18. Aspects

We have learned that the modeling of phenomena by means of objects represents properties in terms of e.g. data-items and methods, and we have learned how classes of objects represent the corresponding domain concepts. Specialization hierarchies are expressed by a class being a subclass of another (super) class. As an example the class `Customer` from before could have been defined as a subclass of a class `Person`, and `Person` would have a `name` attribute of type String.

```
class Person:
   name: var String

class Customer: Person
   addr: var String
   email: var String
```

Altogether this model the situation that a customer *is* a person and that a customer *has* the properties represented by data-items and methods. A `Customer` object *is* a `Person` and it *has* the properties of `Person` *and* the properties defined in `Customer`. We have also seen that an object (**`obj`**) also may have a superclass.

In this section, we will show how to model the situation where phenomena in addition to be classified (and thereby represented by a class) also have some aspects. An aspect is a set of properties that may readily be modelled by data-items, methods, and computations. Many phenomena may have the same aspects, and this calls for the definition of types of aspects. As an aspect may have the same properties as an object, types of aspects may be represented by classes. It will thereby also be possible to classes an subclasses of aspects.

In most cases, classes with aspects will also be subclasses of classes from the domain, e.g. `Customer` being a subclass of `Person`. As we only have tree-structured classification by means of classes and subclasses, i.e. a class can only have one superclass, aspects come in play when a class in addition to a superclass also shall have the properties defined by one or more aspect classes.

In the following we will, by a number of examples, show how to *represent types of aspects by classes*, and to use *composition* to represent aspects of classes and objects. Composition means that an object with aspects simply has an intrinsic/part object for each aspect. An aspect object is specified as an object of an aspect class or as a singular object with an aspect class as superclass.

A class like `Customer` may typically tailor the fact that it is a `Person`, by extending possible virtual methods of class `Person`. We will have the same need for aspects. An aspect may e.g. be that of being addressable. Different objects and classes may be addressable, and each of them may have a slightly different way of being addressable. We will therefore show how to tailor a general type of aspect to different objects and classes.