Matt Duffy

COSC 603

Project 2

**Task 6: 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 is not an omission in Eclipse’s refactoring operation. The variable we are refactoring is a class level variable the variable that does not get refactored is a method level variable. They are two different objects in the program and changing one should not dictate changing the other.

**Task 7: 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.**

Pushing a method or variable up into a superclass or down into the inheriting classes can help clean up reused code in either class. If multiple classes that inherit from the same superclass use the same method or variable it would be good practice to push this up into the superclass for the other classes to inherit.

**Task 8: 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.**

Creating an interface can give classes that implement the interface a number of common components that they can utilize. This allows for one change in the interface class to affect all of the classes implementing it.

**Task 9: Briefly describe your experience with this task including the method signature you extracted and why you chose this one.**

When extracting the for loop into a separate method I chose to include the string array in the method instead of having it passed through the method signature. I chose this because the string array is only used during the loop. Creating this each time the method is called and passing it into the method would be unnecessary, unless the variable needs to be used outside of the method then it should be created in the method.

**Task 10: 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.**

Extracting this into a new variable could help save time. Each time you do this call it has to go into the cell class and call the getColorGroup method. If this method take time to run then multim=le calls can slow this down. However if the method being called isn’t costly to run then you are creating another object that needs to be stored in memory.

**Task 11: 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 methods signature from void to Boolean does not properly refactor. To get this to work return calls were needed to get the code to run properly. This can help make the code more reliable because it can give a better idea of whether a method was run correctly or not. The Boolean allows for an extra check.

**Task13: Briefly describe the refactorings you made (if any) as a result of using JDeodorant.**

For this task I used the suggestions to create new classes for methods that were being called from multiple locations. This will create less spaghetti code and give the programmer a single location to change when a method needs to be rewritten.

**Task 14: Briefly describe the refactorings you made (if any) as a result of using JDeodorant.**

Most of the suggestions that were given from JDeodorant involved extracting new methods. I did this for the creating of arrays that are using throughout the program and for a few other areas that it recommended be taken out of the method.

**A description (2-3 paragraphs) of what you learned from this project and how it relates to some of the topics covered in lecture**

We discussed in class the ripple effect and how it can affect code. This project showed that when refactoring a small part of the code the changes can be felt in other areas. When using automated refactoring like in this exercise the IDE tries to limit the effect on the rest of the code. It will show changes and potential errors that will come about form the refactoring. I also learned that while the IDE does a good job of automating this the ripple effect can be felt in areas that the program cannot pick up. This was shown in the change of the method signature from returning void to returning a Boolean. Eclipse was able to change the appropriate signatures but was not able to add the return statement into the code, this had to be done manually to get the code to work properly again.

The project also touched on some of the different types of maintenance. Using refactoring of the code as a preventative maintenance step to allow for future maintainability. Using the JDeodorant tool and refactoring the project will give it better readability through less interconnecting calls through the classes and using less repeated code.

Running the unit tests gave us a good look at corrective maintenance. While none of the unit test came back with an error and needed to be changed, running the test scripts showed that that the program was still producing the proper output as we refactored different parts of the program.

**A description (2-3 paragraphs) of what you liked about Eclipse’s support for refactoring including its strengths and limitations as well as your impression of JDeodorant**

I liked the way Eclipse does refactoring. Being able to highlight and automatically refactor is a very helpful tool when trying to properly move code form one area to another. I’ve never used automated refactoring before, other than name changes, and it’s nice to see any conflicts that can arise from doing a refactoring. It will definitely make refactoring for me a little easier in the future knowing that these tools are available.

JDeodorant is a nice little tool to offer some help on what may need to be refactored. I’m not sure if I agree with all the recommendations that they give or the manner that they give them. But it does give a little inside on some week areas of code that can be either removed or moved to different location. This is especially true when you are doing too much in a single method, JDeoderant does seem to find these types of code mistakes very easily and highlights them so they are easily refactored with Eclipse.

**A description (1-2 paragraphs) of how/why unit tests are important when doing refactoring**

Unit tests help to check that the refactoring has not affected the desired outcome of the code. When refactoring, you could potentially change too many parts of the code to be able to check on your own. Allowing the unit test to determine whether the code will still produce the desired outcome will help mitigate this. You can however rely too much on unit test. The test may not be able to determine whether the refactoring has affected the code if the outcome lies outside of its scope.