

# Applying UML & Patterns (3<sup>rd</sup> ed.)

## Chapter 20

### MAPPING DESIGNS TO CODE

**This document may not be used or altered without the express permission of the author.**

**Dr. Glenn L. Ray**  
School of Information Sciences  
University of Pittsburgh  
[gray@sis.pitt.edu](mailto:gray@sis.pitt.edu) 412-624-9470

# Design -> Code

- Implementation Model
  - The ultimate deliverables
    - Source code, DB definitions, etc.
  - Agile design does NOT preclude prototyping
  - Design -> code is more than mechanical code generation
    - Especially with agile modeling because many details skipped intentionally
    - Developers may have good ideas that improve design

# Design -> Code

- Two primary tasks
  - Define classes & interfaces
    - Get these from design class diagram
    - Many tools can generate them
  - Define methods

# Fig. 20.1

Constructor usually not  
Shown on OO diagram

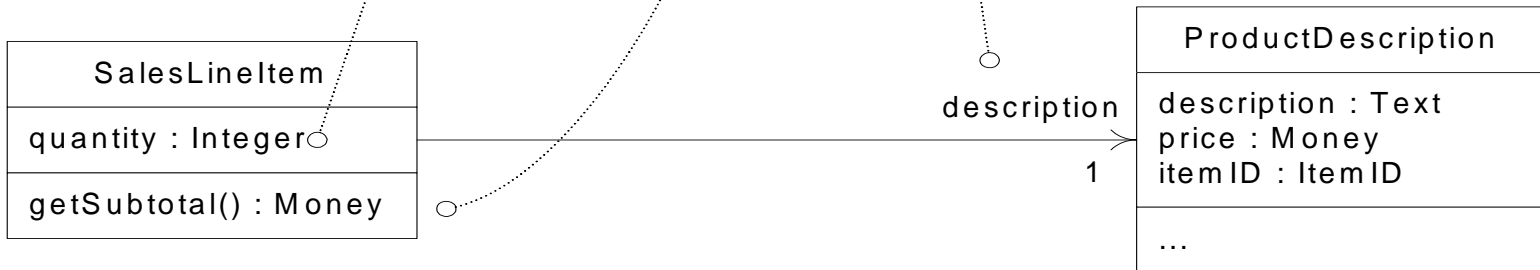
```
public class SalesLineItem
{
    private int quantity;

    private ProductDescription description;

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

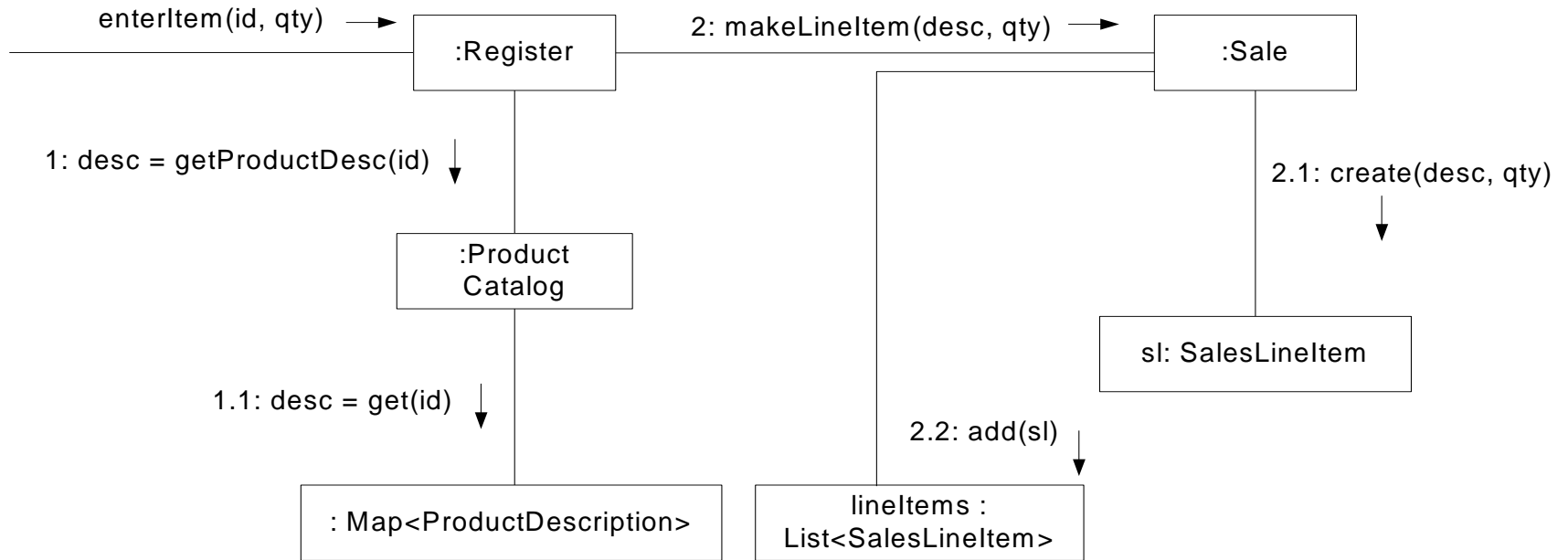
    public Money getSubtotal() { ... }

}
```

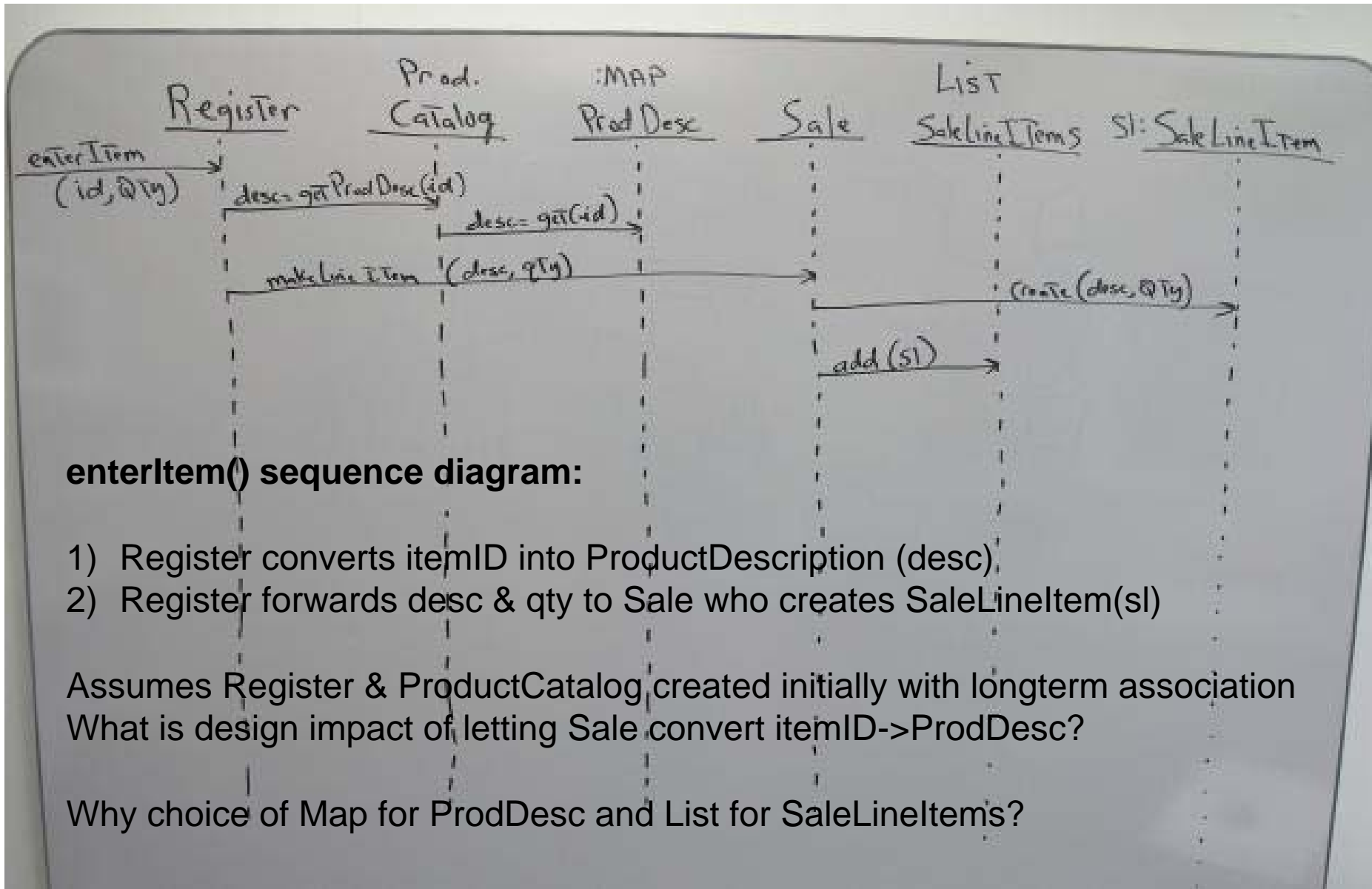


A mapping from class diagram to code

# Fig. 20.2



`enterItem()` system operation



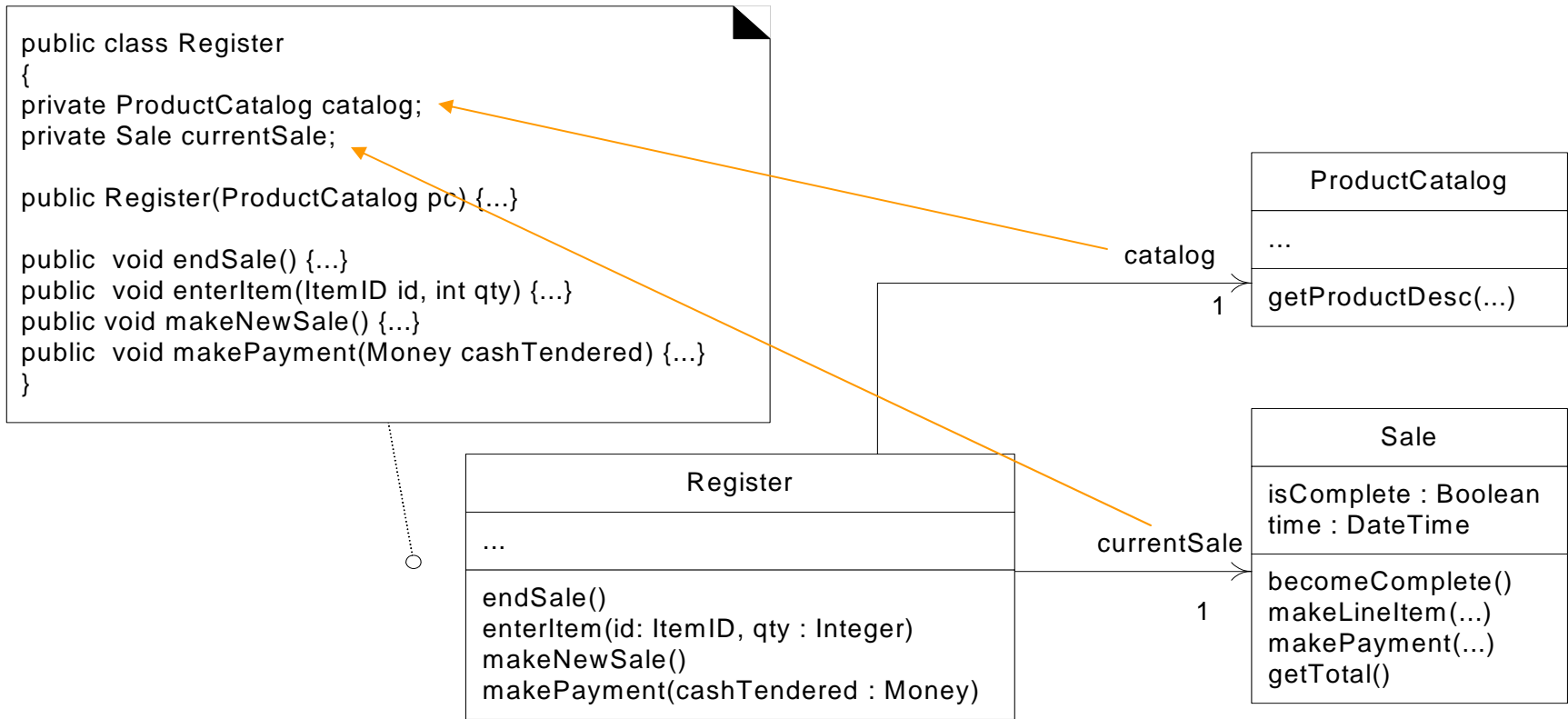
**enterItem() sequence diagram:**

- 1) Register converts itemID into ProductDescription (desc)
- 2) Register forwards desc & qty to Sale who creates SaleLineItem(sl)

Assumes Register & ProductCatalog created initially with longterm association  
 What is design impact of letting Sale convert itemID->ProdDesc?

Why choice of Map for ProdDesc and List for SaleLineItems?

# Fig. 20.3

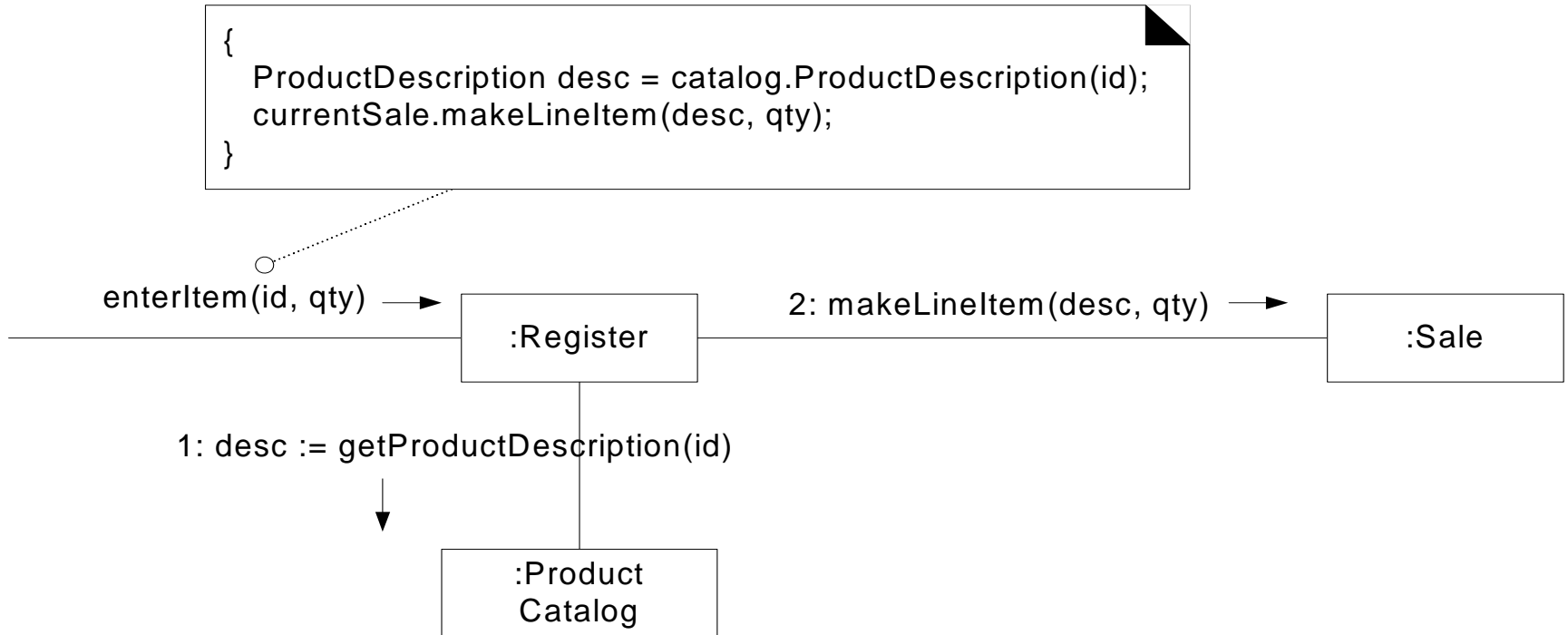


## Definition of Register class

Every method shown in OO diagram maps to code

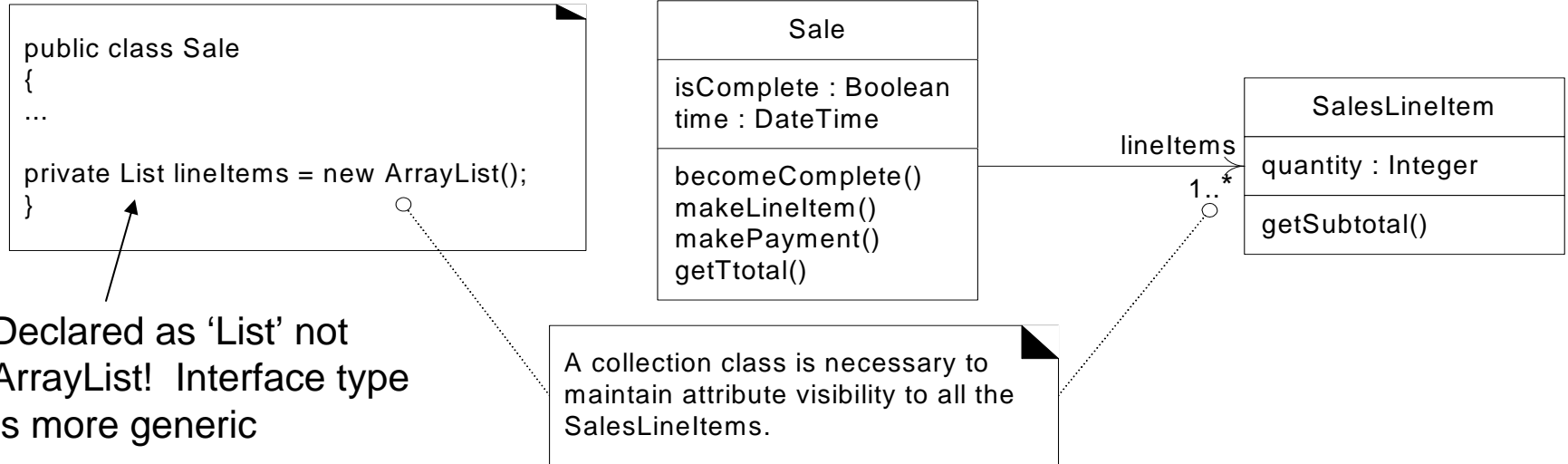
Both associations map to attributes

# Fig. 20.4



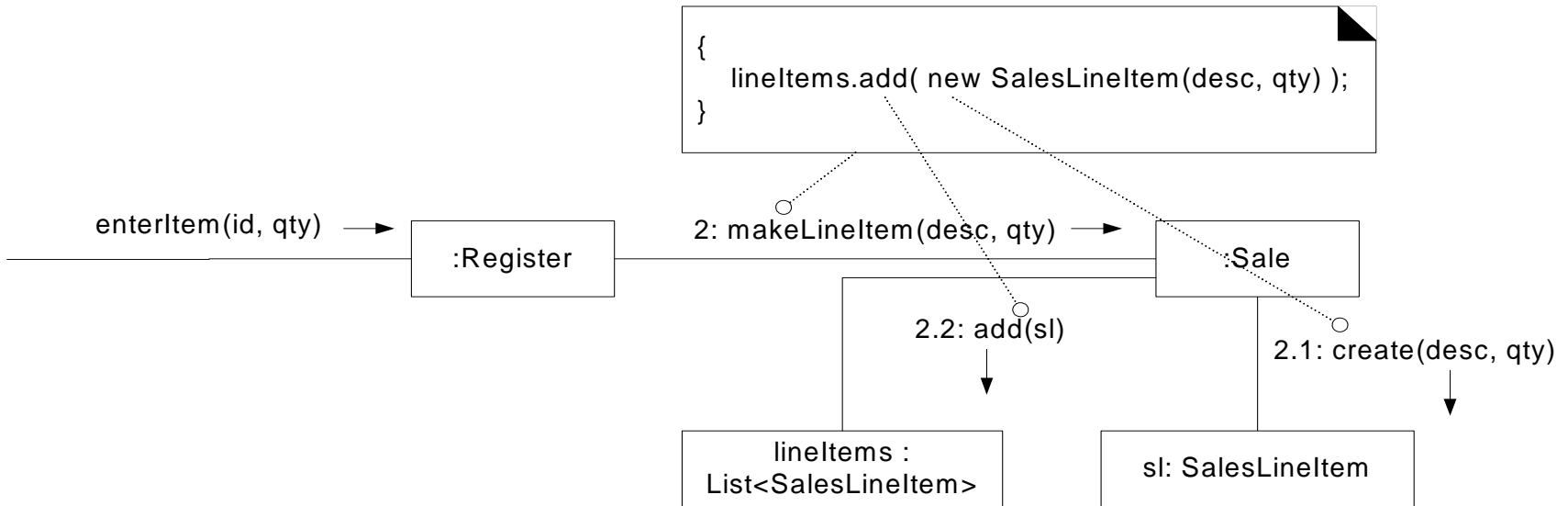
Each message maps to a method call within `enterItem()`

# Fig. 20.5

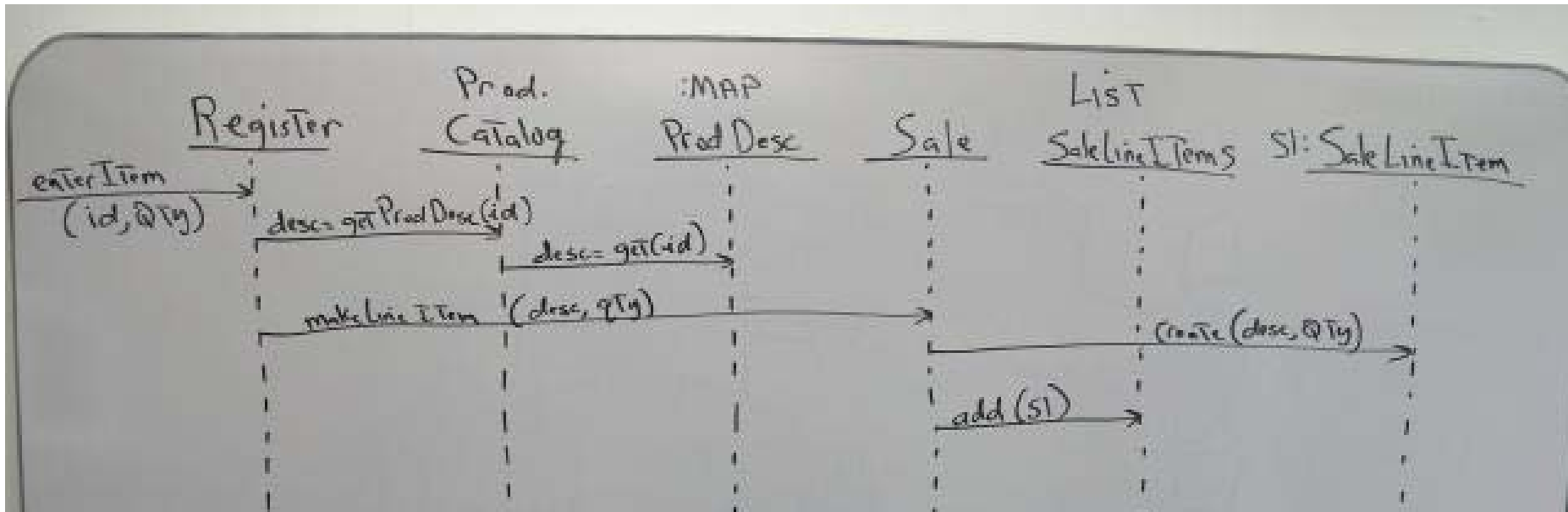


Adding a collection  
Required by 1-to-many associations

# Fig. 20.6



Sale.makeLinItem(), part of the enterItem() interaction diagram  
Notice how simply the method can be implemented!



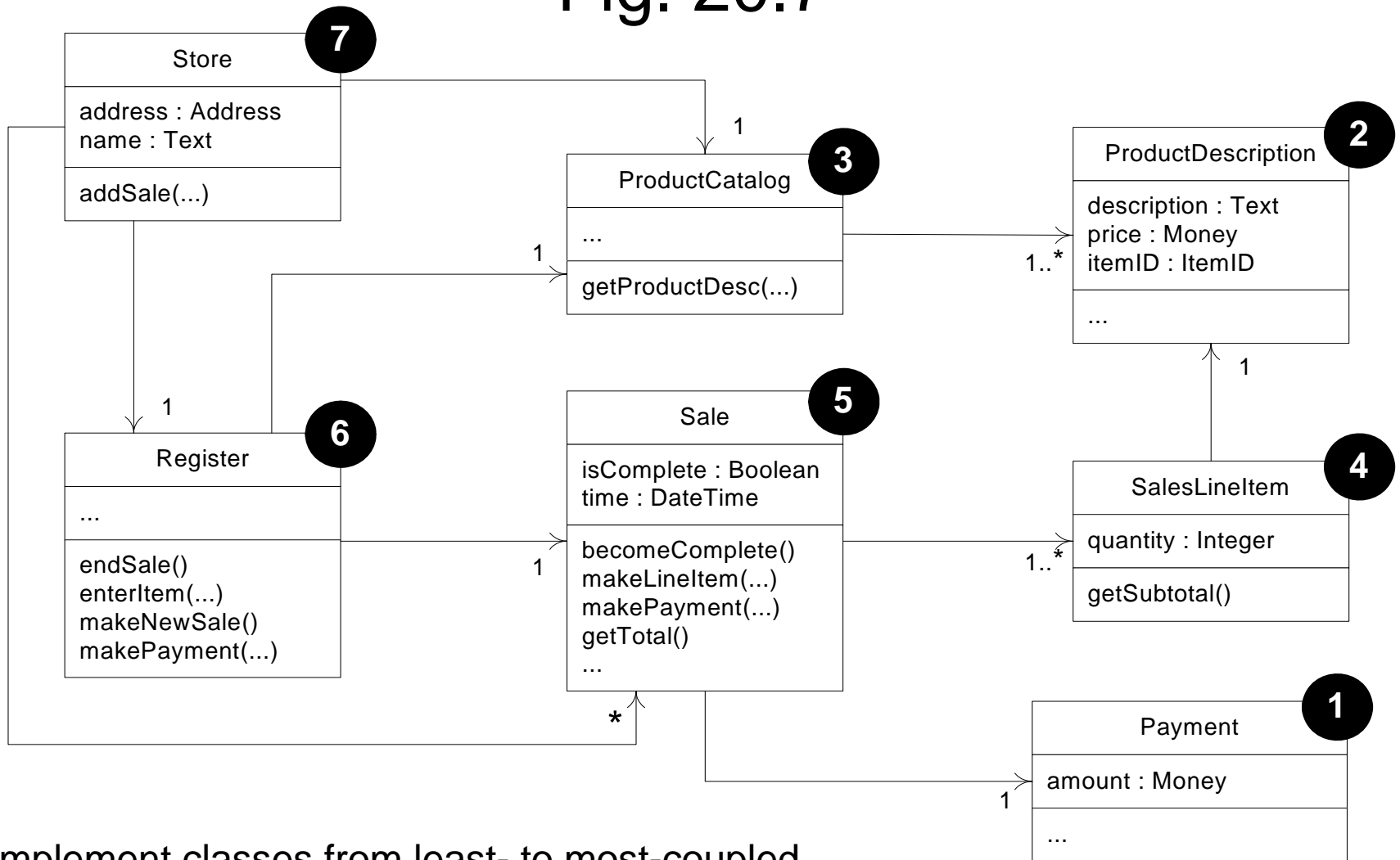
**enterItem() sequence diagram:**

- 1) Register converts itemID into ProductDescription (desc)
- 2) Register forwards desc & qty to Sale who creates SaleLineItem(sl)

Assumes Register & ProductCatalog created initially with longterm association  
 What is design impact of letting Sale convert itemID->ProdDesc?

Why choice of Map for ProdDesc and List for SaleLineItems?

# Fig. 20.7



Implement classes from least- to most-coupled

Payment should be first class implemented, Store should be last

See any errors in diagram?