18.2 Interface-like aspects

The addressable aspect in the previous section was an example of a general aspect with both data-items and methods.

The following example illustrates aspects that only have methods. In other languages this kind of aspects are defined by a special *interface* definition, but we will define also these kinds of aspects by means of classes. Usually interface only have method signatures, i.e. name, parameters and return type, while aspects used for interface may have methods with more than signatures.

We start out by the simple aspect of being printable:

```
class printable:
   print:<
      inner(print)
```

Different classes of objects may have a `printable` aspect. Here we define that `Person` objects are printable:

```
class Person(name: var String):
   asPrintable: obj Printable
      print::
         name.print
```

Because the description of `asPrintable` is nested within `Person`, the data item `name` of `Person` is visible, and it is therefore possible to extend print to print the `name`.

A `Person` object are printed like this:

```
JohnSmith: ref Person
JohnSmith := Person("John Smith")
...
JohnSmith.asPrintable.print
```

This will not surprisingly print “John Smith”. A more interesting example is the case where we have a set of `Person` objects and apply `asPrintable.print` to all of objects in this set:

```
persons: obj Set(Person)
-- insert Person objects in this set, e.g.:
persons.insert(Person("John Smith"))
persons.insert(Person("Liza Jones"))
persons.insert(Person("Mary Pole"))
...
persons.scan
   current.asPrintable.print
```

The same effect *could* have been obtained by defining `Person` as a subclass of `Printable`, but that would classify `Person` to be `Printable`, and that is *not* what we want to express. Subclassing works well for classification, but for aspects we typically would like to have that a class like `Person` can have more than one aspect.

Suppose we also have the notion of being movable:

```
class Movable:
   move(newAddr: ref Address):
      inner(move)
```

then `Person` could also be movable:

```
class Address:
   -"-
   print:<
      -"-

class Person(name: var String):
   addr: obj Address

   asPrintable: obj Printable
      print::
         name.print
         address.print

   asMovable: obj Movable
      move::
         addr.change(newAddr)
```

With the above class definitions, the properties of `Printable` (in this case just the method `print`) is also a property of `Person`, although via the name `asPrintable`. Correspondingly for `move` of `Movable`:

```
Joe: ref Person
...
Joe := Person("Joe")
Joe.asPrintable.print
Joe.asMovable.move(Address("Norway","Oslo","Røahagan", 33))
```

As mentioned, the above aspects, `Printable` and `Movable`, are example of interfaces in other languages, with only methods. As the methods `print` and `move` are defined independently of `Person` (or other classes that should be `Printable` and `Movable`, interfaces usually do not have any statements describing their actions. This is not the case for aspects that work as interfaces.