Unit Eight – Sequence Diagrams

Activity List

		Completed
	1. Read the Activity 8.1 material, ***Introduction to Interaction Sequence Diagrams***, over page	☐
	2. Read the Activity 8.2 material, ***Sequence Diagram Example and Notation,*** over page	☐
	3. Complete the Activity 8.3 exercise, ***Reading a Sequence Diagram,*** over page	☐
	4. Read the activity 8.4 material, ***Further Sequence Diagrams,*** over page	☐
	5. Complete the Activity 8.5, ***Sequence Diagram Exercises,*** over page	☐

Learning Objectives

At the end of this unit, you should be able to:

1. Explain the content and purpose of a UML Sequence Diagram
2. Describe the notation used in a UML Sequence Diagram
3. Apply the notation to build UML Sequence Diagrams

Activity 8.1: Introduction to Interaction Sequence Diagrams

In unit 7, we looked at the basic ideas of object interaction, and learned how to draw UML communication diagrams. Communication diagrams, in the approach to development that we are following, are most often used as an analysis technique, though they can also be used in design. However, they seem to be decreasing in importance, and the primary technique for specifying object interaction is now the rather more complex *sequence diagram*.

Sequence diagrams and communication diagrams are both in the group of UML diagrams known as interaction diagrams. There are two more interaction diagrams: timing diagrams and interaction overview diagrams. We do not cover these in detail on the module, but you have read about them in Bennett et al. (2010), sections 9.5 and 9.6.

Sequence diagrams used to suffer from limitations in what they could easily express, but the notation has been substantially revised in UML 2.0 and now provides a very flexible way of describing the message passing used in a scenario. We expect sequence diagrams to increase in importance at the expense of communication diagrams. Nevertheless, the two techniques do have different emphases, and there is a useful role for both.

In this unit's material, we shall learn how to draw sequence diagrams, and see one of their uses in design, the representation of interactions with a user interface.

First, then, what is the difference between sequence diagrams (sometimes known as *interaction sequence diagrams*) and communication diagrams?

We have seen that communication diagrams closely reflect the static structure of the required system, as they show links between objects. Links are needed to enable messages to be passed, and will correspond to associations on the class diagram. An important use of communication diagrams in analysis is to confirm the required classes and help identify their operations and associations. We also learned how to derive class diagrams from communication diagrams.

Sequence diagrams have a different focus. While they still show objects (lifelines), actors and messages, they do not show links. Their emphasis is on the sequence of messages, rather than on the static structure of the system. On a communication diagram, the sequence of messages can be determined from their message numbers, and the numbering scheme allows great flexibility in showing message ordering, including the use of concurrent messages. In sequence diagrams, messages are not usually numbered, and the ordering is shown by the arrangement of messages on the page. This gives a visual emphasis to the time-ordering of messages. They also have notation for showing time constraints, and so are very useful in applications when these, or concurrency, are required. Nevertheless, there may be occasions when

complex interactions, including those showing concurrency are easier to draw in the less restricted layout of a communication diagram.

USDP and similar processes tend to use communication diagrams in analysis and sequence diagrams for design. However, either may be used at either stage.

As an analysis technique, sequence diagrams are used in some development methods as an alternative to communication diagrams. As not all approaches use control and boundary classes, these may be shown or not as required. Sequence diagrams used in analysis often omit details of notation such as the return types and arguments of operation signatures.

As part of design, sequence diagrams are drawn or extended to show design detail. In design, sequence diagrams are useful to show the working of the user interface. They will show user interactions with actual user interface classes, rather than analysis boundary classes. At this stage the full notation is used.

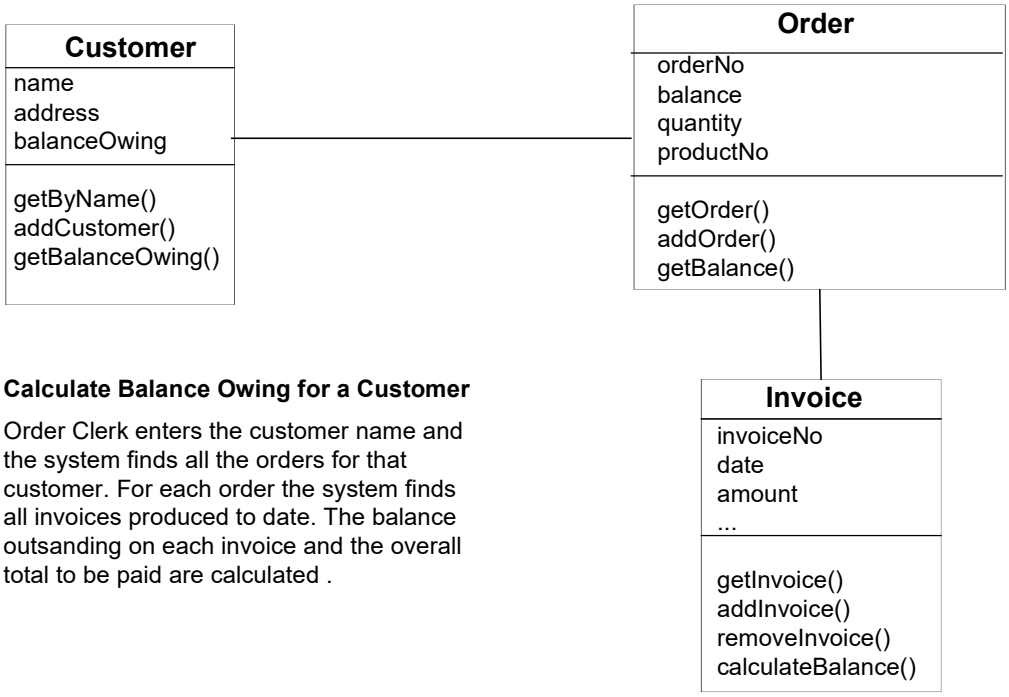
In the next section of reading, we shall see how to draw sequence diagrams.

Activity 8.2: Sequence Diagram Example and Notation

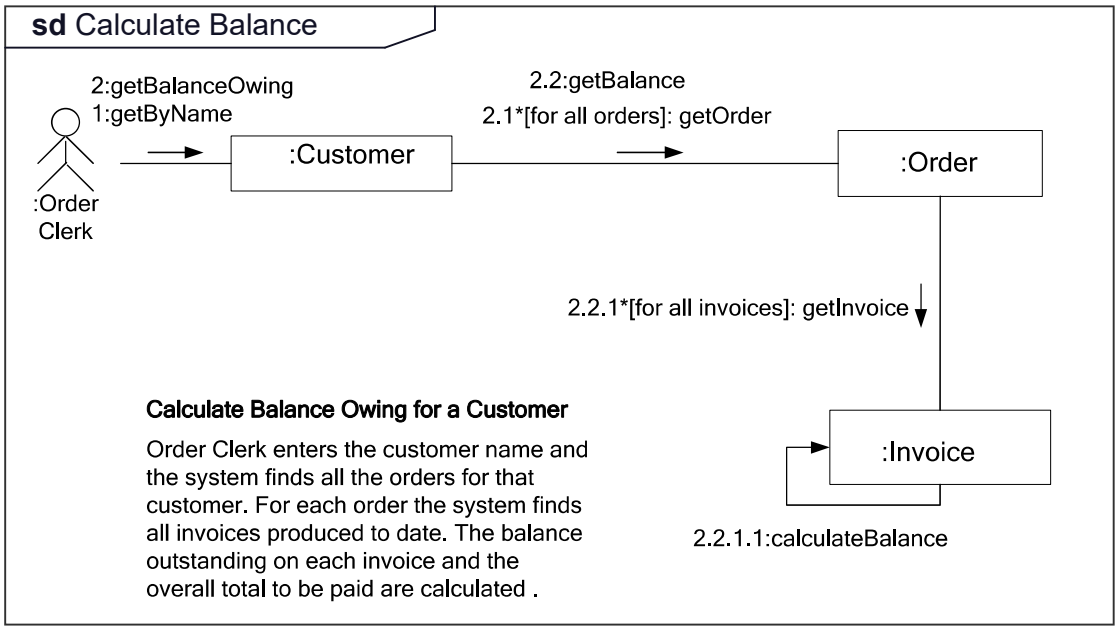
In this section, we shall use an example to look at the main features of sequence diagram notation. Below is a small class diagram and a short scenario. It is concerned with orders and invoices, and the actor would be an Order Clerk. Several invoices can be issued against each order.

Calculate Balance Owing for a Customer

Order Clerk enters the customer name and the system finds all the orders for that customer. For each order, the system finds all invoices produced to date. The balance outstanding on each invoice and the overall total to be paid are calculated .

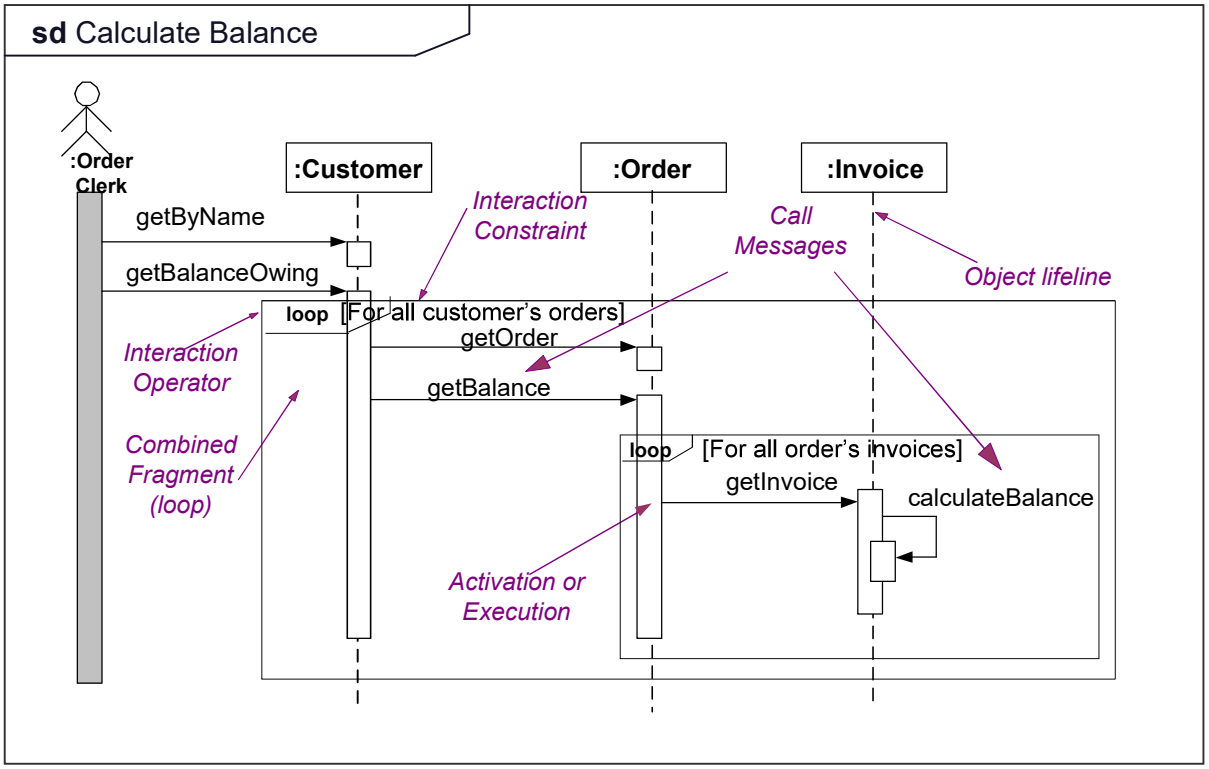


If we were to draw a communication diagram, we might produce the following diagram. Note the iteration of getOrder and getInvoice.



However, this doesn't totally reflect what we want. We need to get the balance of each order, so we would really like each instance of message 2.1 to be followed by an instance of message 2.2. This is not easy to show in communication diagram notation, but, as we shall see, a sequence diagram allows us to represent it clearly and precisely.

Here is a sequence diagram for the same scenario. Some important parts of the notation are identified.

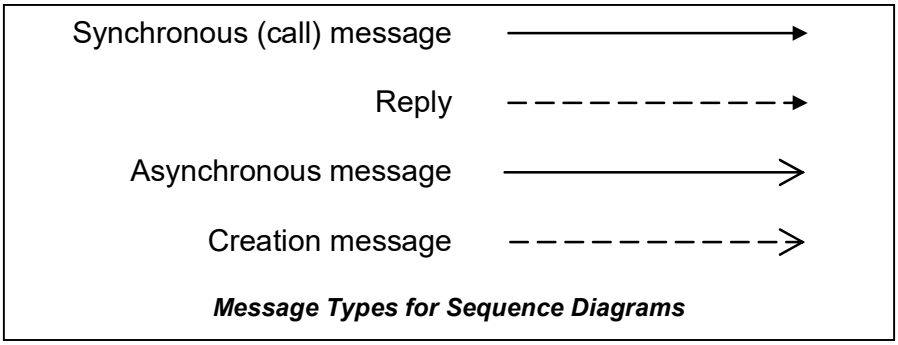


Here we have *lifelines* representing three objects. We have seen lifelines on communication diagrams, but here, an actual line is drawn. It represents a time dimension. Sequence diagrams usually show time in a vertical dimension. The arrows represent messages. *Activations* show when an object is active, though it may not actually be processing: it could be waiting for control to be returned.

Sequence diagrams are drawn in a frame and labelled, in the same way as communication diagrams. However, you will see that parts of the diagram are also in frames, which can be nested. These part-diagrams are called *interaction fragments*, and they are used in various ways. They create sections of functionality that may be combined as building blocks, and they are the means of showing various types of structure, such as loops and branching. An interaction fragment can have an *interaction operator* to show what construct is used, and an *interaction constraint* to govern its operation. Here, we have some nested *loops*.

Note that all the messages shown here are *call* messages. These are synchronous messages that invoke an operation on an object, which may be the sending object or another object. For example, :Customer sends a message to call the getBalance operation of order. Invoice calls its own calculateBalance operation. As these are synchronous messages, the operation that sends them cannot continue until the called operation returns control.

In fact, sequence diagrams differentiate between several types of message. The most common are shown below.



Message Types for Sequence Diagrams

Here we see the four main types of message arrow that are used in sequence diagrams.

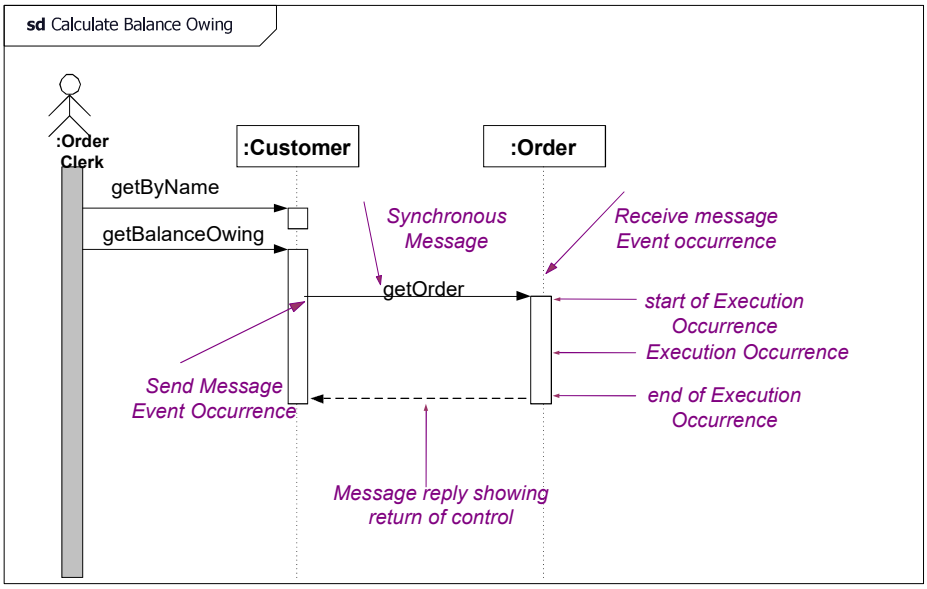
You will remember from our topic on communication diagrams that UML distinguishes between synchronous and asynchronous messages. Synchronous messages are those in which the sending objects awaits a reply, which will return control from the object that received the message, before it can continue with its work. With asynchronous messages, the object sends the message and continues immediately: there is no reply.

You can see from the table that there is a separate type of arrow to show replies. We shall look at one final type of message in this section: the creation message. This is used exclusively for the creation of new objects.

(Note that this is a change from UML 1.x notation. In UML 1.x, there is no separate arrow for creation messages, and a reply is shown as - - - - - - ->.)

Replies, activation and focus of control

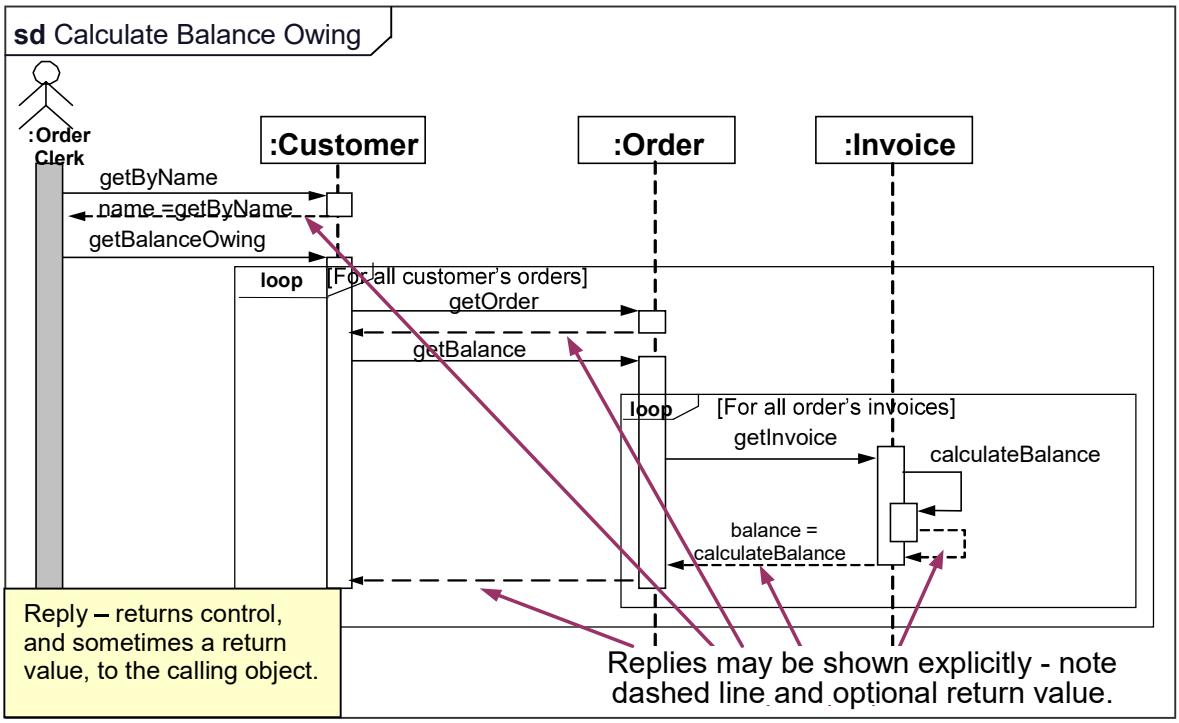
The next diagram is just part of our previous example, and shows a reply message in use. It also indicates the *events* that take place as messages are sent and the called operations execute. You are familiar with the concept of events from your study of state machines. Note how the execution occurrence (or activation) starts when the message is received and ends when a reply is sent to return control to the calling object.



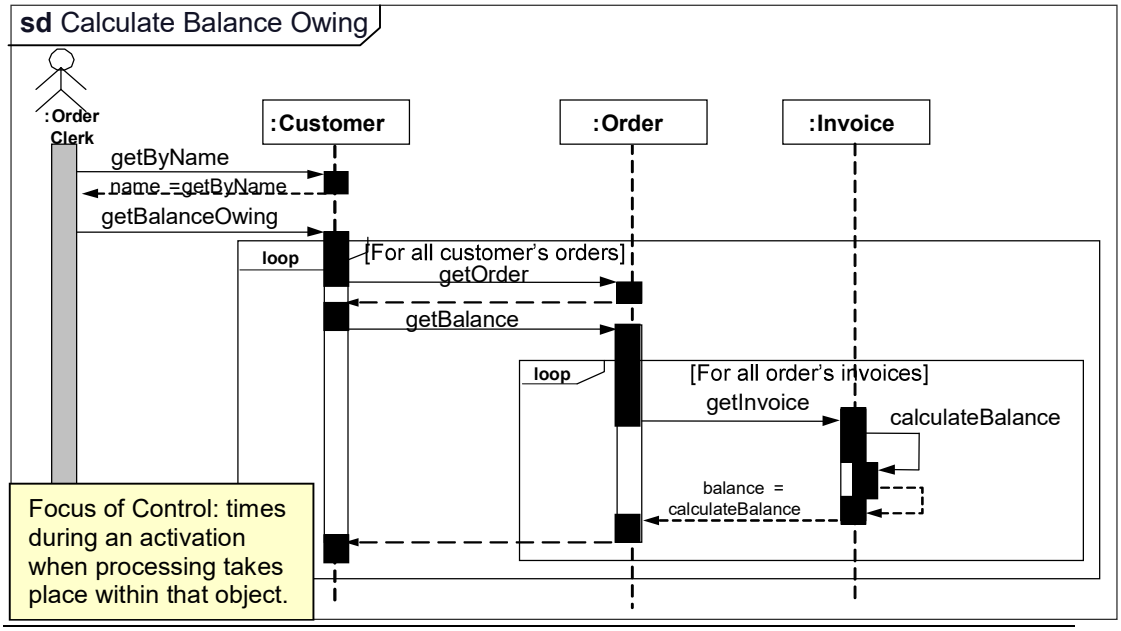
Showing replies is optional – where a synchronous message has been sent, they can be assumed to take place when an activation ends. Replies may simply return control to the calling object, but they may also return a value. In the next version of our example, we show all the possible replies. Note the syntax for showing return values, e.g.

balance=calculateBalance

where calculateBalance is the operation returning the value.



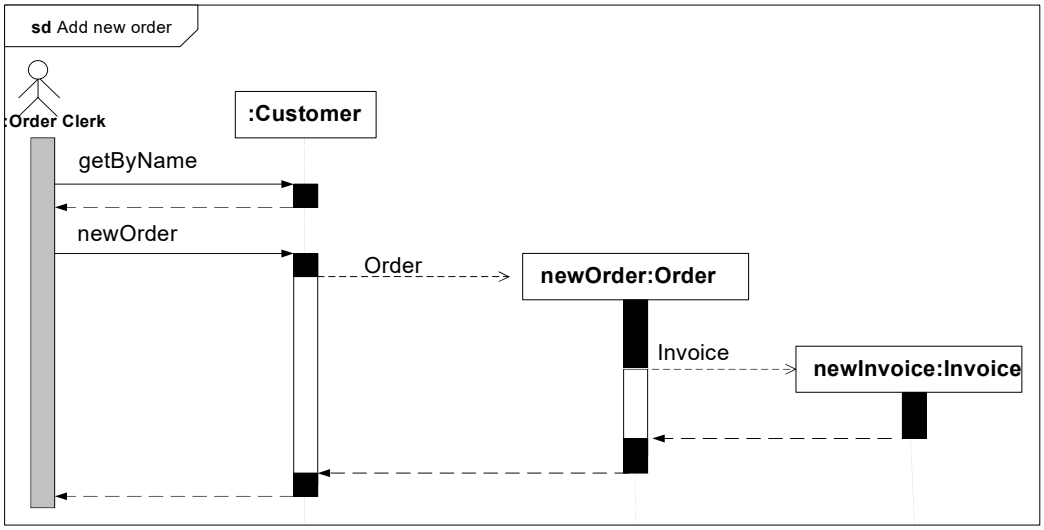
We noted that during an activation, an object may be actively processing or waiting for a reply. When it is actively processing, we say that it has the focus of control. This may be shown explicitly. In the diagram below, filled rectangles show when an object has control; unfilled rectangles show when it is waiting for a return. Notice the relationship between calls and returns and the focus of control: when an object sends a call message, it loses control. When the reply is received, it regains control and starts actively processing again. Where an object calls one of its own operations, the original operation is blocked from continuing (and so shown as an unfilled rectangle) until the nested call returns. Here, calculateBalance illustrates this.



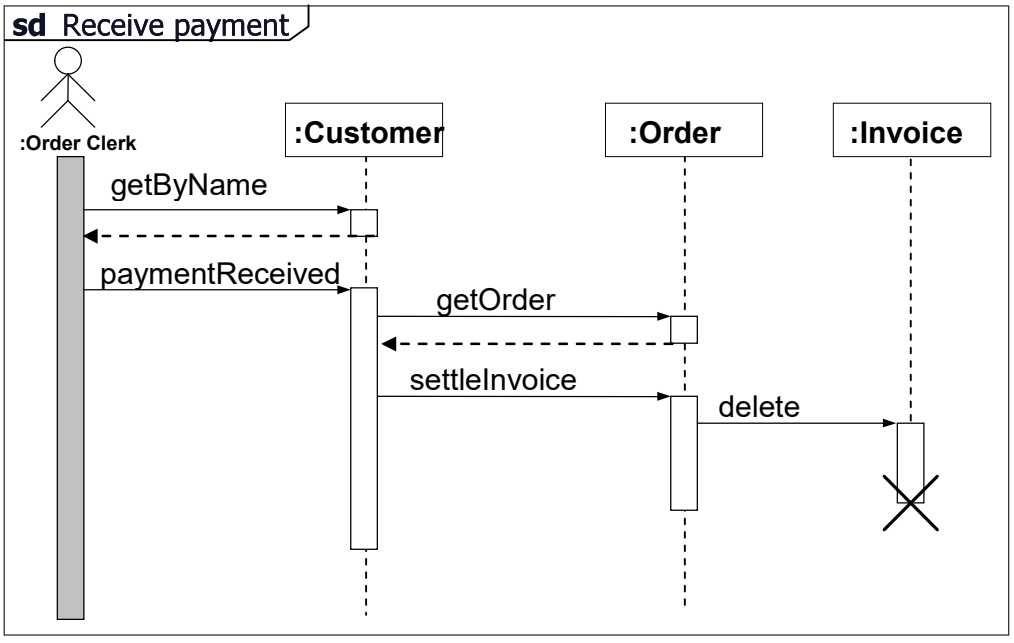
Creating and Destroying Objects

The next two examples show how objects may be created and destroyed. Note the use of the special creation message, which is drawn going into the lifeline's rectangle, rather than the line below. When created, an object becomes active, returning control when its creating operation (constructor) completes. It is common for constructor operations to have the same name as the class whose instances they create.

Here, we have given a name to the new lifelines rather than leaving them as anonymous instances: this is optional and not part of the notation for creating objects.

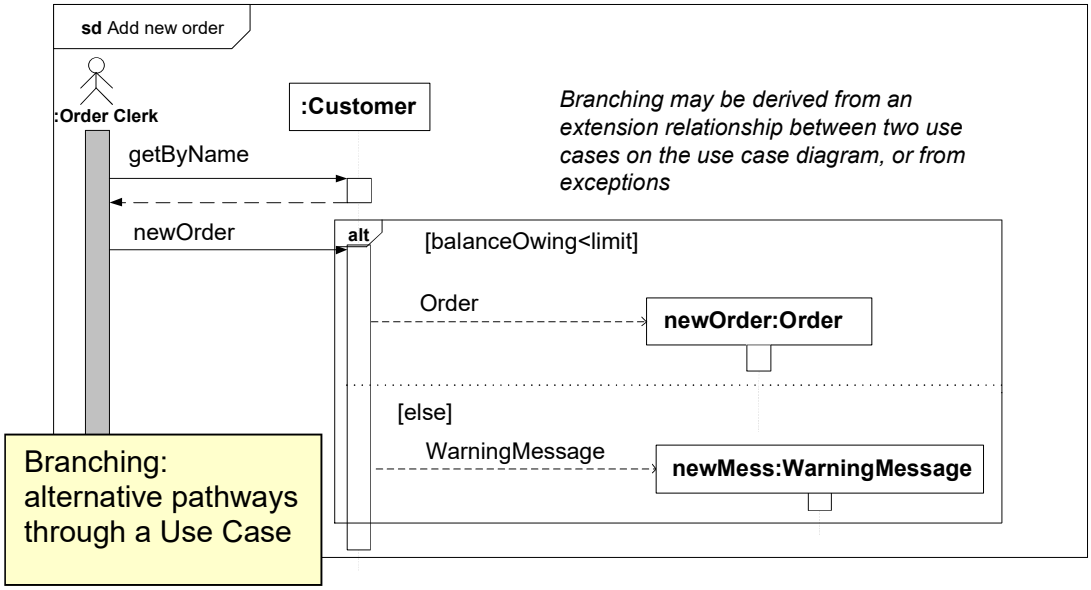


When an object is destroyed, a large cross is placed at the end of its lifeline. Notice that its dashed line does not continue down the page.



Branching

Branching allows us to show alternative courses of action on a diagram. These might be derived from an extension relationship between two use cases on the use case diagram, or from alternative or exceptional scenarios. To show branching, we use a combined fragment with the interaction operator ***alt.***

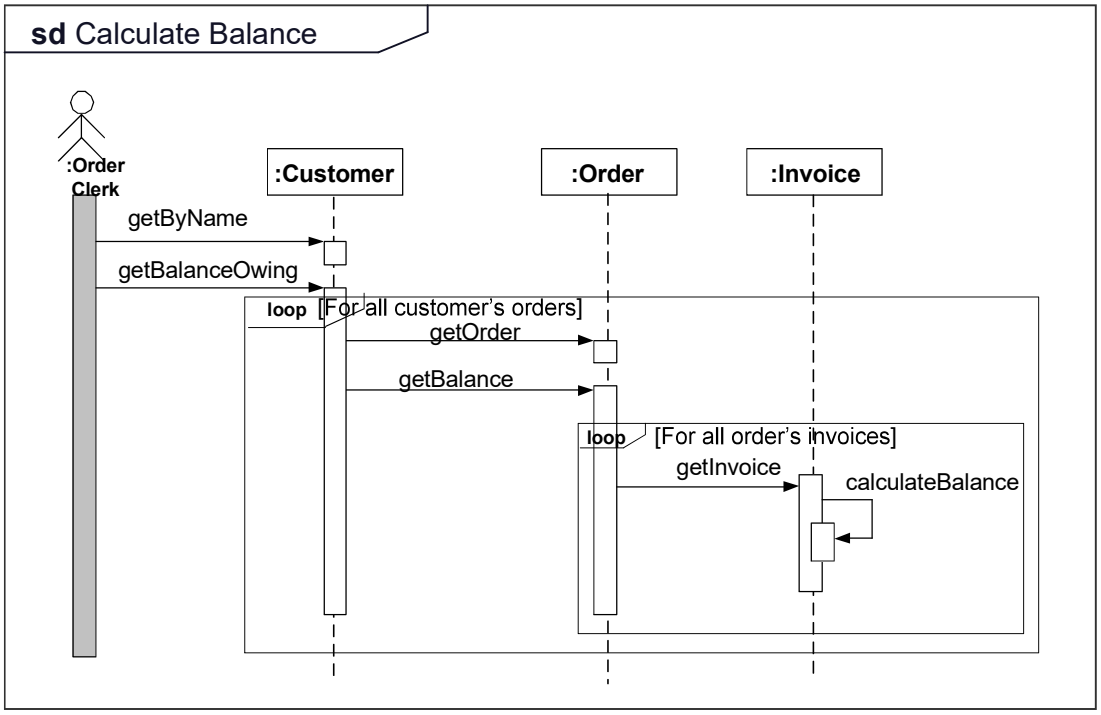


In the example above, we have two possibilities. Which is taken depends on the condition, shown in square brackets, [balanceOwing<limit]. This is examining a value of the :Customer's balanceOwing attribute. The condition is an *interaction constraint.* Other points to notice about the notation: the frame is divided into two, and which part is executed depends on the result of testing the interaction constraint. If it is true, a new order is created. If not – note the [else] condition – a WarningMessage is created.

An **alt** always has at least two compartments showing alternative interactions, and each one is governed by a constraint. If it's a case of either doing something or doing nothing, there's a different interaction operator, **opt**, which means that the interactions shown are optional. This only needs one compartment.

Iteration

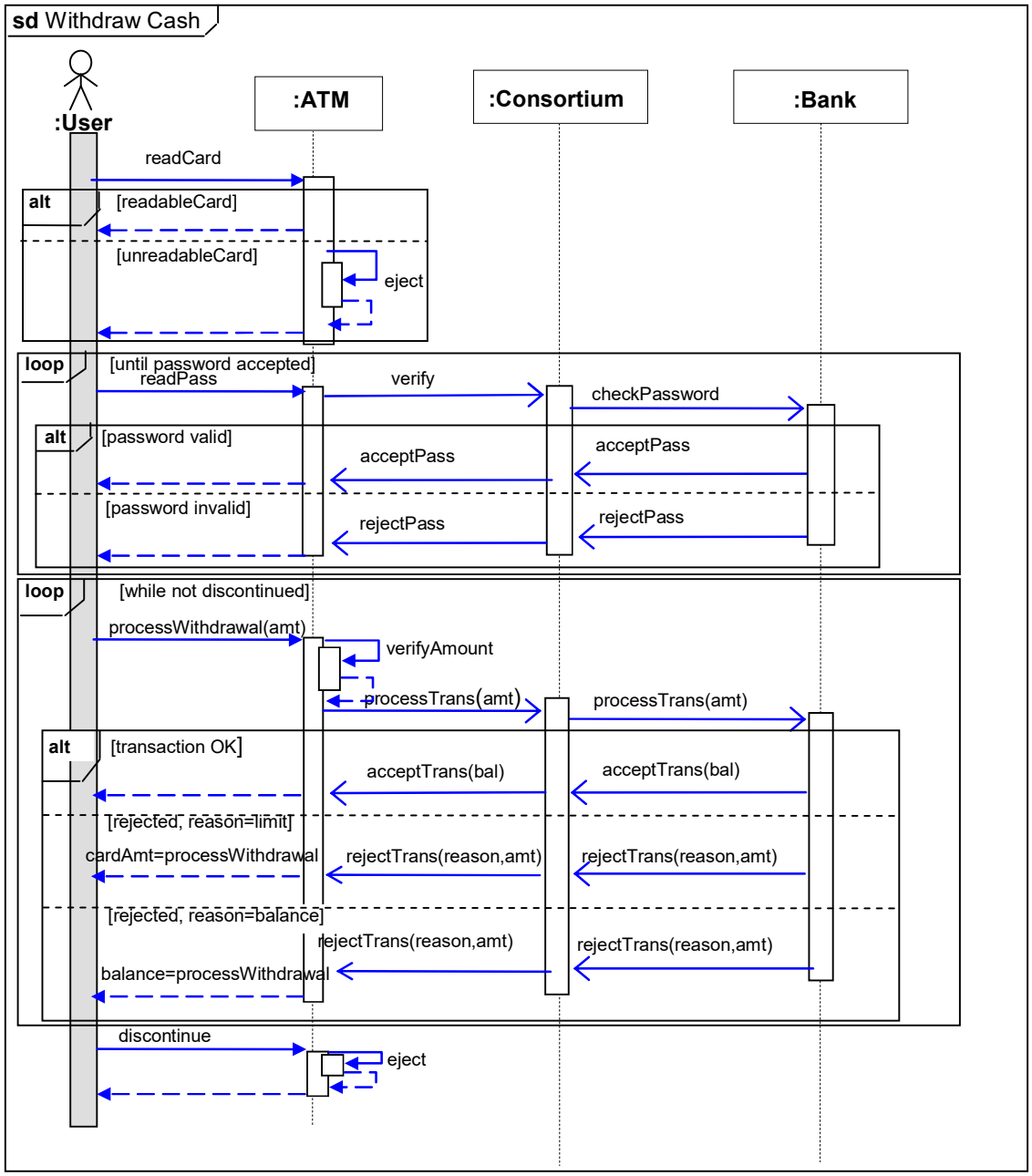
Let us return to our original example to look at another important construction: iteration. If we wish a part of an interaction to be repeated, we use a combined fragment with the interaction operator *loop*, and the condition that controls the loop as the interaction constraint. Everything inside the loop is repeated according to the interaction constraint.



Loops and other combined fragments can be nested to any depth. However, too much complexity can make the diagram unreadable. It would be useful to have a way of simplifying our diagrams.

Activity 8.3: Reading a Sequence Diagram

Study the ATM sequence diagram below, then answer the questions that follow.



a. What *synchronous* messages are sent to :ATM?

b. What *asynchronous* messages are sent to :ATM?

c. What are the possible outcomes of sending a withdrawal (i.e. processTrans(amt)) to the :Consortium?

d. What causes the second loop to stop iterating?

e. It is not obligatory to show the parameters of a message. If message verify had a parameter, what do you think it would be? Rewrite the message to show the parameter, then do the same for any other messages that you think should also include this parameter.

f. Which of the types of message that we have seen is not shown on this diagram?

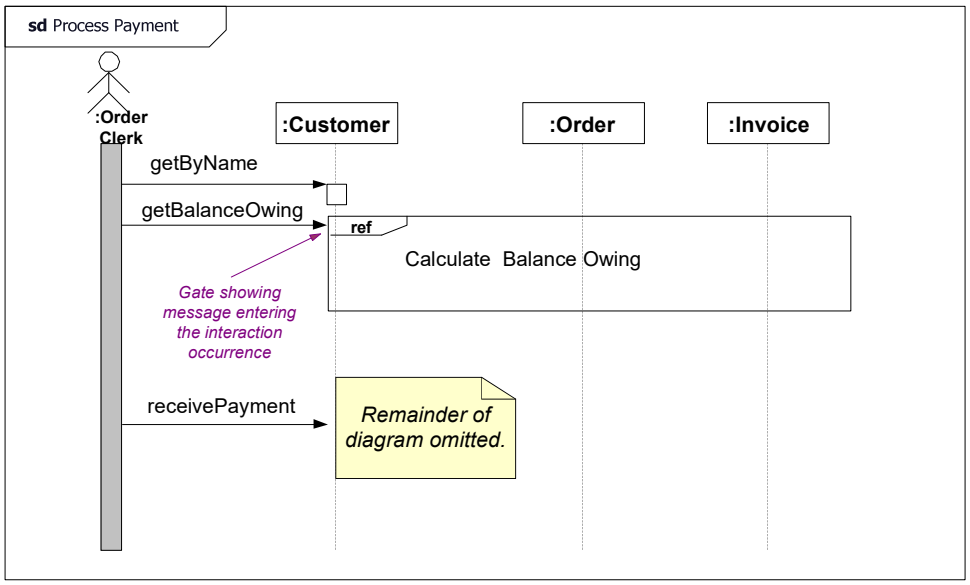
Example solutions can be found in the Appendices.

Activity 8.4: Further Sequence Diagrams

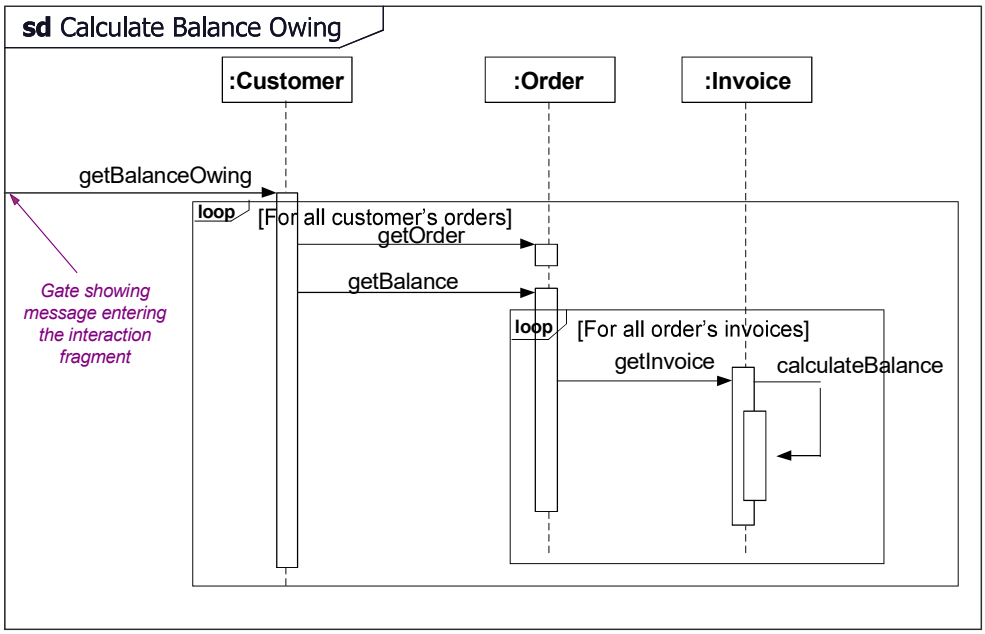
Managing Sequence Diagrams

Our examples have been fairly simple, but some sequence diagrams become very complex. It might even be that part of the logic is used in more than one place – perhaps it represents an «include». In these cases, it can be useful to hide the details of parts of the interaction. They can then be shown in full on a separate diagram.

Let us assume that finding the balance owing can also be part of a larger scenario, for processing payments. It would be useful simply to be able to slot in the functionality for finding balance. We can do it like this.



The ***ref*** tells us that this interaction, Calculate Balance Owing, is defined elsewhere. And here it is….



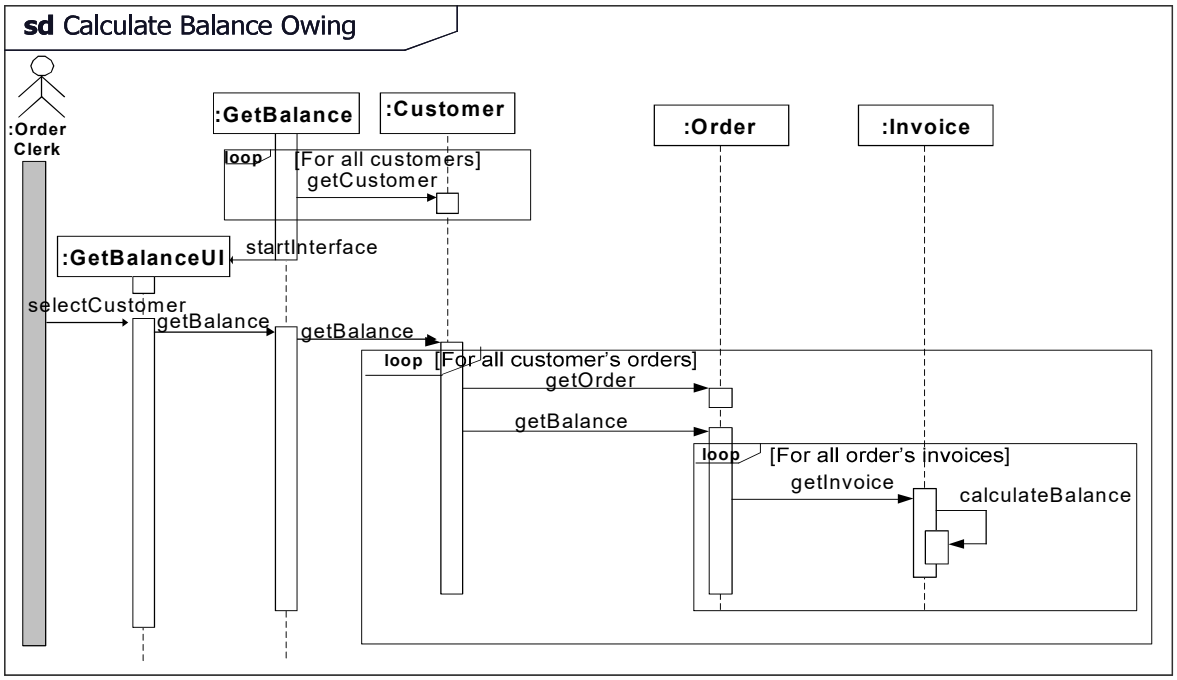
Boundary and Control Objects

You might have wondered why our diagrams do not include boundary and control objects. These can be used on sequence diagrams if required, especially if used at the analysis stage, in which case they are used in the same way as in communication diagrams.

However, not all analysts or all development methods use them. In design, sequence diagrams can show the actual interface objects (windows, buttons…) rather than a single boundary object

For simplicity, **boundary and control objects have been omitted from these examples.**

The following diagram shows our original example with boundary and control objects added. In this version, the interface will display a list of customers and the :OrderClerk will choose one.



Reading:

Read Bennett et al. (2010) section 9.3, or review it if you have already read it. This section covers some additional notation.

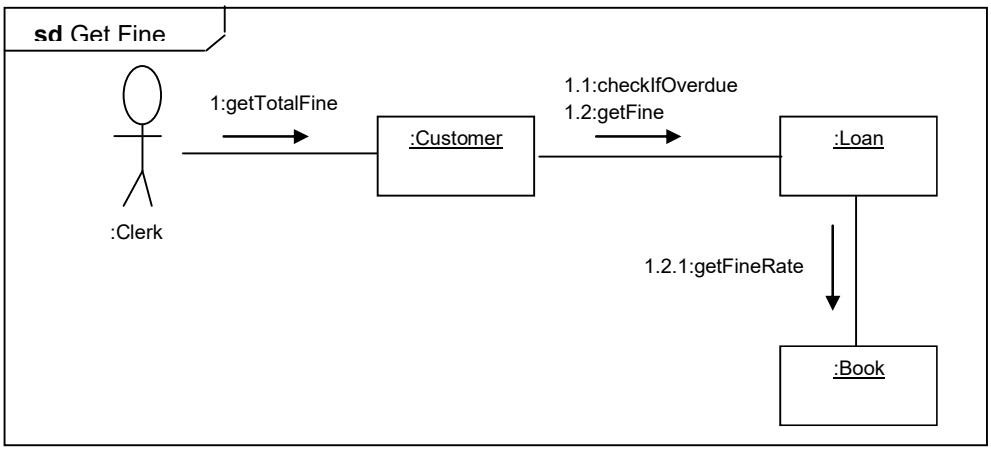
Summary:

The following table sums up what we have learned about the similarities and differences between sequence diagrams and communication diagrams.

Communication Diagrams	Sequence Diagrams
Show the object interactions needed by a scenario.	Show the object interactions needed by a scenario.
Focus is on structure (links) as well as messages.	Focus is on time ordering.
Show lifelines and messages.	Show lifelines and messages.
Show links between objects.	Don't show links.
Don't show activation or focus of control.	Can show activation and focus of control explicitly.
Show message numbers.	Usually don't show message numbers.
Don't show returns explicitly	Do show returns explicitly
Most commonly used in analysis	Used in analysis or design, but especially in design.
A typical use in design is to help work out how object ID's are used.	A typical use in design is to show interaction with a user interface.

Activity 8.5: Sequence Diagram Exercises

1. Below is a communication diagram from unit 7, concerning library fines. Convert it into a sequence diagram. You do not need to add boundary and control classes, but show activations, focus of control, and replies.



2.

a. A netball club has a number of teams, and keeps track of which players belong to which team. Draw a sequence diagram for a scenario concerned with moving a player from one team to another.

   The actor (a coach) indicates the player and the new team in the first input message.
   The system disconnects the player from the current team and attaches them to the new team.

   Show activations and focus of control, and include lifelines for a control object and a boundary object, a :Player and two named lifelines for teams. (Hint: you can use *self* here, as you saw in communication diagrams.)

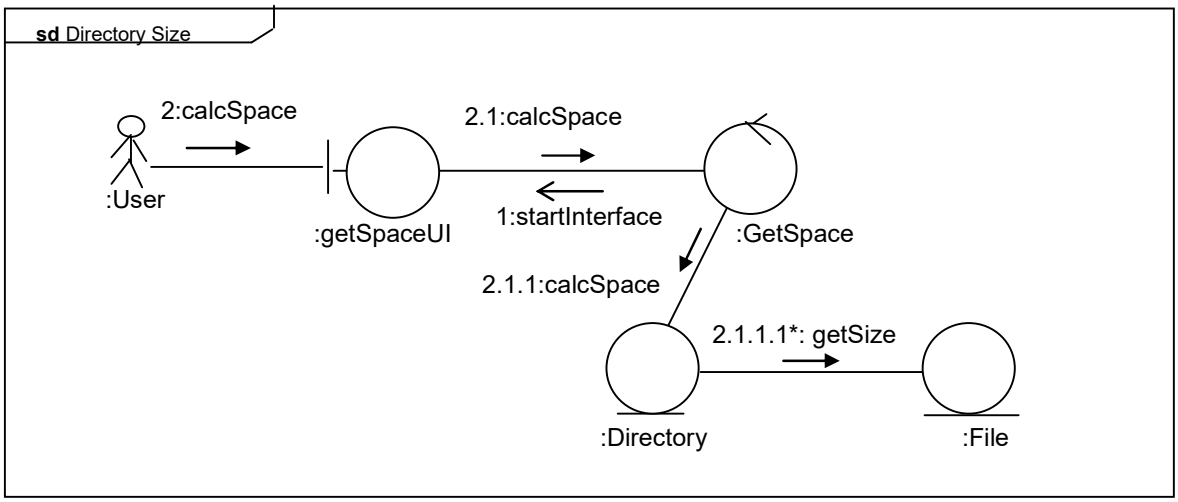
b. Extend the diagram so that the system presents the coach with a list of players to choose from. You will need to use a **loop**.

c. Change your diagram so that before a player is moved, the system checks that there is space in the new team for the player. If there are already eight or more players in the new team, an error message should be created. If not, the player may be moved. (There are seven players in a netball team, so eight allows for a full team and one reserve.)

3. Adapt your diagram from exercise 1 to reflect a changed scenario in which several books are on loan, and they may or may not be overdue. Hint: you

will need to use two different interaction operators. If it seems difficult, deal with the case of several books first, then look at the possibility that they may not be overdue. You might want to remind yourself of the difference between alt and opt.

4. In exercise 7.6, you drew a communication diagram for a simple file management system, in which there are objects representing files and directories. Directories contain files, but not other directories. The diagram (given below) modelled a scenario that calculates the total space used by all the files in a specified directory, in response to a user request. Draw an equivalent sequence diagram. Include the boundary and control objects.



Example solutions can be found in the Appendices.