CSCI 125 Project 3: Zuul

Part 1

For this project you will work on your own to extend your zuul game by completing the following exercises in the textbook. You can start with the *zuul-better* project.

- Ex. 8.22: Rooms can hold any number of items.
- Ex. 8.23: Implement a *back* command. What should happen if you type "back" twice in a row? What should happen if you type "back" at the beginning of the game before the player has moved?
- Ex. 8.28: Extend your implementation to add a *Player* class. (See section 8.12)
- Ex. 8.29: Implement *take* and *drop* commands.
- Ex. 8.30: Players can carry multiple items.
- Ex. 8.31: Add a restriction that allows the player to carry items only up to a specified maximum weight.
- Ex. 8.32: Implement an *items* command that prints out all items currently carried and their total weight.
- Ex. 8.33: Add a "magic cookie" (or item of your choice) to a room and a *eat* command that increases the weight that the player can carry. Attempting to eat something other than the "magic cookie" should result in an appropriate response.

You must complete the exercises to this point to get a C level grade on the project.

For a better quality game (and easy points):

- If nothing in room or inventory, don't just print empty list.
- Report failure if Player/Room doesn't have Item dropped/taken.

This portion of the project is due today: the remainder of the project is due next Tuesday

When you have completed the Zuul Project, select "Create Jar File ..." from the File menu in BlueJ. Check "include source" and "include Bluej project files". Save the file with the name bassett-zuulproject.jar (except use your name instead of bassett). Then upload the "jar" file into Moodle before class.

CSCI 125 Project 3: Zuul

PART 2:

- You may fix and resubmit any items from Part 1 for re-grading. Include these fixes with your submission for Part 2.
- Ex. 8.44: Add locked exits to your game. The player needs to find (or otherwise obtain) a key to unlock the exit and enter a room. (You don't need to add a new class for doors or exits to your game). Make this as general as possible - in any room, any exit could potentially be locked. There are several ways to design this: some are better than others.
- Ex. 8.47: Add characters to your game which can talk and will give you some help if you give them the right item. Don't name this class *Character* because Java already has a class named *Character*.
- **You must complete the exercises to this point to get a B level grade on the project**
- Note that the Game class should do all the printing; other classes can return Strings for the Game class to print.
- Avoid any Null Pointer Exceptions (or any other type of exception)!
- In the interests of modularization and encapsulation, your classes should not reveal the data structures they use to store information. For example, if your Room uses a HashMap to store items, don't create a *getItems* method that returns a HashMap. Instead, ask yourself "what responsibilities should my Room class be able to handle?" For example, you will probably want methods like *containsItem*, *getItem*, *putItem*.
- **For an A level project:** Write your code so that the details of your specific game are isolated to the *createGame()* method (renamed from *createRooms()*). All of the other methods in your project should work for any game that involves Rooms, Players, Items and Characters. The only exception is if you have to add special commands that are game-specific. If you have any questions, be sure to ask me.

YOU MUST ALSO HAND IN:

1. Add **comments** to your code that clearly indicate where the changes were made for each exercise. Include Javadoc for all public classes, public constructors, and public methods.
2. In the README.TXT file in the upper left-hand corner of the BlueJ class diagram, write a **paragraph** or two describing the changes you made to your project and the reasons you chose to do the set of exercises that way. Use terms from the Green boxes scattered throughout the chapter to explain your design considerations.
3. Write a **test script** consisting of a list of commands that can be used to test your changes by playing a game: if using zuul-with-tester, this should use the supported format and be in the default script.txt; otherwise write your script in README.TXT. Describe how the script tests the changes.

When you have completed the Zuul Project, select "Create Jar File ..." from the File menu in BlueJ. Check "include source" and "include Bluej project files". Save the file with the name bassett-zuulproject.jar (except use your name instead of bassett). Then upload the "jar" file into Moodle before class.