7.2 Method details

Objects may have attributes in the form of methods. In the previous chapters, we have seen examples of methods such as `addInterest`, `deposit` and `withdraw`. Here we describe methods in more detail.

A method describes a sequence of statements to be executed. Execution of a statement results in some operations that may change the datums of data-items or compute new datums based on the datums of the data-items of the enclosing object and its possible parameters.

In programming languages in general, a method is often called a procedure, or function. The term method was introduced by Smalltalk and has since then been used instead of procedure in OO languages.
Sometimes the term function is used for a procedure that computes a value only based on its parameters.

A method has the form:

```
MethodName(Parameters) ReturnValue: SuperMethod
   Declarations
   Statements
```

A method has a name as specified by *MethodName*.

A method may have parameters as specified by *Parameters* – if no parameters, the brackets are not needed. The *Parameters* may be one or more declarations of data-items or virtual classes and methods – see section .

In section and chapter , we have seen examples of methods `deposit` and `withdraw` that both have one parameter. In the next example we show an example of a method with two parameters:

```
transfer(from: ref Account, to ref Account, amount: var float):
   from.withdraw(amount)
   to.deposit(amount)
```

A method may also specify a value to be returned by an invocation of the method – *Return*`Value` is the type of this value.

A method may also have a supermethod as specified by *SuperMethod*. A supermethod is similar to a superclass – we explain supermethods in section .

The body of a method consist of a sequence of possible declarations and statements as specified by *Declarations* and *Statements*.

The clause *Return*`Value` is either empty or has the form:

```
-> ReturnDeclaration
```

where *ReturnDeclaration* may be a declaration of a variable (**var** or **ref**) data-item.

In chapter 2 and 3, the method `withdraw` is an example of a method with a return value:

```
   withdraw(amount: var float) -> newB: var float:
      balance := balance - amount
```

```
      newB := balance
```

As can be seen, a method has almost the same structure as a class. The only difference is that a return value may be specified. A class has an implicit return value, being a reference to an object of the class.

Method invocation

In Chapters 2 and 3, we have seen examples of execution — invocation, of methods. Below we show an invocation of transfer from the account of John Smith to the account of Liza Jones.:

```
transfer(account_1010, account_1022, 218)
```

In the next example, we have defined `transfer` within an object `BankSys` together with class `Account` and an instance of `Account`. We have modified `deposit` to also return the new `balance`.

```
BankSys: obj
   class Account(owner: var String):
      balance: var float
      -"-
      deposit(amount: var float) -> newB: var float:
         -"-
      withdraw(amount: var float) -> newB: var float:
         -"-

   transfer(from: ref Account, to ref Account, amount: var float):
      newBalanceFrom, newBalanceTo: var float
      newBalanceFrom := from.withdraw(amount)
      newBalanceTo := to.deposit(amount)

   account_1010: obj Account("John Smith")
   account_1022: obj Account("Liza Jones")
   account_1010.deposit(350)
   transfer(account_1010, account_1022, 218)
```

The transfer method has two local data-items `newBalanceFrom` and `newBalanceTo`, which are assigned the new values of `balance` of `from` and `to` respectively.

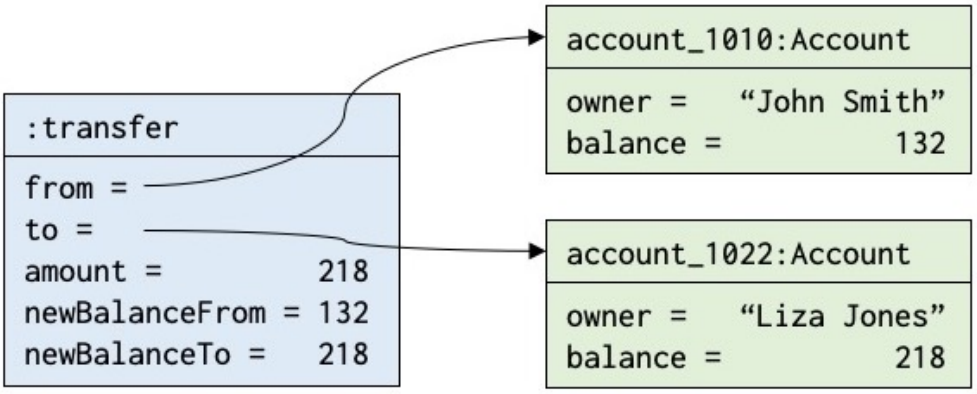
In addition to the declarations of `BankSys`, we have added statements that are executed by `BanksSys`:

```
   account_1010: obj Account("John Smith")
   account_1022: obj Account("Liza Jones")
   account_1010.deposit(350)
   transfer(account_1010, account_1022, 218)
```

A method invocation creates a *method object* that has the structure specified the method. The method object contains possible parameters and possible data-items of the method.

The following snapshot shows the situation at the end of execution of the `transfer` method object:

```
transfer(from: ref Account, to ref Account, amount: var float):
   newBalanceFrom, newBalanceTo: var float
   newBalanceFrom := from.withdraw(amount)
   newBalanceTo := to.deposit(amount)
```



A transfer method object

As mentioned in section , the initial value of `balance` is 0 (zero), which means that before `transfer` is executed `account_1010.balance` is 350 and `account_2022.balance` is 0 (zero).

A method object is sometimes called a *method invocation* or just *invocation*.

There is a strong similarity between method invocation and method object on the one side and class instantiation and object on the other side. Both forms of invocation/instantion create an object and starts executing the statements of the object.

The object (class object or method object) executing a method invocation is denoted the *invoker* of the method object and the method object is sometimes denoted the *invokee*.

Again, we may see that a method invocation is quite similar to a class instantiation. The difference in wording between invocation and instantiation is that for a method, invocation emphasizes that execution of statements is the primary task. For a class, instantiation emphasise that creation of an object is the primary task.