Software Specification & Design

Midterm Material (design patterns not included)

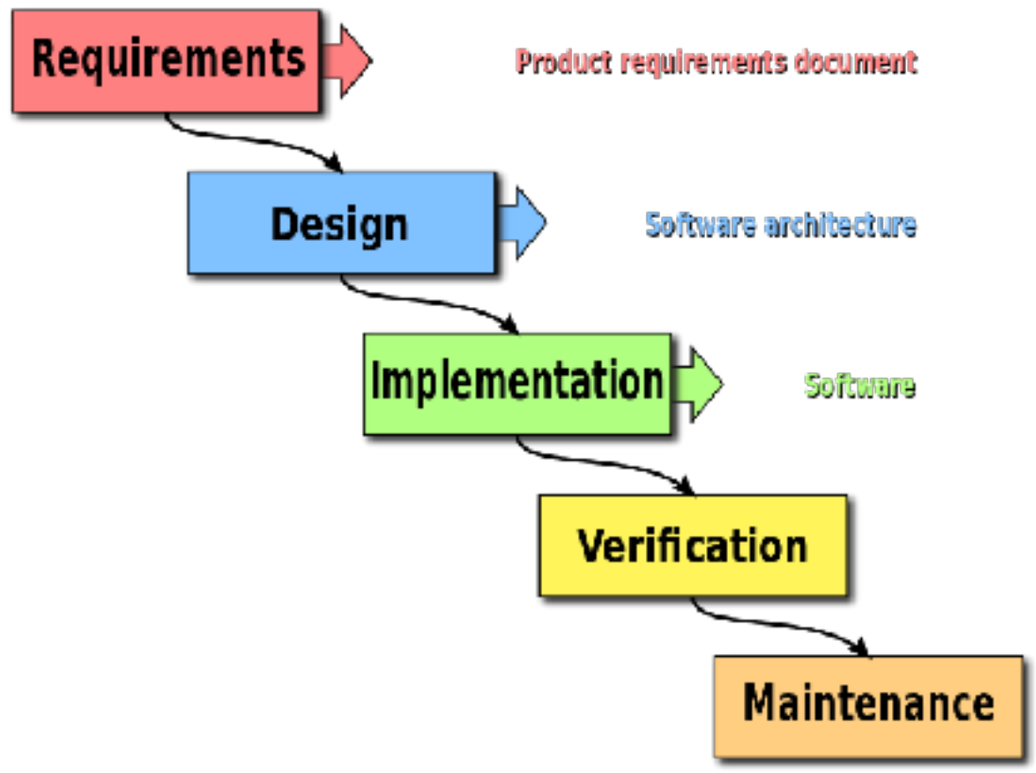
Intro. Software Process

- Aka Software development process
- What is process?
- Do we need process?
- What are examples of popular processes?

Waterfall Model

- Sequential
- Construction industries
- Define most of requirements at the beginning
- Advantages and disadvantages?

Waterfall Model



http://en.wikipedia.org/wiki/Waterfall_model

Waterfall Model

- Problems
  - Users don’t know what they actually want.
  - Too late to go back
  - Requirements change ~ 25% - 50%
  - The bigger project, the more change

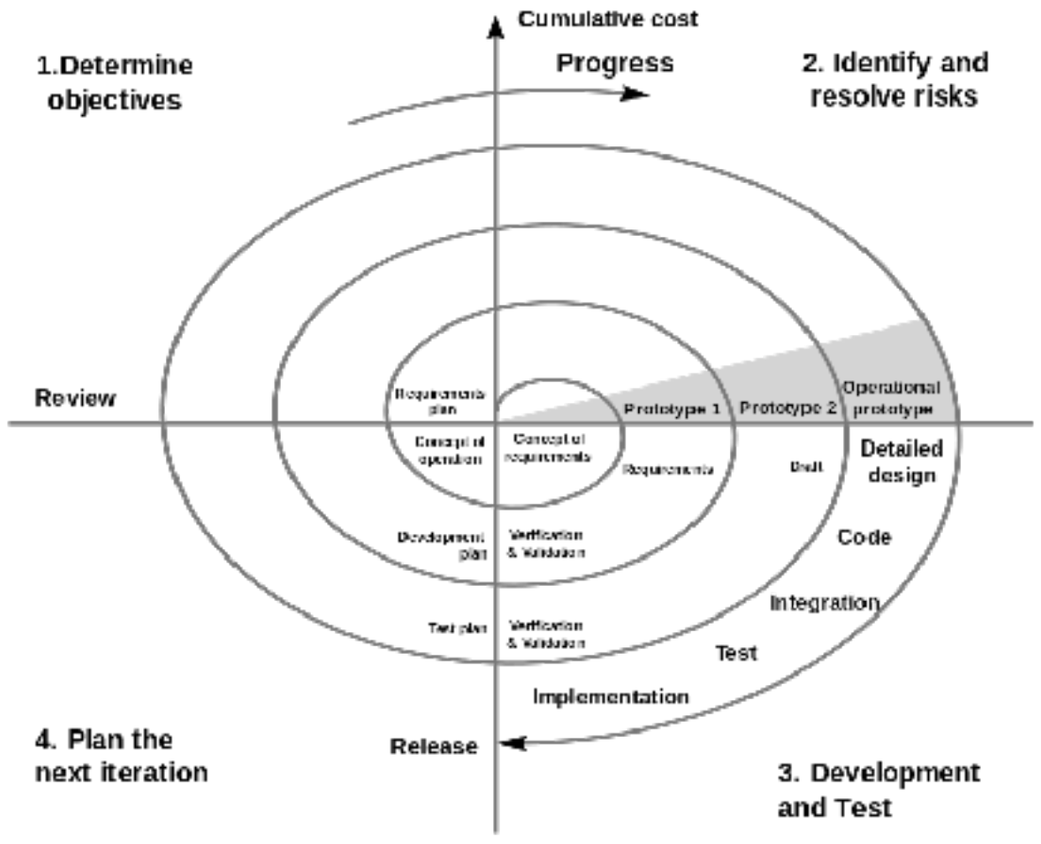
Waterfall Model

- So should we use it?
  - If the requirements are well known, clear and fixed. (Not likely, but possible)
  - You have enough expertise
  - Fix contracts/deliver date/budget

Iterative and Evolutionary Development

- Iterations
- Each can be thought as a mini project
- The system grows over time
- Iterative and Incremental development
  (The names gave different meanings for different people)

Iterative and Evolutionary Development



https://en.wikipedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg

Iterative and Evolutionary Development

- Nature
  - Embrace change
  - Early iterations are far from the true path of the system
  - In late iterations, significant change is rare (But can occur)

Iterative and Evolutionary Development

- Benefits
  - Less project failure
  - Early visible progress
  - Early feedback
  - Reduce complexity

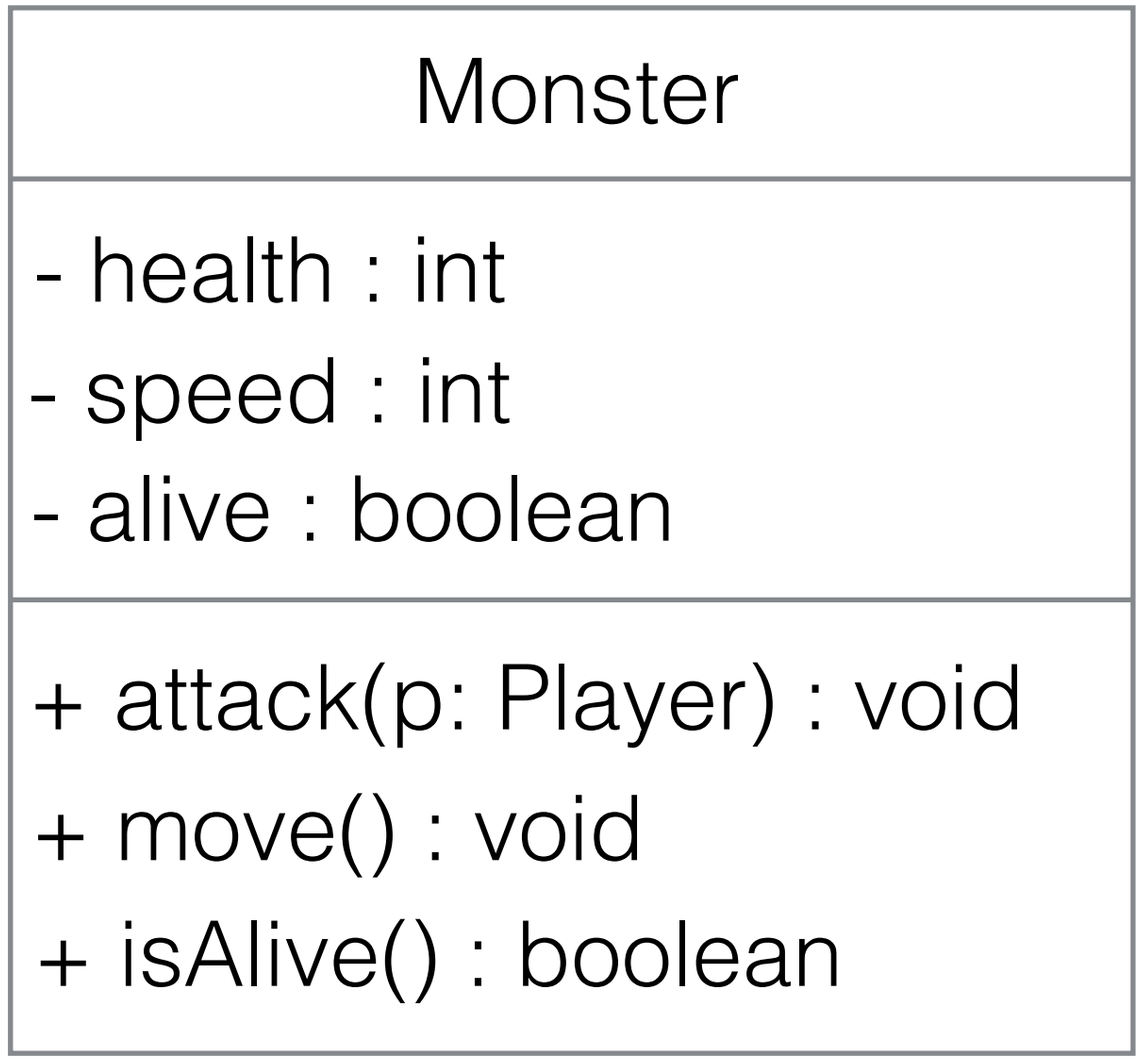
Intro. Software Design

- There are many layers in software design
- From architectural level to implementation level
- Let’s see examples

Java & OOP Review

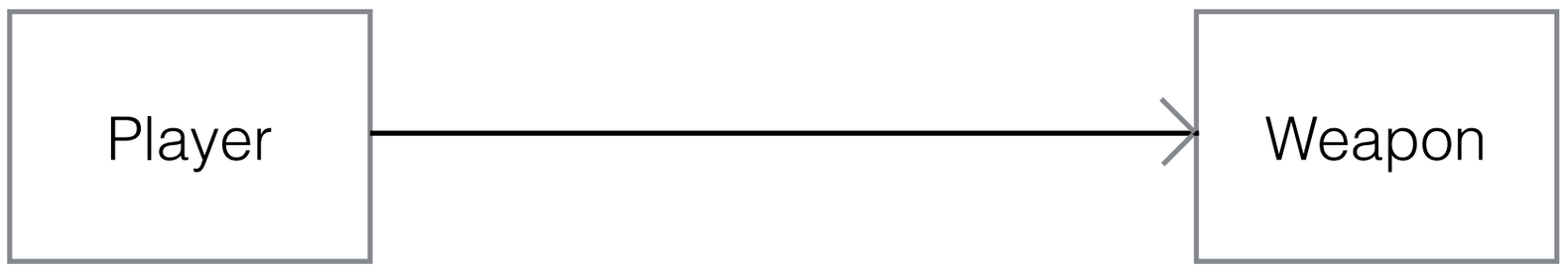
- Before moving on, we will review about
    - Different types of classes in Java
    - Objects and there default methods
    - Inheritance
    - Interface
    - Common classes in Java such as List, Set, Map

UML Class Diagram



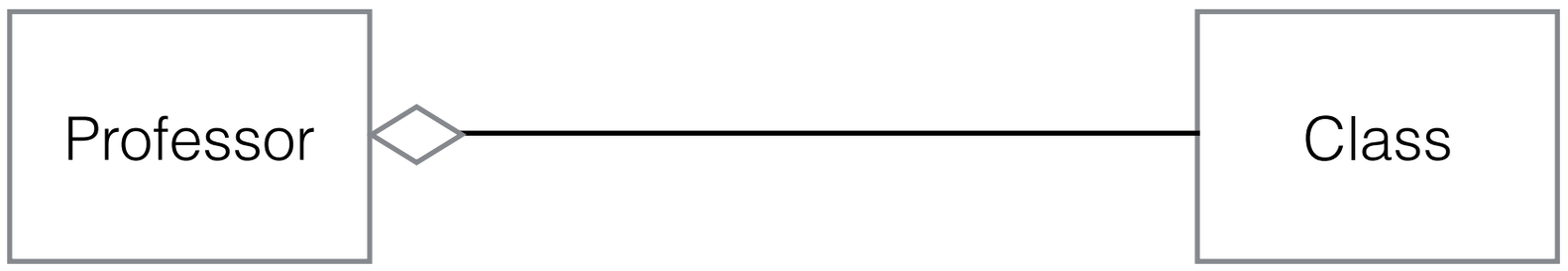
UML Class Diagram

- Direct Association



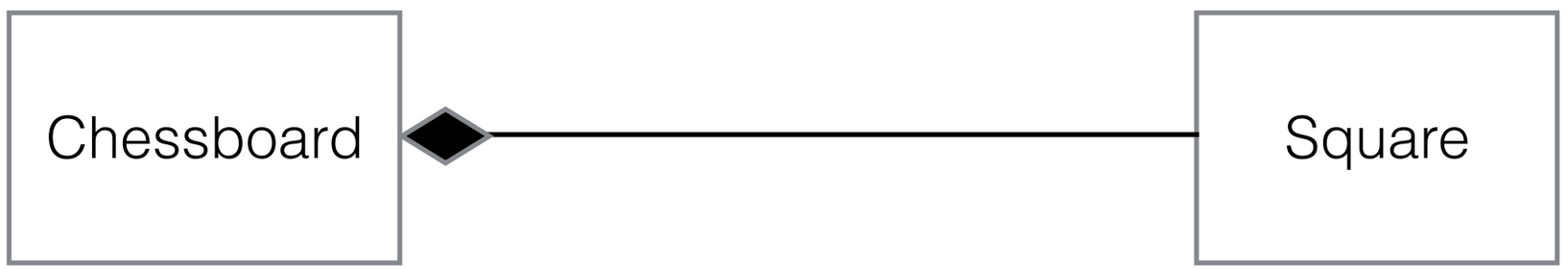
UML Class Diagram

- Aggregation



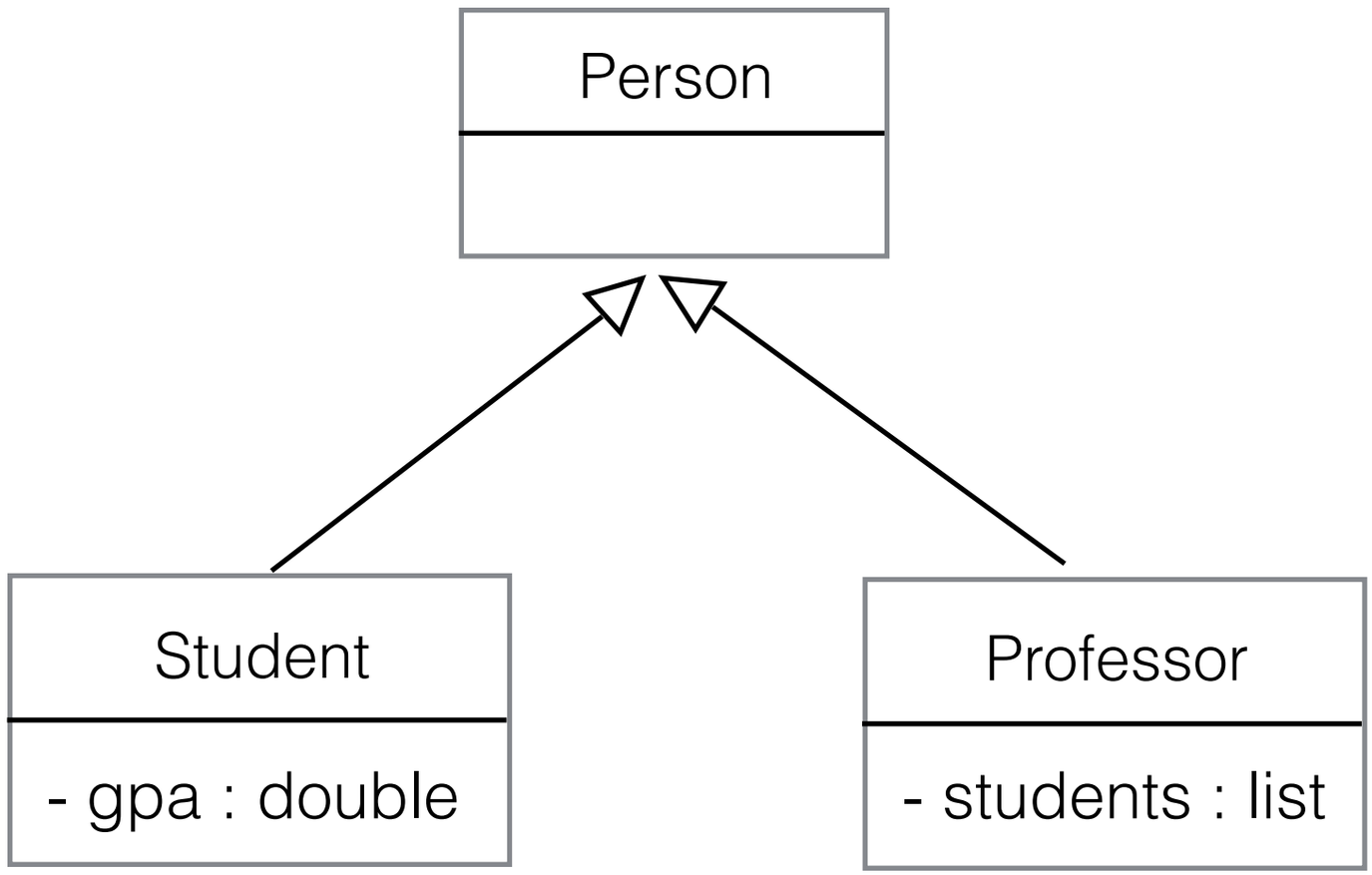
UML Class Diagram

- Composition



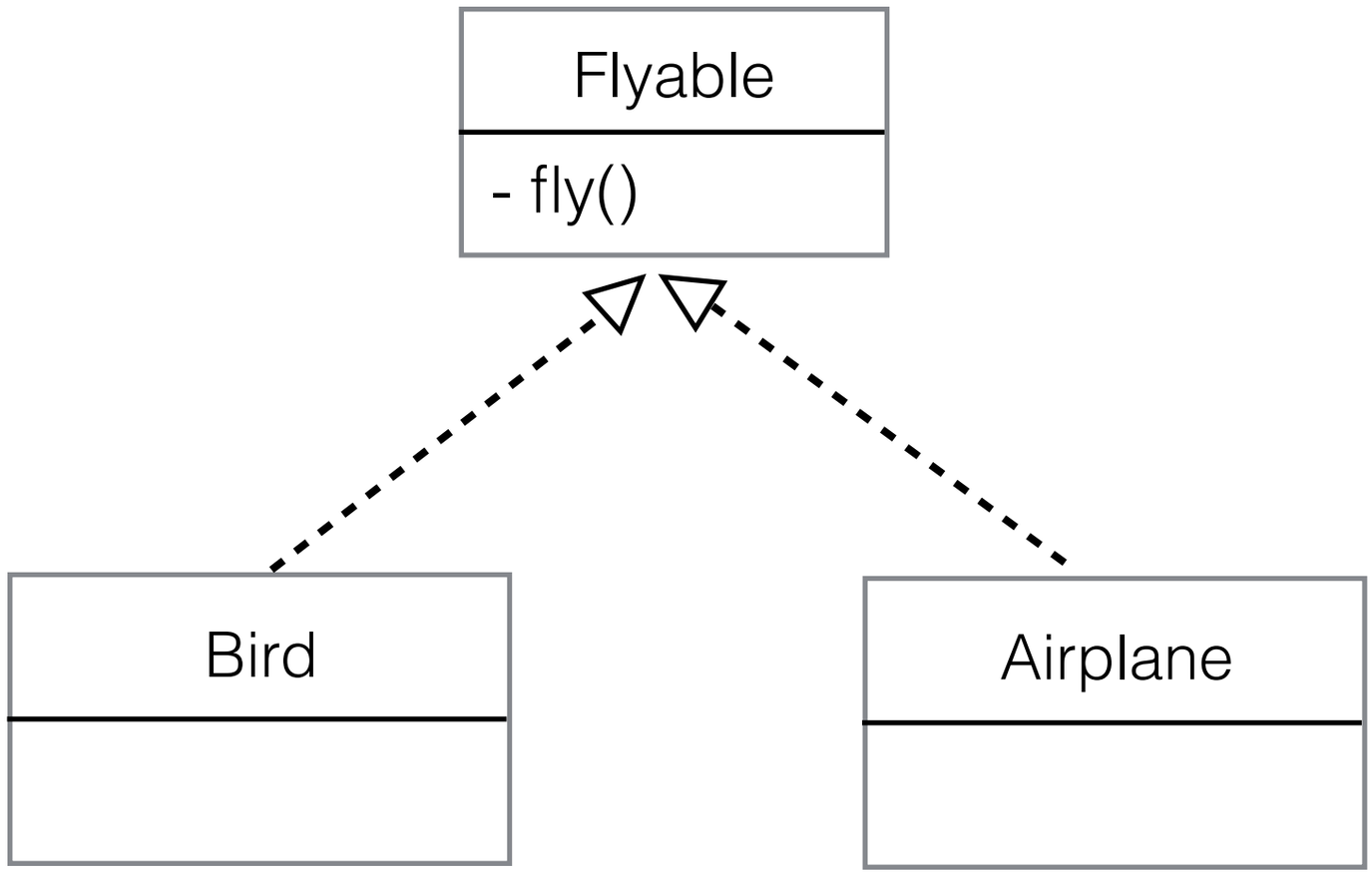
UML Class Diagram

- Generalization



UML Class Diagram

- Realization



From requirements to design

- What are requirements?
- What are types of requirements?
    - Functional
    - Usability
    - Reliability
    - Performance
    - Supportability

Requirements - Functional

- Features & capabilities
- Describe what the system can do
- Example:
    - Users can register/login to the system
    - Users can like their content

Requirements - Usability

- Human factors
- Understandability, Learnability, Operability, Attractiveness
- Example:
    - The text must be visible from 2 meters.
    - All error messages in the system describe how to fix it.

Requirements - Reliability

- Frequency of failure
- Recoverability
- Predictability
- Example:
    - The system can boot up after 1 minutes after failure.
    - The system will fail less than 3 hours a week.

Requirements - Performance

- Response times
- Throughput
- Accuracy
- Example
    - The system can handle up to 10k concurrent users
    - The new feed calculation is always done within 31.5 seconds

Requirements - Supportability

- Adaptability
- Maintainability
- Internationalization
- Example
    - The app supports Thai and English

From requirements to design

- What should be the main requirements for designing software?
- There are many
- For this class, let’s go with Use case for functional requirements

Use case

- Text stories
- Discover and record requirements
- 3 types, brief, casual, fully dressed

Example of use case (1)

- Dice game use case

1. Player roll two dice
2. The system displays results
3. The player win if the sum of two faces is 7. Otherwise, he lose.

Example of use case (2)

- POS - Process Sale :

    1. A customer arrives at a checkout with items to purchase.
    2. The cashier uses the POS system to record each purchased item.
    3. The system presents a running total and line-item details.
    4. The customer enters payment information
    5. The system validates and records.
    6. The system updates inventory.
    7. The customer receives a recipe from the system and then leaves with the item.

Use case

- How to handle branch scenario (alternate flows)
- Example (from the last page)
    - 3a. Invalid item id
        - 1. System signal error and reject entry
        - 2. Cashier handle error

Use case

- How to handle exception
- Example (from the last 2 page)
    - *a At anytime, System crashes
        - 1. Cashier restart the system, logs in, and recover the last state
        - 2. The system resume the last state

Exercise:

Write a use case for snake and ladder game.

From requirements to design

- Now, we have use cases, what next?

- We need some thing to translate use cases into design language.

Domain model

- A visual representation of conceptual classes or real-situation objects in a domain
- Conceptual models, Domain Object Models, Analysis Object Model
- Are not software objects or software classes
- [domain objects] [associations] [attributes]

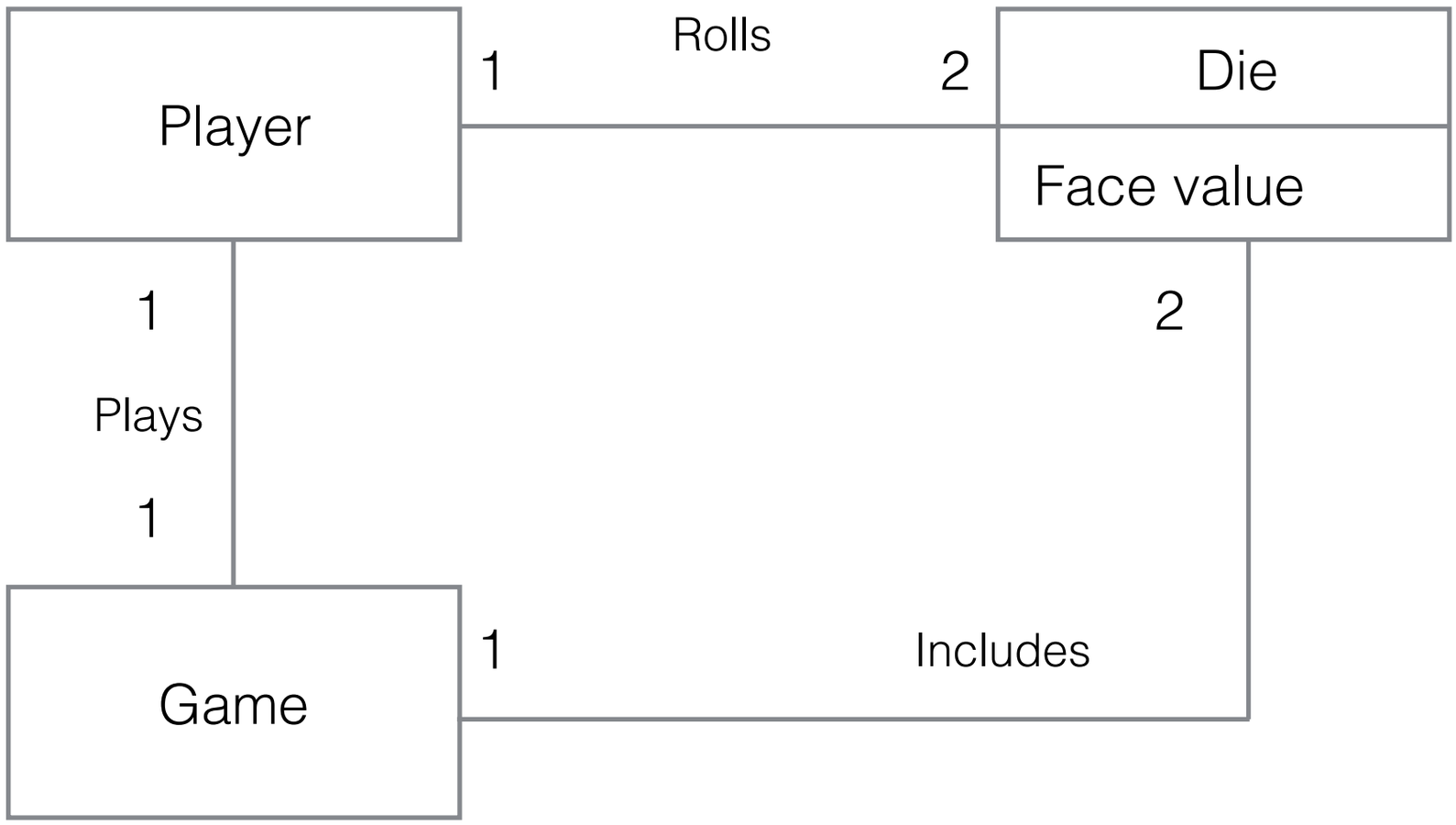
Example use case

- Dice game

Dice Game

- Dice game use case
  - Player roll two dice
  - The system displays results
  - The player win if the sum of two faces is 7. Otherwise, he lose.

Domain model



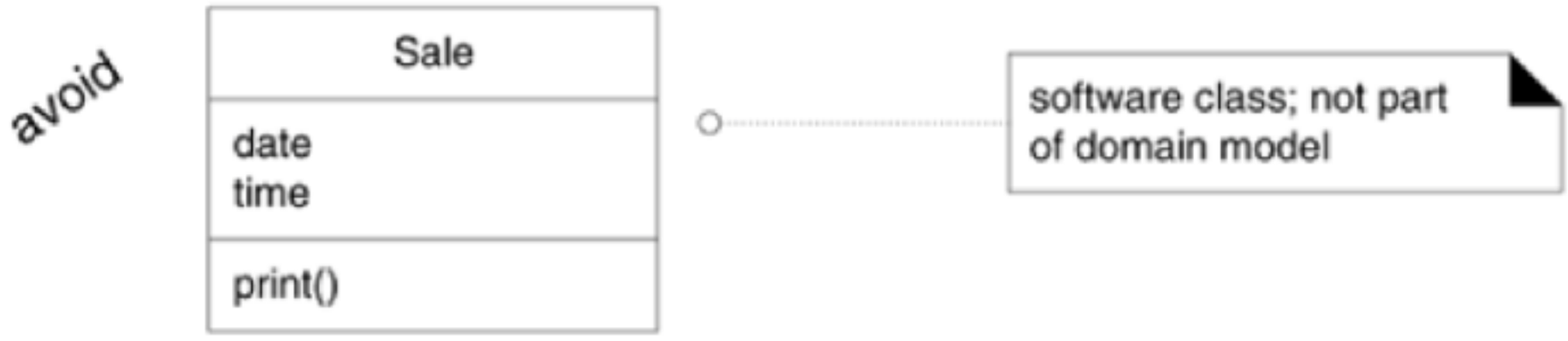
Why create a domain model?

- To understand key concepts of the business
- Get the big picture without worrying about the software details
- Domain model acts as inspirational to create software classes

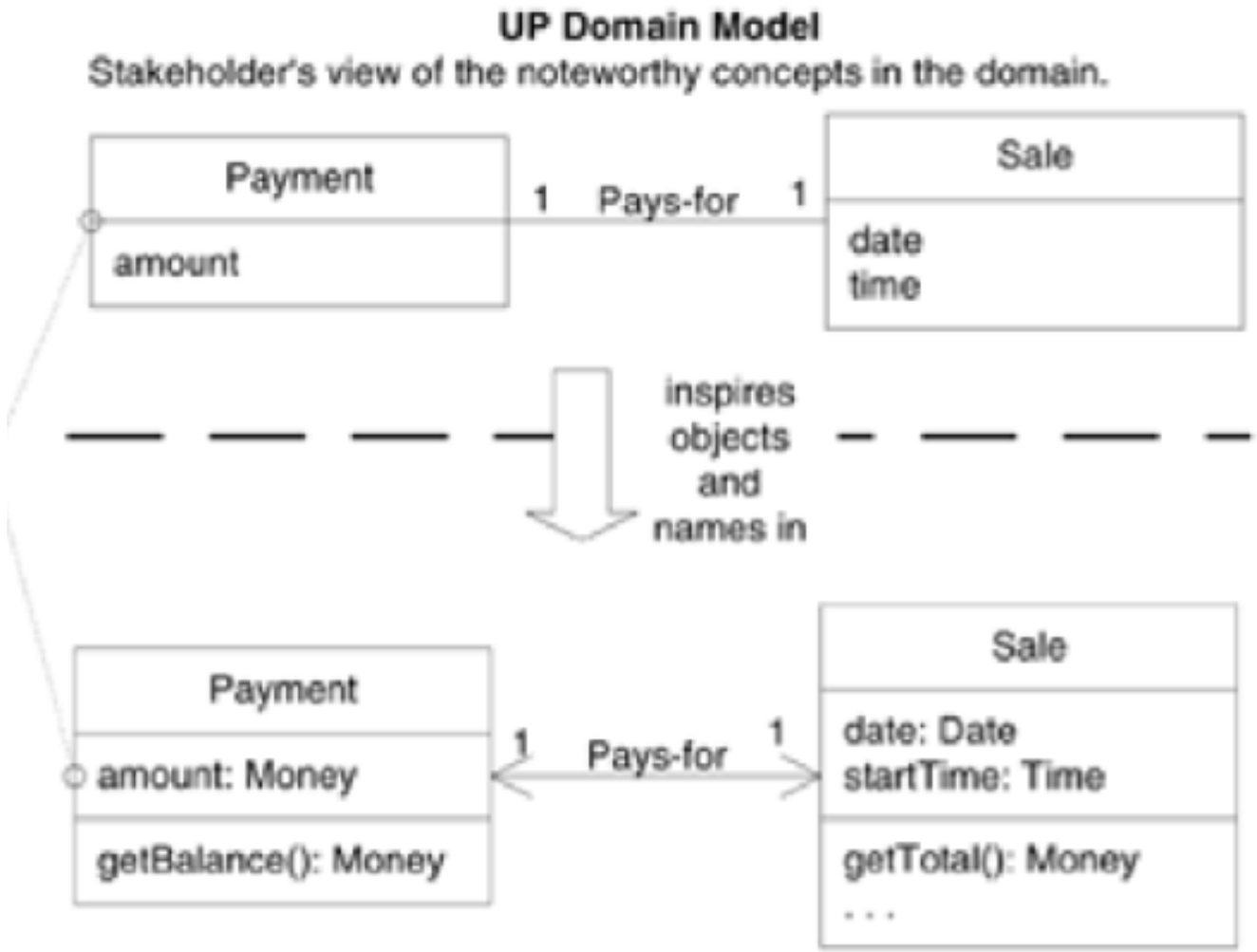
Domain model - Things to avoid

- Describe software artifacts like Window, Database
- Specify method to a model
- See examples

Domain model - Things to avoid



Why create a domain model?



Domain model - From use case

Preconditions: Cashier is identified and authenticated

Postconditions: Sale is saved. Tax is correctly calculated. Account and inventory updated. Commission recorded. Receipt is generated. Payment authorization approvals are record

Main Success Scenario:

1. Customer arrives at POS with items
2. Cashier starts a new sale
3. Cashier enters item id
4. System records sale line item and present item description, price, total

- - - Cashier repeat steps 3-4 until done

5. System presents total with taxes calculated
6. Cashier tells customer the total, and asks for payment
7. Customer pays and system handles payment
8. System logs completed sale and sends sale and payment information

to the external Accounting and inventory system

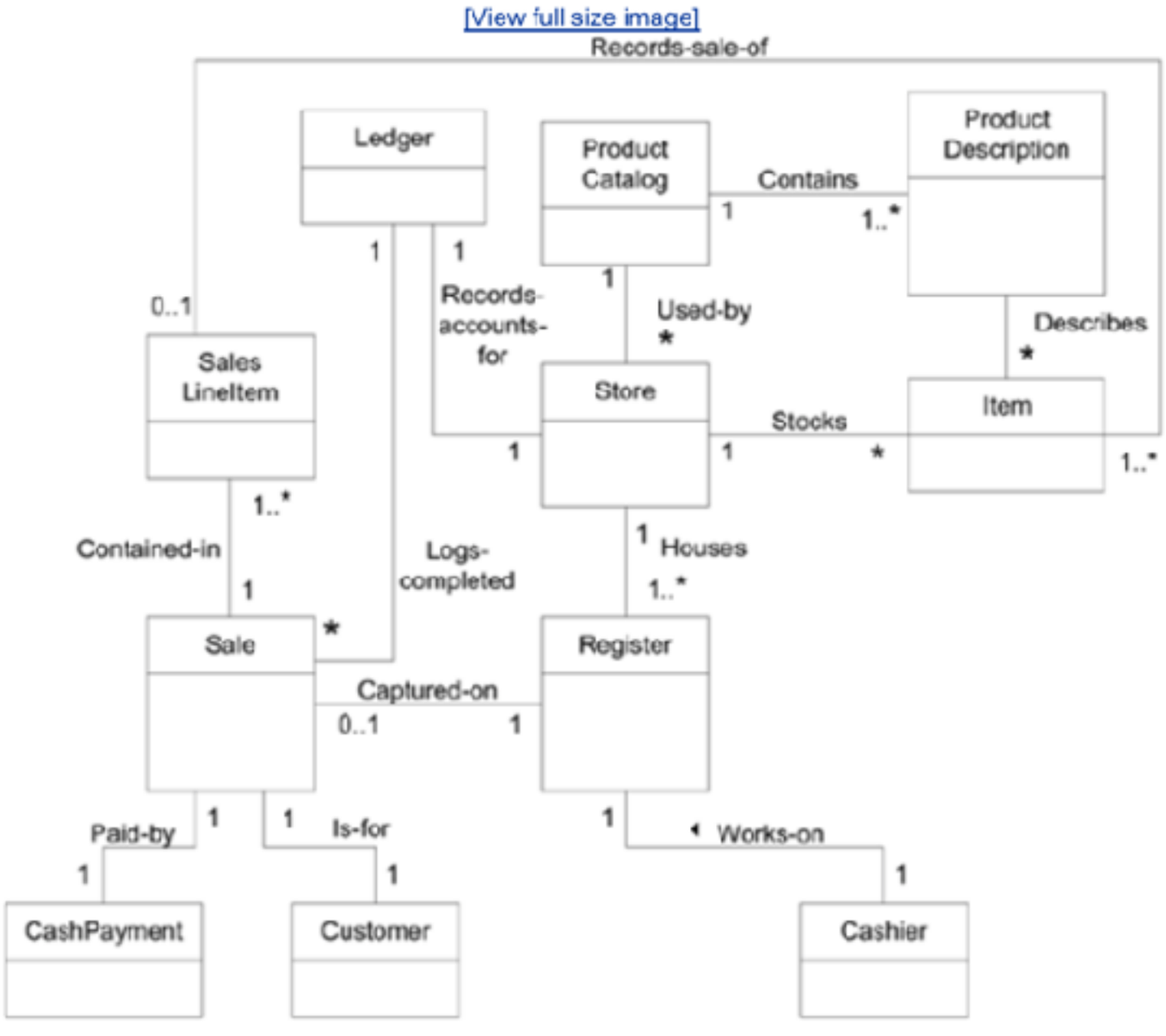
9. System presents receipt
10. Customer leaves with items

Domain model - candidates

Register	Item	Store	Sale
Sales LineItem	Cashier	Customer	Ledger
Cash Payment	Product Catalog	Product Description	

Domain model - POS

[View full size image]



Domain model - Attributes vs classes

- If that thing is raw number or text in the real world it might be an attribute

- In the previous model, What is store?

Association

- When to show association?

- Will the association be implemented in software?

Association

- How should we name association?
- Has and Use are not very good.
- Sale ‘Use’ CashPayment => Bad
- Sale ‘Paid-by’ CashPayment => Better

Association

- Multiplicity, see examples

- Multiple associations are also possible

Attributes

- When to show attributes?

- No foreign keys

GRASP Principle

GRASP

- GRASP consists of
    - **Controller**
    - **Creator**
    - **Information Experts**
    - **High Cohesions**
    - Low Coupling
    - Polymorphism
    - Protected Variations
    - Pure Fabrication
    - Indirection

GRASP: Controller

- Name: Controller
- Problem: What first object beyond the UI layer receives and coordinates a system operation?
- Solution: Assign the responsibility to:
    - The object that represents the overall system
    - The object that represents that particular use case

Controller - Snake and Ladder

- What class should act as a controller?

GRASP: Creator

- Name: Creator
- Problem: Who creates an A?
- Solution: Assign class B the responsibility to create an instance of class A if one of these is true (the more the better)
    - B contains A
    - B records A
    - B closely uses A
    - B has the initialising data for A

Creator - Snake and Ladder

- Who creates a Board object?
- Who creates a Die object?
- Who creates a Piece object?
- Who creates a Player object?
- Who creates a Square object?
- Who creates a Game object?

Creator - Drawing Board

- Who creates all the objects on the board?

- Who creates a drawing board object?

GRASP: Information Expert

- Name: Information Expert
- Problem: What is a basic principle by which to assign responsibilities
- Solution: Assign a responsibility to the class that has the information needed to fulfil it

Information Expert - Snake and Ladder

- Which object should handle piece movement
- Which object knows the piece position
- Which object knows if the piece is at goal square

Information Expert - Drawing Board

- Who should handle mouse clicks?
- Who should maintain the list of all objects?

GRASP: High Cohesion

- Name: High Cohesion
- Problem: How to keep objects focused, understandable, and managable?
- Solution: Assign the responsibility so that cohesion remains high. Use this to evaluate alternatives.

High Cohesion - Snake & Ladder

- What if we don’t have Dice class?

- What if we don’t have Piece class?

High Cohesion - Drawing Board

- What if everything is rendered in Window?

UML Interaction Diagrams

What are interaction diagrams?

- Diagrams we use to show how objects interact via messages.

- Dynamic object modeling

- Two types:

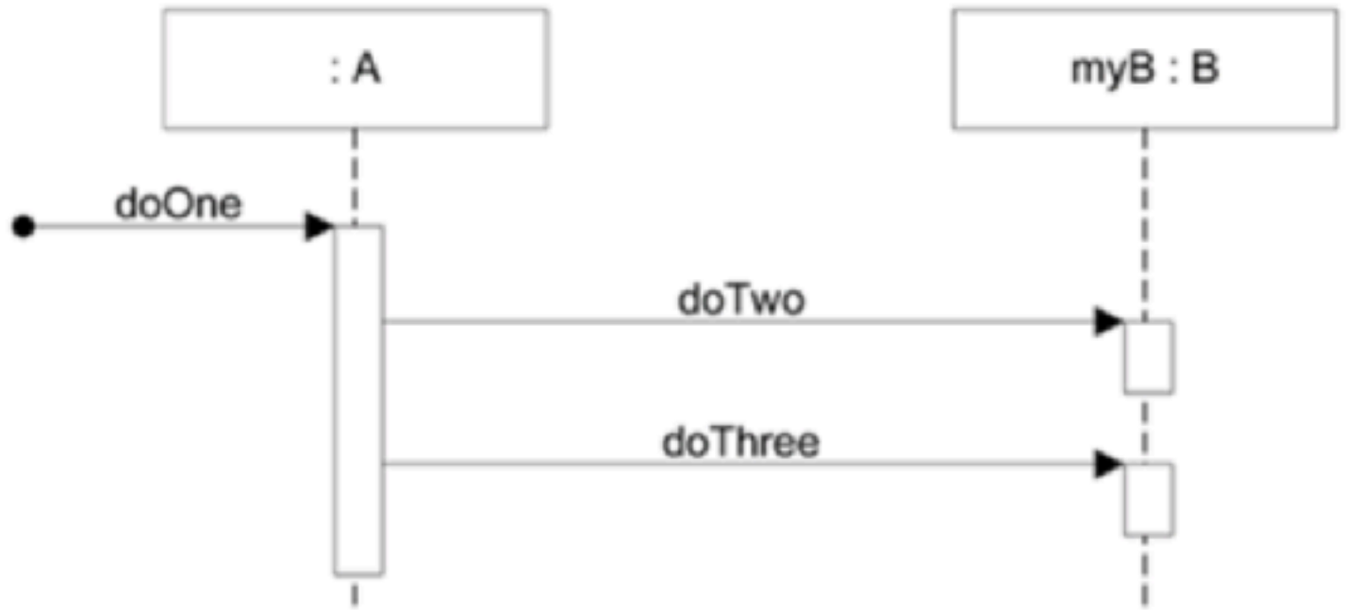
    - Sequence diagrams

    - Communication diagrams

- Today we will start with Sequence diagrams

Sequence Diagrams

Figure 15.1. Sequence diagram.

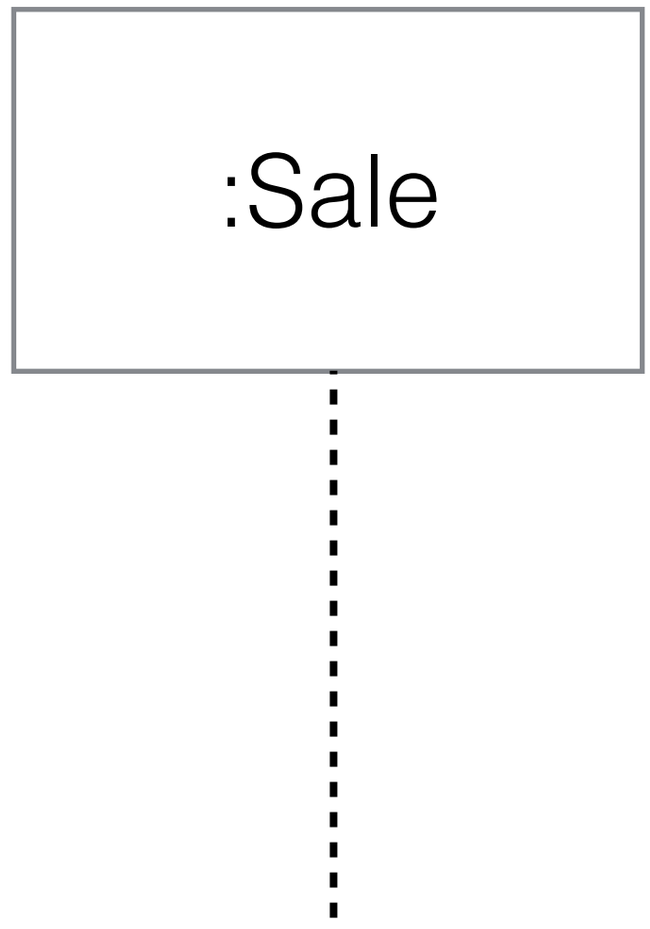


How can we turn this into code?

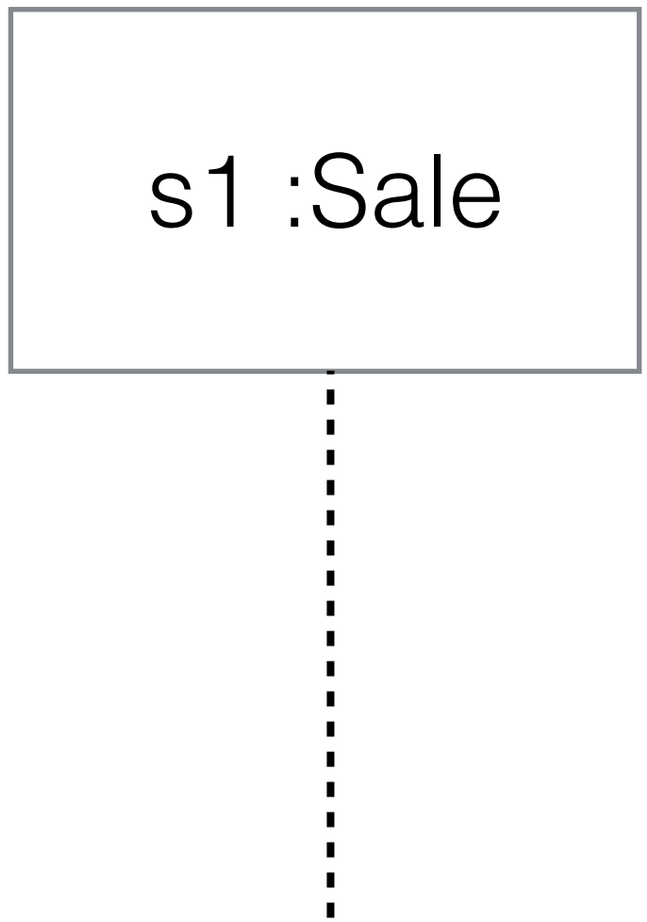
Why interaction diagram is important?

WHY?

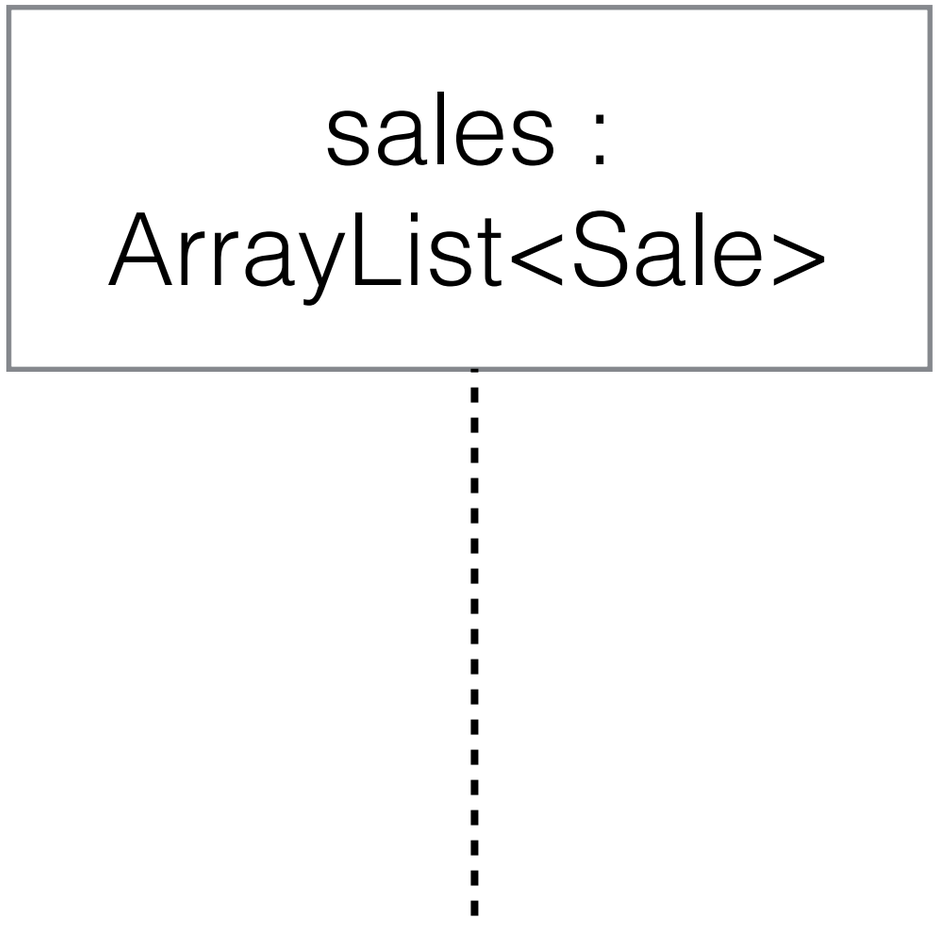
Notations - Life line box



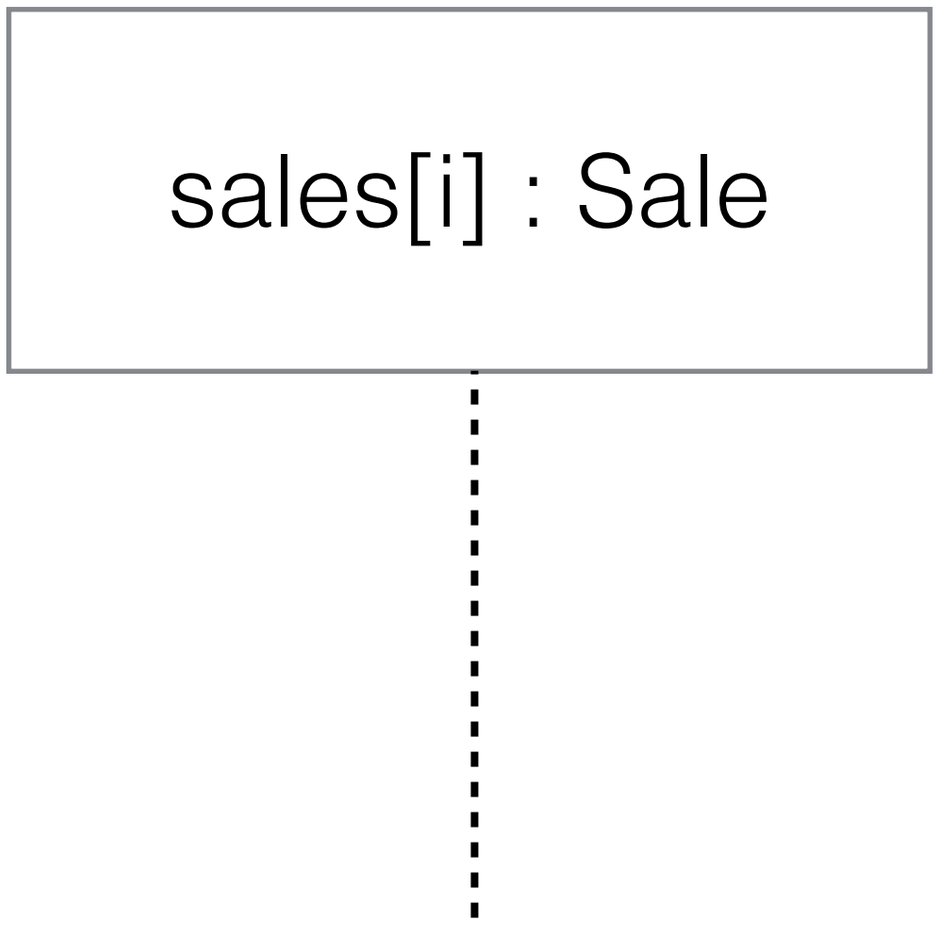
Notations - Life line box



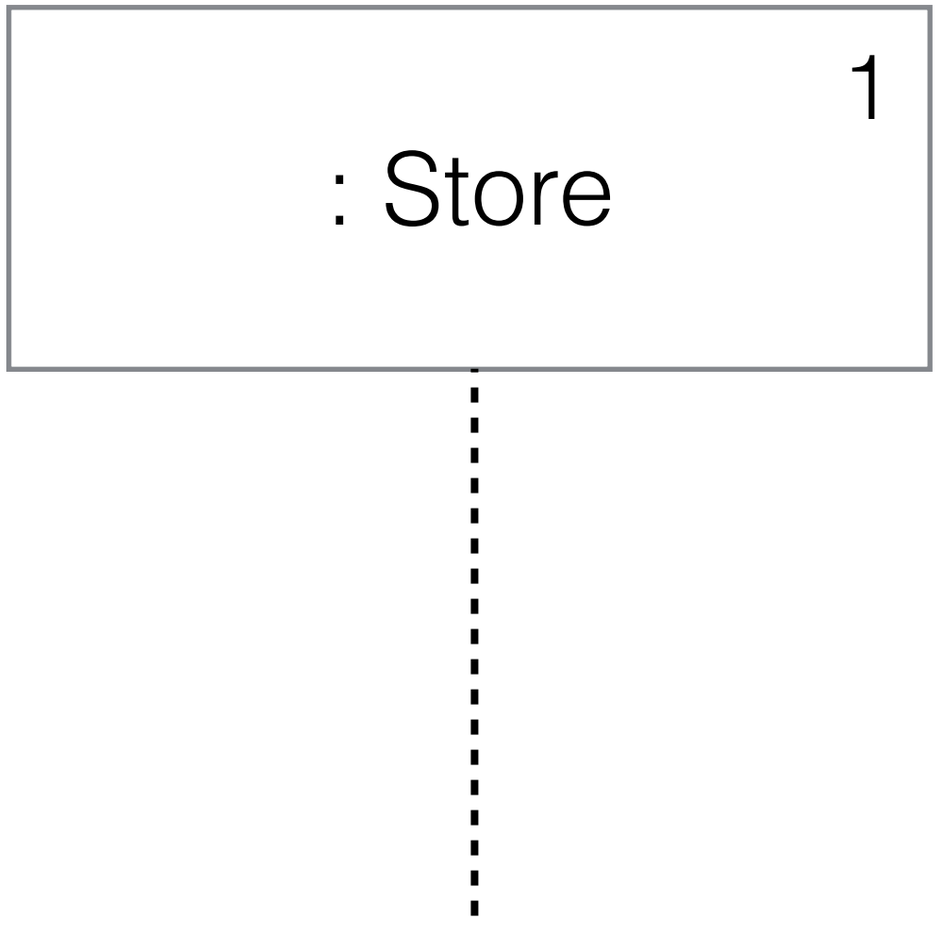
Notations - Life line box



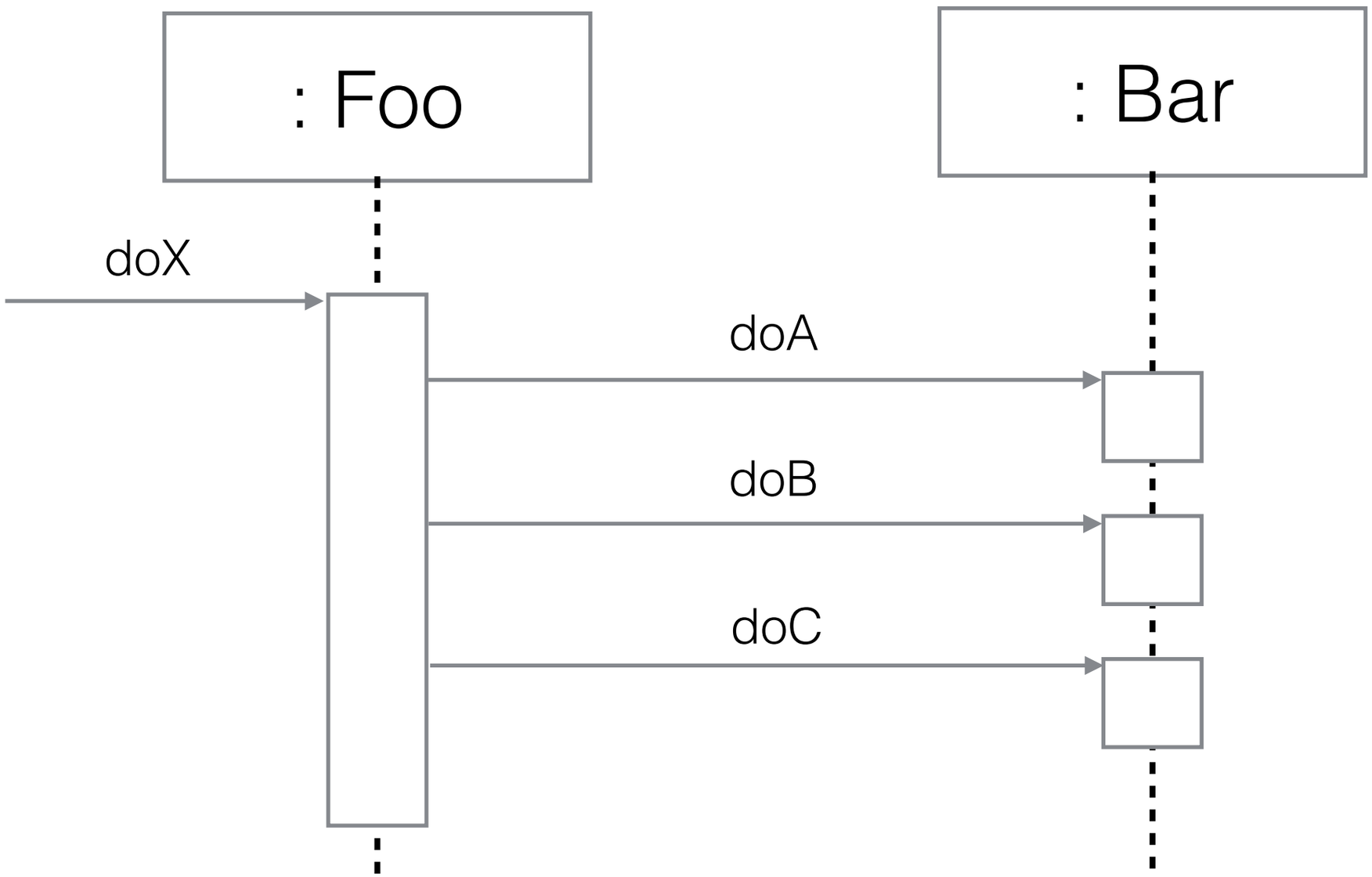
Notations - Life line box



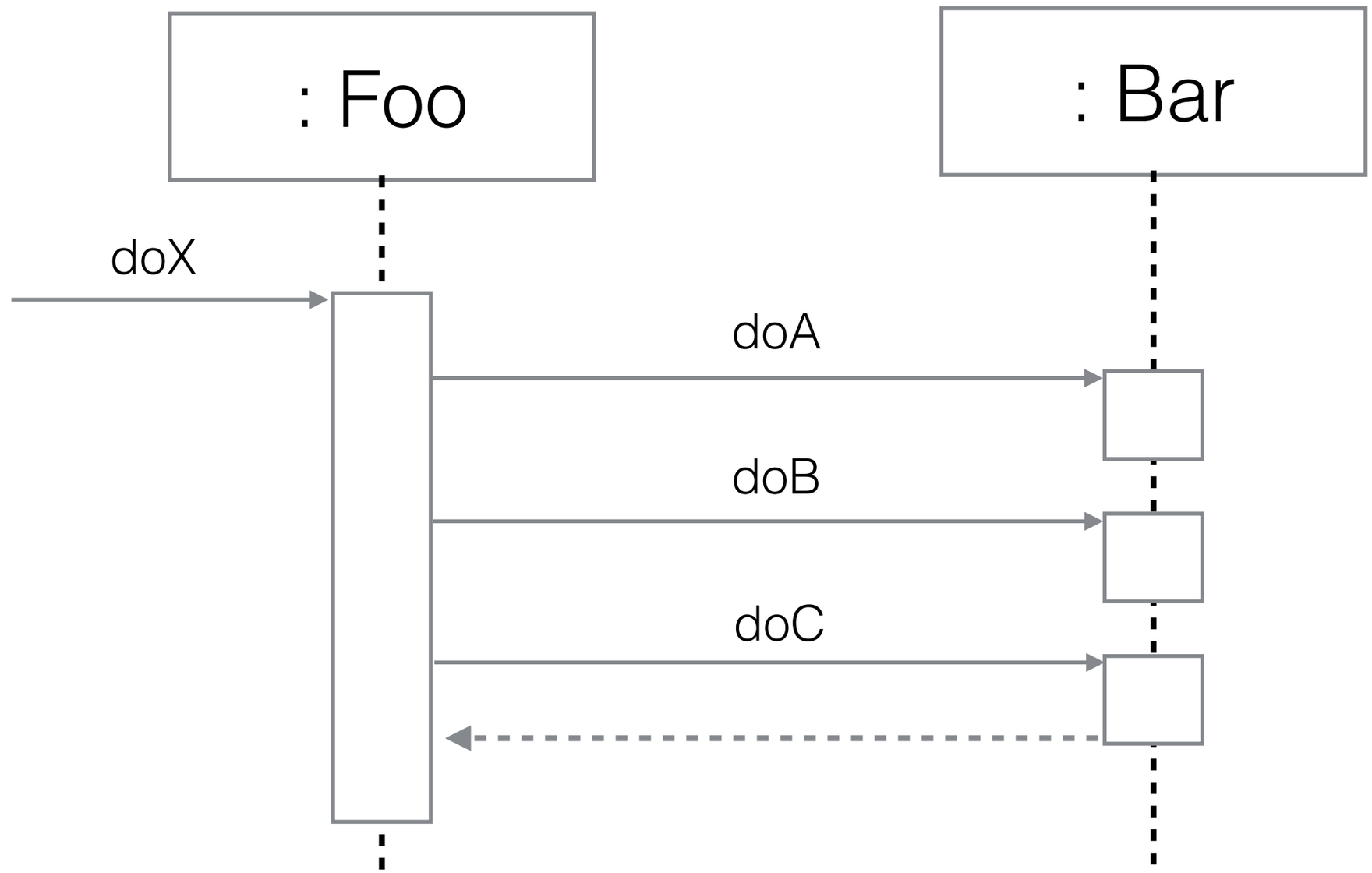
Notations - Singleton



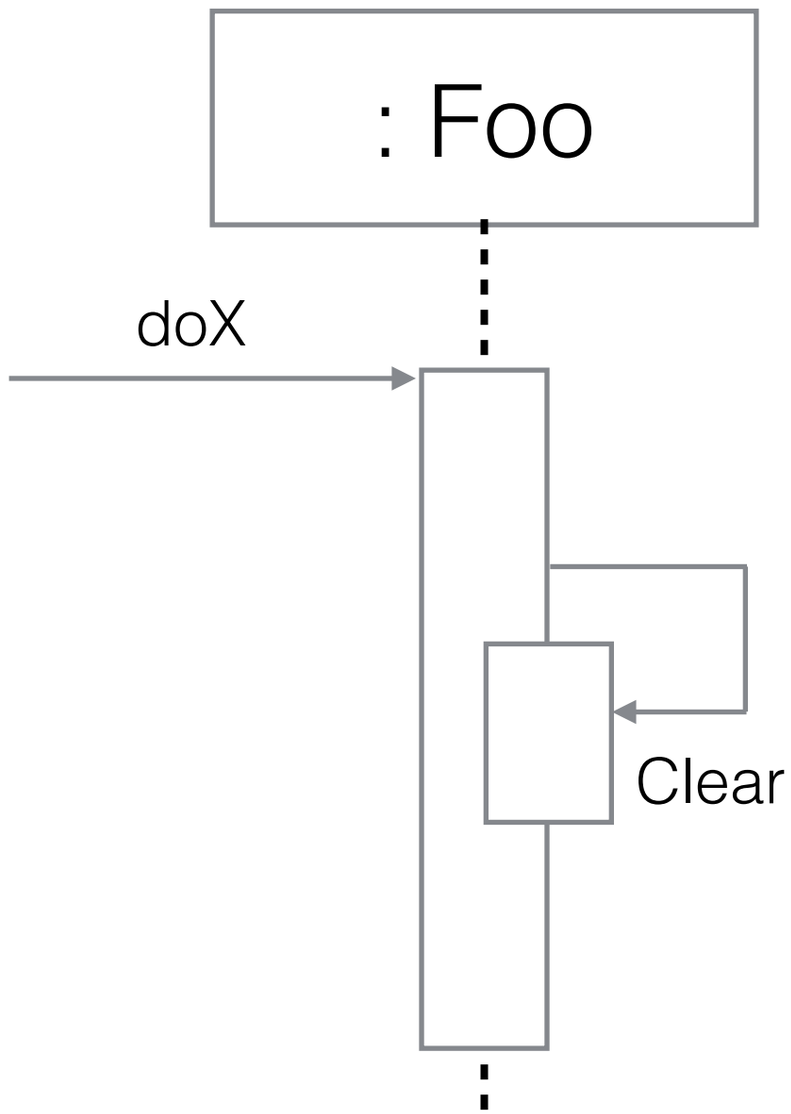
Notations - Message



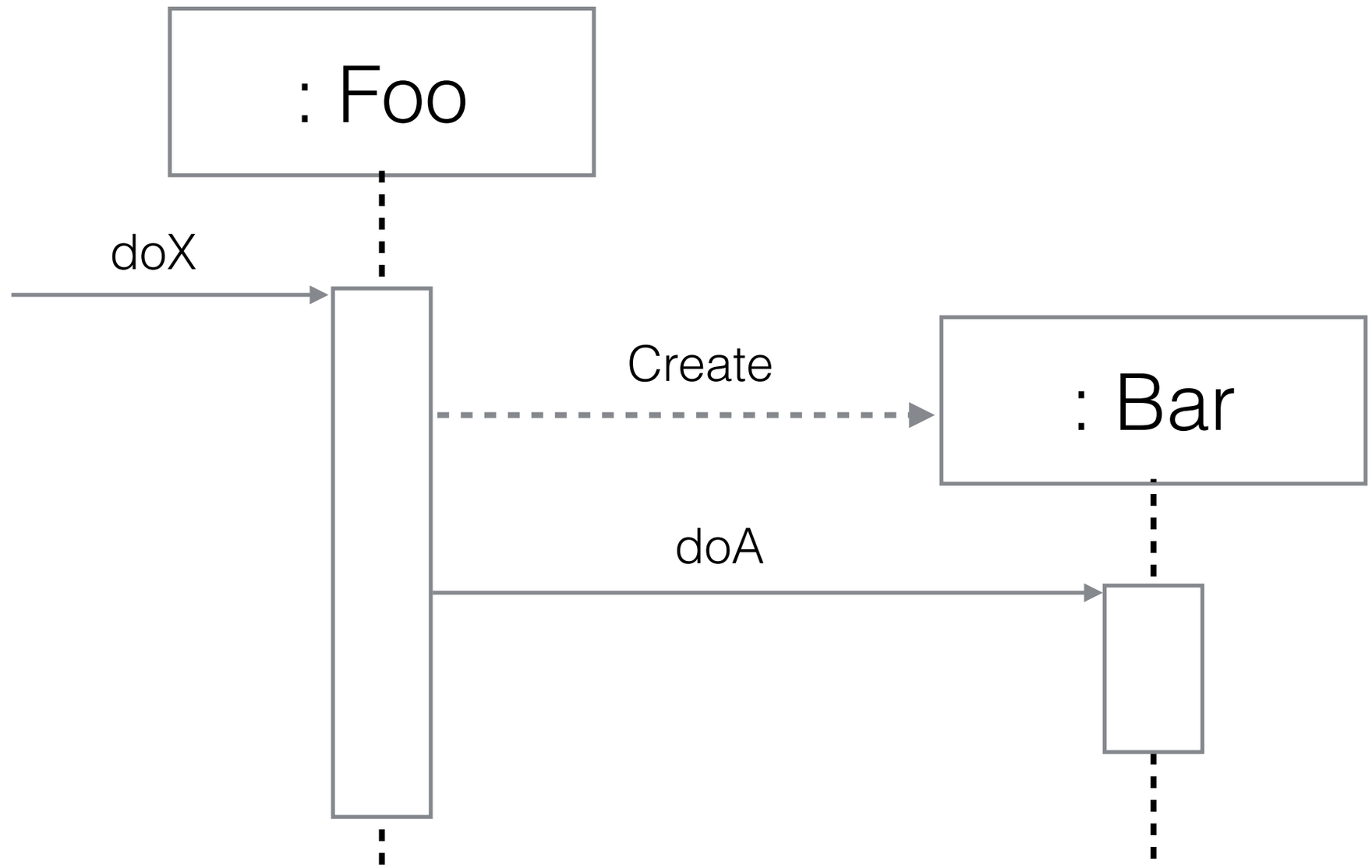
Notations - Returns



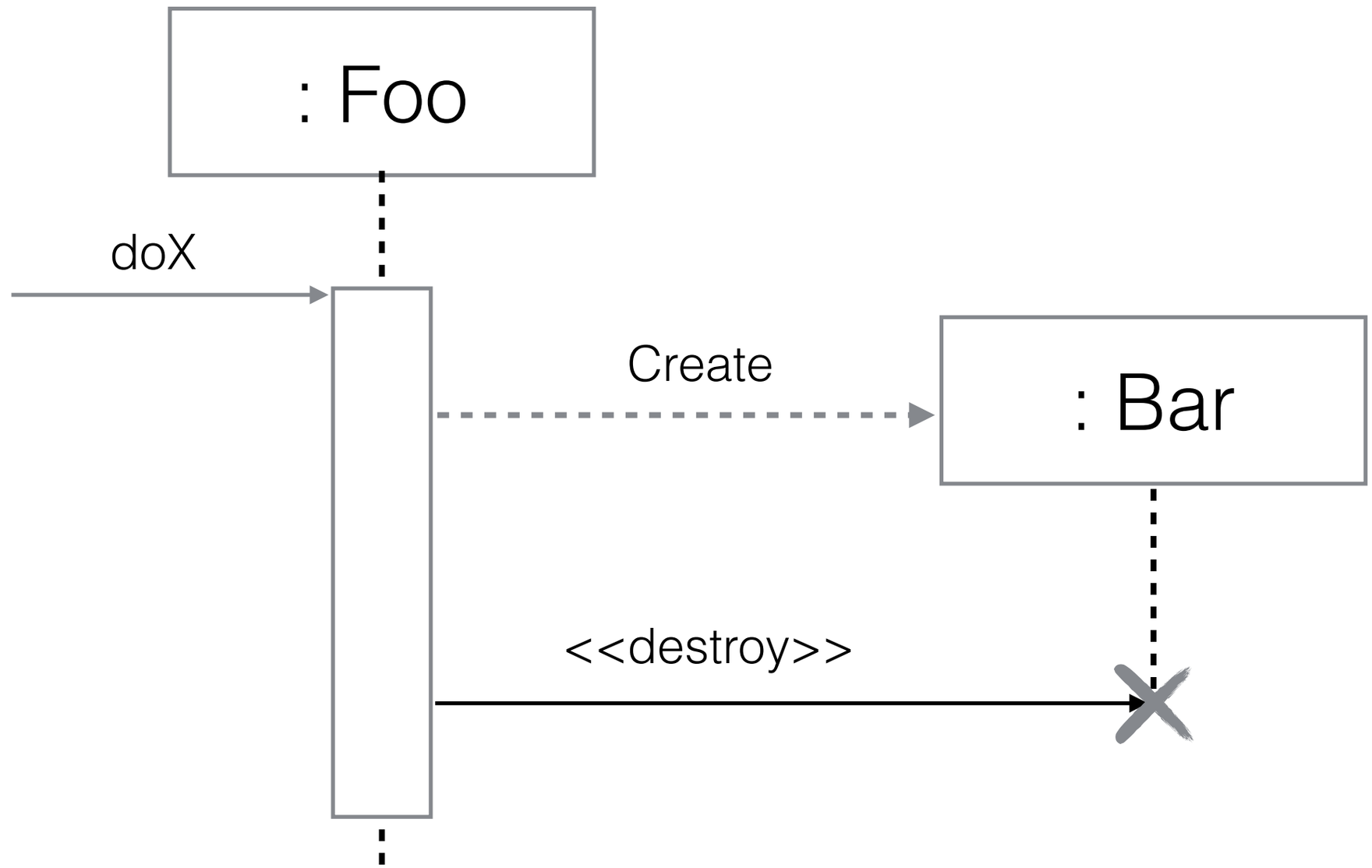
Notations - self/this



Notations - Creation

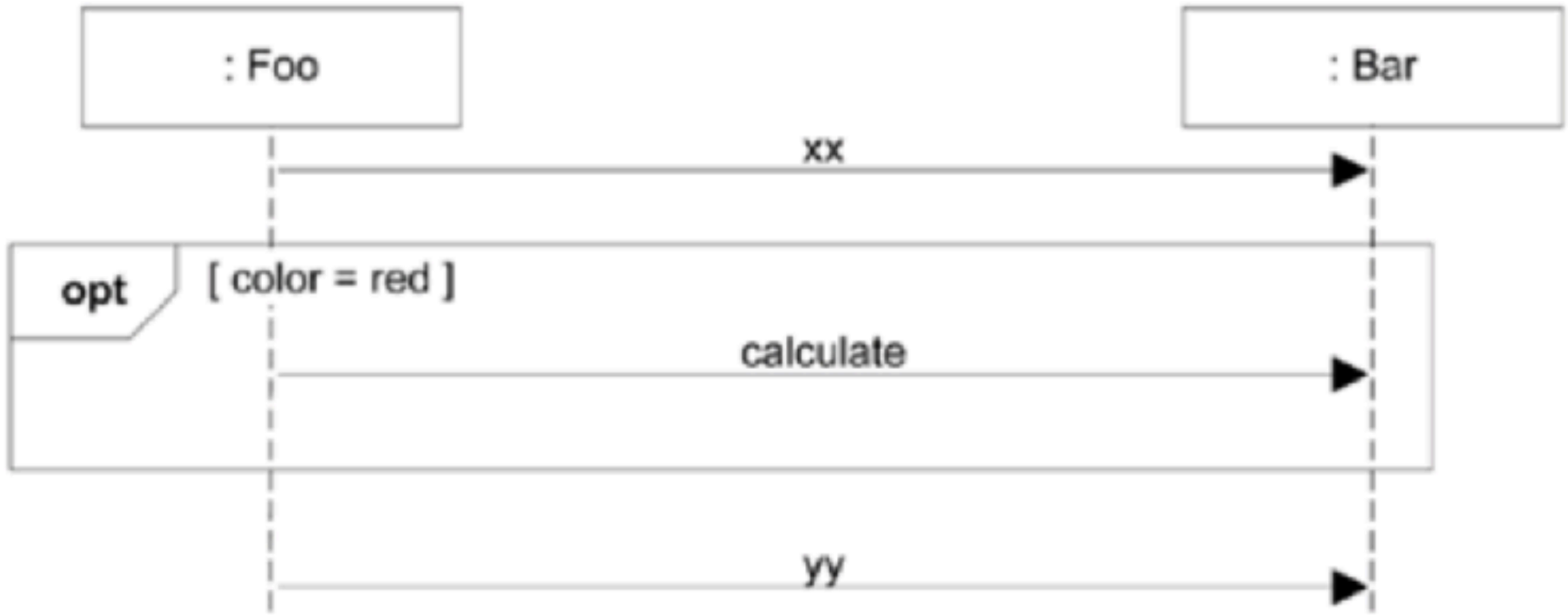


Notations - Termination

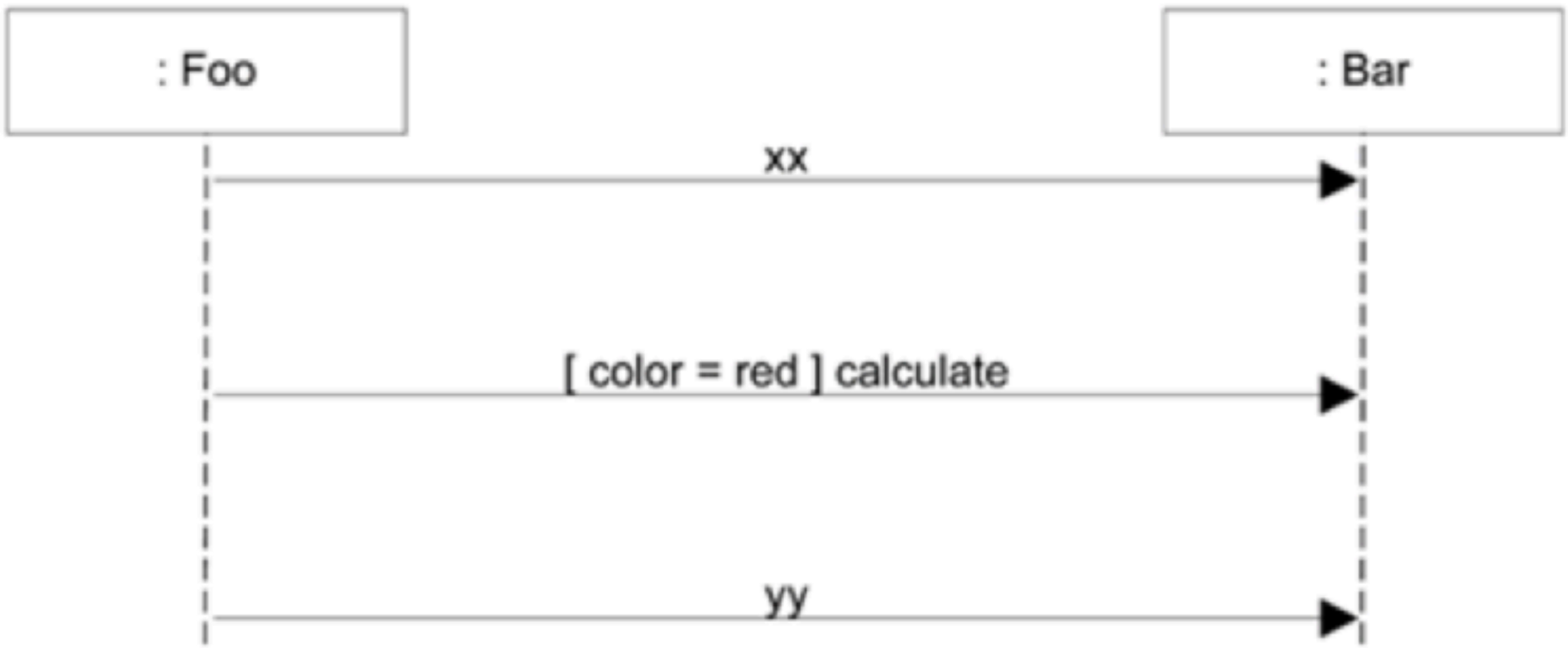


Notations - Loop

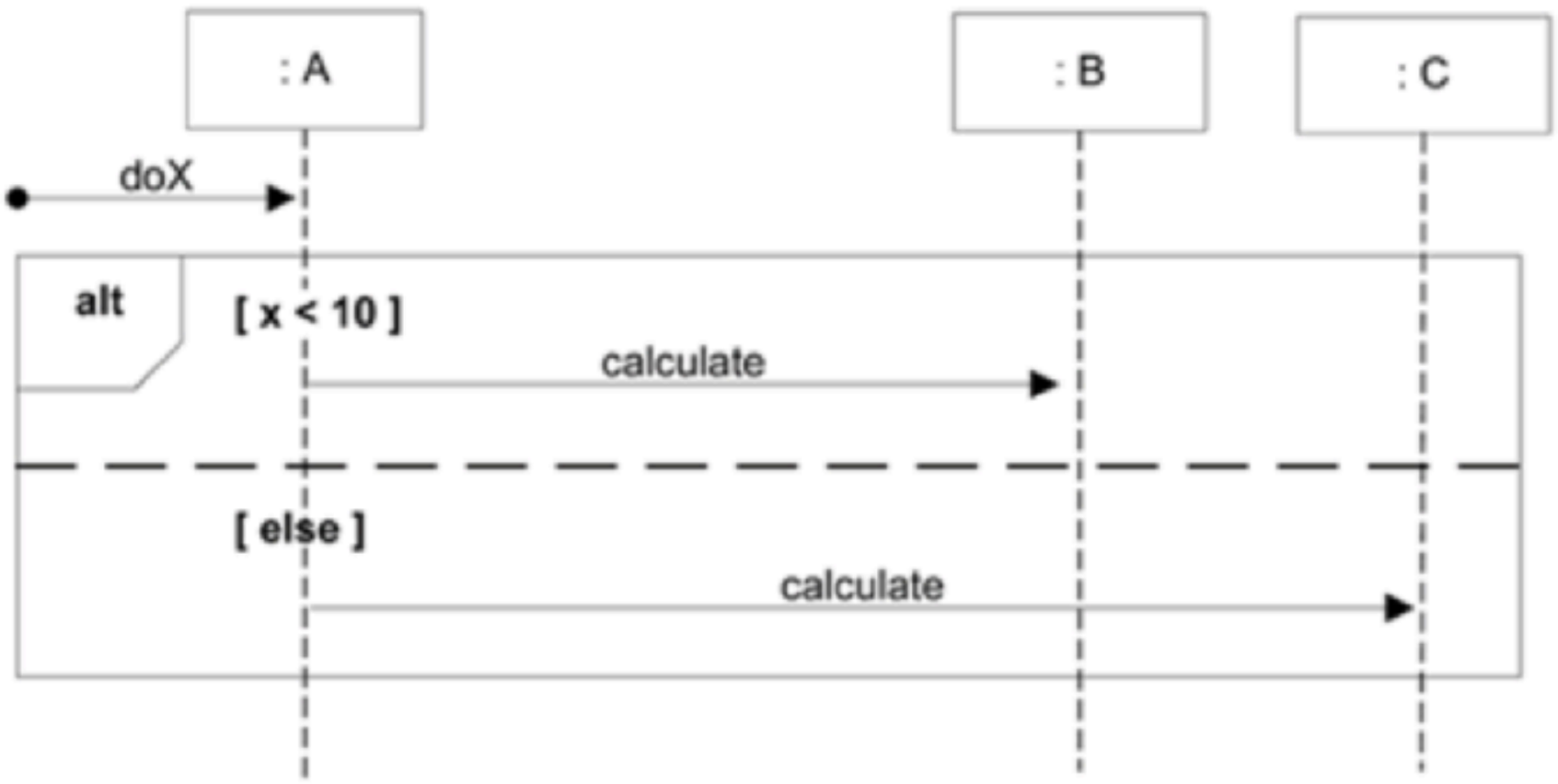
Notations - Conditional



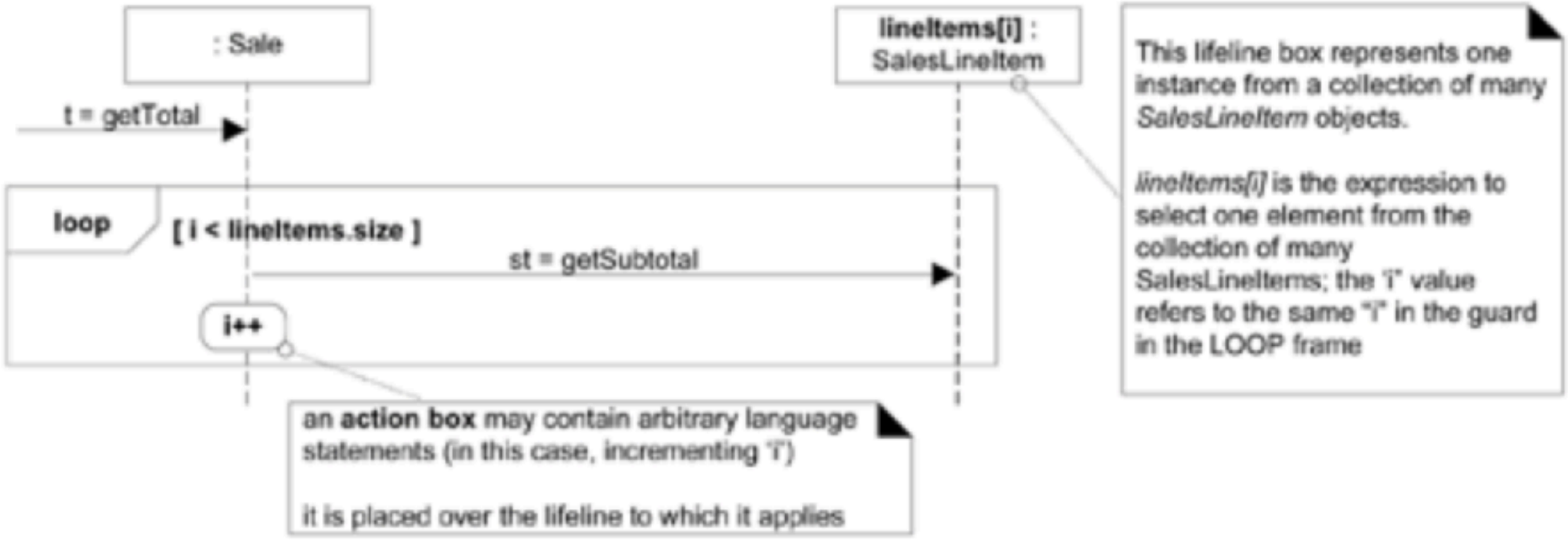
Notations - Conditional (2)



Notations - Conditional (3)

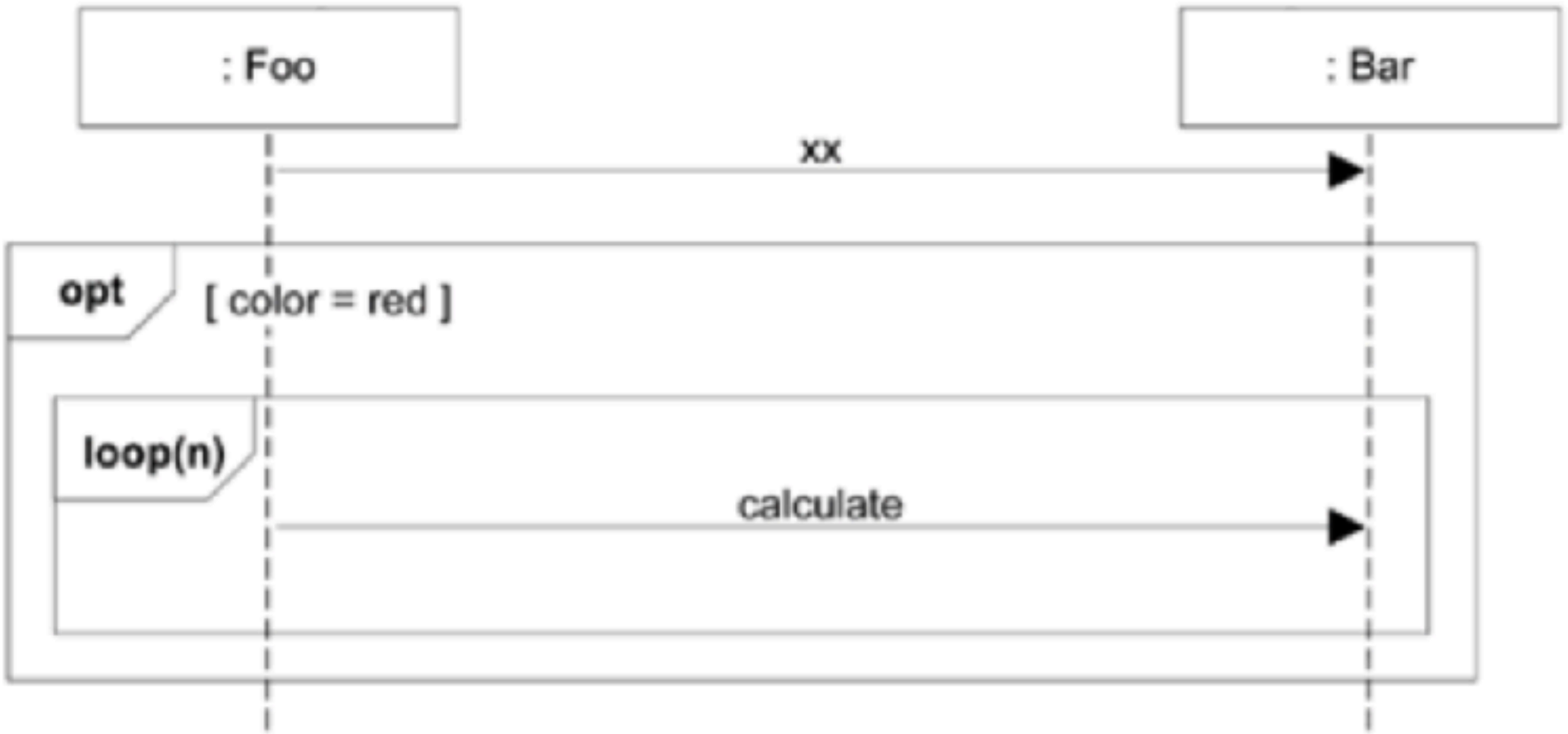


Notations - Iteration

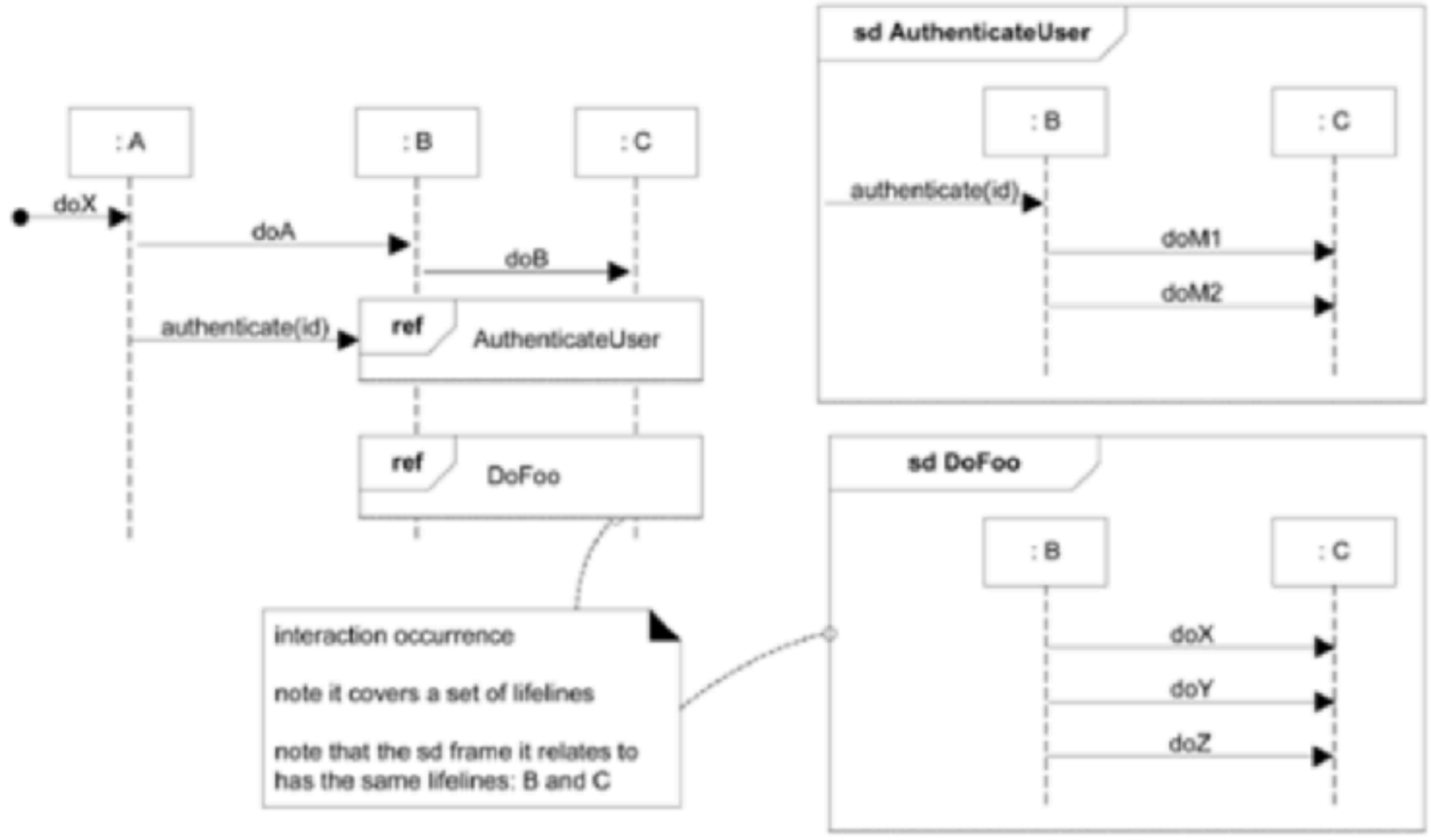


Notations - Iteration (2)

Notations - Nesting



Notations - Reference



Notations - Static method