Objective-Oriented Programming

Week 4

aka OOP,
OO Programming, OO

Object Oriented Programming (OOP)

- In OOP all (or most) code is in classes
- It gives us a new way of thinking about problems and programs.
  - What *classes* of data are we dealing with?
  - What are the *attributes* of each class?
  - What *operations* do we perform on these attributes?
    - i.e. what *methods* should we write?

Classes as “records”

In their simplest form, classes give us a way of collecting related variables into a single object.

Where do classes come from?

- The programmer has to come up with the classes to use in a program. How?
- As with functions:
  - By discovery
    - "I seem to have lots of functions taking these three parameters. Maybe they belong together as a type of object?"
  - By top-down design (OO design)
    - Analyze the problem before starting.

```python
def calculate_area(length, width, height):
    return length * width * height

def calculate_perimeter(length, width, height):
    return 4 * (length + width + height)

def print_dimensions(length, width, height):
    print(f"Length: {length}, Width: {width}, Height: {height}")

# Using the functions
length = 5
width = 3
height = 2

print(calculate_area(length, width, height))
print(calculate_perimeter(length, width, height))
print_dimensions(length, width, height)
```

```python
class Box:
    def __init__(self, length, width, height):
        self.length = length
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.length * self.width * self.height

    def calculate_perimeter(self):
        return 4 * (self.length + self.width + self.height)

    def print_dimensions(self):
        print(f"Length: {self.length}, Width: {self.width}, Height: {self.height}")

# Using the class
box = Box(5, 3, 2)

print(box.calculate_area())
print(box.calculate_perimeter())
box.print_dimensions()
```

By top-down design (OO design)

Analyze the problem before starting.

Noughts and crosses classes

- Some candidates might be:
  - game
  - player
  - board
  - row
  - column
  - diagonal
  - line

I've chosen to run with just these three

- Consider what the *fields* and *methods* of each might be

(aka *instance variables*)

Thinking goes like this ...

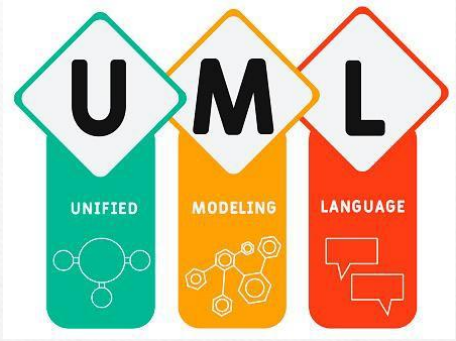
- Let's think about the *Game* class first.
  - What are its *attributes* (instance variables)? That is, what forms a game of noughts and crosses?
  - Well, there are 2 players and a board.
  - Now, what are the *methods* of game? That is, what can you do to/with a game?
    - ✔ Well, you can *play* it.
    - ✔ You keep playing until the *game_is_over*.
    - ✔ And a game is over when it's either won or drawn, so there's another couple (or perhaps just one - *state_of_game()?)*
- What about a *Board*?
  - It needs to have a grid of squares for starters.
  - Methods? Perhaps *make_move()?*
  - And a way to ask if it's won or drawn?
- *Player*?...and so on

Object Oriented Analysis and Design (OOAD)

Outline

- **Unified Modelling Language**
  - Use-Case Diagrams
  - Activity Diagrams
- **Class Design**
  - Identify Class and Objects
- **Class Relationships**
  - Association
  - Aggregation
  - Composition

Unified Modelling Language (UML)



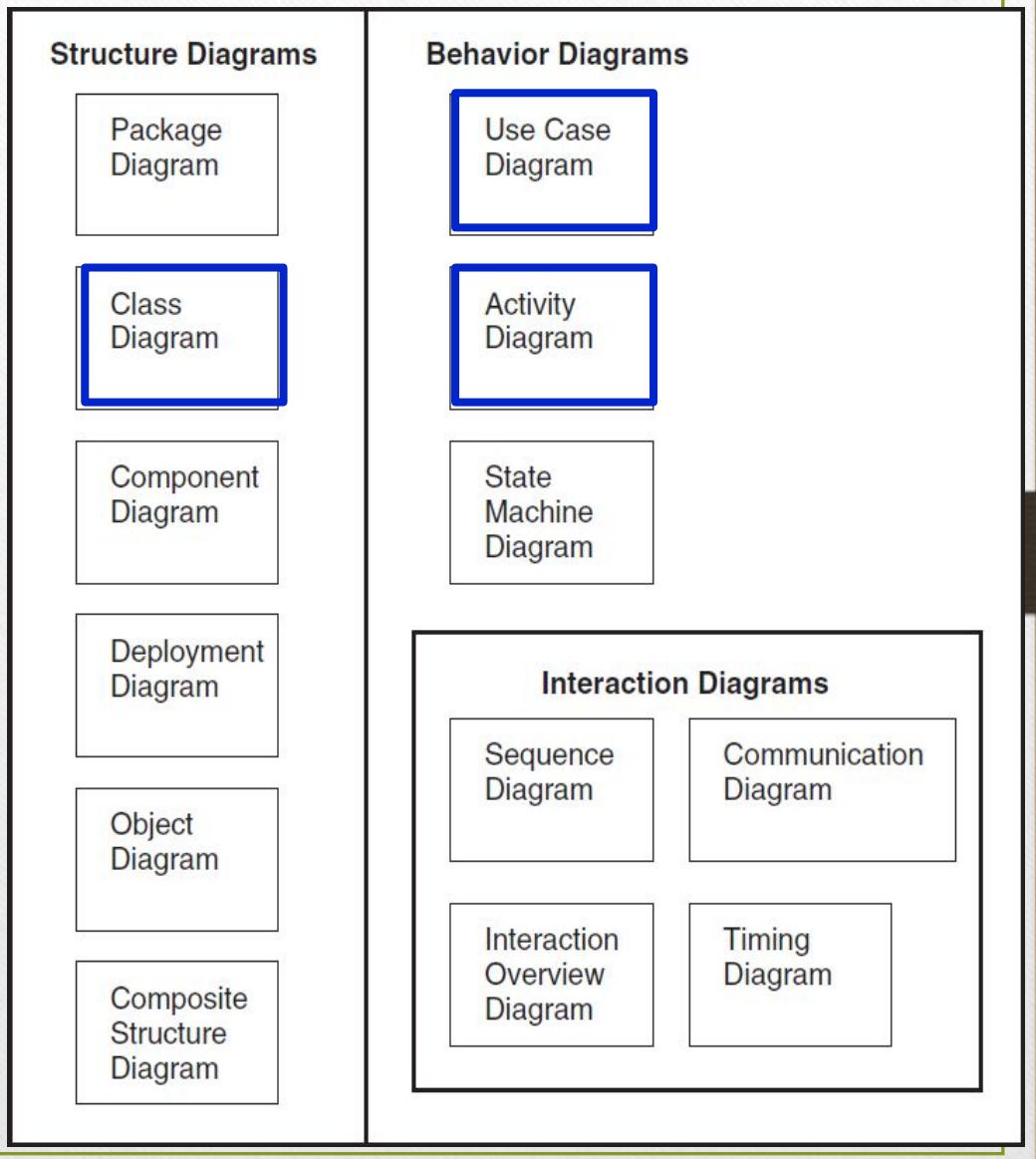
- Language: Express idea, NOT a methodology or procedure
- Modelling: Describing a software system at a high level of abstraction
- Unified: UML has become a world standard

OMG™
Object Management Group®

Unified Modelling Language (UML)

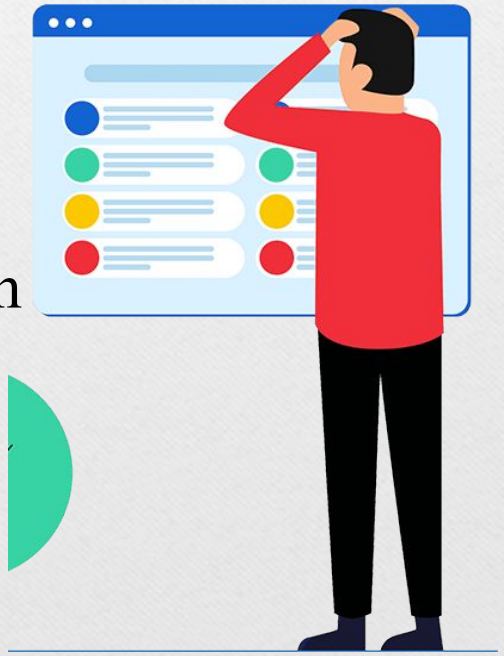
- UML is a Graphical Language for visualising, specifying, constructing, and documenting the artefacts of software systems
- The UML uses mostly graphical notations to express the OO Analysis and Design of software projects.
- Helps obtain an overall view of a system.
- Simplifies the complex process of software design

- Structure Diagrams
  - represent the static aspects of the system
  - document the software architecture
- Behaviour Diagrams
  - represent the dynamic aspect of the system
  - describe the functionality of software systems
- Interaction Diagrams
  - a subset
  - emphasize the flow of control and data among the things in the system being modelled

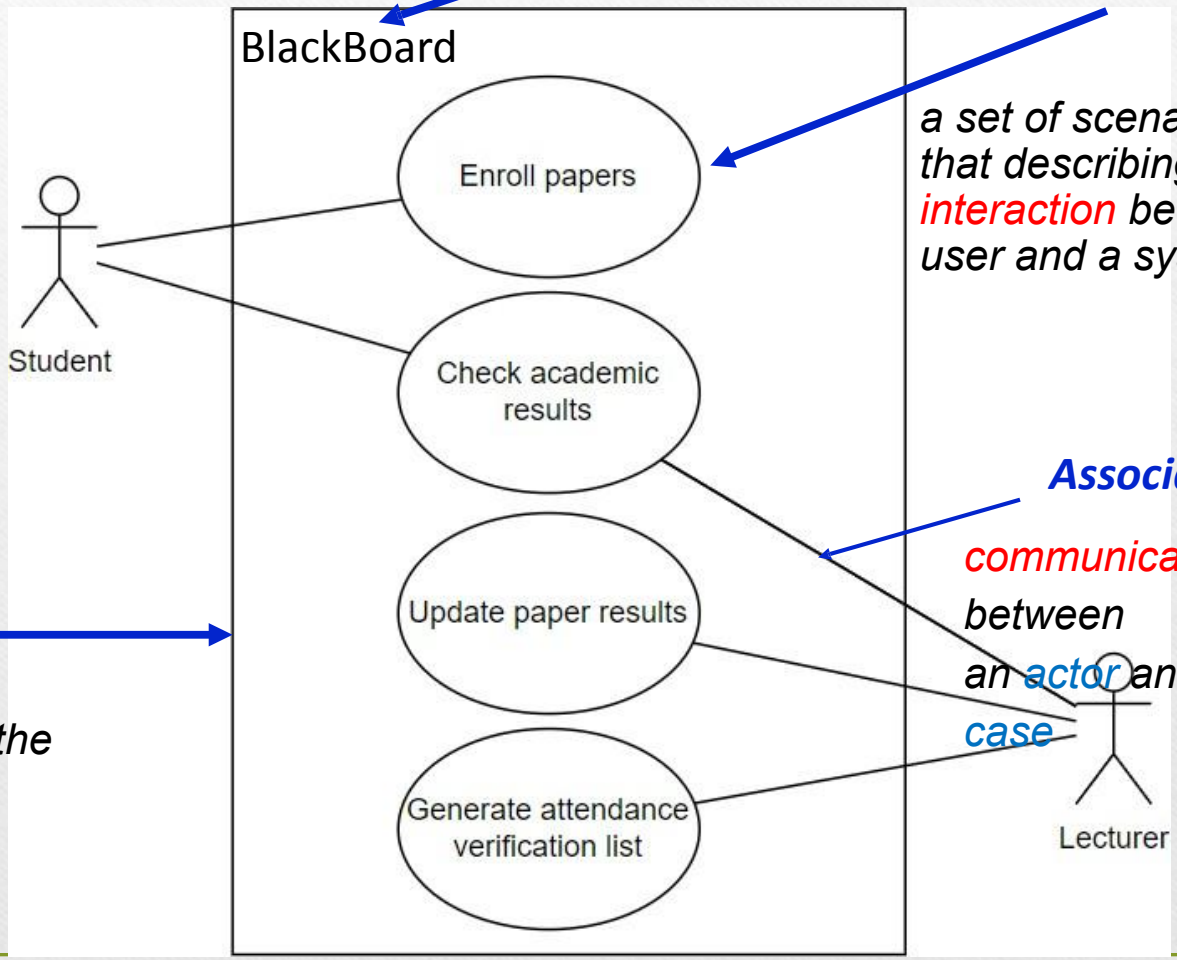


Use Case Diagrams

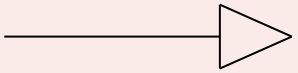
- A use case diagram is a set of use cases
- A use case is a model of the interaction between
  - External users of a software product (actors) and
  - The software product itself
  - More precisely, an actor is a user playing a specific role
- Display who (or what) interacts with the system
- Capturing user requirements

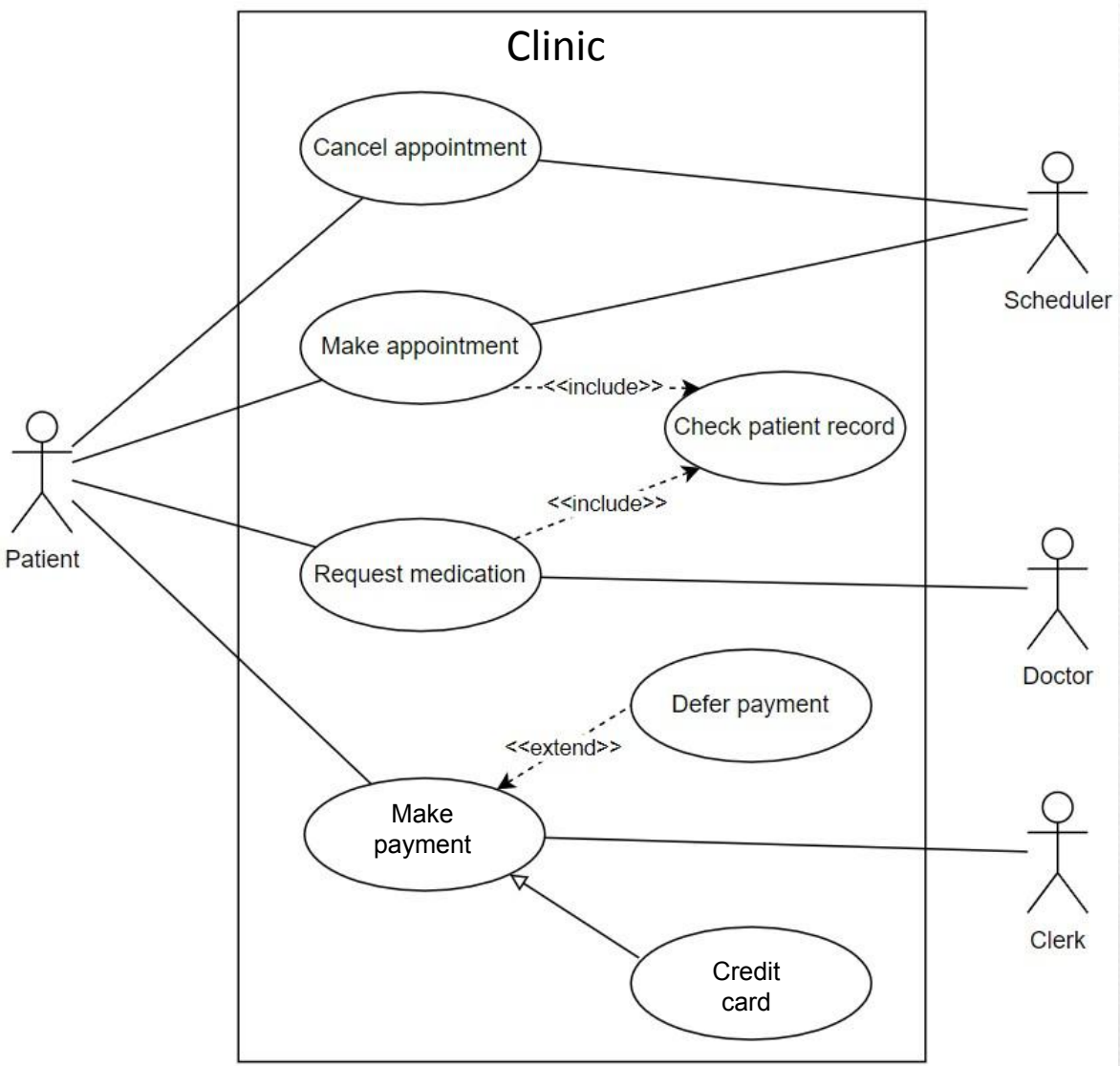


Use Case Diagrams Example



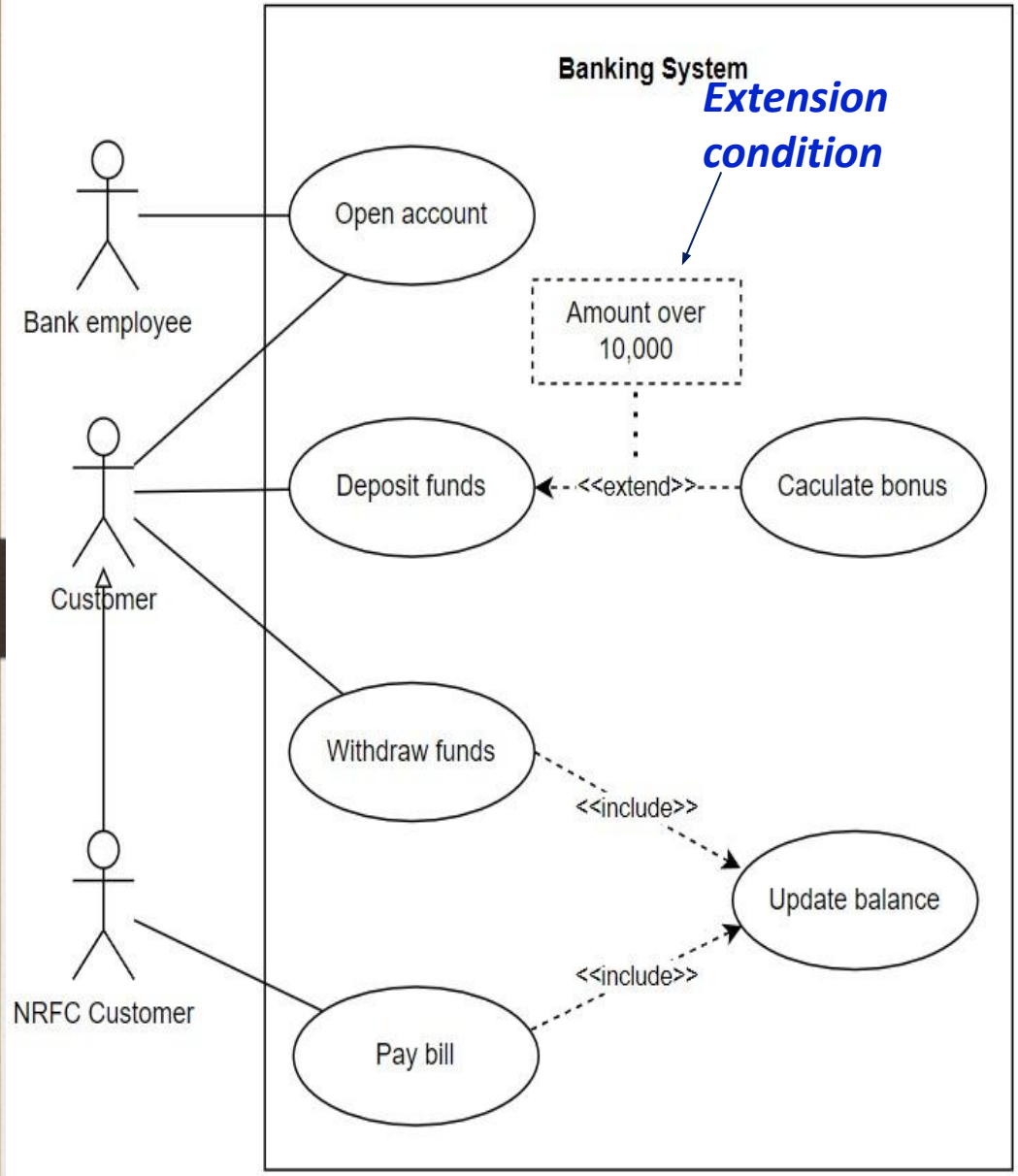
Relationships in Use Case Diagrams

Relationship	Symbol	Meaning
Association		communication between an actor and a use case
Generalisation		use cases can have common behaviours that other use cases (i.e., child use cases) can modify by adding steps or refining others
Include	<<include>>	one use case (the base use case) includes the functionality of another use case (the inclusion use case)
Extend	<<extend>>	one use case (extension) extends the behaviour of another use case (base)



Relationship Example

- Include: Both Make appointment and Request medication include Check Patient Record as a subtask.
- Extend: To complete Pay bill, there is an option to defer payment.



Relationship Example 2

Use Case Descriptions

- User case descriptions tend to be written at two separate levels of detail: brief description and fully developed description.

Use case	Brief use case description
Create customer account	User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.
Look up customer	User/actor enters customer account number, and the system retrieves and displays customer and account data.
Process account adjustment	User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment.

Full Developed Use Case Descriptions

- The fully developed description is the most formal method for documenting a use case.

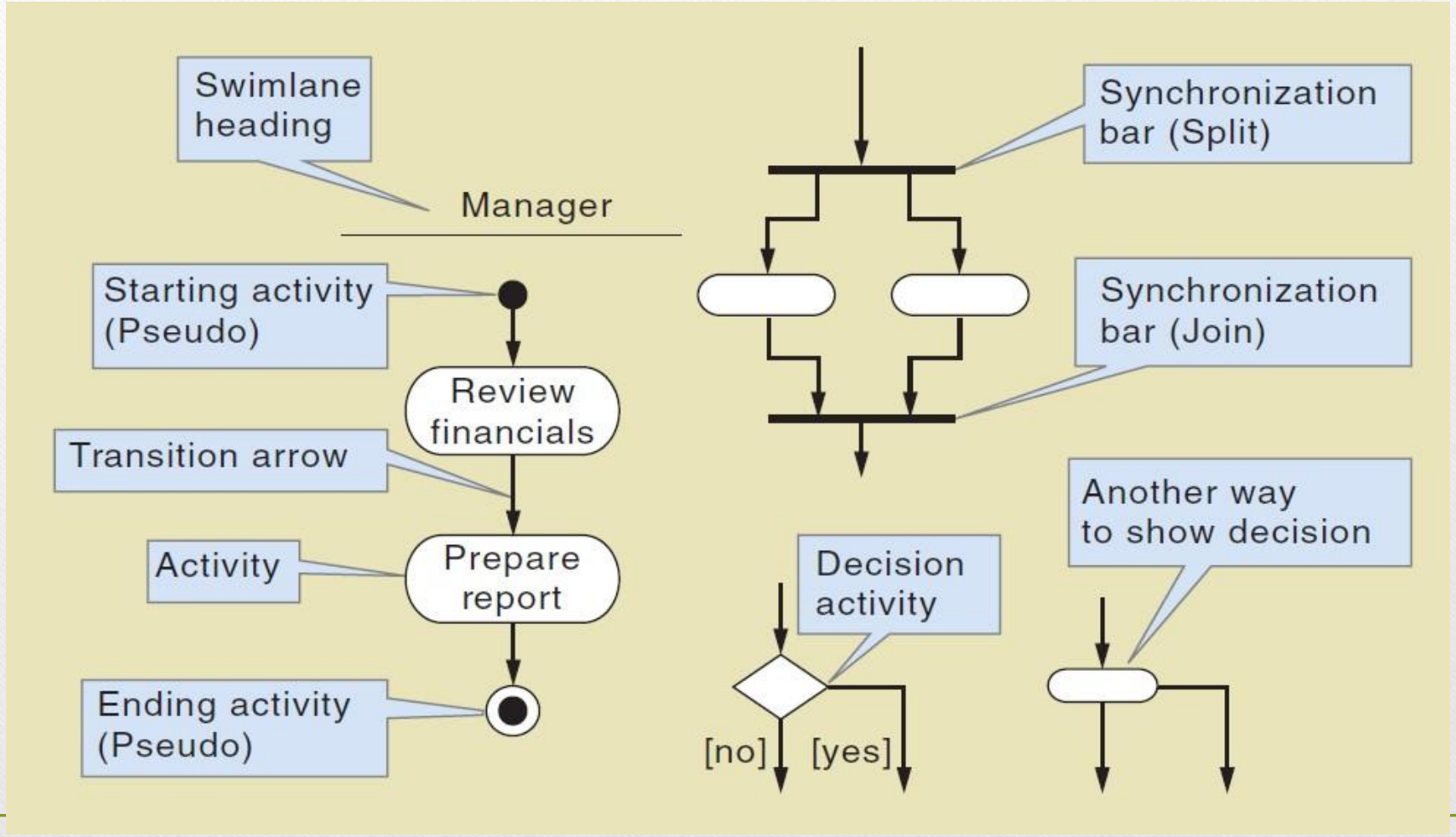
Use case name:	*Create customer account.*
Scenario:	Create online customer account.
Triggering event:	New customer wants to set up account online.
Brief description:	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.
Actors:	Customer.
Related use cases:	Might be invoked by the *Check out shopping cart* use case.
Stakeholders:	Accounting, Marketing, Sales.
Preconditions:	Customer account subsystem must be available. Credit/debit authorization services must be available.
Postconditions:	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.
Flow of activities:	**Actor** / **System** 1. Customer indicates desire to create customer account and enters basic customer information. 1.1 System creates a new customer. 1.2 System prompts for customer addresses. 2. Customer enters one or more addresses. 2.1 System creates addresses. 2.2 System prompts for credit/debit card. 3. Customer enters credit/debit card information. 3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
Exception conditions:	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.

Activity Diagram

- Activity diagram describes user (or system) activities, the person who does each activity, and the sequential flow of these activities.

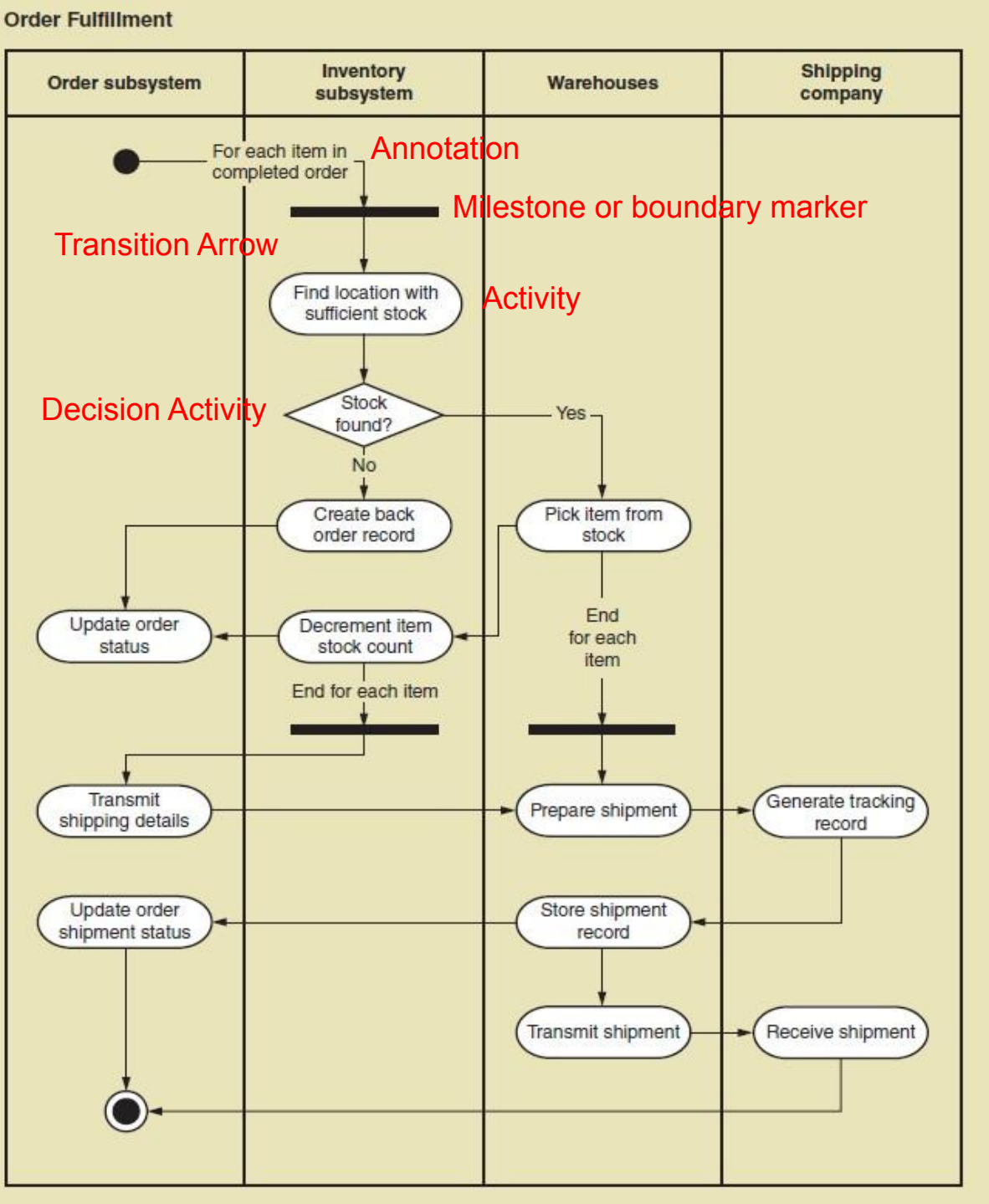
- Activity Diagram **for Use Case** refers to an activity diagram that documents the flow of activities for a particular use case.

Basic Symbols of Activity Diagrams



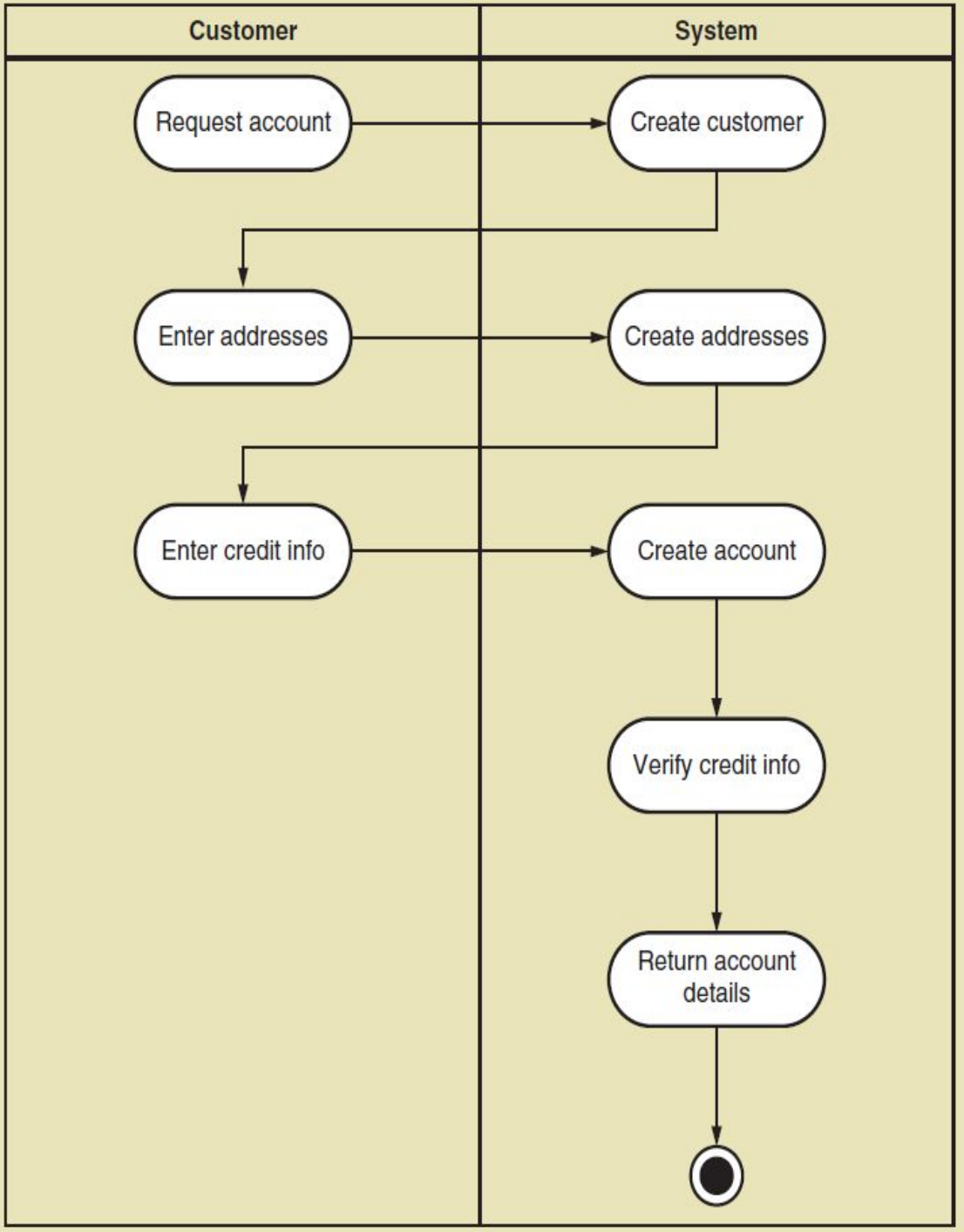
Activity Diagram Example

- workflow of order fulfilment
- The diagram shows the flow of information and control between the order subsystem, inventory subsystem, warehouse(s), and shipper



Activity Diagram for Use Case Example:

- Flow of activities for the Create customer account use case
- Supplement the use case description
- Two swimlanes, one for customer and one for the system
- The customer has three activities, and the system has five activities



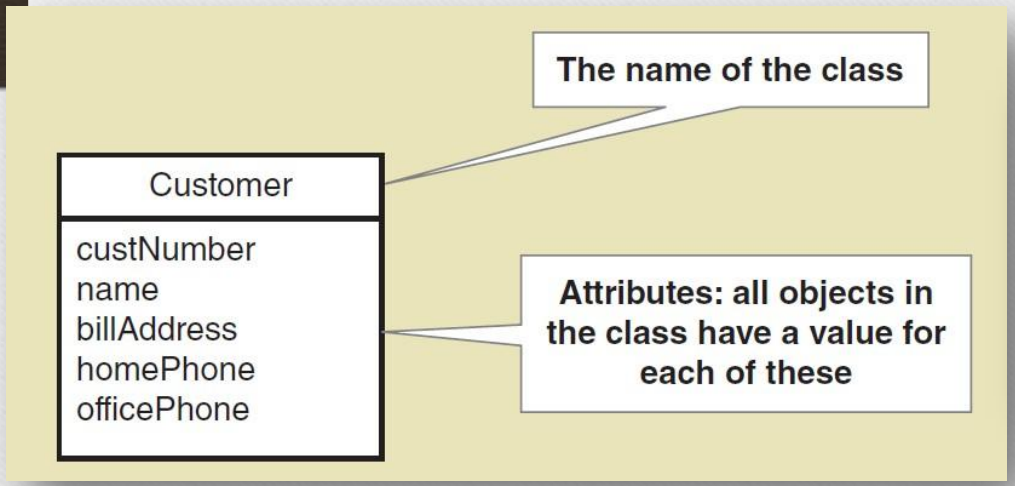
Class Design

Class Design

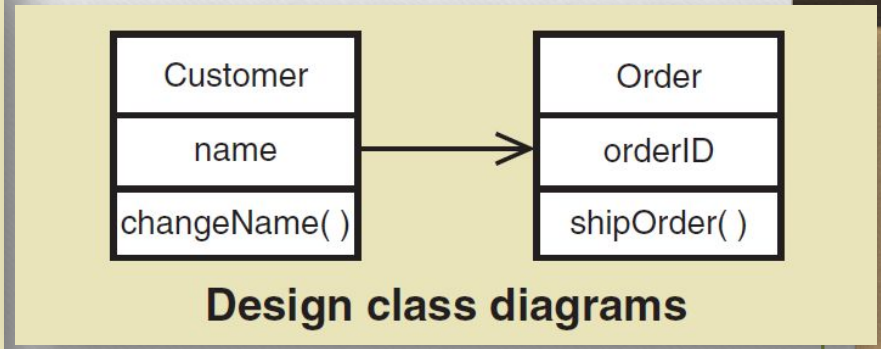
- Class Diagram
- Identify classes for the system
- Describe attributes and methods in each class
- Establish relationships among classes

Class Diagram

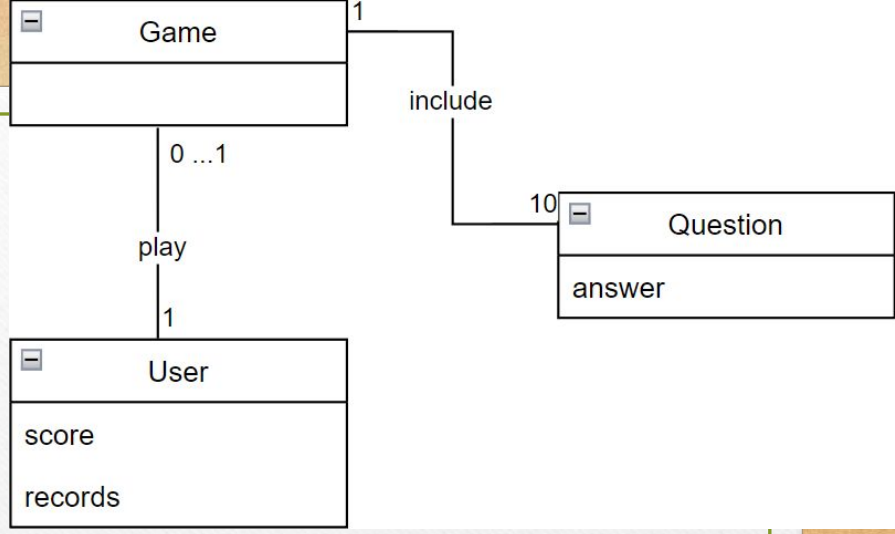
A class diagram is a type of static structure diagram that is used in software engineering to describe the structure of a system by showing its classes, their attributes, methods, and the relationships among objects.



Design models



Classes Diagram



- One way to identify classes is to identify the objects discussed in the program requirements, which are generally nouns
- Example: a math quiz game

Each **game** contains 10 randomly generated math **questions**. The **user** can answer each of the **questions** via the quiz window. If **the answer** is correct/wrong, the **user's score** will be increased/decreased by 10. The **user's score** in the **user's record** will be updated after each **game**.

Obviously, not all nouns are objects, some of them can be attributes of objects

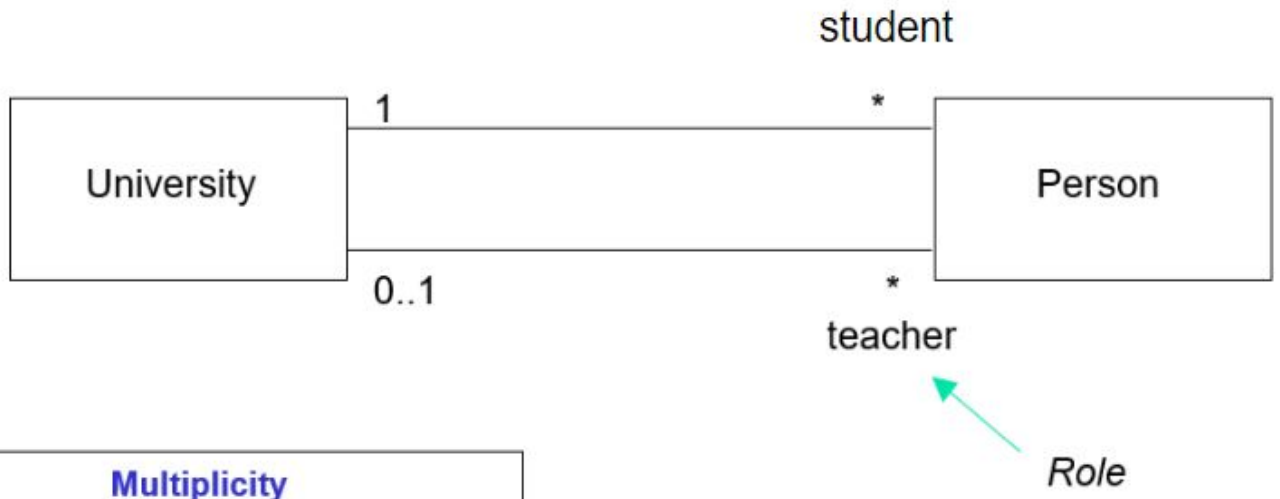
Class Relationship

There are two kinds of Relationships

- Generalization
- Association

- Associations can be further classified as
- Aggregation
- Composition

Association: Multiplicity and Roles



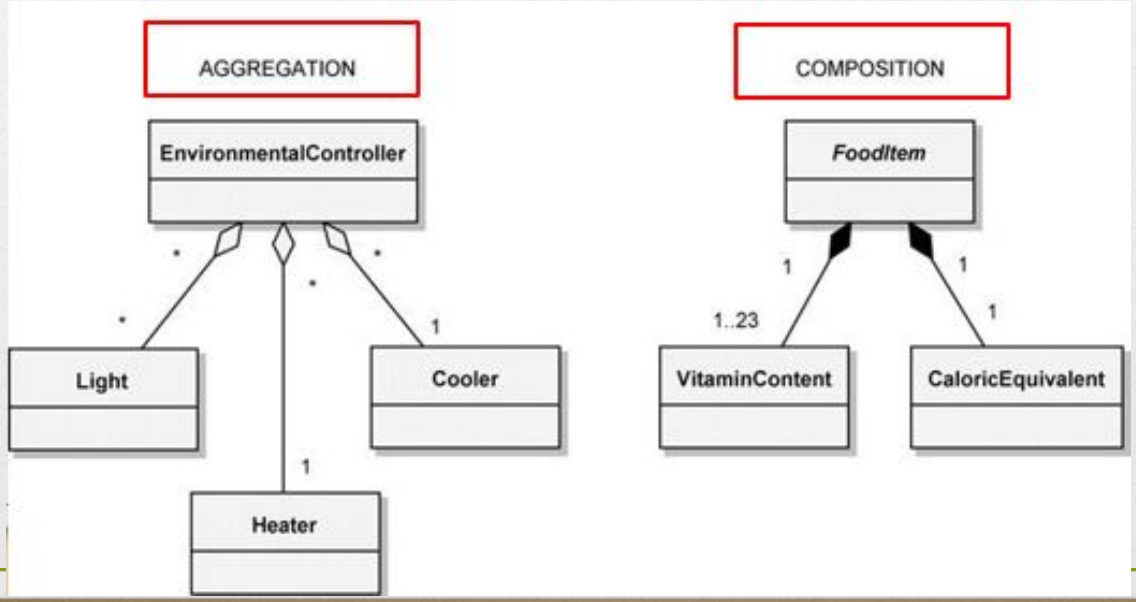
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

- Association
  - Aggregation: represents a "whole-part" relationship where the part can exist independently of the whole. '**Classroom**' and '**Student**'
  - Composition: represents a stronger form of the "whole-part" relationship, where the part is strongly dependent on the whole. The part cannot exist without the whole. '**House**' and '**Rooms**'



More Details

- List of UML tools: http://en.wikipedia.org/wiki/List_of_UML_tools
- More Information: https://www.youtube.com/watch?v=WnMQ8HlmeXc

Thank You