2.5 Methods

The `aAccount_1010` has two data-items `owner`, and `balance` that represent properties associated with a bank account. We also need to represent *activities* associated with bank accounts. Such activities may include calculating possible interest, and deposit and withdraw of money on an account. An activity may be represented by a method.

A *method* is a template for a set of computations. A method may be an attribute of an object just like a data-item. A method has a sequence of statements that when executed carry out a computation that may change the state of the object and/or compute new values from the state of the object.

In the example below, we add a method called `addInterest` that computes the next amount of interest for the account and add it to the `balance`. In addition, we have added a data-item `interestRate` which is the quarterly interest rate for the account. In this simple example, we thus assume that interest is added every quarter. `interestRate` is declared as a variable (**`var`**), since it may change over time.

```
account_1010: obj
   owner: val "John Smith"
   balance: var float
   interestRate: var float
   addInterest:
      balance := balance + (balance * interestRate) / 100
```

The description of method `addInterest` consists of the following assignment statement:

```
balance := balance + (balance * interestRate) / 100
```

The right-side expression is `balance + (balance * interestRate) / 100`.The variable `balance` denotes the current value of `balance` before it is updated by the assignment. The variable `interestRate` similarly denotes the current value of `interestRate` for the account.

Suppose that that `balance` holds the value 350.56 and `interestRate` holds the value 0.7. The execution of the assignment statement then takes place as follows:

1. The variables `balance` and `interestRate` are substituted by their current values resulting in an expression: `350.56 + (350.56 * 0.7) / 100`
2. This expression is evaluated resulting in the value 353.01
3. The value 353.01 is assigned to `balance`, which thus holds the value 353.01 after execution of the assignment statement.

The `addInterest`-method of `account_1010` may be executed by a statement of the form:

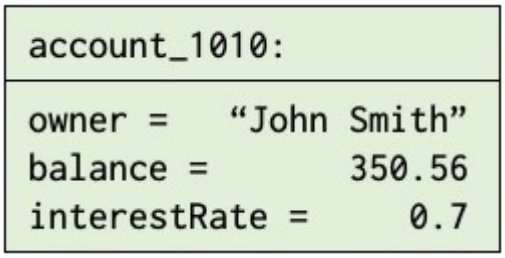
```
account_1010.addInterest
```

Such a statement is called a *method invocation*. As with the assignment to `interestRate` it must be placed in an object. The object executing the method invocation is called the *invoker* of the method. The object where the method is defined is called the *context* of the object.

In the next snapshots we have extended `myFirstProgram` to include an assignment of `account_1010.interestRate` and an invocation of `account_1010.addInterest`.

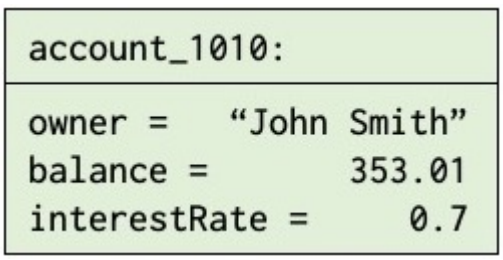
The first snapshot shows the state of `account_1010` after the two assignments, but before the invocation `account_1010.addInterest`. The invoker is `myFirstProgram` and the context is `account_1010`.

```
myFirstProgram: obj
   account_1010: obj
      owner: val "John Smith"
      balance: var float
   account_1010.balance := 350.56
   account_1010.interestRate := 0.7
   ...
   account_1010.addInterest
   ...
```



The second snapshot shows the situation after the invocation of `account_1010.addInterest`.

```
myFirstProgram: obj
   account_1010: obj
      owner: val "John Smith"
      balance: var float
   account_1010.balance := 350.56
   account_1010.interestRate := 0.7
   ...
   account_1010.addInterest
   ...
```



As can be seen, `balance` now has the value 353.01.

As mentioned, we assume that interest is added quarterly. This implies that some other component in the bank system must invoke this method every quarter. The above invocation is only to illustrate how method invocation works.