THE UNIVERSITY OF MELBOURNE

SWEN30006
Software Modelling and Design

MAPPING DESIGNS TO CODE

Larman Chapter 20

Beware of bugs in the above code;
I have only proved it correct, not tried it.

—Donald Knuth

Objectives

On completion of this topic you should be able to:

- Map design artefacts to code in an object-oriented language.

Mapping Designs to Code

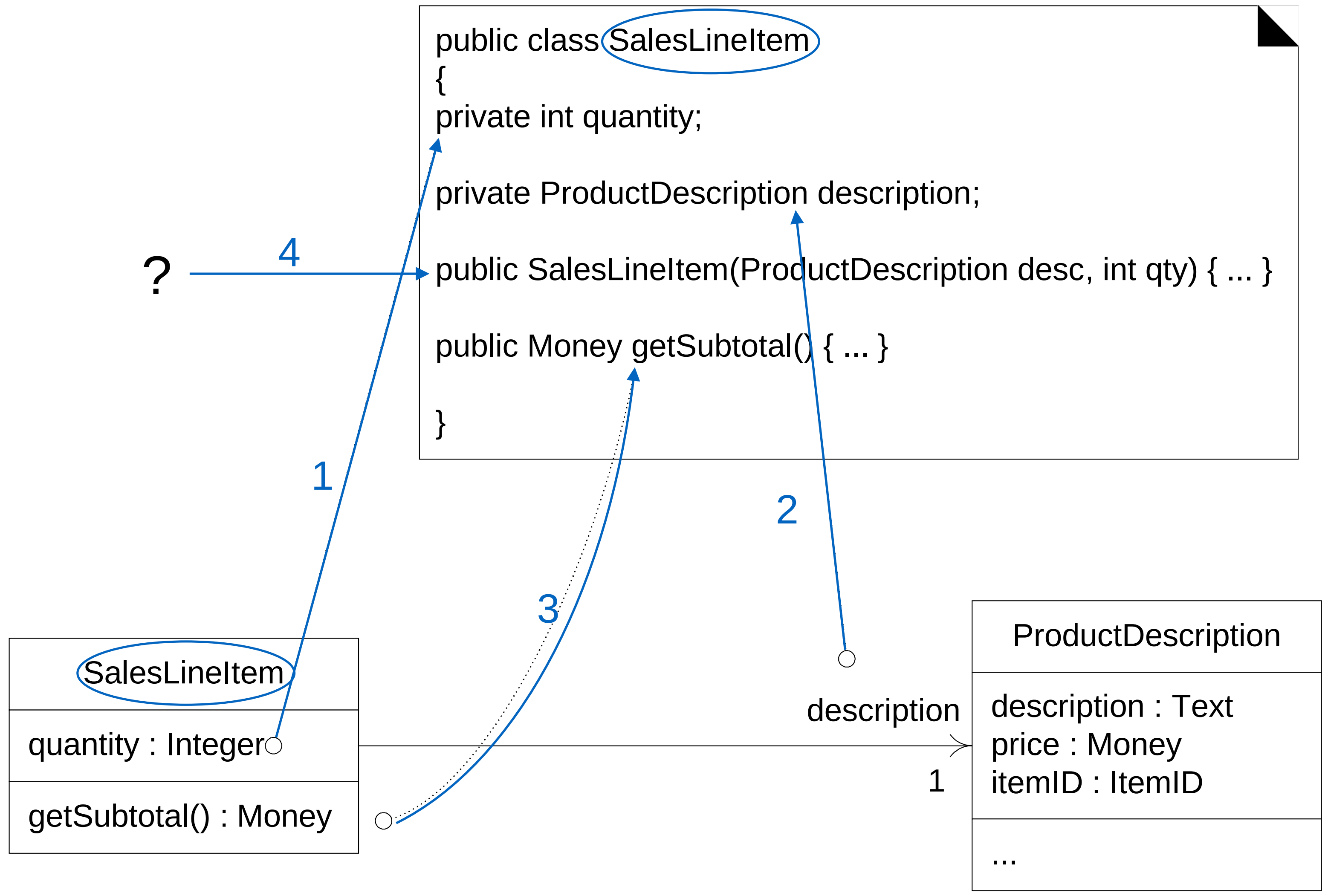
OO implementation requires writing:

- Class and interface definitions
- Method definitions

Mappings covered here are relatively mechanical.

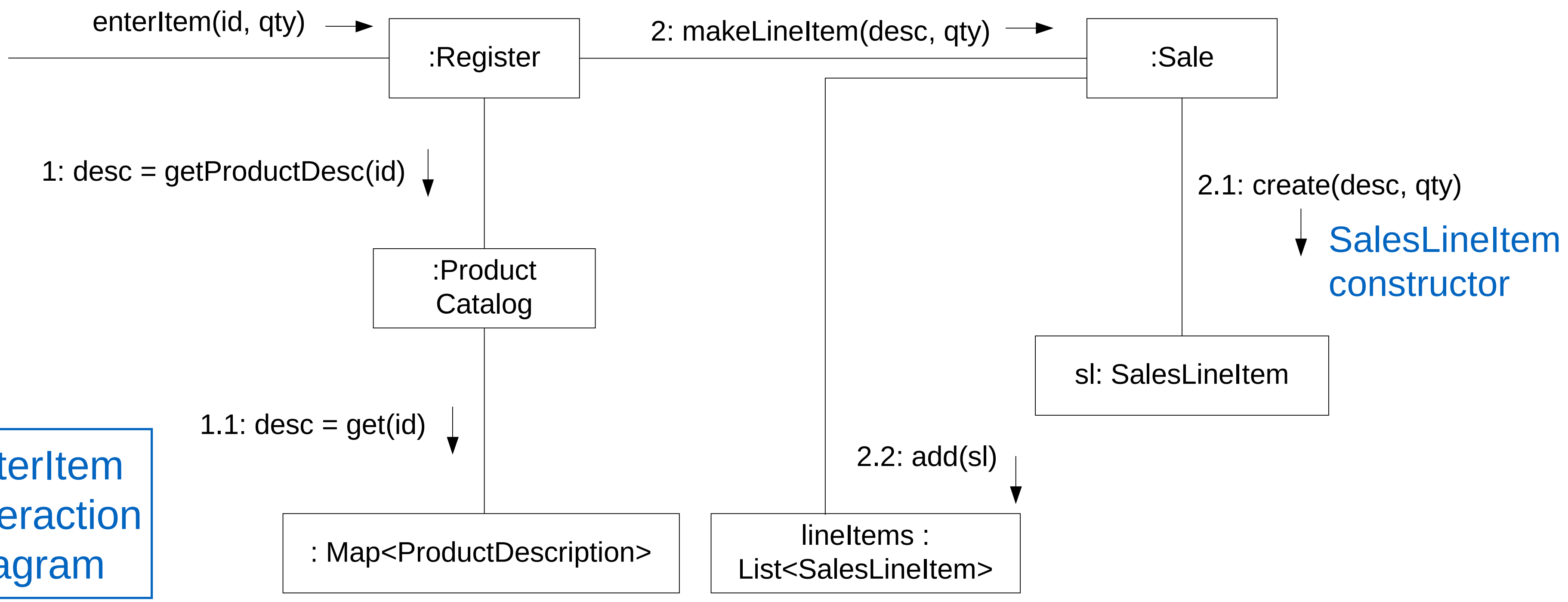
- Design should be a great basis; nonetheless,
- Programming in general is iterative and creative.

Creating Class Definitions from DCDs



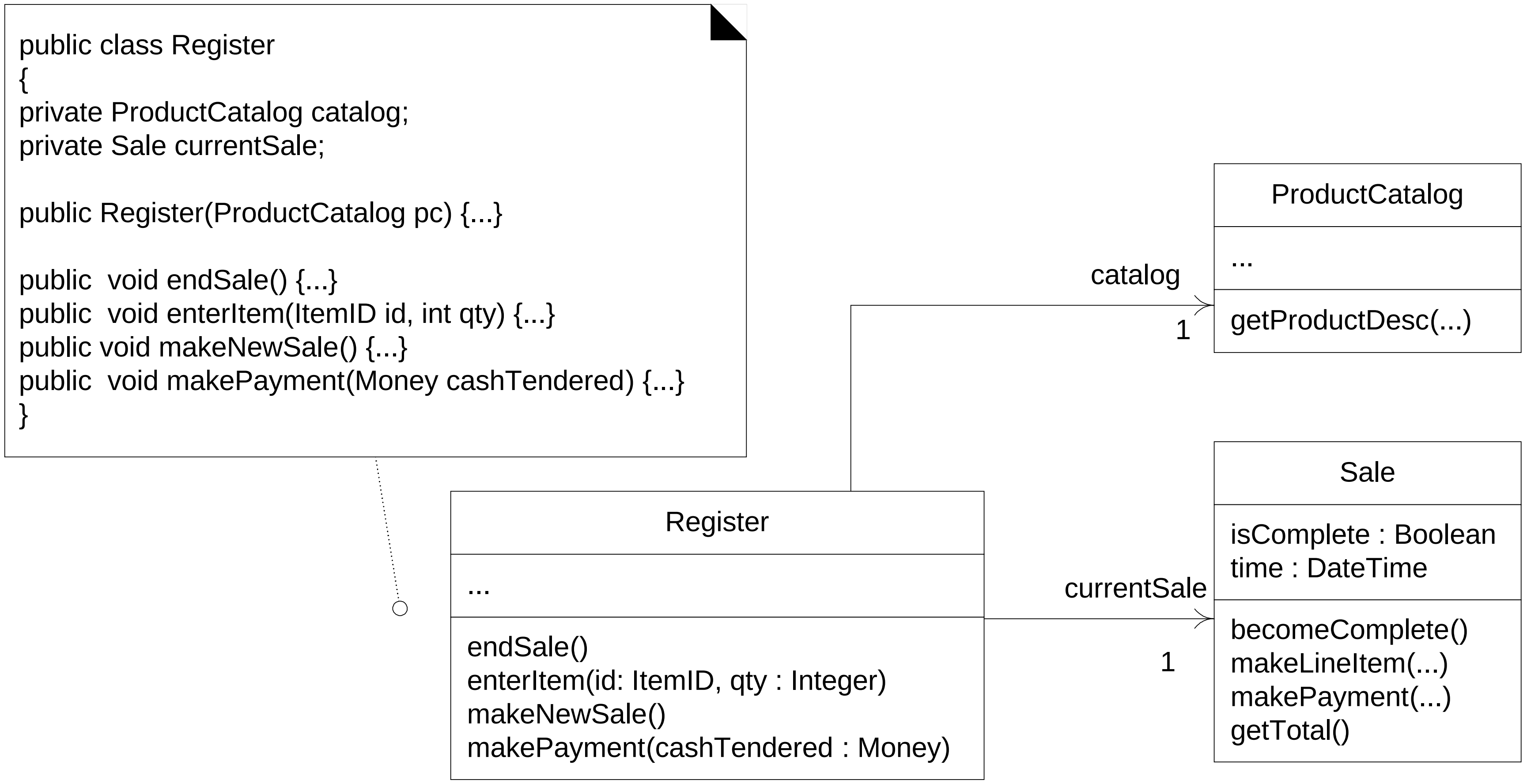
```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

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

public Money getSubtotal() { ... }

}
```

Creating Methods from Interaction Diagrams


enterItem(id, qty)
:Register
2: makeLineItem(desc, qty)
:Sale
1: desc = getProductDesc(id)
2.1: create(desc, qty)
SalesLineItem constructor
:Product Catalog
sl: SalesLineItem
1.1: desc = get(id)
2.2: add(sl)
: Map<ProductDescription>
lineItems : List<SalesLineItem>

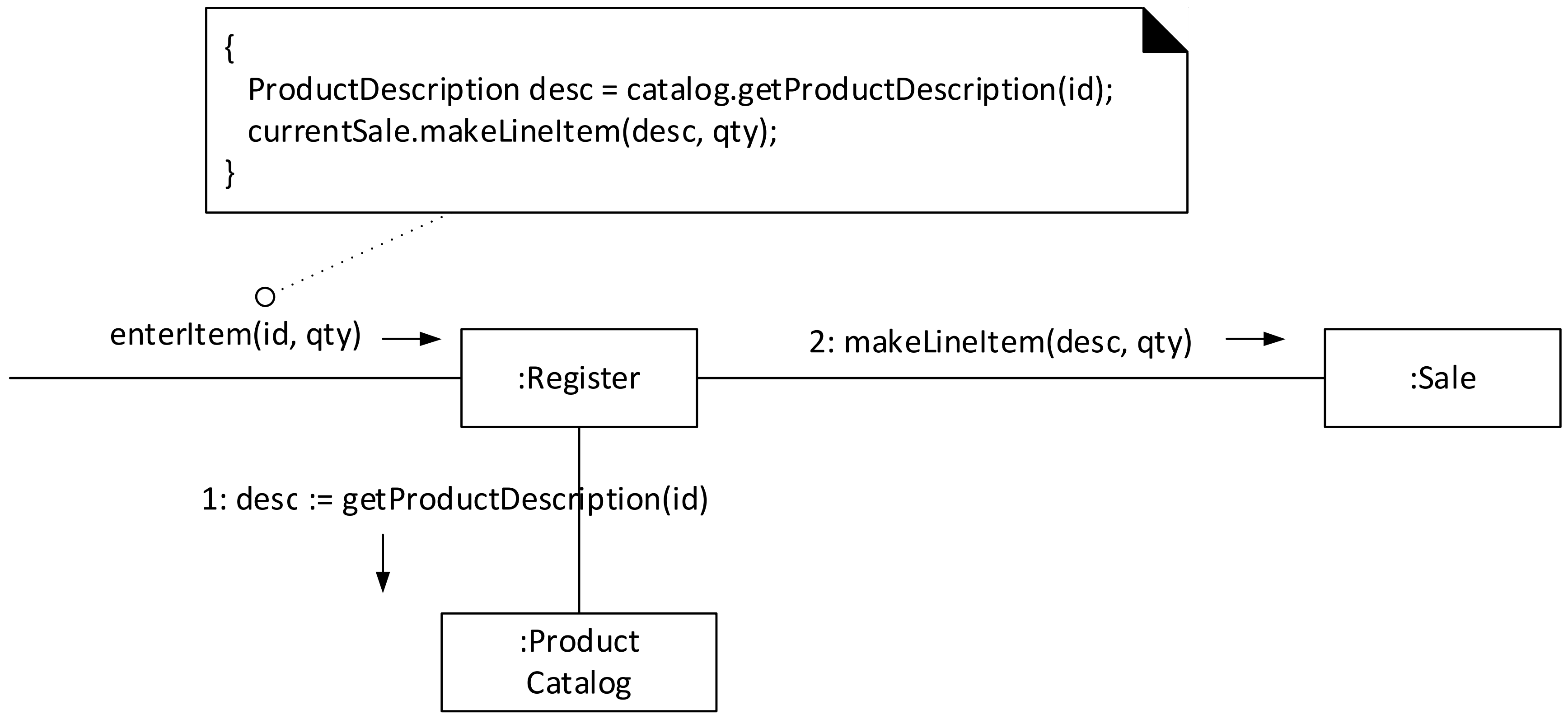
enterItem Interaction Diagram

Create Register Class from DCD



```
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) {...}
}
```

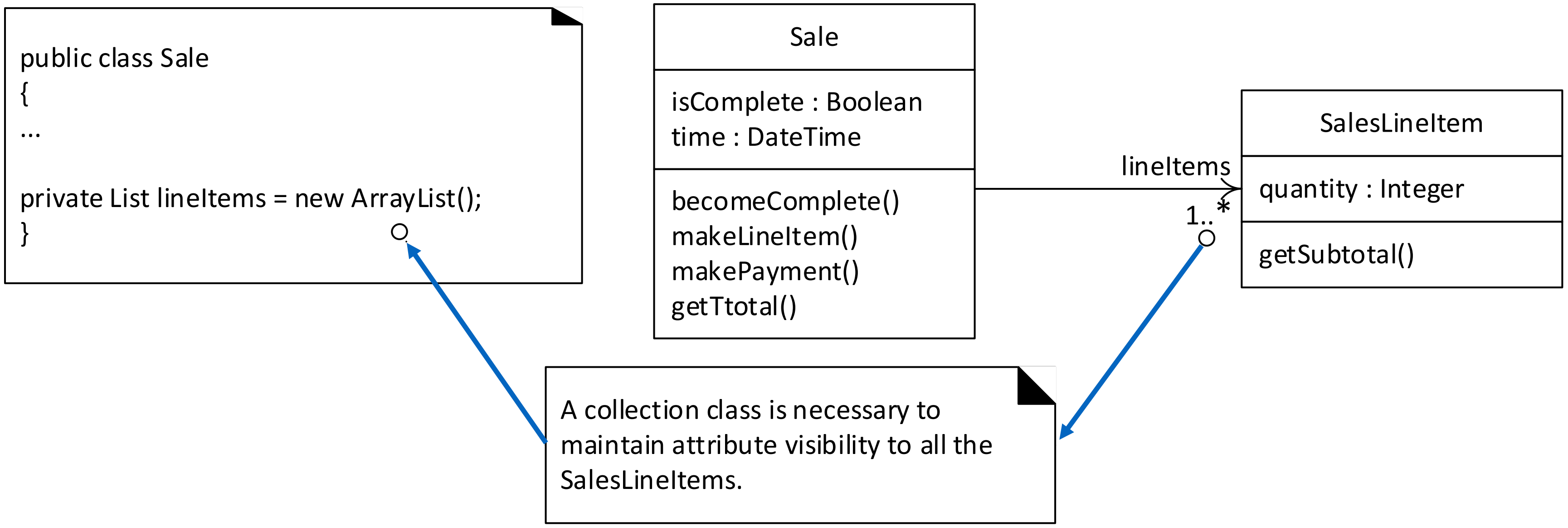
Create Register.enterItem Method



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

enterItem(id, qty)
:Register
2: makeLineItem(desc, qty)
:Sale
1: desc := getProductDescription(id)
:Product Catalog

Multiplicity *: Adding a Collection



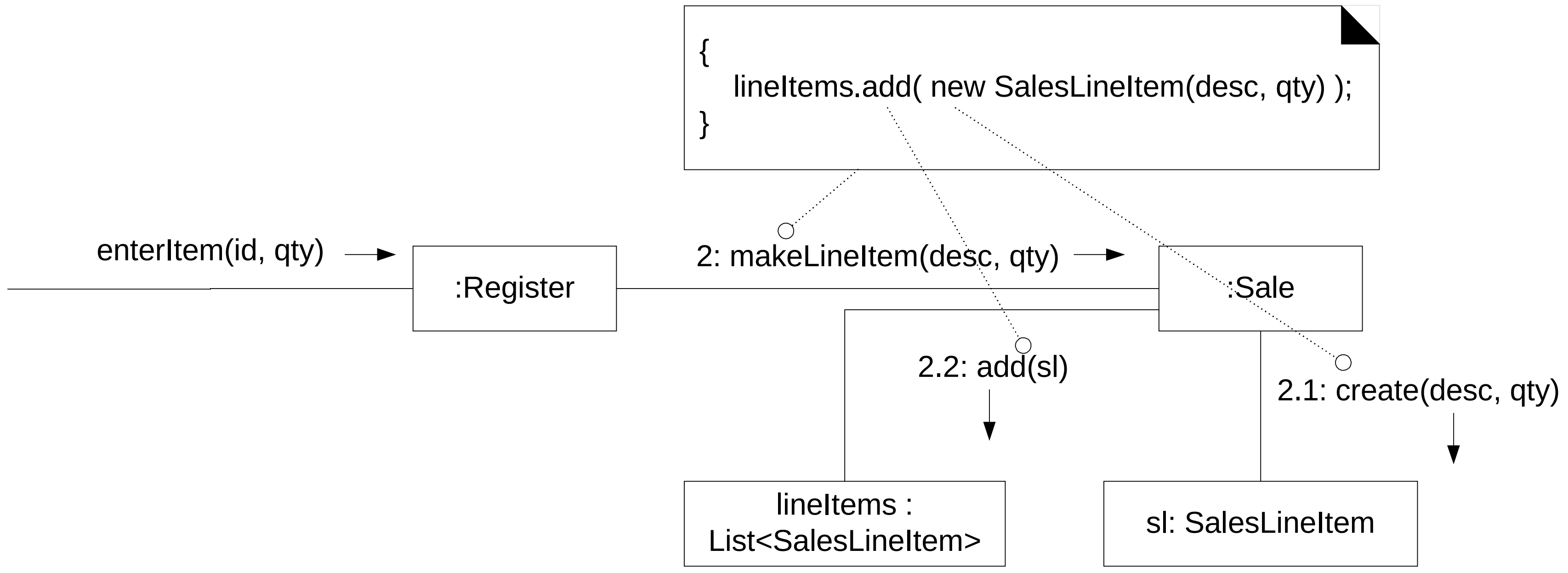
```
public class Sale
{
...

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

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

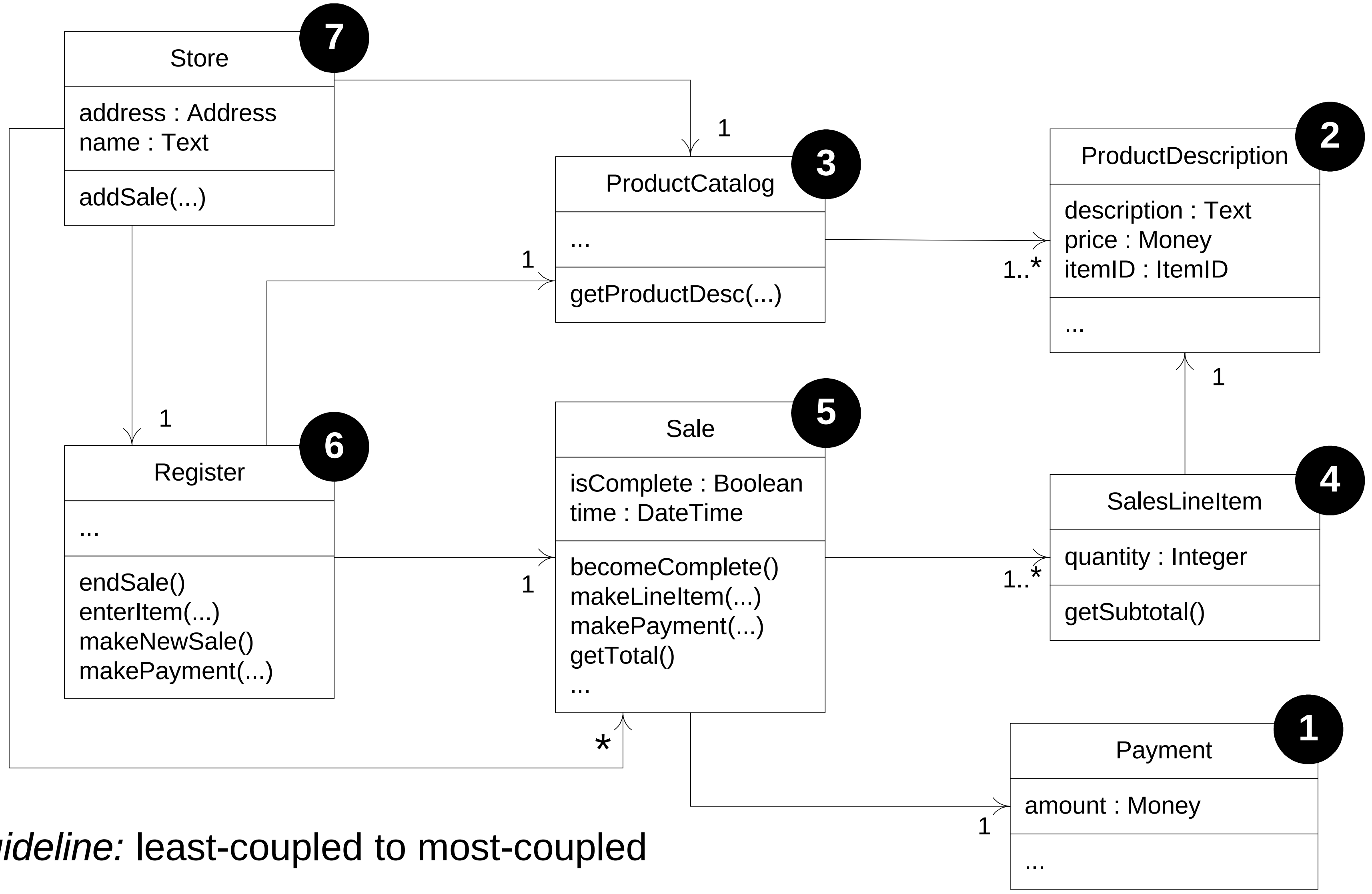
Guidelines:

- Chose collection class supporting required operations
  - E.g. Key-based lookup -> *Map*; Growing ordered list -> *List*
- If it implements an interface, declare in terms of the interface
  - E.g. *Map<String, Integer> m = new HashMap<String, Integer>();*

Create Sale.makeLineItem Method


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

Possible Order: Implement/Test



Guideline: least-coupled to most-coupled

Summary

- Design provides a strong basis for implementation
  - Both static and dynamic models play a role
  - Much of the mapping is mechanical
- Implementation should match design by default
  - Thoughtful variation may be required
- Guideline: Work from least to most coupled