Interaction Diagram Notation

Software Design and Analysis

CSCI 2040

Objectives

- Read basic UML interaction
  - Sequence diagram notation
  - Collaboration diagram notation

Introduction

- The UML includes **interaction diagrams** to illustrate how **objects** interact via messages.
- The term *interaction diagram,* is a generalization of two more specialized UML diagram types;
  - collaboration diagrams
  - sequence diagrams

Usage

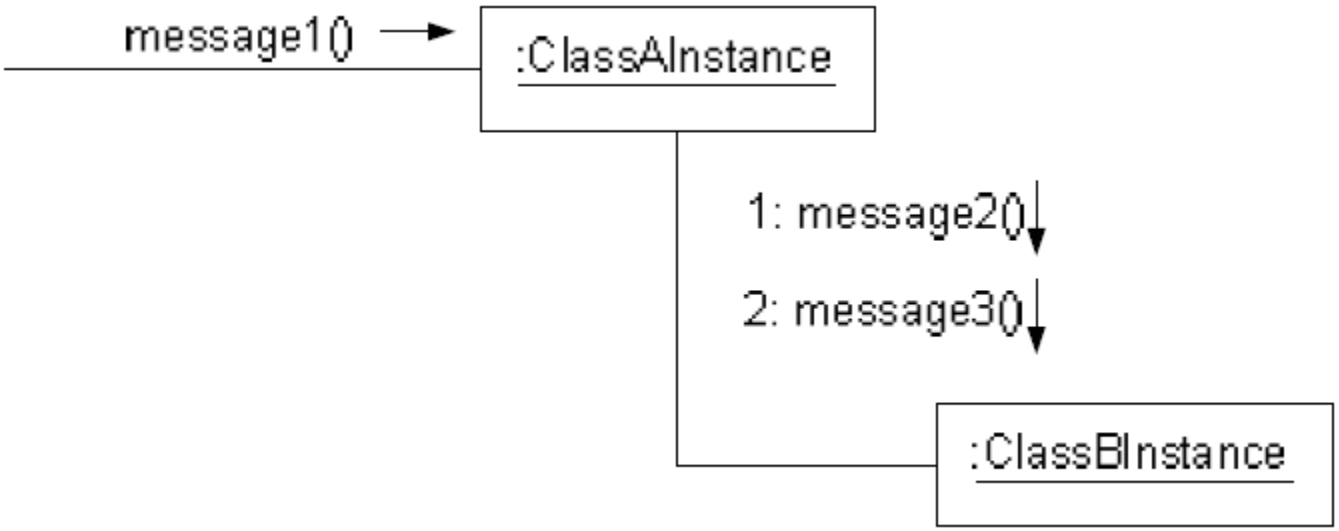
- Both can be used to express similar message interactions.
- Each type has strengths and weaknesses.
- Interaction Diagrams represent dynamic view.
- Most Interaction Diagrams are created during Elaboration phase.
  - They are not usually motivated in inception.

Motivation: Why Draw an SSD

- An interesting and useful question in software design is this: *What events are coming in to our system*? Why?
  - Because we have to design the software to handle these events (from the mouse, keyboard, another system, …) and execute a response.
- Software system reacts to three things:
  - 1) external events from actors (humans or computers), 2) timer events, and
  - 3) faults or exceptions (which are often from external sources).

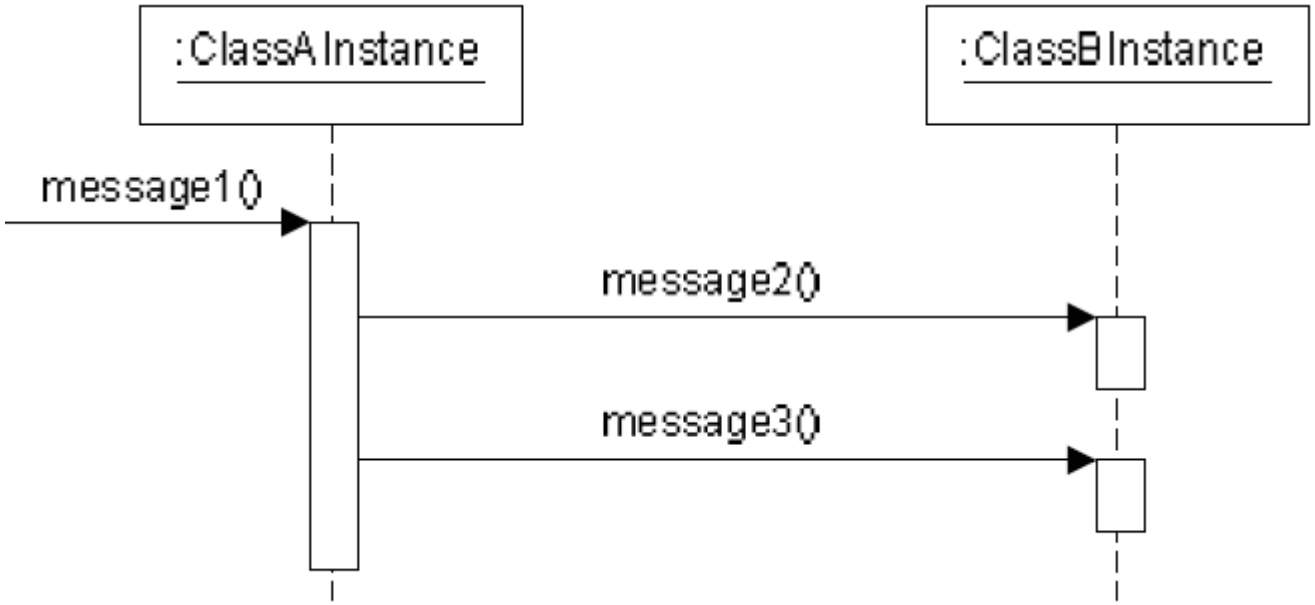
Collaboration Diagram

- **Collaboration diagrams** illustrate object interactions in a graph, in which objects can be placed anywhere on the diagram.
  - They are space efficient.



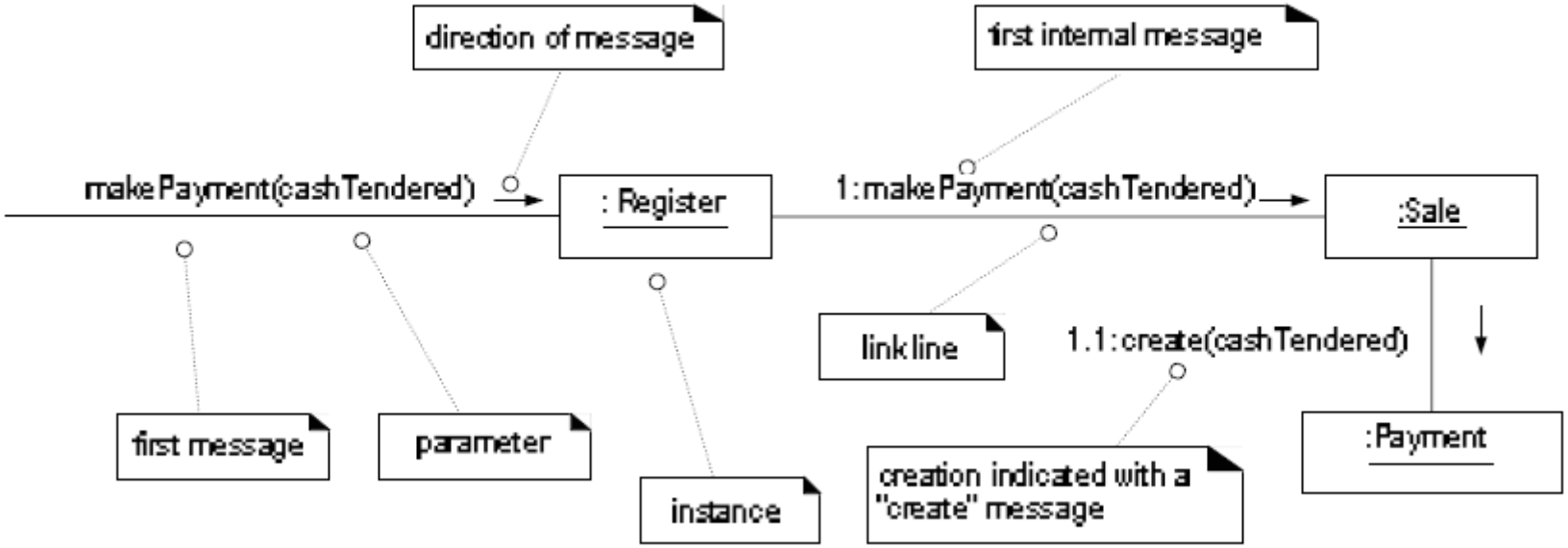
Sequence Diagram

- Most prefer **sequence diagrams** as they clearly illustrate the sequence of messages.



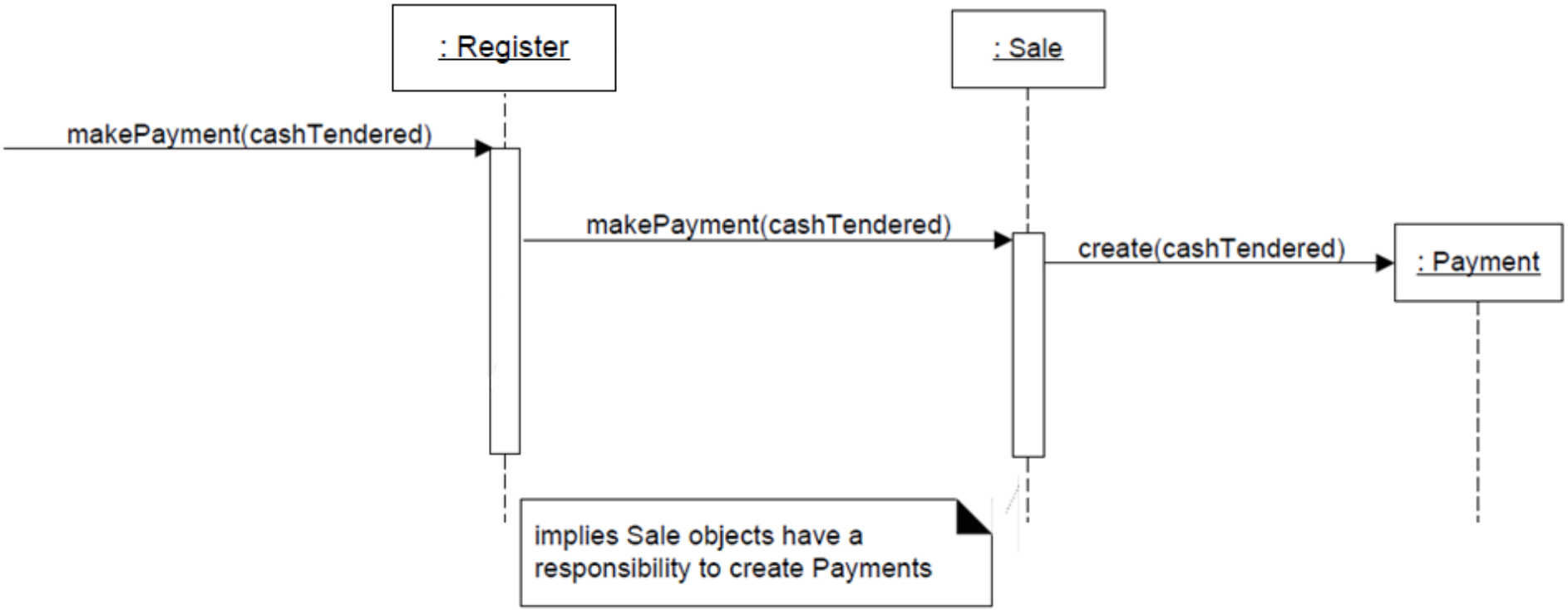
Make Payment Collaboration Diagram

1. The message *makePayment is* sent to an instance of a *Register.*
2. The *Register* instance sends the *makePayment* message to a *Sale* instance.
3. The *Sale* instance creates an instance of a *Payment.*



Make Payment Sequence Diagram

- The sequence diagram has the same intent as the prior collaboration diagram.



Related Code

- what might be some related code for the *Sale* class and its *makePayment* method?

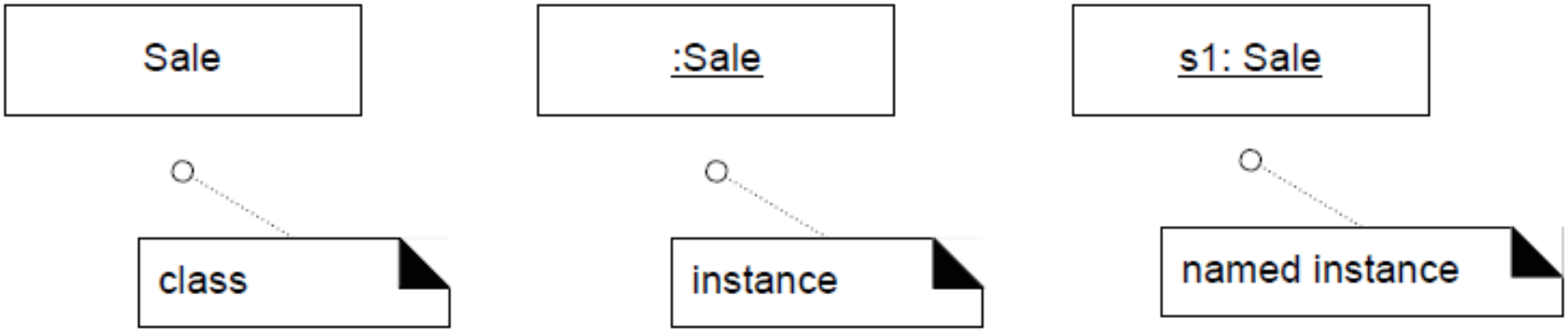
```
public class Sale
{
private Payment payment;

public void makePayment( Money cashTendered )
{
    payment = new Payment( cashTendered );
    //…
}
// …
}
```

Interaction Diagrams are Valuable

- Common starting point for inspiration during programming.
- Create interaction diagrams in pairs, not alone.
  - The collaborative design will be improved, and the partners will learn quickly from each other.
- Patterns can be applied to improve the quality of the design.

Illustrating Classes and Instances

- The UML has adopted a simple and consistent approach to illustrate **instances vs. classes**.
- An instance uses the same graphic symbol as the type, but the designator string is underlined.



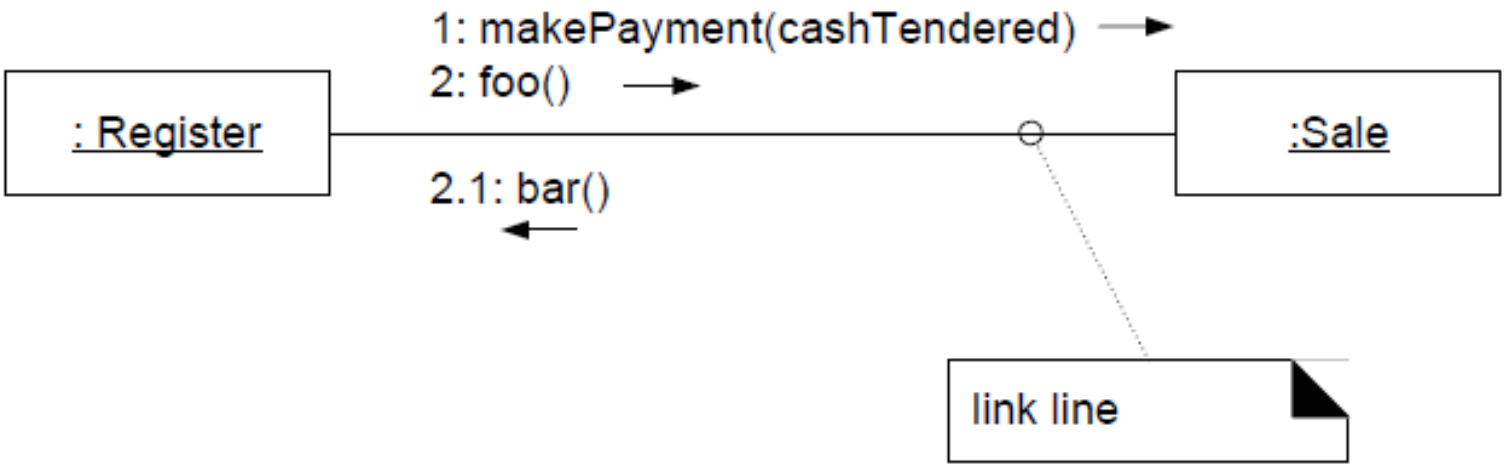
Basic Message Expression Syntax

- The UML has a standard syntax for message expressions:
  - return := message(parameter : parameterType) : returnType
- Type information may be excluded if obvious or unimportant. For example:
  - spec := getProductSpect(id)
  - spec := getProductSpect(id:ItemID)
  - spec := getProductSpect(id:ItemID) ProductSpecification

Collaboration Diagram Notation

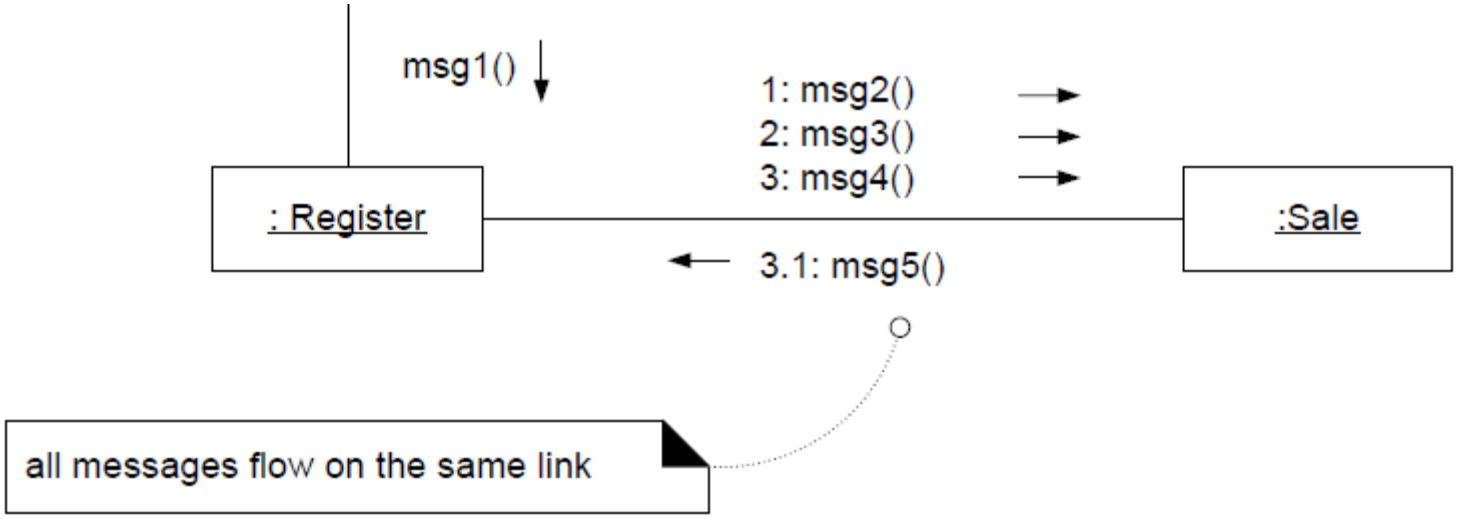
Basic Collaboration Notation

- A **link** is a connection path between two objects.
  - Link is an instance of an association.



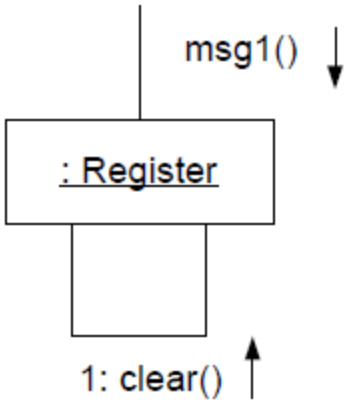
Messages

- Each message between objects is represented with a message expression and small arrow indicating the direction of the message.
  - Note that multiple messages can flow.
  - A sequence number is added to show the sequential order of messages in the current thread of control.

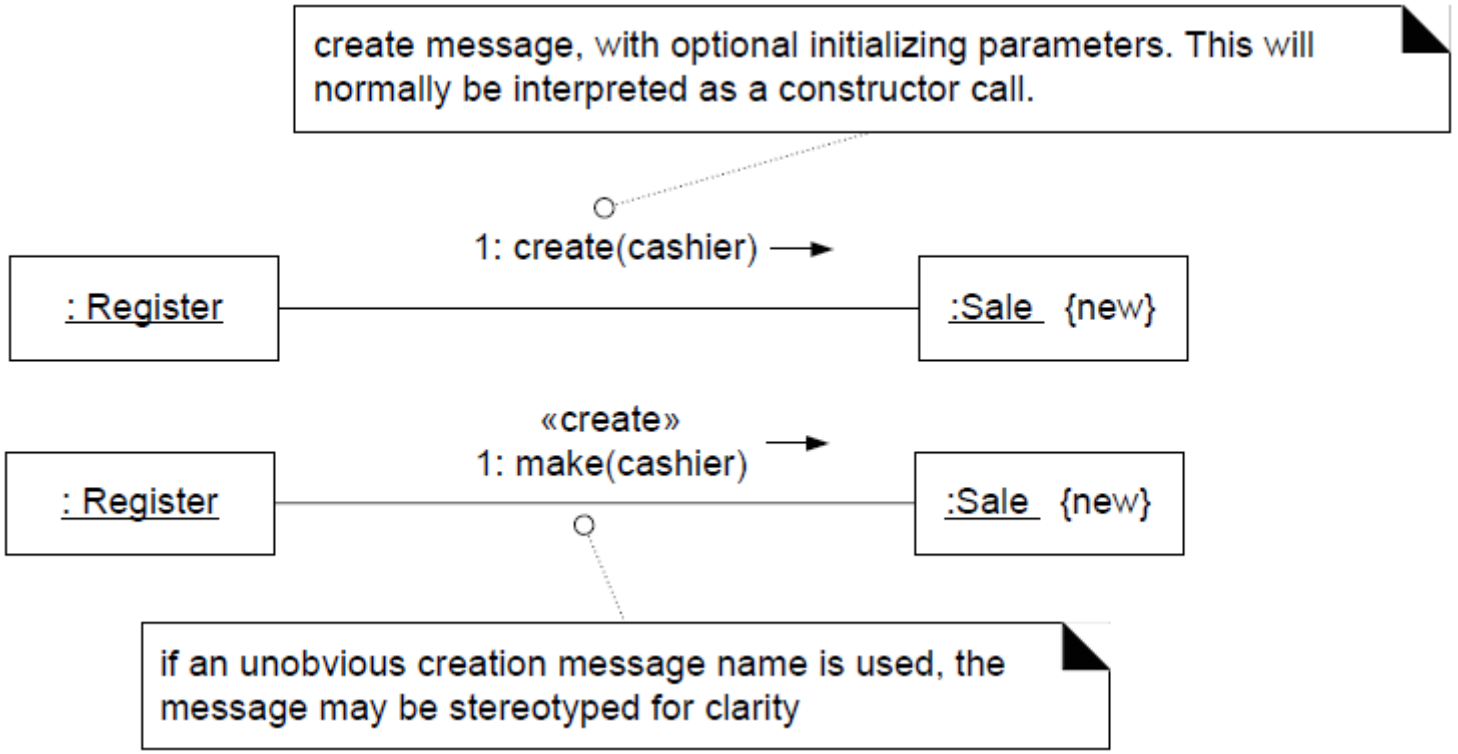


Messages to Self or This

- *A* message can be sent from an object to itself.



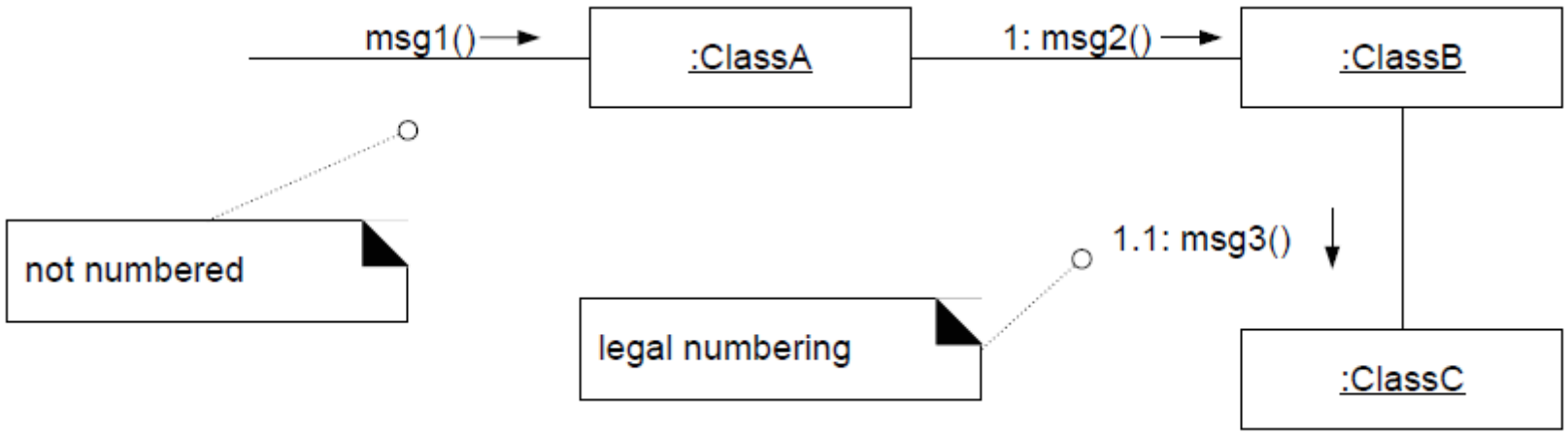
Instance Creation (Constructors)



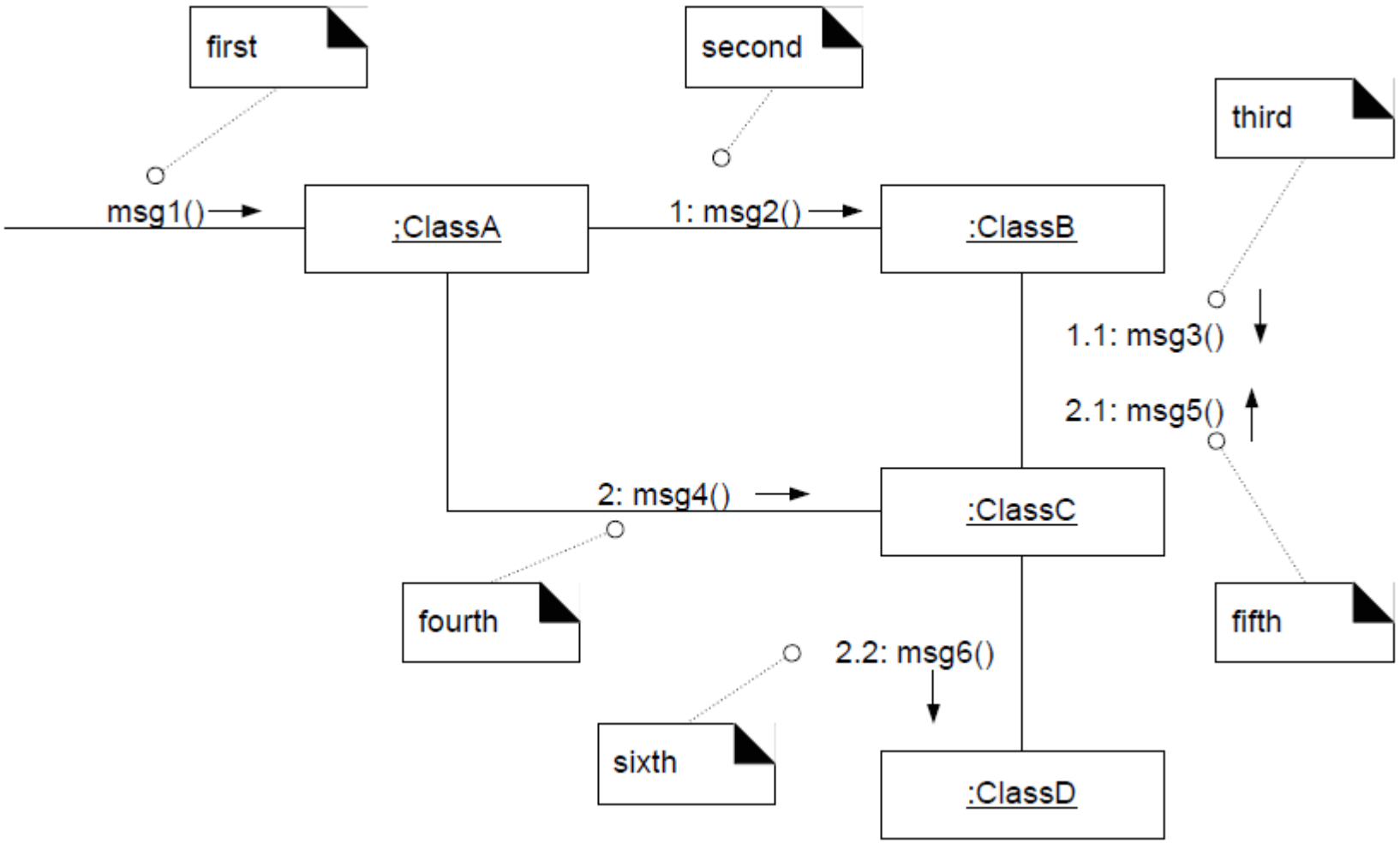
- Furthermore, the UML property *{new}* may optionally be added to the instance box to highlight the creation.

Message Number Sequencing

- The order of messages is illustrated with **sequence numbers**

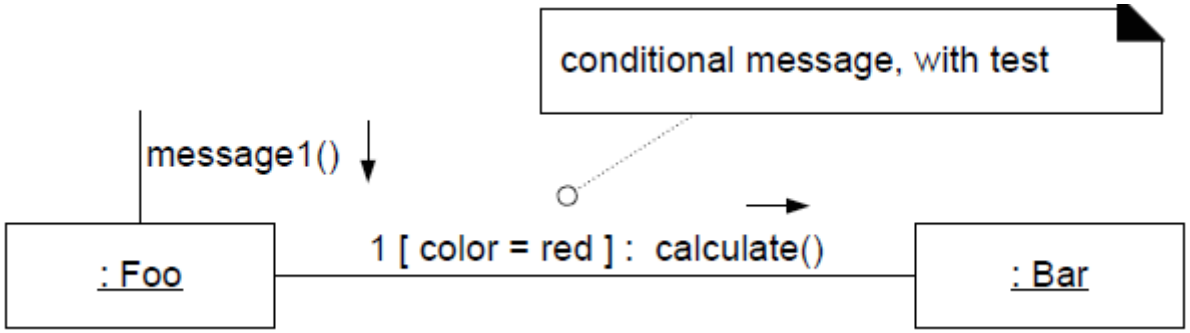


Complex Sequence Numbering



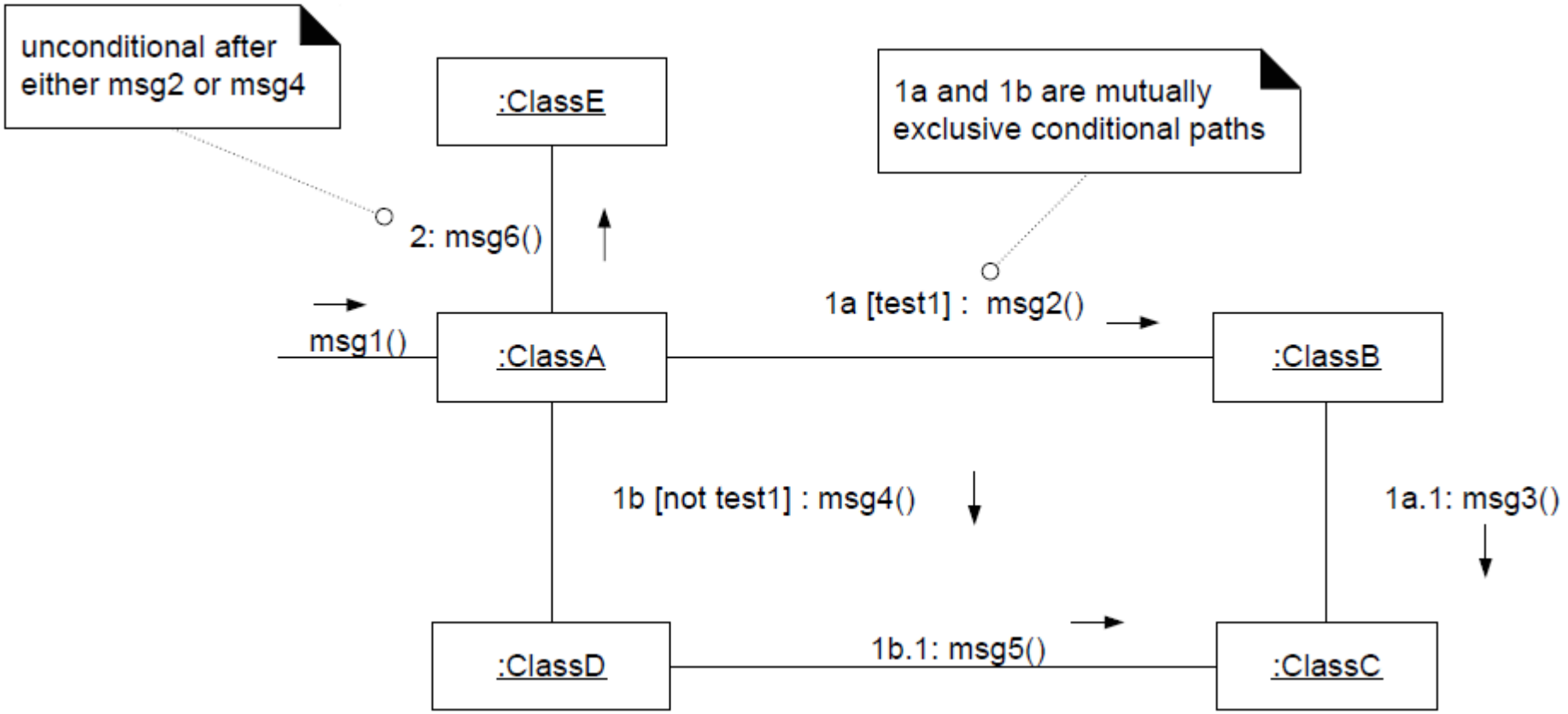
Conditional Messages

- The message is only sent if the clause evaluates to *true.*

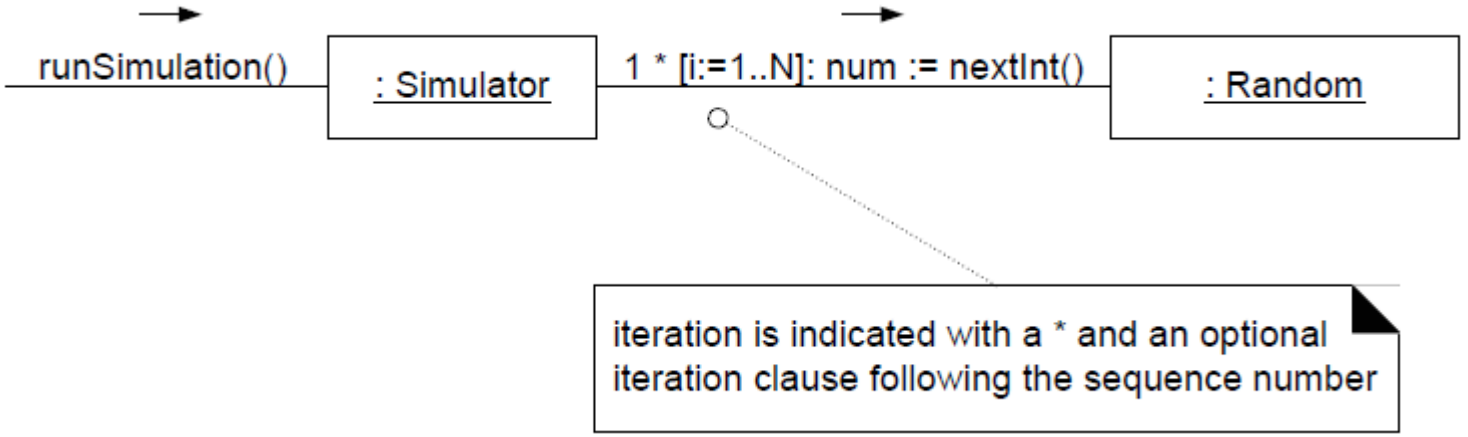


Mutually Exclusive Messages

- The example illustrates the sequence numbers with mutually exclusive conditional paths.
  - In this case it is necessary to modify the sequence expressions with a conditional path letter.

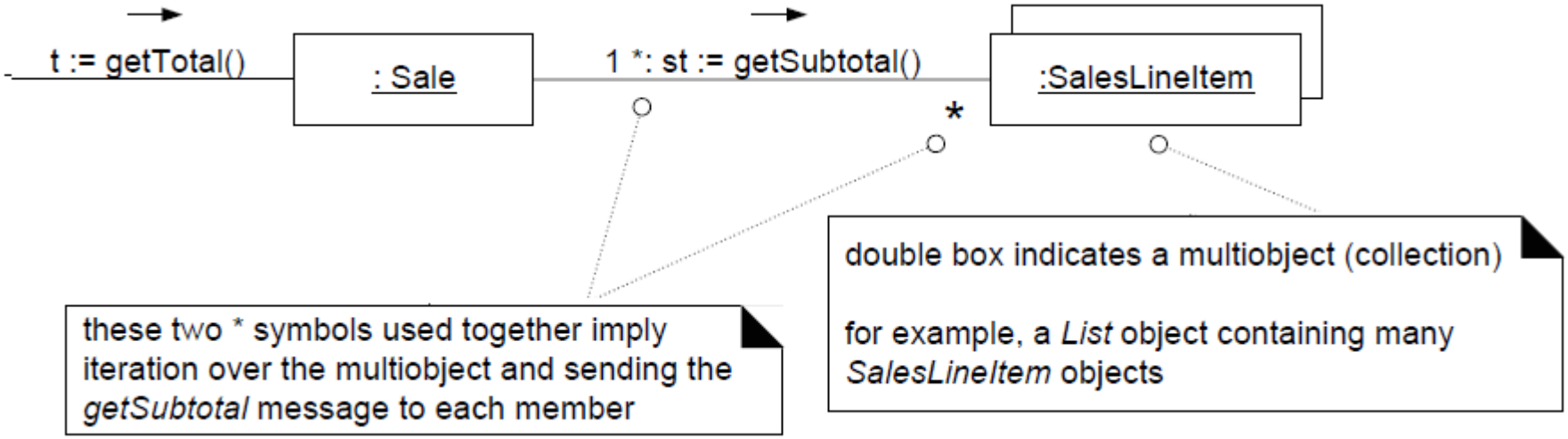


Iteration or Looping



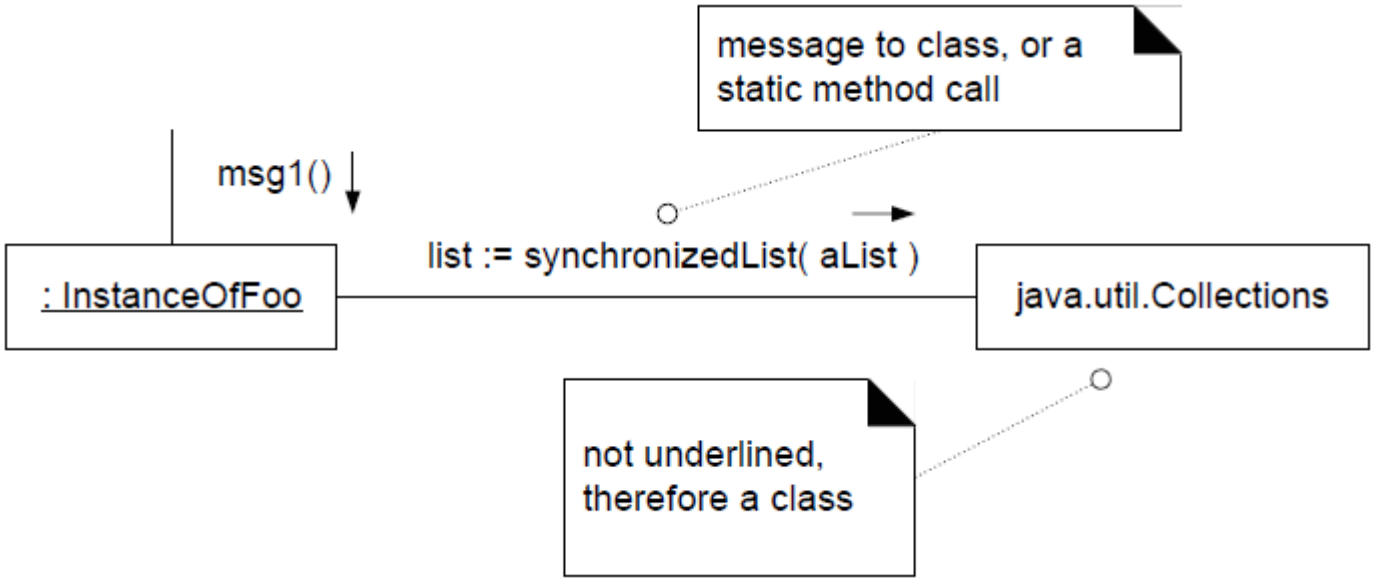
- If the details of the iteration clause are not important to the modeler, a simple just '*' can be used instead.

Iteration Over Collection



Messages to Class Object

- Messages may be sent to a class itself, rather than an instance, to invoke class or **static methods.**



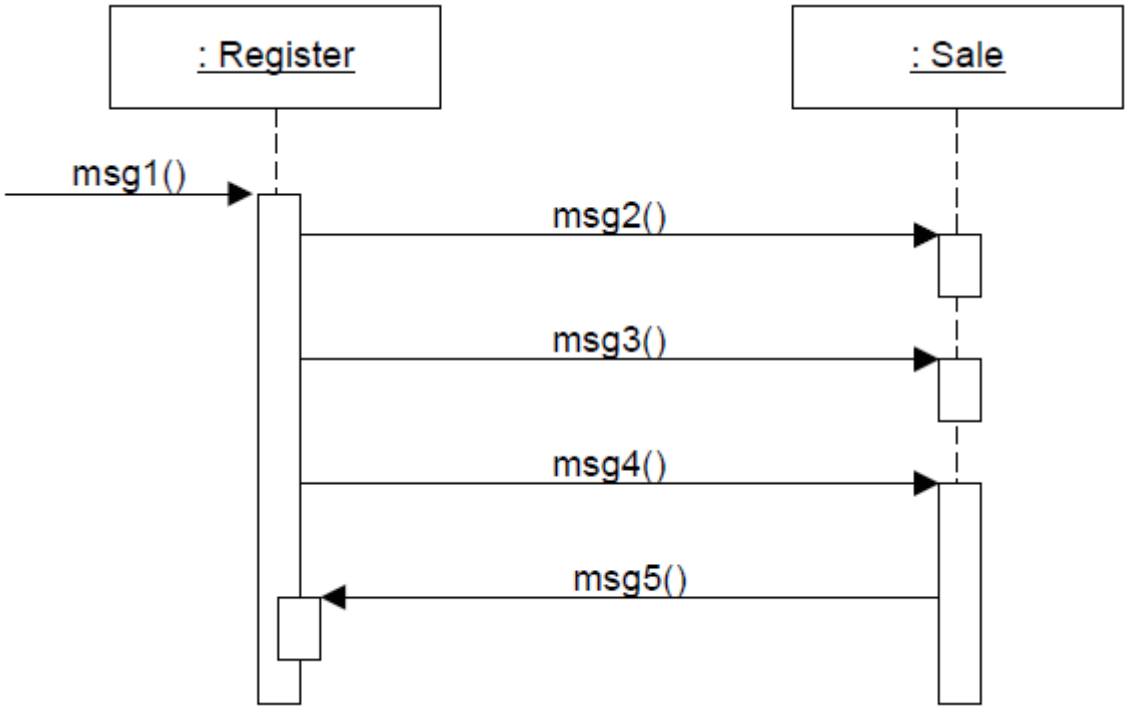
Sequence Diagram Notation

Sequence Diagrams Notation

- Links
  - Unlike collaboration diagrams, sequence diagrams do not show links.
- Messages
  - Each message between objects is represented with a message expression on an arrowed line between the objects.

Messages

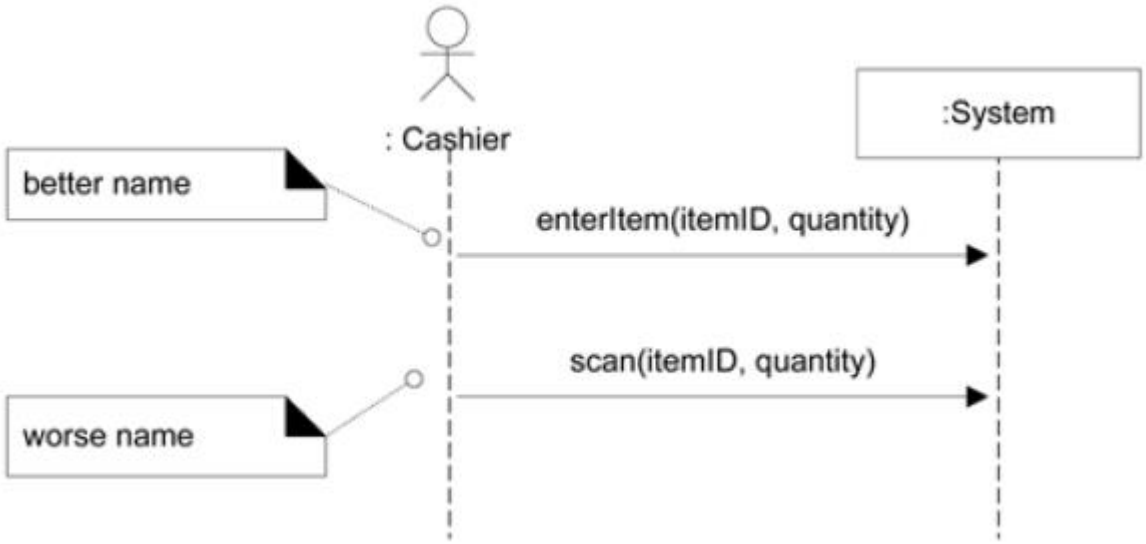
- The time ordering is organized from top to bottom.



- Activation boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message.

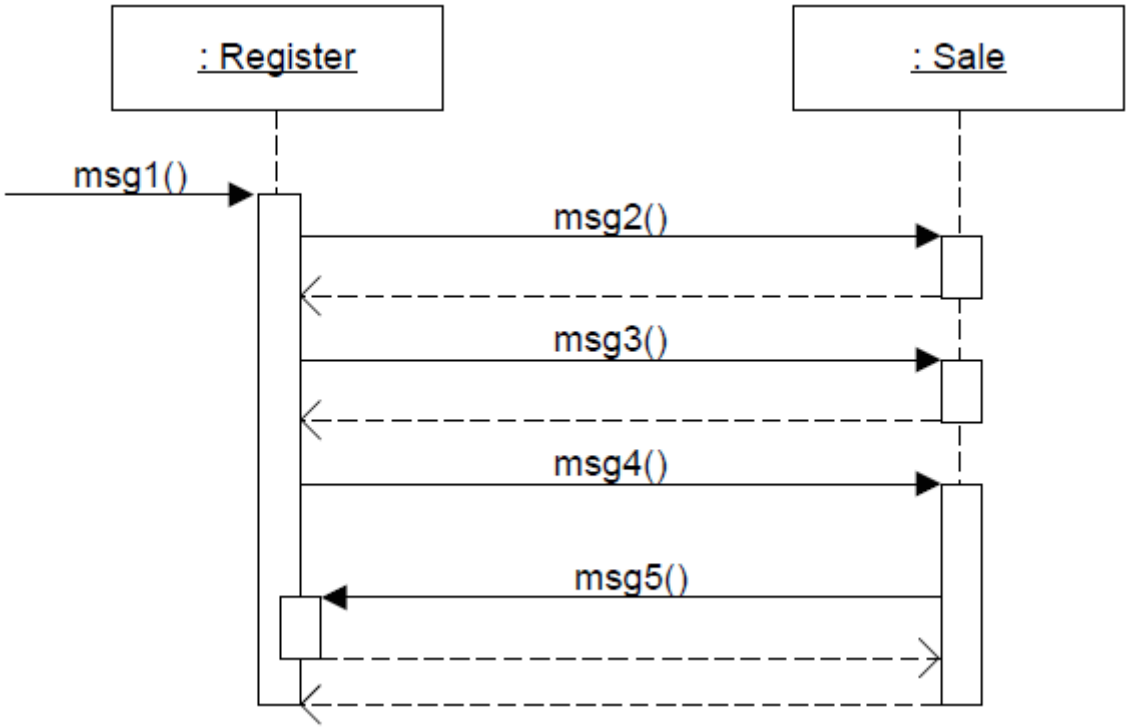
How to Name System Events and Operations?

- System events should be expressed at the abstract level of intention rather than in terms of the physical input device.
  - abstract and noncommittal with respect to design choices about what interface is used to capture the system event.



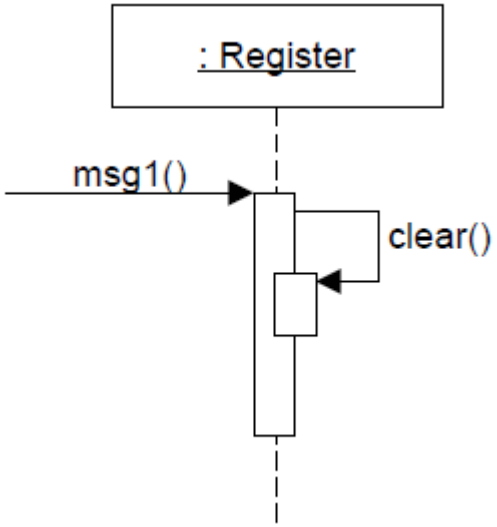
Illustrating Returns

- A sequence diagram may optionally show the return from a message as a dashed open-arrowed line at the end of an activation box.
  - Many practitioners exclude them.
  - Some annotate the return line to describe what is being returned (if anything) from the message.

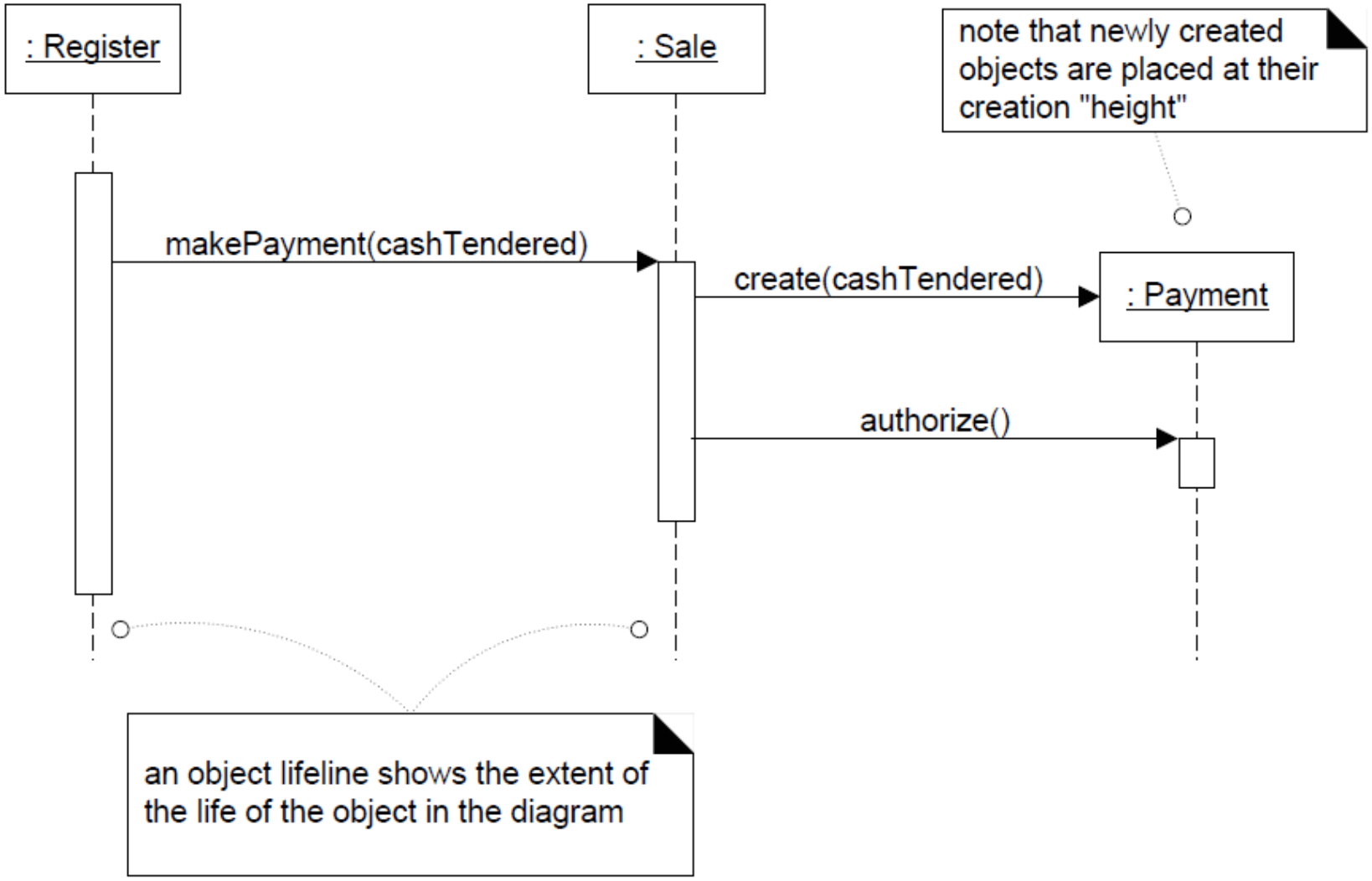


Messages to Self or This

- A message can be illustrated as being sent from an object to itself by using a nested activation box.

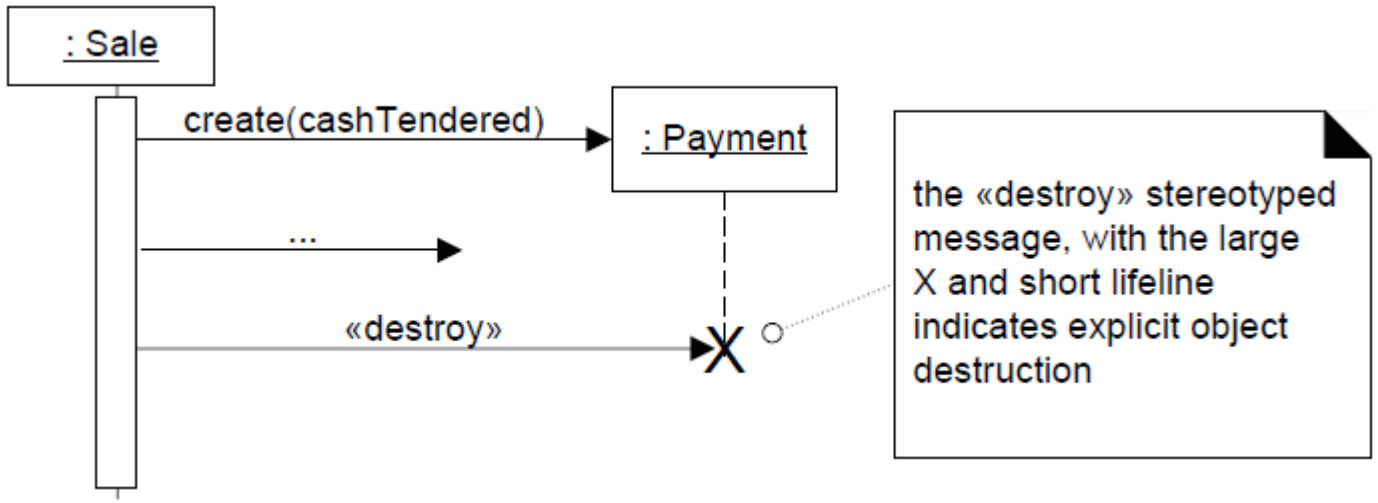


Creation of Instances

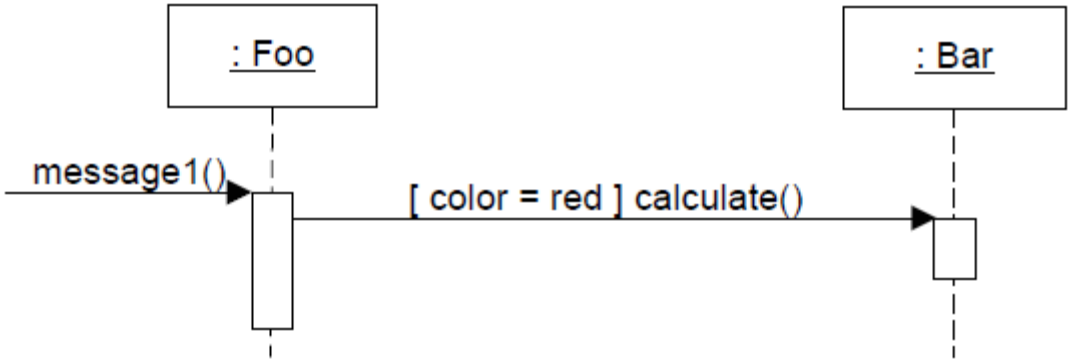


Object Lifelines and Destruction

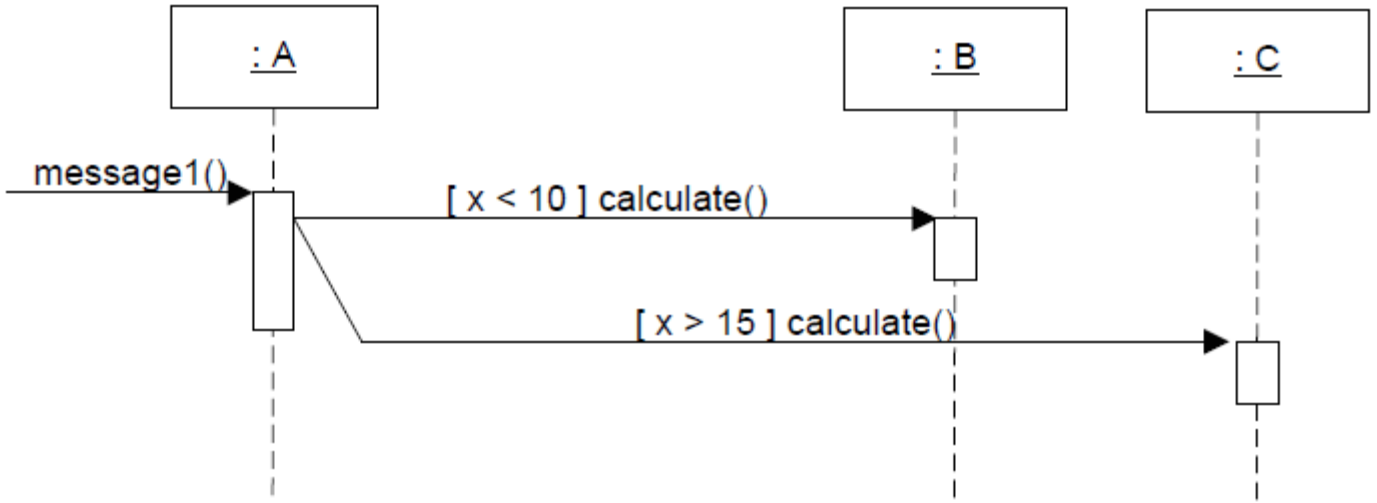
- In some circumstances it is desirable to show explicit destruction of an object (as in C++, which does not have garbage collection);



Conditional Messages

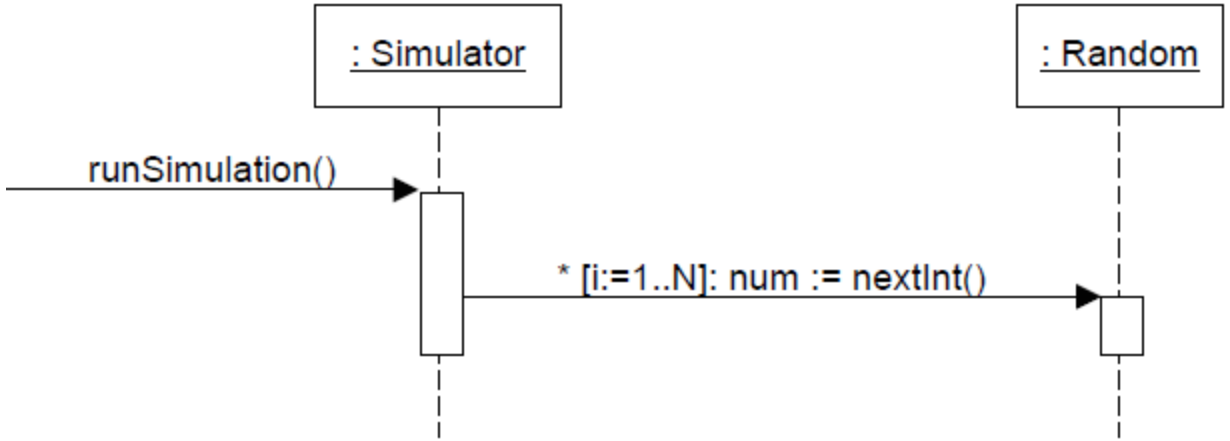


Mutually Exclusive Conditional Messages

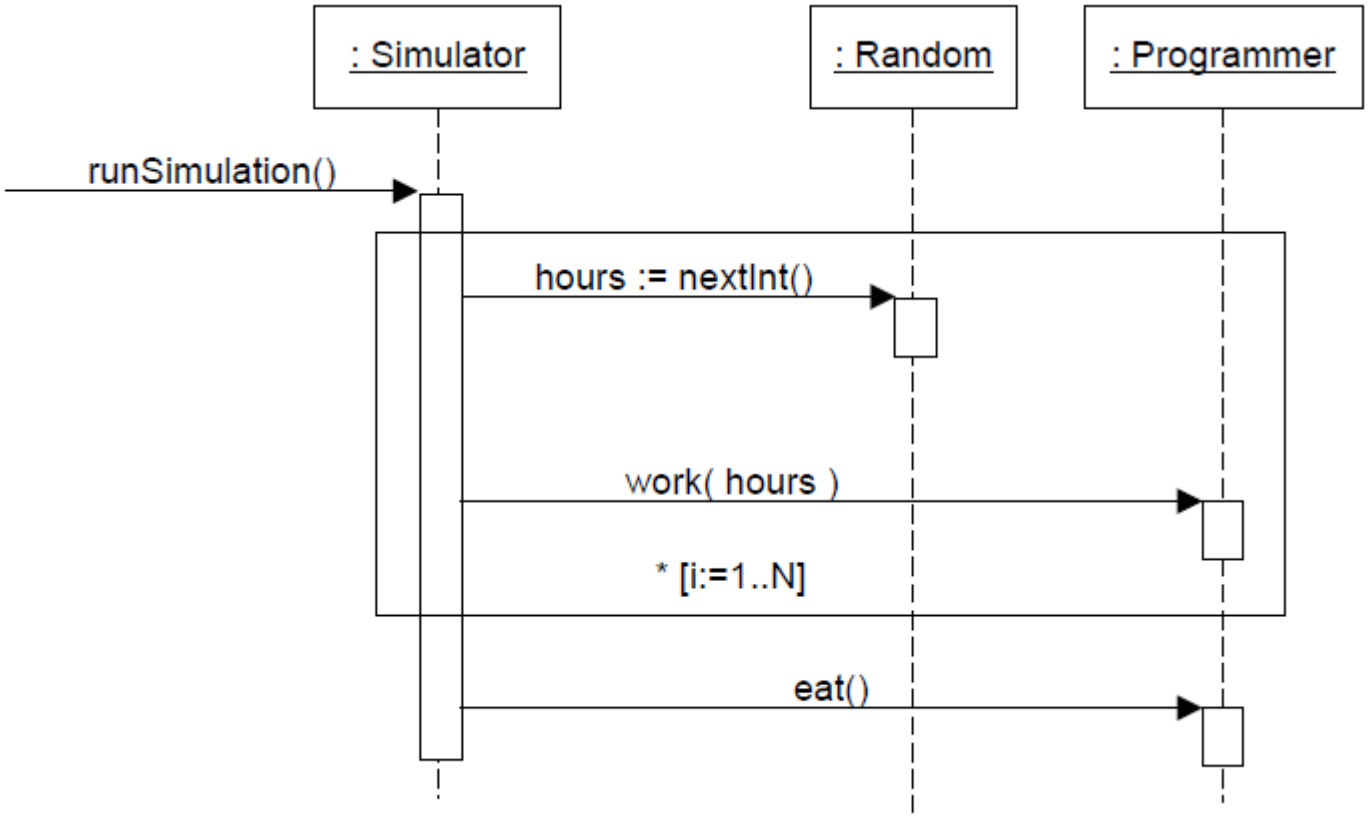


Iteration for Single Message

- Iteration notation for one message

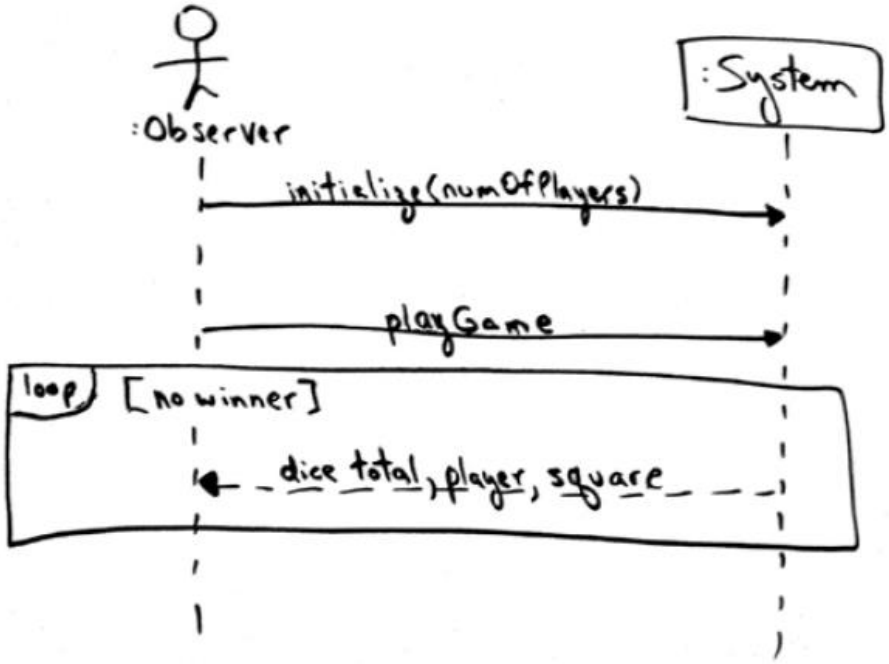


Iteration for Sequence of Messages



Example: Monopoly SSD

- The *Play Monopoly Game* use case is simple, as is the main scenario.
  - The observing person initializes with the number of players, and then requests the simulation of play, watching a trace of the output until there is a winner.

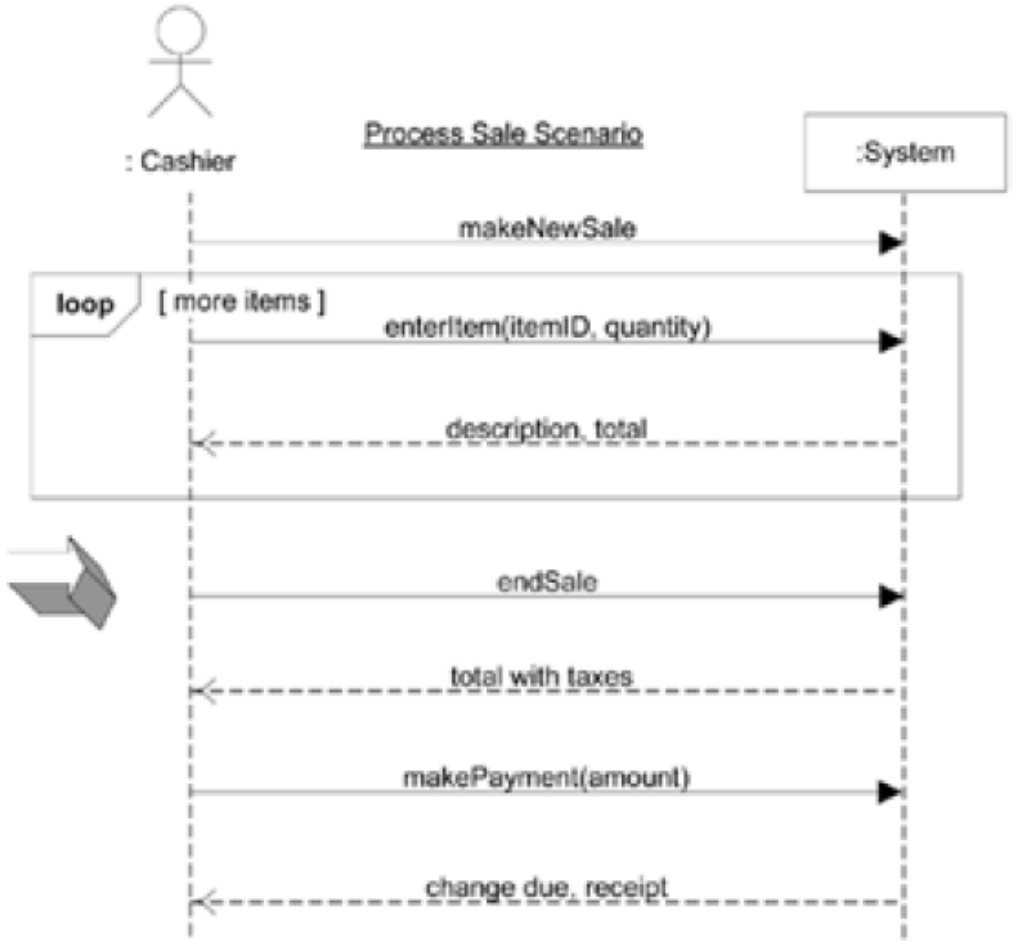


Relationship Between SSDs and Use Cases

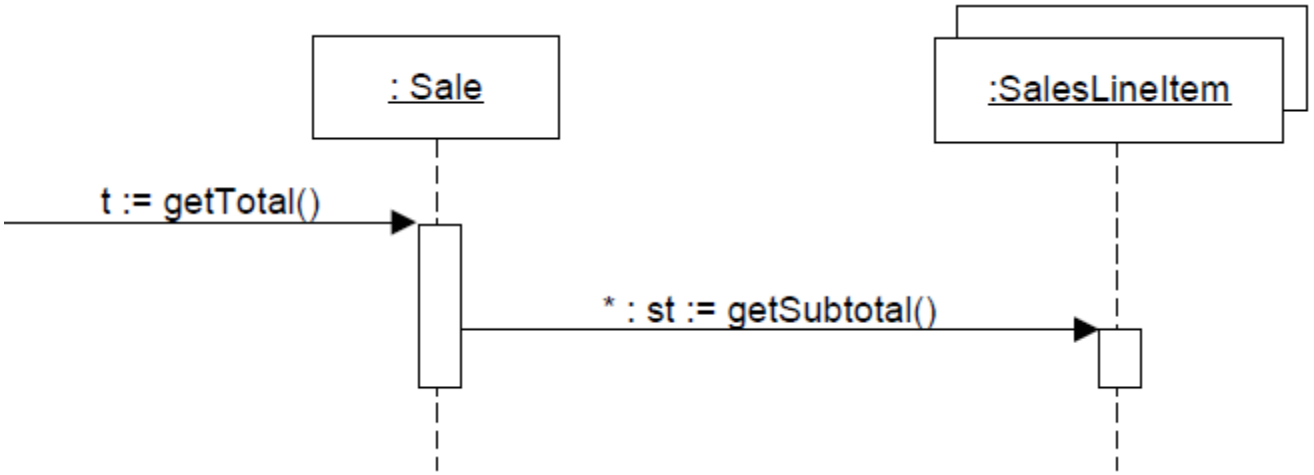
- SSDs are derived from use cases; they show one scenario.

Simple cash-only Process Sale scenario:

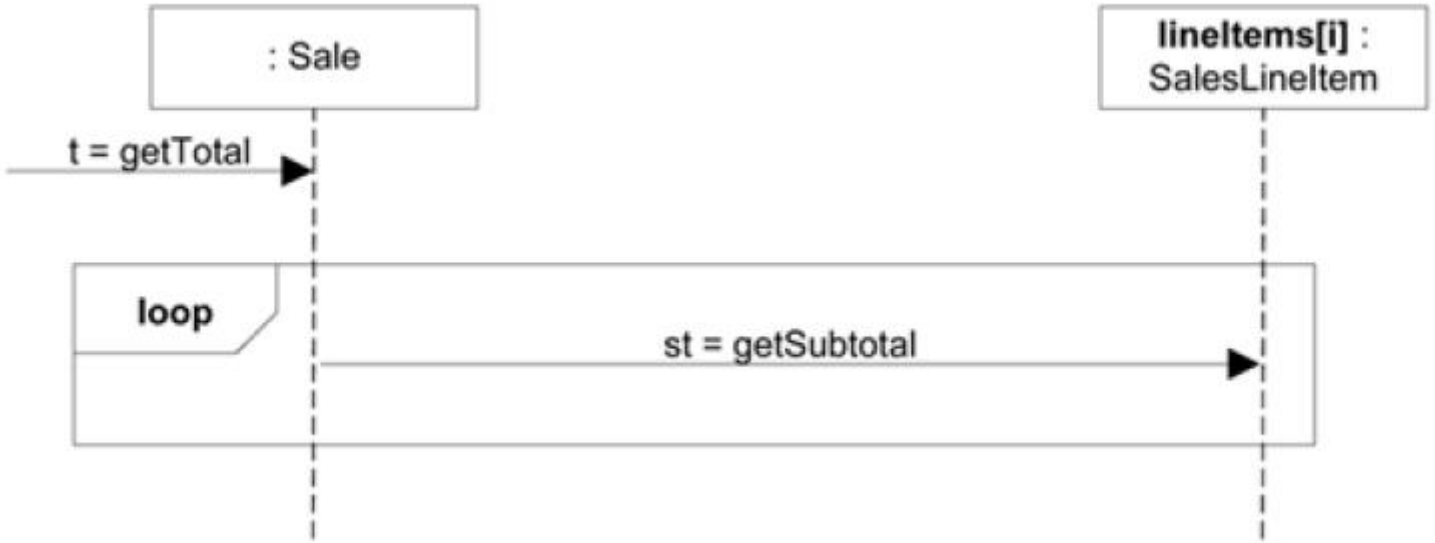
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



Iteration over Multiobject

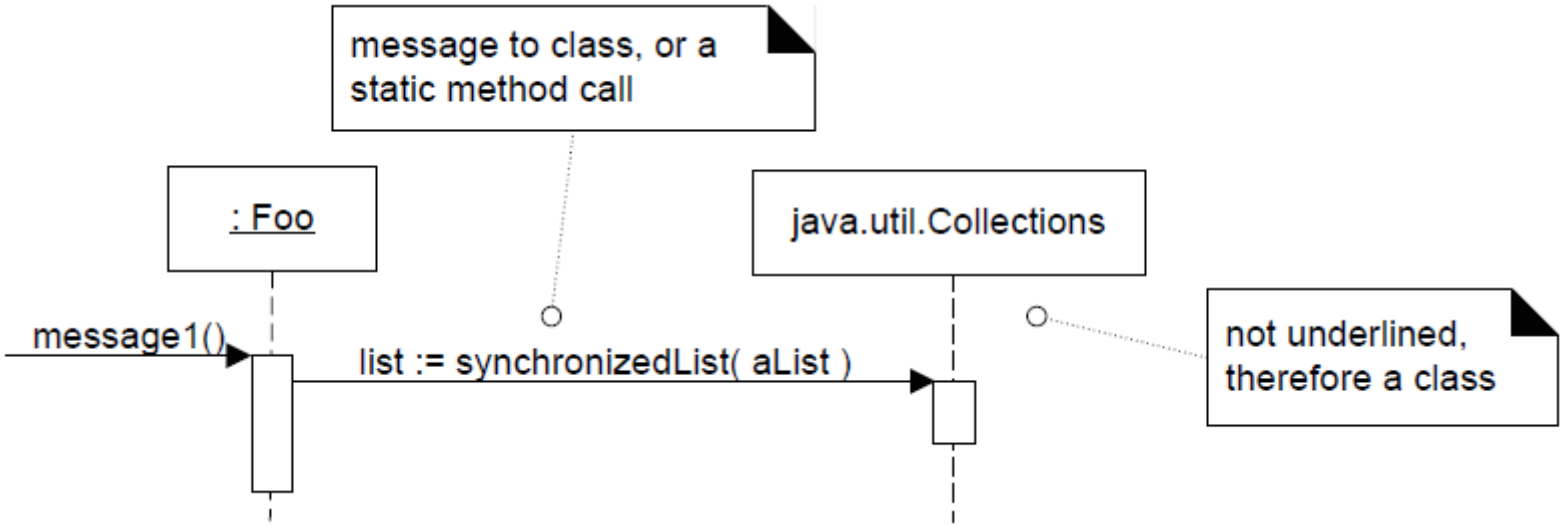


Iteration over a Collection




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


public Money getTotal()
{
   Money total = new Money();
   Money subtotal = null;

   for ( SalesLineItem lineItem : lineItems )
   {
      subtotal = lineItem.getSubtotal();
      total.add( subtotal );
   }
```

Invoking Class or Static Methods



Quiz

- What are the two types of Interaction Diagrams?
- Why Interaction Diagrams are useful?
- What is the advantage of Sequential Diagrams over Collaboration Diagrams?
- How to order messages in Collaboration Diagrams and Sequential Diagrams, respectively?
- How to iterate over single object and Multiobject in SDs?

Actions

- Review Slides.
- Practice interaction notation in MS Visio.
- Read Chapter 15, Interaction Diagram Notation
  - *Applying UML and Patterns*, Craig Larman