UML (Unified Modeling Language) For Communicating Design

CPEN333 – System Software Engineering

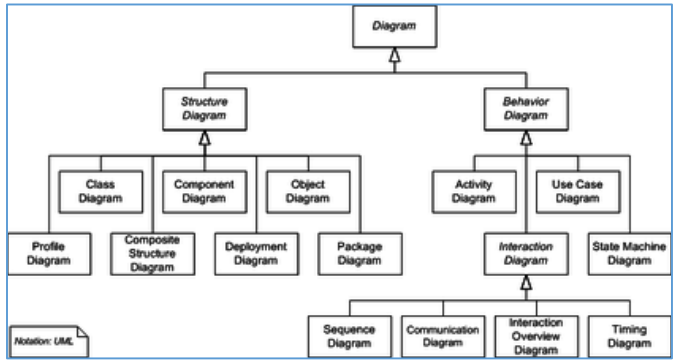
2021 W2

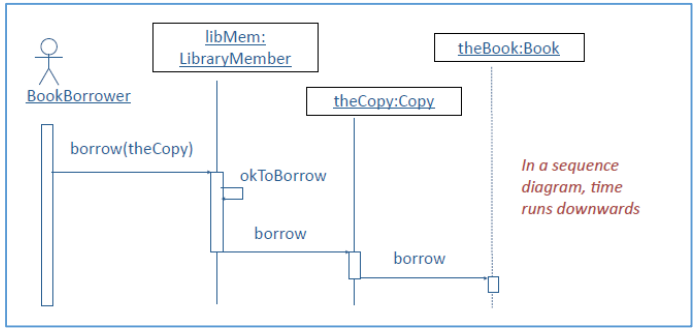
University of British Columbia

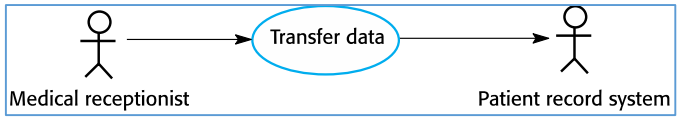
Introduction

- UML (Unified Modeling Language) is a modelling language.

- It helps us communicate the design and visualize the relationship between different parts of a program.
  - UML is programming language independent.







- There are many different types of UML diagrams, in this set of slides, we focus on the UML class diagrams, use cases and sequence diagrams.

Objectives

- Describe UML and the benefits of using UML
- Understand UML diagrams: class, use case, sequence
- Create UML diagrams for your program

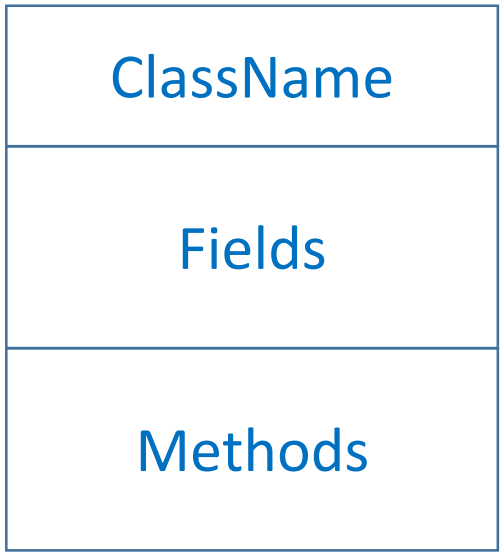
UML Class Diagram

- The UML class diagram allows us to show a set of classes used in our program and their relationship.
- Each class is represented by a diagram and we use links to show their relationships.
- A simple class diagram is a rectangle, just including the name of the class.
  - This model is used when we only want to identify the class in UML, but we would like to hide/abstract its members.
    - Example: *HelloWorld* class

  - We often though need a more complete representation of the class, including its members.

UML Class Diagram (cont.)

- The template of a more complete UML diagram for a class is shown below. It shows the name, fields (data members) and methods (function members).



The name of the class

Class fields (attributes)

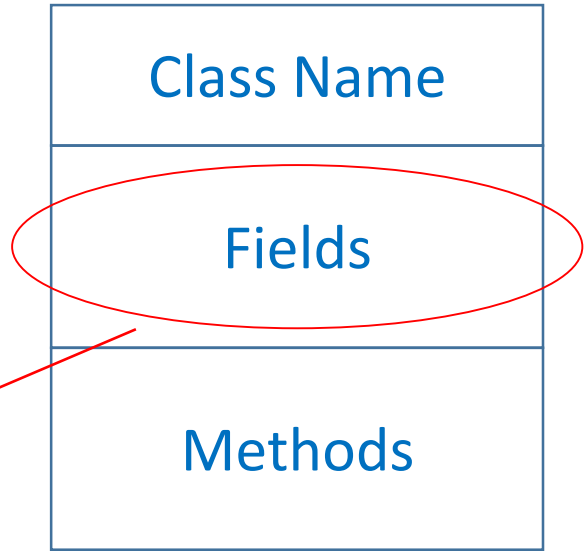
Class methods (operations)

- If there is no attribute or operation, we leave the corresponding space blank.

Fields in a UML Class Diagram

- In the fields (attribute) portion of the UML class diagram, we list all the fields.
  - Each field appears on one line.
  - We use the format

```
fieldName: fieldType
```



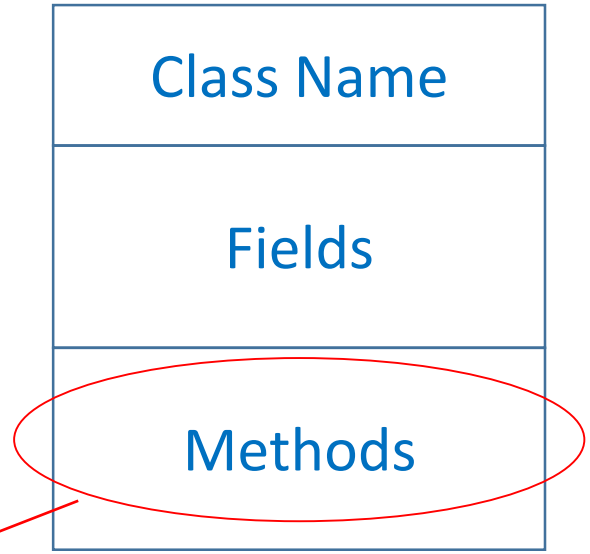
- Example:

```
name: str
gradeList: list
...
```

Methods in a UML Class Diagram

- In the methods portion of the UML class diagram, we list all the methods.
  - Each method appears on one line.
  - We use the format

```
methodName (parameter: type [separated by comma]): returnType
```



- Example:

```
isValidName (name: str): bool
getName (): str
add (a: float, b: float): float
...
```

UML Class Diagram Example

- A BMI class (BMI: Body mass index):

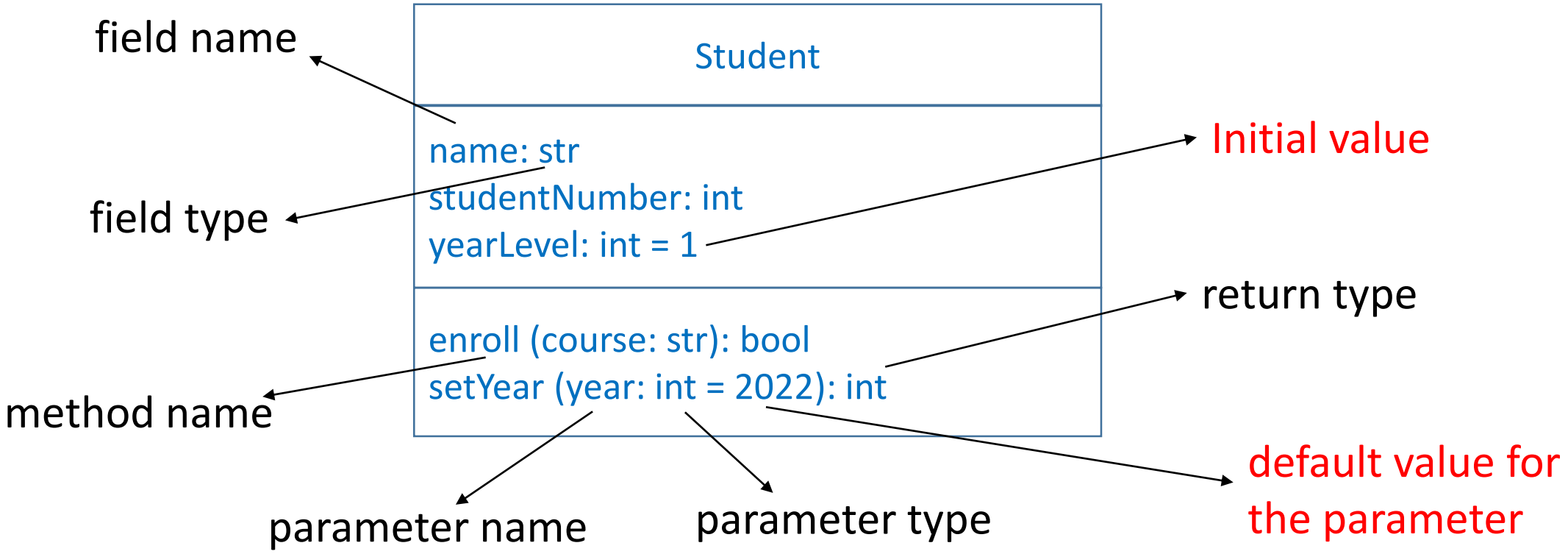
```
class BMI:
    def __init__(self, weight, height):
        self.weight = weight
        self.height = height

    def getBMI(self) -> float:
        bmi = self.weight / (self.height  ** 2)
        return round(bmi * 100) / 100

    def getStatus(self) -> str:
        bmi = self.getBMI()
        status = "Obese"
        if bmi < 18.5:
            status = "Underweight"
        elif bmi < 25:
            status = "Normal"
        elif bmi < 30:
            status = "Overweight"
        return status
```

UML Class Diagram Example 2

- Another class diagram also including:
  - initial values
  - parameter default value



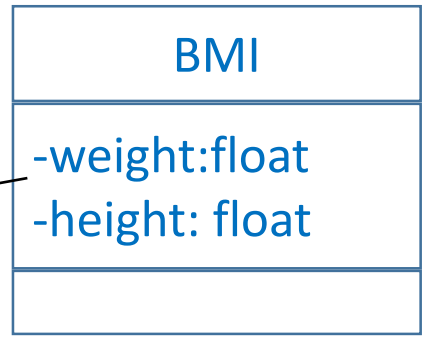
UML Class Diagram Visibility Symbols

- In general, a UML class diagram can also show visibility. In that case, the visibility symbol is put before a field.
  - – for a private member
  - + for a public member

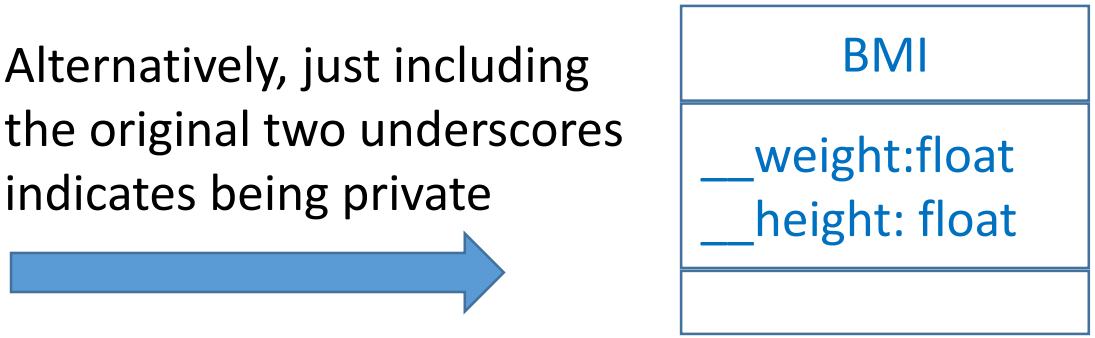
SYMBOL	MEANING	EXPLANATION
+	Public	The member is visible to all code in the application.
–	Private	The member is visible only to code inside the class.

- However for Python classes, we may only want to specify visibility for the fields.
- For example, if `__weight` and `__height` are private fields of a BMI class, we can specify them as follows:

visibility symbol (-) to indicate private members



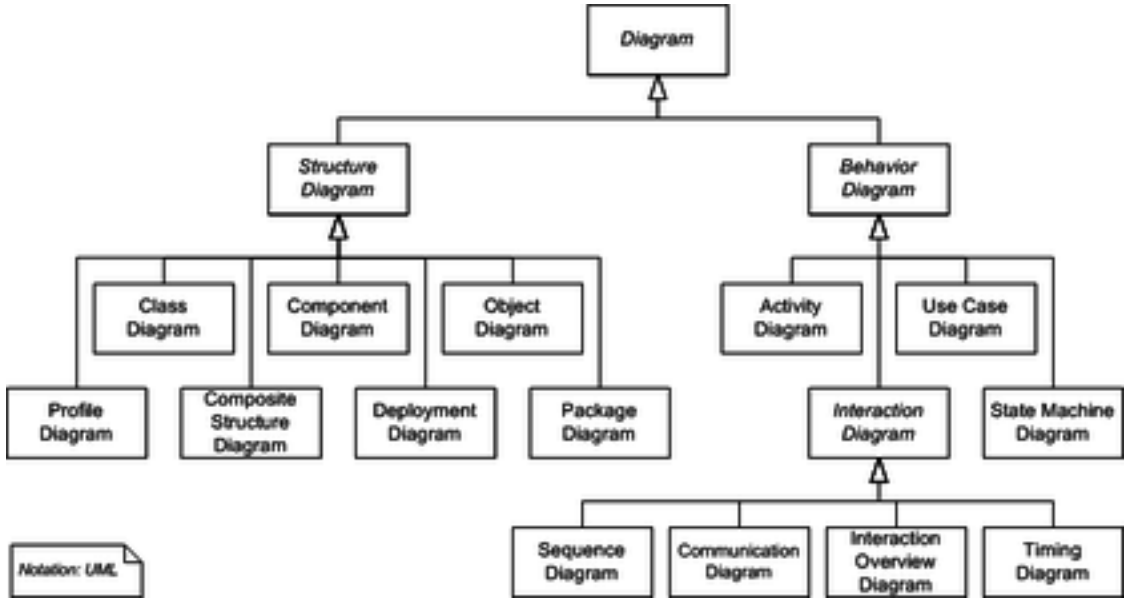
Alternatively, just including the original two underscores indicates being private



UML Relationships

- So far we have mainly considered a class on its own and discussed how to represent it using UML.

- Normally though, there are more than one class and we would like our UML class diagram to also represent the relationships (if any) between those classes.



UML Relationships (cont.)

- There are different types of relationships:
  - Inheritance (Generalization)
  - Association
  - Composition or Aggregation
  - Implementation (realization)
  - …
- In the class diagram, different classes are connected using different types of lines and arrows to show relationships (logical connections).

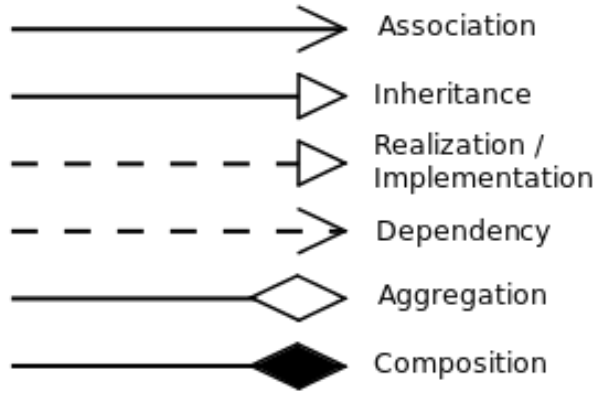


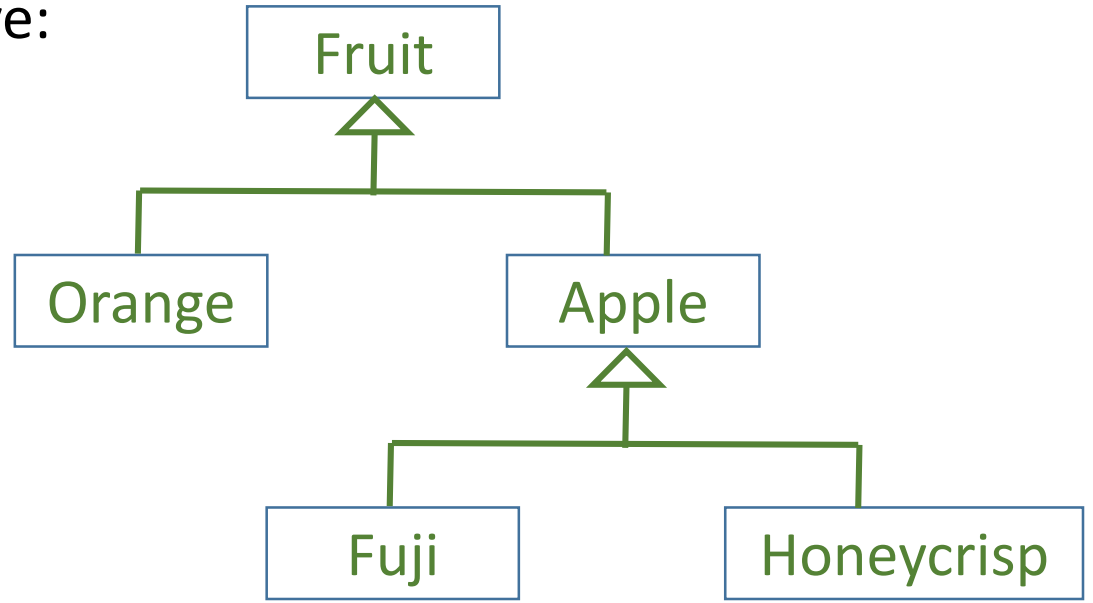
Image source: Wikipedia

UML Relationships: Inheritance

- Inheritance (generalization): we use the following line and arrow type for inheritance which is an *is-a* relationship:

- For example:
  - Suppose we have `Fruit`, `Apple`, `Orange`, `Fuji`, and `Honeycrisp` as classes defined for a program, we could have:

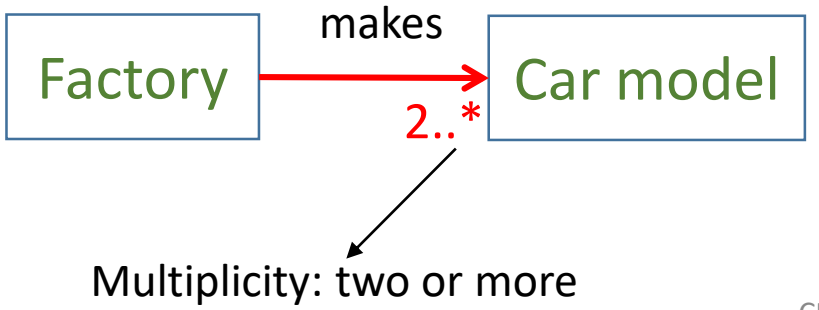
```
class Fruit:
    # ...
class Apple(Fruit):
    # ...
class Orange(Fruit):
    # ...
class Honeycrisp(Apple):
    # ...
class Fuji(Apple):
    # ...
```

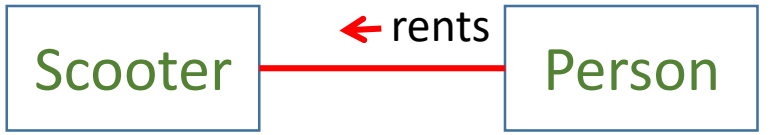


UML Relationships: Association

- Association is a common and general structural relationship that describes how two classes are associated or connected.
  - For example, if two classes in a model have a using or working relationship, that can be represented by an association.

- We use a solid line for the link:
  - The link could have directions to show unary or binary association.
  - Note the alternative ways of using the arrow in association below:





Example

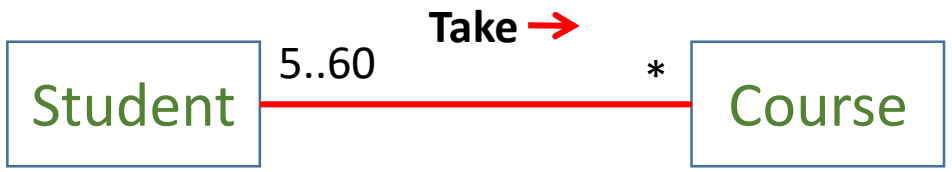
- A student taking a course is an association between the `Student` class and the `Course` class.

```
class Student:
    # ...
    def addCourse(self, course):
        #...
```

type Course

```
class Course:
    # ...
    def addStudent(self, student):
        #...
```

type Student



A student may take any number of course; and a course may have from five to sixty students.

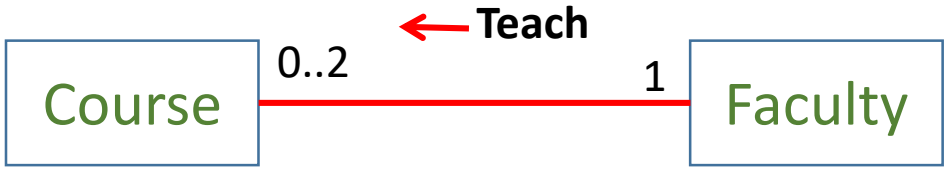
***** means an unlimited number, and **n..m** indicates that the number of objects is between m and n (inclusive).

Example (cont.)

- Also, a faculty member teaching a course is an association between the `Faculty` class and the `Course` class.

```
class Course:
    # ...
    def setFaculty(self, faculty):
        #...
```

```
class Faculty:
    # ...
    def addCourse(self, course):
        #...
```



A faculty member may teach at most two course; and a course is taught by one faculty member.

Example (cont.)

- Considering all three classes:

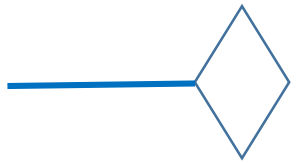
```
class Student:
    # ...
    def addCourse(self, course):
        #...
```

```
class Course:
    # ...
    def setFaculty(self, faculty):
        #...
    def addStudent(self, student):
        #...
```

```
class Faculty:
    # ...
    def addCourse(self, course):
        #...
```

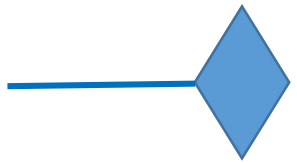
Aggregation and composition

- Aggregation is a special form of association that represents an ownership relationship between two objects.
- Aggregation models has-a relationship.



aggregation

- An object can be owned by several other aggregating objects.
- If an object is exclusively owned by an aggregating object, the relationship is referred to as a composition.



composition

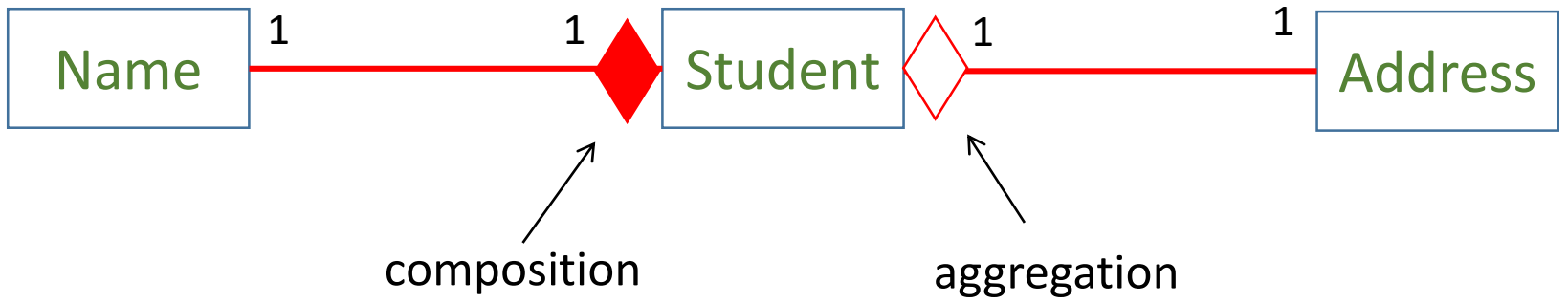
Example

- For example, a student has a name is a composition relationship between the `Student` class and the `Name` class.
- A student has an address is an aggregation relationship between the `Student` and the `Address` class (since an address can be shared by several students).

```
class Student:
    def __init__(self, name, address):
        self.name = name
        self.address = address
```

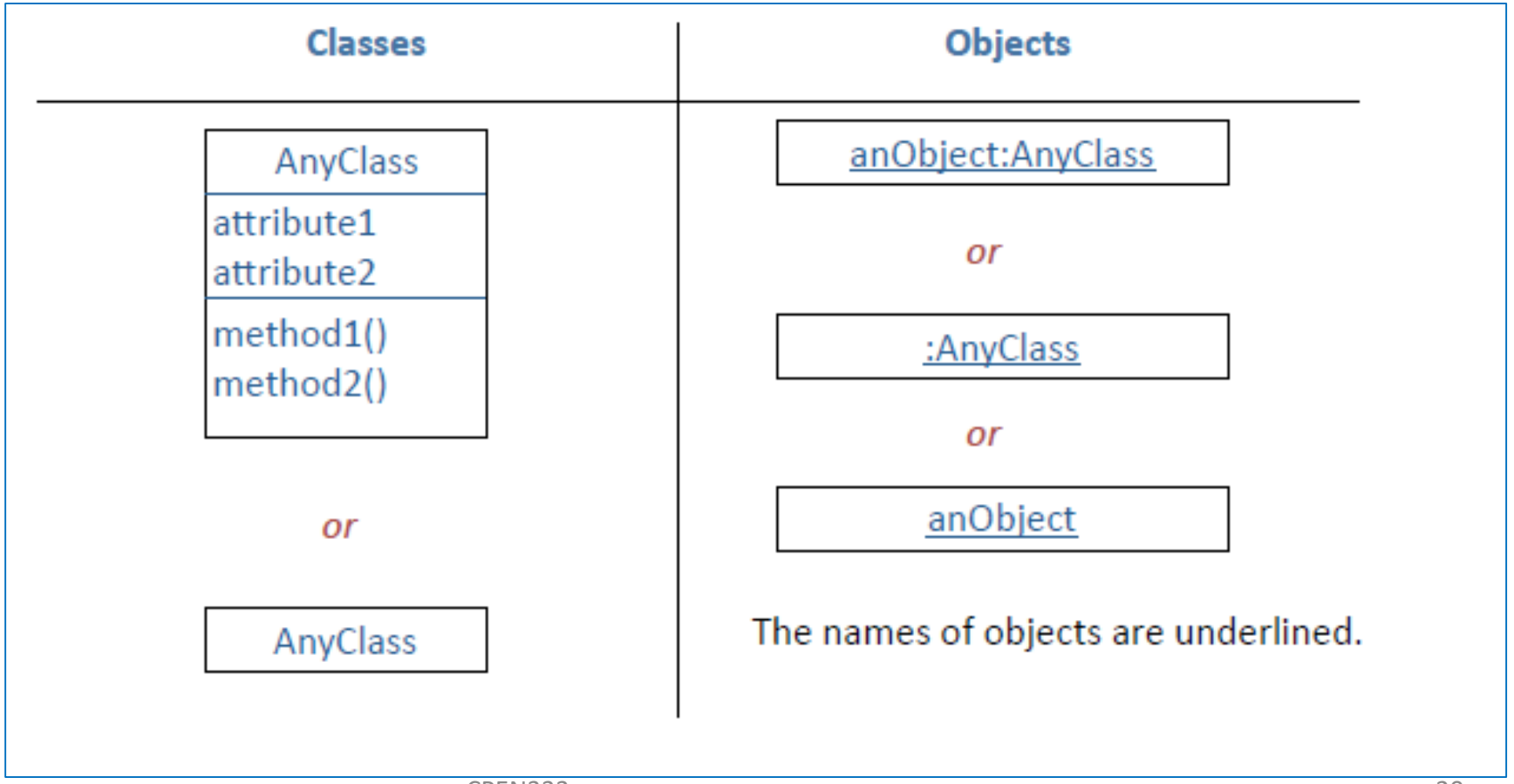
type Address

type Name



UML Notation for Objects

- An *object* diagram represents an instance of a class diagram.
- The figure shows three possible notations (in all, the names are underlined):



Modeling Interaction

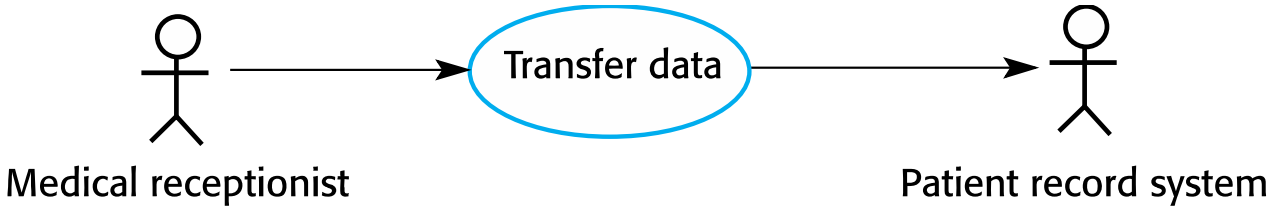
- A class diagram shows a static view (how classes are defined, and arranged …).

- A dynamic view represents the interactions of the objects in a system.
  - Since system behaviours can be complex, the dynamic view tends to look at smaller, discrete pieces of the a system like individual scenarios or operations.
- We will discuss two approaches to interaction modeling:
  - **use case** diagram
  - **sequence** diagram

UML Use Case

- Use case diagram is used to model interaction between a system and external agents, for example human users or other systems.

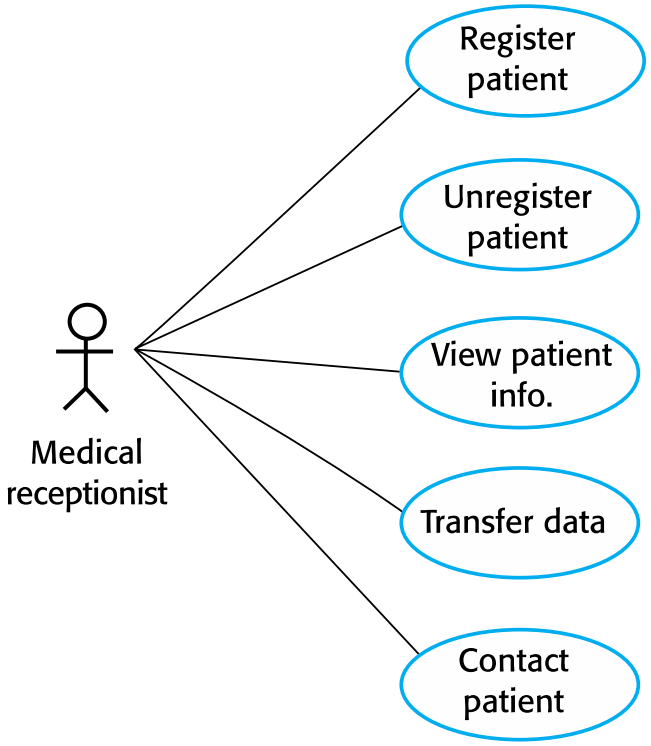
- In its simplest form, a use case is shown as an oval with the actors involved in the use case represented as stick figures (stickman).
  - An actor represents an *external entity* outside of the system we are modelling, such as the users or other systems.



Example: A use case from a medical software system, representing the task of uploading data by a medical staff to another patient record system.

Use Case Example 2

- Let's consider the same medical software system from the previous slide.
- The following diagram shows all the use cases in the system in which the actor "medical receptionist" is involved with.



UML Sequence Diagram

- A sequence diagram is a type of interaction diagrams
- It shows the interaction between the actors and the objects in the system, and the interactions between the objects themselves.
- Sequence diagrams normally show the time ordering of messages

Notations/Definitions

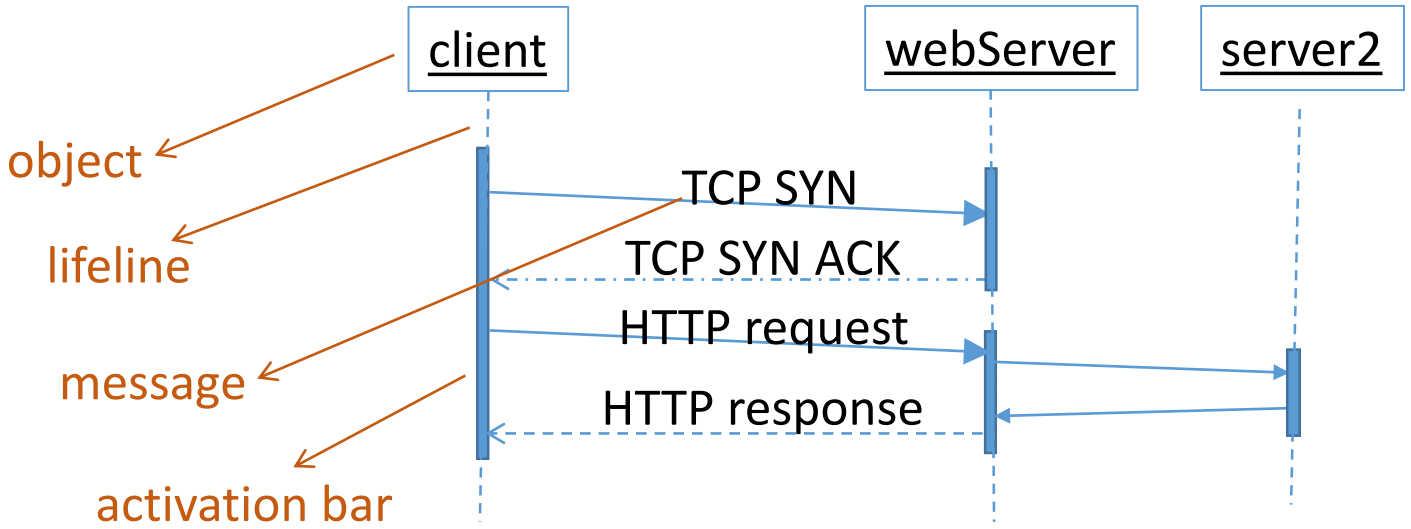
- An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose.

- A message is represented by an arrow.
  - As shown in the example, a call message uses a solid line and a response message uses a dashed line.

- A narrow rectangle, called activation bar, may be added to a lifeline to show when the object is active.

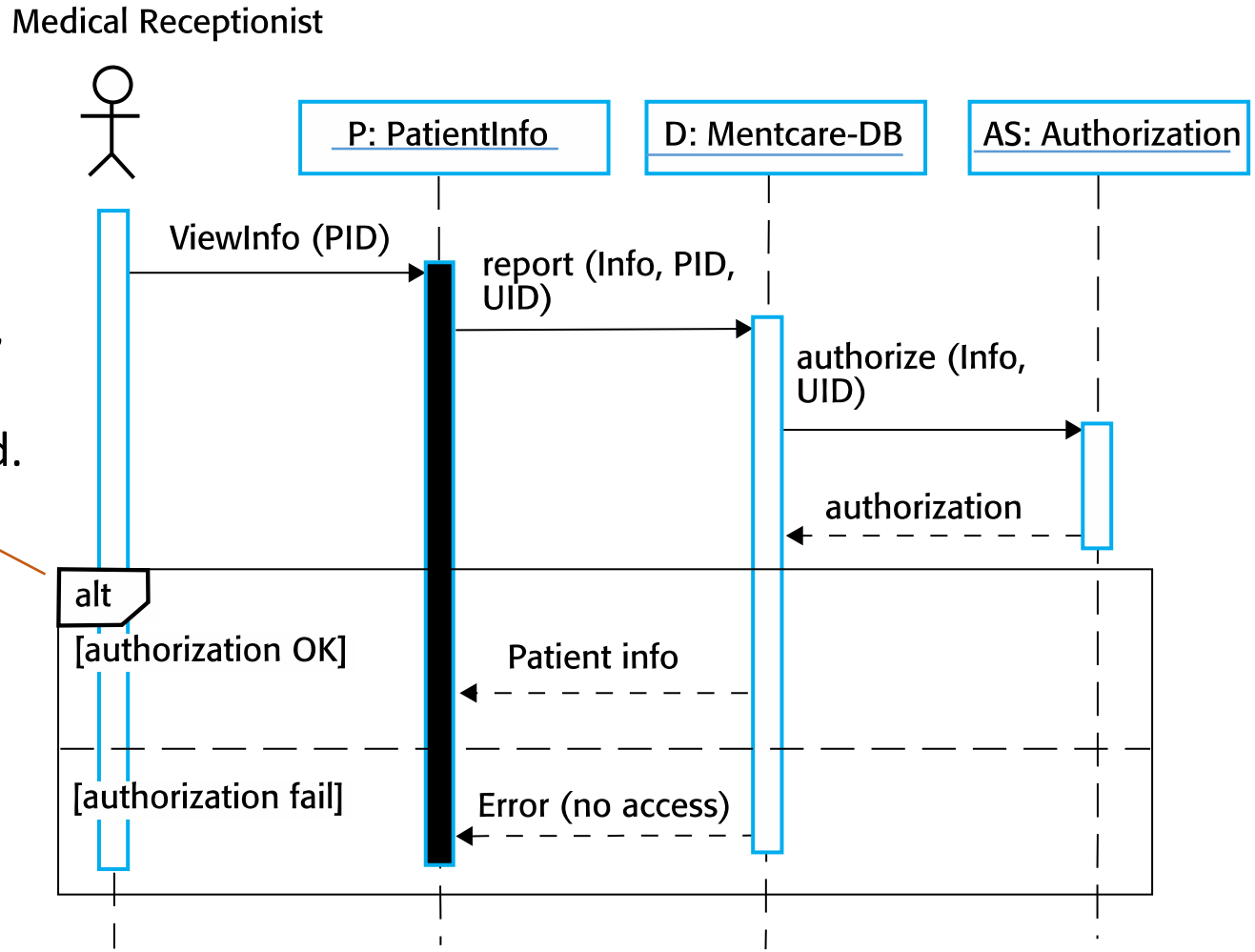
Sequence Diagram Example

- All sequence diagrams are modeled at the object level rather than the class level.
- In a sequence diagram, time runs downward (depicted with the vertical dashed line below each object, also called lifeline).
- Example: A simplified sequence diagram of web browsing



Example 2

- Sequence diagram: patient information



"alt" denotes an alternative, so one of the contained interactions will be executed.

Example 2 (cont.)

1. Medical receptionist triggers the ViewInfo method in an instance P of the PatientInfo object class, supplying PID (patient's ID).

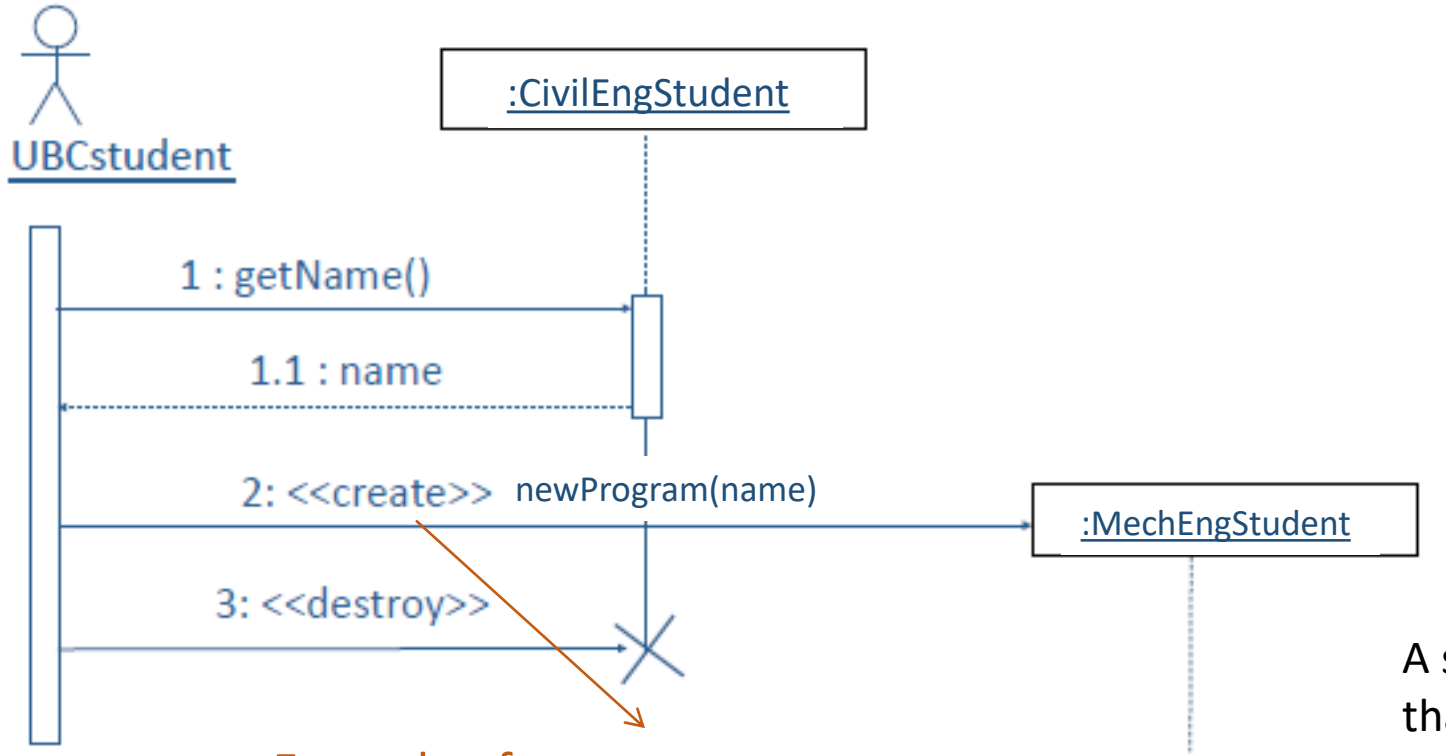
2. The P instance calls the database to return the info required, supplying info, PID and UID (user ID).

3. The database checks with an authorization system (AS) that UID is authorized for this action (i.e. to retrieve the info).

4. If authorized, the patient info is returned, otherwise, an error message is returned.

Example 3

- Sequence diagram: change in UBC program
  - We can add sequence numbers to messages

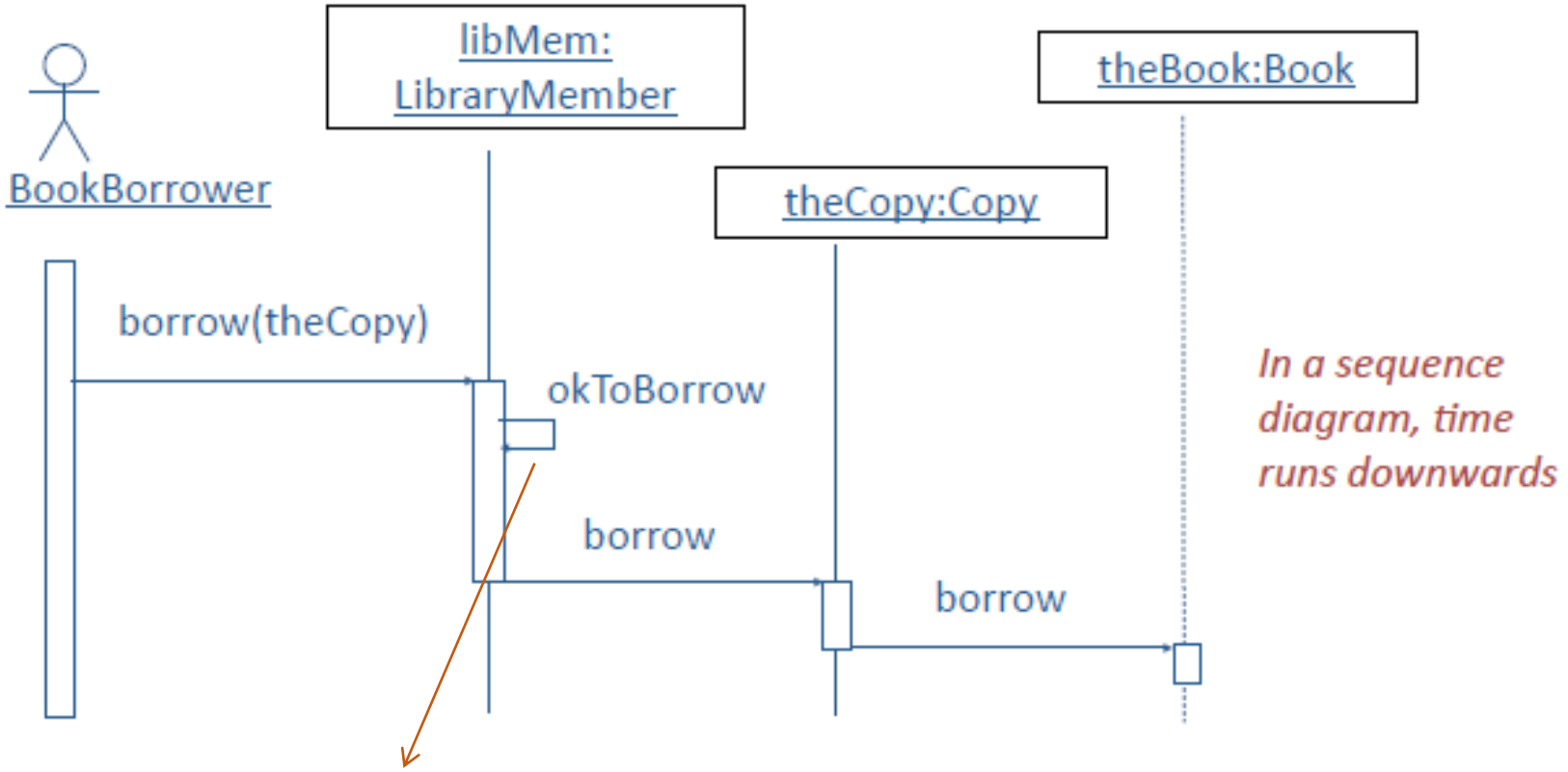


Example of a create stereotype

A stereotype is a model element that identifies the purpose of other model elements

Example 4

- Sequence diagram: borrow a copy of a book



Example of a self-reference

Some references

- UML OMG: https://www.omg.org/spec/UML/About-UML/
  - http://www.omg.org/spec/UML
- Class diagram: https://en.wikipedia.org/wiki/Class_diagram

- UML association: https://www.uml-diagrams.org/association.html

- *Introduction to Programming Using Python*, Y. D. Liang
- *Software Engineering*, I. Sommerville
- *UML Weekend Crash Course*, T.A. Pender, Wiley 2002