CS6312
Lab 02

The World of Zuul

Objectives:

- Modify / update an existing application for better structure and functionality

Overview:

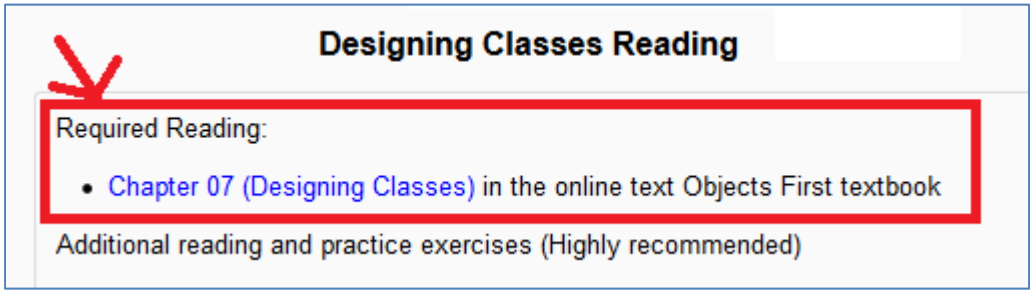
One thing that we rarely get a chance to do in the school environment (which is prominent in the 'real-world') is to take existing code and modify/update it for new functionality. This is the area where appropriate code structure makes life so much easier.

In an effort to give you some experience with this, I've pulled an exercise out of an introductory Java text that I've taught from in the past (don't worry, it's their 'free chapter', so we're not breaking any copyright laws). These authors took the time to develop an application that's REALLY poorly structured but from the user's standpoint, it works just fine. The point is that it's difficult to see the structure issues until you begin to do some seemingly small modifications... then everything breaks down.

One thing that's critical for this exercise is that you understand the background of the application and the code inside the files. The textbook reading (included in this week's Reading section in Moodle) and the first few parts of this exercise will give you a little bit of experience to get started.

As always, be sure to use appropriate data types, variable names, and comments in your code. Don't forget to 'Commit' and comment along the way – at a minimum, you should have a commit (with an appropriate commit message) after most of these steps

1. Get a copy of the document located in this week's reading section on Moodle



2. Read sections 7.1 through 7.6 before downloading or looking at any code

3. Download the starter file from Moodle, extract the files, and open the Java Project in Eclipse using the procedures outlined in the previous exercises.

4. We have included a repository inside this project. Double-check that it is present before continuing (right-click the Project name and choose Team > Show History). You should see exactly one commit from me. Be sure it's there before moving ahead.

5. As it sits now, this code will show Checkstyle issues. Many of them are problems with missing / incomplete Javadoc comments. Go ahead and fix those now. The remaining issues will be fixed as you complete the exercise.

6. Add code to the `ZuulDriver` class that will:
    a. Create an instance of the `Game` class
    b. Use that object to call `play( )`

7. (Based on Ex 7.1)
   Play the game to explore the world created in this game. Use some type of image drawing program (MS Paint, Excel, Word, etc) to draw a digital map of this world. Save your file in the *src* folder of your Java Project. (If you don't save it here, we won't be able to see it for grading!)

8. (Based on Ex 7.5)
   Section 7.4 of the text describes some issues with code duplication present in this program. Implement and use a separate method named `printLocationInfo` in your project as described in this section.

NOTE: Section 7.6 gives an overview of changes, including the use of a `HashMap`. We will *not* be implementing a `HashMap` in this exercise. Please read the information here, but do not implement these changes in your application.

9. (Based on Ex 7.6)
   Section 7.6 of the text describes some issues with coupling present in this program. Implement the changes to the `Room` class by making its instance variables `private`. Then implement and make use of the `getExit` method described. (Hint: you'll need to complete the next step in order to get the code to compile for testing).

   Be aware that the code given in this part of the text does not follow our style guidelines, so please fix it accordingly. That is, because only one exit will be returned for any given direction, we can use an if / else if / else if / else structure, instead of the series of 4 independent if statements shown in the text. Note that this makes the code (1) easier to read and (2) more efficient.

10. (Based on Ex 7.7)
    Make a similar change to the `printLocationInfo` method of `Game` so that the details of the exits are now prepared by the `Room` rather than the `Game`. Define a method in `Room` with the following signature:

```
/**
 * Return a description of the room's exits,
 *    for example, "Exits: north west".
 * @return  A description of the available exits.
 */
public String getExitString()
```

11. Read Sections 7.7 – 7.12

12. (Based on Ex 7.11)
    Implement and use the `getLongDescription( )` method described at the end of Section 7.7.

13. (Based on Ex 7.14)
    Add the functionality for the user to "look" as described in Section 7.9.

14. (Based on Ex 7.16)
    Add the code described in Section 7.9 for the "improved version" of printing out the command words.

15. (Based on Ex 7.18)
    Read Section 7.10 and implement the suggested change to getting the list of command words. NOTE: This one requires that you undo part of the previous exercise, which we don't want you to do. Instead:
    a. Keep the `showAll` method and simply add the method `getCommandList`. Normally it make more sense to replace `showAll` as described, but we want to see your work, so keep it in there.
    b. Keep the `showCommands` method and simply add a method `getCommands`.
    c. Do NOT replace the old call to `showCommands`, instead just comment it out and add the code for this exercise below it

Submitting:

When you are finished, make one final commit to your repository, be sure that all of the class files are saved, close Eclipse, and Zip the folder holding your Eclipse project using 7-Zip. Give your file the name 6312*YourLastName*Lab02.zip. Upload the Zip file to the appropriate link in Moodle.

It's probably worth taking a moment to download the file you uploaded to Moodle and double-checking to be sure that it contains everything you are required to have (including the repository).