8.2 Submethods

Description

In this section, we introduce *submethods*, which makes it possible to organise methods in a hierarchy similar to a class hierarchy using subclasses. As for subclass and superclass, submethod and supermehtod are dual terms. A submethod is a method that has another method as a supermetod, and thus a supermethod is a method being used to define submethods.

One primary use of methods as supermethods is to define new control abstractions, which may be used as control structures like if-then, etc. In chapter , we show examples of how to define such control abstractions, and chapter have further examples.

In section we introduced class `Transaction` and augmented `deposit` and `withdraw` to record each deposit and withdraw on a given `Account`:

```
deposit(amount: var float):
   balance := balance + amount
   transactions.insert(
      Transaction("deposit", clock.today, clock.now, amount))
withdraw(amount: var float) -> newB: var
 float:
   balance:= balance - amount
   transactions.insert(
      Transaction("withdraw", clock.today, clock.now, amount))
```

They both invoke `transactions.insert` with almost identical arguments. Both `deposit` and `withdraw` perform transactions on the `Account` and we would therefore like to reflect this in the model of the bank account.

To do this, we introduce a common supermethod `transact`:

```
transact(amount: var float):
   theTransaction: ref Transaction
   theTransaction := Transaction("", clock.today, clock.now, amount))
   inner(transact)
   transactions.insert(theTransaction)
```

The method `transact` takes care of recording the `transactions` in the `transactions` object. It generates a `Transaction` object and assigns it to the reference variable `theTransaction`. Then it executes an `inner(transact)`, which has the effect that possible statements in submethods of `transact` are executed. We explain `inner` in section below.

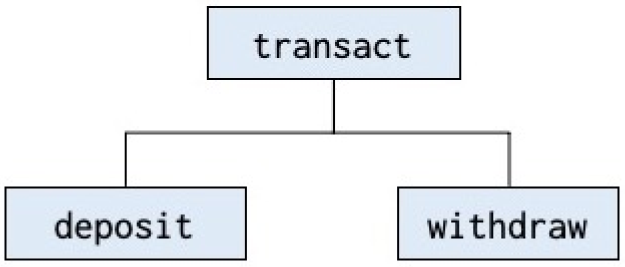
We may now rewrite `deposit` and `withdraw` to become submethods of `transact`:

```
deposit: transact
   balance := balance + amount
   theTransaction.what := “deposit”
withdraw: transact
   balance := balance – amount
   theTransaction.what := “withdraw”
```

By writing `transact` after `deposit:` and `withdraw:` both are defined as submethods of `transact`.

It is the same mechanism as for classes and subclasses; properties of `transact` becomes properties of both `deposit` and `withdraw`, including both data-items and statements.

It is therefore illustrated similarly, methods, however, with another color than classes.

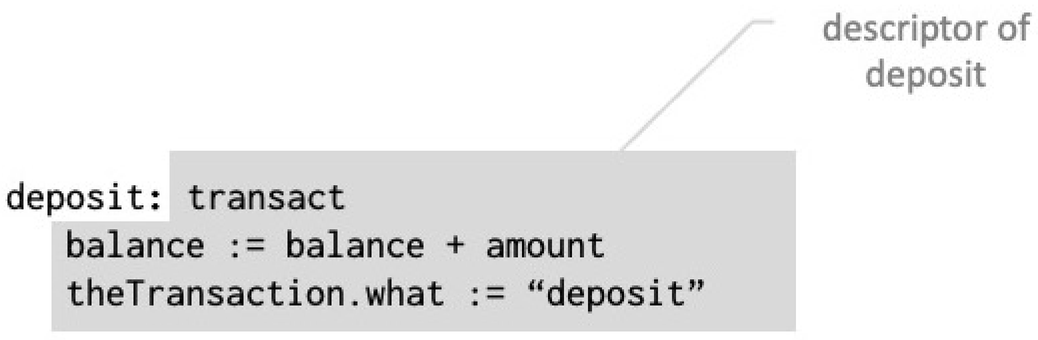


The effect of this is that the code in `transact` works as a wrapper around the code in `deposit` and `withdraw`. Invocation of `deposit` and `withdraw` starts by execution of the statements in `transact`; here execution of `inner(transact)` implies that the statements in `deposit` and `withdraw` are executed; finally the statements after `inner(transact)` are executed.

In the next section, we give a more detailed description of `inner`.

Method descriptor with a supermethod

The following figure shows that the method descriptor for `deposit` includes its supermethod `transact`:



```
deposit: transact
   balance := balance + amount
   theTransaction.what := “deposit”
```