Systems Design and Databases (CIS1018-N)

Week 3

Sequence and Class Diagrams

Teaching Team

Module Leader & Lecturer: Dr Yar Muhammad
Email: Yar.Muhammad@tees.ac.uk
Office: G0.39 (Greig Building)

Tutor:
- Dr Mengda He
- Mr Mansha Nawaz
- Mr Vishalkumar Thakor

My Academic Hub Time Slots:
Monday 10:00 - 11:00 in Room IT1.13 (Europa Building)
Tuesday 13:00 - 14:00 in Room IT1.13 (Europa Building)

- See Blackboard Ultra for online materials: https://bb.tees.ac.uk/

Lectures & IT Labs

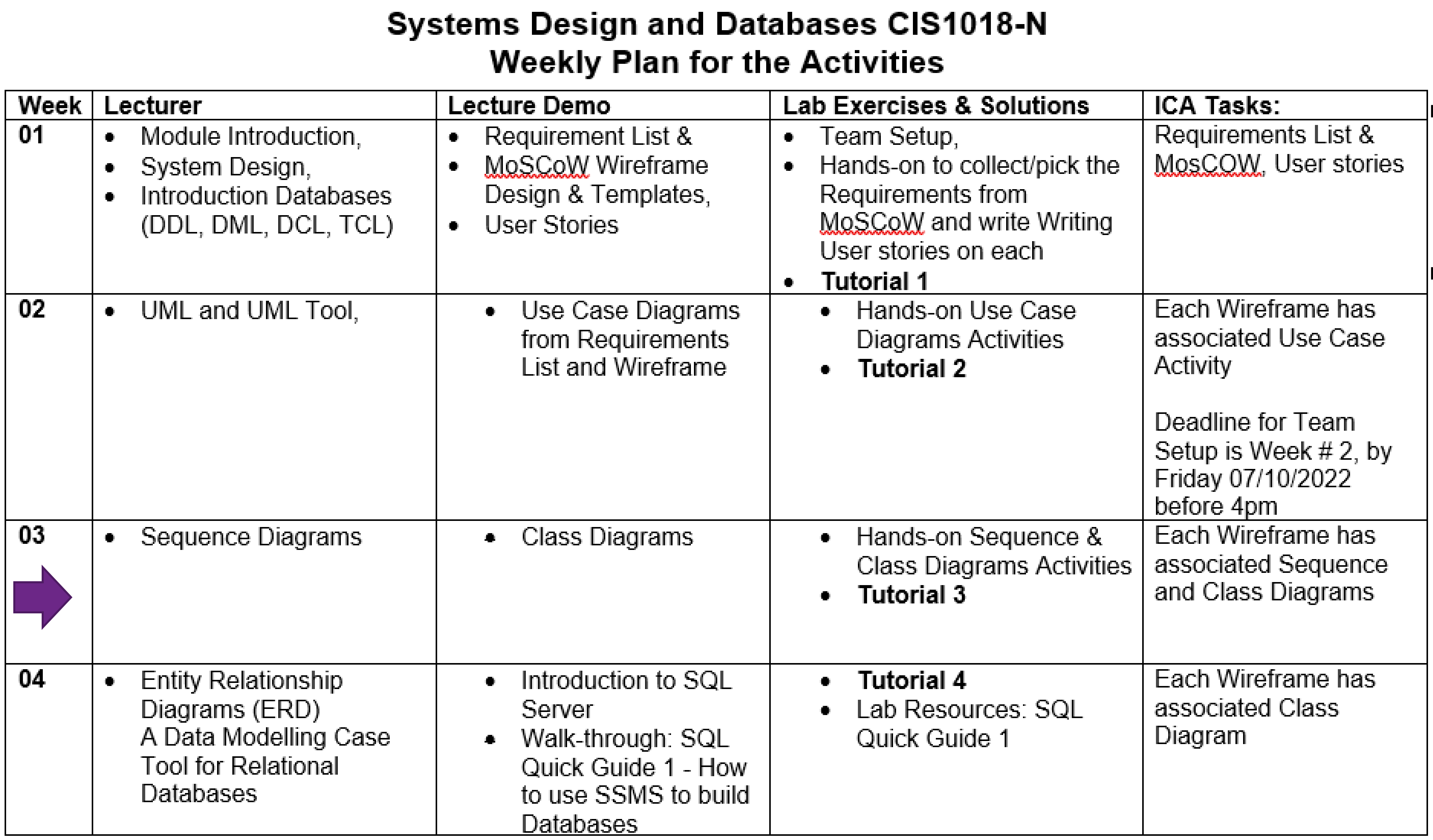
Lectures – Dr Yar Muhammad	Tuesdays @ 2-3 pm	Thursdays @ 1-2 pm
Week 1 – Week 12	CL1.87	

Tutor – Thursday	IT Lab Session Room #: IT2.42
Mr Mansha Nawaz M.Nawaz@tees.ac.uk	Time: 3 – 5 pm

Tutor – Friday	IT Lab Session Room #: OL3
Dr Yar Muhammad Yar.Muhammad@tees.ac.uk	Time: 9 – 11 am & 11 am – 1 pm
Dr Mengda He M.He@tees.ac.uk	Time: 9 – 11 am
Mr Vishalkumar Thakor V.Thakor@tees.ac.uk	Time: 11 am – 1 pm & 1 – 3 pm
Mr Mansha Nawaz M.Nawaz@tees.ac.uk	Time: 1 – 3 pm

Schedule 1/3

Systems Design - UML



Systems Design and Databases CIS1018-N
Weekly Plan for the Activities

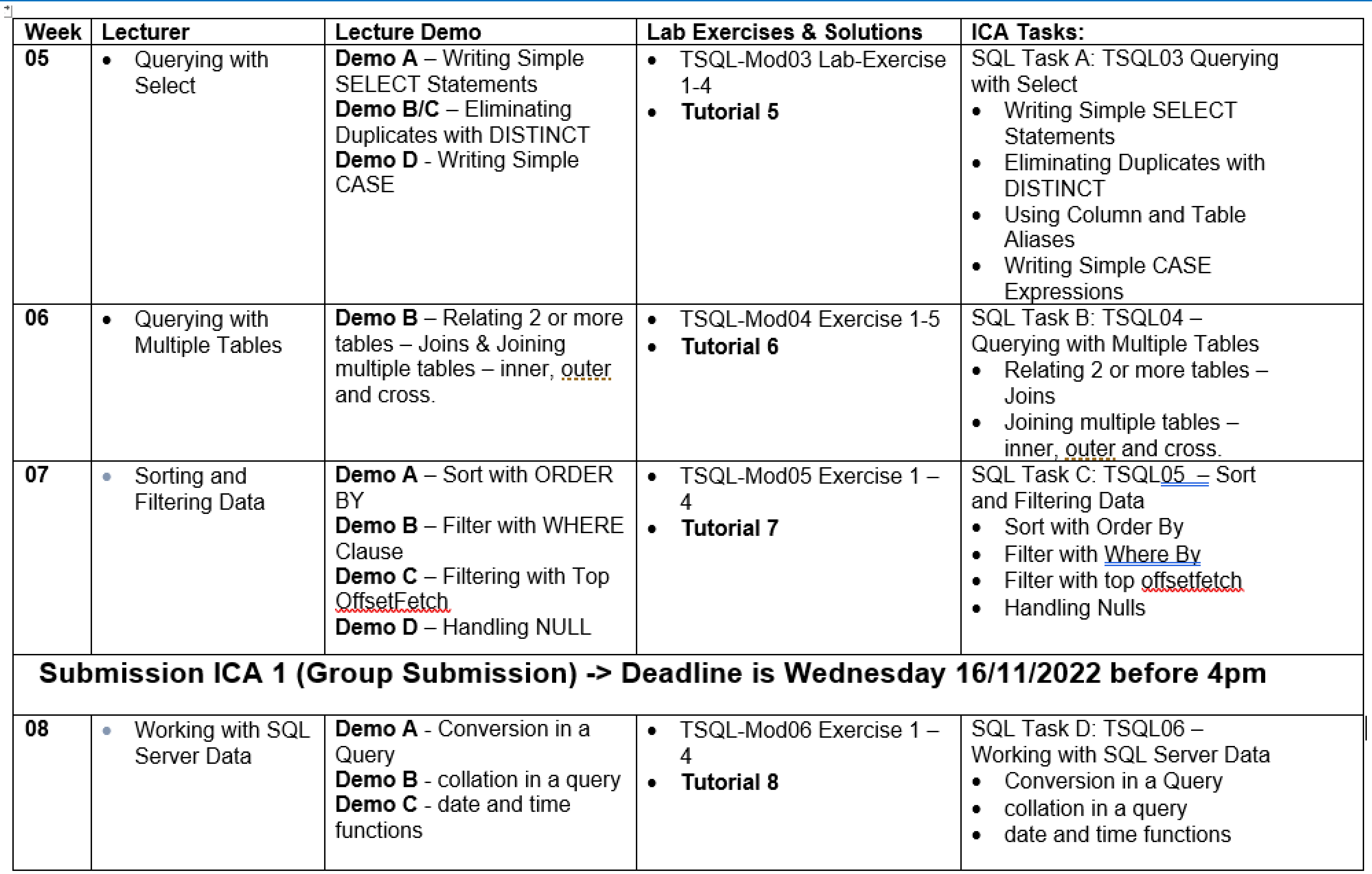
Week	Lecturer	Lecture Demo	Lab Exercises & Solutions	ICA Tasks:
01	• Module Introduction, • System Design, • Introduction Databases (DDL, DML, DCL, TCL)	• Requirement List & • MoSCoW Wireframe Design & Templates, • User Stories	• Team Setup, • Hands-on to collect/pick the Requirements from MoSCoW and write Writing User stories on each • **Tutorial 1**	Requirements List & MosCOW, User stories
02	• UML and UML Tool,	• Use Case Diagrams from Requirements List and Wireframe	• Hands-on Use Case Diagrams Activities • **Tutorial 2**	Each Wireframe has associated Use Case Activity Deadline for Team Setup is Week # 2, by Friday 07/10/2022 before 4pm
03	• Sequence Diagrams	• Class Diagrams	• Hands-on Sequence & Class Diagrams Activities • **Tutorial 3**	Each Wireframe has associated Sequence and Class Diagrams
04	• Entity Relationship Diagrams (ERD) A Data Modelling Case Tool for Relational Databases	• Introduction to SQL Server • Walk-through: SQL Quick Guide 1 - How to use SSMS to build Databases	• **Tutorial 4** • Lab Resources: SQL Quick Guide 1	Each Wireframe has associated Class Diagram

Analysis

Design

Schedule 2/3

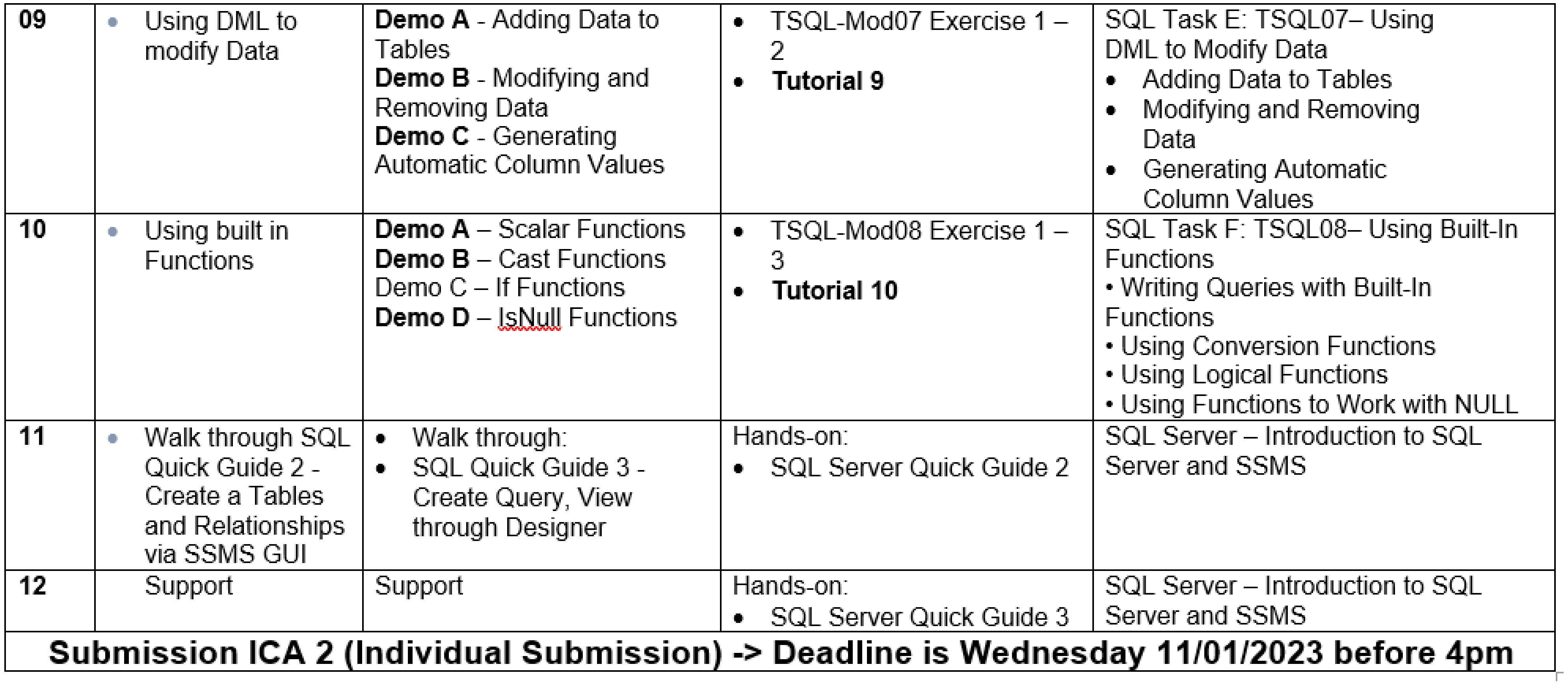
Databases - TSQL



Week	Lecturer	Lecture Demo	Lab Exercises & Solutions	ICA Tasks:
05	• Querying with Select	**Demo A** – Writing Simple SELECT Statements **Demo B/C** – Eliminating Duplicates with DISTINCT **Demo D** - Writing Simple CASE	• TSQL-Mod03 Lab-Exercise 1-4 • **Tutorial 5**	SQL Task A: TSQL03 Querying with Select • Writing Simple SELECT Statements • Eliminating Duplicates with DISTINCT • Using Column and Table Aliases • Writing Simple CASE Expressions
06	• Querying with Multiple Tables	**Demo B** – Relating 2 or more tables – Joins & Joining multiple tables – inner, outer and cross.	• TSQL-Mod04 Exercise 1-5 • **Tutorial 6**	SQL Task B: TSQL04 – Querying with Multiple Tables • Relating 2 or more tables – Joins • Joining multiple tables – inner, outer and cross.
07	• Sorting and Filtering Data	**Demo A** – Sort with ORDER BY **Demo B** – Filter with WHERE Clause **Demo C** – Filtering with Top OffsetFetch **Demo D** – Handling NULL	• TSQL-Mod05 Exercise 1 – 4 • **Tutorial 7**	SQL Task C: TSQL05 – Sort and Filtering Data • Sort with Order By • Filter with Where By • Filter with top offsetfetch • Handling Nulls
Submission ICA 1 (Group Submission) -> Deadline is Wednesday 16/11/2022 before 4pm				
08	• Working with SQL Server Data	**Demo A** - Conversion in a Query **Demo B** - collation in a query **Demo C** - date and time functions	• TSQL-Mod06 Exercise 1 – 4 • **Tutorial 8**	SQL Task D: TSQL06 – Working with SQL Server Data • Conversion in a Query • collation in a query • date and time functions

Schedule 3/3

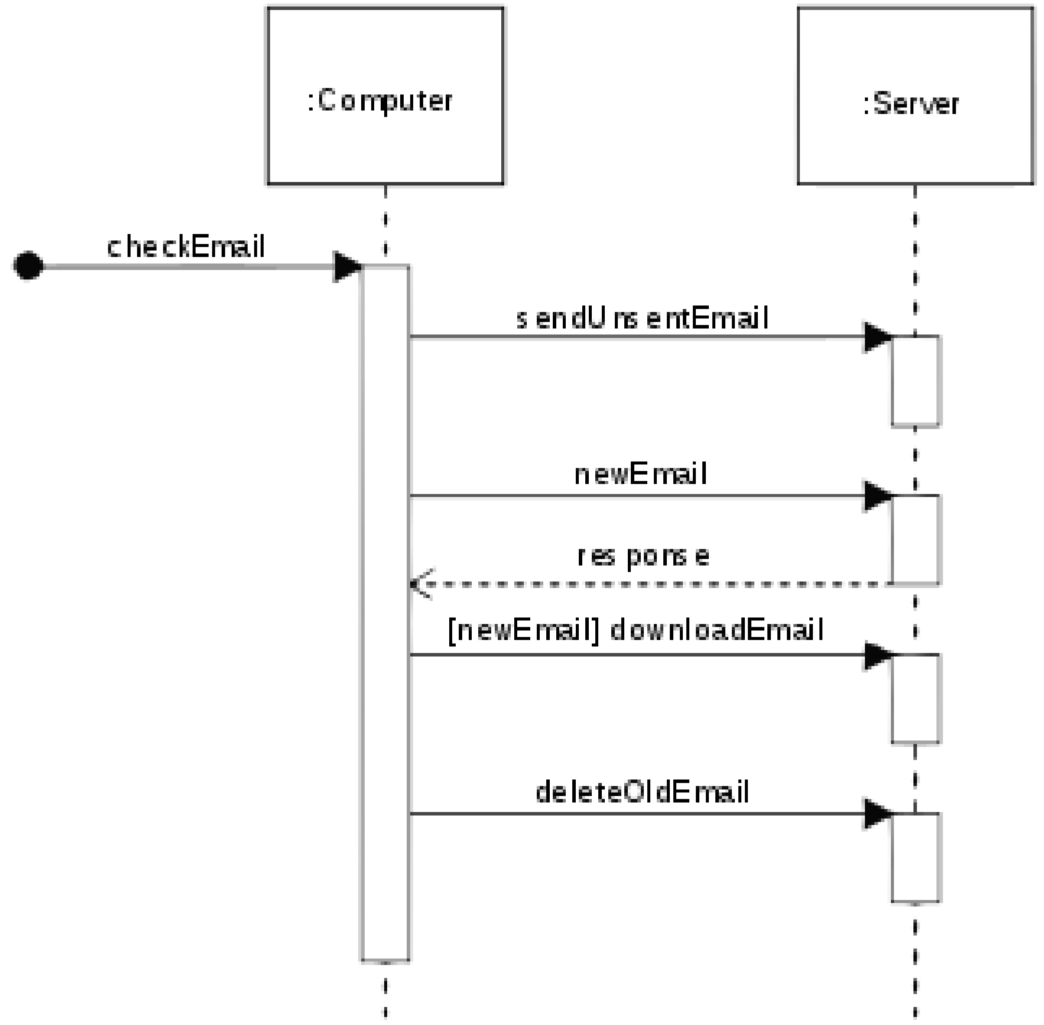
Databases - TSQL



09	• Using DML to modify Data	**Demo A** - Adding Data to Tables **Demo B** - Modifying and Removing Data **Demo C** - Generating Automatic Column Values	• TSQL-Mod07 Exercise 1 – 2 • **Tutorial 9**	SQL Task E: TSQL07– Using DML to Modify Data • Adding Data to Tables • Modifying and Removing Data • Generating Automatic Column Values
10	• Using built in Functions	**Demo A** – Scalar Functions **Demo B** – Cast Functions Demo C – If Functions **Demo D** – IsNull Functions	• TSQL-Mod08 Exercise 1 – 3 • **Tutorial 10**	SQL Task F: TSQL08– Using Built-In Functions • Writing Queries with Built-In Functions • Using Conversion Functions • Using Logical Functions • Using Functions to Work with NULL
11	• Walk through SQL Quick Guide 2 - Create a Tables and Relationships via SSMS GUI	• Walk through: • SQL Quick Guide 3 - Create Query, View through Designer	Hands-on: • SQL Server Quick Guide 2	SQL Server – Introduction to SQL Server and SSMS
12	Support	Support	Hands-on: • SQL Server Quick Guide 3	SQL Server – Introduction to SQL Server and SSMS
Submission ICA 2 (Individual Submission) -> Deadline is Wednesday 11/01/2023 before 4pm				

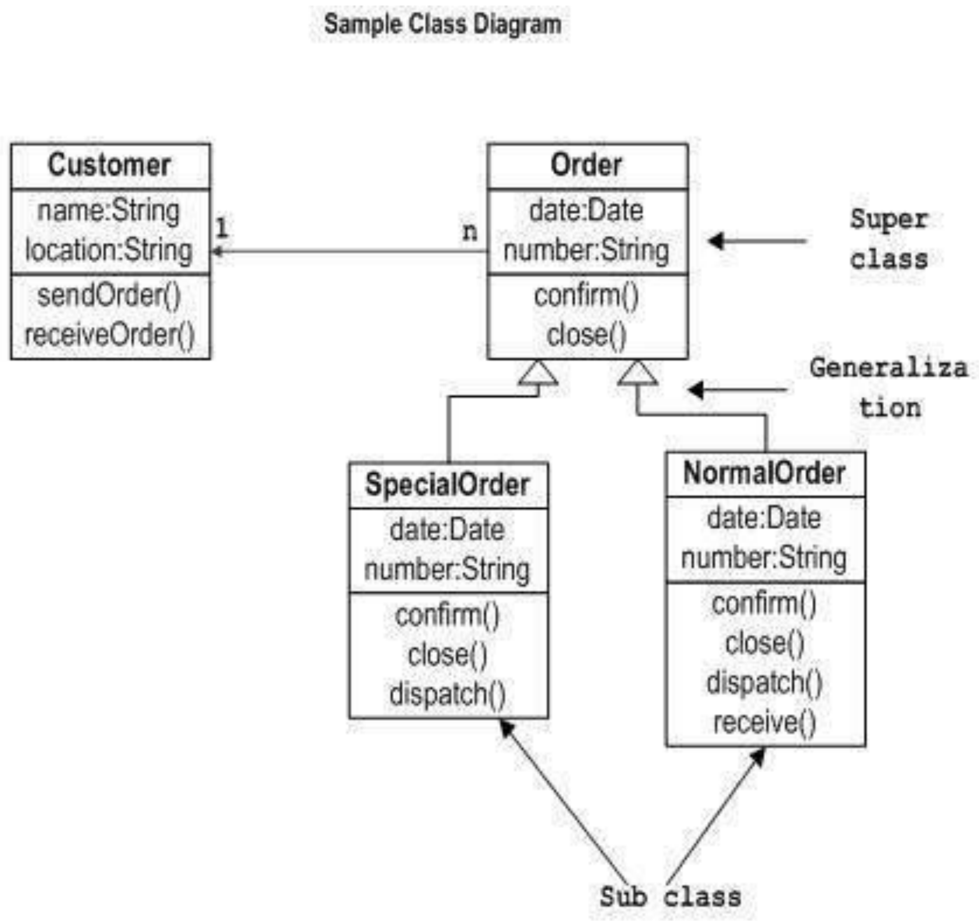
What is Sequence Diagram?

- A sequence diagram is a Unified Modeling Language (UML) diagram that **illustrates the sequence of messages between objects in an interaction.**
- A sequence diagram **consists of a group of objects that are represented by lifelines**, and the messages that they exchange over time during the interaction.



What is Class Diagram

- Class diagram **describes the attributes and operations** of a class and also the constraints imposed on the system.
- The class diagrams are **widely used in the modeling of object oriented systems** because they are the only UML diagrams, which can be **mapped directly with object-oriented languages**.

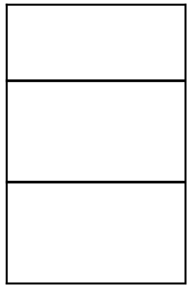


What are the class or objects or things we can identify from our Use Case?

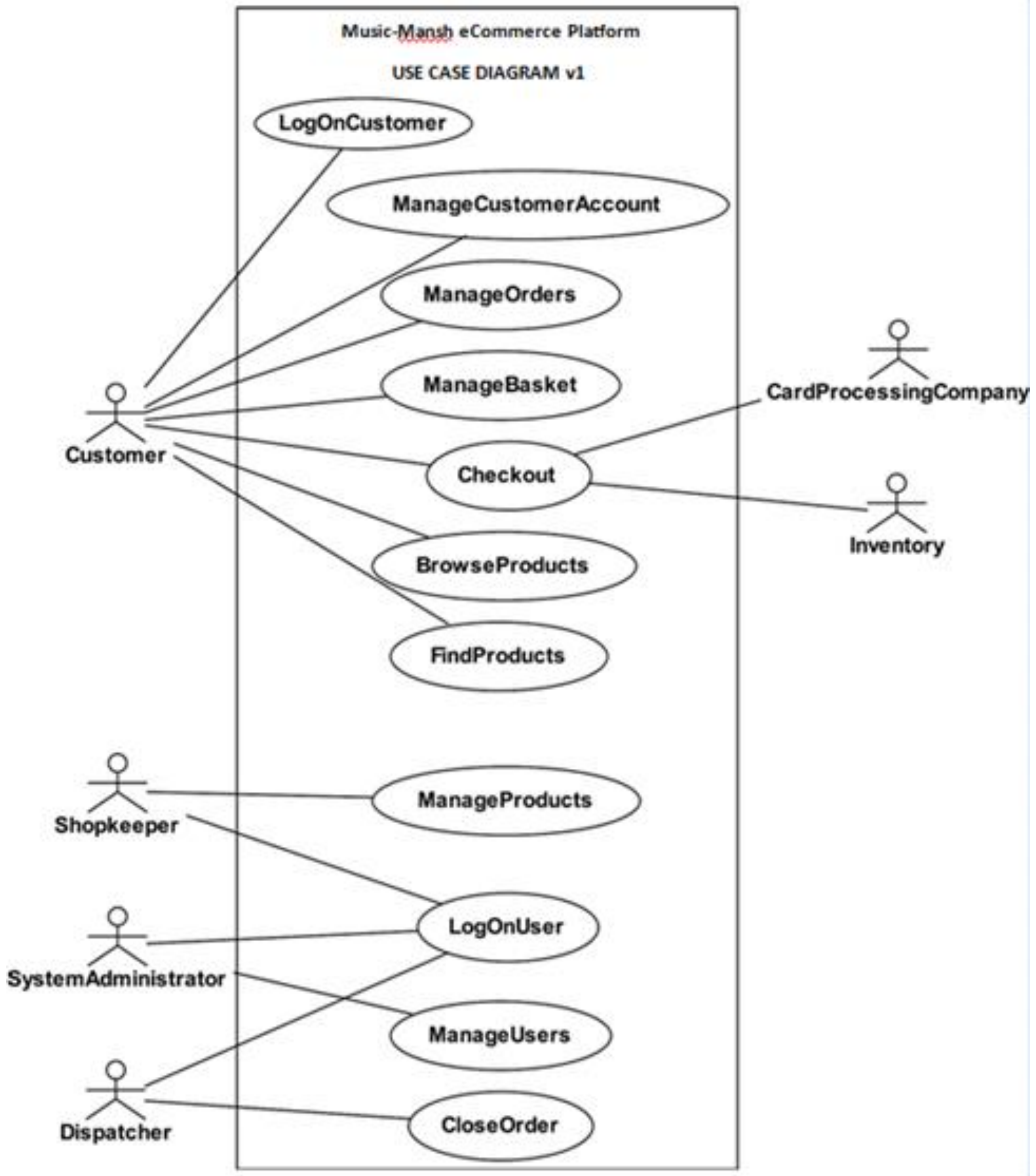
Do we need to record a list from a Database?
Can we identify the **class or object with a unique key**?

Class Objects (Entitles or Things)

- Customers
- Orders
- Basket
- Products
- Users



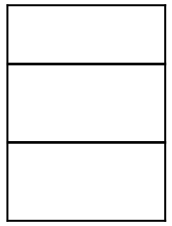
Analysis Class



Identifying Class Objects and Sequence Diagrams

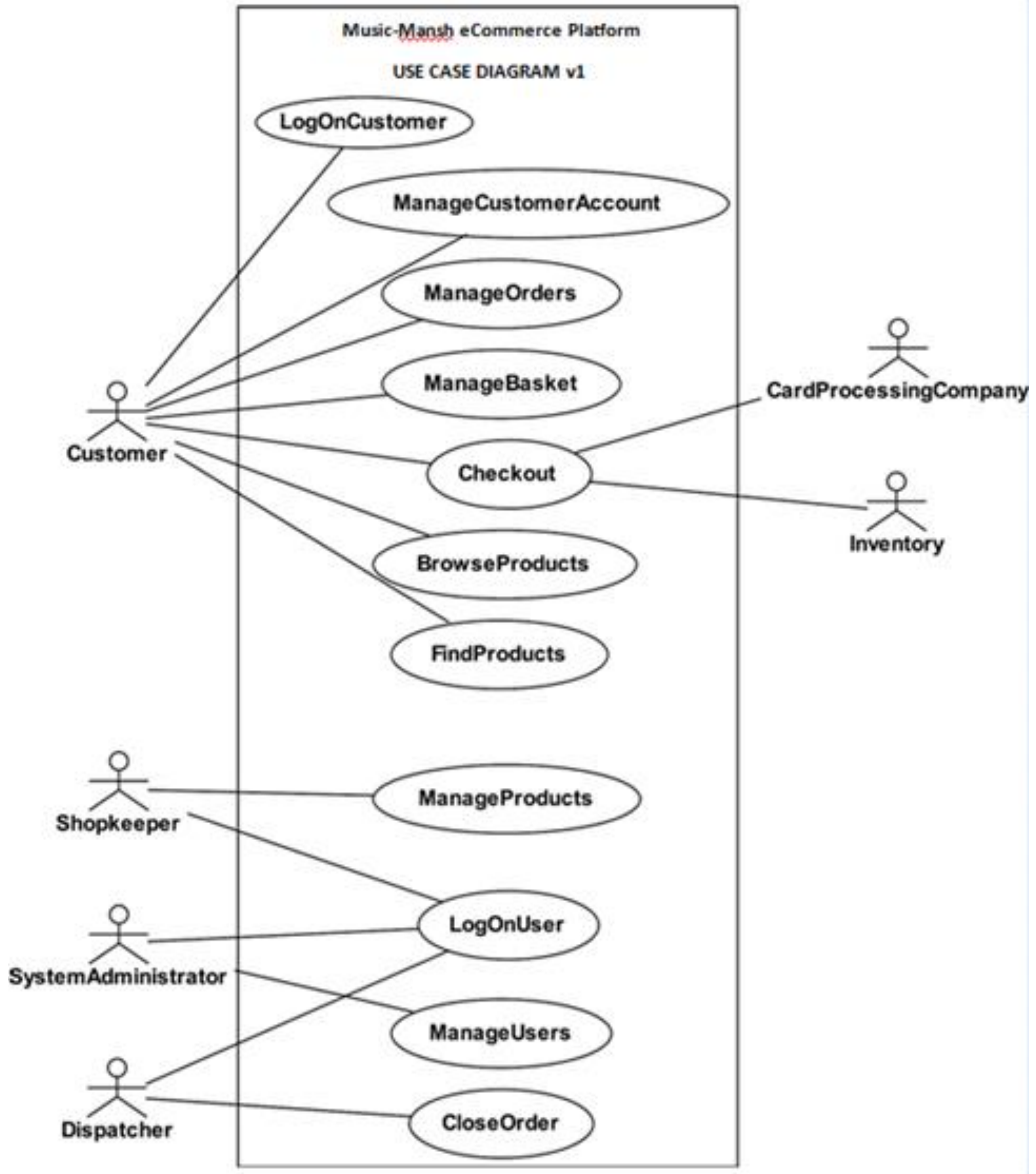
Class Objects (Entitles or Things)

- Customers
- Orders
- Basket
- Products
- Users



Analysis class

- We do not model all Use Cases as a sequence diagram.
- Try to choose a main sequence to follow from the class or object prospective.
- Focus on communication paths such as:
  - Customer adding items to a basket and placing an order.
  - Customer making a payment (needs to interact with Bank system)
  - Users login
  - We can map many sequences given time but focus on 1 or 2 per team member.

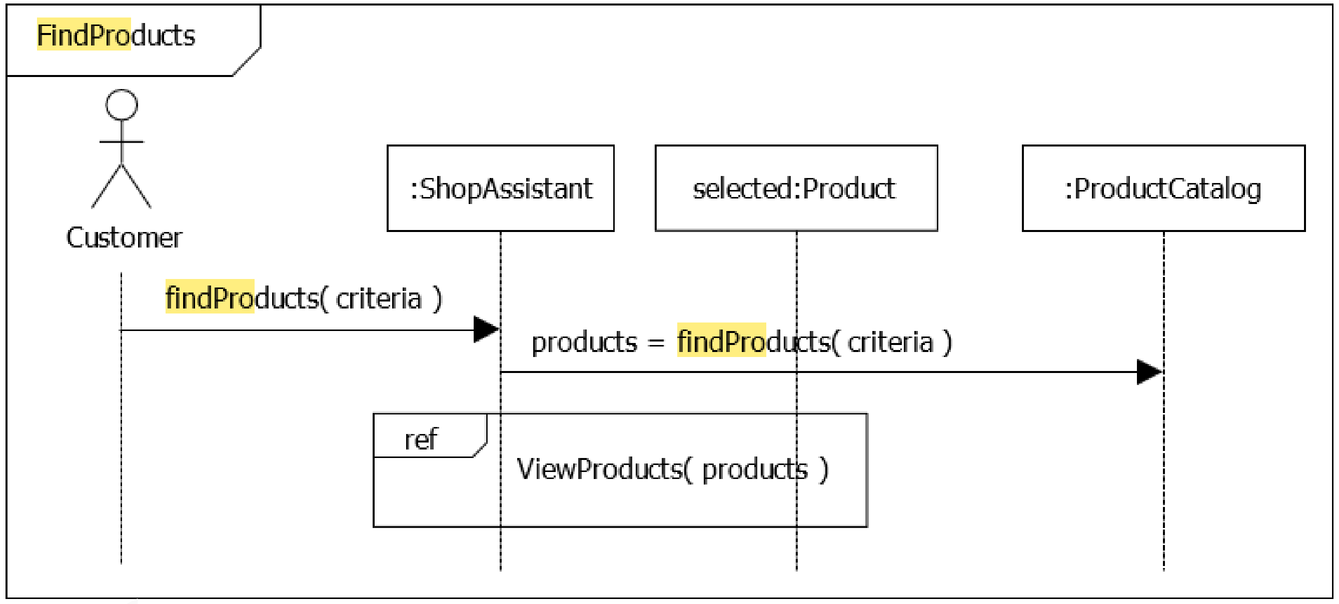


FindProduct – Sequence Diagram

Use Case Descriptors to Sequence Diagrams

Use case: FindProduct
ID: 3
Brief description: The system finds some products based on Customer search criteria and displays them to the Customer.
Actors: Customer
Preconditions: None.
Main flow: 1. The use case starts when the Customer selects "find product". 2. The system asks the Customer for search criteria. 3. The Customer enters the requested criteria. 4. The system searches for products that match the Customer's criteria. 5. For each product found   1. The system displays a thumbnail sketch of the product.   2. The system displays a summary of the product details.   3. The system displays the product price.
Postconditions: None.
Alternative flows: NoProductsFound

"Method" "Class" and "Object"

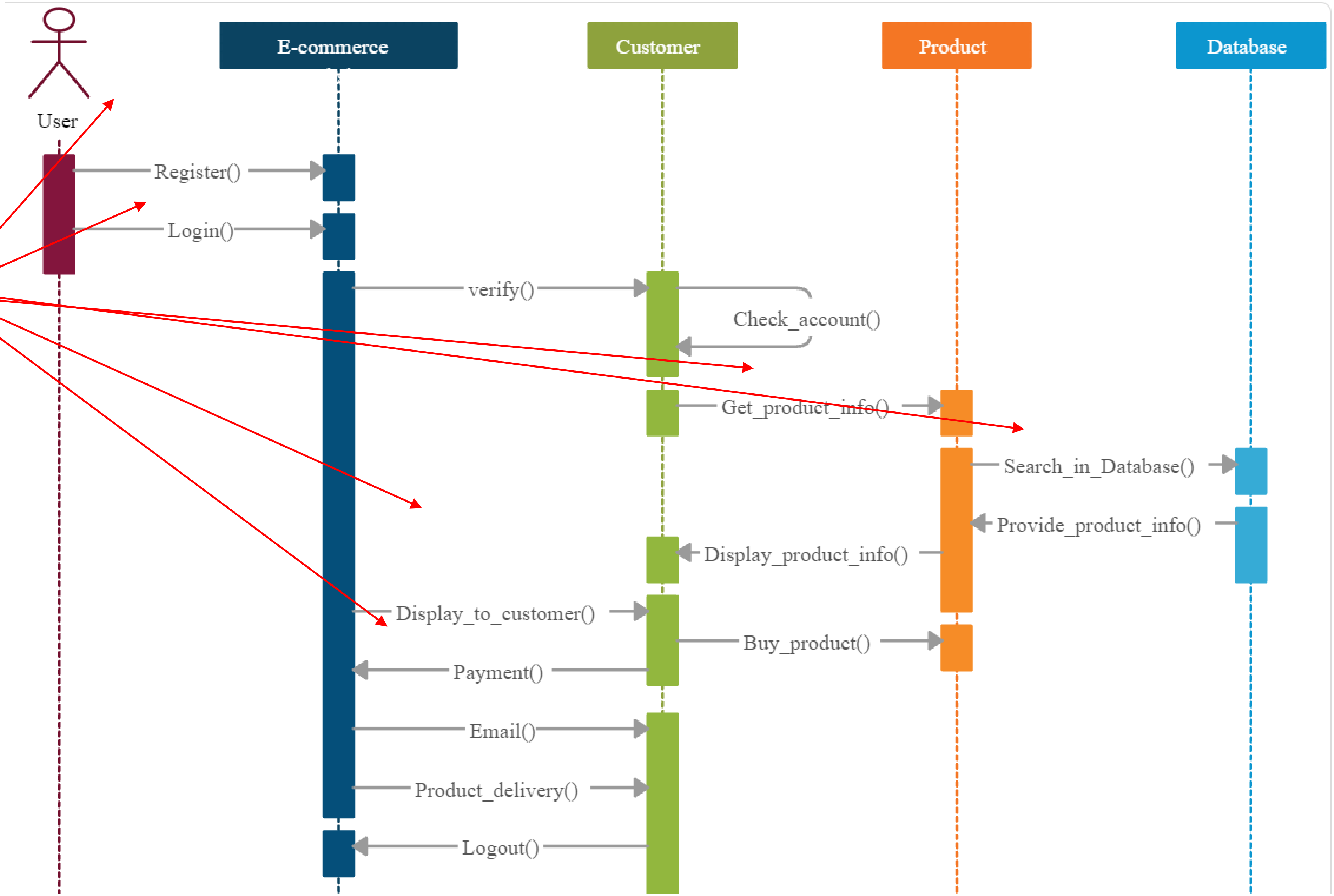


Sequence diagram Example: eCommerce platform overview

1. Model simple flow of Objects through our Systems Design

2. We end up with simple structures to ultimately code as programable :
methods, class and objects

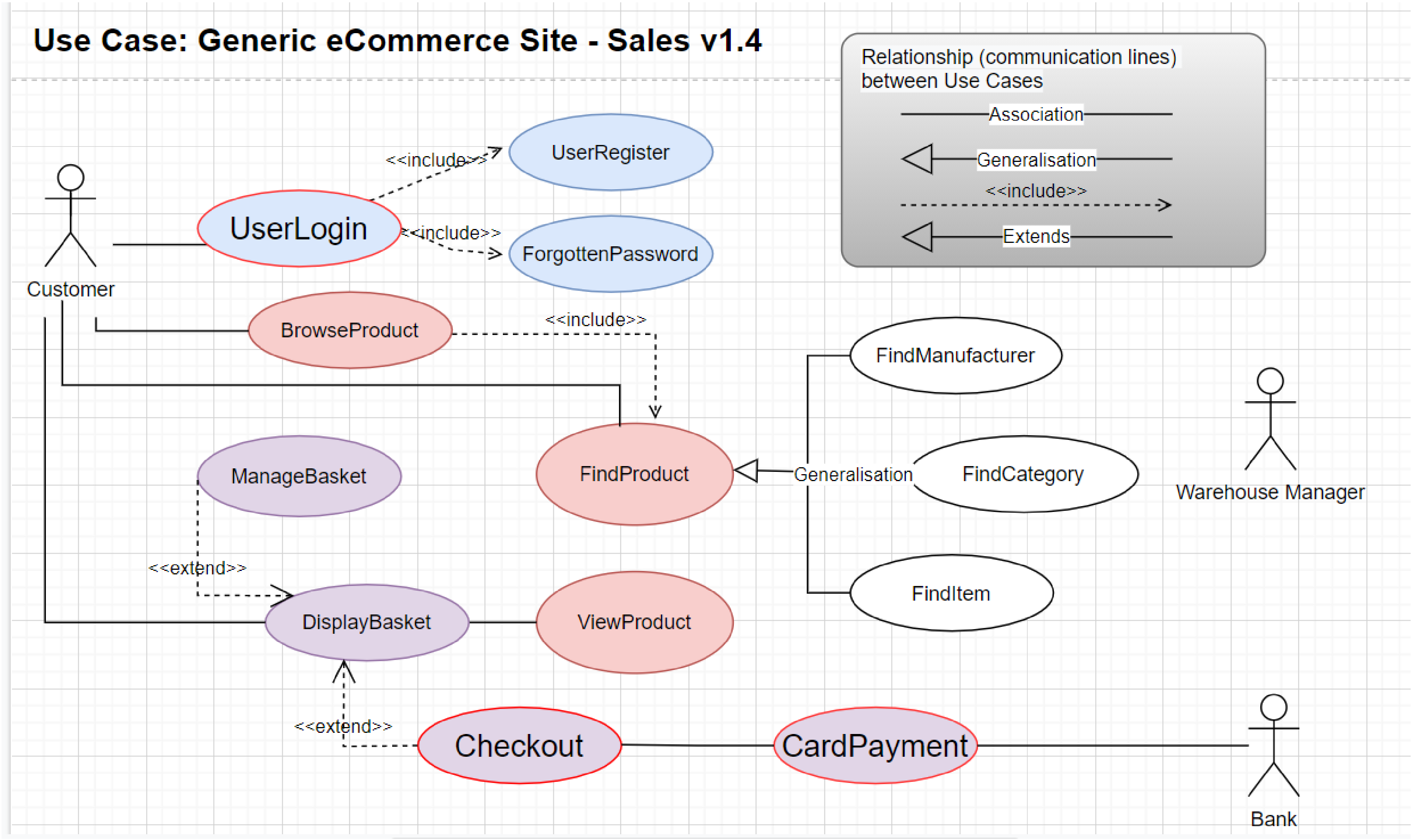
3. Programs components: methods, class making turning our USE CASE objects into CODE
Registration
Login
DisplayCustomer
Payment
OrderConfirmationEmail
ProductDelivery
Product
SearchProduct



Sequence diagram: Example from Use case to Sequence Diagram 1/3

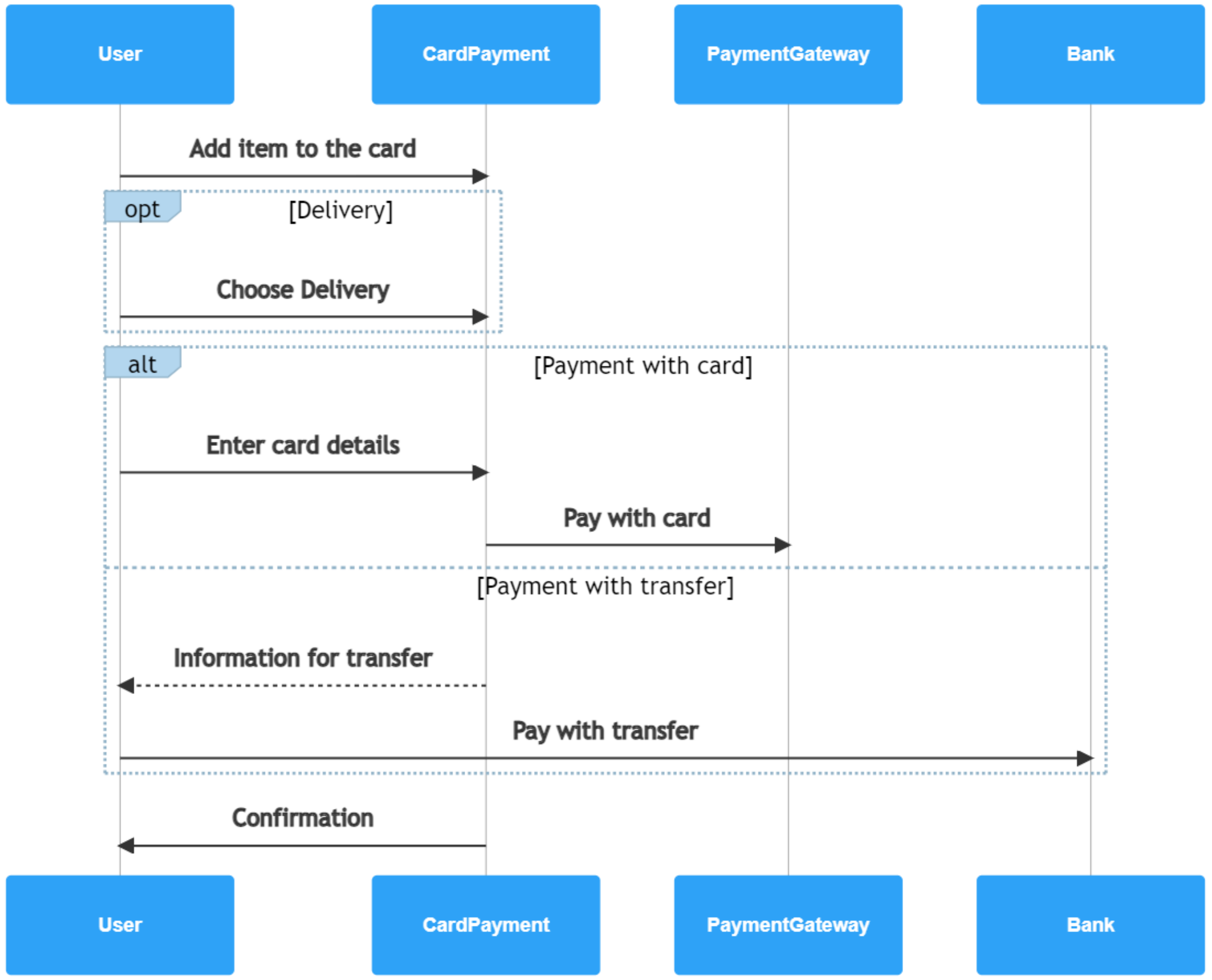
Use Case :

- CardPayment page
- User login page
- Checkout page



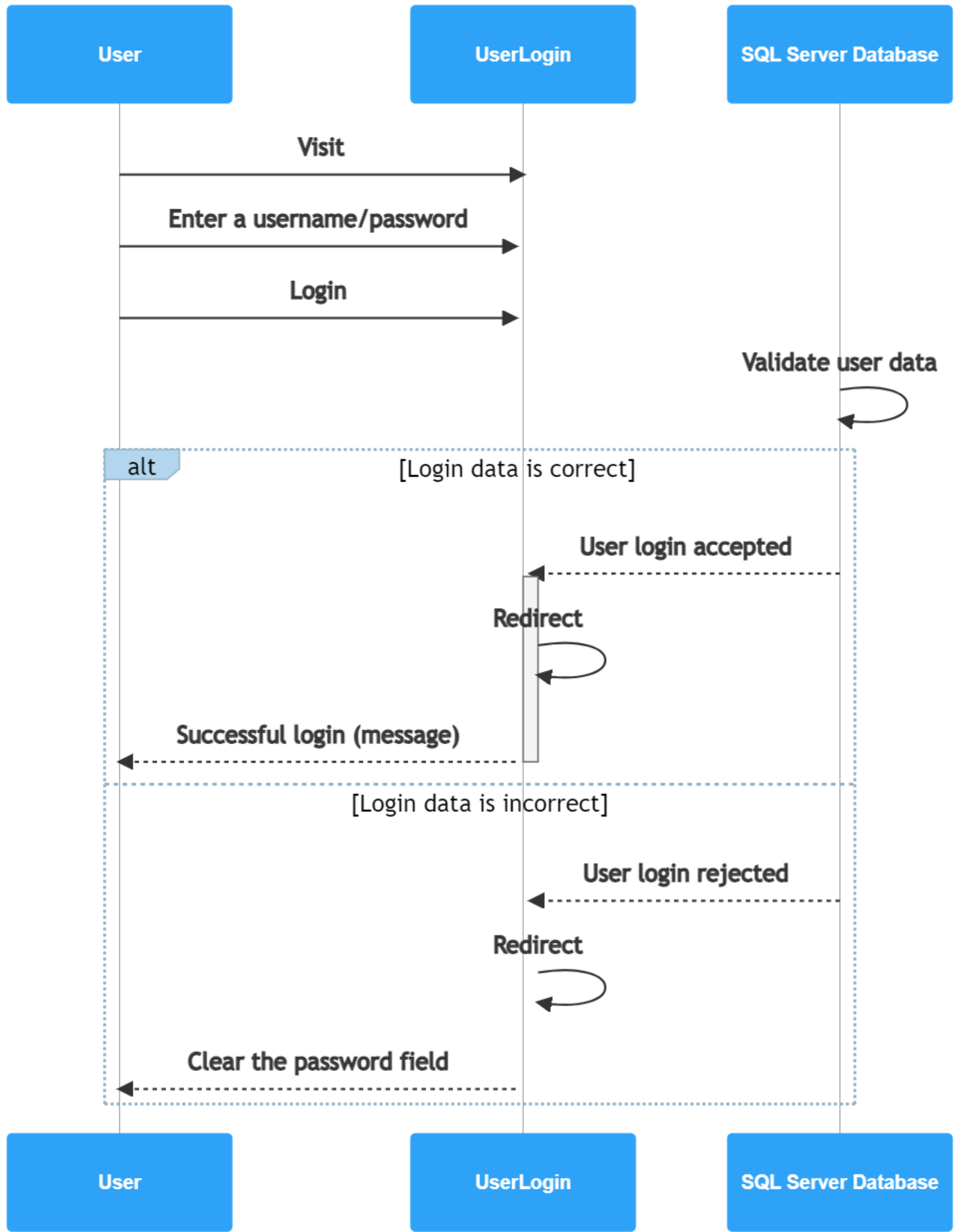
Sequence diagram: Example from Use case to Sequence Diagram 2/3

CardPayment page



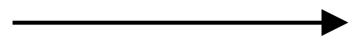
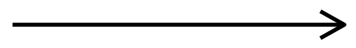
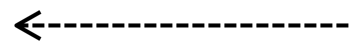
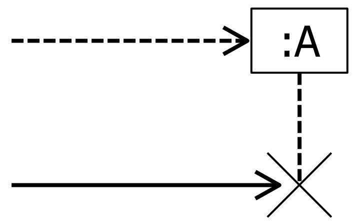
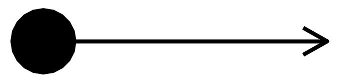
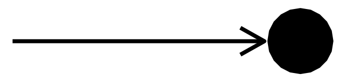
Sequence diagram: Example from Use case to Sequence Diagram 3/3

User login page



Lifeline & Messages

- A lifeline represents a single participant in an interaction
  - Shows how a classifier instance may participate in the interaction
- A message represents a communication between two lifelines

sender ➡ receiver/ target	type of message	semantics
	synchronous message	calling an operation synchronously the sender waits for the receiver to complete
	asynchronous send	calling an operation asynchronously, sending a signal the sender *does not* wait for the receiver to complete
	message return	returning from a synchronous operation call the receiver returns focus of control to the sender
	creation destruction	the sender creates the target the sender destroys the receiver
	found message	the message is sent from outside the scope of the interaction
	lost message	the message fails to reach its destination

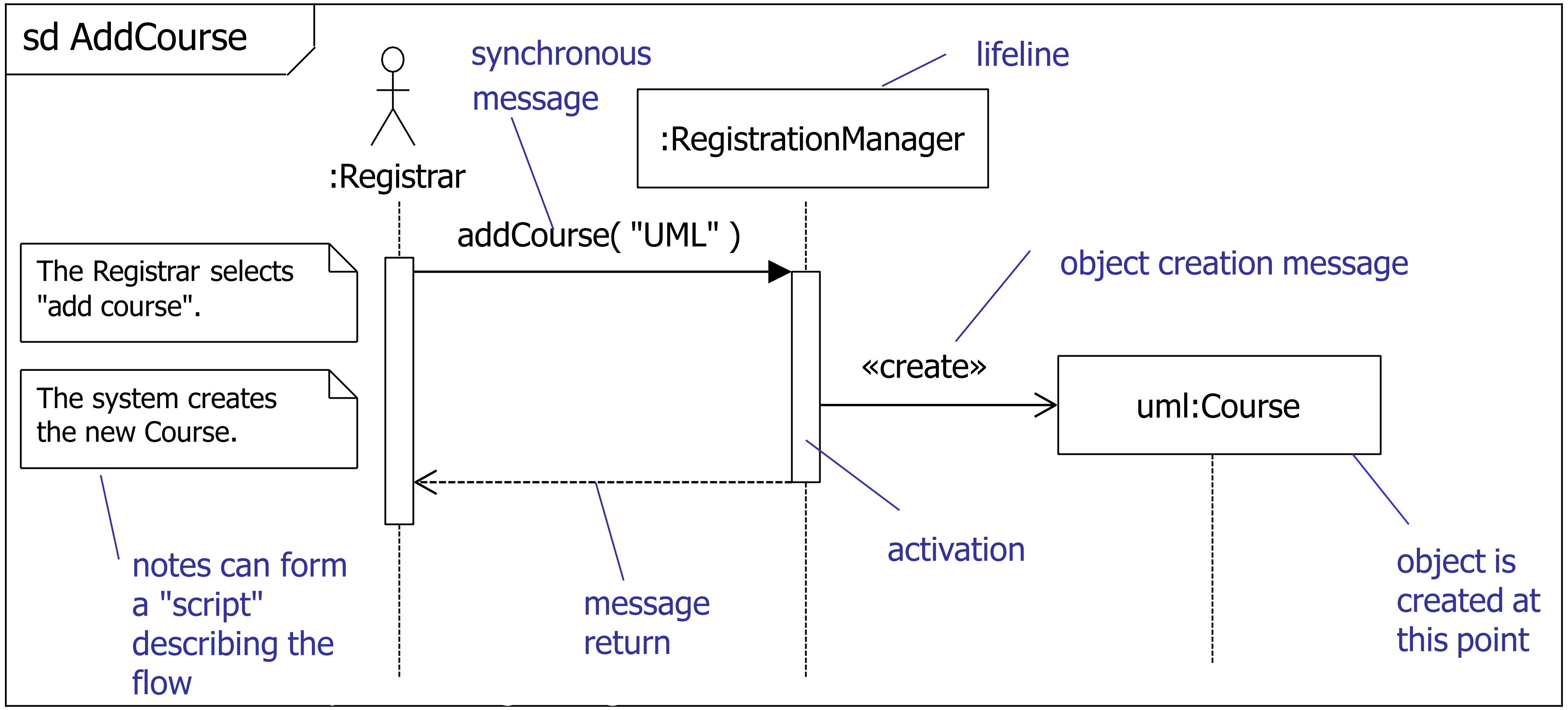
Interaction diagrams

- Sequence diagrams
  - Emphasize time-ordered sequence of message sends
  - Show interactions arranged in a time sequence
  - Are the richest and most expressive interaction diagram
  - Do not show object relationships explicitly - these can be inferred from message sends
- Communication diagrams
  - Emphasize the structural relationships between lifelines
  - Use communication diagrams to make object relationships explicit
- Interaction overview diagrams
  - Show how complex behavior is realized by a set of simpler interactions
- Timing diagrams
  - Emphasize the real-time aspects of an interaction

Sequence diagram syntax - addCourse

- *All* interaction diagrams may be prefixed sd to indicate their type
  - You can generally infer diagram types from diagram syntax

- Activations indicate when a lifeline has focus of control - they are often omitted from sequence diagrams

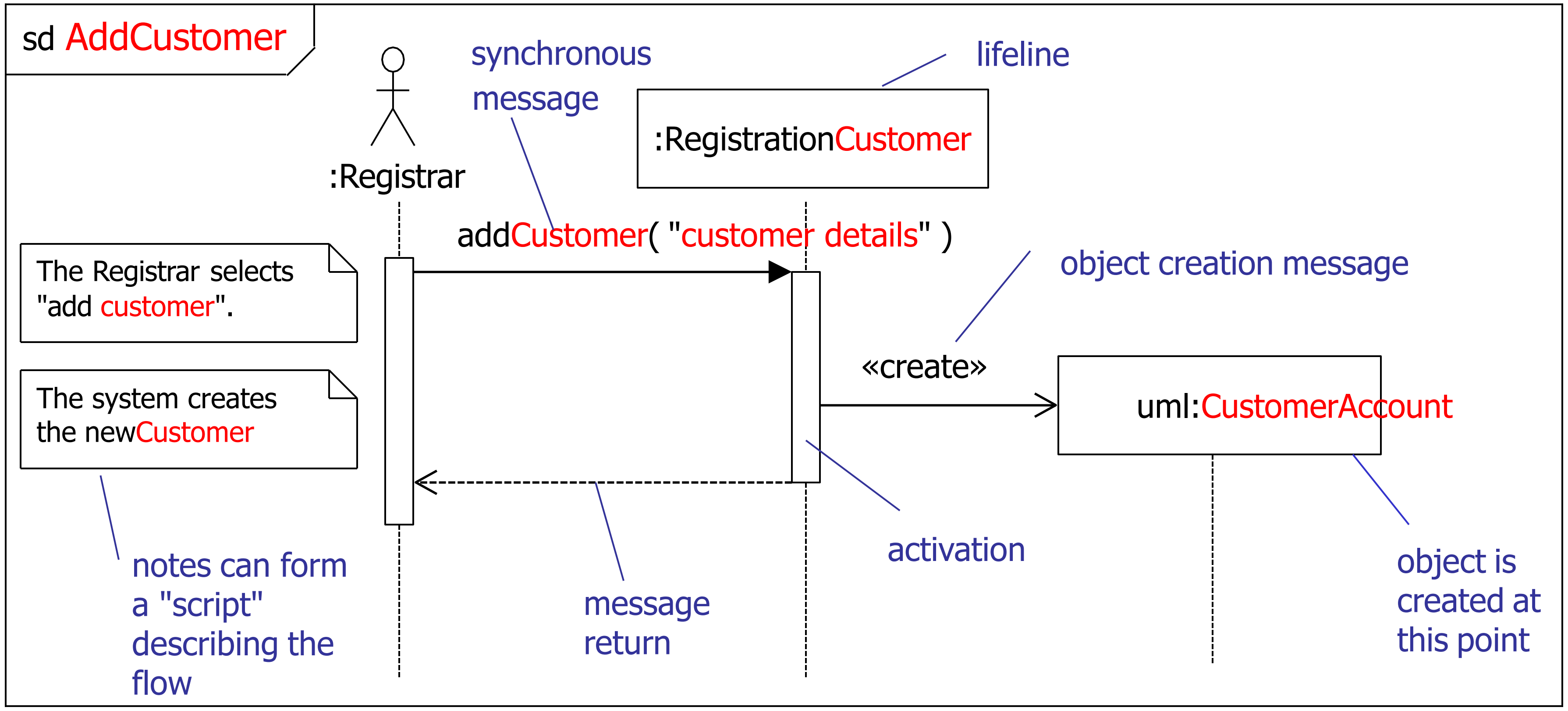


Sequence diagram syntax - AddCustomer

- *All* interaction diagrams may be prefixed sd to indicate their type
  - You can generally infer diagram types from diagram syntax
- Activations indicate when a lifeline has focus of control - they are often omitted from sequence diagrams

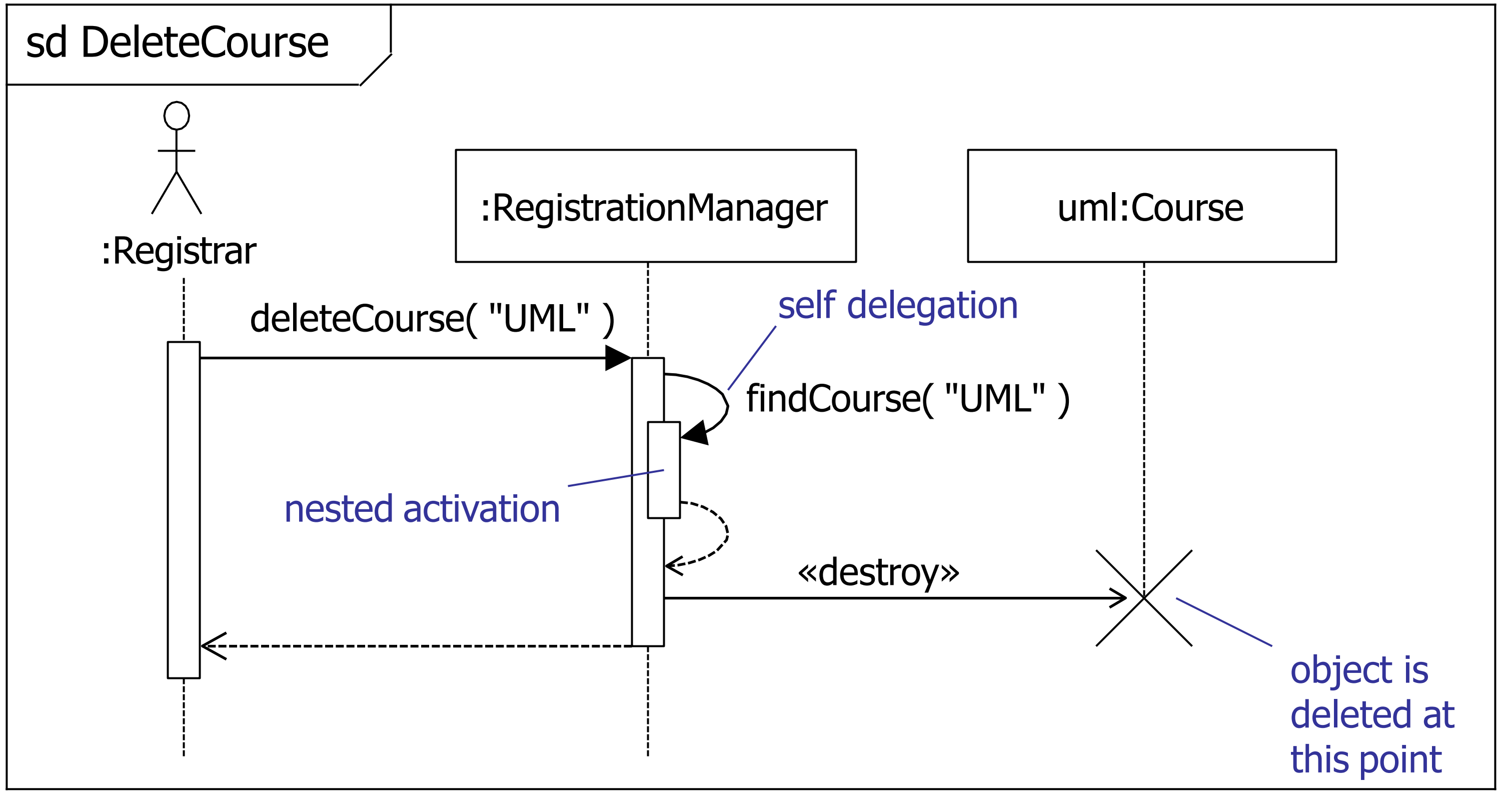
RegistrationManager becomes a class or object to supporting a new Use Case: UserRegistration!

- We **realize** more to the Systems Design.
- However we can add it as a Requirement and link to a generalised Use Case??
- Will this save us adding more Use Cases???

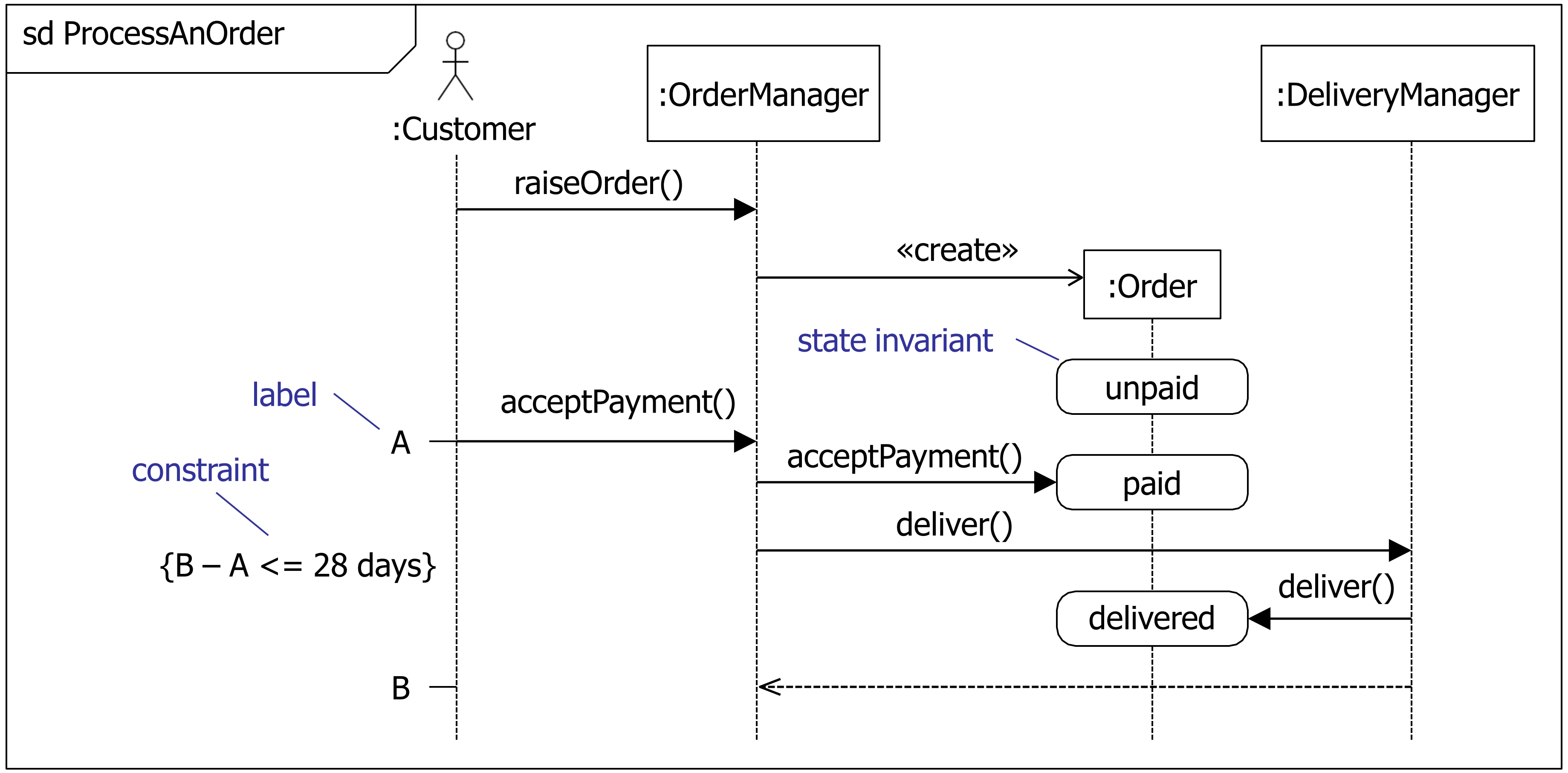

sd AddCustomer
:Registrar
synchronous message
:RegistrationCustomer
lifeline
addCustomer("customer details")
The Registrar selects "add customer".
The system creates the newCustomer
object creation message
«create»
uml:CustomerAccount
activation
notes can form a "script" describing the flow
message return
object is created at this point

Deletion and self-delegation

- Self delegation is when a lifeline sends a message to itself
  - Generates a nested activation
- Object deletion is shown by terminating the lifeline's tail at the point of deletion by a large **X**

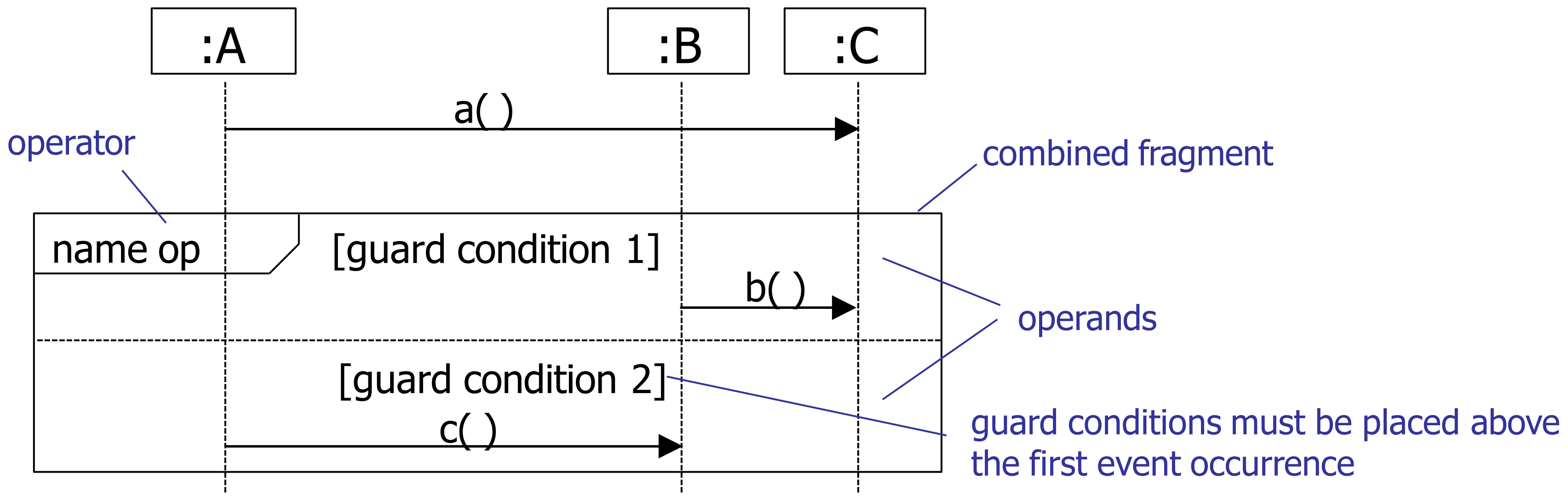


State invariants and constraints



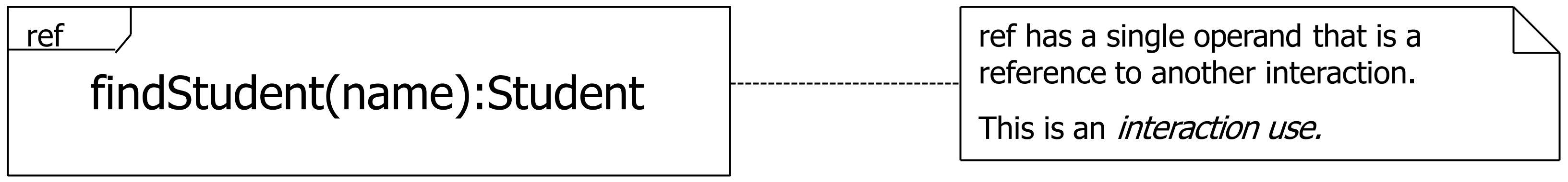
Combined fragments

- Sequence diagrams may be divided into areas called *combined fragments*
- Combined fragments have one or more *operands*
- *Operators* determine how the operands are executed
- *Guard conditions* determine whether operands execute. Execution occurs if the guard condition evaluates to true
    - A single condition may apply to all operands OR
    - Each operand may be protected by its own condition



Common operators

operator	long name	semantics
opt	Option	There is a single operand that executes if the condition is true (like if … then)
alt	Alternatives	The operand whose condition is true is executed. The keyword else may be used in place of a Boolean expression (like select… case)
loop	Loop	This has a special syntax: loop min, max [condition] Iterate min times and then up to max times while condition is true
break	Break	The combined fragment is executed rather than the rest of the enclosing interaction
ref	Reference	The combined fragment refers to another interaction



ref has a single operand that is a reference to another interaction.

This is an *interaction use.*

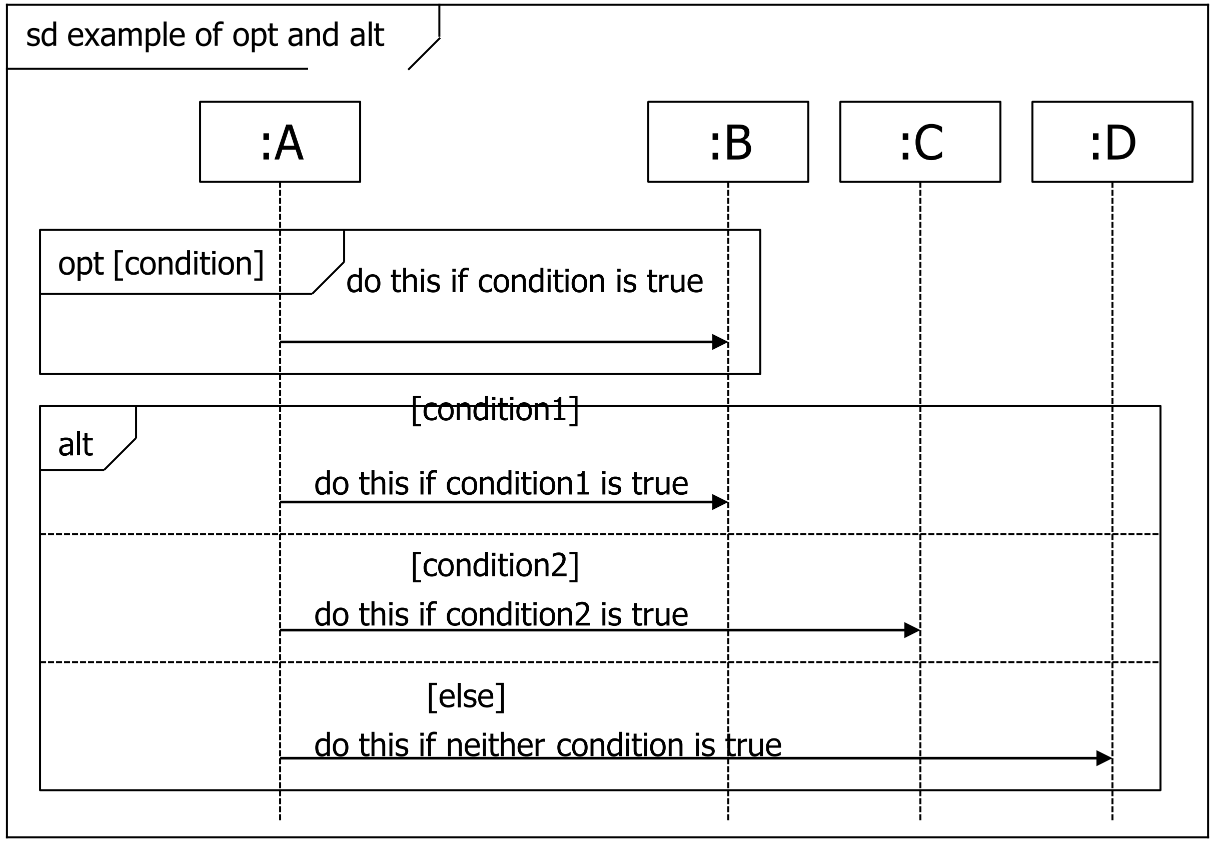
The rest of the operators

- These operators are less common

operator	long name	semantics
par	parallel	Both operands execute in parallel
seq	weak sequencing	The operands execute in parallel subject to the constraint that event occurrences on the *same* lifeline from *different* operands must happen in the same sequence as the operands
strict	strict sequencing	The operands execute in strict sequence
neg	negative	The combined fragment represents interactions that are invalid
critical	critical region	The interaction must execute atomically without interruption
ignore	ignore	Specifies that some messages are intentionally ignored in the interaction
consider	consider	Lists the messages that are considered in the interaction (all others are ignored)
assert	assertion	The operands of the combined fragments are the only valid continuations of the interaction

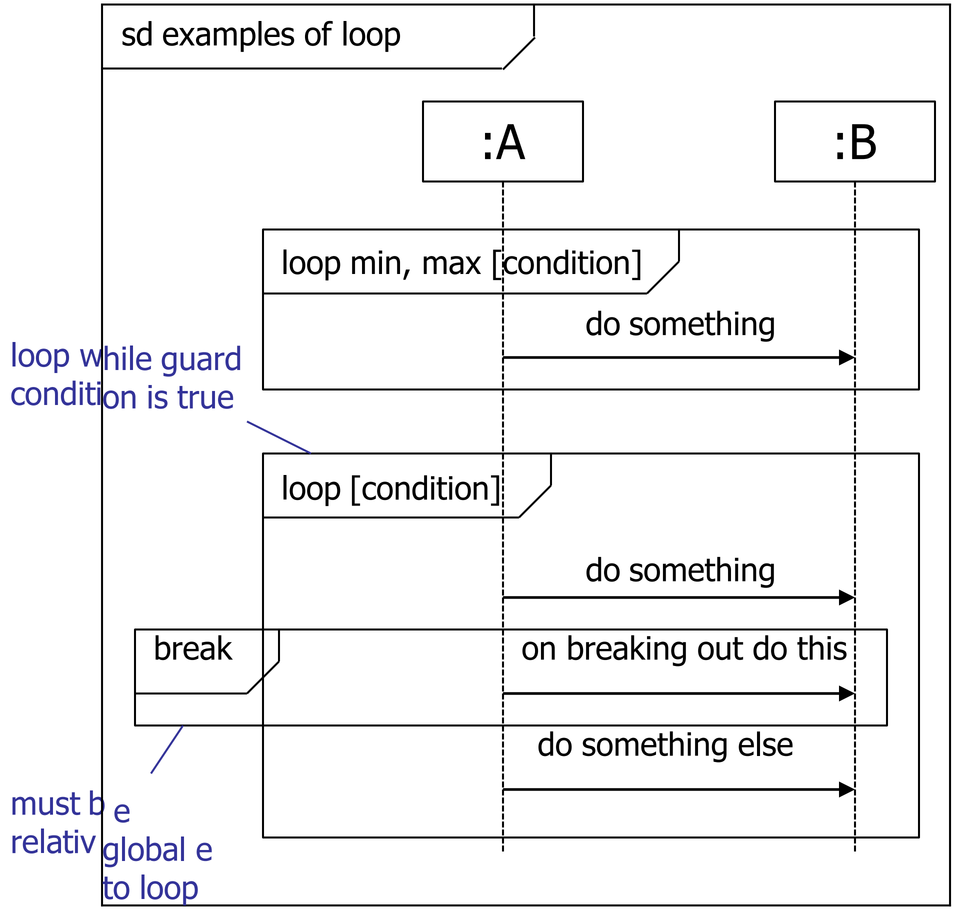
Branching with opt and alt

- **opt** semantics:
  - single operand that executes if the condition is true
- **alt** semantics:
  - two or more operands each protected by its own condition
  - an operand executes if its condition is true
  - use else to indicate the operand that executes if *none* of the conditions are true


sd example of opt and alt
:A
:B
:C
:D
opt [condition]
do this if condition is true
alt
[condition1]
do this if condition1 is true
[condition2]
do this if condition2 is true
[else]
do this if neither condition is true

Iteration with loop and break

- loop semantics:
  - Loop min times, then loop (max – min) times while condition is true
- loop syntax
  - A loop without min, max or condition is an infinite loop
  - If only min is specified then max = min
  - condition can be
    - Boolean expression
    - Plain text expression *provided* it is clear!
- Break specifies what happens when the loop is broken out of:
  - The break fragment executes
  - The rest of the loop after the break does *not* execute
- The break fragment is *outside* the loop and so should overlap it as shown



Loop idioms

type of loop	semantics	loop expression
infinite loop	keep looping forever	loop *

for i = 1 to n
{body}
	repeat (n) times	loop n

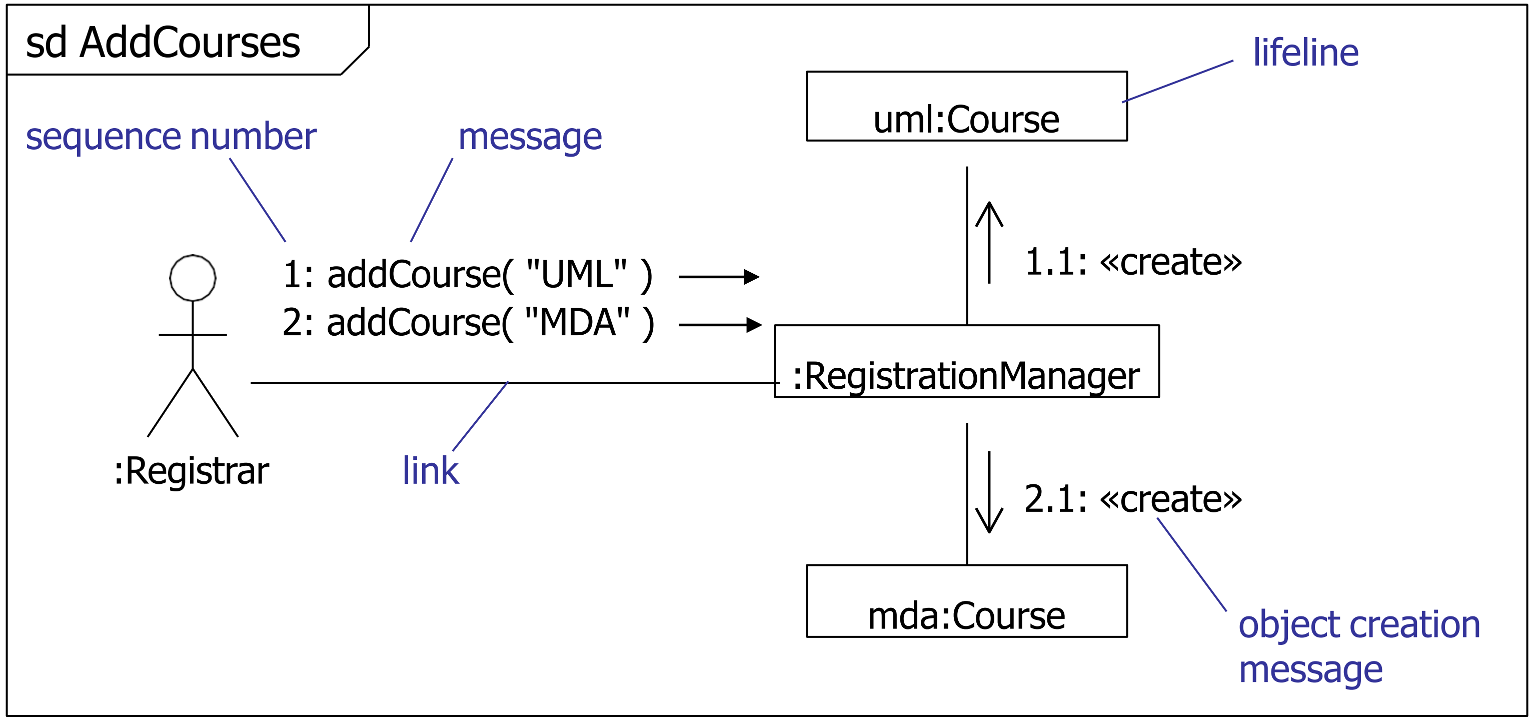
while(booleanExpression)
{body}
	repeat while booleanExpression is true	loop [booleanExpression]

repeat
{body}
while(booleanExpression)
	execute once then repeat while booleanExpression is true	loop 1, * [booleanExpression]

forEach object in collection {body}	Execute the loop once for each object in a collection	loop [for each object in collection]
forEach object in ObjectType {body}	Execute the loop once for each object of a particular type	loop [for each object in :ObjectType]

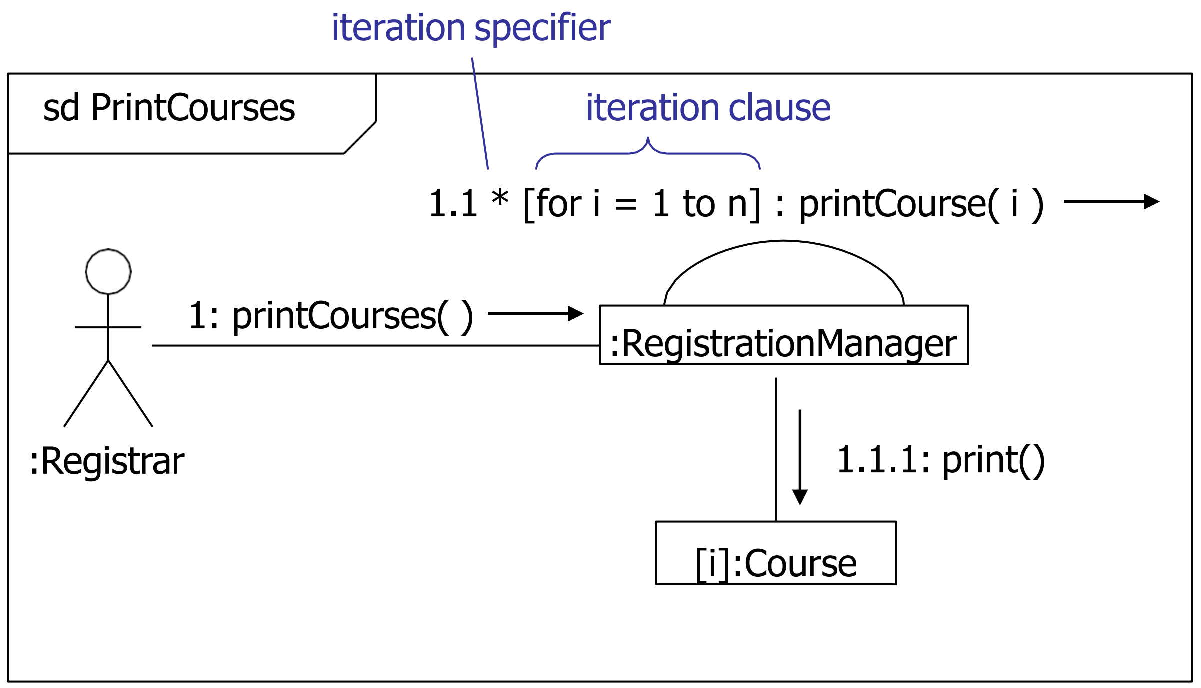
Communication diagram syntax

- Communication diagrams emphasize the structural aspects of an interaction - how lifelines connect together
  - Compared to sequence diagrams they are semantically weak
  - Object diagrams are a special case of communication diagrams



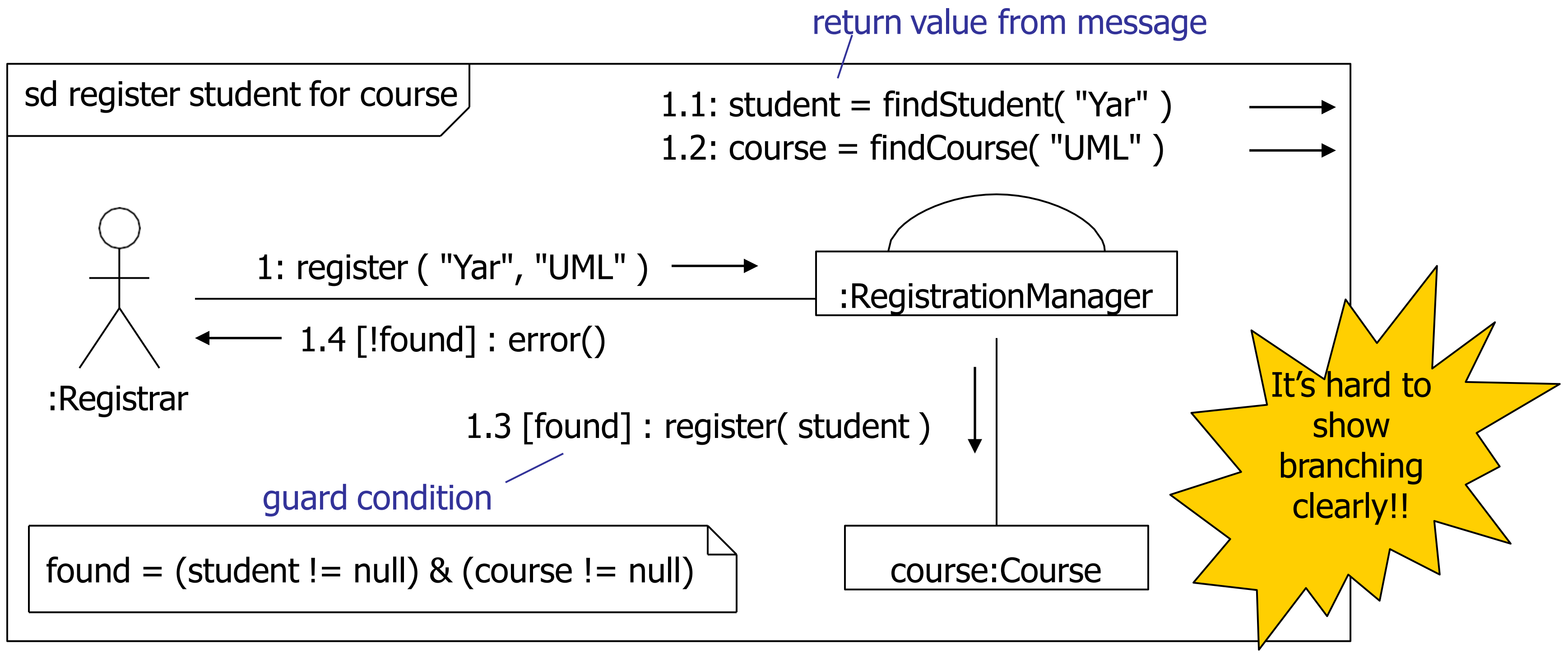
Iteration

- Iteration is shown by using the *iteration specifier* (*), and an optional *iteration clause*
  - There is no prescribed UML syntax for iteration clauses
  - Use code or pseudo code
- To show that messages are sent in parallel use the parallel iteration specifier, *//



Branching

- Branching is modelled by prefixing the sequence number with a *guard condition*
    - There is no prescribed UML syntax for guard conditions!
    - In the example below, we use the variable found. This is true if both the student and the course are found, otherwise it is false


return value from message
sd register student for course
1.1: student = findStudent("Yar")
1.2: course = findCourse("UML")
1: register ("Yar", "UML")
:RegistrationManager
1.4 [!found] : error()
:Registrar
1.3 [found] : register(student)
guard condition
found = (student != null) & (course != null)
course:Course
It's hard to show branching clearly!!

Summary

- In this section we have looked at use case realization using interaction diagrams
- There are four types of interaction diagram:
  - **Sequence diagrams – emphasize time-ordered sequence of message sends**
  - Communication diagrams – emphasize the structural relationships between lifelines
  - Interaction overview diagrams – show how complex behavior is realized by a set of simpler interactions
  - Timing diagrams – emphasize the real-time aspects of an interaction
- We have looked at sequence diagrams
- **Hopefully after Class Diagrams you have a complete eCommerce Systems Design using UML**

DEMO

How to use/find symbols for Sequence Diagram

What are Classes?

Design **classes** are **classes whose design specifications** have been completed to such a degree that they can be drafted for implemented in a Development Toolset.

- The class helps the developer to progress onto the process of actual writing the piece(s) of code

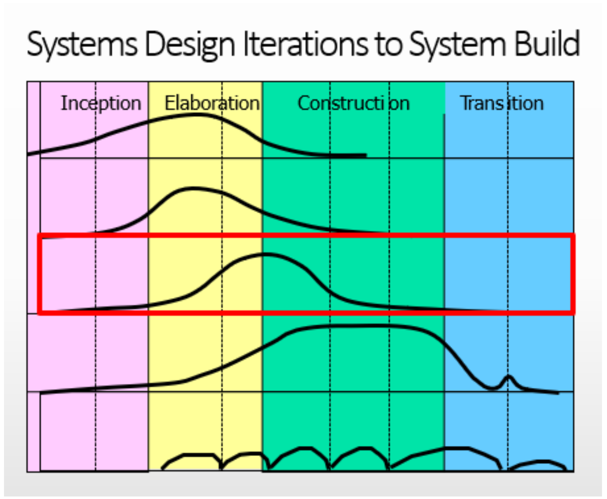
Why are the classes included in the UML?

- The **classes** that are included in the UML are what represent the **building blocks of the same objects or things**.
    - Customer things? User things?
    - Order things? Sales things?
    - Basket things?
    - Despatch things?

- This is the reason why the **class diagrams** are also referred to as the **building blocks** of UML.
- When it comes to the varying elements in **the class diagram**, there are a few we needs to be highlight and considered ….

Design – Class Diagram

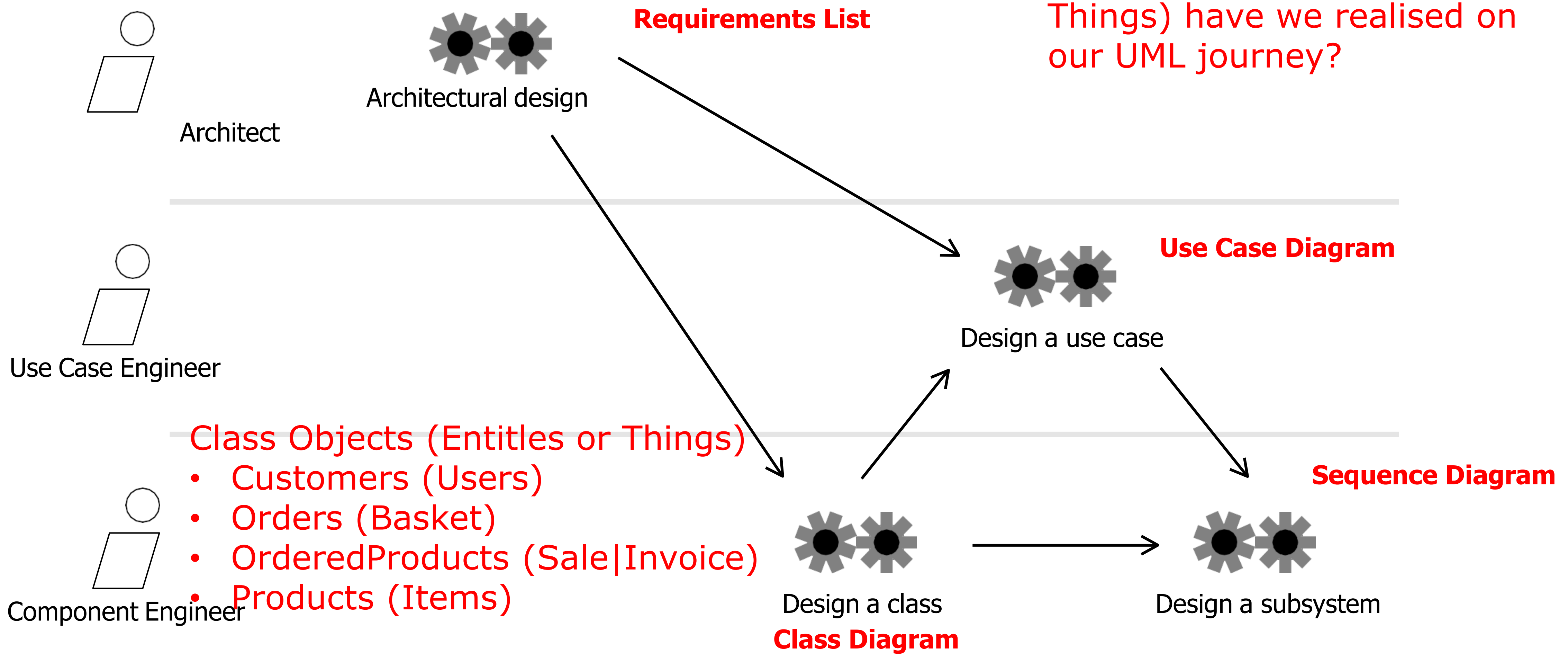
- Defn: In software engineering, a **class diagram** in the **UML** is **a type of static structure diagram** that describes the structure of a system by showing the system's **classes**, their **attributes**, **operations (or methods),** and the **relationships** among **objects**.



- SME: How best to view 'Class - Objects'
- *Things we need to model and build*
- **Class Objects, Data and Sequence Operations?**
  - Customer class – data and its sequences?
  - Order class – data and its sequences?
  - Etc
- **Class, Objects, Entities or Things we need to model & build in a SQL Server Database**
  - Store and Server Related Data via technology such as SQL Server
  - Next week – Design ERD and later Build in SQL Server Database

Workflow – Systems Design

Class Objects (Entitles or Things) have we realised on our UML journey?



Class Objects (Entitles or Things)
- Customers (Users)
- Orders (Basket)
- OrderedProducts (Sale|Invoice)
- Products (Items)

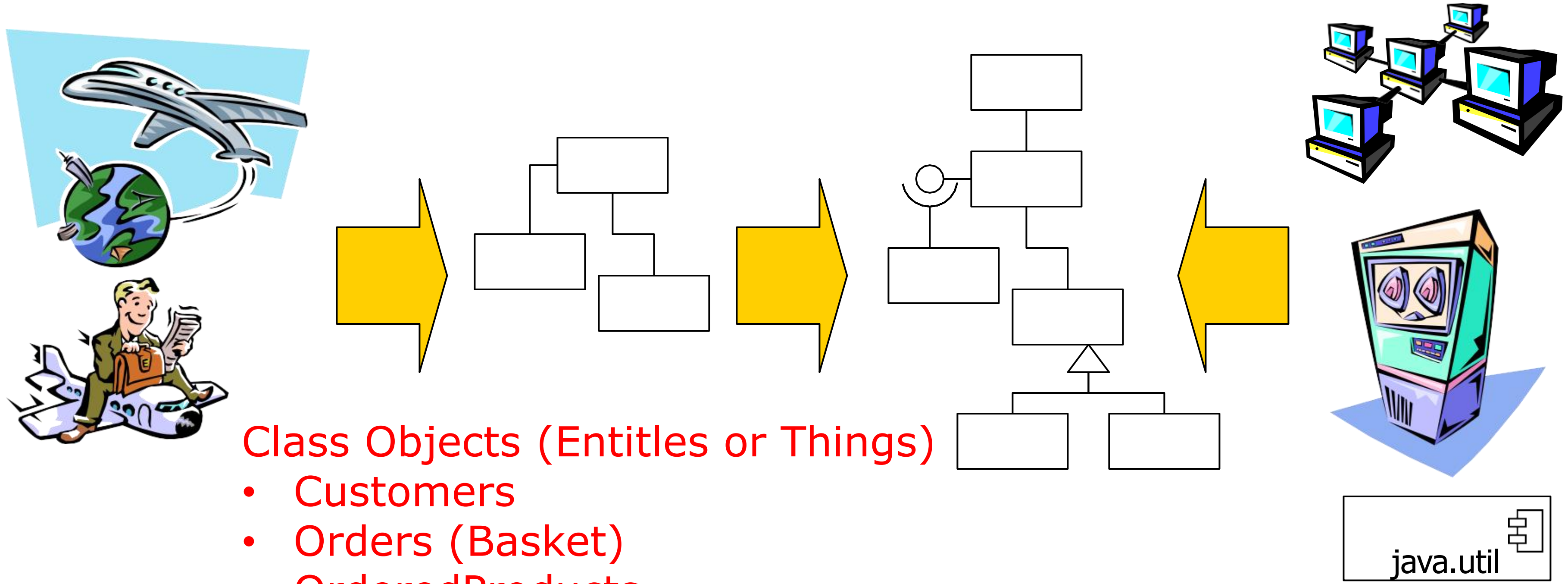
Sources of design classes

Problem domain

Analysis classes

Design classes

Solution domain



Class Objects (Entitles or Things)

- Customers
- Orders (Basket)
- OrderedProducts
- Products

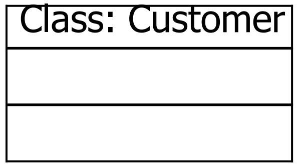
What are the class or objects or things we can identify from our Use Case

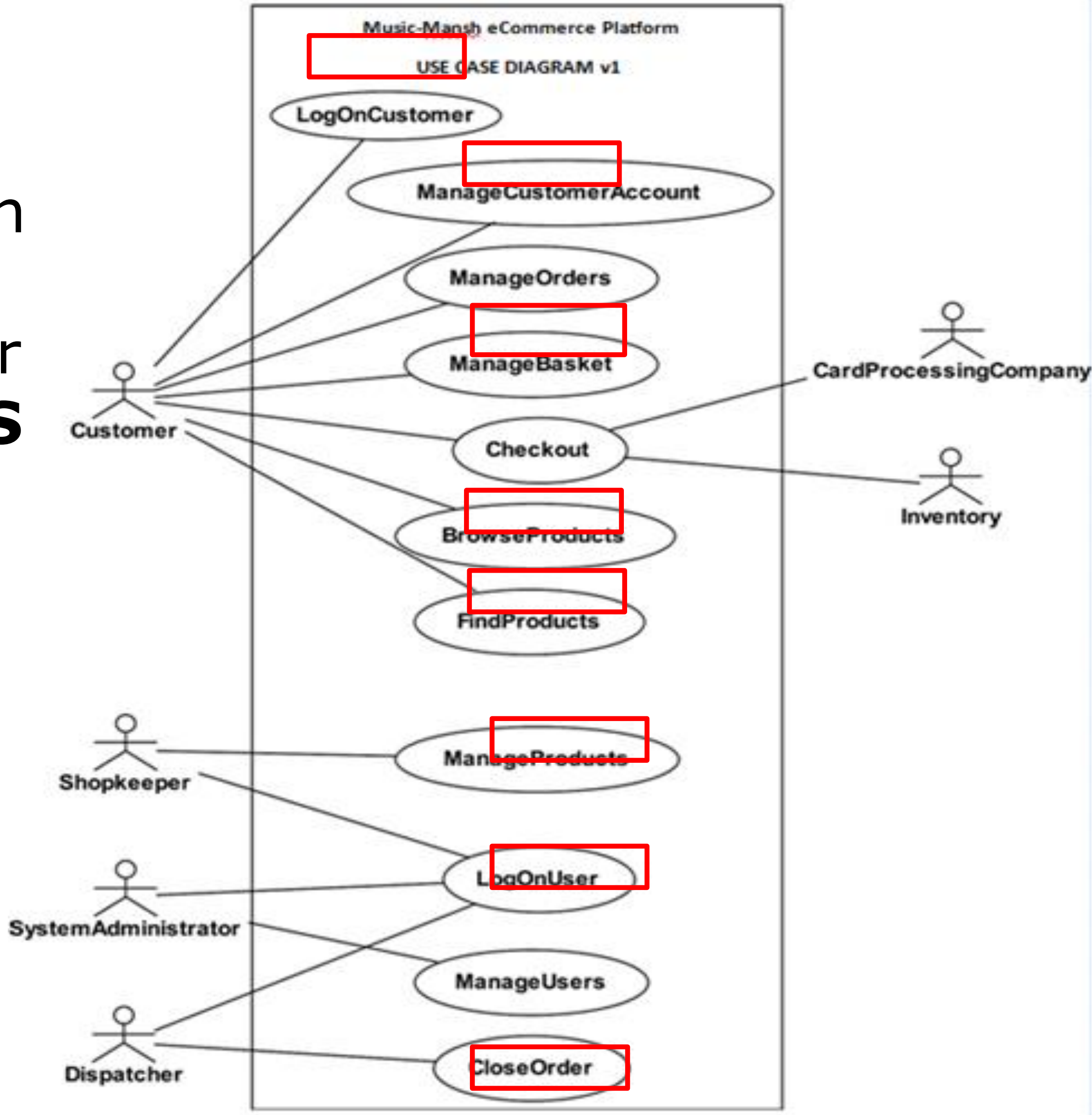
- Do we need to record a list of the above in a Database? **YES**
- Can we identify the class or object with a unique key? **YES**
- Can we identify Class, Object, Entity or Thing identified by name or noun? **YES**

If YES then list the classes of interest for Team discussion.

Class Objects (Entitles or Things)

- Customers?
- Orders?
- Basket?
- *OrderedProducts?*
- Products?
- Users?



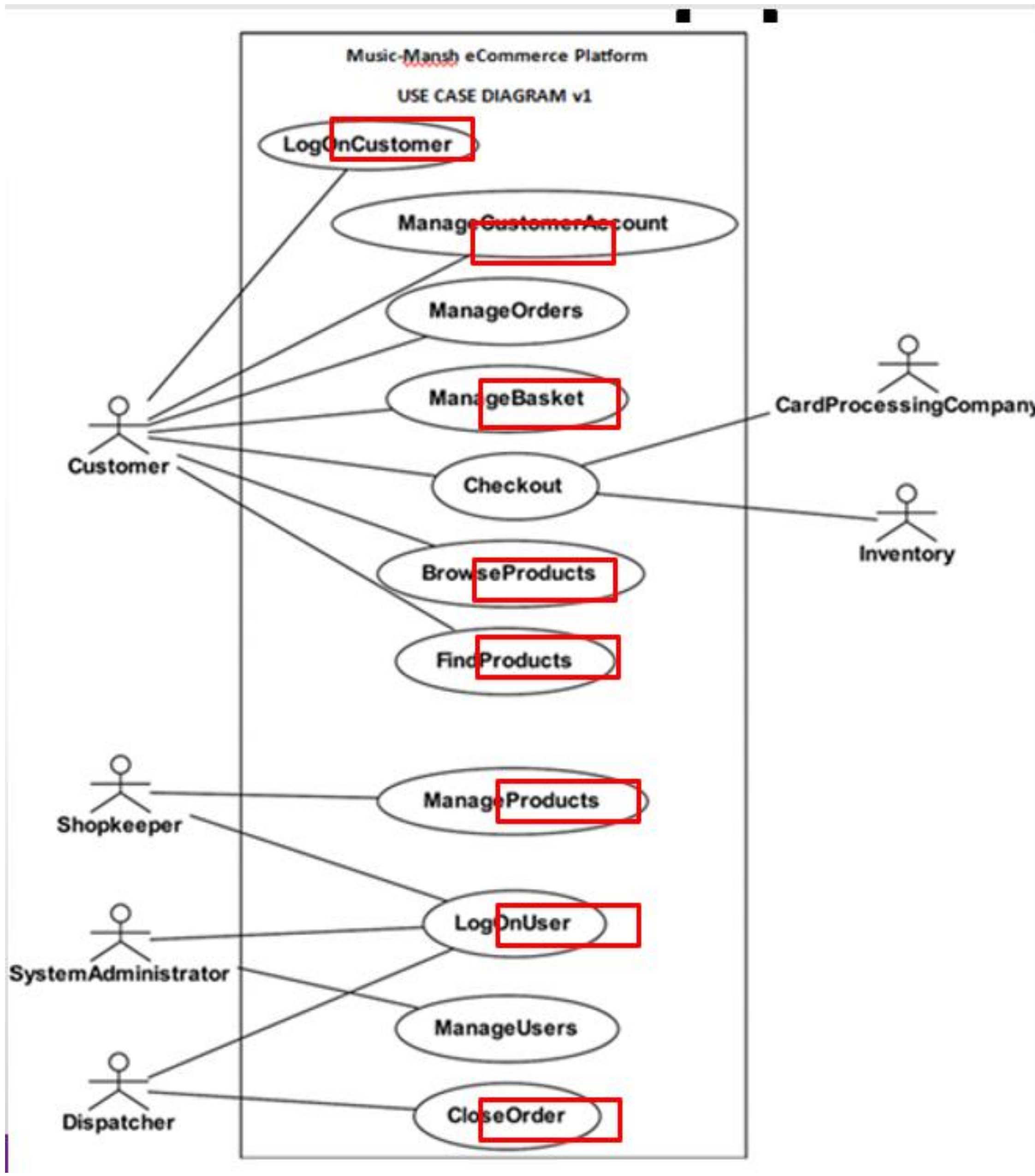


Identifying from Use Case and Sequence Diagrams our Class Object Entity or Things

Class Objects (Entitles or Things)

- Customers
- Orders
- Basket
- Products
- Users

- We may choose a main sequence to follow from the **class** or **object** prospective.
- Focus on communication paths such as:
- **Customer** adding items to a **basket** and placing an **order.**
- **Customer** making a payment (needs to interact with Bank system).
- **Customer** has an **Account**?
- Users login
- We can map many **Class**es but focus on the common ones first.



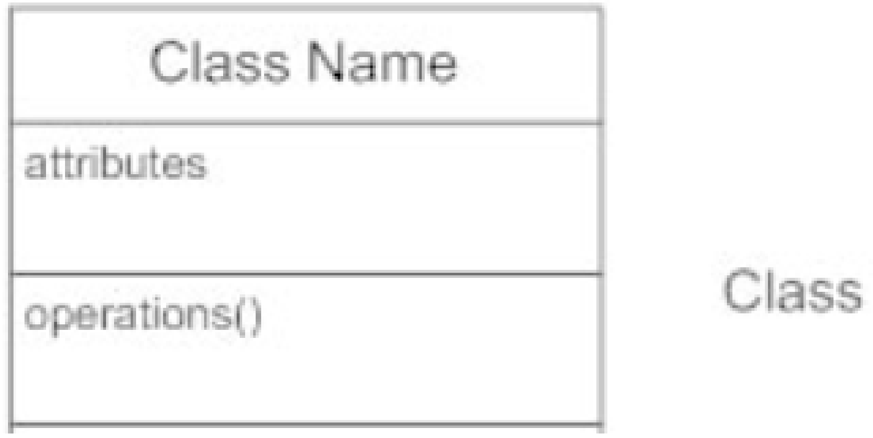
Basic Class Diagram Symbols and Notations

Classes

- Classes represent an abstraction of entities (things or objects) with common characteristics.
- Associations represent the relationships between classes.
- Illustrate classes with rectangles divided into compartments.
- Place the **name of the class** in the **first partition** (centered, bolded, and capitalized),
- List the **attributes** in the **second partition** (left-aligned, not bolded, and lowercase)
- and write **operations** **(sequences)** into the third.

Class Objects (Entitles or Things)

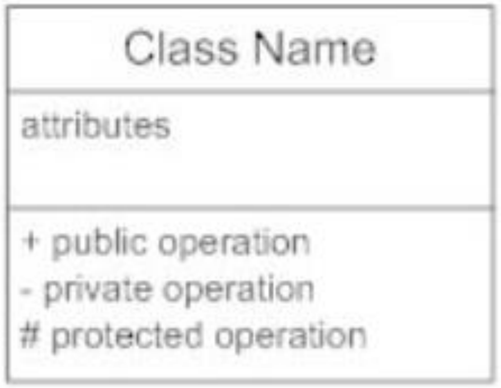
- Customers
- Orders
- Basket
- Products
- Users



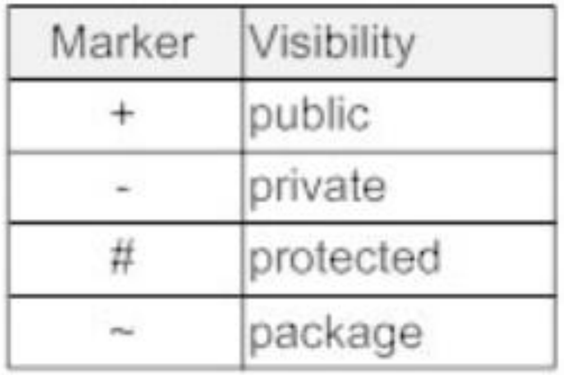
Basic Class Diagram Symbols and Notations

Visibility (to other classes)

- **Useful for the Programmers** – these are at a lower level design and useful for Development in object oriented programming languages – python, java etc
- Note: The Database Developer is primarily only interested in the attributes.
- Use visibility markers to signify who can access the information contained within a class.
- Private visibility, denoted with a **-** sign, hides information from anything outside the class partition.
- Public visibility, denoted with a **+** sign, allows all other classes to view the marked information.
- Protected visibility, denoted with a **#** sign, allows child classes to access information they inherited from a parent class.

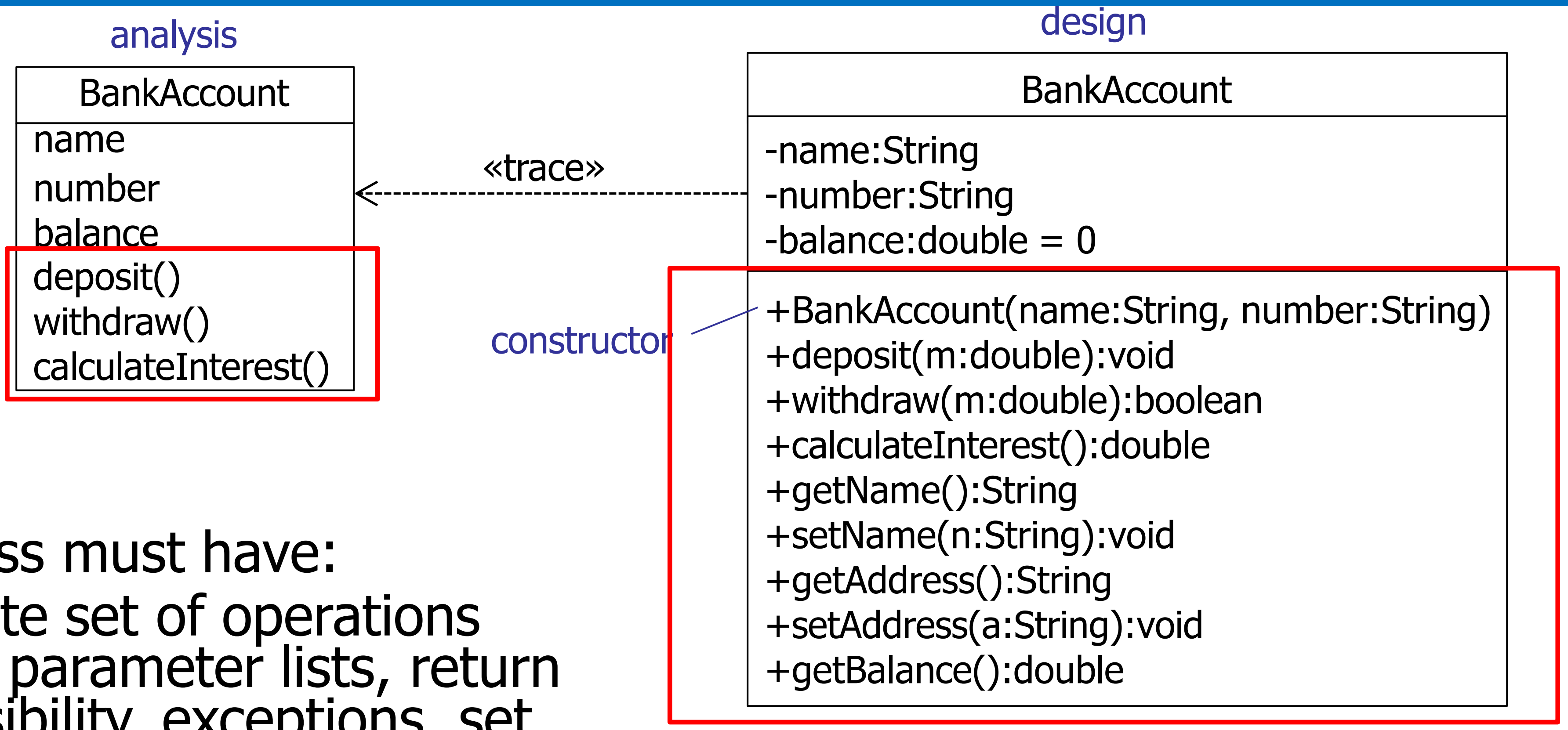


Visibility



Marker	Visibility
+	public
-	private
#	protected
~	package

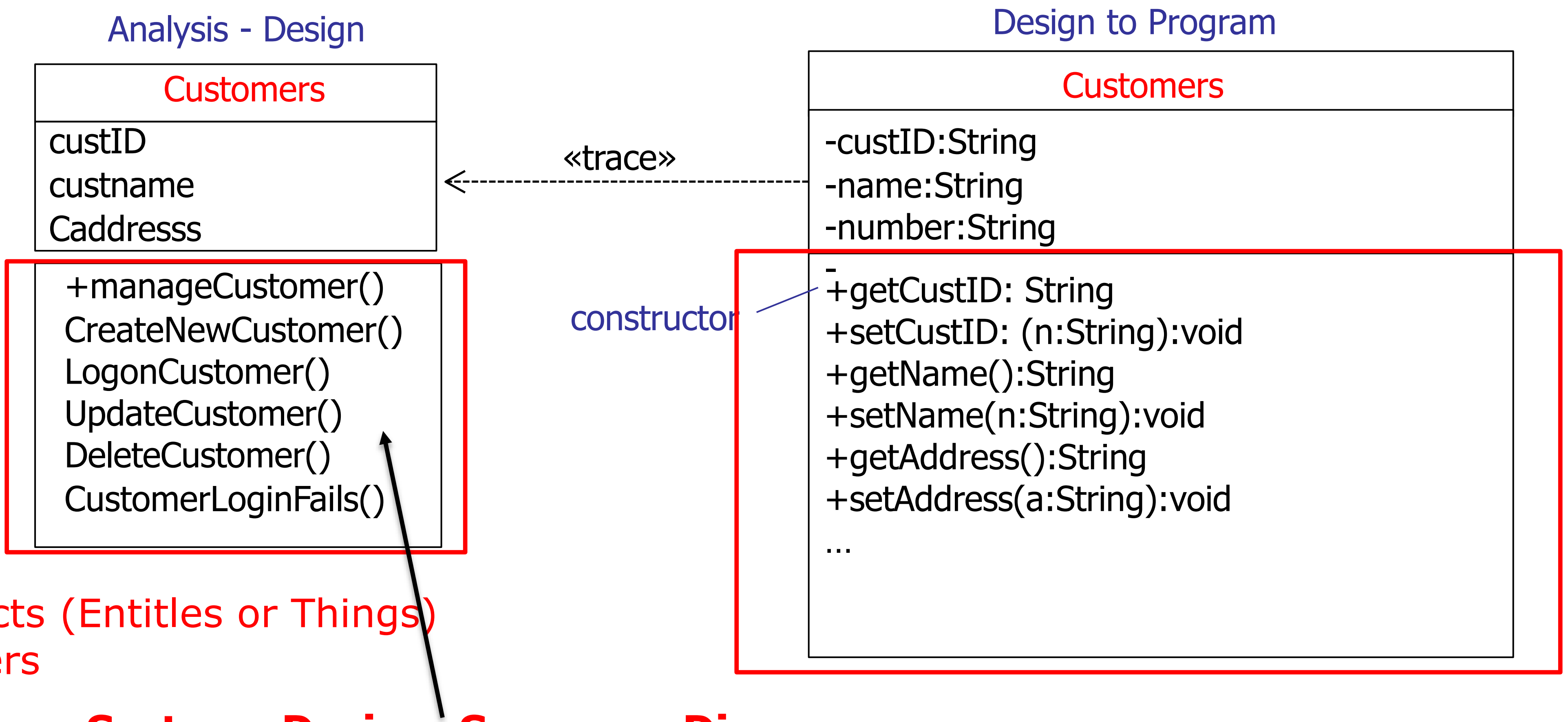
Anatomy of a design class



- A design class must have:
  - A complete set of operations including parameter lists, return types, visibility, exceptions, set and get operations, constructors and destructors
- A complete set of attributes including types and default values

- Class Diagrams we indicate the method, object, or class – the program constructs we need – **Sequence Diagrams help provide the constructs we need!**

Identifying from Use Case and Sequence Diagrams our Class Object Entity or Things



Class Objects (Entitles or Things)

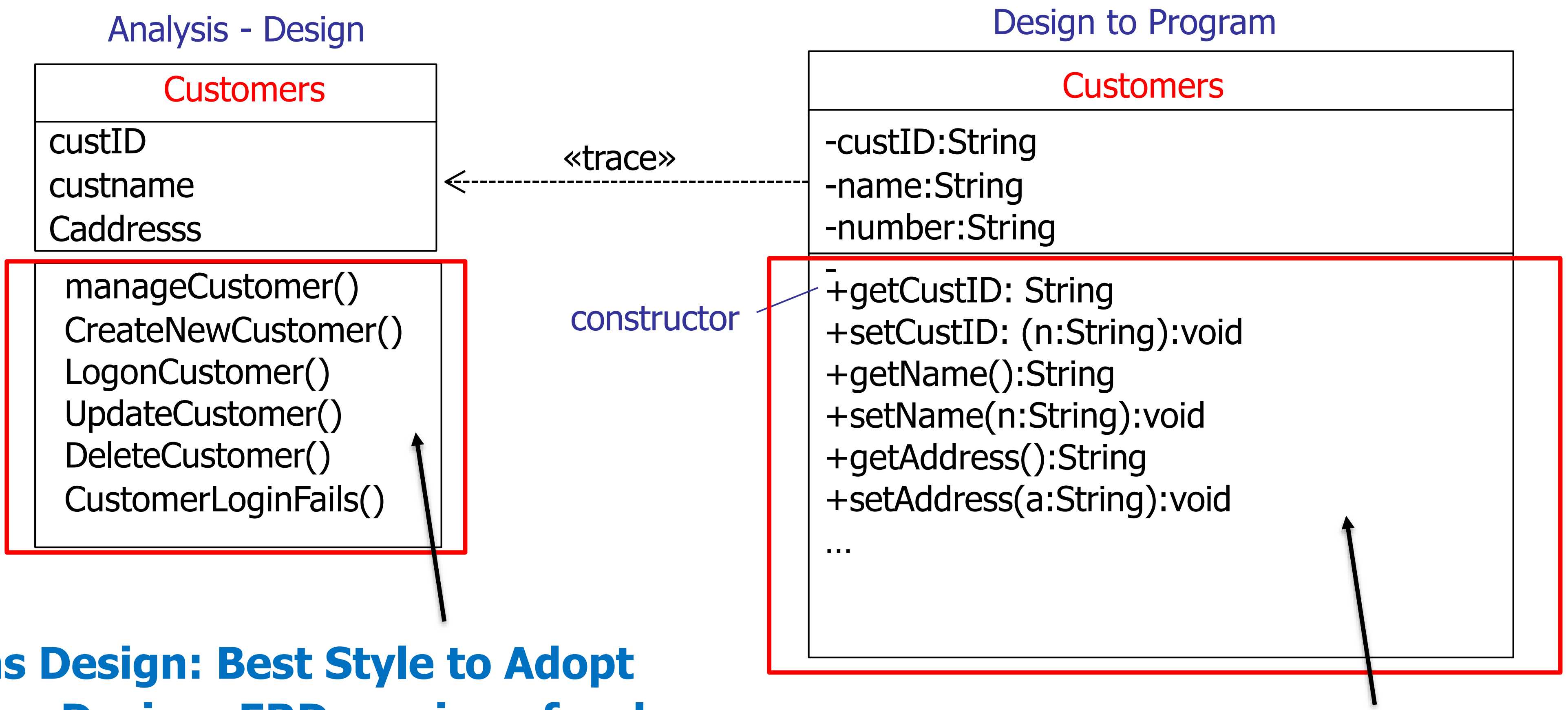
- Customers
- Orders
- Basket
- Products
- Users

- **Systems Design: Sequence Diagrams help provide the constructs we need for Class Diagrams!**

- **Class Diagrams – 2 styles to adopt! Either SD activity or programming construct activity**

Identifying from Use Case and Sequence Diagrams our Class Object Entity or Things

Class Customers



- **Systems Design: Best Style to Adopt**
- **Database Design: ERD version of a class diagram**

- **Useful style to adopt for programming coding methods in python jave etc.**

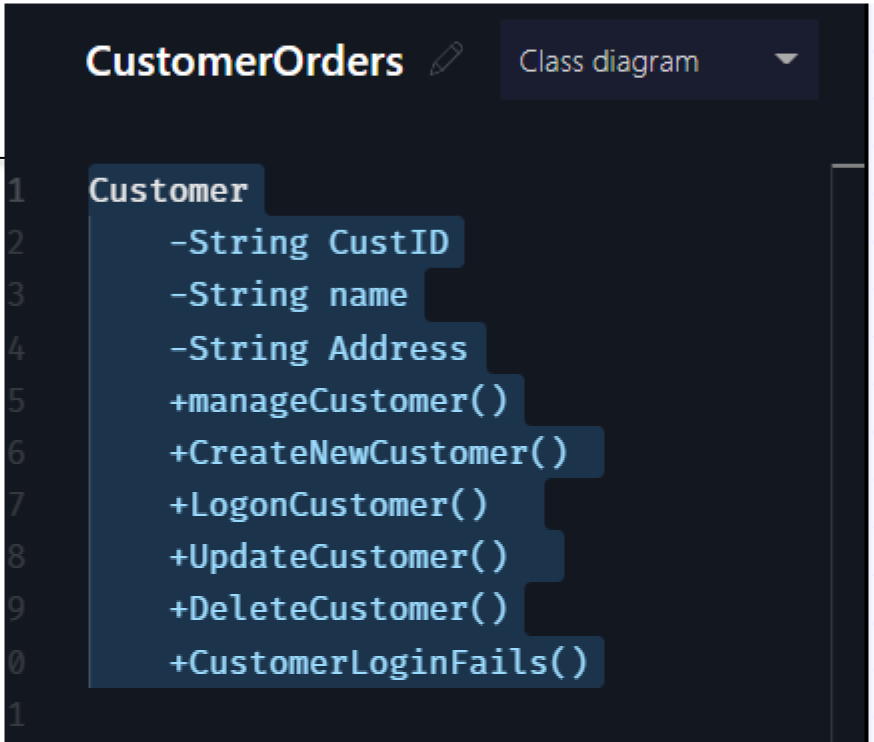
How to Developing Class Diagrams

Customer

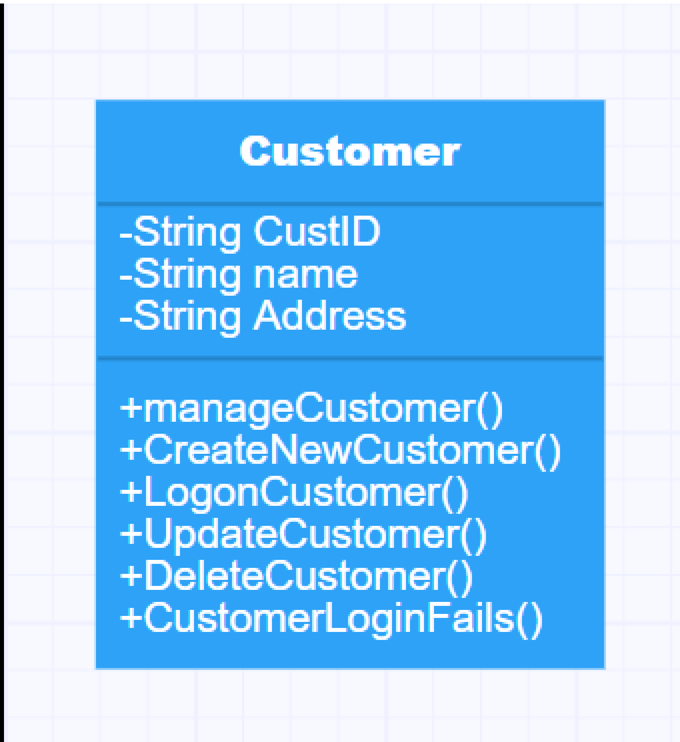
- Class Objects (Entitles or Things)
- Customers
- Orders (Basket)
- OrderedProducts
- Products

DEMO

```
Customer
    -String CustID
    -String name
    -String Address
    +manageCustomer()
    +CreateNewCustomer()
    +LogonCustomer()
    +UpdateCustomer()
    +DeleteCustomer()
    +CustomerLoginFails()
```



```
Customer
    -String CustID
    -String name
    -String Address
    +manageCustomer()
    +CreateNewCustomer()
    +LogonCustomer()
    +UpdateCustomer()
    +DeleteCustomer()
    +CustomerLoginFails()
```

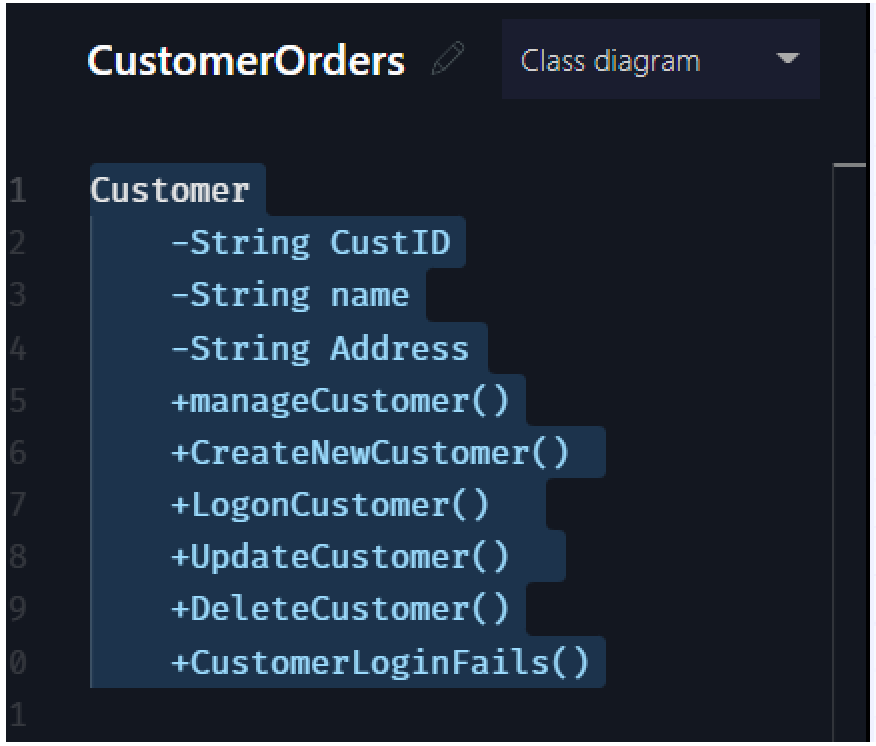


```
Customer
-String CustID
-String name
-String Address
+manageCustomer()
+CreateNewCustomer()
+LogonCustomer()
+UpdateCustomer()
+DeleteCustomer()
+CustomerLoginFails()
```

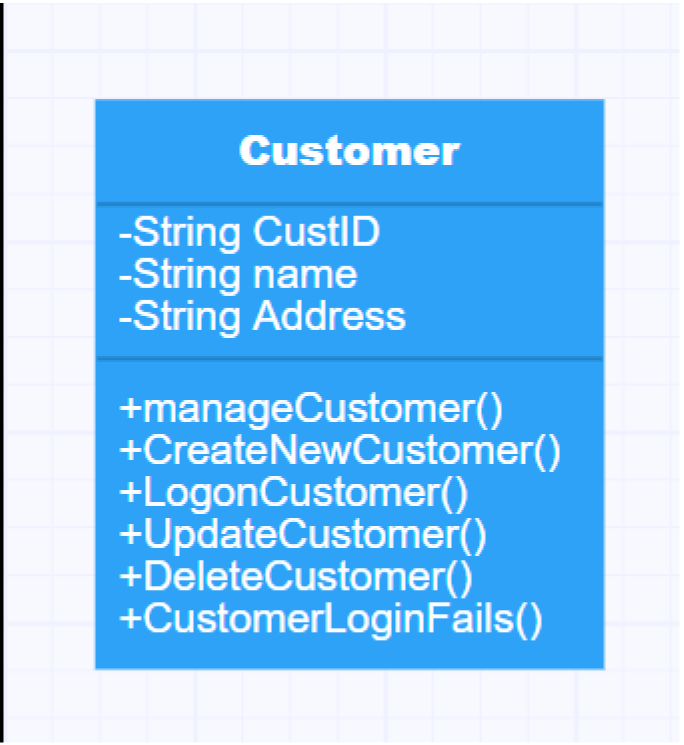
Customer

- Class Objects (Entitles or Things)
- **Customers**
- Orders (Basket)
- OrderedProducts
- Products

```
Customer
    -String CustID
    -String name
    -String Address
    +manageCustomer()
    +CreateNewCustomer()
    +LogonCustomer()
    +UpdateCustomer()
    +DeleteCustomer()
    +CustomerLoginFails()
```



```
Customer
    -String CustID
    -String name
    -String Address
    +manageCustomer()
    +CreateNewCustomer()
    +LogonCustomer()
    +UpdateCustomer()
    +DeleteCustomer()
    +CustomerLoginFails()
```

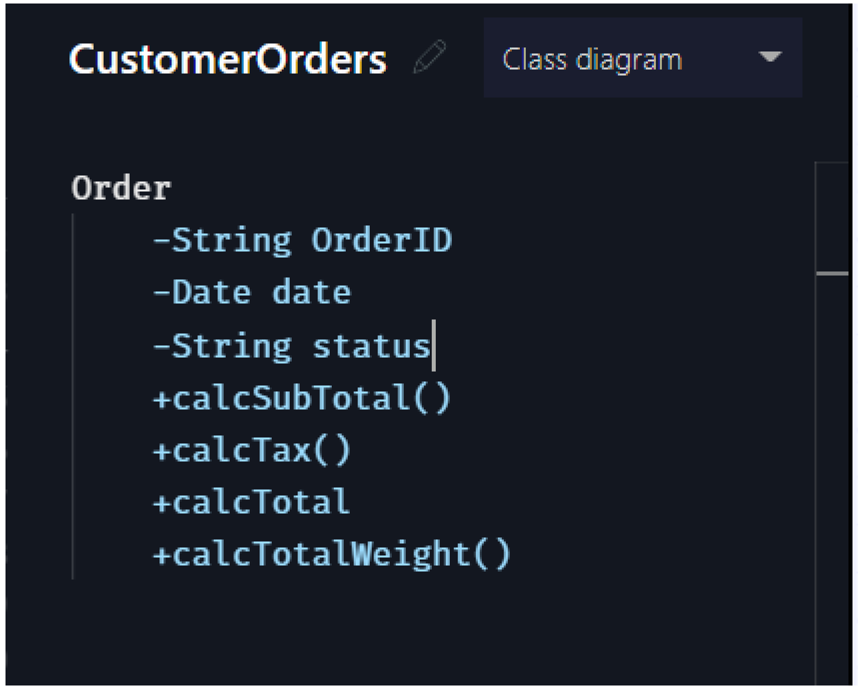


Orders (Basket)

Class Objects (Entitles or Things)

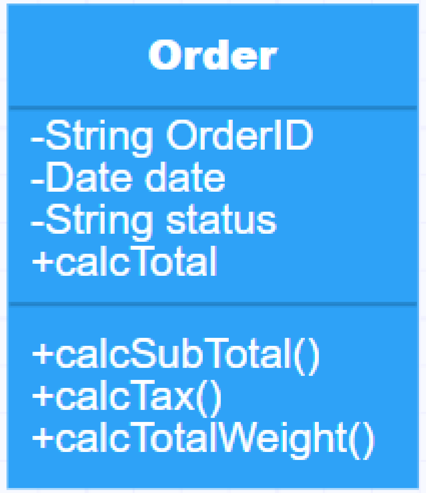
- Customers
- **Orders** (Basket)
- OrderedProducts
- Products

```
Order
        -String OrderID
        -Date date
        -String status
        +calcSubTotal()
        +calcTax()
        +calcTotal
        +calcTotalWeight()
```



CustomerOrders

```
Order
     -String OrderID
     -Date date
     -String status
     +calcSubTotal()
     +calcTax()
     +calcTotal
     +calcTotalWeight()
```



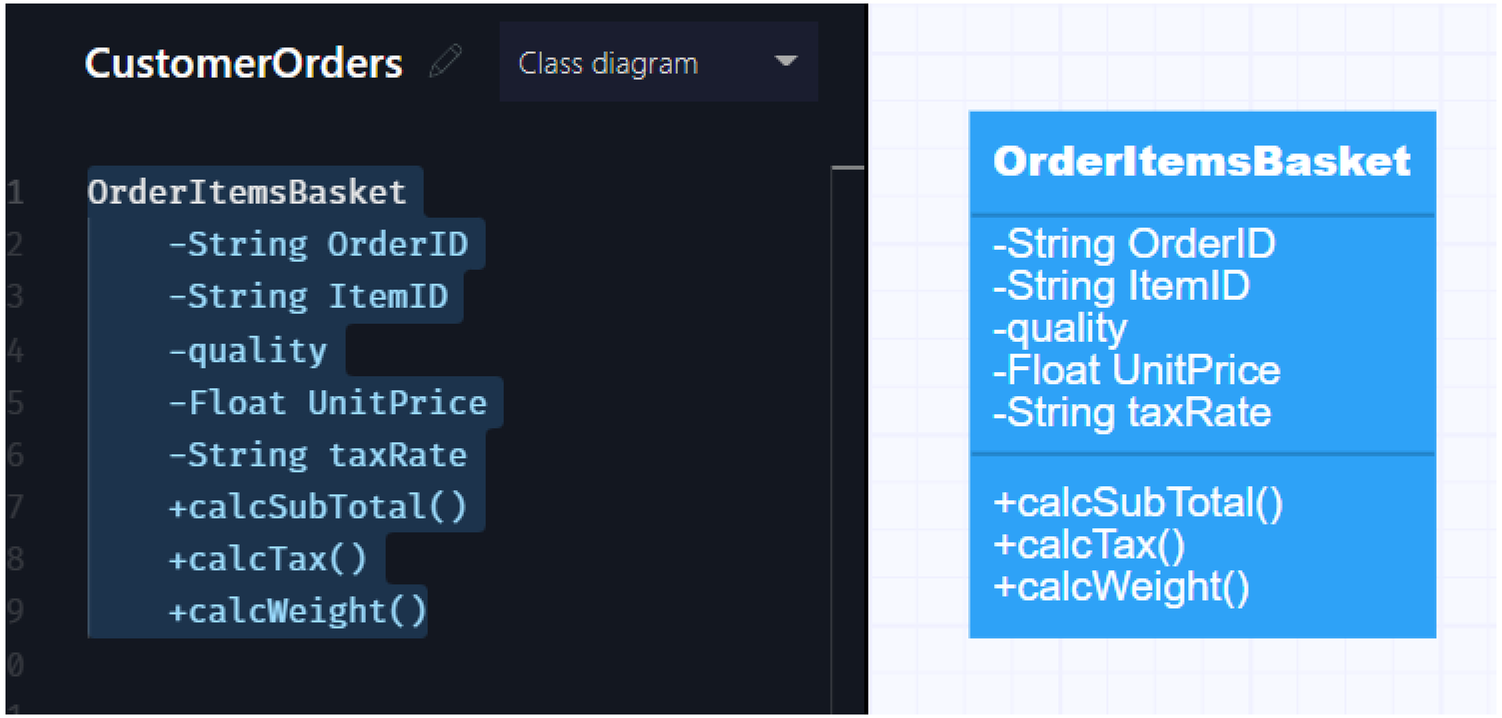
Order
-String OrderID -Date date -String status +calcTotal

+calcSubTotal()
+calcTax()
+calcTotalWeight()

Ordered Products

- **Class Objects (Entitles or Things)**
- Customers
- Orders **(**Basket)
- **OrderedProducts**
- Products

```
OrderItemsBasket
       -String OrderID
       -String ItemID
       -quality
       -Float UnitPrice
       -String taxRate
       +calcSubTotal()
       +calcTax()
       +calcWeight()
```

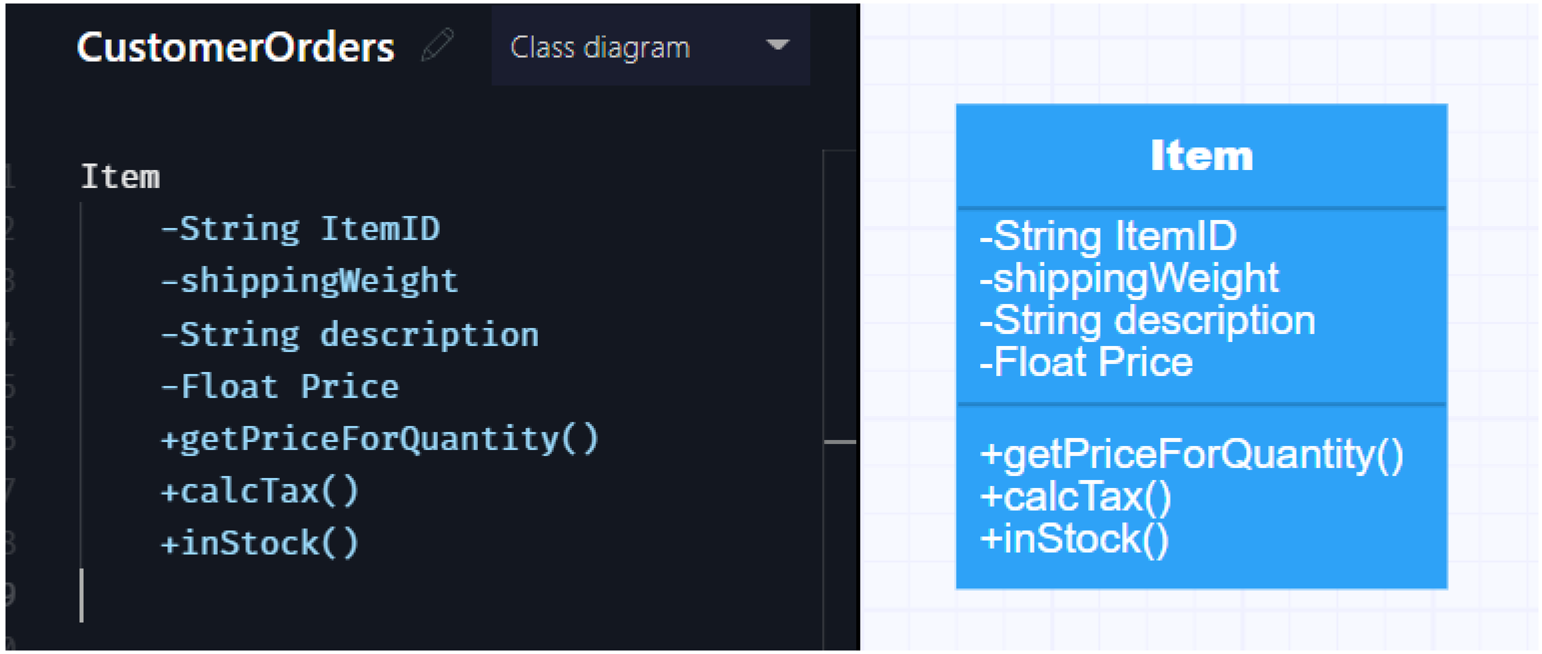


```
OrderItemsBasket
    -String OrderID
    -String ItemID
    -quality
    -Float UnitPrice
    -String taxRate
    +calcSubTotal()
    +calcTax()
    +calcWeight()
```

Items

- Class Objects (Entitles or Things)
- Customers
- Orders **(**Basket)
- OrderedProducts
- Products
- **Items**

```
Item
         -String ItemID
         -shippingWeight
         -String description
         -Float Price
         +getPriceForQuantity()
         +calcTax()
         +inStock()
```



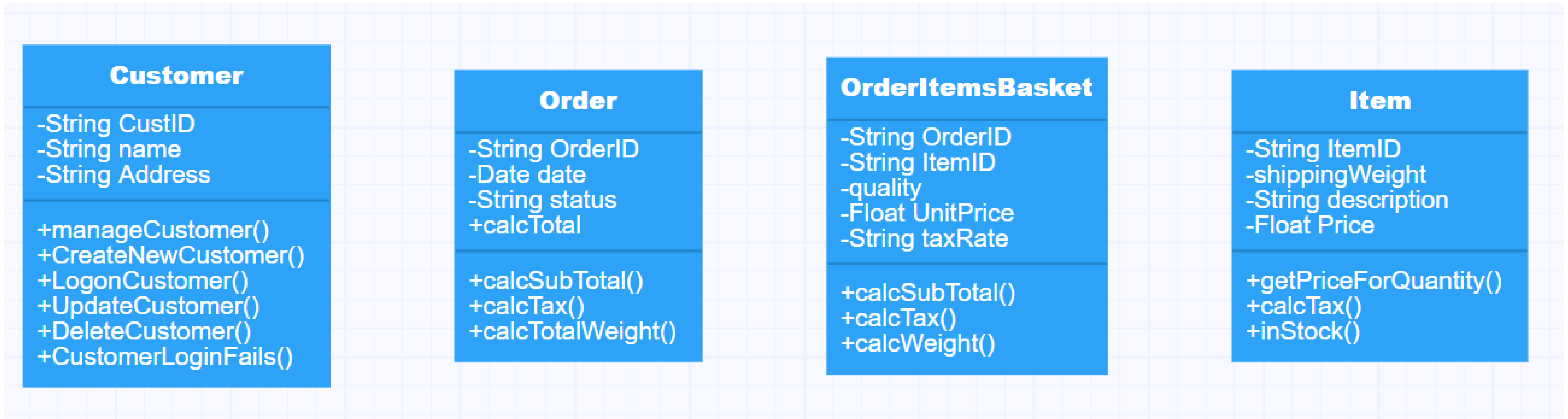
```
Item
    -String ItemID
    -shippingWeight
    -String description
    -Float Price
    +getPriceForQuantity()
    +calcTax()
    +inStock()
```

Item
-String ItemID -shippingWeight -String description -Float Price

+getPriceForQuantity()
+calcTax()
+inStock()

The class diagram – pre associations or relationships

- We need to add associations or relationships to complete the class model
- Working method for associations and relationship degree
- We will be covering relationships in more details when we will focus on SQL Server Database build for the Class Diagram (Entity Relationship Diagram)



The class diagram associations or relationships

- one **customer** will have none, one or more **orders.**
- one **order** must be associated to only one **customer**.
- An order can have many **items**
- An item can be on many orders
- Many to many are mapped into a new class to map the Primary Keys **OrderedItem**.

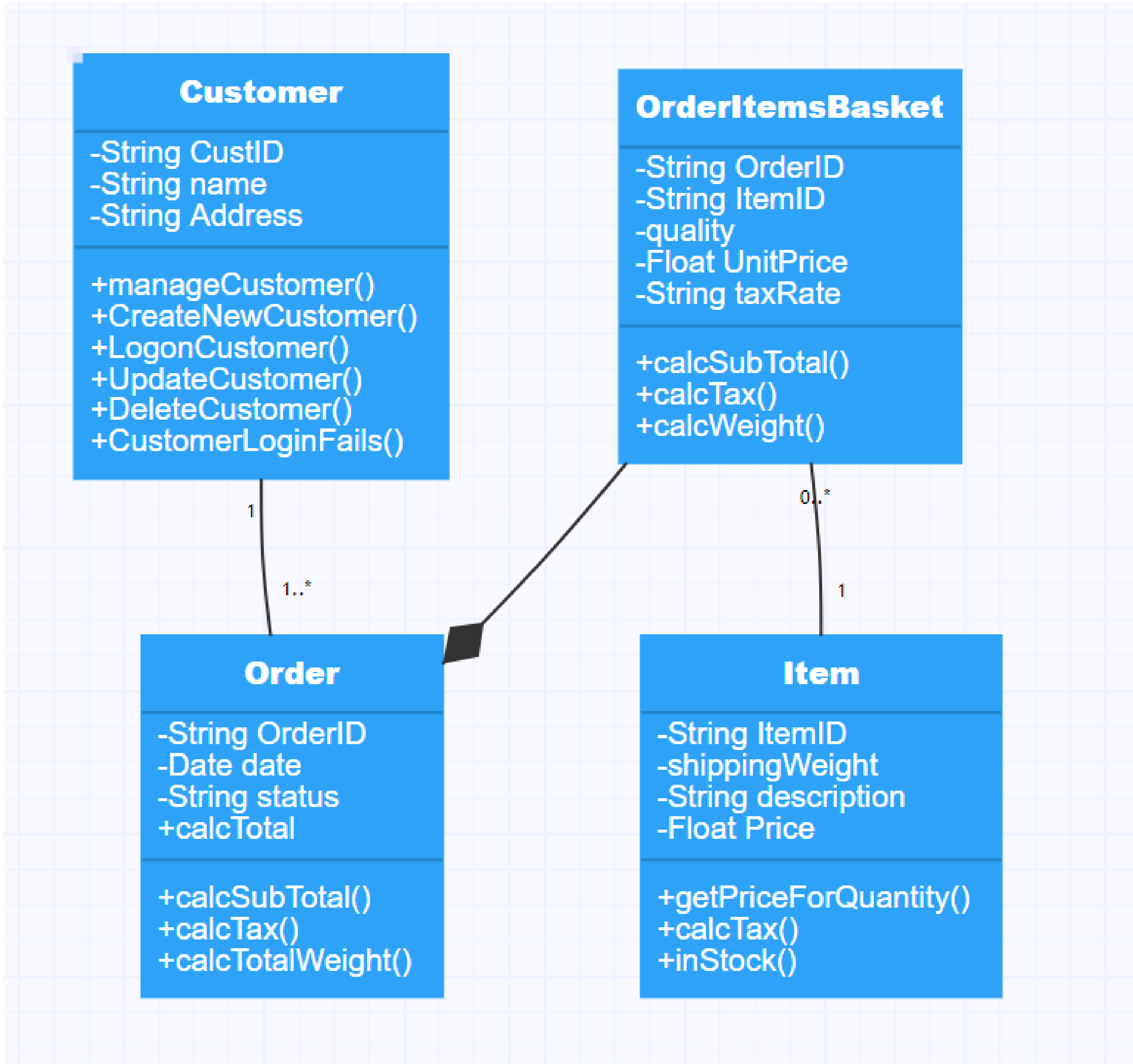
 Our Class Diagram introduces OrderItemsBasket aka OrderItems or OrderLine.

Many to Many M:N always decompose into 2 x 1:m
Will be Covered under ERD

```
Customer {1}--{1..*} Order
Order <*>-- OrderItemsBasket {0..*}--{1} Item
```

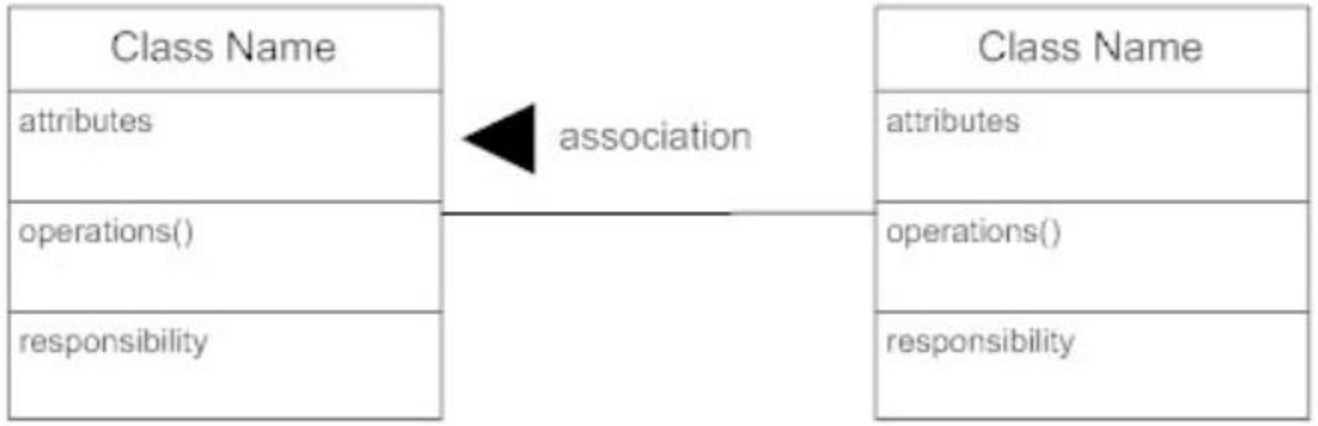
Customer {1}--{1..*} Order

Order <*>-- OrderItemsBasket {0..*}--{1} Item



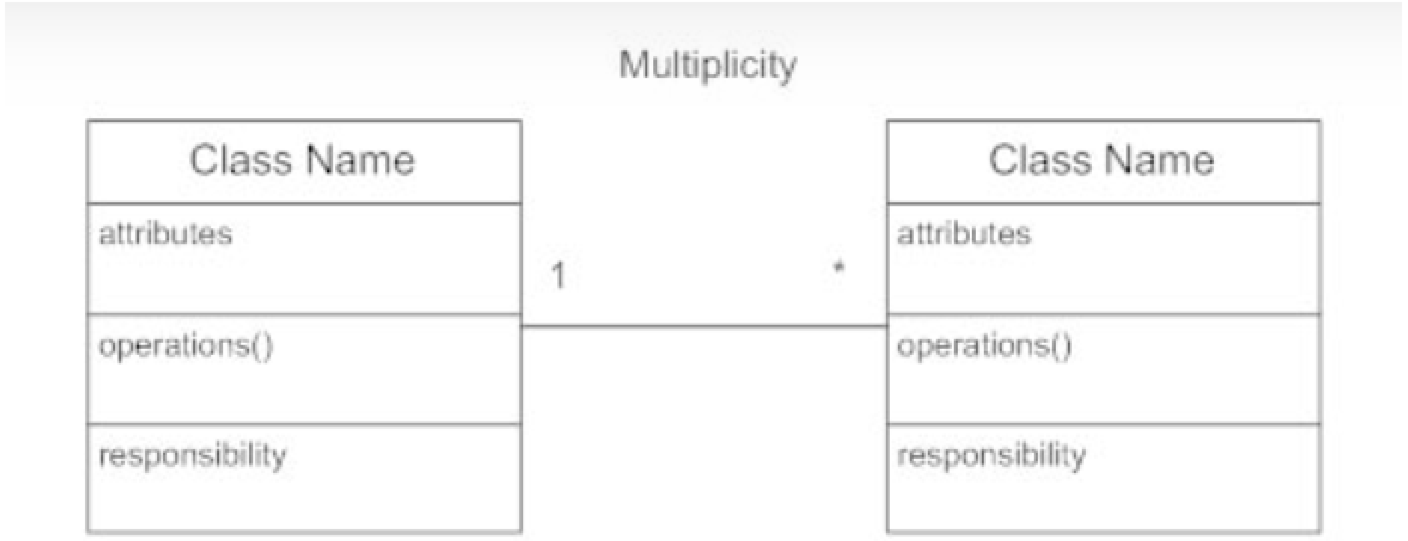
Associations

- **Associations** represent static **relationships between classes**.
  - Place association names above, on, or below the association line.
  - Use a filled arrow to indicate the direction of the relationship.
  - Place roles near the end of an association.
  - Roles represent the way the two classes see each other.
  - It is the Relationship between classes. The connecting line in the diagram below.



Multiplicity (Cardinality) – Relationship Degree

- Place multiplicity or relationship degree notations near the ends of an association.
- These symbols indicate the number of instances of one class linked to one instance of the other class.
- For example, one **customer** will have none, one or more **orders.**
- Note the 0 defines a **customer** may exist without having an **order.**
- But each **order** must be associated to only one **customer**.



Multiplicity (Cardinality) – Relationship Degree

- one **customer** will have none, one or more **orders.**
- one **order** must be associated to only one **customer**.

- An order can have many items
- An item can be on many orders

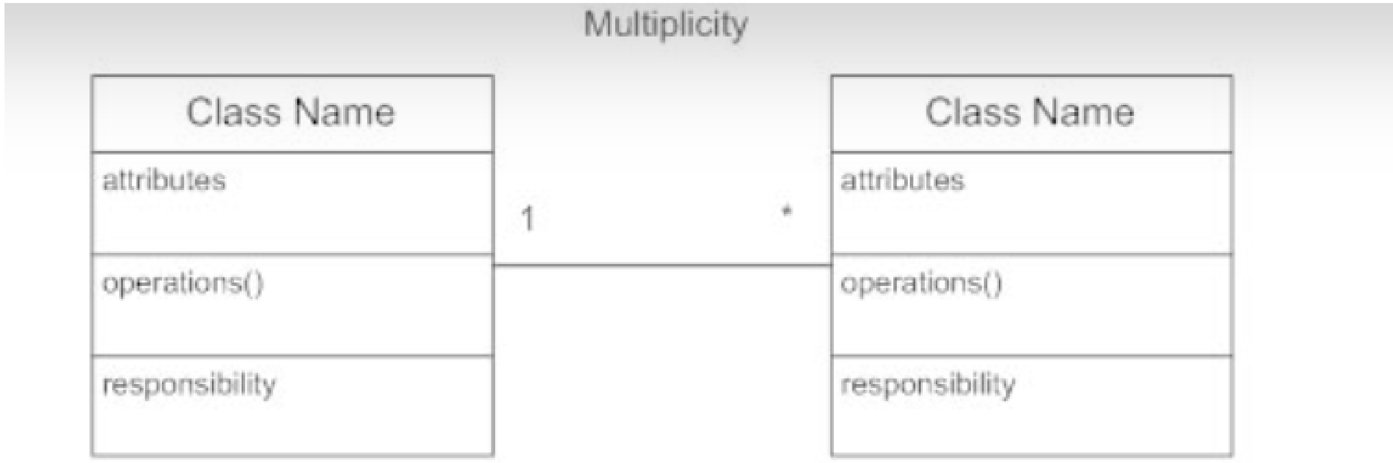
It is a many to many relationship!

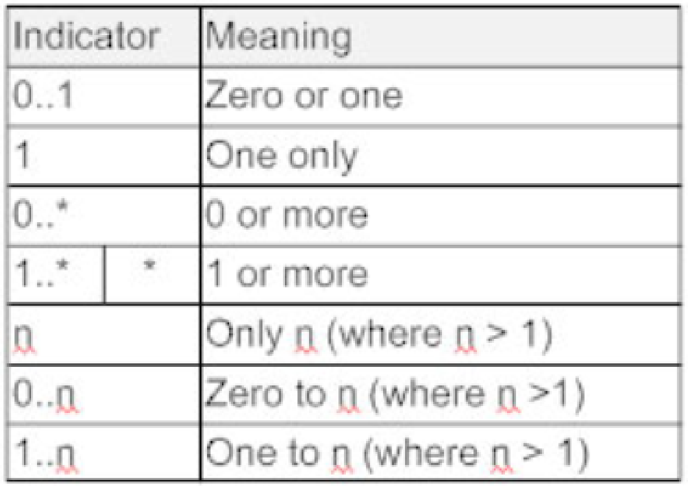
Just for reference:

All relational database systems need many to many relational ships decomposing into:
2 x 1:m relationships.

Normalisation and **Entity Relationship Diagrams** produce the decomposition.

We will cover the many to many decomposition process in next ERD Lecture

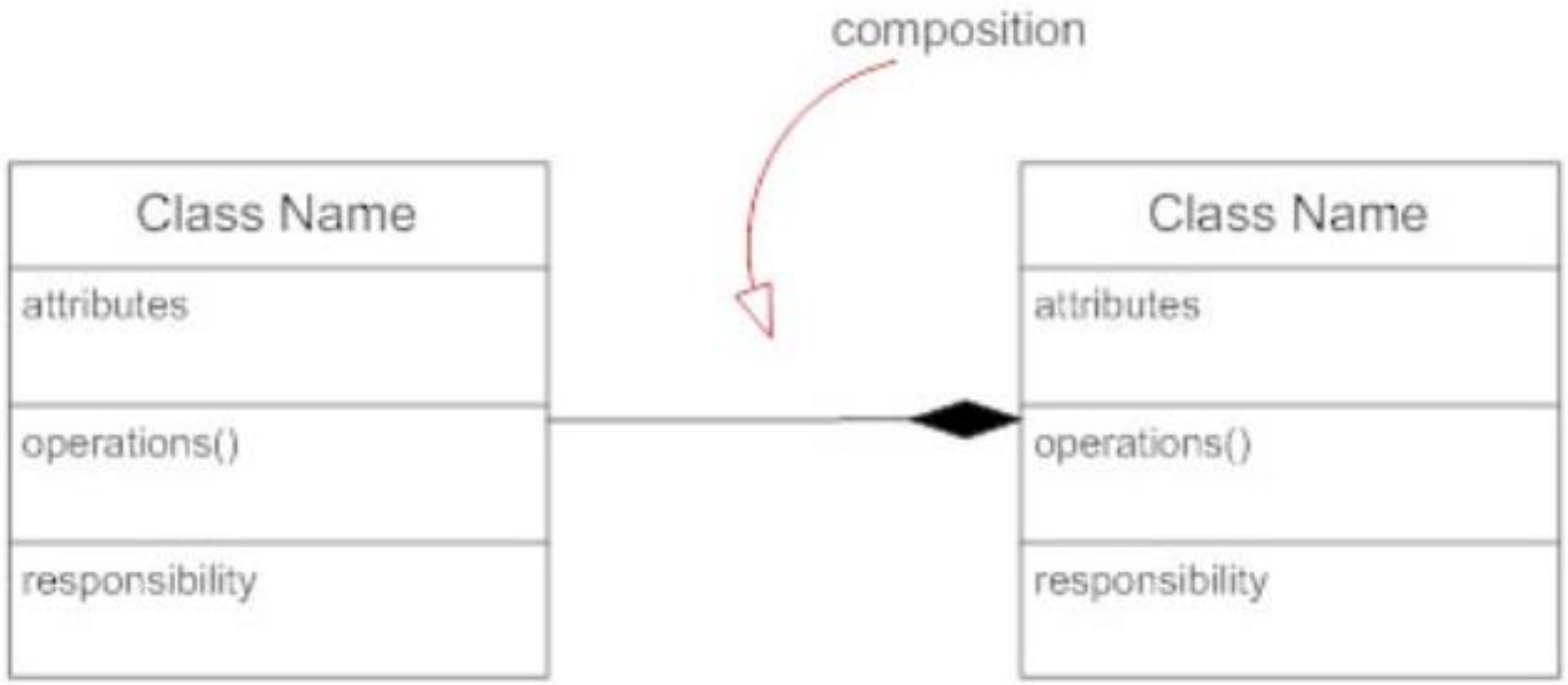




Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

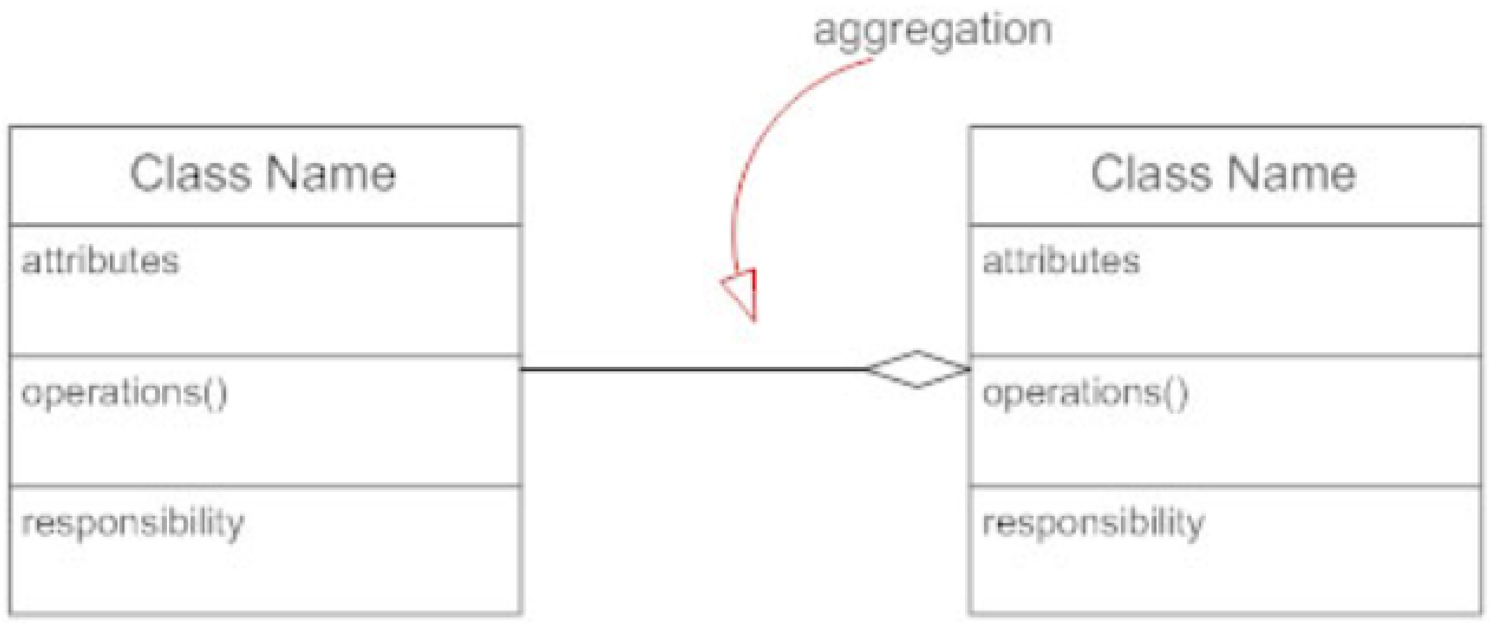
Composition and Aggregation

- Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part.
- **Is this similar to the way we indicate 'extended' use case?**
- Illustrate composition with a filled diamond.



Composition and Aggregation

- To represent a simple **aggregation** relationship use a hollow diamond to represent the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other.
- **Is this similar to the way we indicate ‘include’ use case?**
- The diamond ends in both composition and aggregation relationships point toward the "whole" class (i.e., the aggregation).



Supporting Material

- Highly recommended to walk through in your own time

Sequence Diagram - Video Tutorial

UML Behavioural Diagrams: Sequence - Georgia Tech - Software Development Process– 2mins

SEQUENCE DIAGRAM
interaction diagram that emphasizes the time ordering of messages

https://youtu.be/XIQKt5Bs7II

Sequence Diagram - Video Tutorial

5 Steps to Draw a Sequence Diagram – 9mins

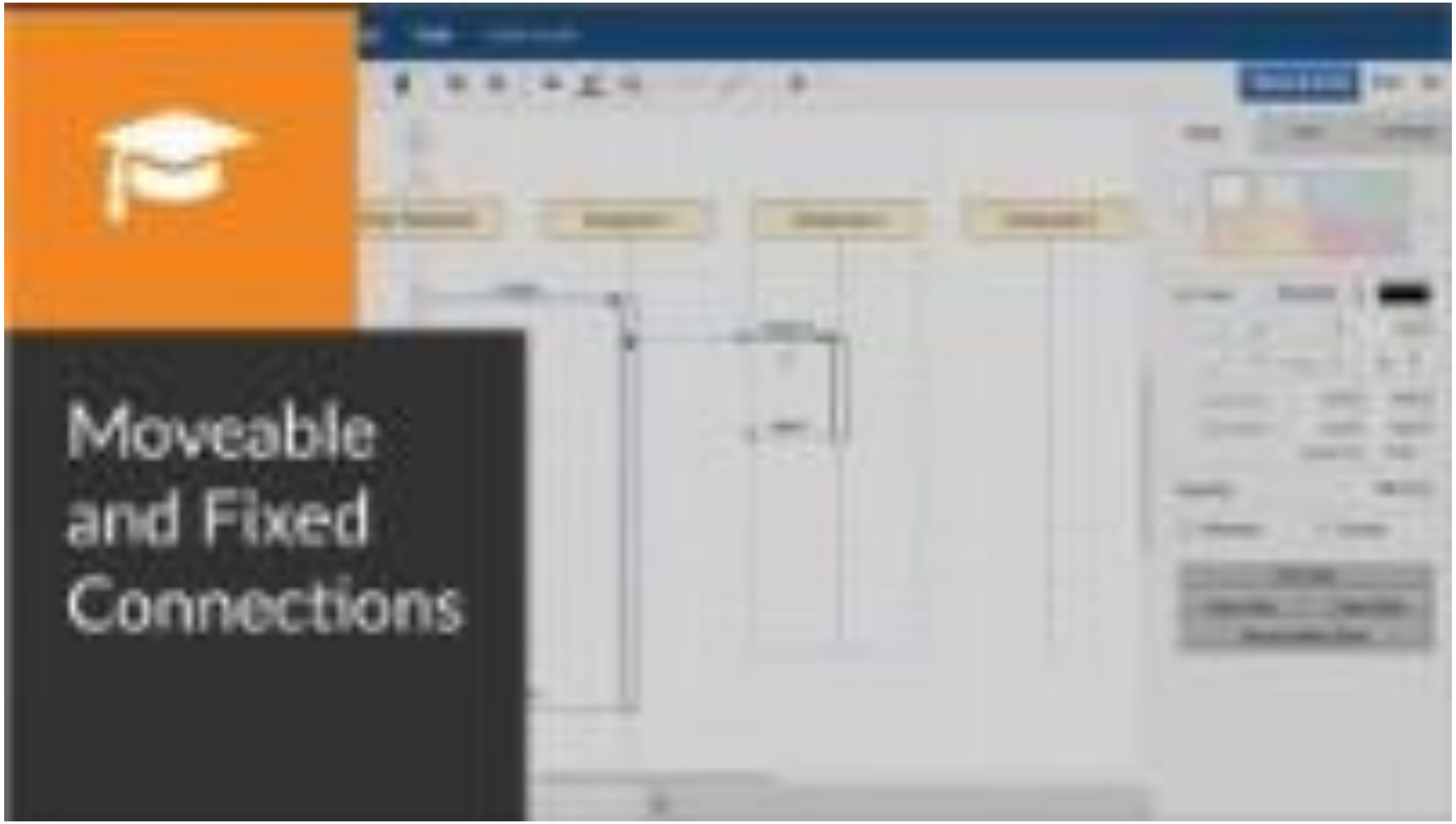
https://youtu.be/pCK6prSq8aw

Sequence Diagram - Step by Step Guide with Example – 26 mins

https://youtu.be/_Mzi1rYtI5U

Sequence Diagram - draw.io Video Tutorial

Timelines: How to use floating and fixed connectors in draw.io diagrams – 2mins



https://youtu.be/XC25MIcxwqU

UML Class Diagram Tutorial

https://youtu.be/UI6lqHOVHic