Object-Oriented Design
- Design Patterns: GRASPs

Zhiming Liu
zhimingliu88@swu.edu.cn
Centre for Research and Innovation in Software Engineering
School of Computer and Information Science
Southwest University, Chongqing, China

Patterns

- Patterns: general principles and idiomatic solutions that guide the creation of software
- Each of these principles or solutions describes a problem(s) to be solved and a solution(s) to the problem(s)

Patterns

- Patterns: general principles and idiomatic solutions that guide the creation of software
- Each of these principles or solutions describes a problem(s) to be solved and a solution(s) to the problem(s)
- A pattern is a problem/solution pair that can be applied to new context, with advice on how to apply it in novel situations:

Patterns

- Patterns: general principles and idiomatic solutions that guide the creation of software
- Each of these principles or solutions describes a problem(s) to be solved and a solution(s) to the problem(s)
- A pattern is a problem/solution pair that can be applied to new context, with advice on how to apply it in novel situations:

  Pattern Name: name given to the pattern
  Solution: description of the solution of the problem
  Problem: description of the problem that the pattern solves

Patterns

- Patterns: general principles and idiomatic solutions that guide the creation of software
- Each of these principles or solutions describes a problem(s) to be solved and a solution(s) to the problem(s)
- A pattern is a problem/solution pair that can be applied to new context, with advice on how to apply it in novel situations:

Pattern Name:	name given to the pattern
Solution:	description of the solution of the problem
Problem:	description of the problem that the pattern solves

We study Expert, Creator, Low Coupling, High Cohesion, and Controller

Expert

Pattern Name:	Expert
Solution:	**Assign a responsibility to the information expert – the class that has *information* necessary to fulfill the responsibility**.
Problem:	What is the most basic principle by which responsibilities are assigned in OOD?

Example in POST

- Some class needs to know the grand total of a sale
- To assign responsibilities, we had better to state the responsibility clearly

Example in POST

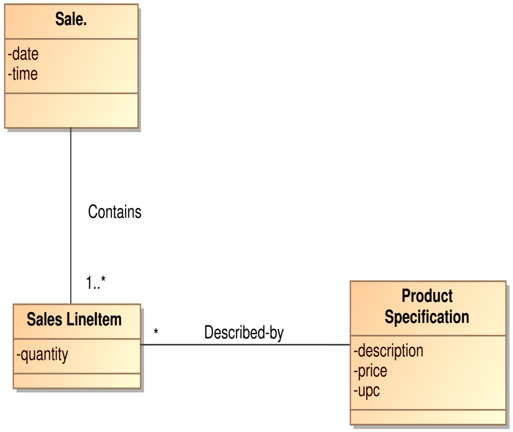
- Some class needs to know the grand total of a sale
- To assign responsibilities, we had better to state the responsibility clearly
  Who should be responsible for knowing the grand total of the sale?

Example in POST

- Some class needs to know the grand total of a sale
- To assign responsibilities, we had better to state the responsibility clearly
  Who should be responsible for knowing the grand total of the sale?
- Expert Pattern: we should look for the class which has the information needed to determine the total

Where to look for it?

Class Diagram



Decide the following

- What information is needed to determine the grand total?

Decide the following

- What information is needed to determine the grand total?
  - All the *SalsLineItem* instances of the *Sale* instance, and

Decide the following

- What information is needed to determine the grand total?
    - All the *SalsLineItem* instances of the *Sale* instance, and
    - The sum of their subtotal.

Decide the following

- What information is needed to determine the grand total?
  - All the *SalsLineItem* instances of the *Sale* instance, and
  - The sum of their subtotal.
- Who knows this information?

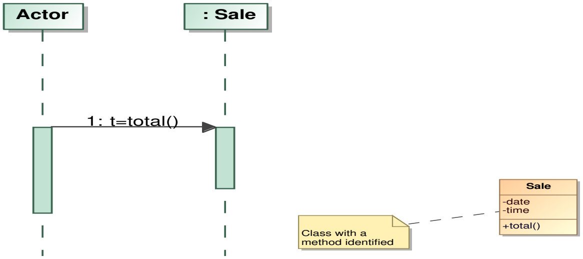
Decide the following

- What information is needed to determine the grand total?
  - All the *SalsLineItem* instances of the *Sale* instance, and
  - The sum of their subtotal.
- Who knows this information?

Knowledge and skills of algorithms design are needed, or a formulation of total, are needed

The Expert is *Sale*

- Assignment of responsibility is done in the context of the creation of collaboration diagrams.
- Start working on the OSD related to the assignment of responsibility for determining the grand total to *Sale*:



A design class diagram is also created

Continue with the Diagram

- Now, *how does the Sale carries out this responsibility*?

Continue with the Diagram

- Now, *how does the Sale carries out this responsibility*?
- it needs to get the subtotal of each line item
- *who should be responsible to return the subtotal of a line item*?

Continue with the Diagram

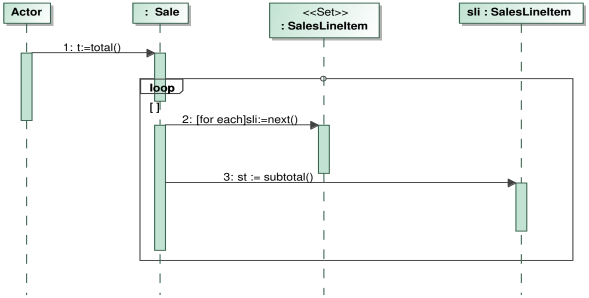
- Now, *how does the Sale carries out this responsibility*?
- it needs to get the subtotal of each line item
- *who should be responsible to return the subtotal of a line item*?
- *what is the information needed to determine the line item subtotal*?

Continue with the Diagram

- Now, *how does the Sale carries out this responsibility*?
- it needs to get the subtotal of each line item
- *who should be responsible to return the subtotal of a line item*?
- *what is the information needed to determine the line item subtotal*?
- **answer:** *SalesLineItem.quantity* and *ProductSpecification.price*.
- *who is the information expert for returning the subtotal of each line item?*

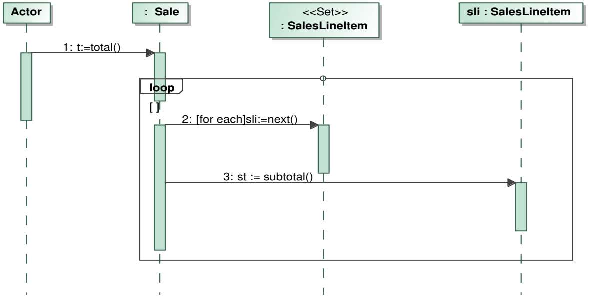
The Diagram

Answer: *SalesLineItem* is the expert for knowing and returning its subtotal, red but the sale may contain a number of *SalesLineItem* objects



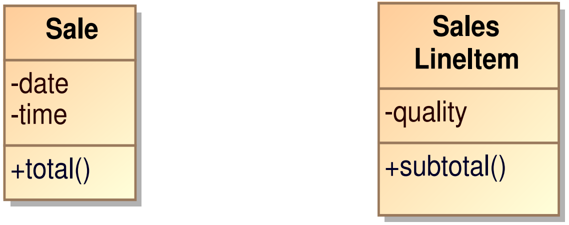
The Diagram

Answer: *SalesLineItem* is the expert for knowing and returning its subtotal, red but the sale may contain a number of *SalesLineItem* objects



Design class diagram?

Design Class Diagram



Continue with the Diagram: *subtotal*()

- to carry the responsibility of **returning** its subtotal, *SalesLineItem* needs to know **where** to get the price

Continue with the Diagram: *subtotal*()

- to carry the responsibility of **returning** its subtotal, *SalesLineItem* needs to know **where** to get the price
- **who is the information expert on answering the price**?

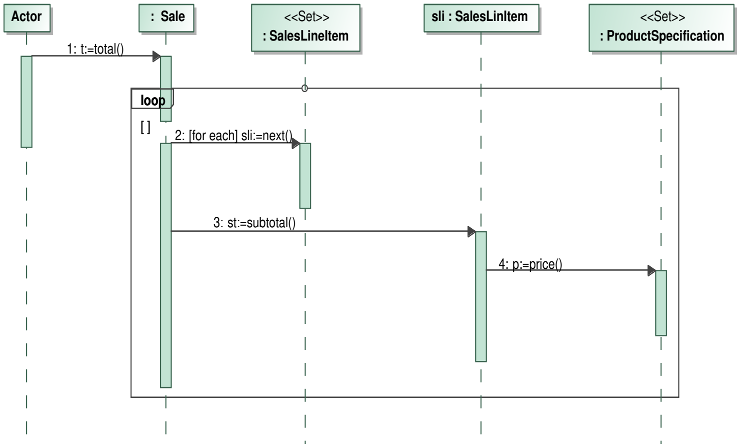
Continue with the Diagram: *subtotal*()

- to carry the responsibility of **returning** its subtotal, *SalesLineItem* needs to know **where** to get the price
- **who is the information expert on answering the price**?
- **answer:** *ProductSpecification*

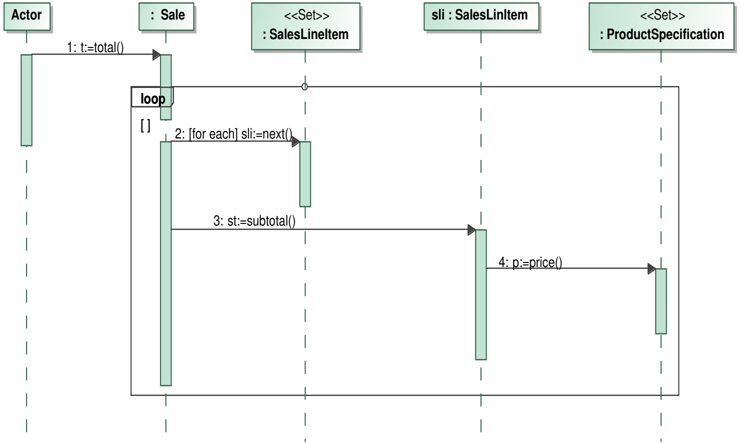
Continue with the Diagram: *subtotal*()

- to carry the responsibility of **returning** its subtotal, *SalesLineItem* needs to know **where** to get the price
- **who is the information expert on answering the price**?
- **answer:** *ProductSpecification*
- **a message must be sent to it asking for its price**

Complete the Design of *total*()

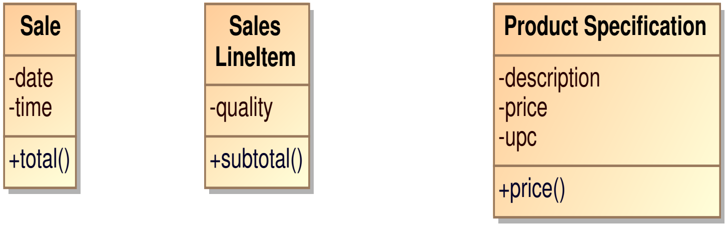


Complete the Design of *total*()



The design class diagram?

Design Class Diagram



associations are omitted!

Summary: Experts involved Total

Classes	**Responsibilities**
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

Summary: Experts involved Total

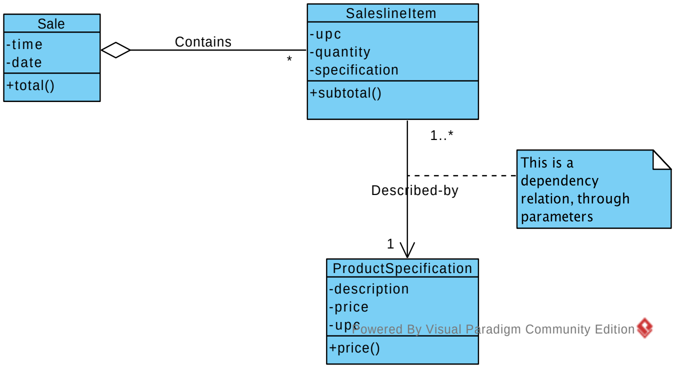
Classes	**Responsibilities**
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

These responsibilities identified and assigned while drawing the OSD

Design Class Diagram

Also create a **design class diagram** while creating a OSD, that refines the **conceptual class diagram** by

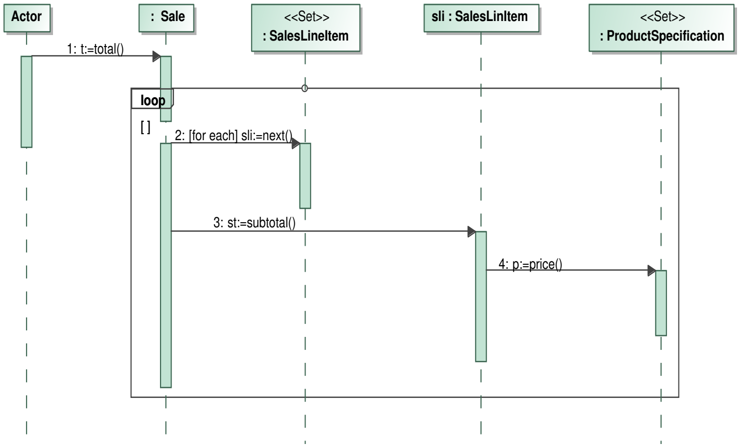
- adding methods to classes, and
- **navigation direction** of associations – **visibility** and **dependency**



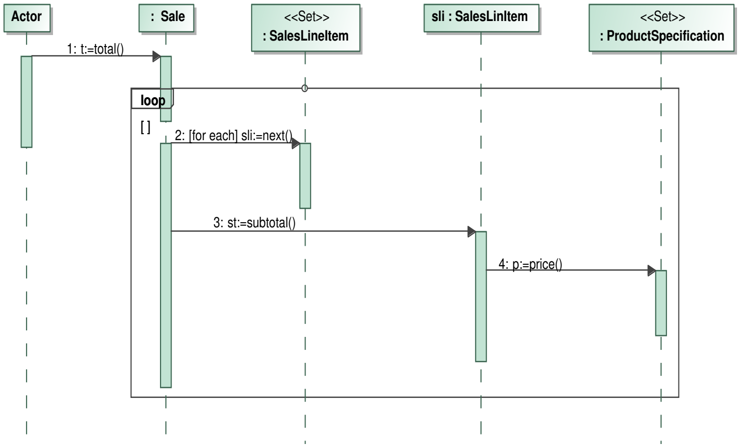
Remarks on Expert

- Expert is used more than any other pattern
- Express the intuition that objects do things related to the information they have
- Fulfilment of a responsibility often requires information spread across different partial experts.
- Then, objects need to interact to exchange information and to share the work
- Software object does those operations which are normally done by the domain object it represents (do it myself strategy)

Complete the Design of *total*()

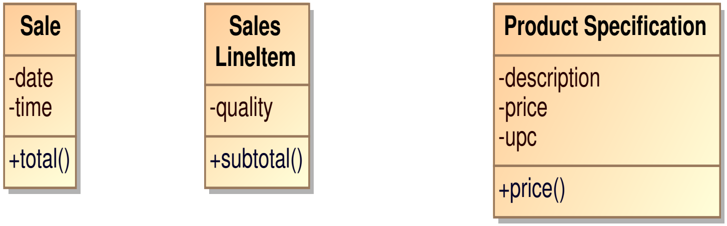


Complete the Design of *total*()



The design class diagram?

Design Class Diagram



associations are omitted!

Summary: Experts involved Total

Classes	**Responsibilities**
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

Summary: Experts involved Total

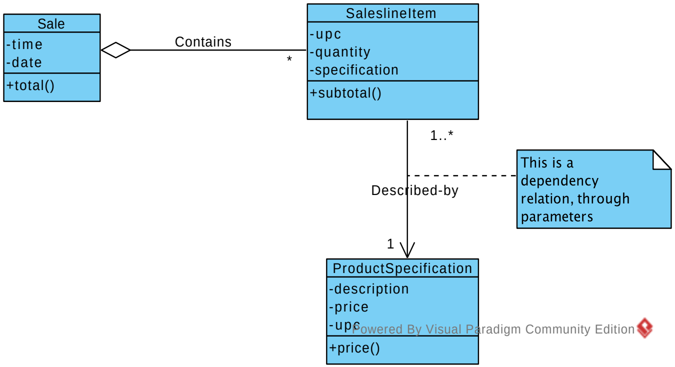
Classes	**Responsibilities**
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

These responsibilities identified and assigned while drawing the OSD

Design Class Diagram

Also create a **design class diagram** while creating a OSD, that refines the **conceptual class diagram** by

- adding methods to classes, and
- **navigation direction** of associations – **visibility** and **dependency**



Remarks on Expert

- Expert is used more than any other pattern
- Express the intuition that objects do things related to the information they have
- Fulfilment of a responsibility often requires information spread across different partial experts.
- Then, objects need to interact to exchange information and to share the work
- Software object does those operations which are normally done by the domain object it represents (do it myself strategy)

Creator Pattern

- Creation of objects is one of the most common functionality in an OO system

Creator Pattern

- Creation of objects is one of the most common functionality in an OO system
- It is important to have a general principle for the assignment of creation responsibility

Creator Pattern

- Creation of objects is one of the most common functionality in an OO system
- It is important to have a general principle for the assignment of creation responsibility
- **Creator**: the object B which invokes the creation of object A is called the creator of A

Pattern Name: Creator

Solution: **Assign class B the responsibility to create an instance of a class A if one of the following is true:**

- B **aggregates** A objects.
- B **contains** A objects.
- B **records** instances of A objects.
- B **closely uses** A objects.
- B **has the initialising data** that will be passed to A when it is created.

Problem: What should be responsible for creating a new instance of some class?

The object that has the direct reference to *B* is usually the creator of *B*

Example in POST: *enterItem*()

- Suppose there is a need to "Create a SalesLineItem",

Example in POST: *enterItem*()

- Suppose there is a need to "Create a SalesLineItem",
- Who should be responsible for creating *SalesLineItem*?

Example in POST: *enterItem*()

- Suppose there is a need to "Create a SalesLineItem",
- Who should be responsible for creating *SalesLineItem*?
- Follow Creator Pattern, we should look for a class that

Example in POST: *enterItem*()

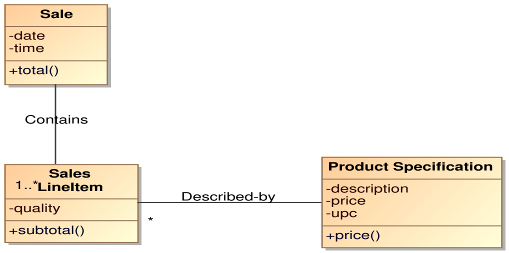
- Suppose there is a need to "Create a SalesLineItem",
- Who should be responsible for creating *SalesLineItem*?
- Follow Creator Pattern, we should look for a class that

  aggregates, *contains*, or *records SalesLineItem*

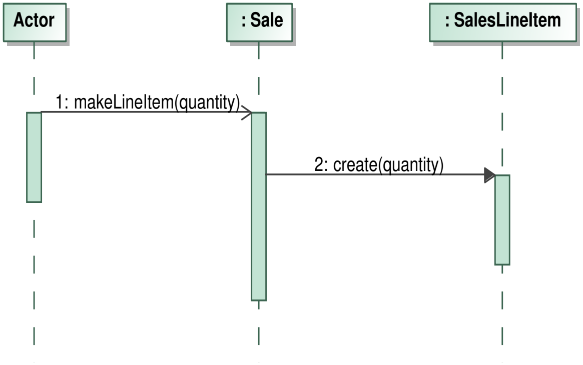
Example in POST: *enterItem*()

- Suppose there is a need to "Create a SalesLineItem",
- Who should be responsible for creating *SalesLineItem*?
- Follow Creator Pattern, we should look for a class that

  aggregates, *contains*, or *records SalesLineItem*



Answer: *Sale*



Add *makeLineItem*() method in class *Sale* too

Which object should be creator of the *Payment* of *Sale*?

- Consider Creator Pattern

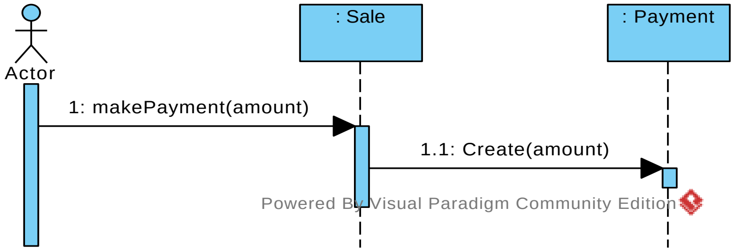
Which object should be creator of the *Payment* of *Sale*?

- Consider Creator Pattern

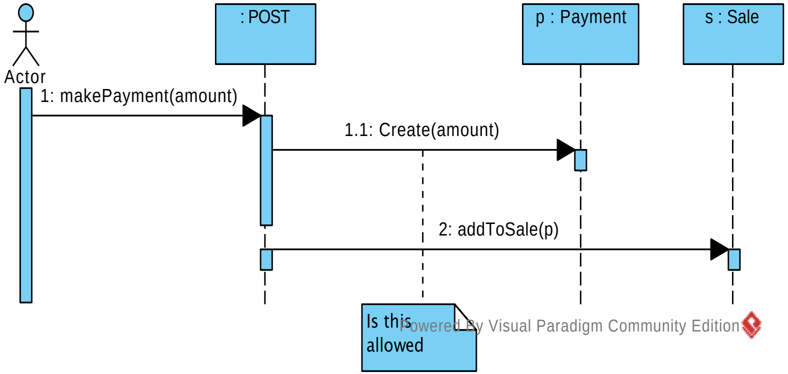
- Refer to the conceptual class diagram

Which object should be creator of the *Payment* of *Sale*?

- Consider Creator Pattern
- Refer to the conceptual class diagram



An Alternative Design?



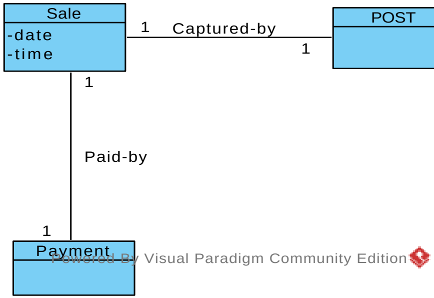
Why not right?

Why not right?

No association between *POST* and *Payment*

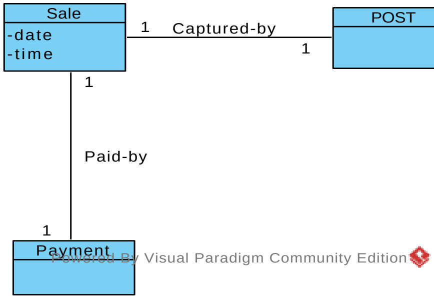
Why not right?

No association between *POST* and *Payment*



Why not right?

No association between *POST* and *Payment*



How about add an association?