GRASP OO design principles

General Responsibility Assignment Software Patterns

- Describe fundamental principles of object design

Responsibility for

- **Doing something**
- **Knowing things**

Object-oriented design

”Deciding what *methods belong where* and how *objects should interact* carries consequences and should be taken seriously.”

[Larman] p. 271

Which consequences?

How to decide?

GRASP – patterns considered

- **Creator**
- **Information Expert**
- **Low coupling**
- **Controller**
- **High cohesion**

Creator

Which object must have the responsibility to create new instances of a class **A**?

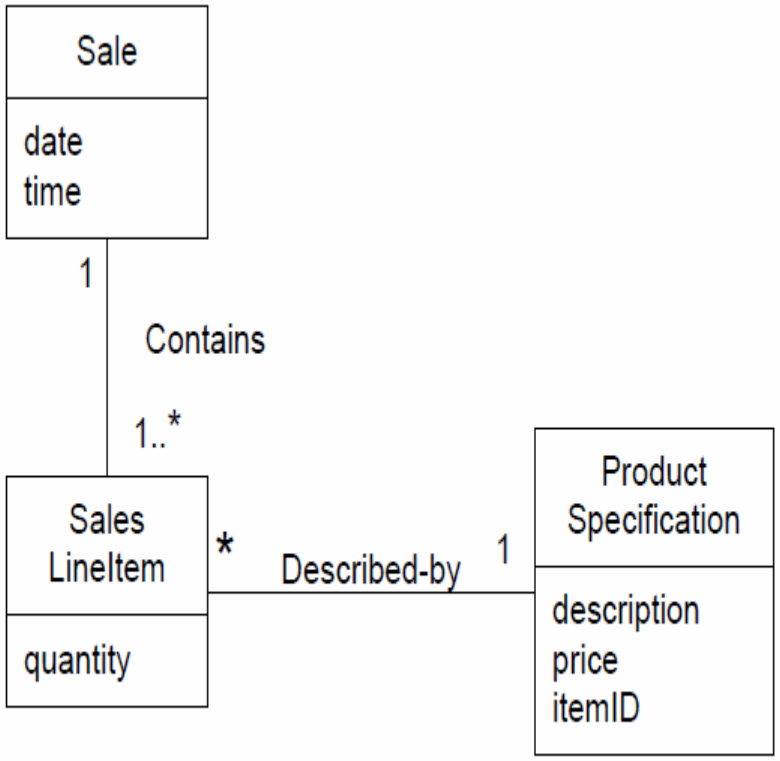


Figure 16.7 Partial domain model.

Who should be responsible for creating a *SalesLineItem* instance?

By Creator, we should look for a class that aggregates or contains *SalesLineItem* instances

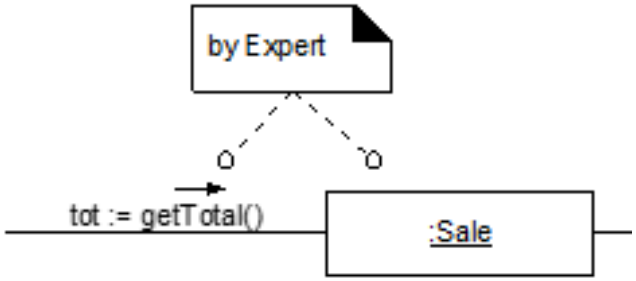
Information Expert

- **Problem**:
  What is a basic principle by which to assign responsibilities to objects?

- **Solution: Assign responsibility to an object having the information (the knowing) needed to fulfill the job**
  - E.g. which object should be assigned responsibility for calculating a total?
    1. Which information is necessary (algorithm)?
    2. Which object has most of the information?

Information Expert calculation *total*

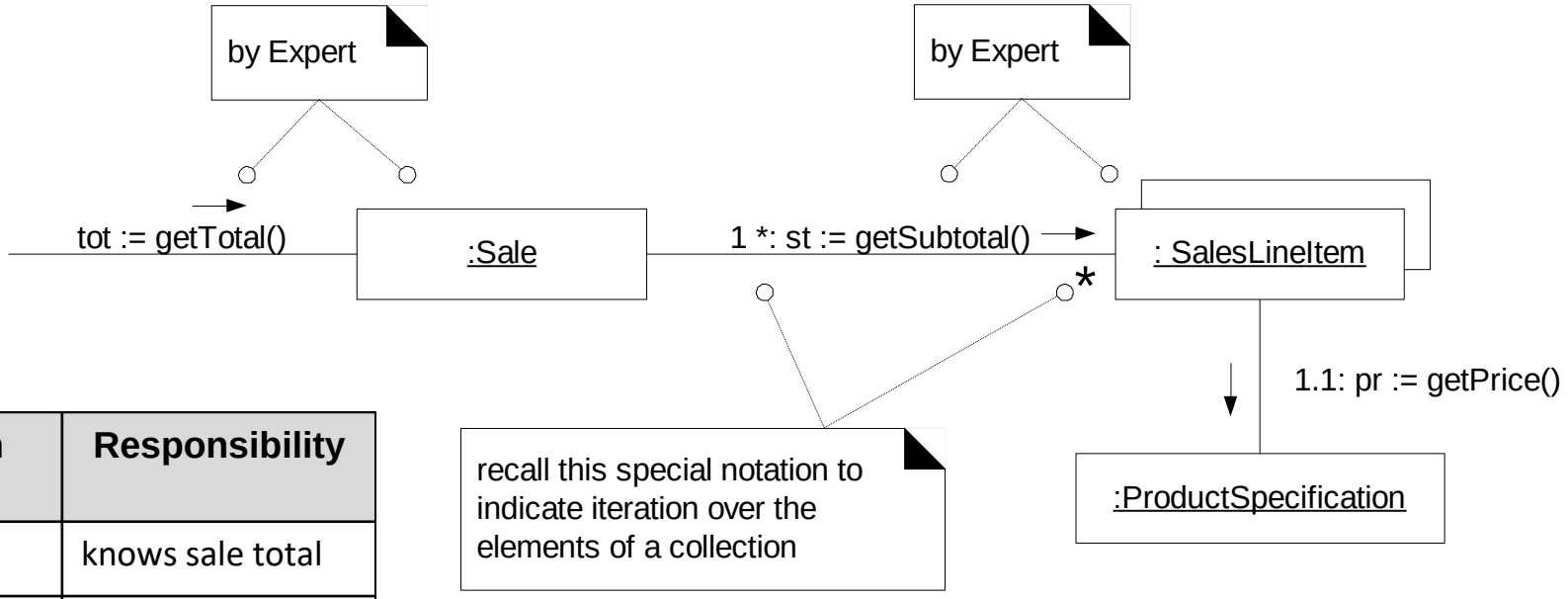
Assign responsibility to an object having the information (the knowing) needed to fulfill the job



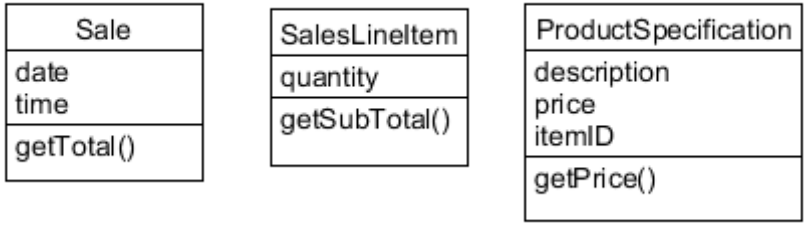
Sale has most of the information

.. but not all

Information Expert calculation *total*



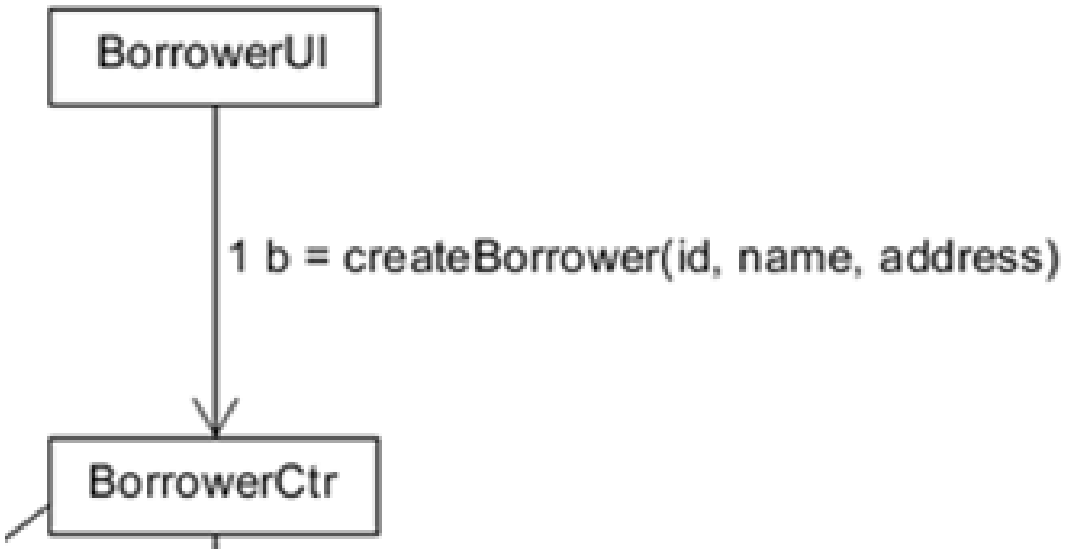
Design class	Responsibility
Sale	knows sale total
SalesLineItems	knows line item subtotal
ProductSpecification	knows product price



Controller

Who should be responsible for handling UI events?

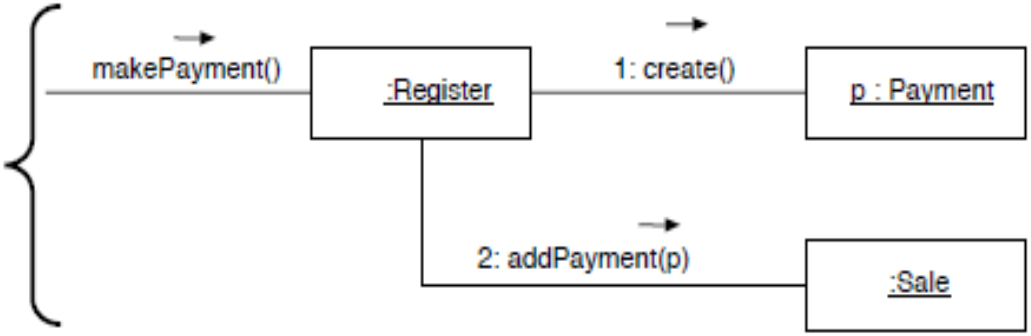
- **Represents a receiver or handler of all system events of a *use case* scenario**

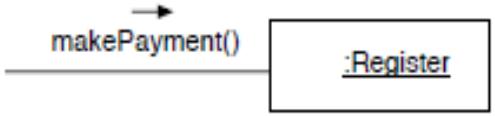


Low Coupling
assigning responsibility for makePayment

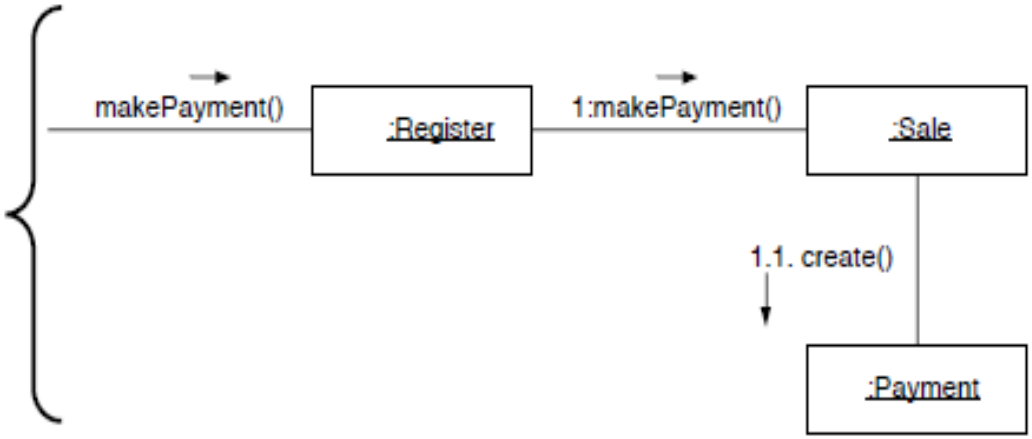
–How can you support low dependency between the system components/classes
–How can you promote reuse?

Expert pattern:
Assign responibility to the object having the necessary information





Low coupling pattern:
Assign responsibility so that coupling remains low



Low Coupling - reuse of controllers

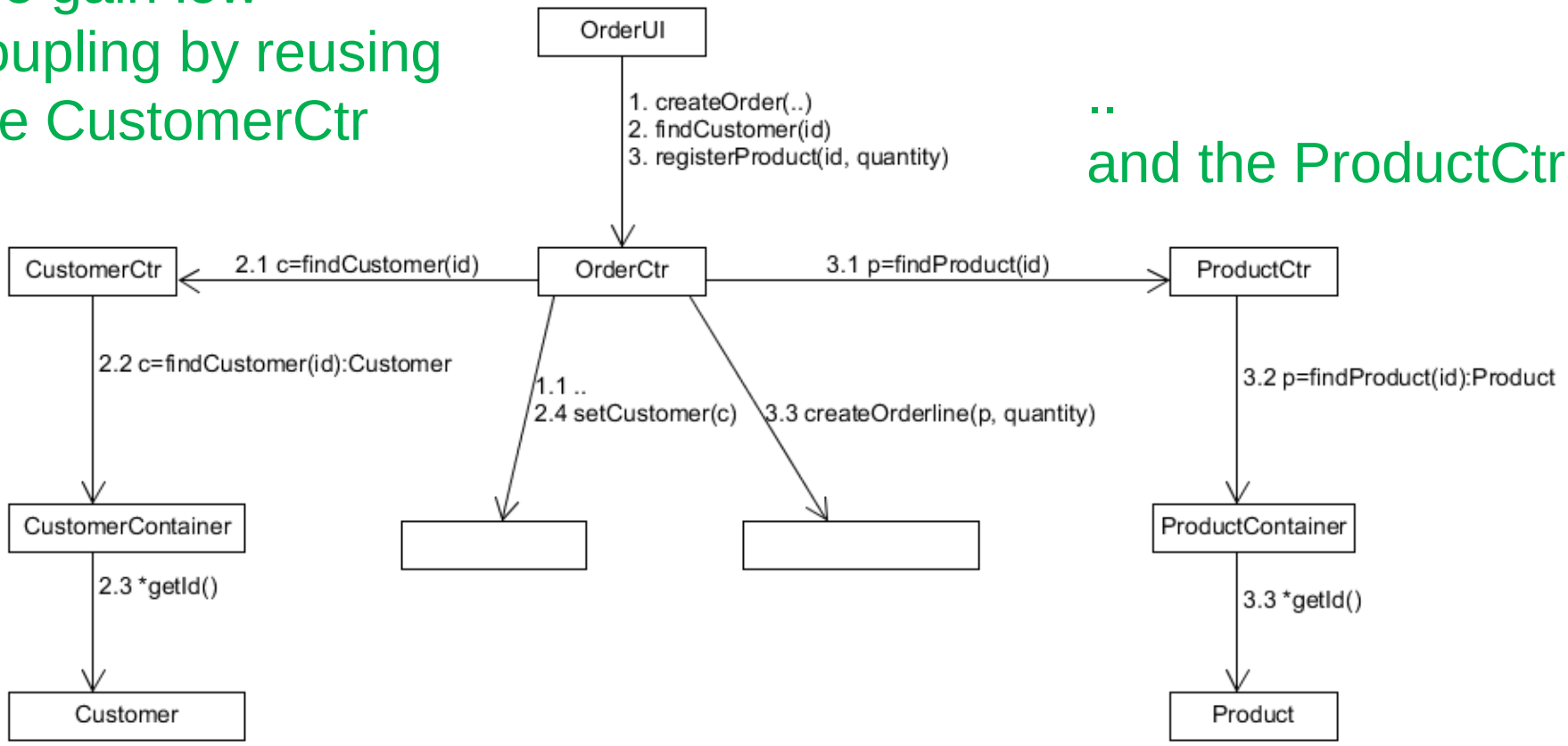
– How can you support low dependency between the system components/classes
– How can you promote reuse?

We gain low coupling by reusing the CustomerCtr

..

..

and the ProductCtr



Cohesion

- Cohesion measures how strongly related and focused the responsibilities of an element are

Why do we want classes to have high cohesion?

High Cohesion?

If we consider Creator Pattern, since Register records a Payment in the real-world domain, then Register should create a Payment instance… → low cohesion

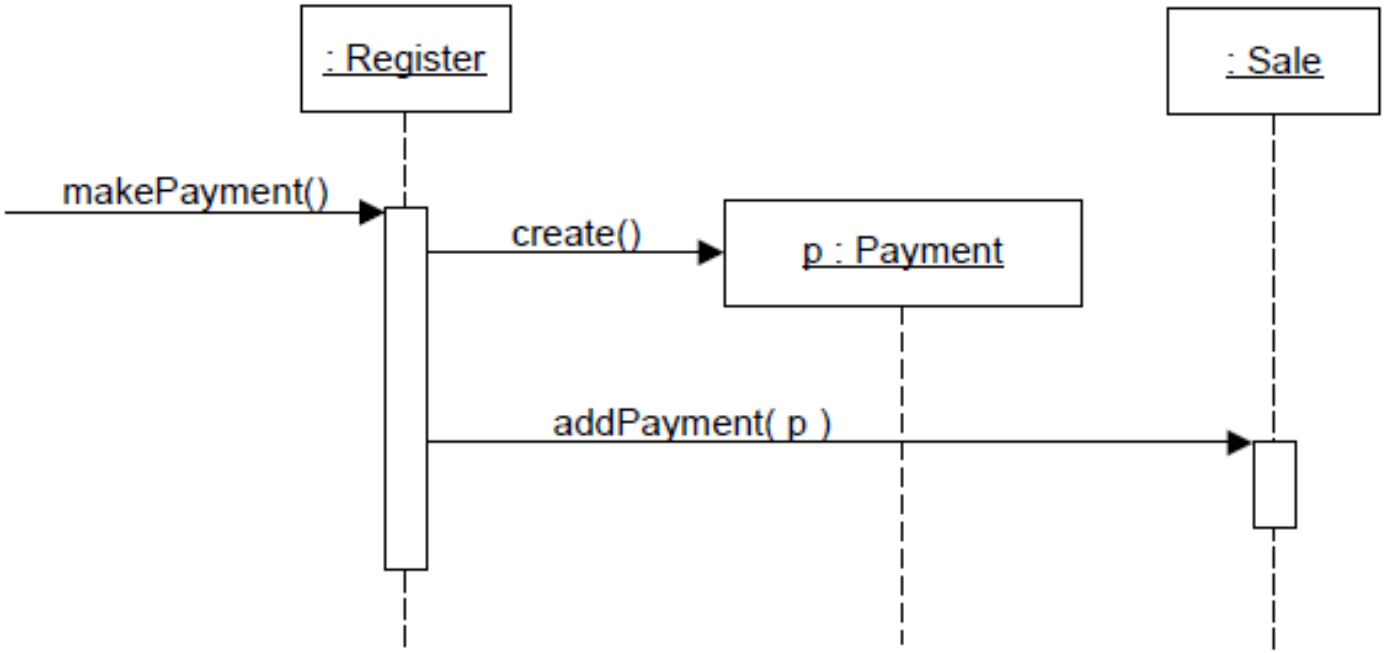


Figure 16.11 Register creates Payment.

High Cohesion

How to keep classes focused and manageable?

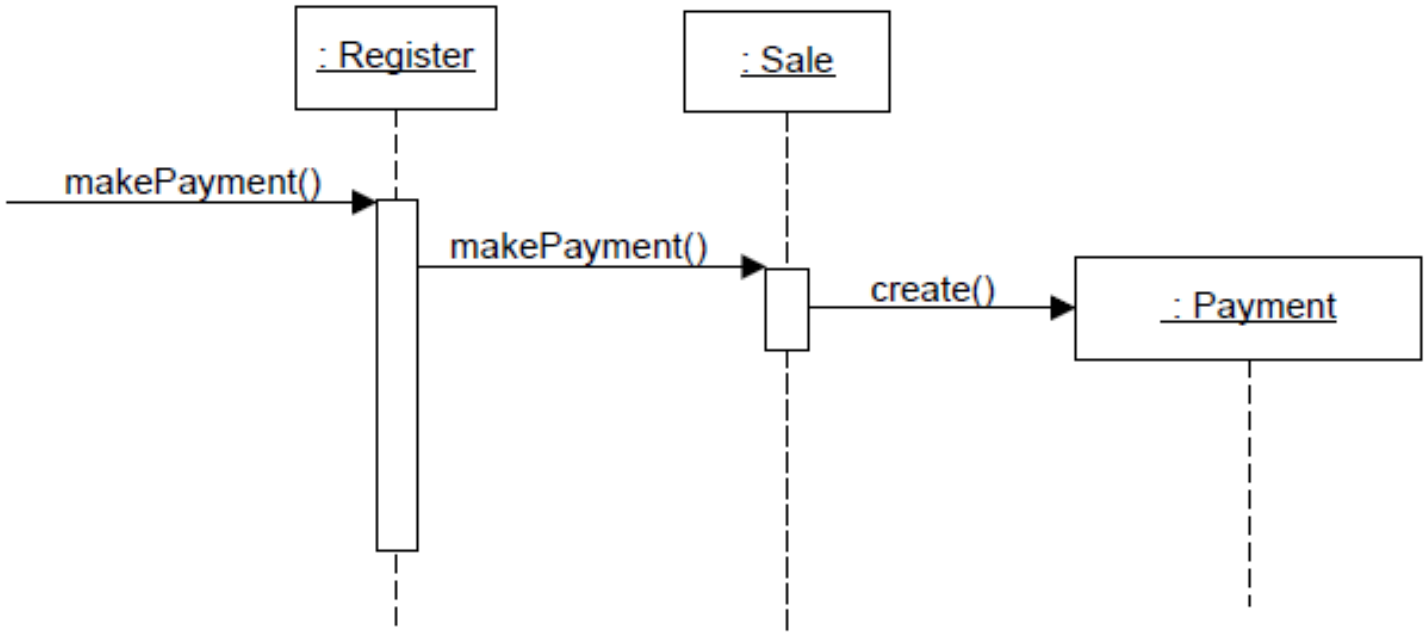


Figure 16.12 Sale creates Payment