CO520 Further Object-Oriented Programming
Assessment 2: Design & Inheritance

Olaf Chitil

Submission by Moodle on Tuesday 18th February 2020 at 23:55, Week 18

Introduction

You work with the given project `hauntedCastleStart`. This is yet another variant of the game of Zuul that is described in Chapter 8 of the module text book. This variant uses a different scenario, namely a haunted castle. You will extend the project towards a more interesting game.

Note the difference in class structure. There is a split between the classes `Game` and `GameMain`. The latter is the textual user interface of the game, the former is the actual game independent of any particular user interface (we could use it for example with a graphical user interface where commands are selected from menus). Furthermore, commands are structured rather differently from the book, using lambdas. All knowledge about the command language is in the `Parser`. There are several classes and methods that currently do not serve any purpose. They are there for you to extend as described below. Start by familiarising yourself with the source files of the project.

The project contains assertions for some contracts and test classes. These are part of the specification of what you have to implement. Many of the tests fail in the given project. Through your work more and more of these tests should pass; if you do all tasks, then nearly all tests should pass (Adding dual ghosts can make a few tests fail that passed beforehand!). You should never remove or change these contracts or test classes. You can add to them, but this is not required.

The assessment will be marked out of 100 and provisional marks have been associated with the individual components. Most of the marks will be obtained by passing tests of a test suite that will be added to the project after your submission. This test suite will include some of the tests that are already in the project, but there will be many more tests.

Make sure that you do not only write code, but that you also write or update appropriate comments for all classes and methods. Furthermore, you have to keep to the program style guide of the book. After marking out of 100 as indicated below, between 0% and 20% may be subtracted if you violate the style guide. In extreme cases even more than 20% can be subtracted.

Note that there are dependencies between the tasks given below, but it is possible to do some of them in a different order.

Note that you have to submit the assessment before the deadline; there are no “grace days”.

Adding a Goal (10 marks)

The aim of the game shall be to reach the bedroom. After the player has entered the bedroom a congratulatory message shall be printed and the game ends.

All your changes will be in the classes `Game` and `Player`. You will need a new field in the class `Player`. Make changes in the `goRoom` method.

Adding Time (10 marks)

Playing the game shall have a time limit. Do not use real time but instead count the number of movements of the player. The initial time limit shall be 12. If the goal has not been reached within the time limit, then the player loses. Losing means that a corresponding message is printed and the game ends.

Note that if in the last time step the player reaches the goal, then the player should win, not lose because time runs out.

Make only changes in the classes `Game` and `Player` and keep them minimal.

Adding a `look` Command (10 marks)

Add the command `look` to the commands that the game understands. The purpose of `look` is to print out the description of the room and the exits again (we "look around the room"). This could be helpful if we have entered a sequence of commands in a room so that the description has scrolled out of view and we cannot remember where the exits of the current room are.

You will have to add to the class `Game` and modify the class `Parser`.

Adding Ghosts (10 marks)

A ghost has a description.

Complete the implementation of class `Ghost`. Its `toString()` method shall return the description.

Adding Characters to Rooms (20 marks)

We extend the game such that a room knows about the characters inside it.

Extend the class `Room` with a field such that a room can contain zero or many characters. Add a method `addCharacter` that adds a character to a room and a method `removeCharacter` that removes a character from a room.

Modify the constructor and the method `move` in class `Character`, so that they ensure that the presence or absence of a character is correctly recorded in a room.

Extend the `getLongDescription` method to include a list of all the characters in the room. For this purpose use the `toString()` method of the character. Note that there will be at least one character in a room of the castle, namely the player themselves.

Finally, add two different objects of class `Ghost` to some rooms in the scenario by changing class `Game`.

Making Ghosts Move Randomly (15 marks)

Every time the player performs a "go" command, every ghost shall move in the castle to a random room. Because ghosts can move through walls, a ghost can go in one step to any room; it does not have to follow exits.

Implement the method `Ghost.goRandom` and ensure that it is called from `Game`. To help with moving all ghosts randomly, add two collections to class `Game`: one shall contain all rooms and the other one all ghosts.

Challenge Part

Adding a Solid Ghost (10 marks)

A solid ghost is like a ghost, but it cannot go through walls. So when going to another random room it can only follow an exit to the next room.

Complete the implementation of class `SolidGhost`.

Add one solid ghost to the scenario in class `Game`.

Adding a Dual Ghost (15 marks)

The dual of a direction is the opposite direction. For example, the dual of north is south and the dual of up is down. The dual of a dual is again the original direction.

A dual ghost changes a room in that it changes all exits to their duals. For example, usually the west exit of the great hall leads to the hall and the south exit of the great hall leads to the chapel. When a dual ghost is in the great hall, suddenly the two exits of the great hall are east and north; the east exit leads to the hall and the north exit leads to the chapel. When a dual ghost leaves a room, all exits return to normal. When there are two dual ghosts in a room, the room has exits as normal; so two dual ghosts cancel out each other's effect.

Implement a `dual` method in enum `Direction`. Implement a `dual` method in class `Room`, that changes the room's exits as described above. Complete the implementation of class `DualGhost`, so that its movement between rooms has the described effect on rooms' exists.

Add one dual ghost to the scenario in class `Game`.

Submission

Submit your whole BlueJ project as a single `.jar` or `.zip` file on the Moodle webpage. Make sure that the source files are included.

If you use a development environment different from BlueJ, such as Netbeans or Eclipse, then you have to convert it into BlueJ format. **Do not put your project classes into any Java packages, because that will cause all tests added for marking to fail.**

General Advice

Comments

Comments are fundamental to understanding programs, whether the programs are written by other people (especially in a development team) or we ourselves have forgotten details after some time.

Every class should have a meaningful class comment. If you modify a given class, you should add yourself as an author and update the version. The given projects use dates for versions.

Also every method should have some basic comments so that anyone using a class can understand what a method does without having to read its implementation.

Submission Format

The assessment sheet clearly specifies that your project has to be submitted as a `.jar` or `.zip` file. In the past some students used the `.rar` or `.7z` file formats. These are different file formats. We may simply disregard such submissions and give 0 marks. BlueJ can produce `.jar` files.

Many file compression programs offer the choice between different target file formats and you should make sure to choose `.zip` (or `.jar`, which is nearly the same).

Program Design

Good design has been a topic of the module and we do expect it in all assessments. The given project already fixes many design decisions, especially on the class level. For example, the project has a clear division between classes for the user interface and classes for the actual game (`Game` and `Room`). The classes for the actual game should not produce any output (e.g. via `System.out.print(...)`, but only return `String`s, which are then handled by the user interface part. You, however, still decide on the design of the implementation of classes. Most importantly, you should aim to avoid code duplication. Also ensuring that each method has a clear responsibility and that code could be changed easily in the future should be your aim.

Plagiarism and duplication of material

The work you submit must be your own. We will run checks on all submitted work in an effort to identify possible plagiarism, and take disciplinary action against anyone found to have committed plagiarism.

Some guidelines on avoiding plagiarism

One of the most common reasons for programming plagiarism is leaving work until the last minute. Avoid this by making sure that you know what you have to do (that is not necessarily the same as how to do it) as soon as an assessment is set. Then decide what you will need to do in order to complete the assignment. This will typically involve doing some background reading and programming practice. If in doubt about what is required, ask a member of the course team.

Another common reason is working too closely with one or more other students on the course. Do not program together with someone else, by which I mean do not work together at a single PC, or side by side, typing in more or less the same code. By all means discuss parts of an assignment, but do not thereby end up submitting the same code.

It is not acceptable to submit code that differs only in the comments and variable names, for instance. It is very easy for us to detect when this has been done and we will check for it.

Never let someone else have a copy of your code, no matter how desperate they are. Always advise someone in this position to seek help from their class supervisor or lecturer. Otherwise they will never properly learn for themselves.

It is not acceptable to post assignments on sites such as RentACoder and we treat such actions as evidence of attempted plagiarism, regardless of whether or not work is payed for.

Further advice on plagiarism and collaboration is also available

You are reminded of the rules about plagiarism that can be found in the Stage I Handbook. These rules apply to programming assignments. We reserve the right to apply checks to programs submitted for assignment in order to guard against plagiarism and to use programs submitted to test and refine our plagiarism detection methods both during the course and in the future.