Observer Pattern

Contents

1. Problem
2. Solution Idea
3. Class Diagram
4. Concrete Example
5. Benefits
6. What you need to remember

Problem

- Given two objects. One of them (observer) wants to know when something happens to the other object (the subject).

- Example: DataTable and LineGraph. LineGraph wants to know when DataTable entries are changed.

Problem continued

- One solution – polling! (why not a good idea?)

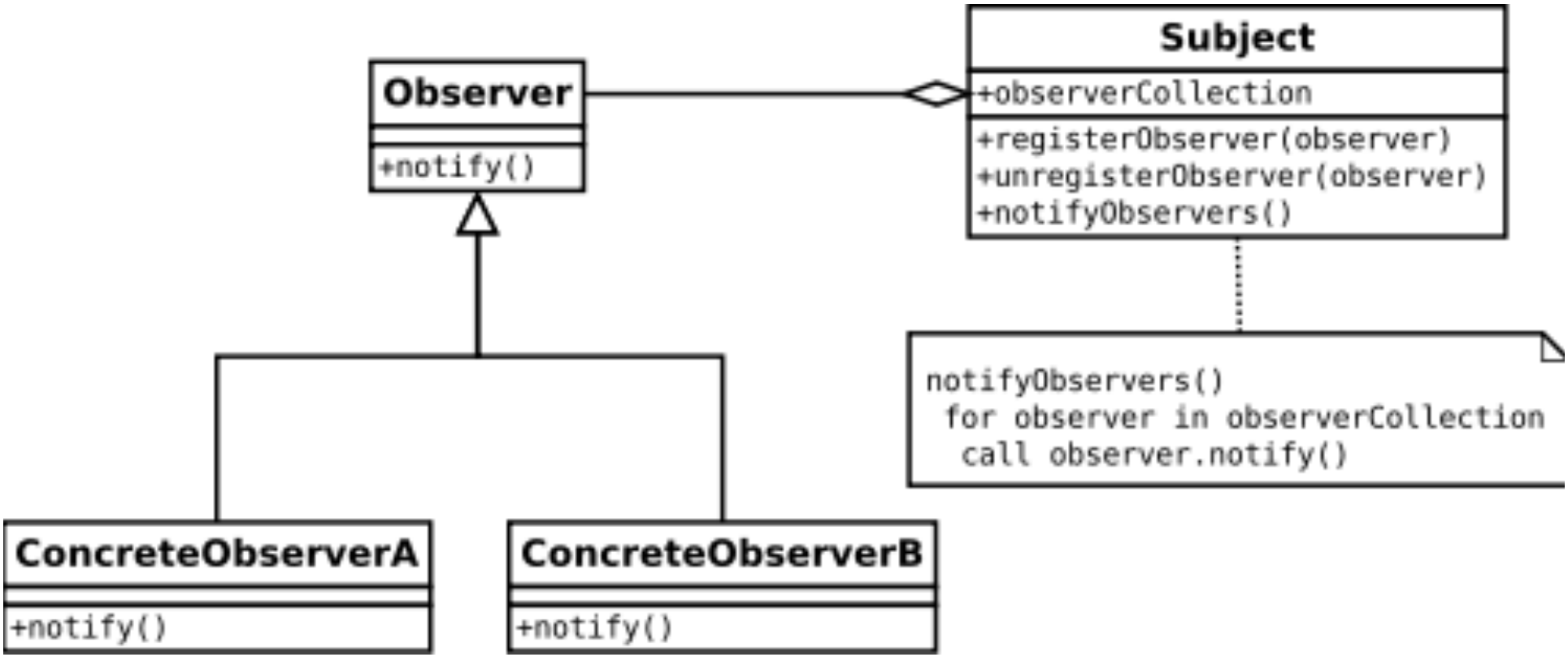
- Elements of a good solution:
  - Subject code should be unaware of specific observers (and their specifics) – why?
  - Many observers should be able to observe the same subject.
  - An observer should be able to observe multiple subjects.

Solution Idea

- Subject has a list of observers (which is an interface that specific observer objects will have to implement – that's why it does not know about specific observers).

- When some change happens a fireEvent method is called.
  - The method goes over the list of observers and calls their handle/notify methods one by one.
  - observers can register/unregister themselves from a subject

Class Diagram

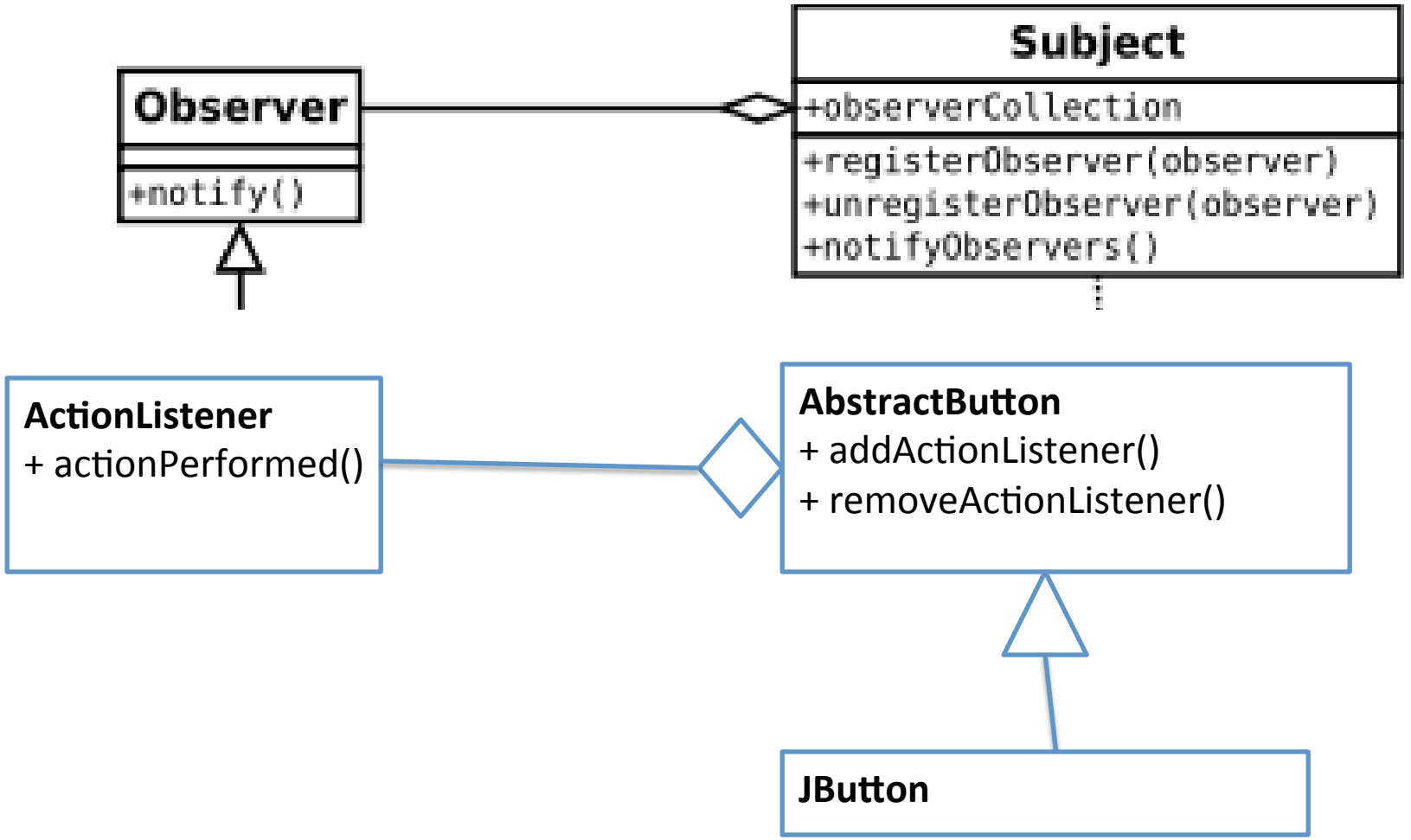


CONCRETE example

- Jbutton is Subject
- addActionListener() is registerObserver()
- removeActionListener() is unregisterObserver()

- ActionListener Interface is **Observer**
- actionPerformed method is **notify()**

Class Diagram



BENEFITS

1. when subject is changed (or event happens)… all observers are notified of change automatically.
2. new observers can be added without ANY change in code for subject.
3. an observer can observer multiple subjects for events
4. subject doesn't know about specific observers (they are very loosely coupled).

What you need to remember

- You can use this pattern in your own code as well!
  - Note that you can use java.util.Observer and java.util.Observable interface in your own code (i.e. non Java Swing code)
- Understand all the benefits of using this pattern.