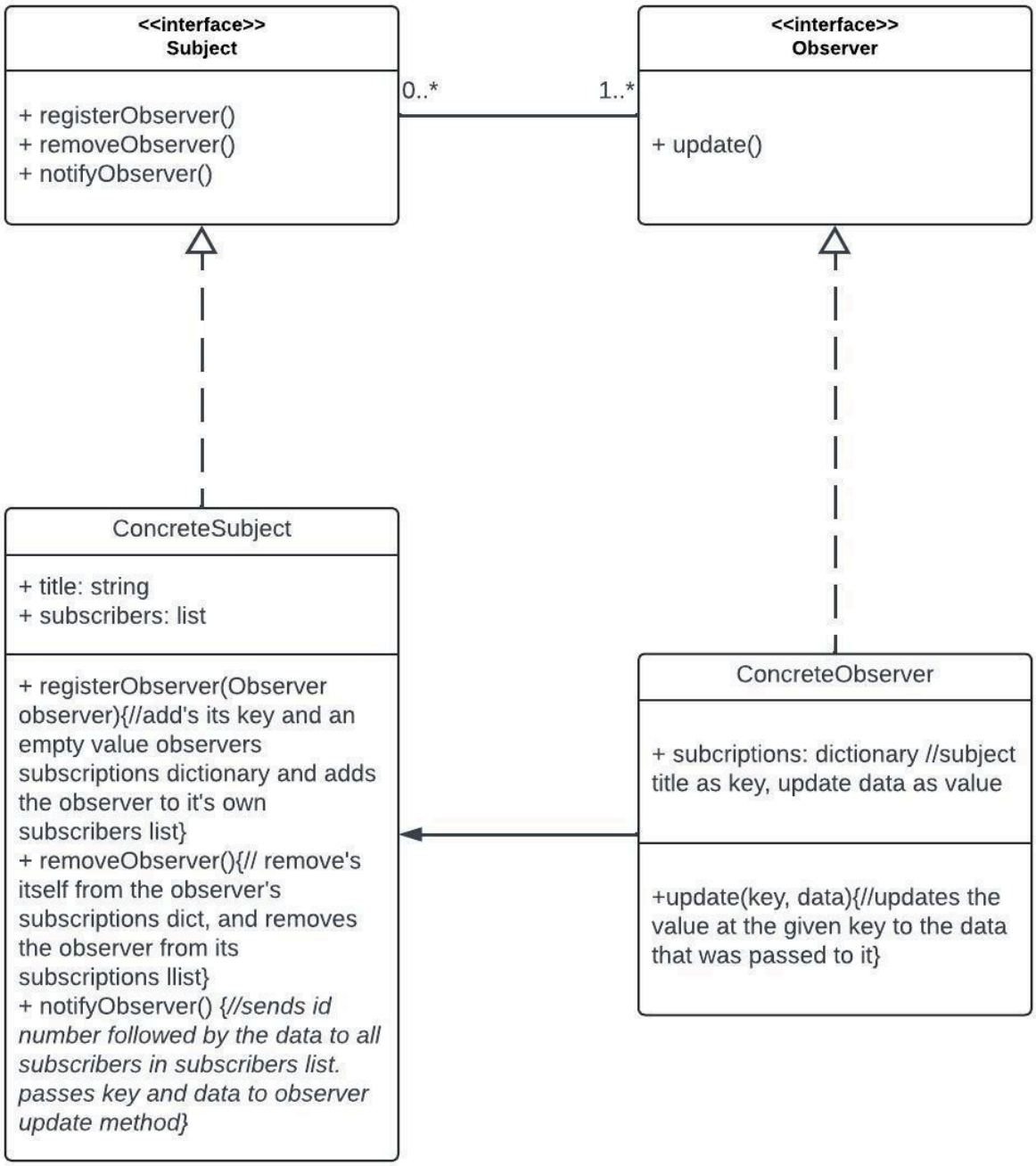

This version of the Observer pattern is one that I came up with, and supports a many-to-many relationship between subjects and observers. It does this in 3 key ways: a title for the subject, a subscriptions dictionary for the observer, and a slight change in the way that notify function works.

The title attribute of the subject allows specification between multiple subjects. For example, an observer could have 3 different subscription, and it could differentiate them by that title, such as “sports”, “weather”, and “news”. The title also works well because it can be specific. If you had *m* number of weather stations, the titles could be “station1”, “station2”, all the way up to “station*m*”. The pro of the title is that it allows for differentiation between infinite observers without changing the observer interface. A con of the title is that you need a way to handle, or prevent, duplicates.

A dictionary that stores all of an observers subscriptions builds on this. With the title, the subjects have a way to be differentiated. The subscriptions dictionary actually handles that differentiation. An observer stores its subscriptions with the subject’s title as the dictionary key, and the subject’s data as the dictionary value. This allows an observer to keep track of, and store data from, *m* subjects. A pro of this is that the dictionary is easy to implement and doesn’t change much about the observer interface. It allows subjects and observers to still be loosely coupled. Dictionary operations are also fast, which is useful as the dictionary grows. A con is that there is potentially a lot of data for the observer to store. If the amount of data to store is too much for the observer, a better option may be to have a class or interface external to the observer that stores the data, but that the observer can reference. That would potentially be more complicated to implement though.

Finally, the title and the dictionary, allow for a very easy notify function. When a subject wants to notify its obserers it can still go through it’s list and call update() on all of its observers. The subject title and data are then passed to that update function. The function finds the subject’s dictionary entry based on its key, and updates the value at that key with the new data. The pros of this is that for *n* subscriber, notify should operate in O(n) time, competing two O(1) operations per observer (one lookup, one insertion). Other pros are that this is simple to implement, doesn’t require additional classes or interfaces, doesn’t change the subject interface much, and keeps observers and subjects loosely coupled. So far, I have not been able to think of a reasonable con to this approach.