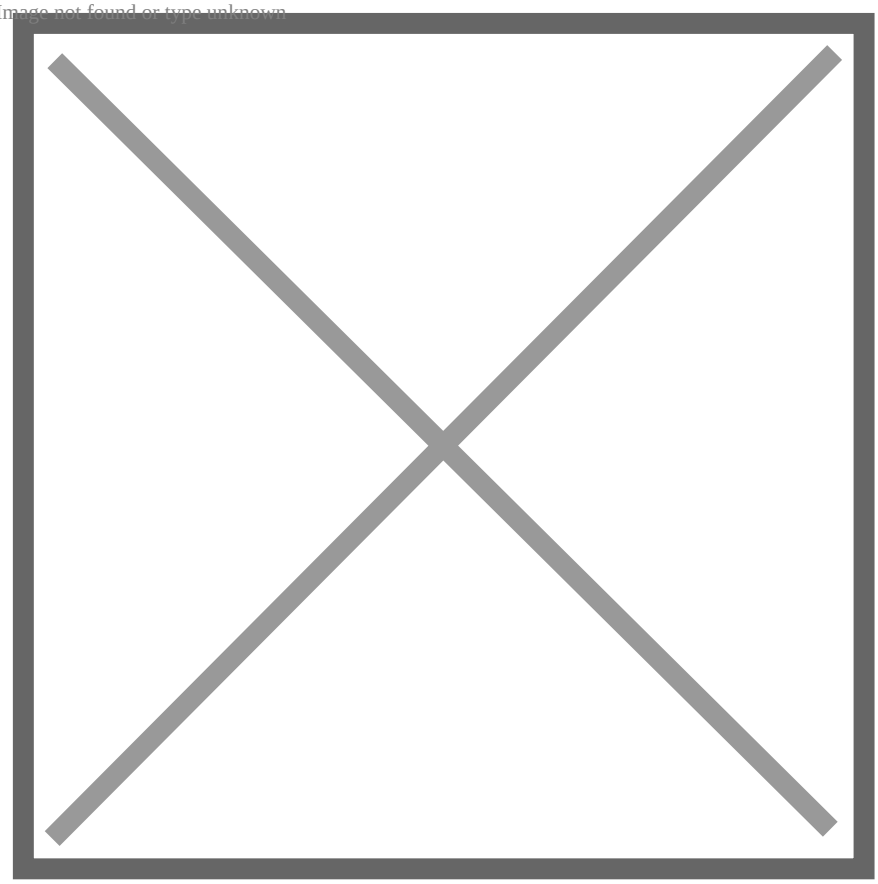
20. Representing non-tree-structured classification hierarchies

Description

The subclass mechanism as described in chapter makes it possible to represent tree-structured classification hierarchies. In practice, many classification hierarchies are in fact tree-structured, but there are of course examples of the opposite. In this chapter, we show examples of such hierarchies.

The classification of geometric figures may result in a non-tree structured hierarchy as shown below. Here the concept Square has two generalizations, Rectangle and Rhombus.

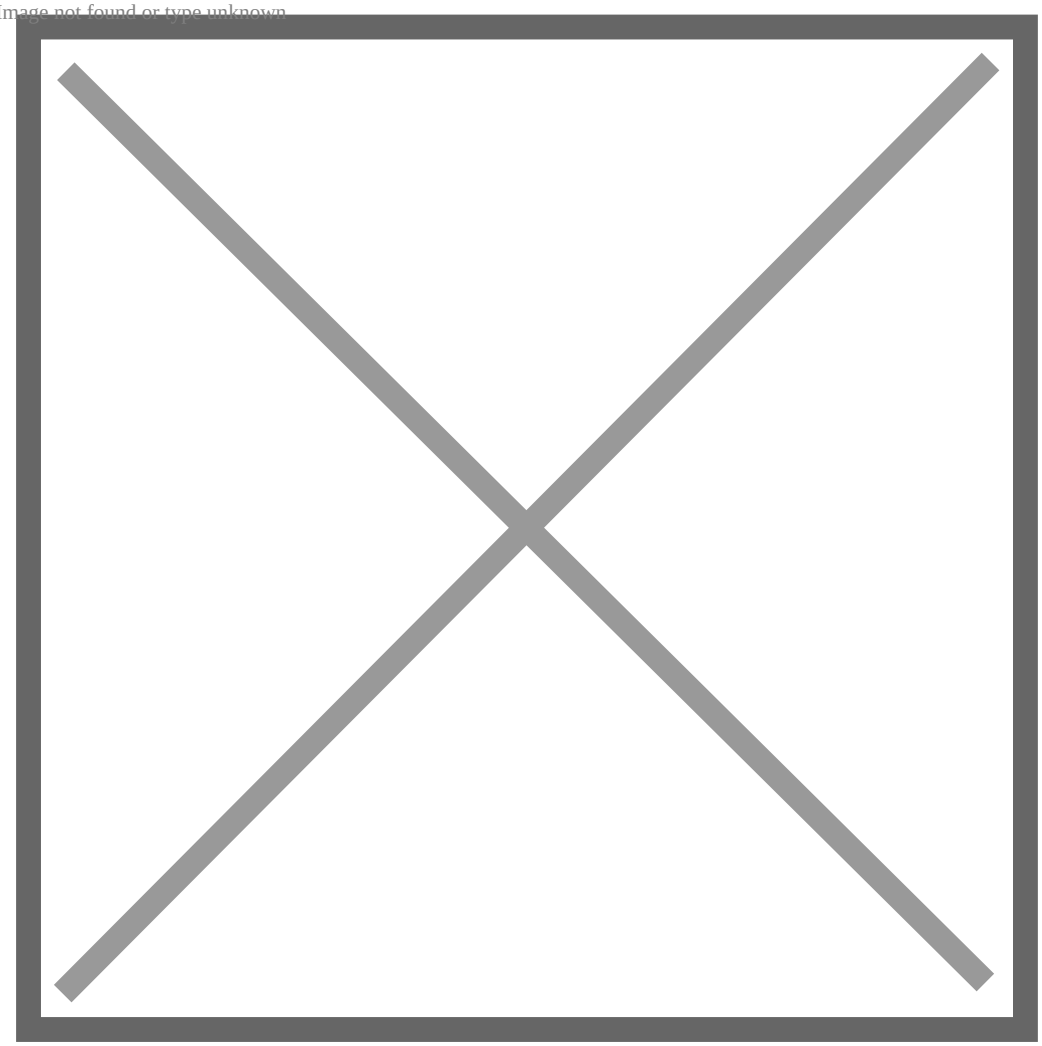
Image not found or type unknown



Non-tree Classification

There are situations where it may be desirable to classify phenomena according to independent properties. People may e.g. be classified according to their nationality and profession as shown in the next figure. This leads to two or more independent classification hierarchies of the same phenomena (in this case people).

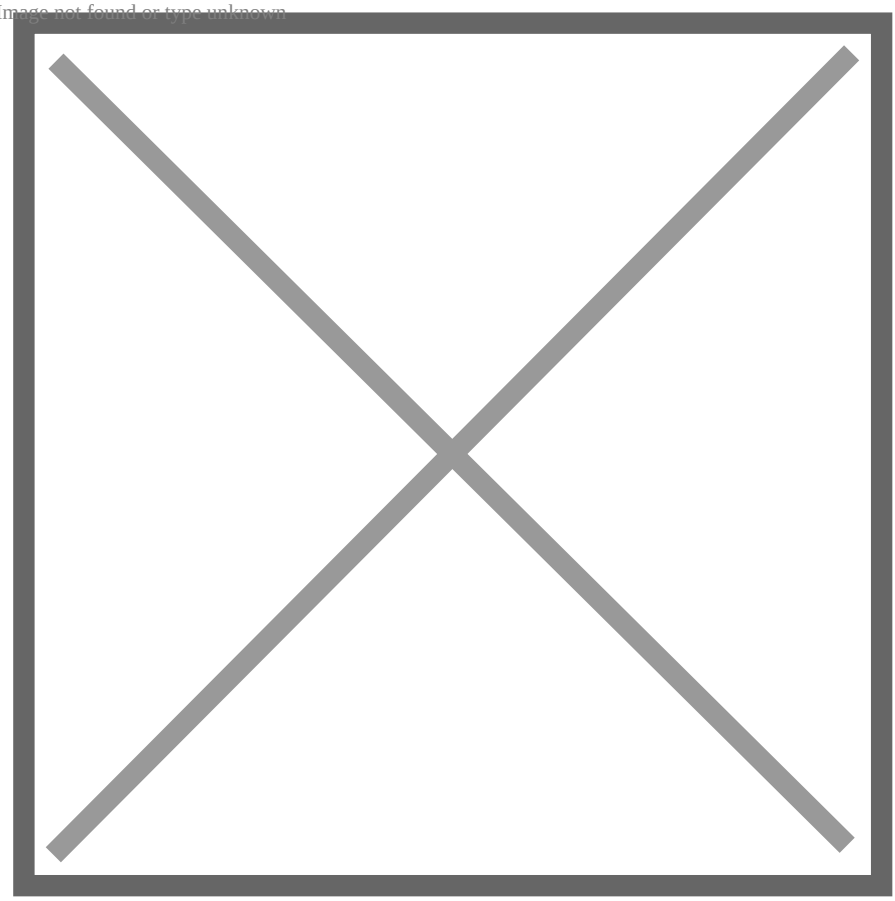
Image not found or type unknown



Independent Classification

A phenomenon may be member of different concepts over time. At one point of time, person may be a student, then a programmer and later a manager as illustrated by Fig. 10. This form of *dynamic classification* is not supported by mainstream programming languages. In some languages, like Smalltalk, it is possible to change the class pointer of an object, but this may be an unsafe way of programming and for languages with static type checking, it breaks the type checking.

Image not found or type unknown



Dynamic Classification

Language support

All languages have limitations for describing models and the modeler/programmer must be aware of these limitations. If your domain e.g. require non-tree-structured classification hierarchies and your language only support tree-structured classification, you must find a work-around and you need to document that a work-around has been made.

Aspects of a phenomena as described in chapter xxxx, may be viewed as a kind of classification of phenomena characterized by a given aspect. The representation of aspects by means of part-objects combined with nested classes is in many cases a good solution, buy may also be seen as a work-around.

Hvad siger vi om geometri eksemplet? +++ ja, det er jo ikke MI, men faktisk en mulig klassifikation. Minder meg om at vi i aspects-kapitlet måske burde have et eksemple på ren MI som fikses med aspects. De eksempler vi har er mere a la traits

For independent classification hierarchies, aspects may often be used. Given the concept of a person represented by a class Person. Nationality and profession may then be viewed as different aspects of a person and thus be represented by part-objects. +++ måske vi skulle bruge disse som eksempler på aspects?

In the case of dynamic classification, part-objects cannot be used to represent the actual aspect since such an object is the same for the life-time of the enclosing object. Here a variable reference may sometimes be used, but if the type of the aspect shall be defined using the context of the phenomena/object having the aspect, the class type of the reference must be defined as a local class. But defining the profession hierarchy as local classes of class Person limits profession to be aspects of only person. +++ local class kan vel være sub af mere generelle

Det næste er ude af tråd med vores stil da det kommenterer på andre sprog. Men det er jo et emne der fylder meget så måse det er ok det står her. Men det kan reduceres til en enkelt sætning der henviser til det kapitel vi måtte skrive om mainstream languages.

No mainstream object-oriented language (if any language at all) has a suitable mechanism for representing independent classification hierarchies, although many languages have mechanism intend for this purpose.

Some language support so-called multiple inheritance, which allows a class to have more than one superclass – examples of such languages are Eiffel, CLOS and C+. This easily leads to a combinatorial explosion of classes like NorwegianNurse, JapaneseNurse, NorwegianEngineer, etc. which in most cases gives a very tangled subclass structure.

Java has an interface-mechanism which is a simplified version of multiple inheritance. An interface is a class with only method-signatures with an empty body. A class may have one superclass but implement zero or more interfaces.

Other mechanism are so-called *traits* and *mixins*. +++ more

For a detailed description of multiple inheritance, interfaces, traits and mixing, the reader should learn to what extent such mechanism are supported by a given language to be used.