Modeling

Programmeertechnieken, Tim Cocx

Universiteit
Leiden
The Netherlands

Software development Lifecycle

- (Requirements) analysis:
  - Requirements gathering and description
  - Construction of analysis model(s)
  - ["**What** is supposed to be done"]
- Design
  - Construction of design model(s)
  - ["**How** are we going to approach it"]
- Coding:
  - Programming/ (unit) testing
- Deploy:
  - Start using the software
- Support
  - Usage and maintenance

Analysis

- Requirements gathering (elicitation)
- Create analysis model
  - Denotes **What the system does, not how**.
  - Is (more or less) understandable by the domain
  - Domain: customers/ users/ usage environment
  - Shows the domain's needs
  - Models reality / the domain
  - In this course: class diagram
  - Target group: the domain and developers

Design

- Determine the way the software is going to be built
  - First major structural decisions, later; details
- Create design model
  - Shows **how the system is built**
  - For yourself (thinking aid) and co-workers (discussion aid)
  - For later (when changes come): documentation
  - In this course: class diagram
  - Target group: developers

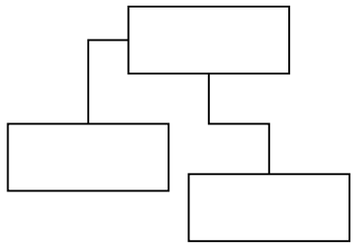
Modeling steps

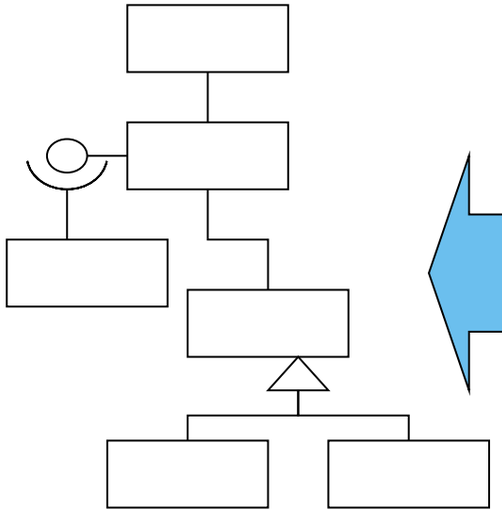
Problem domain

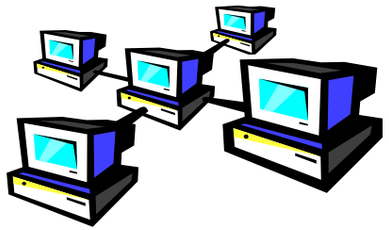
Analysis model

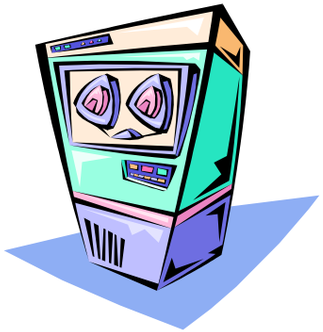
Design model

Solution domain



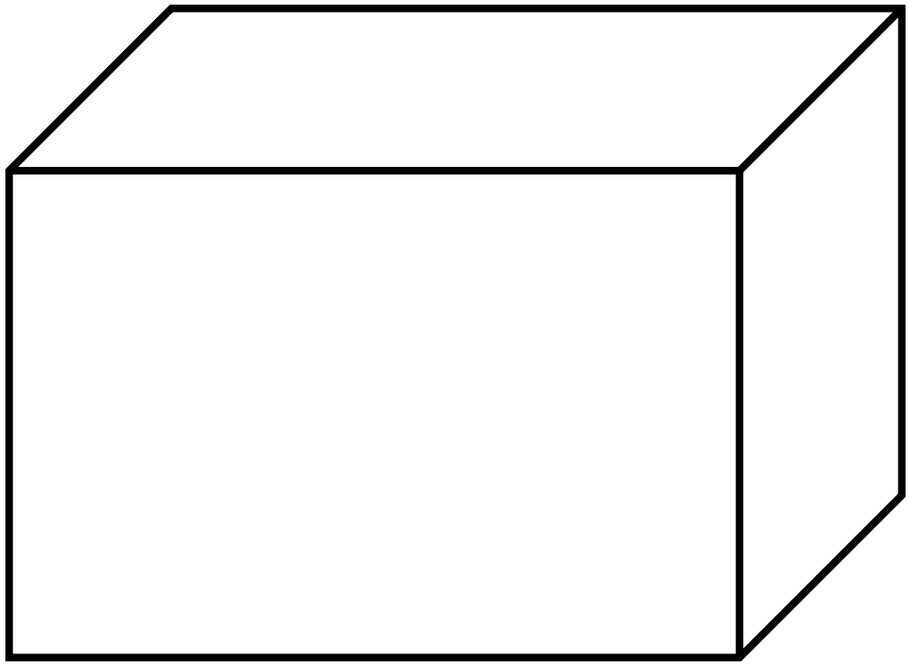






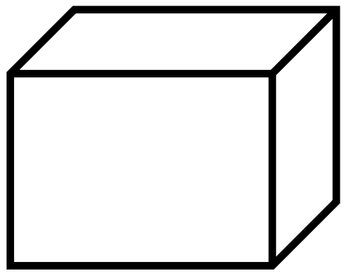
Modeling

What is this?



Modeling

- Create an image of a piece of reality
  - With a certain purpose
  - According to a pre-determined technique
  - Depending on the purpose, details can be omitted



Example model

- Model of a railroad:

Example model

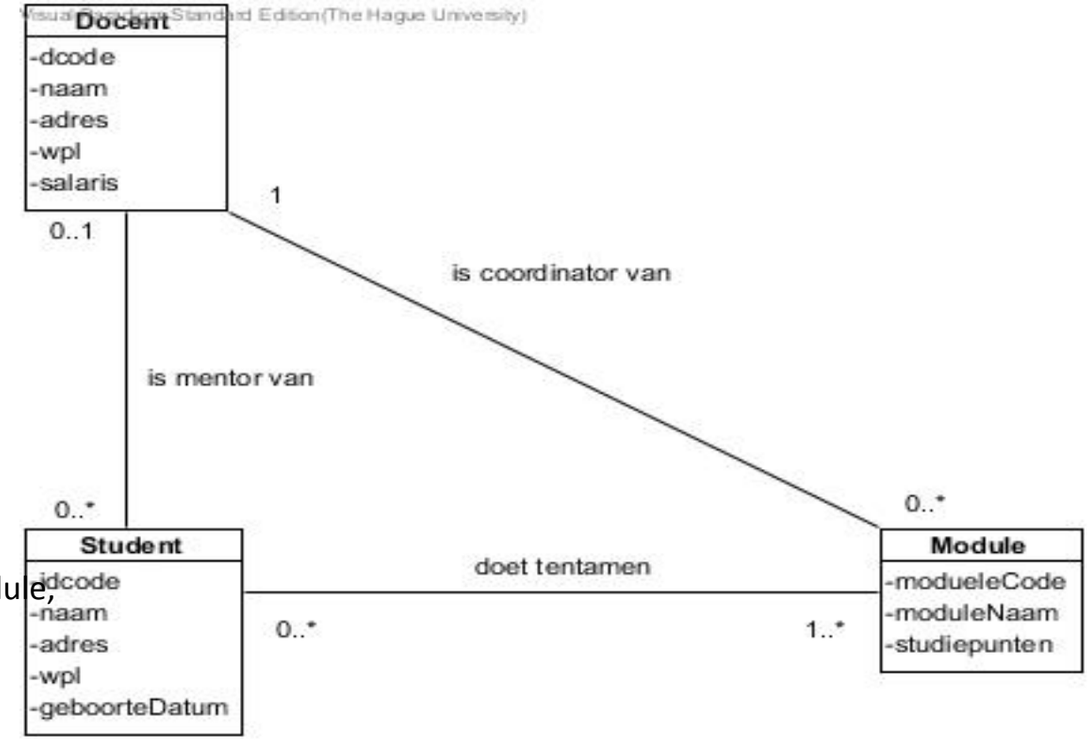
- Conceptual model of a database

student (idcode, naam, adres, wpl, geboortedat, *d_code*)
d_code is vreemde sleutel, verwijst naar d_code in docent,
null is toegestaan.

docent (dcode, naam, adres, wpl, salaris)

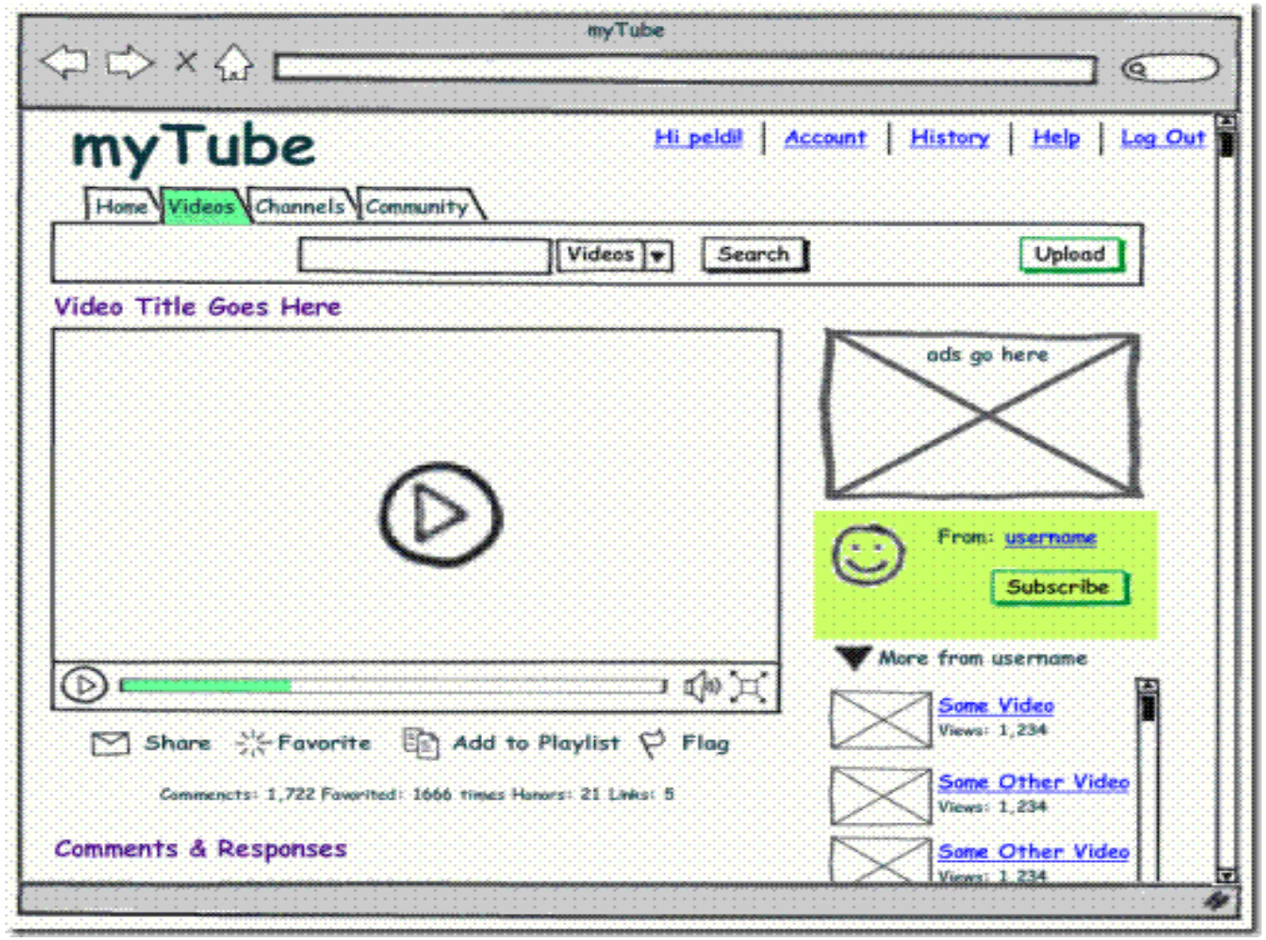
module (moduleCode, moduleNaam, studiepunten, *coordinator*)
coordinator is vreemde sleutel, verwijst naar d_code in docent,
null is NIET toegestaan.

tentamen (*idcode*, *modulecode*)
idcode is vreemde sleutel, verwijst naar idcode in student,
null niet toegestaan
modulecode is vreemde sleutel, verwijst naar modulecode in module;
null niet toegestaan.



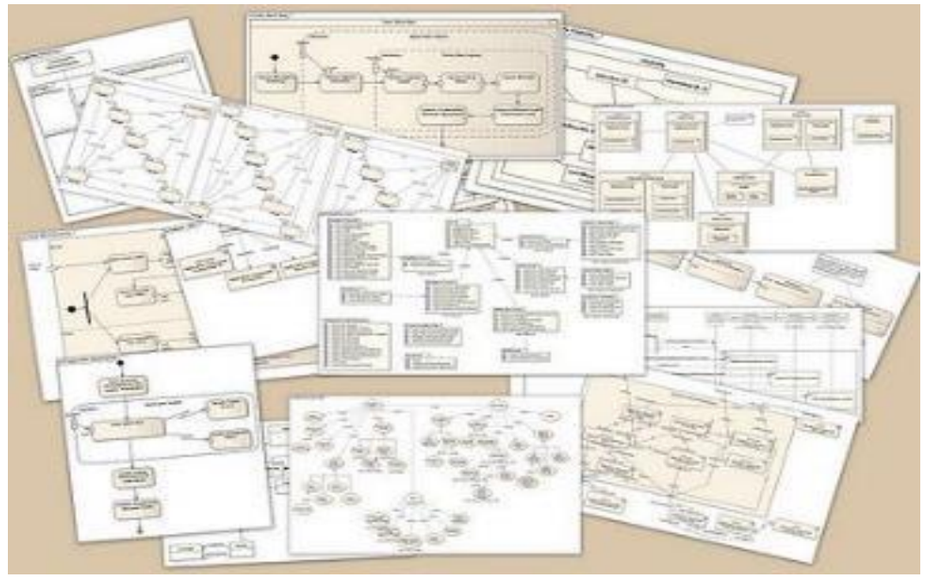
Example model

- Graphical design model of a website (‘wireframe’, ‘mock-up’)



Unified Modeling Language

- De *Unified Modeling Language* (UML) is the ‘de facto’ standard to model software.
  - Class diagram
  - Use Case diagram
  - Sequence diagram
  - State Transition diagram
  - Activity diagram
  - Etc.
- UML is a ‘drawing-language’ showing how to create these diagrams



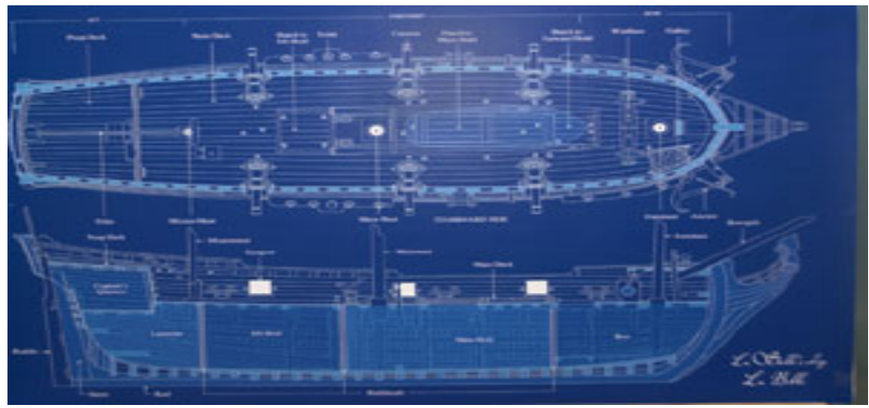
Class diagram

- A class diagram is a blueprint

Class diagram

As systemmodel

- Describes the **interaction** between **objects**



Case 1: Mario

Wat are the objects?

Case 1: Mario

Case 1: Mario

- Mario
- Luigi
- Toad1
- Toad2
- Yoshi Gr
- Yoshi Ro
- Hammer n 1
- Hammerman 2
- Hammer 1
- Hammer 2
- Hammer 3
- Coin 1
- Coin 2
- Coin 3

- Mushroom
- Mystery block
- Row of blocks 1
  - Block1
  - Etc.
- Ro f blocks 2
  - B l1
  - Et
- Floor
- Roof
- Platform 1
- Platform 2
- Platform 3
- Platform 4

Bad way

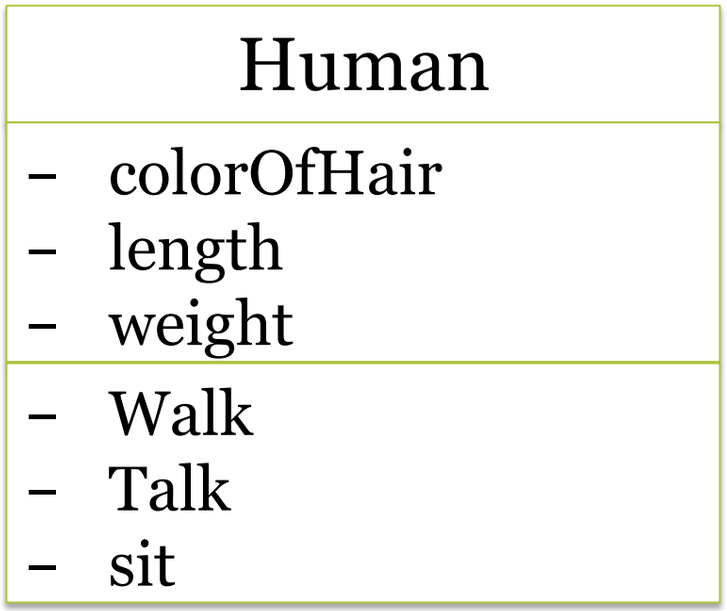
Case 1: Mario

- Making a list with objects becomes a little bit unruly quite
  - Imagine you have 200 coins in one level!
  - What if we also want to describe the properties (e.g.: location) of every object?
- A lot of object are (approximately) the same!
  - Lets combine those objects under one ‘blueprint’!
  - This is a class.

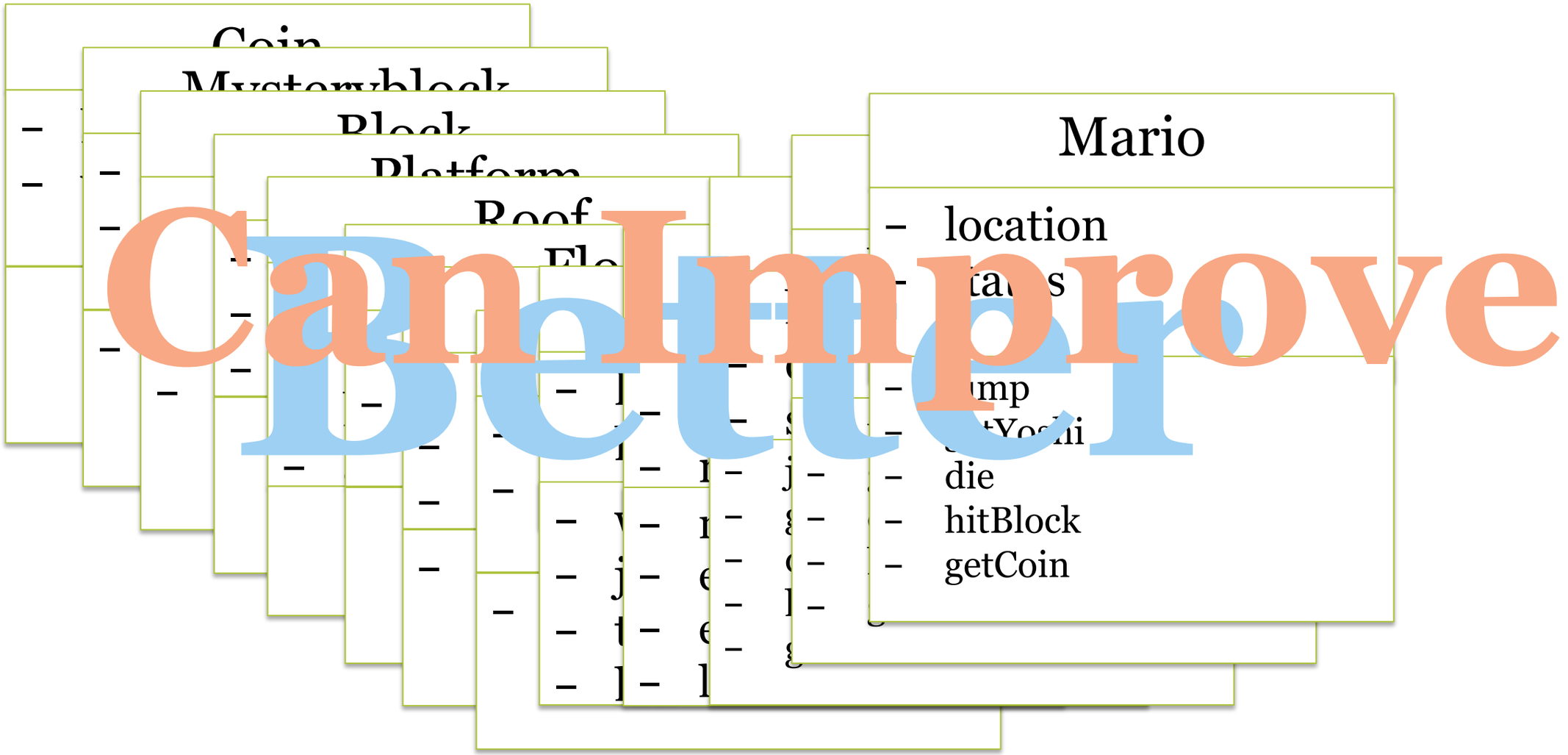
Class

- A *class* describes a blueprint for a collection of individual *objects*
- Example: The class ‘Human’ describes all of us.
  - We are ‘objects of the class Human’ (notice the capital letter ‘H’!)
- A class describes:
  - Attributes: properties
  - Methods: skills
- Methods for Human:
  - Walk, talk, sit
  - Methods are exactly the same for every object of the class
- Attributes Human:
  - Color of hair, length, weight
  - Attributes (can) differ per object

Class UML syntax



Case 1: Mario



Inheritance

- A child:
  - Has a length, weight, color of hair and favorite toy
- A

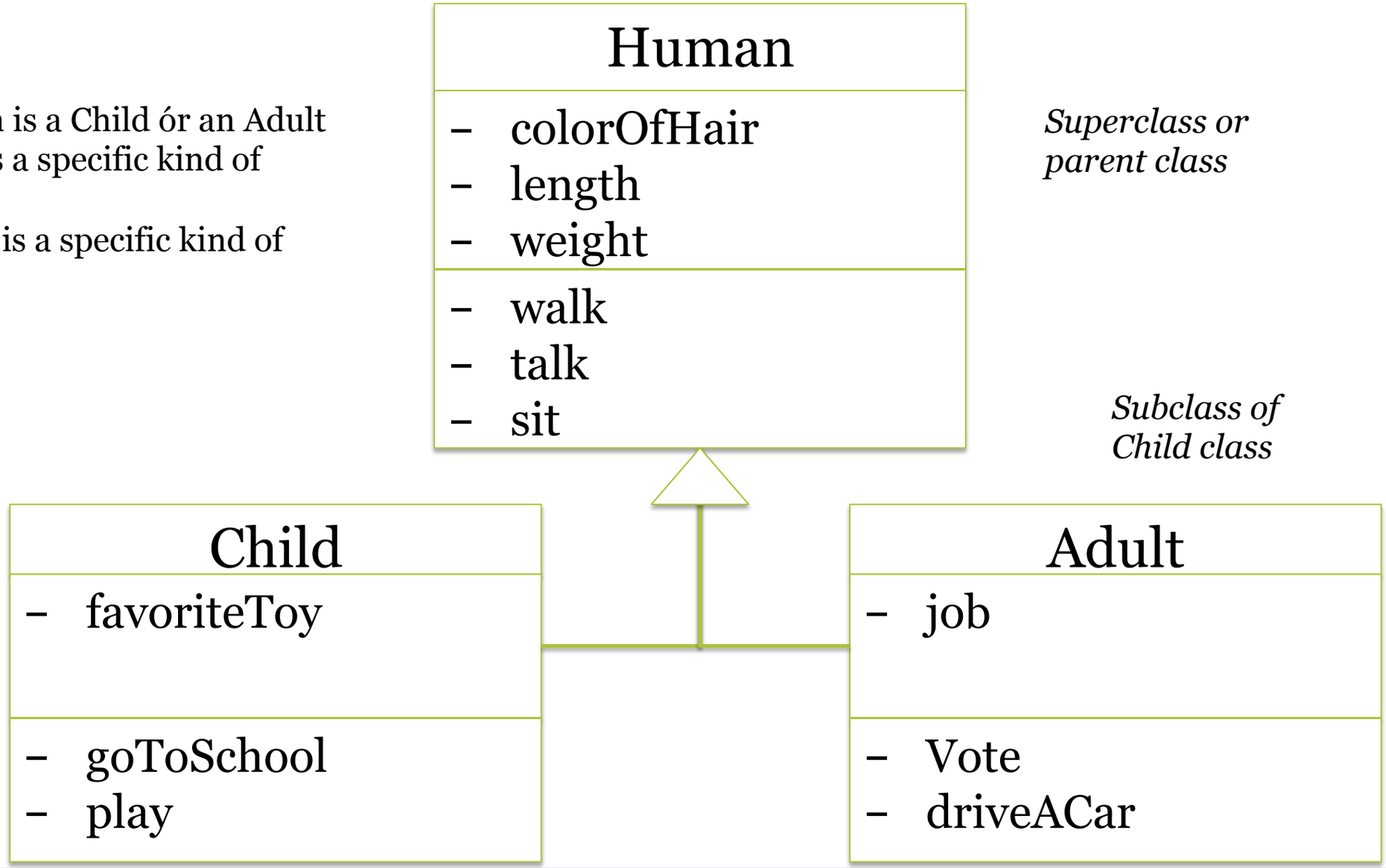
Child
– colorOfHair – length – weight – favoriteToy
– walk – talk – sit – play – goToSchool

Adult
– colorOfHair – length – weight – job
– walk – Talk – sit – vote – driveACar

What can be done better?

Inheritance UML syntax

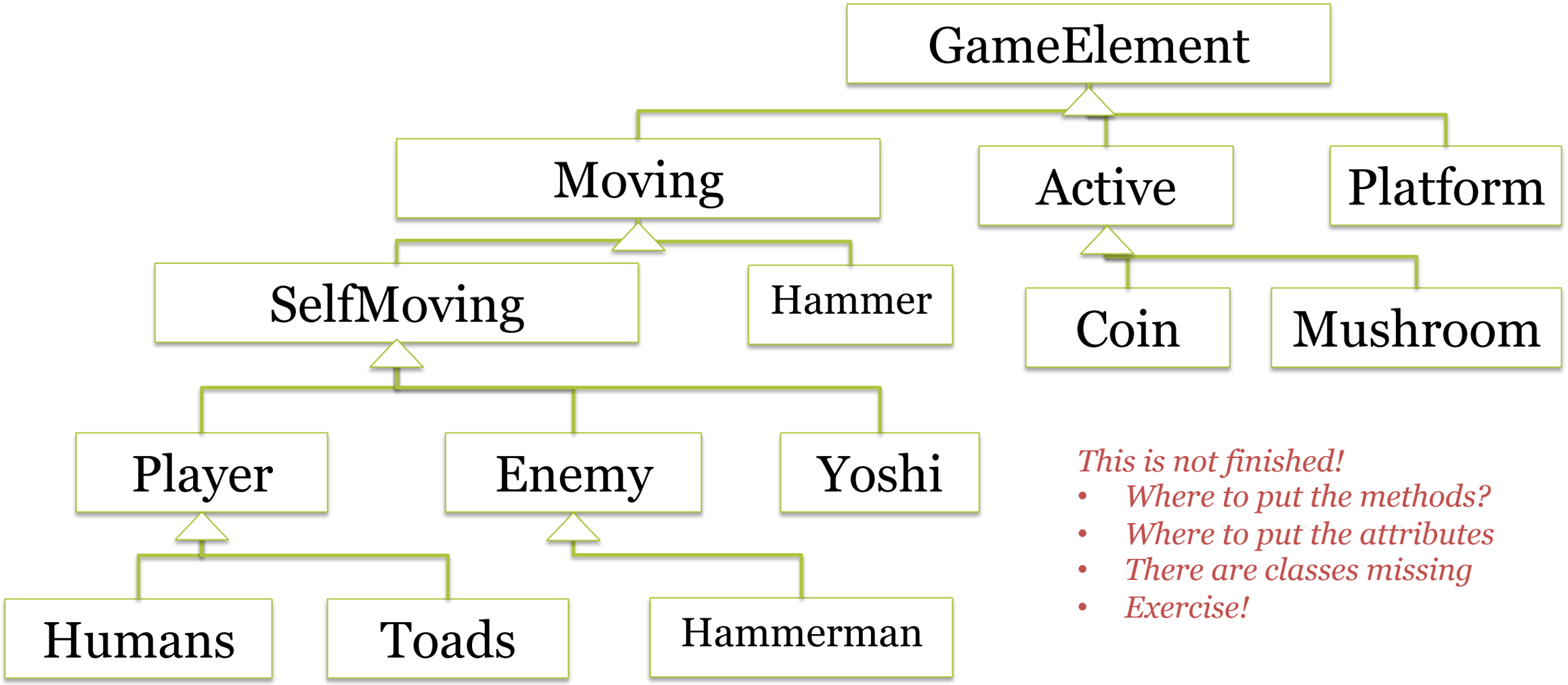
- A Human is a Child ór an Adult
- A Child is a specific kind of Human
- An Adult is a specific kind of Human



Inheritance theory

- inheritance: also *Generalization*
- A subclass inherits all attributes of its super class(es).
  - Can be more (*grandfather class*)
- A subclass inherits all methods of its super class(es)
- ‘Downwards’: specialization
- ‘Upwards’: generalization

Case 1: Mario



This is not finished!

- *Where to put the methods?*
- *Where to put the attributes*
- *There are classes missing*
- *Exercise!*

Class Diagram: procedure

- Read the analysis report
- Nouns
  - Class
  - Attribute
  - Don’t model
- Verbs
  - Methods
  - Something else
  - Don’t model

Case 2: College

The following data needs to be entered for new students: name, student code, date of birth and study coach (at the time of registration every student gets assigned a teacher as his or her coach). Students have a list of grades, study and do exams. After every period, the grades for the courses and the date of the exam must be entered. The system calculates the average result of the grades. Teachers have a name, date of birth and building. They grade exams and assess students

Case 2: College

The following **data** needs to be *entered* for new **students**: **name**, **student code**, **date of birth** and **study coach** (at the **time** of **registration** every **student** *gets assigned* a **teacher** as his or her **coach**). **Students** *have* a **list of grades**, *study* and *do* ***exams***. After every **period**, the **grades** for the **courses** and the **date of the exam** must be *entered*. The **system** *calculates* the **average result** of the **grades**. **Teachers** *have* a **name**, **date of birth** and **building**. They *grade* ***exams*** and *assess* ***students***

Case 2: College

What is what?

noun:

- students
- code
- data
- name
- date of birth
- coach
- list of grades
- teacher
- period
- Grade
- course
- date of the exam
- system
- average
- building

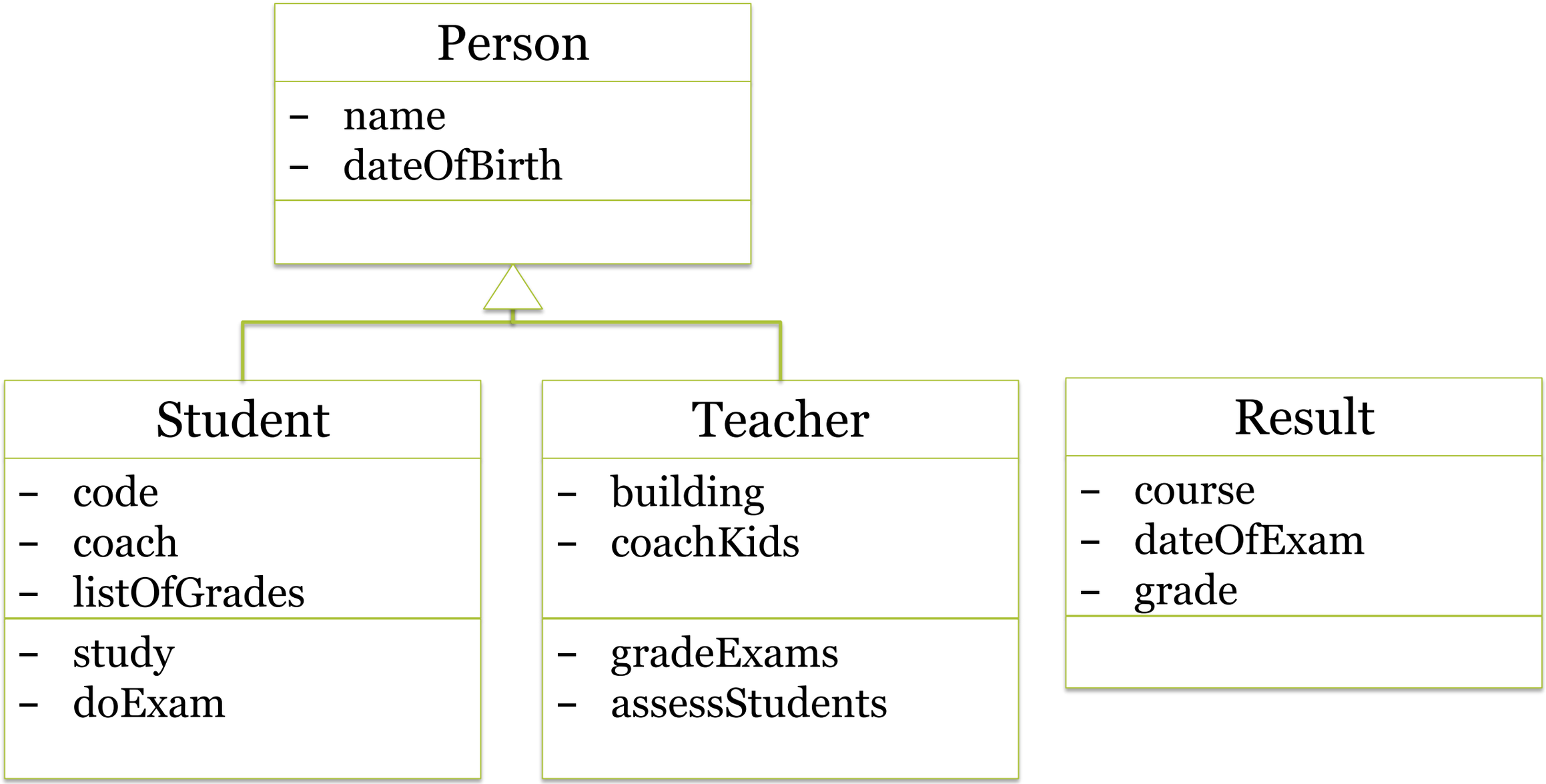
ve

- entered
- gets assigned
- have
- do exames
- study
- enter
- calculate
- grade exam
- Assess students

Case 2: College

- Student
  - name
  - date of birth
  - code
  - coach
  - list of grades
  - *do exam*
  - *study*
- Teacher
  - coach-kids
  - name
  - date of birth
  - building
  - *assess students*
  - *grade exam*
- Result
  - grade
  - Date of exam
  - course

Case 2: College



Case 1: Mario

Interaction??

Case 1: Mario

Which objects interact?

Case 1: Mario

- Mysteryblock *contains* Mushroom
- Player *rides* Yoshi (←parent class!)
- Hammerman *throws* Hammer
- Hammer '*kills*' Player
- Player *gets* Coin
- Etc.

- How do we model that?

Two Choices

- A short term relation (one night stand):
  - One of the classes *depends* on the existence of the other class to do its job

Dependency

- A long term relation
  - The classes are involved with each other for a longer time, remember each others existence and are therefore *associated*

Association

Dependency UML Syntax

- A dependency is denoted by a dotted arrow
- A dependency always has a *stereotype* attached: what kind of dependency it is.
  - Syntax << *stereotype*>>

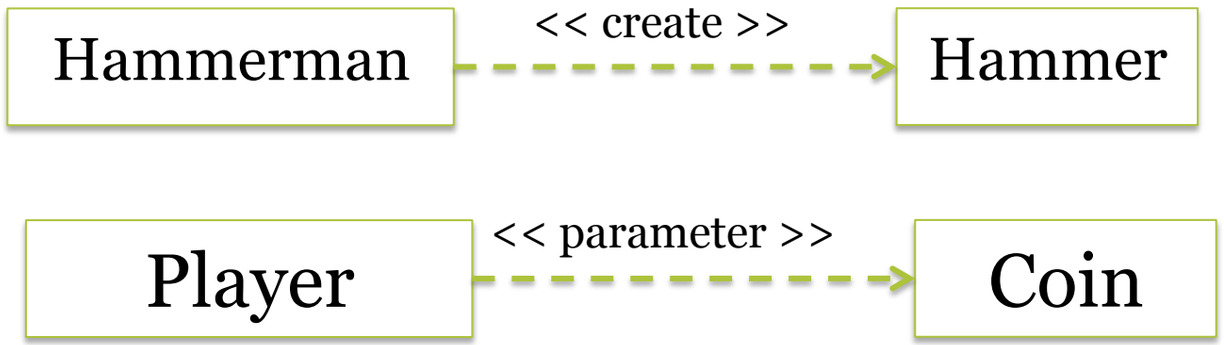
- If there is no text with it a << use >> dependency is assumed

<< parameter >>

Dependencies: usage stereotypes

- << use >>: the class is being used in an unspecified manner
  - Rather not, unless it's used as local variable
- << parameter >>: the class is a parameter in one of the other classes methods
- << create >>: the class is created by the other class

Dependencies: Examples

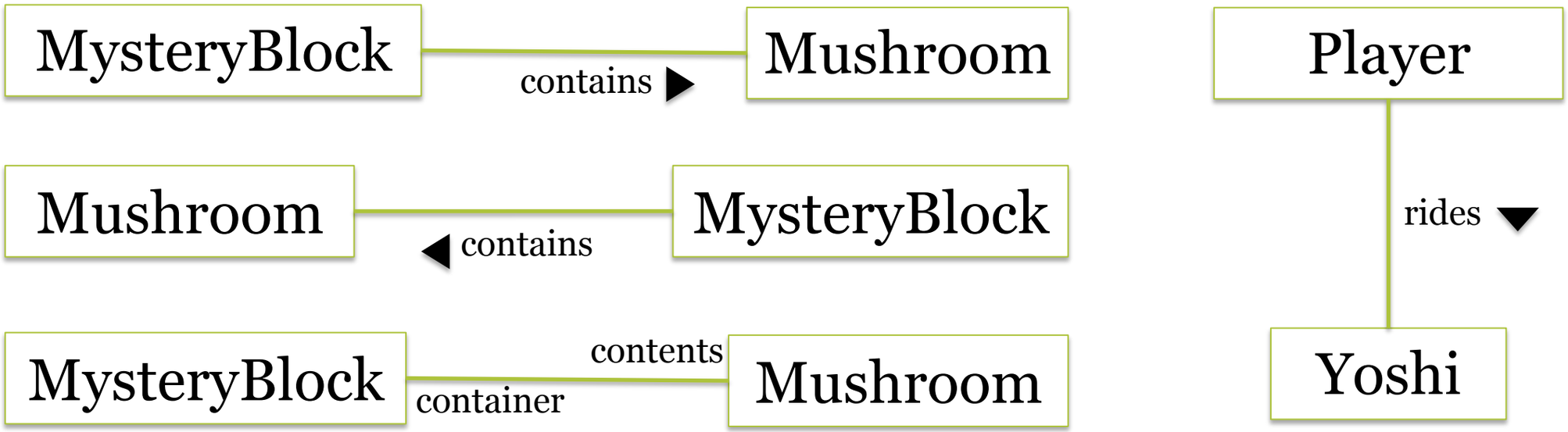


Association UML syntax

- When two classes ‘know’ one another they are *associated*
- Associations are (naturally) inherited by sub classes
- Notation is a line.

Association UML syntax

- An association always has a description
  - This is a name with reading direction, ór
  - A division of roles on both sides



Case 1: Mario

Can Mario ride multiple Yoshis (at the same time)?

Can a mystery block contain multiple mushrooms?

Can a hammerman throw multiple hammers?

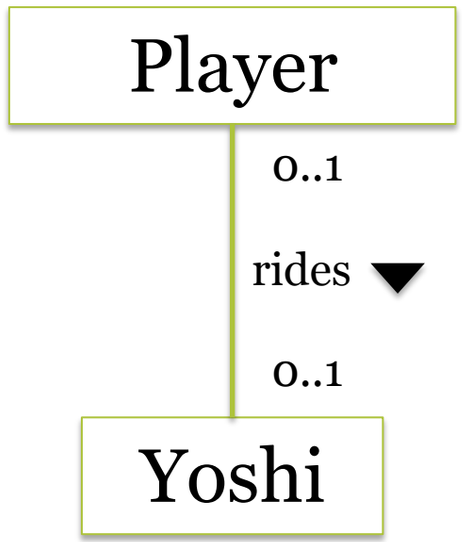
Multiplicity UML syntax

- *Multiplicity* shows the quantities in an association:
  - 1 → exactly 1
  - 99 → exactly 99
  - 5..55 → a value between 5 and 55
  - * → multiple(=potentially infinite, but also 0)
  - 4..* → 4 or more
- A dependency has no multiplicity (nothing is remembered)!
- Multiplicity is given on both sides

Multiplicity UML syntax

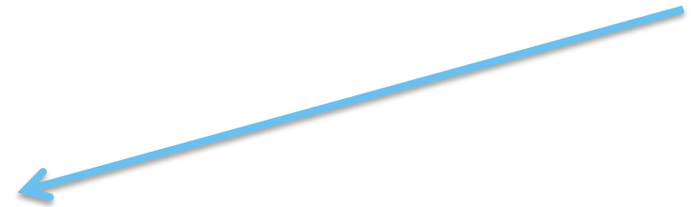
- A MysteryBlock contains 0 or more Coins
- A Coin is stored in 0 or 1 MysteryBlocks

- A player rides 0 or 1 Yoshis
- A Yoshi is ridden by 0 or 1 Players



Class diagram: procedure

- Nouns
  - Class
  - Attribute
  - Don’t model
- Verbs
  - Methods
  - Something else
  - Don’t model



Associatie !
(dependency)

Case 2: College

The following **data** needs to be *entered* for new **students**: **name**, **student code**, **date of birth** and **study coach** (at the **time** of **registration** every **student** *gets assigned* a **teacher** as his or her **coach**). **Students** *have* a **list of grades**, *study* and *do* ***exams***. After every **period**, the **grades** for the **courses** and the **date of the exam** must be *entered*. The **system** *calculates* the **average result** of the **grades**. **Teachers** *have* a **name**, **date of birth** and **building**. They *grade* ***exams*** and *assess* ***students***

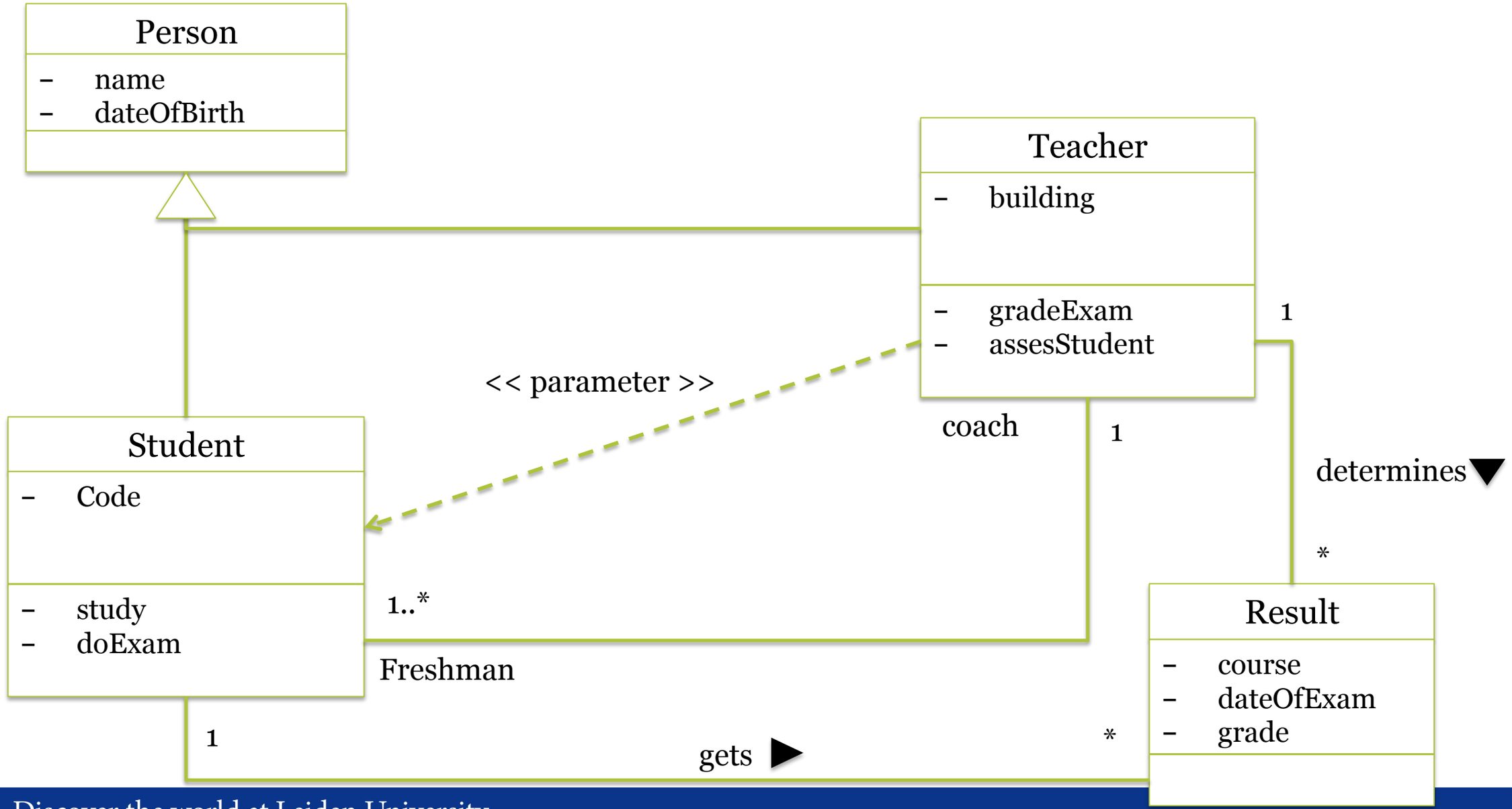
Case 2: College

- Student
  - name
  - date of birth
  - code
  - coach
  - list of grades
  - *do exam*
  - *study*
- Teacher
  - coach-kids
  - name
  - date of birth
  - building
  - *assess students*
  - *grade exam*
- Result
  - grade
  - Date of exam
  - course

Case 2: Studeren

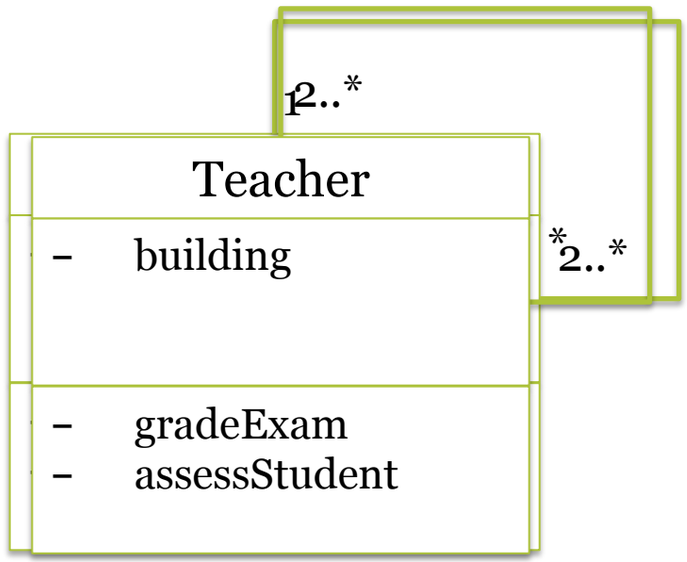
- Student
  - naam
  - Geboortedatum
  - Studentcode
  - Slb'er
  - cijferlijst
  - *Inschrijven*
  - *Tentamen maken*
  - *studeren*
- Docent
  - SLB-studenten
  - Naam
  - Geboortedatum
  - Vestiging
  - *Student beoordelen*
  - *Cijfers berekenen*
- Resultaat
  - Cijfer
  - Toetsdatum
  - Blok

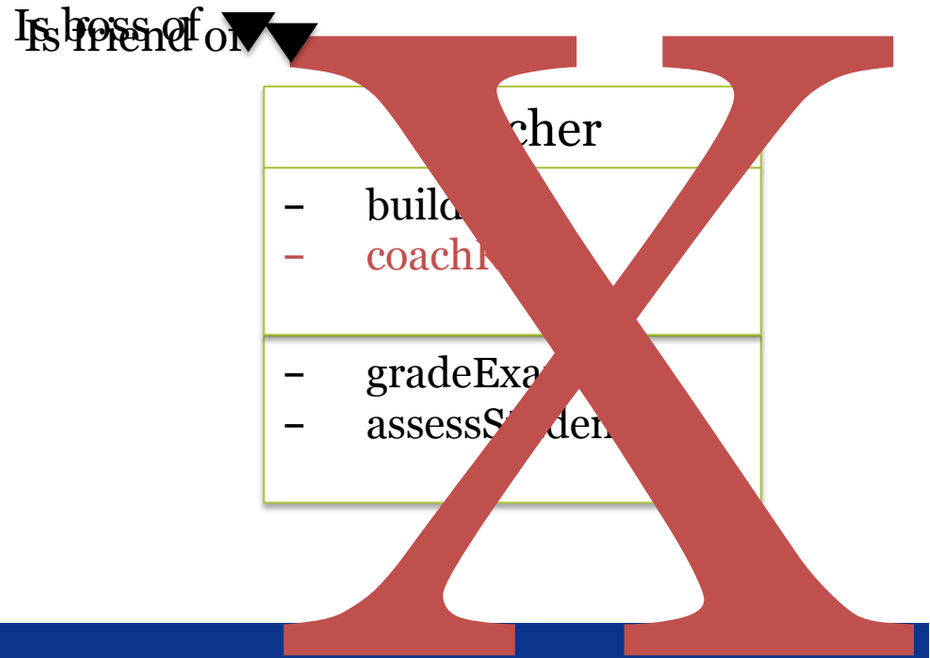
Case 2: Studeren



Associations: observations and rules

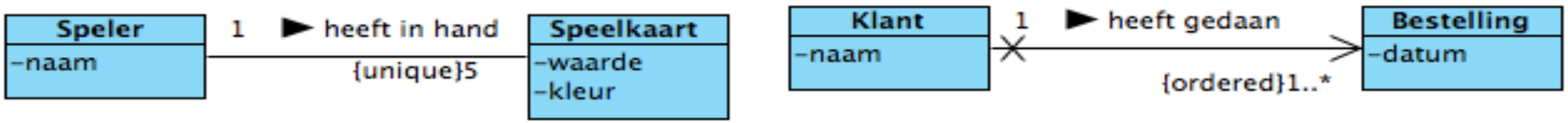
- Associations can also refer to the same class: *unary association*.
  - A student has 2 or more friends
  - A teacher manages other teachers
- Every class in the diagram is connected to at least one other class
- If a class references another class in the diagram it's always an association, not an attribute





Properties of multiplicity

- Sometimes you want to assign a property to a *-association
  - The list must be sorted (not possible to show ‘on what’)
  - Every object is represented in the association only once.
- Such a property is denoted by {property}
  - Ordered / **unordered** → sorting
  - unique / **nonunique** → unicity



Case 3: Kebab

DönerKings is a large brand of kebab-bakers. Every branch has at least 2 employees, with a certain salary and a name. They are hired to bake, fill and sell buns. Some employees manage 2 other employees. It is possible to have more than one boss. Branches sell buns and Turkish pizzas (we know the price of both). Buns have a certain content (chicken or veal) and pizzas are sold with different diameters. Both can be eaten. Sometimes branches expand, which means hiring more employees. Buns and pizzas always contain 3 ingredients,that have a certain expiration date, sometimes they spoil. Oh yes! Branches have an owner. That is one of the employees.

Case 3: Kebab

DönerKings *is* a large **brand** of **kebab-bakers**. Every **branch** *has* at least 2 ***employees***, with a certain **salary** and a **name**. They are *hired* to *bake*, *fill* and *sell* **buns**. Some **employees** *manage* 2 other **employees**. It *is* possible to *have* more than one **boss**. **Branches** *sell* **buns** and **Turkish pizzas** (we *know* the **price** of **both**). **Buns** *have* a certain **content** (**chicken** or **veal**) and **pizzas** are *sold* with different **diameters**. **Both** can *be eaten*. Sometimes **branches** *expand*, which means *hiring* more **employees**. **Buns** and **pizzas** always *contain* 3 **ingredients**, that *have* a certain **expiration date**, sometimes they *spoil*. Oh yes! **Branches** *have* an **address** and **owner**. That *is* one of the **employees**.

Case 3: Kebab

- DönerKings
- brand
- Kebab-bakers
- Branch
- Employees
- Salary
- Name
- buns
- Boss
- Turkish Pizzas
- Price
- Content
- Chicken
- Veal
- Pizzas
- Diameter
- Both
- Ingredients
- Expiration date
- Address
- Owner

First get rid of synonyms and superfluous words

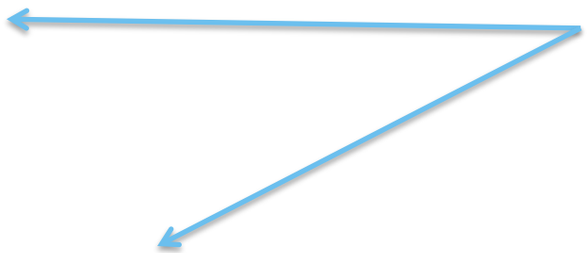
Case 3: Kebab

- ~~DönerKings~~ — Name of customer
- ~~brand~~ — Explanation of customer
- ~~Kebab-bakers~~ — synonym
- Branch
- Employees
- Salary
- Name
- buns
- ~~Boss~~ — Part of verb
- ~~Turkish Pizzas~~ — synonym
- Price
- Content
- Chicken — Possible value of attribute
- ~~Veal~~ — Possible value of attribute
- ~~Pizzas~~
- Diameter
- ~~Both~~ — Language construct(hint!)
- Ingredients
- Expiration date
- Address
- Owner

Now: make singular and sort

Case 3: Kebab

- Branch
  - address
  - owner
- Employee
  - salary
  - name
  - subordinates
- Product
  - price
- Bun
  - content
- Turkish Pizza
  - diameter
- Ingredient
  - expirationDate



References another class (→Association)

Super class!

Case 3: Kebab

- Branch
  - address
  - owner
- Employee
  - salary
  - name
  - subordinates
- Product
  - price
- Bun
  - content
- Turkish Pizza
  - diameter
- Ingredient
  - expirationDate

Different?

- Has employees= hire
- Bake
- Sell
- Fill buns
- Manage= have boss
- Sell
- ~~Know~~
- Be eaten
- Expand
- hire
- ~~have~~
- spoil
- contain

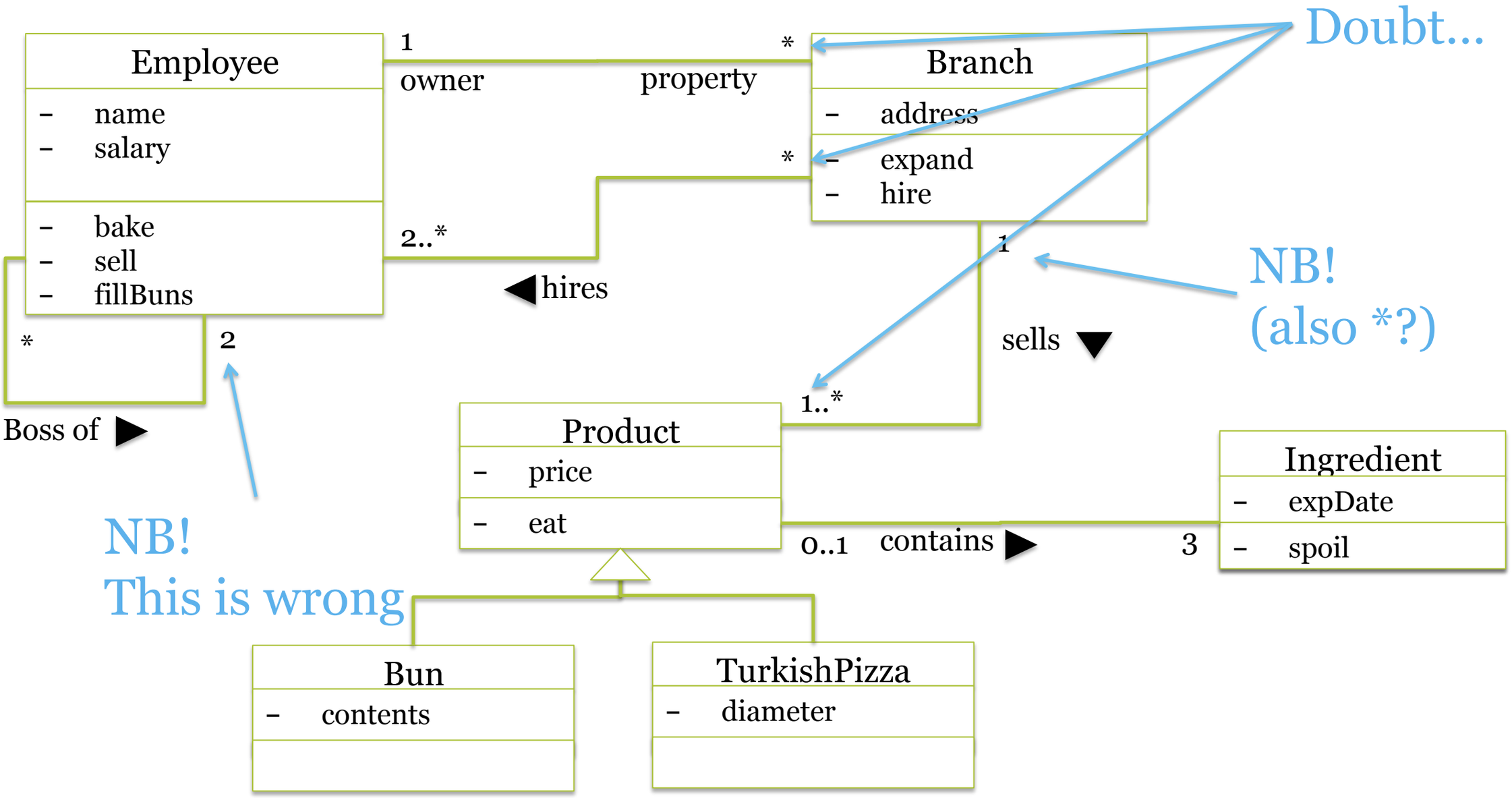
Case 3: Kebab

- Branch
  - address
  - *expand*
  - *hire*
- Employee
  - salary
  - Name
  - *bake*
  - *sell*
  - *fillBuns*
- Product
  - price
  - *eat*
- Bun
  - content
- Turkish Pizza
  - diameter
- Ingredient
  - expirationDate
  - *spoil*

Associations / Dependencies:

- Employee←→ branch
  - hire
  - owner
- Employee ←→ Employee
  - manage
- Branch ←→ Product
  - sell
- Product ←→ Ingredient
  - contain

Case 3: Kebab


Employee
- name
- salary
- bake
- sell
- fillBuns
1
owner
*
property
Branch
- address
- expand
- hire
*
2..*
hires
1
sells
Doubt...
NB!
(also *?)
*
2
Boss of
1..*
Product
- price
- eat
Ingredient
- expDate
- spoil
0..1
contains
3
NB!
This is wrong
Bun
- contents
TurkishPizza
- diameter

Next up...

- Start assignment 3