6 DESIGN BY CONTRACT AND ASSERTIONS

If classes are to deserve their definition as abstract data type implementations, they must be known not just by the available operations, but also by the formal properties of these operations, which did not yet appear in the preceding example.

The role of assertions

Eiffel encourages software developers to express formal properties of classes by writing **assertions**, which may in particular appear in the following roles:

- Routine **preconditions** express the requirements that clients must satisfy whenever they call a routine. For example the designer of *ACCOUNT* may wish to permit a withdrawal operation only if it keeps the account's balance at or above the minimum. Preconditions are introduced by the keyword **require**.
- Routine **postconditions**, introduced by the keyword **ensure**, express conditions that the routine (the supplier) guarantees on return, if the precondition was satisfied on entry.
- A class **invariant** must be satisfied by every instance of the class whenever the instance is externally accessible: after creation, and after any call to an exported routine of the class. The invariant appears in a clause introduced by the keyword **invariant**, and represents a general consistency constraint imposed on all routines of the class.

With appropriate assertions, the class *ACCOUNT* becomes:

```
class ACCOUNT create
        make
feature
        ... Attributes as before:
                balance, minimum_balance, owner, open ...
        deposit (sum: INTEGER) is
                        -- Deposit sum into the account.
                require
                        sum >= 0
                do
                        add (sum)
                ensure
                        balance = old balance + sum
                end
```

```
        withdraw (sum: INTEGER) is
                -- Withdraw sum from the account.
            require
                sum >= 0
                sum <= balance – minimum_balance
            do
                add (–sum)
            ensure
                balance = old balance – sum
            end
        may_withdraw ...  -- As before
feature {NONE}
        add ... -- As before
        make (initial: INTEGER) is
                -- Initialize account with balance initial.
            require
                initial >= minimum_balance
            do
                balance := initial
            end
invariant
        balance >= minimum_balance
end -- class ACCOUNT
```

The notation **old** *expression* is only valid in a routine postcondition. It denotes the value the *expression* had on routine entry.

Creation procedures

In its last version above, the class now includes a creation procedure, *make*. With the first version , clients used creation instructions such as **create** *acc1* to create accounts; but then the default initialization, setting balance to zero, violated the invariant. By having one or more creation procedures, listed in the **create** clause at the beginning of the class text, a class offers a way to override the default initializations. The effect of

```
create acc1.make (5_500)
```

is to allocate the object (as with the default creation) and to call procedure *make* on this object, with the argument given. This call is correct since it satisfies the precondition; it will ensure the invariant.