OOSE

POST Case study Design Initiation Communication Diagrams

Pádraig de Burca
padraig.deburca@ncirl.ie

Aim and Objectives

Aim

- Appreciate the Design phase of building a POST system

Objectives

- To understand how to create:
  - Interaction Diagrams
    - Communication Diagrams
  - Classes and Instances
  - Links
  - Messages
  - Return Types
  - Self
  - Iterations
  - Instances
  - Message Number sequences
  - Collections
  - Messages to Class Objects

Module Syllabus

Communication Diagrams (15%)

- Guidelines
- Classes and instances
- Links
- Messages
- Return types
- Iterations
- Creation of instances
- Message number sequencing
- Conditional messages
- Collections
- Class messages

Design phase initiation

During an iterative development cycle it is possible to move to a design phase, once the documents in the analysis phase are complete.

During the design phase a **solution** based upon the object-oriented paradigm is developed.

The heart of this solution is the creation of **interaction diagrams**, which illustrate how objects will communicate in order to fulfill the requirements.

Design class diagrams are then drawn to summarize the definition of the classes that are to be implemented in software.

Design phase initiation

A preliminary best guess at post-conditions for the system operations are described in the operation **contracts**.

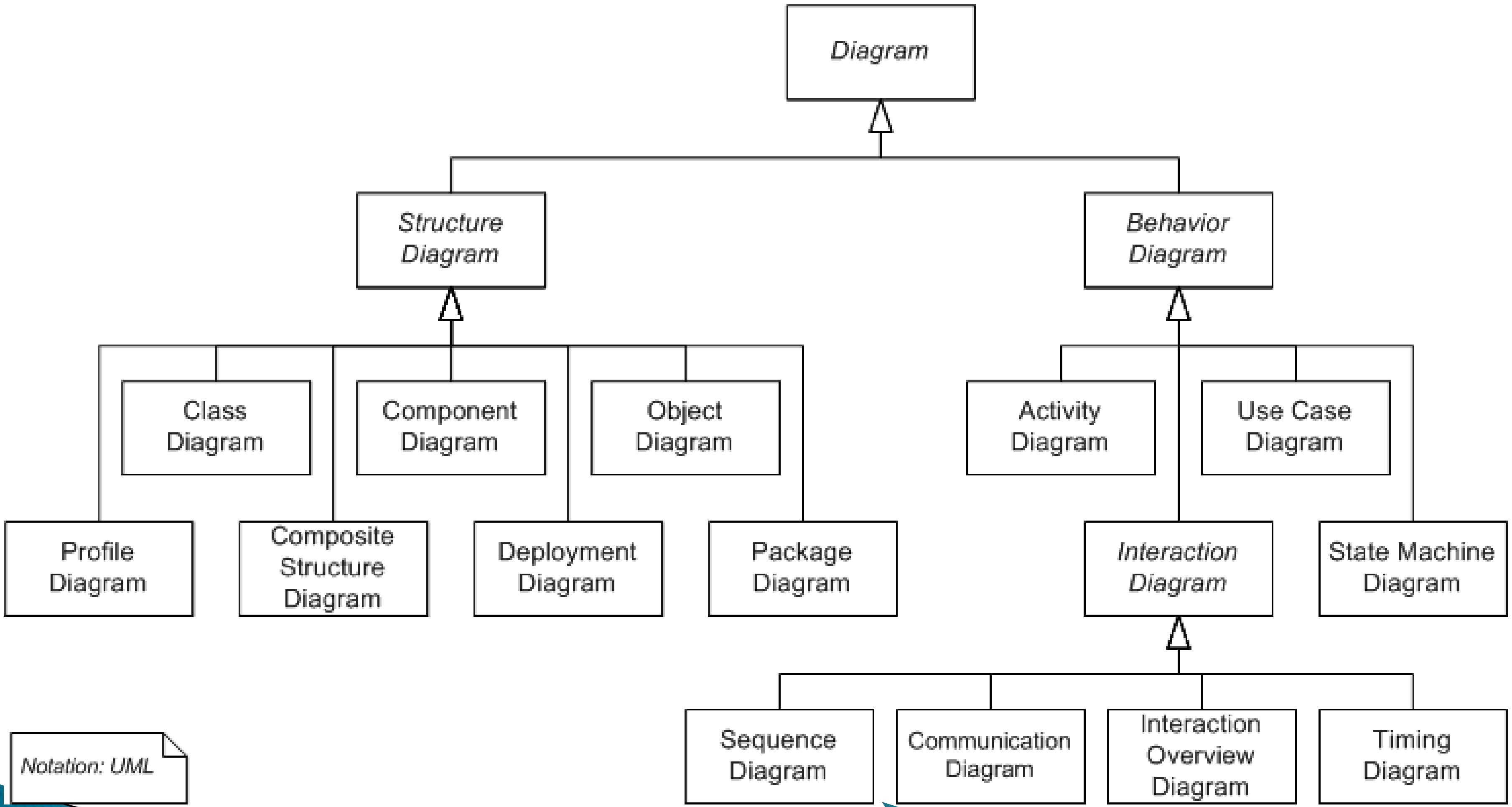
However, the contracts do not show a solution of **how** software objects are going to fulfill the post-conditions.

The UML includes **interaction diagrams** to illustrate how objects interact via messages to fulfill tasks.

The creation of interaction diagrams is dependent upon the prior creation of the following artifacts:

- **Conceptual class model** – from this, the designer may choose to define software classes corresponding to concepts.
- System operation **contracts** – from these, the designer identifies the responsibilities and post-conditions that the interaction diagrams must fulfill.

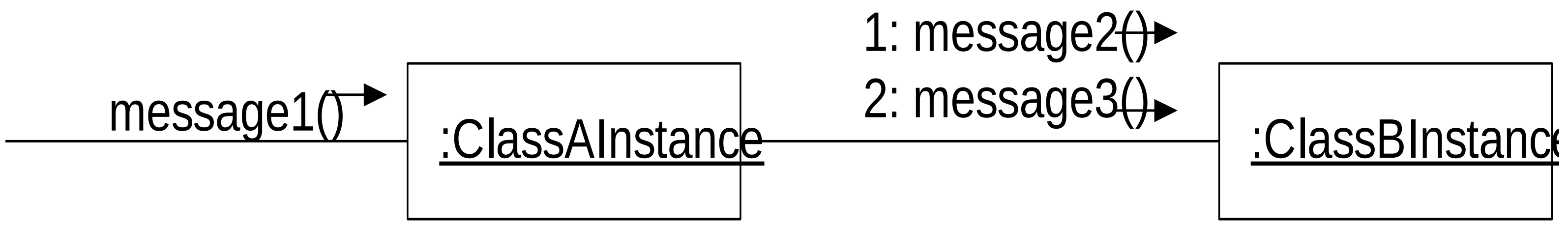
Object Oriented diagrams



Types of OOSE diagrams

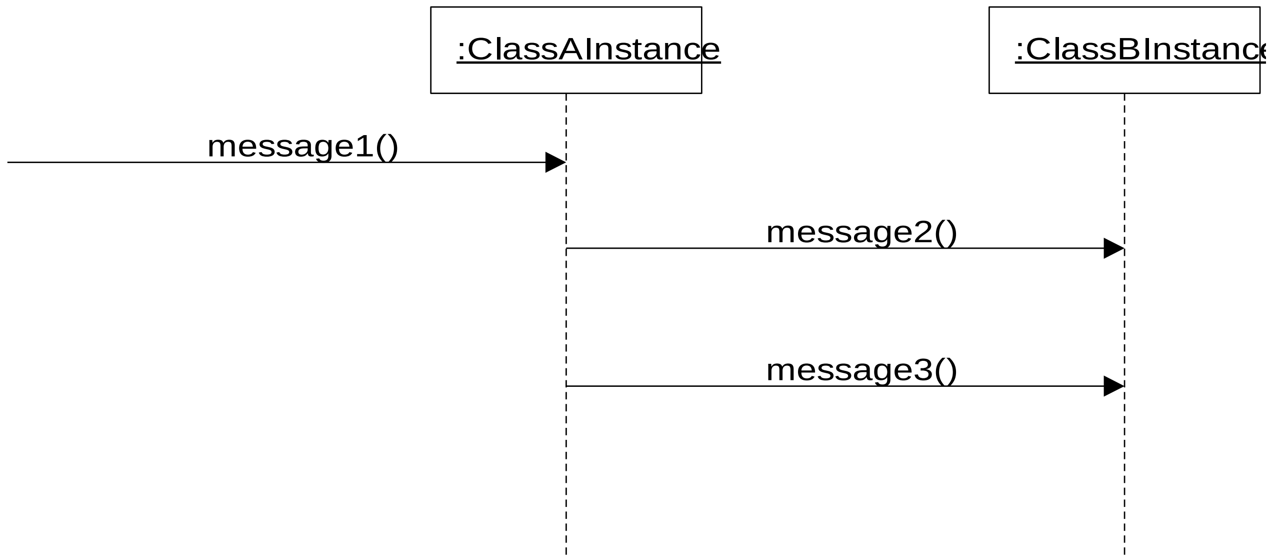
Design phase

- Interaction diagrams
  - An interaction diagram illustrates the message interactions between objects.
  - The UML defines two kinds of interaction diagram, either of which can be used to express similar or identical message interactions.
    - **Communication diagrams** illustrate object interactions in a graph or network format:



Design phase

- Interaction diagrams
  - As we have already seen, **sequence diagrams** illustrate interactions in a kind of fence format:



  - In this course, Communication diagrams will be emphasized over sequence diagrams because of their ability to convey more information

Design phase

- Interaction diagrams
  - Interaction diagrams are one of the most important artifacts created in object-oriented analysis and design.
  - **The amount of time and effort spent on their generation should absorb a significant percentage of the overall project effort.**
  - Codified **patterns**, principles and idioms can be applied to improve the quality of their design.
- UML Sequence diagrams and Communication diagrams use common elements and express similar information, however, Communication diagrams are more focused on showing the collaboration of objects rather than the time sequence in detail.

Communication Diagrams

.. are used for describing communication between objects or communication partners commonly executing a system operation.

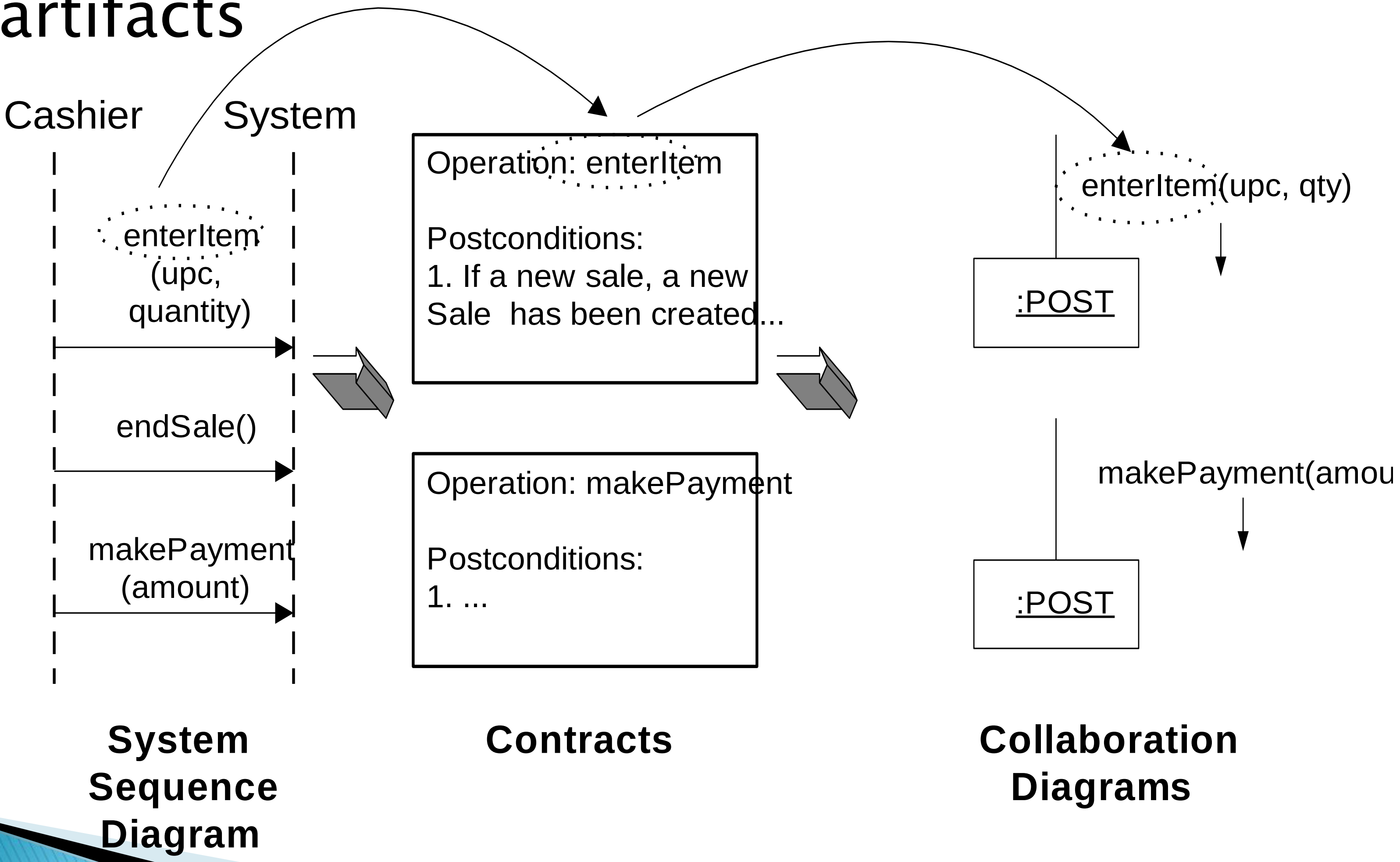
.. are most useful for representing interrelations between objects and for procedural design.

.. are designed by systems analysts, designers and implementers during analysis workflow.

- During the following phases of software development, they are further elaborated and completed.

Communication Diagrams

- Communication diagrams and other artifacts



Communication Diagrams

- Communication diagrams and other artifacts
  - The use cases suggest the system events which are explicitly shown in system sequence diagrams.
  - An initial best guess at the effect of the system events is described in system operation contracts.
  - The system events represent messages that **initiate** interaction diagrams, which illustrate how objects interact to fulfill the required tasks.
  - The interaction diagrams involve message interaction between objects defined in the conceptual model, plus other classes of objects.

Communication Diagram guidelines

Create a separate diagram **for each system operation** under development in the current development lifecycle.

For each system operation message, make a diagram with it at the **starting message**,

If the diagram gets complex, split it into smaller diagrams.

Using the operation contract responsibilities and post-conditions, and use case description as a starting point, design a system of interacting objects to fulfill the tasks.

Apply **patterns** to develop a good design.

Patterns

- Experienced object-oriented developers (and other software developers) build up a repertoire of general principles that guides them in the creation of software.
- These principles, if codified in a structured format describing the problem and solution, and given a name, may be called **patterns**.
- Patterns do not usually contain new ideas!

Patterns

All patterns ideally have meaningful names:

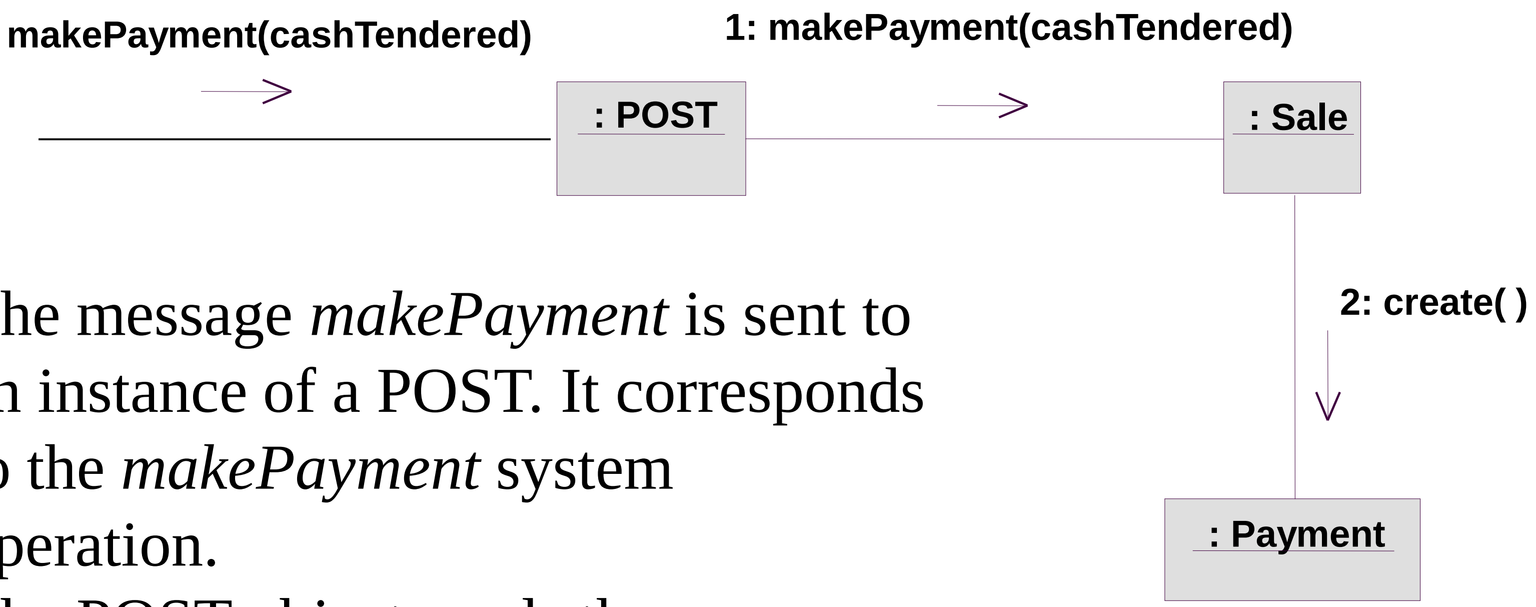
- This supports incorporating a concept into our understanding and memory
- It facilitates communication.
- Naming a complex idea such as a pattern is an example of the power of abstraction – reducing a complex form to a simple one by eliminating detail.

We will look in detail at the following patterns (next week):

- **Expert**
- **Creator**
- **Controller**

Communication Diagrams

- POST Communication Diagram: makePayment

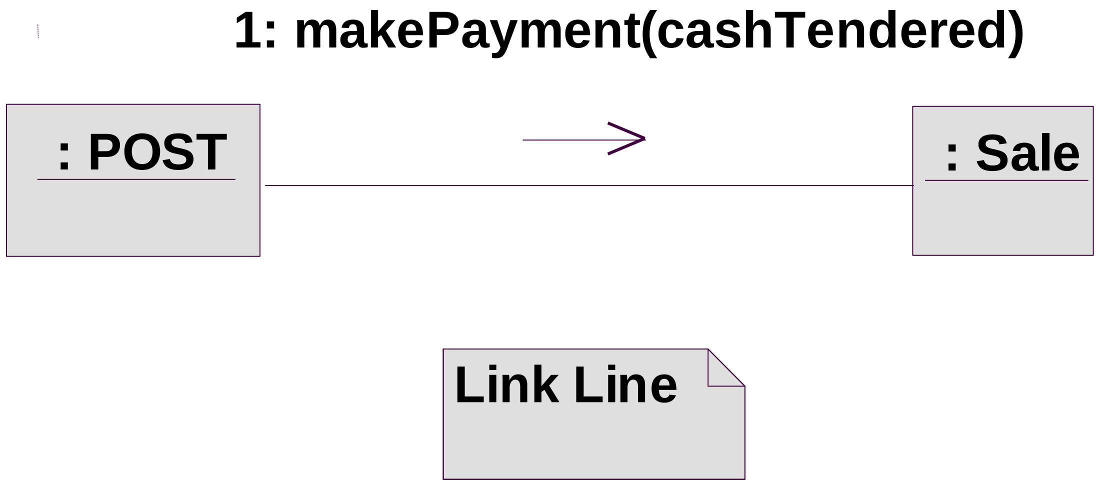


1. The message *makePayment* is sent to an instance of a POST. It corresponds to the *makePayment* system operation.
2. The POST object sends the message to a *Sale* instance.
3. The *Sale* instance creates an instance of a *Payment*.

There is a flaw in this example – we'll come back to it later!

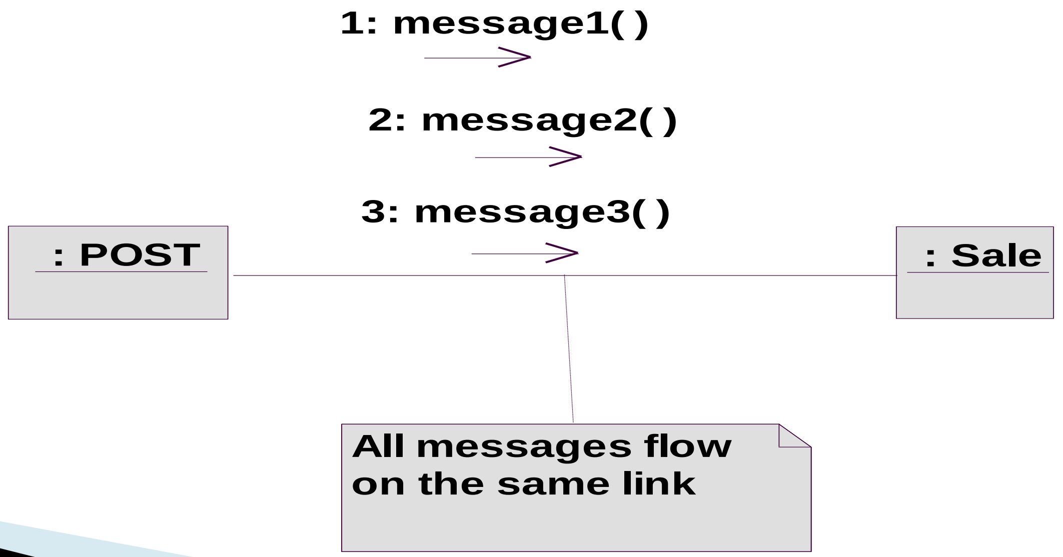
Communication Diagrams

- Illustrating links
  - A link is a path between two instances; more formally, a link is an instance of an association.
  - Viewing two instances in a client/server relationship, a path of navigation from the client to server means that messages may be sent from the client to the server.



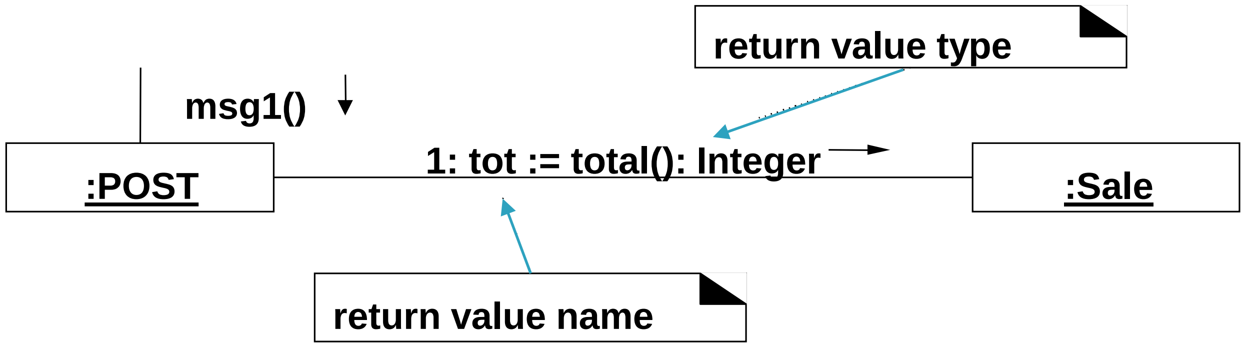
Communication Diagrams

- Illustrating Messages
  - Messages between objects are represented via a labeled arrow on a link line.
  - **Any** number of messages may flow along this link.
  - A sequence number is added to show the sequential order of messages.



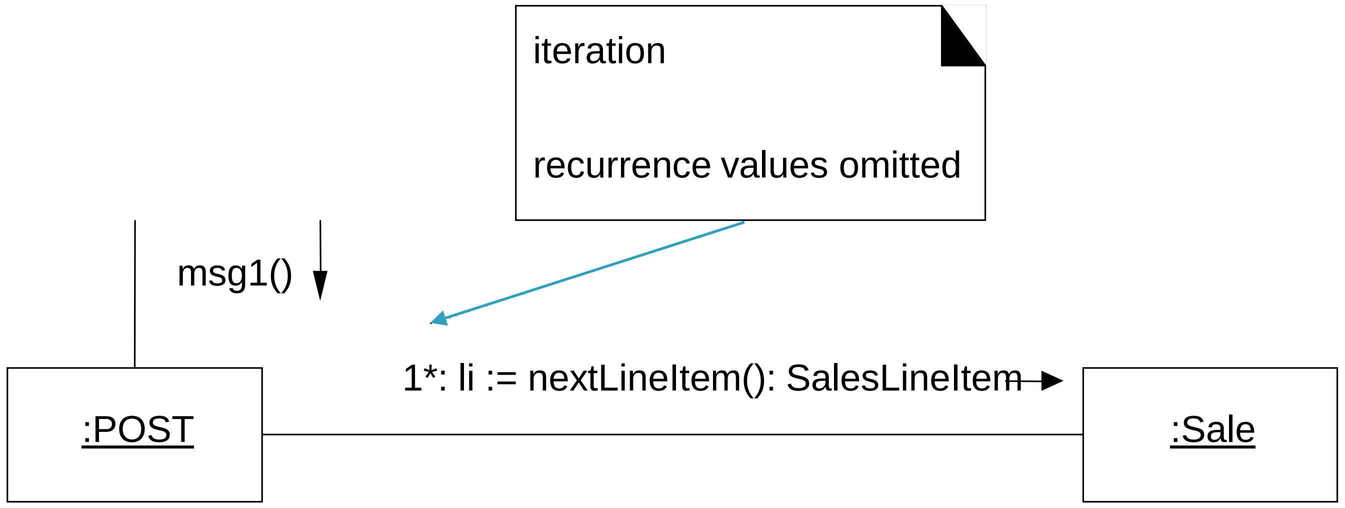
Communication Diagrams

- Illustrating a return type
  - A return value may be shown by preceding the message with a return variable name and an assignment operator (:=)



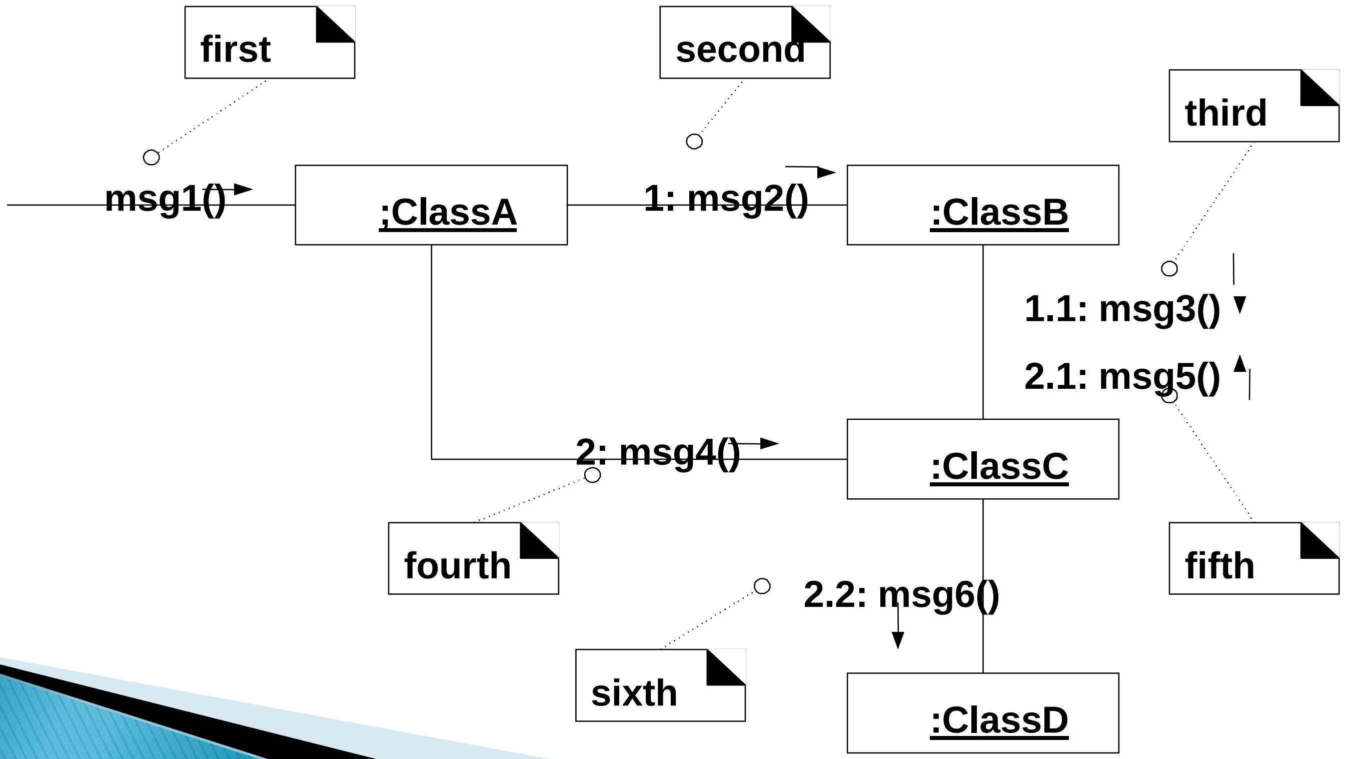
Communication Diagrams

- Illustrating Iteration
  - Iteration is indicated by following the sequence number with a *.
  - This expresses that the message is being sent repeatedly, in a loop, to the receiver:


iteration
recurrence values omitted
msg1()
1*: li := nextLineItem(): SalesLineItem
:POST
:Sale

Communication Diagrams

- Illustrating Message Number sequencing
  - The first message is not numbered. Thus, msg1() is unnumbered.
  - The order and nesting of subsequent messages is shown with a legal numbering scheme:

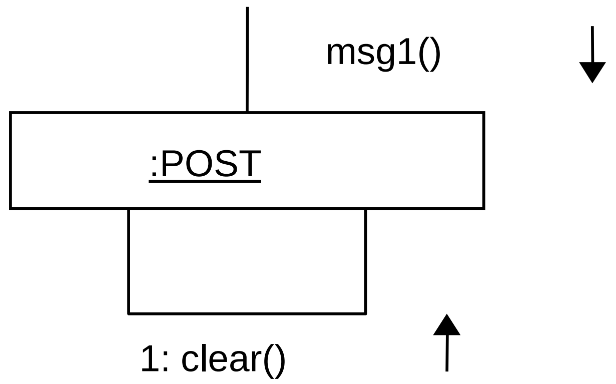


Video

- UML 2 Communication Diagrams - YouTube

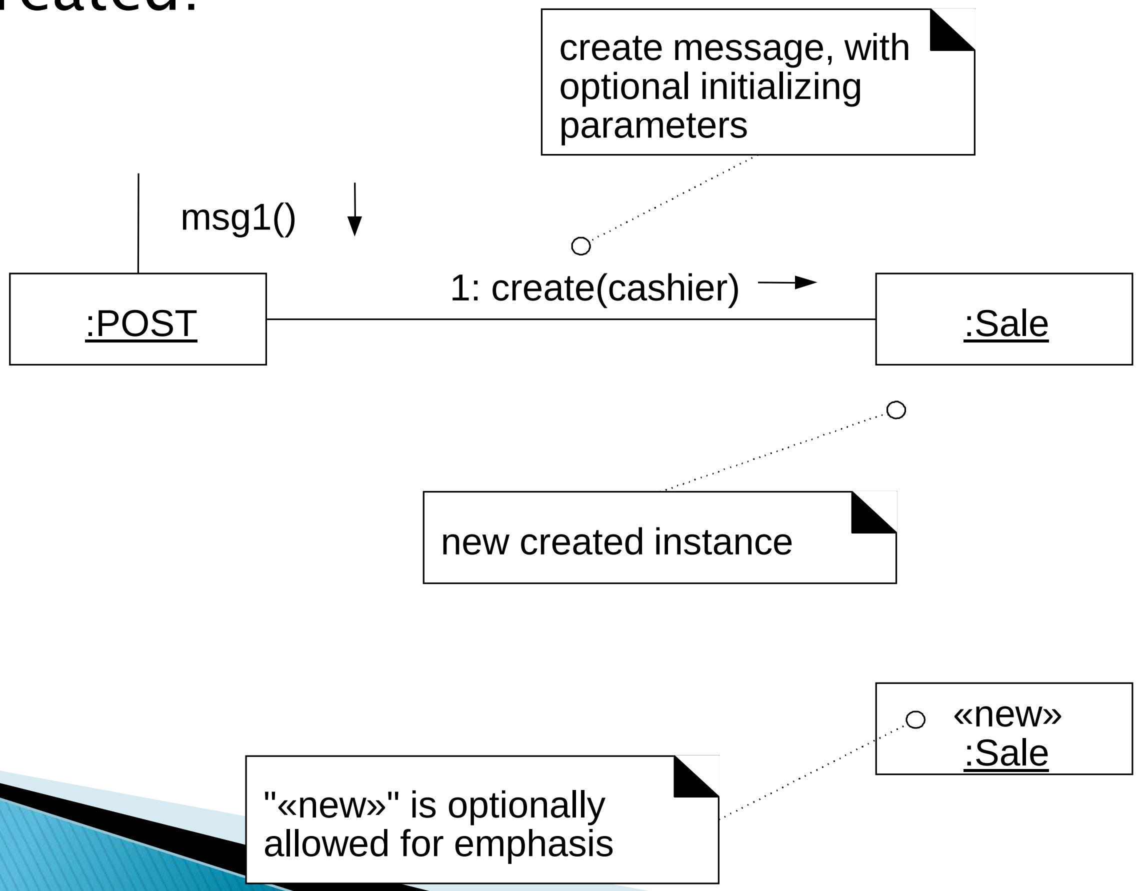
Communication Diagrams

- Illustrating messages to "self"
  - A message can be sent from an object to itself



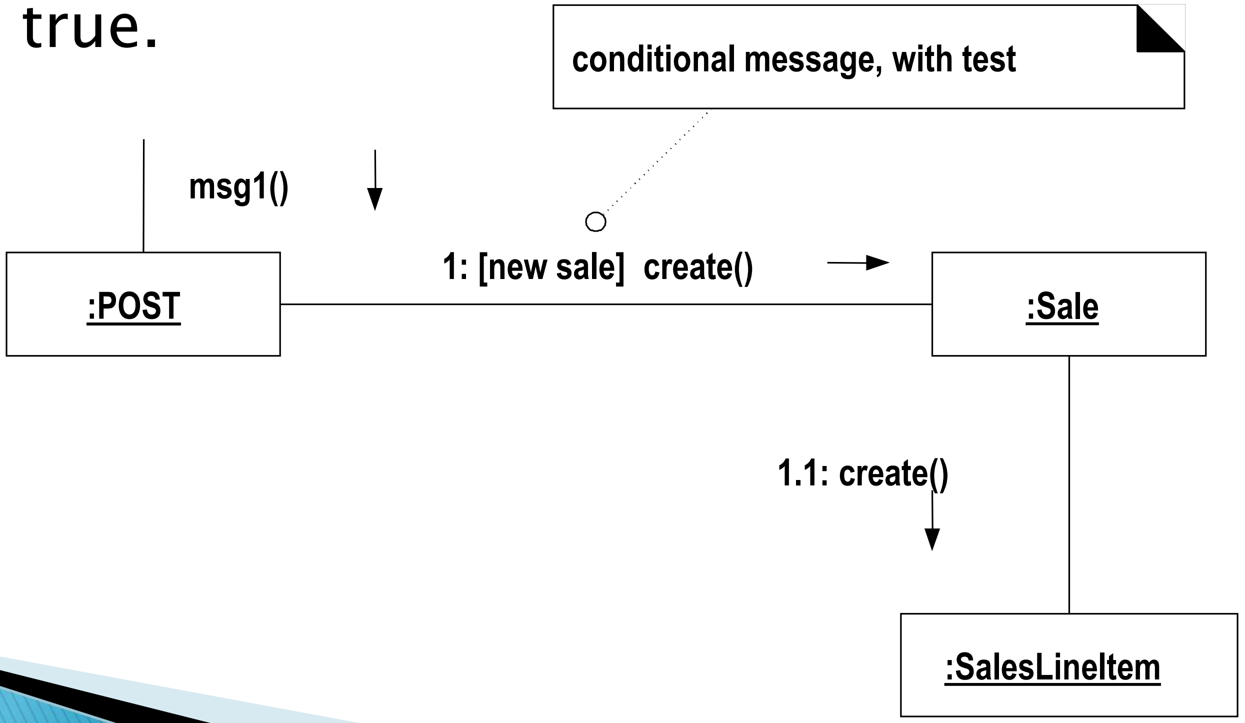
Communication Diagrams

- Illustrating creation of instances
  - The language independent creation message is create, shown being sent to the instance being created:


create message, with optional initializing parameters
msg1()
1: create(cashier)
:POST
:Sale
new created instance
«new» :Sale
"«new»" is optionally allowed for emphasis

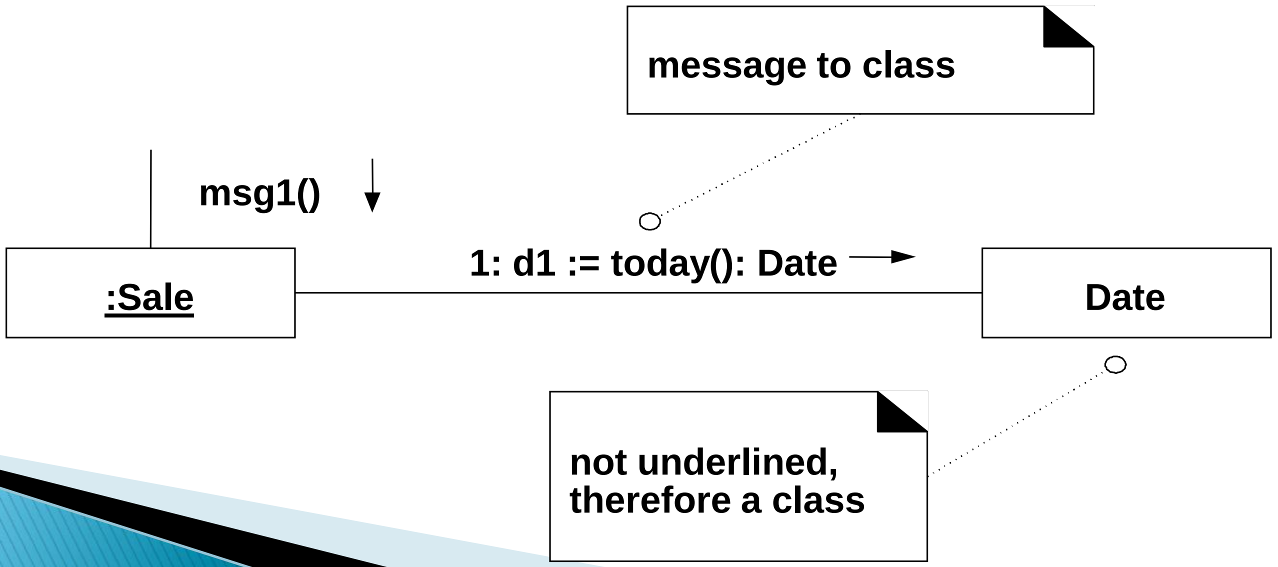
Communication Diagrams

- Illustrating Conditional messages
  - A conditional message is shown by following a sequence number with a conditional clause in square brackets.
  - The message is only sent if the clause evaluates to true.



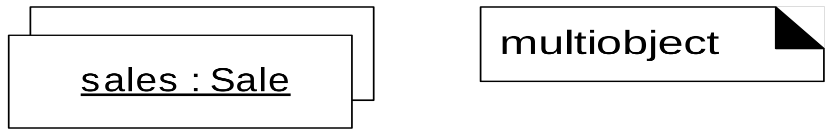
Communication Diagrams

- Illustrating messages to a class
  - Messages may be sent to a class itself, rather than an instance, in order to invoke class methods:



Communication Diagrams

- Illustrating Collections
  - A **multiobject**, or a set of instances, may be shown with a stack icon:



Design phase

There is a **great variability** in the potential quality of object interaction design and **responsibility** assignment.

Poor choices lead to systems which are fragile and hard to maintain, understand, reuse, or extend.

A skillful implementation is founded on the **principles of good object-oriented design.**

Some of these principles, applied during the creation of interaction diagrams and responsibility assignment, are codified in **patterns**.

Design phase: Responsibilities

Responsibilities are related to the **obligations** of an object in terms of it’s **behaviour.**

Basically, these responsibilities are of the following two types:

- **Doing** responsibilities of an object include:
  - doing something itself
  - initiating action in other objects
  - controlling and coordinating activities in other objects.
- **Knowing** responsibilities of an object include:
  - Knowing about private encapsulated data
  - Knowing about related objects

Design phase : Responsibilities

Responsibilities are assigned to objects during object-oriented design.

For example, it may be decided that:

- a *Sale* object is responsible for printing itself (a <u>doing</u>)
- a *Sale* object is responsible for knowing it's date (a <u>knowing</u>)

Responsibilities related to "knowing" are often inferable from the conceptual model, because of the attributes and associations it illustrates.

Interaction diagrams show choices in assigning responsibilities to objects.

Summary

Interaction Diagrams – Communication Diagrams

Interaction diagrams illustrate how objects interact via messages to fulfill tasks

Classes and Instances – instance uses the same graphic symbol as the type, but the name is underlined.

Links – path between two instances; more formally, a link is an instance of an association.

Messages – Messages between objects are represented via a labeled arrow on a link line.

Return Types – A return value may be shown by preceding the message with a return variable name and an assignment operator (:=)

Summary

Self – A message can be sent from an object to itself

Iterations – Iteration is indicated by following the sequence number with a *.

Instances – creation message is create, shown being sent to the instance being created.

Message Number sequences – The first message is not numbered. The order and nesting of subsequent messages is based on a legal numbering scheme.

Collections – A **multiobject**, or a set of instances, is shown with a stack icon

Messages to Class Objects – Messages may be sent to a class itself, rather than an instance, in order to invoke class methods

More Help

- Communication Diagrams in UML 2
- Agile Introduction to UML 2 Communication Diagrams
- Agile Communication Diagram Guidelines
- Visual Paradigm Communication Diagram – UML 2 Diagrams
- SAD with UML ch. 7

Communication Diagrams in Rational Rose

- IBM Rational Rose uses UML 1 rather than UML 2.
  - UML 1 – Collaboration Diagrams
  - UML 2 – Communication Diagrams
- Use Rational Rose in our next lab to draw both the Sequence Diagram and the Collaboration diagram shown in the video tutorials.
  - Creating Sequence Diagrams
  - Creating Collaboration diagrams
- Note difference between System Sequence Diagram & Sequence Diagram.