5.1.1-old Reference assignment and comparison

Description

Here we will summarise assignment of references in general and also describe comparison of references. We use class `Customer` for the example and we use the following demo object:

```
aClerk: obj
   JohnSmithProfile: obj Customer("John Smith")
   LizaJonesProfile: obj Customer("Liza Jones")
   customerA, customerB: ref Customer
   customerA := JohnSmithProfile
   customerB := LizaJonesProfile
   customerA := customerB
```

Figure 5.1.1 Customer references

In the above example, we have two `Customer` objects `JohnSmithProfile` and `LizaJonesProfile`, and two reference variables `customerA` and `customerB`. The following snaphots illustrates the effect of reference assignments.

The first snapshot shows the situation after generation of `aClerk` – marked by the red arrow (–>). Here `JohnSmithsProfile` refers to `Customer("John Smith")` and `LizaJonesProfile` refers to `Customer("Liza Jones")`. The reference variable `customerA` and `customerB` are both none:

```
 aClerk: obj
    JohnSmithProfile: obj Customer("John Smith")
    LizaJonesProfile: obj Customer("Liza Jones")
    customerA, customerB: ref Customer
--> customerA := JohnSmithProfile
    customerB := LizaJonesProfile
    customerA := customerB
```

Snapshot

Snapshot A

The next snapshot shows the situation after the assignment `customerB := LizaJonesProfile`. As can be seen, `customerB` and `LizaJonesProfile` now both refer to the same object:

```
 aClerk: obj
    JohnSmithProfile: obj Customer("John Smith")
    LizaJonesProfile: obj Customer("Liza Jones")
    customerA, customerB: ref Customer
    customerA := JohnSmithProfile
    customerB := LizaJonesProfile
--> customerA := customerB
```

Snapshot

Snapshot B

The final snapshot shows the situation after execution of `customerA := customerB`. As can be seen, `customerA` and `customerB` now both refer to `LizaJonesProfile`.

```
 aClerk: obj
    JohnSmithProfile: obj Customer("John Smith")
    LizaJonesProfile: obj Customer("Liza Jones")
    customerA, customerB: ref Customer
    customerA := JohnSmithProfile
    customerB := LizaJonesProfile
    customerA := customerB
-->
```

Snapshot

Snapshot C

Comparison

We may also compare references using = (equality) and <> (inequality).

Below, we show the value of some reference expressions using = (equality) and <> (inequality) at Snapshot B and Snapshot C above.

For the situation at *Snapshot* B above, we have the following:

Og her bruges kommentarer næste gang

```
JohnSmithProfile = LizaSmithProfile    -- false, they refer to different objects
JohnSmithProfile <> LizaSmithProfile   -- true, they refer to the same object
JohnSmithProfile = customerA           -- true, they refer to the same object
JohnSmithProfile <> customerB          -- false, they refer to different objects
```

The first comment `-- false, they refer to different objects` is meant to say that the expression `JohnSmithProfile = LizaSmithProfile` evaluates to the value `false`.

The situation at Snapshot C after the assignment `customerA := customerB` is as follows:

```
JohnSmithProfile = customerA   -- false
customerA = customerB          -- true
LizaJonesProfile = customerA   -- true
```

Assignment between data items being references is called r*eference assignment* and comparison of references is called *reference comparison*.

Reference assignment and reference comparison is fundamentally different from assignment between data items representing values.

The `withdraw` method has a statement:

```
newB := balance
```

Here the value hold by `balance` is copied to `newB`, which then holds the same value as `balance`. The data items `newB` and `balance` are not references to some objects. As we shall se in section X, they are a special kind of objects called *value objects* that may represent values – in section , we describe *value assignment* and *value comparison*.

Type rule

As shown above, we may assign a reference to a `Customer` object to a reference variable that has the type `Customer`. It is not possible to assign a reference to an `Account` object to the reference variable `aCustomer`.

In general the type of an expression in an assignment must be the same as the type of the reference variable being assigned to. This is also the case for passing an expression as an argument to a parameter of a method being a reference.

The above rule does also apply to comparisons using = (equality) and <> (inequality) where both arguments must be of the same type

Consider the following example:

```
aCustomerA, aCustomerB: ref Customer
anAccountA, anAccountB: ref Account
B: var Boolean
aCustomerA := Customer("John Smith")   -- legal
anAccountA := Account(aCustomerA)      -- legal
aCustomerA := anAccountA               -- illegal
anAccountB := Account(anAccountA)      -- illegal
B := aCustomerA = aCustomerB           -- legal
B := aCustomerA <> anAccountA          -- illegal
```

The assignment `anAccountB := Account(anAccountA)` is illegal since the `owner` parameter of `Account` is of type `Customer` whereas the argument `anAccountA` is of type `Account`.

The purpose of the type rule is two fold: from a programming and modeling point of view it does not make sense to allow assignments like `aCustomerA := anAccountA`.

Secondly the type rule is necessary to prevent errors at run-time. Assume that we allow the assignment then we may write code as

```
aCustomerA := anAccountA
aCustomerA.addAccount(JohnSmithProfile)
```

This does not makes sense since a `Customer` object does not have an `addAccount` method.

In chapter , we extend the type rule for assignment, parameter transfer and comparison.

Parameter passing

Passing a parameter as part of a method invocation or class invocation is similar to assignment in the sense that the actual parameter is assigned to the formal parameter of the method or class respectively. This also means that the type rules for parameter passing is the same as the type rule for assignment.