UML Distilled

from UML Distilled by Martin Fowler

Introduction to UML

UML Distilled by Fowler, chapter 1

The Unified Modeling Language (UML) is a family of graphical notations, backed by a single meta-model, that help in describing and designing software systems, particularly software systems built using the object-oriented style.

UML, Fowler

Ways of using UML

- UML as Sketch – Developers use UML to help communicate some aspects of a system (this is the most common use)

- UML as Blueprint – Focus is on completeness. All design decisions are laid out
    - *Forward Engineering* – Create UML diagrams, then write code based on those diagrams
    - *Reverse Engineering* – Generate UML diagrams from existing code to help understand it

- UML as Programing Language – Write your entire program in UML. Requires sophisticated tools. Not mainstream

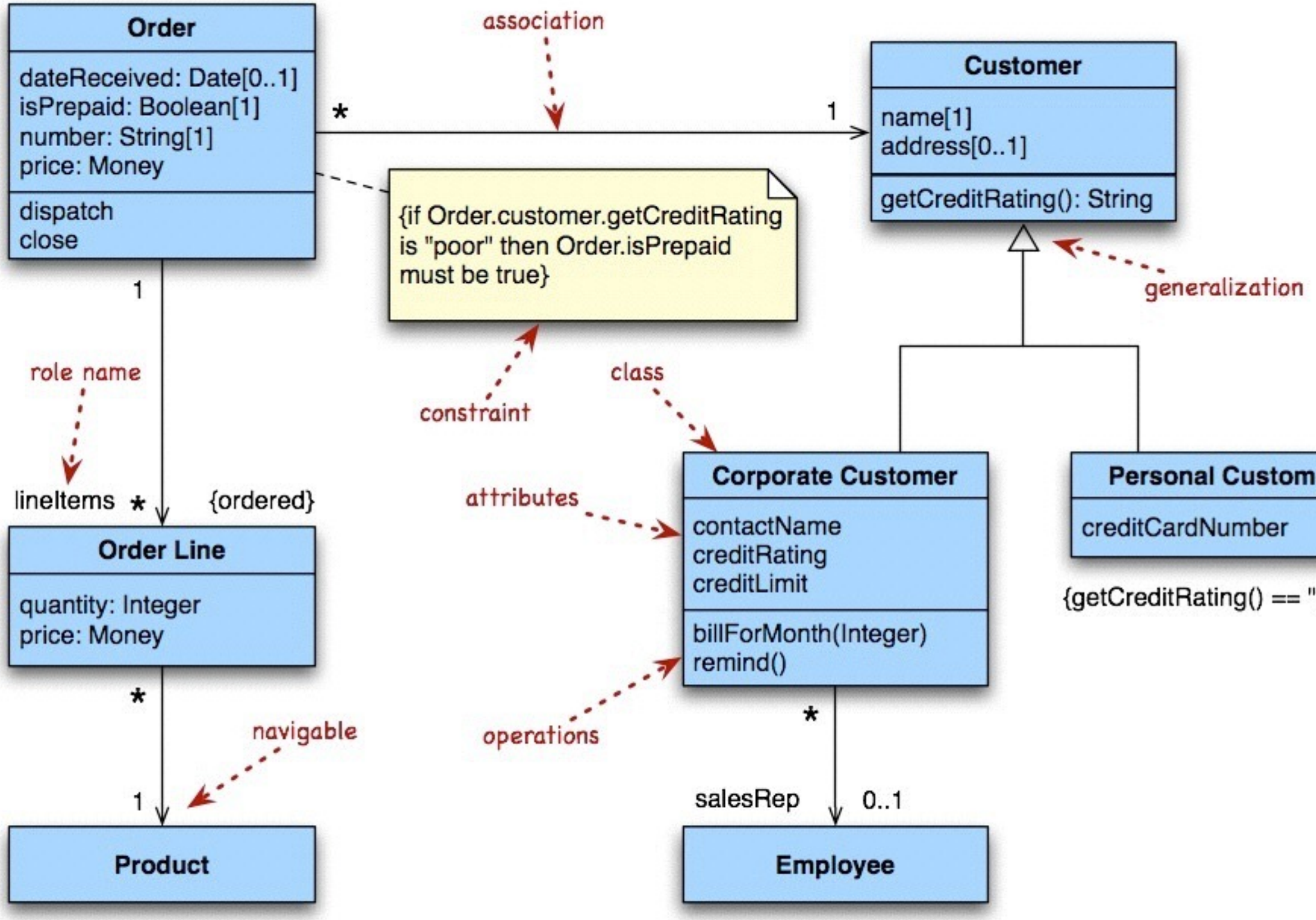
Obviously, good and legal is best, but you're better off putting your energy into having a good design rather than worrying about the arcana of UML.

You cannot look at a UML diagram and say exactly what the equivalent code would look like. However, you can get a rough idea of what the code would look like. In practice, that's enough to be useful.

UML, Fowler

Class Diagram Essentials

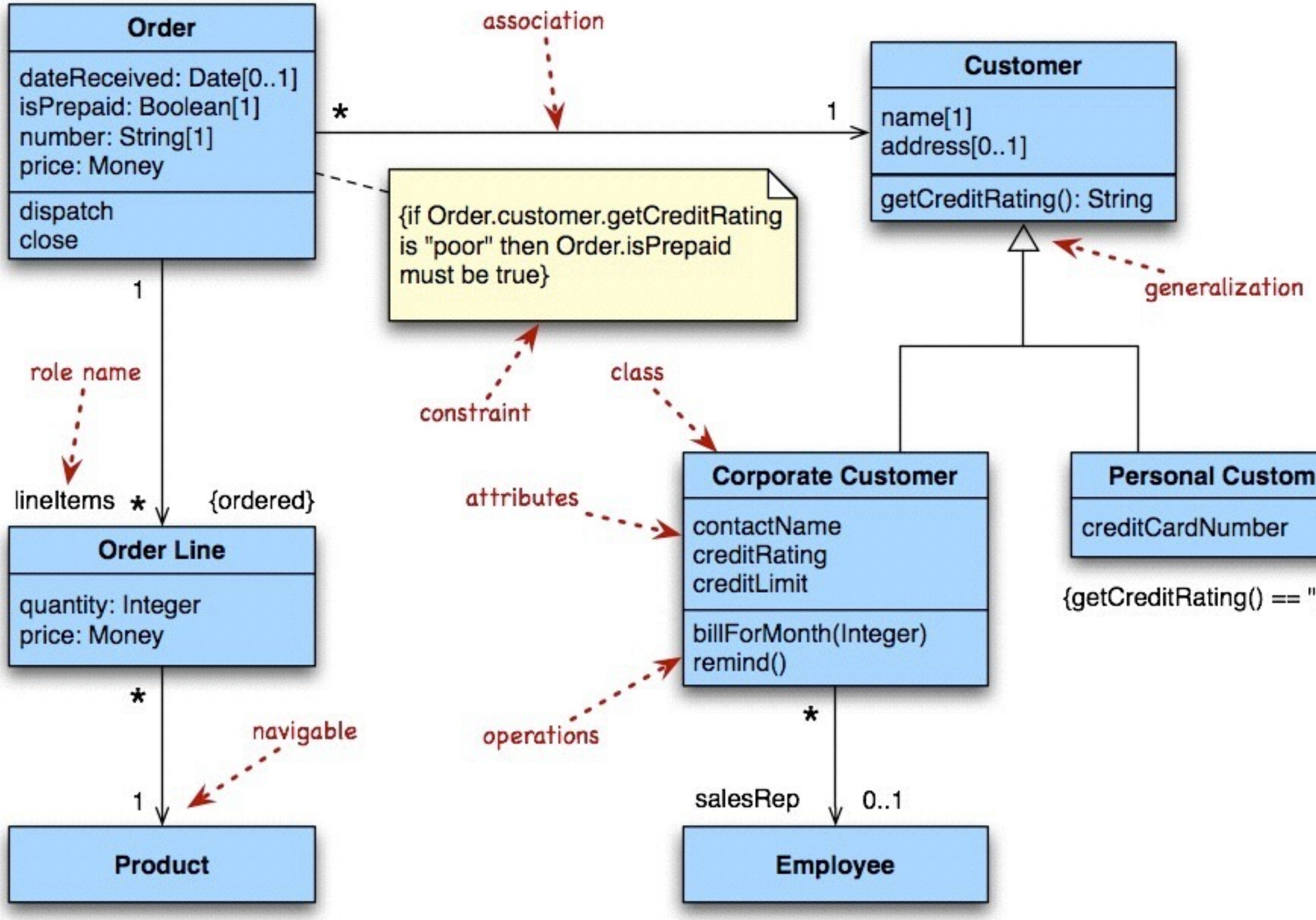
UML Distilled by Fowler, chapter 3


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String[1]
price: Money
dispatch
close
association
*
1
Customer
name[1]
address[0..1]
getCreditRating(): String
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}
generalization
role name
class
constraint
1
lineItems
*
{ordered}
Order Line
quantity: Integer
price: Money
*
navigable
1
Product
attributes
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
operations
*
salesRep
0..1
Employee
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}

Pause and Think

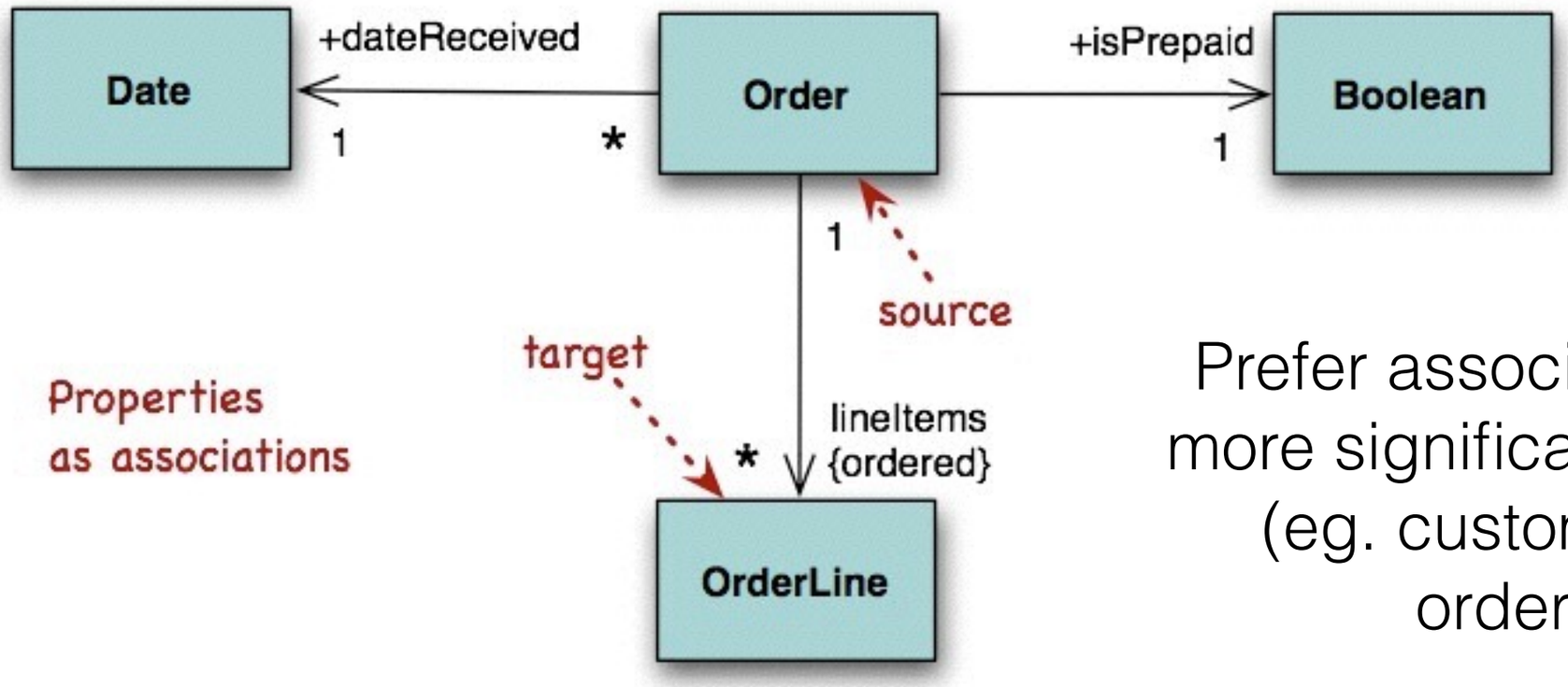
How would you implement the following attribute in Java?
Specifically, how would you handle the [0..1]?

dateReceived: Date [0..1]


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String[1]
price: Money
dispatch
close
association
*
1
Customer
name[1]
address[0..1]
getCreditRating(): String
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}
generalization
role name
class
constraint
1
lineItems
*
{ordered}
Order Line
quantity: Integer
price: Money
*
navigable
1
Product
attributes
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
operations
*
salesRep
0..1
Employee
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}

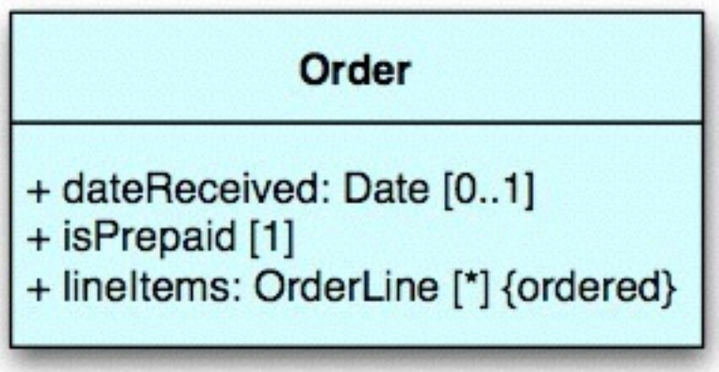
Pause and Think

How would you implement the following HAS-A relationship in Java?

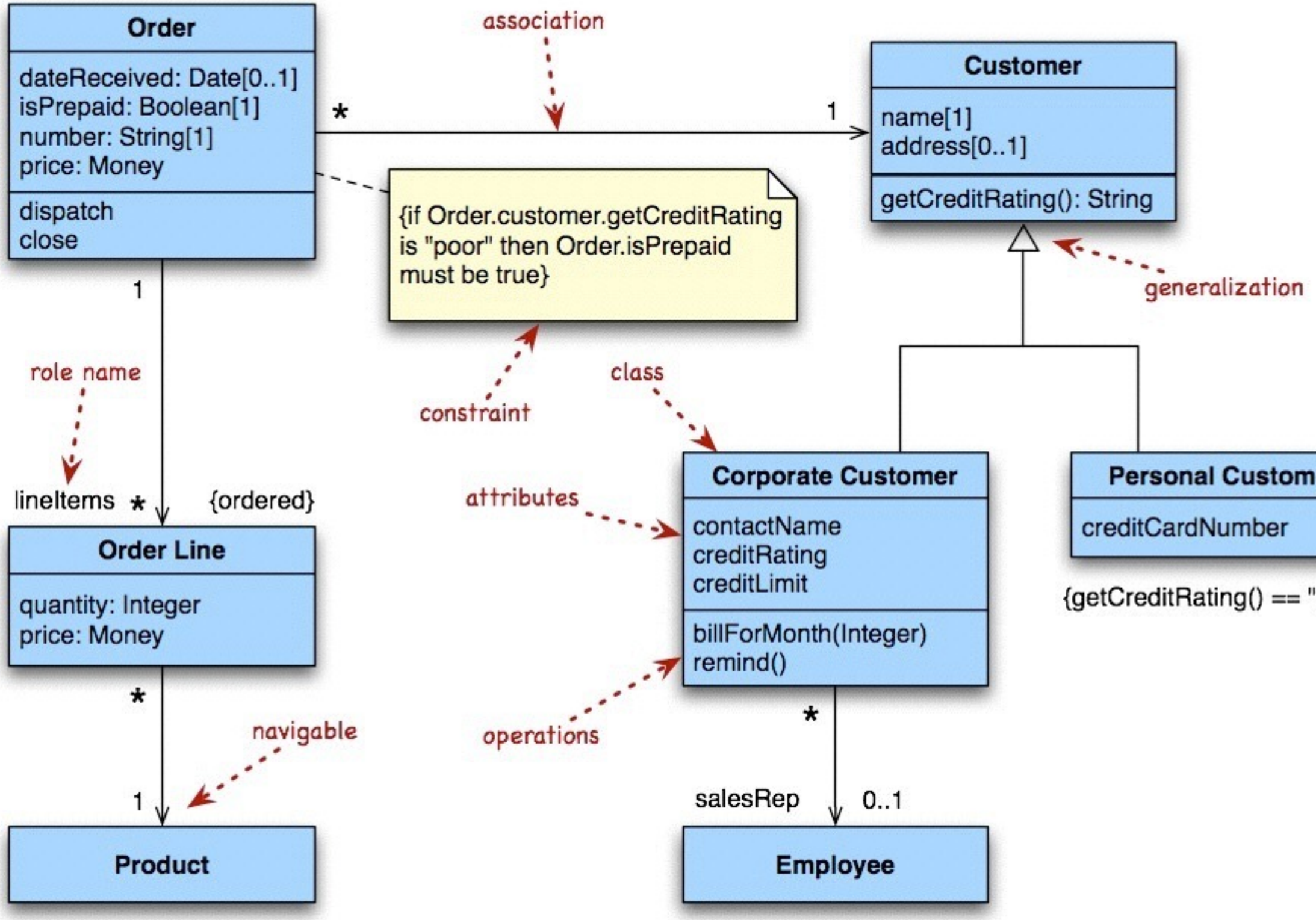


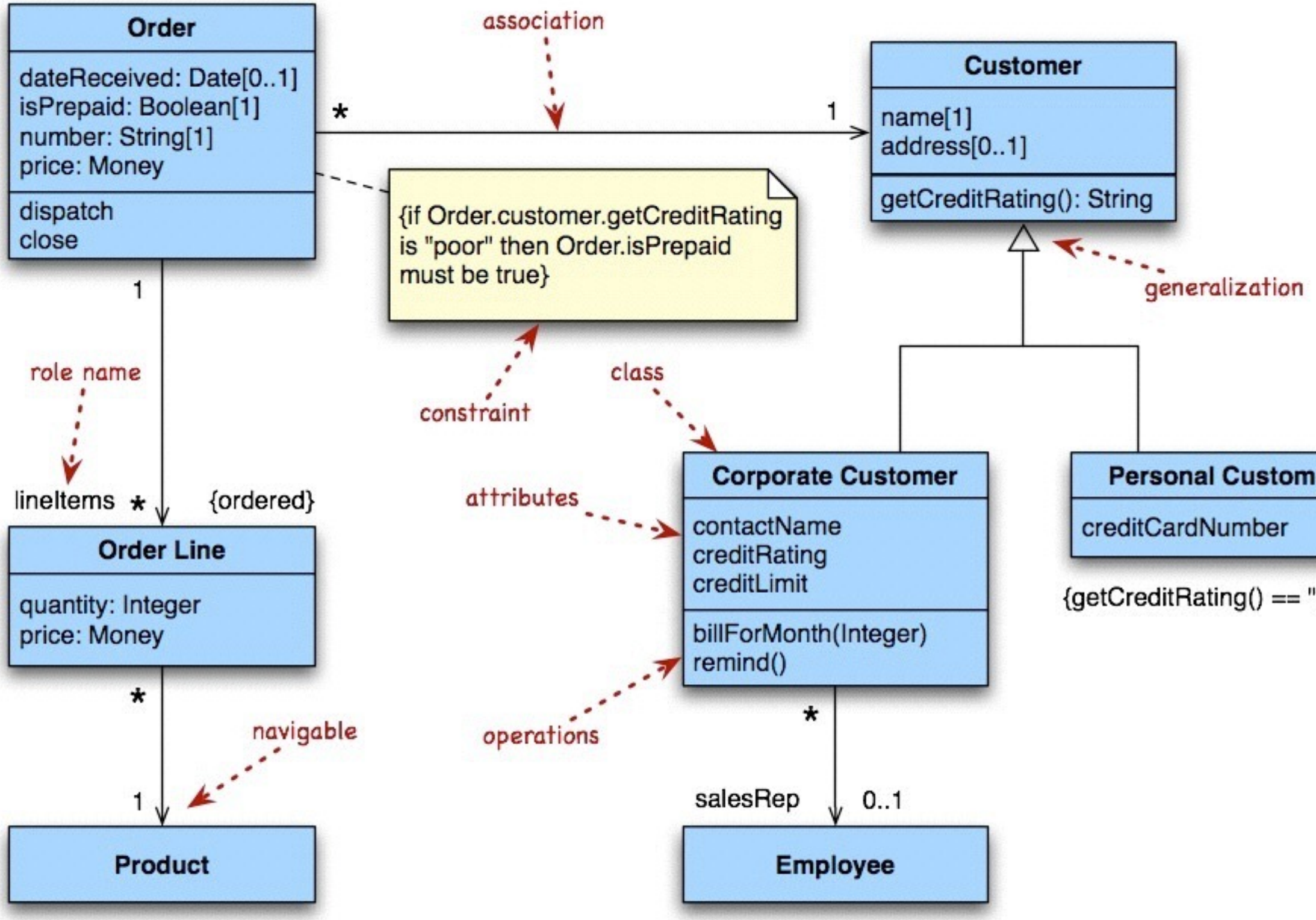
Prefer associations for more significant classes (eg. customers or orders)

Properties
as attribues



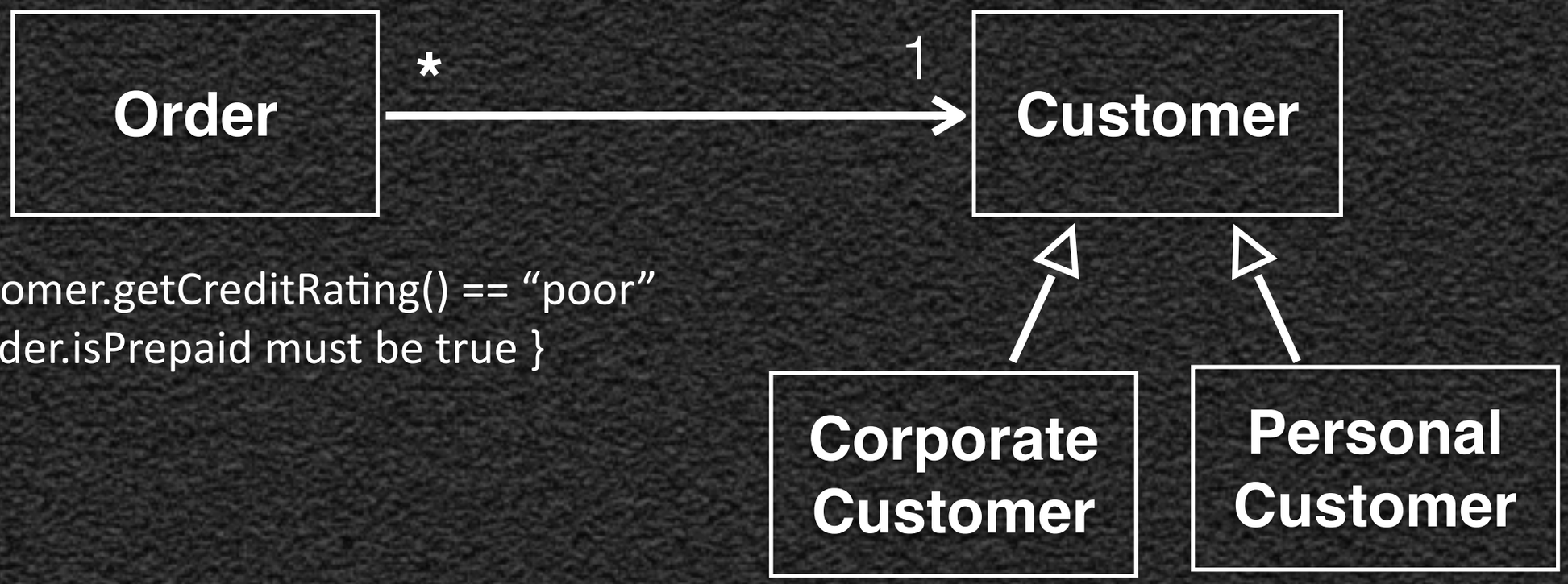
Prefer attributes for small objects (eg. dates or Booleans)


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String[1]
price: Money
dispatch
close
association
*
1
Customer
name[1]
address[0..1]
getCreditRating(): String
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}
generalization
role name
class
constraint
1
lineItems
*
{ordered}
Order Line
quantity: Integer
price: Money
*
navigable
1
Product
attributes
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
operations
*
salesRep
0..1
Employee
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String[1]
price: Money
dispatch
close
association
*
1
Customer
name[1]
address[0..1]
getCreditRating(): String
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}
generalization
role name
class
constraint
1
lineItems
*
{ordered}
Order Line
quantity: Integer
price: Money
*
navigable
1
Product
attributes
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
operations
*
salesRep
0..1
Employee
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}

Pause and Think

What can we say about <u>corporate</u> customers?

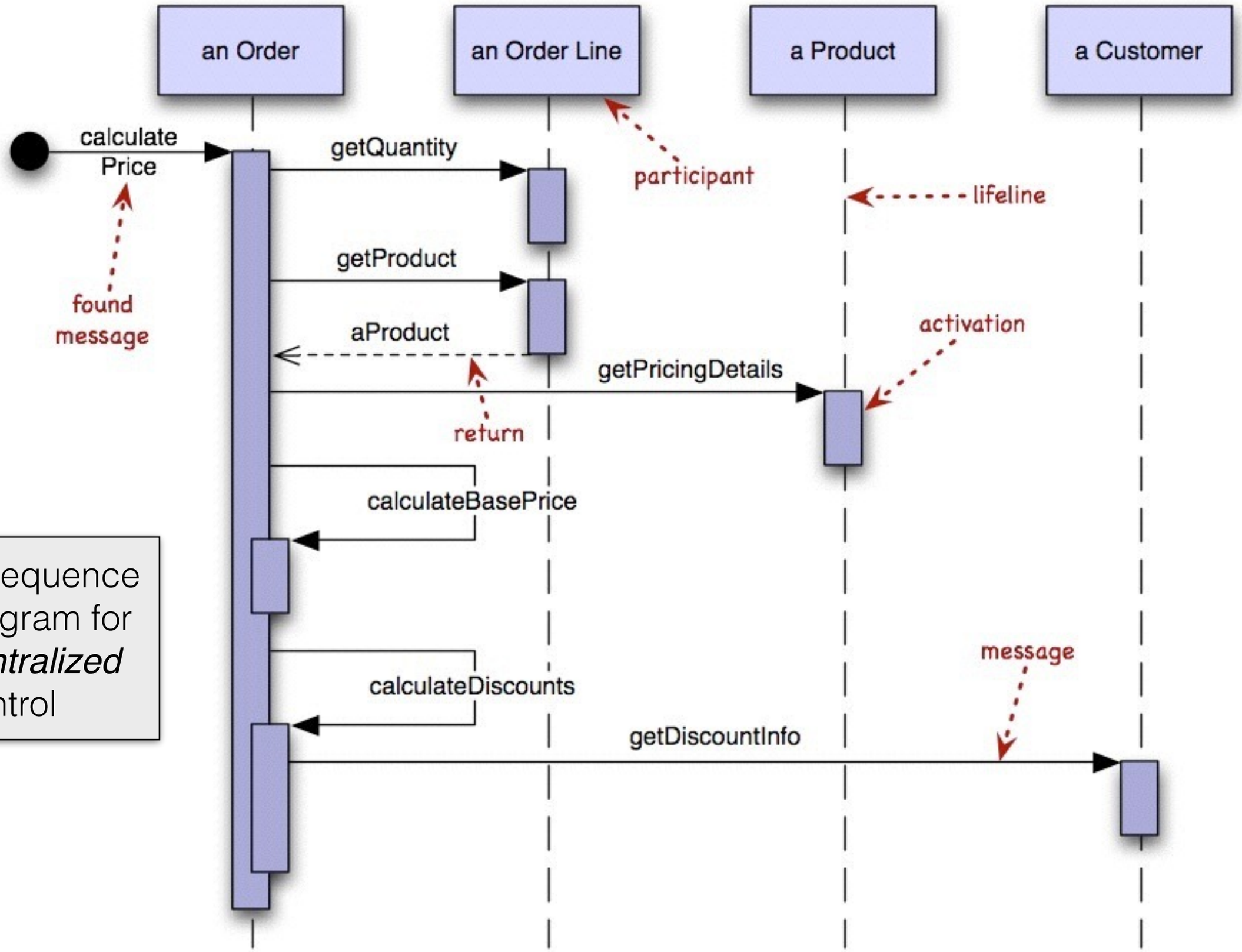


{ if Order.customer.getCreditRating() == “poor”
then Order.isPrepaid must be true }

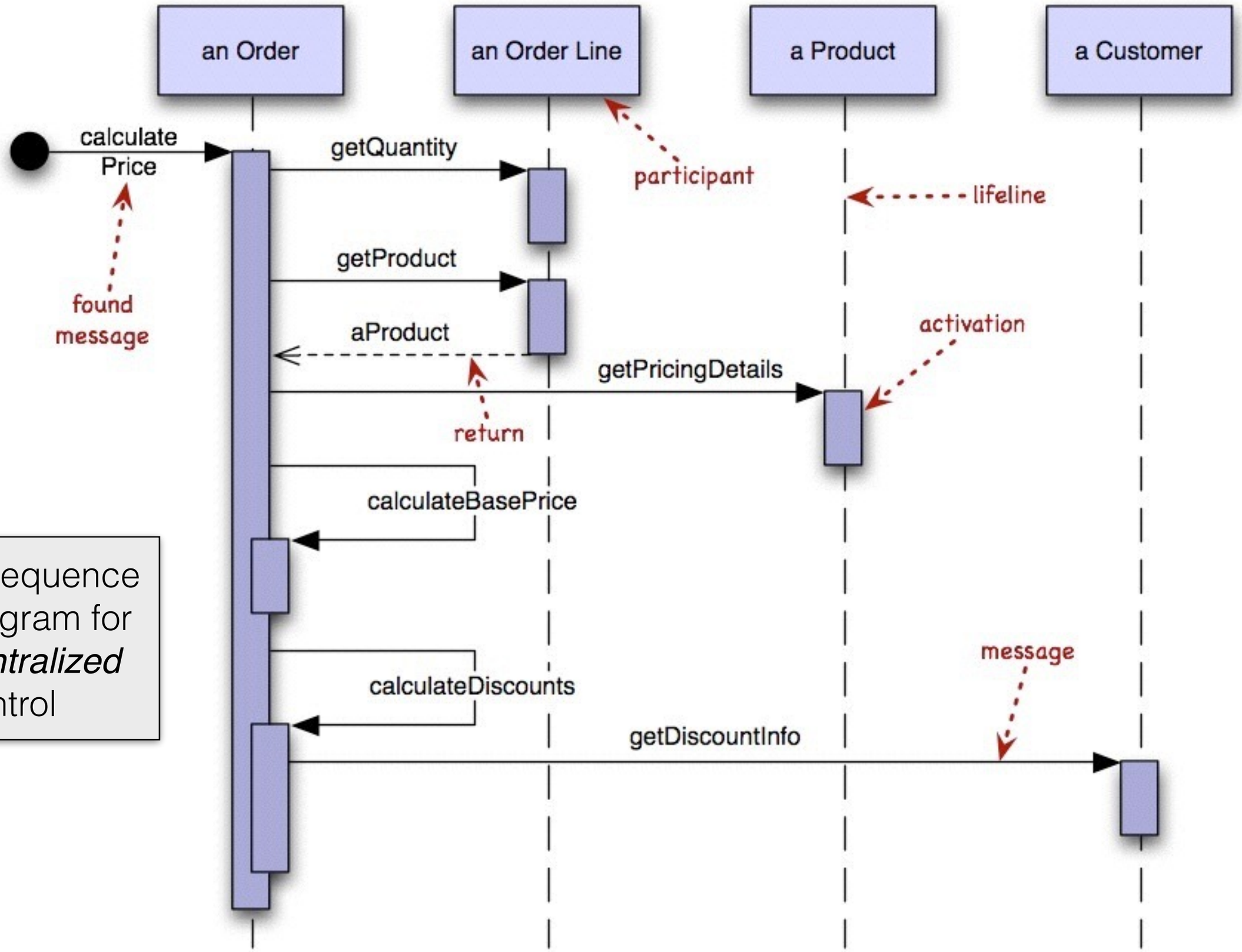
{ getCreditRating() == “poor” }

Sequence Diagrams

UML Distilled by Fowler, chapter 4

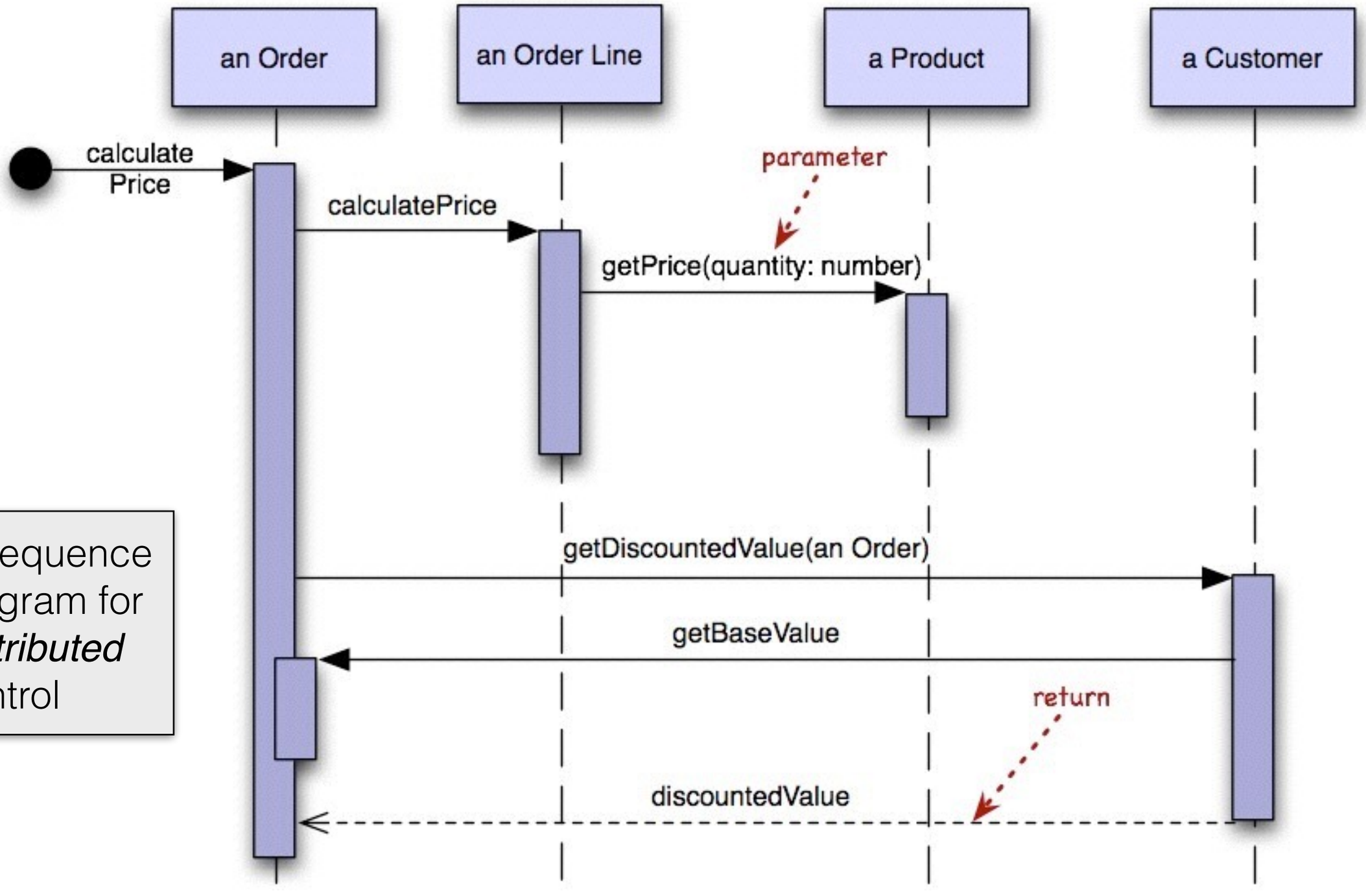


A sequence diagram for *centralized* control



A sequence diagram for *centralized* control

A sequence diagram for *distributed* control



Pause and Think

What do the two different sequence diagrams – centralized and distributed – for the same method (calculatePrice) represent?

A. Two different ways a visualizing the same method – calculatePrice – in the Object class

B. Two different executions of the same method?

C. Two completely different ways of implementing the calculatePrice method

Object bigots like me strongly prefer distributed control. One of the main goals of good design is to localize the effects of change.

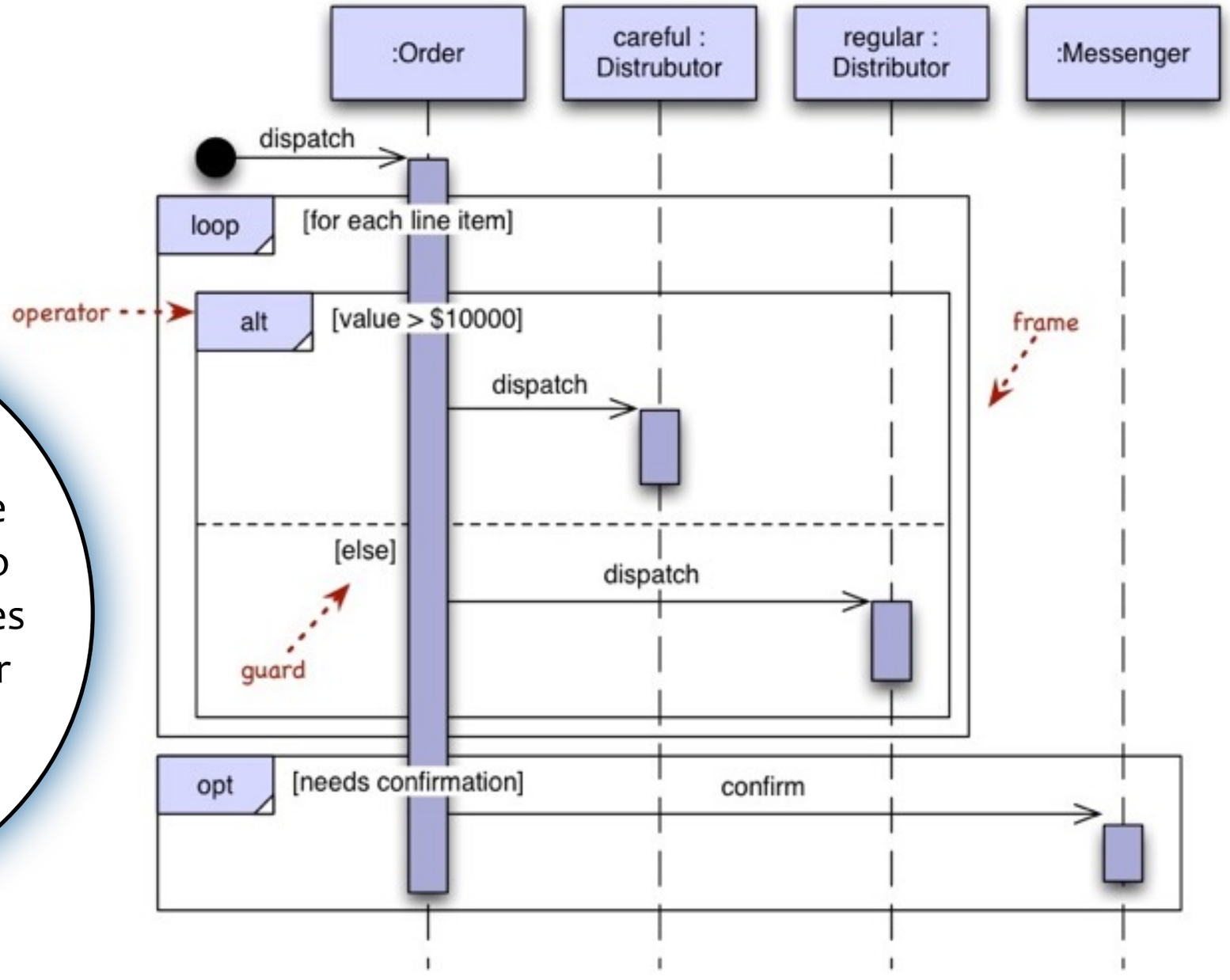
Data and the behavior that accesses that data often change together. So putting the data and the behavior that uses it together in one place is the first rule of object-oriented design.

UML, Fowler

A common issue with sequence diagrams is how to show looping and conditional behavior.

This isn't what sequence diagrams are good at. If you want to show control structures like this, you are better off with an activity diagram.

UML, Fowler



Interaction frames – Use only if you feel you are *forced* to show flow structures in a sequence diagram.

Class Diagrams: Advanced Concepts

UML Distilled by Fowler, chapter 5

Aggregation and Composition

Aggregation is the part-of relationship. It's like saying that a car has an engine and wheels as its parts.

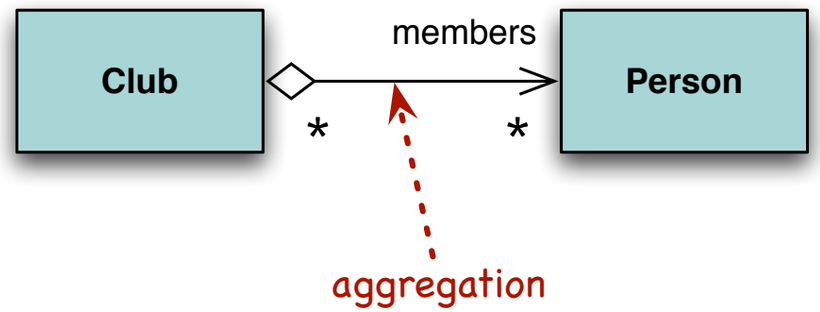
This sounds good, but the difficult thing is considering what the difference is between aggregation and association.

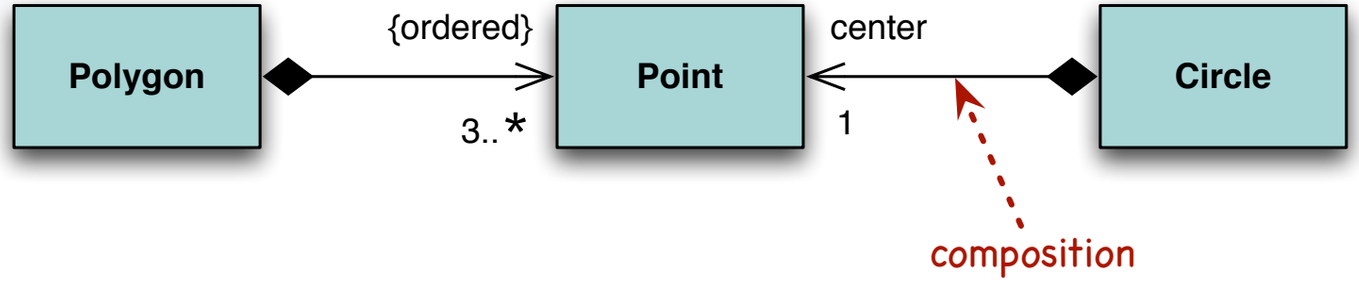
Aggregation can occur when a class is a collection or container of other classes, but if the container is destroyed, its contents are not.

– *Wikipedia*

Composition is a good way of showing properties that own by value. If you delete the polygon, it should automatically ensure that any owned points are also deleted.

UML, Fowler

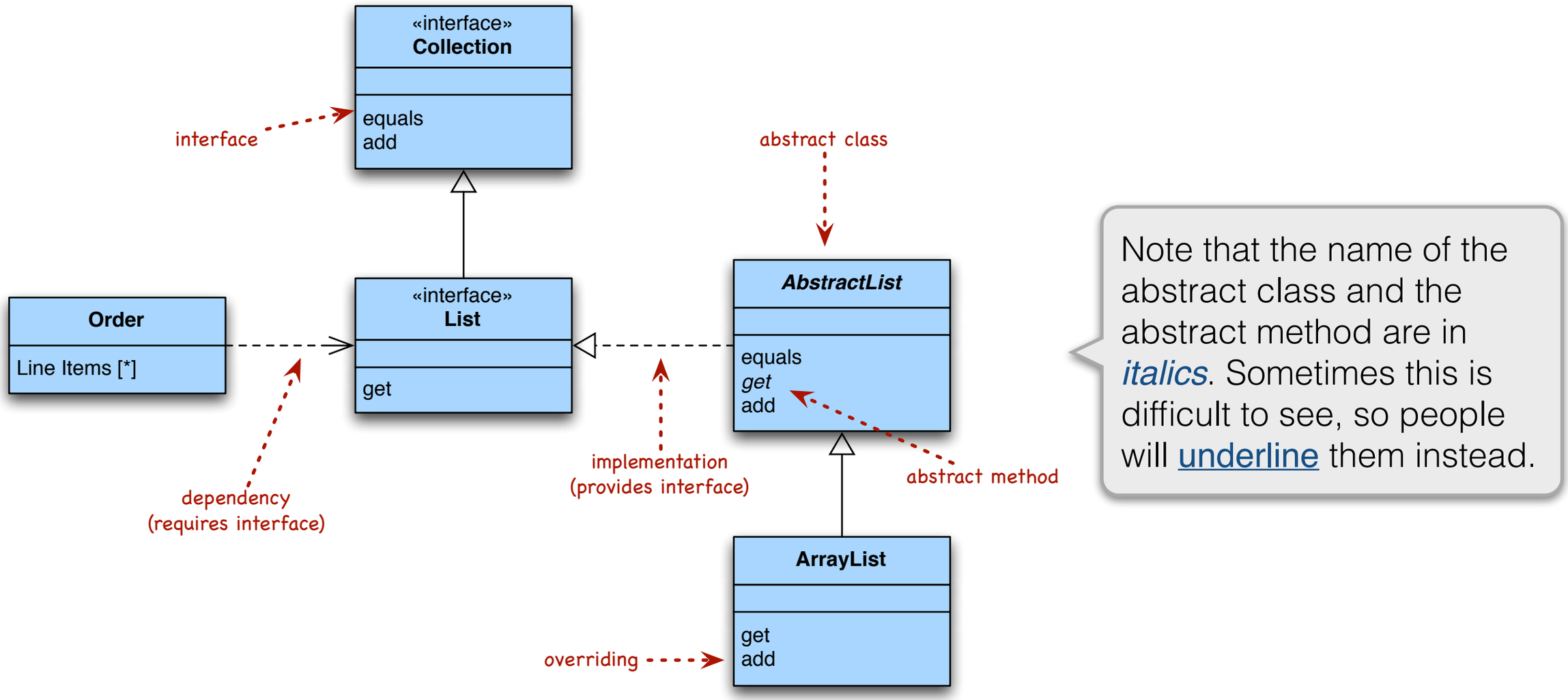




Pause and Think

Which relationship makes more sense?

Interfaces & abstract classes

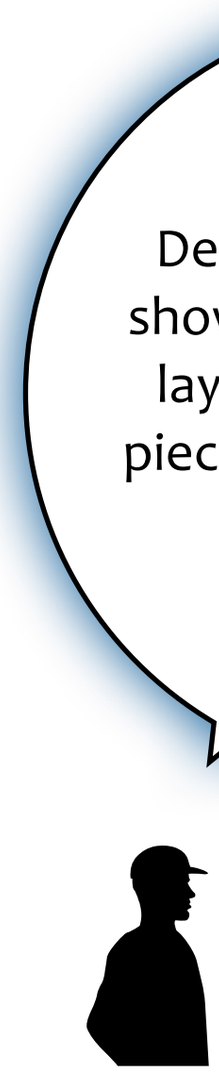


Deployment Diagrams

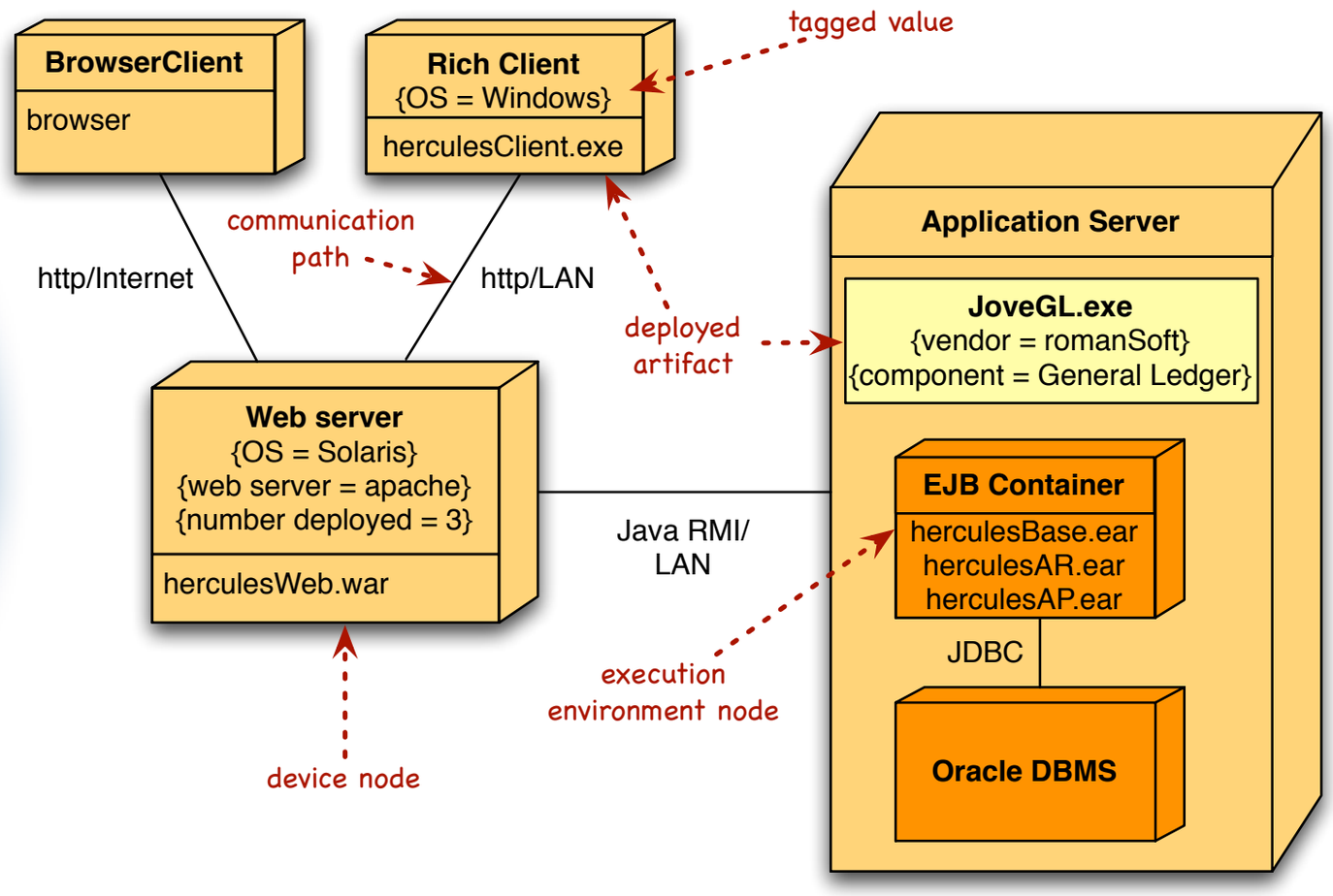
UML Distilled by Fowler, chapter 8

Deployment diagram

Deployment diagrams show a system's physical layout, revealing which pieces of software run on what pieces of hardware.



UML, Fowler



Use Cases

UML Distilled by Fowler, chapter 9

Buy a Product

- The customer browses the catalog and adds desired items to the shopping basket. When the customer wishes to pay, they give shipping and credit card information and confirm the sale.
- The system checks authorization on the credit card and confirms the sale immediately and with a follow-up email.

Buy a product

Goal Level: Sea Level

Main Success Scenario:

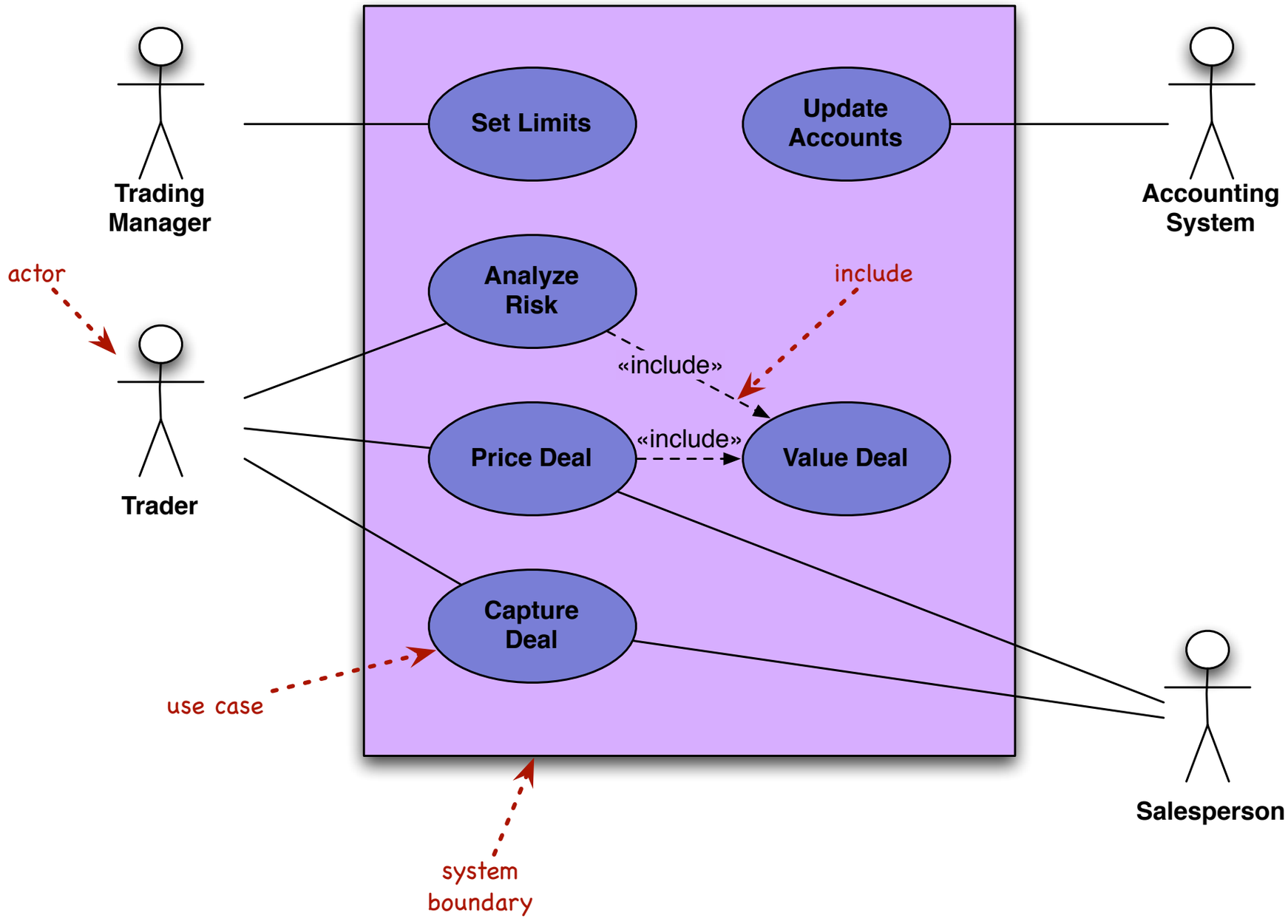
1. Customer browses catalog and selects items to buy
2. Customer goes to checkout
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming e-mail to customer

Extensions:

3a: Customer is regular customer
   1: System displays current shipping, pricing, and billing information
   2: Customer may accept or override these defaults, returns to MSS at step 6

6a: System fails to authorize credit purchase
   1: Customer may reenter credit card information or may cancel

Use case diagram



Although the diagram is sometimes useful, it isn't mandatory. In your use case work, don't put too much effort into the diagram. Instead, concentrate on the textual content of the use cases.

UML, Fowler

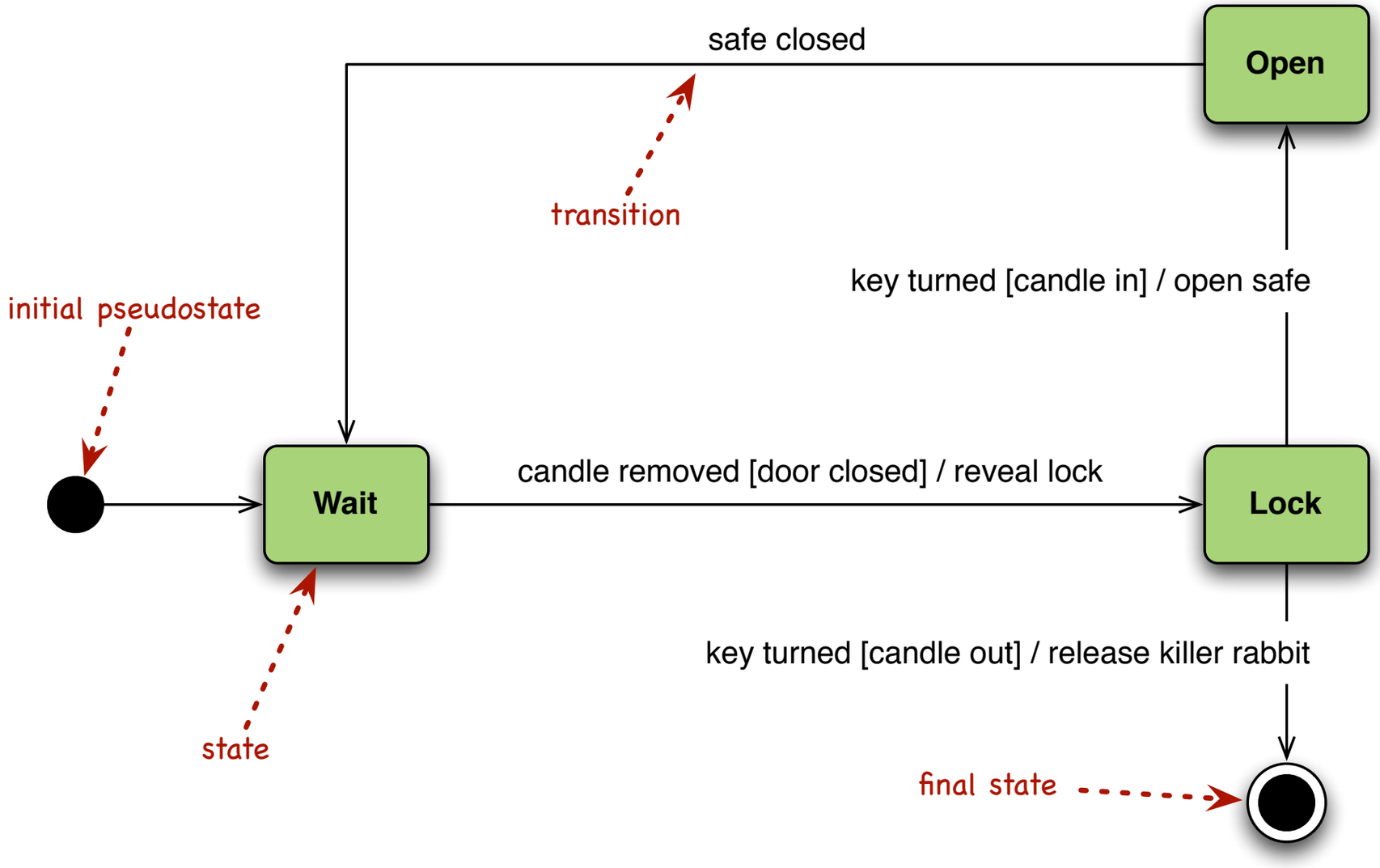
State Machine Diagrams

UML Distilled by Fowler, chapter 10

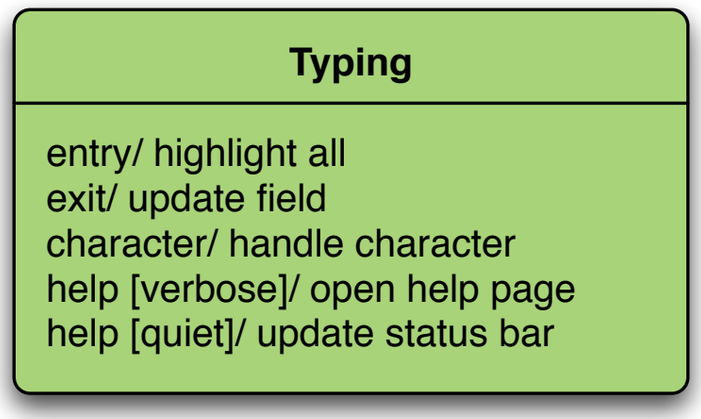
In object-oriented approaches, you draw a state machine diagram for a single class to show the lifetime behavior of a single object.

UML, Fowler

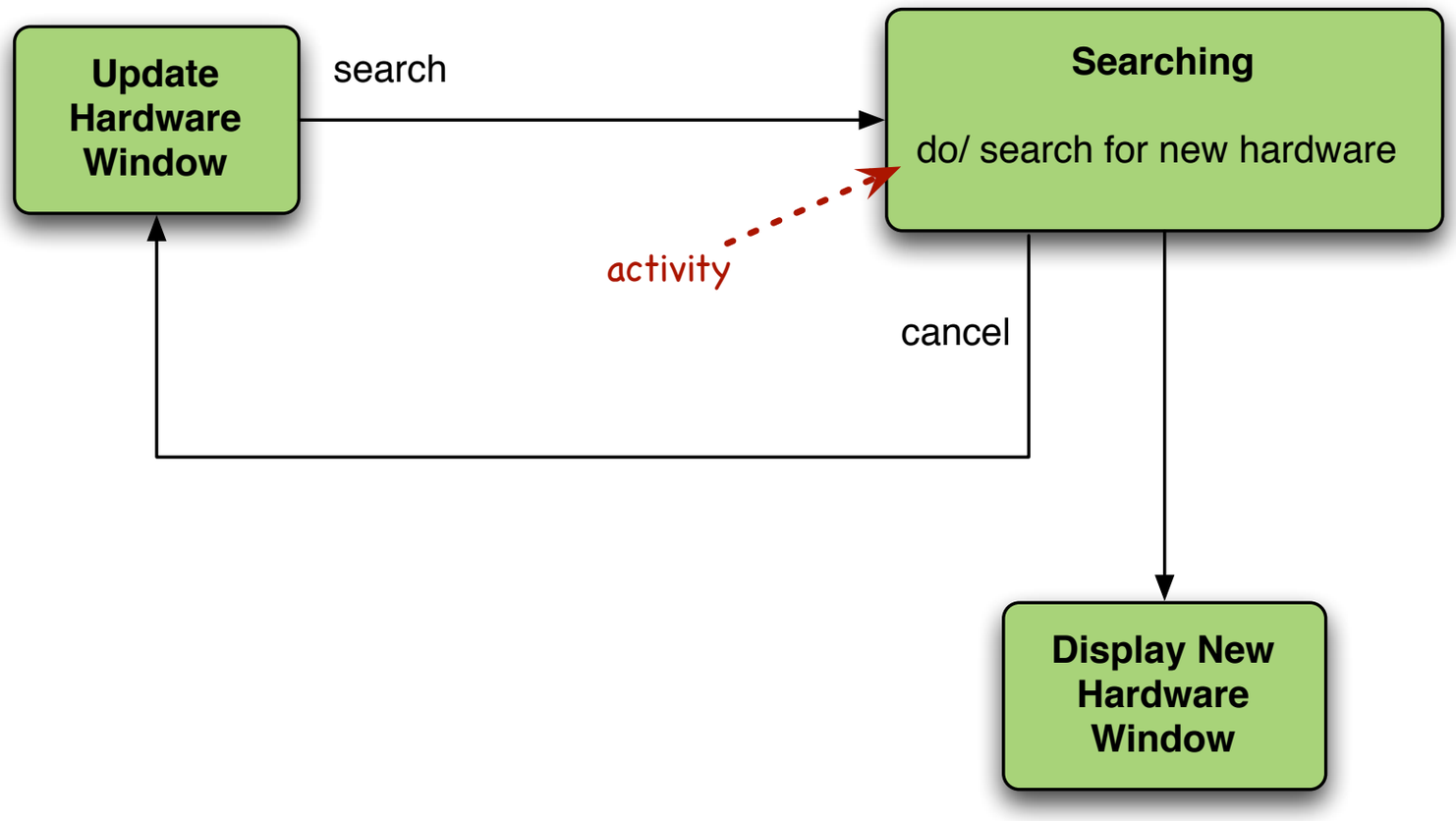
Simple state machine diagram



Internal activities

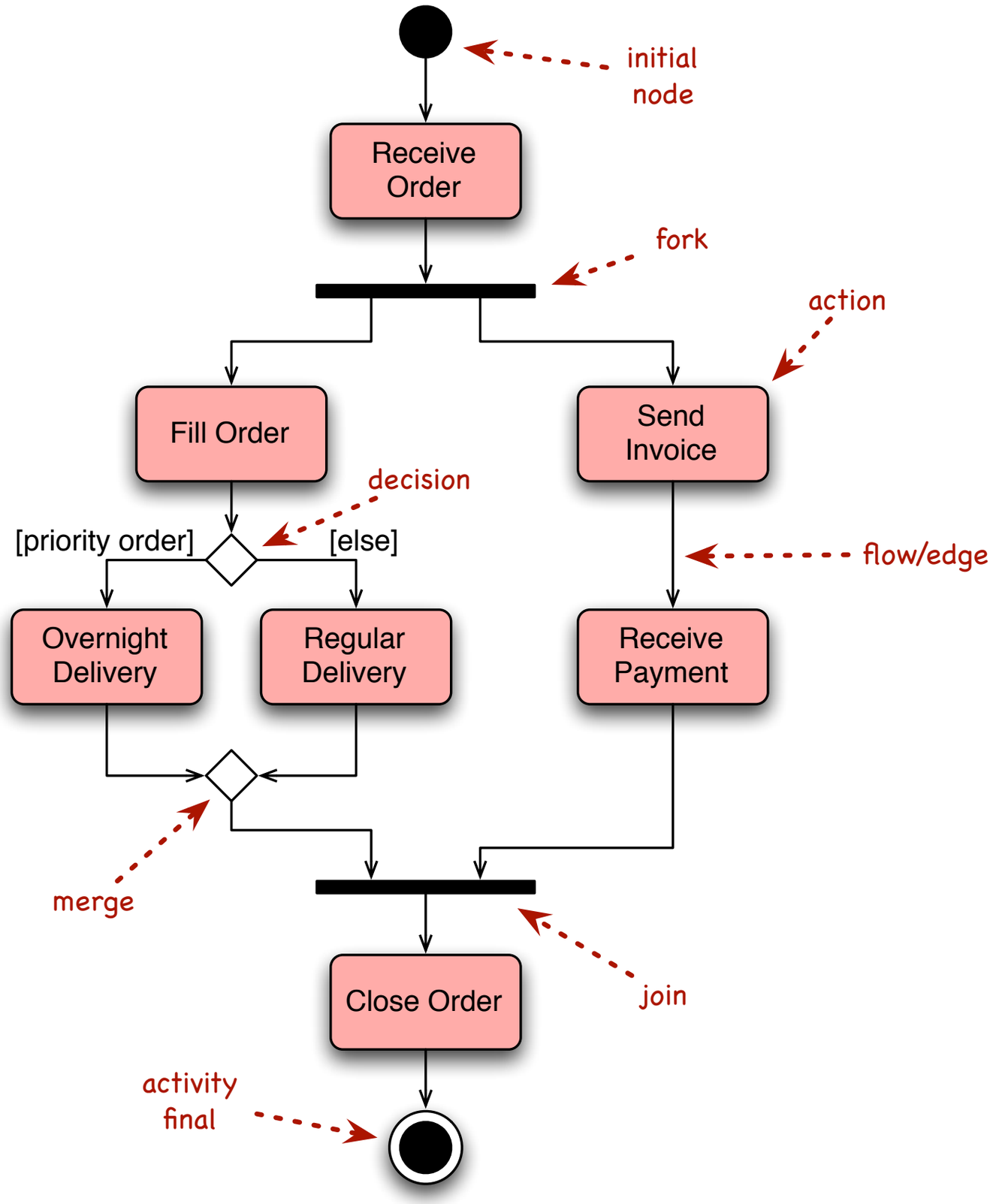


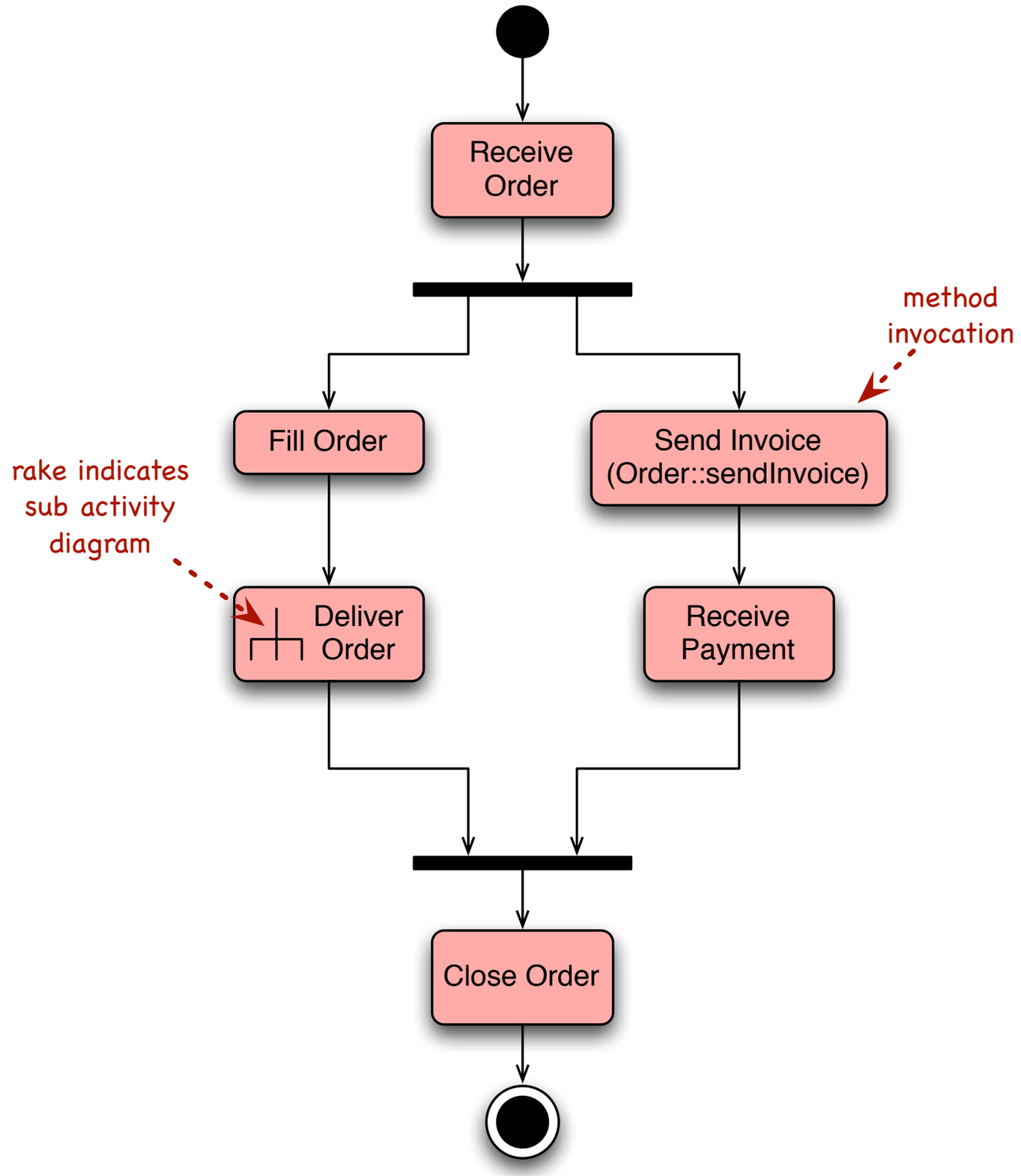
State with an activity



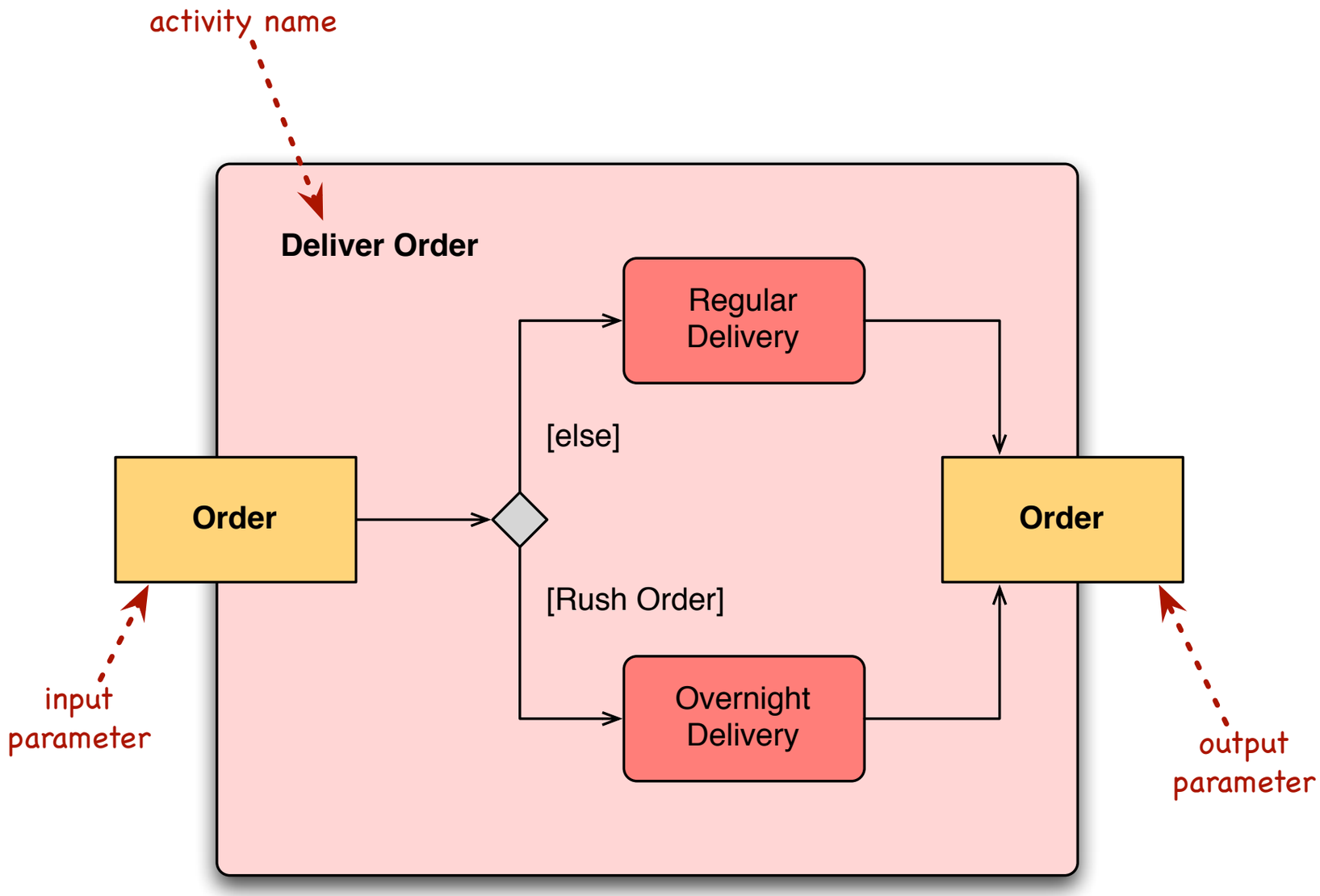
Activity Diagrams

UML Distilled by Fowler, chapter 11


initial node
Receive Order
fork
action
Fill Order
Send Invoice
decision
[priority order]
[else]
flow/edge
Overnight Delivery
Regular Delivery
Receive Payment
merge
Close Order
join
activity final


Receive Order
Fill Order
Send Invoice (Order::sendInvoice)
method invocation
rake indicates sub activity diagram
Deliver Order
Receive Payment
Close Order

Subsidiary activity diagram



Swimlane Diagram