CO520 Further Object-Oriented Programming
Assessment 1: Designing Classes

Olaf Chitil

Submission by Moodle on Thursday 15th February 2018 at 23:55, Week 17

Introduction

You work with the given project `londonStart`. 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 some parts of London. 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). Note that the classes `Game` plus `Room` are not dependent on any other class. Furthermore, commands are structured rather differently from the book, using lambdas. All knowledge about the command language is in the `Parser`. Start by familiarising yourself with the source files of the project.

The project contains assertions for some contracts and a small test class. You should never remove these (you can add to them, but this is not required). You may have to modify existing tests when adding a new functionality changes an expected result in a test.

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

The assessment will be marked out of 100 and provisional marks have been associated with the individual components.

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 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 Trafalgar Square. After the player has entered Trafalgar Square a congratulatory message shall be printed and the game ends.

All your changes will be in the `Game` class. You will need a new field in the class. To implement the new functionality, 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 "go" commands. 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 class `Game` and keep them minimal.

Adding a `look` Command (15 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 make a small change in class `Game` and modify the class `Parser`.

Adding Items (15 marks)

An item has a description. In the following we assume that the description always is a single word.

To simplify future extensions, write a new *enumeration class* `Item`. This class shall use parameterised values, like the command interface described in Section 8.13.2 of our text book. The class contains the description. Its `toString()` method shall return the description.

Define three different `Item` objects: `SANDWICH`, `CRISPS` and a `DRINK`.

Note that in the following we assume that any item appears only once in the whole game.

Adding Characters (15 marks)

A character has a description. In the following we assume that the description always is a single word. A character holds one or zero items.

To simplify future extensions, write a new *enumeration class* `Character`. This class shall use parameterised values, like the command interface described in Section 8.13.2 of our text book. The class contains the description and an item; the latter may be `null`. Its `toString()` method shall return the description and the description of the item that it holds (if it holds an item).

Define four different `Character` objects: `LAURA`, `SALLY`, `ANDY` and `ALEX`. Give each one item; one character gets no item. You can freely choose the allocation.

Additionally the class shall have a `take(Item item)` method, which takes the indicated item from the character, if the character has that item. Also the method returns `true`, if it is successful in taking the item from the character, and `false` otherwise.

Note that in the following we assume that any character appears only once in the whole game.

Adding Characters to Rooms (13 marks)

We extend the game such that each character is in a room. 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. Extend the `getLongDescription` method to include a list of all the characters in the room (using their `toString()` method). Add a `take(Item item)` method, which takes the indicated item from any character in the room, if some character has that item. Also the method returns `true`, if it is successful in taking the item from a character, and `false` otherwise.

Finally, in the game, add the four characters to any rooms of your choice.

Challenge Part

Adding a `take` Command (15 marks)

The player shall be able to carry any number of items.

Add a method to the class `Game` for the player to `take` an `Item` object from a character in the current room. If the item is not in the current room, then the `take` method should return an appropriate message. If the `take` method is successful, then it removes the item from the room and returns an appropriate message. So overall the returned message either confirms the action or states why it could not be done.

Subsequently add a `take` command to the whole game. By entering the command `take` ***`item`*** the player takes ***`item`*** in the current room. Here ***`item`*** shall be the description of an item.

Adding an `eat` Command (7 marks)

If the player has sandwich, crisps and a drink when the `eat` command is executed, then the game shall end with a congratulatory message. If not all conditions are met, then only an appropriate message is printed and the game continues.

Deadline and Submission

The deadline is 23:55 on Thursday 15th February 2018 at 23:55 (Week 17).

Submit your whole BlueJ[1] project as a single `.jar` or `.zip` file on the Moodle webpage. Make sure that the source files are included.

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` file format. This is a different file format. 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).

[1] If you use a development environment different from BlueJ, such as Netbeans or Eclipse, you have to convert it into BlueJ format.

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.