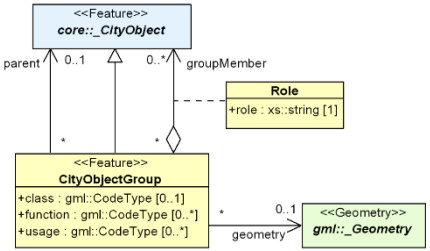
UML class diagrams in a nutshell



Giorgio Agugiaro

Last update: 31 December 2024

License

UML in a nutshell

UML stands for ***Unified Modelling Language*** and is a general-purpose, development modelling language in the field of software engineering that is intended to provide a standard way to visualise the design of a system.

When working with **class diagrams**, three are the main items to look for:

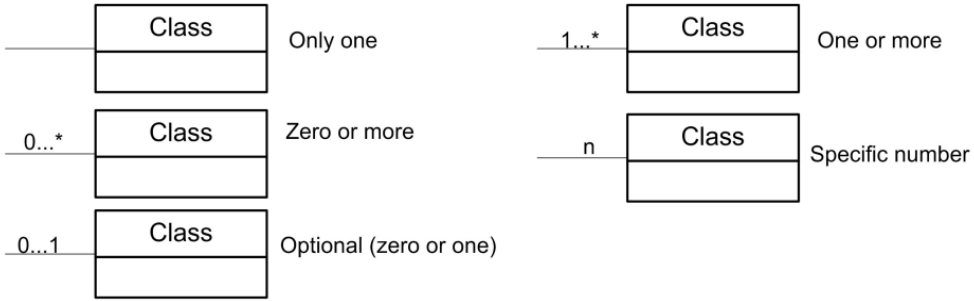
- **Classes** ("what?")
  - Attributes
  - Methods (not treated in these slides)
- **Multiplicity** ("how many?")
- **Relations** ("how do classes relate to each other?")

UML in a nutshell

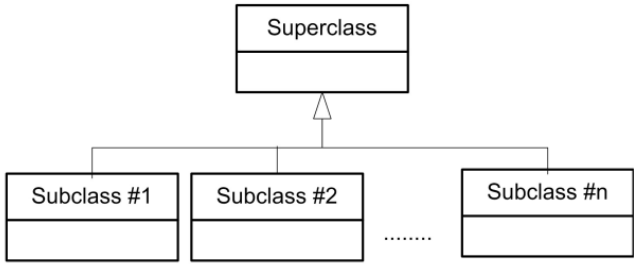
This is a simple way of reading such a relation, e.g. "Class 1 ***has*** Class 2". Alternatively, it is explicitly indicated by the "Role" word, always written near the target.

Association between classes

Association multiplicity



Class inheritance (subtyping)



"is a (superclass of)"

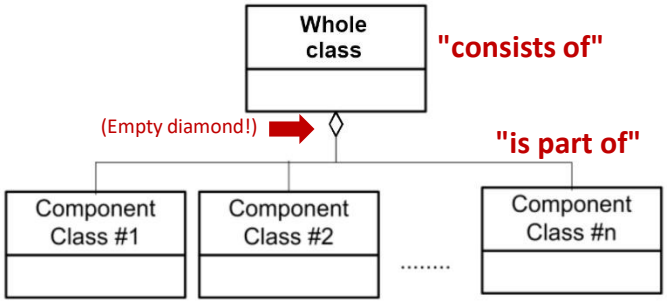
"is a (subclass of)"

UML in a nutshell

Aggregation represents a *weak* form of "has-a" or "whole/part" relationship between the aggregate (the whole) and its parts.

- The life cycle of the parts is not tightly coupled to the life cycle of the aggregate. If the aggregate is destroyed, its parts can continue to exist
- The contained objects can be part of multiple aggregates

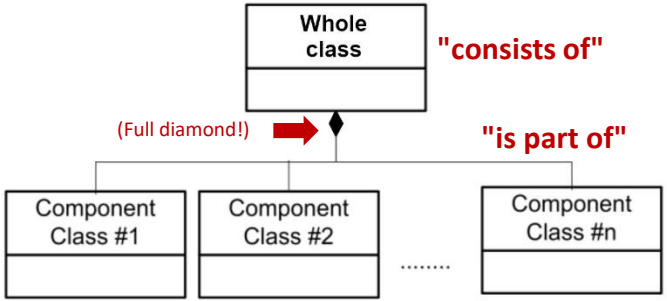
Aggregation between classes



Composition represents a *strong* form of the "has-a" or "whole/part" relationship, characterized by a strong ownership and a coincidental lifecycle between the whole and its parts.

- The whole is responsible for the creation and destruction of its parts. The lifecycles of the parts are tightly coupled with the lifecycle of the whole.
- Each part is contained in only one whole at a time and cannot be shared among multiple wholes

Composition between classes



UML in a nutshell

- Examples of **composition** (left) and **aggregation** (right) between classes

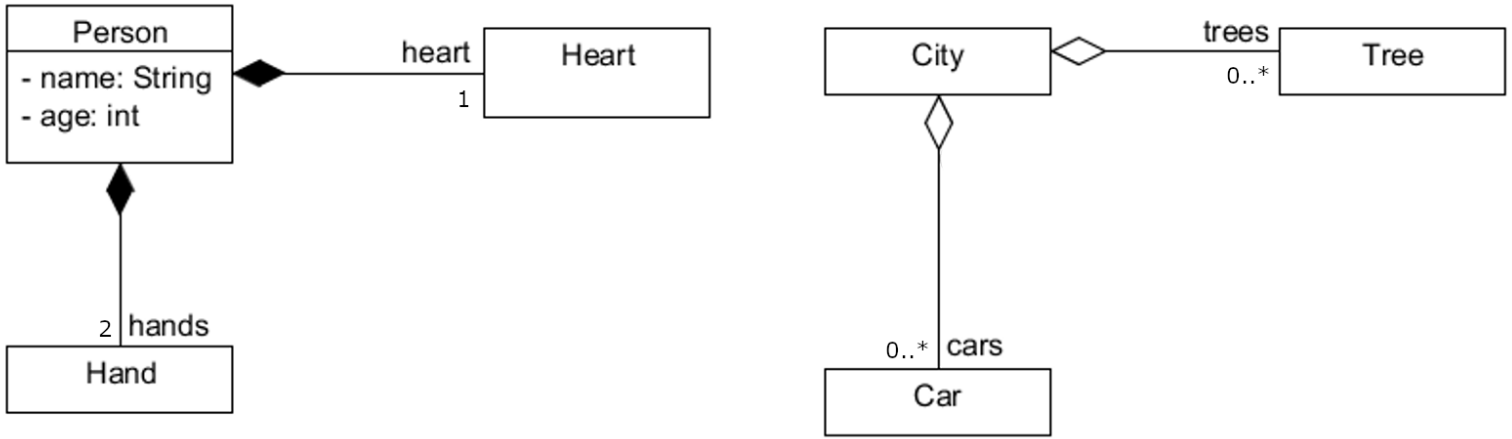


Image source: Adapted from https://i.sstatic.net/WC2eJ.png

explained with...

Class

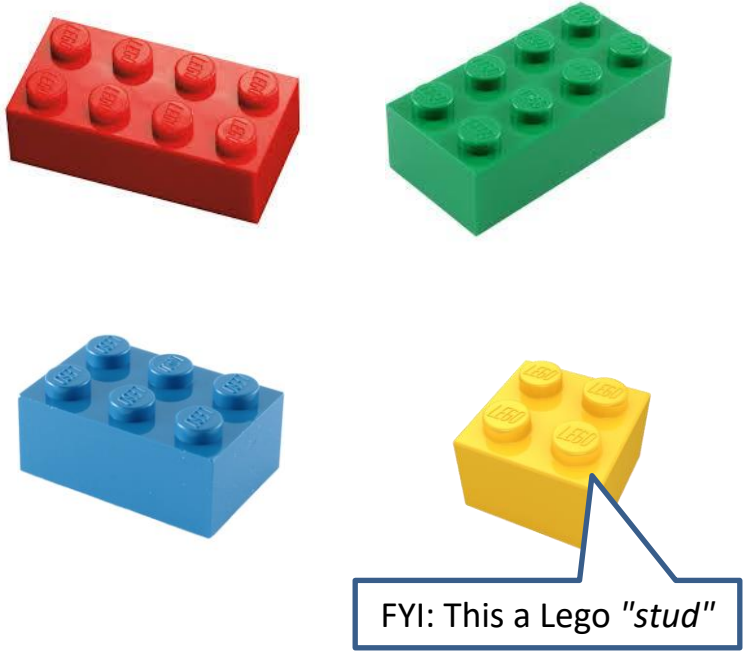
«FeatureType»
LegoBrick

+ name: CharacterString [0..*]
+ fabricationYear: int
+ description: CharacterString [0..1]
+ inProduction: boolean
+ productionDate: Date
+ width_A_studs: int
+ width_B_studs: int
+ height: Length
+ width_A: Length
+ width_B: Length
+ geometry: GM_Solid
+ colour: BrickColourTypeValue

«enumeration»
BrickColourTypeValue

yellow
red
blue
green
grey
white
black

Objects
(*Instances* of the class)



Attributes / Properties

«FeatureType»
LegoBrick

+ name: CharacterString [0..*]
+ fabricationYear: int
+ description: CharacterString [0..1]
+ inProduction: boolean
+ productionDate: Date
+ width_A_studs: int
+ width_B_studs: int
+ height: Length
+ width_A: Length
+ width_B: Length
+ geometry: GM_Solid
+ colour: BrickColourTypeValue

Multiplicity

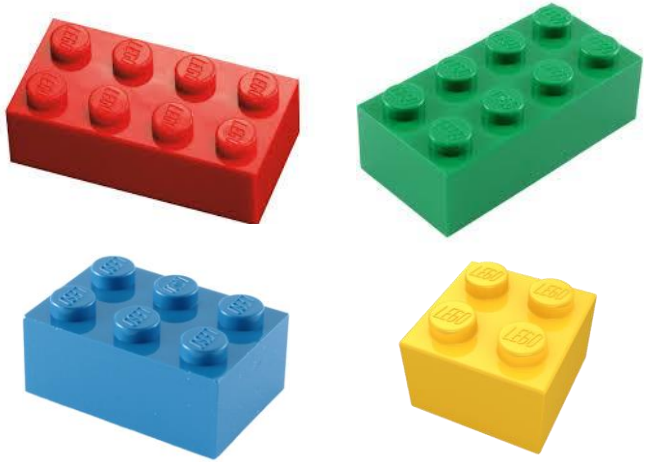
Attributes / Properties **type**: Can have a simple or complex structure.
E.g. length = double (for the value) + CharacterString (for the Unit of Measure)

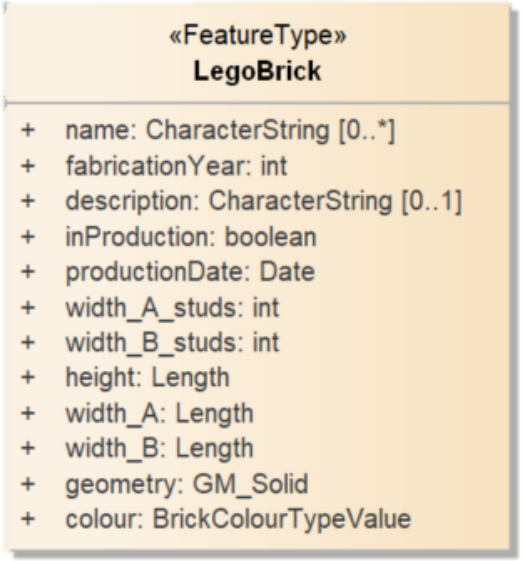
«enumeration»
BrickColourTypeValue

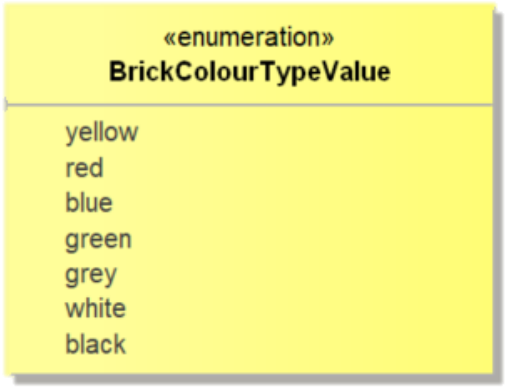
yellow
red
blue
green
grey
white
black

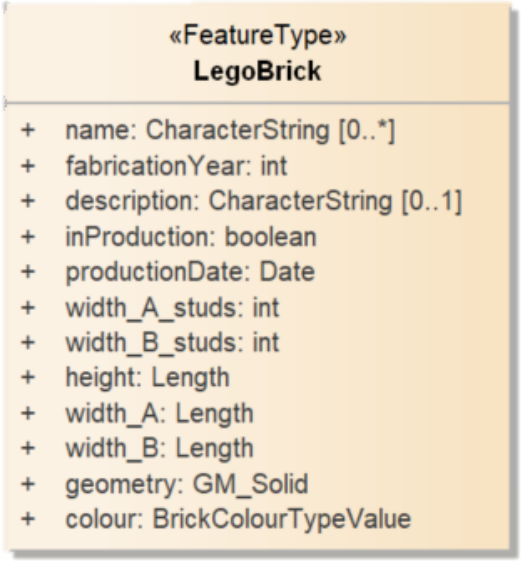
Enumerations contain *closed* sets of possible values
(**Codelists** contain *open* sets of possible values)

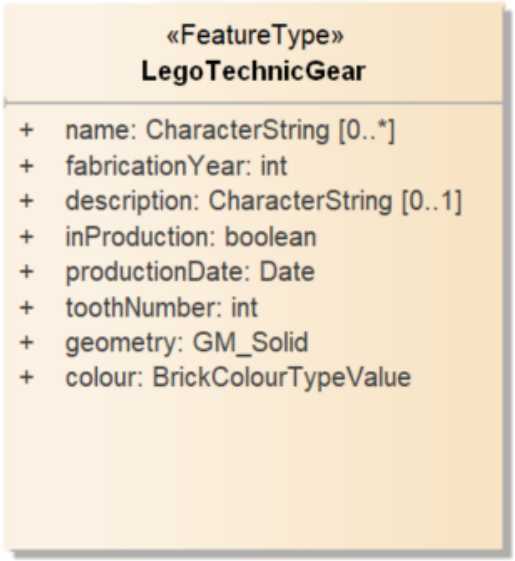
How to represent in UML that Lego does not produce only bricks, but also other elements?

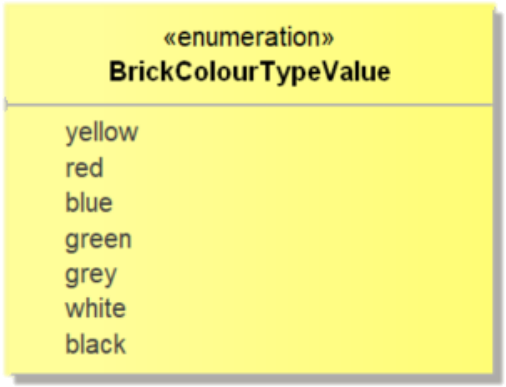






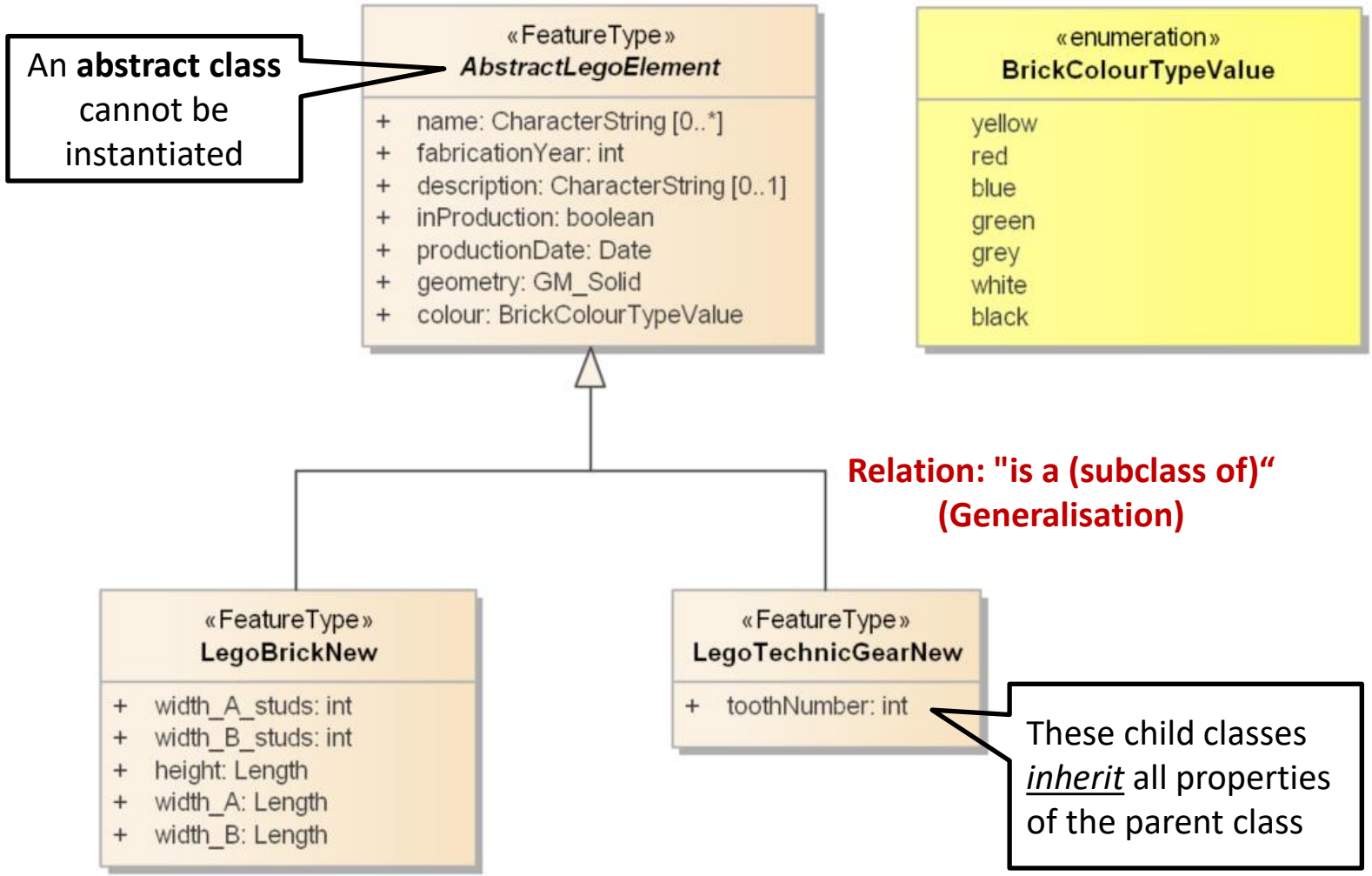


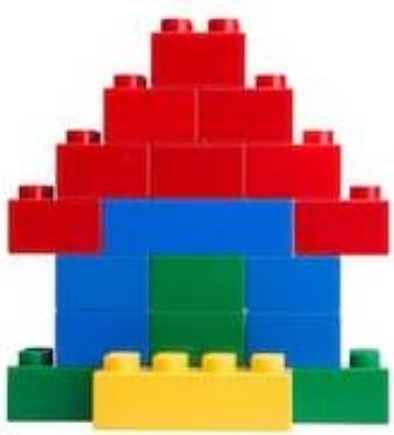




We can simply add another class. However, the two classes share some properties.

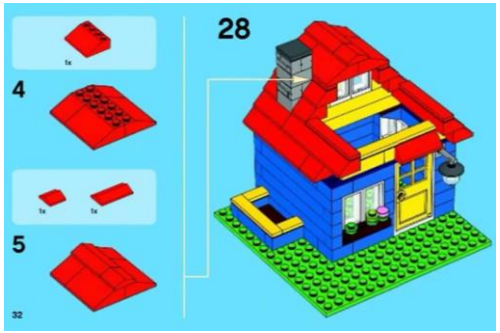
Is there a more elegant way to model them?

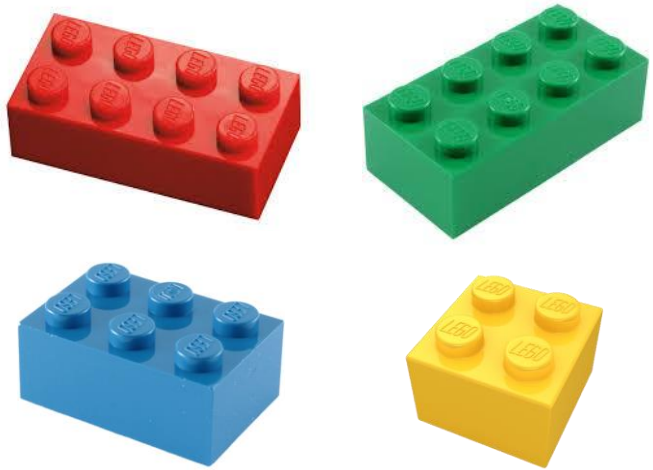


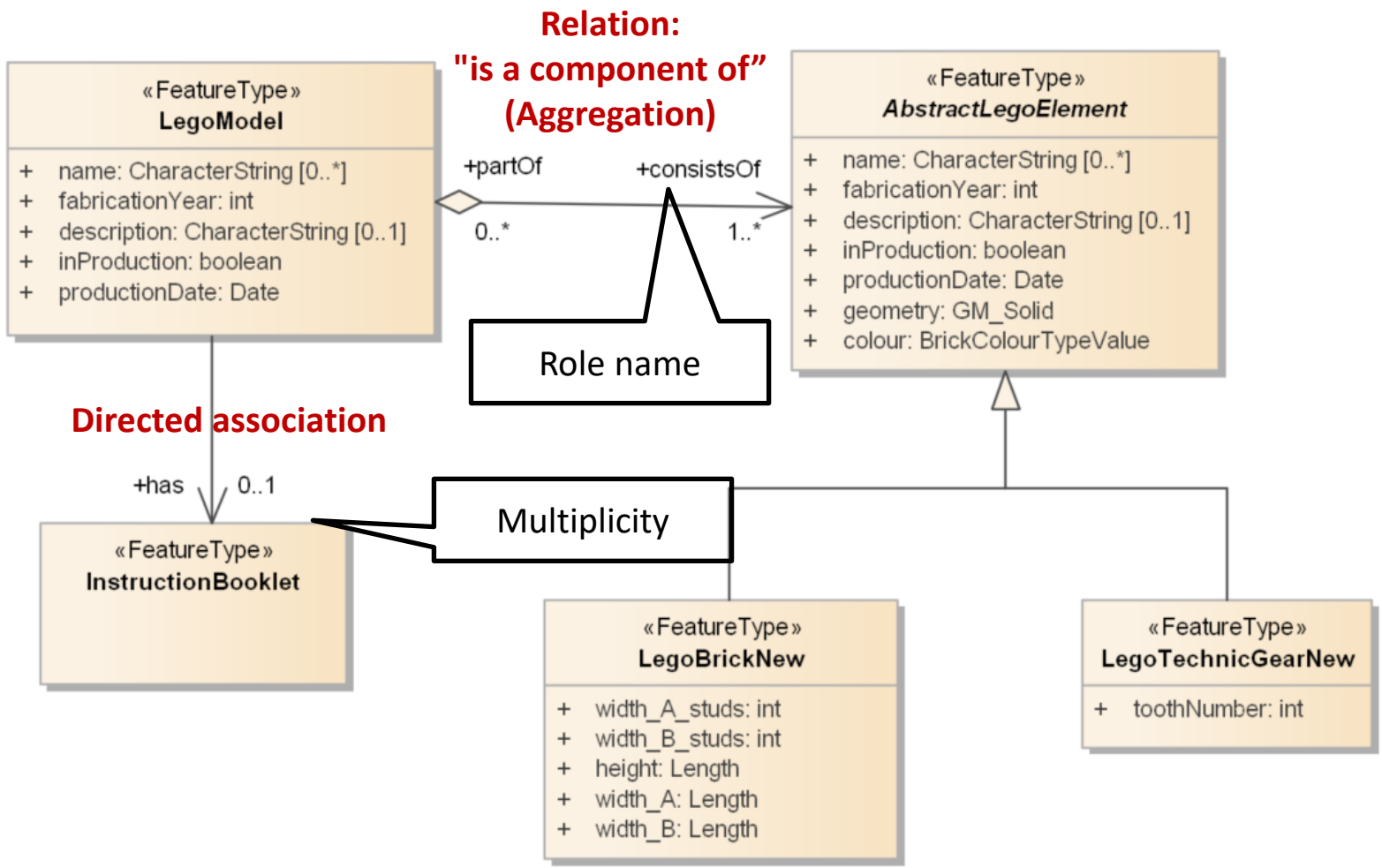


How to represent in UML that:

1) a Lego model (e.g. a house) is made of Lego bricks, and
2) a set of building instructions is generally also included when you buy it?





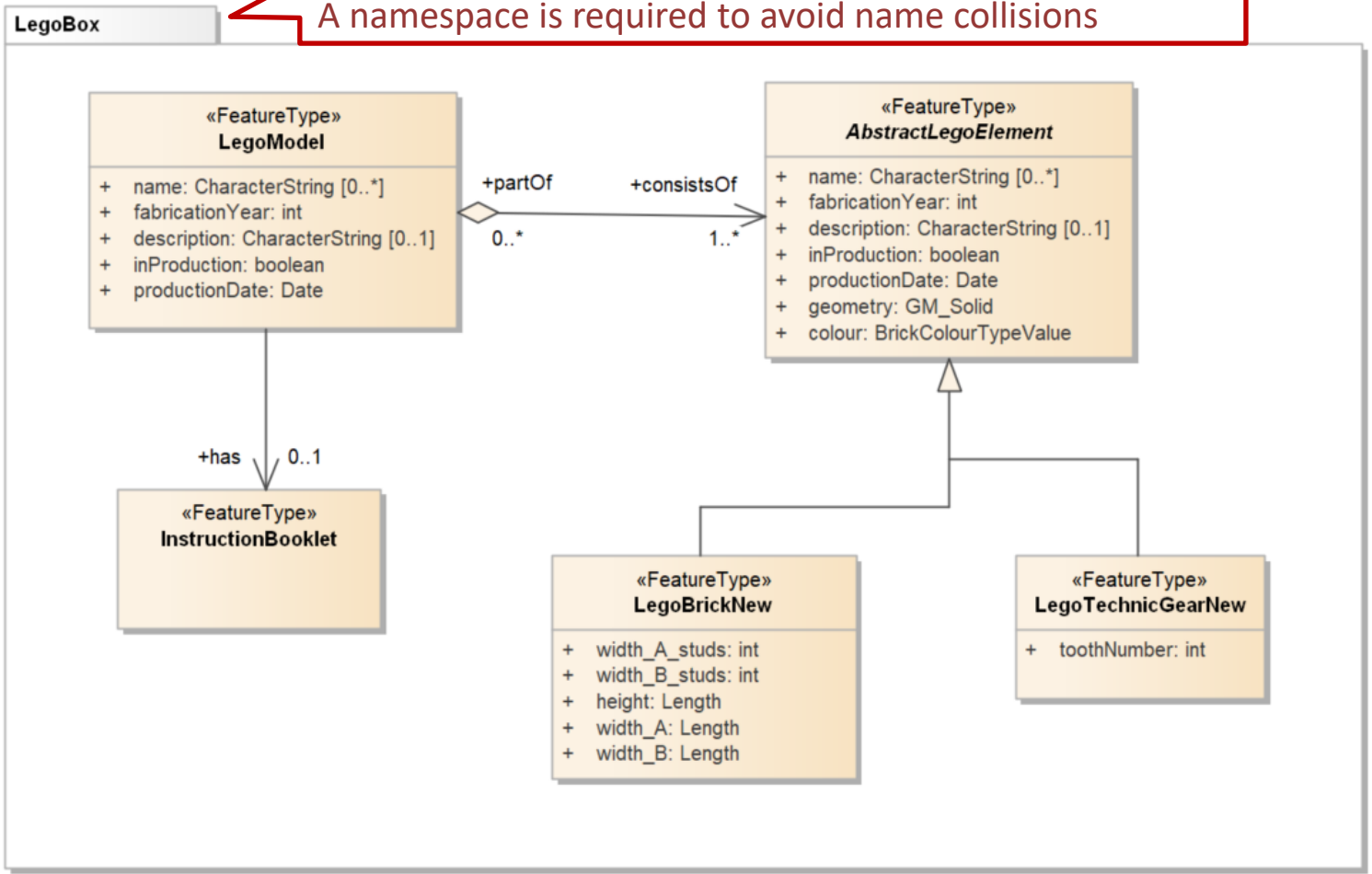


TU Delft 3Dgeoinfo

- Intro
- Classes
- Relations
- **Packages**
- Namespaces

Finally, wow to represent in UML that the bricks and the building instructions are contained in a box?

A package groups elements that belong together.
It also provides a *namespace* for the grouped elements.
A namespace is required to avoid name collisions



Namespaces

- Imagine that we are modelling class "**EARTH**".
- Look at these are 3 examples of different meanings of the word "earth": if we create 3 classes, all called "Earth", we (or the computer) may not know anymore what is what...

Planet

Soil

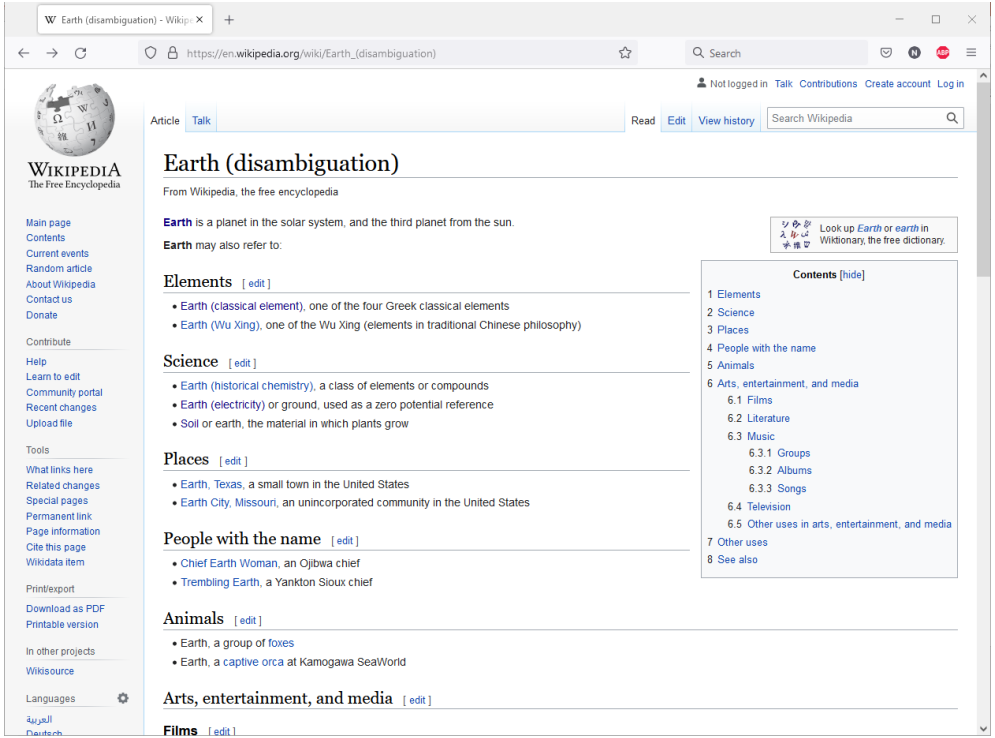
Electrical engineering

Images source: Wikipedia (starting from https://en.wikipedia.org/wiki/Earth_(disambiguation))

Namespaces

- Hence, we need a *namespace*!!
- You can think of a namespace as a sort of "surname" for a person. It helps reducing the cases of confusion with two or more people with the same name...

- So, we may end up with:
  - Astronomy:Earth
  - Geology:Earth
  - Electricity:Earth

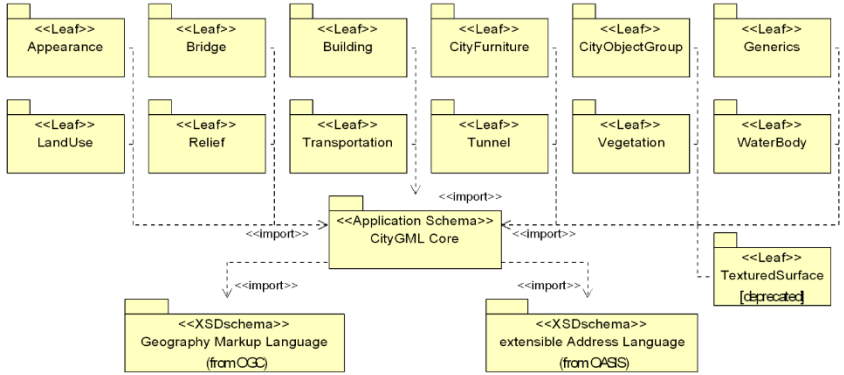


https://en.wikipedia.org/wiki/Earth_(disambiguation)

Namespaces

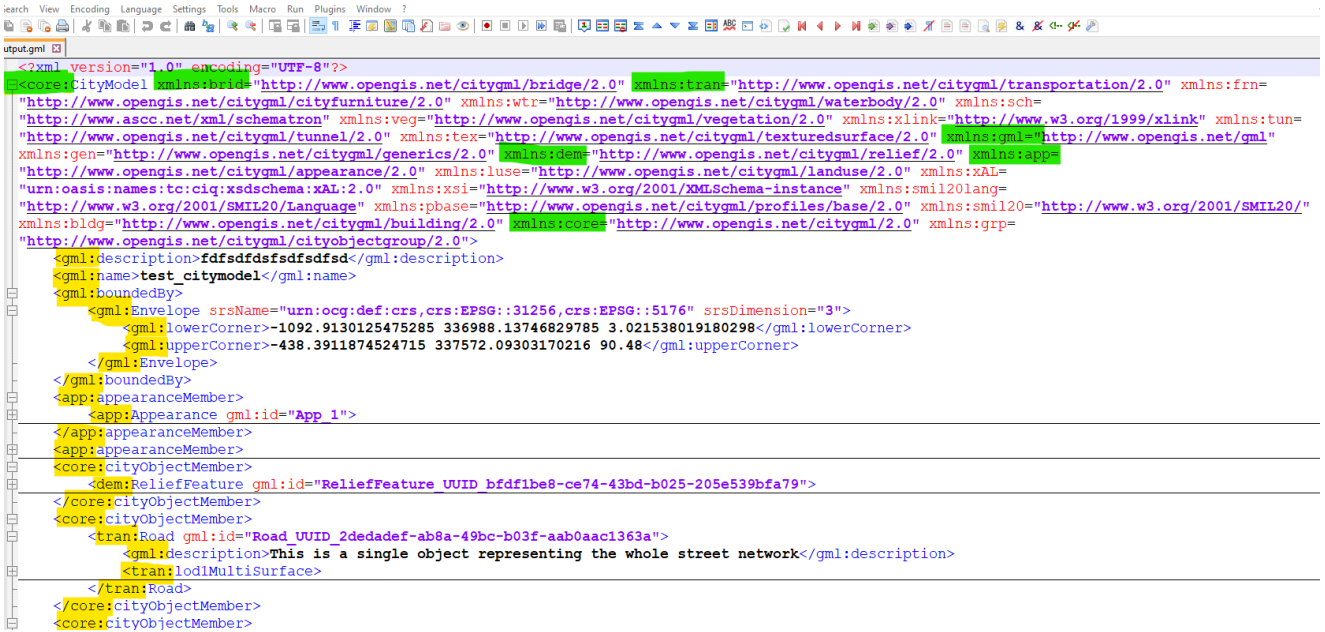
- In UML, the concept of package is used to represent the namespace
- All classes within the same package are intended to belong together and share some common characteristics
- In the case of class earth example, you'd need three different packages in order to separate the 3 different earth classes and avoid semantic misunderstanding

- In CityGML, for example, there are several packages. The one called "Building" contains all classes to model a building, the one called "Terrain" contains all classes to model a digital representation of the terrain, etc.



Namespaces

- In an XML document, the definition of the namespaces is given in the header, using the tag **xmlns** (stands for... xml namespace ☺)
- For classes (and their properties/attributes) it is then indicated before the name using a "**:**" (colon) as separator



Thank you for your attention!

Dr. Giorgio Agugiaro
g.agugiaro@tudelft.nl
3D Geoinformation Group
TU Delft
The Netherlands
https://3d.bk.tudelft.nl/gagugiaro