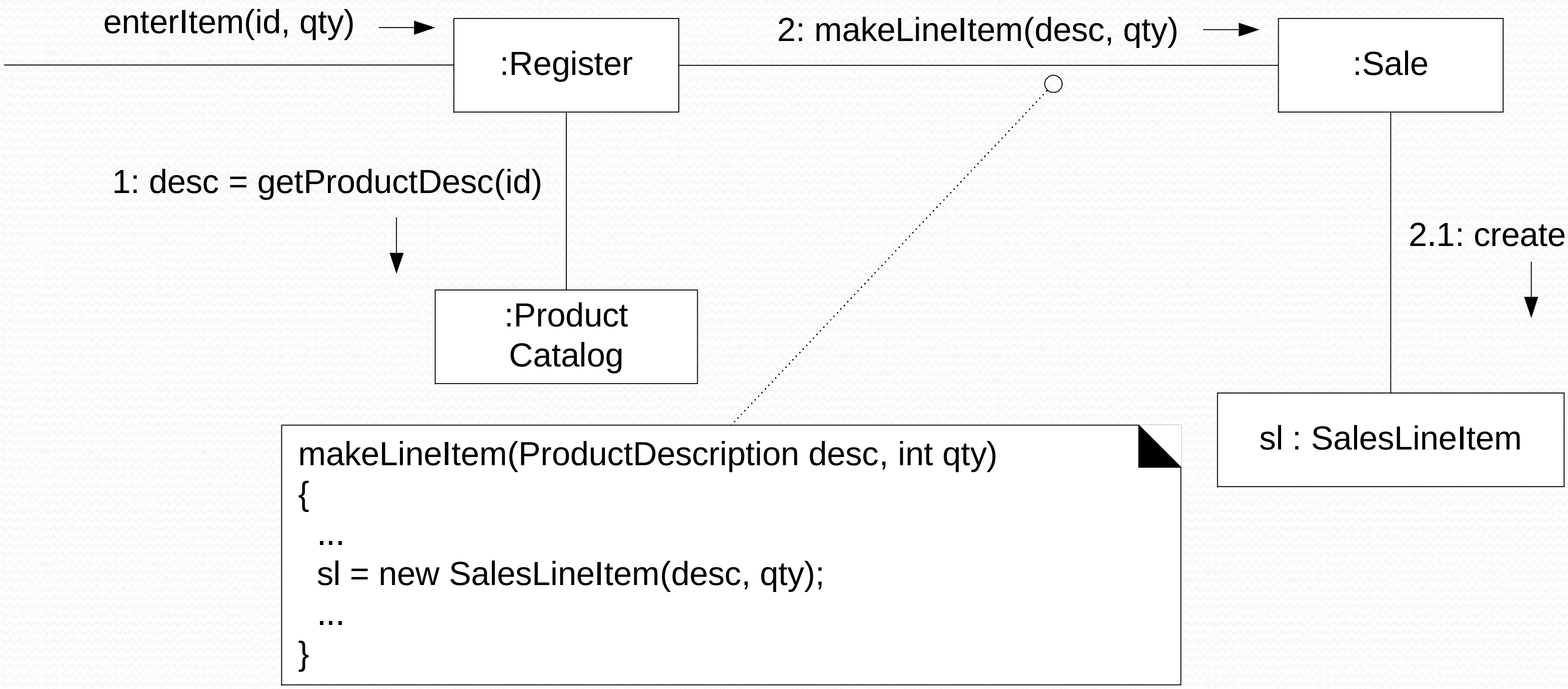
Parameter Visibility

- Parameter visibility exists when B is passed as a parameter to a method of A

- Temporary visibility as it persists only within the scope of the method

Parameter Visibility

Example of Register instance sends makeLineItem message to a Sale instance, a ProductDescription instance is passed as a parameter and Sale has Parameter visibility to a ProductDescription.

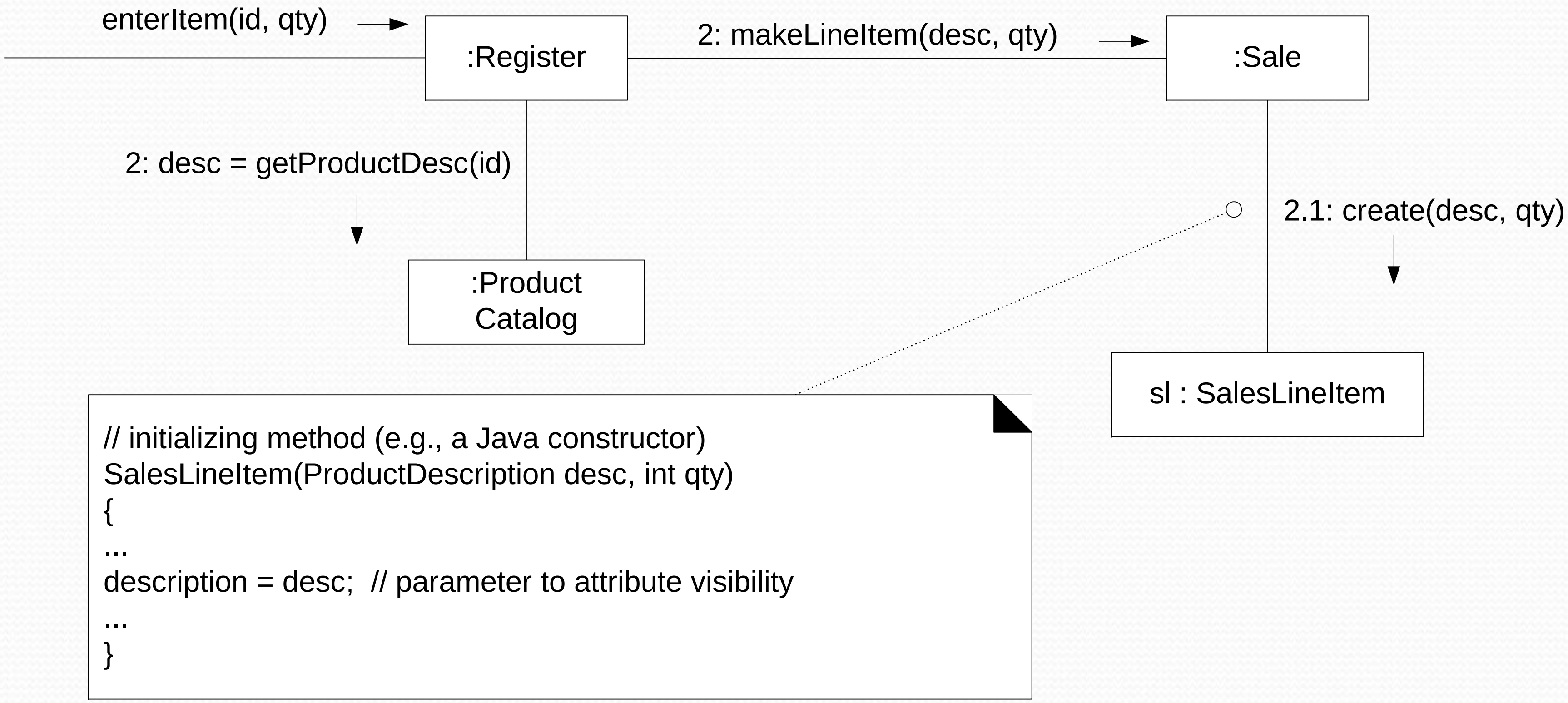


```
makeLineItem(ProductDescription desc, int qty)
{
  ...
  sl = new SalesLineItem(desc, qty);
  ...
}
```

Larman fig. 19.3

Parameter to Attribute Visibility

Common to transform Parameter visibility to Attribute visibility. Example: Sales creates a new SalesLineItem and passes the PRoductDescription to its initializing method (Java Constructor) where the parameter is assigned to an attribute.



```
// initializing method (e.g., a Java constructor)
SalesLineItem(ProductDescription desc, int qty)
{
...
description = desc;  // parameter to attribute visibility
...
}
```

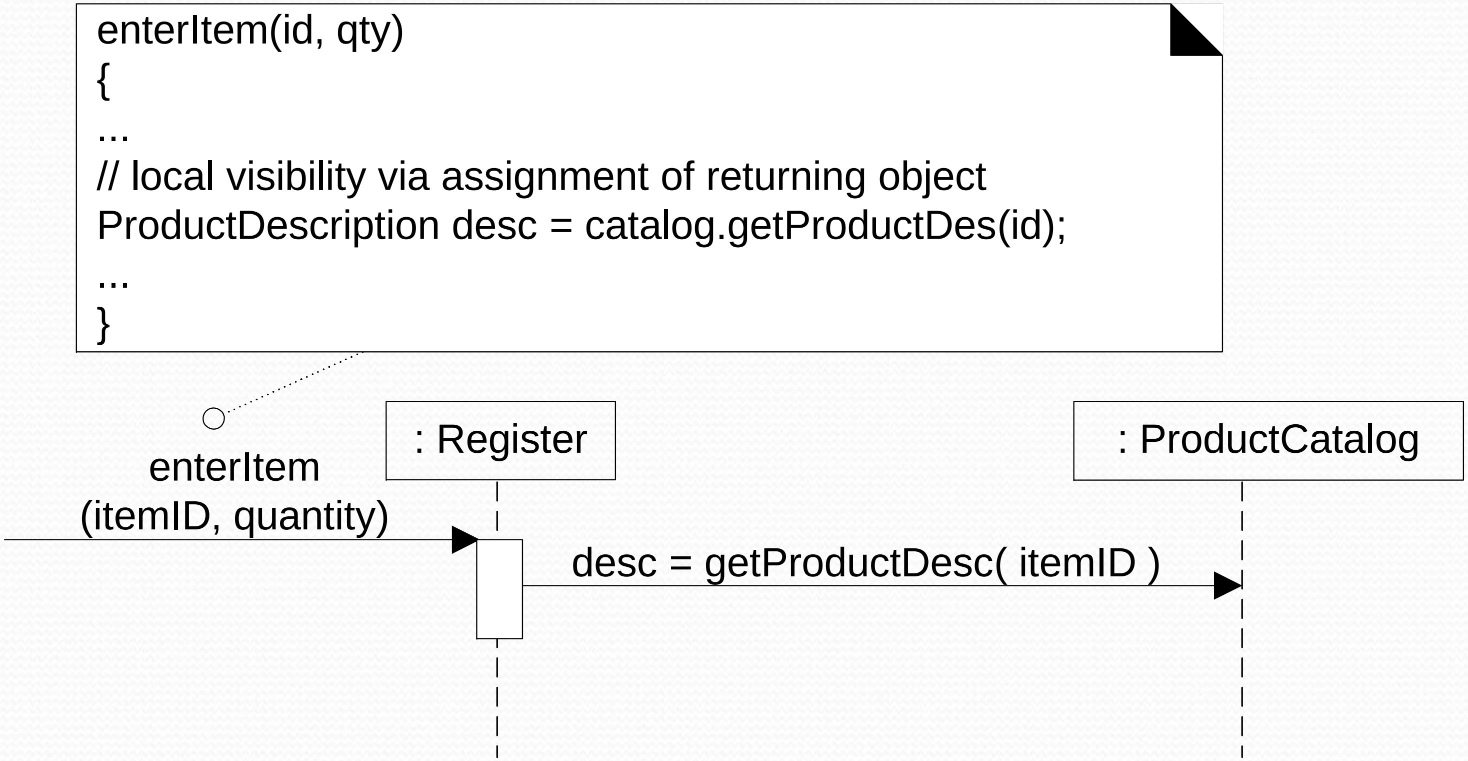
Larman fig. 19.4

Local Visibility

- Local visibility exists when B is declared as a local object within a method of A
- Temporary visibility as it persists only within the scope of the method
- Create a new instance and assign to a local variable
- Assign the returning object form a method invocation to a local variable
- Common to transform Local visibility to attribute visibility

Local Visibility

Example of Register’s enterItem method



```
enterItem(id, qty)
{
...
// local visibility via assignment of returning object
ProductDescription desc = catalog.getProductDes(id);
...
}
```

enterItem
(itemID, quantity)

: Register

: ProductCatalog

desc = getProductDesc(itemID)

Larman fig. 19.5

Global Visibility

- Global visibility exists when B is global to A
- Permanent visibility as it persists as long as A and B exist
- Assign an instance to a global variable (e.g. in C++) or use Singleton pattern

Remember Operations / Messages

- UML operation is a declaration with a name, parameters, return type, exceptions list, and possibly a set of constraints of pre and post-conditions.

  Visibility name (parameter-list) :return type {property-string}

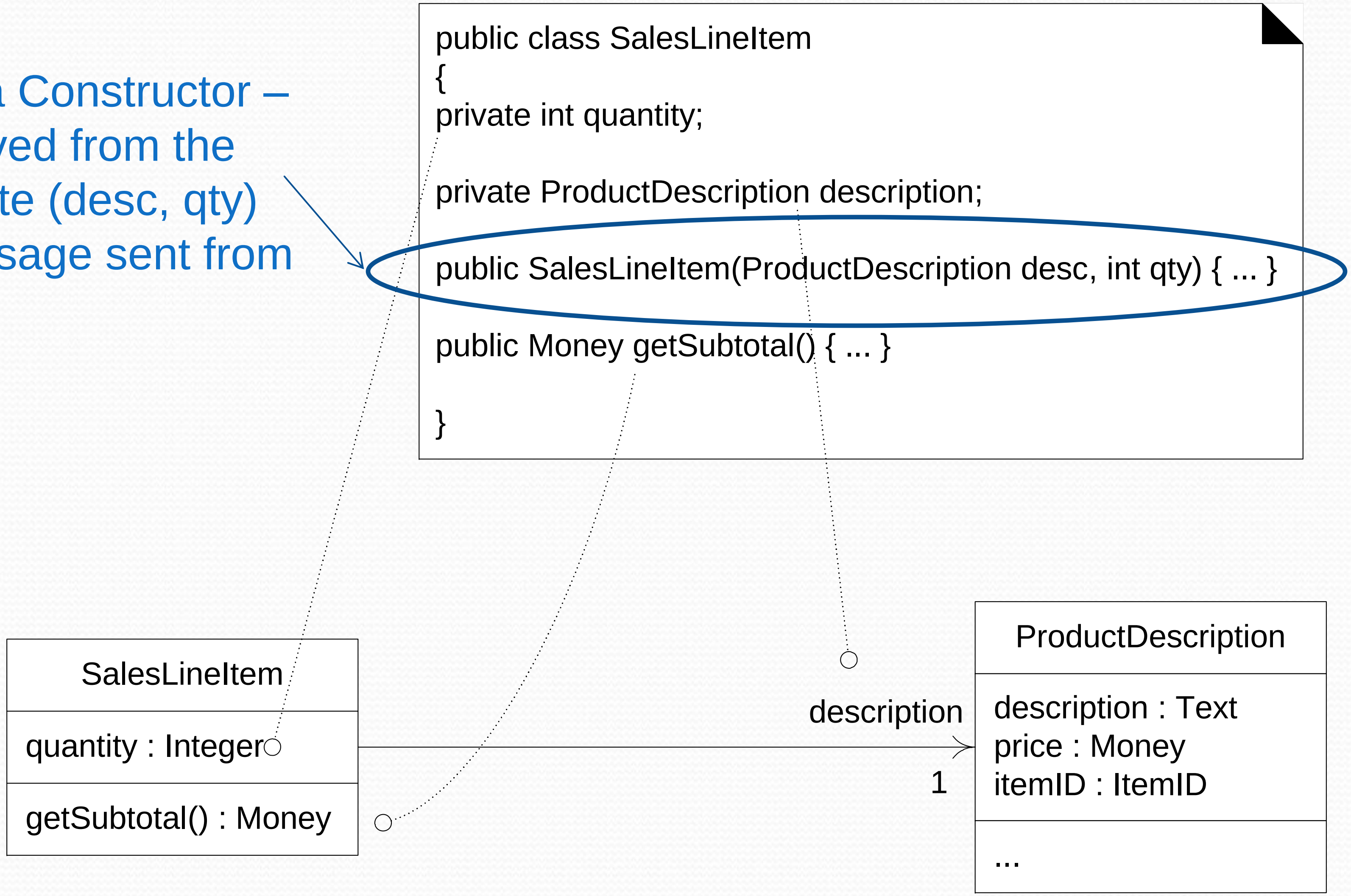
- Exceptions can be indicated in the property string in the declaration
- Visibility of operation is public by default
- Methods are implementation of operations

Ready To Map Design To Code

- DCD and Interaction Diagrams are pre-requisites
- Ready for UML Implementation Model
- Writing source code for class / interface definitions and method definitions
- Programming may lead to further changes and modifications to the Design Model – an iterative process

Map SalesLineItem Class To Code

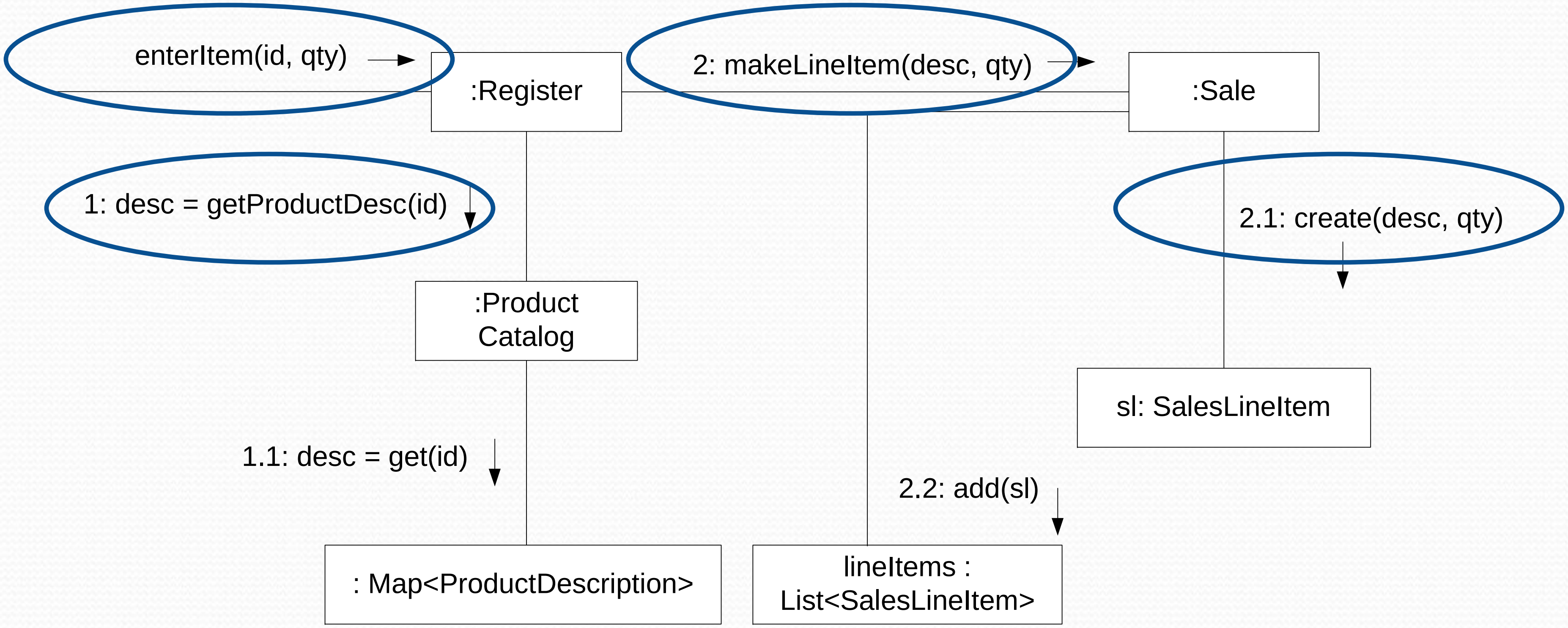
Java Constructor – derived from the create (desc, qty) message sent from Sale



```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

Larman fig. 20.1

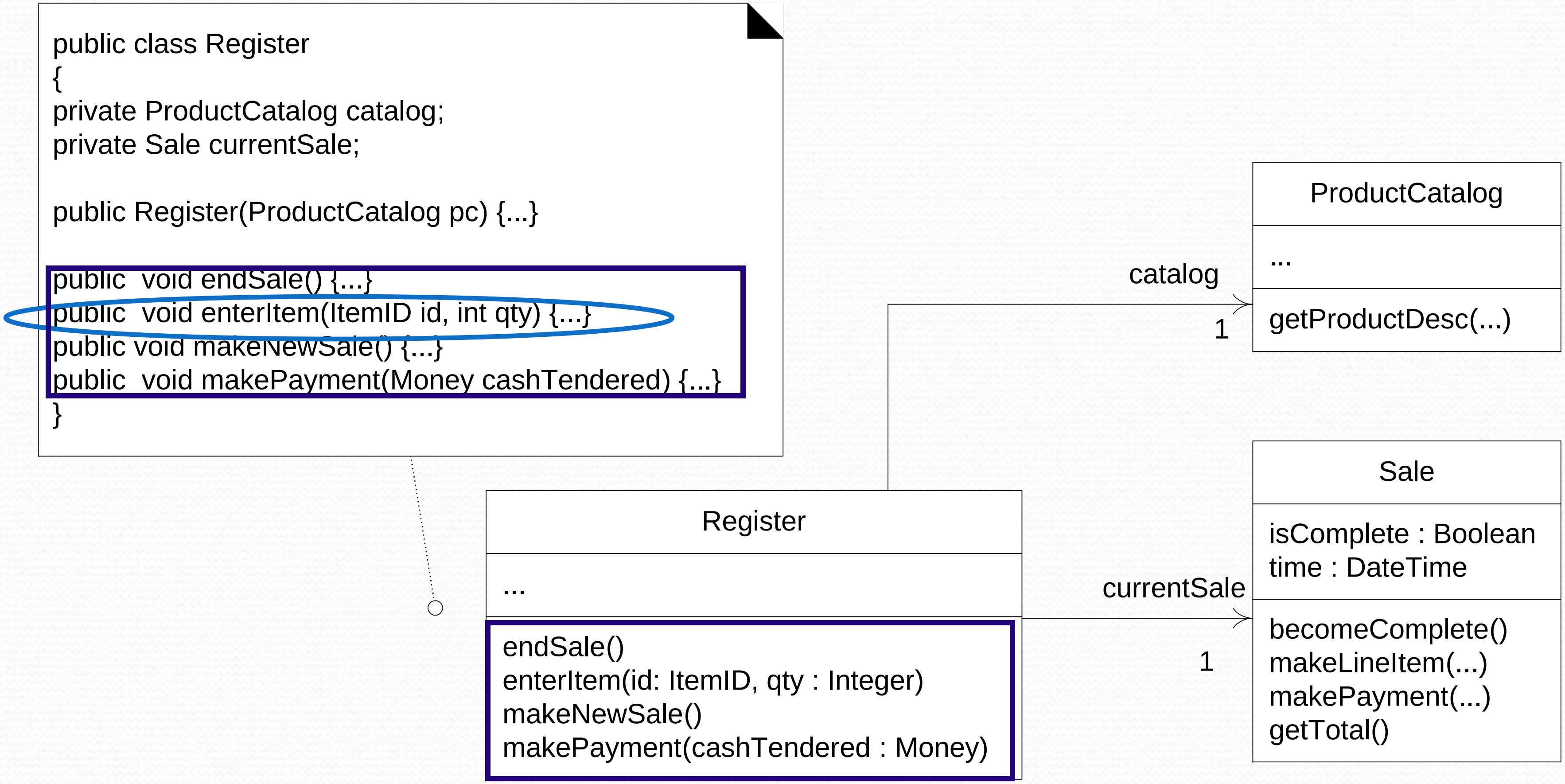
Interaction Diagram References



Map each sequenced message to a statement in a Java method

Larman fig. 20.2

Map Register (fascade) Class To Code

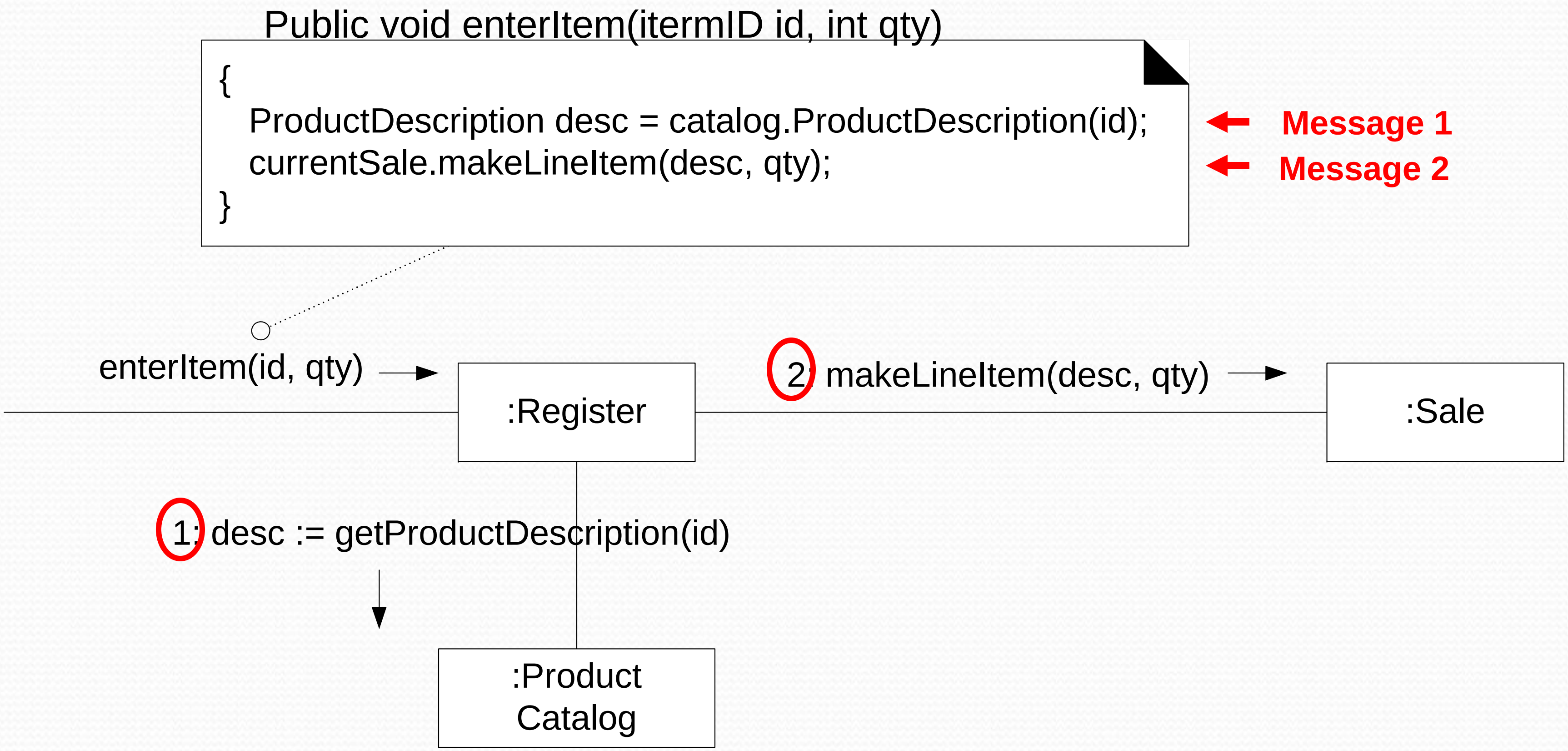


```
public class Register
{
private ProductCatalog catalog;
private Sale currentSale;

public Register(ProductCatalog pc) {...}

public  void endSale() {...}
public  void enterItem(ItemID id, int qty) {...}
public void makeNewSale() {...}
public  void makePayment(Money cashTendered) {...}
}
```

An enterItem method is defined in the Register Class since the enterItem message is sent to the Register

Larman fig. 20.3

enterItem Method defined in Register



Public void enterItem(itermID id, int qty)

```
{
   ProductDescription desc = catalog.ProductDescription(id);
   currentSale.makeLineItem(desc, qty);
}
```

Message 1

Message 2

enterItem(id, qty) →

:Register

2: makeLineItem(desc, qty) →

:Sale

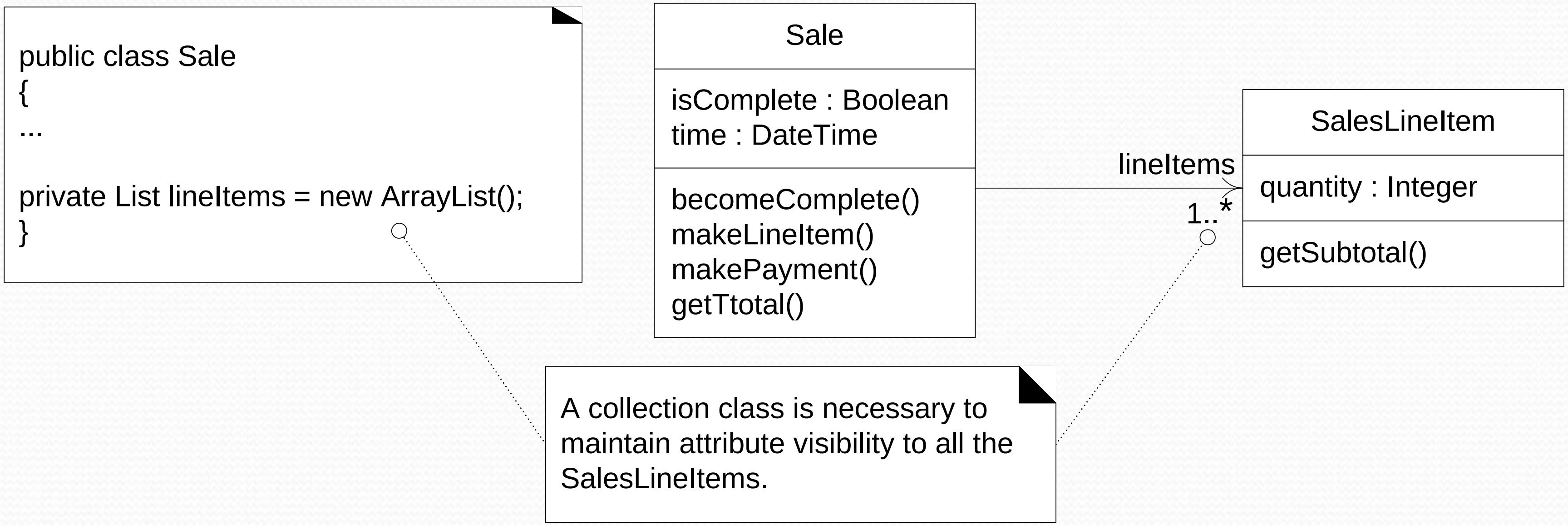
1: desc := getProductDescription(id) ↓

:Product Catalog

Example of the enterItem method defined in Register Class

Larman fig. 20.4

Collection in Sale Class



```
public class Sale
{
...

private List lineItems = new ArrayList();
}
```

Sale

isComplete : Boolean
time : DateTime

becomeComplete()
makeLineItem()
makePayment()
getTtotal()

lineItems

1..*

SalesLineItem

quantity : Integer

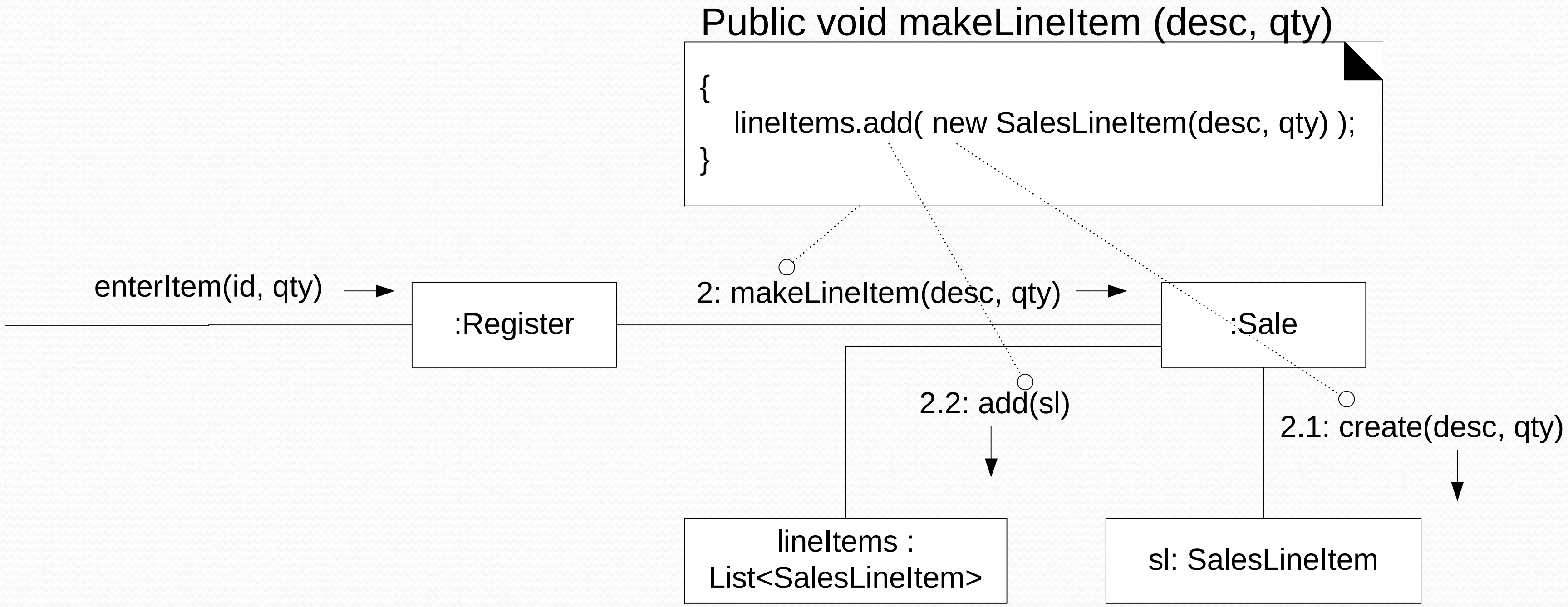
getSubtotal()

A collection class is necessary to maintain attribute visibility to all the SalesLineItems.

Use List or Map or Array for one-to-many relationships

Larman fig. 20.5

makeLineItem Method



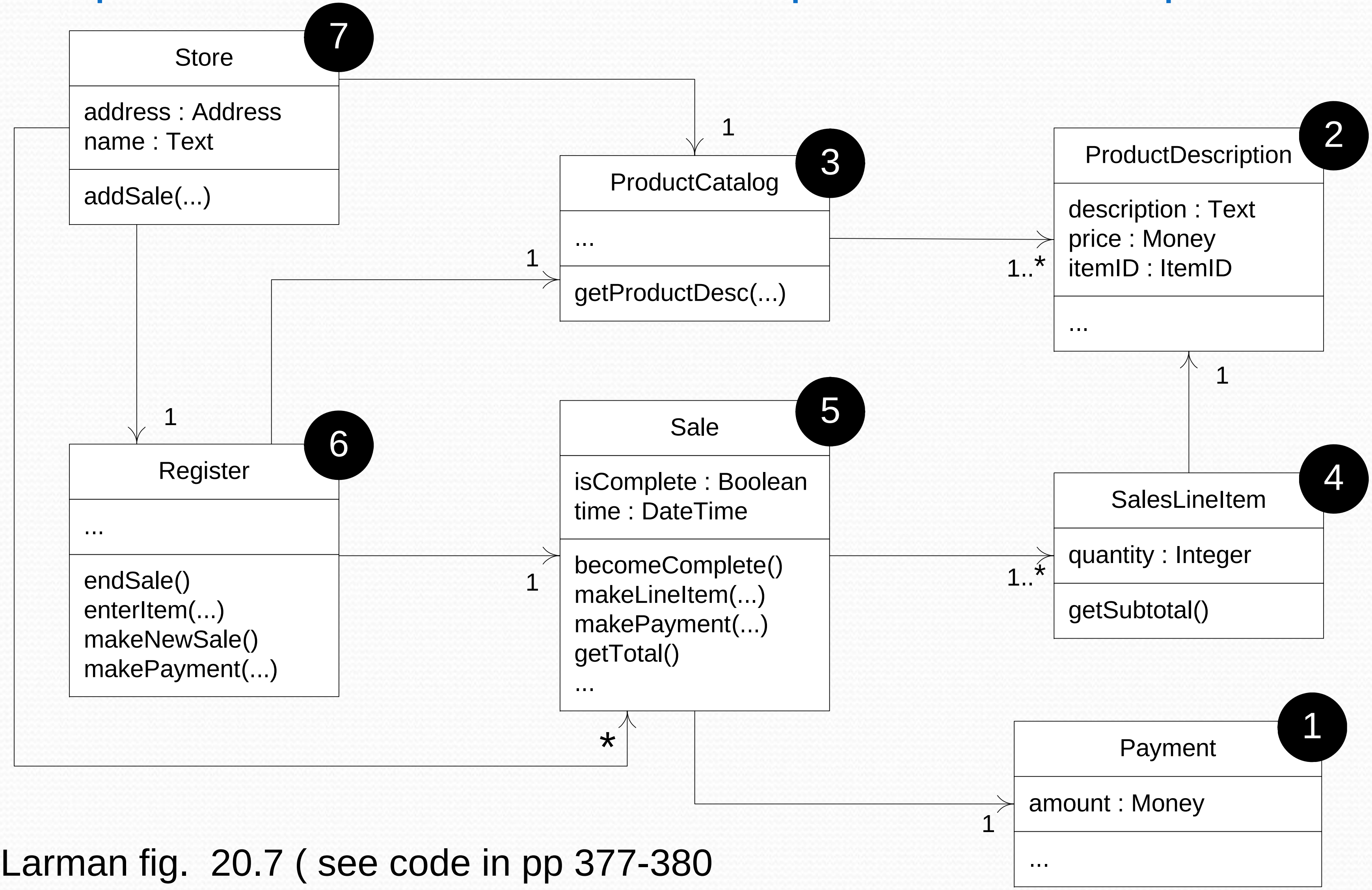
Public void makeLineItem (desc, qty)

```
{
    lineItems.add( new SalesLineItem(desc, qty) );
}
```

Larman fig. 20.6

Implementation Of DCD Classes

Implement classes from least-coupled to most-coupled



Larman fig. 20.7 (see code in pp 377-380