Matthew Thomas
CSCI e97 — Assignment 4 — Results

Did the application of the design patterns help or hinder your design and implementation? Please explain how.
For the most part, the design patterns greatly helped my design and implementation. While the Visitor pattern in particular was a bit tricky, all three I think were useful and instructive in completing the assignment.

The Singleton pattern was helpful when implemented "correctly". The concept was brought up in earlier assignments but my implementation had consisted of only instantiating one class myself. Of course, this doesn't prevent additional instances from being created, and following the pattern as outlined in *Head First Design* solves that issue. Conceptually it is easy to understand, but took a little debugging during implementation to get the interfaces talking to each other correctly. After that, being able to call AuthenticationService.getInstance() helped simplify the code.

The Composite pattern was most helpful in the design phase in terms of figuring out how to structure the Roles and Permissions. Having the constraint of needing to use the pattern helped guide my thinking and probably led to the best solution even had the composite pattern not been a requirement.

The Visitor pattern as I mentioned was the most challenging, but I do think it was the most rewarding. This was demonstrated as it came time for testing and checking my implementation. Unlike previous assignments where we had to implement several getItem() or printItem() methods, having the InventoryVisitor be able to handle the traversal saved a lot of time. This way, instead of having to write all new methods after completing a new class, I could just add a line to on the Visitor methods to incorporate the change and get the functionality "for free". Likewise for the AuthVisitor, designing and implementing once and then being able to utilize to check my work was a great time savings. Given the challenges I faced, I probably would not have continued down the Visitor pattern path had it not been a requirement, but I'm glad I stuck through it and got it working.

How could the design have been better, clearer, or made the implementation easier?
I think the design could have been better had I understood the Visitor pattern better from the start. My original design hand-waved over some of the methods needed to check access and permissions and defaulted to an idea of "the Visitor will take care of that" without a full picture of *how*. This was a topic of discussion in several section meetings, and a few Piazza threads, that really helped shaped my understanding of the *how*. In particular, Eric's comment on my design that "the AccessCheck Visitor should locate the matching auth token, then navigate to the associated user, and then check the associated entitlements…" helped clarify in my head, design, and then implementation the mechanics behind how a Visitor travels from piece to piece.

Any implementation changes that you made to your design and how the continue to support the requirements.
One implementation change that I made to the design was in response to the bootstrapping problem brought up in Piazza post @267. This post brought up two changes I ended up implementing: 1) adding AuthToken restrictions to most Authentication Service API methods; and 2) creating a "root user" and special initialization method.

While API restriction to the Authentication Service was not explicitly mentioned in the requirements, it seemed like a good idea to include. Not only does this addition mimic the Store Model Service in terms of restricting access, but it also makes sense for the Authentication Service to be authenticated. This continues to support the requirements since it does not negatively impact the functioning of other features.

The second part was trickier and involved more design considerations. Only one root user is allowed to be created, and once a root user exists, an AuthToken is needed to access restricted methods. This solves the problem of creating the permissions needed to permit users to access restricted methods. My implementation continues to support the requirements by keeping the Store Model/Controller Service interactions the same. All the Authentication Service adds is a "special" user to perform a few setup actions to get the rest of the system set up.

Is the design process getting easier?
Overall, I do think the design process has been getting easier. Most of the written elements in the design have become second-hand, and the diagraming is also getting easier the more I do it. I find the assignments have done a great job building off of the previous ones in terms of design skills. Assignment 2 (store model) dealt a lot with modeling and getting my head about UML and classes. Assignment 3 (store controller) then added a sequence diagram which I found challenging to start but got easier in assignment 4. It also added patterns from *Head First Design* which was somewhat challenging to start, but has also gotten easier to model in this assignment. Some parts of the design are still challenging, but at this point in the course it is more a conceptual challenge (e.g. what is the best design) rather than a practical one (e.g. *how* do I create the class diagram).

Did the design review help improve your design?.
As mentioned below in the comments section, my peer review partners did not get back to me until late in the assignment. I contacted Eric Gieseke about this situation and he replied with some helpful comments. His overall impression was the design was well thought out, but his specific recommendations (like the note about the Visitor I mentioned in "How could the design have been better…") and structuring the class diagram helped with fine-tuning the design and where I should direct my focus.

Your comments for your review partners.

I originally had two members in Group 17 for peer review, Ramnath Pillai and James McCrary. James never replied to any contact message and no longer appears in the group on Canvas. I contacted Eric Gieseke for input and ended up receiving his feedback, as well as a response from Ramnath. Ramnath provided several diagrams for his design which I provided comments on in the group discussion post. Comments are below.

My comments on Ramnath's design:

> Hi Ramnath,
>
> I took a look at your class diagram and sequence diagrams and overall think you have a really good design. I did see a few areas I think you should keep in mind as you proceed with your implementation.
>
> I don't see a Resource class. The requirements mention them separately and I've viewed them as being reusable (e.g. many roles can reference the same Resource). Likewise, I've found it useful in my implementation to have an association between the AuthService and a list of Resources to maintain uniqueness.
>
> Something I've found during implementation is it has been useful to keep separate lists of Roles and Permissions associated with the AuthService. My original idea was similar to yours in keeping just a single "entitlements" list associated in the AuthService, but I would suggest maybe breaking that up into separate Roles/Permissions lists.
>
> Another thing I've come across is, like you I started with creating a separate Credential class to track face, voice, and password. My class ended up being just a single string with a getter() method so I thought it was simpler in my implementation to redesign it as just three properties in the User. However you end up implementing the design, you should clarify a User can have more than one Credential. My design allows for only one of each kind (e.g. one voice, one face, one password), and if another is provided the old one is removed. Yours may differ, but I think it'd be useful to mention.
>
> I also noticed you have the AuthenticationToken associated only with the CheckAccessVisitor and not the User. Without reading more, I'm unclear how this relationship would function and meet requirements. Be sure to clarify in your final design/implementation.
>
> A final note on the class diagram: I don't see any of the Exception classes included, or any properties/methods on any of the classes. I think it's useful (/ required ??) for the diagram, but make sure to include a dictionary as well.

Regarding your sequence diagrams, they look very good as far as I can tell. The only bit I noticed is something I've struggled with and haven't fully figure out myself yet: the Visitor returning information. In your checkAccess diagram, you have the visitor returning true/false? And the inventory diagram you have the Visitor return void and then a call to getCumulatedInfo(). I do have something similar, but what I haven't figured out yet is how to separate these functionalities while keeping a "sparse" interface that doesn't reference specifics.

Overall I think you did a great job Ramnath. Best of luck on the implementation!

Best,
Matthew

Comments from peer design review and optionally the functional review.
See my note above about the peer review. I received feedback from Eric Gieseke. Ramnath never returned feedback on the design. If he responds after submitting my assignment, I will post to the comments portion of the submission on Canvas.

Comments from Eric Gieseke:

Hi Matthew,

Please see my comments below.

Class diagram:
Exception classes should be referenced by the StoreAuthenticationService Interface.
Make better use of space, so that it is easier to read.

In the class dictionary, you should mention what classes a class extends or interfaces that it implements.

The AccessCheck Visitor should locate the matching auth token, then navigate to the associated user, and then check the associated entitlements to search for the matching permission/resource.

This looks like a very well thought out and documented design document that addresses the given requirements. Nice work!

Eric