The Abuse of Design Patterns

Mathieu Nayrolles

mathieu.nayrolles@gmail.com

March 24, 2014

From SOEN-6461

- Design Styles ☺
- Design Patterns ☺
- Anti-Patterns ☹

As a conscientious students / software professionals we will now:

- Think which style fit the best our domain model.
- Apply design patterns to solve our implementation problems.
- Have Anti-Patterns in mind while designing / implementing and try to avoid them.

Case Study — In Java

System.out.println("hello world");

- Seems too simple simple.
- I maybe miss something here and the professor / client will point it out.
- Can I do this with some design patterns ?

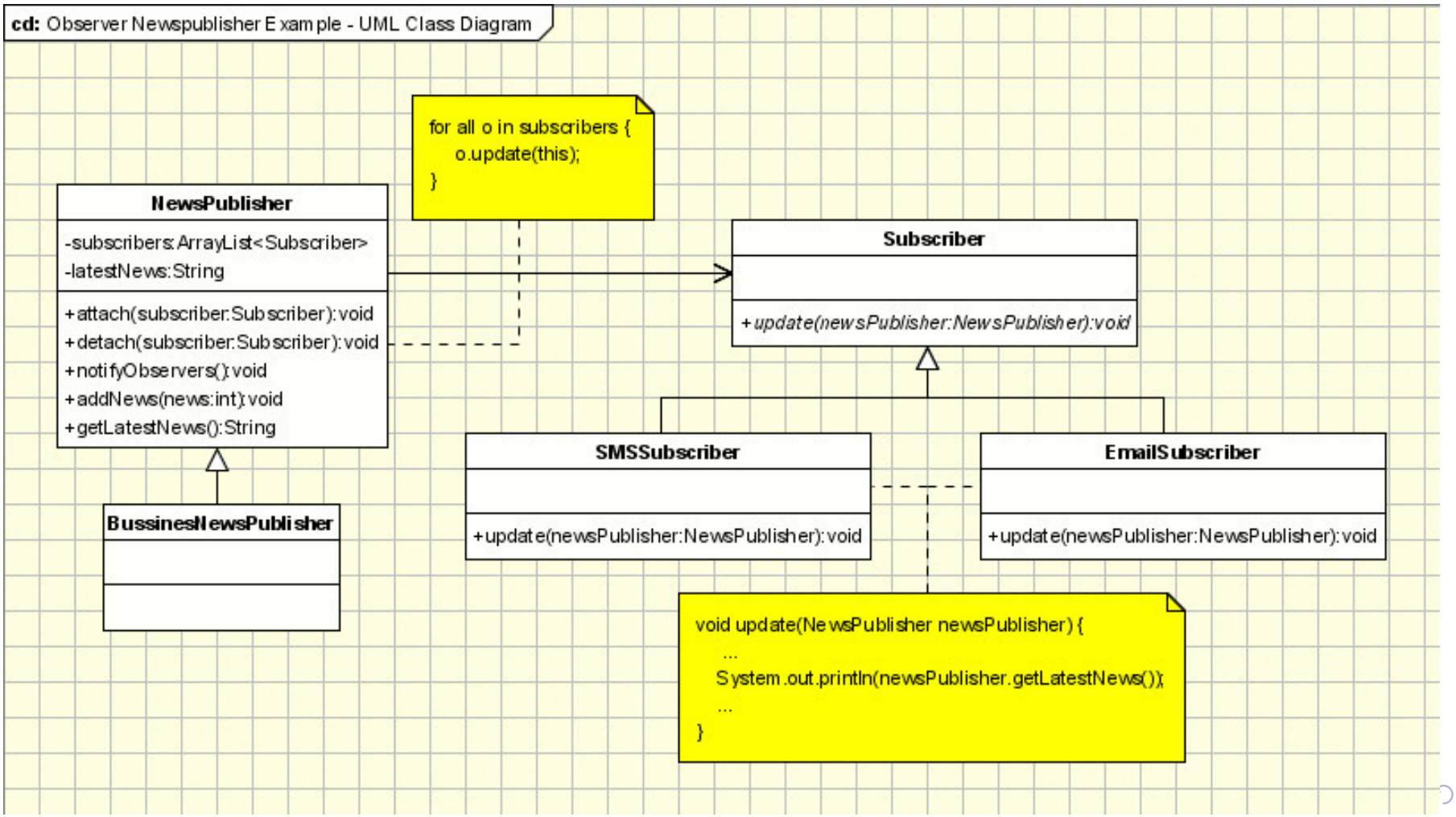
The candidate patterns:

- Observer
- Command
- Abstract Factory
- Singleton

4 design patterns! Obviously, a great design. Let's see.

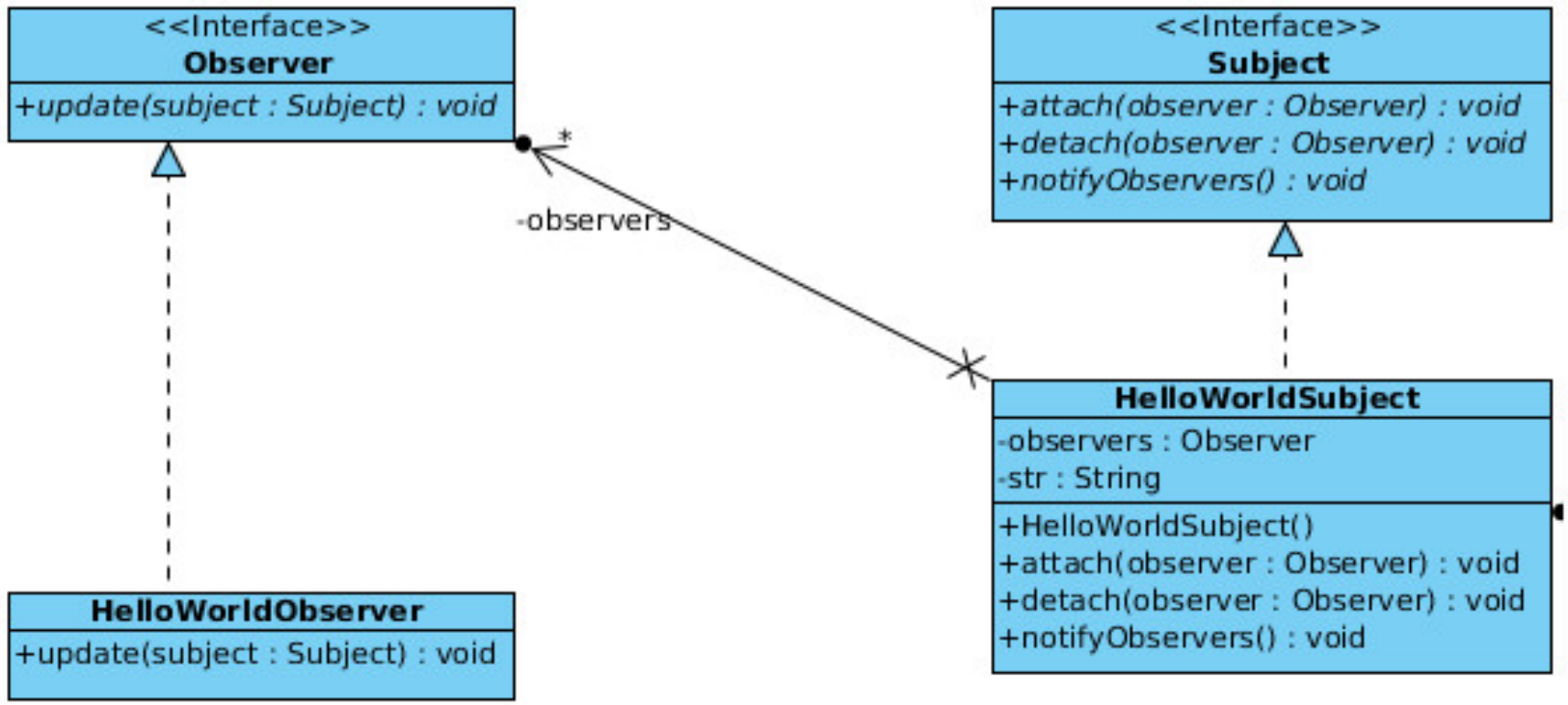
Candidate patterns — Observer

Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



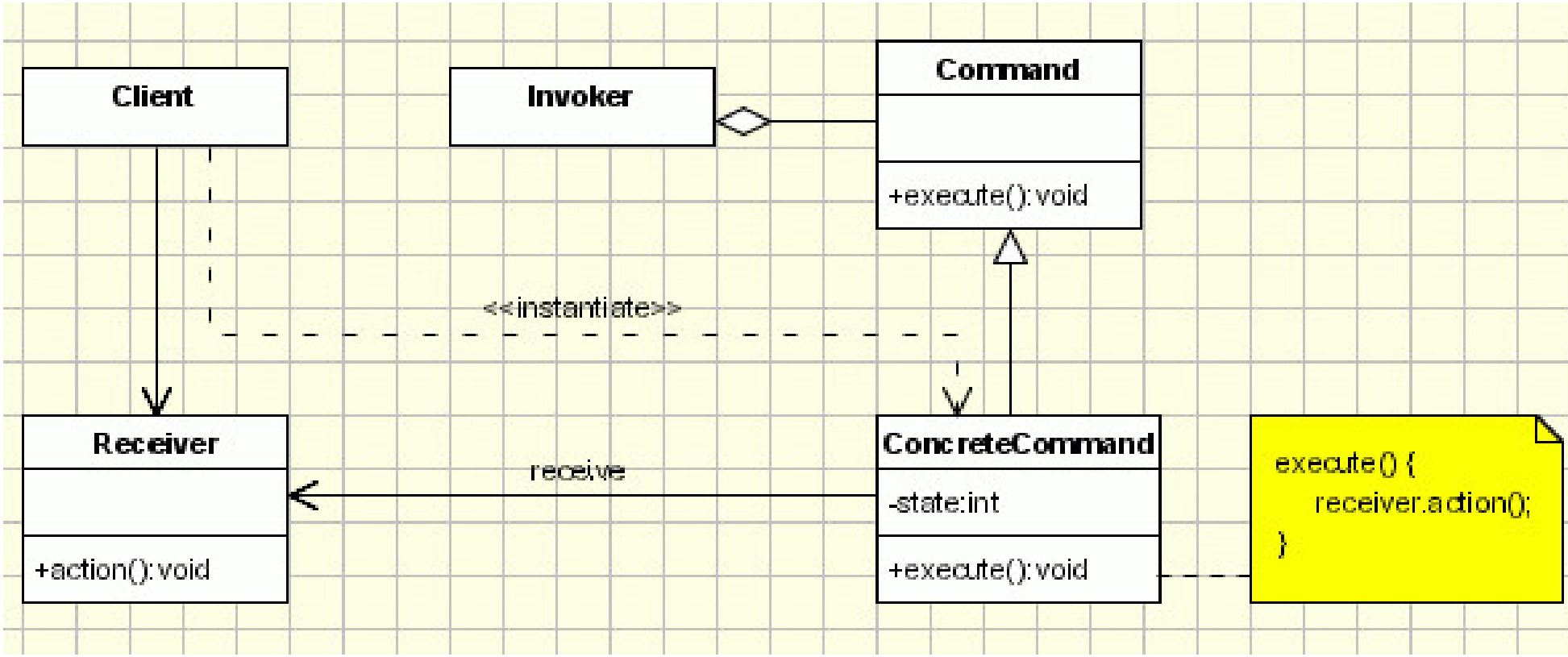
Candidate patterns — Observer (cont'd)

- Two interfaces Subject and Observer to add Observer.
- Two classes HelloWorldSubject and HelloWorldObserver that implements them.



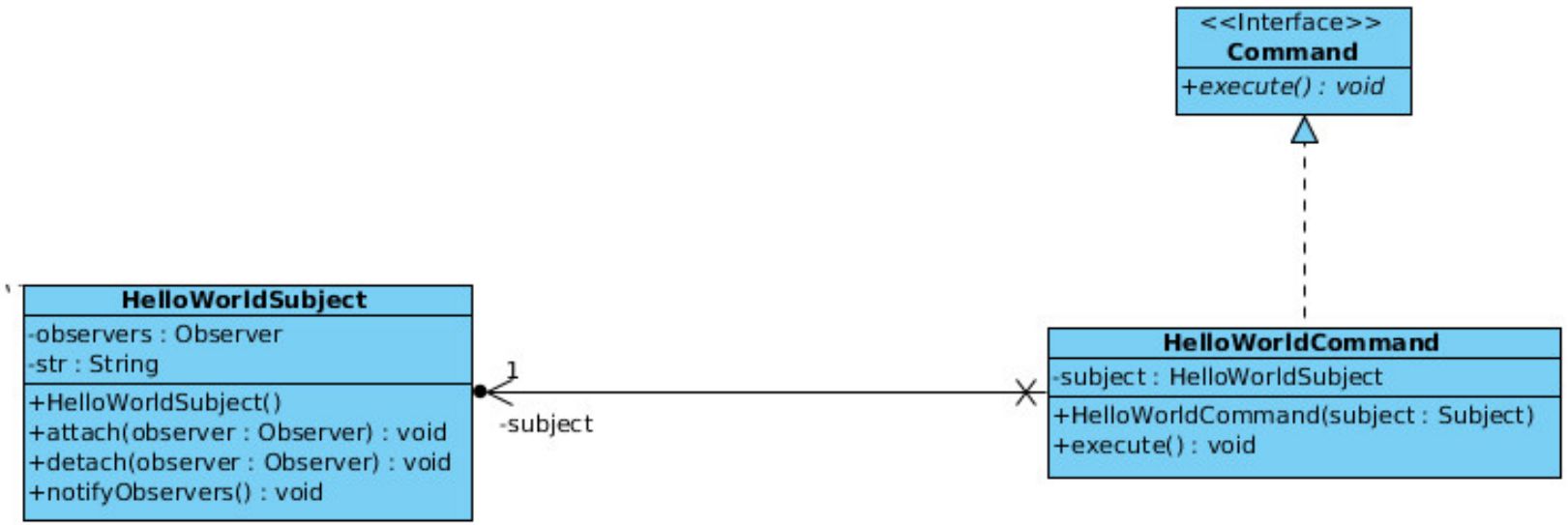
Candidate patterns — Command

- Encapsulate a request in an object
- Allows the parameterization of clients with different requests
- Allows saving the requests in a queue



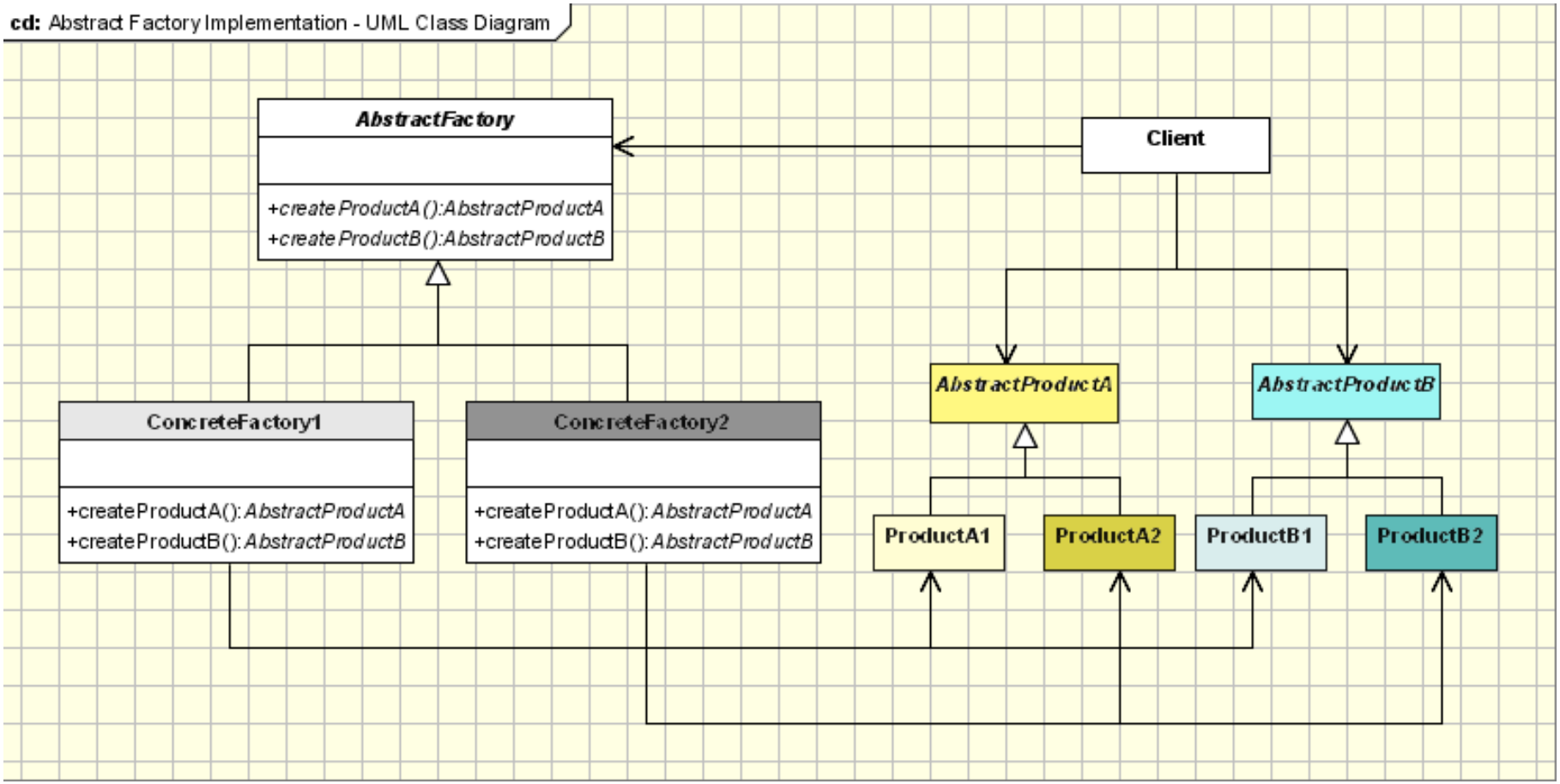
Candidate patterns — Command (cont'd)

We want the string "hello world" to be passed as a command.



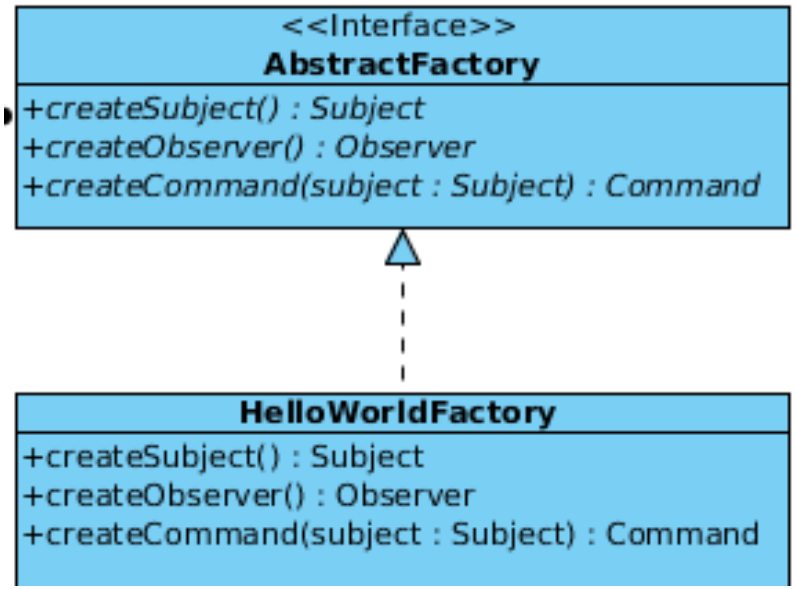
Candidate patterns — Abstract Factory

Abstract Factory offers the interface for creating a family of related objects, without explicitly specifying their classes.



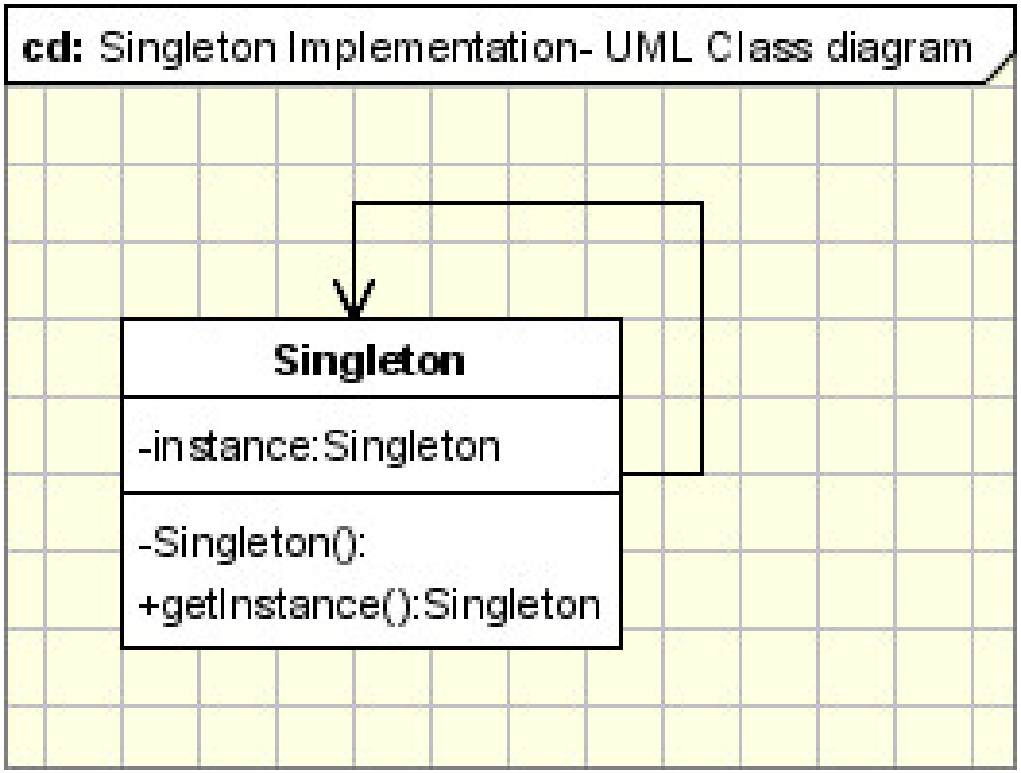
Candidate patterns — Abstract Factory (cont'd)

We create the observer, the subject and the command with this abstract factory.



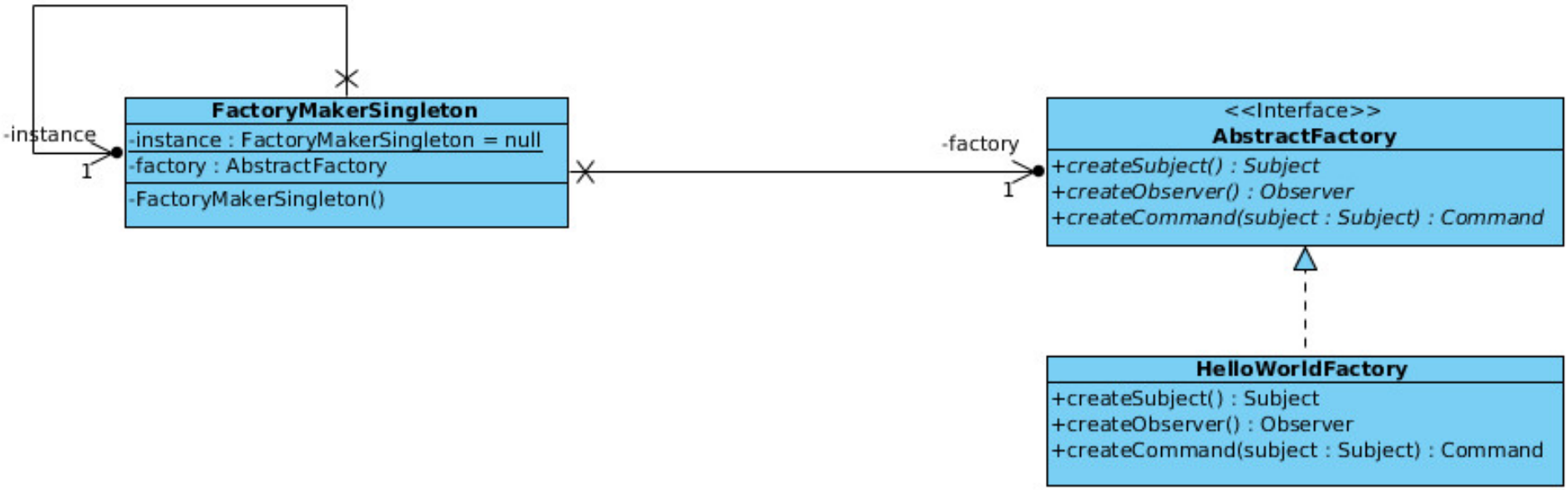
Candidate patterns — Singleton

- Ensure that only one instance of a class is created.
- Provide a global point of access to the object.

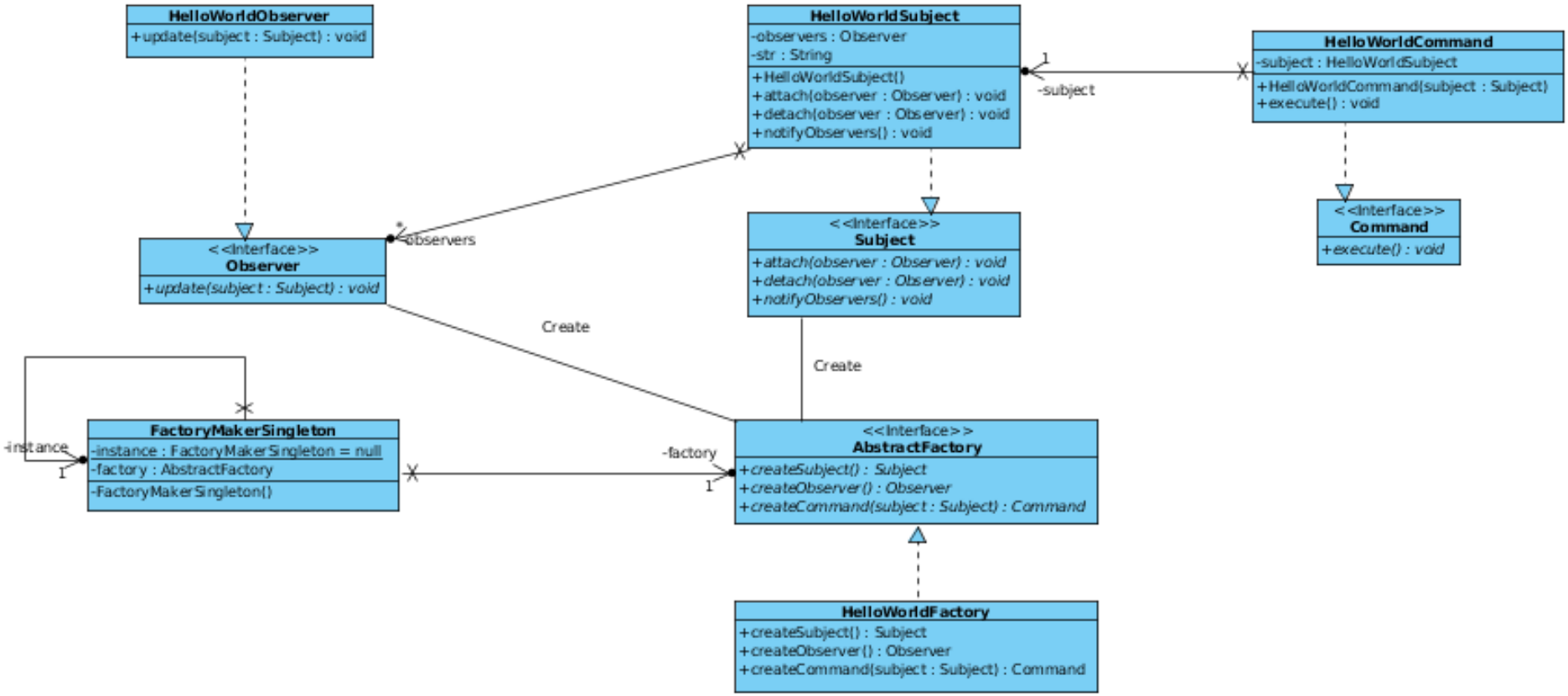


Candidate patterns — Singleton (cont'd)

We want to be sure that only one factory is created. To do so, we create a FactoryMakerSingleton.



Final Design



- Is it a great design ? (4 DP)
- If so, why ? If not, why ?

Conclusion

- The code is too complex for a hello world program.
- It contains the flexibility that we will never need.
- The time spent in designing and implementing is a total waste.
- There is a class explosion (9, without the Main).
- It violates the KISS principle.
- This does not serve the purpose of the problem.

Use patterns where they are natural, do not try to use them anyway. Design Patterns is a **great tool** to build great software. Use them **wisely**, do **not** abuse them.

Some citations

"When people starts learning design patterns, they try to use patterns everywhere. They try to use patterns anyway, it does not matter whether a pattern is required or not. They think that the more patterns are used, the better is the design. The outcome is a code with unnecessary complexity."

— **Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra, Elisabeth Robson** in *Head First Design Patterns* (2004)

Some citations (cont'd)

"One comment I saw in a news group just after patterns started to become more popular was someone claiming that in a particular program they tried to use all 23 GoF patterns. They said they had failed, because they were only able to use 20. They hoped the client would call them again to come back again so maybe they could squeeze in the other 3.

Trying to use all the patterns is a bad thing, because you will end up with synthetic designs-speculative designs that have flexibility that no one needs. These days software is too complex. We can't afford to speculate what else it should do. We need to really focus on what it needs."

— **Erich Gamma** in *How to Use Design Patterns* (2005)