Designing for Visibility & Mapping to Code

CSSE 574: Session 4, Part 3

Steve Chenoweth

Phone: Office (812) 877-8974

Cell (937) 657-3885 Email: chenowet@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Agenda

- **Designing for Visibility**
- **Mapping Designs to Code**

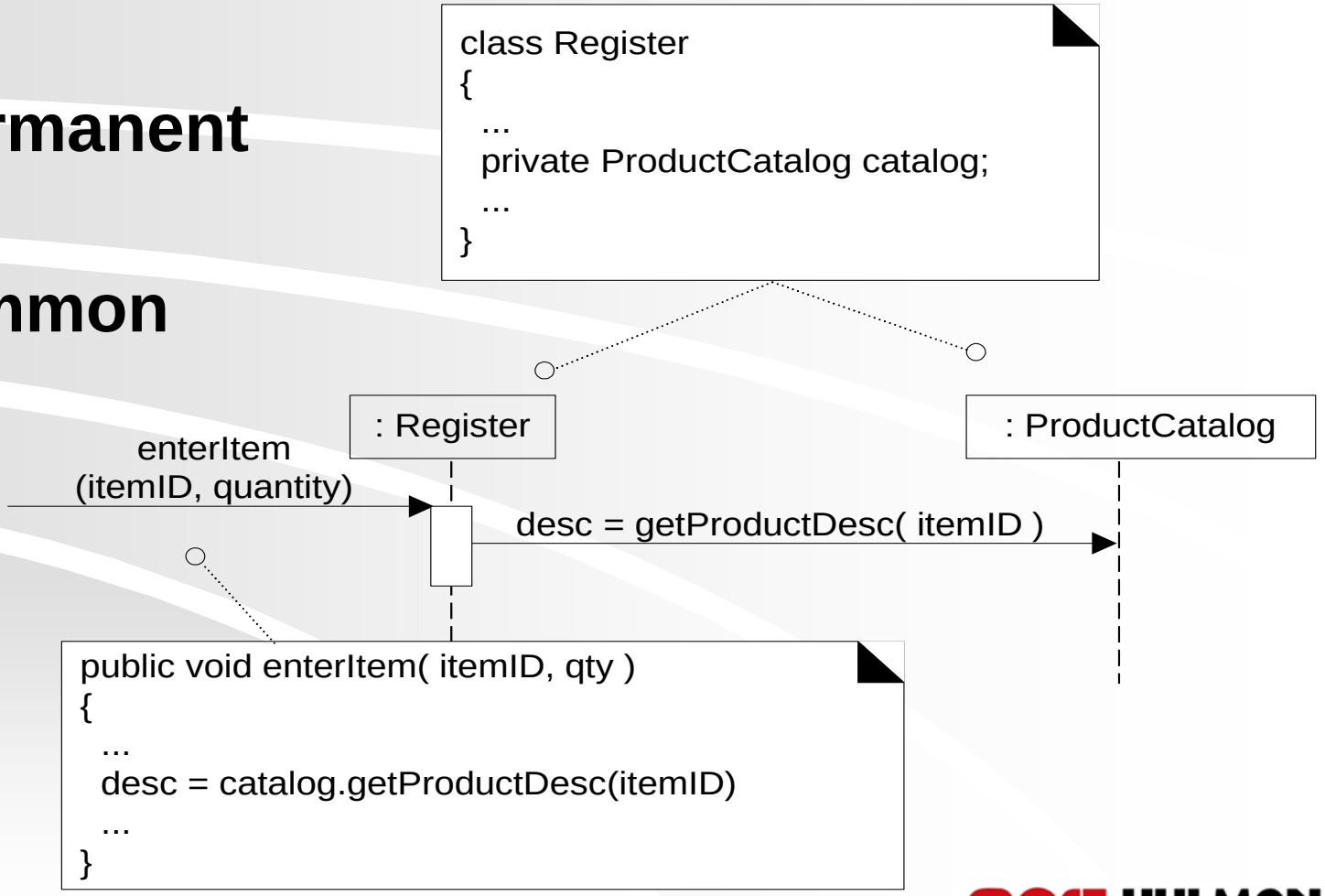
Visibility – How the drive back to TH felt after class Wed, 12-15-2010. From http://www.hydrometeoindustry.org/catalogue/ProductSheets/Vaisala/RoadWeather/VaisalaRoadWeatherSheet.htm.

Visibility

- An object *B* is visible to an object *A* if *A* can send a message to *B*
- Related to, but not the same as:
  - Scope
  - Access restrictions (*public*, *private*, etc.)
- What are four common ways that *B* can be visible to *A*?

Attribute Visibility

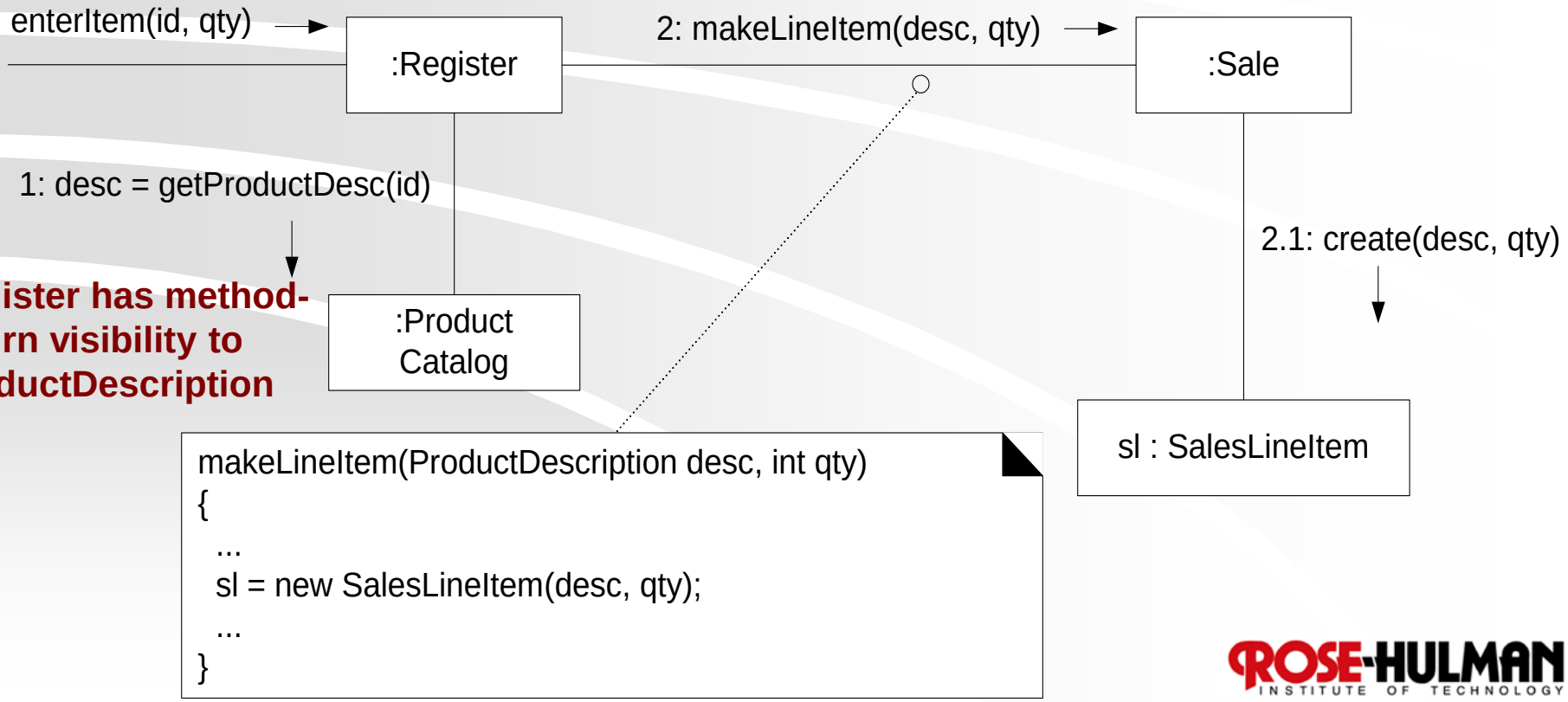
- **Object *A* has attribute visibility to object *B* if … *A* has an attribute that stores *B***
- **Quite permanent**
- **Most common**



Parameter Visibility

Object *A* has parameter visibility to object *B* if … *B* is passed in as an argument to a method of *A*

- Not permanent, disappears when method ends
- Second most common
- Methods often convert parameter visibility to attribute visibility



Local Visibility

- Object *A* has local visibility to object *B* if …
  - *B* is referenced by a local variable in a method of *A*
- Not permanent, disappears when leaving variable's scope
- Third most common
- Methods often convert local visibility to attribute visibility

Global Visibility

- Object *A* has global visibility to object *B* if …
  - *B* is stored in a global variable accessible from *A*
- Very permanent
- Least common (but highest coupling risk)

Cartoon of the Day

Used with permission. http://notinventedhe.re/on/2009-9-23

Before we get into Code

- Created Domain Model from requirements and use cases
- Used System Sequence Diagrams to identify system operations
- Clarified system operations with Operation Contracts
- Assigned “doing” responsibilities with Interaction Diagrams (Communication and Sequence Diagrams)
- Assigned “knowing” responsibilities with Design Class Diagrams

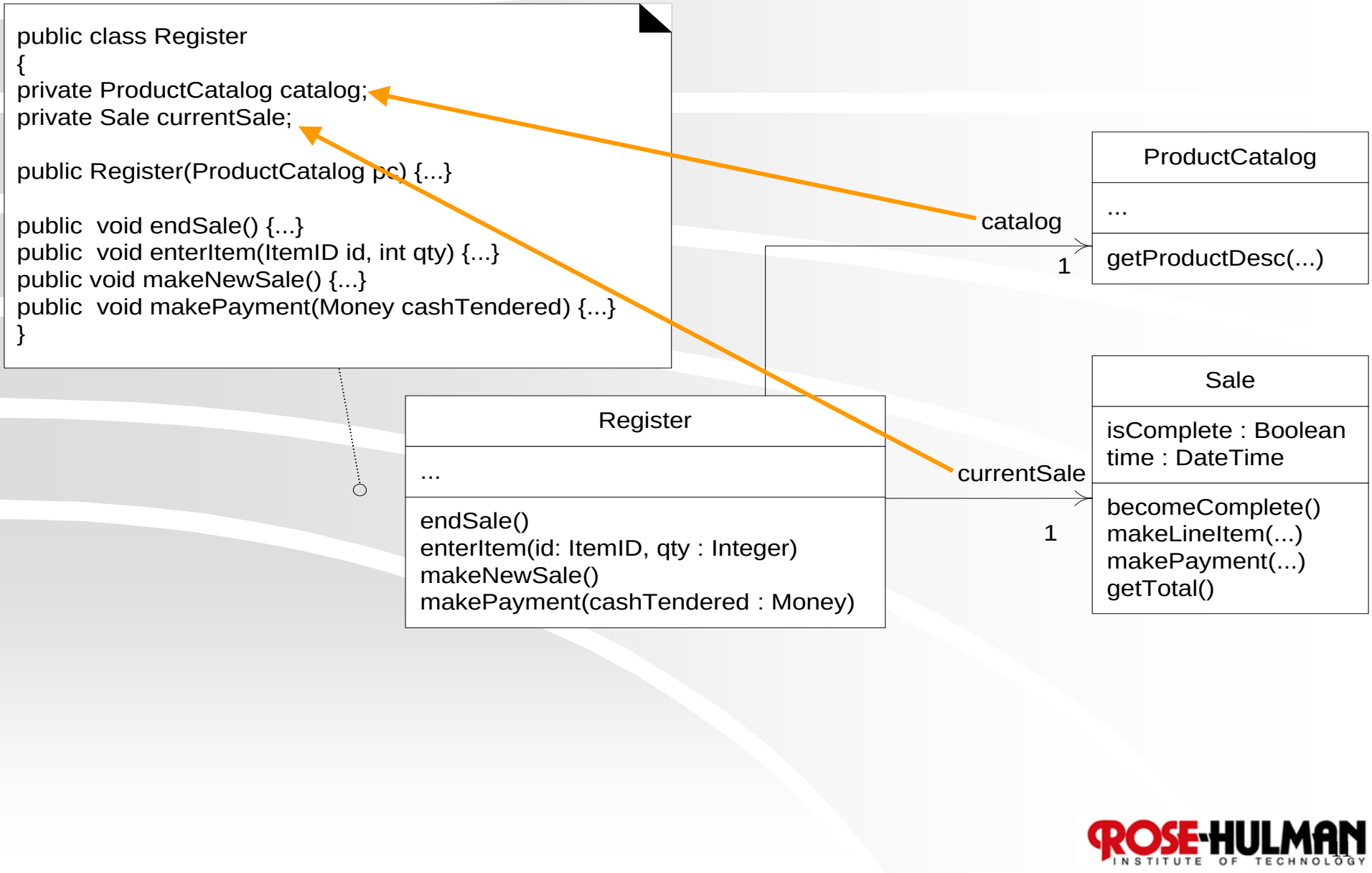
Depending on the system, many of these steps might just be sketches!

Moving from Design to Code

- Design provides starting point for Coding
  - DCDs contain class or interface names, superclasses, method signatures, and simple attributes
- Two primary tasks
  1. Define classes & interfaces
  2. Define methods
- Elaborate from associations to add reference attributes

Example: Defining Register Class

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

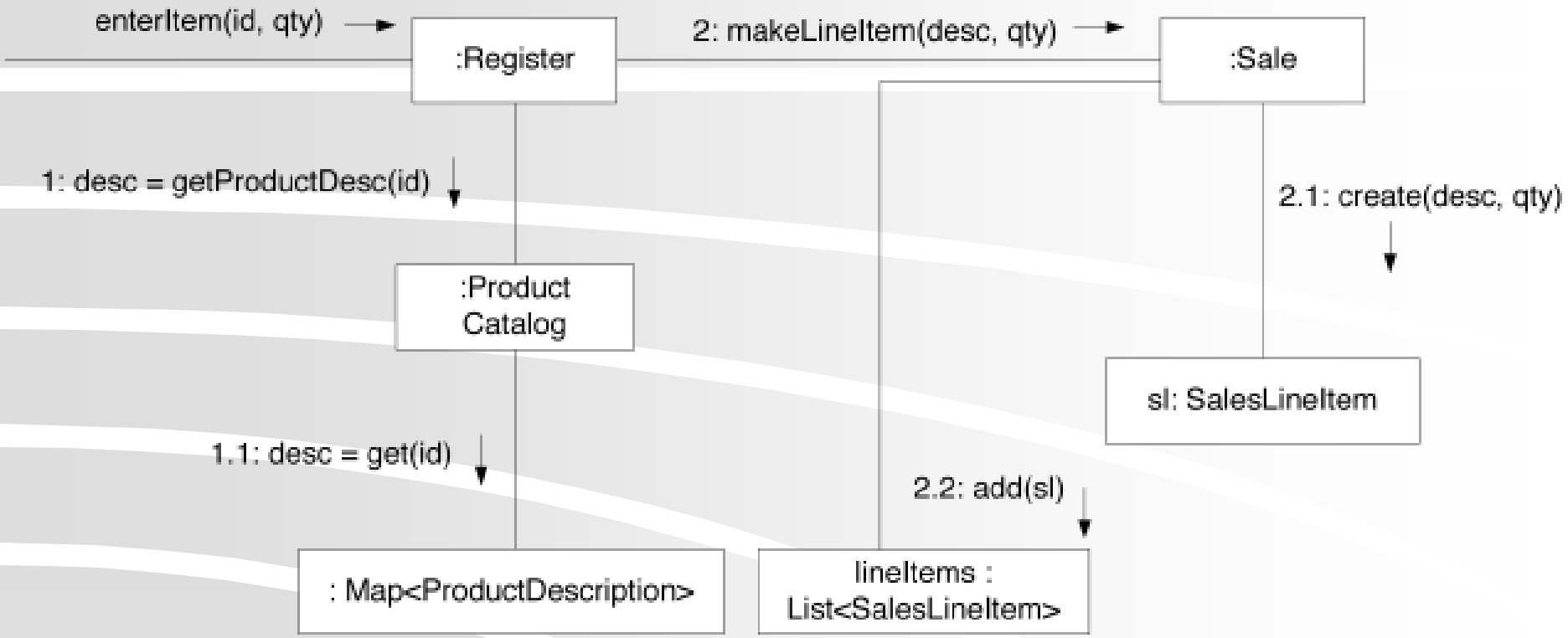


Create Class Definitions from DCDs

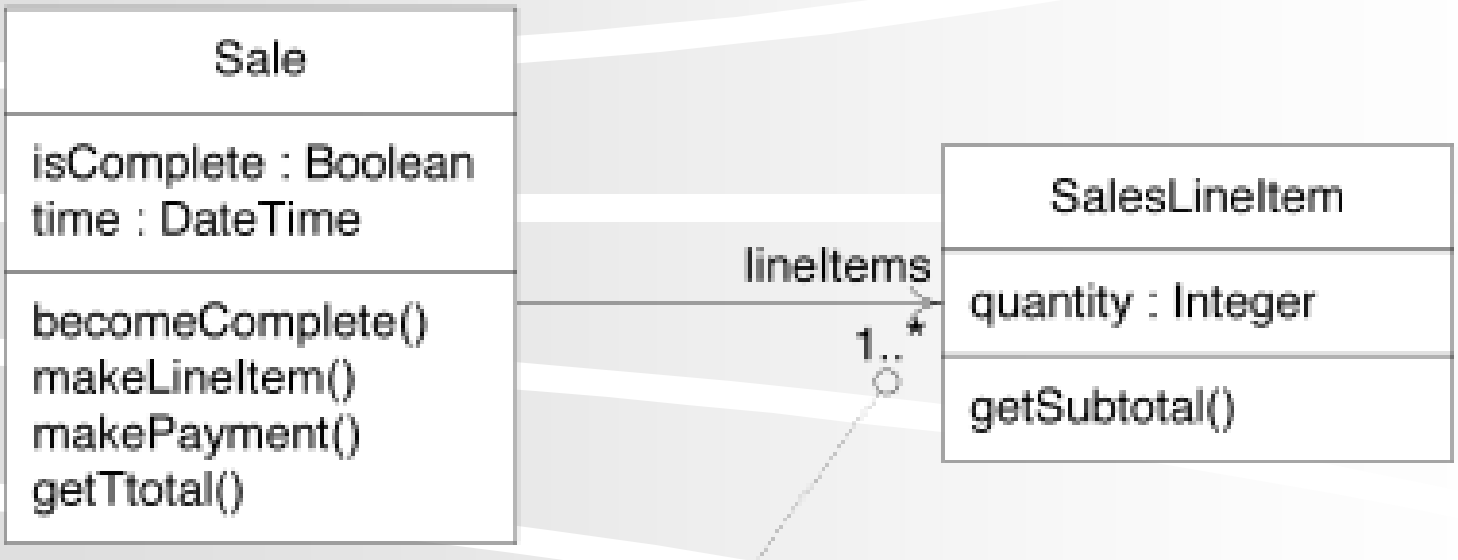
```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

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

public Money getSubtotal() { ... }

}
```

Don't write the code on your diagram. Write it in your IDE.

Create Methods from Interaction Diagrams



Collections

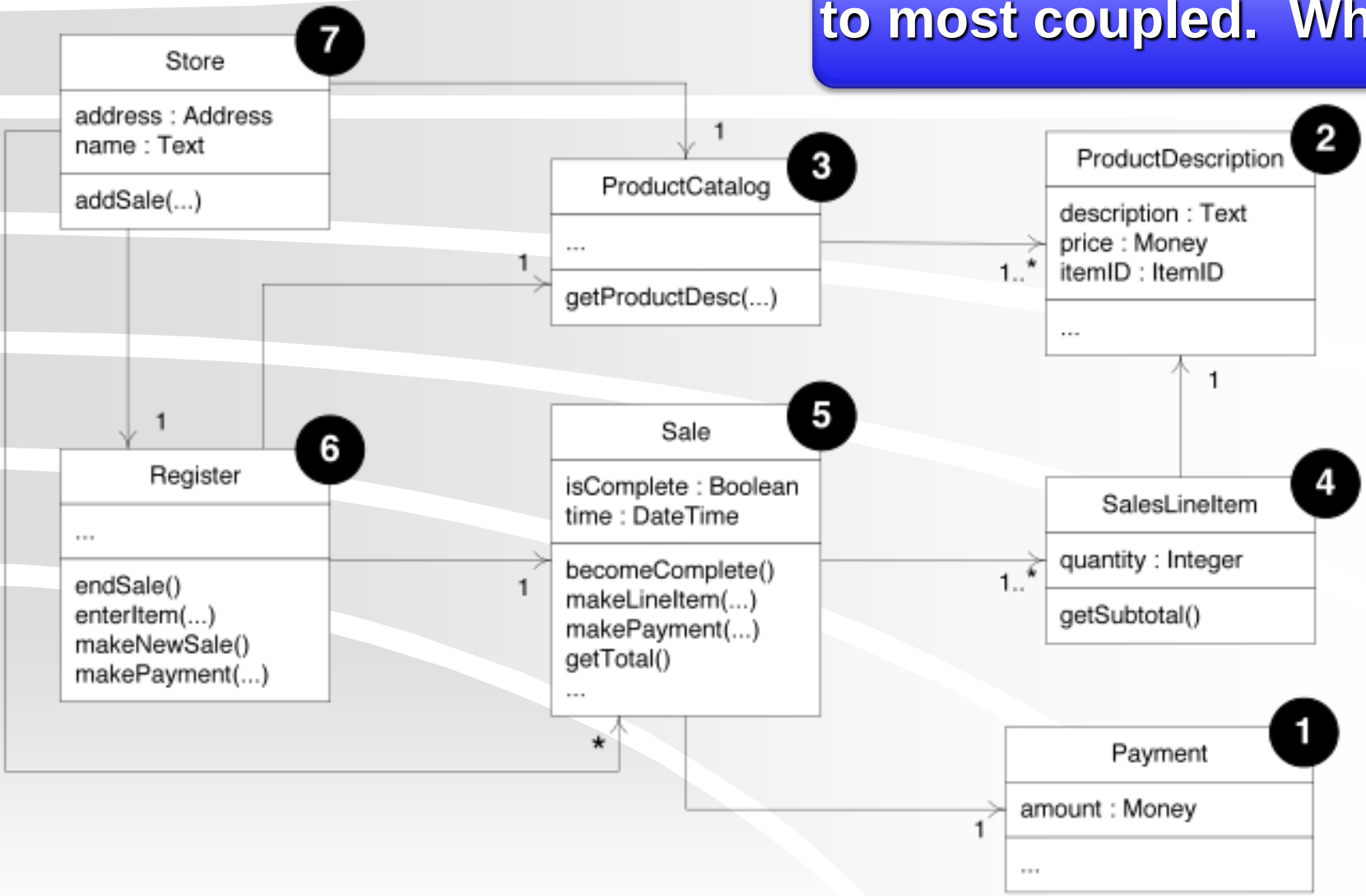



```
public class Sale {
        …
  private List<SalesLineItem> lineItems = new ArrayList<SalesLineItem>();
        …
}
```

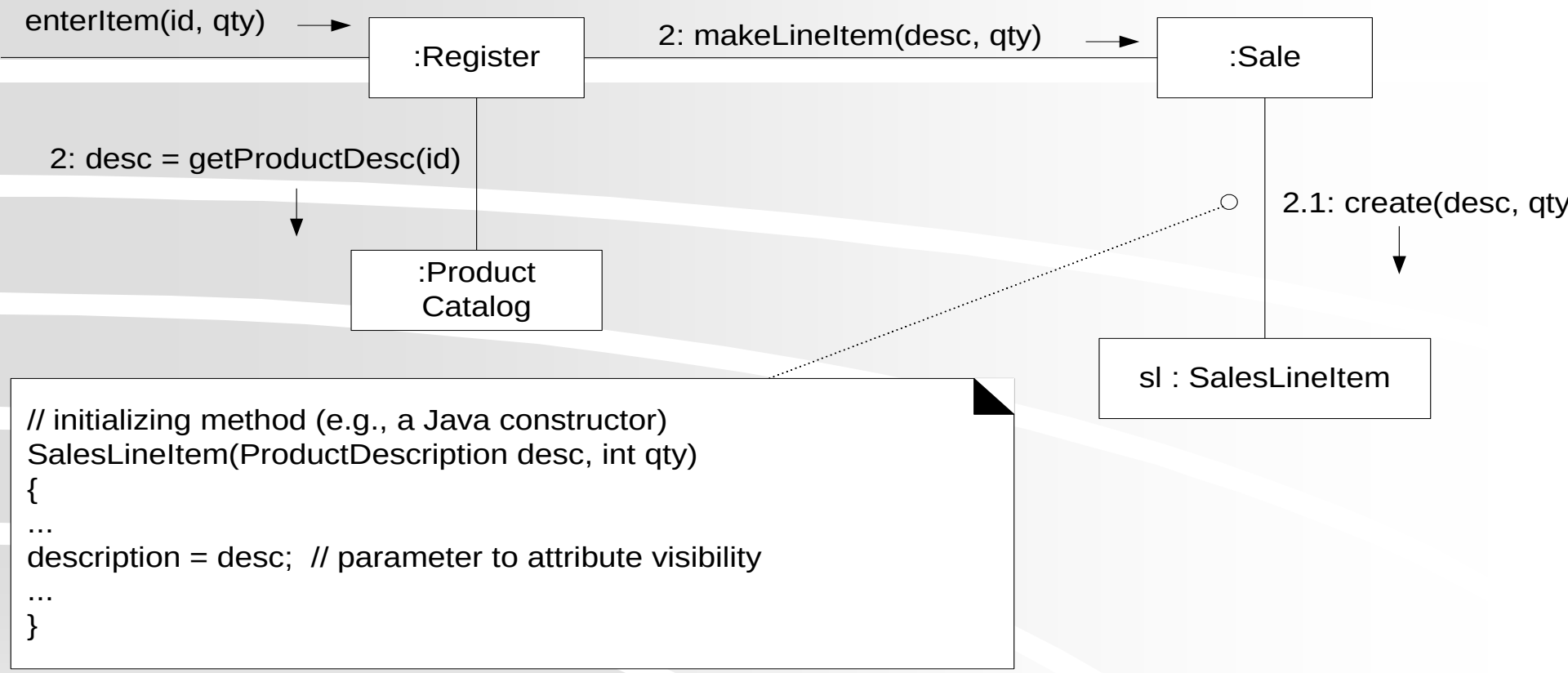
Guideline: If an object implements an interface, use the interface type for the variable.

What Order?

Typically, least coupled to most coupled. Why?



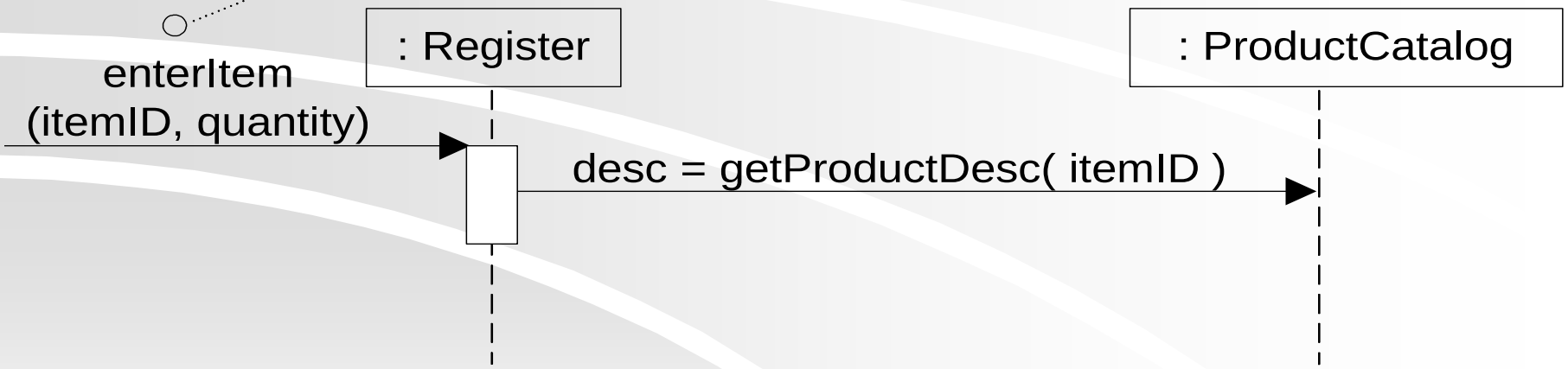
Parameter Visibility Converted to Attribute Visibility



SalesLineItem has association with ProductDescription

Local Visibility: Register Assigns Method Return to A Local Variable

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



Register has local visibility to ProductDescription

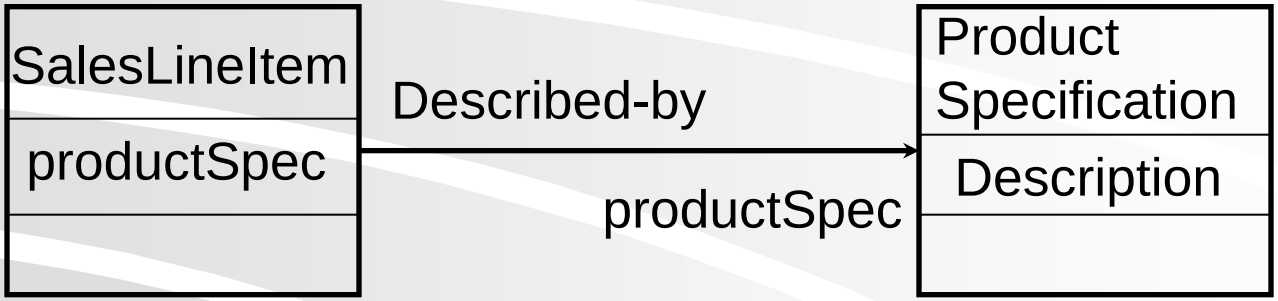
Mapping from Class Diagram to Code

Constructor usually not Shown on diagram

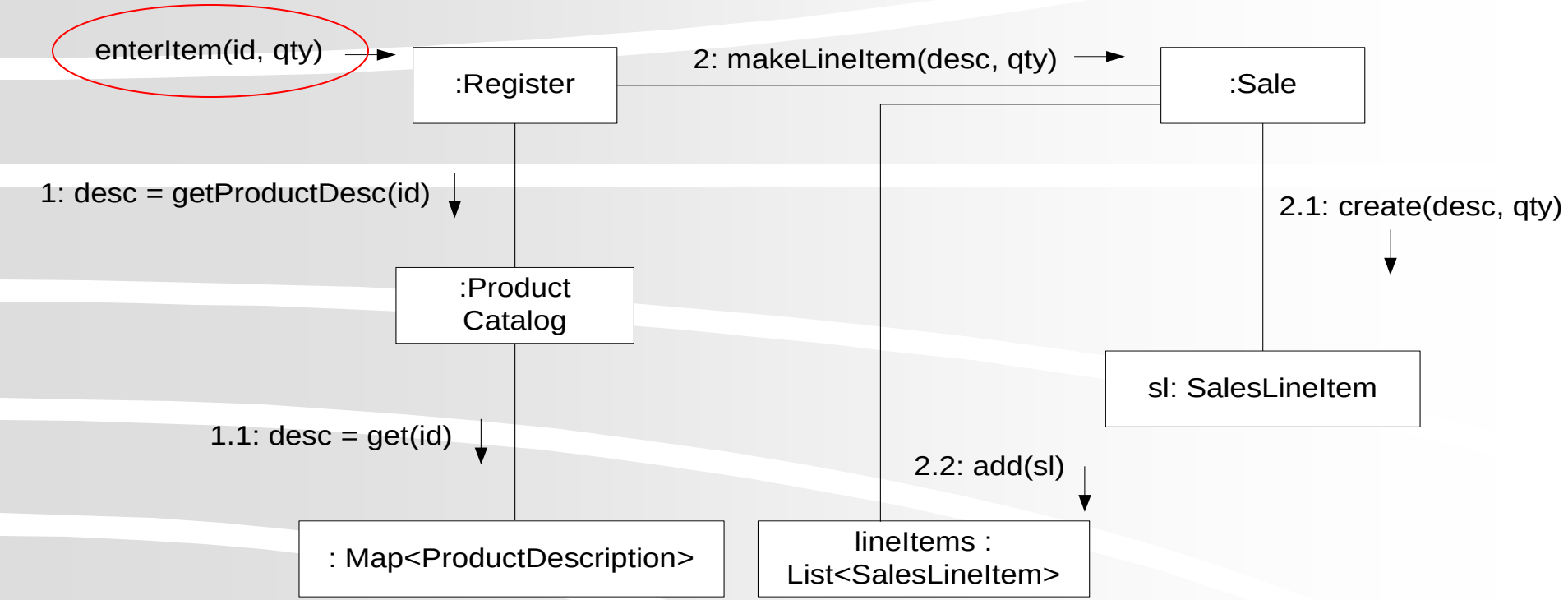
```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

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

public Money getSubtotal() { ... }

}
```

Reference Attributes and Role Names

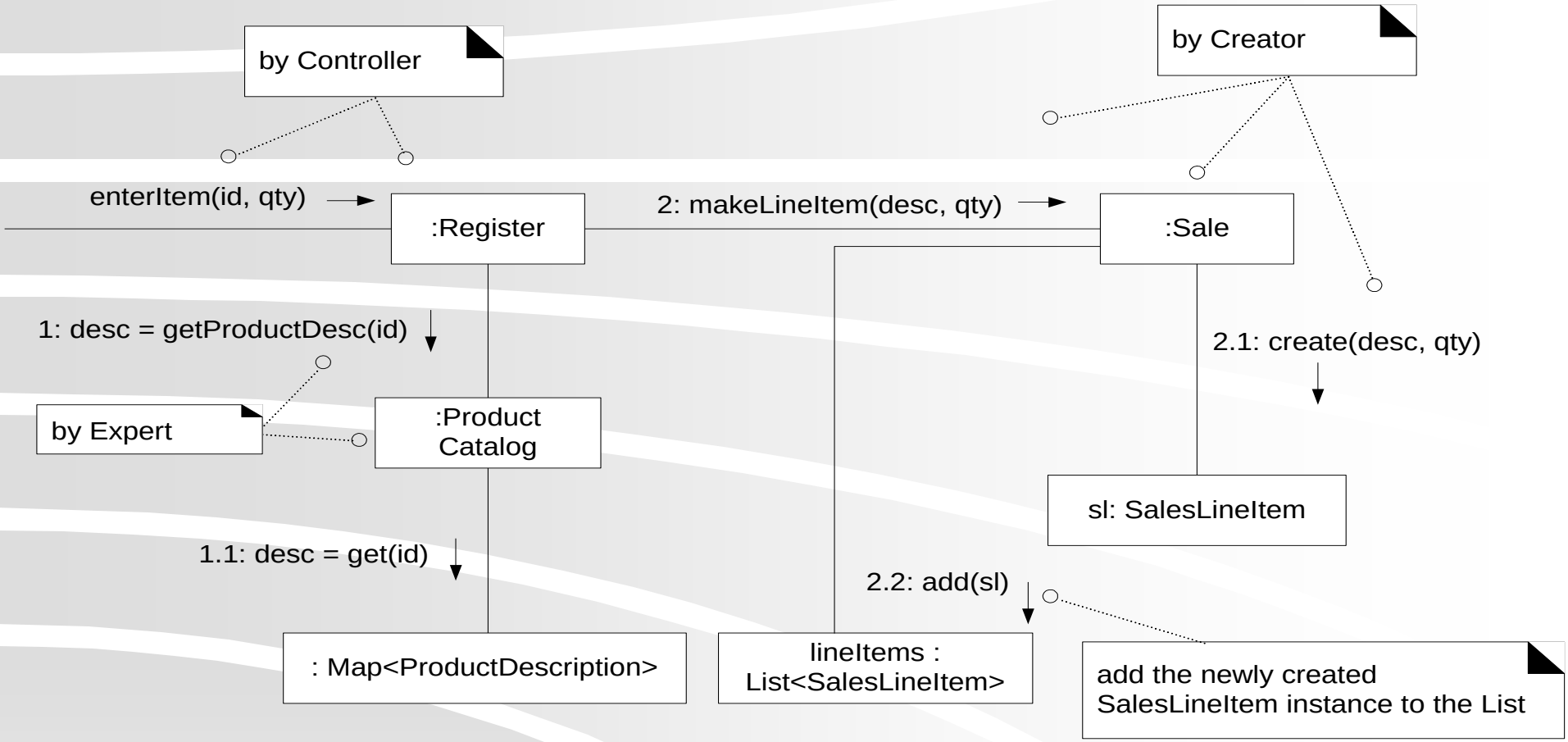
- **Reference Attributes are indicated by associations and navigability in a class diagram**
- **Example: A product specification reference on a Sales Line Item**



Consider enterItem() System Operation

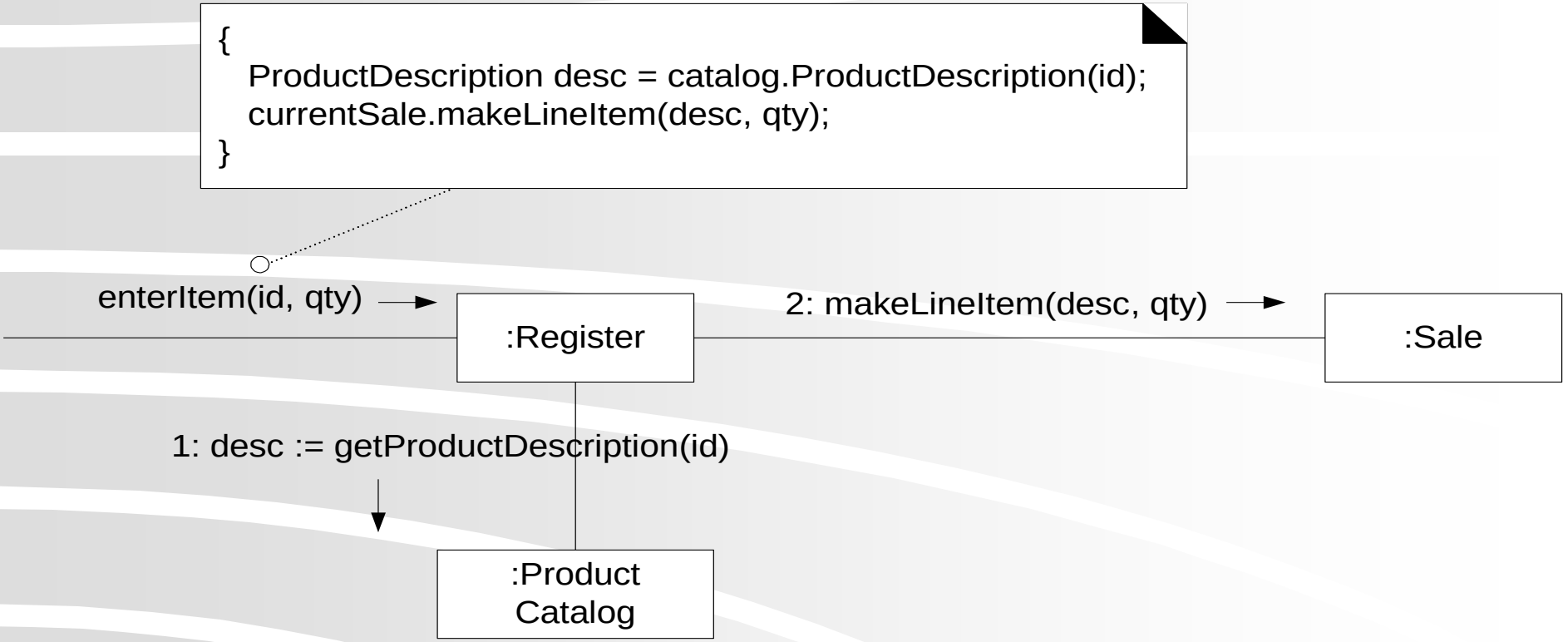


Dynamic View: enterItem() Communication Diagram



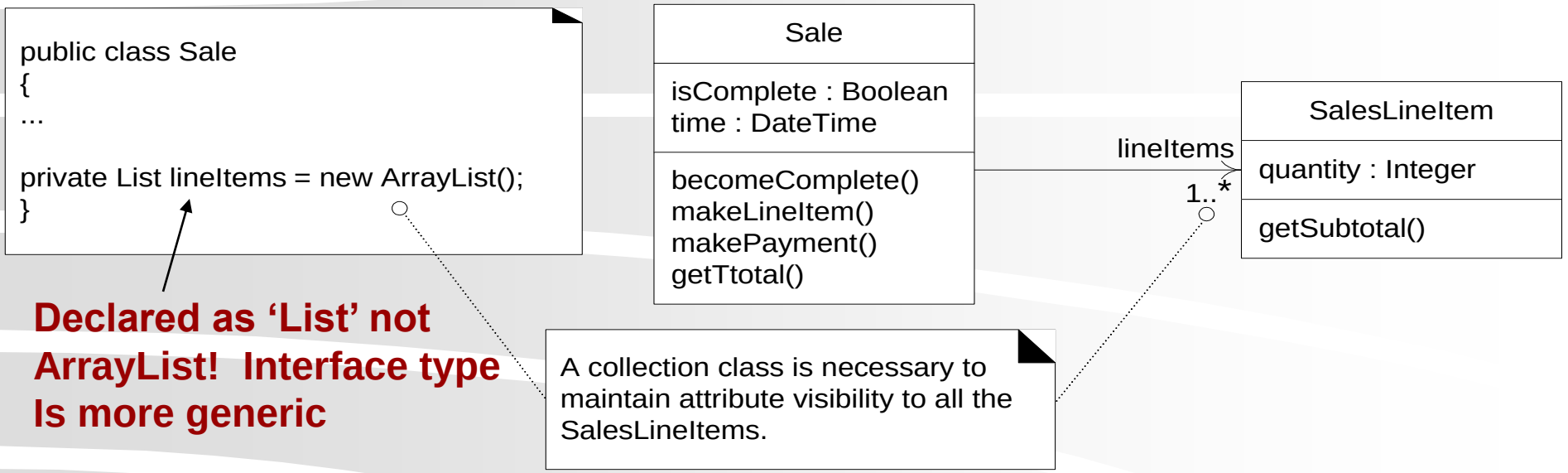
Why choice of Map for ProdDesc and List for SaleLineItems?

Mapping Messages to Methods



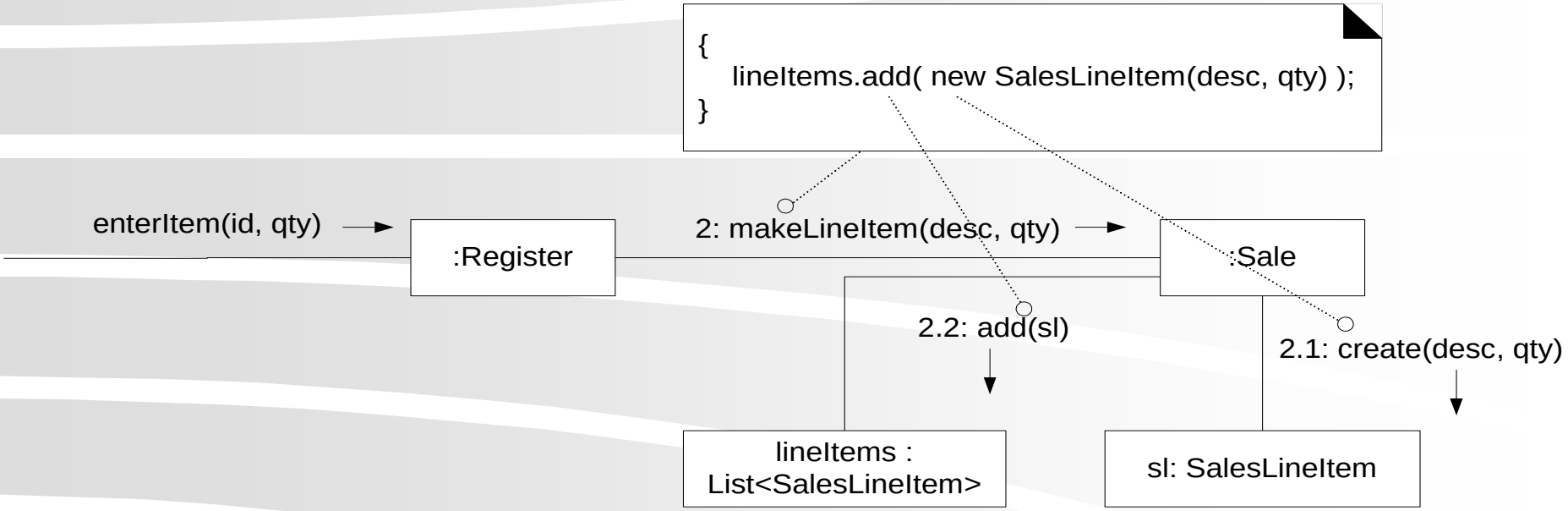
Each message maps to a method call within enterItem()

Adding Collection from 1 to Many Association



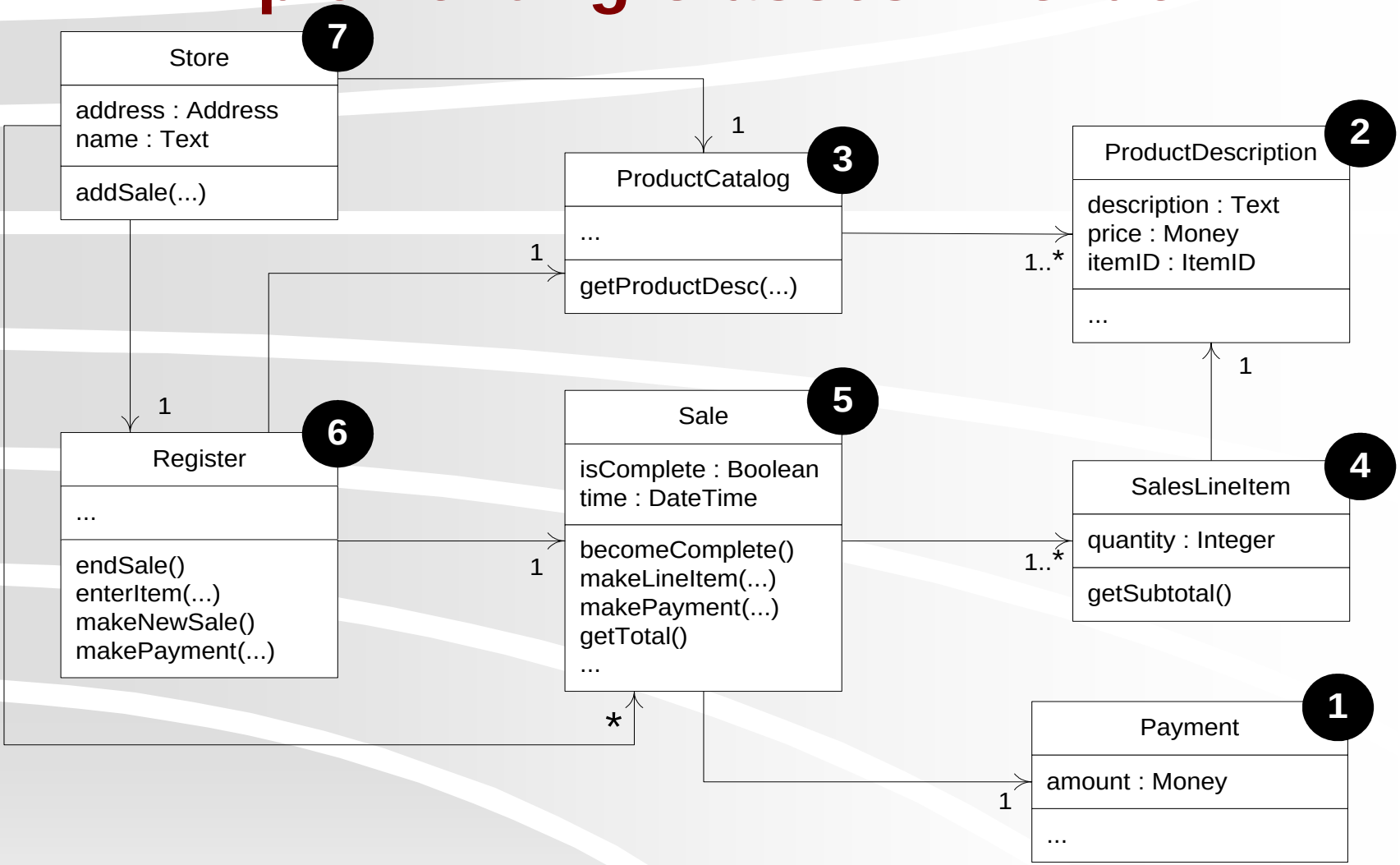
Declared as 'List' not ArrayList! Interface type Is more generic

Implementing Methods from IDs

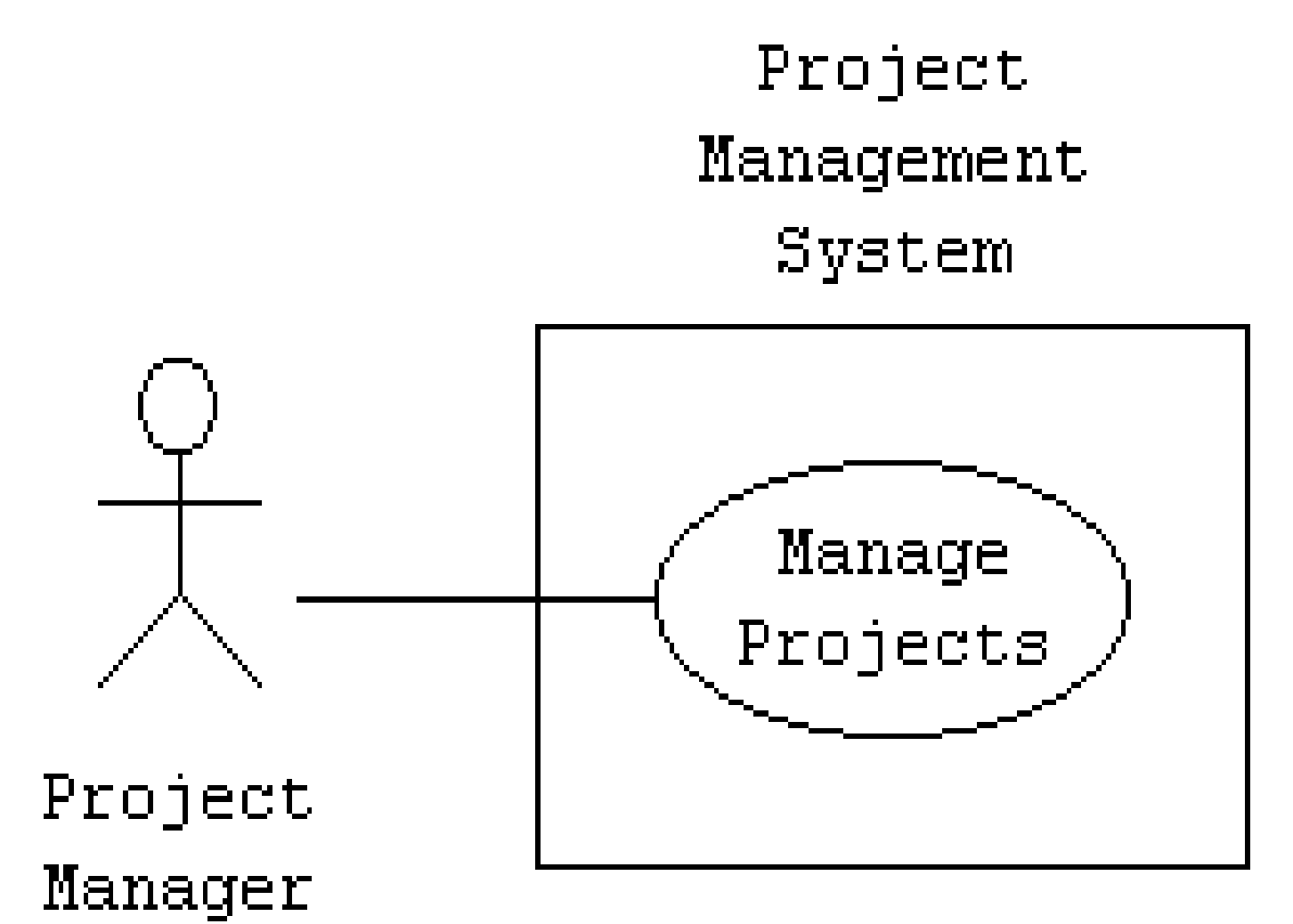


- **Sale.makeLineItem(), part of the enterItem() interaction diagram**
- **Notice how simply the method can be implemented!**

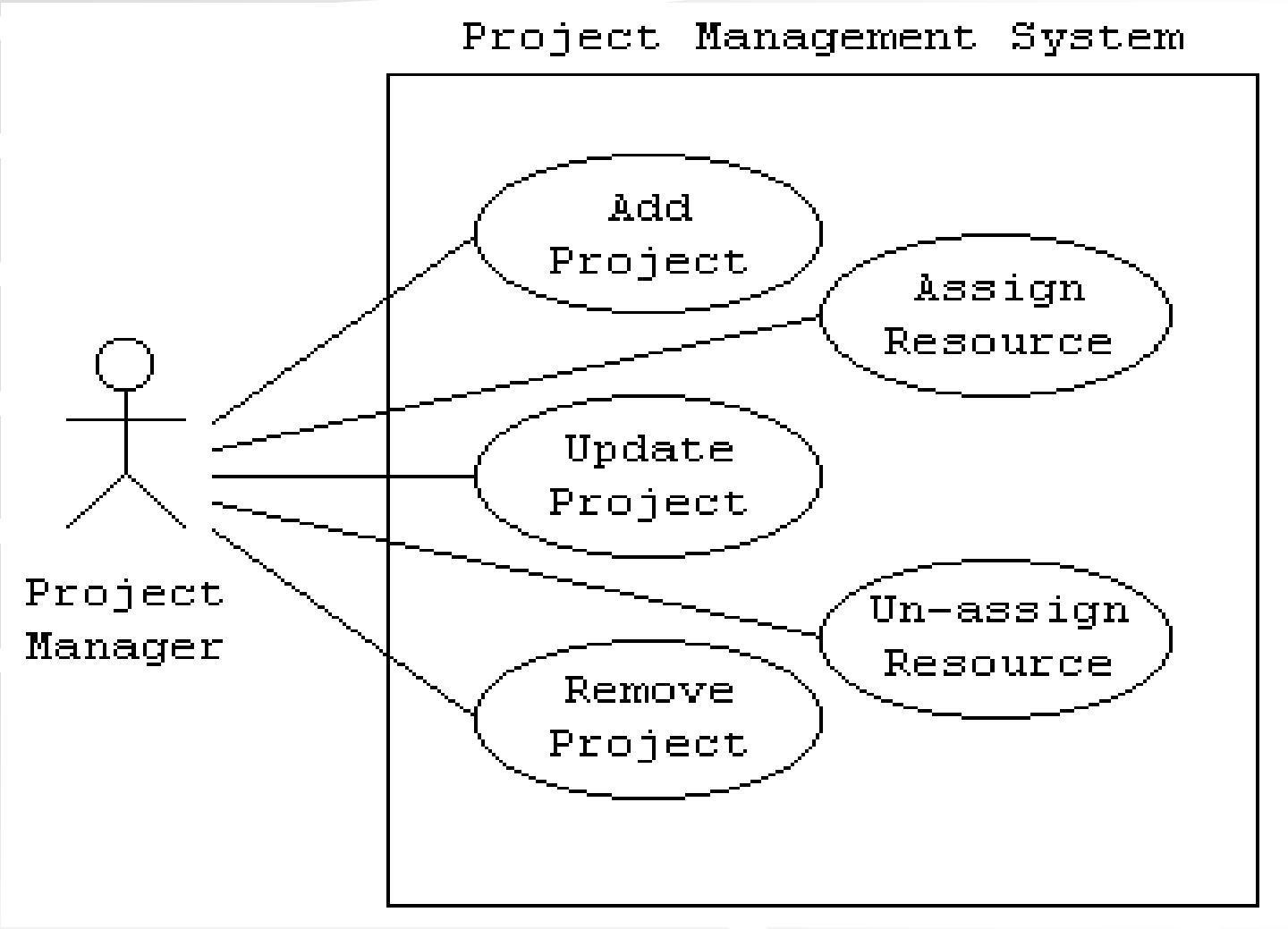
Implementing Classes in Order



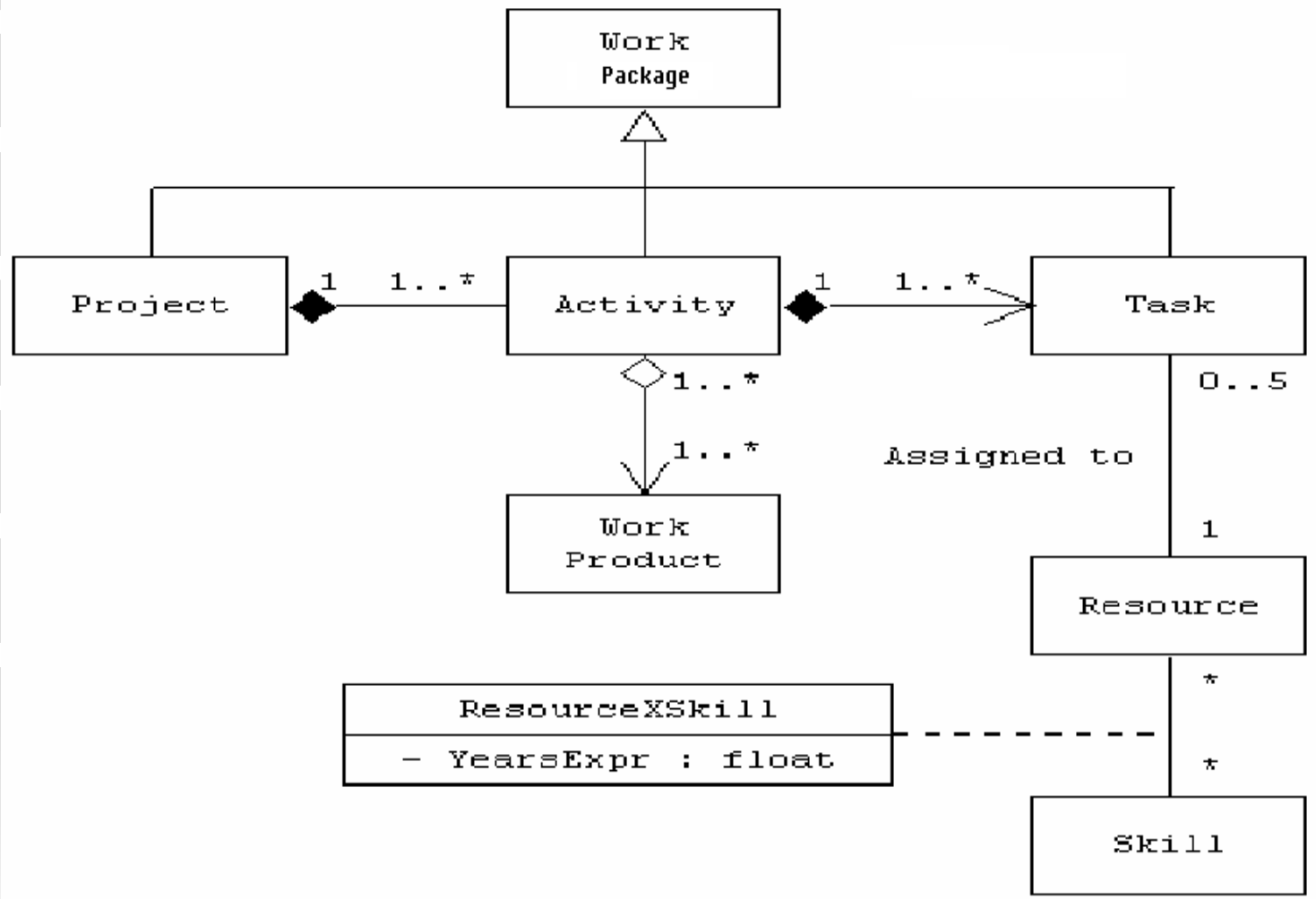
Working Example: PM



PM: Use Case Diagram



PM: Class Diagram



PM: Class to Code

- class WorkPackage;
- class Project;
- class Activity;
- class Task;
- class WorkProduct;
- class Resource;
- class Skill;
- class ResourceXSkill;

PM: Class to Code

```
class WorkPackage
  { // Details omitted };

 class Project : public WorkPackage
 { private: CollectionByVal<Activity> theActivity; };

class Activity : public WorkPackage
 { private: Project *theProject;
              CollectionByVal<Task> theTask;
   CollectionByRef<WorkProduct>
                                   theWorkProduct; };
```

PM: DCD Mapping

```
                    Project
-------------------------------------------------------
Name : char * {private}
- Descr : char *
- StartDate : Date
- NumberOfProjects : int = 0
-------------------------------------------------------
+ <<constructor>> Project (Name : char *) : Project
+ <<constructor>> Project (void) : Project
+ <<destrcutor>> ~Project (void)
+ getName (void) : char *
+ setName (theName : char *) : void
setDescr (Descr : char *) : void {public}
getDescr (void): char * {public}
+ setStartDate (theStartDate : Date) : void
getStartDate (void) : Date {public}
# hasActivites (void) : bool
+ addActivity (theActivity : const Activity &) : void
+ getAllAcitivities (void) : CollectionByRef<Activity>
+ getNumberOfProjects (void) : int
+ save (void) : void
+ load (Name : char *) : void
```

PM: DCD Code

```
class Project
 { private:
     char *Name;
     char *Descr;
     Date StartDate;
     static int NumberOfProjects;
  public:
    Project (char *Name);
    Project (void); ~Project (void);
    char *getName (void);
    void setName (char
      *theName);
    void setDescr (char *Descr);
    char *getDescr (void);
    void setStartDate (Date
                       theStartDate);
```

```
Date getStartDate (void);
   void addActivity (const Activity
     &theActivity);
  CollectionByRef<Activity>
     getAllAcitivities (void);
   static int getNumberOfProjects (void);
   void save (void);
   void load (char *Name);
protected:
   bool hasActivities (void); };

 int Project::NumberOfProjects = 0;
```

PM: Sequence Diagram