18.3.1 The alarm example

Description

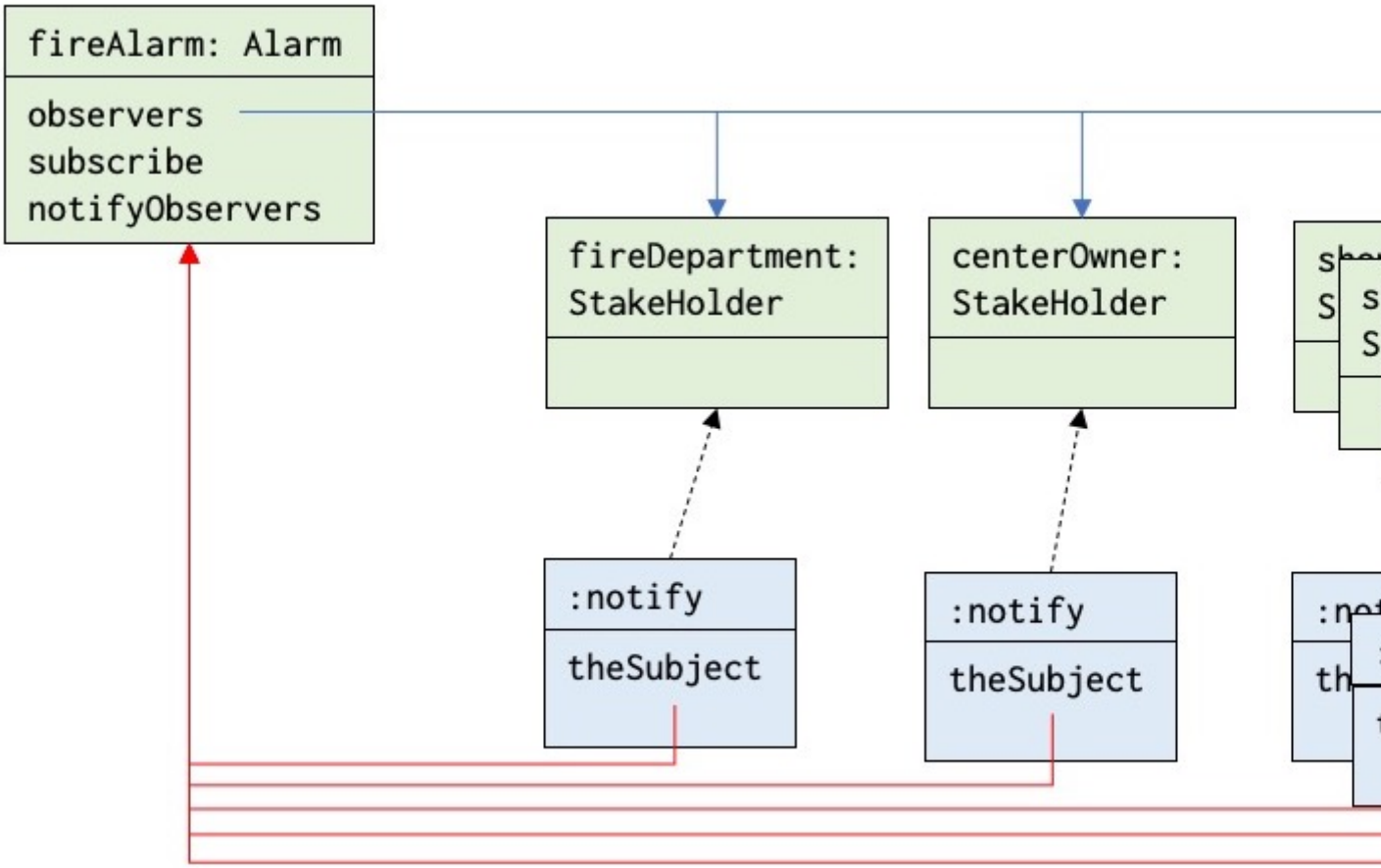
In order to simplify this first example, we do not represent the subject and observer aspects as aspects in the form of objects as in the previous sections. We give an example of this later in this section.

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

```
class Alarm(id: var String): Subject
   whatWentWrong:
      " - smoke detected\n".print
   alert:
      notifyObservers
class Stakeholder(id: var String): Observer
   ObservedSubject::< Alarm
   notify::
       theSubject.whatWentWrong
   fireAlarm.subscribe(this(StakeHolder))
fireAlarm: obj Alarm("fireAlarm")
fireDepartment: obj Stakeholder("FireDepartment")
centerOwner: obj Stakeholder("CenterOwner")
shopA: obj StakeHolder("ShopA")
shopB: obj StakeHolder("ShopB")
shopC: obj StakeHolder("ShopC")
```

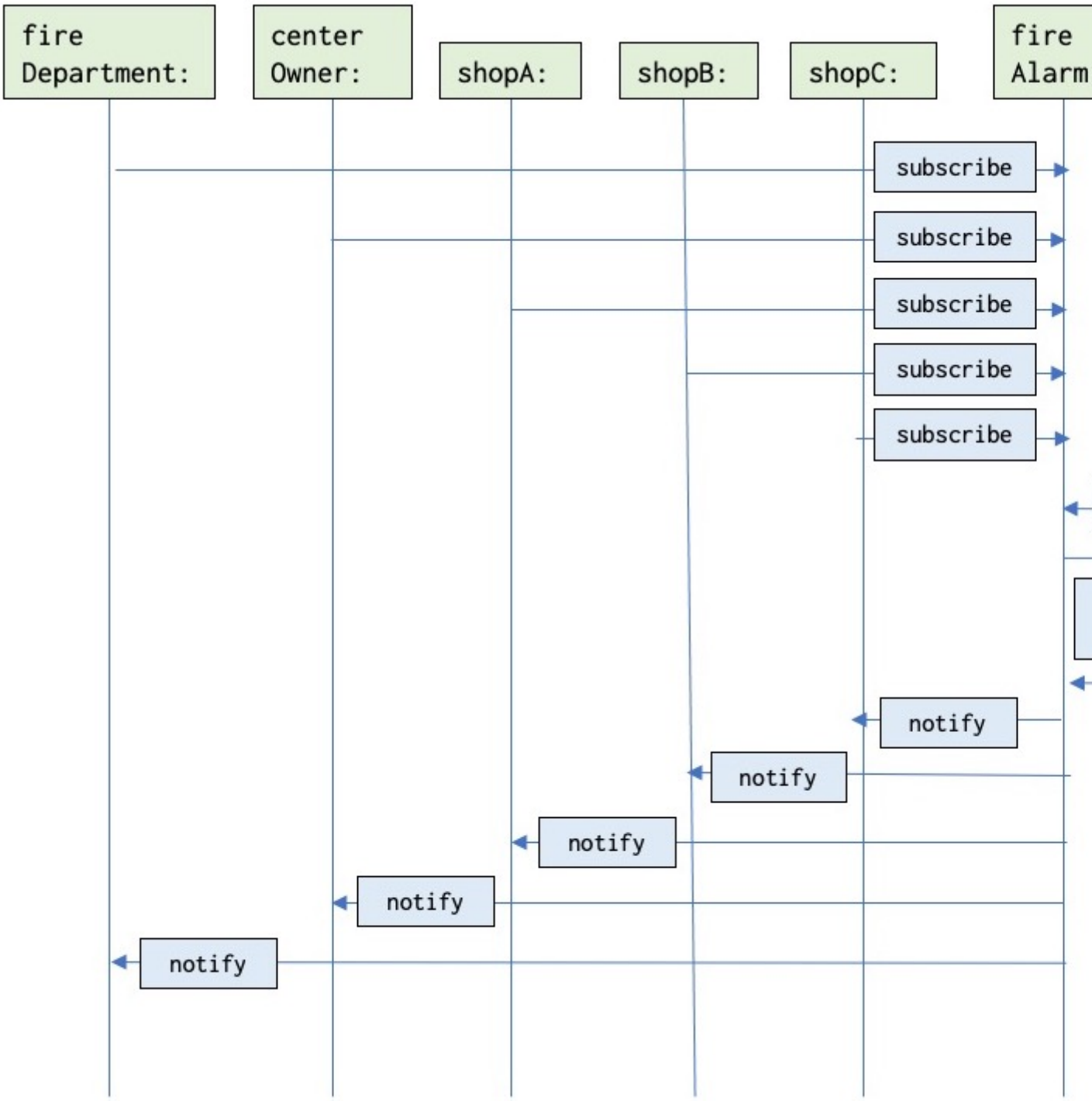
- Class `Alarm` has an `id` identifying the alarm.
- It has a method `whatWentWrong` that may supply information to an `Observer` when notified.
- It has a method `alert` that is invoked by a fire censor in case a fire is detected.
- Class `StakeHolder` is a subclass of `Observer`.
- It has an `id` identifying the `Observer`.
- It makes further binding of `notify`, which invokes `theSubject.whatWentWrong`.
- 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 invokes `fireAlarm.subscribe(this(StakeHolder))` and thereby subscribe to events from `fireAlarm`.
- The object `fireAlarm` is declared as an instance of class `Alarm`.
- The `Observer` objects `FireDepartment`, etc. are declared as instances of class `Stakeholder`.

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.



SubjectObserver/

When the `Stakeholder` objects are generated, they all subscribe to events from the `fireAlarm`. At some point in time an external sensor may detect a possible fire and invoke `fireAlarm.alert`, which then invokes `notifyObservers`, which in turn invoke `notify` on all `Stakeholder` objects. This is illustrated by the following OSD.



fireAlarmOSD/

The invocations of `notify` 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 `FireDepartment` will then observe all the `FireAlarms` whereas the shops and owner of a given shopping center only observe the alarm in that center.