

## 18.3.2 Representing the observer pattern as aspects

### Description

The example in the previous section is meant for illustrating the observer pattern, but from a modeling point-of-view, it does not make sense that `fireAlarm` is a subclass of `Subject` and that `FireDepartment`, etc. are subclasses of `Observer`.

In this section, we will show how the observer pattern may be represented as aspects. For this purpose, we will use the bank example. A scenario that calls for adding subject/observer aspects to the bank system is the following: Every day the bank checks if there has been some suspicious transactions on accounts, and in case, both a special alarm part of the bank *and* the actual customer are notified of the event. This is done by giving each account a subject aspect and both alarm and each customer an observer aspect.

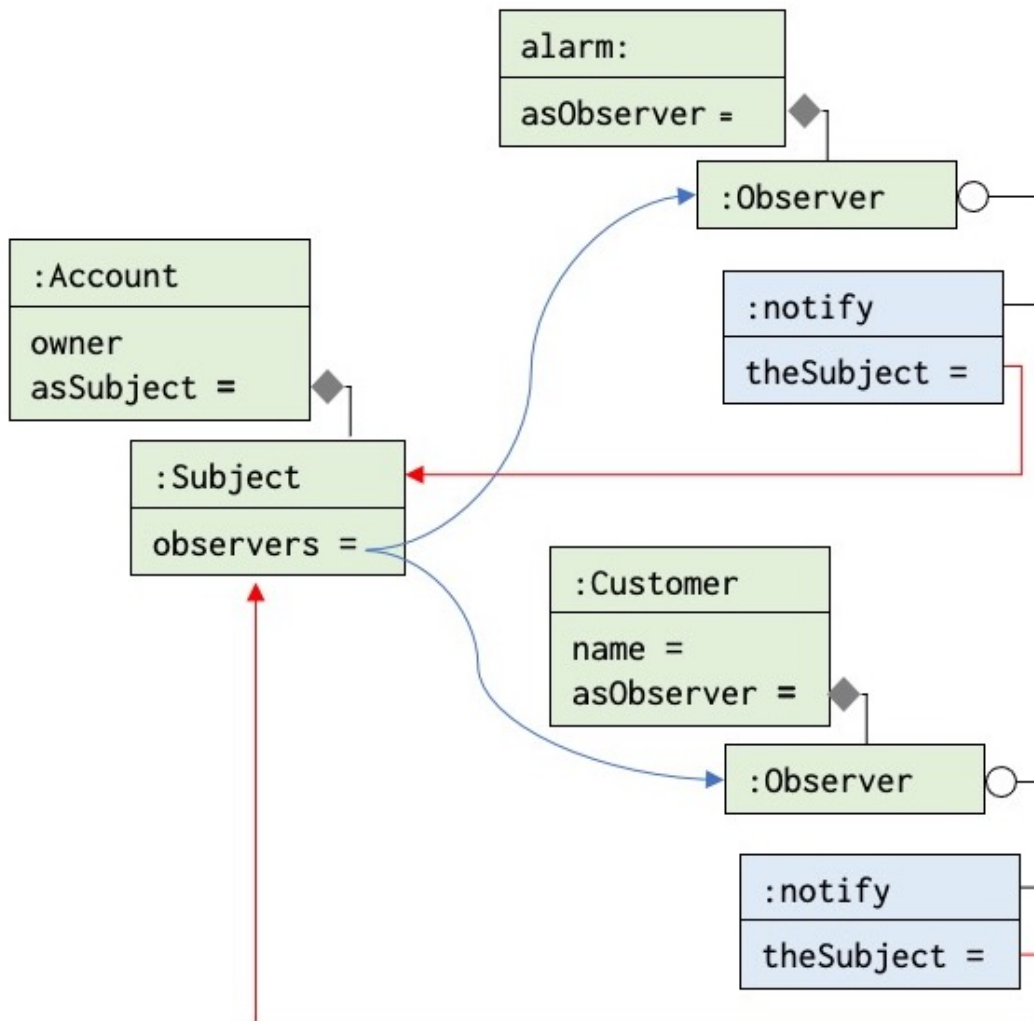
`Account` objects have a subject aspect represented by the intrinsic object `asSubject`, and `Customer` objects and the `alarm` object have an observer aspect represented by the intrinsic object `asObserver`:

```
BankSysEx: obj
  class Account(owner: ref Customer):
    "_"
    asSubject: obj Subject

  class Customer(id: var Integer):
    "_"
    asObserver: obj Observer

  alarm: obj
    asObserver: obj Observer
  ...
```

The subject and observer aspects are now represented as properties of intrinsic objects of `Account` objects, of `Customer` objects, and of the `alarm` object. This is illustrated in the following figure, in the first round with the types of the intrinsic objects being `Subject` and `Observer`:



As for the Firealarm example, we also have to extend ObservedSubject to include information about the accounts. The type of asSubject object cannot just be Subject, as it has to report events that has to do with Account-objects. Since we have two types of observers, Customer objects and the alarm, we introduce a class AccountSubject being a subclass of Subject, and we use the class as the type of asSubject:

```

class AccountSubject(whichAccount: ref Account): Subject
  issueWithTransactions:
    "There is an issue with the account of: ".print
    whichAccount.owner.print
    inner(AccountSubject)

class Account(owner: ref Customer):
  "-"
  asSubject: obj AccountSubject(this(Account))
    
```

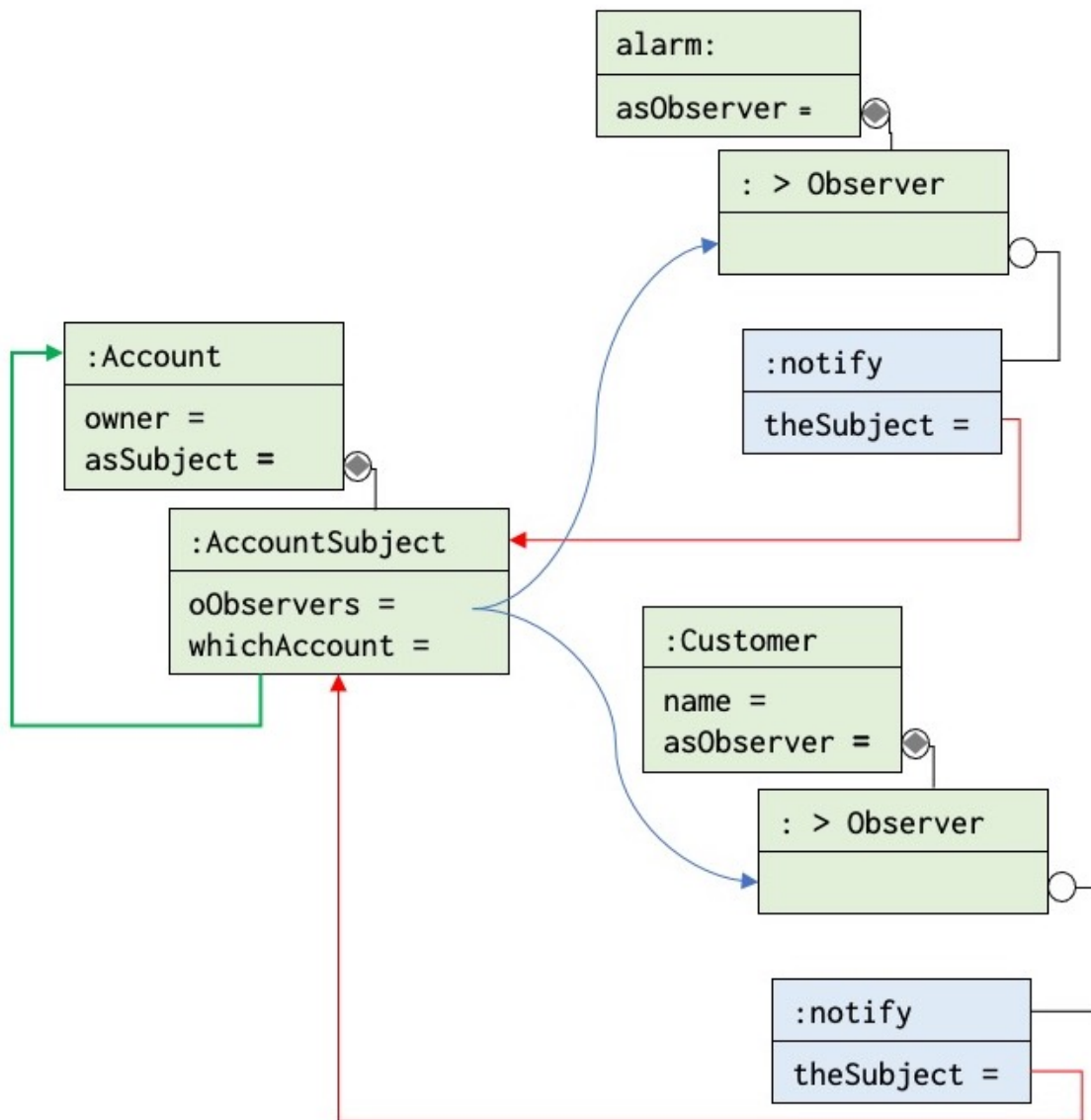
In a similar way, the asObserver objects of Customer and alarm are defined as singular objects derived from class Observer:

```

class Customer(name: var String):
  asObserver: obj Observer
    class ObservedSubject:: AccountSubject
    notify::
      theSubject.issueWithTransactions
  ...
alarm: obj
    
```

```

asObserver: obj Observer
class ObservedSubject:: AccountSubject
notify::
    theSubject.issueWithTransactions
    
```



Next we show how `Customer` objects and `alarm` may subscribe to events in `Account` objects. This is done in a method `newAccount` in `BankSysEx`:

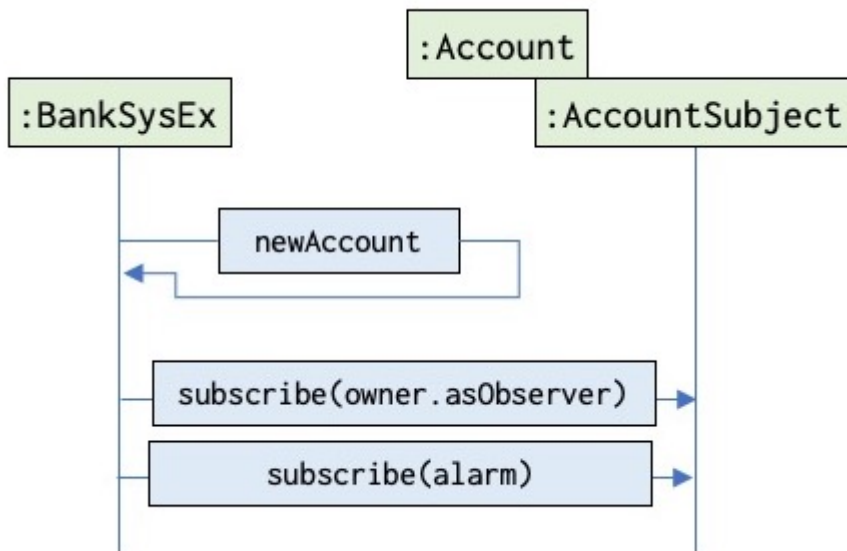
```

newAccount(owner: ref Customer):
    acc: ref Account
    acc := Account(owner)
    theAccountsFile.insert(acc)
    acc.asSubject.subscribe(owner.asObserver)
    acc.asSubject.subscribe(alarm)
    
```

The method `newAccount` creates a new `Account` for the `Customer` referred to by `owner`.

- First a new `Account` object is generated and a reference to it is assigned to `acc`.
- Next this reference is inserted in the `theAccountsFile`, which is the set of accounts of `BankSysEx`.
- Then the `owner` aspect (`owner.asObserver`) of the new `Account` is subscribed to the subject aspect of `acc` (`acc.asSubject`).

- Finally the alarm subscribes to the subject aspect of acc.



To sum up, the overall structure of BankSysEx is:

```

BankSysEx: obj
class AccountSubject(whichAccount: ref Account): Subject
    ""
class Account(owner: ref Customer):
    ""
    asSubject: obj AccountSubject

class Customer(id: var Integer):
    ""
    asObserver: obj Observer
    ObservedSubject:: AccountSubject
    ""

alarm: obj
    asObserver: obj Observer
    ObservedSubject:: AccountSubject
    ""
newAccount(owner: ref Customer):
    ""
accountsFile: obj OrderedList(Account)
...
    
```