Assignment #7

Practice with Refactoring in Eclipse (or Netbeans)

CS 4354 Summer II 2014
Instructor: Jill Seaman

Due: in class **Tuesday, 8/5/2014** (upload electronic copy by 12:00 noon).

---

1. **SETUP**: We'll use a Monopoly game written in Java (complete with JUnit testcases) developed at North Carolina State University. I have modified the code slightly for the purposes of this assignment.

   Get a copy of this project's source files (Monopoly4.zip) from the class website. Create a new Java project, then import these files as follows:
   - Create a new Java project.
   - Make sure that project has a folder named src in it. If not, select the project name, then right-click and choose New -> Source Folder. In the pop-up window, name it src.
   - Right-click on the folder src and choose Import, General, **Archive File**
   - Browse and choose the Monopoly4.zip file, and then hit Finish.

   You should see the java files in two packages under src (once you open src).

   Set up and run JUnit tests:
   - Right-click on the Project, choose Build Path > Add Library > JUnit. Then select either JUnit 4 or JUnit 3 (the code is written in JUnit 3 style).
   - Right-click on the Project, choose Build Project.
   - Right-click on the Project, choose Run As > JUnit Test. Make sure all the tests run and pass (44/44).

   Let's play Monopoly! Try clicking on the GUI package, then right-clicking and choosing Run Java Application….

   Now let's Refactor:

2. **Rename class field:**
   - Locate and open class Cell (which represents a square on the game board) in the monopoly package.
   - [Eclipse] Highlight the name of the class and select Navigate > Open Type Hierarchy to see its subclasses in the Type Hierarchy pane. You can also see that Cell has a field named **player** of type Player. This name is not very descriptive of the role of the Player with respect to the Cell.

- This Player is really the owner of that Cell, so use the Refactor > Rename option to change the name to owner [Eclipse] (click on the triangle to open the Rename Dialog). Select to update references, comments and rename the setters and getters. Use the Preview option to see what will change before you select ok to finish.
- Save, Select [Eclipse] Project > Build Project to rebuild, then Run the JUnit tests again to make sure nothing is broken.

3. Extract Local variable:

- Go to GameBoard.addCell(PropertyCell). See that the expression `cell.getColorGroup()` is used twice? Highlight one of those usages and then use [Eclipse] Extract Local Variable from the refactoring menu. Note that Eclipse suggests names for the local variable.
- Explore what options are offered, and carry out the refactoring and make sure you understand what has changed.
- Netbeans: I'm not sure this is available, it might be Introduce Variable, but in any case you can do it "by hand".
- Save, build project, run JUnit tests.
- Is it always OK to do this to a function call like this? Could it affect the correctness of the program?

4. Extract Method:

- Go to GameMaster.btnGetOutOfJailClicked(). Select all of the statements inside the if block, and use Extract Method [Netbeans: Introduce Method?]. Call the new (private) method `setAllButtonsDisabled`. Make sure it replaces the additional occurrence of the statements (in the previous method) with a call to the new method as well (you are replacing duplicate code with a method).
- Go to PropertyCell.getRent(). Notice the for loop. Extract a method containing the for loop and optionally the declaration of the array named `monopolies` that occurs before it. Choose the one that gives you the FEWEST parameters.
- Save, build project, run JUnit tests.

5. **Extract Subclass:**

If you investigate the Cell hierarchy, you may notice that the fields `owner` and `available` (and their getters and setters) are used only in three subclasses: `PropertyCell`, `RailRoadCell` and `UtilityCell` (but there are a total of 8 subclasses). We will use a Refactoring called "Extract Subclass" to make a special subclass of Cell that will be the superclass of these three classes. This must be done in steps:

(1) Click on one of the three subclasses that uses the owner and available and use the Refactor > **Extract Superclass** option. Call it `OwnedCell`. Be sure to indicate all three subclasses. This will create the new class and add it to the Cell type hierarchy. (Save, Build Project to check for compiler errors).

(2) Use Push Down Field to push the **`available`** field (from the Cell class) and its getters and setters to the OwnedCell class. In Eclipse, after you select Preview, un-select the classes you do not want to receive the pushed down field (hint:

`available` is used only in the OwnedCell subclasses). If you don't or can't do this, after you do push down field, you'll have to edit each of the non-OwnedCell classes to remove the field and getters/setters.

(3) Now when you build the project, you will have some compiler errors. Fix them:
GoCell: delete the statement.
GameMaster: in the else, check if cell is an instanceof OwnedCell, then cast cell to OwnedCell inside the else block (using a temp var or lots of parentheses).
Player: put the entire method body inside an if-stmt that checks if getPosition is an instance of OwnedCell. Cast it to OwnedCell before checking isAvailable.

(4) Save changes, build project and run JUnit tests (fix any mistakes).

(5) Redo steps (2)-(4) to Push Down the **`owner`** field.

Pay attention to the code in the Player class (as you fix the errors). Note where it checks instances for one of the three OwnedCell subclasses in the buyProperty and sellProperty methods. We could probably also use refactoring to change the parameter of these methods to OwnedCell, and then use polymorphism to move the subclass-specific code to the subclasses. (But that exercise is left for another time . . .).

NOTES:

- This assignment is to be done during class on Tuesday 8/5 in groups of 2-4 people.
- You should be able to use Eclipse or Netbeans
- There are no style guidelines.
- Make sure your code compiles and that the JUnit tests still pass before submitting it.
- I will make a list of who is working in each group during the lab exercise.

Submit:

Please combine the (modified) *.java files from the project into a single zip file (assign7.zip). Submit an **electronic copy**, using the Assignments tool on the TRACS website for this class, before the end of class.