UML Class Diagrams

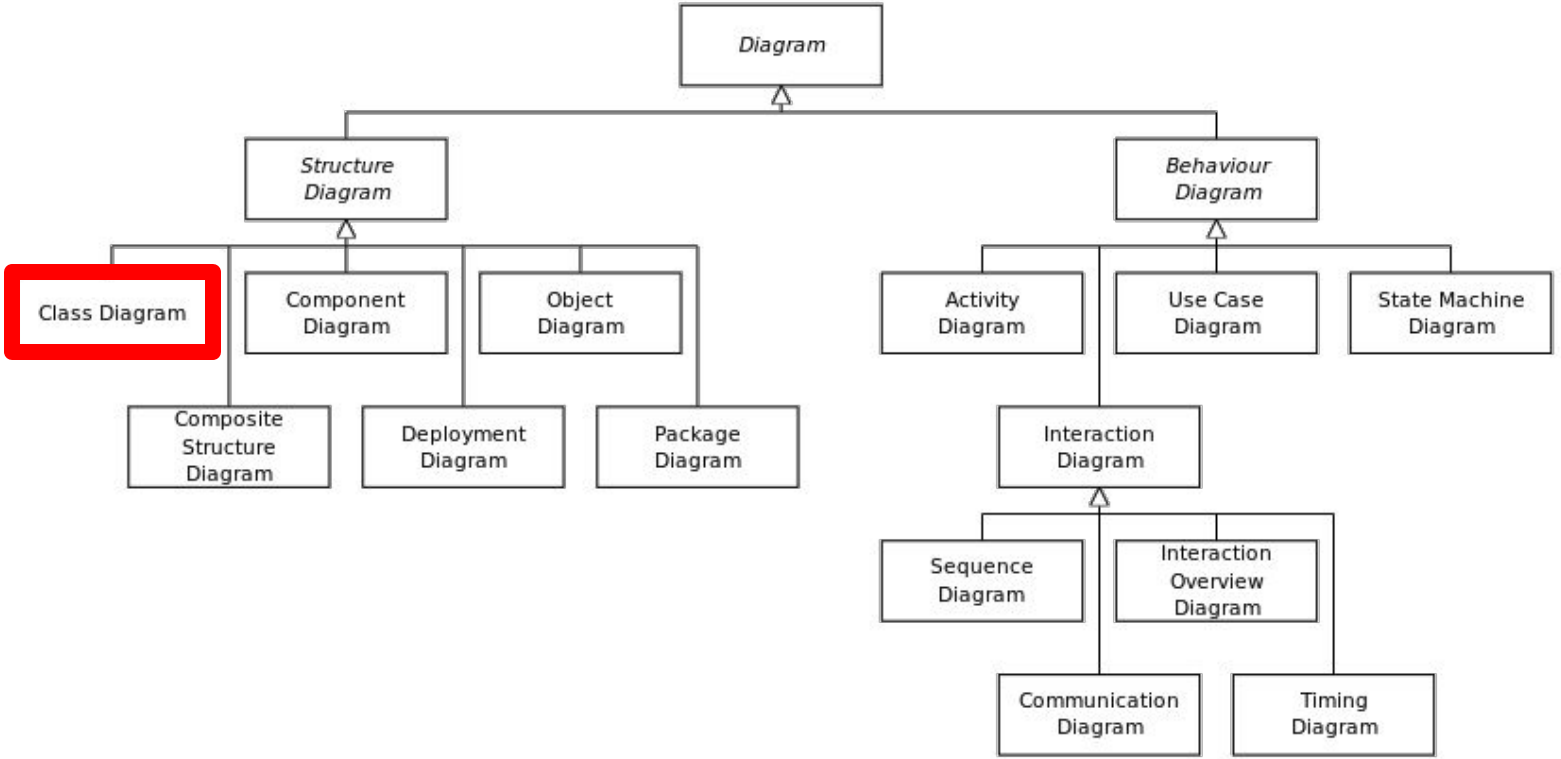
A.Y. 2018/2019

Premise

“As always, there is never a <<correct>> solution to any modelling problem. It’s more that some models are more precise, and more informative, than others. You may find your solutions differ slightly from the ones below, but these solutions demonstrate good practice in model building.”

prof. Paul Krause, University of Surrey

Class diagrams



Analysis Vs. Design

Analysis is about understanding a model by identifying the key concepts the describe a problem in a certain domain.

Design is about adapting the model resulting from the analysis in a way it can be implemented.

In other words:

- The analysis is closer to the problem
- The design is closer to the solution

Analysis: how to

- Extracting a set of *classes* from the problem specification
- Understanding the role of those classes: what kind of attributes and operations they provide?
- Recording a map of classes with their relations
- Modelling the dynamics of classes (i.e. behavioural UML models)
- Performing incremental refinements

Design: how to

- The model resulting from the analysis is refined
- UML diagrams are transformed in order to introduce more details
- Specific constraints are taken into account such as the target platform, the programming language, and non-functional requirements

Analysis classes

An analysis class models a concept or an entity of the problem

- If use cases are well specified it is easier to get analysis classes

A method is Class-Responsibility-Collaboration

Class-Responsibility-Collaboration

Class-Responsibility-Collaboration (CRC) cards are a brainstorming tool used in the design of object-oriented software.

They were originally proposed by Ward Cunningham and Kent Beck as a teaching tool, but are also popular among expert designers and recommended by extreme programming supporters.

Members of a brainstorming session will write up one CRC card for each relevant class/object of their design.

CRC card components

A CRC card is partitioned into three areas:

- On top of the card, the class name
- On the left, the responsibilities of the class
- On the right, collaborators (other classes) with which this class interacts to fulfill its responsibilities

Example of CRC card

Class name:	Superclass:	Subclasses:
Responsibilities	**Collaborations**	

Example of CRC card (contd.)

Class name: Real estate owner	**Superclass:** Person	**Subclasses:** Apartment owner
Responsibilities	**Collaborations**	
Invites	Invitation, Person, List, ...	
Buys	Money, Shop, Food, Furniture, ...	
Cleans	Duster, Sponge, ...	

Exercise: a brewery

Compile a CRC card for representing the following domain:

- A brewery is frequented by customers and staff members. Staff members collect orders. Payments are performed at the cash desk and members collect are enabled to perform cash operations. The brewery manager is responsible, besides normal service, to check the availability of beers in the fridge and, in case, to refill the fridge with other beers.

Which are the key concepts?

Exercise: brewery key concepts

Compile a CRC card for representing the following domain:

- A **brewery** is frequented by **customers** and **staff members**. Staff members collect **orders**. **Payments** are performed at the **cash desk** and staff members are enabled to perform cash operations. The **brewery manager** is responsible, besides normal service, to check the availability of beers in the **fridge** and, in case, to refill the fridge with other beers.

Exercise: CRC card for staff members

Class name: Staff member	**Superclass:** Person	**Subclasses:** Staff manager
Responsibilities	**Collaborations**	
Records orders	Order, Customer	
Serves beers	Beer, Customer, Fridge	
Manages payment	Customer, Cash	

Exercise: CRC card for staff manager

Class name:	**Superclass:**	**Subclasses:**
Staff manager	Staff member	

Responsibilities	**Collaborations**
Checks beers in fridge	Fridge
Adds beers in fridge	Beer, Fridge

Exercise: CRC card for customer

Class name: Customer	**Superclass:** Person	**Subclasses:**
Responsibilities	**Collaborations**	
Orders beer	Order, Staff member	
Enjoys beer	Beer, Staff member	
Pays	Staff member	

Exercise: CRC card for beer

Class name:	**Superclass:**	**Subclasses:**
Beer		
Responsibilities	**Collaborations**	
Knows its price		

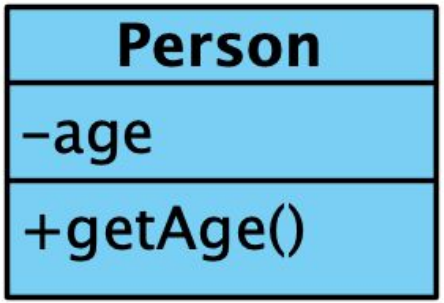
Exercise: CRC card for fridge

Class name: Fridge	**Superclass:**	**Subclasses:**
Responsibilities	**Collaborations**	
Knows the amount of beer available	Beer	
Allows to add beer		

Notation for class

A class is represented as a rectangle with internal slots for:

- Class name (UpperCamelCase) - mandatory
- Attributes (lowerCamelCase) - optional
- Operators (lowerCamelCase) - optional



Signature for attributes and operations

Attributes: visibility name:type=initial value

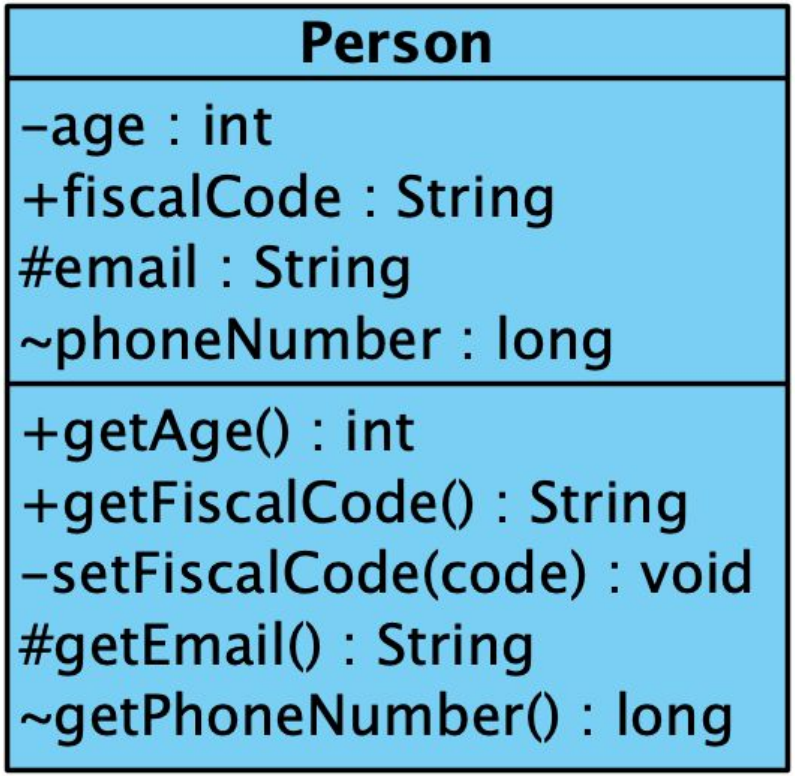
Operations: visibility name(parameter name:parameter type, ...): return type

- Only names are mandatory
- Analysis classes typical provide only the most relevant attributes and operations. In most of the cases analysis classes provide record only names for attributes and operations.
- The assignment of initial values can point out restrictions as gathered from a problem
- Design classes provide a full specification that can be directly implemented

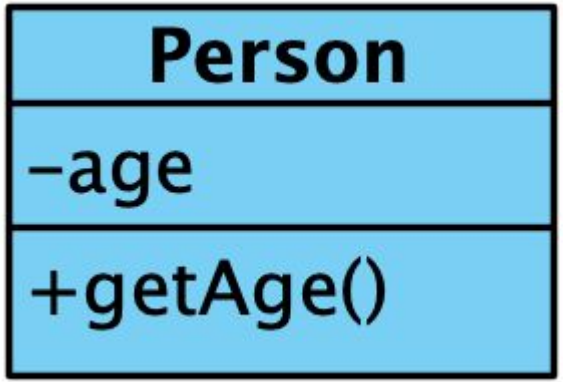
Visibility types for attributes and operations

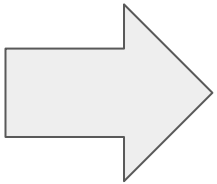
+ Public: a public element is visible to all elements that can access the contents of the namespace that owns it.
- Private: element is only visible inside the namespace that owns it.
Protected: element is visible to elements that have a generalization relationship to the namespace that owns it.
~ Package: element is only visible by elements within a package and its sub-packages.

Visibility types for attributes and operations (contd.)



From UML to Python





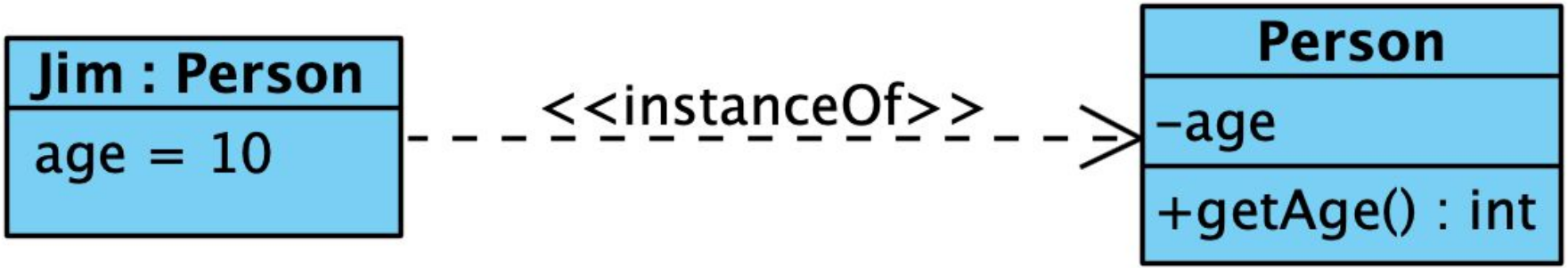
```
class Person():
    def __init__(self,age):
        self.age = age

    def getAge(self):
        return self.age
```

Objects

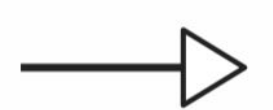
Objects are class instances and their notation is similar to that used for representing classes.

Object names are recorded as: InstanceName : ClassName

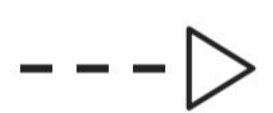
Objects have no slot for operations



Interactions



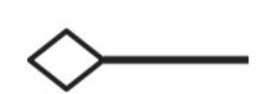
Generalization



Realization

Association

Dependency

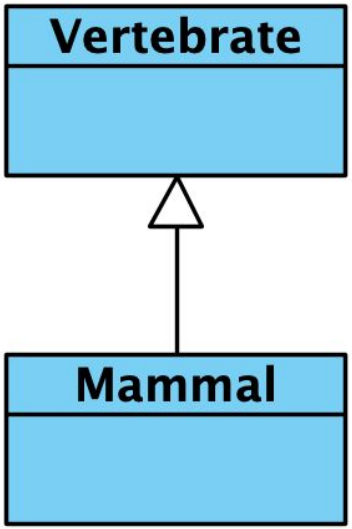


Aggregation

Composition

Generalization

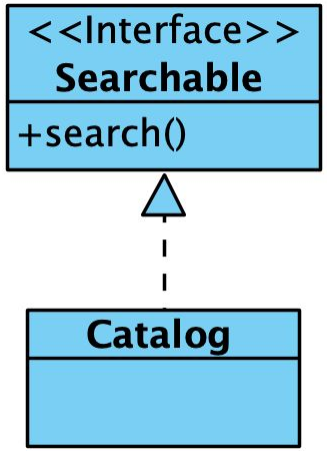
The process of a child or subclass taking on the functionality of a parent or superclass, also known as inheritance. It's symbolized with a straight connected line with a closed arrowhead pointing towards the superclass.



Realization

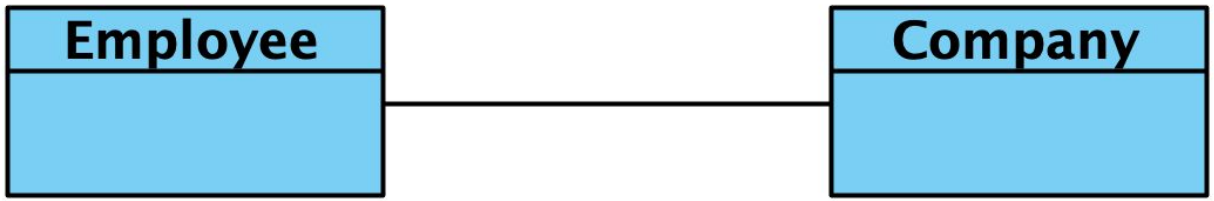
A semantic relation used for representing a provider that exposes an interface and a client that realizes the interface.

The canonical example is the relation between an interface and a class implementing such an interface.



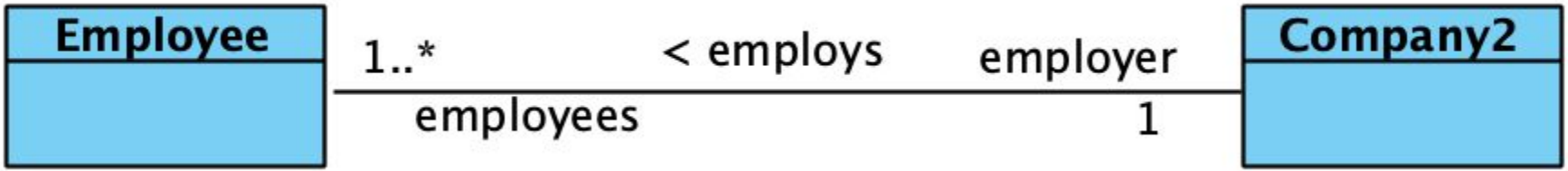
Association

The relationship between two classes. Both classes are aware of each other and their relationship with the other. This association is represented by a straight line between two classes.



Association: ornaments

- Name
- Directional triangle: specifies the direction used for reading the relation
- Roles
- Multiplicity

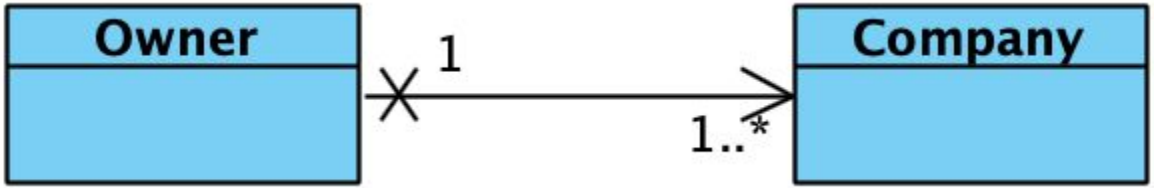


Unidirectional associations

Useful for specifying which elements of an association knows about the association.

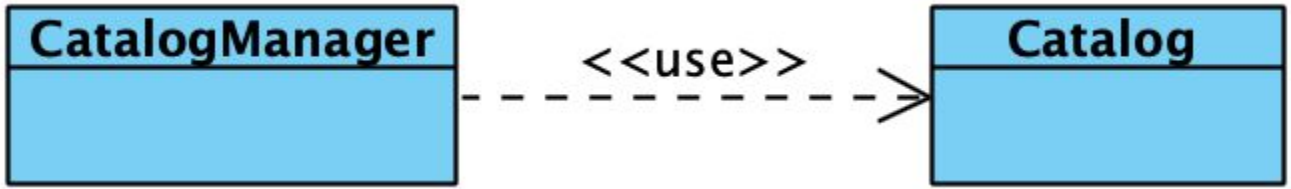
- The arrows specifies the navigability
- The square specifies the lack of navigability

An Owner knows what are his companies, but company does not know who is its owner.



Dependency

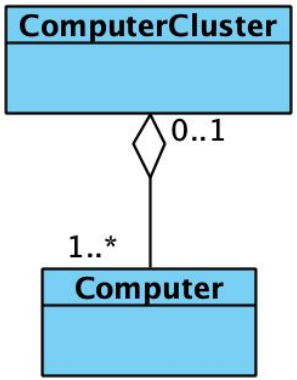
Dependency is a directed relationship which is used to show that some element or a set of elements requires, needs or depends on other model elements for specification or implementation. Because of this, dependency is called a supplier - client relationship, where supplier provides something to the client, and thus the client is in some sense incomplete while semantically or structurally dependent on the supplier element(s). Modification of the supplier may impact the client elements.



Aggregation

A binary association between a property and one or more composite objects which group together a set of instances. Aggregation has the following characteristics:

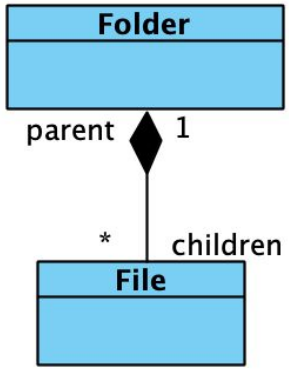
- it is binary association
- it is asymmetric - only one end of association can be an aggregation
- it is transitive - aggregation links should form a directed, acyclic graph, so that no composite instance could be indirect part of itself



Composition

Composite aggregation (composition) is a "strong" form of aggregation with the following characteristics:

- it is binary association
- it is a whole/part relationship
- a part could be included in at most one composite (whole) at a time
- if a composite (whole) is deleted, all of its composite parts are "normally" deleted with it.



Example: Library Domain Model

Library Domain Model describes main classes and relationships which could be used during analysis phase to better understand domain area for Integrated Library System (ILS), also known as a Library Management System (LMS).

Each physical library item - book, tape cassette, CD, DVD, etc. could have its own item number. To support it, the items may be barcoded. The purpose of barcoding is to provide a unique and scannable identifier that links the barcoded physical item to the electronic record in the catalog. Barcode must be physically attached to the item, and barcode number is entered into the corresponding field in the electronic item record.

Barcodes on library items could be replaced by RFID tags. The RFID tag can contain item's identifier, title, material type, etc. It is read by an RFID reader, without the need to open a book cover or CD/DVD case to scan it with barcode reader.

Example: Library Domain Model - class diagram