TDT 4240
Software Architecture

Exercise 2: “Patterns”

David Rozas
María Fernández

Changes in the code

Exercise 1A

To show both billboards when the program starts, it was only necessary to create an instance of the class LargeBillBoard in the class MovieInfoSource and call its method showNextMovie into the method showMovieInfo of the class MovieInfoSource.
We have also made a copy of the source file text (data_utf8.txt) with UTF-8 as character-set, in order to display properly Norwegian characters.

Exercise 1B

For this exercise we created a class NovaBillboard, with similar features as SimpleBillboard. We had to make a decision about where to put the logic of the program: in the method showNextMovie of NovaBillboard or in the method showMovieInfo of the class MovieInfoSource.
We decided to put it into the NovaBillBoard class in order to respect the encapsulation philosophy. In this way, each class is responsible of implement its own information. The disadvantage of this solution is all the information is broadcasted to all the Billboards in any case, so this could be a problem in distributed systems in the link between source and billboards (overload, bottleneck, ...)
Another approach could have been to place the logic of each billboard in showMovieInfo. In this way, the source would have decided which information is sent to whom, saving resources in this flow of data between source and billboards. But this solution is not scalable, so we have considered that this feature is more important.

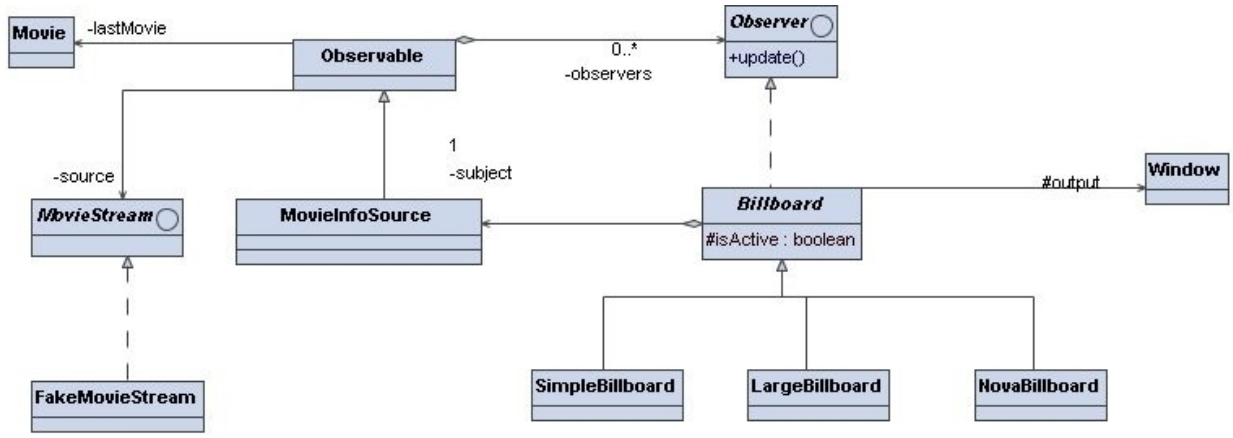
Exercise 1C

For implementing this section we created a method called isVisible in the abstract class Billboard. We had to add some methods in Window class to be able to dispose, get the title and make the frame visible.
In order to force the exit of the program in case of all billboards are disposed some logic was added in MovieInfoSource.

Part 3

Exercise 3A

We have considered that the observable object is MovieInfoSource, the observers should be the Billboards, and every time last movie changes the observers are notified. Each Billboard subclass heritages this from its parent class, so the update method is only implemented in the abstract class.
We have also changed the method main in Billboard to add only as observers the active billboards.
An UML-diagram showing the most important classes is showed below.



Exercise 3B

After using the model/observer pattern, the addition of new billboards would be much easier, because regarding to the main logic of the program we have only to instantiate and add them as observers, without being necessary to change any other part of the code in the internal classes. Meanwhile in the former model it would have been necessary to create a new attribute and perform some changes in MovieInfoSource, increasing the level of coupling.

Exercise 3C

As we previously mentioned, the use of this pattern provides better scalability, flexibility, expandibility, etc.

Exercise 3D

In this case the use of this pattern becomes absolutely indispensable, because the initial code would not scale, and it would be very hard to maintain, test, debug, etc.