Jess Rudolph

Project 2

**Task 6**

For this task, briefly describe if this omission is an oversight on the part of Eclipse’s refactoring operation and why or why not. Additionally, briefly describe how (or if) this operation is any different than a simple find all and replace.

This omission wherein the parameter is not renamed after the refactoring, is an oversight on the part of Eclipse’s refactoring operation. I believe this because the parameter was supposed to be part of the renaming refactor operation – the parameter is part of what the operation is supposed to rename. This is different from a find all and replace because it allows you to select exactly what you want to rename instead of renaming everything.

**Task 7**

For this task, briefly describe your experience with this task and for which design smells pushing down or pulling up a class’s field(s) and/or associated methods could help make the code more maintainable and why.

Push up and pull down can be used to refactor code. For example, if there are attributes in the superclass that need to be moved to a subclass, this can be done easily. This would make the code more maintainable because these adjustments could be made.

**Task 8**

For this task, briefly describe your experience with this task and for which design smells extracting an interface could help make the code more maintainable and why. In your description, be sure to include a description of which methods you extracted into the interface and what new files were created in this operation.

I extracted the getTheOwner() and setTheOwner() methods when extracting the interface. I chose those methods because the owner information is necessary for the players to own anything.

**Task 9**

For this task, briefly describe your experience with this task including the method signature you extracted and why you chose this one.

I extracted the private int calculateMonopoliesRent(int rentToCharge) method signature because it is needed to calculate the rent.

**Task 10**

For this task, briefly describe your experience with this task and for which design smells creating a local variable from repeated code could help make the code more maintainable and why. In addition, comment on whether it is always OK to do this to a function call and whether it could affect the correctness of a program.

I found creating a local variable from repeated code to be very helpful. This makes for more maintainable code by aiding to eliminate the design smell of duplicated code. It may not always be okay to utilize this function because sometimes a human may need to manually make these corrections for better accuracy.

**Task 11**

For this task, briefly describe your experience with this task and for which design smells changing a method’s signature could help make the code more maintainable and why. In addition, comment on why things are changing in other class than just Cell.java and how this affected the definitions of any other classes besides Cell.java.

Changing a method’s signature changes the parameters, visibility, and return type of a method. In this instance, the change effected other classes because the method signature change was made to the superclass and therefore, all of the subclasses that call that class were also affected. The code would need to be manually refactored such that those methods have a return type of Boolean.

**Task 13**

I was not able to find any refactorings while utilizing the JDeodorant plugin for the Monopoly program.

**Task 15**

I learned a lot from this project about refactoring and the tools that can be utilized to implement refactoring code. This assignment relates to the lessons in the lecture about design smells. Specifically, it relates to detecting and resolving problems like duplicated code, methods that are too long, proper documentation (javadocs), renaming resources, and class hierarchy modifications.

I’m really happy that Eclipse has the JDeodorant and other resources available for refactoring so that I may utilize them as needed. However, there are limitations and times when the code may be too complex to utilize these tools and rectoring changes need to be done manually. The greatest limitation is that when changing the method signature there is likely portion of the code that will need to be manually refactored. I wouldn’t call this too much of a limitation, though, because you can easily see where those changes caused errors in your code to identify them and fix them with ease. Overall, I’m very impressed with these features.

Unit tests are very important when refactoring. They should be conducted before and after you refactor the code so you know immediately what changes (refactoring) caused new errors to arise (if any). It definitely aids in locating those errors without having to trace through all of the recent changes you’ve made and guessing which one(s) caused the error.