18.3 Subject and Observer Aspects

In this section, we introduce two related aspects, *subject* and *observer*, which have to be used in tandem. Objects of classes with a subject aspect may be observed by a set of objects with an observer aspect. An observer object subscribes to certain events in the subject object, events that typically change the state of the subject object. In a case of such an event, the observer will be notified.

As a first example, we use a a fire alarm in a shopping center with a number of shops. The fire alarm may be considered a subject to be observed by a number of stakeholders that include the nearest fire station, the owner of the shopping center and the shops in the center. As a second example, we show how to add subject and observer aspects to the bank system to monitor suspicious transactions on the accounts of the bank.

> In programming a *design pattern* describes a well-defined functionality of a program in terms of how to structure a program fragment implementing the functionality. Observer is an example of a design pattern.

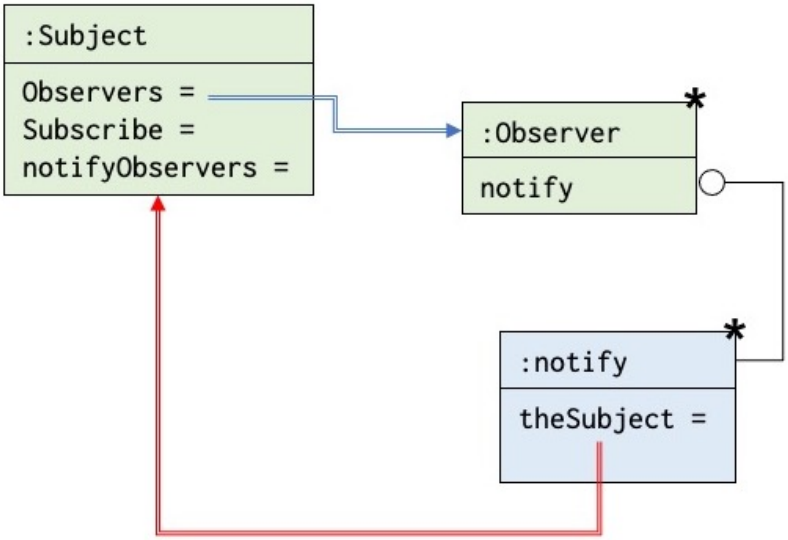
Before we show how to represent the examples, we introduce some general classes for representing subjects and observers represented by `Subject` and `Observer`. Objects being subject to observation shall have the properties of `Subject`, and observer objects shall have the properties of `Observer`. The structure and behaviour of class `Observer` and class `Subject` is known as the *Observer Design Pattern* – we thus place the two classes in a module called `ObserverPattern`:

```
ObserverPattern: obj
   class Subject:
      observers: obj Set(#Observer)
      subscribe(obs: ref Observer):
         observers.insert(obs)
      notifyObservers:
         observers.scan
            current.notify(this(Subject))
      inner(Subject)
   class Observer:
      class ObservedSubject:< Subject
      notify(theSubject: ref ObservedSubject):<
         inner(notify)
```

- A `Subject` have a `Set` `observers` containing `Observer` objects.
- It has a method `subscribe` that makes an `Observer` object an element of `observers`;
- It has a method `notifyObservers` that is called in case the observeres should be notified about changes, so it will for each `observer` in `observers` invoke its `notify`.
- An `Observer` has a local virtual class `ObservedSubject` that can be extended with further attributes in subclasses of the observed `Subject`.
- It has a method `notify`, which has a parameter, `theSubject` of type `ObservedSubject`, which is a reference to the `Subject` that invokes `notify`.

The figure below illustrates that a `Subject` has a set (`observers`) of references to a number of

Observer objects; this is illustrated with the '*' in rightmost top of the Observer object. The figure also illustrate that when notify is invoked on these Observer objects, the parameter theSubject of each notify method object is set to denote the Subject object that invoked notify:



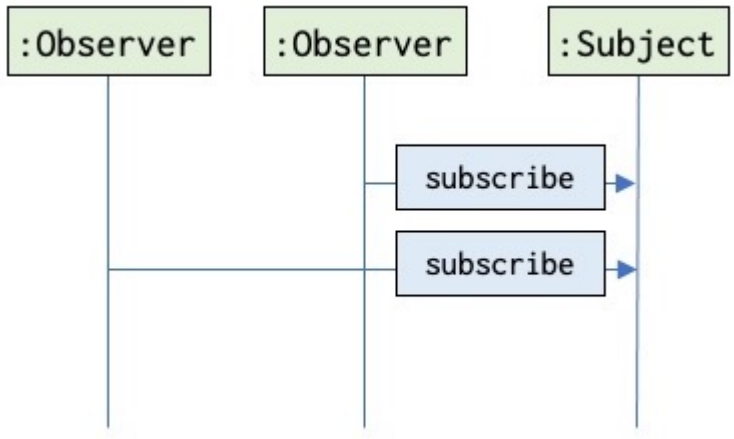
SubjectObserver

The dashed line indicates that each notify method object is an instance of the method notify of a given Observer object.

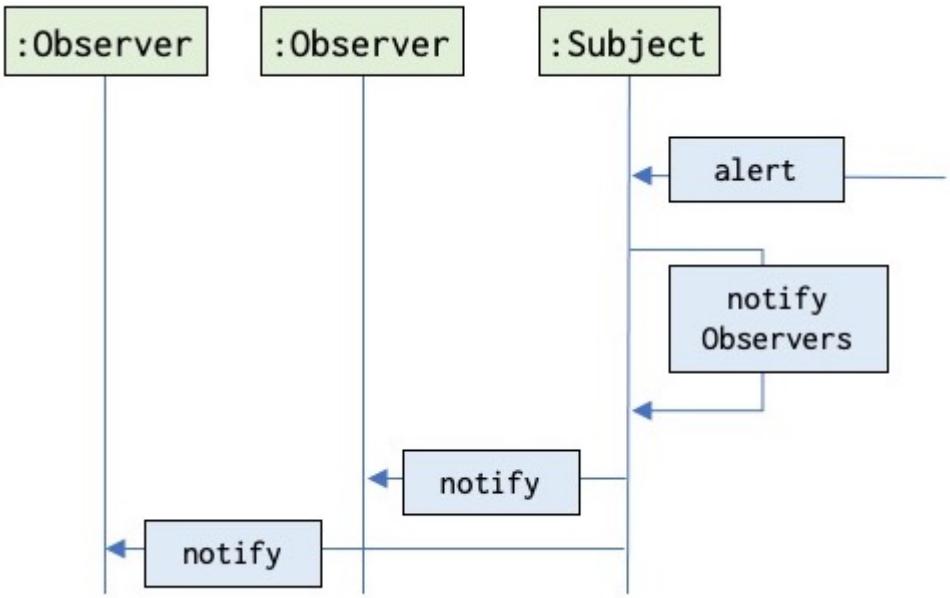
The following two OSDs show what happens between objects of classes Observer and Subject.

Observer objects subscribes to events in a given Subject object, by invoking the method subscribe.

For a given Subject object there may be a number of Observer objects; here we have just illustrated the case with two Observer objects.



In case of an event in the Subject object, either caused by a state change in the Subject object, or by an external alert event (e.g. from a sensor), the Subject object invokes its notifyObserver method leading to invocations of notify of all Observers.



The classes `Subject` and `Observer` are rather abstract so in the next section we will show how the above example of a fire alarm in a shopping center may be modeled using the `ObserverPattern`. We show how to add subject and observer aspects to the bank system in the section following the next section.