Inf2-SEPP: Lecture 12 Part 1: Design Patterns: Observer, Singleton

Cristina Adriana Alexandru

School of Informatics
University of Edinburgh

Previous lecture

- Design patterns
  - Introduction, cautions
  - The MVC Pattern
  - The Command Pattern (a behavioural pattern)

This lecture

Design patterns continued

- The Observer Pattern (a behavioural pattern) and the Singleton Pattern (a creational pattern)
  - The problem
  - Details
  - Benefits
  - Drawbacks

Observer pattern: the problem

Context: detailed design

Problems:

1. Maintaining state consistency between a set of cooperating classes, i.e. *dependant* classes being informed about the state changes of *subject* classes.
2. Easily adding and removing dependants without changing the subjects (i.e. not knowing of who dependants are).

Example: Changing the way information on students (the subjects) is presented in a bar chart vs pie chart (dependants).

Observer is often used in event driven software, and in MVC pattern to represent the 'view' part.

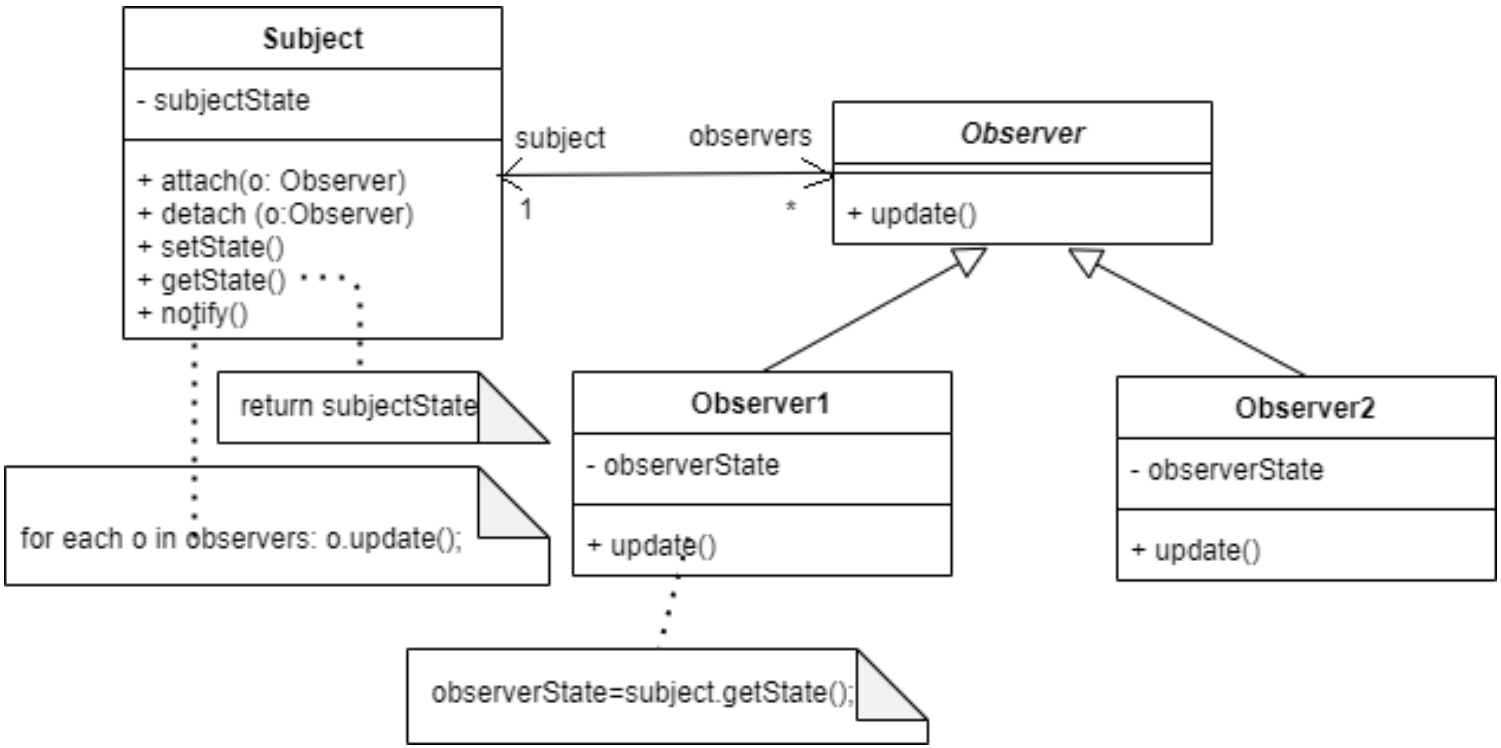
Observer pattern: the problem

Naïve solution for problem 1): associating each subject with each of its dependants

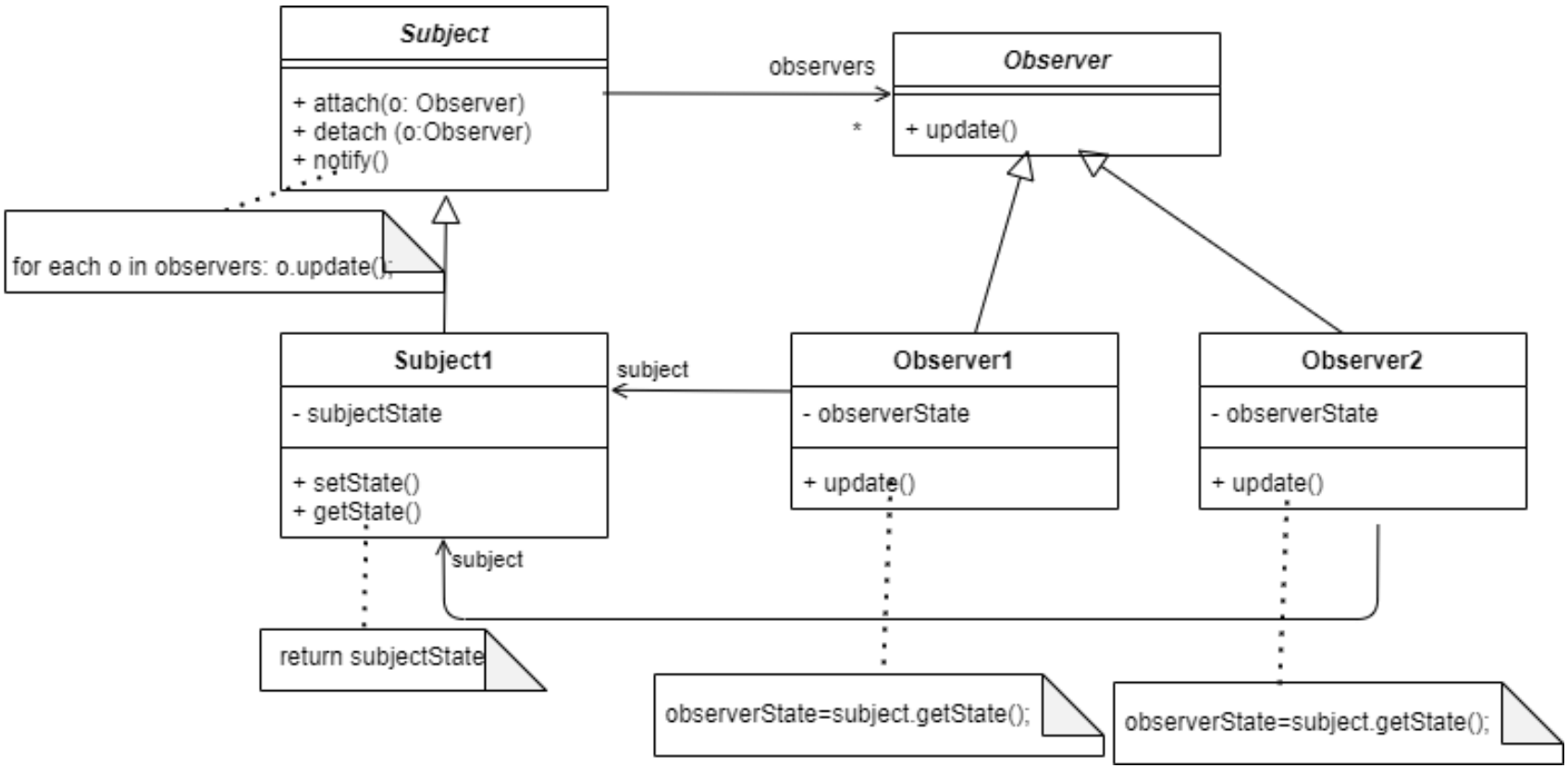
BUT this leads to *tight coupling*, not respecting problem 2):

- The subject must know of its depepdants and their number
- The subjects may need updating when dependants are updated.

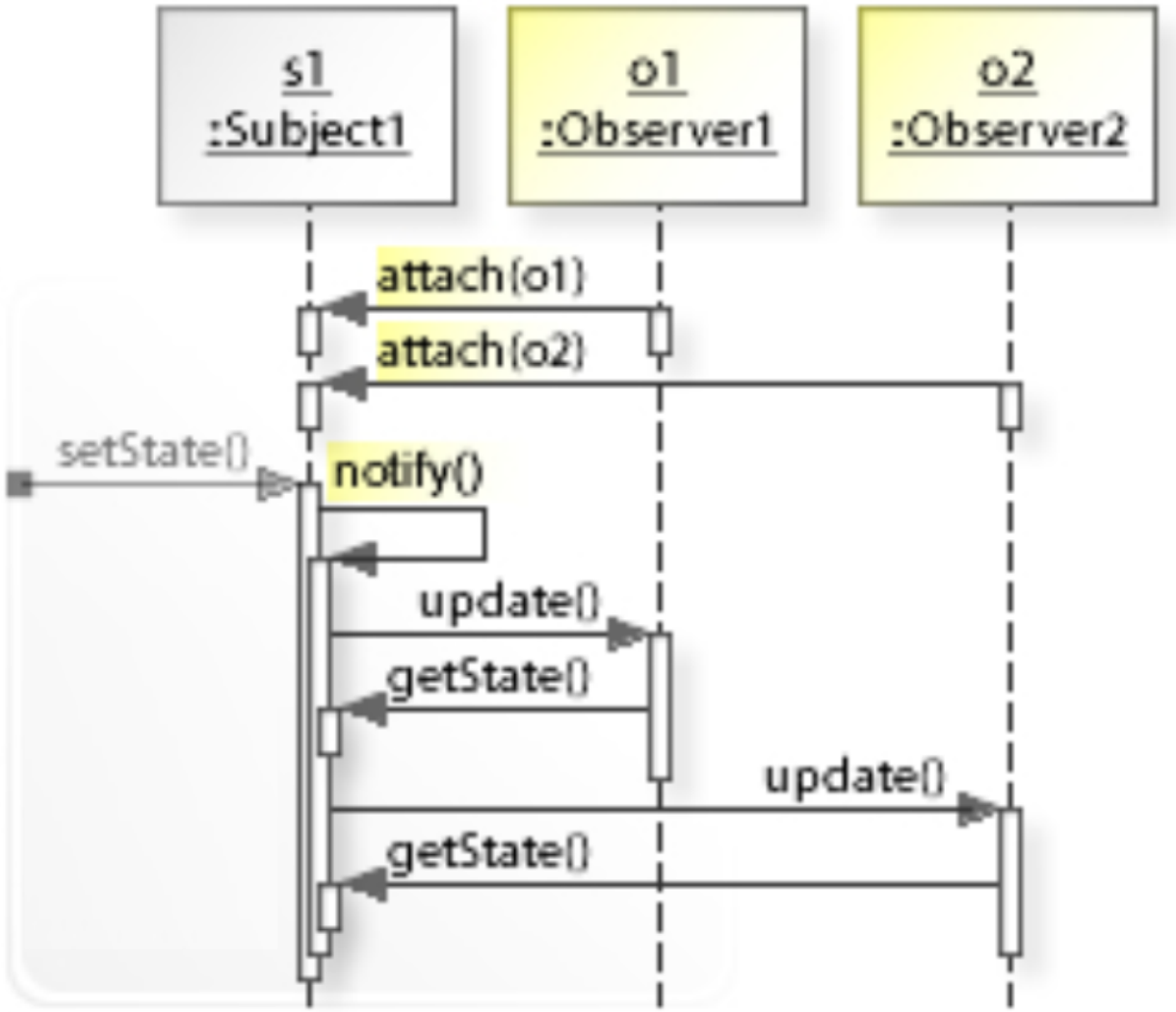
Observer pattern with one subject



Observer pattern with potentially more subjects



Observer pattern with potentially more subjects



Observer pattern: Advantages and disadvatages

Advantages:

- Abstract, minimal subject-observer classes coupling
- Support for broadcasting, without the subject needing to know and inform each observer

Disadvantages:

- May lead to cascades of updates which are difficult to track
- Costly in terms of space if many subjects and few observers; One solution: use of hash maps, costly in terms of time.
- Ending up with dangling references to deleted subjects/ observers; One solution: notifying when deleted
- Risk of having an inconsistent subject state before notification

Singleton Pattern: The problem

Context: detailed design

There are situations in which we want for a class to:

1. Have a single instance
   E.g. A single log of all all system actions may be wanted for a system at any one time.
2. Offer global access to it
3. Protect it from being overwritten

Singleton Pattern: The problem

Naïve partial solution to problem 2 in other programming languages than Java: global variables can make objects accessible

BUT:

- You could still instantiate several such objects (breaks problem 1)
- They can be overwritten, so very unsafe (breaks problem 3)

The Singleton Pattern is used to address this problem. It is often used for logging, driver objects, caching, and in many other patterns.

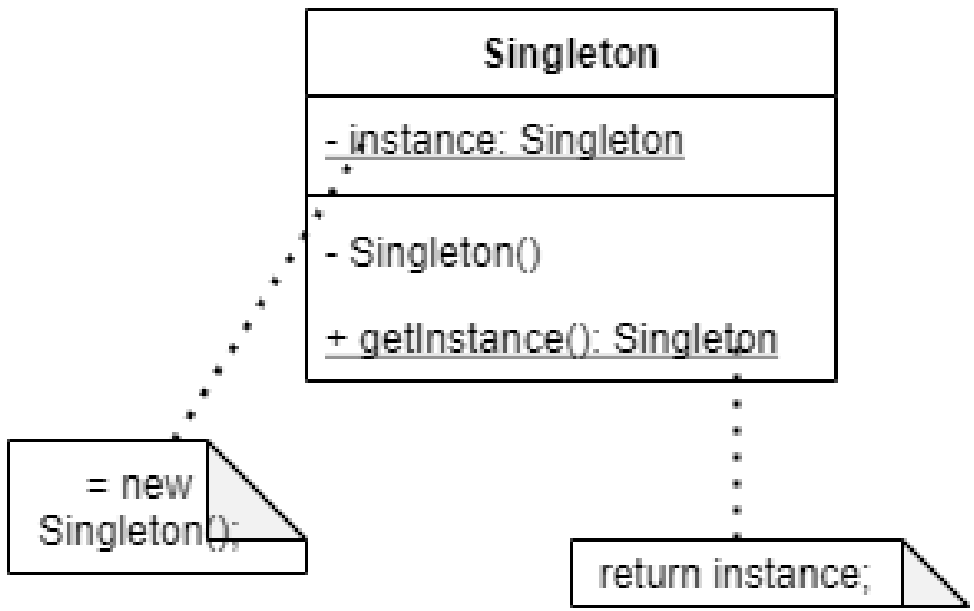
Singleton Pattern: Details

There are many versions of Singleton, but they all share the following main ideas:

- Make the class itself responsible to keep track of its sole instance, by hiding its constructor (using *private* in Java)
- The class offers a way to access the instance, through a static operation (getInstance()) which returns the sole instance of the class

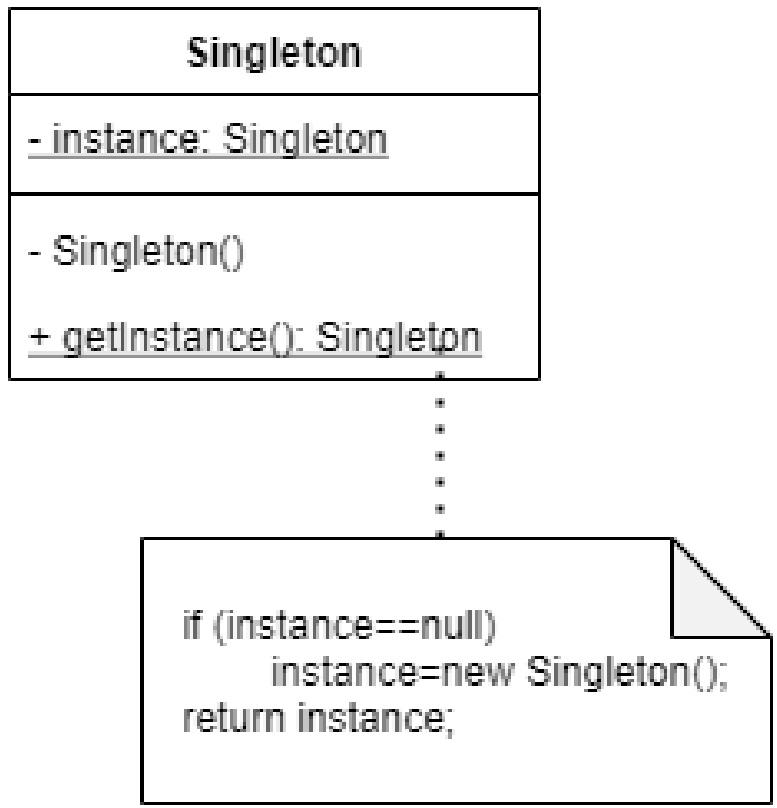
Singleton Pattern: Eager initialisation

The instance is created at first loading of the class (even if not needed).



Singleton Pattern: Lazy initialisation (for single threaded systems)

The instance is created the first time the global creation operation is called.



Singleton Pattern: Advantages and disadvatages

Advantages:

- Offers controlled global access to a sole instance of a class
- The object is initialised only once
- Preferred over global variables: avoids polluting the name space, permits lazy allocation and initialisation
- Can be easily changed to allow more instances of the class, by editing the getInstance() operation

Disadvantages (leading some to frame it as an anti-pattern):

- It is frequently misused, adding unnecessary restrictions
- Introduces global state, potentially unsafe
- Leads to tight coupling between classes in your application
- Multiple threads may create multiple objects
- Its private constructor and static operation make it difficult to produce mock objects needed for unit testing

Resources

Essential: Read about the Observer and Singleton patterns:

- If you can get a copy of Gamma, E., 1995. Design patterns: elements of reusable object-oriented software. Pearson Education India: p. 326-337 "Observer", p. 144-146 "Singleton"
- On the Observer pattern: from Source Making and Wikipedia
- On the Singleton pattern: from Refactoring Guru and Wikipedia