8.1.3 Example

Here we show an example of writing general [redacted] using the extended type rule.

As an example, a customer may have different kinds of accounts. In the example below we add a `SavingsAccount` and a `CreditAccount` to `JohnSmithProfile`. We also add a method `totalBalance` to `Customer` – this method computes the sum of the balance of all accounts of a given customer.

```
class Customer
   -"-
   totalBalance -> bal: var float:
      accounts.scan
         bal := current.balance
JohnSmithsProfile: obj Customer("John Smith")
anAccount: ref Account
anAccount := SavingsAccount(JohnSmithProfile)
JohnSmitProfile.addAccount(anAccount)
anAccount.deposit(300)
...
aCreditAccount: ref CreditAccount
aCreditAccount := CreditAccount(JohnSmithProfile)
JohnSmitProfile.addAccount(aCreditAccount)
aCreditAccount.maxCredit := 999
...
console.print("Total balance sum: " + JohnSmithProfile.totalBalance
```

Here we see examples of assigning a reference to an object to a variable of a more general type. One example is:

```
anAccount := SavingsAccount(JohnSmithProfile)
```

The expression `SavingsAccount(JohnSmithProfile)` generates an object of type `SavingsAccount` – the reference to this object is then assigned to `anAccount` which has the more general type `Account`.

Another example is:

```
JohnSmitProfile.addAccount(aCreditAccount)
```

Here a reference variable of type `CreditAccount` is assigned to the parameter of `addAccount`, which is of type `Account`.

The example also shows that it is possible to write code that works for all kinds of `Account` objects. This is the case for the method `addAccount` that adds the `Account` being passed as argument to the `accounts` array of the `Customer` object.

Another example is the new method `totalBalance`, which has the statement:

```
accounts.scan
   bal := current.balance
```

This statement scans through all objects in the accounts array – `current` refers to the current `Account` object, and the type of `current` is `Account`. The expression `current.balance` access the `balance` attribute of the account object bering referred to by `current`. This works since all objects in `accounts` have a `balance` attribute independently of being an instance of `Account`, `SavingsAccount` or `CreditAccount`.

Consider the statement

```
aCreditAccount.maxCredit := 999
```

Here the attribute `maxCredit` is accessed. This is possible since `aCreditAccount` is of type `CreditAccount` and thus must refer to an object of type `CreditAccount`.

If we had used the variable `anAccount` instead (as we did for generating the `SavingsAccount`), we cannot access the attribute `maxCredit` since only `CreditAccounts` has this attribute.

In section on virtual methods, we show how to write general code that depends on objects that are instances of different subclasses.