18.3.1 The alarm example

Description

In order to simplify this first example, we do not represent the subject and observer aspects as aspects in the form of objects as in the previous sections. We give an example of this later in this section.

The fire alarm is represented by an object of class `Alarm`, a subclass of `Subject`. The fire station, center owner, and shops are represented by instances of class `Stakeholder` which in turn is a subclass of `Observer`.

```
AlarmSystem: obj
   class Stakeholder(id: var String): Observer
      ObservedSubject::< ShoppingMall.Alarm
      notify::<
          theSubject.whatWentWrong
          ...
      setUp:
         ShoppingMall.fireAlarm.subscribe(this(StakeHolder))
   FireStation: obj Stakeholder("FireStation")
   CenterOwner: obj Stakeholder("CenterOwner")
   ShoppingMall: obj
      class Alarm(id: var String): Subject
         whatWentWrong::
            " - smoke detected\n".print
         alert:
            notifyObservers
      fireAlarm: obj Alarm("fireAlarm")
      aSensor: obj Sensor
         ...; fireAlarm.alert; ...
      ShopA: obj StakeHolder("ShopA")
      ShopB: obj StakeHolder("ShopB")
      ShopC: obj StakeHolder("ShopC")
      setUp:
         ShopA.setUp
         ShopB.setUp
         ShopC.setUp
   FireStation.setUp
   CenterOwner.setUp
   ShoppingMall.setUp
```

- The `AlarmSystem` has a class `StakeHolder` that is a subclass of `Observer`.
  - It has an `id` identifying the `Observer`.
  - It makes further binding of `notify`, which invokes `theSubject.whatWentWrong`. The dots (`'...'`) indicates that some further actions may be needed. We discuss that at the end of this section.
  - It makes a further binding of class `ObservedSubject` to `Alarm`.
  - Note that `theSubject` (the parameter of `notify`) is of type `ObservedSubject`, and since it is bound to `Alarm`, the method `whatWentWrong` may be invoked.
  - It has a method `setUp`, which invokes `ShoppingMall.fireAlarm.subscribe(this(StakeHolder))` and thereby makes `this(StakeHolder)` subscribe to events from `fireAlarm`.
- The `AlarmSystem` has `Observer` objects `FireStation`, and `CenterOwner`, which are declared as instances of class `StakeHolder`.
- The `AlarmSystem` has a singular object `ShoppingMall`.
  - It has a local class `Alarm`, which has an `id` identifying the alarm.
    - It has a method `whatWentWrong` that may supply information to an `Observer` when notified.
    - It has a method `alert` that is invoked by a fire censor in case a fire is detected.
  - It has an object `fireAlarm` that is declared as an instance of class `Alarm`.
  - It has a local object `aSensor`, which is an instance of class `Sensor` (not shown).
  - It has local objects `ShopA`, `ShopB`, and `ShopC` representing three shops in the mall.
  - It has method `setUp`, which invokes `setUp` for the three `Shop` objects and thereby makes them subscribe to

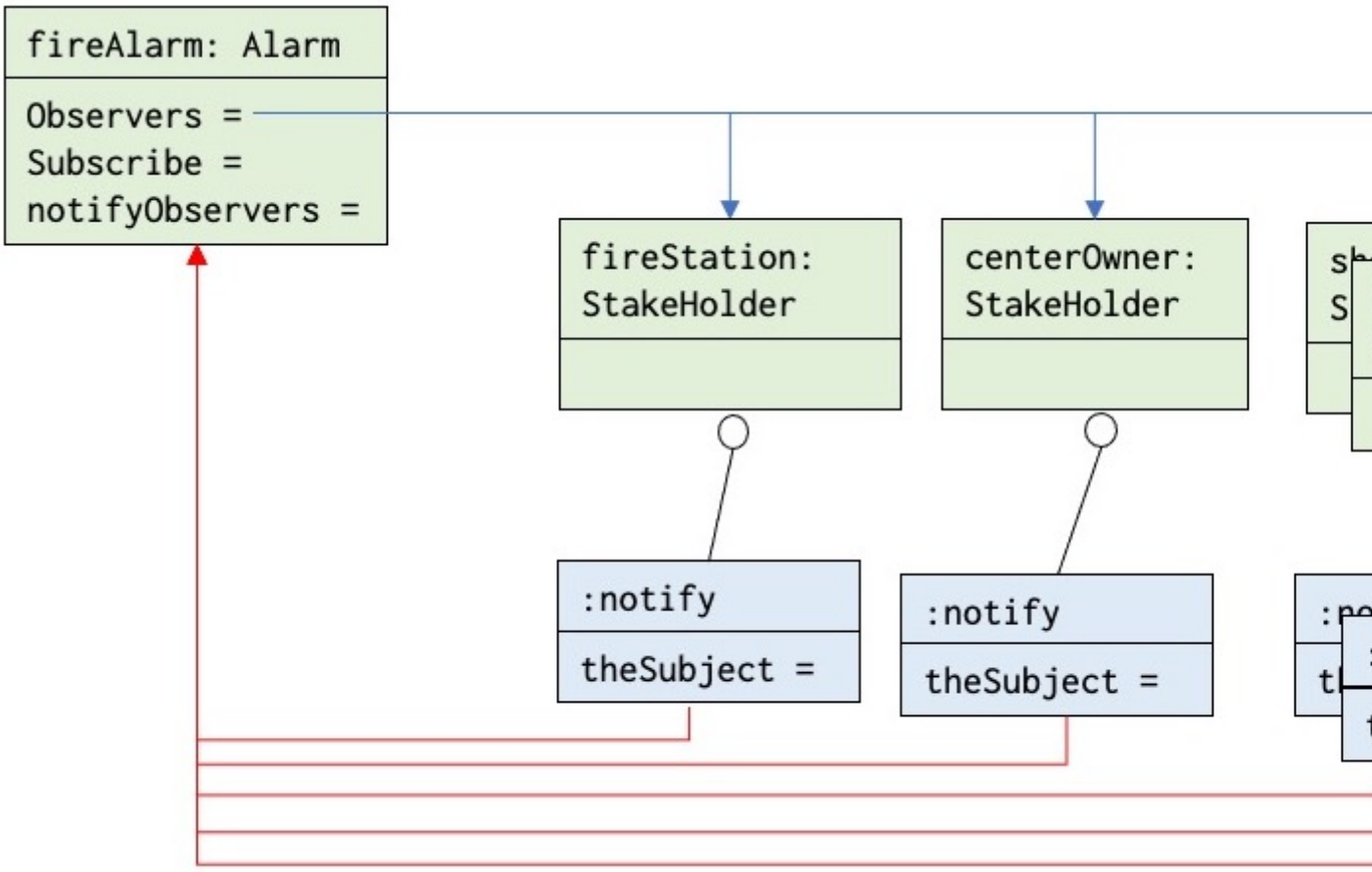
events from the `fireAlarm`.

- Finally, the `AlarmSystem` invokes `setUp` for the `FireStation`, `CenterOwner` and `ShoppingMall`.

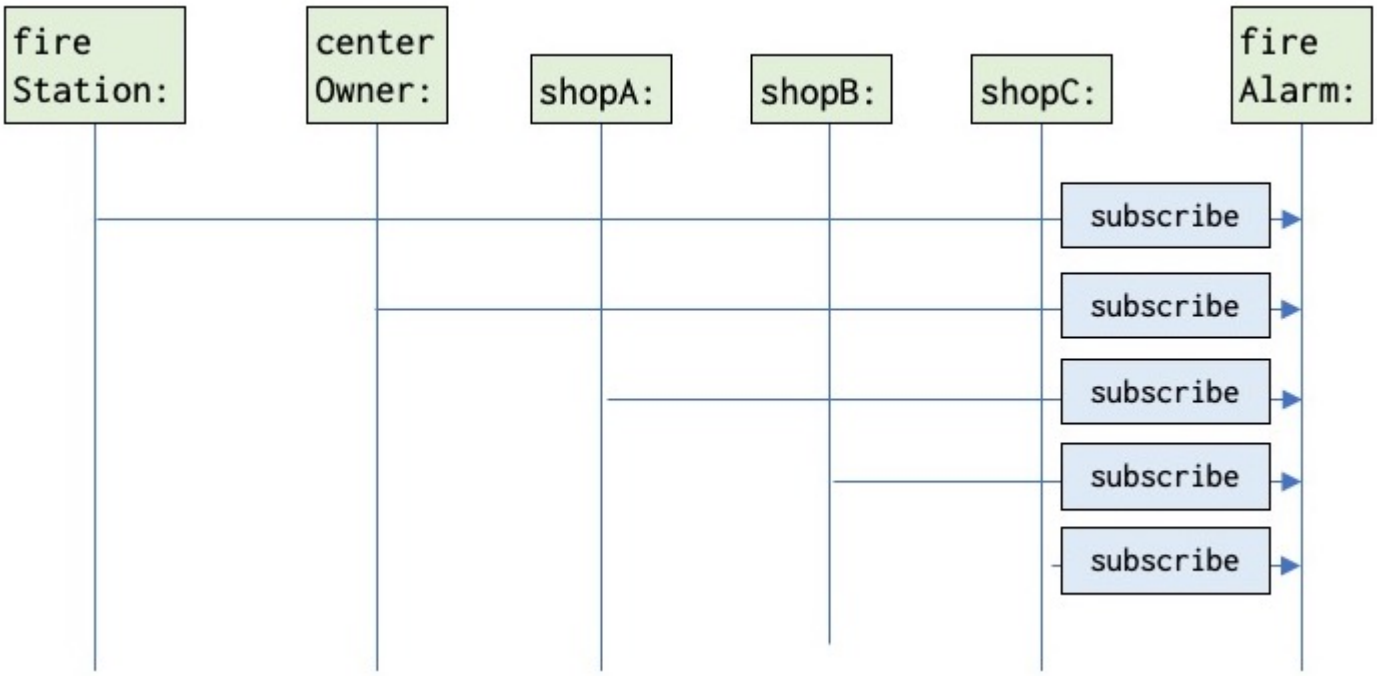
In the example we have shown just one instance of a a `Sensor` object – in practice there may be many sensors in a shopping mall. We have not described how a `Sensor` object is activated to invoke `fireAlaram.alert` – this depends on the actual sensor hardware.

The next diagram illustrates the situation where the subject `fireAlarm` has references to all the `Observers`, and the argument, `theSubject` of invocations of `notify` on an `Observer` refers to the `fireAlarm` object.

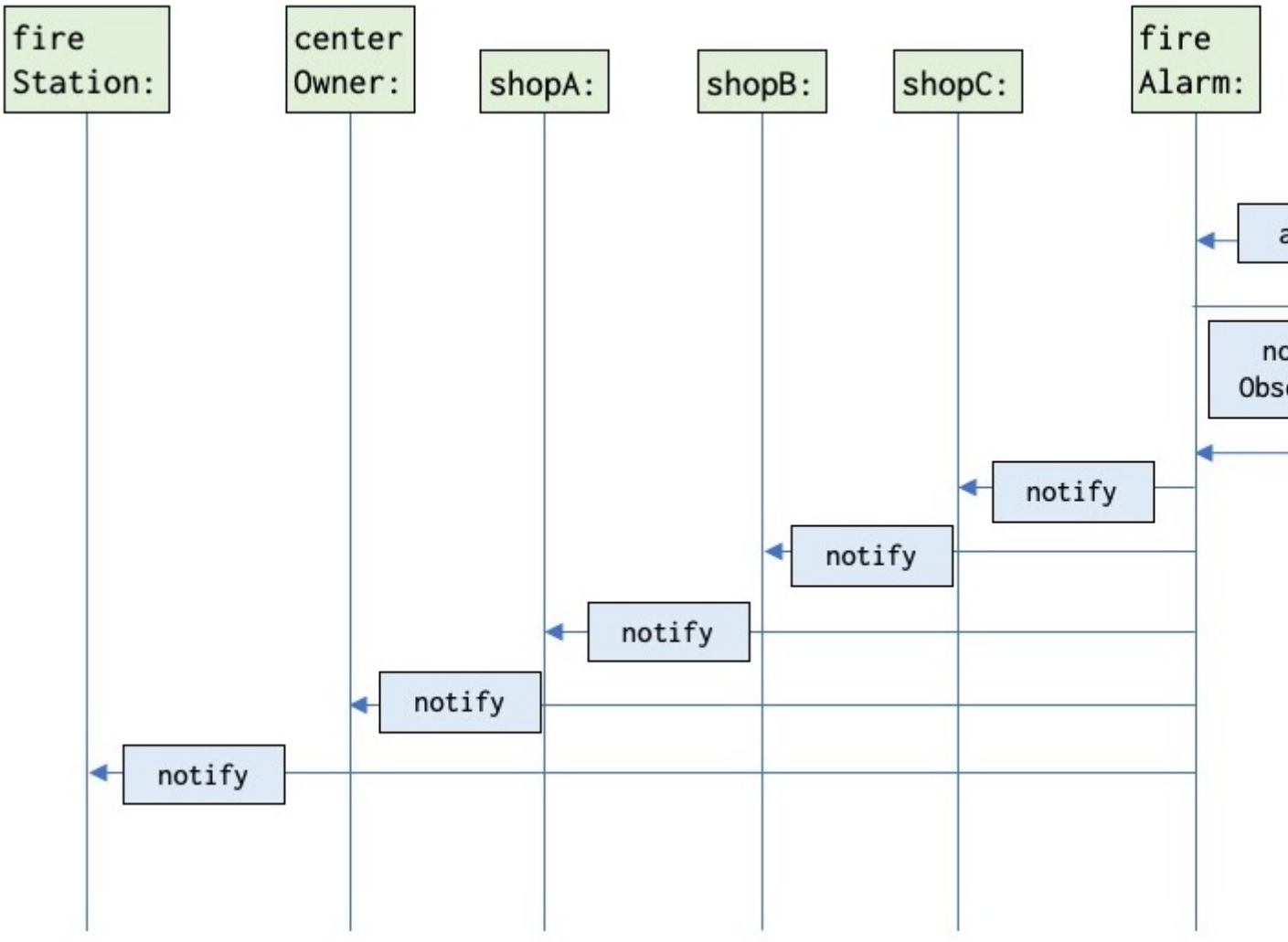
The `ShoppingMall` is an example of composition where `fireAlarm`, `aSensor`, `ShopA`, `ShopB` and `ShopC` are components being parts of the `ShoppingMall`.



`Stakeholder` objects subscribe to events from the `fireAlarm`. This includes the `fireStation`, the `centerOwner`, and the shop stake holders:



Fire sensors located at places in the shopping center are represented by objects of class `Sensor`. When a sensor detects a fire, the corresponding `Sensor` object invoke `fireAlarm.alert`, which then invokes `notifyObservers`, which in turn invoke `notify` on all `Stakeholder` objects. This is illustrated by the following OSD, however, just showing one `Sensor` object.



The `notify` invocations all invoke `whatWentWong` on the `fireAlarm`, but this is not shown in the above diagram.

In the above example, the `fireAlarm` is the only subject being observed. It is of course possible to add more `fireAlarm` objects at different places in the shopping center or at other locations. The `FireStation` will then observe all the `fireAlarms` whereas the shops and owner of a given shopping center only observe the alarms in that center.

Connections to the physical shopping center

As mentioned in the start of this section, a notification of an observer may involve more action than described in the further binding of `notify` in class `StakeHolder`, since the different stakeholders may need to be informed in different ways.

For the `FireStation`, an alarm may be activated in the living room of the fire fighters, the address of the `ShoppingMall` must be communicated to the GPS in the fire engines, etc. For the `ShopOwner`, an sms may have to sent to the shop owner. For the shops, the owner and personel of a shop must be informed, etc.

As an example we may add an `inner` to `invoke` in `StakeHolder` and introduce a further binding of `invoke` in `CenterOwner`:

```
AlarmSystem: obj
   class Stakeholder(id: var String): Observer
      ObservedSubject::< Alarm
      notify::<
          theSubject.whatWentWrong
          inner(notify)
```

```
   -"-
-"-
CenterOwner: obj Stakeholder("CenterOwner")
   notify::<
       -- send sms to center owner
```

We have just inserted a comment about sending an sms to the owner of the center. In order to do that, we need an object that is connected to a real mobile phone device.

Remember that the objects in the example are representations of the real physical phenomena and their connection to these depends on the details of how the `AlarmSystem` is connected to the physical shopping mall.