9.1 Using fat comma for specifying parameters

Description

For classes and methods with many parameters it may sometimes be difficult to remember which parameter is which in the list.

To improve readability it is possible to associate a name with a parameter – such a name is called a *fat comma* since it is used instead of a comma.

We have seen examples of using fat comma for method invocations and one such example is the `put` method of the `Array` class introduced in section .

```
a: obj Array(100,integer)
a.put(7):at[3]
```

The statement `a.put(7):at[3]` describes the invocation of a method `put:at` with two arguments 7 and 3. This method assigns the value 7 at index 3 in the array `a`.

If we instead just use traditional comma-based syntax, this statement would have to written like:

```
a.put(7,3)
```

With this syntax it may not be clear whether the index is the first or second parameter. Using the fat comma syntax, this is not a problem.

The overall structure of the `Array` class is as follows:

```
class Array(range: var integer, element:< Object):
   get[inx: var integer]:
      :::
   put(e: var element):at[inx: var integer]:
      :::
```

Class `Array` has two parameters `range`, the number of elements of the `Array` and `element`, the type of the elements in the `Array`. It has two attributes, the method `get` and the method `put:at`, which are described in section

We have seen other examples of using fat comma such as the control method `if:then`, `if:then:else`, and `for:to:repeat`. We show the overall structure of `if:then:else` below in this section and a more detailed description is given in section .

The general form of declaring a method using fat comma is:

```
Id1(parameter1):id2(paramater2):id3(parameter3):
   ...
```

This method has the name `Id1:id2:id3` and has three parameters specified by *parameter1*, *parameter2*, and *parameter3*.

An invocation has the form:

```
Id1(exp1):id2(exp2):id3(exp3)
```

where *exp1*, *exp2*, and *exp3* are the arguments being supplied.

There may be an arbitrary number of parameters – for brevity, we have just shown 3 parameters.

It is possible to use different parentheses like `(`, `)`, `[`, `]`, `{`, and `}` as in:

```
put(e):at[inx]:do{S}
```

By convention, `(`, and `)` are used for a parameter representing a datum in general, `[`, and `]` are used for a parameter representing an index, and `{`, and `}` are used for a parameter being a virtual method or class. However, these are not rules enforces by the compiler and the programmer may use whatever parameters he/she prefers, but we recommend following the conventions.

For a parameter defined using curly brackets `{,` and `}`, these may in an invocation be replaced by indentation of the actual argument.

Consider the `if:then:else`, which has the following overall structure:

```
if (cond):then{thenPart:< object}else{ elsePart:< Object}:
   :::
```

An `if:then:else` may be invoked as follows using curly brackets:

```
if (a < b) :then { a := a - b }:else { b := b - a }
```

If indentation is used instead of the curly brackets, this `if:then:else` may be written as follows:

```
if (a < b) :then
   a := a - b
:else
   b := b - a
```

Fat comma may be used to specify the parameters of classes as well as methods.