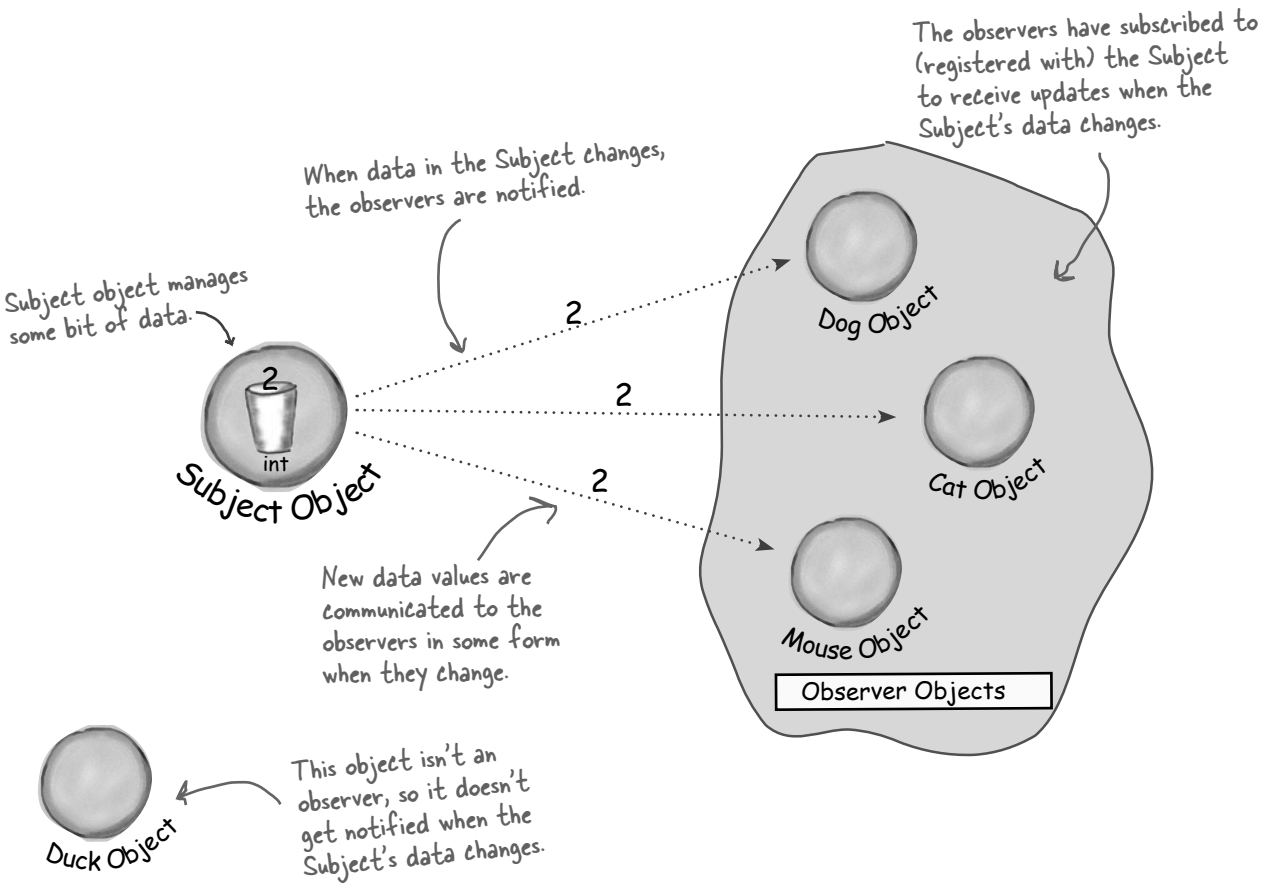
# Publishers + Subscribers = Observer Pattern

**If you understand newspaper subscriptions, you pretty much understand the Observer Pattern, only we call the publisher the SUBJECT and the subscribers the OBSERVERS.**

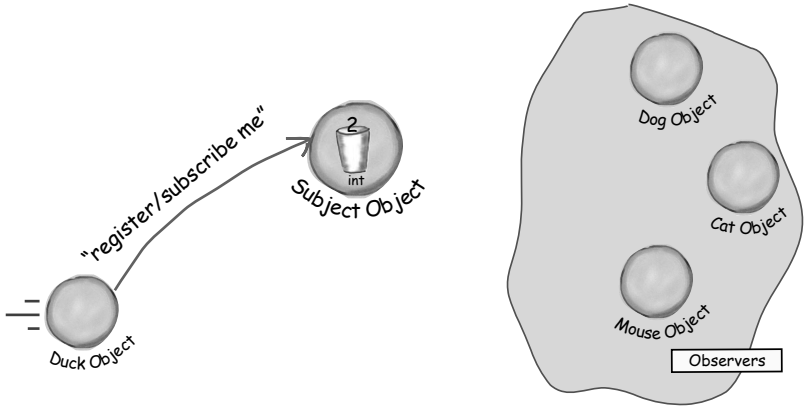
**Let's take a closer look:**



# A day in the life of the Observer Pattern

**A Duck object comes along and tells the Subject that it wants to become an observer.**

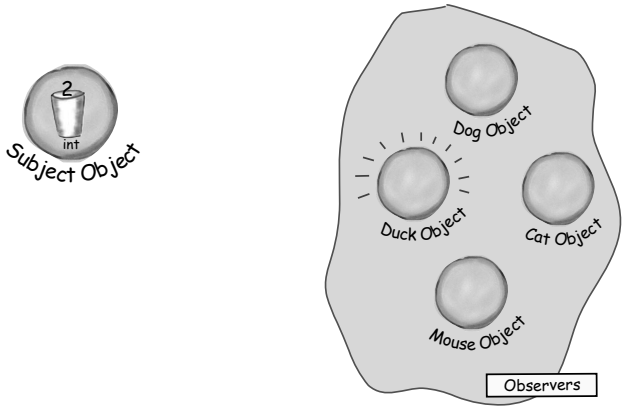
Duck really wants in on the action; those ints Subject is sending out whenever its state changes look pretty interesting...




---

**The Duck object is now an official observer.**

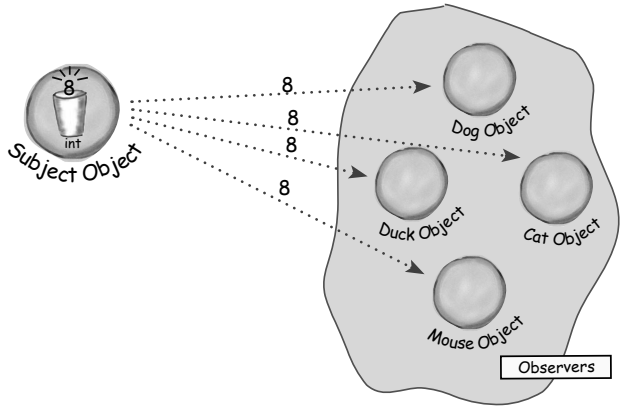
Duck is psyched... he's on the list and is waiting with great anticipation for the next notification so he can get an int.




---

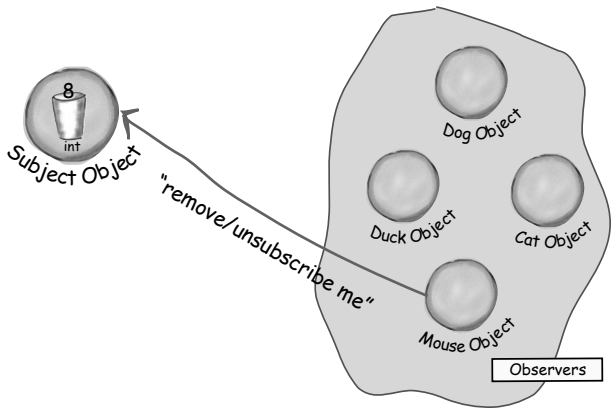
**The Subject gets a new data value!**

Now Duck and all the rest of the observers get a notification that the Subject has changed.



**The Mouse object asks to be removed as an observer.**

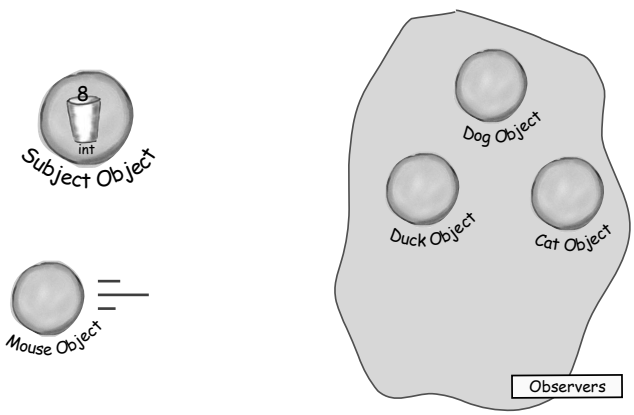
The Mouse object has been getting ints for ages and is tired of it, so it decides it's time to stop being an observer.




---

**Mouse is outta here!**

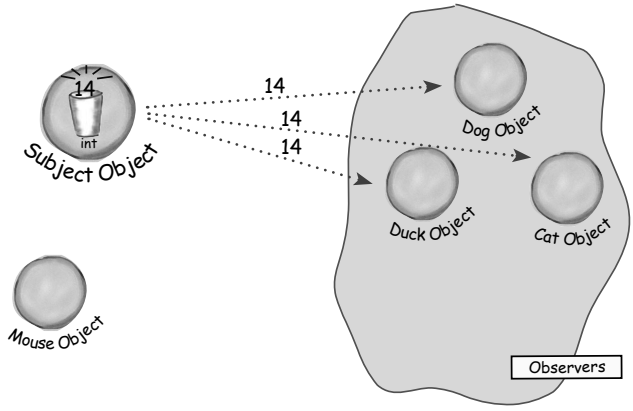
The Subject acknowledges the Mouse's request and removes it from the set of observers.




---

**The Subject has another new int.**

All the observers get another notification, except for the Mouse who is no longer included. Don't tell anyone, but the Mouse secretly misses those ints... maybe it'll ask to be an observer again some day.



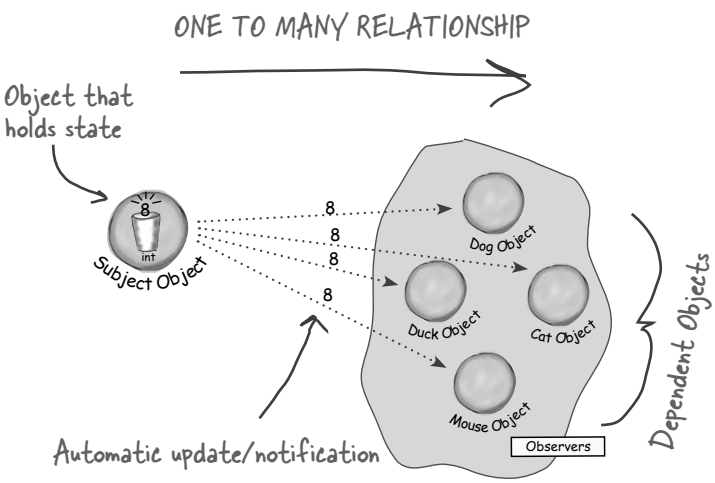
# The Observer Pattern defined

When you're trying to picture the Observer Pattern, a newspaper subscription service with its publisher and subscribers is a good way to visualize the pattern.

In the real world however, you'll typically see the Observer Pattern defined like this:

> **The Observer Pattern** defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

Let's relate this definition to how we've been talking about the pattern:




The subject and observers define the one-to-many relationship. The observers are dependent on the subject such that when the subject's state changes, the observers get notified. Depending on the style of notification, the observer may also be updated with new values.

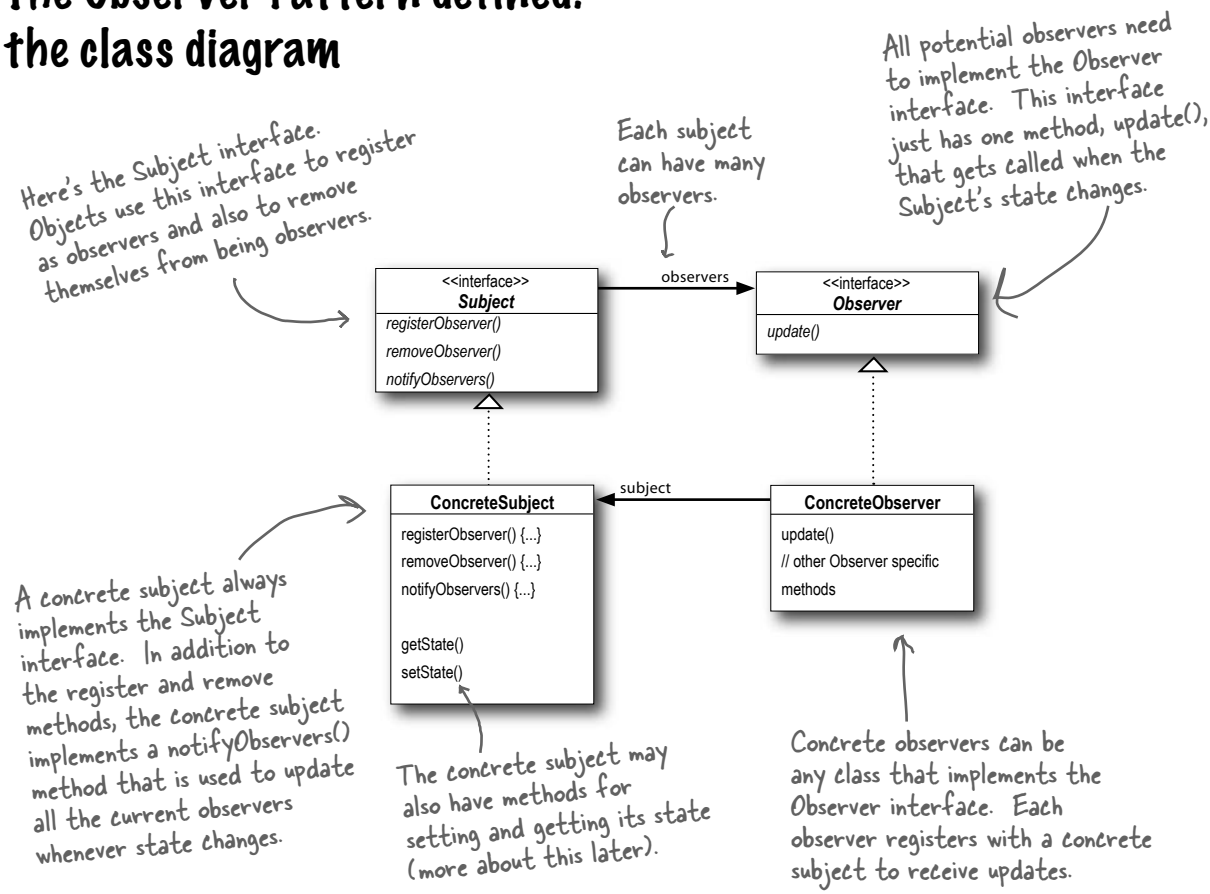
As you'll discover, there are a few different ways to implement the Observer Pattern but most revolve around a class design that includes Subject and Observer interfaces.

Let's take a look...

**The Observer Pattern defines a one-to-many relationship between a set of objects.**

**When the state of one object changes, all of its dependents are notified.**

# The Observer Pattern defined: the class diagram




## there are no Dumb Questions

**Q: What does this have to do with one-to-many relationships?**

A: With the Observer pattern, the Subject is the object that contains the state and controls it. So, there is ONE subject with state. The observers, on the other hand, use the state, even if they don't own it. There are many observers and they rely on the Subject to tell them when its state changes. So there is a relationship between the ONE Subject to the MANY Observers.

**Q: How does dependence come into this?**

A: Because the subject is the sole owner of that data, the observers are dependent on the subject to update them when the data changes. This leads to a cleaner OO design than allowing many objects to control the same data.