Exercise: reverse engineering UML from code

October 15, 2015

Purpose

Check that you understand the semantics of UML – when a given body of Java code and a collection of UML models is consistent – by asking you to develop UML for selected parts of a body of code. This is sometimes called reverse engineering a model from a body of code, and done in order to have documentation going forwards. Here the main purpose is to bring up any questions or misunderstandings you may have about the meaning of the UML.

Besides that, I hope you can get further value out of studying this system. You may find this software good as a concrete context in which to consider the design principles and patterns we cover later in the course. Consider the code quality, too: can you identify things you'd ask to be modified if you were reviewing the code? (Consider contributing to the project!)

You will need access to the source code of the Borg Calendar, whose page is here:
`https://mikeberger.github.io/borg_calendar/`

Browsing the code on GitHub will suffice for this exercise, though you might like to download the code and build it, or even use it. You may also find the Javadocs (link at the above URL) helpful.

Building Borg Calender (if you wish to)

There are build instructions are down in the source directory at
`BORGCalendar/swingui/src/main/resources/resource/BUILD.txt`
but these are incomplete, so here are some supplementary notes, based on some welcome help I got from the system's author, Mike Berger!

Borg, and its dependency ical4j-connector, are maven projects. Learning maven is not an objective of this course, and anyway many of you are probably more familiar with maven than I am, but if not, it's worth learning a little about. For SEOC's purposes, though, following these instructions will suffice.

First grab and unpack the zip from
`https://github.com/mikeberger/ical4j-connector/tree/0.9.4.3-borg`
Build it *without compiling its tests* (because doing so doesn't work!) like this:
`mvn install -Dmaven.test.skip=true`

Then grab and unpack the zip from
`https://mikeberger.github.io/borg_calendar/`
and build it in the normal way, like this:
`mvn clean install`
(actually, you don't need the clean if you're only doing it from a fresh download, but it's conventional).

All being well, both of those stages should complete with BUILD SUCCESS, and then you can run borg (regardless of operating system you're on) by doing
`java -jar borg.jar`
from directory
`BORGCalendar/install/target/installer`

(The release notes in the same directory tell you how to do a more idiomatic thing for several OSs, in case you actually end up wanting to use this software!)

How to approach the exercises

For all of these, the purpose is to learn, not to produce a perfect final artefact. (I do not have a set of perfect final artefacts myself.) If you're spending a lot of time doing mindless boring aspects of this, change what you're doing. There are two distinct reasons for this: first, your time as students is limited and you should maximise the learning:time ratio; second, developers' time is also limited, and I want you to think about how people can do modelling cost-effectively.

Once you have attempted a question, and either feel happy with your solution or have got stuck, please go and join, or start if it hasn't been started yet, a discussion about it in Piazza.

1. Study the code in package `net.sf.borg.model.entity`. Draw a UML class diagram representing the contents of this package. Think about the usefulness of what you're doing: e.g., feel free to write "..." rather than list a zillion attributes in a class. But do think about how to precisely represent as many different kinds of information as you can, and see what questions arise as you do so.

2. Now consider package `net.sf.borg.model`, and do the same there. What are the relationships between packages `net.sf.borg.model` and `net.sf.borg.model.entity` (a) in Java terms (b) conceptually?

3. Draw a single-scenario sequence diagram to represent what happens when an object of class `Memo` receives message `decrypt(''password'')`. Assume that in this scenario `isEncrypted()` returns `true`.

4. Draw a sequence diagram using fragments to represent all possible scenarios when an object of class `MemoModel` receives message `getMemos()`.

5. (Much harder.) Next look at class `TaskModel` in package `net.sf.borg.model`. Study the `saveProject()` method. Note that the code is quite complex, but the Javadoc contains no detail about what work is being done and why. Consider broadly what kind of documentation you would like to inherit, if you were to take over as the maintainer of this project. Consider developing a sequence diagram for the behaviour of this method (in all scenarios, using fragments to represent alternatives, etc.). Will you represent every detail of the behaviour, or will you omit some details? Can you identify a principled way to decide what to include and what to omit, so as to produce a diagram that might be of use to a maintainer? Try to do so, and draw the resulting sequence diagram.

6. Draw a protocol state diagram for class `Memo`.

7. Draw a protocol state diagram for class `ReminderPopup`. Here you'll have a trickier job to decide what the states are, and you'll need to look at the superclasses as well as this class's own code.

8. Explore the functionality of the Borg Calendar to the point where you can describe an interesting use case for it, with some exceptional or alternate courses. Draw an activity diagram to describe the use case, paying attention to where there are restrictions on the order of activities and where not.

9. (Design, rather than UML.) Have a look at the main `Borg` class, especially its `main` method. What's going on with the `getReference` method? How else could this code have been written? Why do you think it's designed this way it is? Do you approve?

10. (Even more optional than the above.) Choose a UML tool to download and play with. (The lists of them all tend to get outdated, but two possibly useful starting points are `http://modeling-languages.com/uml-tools/` and `https://en.wikipedia.org/wiki/Category:UML_tools`.) Have a go at recreating some of your diagrams in the tool. Comment on Piazza on what you tried and how you got on.