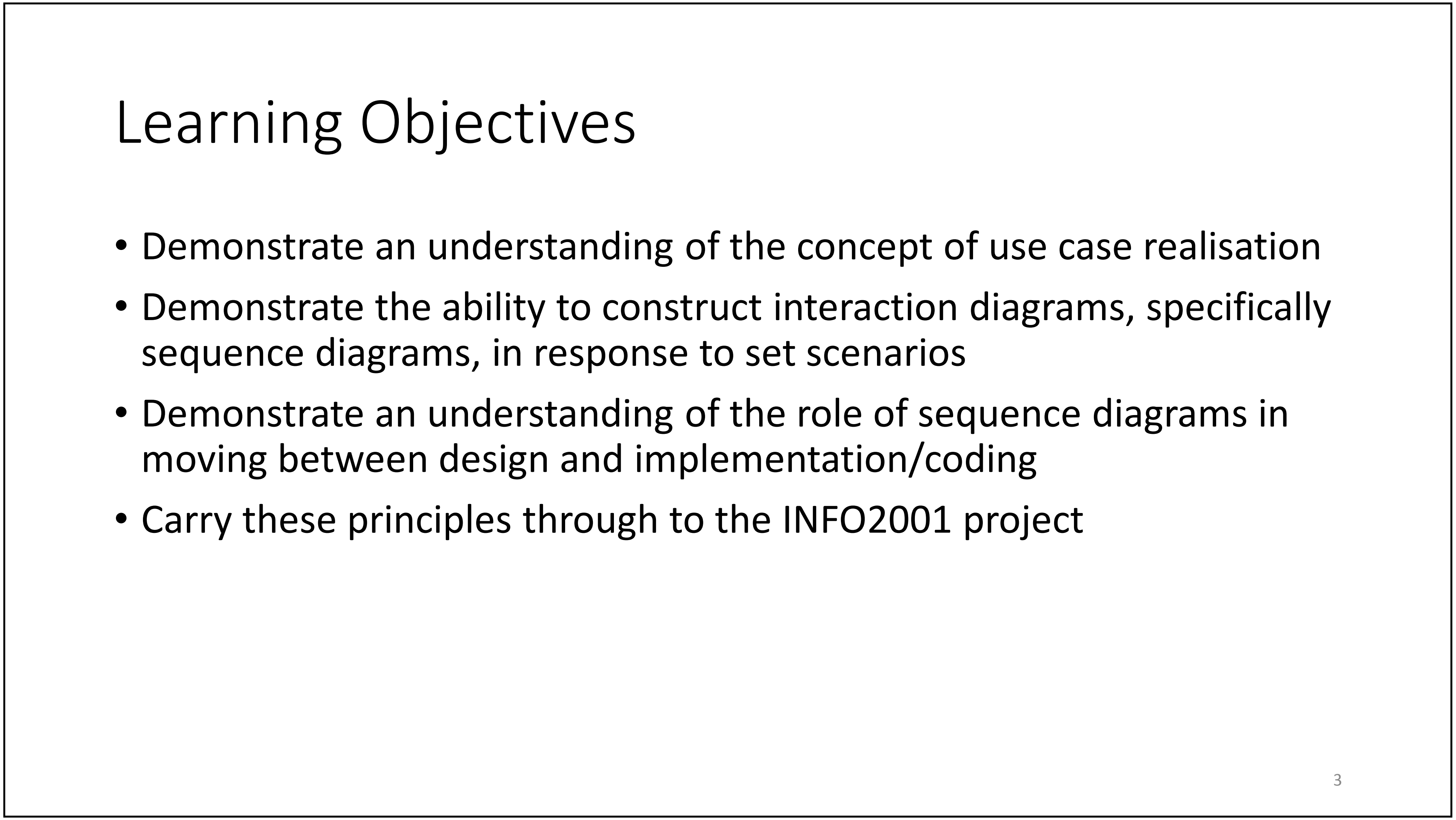
INFO2001
Topic 5: Systems Design
(Use Case Realisation)

Mitchell Hughes
Room CLM131
mitchell.hughes@wits.ac.za
@mitchell_hughes

Lecture Materials - Credits

- These lecture materials are largely based on:
    - Satzinger, J., Jackson, R. & Burd, S. (2012). *Introduction to Systems Analysis and Design: An Agile, Iterative Approach*. (6th Ed.). Course Technology, Cengage Learning.
    - Larman, C. (2005). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* (3rd Ed.). Pearson.
- Unless specified, all images used come from Google Images



Learning Objectives

- Demonstrate an understanding of the concept of use case realisation
- Demonstrate the ability to construct interaction diagrams, specifically sequence diagrams, in response to set scenarios
- Demonstrate an understanding of the role of sequence diagrams in moving between design and implementation/coding
- Carry these principles through to the INFO2001 project

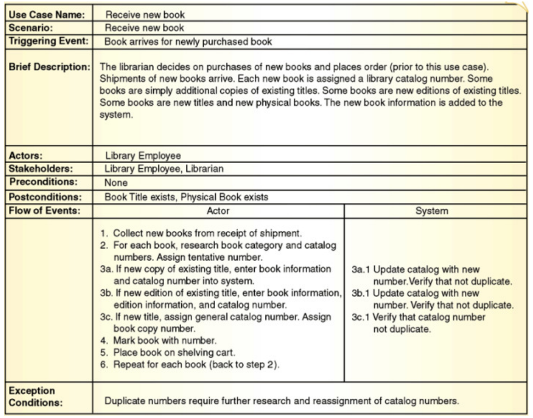
A reminder about the design phase…

- Design is the “bridge” between analysis and implementation
- Specifies the structure of how the system will be written and how it will function, but stops short of actually writing the code
- Specifies the “how”, rather than the “what”
  - How objects will collaborate (work together) or interact with one another to fulfil the requirements of the system

Fitting the pieces together…

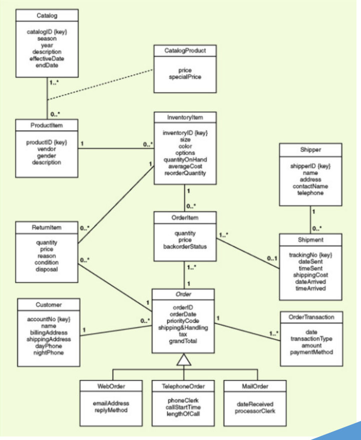
5

Use Case Description

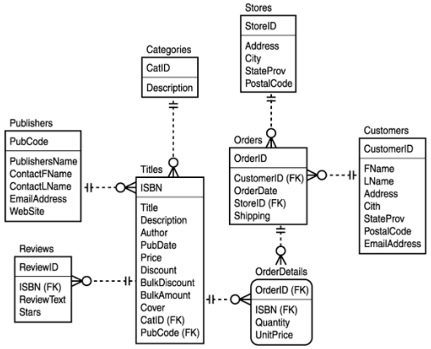


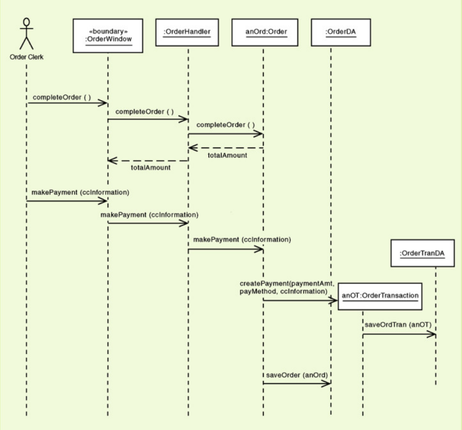
Use Case Name:	Receive new book	
Scenario:	Receive new book	
Triggering Event:	Book arrives for newly purchased book	
Brief Description:	The librarian decides on purchases of new books and places order (prior to this use case). Shipments of new books arrive. Each new book is assigned a library catalog number. Some books are simply additional copies of existing titles. Some books are new editions of existing titles. Some books are new titles and new physical books. The new book information is added to the system.	
Actors:	Library Employee	
Stakeholders:	Library Employee, Librarian	
Preconditions:	None	
Postconditions:	Book Title exists, Physical Book exists	
Flow of Events:	Actor	System
	1. Collect new books from receipt of shipment. 2. For each book, research book category and catalog numbers. Assign tentative number. 3a. If new copy of existing title, enter book information and catalog number into system. 3b. If new edition of existing title, enter book information, edition information, and catalog number. 3c. If new title, assign general catalog number. Assign book copy number. 4. Mark book with number. 5. Place book on shelving cart. 6. Repeat for each book (back to step 2).	3a.1 Update catalog with new number.Verify that not duplicate. 3b.1 Update catalog with new number. Verify that not duplicate. 3c.1 Verify that catalog number not duplicate.
Exception Conditions:	Duplicate numbers require further research and reassignment of catalog numbers.	

Class Diagram

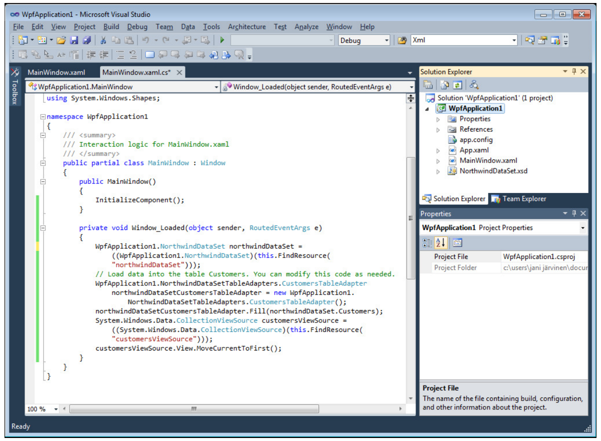


Entity Relationship Diagram (ERD)





Interaction (Sequence) Diagram

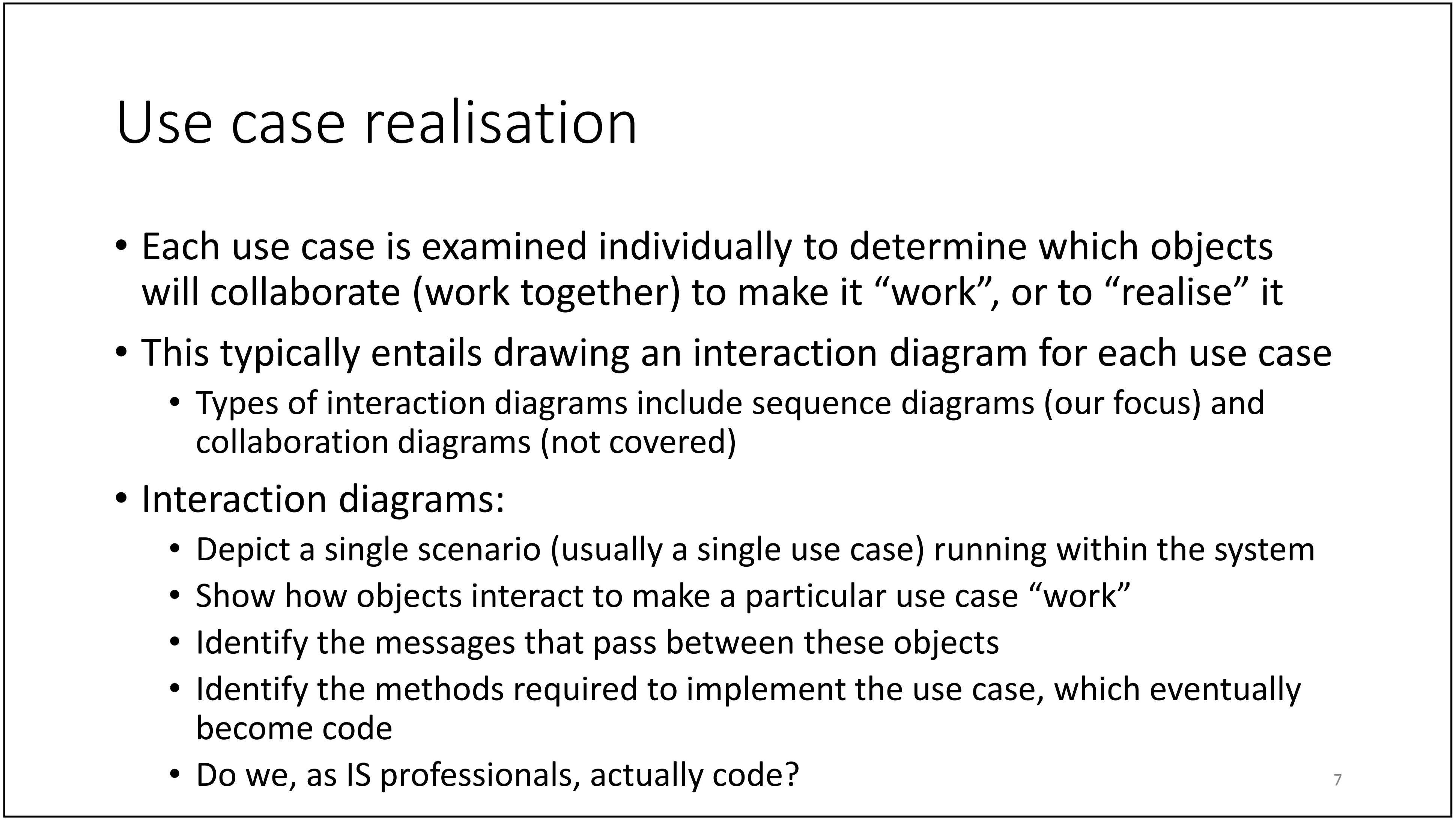


```
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            WpfApplication1.NorthwindDataSet northwindDataSet =
                ((WpfApplication1.NorthwindDataSet)(this.FindResource(
                "northwindDataSet")));
            // Load data into the table Customers. You can modify this code as needed.
            WpfApplication1.NorthwindDataSetTableAdapters.CustomersTableAdapter
                northwindDataSetCustomersTableAdapter = new WpfApplication1.
                    NorthwindDataSetTableAdapters.CustomersTableAdapter();
            northwindDataSetCustomersTableAdapter.Fill(northwindDataSet.Customers);
            System.Windows.Data.CollectionViewSource customersViewSource =
                ((System.Windows.Data.CollectionViewSource)(this.FindResource(
                "customersViewSource")));
            customersViewSource.View.MoveCurrentToFirst();
        }
    }
}
```

Project File	WpfApplication1.csproj
Project Folder	c:\users\jani järvinen\docur

Project File
The name of the file containing build, configuration, and other information about the project.

Implementation (Coding)

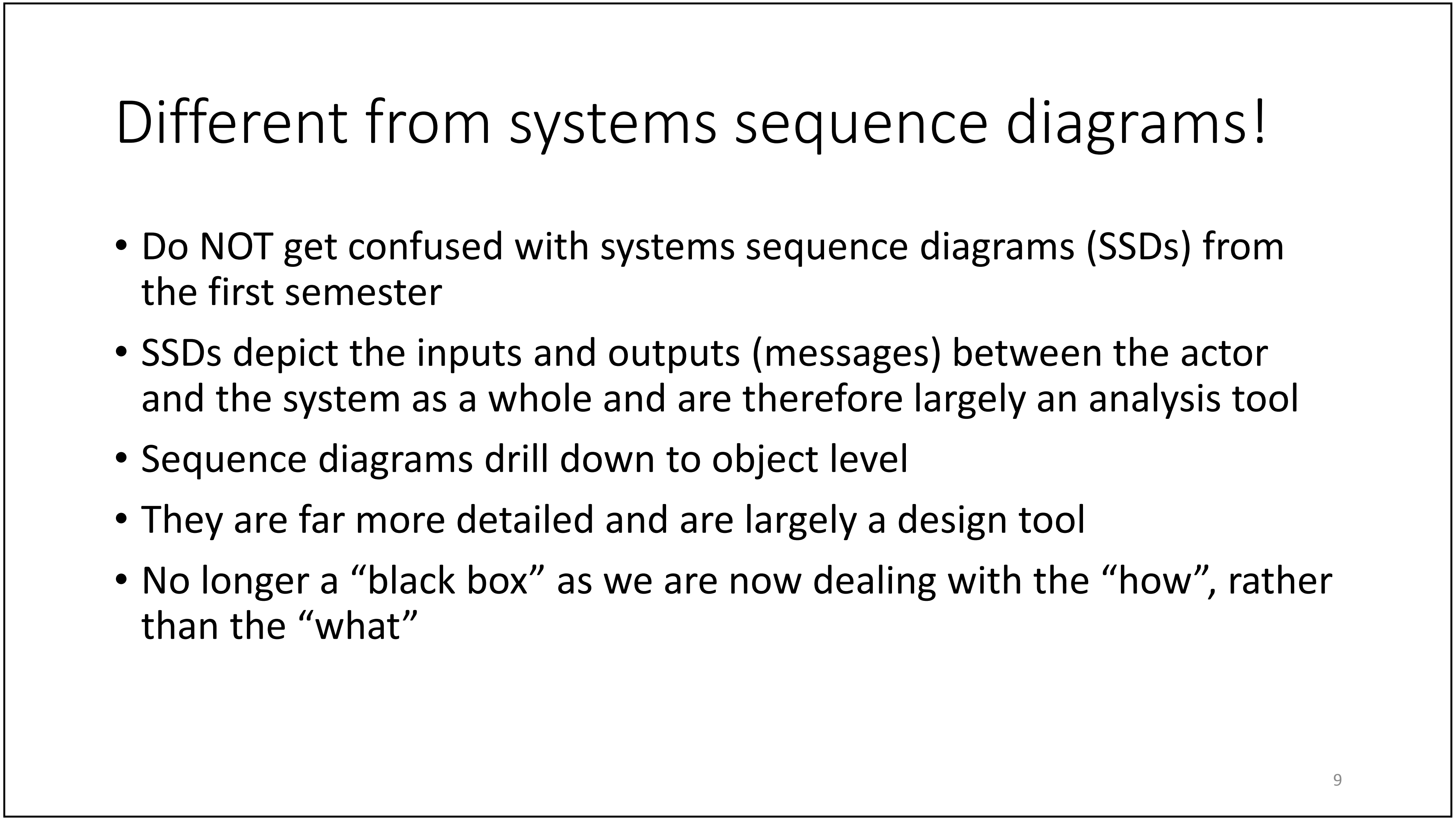


Use case realisation

- Each use case is examined individually to determine which objects will collaborate (work together) to make it “work”, or to “realise” it
- This typically entails drawing an interaction diagram for each use case
  - Types of interaction diagrams include sequence diagrams (our focus) and collaboration diagrams (not covered)
- Interaction diagrams:
  - Depict a single scenario (usually a single use case) running within the system
  - Show how objects interact to make a particular use case “work”
  - Identify the messages that pass between these objects
  - Identify the methods required to implement the use case, which eventually become code
  - Do we, as IS professionals, actually code?

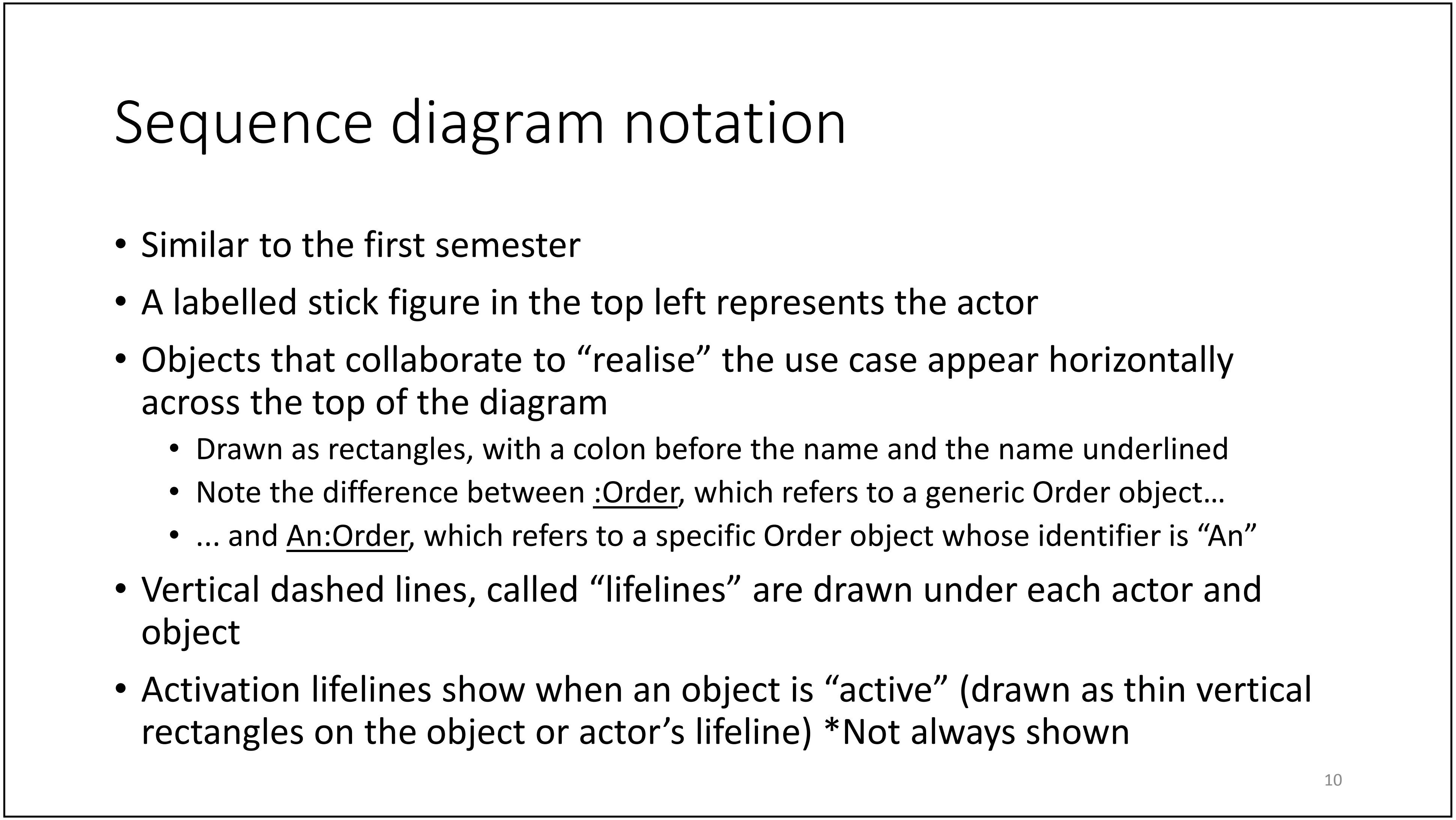
Sequence diagrams

- The most commonly used type of interaction diagram
- Largely derived from use case descriptions (and accompanying use case diagrams) and class diagrams
- Time-ordered, hence “sequence”
- Show the sequence of messages that is sent between objects to “realise” each use case
- Typically one use case per sequence diagram, but can also include or “invoke” other use cases through <<includes>> and <<extends>>
- Should be (☺) used by programmers to code the system, as they provide a visual “bridge” between analysis and implementation



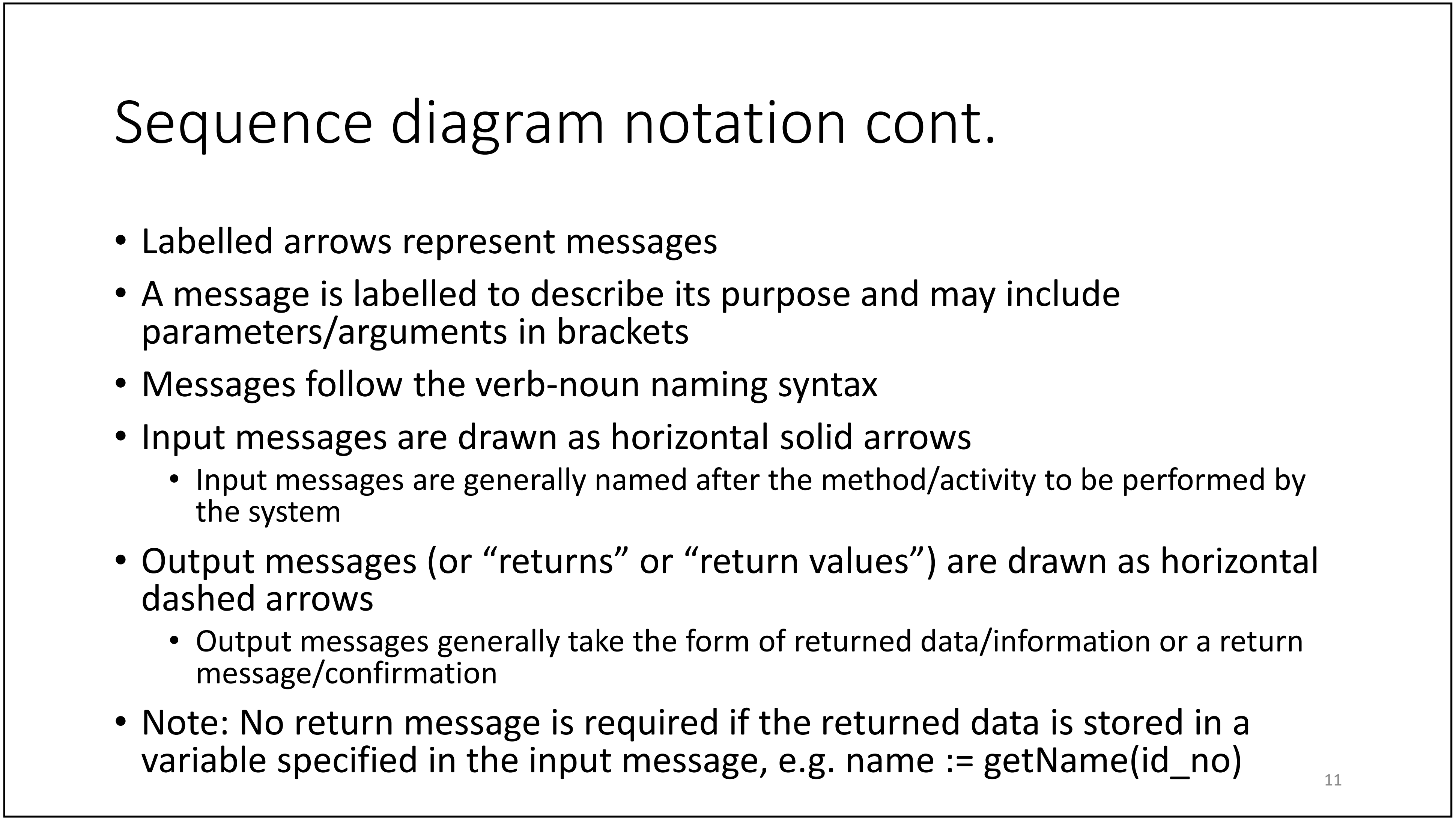
Different from systems sequence diagrams!

- Do NOT get confused with systems sequence diagrams (SSDs) from the first semester
- SSDs depict the inputs and outputs (messages) between the actor and the system as a whole and are therefore largely an analysis tool
- Sequence diagrams drill down to object level
- They are far more detailed and are largely a design tool
- No longer a “black box” as we are now dealing with the “how”, rather than the “what”



Sequence diagram notation

- Similar to the first semester
- A labelled stick figure in the top left represents the actor
- Objects that collaborate to "realise" the use case appear horizontally across the top of the diagram
  - Drawn as rectangles, with a colon before the name and the name underlined
  - Note the difference between <u>:Order</u>, which refers to a generic Order object…
  - ... and <u>An:Order</u>, which refers to a specific Order object whose identifier is "An"
- Vertical dashed lines, called "lifelines" are drawn under each actor and object
- Activation lifelines show when an object is "active" (drawn as thin vertical rectangles on the object or actor's lifeline) *Not always shown



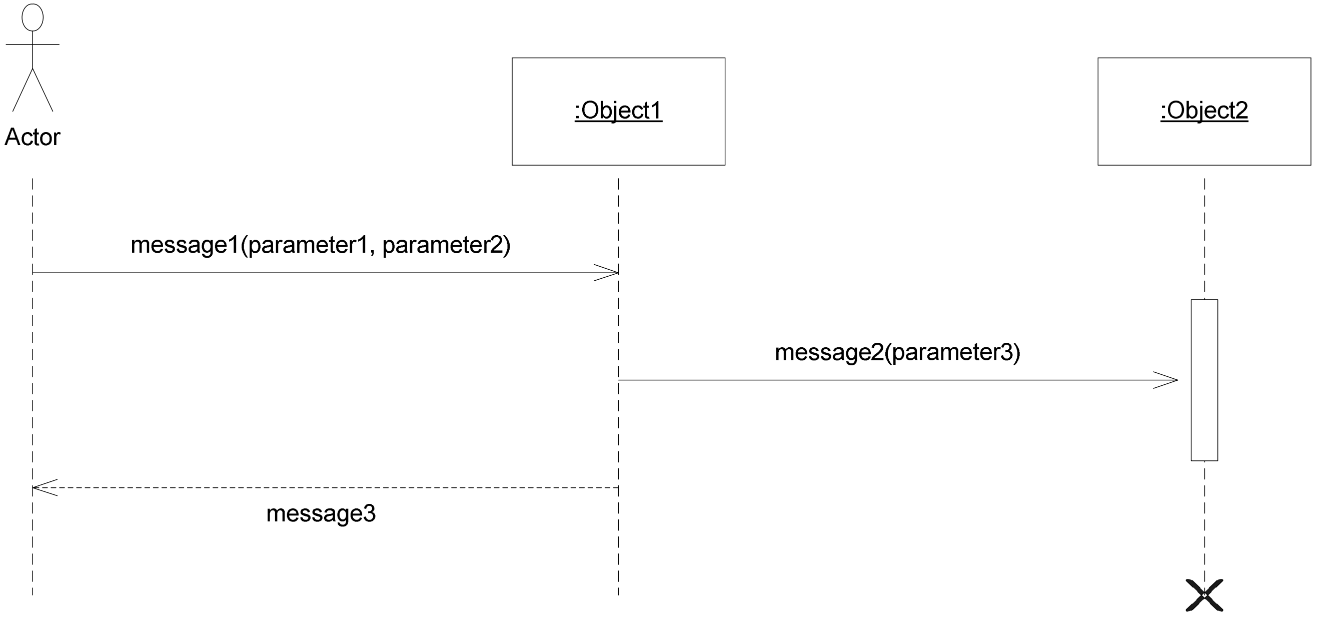
Sequence diagram notation cont.

- Labelled arrows represent messages
- A message is labelled to describe its purpose and may include parameters/arguments in brackets
- Messages follow the verb-noun naming syntax
- Input messages are drawn as horizontal solid arrows
  - Input messages are generally named after the method/activity to be performed by the system
- Output messages (or “returns” or “return values”) are drawn as horizontal dashed arrows
  - Output messages generally take the form of returned data/information or a return message/confirmation
- Note: No return message is required if the returned data is stored in a variable specified in the input message, e.g. name := getName(id_no)

How to “read” a sequence diagram

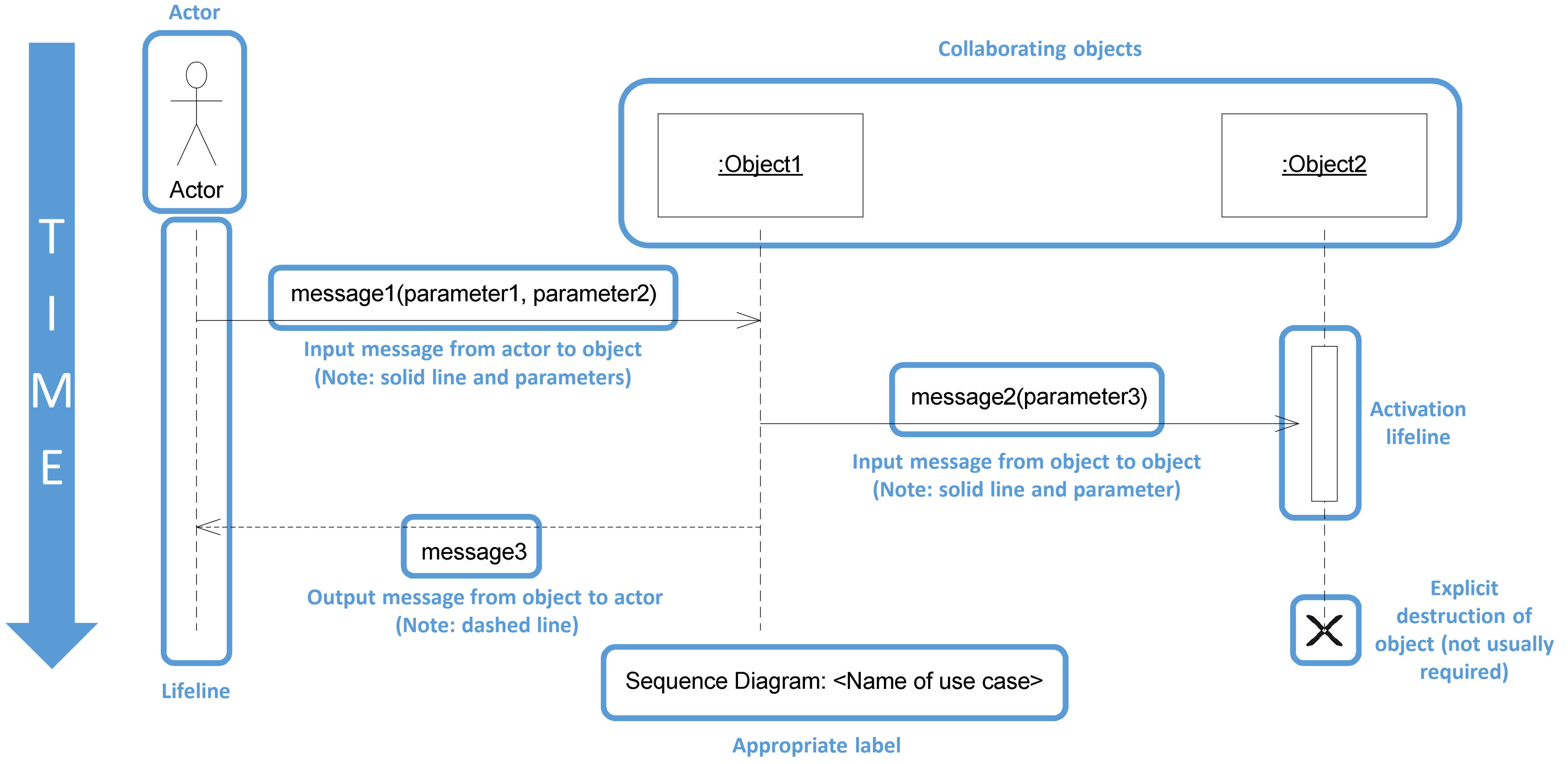
- Read horizontally (across) to see which actors and objects are collaborating to “realise” the use case
- Read vertically (down) to see when things are happening (the further down you read, the further along in time things are happening)
- Arrows represent messages being sent and received by actors and objects

A generic sequence diagram

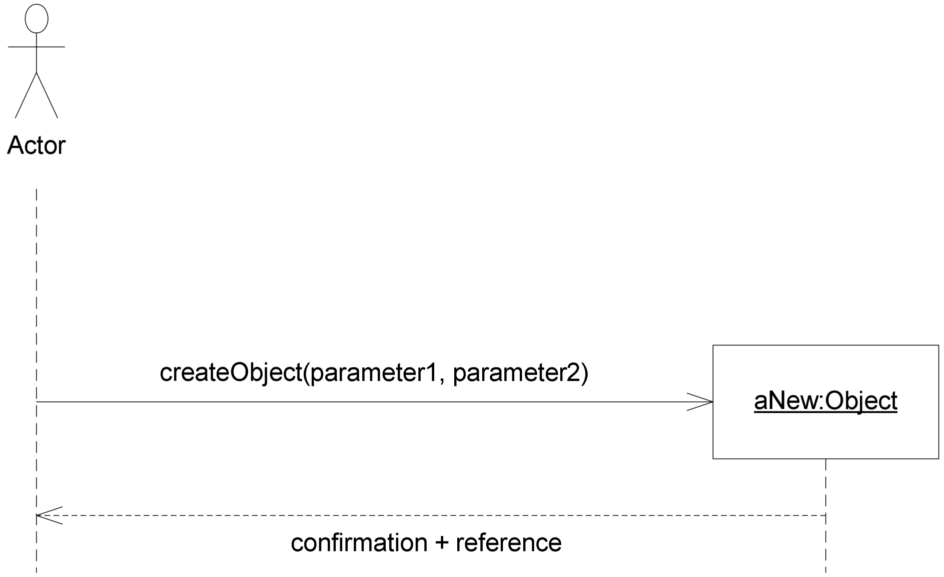


Sequence Diagram: <Name of use case>

A generic sequence diagram

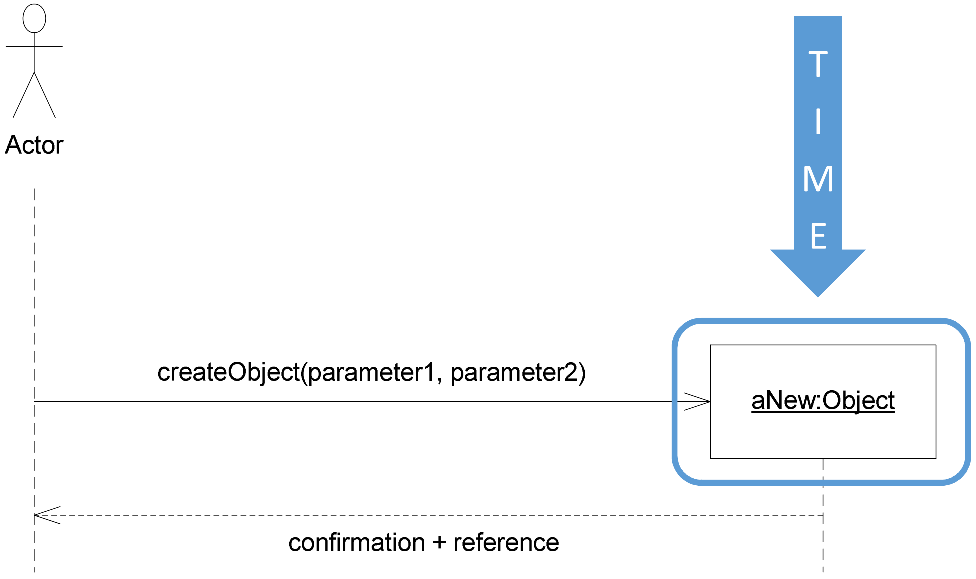

Actor
T I M E
Actor
Collaborating objects
:Object1
:Object2
message1(parameter1, parameter2)
Input message from actor to object
(Note: solid line and parameters)
message2(parameter3)
Input message from object to object
(Note: solid line and parameter)
Activation lifeline
message3
Output message from object to actor
(Note: dashed line)
Explicit destruction of object (not usually required)
Lifeline
Sequence Diagram: <Name of use case>
Appropriate label

Example: Creating an new instance of an object, i.e. a Create use case



Sequence Diagram: Create <name of object>

Example: Creating an new instance of an object, i.e. a Create use case

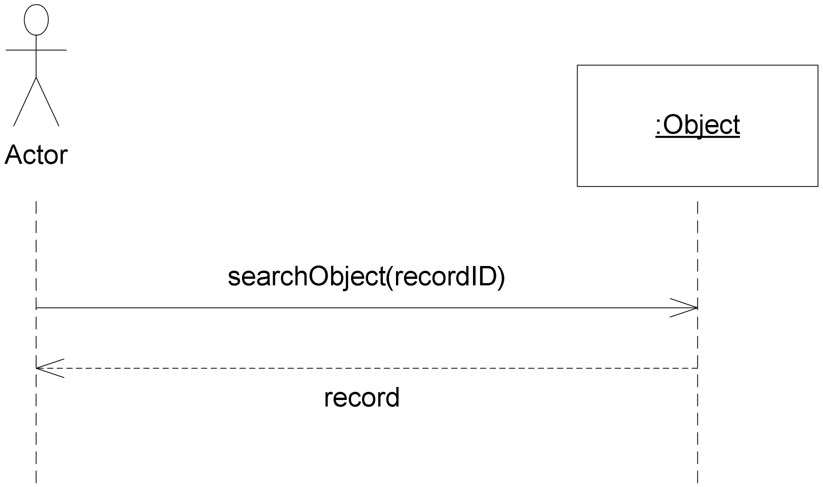


New object created

Note: Object does NOT exist at start of use case

Sequence Diagram: Create <name of object>

Example: Retrieving objects, i.e. a Read or Search use case



Actor

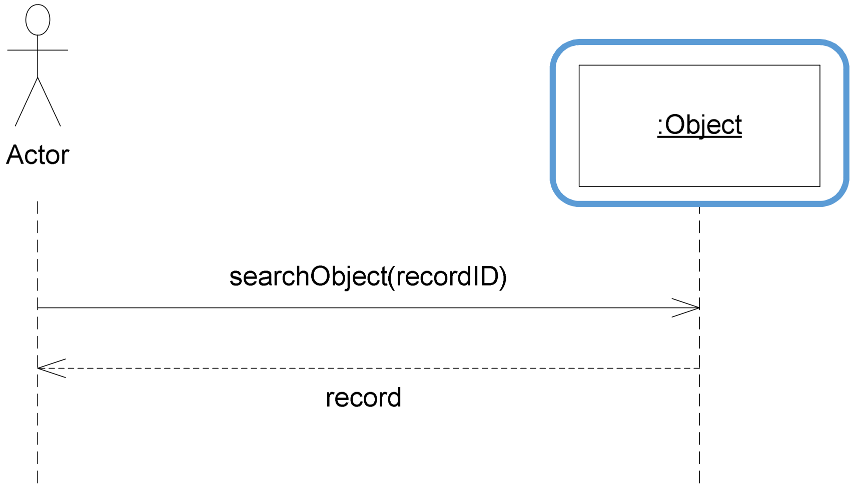
:Object

searchObject(recordID)

record

Sequence Diagram: Read <name of object> or Search <name of object>

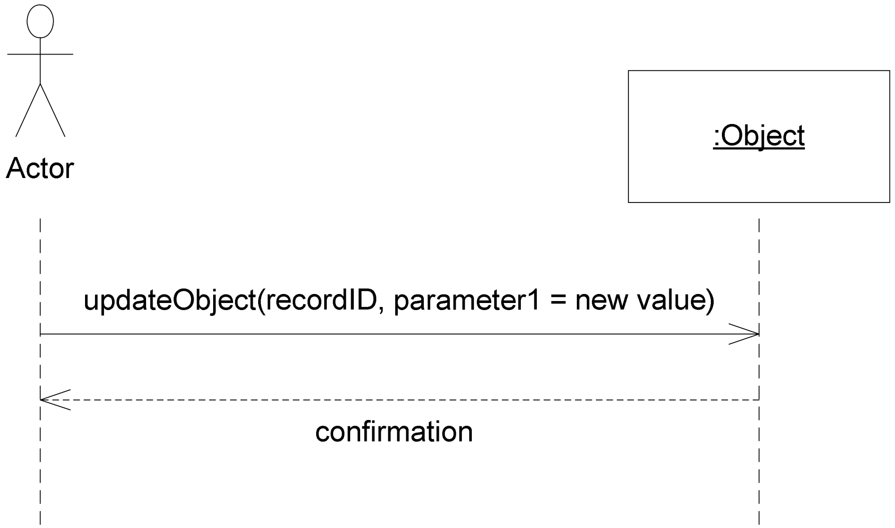
Example: Retrieving objects, i.e. a Read or Search use case



Note: Object exists at start of use case

Sequence Diagram: Read <name of object> or Search <name of object>

Example: Updating objects, i.e. an Update use case



Actor

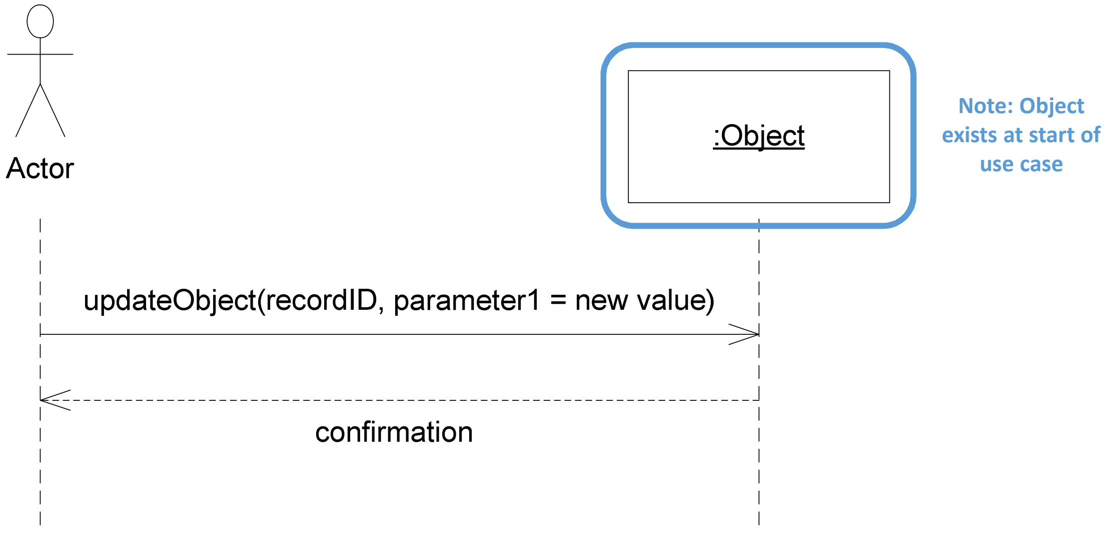
:Object

updateObject(recordID, parameter1 = new value)

confirmation

Sequence Diagram: Update <name of object>

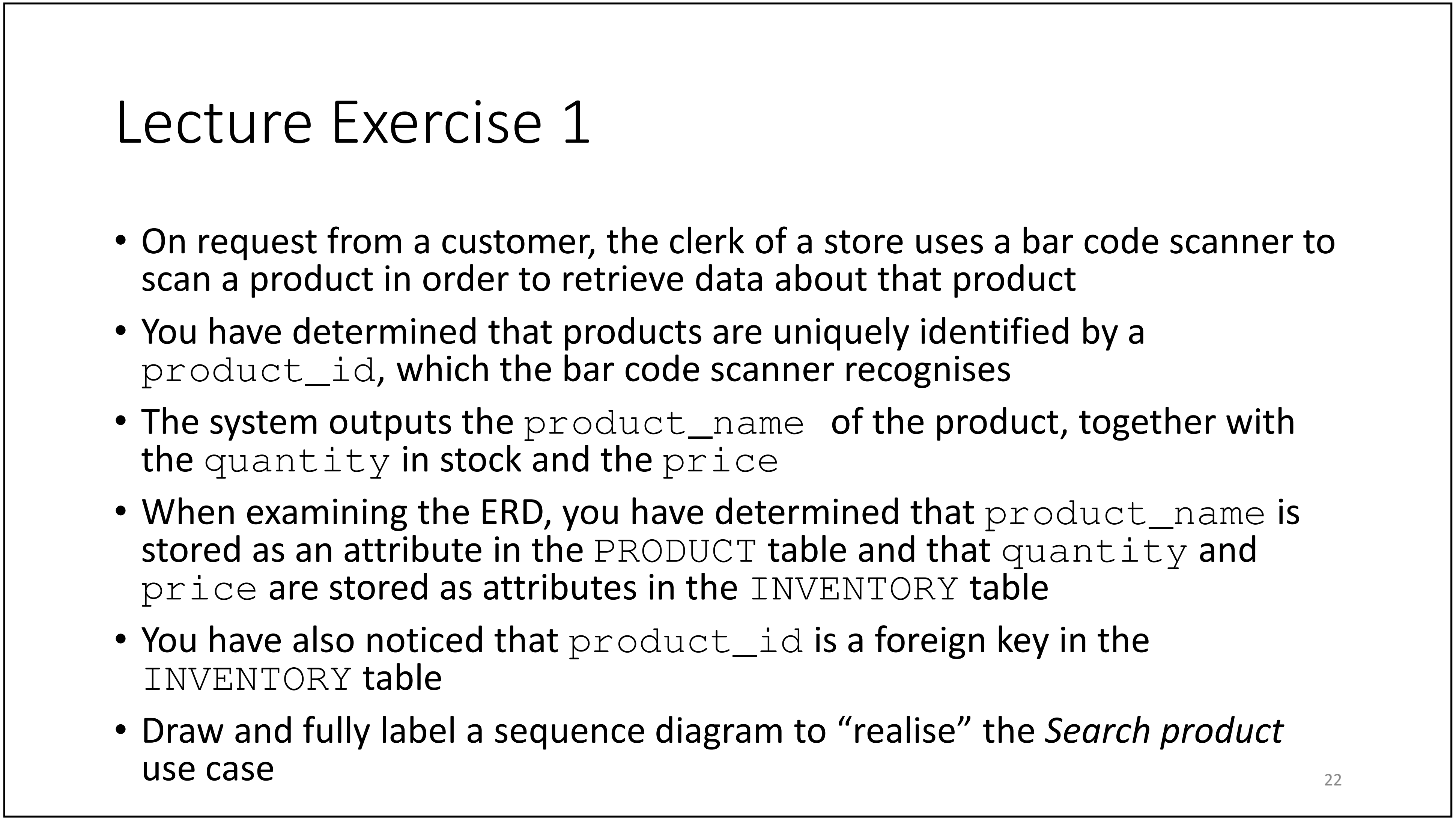
Example: Updating objects, i.e. an Update use case


Actor
:Object
Note: Object exists at start of use case
updateObject(recordID, parameter1 = new value)
confirmation

Sequence Diagram: Update <name of object>

Creating sequence diagrams

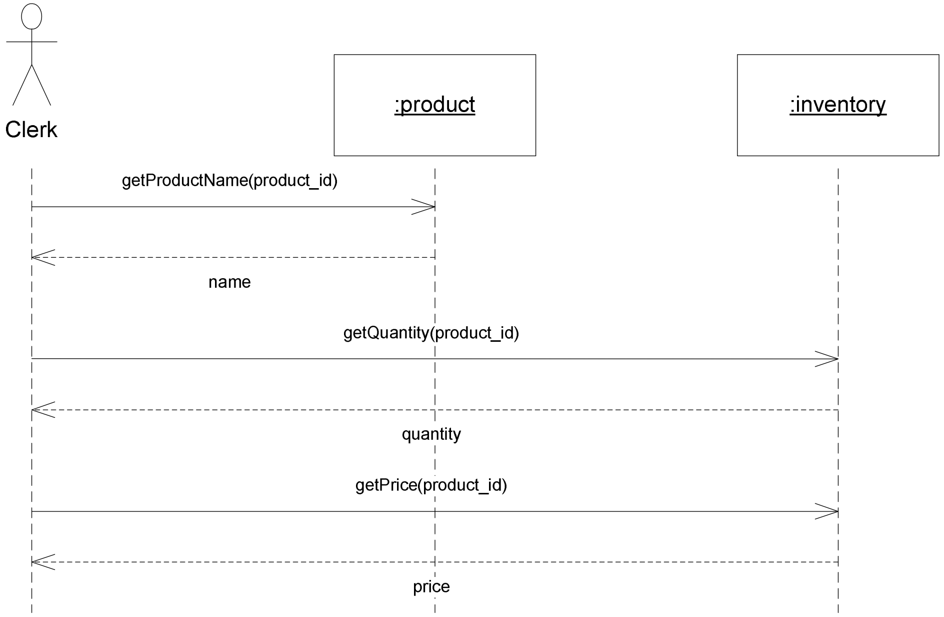
- The first step is to examine and understand the use case description, focusing particularly on the objects that will be collaborating to “realise” the use case
- These are often specifically mentioned as data stores inside a well-written flow of activities
- The second step is to identify the messages and parameters that pass between these objects
- The third step is to examine the use case’s post-conditions to determine success conditions
- The final step is to model these objects, messages and rules in a sequence diagram



Lecture Exercise 1

- On request from a customer, the clerk of a store uses a bar code scanner to scan a product in order to retrieve data about that product
- You have determined that products are uniquely identified by a `product_id`, which the bar code scanner recognises
- The system outputs the `product_name` of the product, together with the `quantity` in stock and the `price`
- When examining the ERD, you have determined that `product_name` is stored as an attribute in the `PRODUCT` table and that `quantity` and `price` are stored as attributes in the `INVENTORY` table
- You have also noticed that `product_id` is a foreign key in the `INVENTORY` table
- Draw and fully label a sequence diagram to “realise” the *Search product* use case

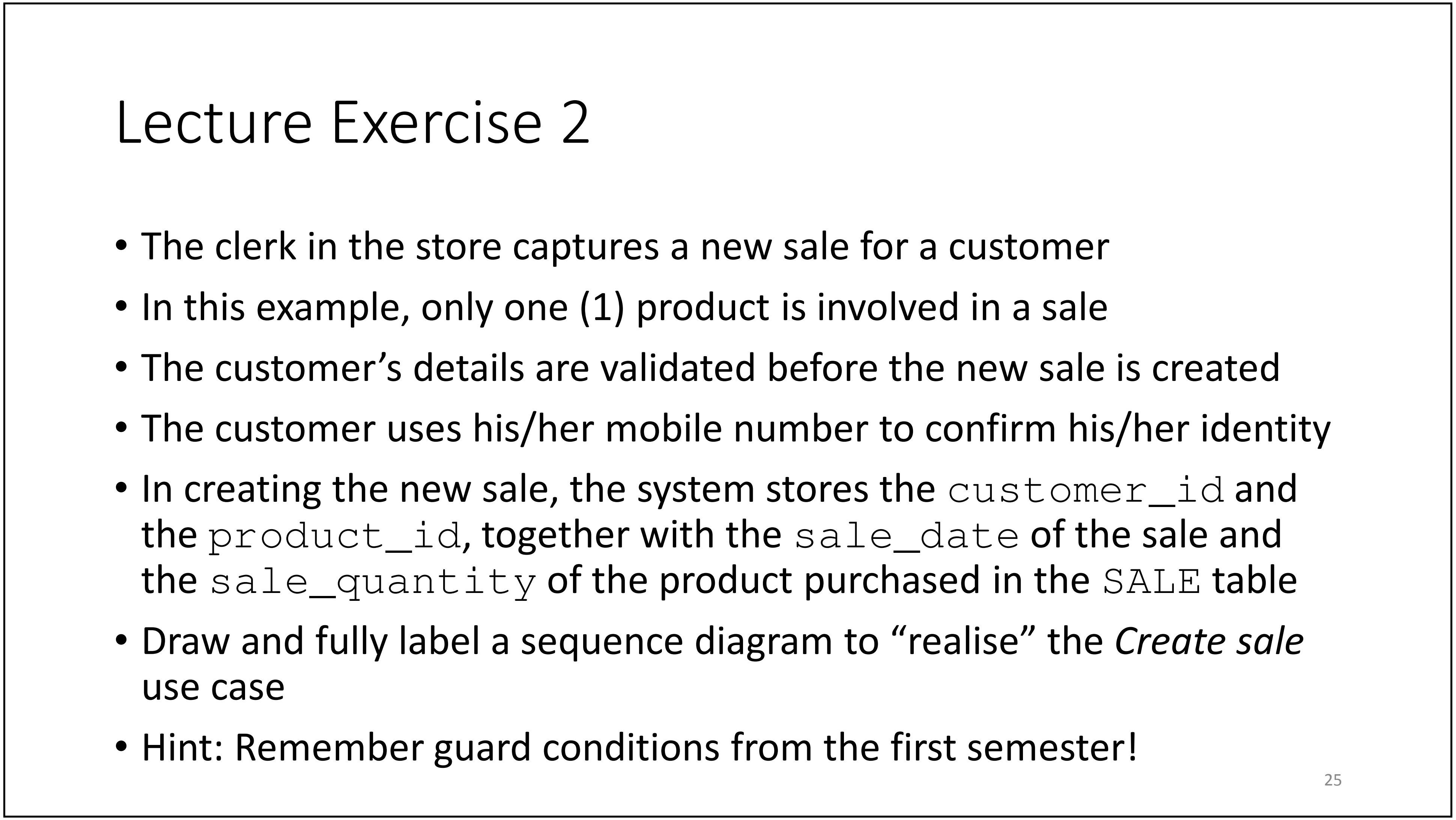
Lecture Exercise 1 - Suggested Solution



Sequence Diagram: Search product (or Read product)

How do sequence diagrams relate to implementation/coding?

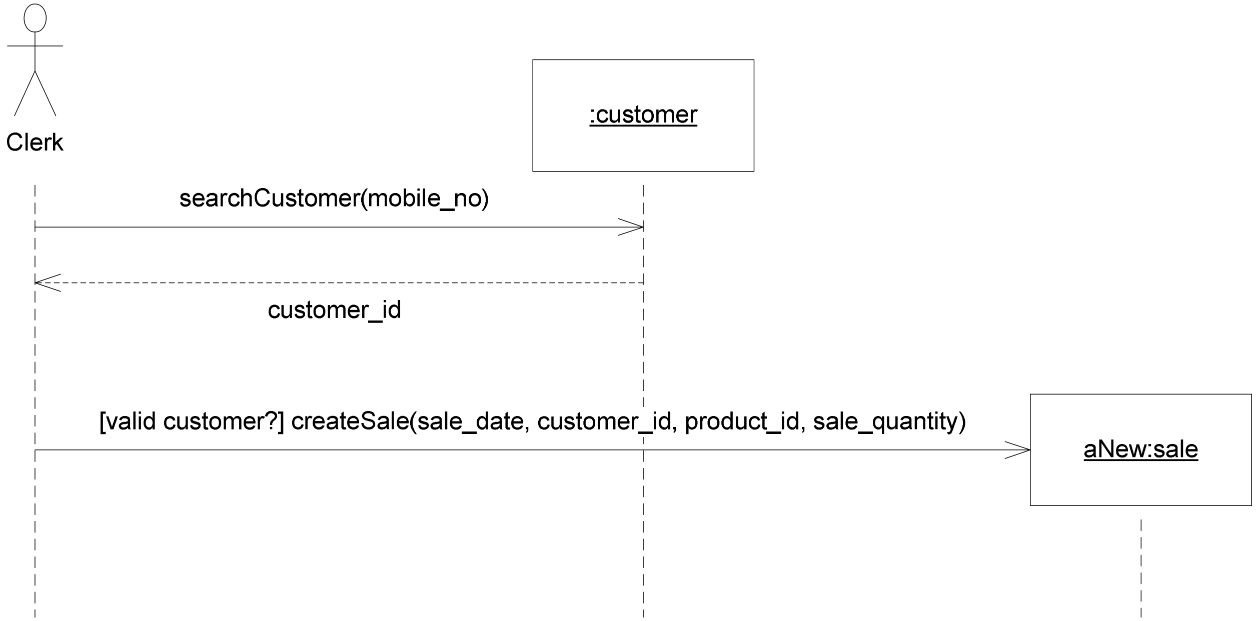
- The messages (methods) and parameters identified can now be implemented/coded using C# and SQL statements
- For example, we would require a SQL statement that uses `product_id` to retrieve `product_name` from the `PRODUCT` table and `quantity` and `price` from the `INVENTORY` table



Lecture Exercise 2

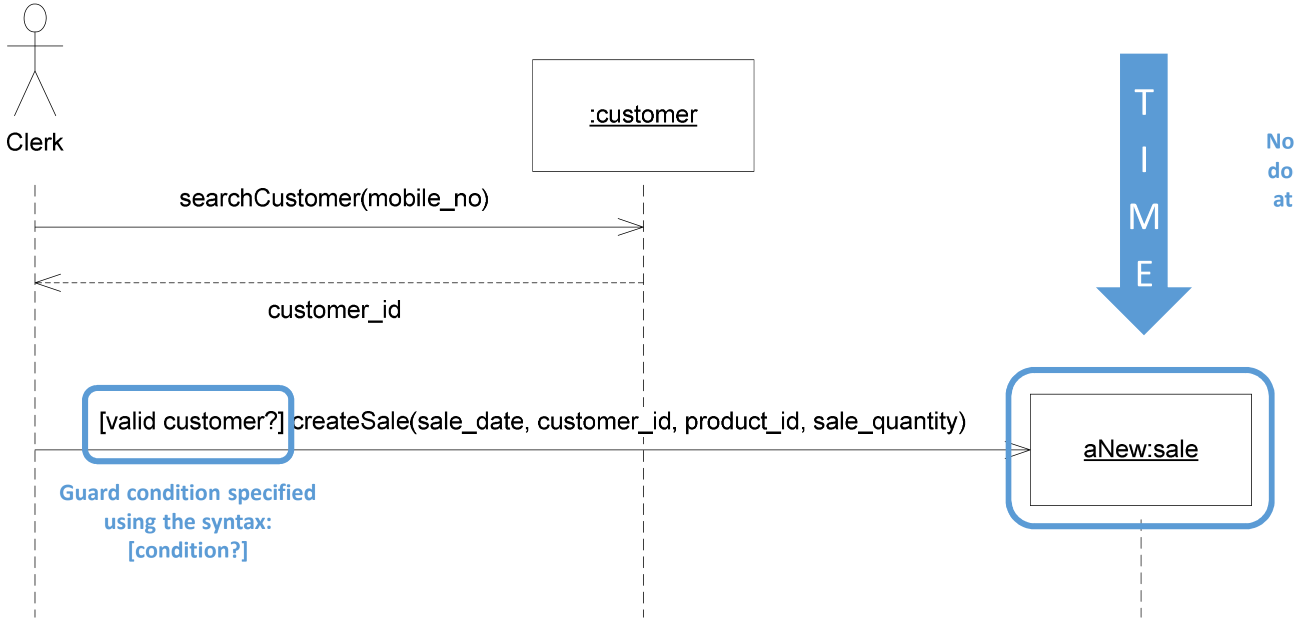
- The clerk in the store captures a new sale for a customer
- In this example, only one (1) product is involved in a sale
- The customer's details are validated before the new sale is created
- The customer uses his/her mobile number to confirm his/her identity
- In creating the new sale, the system stores the `customer_id` and the `product_id`, together with the `sale_date` of the sale and the `sale_quantity` of the product purchased in the `SALE` table
- Draw and fully label a sequence diagram to "realise" the *Create sale* use case
- Hint: Remember guard conditions from the first semester!

Lecture Exercise 2 - Suggested Solution


Clerk
:customer
searchCustomer(mobile_no)
customer_id
[valid customer?] createSale(sale_date, customer_id, product_id, sale_quantity)
aNew:sale

Sequence Diagram: Create sale

Lecture Exercise 2 - Suggested Solution


Clerk
:customer
searchCustomer(mobile_no)
customer_id
[valid customer?] createSale(sale_date, customer_id, product_id, sale_quantity)
aNew:sale
T
I
M
E

Note: New sale does NOT exist at start of use case

Note: The identifier referring to a specific new sale being created

Guard condition specified using the syntax: [condition?]

Sequence Diagram: Create sale

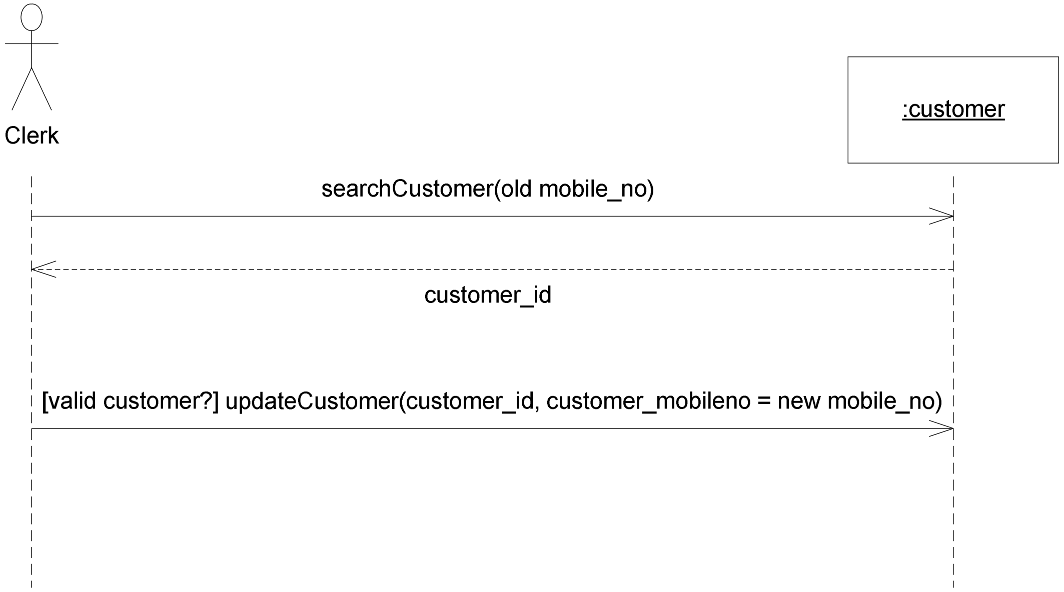
What about implementation/coding?

- Creating a sale would require a SQL `INSERT` statement that creates a new record in the `SALE` table using `sale_date`, `customer_id`, `product_id` and `sale_quantity`

Lecture Exercise 3

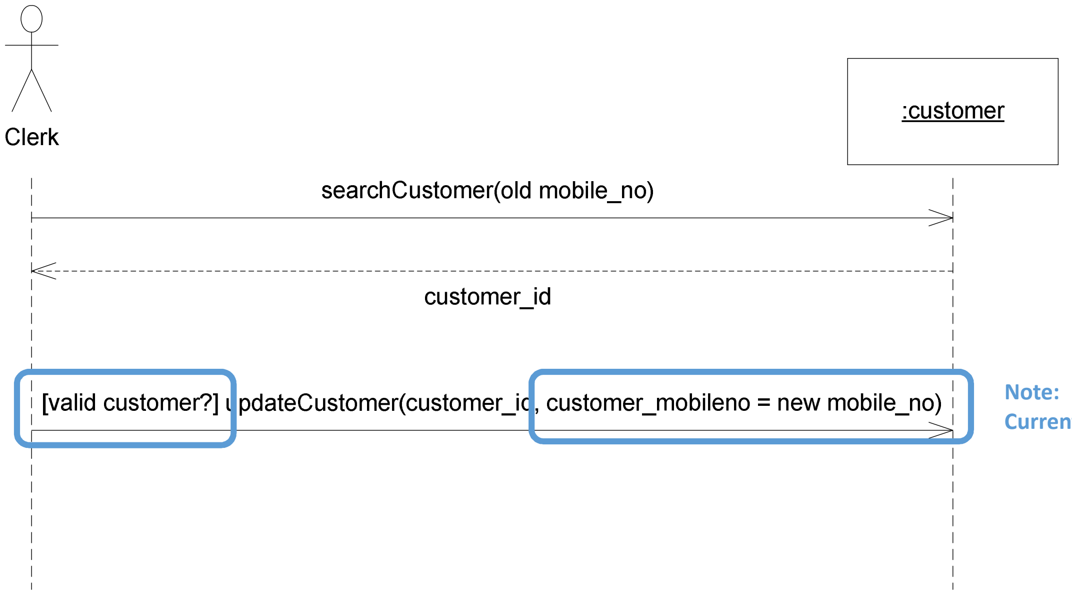
- The clerk in the store updates the mobile number for an existing customer so that he/she can continue shopping
- The customer’s details are retrieved using the old mobile number before the update is made
- Draw and fully label a sequence diagram to “realise” the *Update customer* use case

Lecture Exercise 3 - Suggested Solution


Clerk
:customer
searchCustomer(old mobile_no)
customer_id
[valid customer?] updateCustomer(customer_id, customer_mobileno = new mobile_no)

Sequence Diagram: Update customer

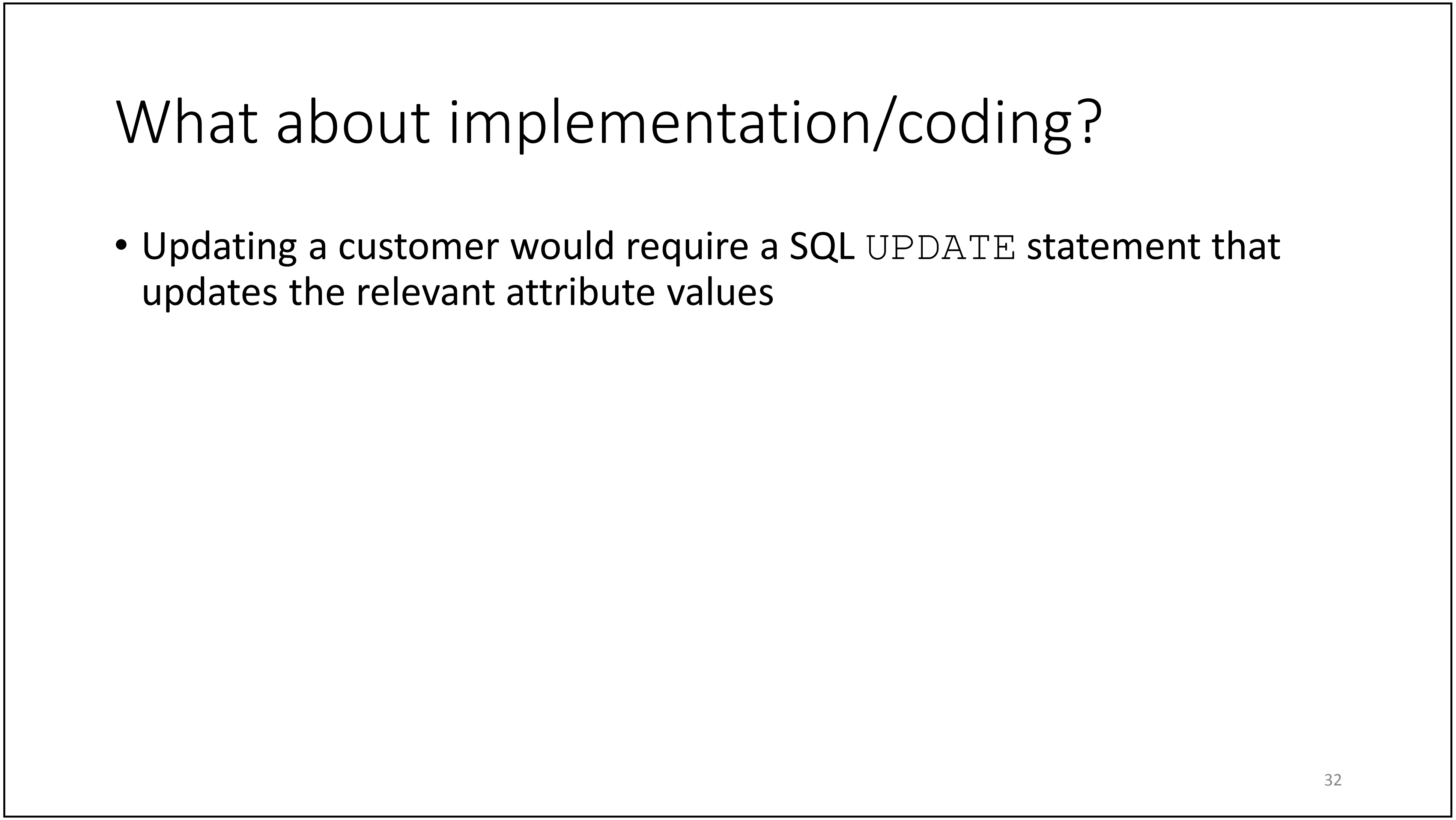
Lecture Exercise 3 - Suggested Solution


Clerk
:customer
searchCustomer(old mobile_no)
customer_id
[valid customer?] updateCustomer(customer_id, customer_mobileno = new mobile_no)

Guard condition specified using the syntax: [condition?]

Note:
Current attribute value replaced by new

Sequence Diagram: Update customer



What about implementation/coding?

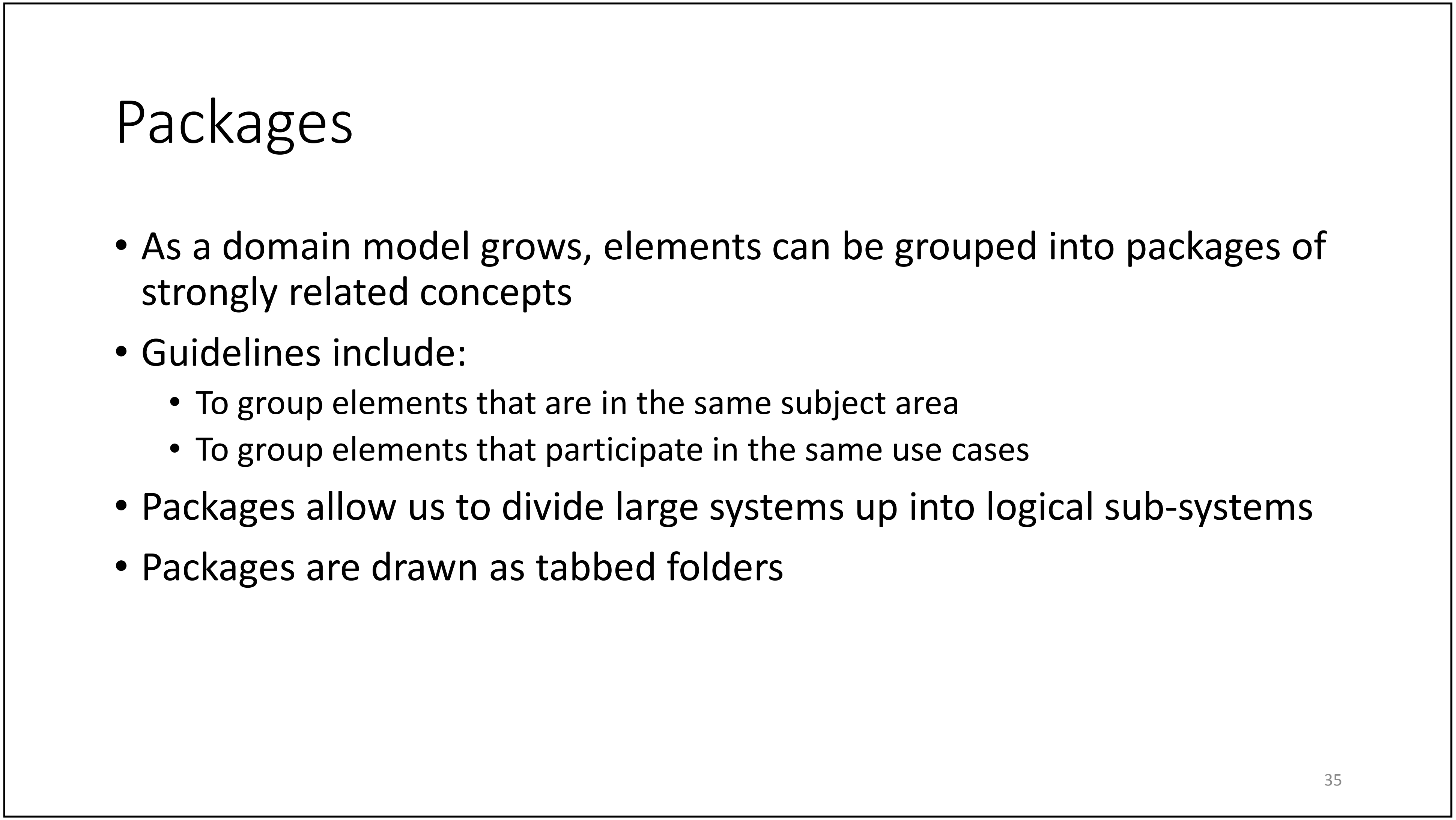
- Updating a customer would require a SQL `UPDATE` statement that updates the relevant attribute values

Adding detail to sequence diagrams

- Sequence diagrams can be extended beyond showing just entity/domain classes to include:
  - Boundary objects (usually in the form of forms, windows or screens)
  - Controller class objects (catch messages from the boundary class and route/send them to the appropriate entity/domain class, i.e. they act as “switchboards”)
  - Data access objects (make connections to an underlying database)
  - We are not concerned with data access objects at second year level

Naming controller objects

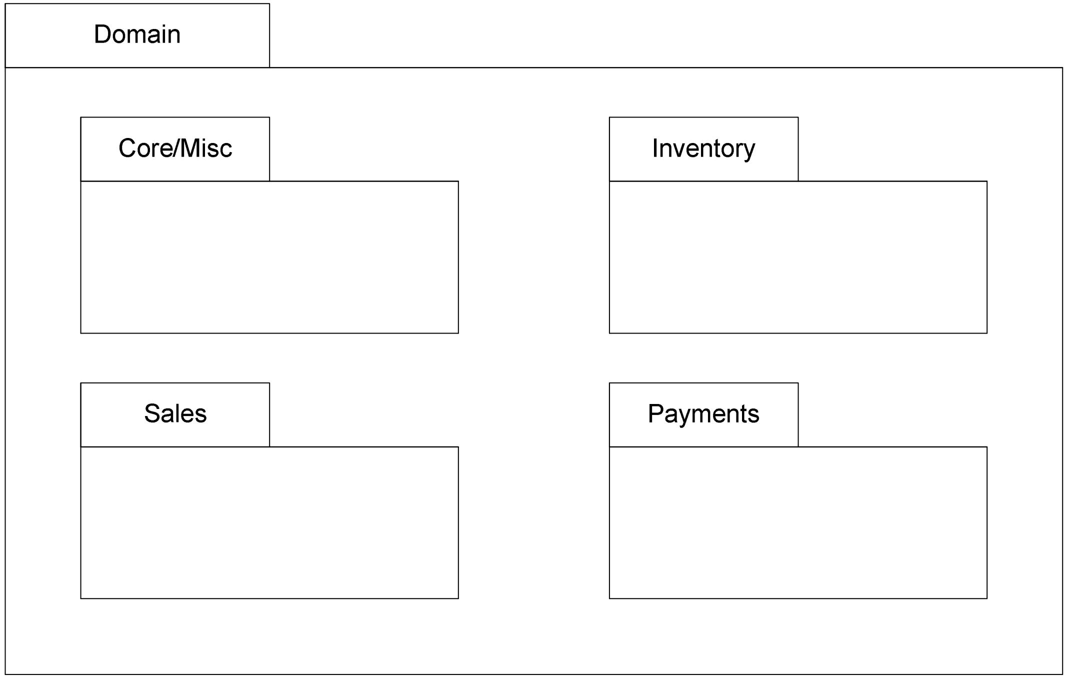
- Controller objects are usually given the generic suffix, “Handler”
- They usually take their prefix from its appropriate PACKAGE name



Packages

- As a domain model grows, elements can be grouped into packages of strongly related concepts
- Guidelines include:
  - To group elements that are in the same subject area
  - To group elements that participate in the same use cases
- Packages allow us to divide large systems up into logical sub-systems
- Packages are drawn as tabbed folders

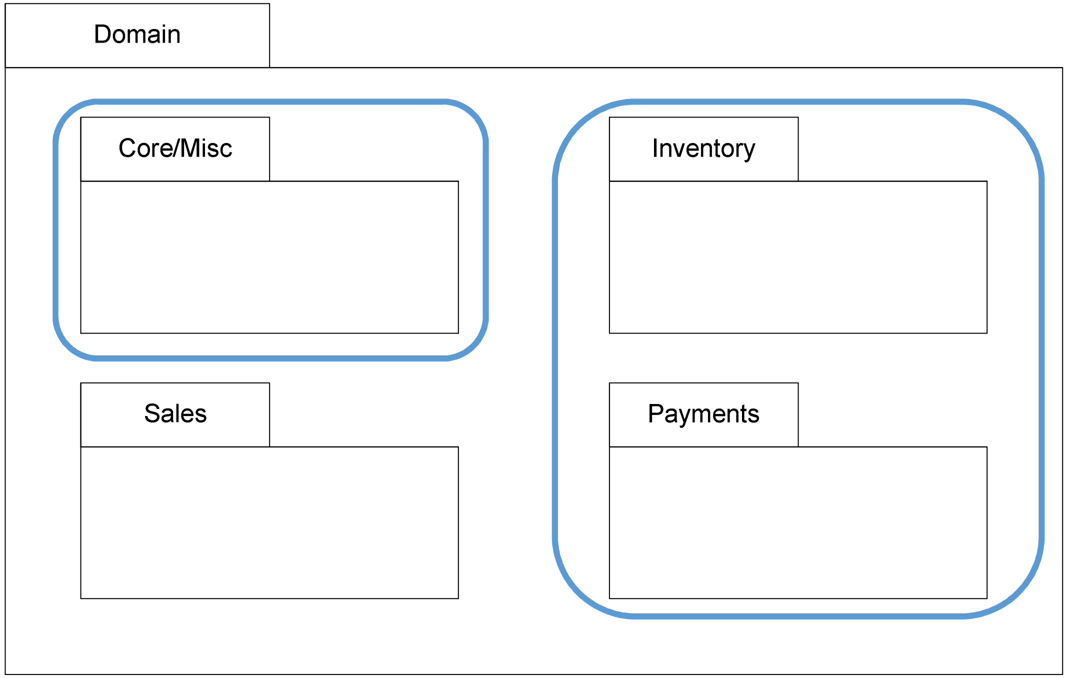
Example package organisation



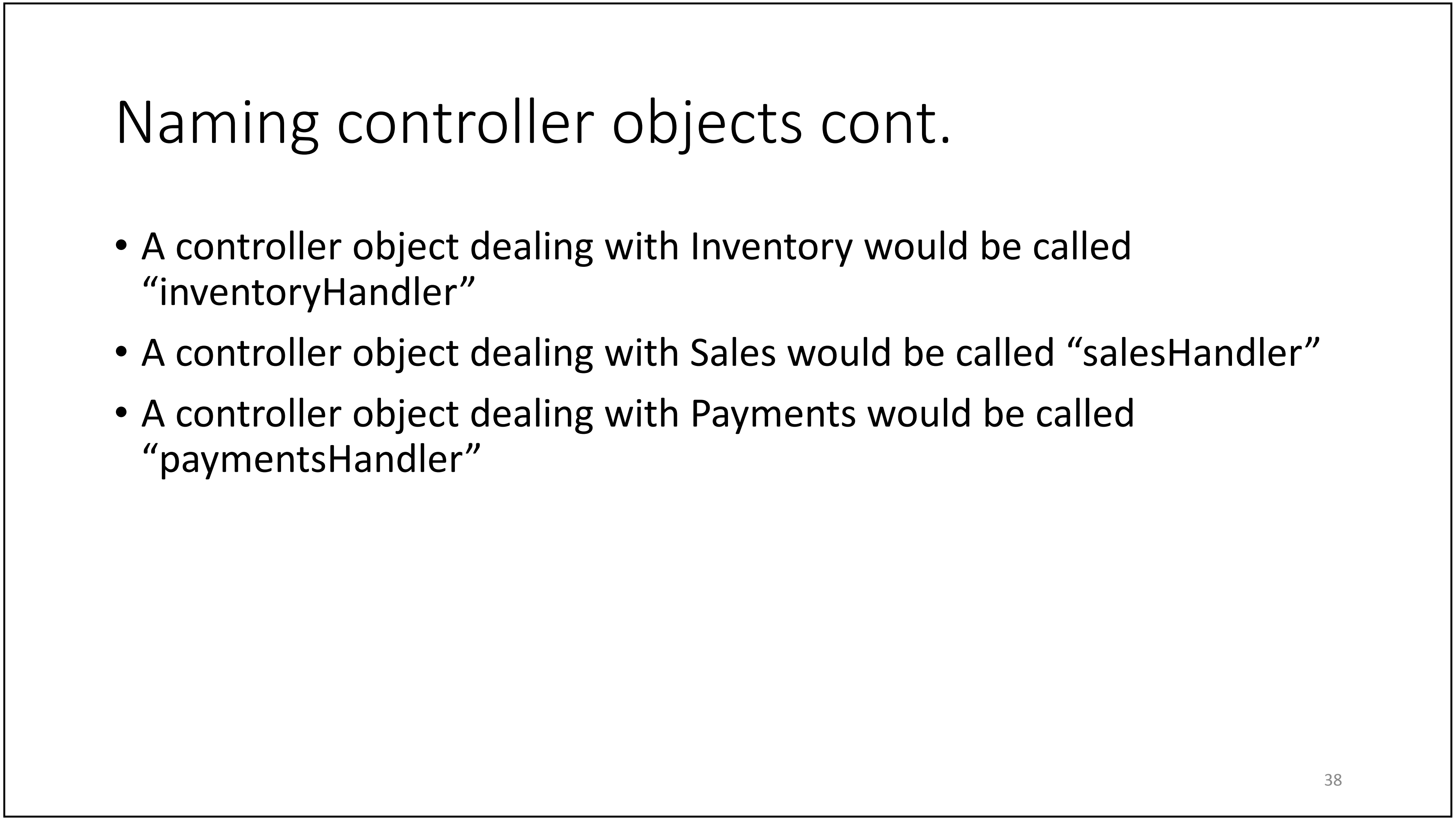
Example package organisation

“Core/Misc” owns widely shared concepts or those without an obvious home

Usually just referred to as “Core”



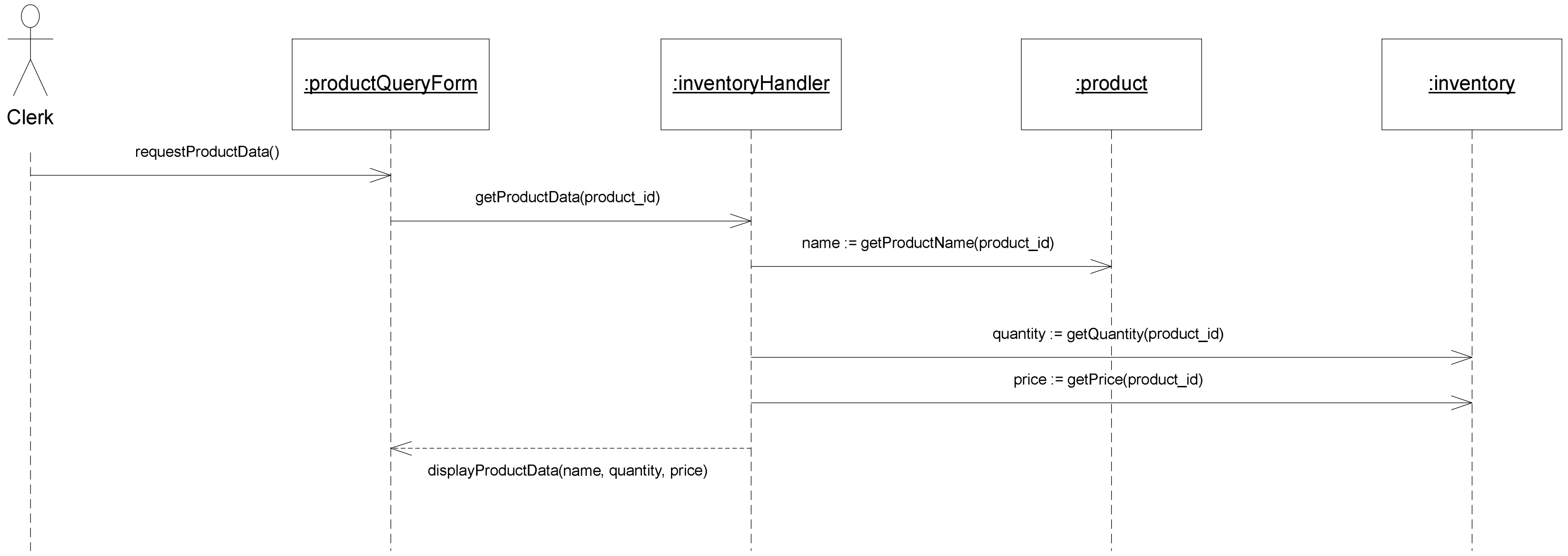
Examples of related business concepts grouped into packages



Naming controller objects cont.

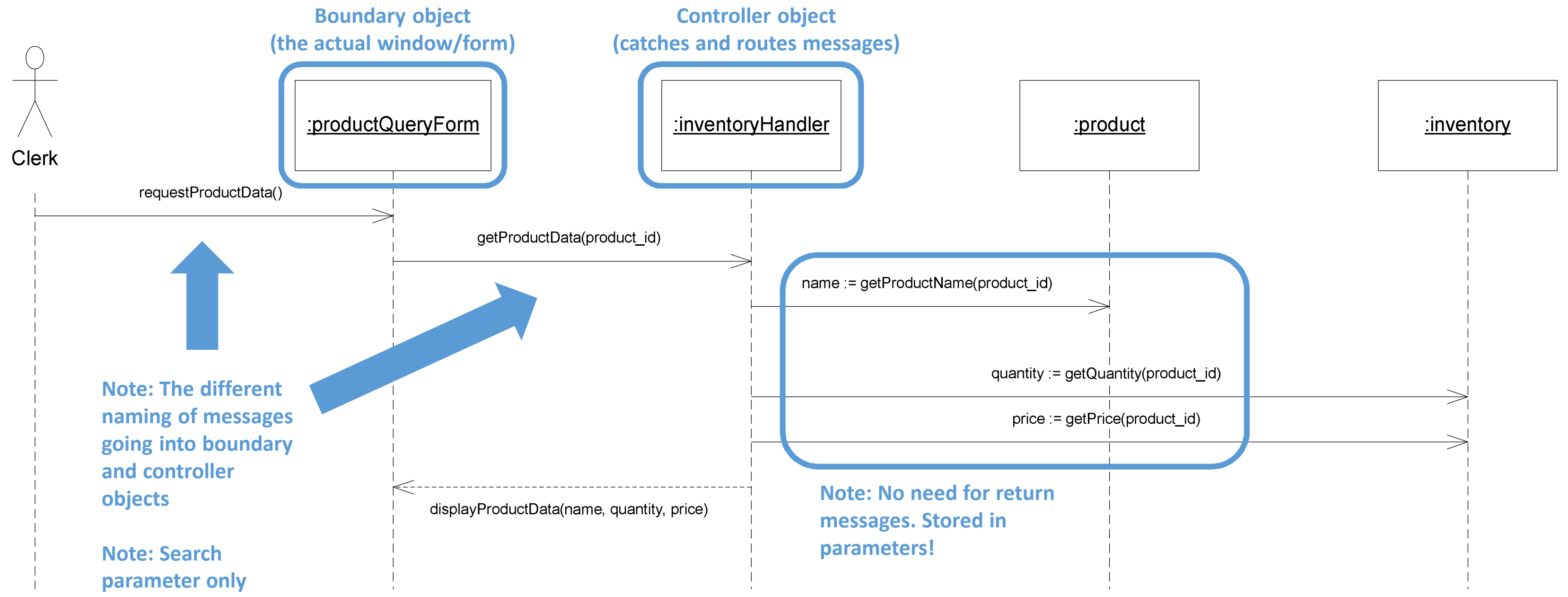
- A controller object dealing with Inventory would be called "inventoryHandler"
- A controller object dealing with Sales would be called "salesHandler"
- A controller object dealing with Payments would be called "paymentsHandler"

Extending Lecture Exercise 1


Clerk
:productQueryForm
:inventoryHandler
:product
:inventory
requestProductData()
getProductData(product_id)
name := getProductName(product_id)
quantity := getQuantity(product_id)
price := getPrice(product_id)
displayProductData(name, quantity, price)

Sequence Diagram: Search product (or Read product)

Extending Lecture Exercise 1


Clerk
:productQueryForm
:inventoryHandler
:product
:inventory
requestProductData()
getProductData(product_id)
name := getProductName(product_id)
quantity := getQuantity(product_id)
price := getPrice(product_id)
displayProductData(name, quantity, price)

Boundary object (the actual window/form)

Controller object (catches and routes messages)

Note: The different naming of messages going into boundary and controller objects

Note: Search parameter only available AFTER form has been presented to the actor, hence empty brackets on first message

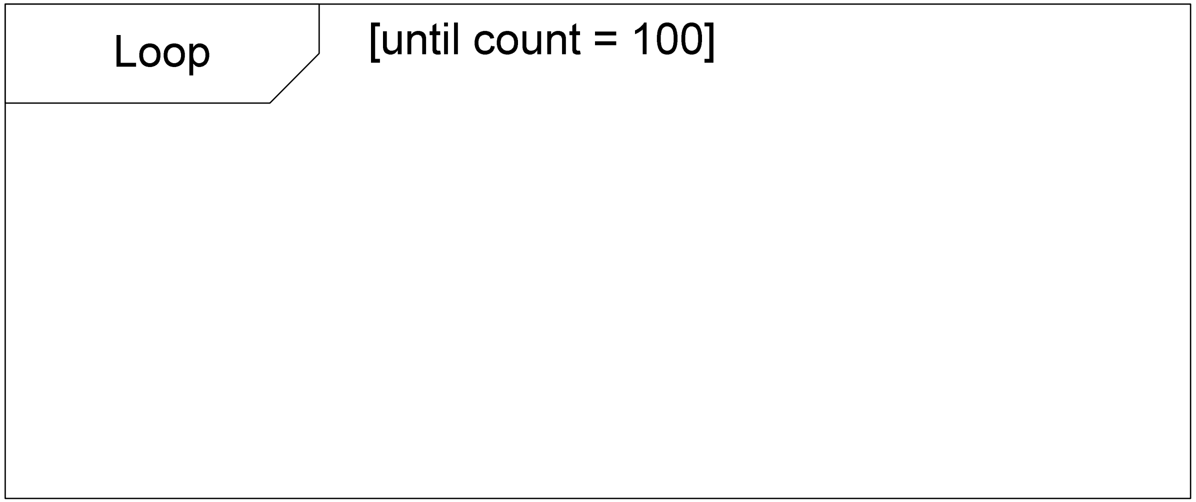
Note: No need for return messages. Stored in parameters!

Sequence Diagram: Search product (or Read product)

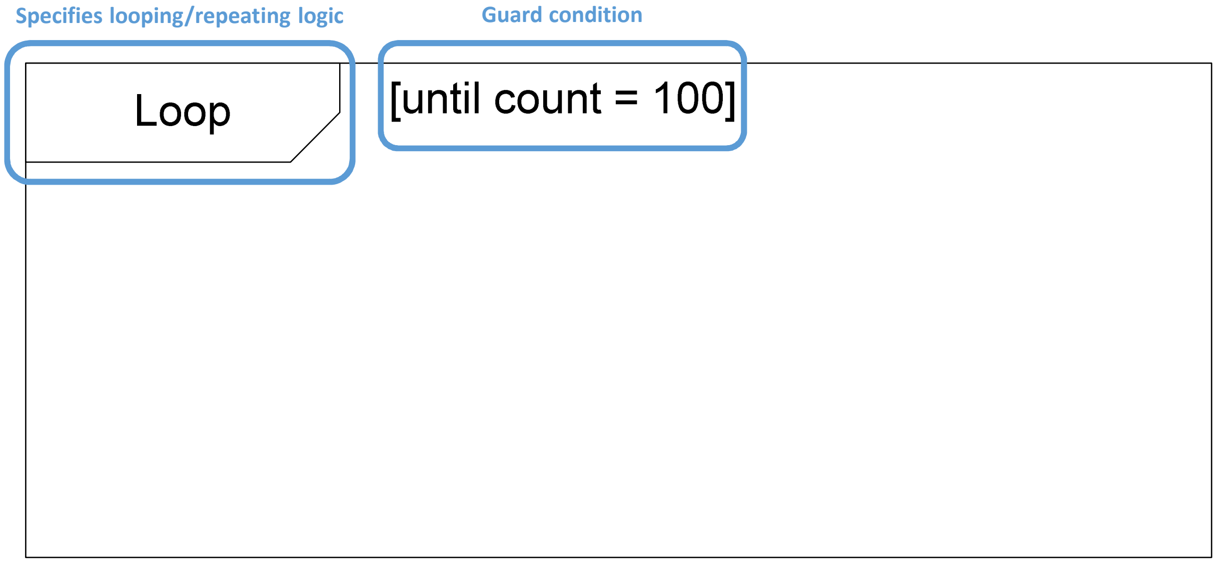
Adding more detail to sequence diagrams

- As with SSDs, processing logic is implemented on a sequence diagram using interaction frames:
  - For looping/repetition, use a Loop frame
  - For optional (true/false), use an Opt frame
  - For alternative (if-then-else), use an Alt frame
- Whatever is inside the interaction frame will only execute until (Loop) or if (Opt, Alt) a certain condition is met
- This condition is called a guard condition
- Guard conditions are indicated by square brackets, i.e. [condition]

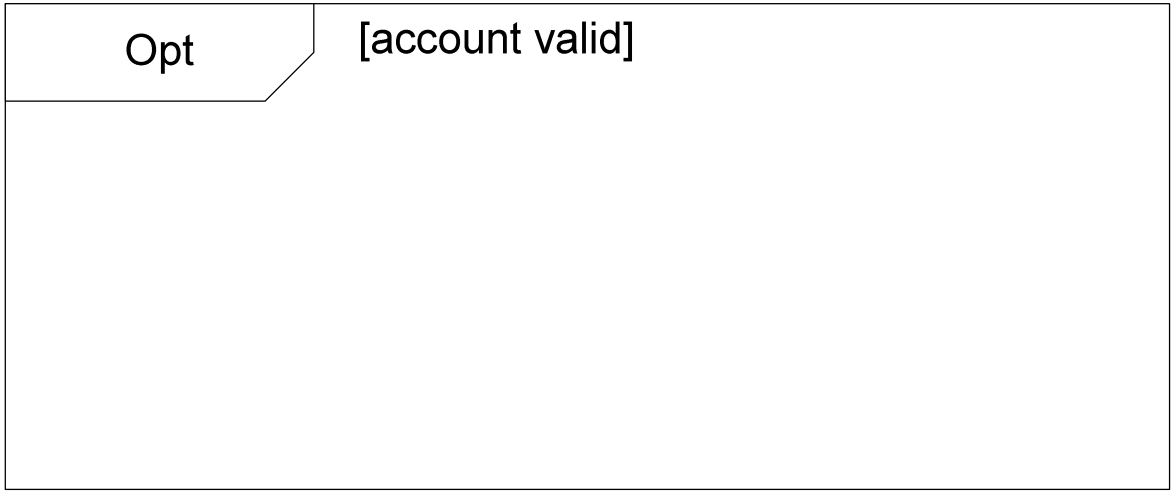
Loop frame example



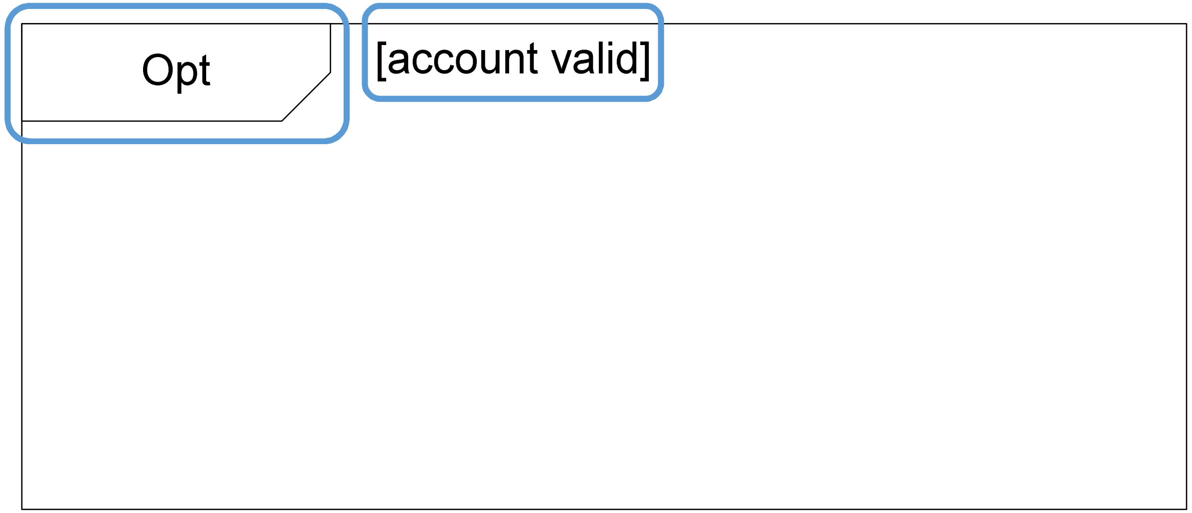
Loop frame example



Opt frame example



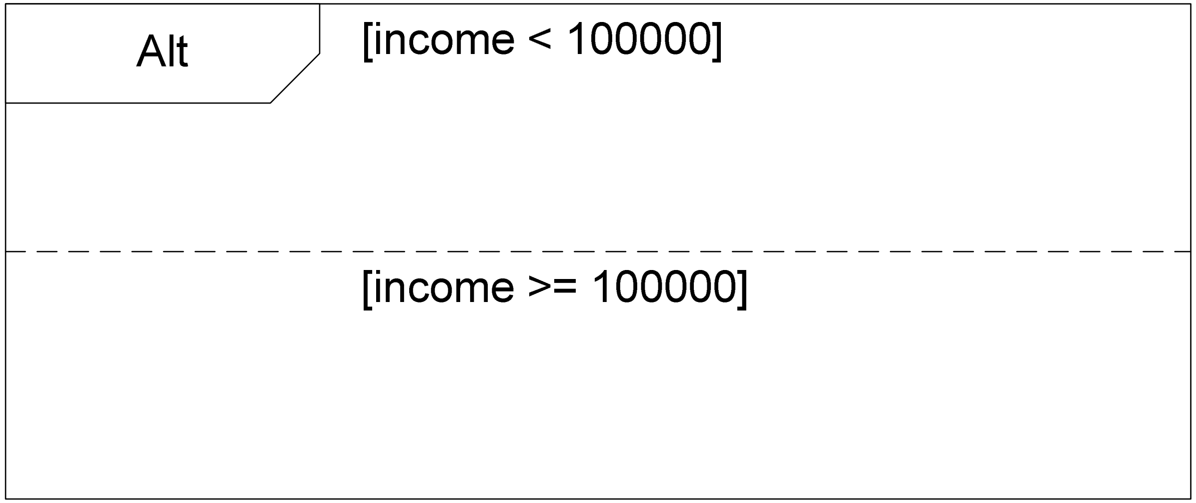
Opt frame example



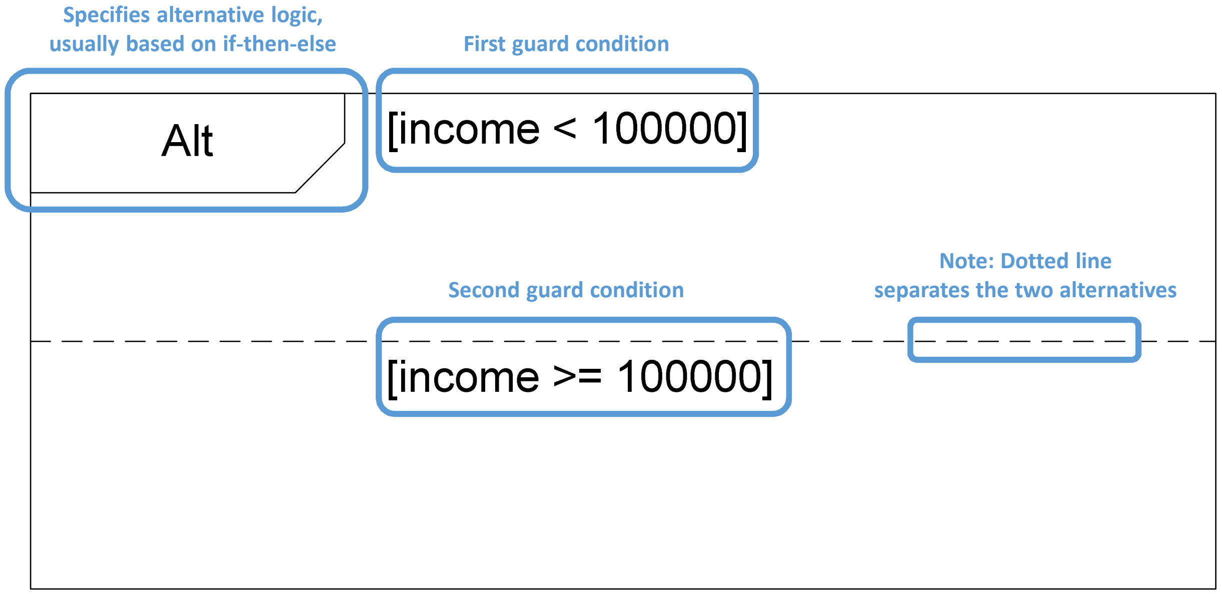
Specifies optional processing, usually based on True/False

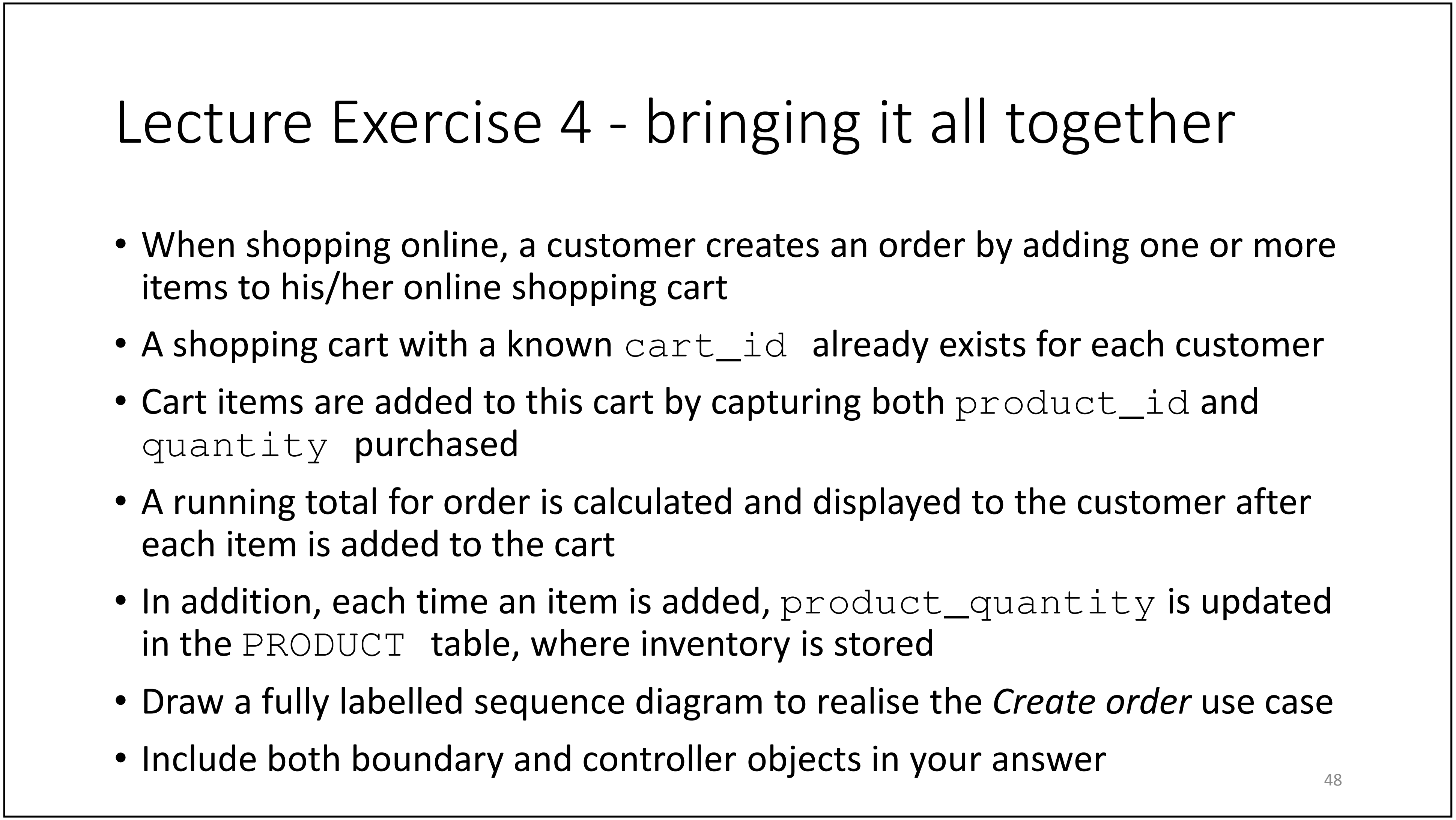
Guard condition

Alt frame example



Alt frame example



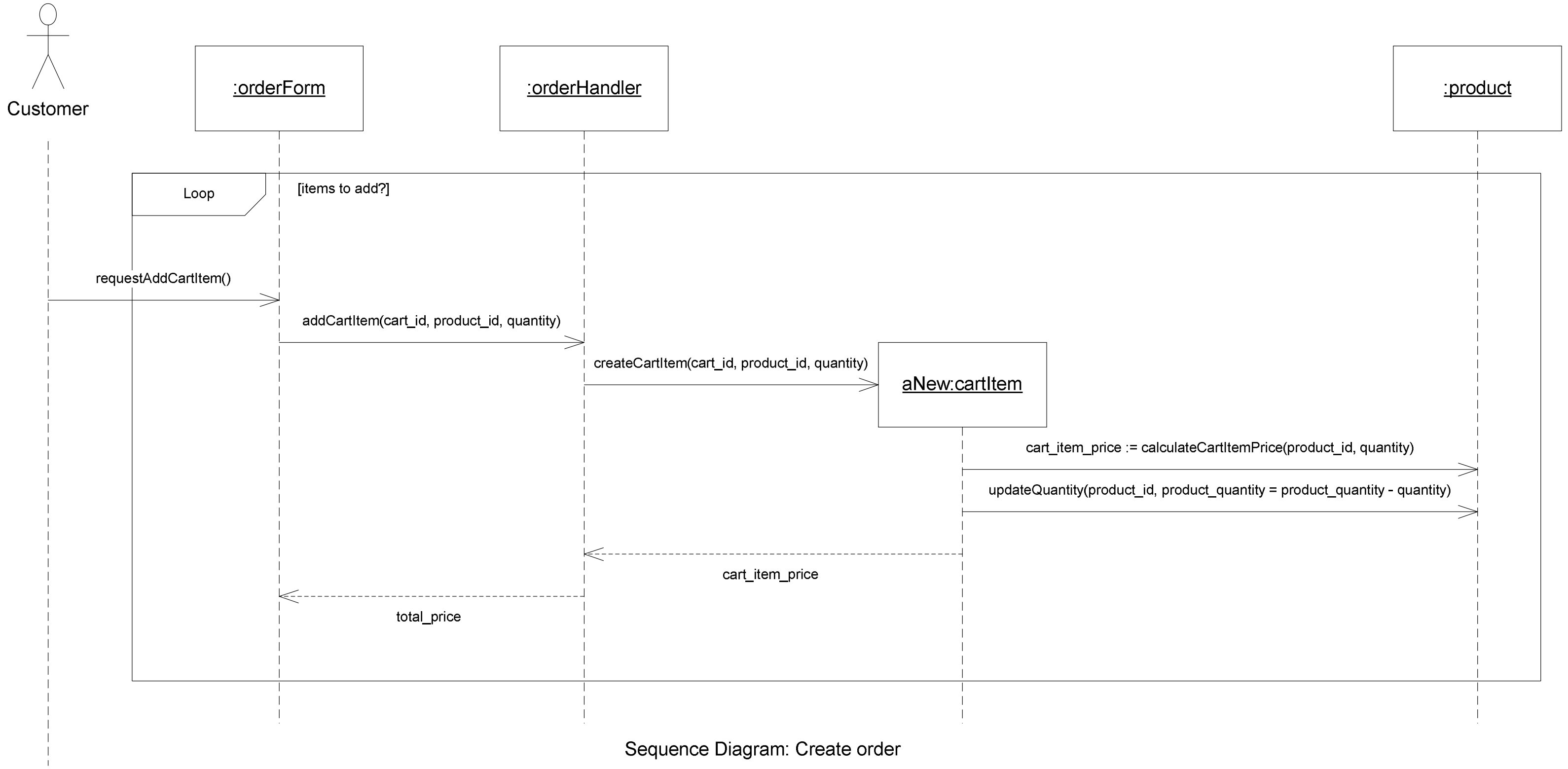


Lecture Exercise 4 - bringing it all together

- When shopping online, a customer creates an order by adding one or more items to his/her online shopping cart
- A shopping cart with a known `cart_id` already exists for each customer
- Cart items are added to this cart by capturing both `product_id` and `quantity` purchased
- A running total for order is calculated and displayed to the customer after each item is added to the cart
- In addition, each time an item is added, `product_quantity` is updated in the `PRODUCT` table, where inventory is stored
- Draw a fully labelled sequence diagram to realise the *Create order* use case
- Include both boundary and controller objects in your answer

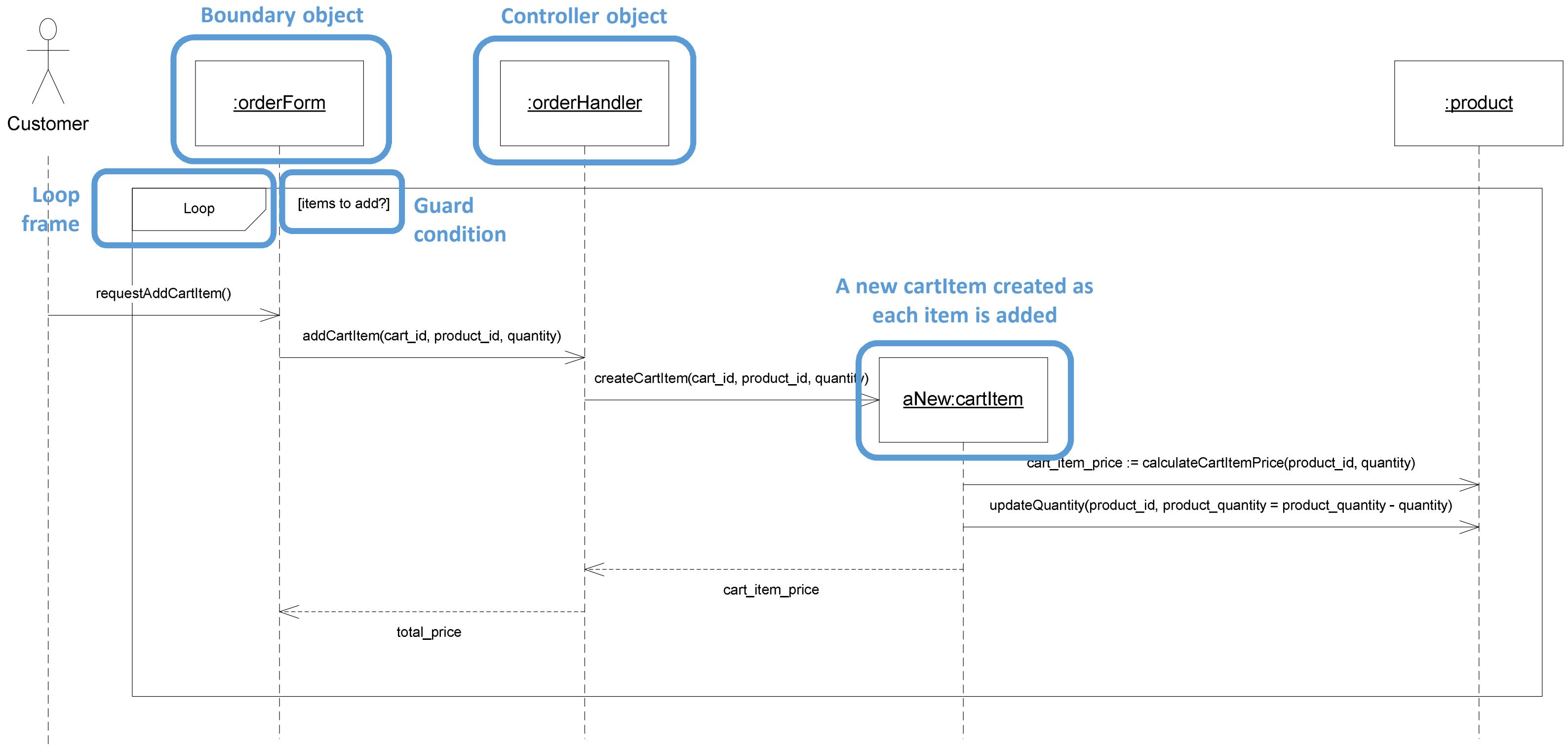
48

Lecture Exercise 4 - Suggested Solution


Customer
:orderForm
:orderHandler
:product
Loop
[items to add?]
requestAddCartItem()
addCartItem(cart_id, product_id, quantity)
createCartItem(cart_id, product_id, quantity)
aNew:cartItem
cart_item_price := calculateCartItemPrice(product_id, quantity)
updateQuantity(product_id, product_quantity = product_quantity - quantity)
cart_item_price
total_price

Sequence Diagram: Create order

Lecture Exercise 4 - Suggested Solution



Boundary object

Controller object

Customer

:orderForm

:orderHandler

:product

Loop frame

Loop

[items to add?]

Guard condition

A new cartItem created as each item is added

requestAddCartItem()

addCartItem(cart_id, product_id, quantity)

createCartItem(cart_id, product_id, quantity)

aNew:cartItem

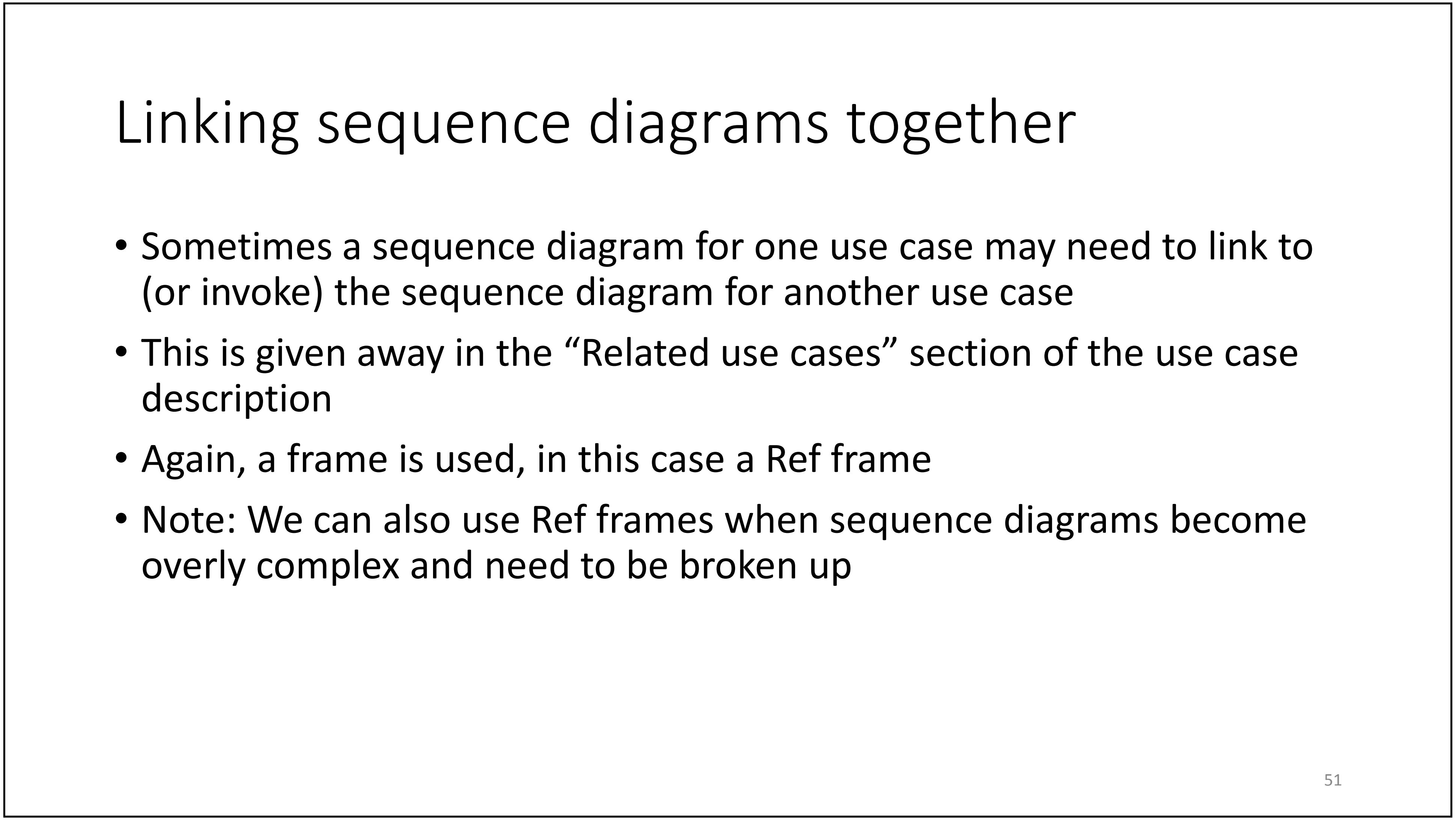
cart_item_price := calculateCartItemPrice(product_id, quantity)

updateQuantity(product_id, product_quantity = product_quantity - quantity)

cart_item_price

total_price

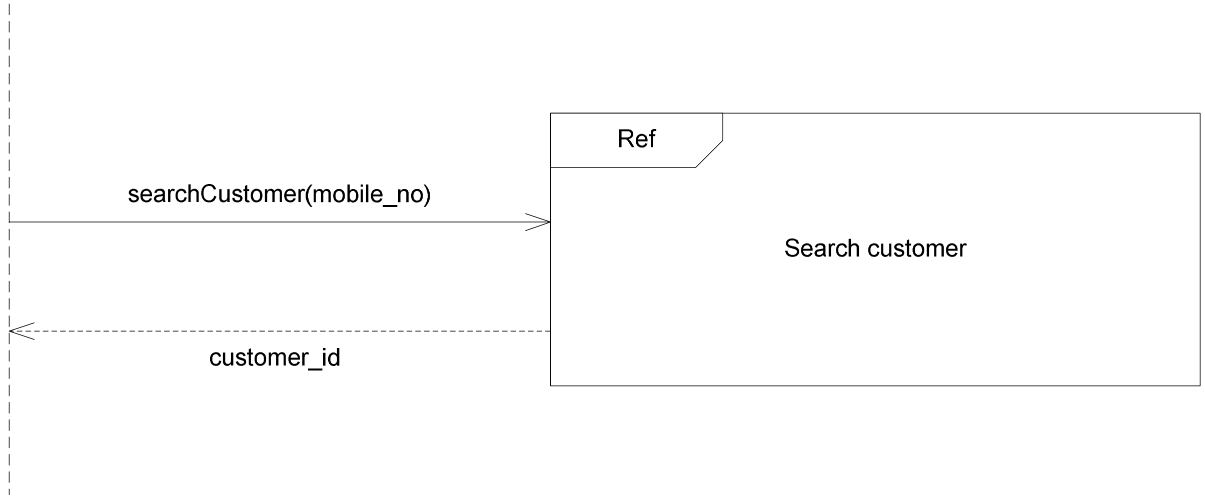
Sequence Diagram: Create order



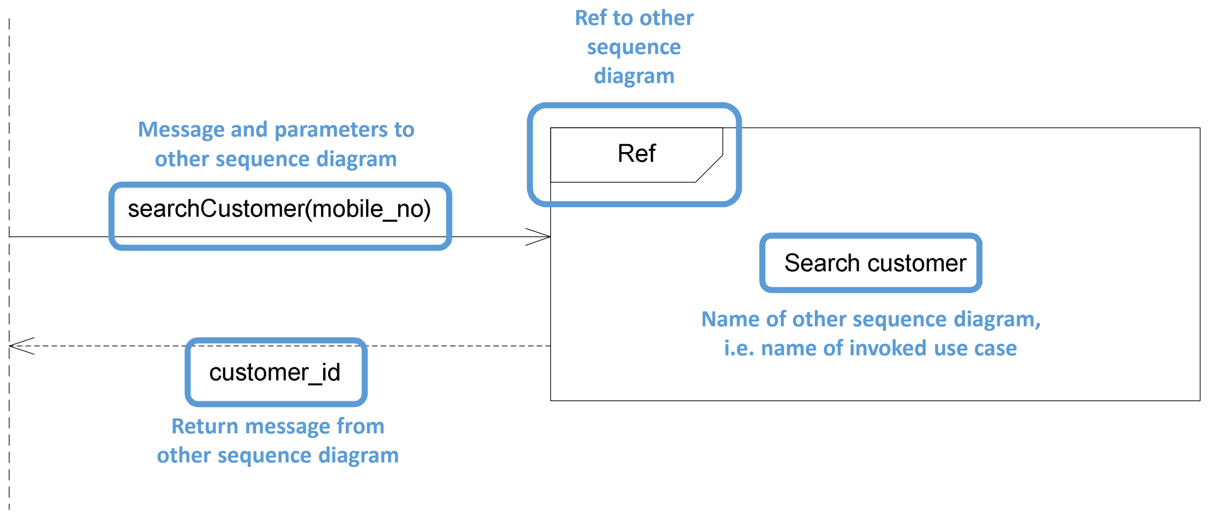
Linking sequence diagrams together

- Sometimes a sequence diagram for one use case may need to link to (or invoke) the sequence diagram for another use case
- This is given away in the “Related use cases” section of the use case description
- Again, a frame is used, in this case a Ref frame
- Note: We can also use Ref frames when sequence diagrams become overly complex and need to be broken up

Ref frame example



Ref frame example



Ref to other sequence diagram

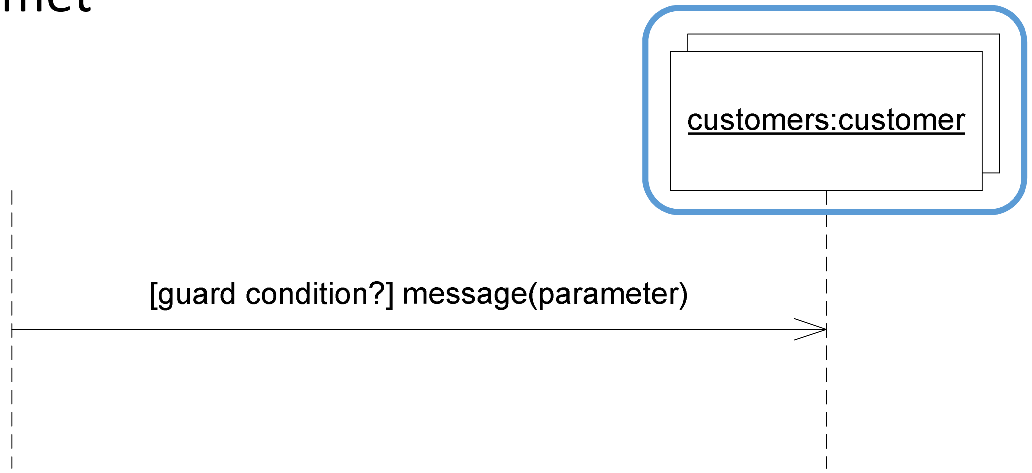
Message and parameters to other sequence diagram

Name of other sequence diagram, i.e. name of invoked use case

Return message from other sequence diagram

More useful notation: Multi-objects (collections)

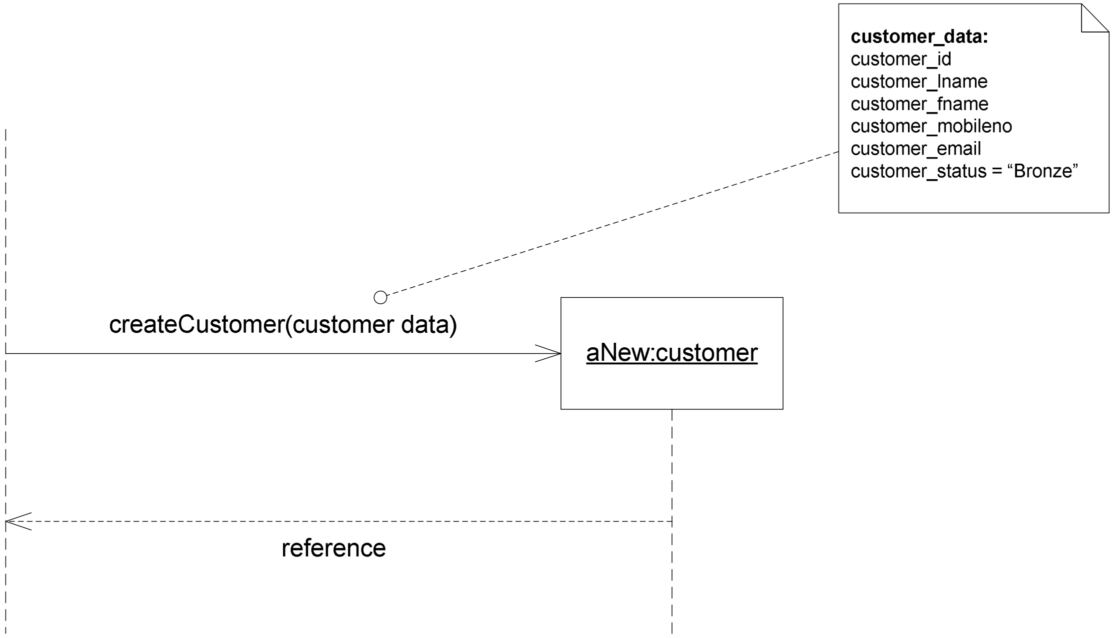
- An actor can interact with multiple elements of a collection using a multi-object
- The message goes to each element of the collection until the guard condition is met



This is a collection of customer objects called "customers"

More useful notation: UML notes

- UML notes allow us to produce less cluttered sequence diagrams



Examinable materials for this topic

- These lecture slides and lecture exercises
- Satzinger *et al.*, Chapter 11, pages 301 to 323