Patterns

Hey guys,

we implemented the Factory-Pattern.

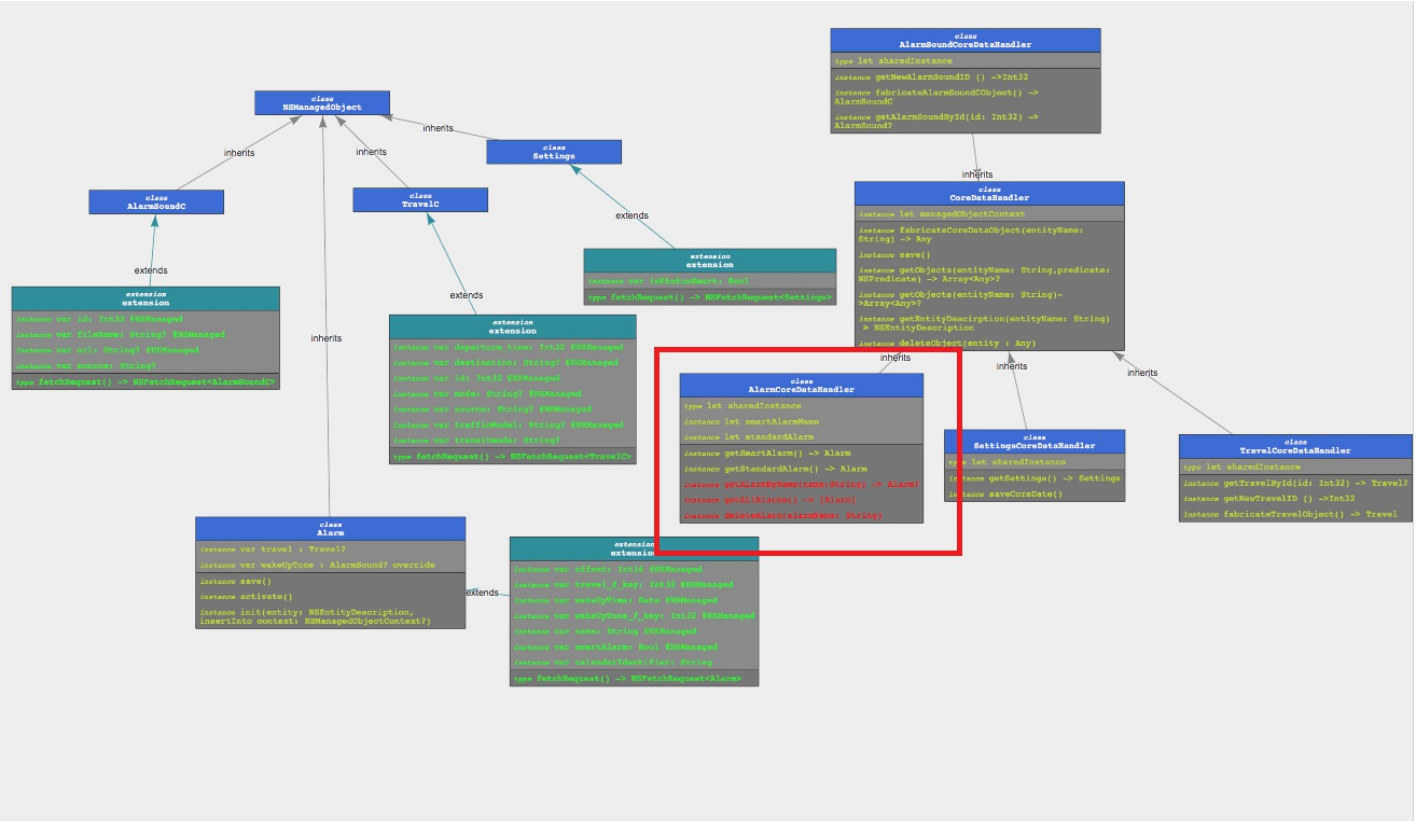
Our reasons for the Factory-Pattern:
– Wrap the constructor
– Generate a Core Data Object when generating a class instance

See the following code snippet showing the implementation:

```
func fabricateAlarm() -> Alarm {
    let alarm = self.fabricateCoreDataObject(entityName: "Alarm") as! Alarm
    alarm.id = getNewAlarmID()
    return alarm
}
```

(source)

See the marks in our class diagram:



(source)

Warm regards

← Refactoring – Fowler

Metrics →

2 thoughts on “Patterns”

Dannynator says:

18. May 2017 at 07:56

Hey guys,
nice job implementing design patterns! You showed the changes in the code and the spot where you applied those changes in the overall class diagram, just as required in the GC. It would also be very nice if you could explain the Factory-Pattern itself, why is it necessary and why it fits into your application. Otherwise it is a bit hard for someone outside of the project to get a clear idea of what you have done there 🙂
Also, when i click on the image it still kinda stays pretty small, so that it is not easy to read the text. Is it possible to make it a bit bigger or maybe scalable?
Hope your project is doing well and wish you all the best:)
Cheers!

Reply

Enrico Kaack says:

18. May 2017 at 08:17

Hey,

nice to see you are using the factory-pattern. It would be cool to see a previous/after UML diagramm to see the difference.

Best,
Enrico

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

POST COMMENT