Unit 3

Object Oriented design

3.1 Analysis to Design

- The requirements and object-oriented analysis has focused on learning to *do the right thing; that is, understanding some of the outstanding goals for the system*, and related rules and constraints.
- By contrast, the following design work will stress *do the thing right; that is, skillfully designing a solution to* satisfy the requirements for this iteration.

- In iterative development, a transition from primarily a requirements focus to primarily a design and implementation focus will occur in each iteration. Furthermore, it is natural and healthy to discover and change some requirements during the design and implementation work of the *early iterations. These discoveries* will both clarify the purpose of the design work of this iteration and refine the requirements understanding for future iterations

Operation Contracts

- Use cases are the primary mechanism in the UP to describe system behavior, and are usually sufficient. However, sometimes a more detailed description of system behavior has value. Contracts describe detailed system behavior in terms of state changes to objects in the Domain Model, after a system operation has executed.

Writing Operation Contracts

- Operation**—name of operation (parameters)**
- Cross Reference**—the Use Cases in which the** OC occurs
- Preconditions**—noteworthy assumptions about** state of system or objects in DM before execution
- Postconditions**—state of objects in DM *after*** execution of operation

Postconditions

- Most important part of OCs!
- Include changes in state of DM
- Book uses categories (note that the names are for reference only):
  - Instance creation or deletion
  - Association formed or broken
  - Attribute modification

This is a sample OC for "enterItem"

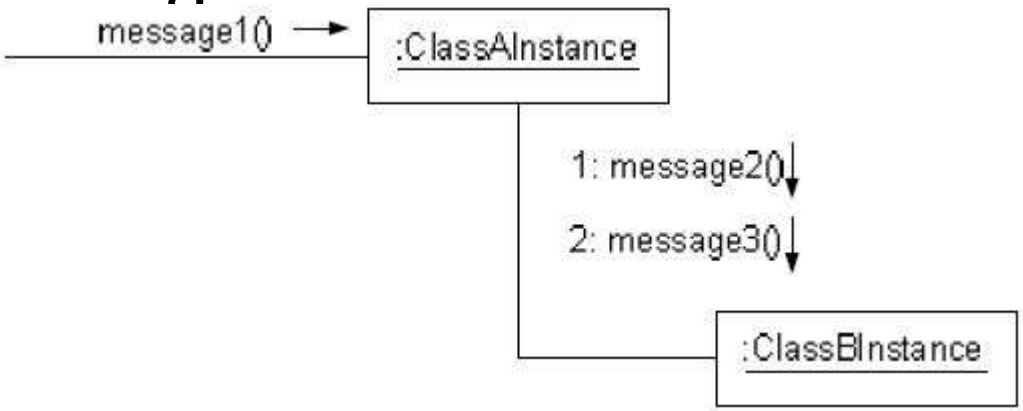
Contract CO2: enterItem

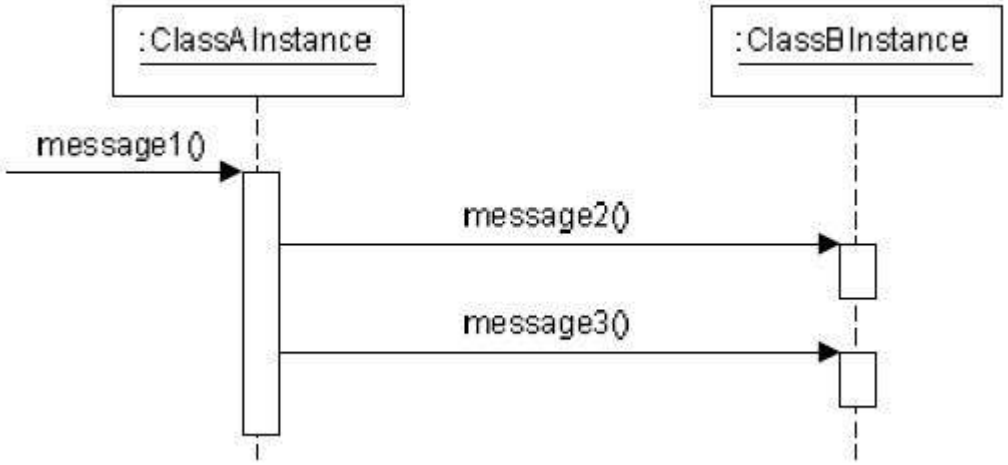
Operation:	enterItem(itemID: ItemID, quantity: integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	– A SalesLineItem instance sli was created *(instance creation)*. – sli was associated with the current Sale *(association formed)*. – sli.quantity became quantity *(attribute modification)*. – sli was associated with a ProductDescription, based on itemID match *(association formed)*.

Interaction diagram

- 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; both can be used to express similar message interactions:
  i. collaboration diagrams
  ii. sequence diagrams
- **Collaboration diagrams** Collaboration diagrams are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case. Along with sequence diagrams, collaboration are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces.

Collaboration Driagram





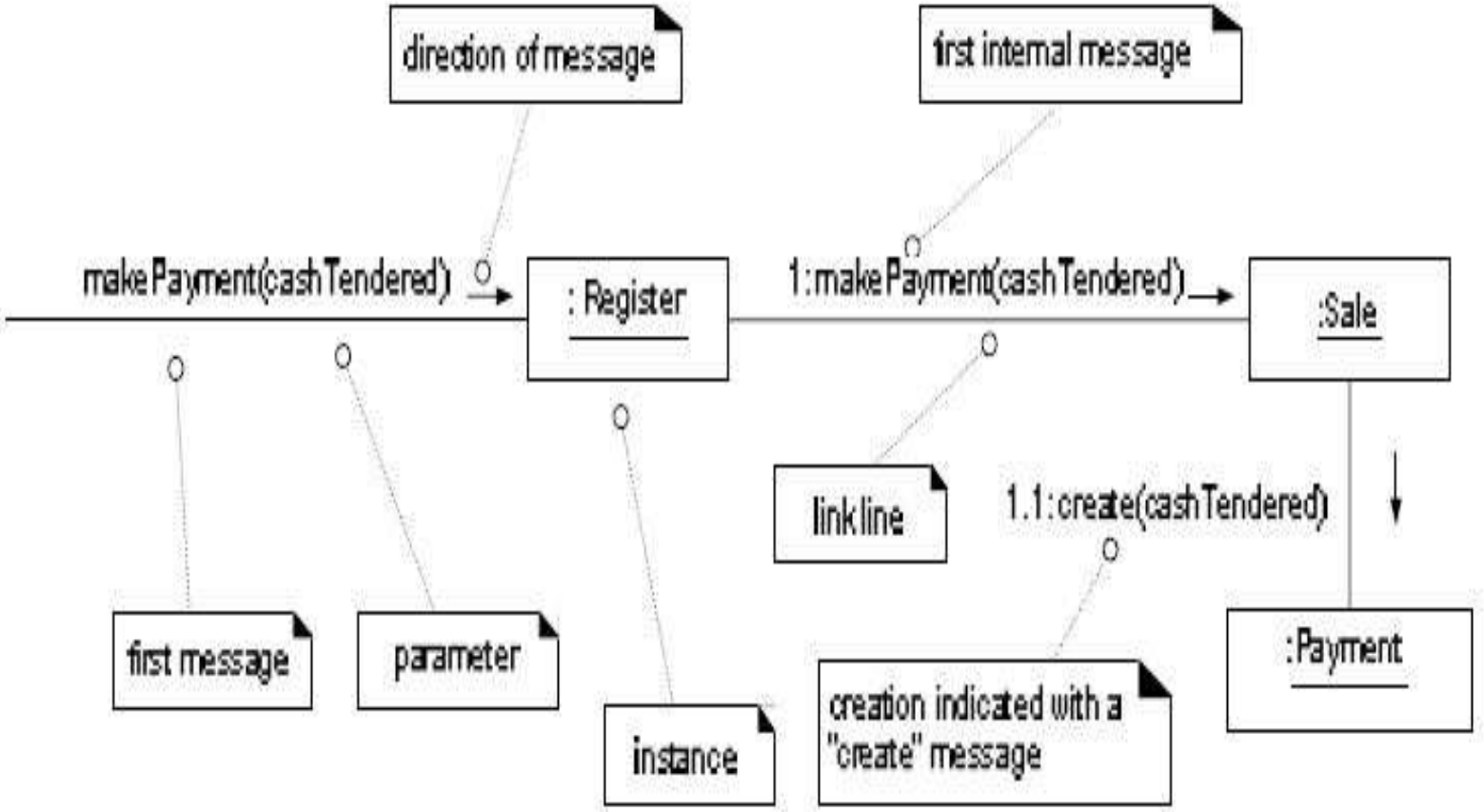
Sequence Diagram

strengths and weaknesses

Comparison

Type	Strengths	Weaknesses
sequence	clearly shows sequence or time ordering of messages simple notation	forced to extend to the right when adding new objects; consumes horizontal space
collaboration	space economical—flexibility to add new objects in two dimensions better to illustrate complex branching, iteration, and concurrent behavior	difficult to see sequence of messages more complex notation

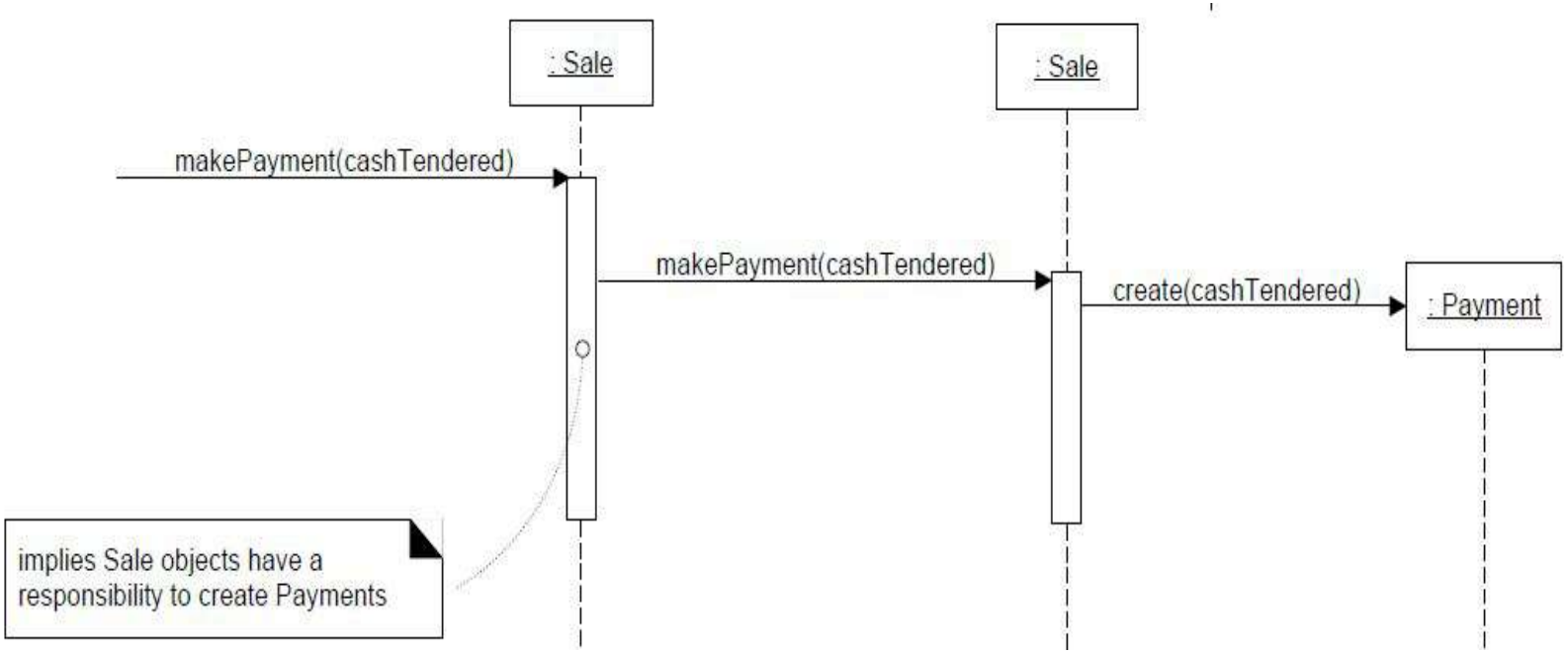
Example Collaboration Diagram: makePayment



The collaboration diagram shown in above Figure is read as follows:

1. The message *makePayment is sent to an instance of a Register. The sender is* not identified.
2. The *Register instance sends the makePayment message to a Sale instance.*
3. The *Sale instance creates an instance of a Payment.*

Example Sequence Diagram: makePayment



The sequence diagram shown in Figure has the same intent as the prior collaboration diagram.

Basic Collaboration Diagram Notation

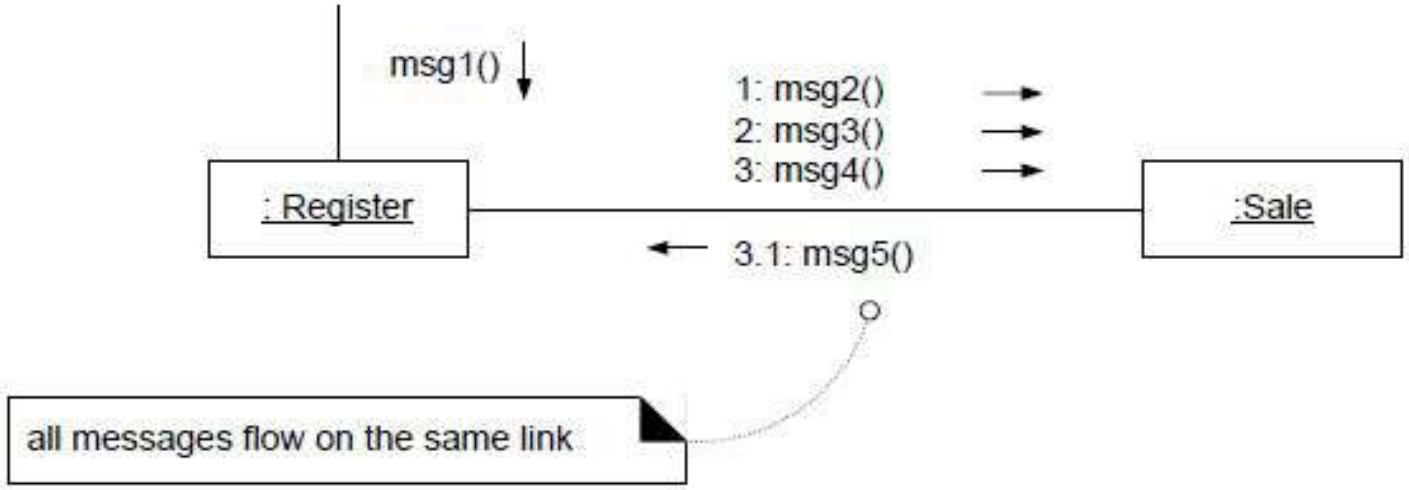
1.Objects: The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.
In the collaboration diagram, objects are utilized in the following ways:

- The object is represented by specifying their name and class.
- It is not mandatory for every class to appear.
- A class may constitute more than one object.
- In the collaboration diagram, firstly, the object is created, and then its class is specified.
- To differentiate one object from another object, it is necessary to name them.

2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.

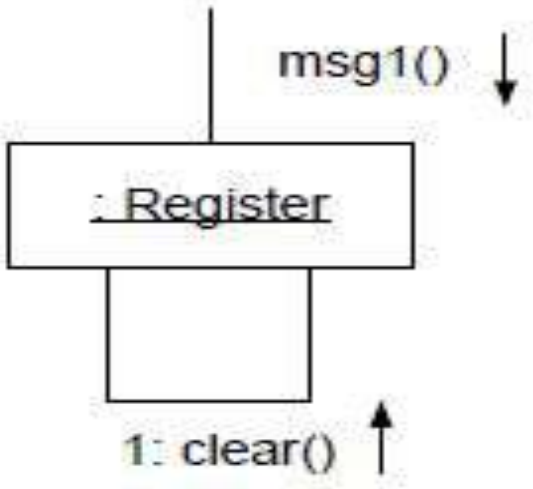
4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

4.1 A sequence number is added to show the sequential order of messages in the current thread of control



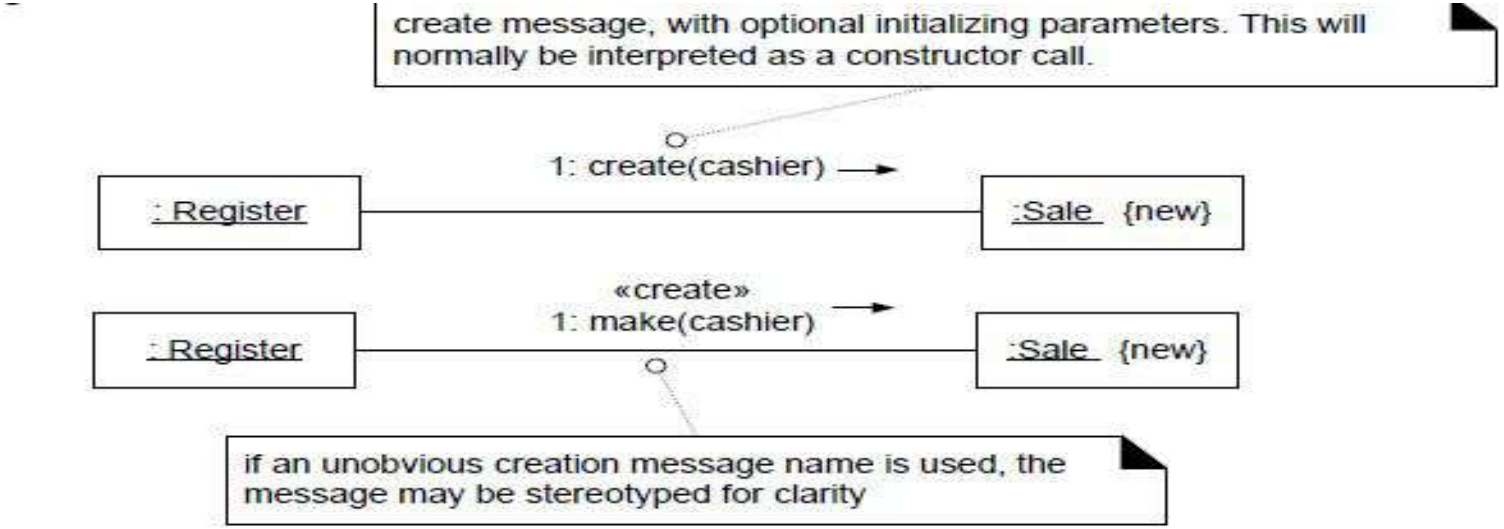
4.2 Messages to "self" or "this"

- *A message can be sent from an object to itself*
  This is illustrated by a link to itself, with messages flowing along the link.



4.3 Creation of Instances

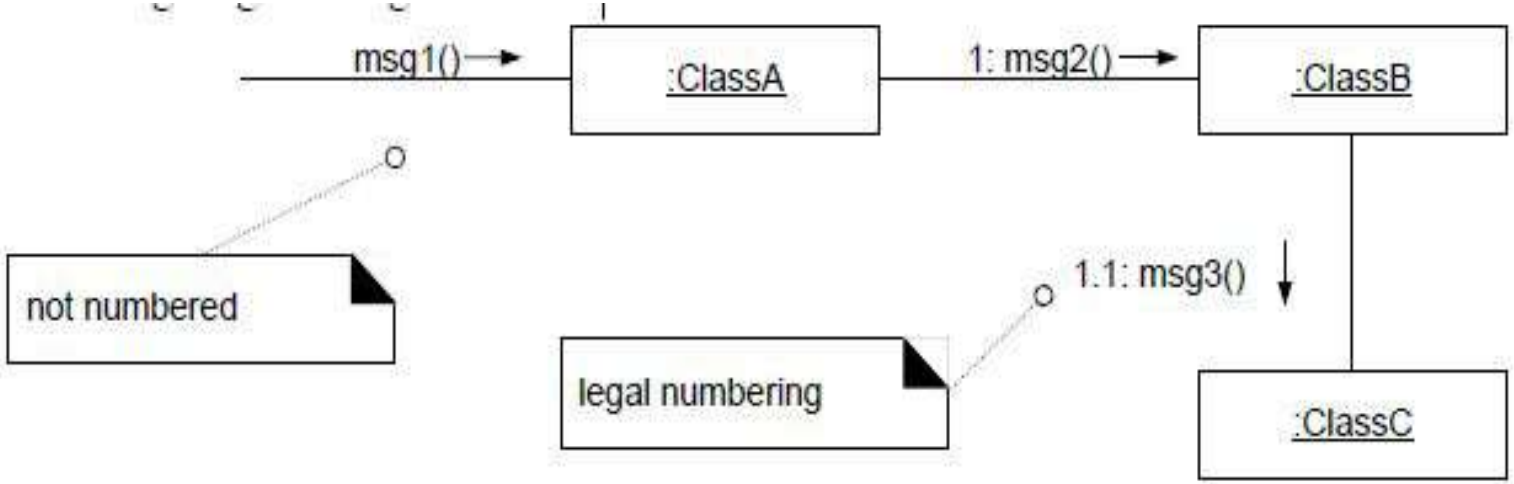
- Any message can be used to create an instance, but there is a convention in the UML to use a message named *create for this purpose. If another (perhaps less* obvious) message name is used, the message may be annotated with a special feature called a UML stereotype, like so: *«create».*
- The *create message may include parameters, indicating the passing of initial* values.



4.4 Message Number Sequencing

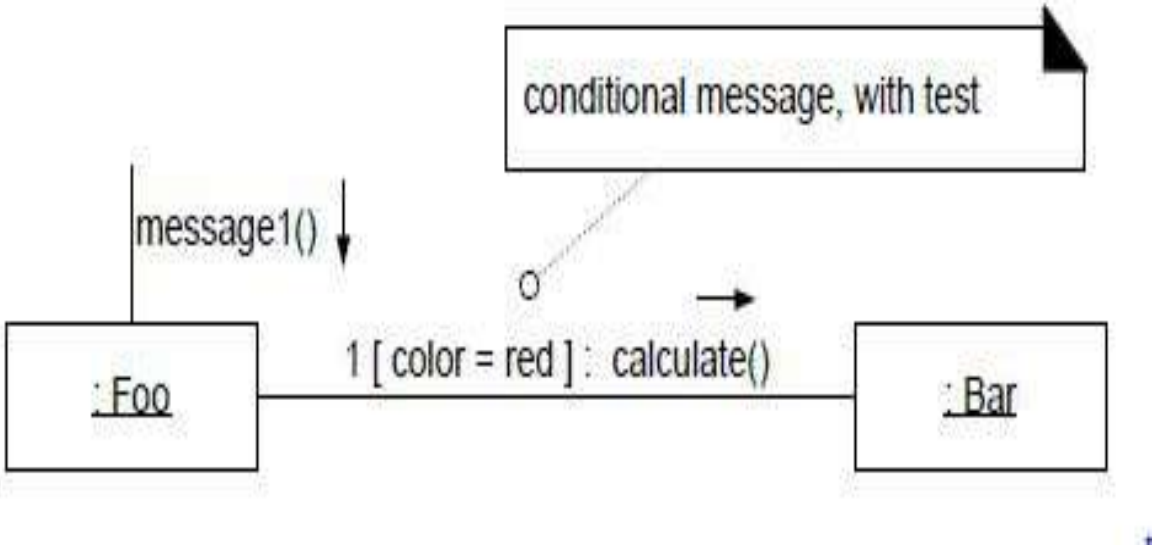
- The order of messages is illustrated with **sequence numbers, as shown in Figure.** The numbering scheme is:

1. The first message is not numbered. Thus,*msg1() is unnumbered.*
2. The order and nesting of subsequent messages is shown with a legal numbering scheme in which nested messages have a number appended to them. Nesting is denoted by prepending the incoming message number to the outgoing message number.

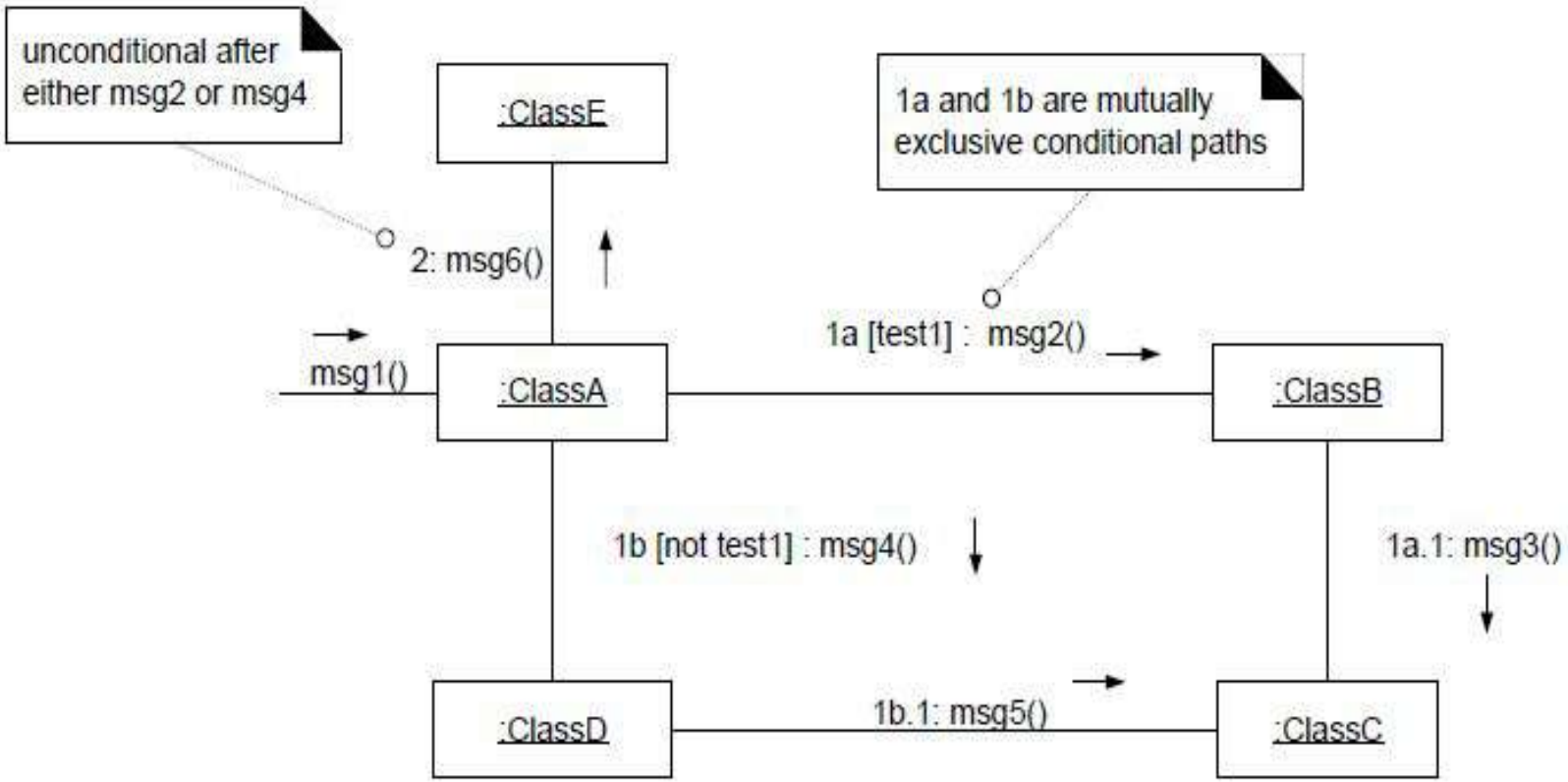


4.5 Conditional Messages

- A conditional message is shown by following a sequence number with a conditional clause in square brackets, similar to an iteration clause. The message is only sent if the clause evaluates to *true.*

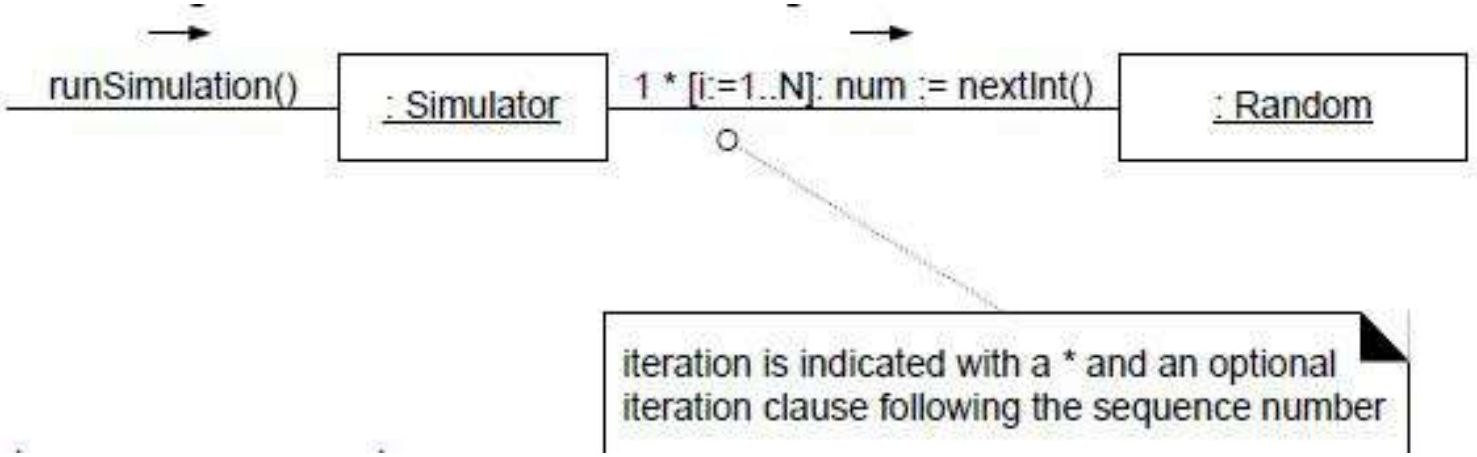


4.6 Mutually Exclusive Conditional Paths



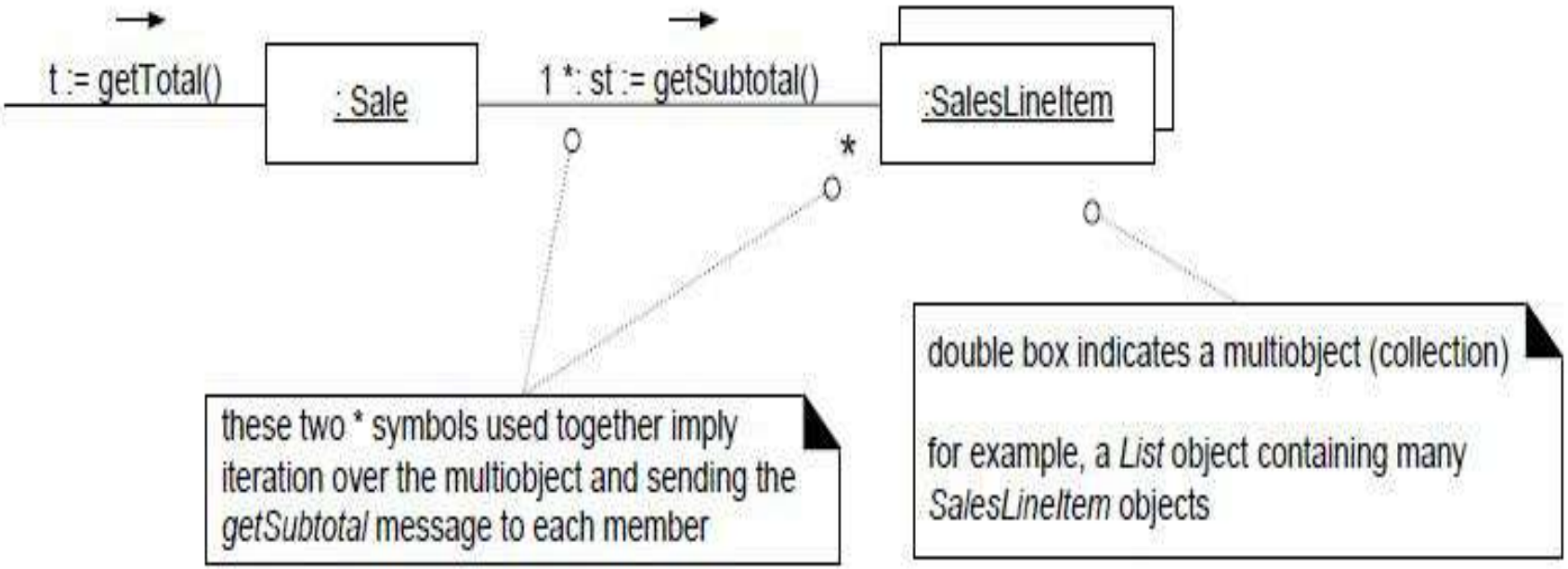
4.7Iteration or Looping

- Iteration notation is shown in Figure. If the details of the iteration clause are not important to the modeler, a simple ’*’ can be used.



4.8 Iteration Over a Collection (Multiobject)

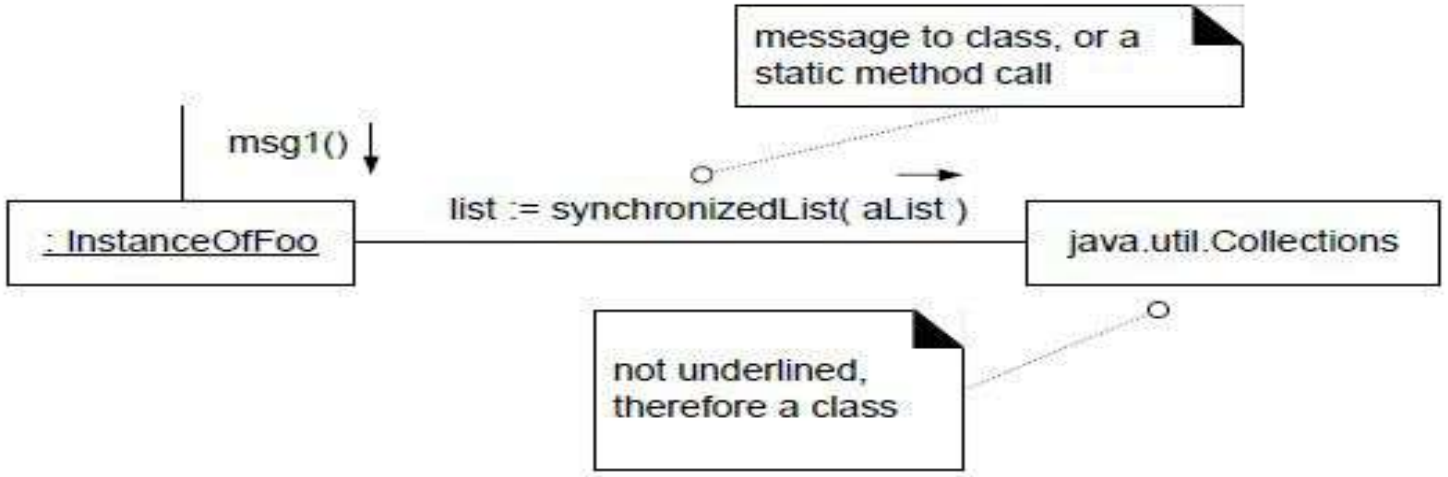
- A common algorithm is to iterate over all members of a collection (such as a list or map), sending a message to each. Often, some kind of iterator object is ultimately used, such as an implementation of *java.util.Iterator or a C++ standard library* iterator. In the UML, the term **multiobject is used to denote a set of** instances.a collection. In collaboration diagrams, this can be summarized as shown in Figure

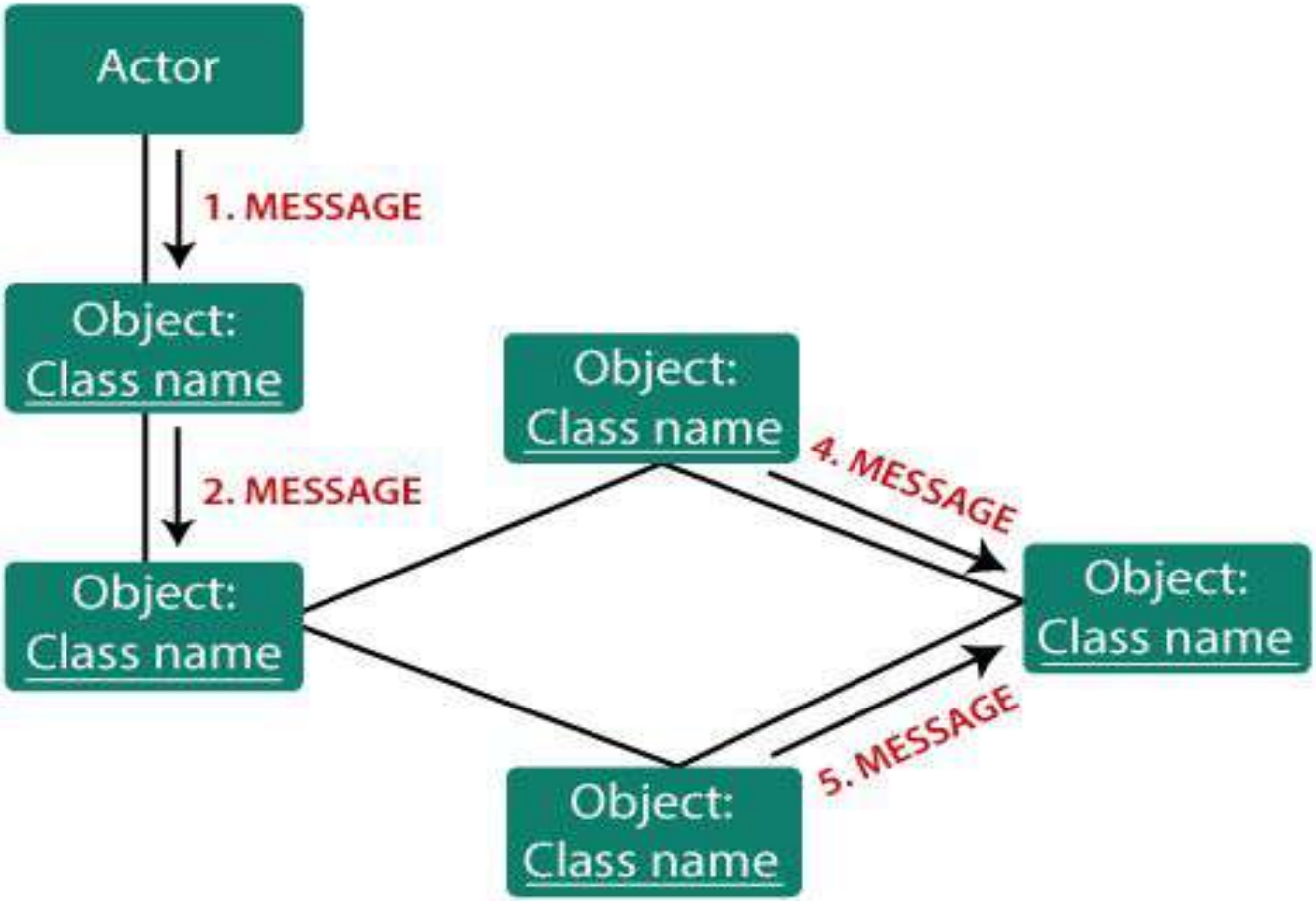


The "*" multiplicity marker at the end of the link is used to indicate that the message is being sent to each element of the collection, rather than being repeatedly sent to the collection object itself.

4.9 Messages to a Class Object

- Messages may be sent to a class itself, rather than an instance, to invoke class or **static methods. A message is shown to a class box whose name is not underlined,** Indicating the message is being sent to a class rather than an instance (see Figure).




Actor
1. MESSAGE
Object:
Class name
2. MESSAGE
Object:
Class name
Object:
Class name
4. MESSAGE
Object:
Class name
5. MESSAGE
Object:
Class name