AIT 642 / COSC 603

Software Maintenance & Testing

Spring 2016

**Project #2 – Refactoring and Design Smells**

***Task 6* –** I think that’s because its just a parameter name and that’s why its not changed? The difference between Eclipse refactor/rename and simple find/replace is that Eclipse understands the code semantically and is able to identify references to a specific method, variable, or class names which ensures that method, variable, and class names indicate their intent clearly. In the omission case the find/replace would change that too.

***Task 7* -**I guess the refactoring/pushing down for this task was itself a design smell because we caused code duplications so if something like that already existed in our code then pulling up could be a good solution to fix that smell design and make the code more maintainable. And pull down can be used when a method is not universally used anymore and is used by only one (or a few) subclass.

This refactoring process was very fast and easy specially to redo the change. And eclipse is really smart and could suggest other changes required.

***Task 8 -*** I like eclipse because it knows what should be done to fix things. Extracting an interface could help to prevent insufficient modularization and also growing a class into a large one. I read it helps with generalization too but couldn’t figure it out how and what it meant?

Methods I extracted into the *IOwnable* interface were getName(), setTheOwner(), getTheOwner() and IOwnable.java is created but it has a question mark on it and it hasn’t been added to GitHub repository with my commit?

***Task 9 -*** In this refactoring experience I learned how to simplify a method by extracting a method from that. I used the second signature because it made my getRent() method simpler and easier to understand.

***Task 10 -*** It was easy to create the local variable but I couldn’t figure it out how this was helpful? Just to prevent duplicated code? Because duplicated code makes the code long and hard to work with. However, introducing variables in this way provides several benefits. First, by providing meaningful names to the expressions, it helps to understand easier what the code is doing. Second, it makes it easier to debug the code, because we can easily inspect the values that the expressions return. Finally, in cases where multiple instances of an expression can be replaced with a single variable, this can be more efficient. Probably it can’t be always OK to create a local variable because it could affect the correctness of a program.

***Task 11 -*** It was easy to change the method signature and after refactoring I needed to change several methods in different classes which were using the method from abstract class Cell. And most of them were needed me to add a return statement as suggested by eclipse. I’m not sure but this refactoring should help with the smell “too many parameters” probably because it deals with parameters mostly. And it’s hard and confusing to work with the code that has too many parameters.

***Task 13 -*** Well the first one I ran was Feature Envy which found five bad smells and the refactoring suggestions were moving some methods and I just applied whatever it was suggesting and all worked except the one method: sendToJail() that caused errors in GameMaster class and I didn’t know how to fix It so I undo it and it was OK afterwards.

The second Type Checking listed three things: two inheritance hierarchy and one constant variable [TYPE\_CC] but didn’t suggest any refactoring so I didn’t do anything.

The Long Method didn’t list anything.

The God Class listed seventeen things and no refactoring suggestions. So at first I did nothing but after I commit my project I found out that if I clicked on the small triangle it shows the methods that needed to change. I applied the suggested refactoring for couple of them and noticed they were mostly test classes so I didn’t know if they needed refactoring or not? So I didn’t commit that version again.

***Task 14 -*** I used the JDeodorant for my first project but it didn’t find much design smell except couple of them for type checking I guess and it suggested me to refactor my variables and I did.

***Task 15 – Summing it All Up.***

This was an interesting project and I found it helpful to understand what we learned in class about refactoring as an important part of software maintenance. I think now I have a better understanding of what it is about and how it can help us to improve the software maintainability by detecting and removing design smells.

Another interesting thing was learning and working with tools which are smart and powerful and can make the refactoring tasks very easy and fast.

Well, I liked them all. Eclipse is smart and can do a lot! I enjoyed working with it and JDeodorant too. Eclipse’s support for refactoring is really powerful and logical. It helped me to understand how different types of refactoring techniques work. I found it to be a safe tool to use for refactoring tasks specially by showing us the consequences of each change we want to make through the its Preview feature.

Eclipse’s automated features makes the refactoring operation very easy and fast and of course makes less mistakes with regard to change the references, associated methods, etc. But still in general it might make mistakes and we need to understand the change we decide to make well before applying it. I’m glad we use a version control machine because I found it difficult to undo a change when we make several changes.

JDeodorant is useful and smart too but I’m not sure if all its findings are correct? And also to what extend its refactoring suggestions are reliable?

I found unit testing a very important part of this project specially for me that I had no idea what I was doing during the first couple of tasks. So it was the only thing that could help me make sure things were working and I didn’t break anything before moving to next task.