CS-162 Lab #3: More Inheritance

General

At the top of the every document that you create (word processing files or source files) include:

```
/**
 * Description of the class or document.
 *
 * @author YOUR NAME
 * @version CS162 Lab #, mm/dd/yyyy (replace with the last edit date)
*/
```

Submit in this lab via Moodle using the "Lab #" link. Your assignment must be uploaded by the assigned due date and time. Moodle will automatically close the link at that time. It is strongly highly recommend you do not wait until the very last minute to submit your work.

Concepts

This lab will extend your experience with designing and programming using Java Inheritance. This lab will focus on the use of inheritance to create specialize behaviors for similar classes of objects. Additionally you will work on your design and planning skills as you will have to come up with your own approaches to refactoring and reengineering the Zuul game to add these new features.

Background

Read chapter #9 and review the online tutorials on inheritance. Study the example in chapter #9 that discusses the implementation of a "transporter room" for the Zuul game that we worked on last term. This is section 9.11. DO NOT START THE LAB UNTIL YOU HAVE READ AND STUDIED THIS SECTION OF THE TEXT. If you have your version of Zuul better from last term that has your up & down commands, then use your modified code for this lab. Otherwise, just get a fresh copy of "ZuulBetter" from Chapter #6 of the code from the text.

Assignment

Implement a transporter room with inheritance in your version of the zuul project (see Exercise 9.8 in the text and read section 9.11 before starting this lab).

Additional Requirements:

1. Start with the modified code from your zuul project from last term (the zuul_better version) with the upstairs/downstairs modifications. If you were not in the class last term, then start with the "zuul_better" project from chapter #6.
2. Add JavaDoc comments for all NEW METHODS and CLASSES that you write. Build JUnit Test classes for one or two NEW CLASSES and create at least one test method for each test class to convince yourself that everything is working properly.
3. Write a MAIN method that will set up and run your Game; you only need one class with a MAIN method; decide which class is the best to add this into and have the main create the appropriate object instances in order to execute the game.
4. Create at least 2 additional new rooms that are instances of transporter rooms.

Helpful Notes:

You are to make as few changes to your existing code as possible (if fact, it is possible to do this with only creating one or two new classes and using inheritance as described in Chapter #9). The only major changes to your existing code (other than creating the new classes) that you should do is to add 2 or 3 "transporter" rooms to the game in the Game class and a couple of other very minor modifications (maybe a new field, and new parameters to the room constructor).

In your design, be sure that you remember that the real purpose of the lab is for you to get further experience with Java's inheritance. So view it as a lab requirement that you MUST at least create a new class that inherits from the Room class (the TransporterRoom class).

The REAL challenge is to write the "findRandomRoom()" method. As the text points out, this is probably best done by creating an additional class that will have an "ArrayList" or "array" of the rooms; when each room is created the constructor for the Room class should add itself (a room) to the list of rooms.

One approach to this would be to create an instance of a container class as a ***class variable*** in the Room class (a "static" variable). This will allow all instances of Room/TransporterRoom to have a single, shared ArrayList that contains references to all of the rooms to make your random selection from. In the constructor of the Room class, each new instance is added to the ArrayList class field. Then create the new Transporter class that inherits from Room and write a FindRandomRoom method and override the getExit() method.

An alternate approach (and a somewhat better design) would be to create a new class to represent the "virtual world or map" of the game. This class would have a container field that would hold references to all of the rooms, along with at least an add() method and the findRandomRoom method. Room class instances would have to be made aware of this new map class and would again add themselves to the map at the end of the room constructor method. You will of course also need the Transporter room class that overrides the getExit() method.

Get yourself re-familiarize with the zuul code as we will be doing several labs with it this term.

Submission:

Submit your entire BlueJ project (zip only, no tar, 7zip, or other file compression types) folder for this lab into the CS162, Lab2 upload link (just as was done in CS161).