

18.1 Addressable aspect

The fact that a number objects from different classes may be addressable is represented by defining a class Address:

```
class Address:
  country: var String
  town: var String
  street: var String
  streetNo: var Integer
  print:
    country.print
    town.print
    street.print
    streetNo.print
  change(newAddr: ref Address):
    country := newAddr.country
    town := newAddr.town
    street := newAddr.street
    streetNo := newAddr.streetNo
```

Address objects have the data-items country, town, street, and streetNo. The address may be changed by the change method, and it may be printed by the print method.

Customers being addressable is represented by Customer objects having an Address object:

```
class Customer:
  -" -
  addr: obj Address
```

The fact that a Customer object has an addr object as above implies that the attributes of the Address object are "kind of attributes of the Customer as well", however, they are only indirect attributes as they have to be accessed through addr, like e.g.

```
aCustomer.addr.print
```

where aCustomer is a reference to a Customer object. The same would be the case inside the class Customer:

```
class Customer: Person
  -" -
  addr: obj Address
  ...
  addr.print
  ...
```

One could argue that for Address to be an aspect of Customer, the properties of Address should be directly accessible and not via the name addr of the part object. This would have been the case if Customer had been a subclass of Address, but that is obviously a wrong way to model this situation. A customer would typically *be a type* of e.g. person and it will *have* an address. From a

modeling approach, a customer *is not* an address, but a customer *has* an address.

The `addr` object is a simple means to represent an aspect like address, as we use an already known mechanism of an intrinsic object (**obj**). However, it should be possible not only to apply a given aspect like `Address` to any class of object, but also to tailor the aspect to the class to which it is applied.

The `print` method of `Address` simply prints the address. If we also would like to print the name of the customer, then we can not define `addr` simply as an `Address` object.

The usual way of making the `print` method so that it can be tailored in different `Address` aspects of different classes or objects is to define it as a virtual method:

```
class Address:
  -:-
  print:<
    inner(print)  -- whatever to print from the class/object that has
                  -- an Address aspect
    country.print
    town.print
    street.print
    streetNo.print
```

Instead of representing an `Address` aspect just by an `Address` object, the aspect is then rather represented by a singular intrinsic object with `Address` as a superclass and with an extension of the virtual `print`:

```
class Customer: Person
  -"-
  addr: obj Address
  print::
    name.print
```

Because `print` is extended in the context of `Customer`, i.e. nested in the description of `Customer`, the `name` data-item of `Customer` is visible and can therefore be printed in the extension of `print`.

Aspects may typically be applied to more than one class. Here it is applied to class `Bank`:

```
class Bank:
  name: var String
  addr: obj Address
  print::
    name.print
```

In this case `print` will print the name of the bank.

Suppose that we have a class `BankCustomer` that is subclass of `Customer`:

```
class BankCustomer: Customer
  bankName: var String  -- the name of the bank
                  -- where the bank customer is a customer
```

Suppose that we still want to represent the `Address` as an aspect of `Customer`, *and* as an aspect of

class `BankCustomer`, but still so that the aspect is tailorable in both `Customer`, in `BankCustomer`, and in further subclasses of `BankCustomer`. Then the above definition of `addr` will not do; we will then have to define the class of `addr` by a virtual class `AddrType`:

```
class Customer: Person
  class AddrType:< Address
    print::<
      inner(print)
      name.print

  addr: obj AddrType
```

```
class BankCustomer: Customer
  bankName: var String
  class AddrType::<
    print::<
      inner(print)
      bankName.print
```

As the virtual class `AddrType` is defined defined in the context of `Customer`, the virtual `print` has access to the name of `Customer`, so `name.print` will print the name of the `Customer`. Note that the `print` method of `AddrType` is an extension of `print` in `Address`, but still virtual (`::<`) so that it can be further extended in the extension of `AddrType` in `BankCustomer`. This extension of `AddrType` is defined in the context of `BankCustomer`, and `bankName` is therefore visible in the further extension of `print`.