2.8 The second program

Description

In this section, we show how to represent activities of a clerk in the bank system. We may do this by introducing an object representing a clerk. In the example, the clerk makes a few transactions on the `account_1010`:

```
aClerk: obj
   handle:
      newBalance: var float
      account_1010.interestRate := 0.7
      newBalance := account_1010.deposit(100)
      newBalance := account_1010.deposit(225)
      account_1010.addInterest
      newBalance := account_1010.withdraw(111)
      console.print("The new balance is:" + newBalance)
```

The activities of a clerk is modelled by a method, `handle`, of the `aClerk` object. The description of the method `handle` consists of the following items:

1. A declaration of a local variable `newBalance`.
2. An assignment statement `account_1010.interestRate := 0.7`.
3. Two invocations of the `deposit` method on `account_1010`, with arguments `100` and `225`.
   - The `balance` is now 325
4. An invocation of the `addInterest` method on `account_1010`.
   - The `balance` is now 325 + 325 * 0.7% = 327,28
5. An invocation of `withdraw` on `account_1010` with argument `111`.
   - The `balance` is now 216,28
6. And finally a statement printing the new `balance` of the account.

As mentioned, objects exist within a computer, and the bank clerks need to be able to manipulate the objects. In order to do this, a given object model must be able to receive information from outside the computer and to deliver information to the outside. This is called *input* and *output* (abbreviated *I/O*) to and from an object model.

As a simple start on handling I/O, we assume that our system has an object `console` that represents a window on the screen of the computer executing the bank system. The `console` objects has a method `print` for printing strings and numbers in the associated window.

The clerk executes the following statment as part of `handle`:

```
console.print("The new balance is: " + newBalance)
```

The argument, `"The new balance is: " + balance` of `print` evaluates to the string `"The new balance is: 216.28"` and this string is printed in the window.

The operator + concatenates two strings. An expression `"Hello " + "world"` evaluates to the `String "Hello World"`. An operator of type `float` like `newBalance` will be transformed to a `String` representing its value. For details see .

Next we show how to make a program containing `aClerk` and `account_1010` in the form of a program `mySecondProgram`. In addition `mySecondProgram` contains a statement `aClerk.handle`.

```
mySecondProgram: obj            aClerk: obj
      handle:
         -"-
   account_1010: obj
      -"-
   aClerk.handle
```

Notation: we use `-"-` to stand for code that is not shown, but has been shown in a previous example. One may think of this symbol as an extended ditto mark. See chapter .

The description of `mySecondProgram` is an example of a program that may be executed – it has the following items:

- A declaration of the object `aClerk`.
- A declaration of the object `account_1010`.
- A statement `aClerk.handle`.

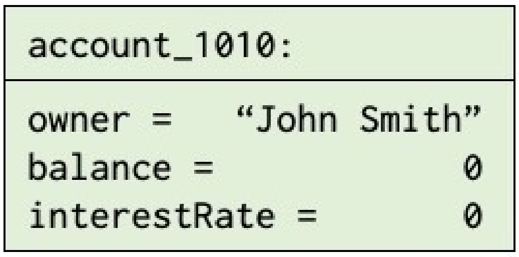
When this program is executed, the object `mySecondProgram` is generated; as part of this generation the two objects `aClerk` and `account_1010` are generated, and finally there is an invocation of `aClerk.handle`.

The following snapshots illustrate the dynamics of the execution of this program.

The first snapshot shows the state of `account_1010` after `mySecondProgram`, `aClerk` and `account_1010` have been generated, and the statement `aCleark.handle` is being executed:

- A `handle` method object has been generated.
- The point of execution is before the statement `account_1010 := 0.7` in `handle`.
- As can be seen, `interestRate = 0.7` and `balance = 0`.

```
mySecondProgram: obj            aClerk: obj
      handle:
         newBalance: var float
 -->     account_1010.interestRate := 0.7
         newBalance := account_1010.deposit(100)
         newBalance := account_1010.deposit(225)
         account_1010.addInterest
         newBalance := account_1010.withdraw(111)
         console.print("The new balance is:" + newBalance)
   account_1010: obj
      -"-
   aClerk.handle
```


account_1010:

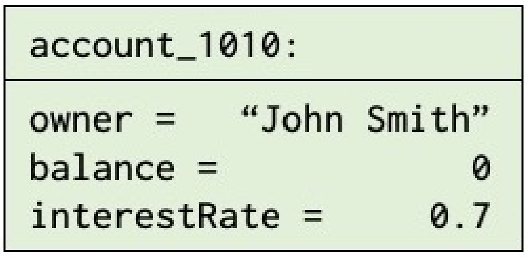
owner = "John Smith"
balance = 0
interestRate = 0

The next snapshot shows the situation after `handle` has executed `account_1010.interestRate := 0.7` and the statement `newBalance := account_1010.deposit(100)` is being executed.

- The point of execution is at the statement `balance := balance + amount`) in `deposit.`
- So far we still have `balance = 0`.

```
mySecondProgram: obj            aClerk: obj
      -"-
   account_1010: obj
      -"-
      deposit(amount: var float):
-->      balance := balance + amount
```

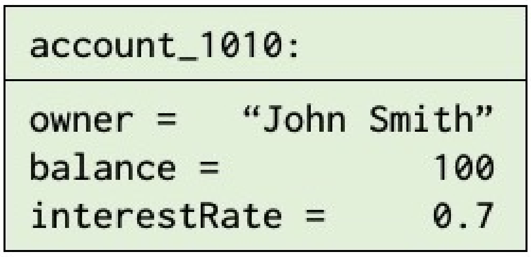
```
   aClerk.handle
```



The next snapshot shows the situation after execution of `deposit` has returned.

- The point of execution is at the statement `account_1010.deposit(225)` in `handle`.
- As can bee seen, `balance` now has the value 100.

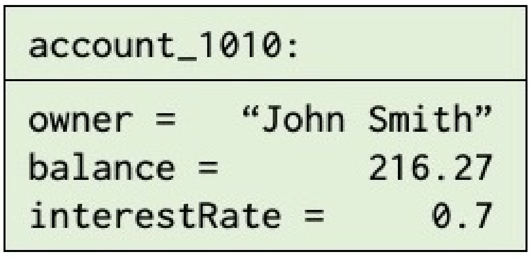
```
mySecondProgram: obj            aClerk: obj
      handle:
         newBalance: var float
         account_1010.interestRate := 0.7
         newBalance := account_1010.deposit(100)
-->      newBalance := account_1010.deposit(225)
         account_1010.addInterest
         newBalance := account_1010.withdraw(111)
         console.print("The new balance is:" + newBalance)
   account_1010: obj
      -"-
   aClerk.handle
```



The final snapshot shows the situation just be fore the statement `console.print("...")` in `handle`.

- As can be seen, the value of a `balance` is now 216.27.

```
mySecondProgram: obj            aClerk: obj
      handle:
         newBalance: var float
         account_1010.interestRate := 0.7
         newBalance := account_1010.deposit(100)
         newBalance := account_1010.deposit(225)
         account_1010.addInterest
         newBalance := account_1010.withdraw(111)
-->      console.print("The new balance is:" + newBalance)
   account_1010: obj
      -"-
   aClerk.handle
```



```
account_1010:

owner =   “John Smith”
balance =       216.27
interestRate =     0.7
```

The above clerk object is only for an illustrative purpose and a does not represent a real clerk in a bank system. We elaborate on the example later in this book.

Terminology: singular object

In this chapter, we have described three objects, `account_1010`, `aClerk`, and `mySecondProgram`. They are all examples of what is called a *singular object* since there is only one of its kind for each of them. In the next section, we introduce the *class* mechanism, which is a template that may be used to generate many objects that have the structure as defined by the template.