Formalization of UML Class Diagrams in First Order Logic

Giuseppe De Giacomo

Sapienza Università di Roma
Laurea Magistrale in Ingegneria Informatica

Let's start with an exercise ...

Requirements: We are interested in building a software application to manage filmed scenes for realizing a movie, by following the so-called "Hollywood Approach".

Every **scene** is identified by a code (a string) and it is described by a text in natural language.

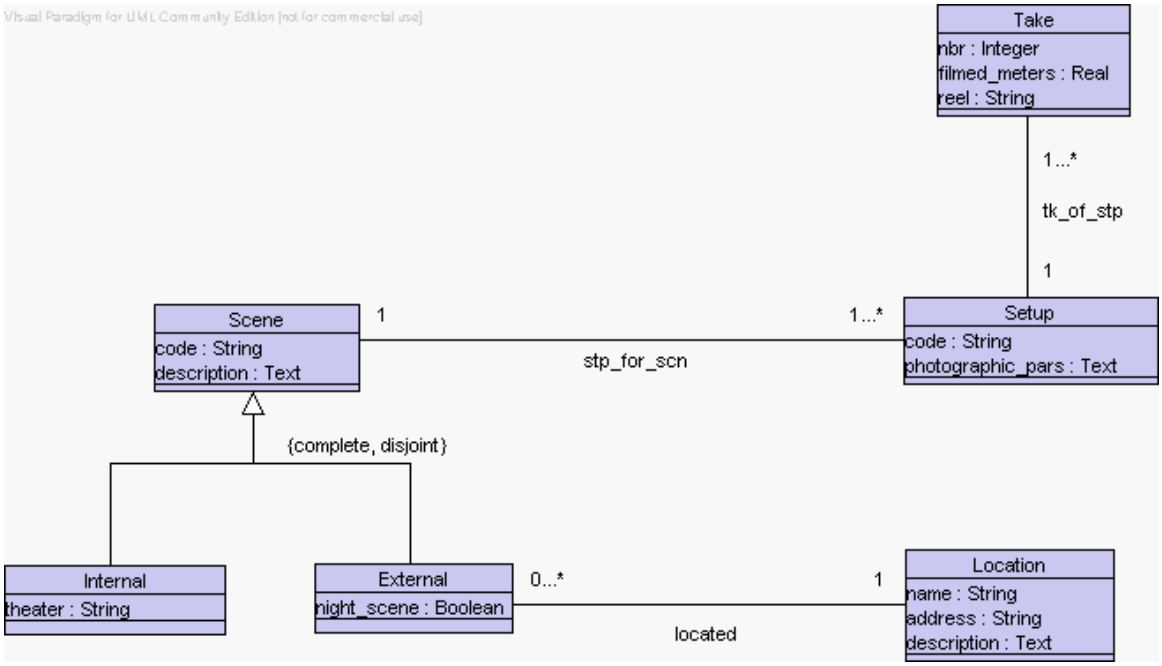
Every scene is filmed from different positions (at least one), each of this is called a **setup**. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

For every setup, several **takes** may be filmed (at least one). Every take is characterized by a (positive) natural number, a real number representing the number of meters of film that have been used for shooting the take, and the code (a string) of the reel where the film is stored. Note that a take is associated to a single setup.

Scenes are divided into **internals** that are filmed in a theater, and **externals** that are filmed in a **location** and can either be "day scene" or "night scene". Locations are characterized by a code (a string) and the address of the location, and a text describing them in natural language.

Write a precise specification of this domain using any formalism you like.

Solution 1: ... use conceptual modeling diagrams (UML)!!!



Solution 1: ... use conceptual modeling diagrams (discussion)

Good points:

- Easy to generate (it's the standard in software design)
- Easy to understand for humans
- Well disciplined, well-established methodologies available

Bad points:

- No precise semantics (people that use it wave hands about it)
- Verification (or better validation) done informally by humans
- Machine incomprehensible (because of lack of formal semantics)
- Automated reasoning out of question
- Limited expressiveness

Solution 2: ... use logic!!!

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stp\_for\_scn(x, y)$, $ck\_of\_stp(x, y)$, $located(x, y)$, ...
Axioms:

$\forall x, y.\ code_{Scene}(x, y) \supset Scene(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Scene(x) \wedge Text(y)$

$\forall x, y.\ code_{Setup}(x, y) \supset Setup(x) \wedge String(y)$
$\forall x, y.\ photographic\_pars(x, y) \supset Setup(x) \wedge Text(y)$

$\forall x, y.\ nbr(x, y) \supset Take(x) \wedge Integer(y)$
$\forall x, y.\ filmed\_meters(x, y) \supset Take(x) \wedge Real(y)$
$\forall x, y.\ reel(x, y) \supset Take(x) \wedge String(y)$

$\forall x, y.\ theater(x, y) \supset Internal(x) \wedge String(y)$

$\forall x, y.\ night\_scene(x, y) \supset External(x) \wedge Boolean(y)$

$\forall x, y.\ name(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ address(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Location(x) \wedge Text(y)$

$\forall x.\ Scene(x) \supset (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
. . .

$\forall x, y.\ stp\_for\_scn(x, y) \supset Setup(x) \wedge Scene(y)$
$\forall x, y.\ tk\_of\_stp(x, y) \supset Take(x) \wedge Setup(y)$
$\forall x, y.\ located(x, y) \supset External(x) \wedge Location(y)$

$\forall x.\ Setup(x) \supset 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\ Scene(y) \supset 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\ Take(x) \supset 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\ Setup(y) \supset 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\ External(x) \supset 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\ Internal(x) \supset Scene(x)$
$\forall x.\ External(x) \supset Scene(x)$
$\forall x.\ Internal(x) \supset \neg External(x)$
$\forall x.\ Scene(x) \supset Internal(x) \vee External(x)$

Solution 2: ... use logic (discussion)

Good points:

- Precise semantics
- Formal verification
- Machine comprehensible
- Virtually unlimited expressiveness

Bad points:

- Difficult to generate
- Difficult to understand for humans
- Too unstructured (making reasoning difficult), no well-established methodologies available
- Automated reasoning may be impossible

Solution 3: ... mix them!!!

Note these two approaches look as being orthogonal! But they can in fact be integrated!!!
Basic idea:

- Assign formal semantics to constructs of the conceptual design diagrams
- Use conceptual design diagrams as usual, taking advantage of methodologies developed for them in Software Engineering
- Read diagrams as logical theories when needed, i.e., for formal understanding, verification, automated reasoning, etc.

Added values:

- inherit from conceptual modeling diagrams: ease-to-use for humans
- inherit from logic: formal semantics and reasoning tasks, which are needed for formal verification and machine manipulation

Solution 3: ... mix them!!! (cont.)

Important point: by using conceptual modeling diagrams one gets logical theories of a specific form.

- One gets limited (or better, well-disciplined) expressiveness
- One can exploit the particular form of the corresponding logical theory to simplify reasoning, hopefully getting:
  - decidability
  - reasoning procedures that match intrinsic computational complexity

In this part of the course ...

We will illustrate what we get from integrating logic with conceptual modeling diagrams.
We will use ...

- as conceptual modeling diagrams:
  - UML Class Diagrams
- as logic:
  - First-Order Logic to formally capture semantics and reasoning
  - Description Logic to understand the computational properties of reasoning.

Unified Modeling Language (UML)

UML stands for Unified Modeling Language. It was developed in 1994 by unifying and integrating the most prominent object-oriented modeling approaches of that age:

- Booch
- Rumbaugh: Object Modeling Technique (OMT)
- Jacobson: Object-Oriented Software Engineering (OOSE)

History:

- 1995, version 0.8, Booch, Rumbaugh; 1996, version 0.9, Booch, Rumbaugh, Jacobson; version 1.0 BRJ + Digital, IBM, HP, ...
- Best known version: 1.2–1.5 (1999–2004)
- Current version: 2.0 (2005)
- 1999/today: de facto standard object-oriented modeling language

References:

- Grady Booch, James Rumbaugh, Ivar Jacobson, "The unified modeling language user guide", Addison Wesley, 1999 (second edition, 2005)
- `http://www.omg.org` → UML
- `http://www.uml.org`

UML Class Diagrams

In this course we deal with one of the most prominent components of UML: UML Class Diagrams.

A UML Class Diagram is used to represent explicitly the information on a domain of interest (typically the application domain of a software).

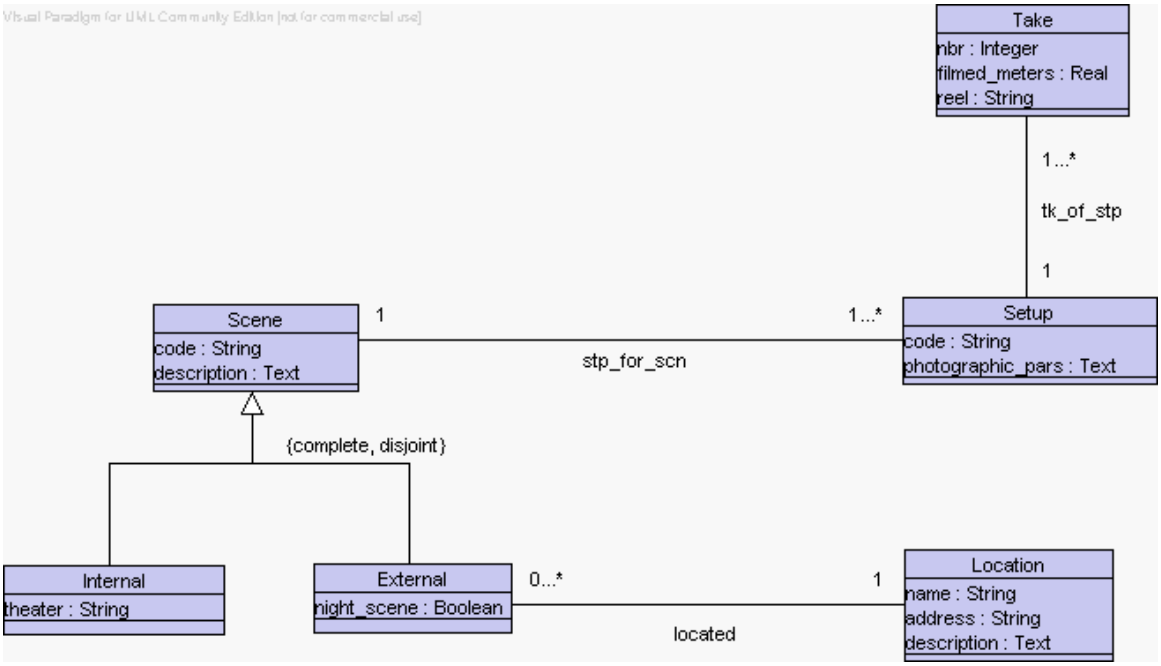
Note: This is exactly the goal of all conceptual modeling formalism, such as the Entity-Relationship Schemas (standard in Database design) or Ontologies (now in vogue due to the Semantic Web – see later) .

UML Class Diagrams (cont.1)

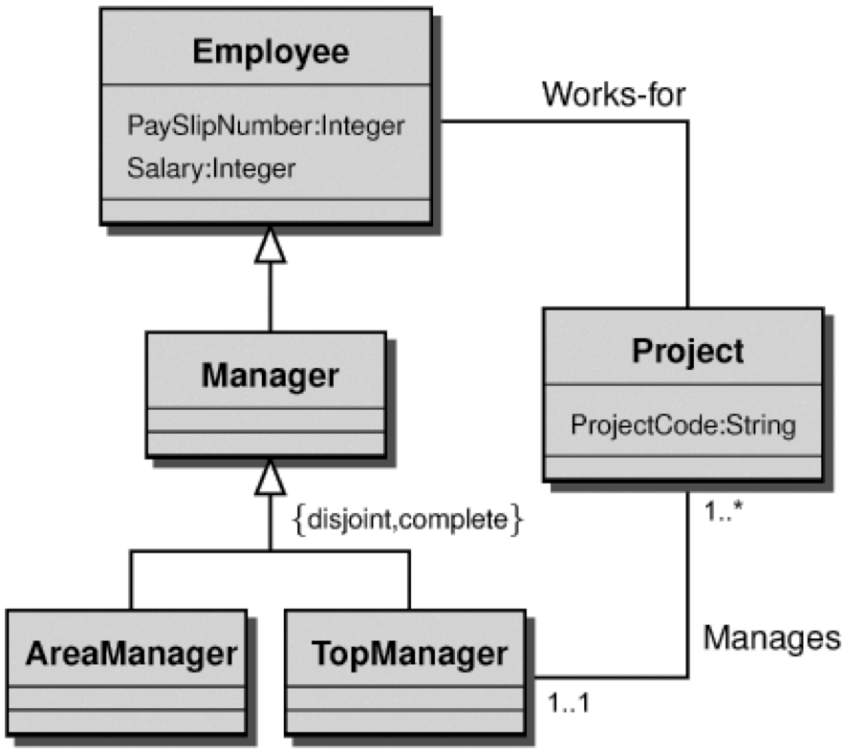
The UML class diagram models the domain of interest in terms of:

- objects grouped into classes
- (simple) properties of classes ("attributes", "operations")
- relationships (associations) between classes
- sub-classing i.e., ISA/Generalization relationships

Example of an UML Class Diagram



Another Example of an UML Class Diagram



UML Class Diagrams (cont.2)

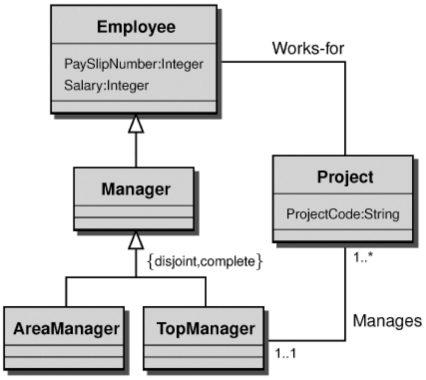
In fact UML class diagrams are used in various phase of a software design:

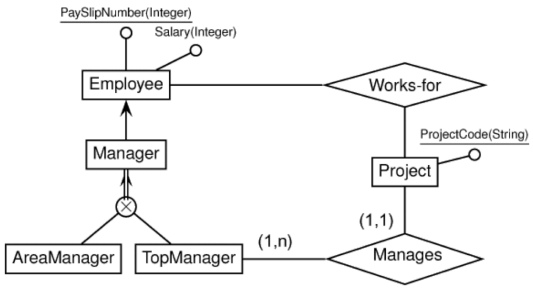
1. during the so-called analysis, where an abstract precise view of the domain of interest needs to be developed – the so-call "conceptual perspective"
2. during software development to maintain an abstract view of the software to be developed – the so-called "implementation perspective"

In this course we focus on 1!

UML Class Diagrams and ER Schemas

UML class diagrams are heavily influenced by Entity-Relationship Schemas.
Example of UML vs. ER:





UML Class Diagrams and ER Schemas (cont.)

Differences concern mostly the features needed for the implementation perspective such as: public, protected, and private qualifiers for operations and attributes.

But also cardinality constrains on participation to non-binary relationship relationships – better defined in ER (see later).

Note: what we learn in this course on UML Class Diagrams holds for ER Schema as well!!!

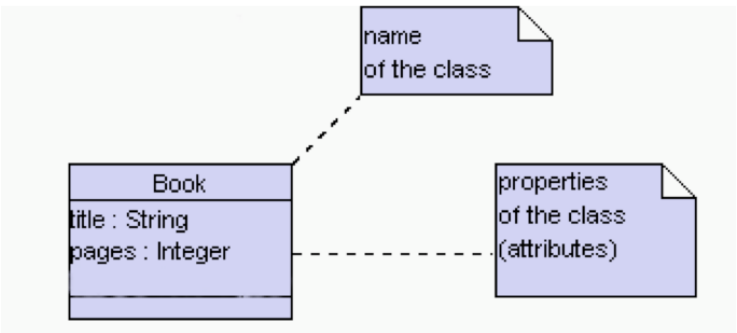
Classes in UML

A class in UML models a set of objects (its "instances") that share certain common properties: attributes, operations, etc.

Each class is characterized by:

- a name (which must be unique in the whole class diagram)
- a set of (local) properties, namely attributes and operations (see later).

Example:



Classes in UML: instances

The objects that belong to a class are called instances of the class. They form a so-called instantiation (or extension) of the class.
Example:
Here are some possible instantiations for our class *Book*.

$$\{book_1, book_2, book_3, \ldots\}$$

$$\{book_\alpha, book_\beta, book_\gamma, \ldots\}$$

Which is the actual instantiation? *We will know it only at run-time!!! – we are now at design time!*

Classes in UML: semantics

A class represent set of objects ... but which set? We don't actually know.

So, how can we assign such a semantics to a class?

Use a FOL unary predicate!!!

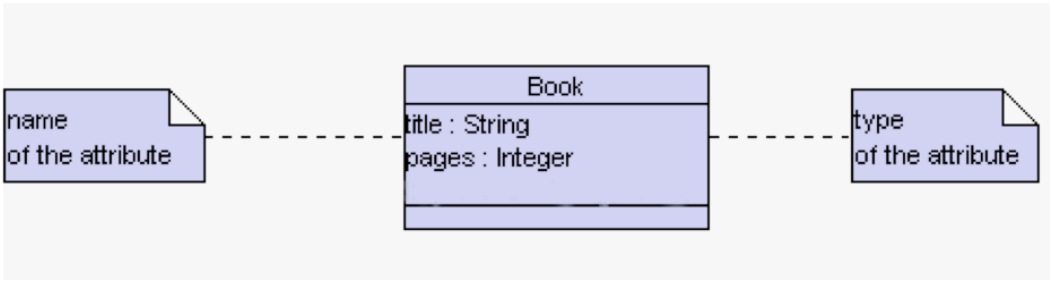
Example:
For our class *Book*, we introduce a predicate $Book(x)$.

Attributes

An attribute models a local property of a class.
It is characterized by:

- a name (which is unique only in the class it belongs to)
- and a type (a collection of possible values)
- and possibly a multiplicity

Example:



Attributes (cont.1)

Attributes without explicit multiplicity are:

- mandatory (must have at least a value)
- single-valued (must have at most a value)

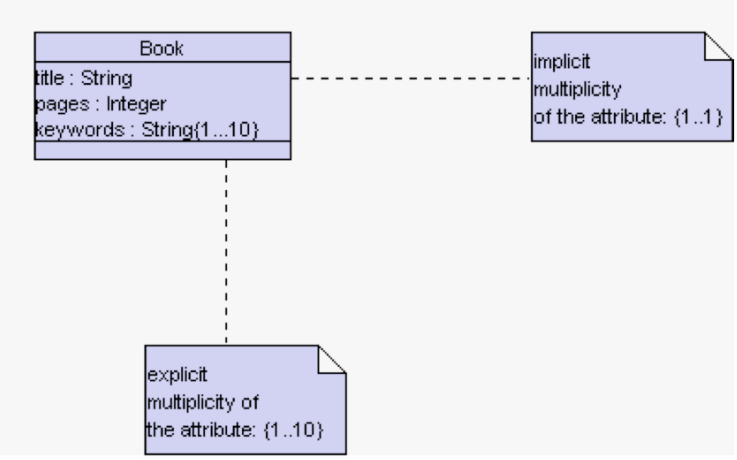
That is, they are functions from the instances of the class to the values of the type they have.

Example:
$book_1$ has as value for the attribute name the *String*: "*The little digital video book*".

Attributes (cont.2)

More generally attributes may have an explicit multiplicity, i.e., a minimal and maximal number of values.

Example:



When multiplicity is implicit then it is assumed to be $1 \ldots 1$.

Attributes: formalization

Since attributes may have a multiplicity different from $1 \ldots 1$ they are better formalized as binary predicates, with suitable assertions representing types and multiplicity:
Given an attribute a of a class C with type T and multiplicity $i \ldots j$ we capture it in FOL as a binary predicate $a_C(x, y)$ with the following assertions:

- Assertion for the attribute type

$$\forall x, y.\ a(x, y) \supset C(x) \wedge T(y)$$

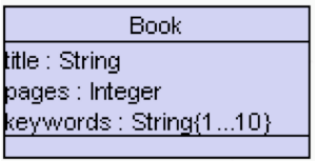
- Assertion for the multiplicity

$$\forall x.\ C(x) \supset (i \leq \sharp\{y \mid a(x, y)\} \leq j)$$

Note: this is a shorthand for a FOL formula expressing cardinality of the possible values for y.

Attributes: formalization (cont.)

Example:

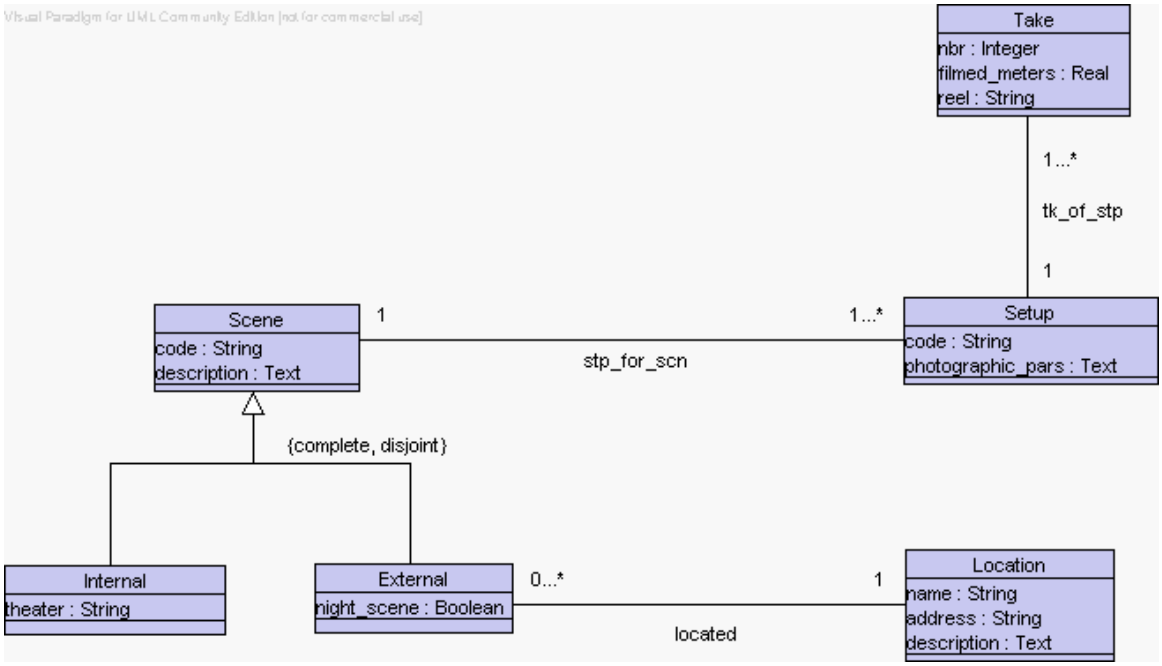


$\forall x, y.\ title(x, y) \supset Book(x) \wedge String(y)$
$\forall x.\ Book(x) \supset (1 \leq \sharp\{y \mid title(x, y)\} \leq 1)$

$\forall x, y.\ pages(x, y) \supset Book(x) \wedge Integer(y)$
$\forall x.\ Book(x) \supset (1 \leq \sharp\{y \mid pages(x, y)\} \leq 1)$

$\forall x, y.\ keywords(x, y) \supset Book(x) \wedge String(y)$
$\forall x.\ Book(x) \supset (1 \leq \sharp\{y \mid keywords(x, y)\} \leq 10)$

In our example ...



In our example ...

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stp\_for\_scn(x, y)$, $ck\_of\_stp(x, y)$, $located(x, y)$, ...
Axioms:

$\forall x, y.\ code_{Scene}(x, y) \supset Scene(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Scene(x) \wedge Text(y)$

$\forall x, y.\ code_{Setup}(x, y) \supset Setup(x) \wedge String(y)$
$\forall x, y.\ photographic\_pars(x, y) \supset Setup(x) \wedge Text(y)$

$\forall x, y.\ nbr(x, y) \supset Take(x) \wedge Integer(y)$
$\forall x, y.\ filmed\_meters(x, y) \supset Take(x) \wedge Real(y)$
$\forall x, y.\ reel(x, y) \supset Take(x) \wedge String(y)$

$\forall x, y.\ theater(x, y) \supset Internal(x) \wedge String(y)$

$\forall x, y.\ night\_scene(x, y) \supset External(x) \wedge Boolean(y)$

$\forall x, y.\ name(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ address(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Location(x) \wedge Text(y)$

$\forall x.\ Scene(x) \supset (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
. . .

$\forall x, y.\ stp\_for\_scn(x, y) \supset Setup(x) \wedge Scene(y)$
$\forall x, y.\ tk\_of\_stp(x, y) \supset Take(x) \wedge Setup(y)$
$\forall x, y.\ located(x, y) \supset External(x) \wedge Location(y)$

$\forall x.\ Setup(x) \supset 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\ Scene(y) \supset 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\ Take(x) \supset 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\ Setup(y) \supset 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\ External(x) \supset 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\ Internal(x) \supset Scene(x)$
$\forall x.\ External(x) \supset Scene(x)$
$\forall x.\ Internal(x) \supset \neg External(x)$
$\forall x.\ Scene(x) \supset Internal(x) \vee External(x)$

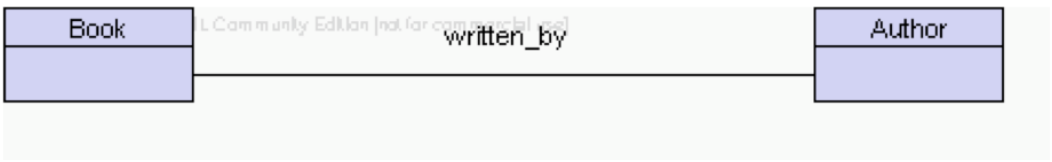
Associations

Relationships between classes are modeled in UML Class Diagrams as Associations.

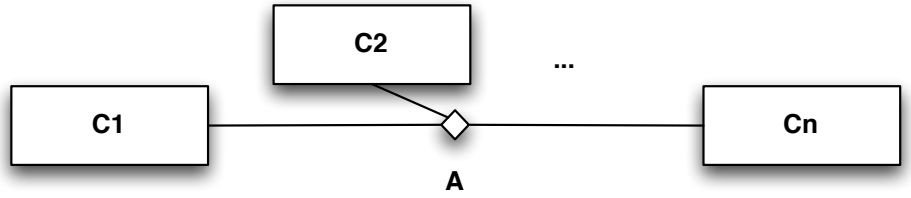
An association in UML is a relation between the instances of two or more classes.

Association model properties of classes that are non-local, in the sense that they involve other classes.
An association between two classes is a property of both classes.

Example:



Associations: formalization



We can represent an n-ary association A among classes $C_1, \ldots, C_n$ as n-ary predicate A in FOL.
We assert that the components of the predicate must belong to correct classes:

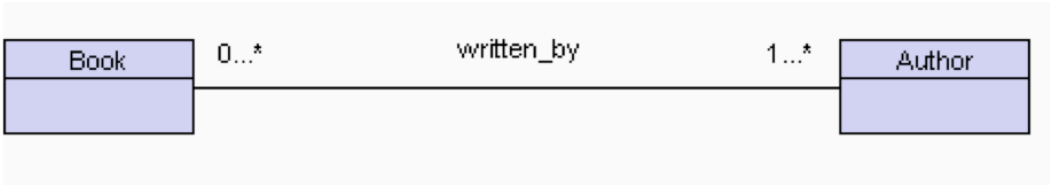
$$\forall x_1, \ldots, x_n.\ A(x_1, \ldots, x_n) \supset C_1(x_1) \wedge \ldots \wedge C_n(x_n)$$

Example:

$$\forall x_1, x_2.\ \mathit{written\_by}(x_1, x_2) \supset \mathit{Book}(x_1) \wedge \mathit{Author}(x_2)$$

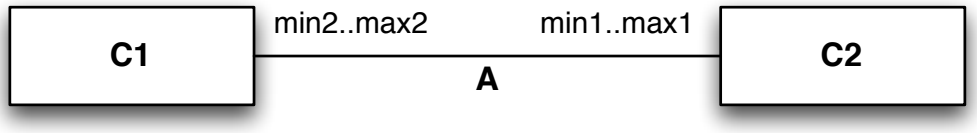
Associations: multiplicity

On binary associations we can place multiplicity constrains as we did for attributes:
Example:



Note: UML multiplicities for associations are look-across and are not easy to use in an intuitive way for n-ary associations, so typically they are not used at all.
In contrast, in ER Schemas, multiplicities are not look-across and are easy to use, and widely used.

Associations: formalization (cont.)



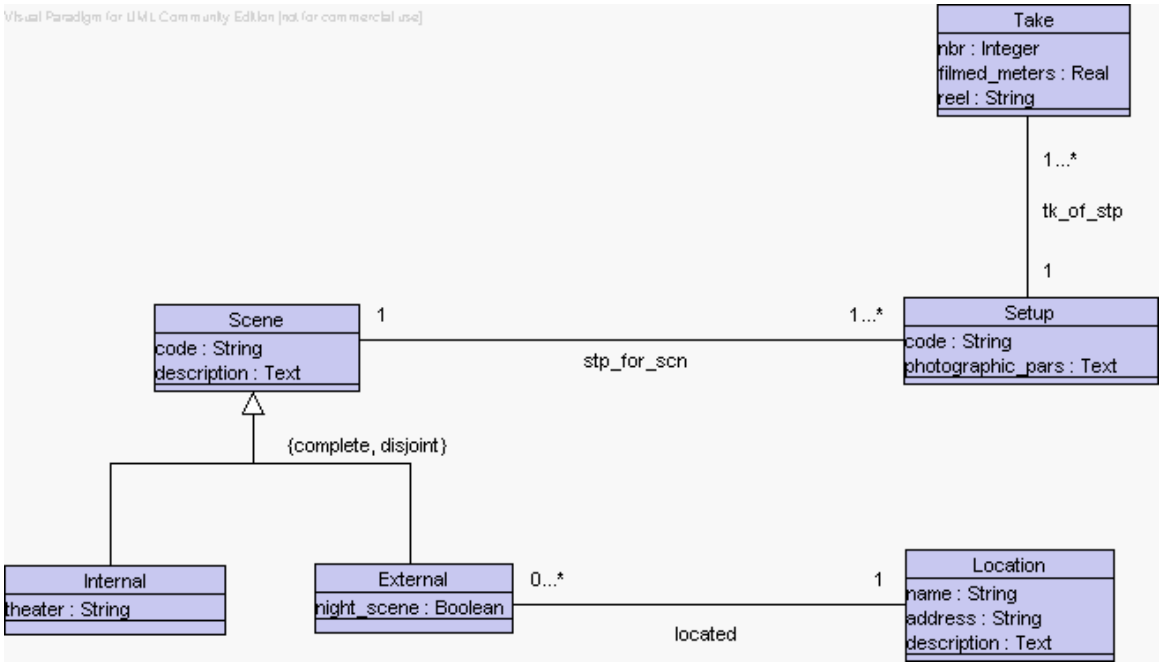
Multiplicities of binary associations are easily expressible in FOL:

$$\forall x_1.\ C_1(x_1) \supset (min_1 \leq \sharp\{x_2 \mid A(x_1, x_2)\} \leq max_1)$$
$$\forall x_2.\ C_2(x_2) \supset (min_2 \leq \sharp\{x_1 \mid A(x_1, x_2)\} \leq max_2)$$

Example:

$$\forall x.\ Book(x) \supset (1 \leq \sharp\{y \mid written\_by(x, y)\})$$

In our example ...



In our example ...

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stp\_for\_scn(x, y)$, $ck\_of\_stp(x, y)$, $located(x, y)$, ...
Axioms:

$\forall x, y.\ code_{Scene}(x, y) \supset Scene(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Scene(x) \wedge Text(y)$

$\forall x, y.\ code_{Setup}(x, y) \supset Setup(x) \wedge String(y)$
$\forall x, y.\ photographic\_pars(x, y) \supset Setup(x) \wedge Text(y)$

$\forall x, y.\ nbr(x, y) \supset Take(x) \wedge Integer(y)$
$\forall x, y.\ filmed\_meters(x, y) \supset Take(x) \wedge Real(y)$
$\forall x, y.\ reel(x, y) \supset Take(x) \wedge String(y)$

$\forall x, y.\ theater(x, y) \supset Internal(x) \wedge String(y)$

$\forall x, y.\ night\_scene(x, y) \supset External(x) \wedge Boolean(y)$

$\forall x, y.\ name(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ address(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Location(x) \wedge Text(y)$

$\forall x.\ Scene(x) \supset (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
. . .

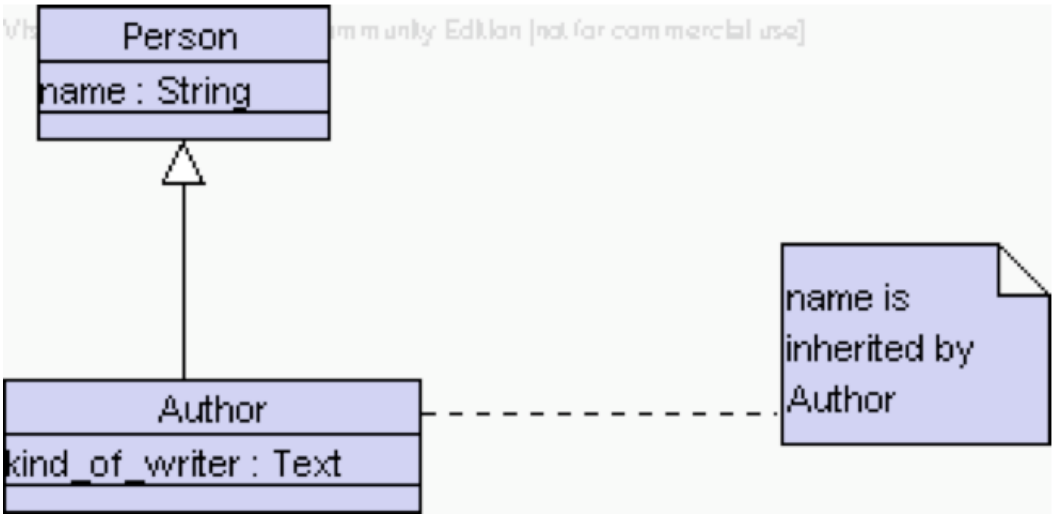
$\forall x, y.\ stp\_for\_scn(x, y) \supset Setup(x) \wedge Scene(y)$
$\forall x, y.\ tk\_of\_stp(x, y) \supset Take(x) \wedge Setup(y)$
$\forall x, y.\ located(x, y) \supset External(x) \wedge Location(y)$

$\forall x.\ Setup(x) \supset 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\ Scene(y) \supset 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\ Take(x) \supset 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\ Setup(y) \supset 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\ External(x) \supset 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\ Internal(x) \supset Scene(x)$
$\forall x.\ External(x) \supset Scene(x)$
$\forall x.\ Internal(x) \supset \neg External(x)$
$\forall x.\ Scene(x) \supset Internal(x) \vee External(x)$

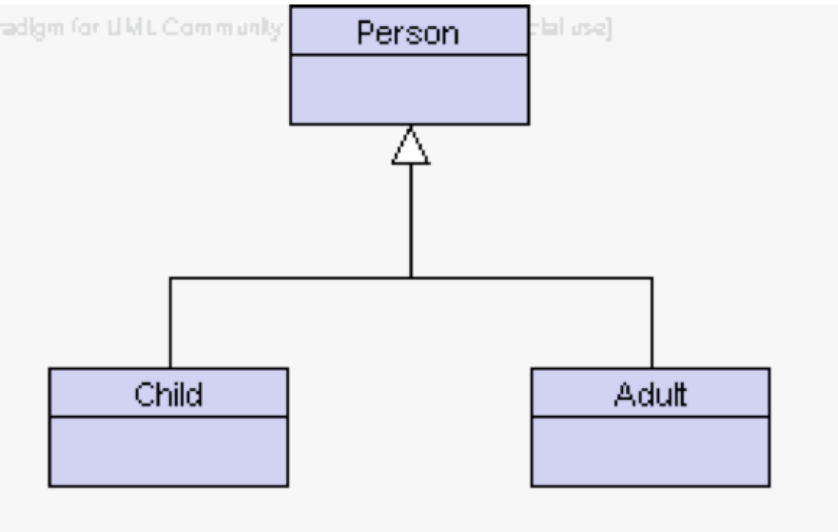
ISA/Generalization

The ISA relationship is of particular importance in conceptual modeling: a class C ISA a class C' if every instance of C is also an instance of C'. In UML the ISA relationship is modeled through the notion of generalization.
Example:



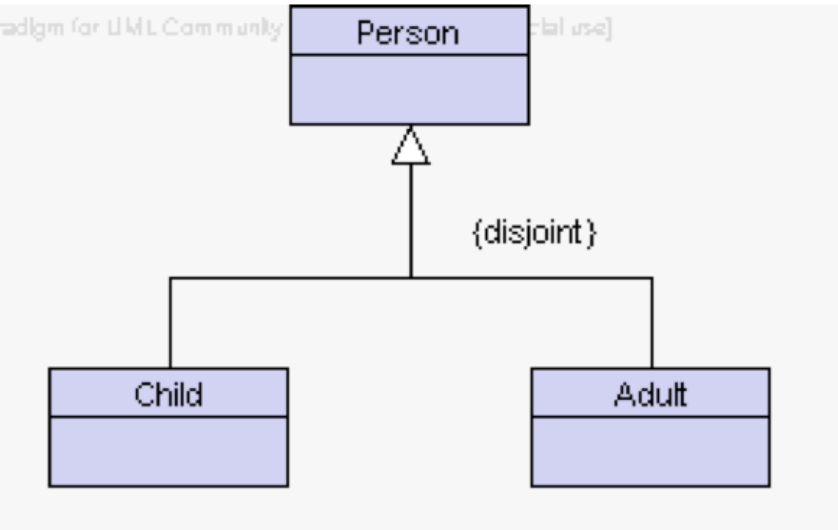
Generalization (cont.1)

A generalization involves a superclass (base class) and one or more subclasses: every instance of each subclass is also an instance of the superclass.
Example:



Generalization (cont.2)

The ability of having more subclasses in the same generalization, allows for placing suitable constraints on the classes involved in the generalization:
Example:

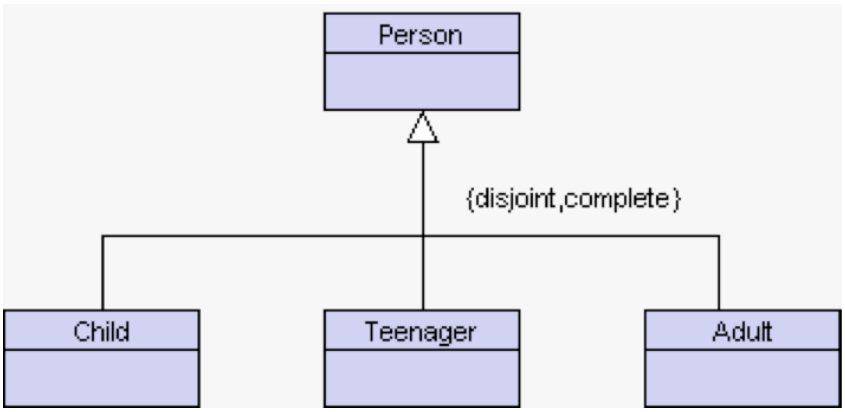


Generalization (cont.3)

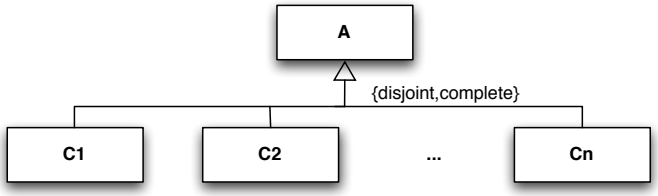
The most notable and used constraints are disjointness and completeness:

- disjointness asserts that different subclasses cannot have common instances (i.e., an object cannot be at the same time instance of two disjoint subclasses).
- completeness (aka "covering") asserts that every instances of the superclass is also an instance of at least one of the subclasses.

Example:



Generalization: formalization



ISA:

$$\forall x.\ C_i(x) \supset C(x), \qquad \text{for } i = 1, \ldots, n$$

Disjointness:

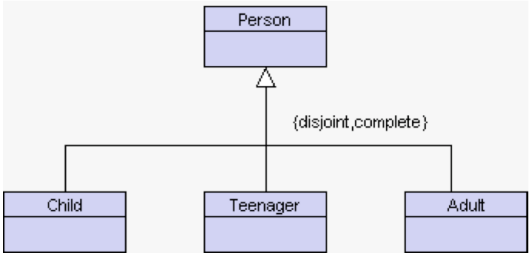
$$\forall x.\ C_i(x) \supset \neg C_j(x), \qquad \text{for } i \neq j$$

Completeness:

$$\forall x.\ C(x) \supset \bigvee_{i=1}^{n} C_i(x)$$

Generalization: formalization (cont.)

Example:

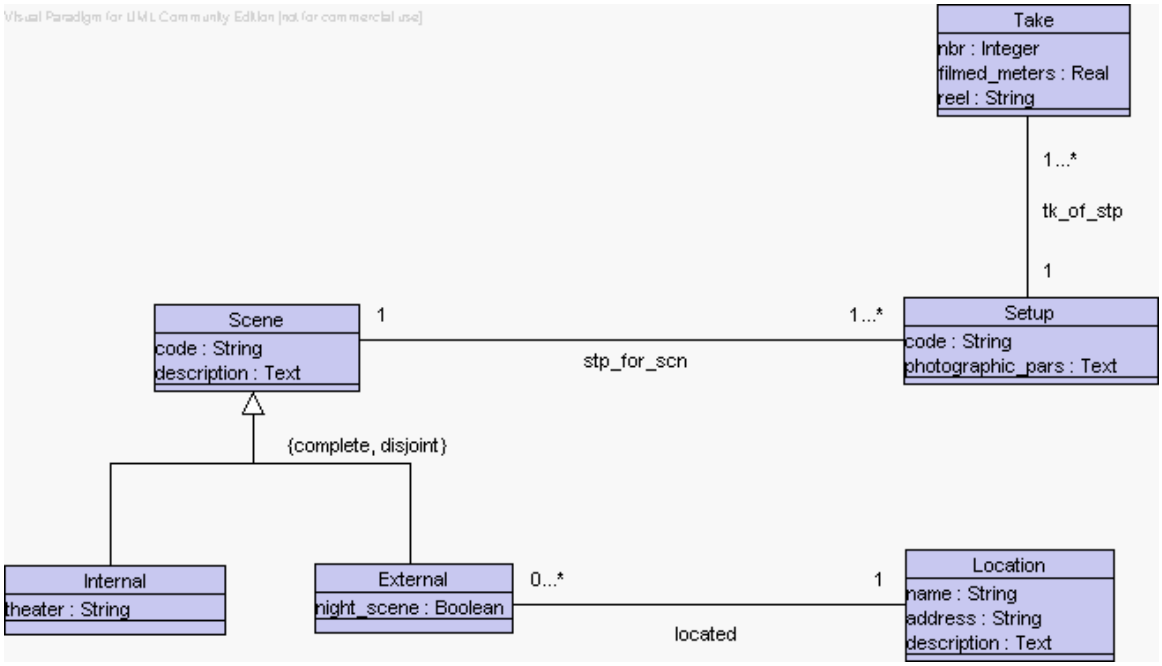


$\forall x.\ Child(x) \supset Person(x)$
$\forall x.\ Teenager(x) \supset Person(x)$
$\forall x.\ Adult(x) \supset Person(x)$

$\forall x.\ Child(x) \supset \neg Teenager(x)$
$\forall x.\ Child(x) \supset \neg Adult(x)$
$\forall x.\ Teenager(x) \supset \neg Adult(x)$

$\forall x.\ Person(x) \supset (Child(x) \vee Teenager(x) \vee Adult(x))$

In our example ...



In our example ...

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stp\_for\_scn(x, y)$, $ck\_of\_stp(x, y)$, $located(x, y)$, ...
Axioms:

$\forall x, y.\ code_{Scene}(x, y) \supset Scene(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Scene(x) \wedge Text(y)$

$\forall x, y.\ code_{Setup}(x, y) \supset Setup(x) \wedge String(y)$
$\forall x, y.\ photographic\_pars(x, y) \supset Setup(x) \wedge Text(y)$

$\forall x, y.\ nbr(x, y) \supset Take(x) \wedge Integer(y)$
$\forall x, y.\ filmed\_meters(x, y) \supset Take(x) \wedge Real(y)$
$\forall x, y.\ reel(x, y) \supset Take(x) \wedge String(y)$

$\forall x, y.\ theater(x, y) \supset Internal(x) \wedge String(y)$

$\forall x, y.\ night\_scene(x, y) \supset External(x) \wedge Boolean(y)$

$\forall x, y.\ name(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ address(x, y) \supset Location(x) \wedge String(y)$
$\forall x, y.\ description(x, y) \supset Location(x) \wedge Text(y)$

$\forall x.\ Scene(x) \supset (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
. . .

$\forall x, y.\ stp\_for\_scn(x, y) \supset Setup(x) \wedge Scene(y)$
$\forall x, y.\ tk\_of\_stp(x, y) \supset Take(x) \wedge Setup(y)$
$\forall x, y.\ located(x, y) \supset External(x) \wedge Location(y)$

$\forall x.\ Setup(x) \supset 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\ Scene(y) \supset 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\ Take(x) \supset 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\ Setup(y) \supset 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\ External(x) \supset 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\ Internal(x) \supset Scene(x)$
$\forall x.\ External(x) \supset Scene(x)$
$\forall x.\ Internal(x) \supset \neg External(x)$
$\forall x.\ Scene(x) \supset Internal(x) \vee External(x)$

Association classes

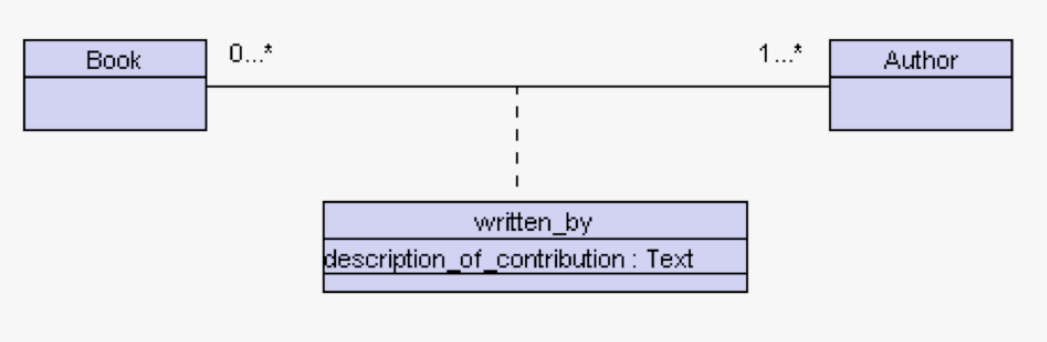
Sometimes we may want to assert properties of associations. In UML to do so we resort to Association Classes.

That is, we associate to an association a class whose instances are in bijection with the tuples of the association.

Then we use the association class exactly as a UML class (modeling local and non-local properties).

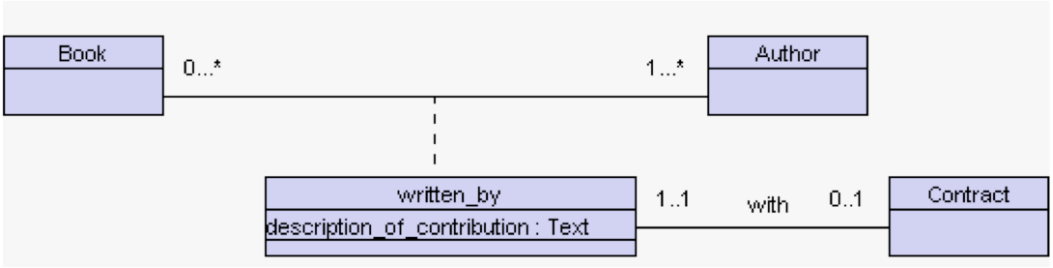
Association classes (cont.1)

Example:

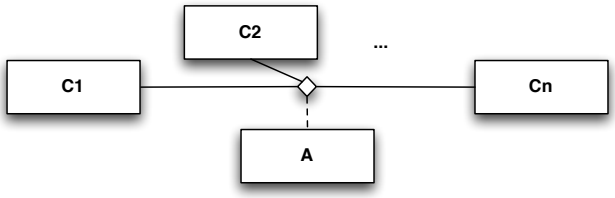


Association classes (cont.2)

Example:



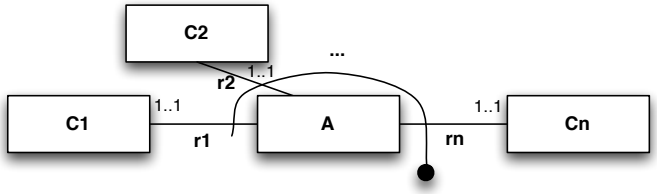
Association classes: formalization



The process of putting in correspondence objects of a class (the association class) with tuples in an association is formally described as reification.
That is:

- we introduce a unary predicate A for the association class A
- we introduce n new binary predicates $r_1, \ldots, r_n$, one for each of the components of the association
- we introduce suitable assertions so that objects in the extension of unary-predicate A are in bijection with tuples in n-ary association A.

Association classes: formalization (cont.1)



FOL Assertions needed for stating bijection between association class and association:

$$\forall x, y.\ r_i(x, y) \supset A(x) \wedge C_i(y), \qquad \text{for } i = 1, \ldots, n$$

$$\forall x.\ A(x) \supset \exists y.\ r_i(x, y), \qquad \text{for } i = 1, \ldots, n$$

$$\forall x, y, y'.\ r_i(x, y) \wedge r_i(x, y') \supset y = y', \qquad \text{for } i = 1, \ldots, n$$

$$\forall y_1, \ldots, y_n, x, x'.\ \textstyle\bigwedge_{i=1}^{n}(r_i(x, y_i) \wedge r_i(x', y_i)) \supset x = x'$$

Association classes: formalization (cont.2)

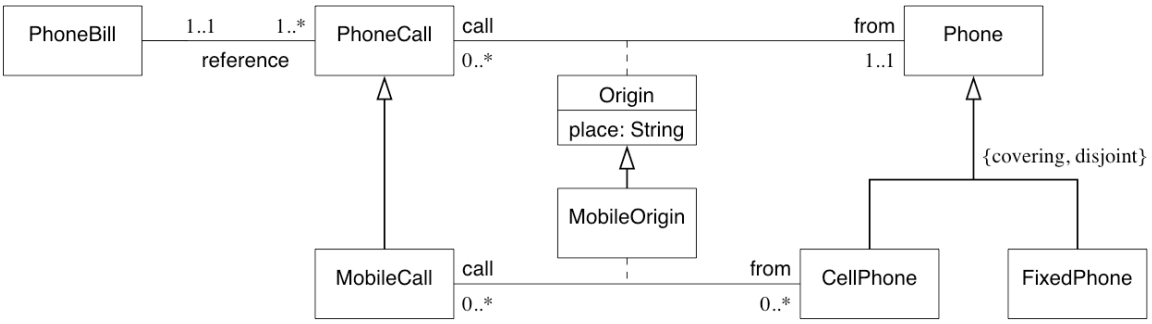
Example:

$\forall x, y.\ rw_1(x, y) \supset written\_by(x) \wedge Book(y)$
$\forall x, y.\ rw_2(x, y) \supset written\_by(x) \wedge Author(y)$

$\forall x.\ written\_by(x) \supset \exists y.\ rw_1(x, y)$
$\forall x.\ written\_by(x) \supset \exists y.\ rw_2(x, y)$
$\forall x, y, y'.\ rw_1(x, y) \wedge rw_1(x, y') \supset y = y'$
$\forall x, y, y'.\ rw_2(x, y) \wedge rw_2(x, y') \supset y = y'$

$\forall x, x', y_1, y_2.\ rw_1(x, y_1) \wedge rw_1(x', y_1) \wedge rw_2(x, y_2) \wedge rw_2(x', y_2) \supset x = x'$

Exercise



Write the diagram in FOL.

Exercise: solution

$\forall x, y.\ place(x, y) \supset Origin(x) \wedge String(x)$
$\forall x.\ Origin(x) \supset 1 \leq \sharp\{y \mid place(x, y)\} \leq 1$

$\forall x, y.\ call(x, y) \wedge \supset Origin(x) \wedge PhoneCall(y)$
$\forall x, y.\ from(x, y) \supset Origin(x) \wedge Phone(y)$
$\forall x.\ Origin(x) \supset \exists y.\ call(x, y)$
$\forall x.\ Origin(x) \supset \exists y.\ from(x, y)$
$\forall x, y, y'.\ call(x, y) \wedge call(x, y') \supset y = y'$
$\forall x, y, y'.\ from(x, y) \wedge from(x, y') \supset y = y'$
$\forall x, x', y_1, y_2.\ call(x, y_1) \wedge call(x', y_1) \wedge from(x, y_2) \wedge from(x', y_2) \supset x = x'$
$\forall x.\ PhoneCall(x) \supset 1 \leq \sharp\{z \mid call(z, x)\} \leq 1$

$\forall x.\ MobileOrigin(x) \supset Origin(x)$
$\forall x, y.\ MobileOrigin(x) \wedge call(x, y) \supset MobileCall(y)$
$\forall x, y.\ MobileOrigin(x) \wedge from(x, y) \supset CellPhone(y)$

$\forall x, y.\ reference(x, y) \supset PhoneBill(x) \wedge PhoneCall(y)$
$\forall x.\ PhoneBill(x) \supset 1 \leq \sharp\{y \mid reference(x, y)\}$
$\forall y.\ PhoneCall(y) \supset 1 \leq \sharp\{x \mid reference(x, y)\} \leq 1$

$\forall x.\ MobileCall(x) \supset PhoneCall(x)$
$\forall x.\ CellPhone(x) \supset Phone(x)$
$\forall x.\ FixedPhone(x) \supset Phone(x)$
$\forall x.\ CellPhone(x) \supset \neg FixedPhone(x)$
$\forall x.\ Phone(x) \supset CellPhone(x) \vee FixedPhone(x)$

Exercise: another solution

$\forall x, y.\ place(x, y) \supset Origin(x) \wedge String(x)$
$\forall x.\ Origin(x) \supset 1 \leq \sharp\{y \mid place(x, y)\} \leq 1$

$\forall x, y.\ call(x, y) \wedge \supset Origin(x) \wedge PhoneCall(y)$
$\forall x, y.\ from(x, y) \supset Origin(x) \wedge Phone(y)$
$\forall x.\ Origin(x) \supset \exists y.\ call(x, y)$
$\forall x.\ Origin(x) \supset \exists y.\ from(x, y)$
$\forall x, y, y'.\ call(x, y) \wedge call(x, y') \supset y = y'$
$\forall x, y, y'.\ from(x, y) \wedge from(x, y') \supset y = y'$
$\forall x, x', y_1, y_2.\ call(x, y_1) \wedge call(x', y_1) \wedge from(x, y_2) \wedge from(x', y_2) \supset x = x'$
$\forall x.\ PhoneCall(x) \supset 1 \leq \sharp\{z \mid call(z, x)\} \leq 1$

$\forall x.\ MobileOrigin(x) \supset Origin(x)$
$\forall x, y.\ call_{mo}(x, y) \supset call(x, y)$
$\forall x, y.\ from_{mo}(x, y) \supset from(x, y)$
$\forall x, y.\ call_{mo}(x, y) \supset MobileOrigin(x) \wedge MobileCall(y)$
$\forall x, y.\ from_{mo}(x, y) \supset MobileOrigin(x) \wedge CellPhone(y)$
$\forall x.\ MobileOrigin(x) \supset \exists y.\ call_{mo}(x, y)$
$\forall x.\ MobileOrigin(x) \supset \exists y.\ from_{mo}(x, y)$
$\forall x, y, y'.\ call_{mo}(x, y) \wedge call_{mo}(x, y') \supset y = y'$ these are redundant
$\forall x, y, y'.\ from_{mo}(x, y) \wedge from_{mo}(x, y') \supset y = y'$
$\forall x, x', y_1, y_2.\ call_{mo}(x, y_1) \wedge call_{mo}(x', y_1) \wedge from_{mo}(x, y_2) \wedge from_{mo}(x', y_2) \supset x = x'$

$\forall x, y.\ reference(x, y) \supset PhoneBill(x) \wedge PhoneCall(y)$
$\forall x.\ PhoneBill(x) \supset 1 \leq \sharp\{y \mid reference(x, y)\}$
$\forall y.\ PhoneCall(x) \supset 1 \leq \sharp\{x \mid reference(x, y)\} \leq 1$

$\forall x.\ MobileCall(x) \supset PhoneCall(x)$
$\forall x.\ CellPhone(x) \supset Phone(x)$
$\forall x.\ FixedPhone(x) \supset Phone(x)$
$\forall x.\ CellPhone(x) \supset \neg FixedPhone(x)$
$\forall x.\ Phone(x) \supset CellPhone(x) \vee FixedPhone(x)$

Other constraints

UML allows for other forms of constraints, such as specifying class identifiers, functional dependencies for associations, across ISA typing, etc.
Example:
In our "Phone Calls" example, we may want to add the following constraints:

*MobileCall*s must have as *Origin* a *CellPhone*
*CellPhone*s can be the *Origin* of *MobileCall* only

We can express these in FOL:

$$\forall x, y.\ \mathit{MobileCall}(x_c) \land \mathit{Origin}(z) \land \mathit{call}(z, x_c) \land \mathit{from}(z, x_p) \supset \mathit{CellPhone}(x_p)$$
$$\forall x, y.\ \mathit{CellPhone}(x_p) \land \mathit{Origin}(z) \land \mathit{call}(z, x_c) \land \mathit{from}(z, x_p) \supset \mathit{MobileCall}(x_c)$$

Actually we can even express them by suitably modifying the diagram by adding ad-hoc classes (loosing readability) – see later.

Other constraints (cont.)

- More generally, one can write full FOL assertions as constraints by using the UML Object Constraint Language (OCL).
- However excessive use of OCL constraints is considered an indication of bad design, since it moves the semantics from the diagram to the OCL constraints, and this may compromise the understandability of the diagram.
- From a formal point of view the use of OCL constraints makes reasoning on a class diagram undecidable.

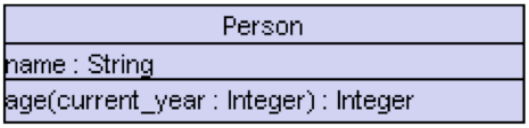
In this course, we do not deal with general OCL constraints.

Class Operations

Apart from attributes, classes may include operations.
An operation of a class is a function from the objects of the class to which the operation is associated and possibly additional parameters, to objects or values.
An operation definition for a class C has the form

$$f(P_1, \ldots, P_m) : R$$

where f is the name of the operation, $P_1, \ldots, P_m$ are the types of the m parameters, and R is the type of the result.
Example:



Observe that only the signature (i.e., the name of the function and the number and the types of parameters, where the object of invocation is an implicit parameter) and the return type of the function is represented in UML, not the actual definition.

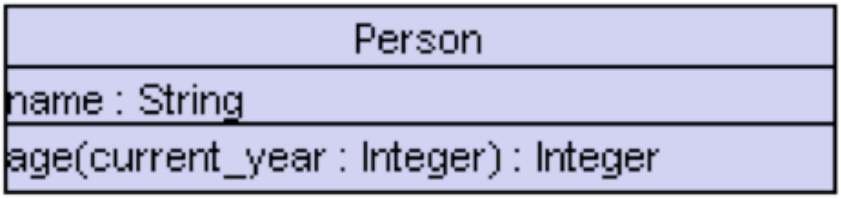
Class Operations: formalization

Formally, such an operation corresponds to an $(1 + m + 1)$-ary predicate $f_{P_1,\ldots,P_m}$, in which the first argument represents the object of invocation, the next m arguments represent the additional parameters, and the last argument represents the result.
Observe that the name of the predicate depends on the whole signature, i.e., it includes the types of the parameters.
The predicate $f_{P_1,\ldots,P_m}$ has to satisfy the following FOL assertions:

$$\forall x, p_1, \ldots, p_m, r.\ f_{P_1,\ldots,P_m}(x, p_1, \ldots, p_m, r) \supset \bigwedge_{i=1}^{m} P_i(p_i)$$
$$\forall x, p_1, \ldots, p_m, r, r'.\ f_{P_1,\ldots,P_m}(x, p_1, \ldots, p_m, r) \wedge f_{P_1,\ldots,P_m}(x, p_1, \ldots, p_m, r') \supset r = r'$$

$$\forall x, p_1, \ldots, p_m, r.\ C(x) \wedge f_{P_1,\ldots,P_m}(x, p_1, \ldots, p_m, r) \supset R(r)$$

Class Operations: formalization



$\forall x, p, r.\ age_{Integer}(x, p, r) \supset Integer(p)$
$\forall x, p, r, r'.\ age_{Integer}(x, p, r) \wedge age_{Integer}(x, p, r') \supset r = r'$
$\forall x, p, r.\ Person(x) \wedge age_{Integer}(x, p, r) \supset Integer(r)$

Class Operations: discussion

- UML allows for the overloading of operations, which takes place between two or more functions having the same name but different signatures.
- Overriding takes place when two operations have the same signature, but behave in different ways. In UML class diagrams for the conceptual perspective, where the bodies of operations are not specified, overriding may only show up as a restriction on the type of the result.
- The above formalization of operations correctly captures both overloading and overriding.

Some common assumptions ...

- Sometimes, in UML class diagrams, it is assumed that all classes not in the same hierarchy are a priori disjoint.
- Here we do not force this assumption; instead we allow two classes to have common instances.
- When needed, disjointness can be enforced by means of explicit disjointness constraints.
- Similarly, we do not assume that objects in a hierarchy must belong to a single most specific class.
- Hence, two classes in a hierarchy may have common instances, even when they do not have a common subclass.
- Again, when needed, suitable covering and disjointness assertions that express the most specific class assumption can be added to a class diagram.

Current CASE tools: no reasoning

The design of UML class diagrams modeling complex real world domains is facilitated by automated CASE tools.

Currently, CASE tools support the designer with:

- user friendly graphical environment
- forms of syntax checking
- management of repositories of diagrams, generated code, etc.

But they offer no form of automated reasoning on the diagram.

Enhancing current CASE tools

The fact that UML class diagrams can be re-expressed in FOL allows for building CASE tools that go far beyond the kind of support reported above.

The designer can use the FOL formalization to formally check relevant properties of class diagrams so as to assess the quality of the diagram according to objective quality criteria.

Forms of reasoning: class consistency

A class is consistent, if the class diagram admits an instantiation in which the class has a non-empty set of instances.
Intuitively, the class can be populated without violating the conditions imposed by the class diagram.
The inconsistency of a class may be due to a design error or due to over-constraining.
An inconsistent class weakens understandability of the diagram. It is an indication of an error.
Once detected, the designer may remove the inconsistency by relaxing some conditions, or by deleting the class.

Forms of reasoning: class consistency – formalization

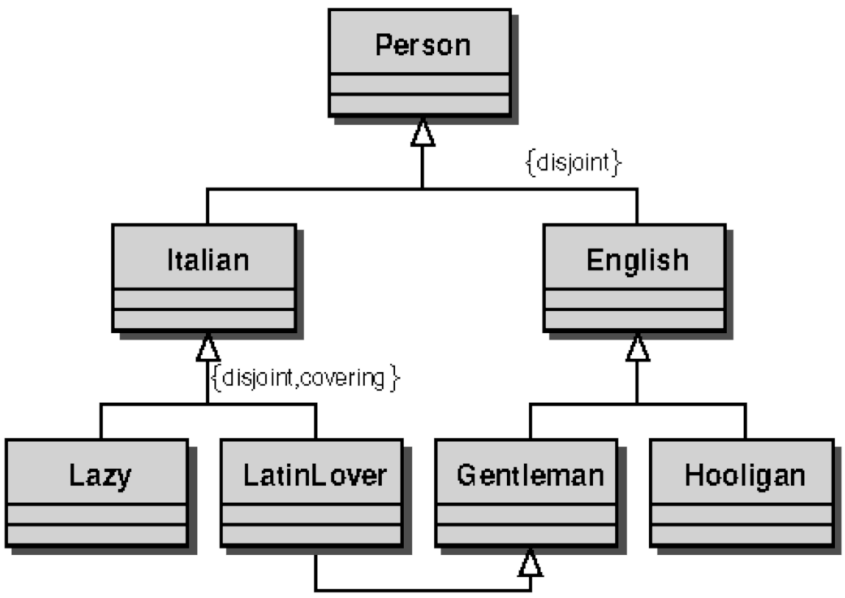
Let Γ be the set of FOL assertions corresponding to the UML Class Diagram, and $C(x)$ the predicate corresponding to a class C of the diagram.
Then C is consistent iff

$$\Gamma \not\models \forall x.\ C(x) \supset \mathit{false}$$

i.e., there must exist a model of Γ were the extension of $C(x)$ is not the empty set.

Note: FOL reasoning task: logical implication

Example (by E. Franconi)



$$\Gamma \models \forall x.\ \mathit{LatinLover}(x) \supset \mathit{false}$$

Forms of reasoning: whole diagram consistency

A class diagram is consistent, if it admits an instantiation, i.e., if its classes can be populated without violating any of the conditions imposed by the diagram.
Note the empty extension for all classes is not considered an admissible instantiation.
Then, the diagram is consistent if at least one of its classes admits a nonempty extension.

When the diagram is not consistent, the definitions altogether are contradictory, since they do not allow any class to be populated.

Forms of reasoning: whole diagram consistency – formalization

Let Γ be the set of FOL assertions corresponding to the UML Class Diagram.
Then, the diagram is consistent iff

Γ is satisfiable

i.e., Γ admits at least a model (remember that FOL models cannot be empty).

Note: FOL reasoning task: satisfiability – reducible to logical implication:

$\Gamma \not\models \mathit{false}$.

Forms of reasoning: class subsumption

A class C_1 subsumes a class C_2, if the class diagram implies that C_1 is a generalization of C_2.
Note that this means that C_2 inherits all properties of C_1 (cf. ISA).

This suggests the possible omission of an explicit generalization. Alternatively, if not all instances of the more specific class are supposed to be instances of the more general class, then there is an error in the diagram.

Note that class subsumption is also the basis for a classification of all the classes in a diagram. Indeed a classification is obtained by checking subsumption between all classes in the diagram.

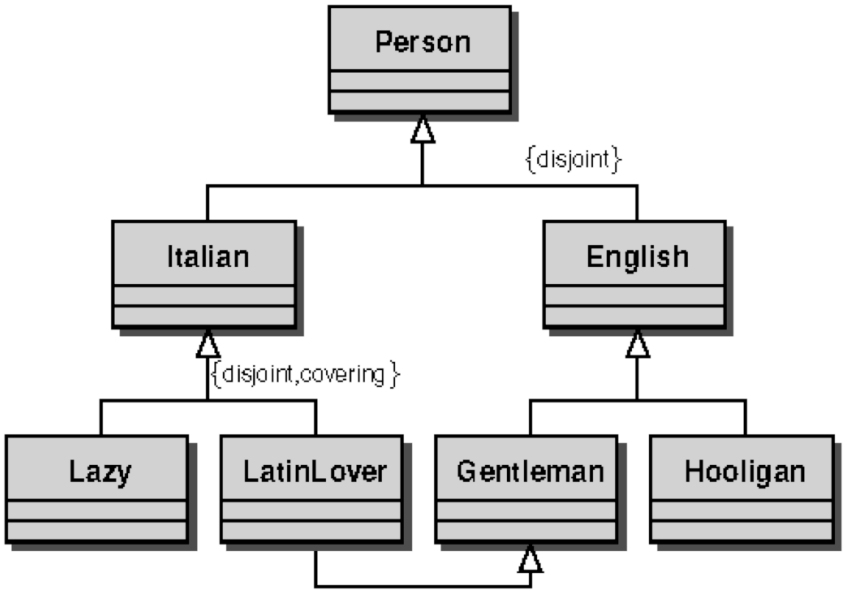
Forms of reasoning: class subsumption – formalization

Let Γ be the set of FOL assertions corresponding to the UML Class Diagram, and $C_1(x), C_2(x)$ the predicates corresponding to the class C_1, C_2 of the diagram.
Then C_1 subsumes C_2 (or C_2 is subsumed by C_1) iff

$$\Gamma \models \forall x.\ C_2(x) \supset C_1(x)$$

Note: FOL reasoning task: logical implication

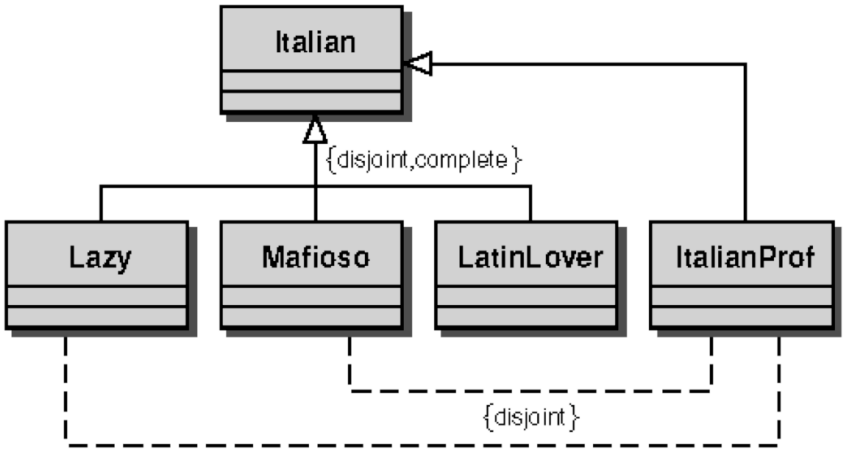
Example



$\Gamma \models \forall x.\ \mathit{LatinLover}(x) \supset \mathit{false}$

$\Gamma \models \forall x.\ \mathit{Italian}(x) \supset \mathit{Lazy}(x)$

Another Example (by E. Franconi)



(reasoning by cases)

$\Gamma \models \forall x.\ \mathit{ItalianProf}(x) \supset \mathit{LatinLover}(x)$

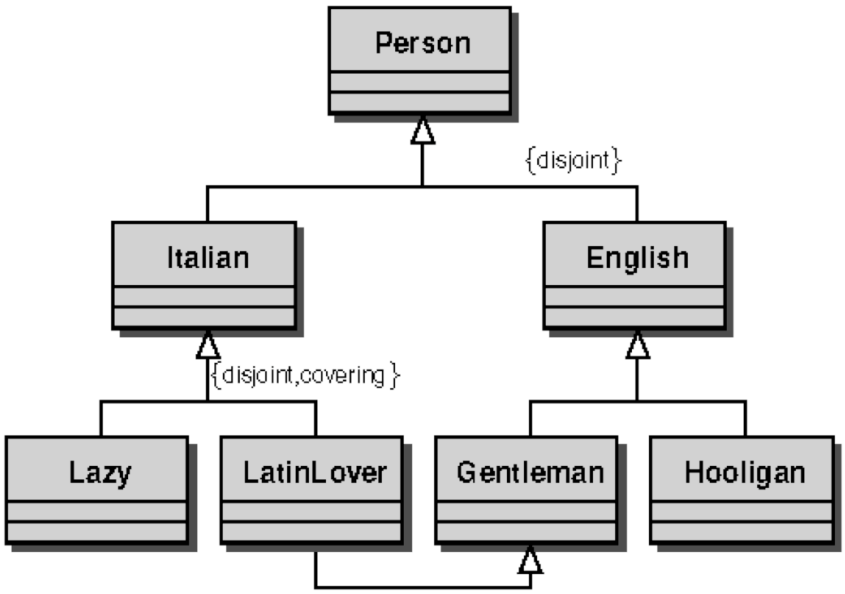
Forms of reasoning: class equivalence

Two classes are equivalent if they denote the same set of instances whenever the conditions imposed by the class diagram are satisfied.
In other words the two classes subsume each other.

If two classes are equivalent then one of them is redundant.

Determining equivalence of two classes allows for their merging, thus reducing the complexity of the diagram.

Example



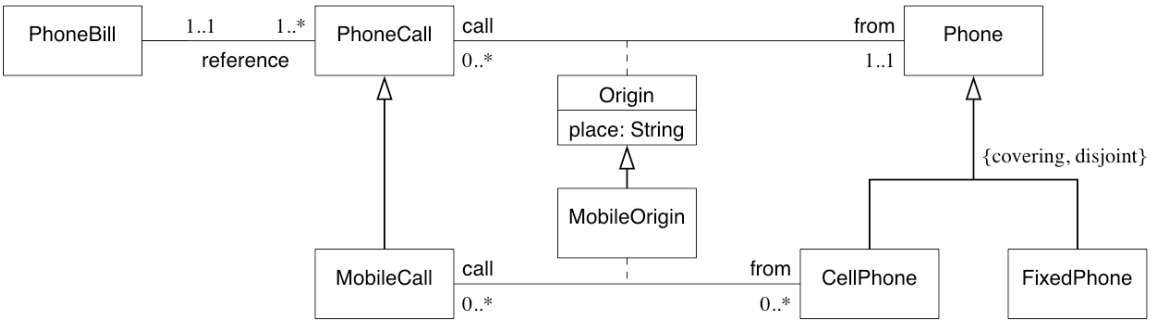
$\Gamma \models \forall x.\ \mathit{LatinLover}(x) \supset \mathit{false}$
$\Gamma \models \forall x.\ \mathit{Italian}(x) \supset \mathit{Lazy}(x)$
$\Gamma \models \forall x.\ \mathit{Lazy}(x) \equiv \mathit{Italian}(x)$

Forms of reasoning: refinements of properties

The properties of various classes and associations may interact to yield stricter multiplicities or typing than those explicitly specified in the diagram.

Detecting such cases allows the designer for refining the class diagram by making such properties explicit, thus enhancing the readability of the diagram.

Example



$$\forall x_c.\ \mathit{MobileCall}(x_c) \supset \sharp\{z \mid \mathit{MobileOrigin}(z) \wedge \mathit{call}(z, x_c)\} \leq 1$$

Forms of reasoning: implicit consequence

More generally ...
A property $\mathcal{P}$ is an (implicit) consequence of a class diagram if $\mathcal{P}$ holds whenever all conditions imposed by the diagram are satisfied.

Determining implicit consequences is useful:

- to reduce the complexity of the diagram by removing those parts that implicitly follow from other ones
- to make properties explicit, for enhancing readability.

Note that all the above reasoning tasks can be seen as special cases of implicit consequences!!!

Forms of reasoning: implicit consequence

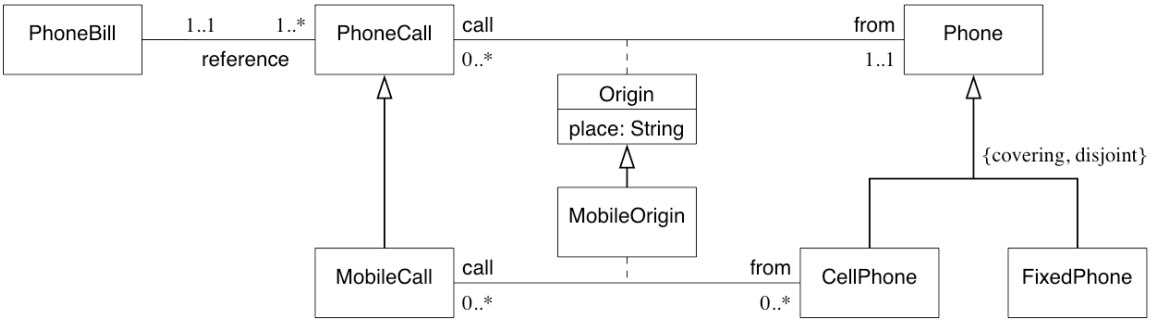
Let Γ be the set of FOL assertions corresponding to the UML Class Diagram, and $\mathcal{P}$ (the formalization in FOL of) the property of interest Then $\mathcal{P}$ is an implicit consequence iff

$$\Gamma \models \mathcal{P}$$

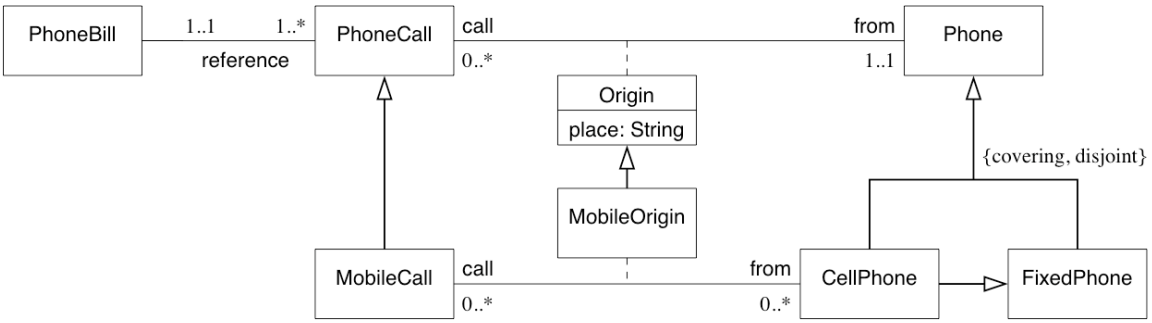
i.e., the property $\mathcal{P}$ holds in every model of Γ.

Note: FOL reasoning task: logical implication

Exercise

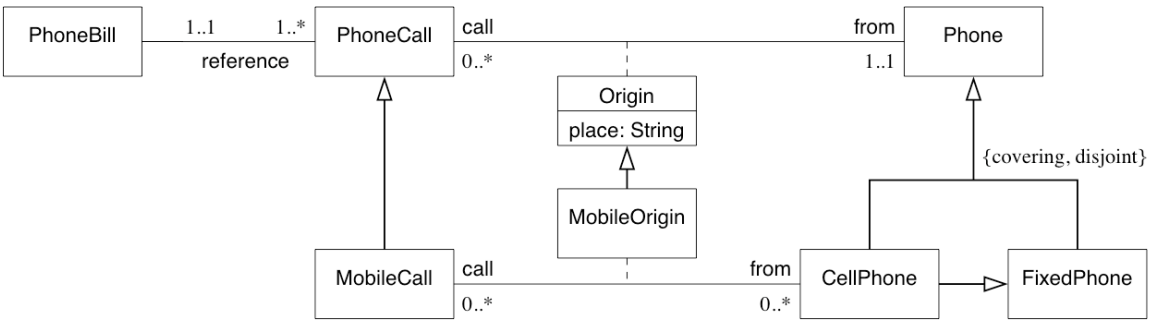


Exercise



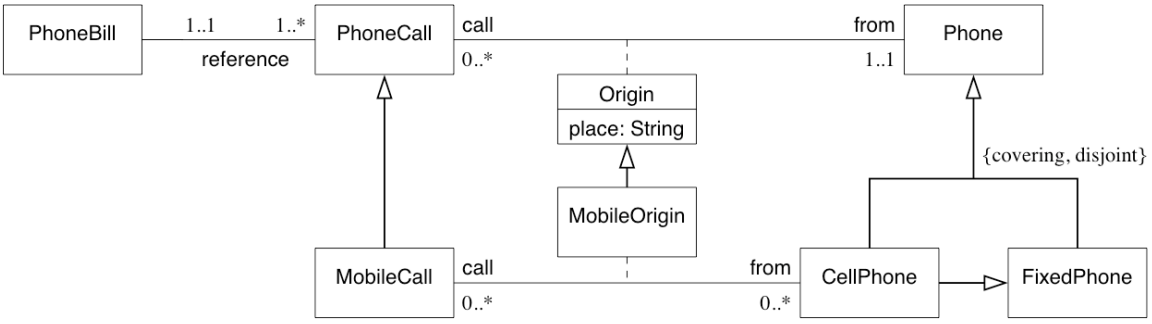
Which implicit consequences hold?

Exercise: solution



$\Gamma \models \forall x.\ \mathit{CellPhone}(x) \supset \mathit{false}$
$\Gamma \models \forall x.\ \mathit{MobileOrigin}(x) \supset \mathit{false}$
$\Gamma \models \forall x.\ \mathit{Phone}(x) \supset \mathit{FixedPhone}(x)$
$\Gamma \models \forall x.\ \mathit{Phone}(x) \equiv \mathit{FixedPhone}(x)$

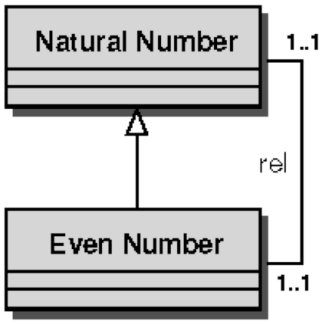
Exercise: solution



$+$ additional constraint
$\forall x, y.\ \mathit{MobileCall}(x_c) \wedge \mathit{Origin}(z) \wedge \mathit{call}(z, x_c) \wedge \mathit{from}(z, x_p) \supset \mathit{CellPhone}(x_p)$

$\Gamma \models \forall x.\ \mathit{CellPhone}(x) \supset \mathit{false}$
$\Gamma \models \forall x.\ \mathit{MobileOrigin}(x) \supset \mathit{false}$
$\Gamma \models \forall x.\ \mathit{Phone}(x) \supset \mathit{FixedPhone}(x)$
$\Gamma \models \forall x.\ \mathit{Phone}(x) \equiv \mathit{FixedPhone}(x)$
$\Gamma \models \forall x.\ \mathit{MobileCall}(x) \supset \mathit{false}$

Unrestricted vs. finite model reasoning



The classes *NaturalNumber* and *EvenNumber* are in bijection.
this implies: "the classes *NaturalNumber* and *EvenNumber* contains the same number of objects"

Unrestricted vs. finite model reasoning (cont.)

- If the domain is finite then:

$$\forall x.\ NaturalNumber(x) \supset EvenNumber(x)$$

- if the domain is infinite we do not get the subsumption!

Finite model reasoning: look only at models with finite domains (very interesting for Databases).

In UML Class Diagrams finite model reasoning is different form unrestricted reasoning.

Main questions

The examples of reasoning we have seen could be easily carried out on intuitive grounds.

More importantly, since they correspond to logical reasoning tasks on the FOL theory corresponding to an UML Class Diagram they can be formalized and formally verified.

Two main question remain open ...

Main questions (cont.)

...

- *Can we develop sound, complete, and* **terminating** *reasoning procedures for reasoning on UML Class Diagrams?*
  To answer this question we will look at Description Logics and show that reasoning on UML Class Diagrams can be done in EXPTIME (and actually carried out by current DLs-based systems such as FACT++, PELLET or RACER-PRO).
- *How hard is it to reason on UML Class Diagrams in general?*
  We will show that reasoning on UML Class Diagrams is in fact EXPTIME-hard!
  This is somewhat surprising, since UML Class Diagrams are so widely used and yet reasoning on them (and hence fully understand the implication the may give rise to) is not easy at all in general.

Note that these results hold for Entity-Relationship Schemas as well!!!