9.4 Iterators

Description

An iterator is a control structure that iterators over the members of a collection like `Set`, `OrderedList` or `Array`. For each member of a given collection a list of statements is executed with the current member as an argument.

Defining iterators

For collections like class `Set`, it is often useful to perform an action on all elements of a given `Set`. In the `Bank` example, we may add interest to all accounts using the following statement:

```
theAccountsFile.scan
   current.addInterest
```

The object `theAccountsFile` is defined as an attribute of the bank system defined in section . It is a `Set` that contains all the `Account`-objects of the bank.

Here `scan` is a control method in the form of an iterator defined as an attribute of class `Set`. The reference variable `current` is an attribute of `scan` and refers to current element of the iteration. The effect of `theAccountFile.scan ...` is that `current.addInterest` is executed for each `Account`-reference in `theAccountsFile` and for each iteration, `current` refers to the current element.

In the next example, we show how to define `scan`. To do this, we need to supply details of how to represent the `Account`-objects in `theAccountsFile`. For this purpose, we add an array, `members` to hold references to the `Account`-objects.

For simplicity, we assume that there is a maximum of 500 accounts, defined by the constant `max`. The integer variable `top` keeps track of the current no. of accounts.

The method `insert`, checks that its parameter `element` is not already in the `Set`. Then it increments `top` and if `top <= max`, `element` is inserted into `members`. Otherwise an error message is printed.

We may now define scan as shown below:

```
class Set(class ElementType:< Object):   +++ Member?
   insert(element: ref ElementType):
      if (not has(element)) :then
         top := top + 1
         if (top < max) :then
            members(element):at[inx]
         :else
            console.print("No space for more accounts")
   has(element: ref ElementType): ...
   remove(element: ref ElementType): ...
   scan:
      current: ref ElementType
      for (1):to(top):repeat
         current := members[inx]
         inner(scan)
   max: val 500
   members: obj Array(max, ElementType)
   top: var integer
   ...
```

As can be seen, `scan` goes through the elements of `members` and for each element, `members[inx]` is assigned to `current` and then an `inner(scan)` is executed.