CO520 Further Object-Oriented Programming
Assessment 3: Functional Abstractions

Stefan Kahrs

Submission by Moodle on Thursday 22$^{\text{th}}$ March (Week 22) at 23:55

Introduction

This assessment is designed to help you explore the features associated with functional abstraction, as well as putting other things together you have seen so far. Generally, you should avoid adding named methods to classes when the same task can be achieved using anonymous methods. However, you should add methods to exploit common code patterns that reduce overall code size.

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

You will take an existing project as a starting point, and make modifications as required, or as you see fit.

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.

Familiarisation

Familiarise yourself with the source files of the project. The project is based on (a solution of) assessment 1, i.e. it is another variant of the ZUUL project.

Characters (30 marks)

Redesign the Character class, along the following cocepts.

Every character should have a `moveProbability` which is a floating point number between 0.0 and 1.0. This should be made a parameter of their constructor. Instead of carrying one or zero items as now, each character has a list of items. Provide a method `getList` giving access to that list. Initially, the character should carry the same items as now, but now in form of a list collection. As a consequence, modify the `take` method accordingly.

Currently characters can only be taken from, they cannot receive items. Add a method `receive(Item e)` that permits a character to receive an item. The method returns a boolean, indicating a successful reception when true.

Add three additional characters to the class, the `CookieMonster`, the `CrispGiver`, and the `Player`. The CookieMonster and CrispGiver characters do not carry items, but they have special powers and behaviours. The CrispGiver and CookieMonster have a moveProbability of 1.0, the Player one of 0.0, everyone else one of 0.5.

When a character enters a room, it should exhibit its default behaviour, specified by the method `enterRoom(Room r)`. Most characters do nothing in particular, except two of our special new characters: the CrispGiver gives to everyone in the room crisps (using their 'receive' method), the CookieMonster takes crisps if they are available.

Also write a method `boolean automove(double d)`. This should return true if and only the the moveProbability of the character is smaller than d.

Room (20 marks)

We want to be able to pick a random exit from a room. This means we want a method `Room randomExit()` that picks any of the exits of the room with equal probability. Should a room have no exits the method should return `null`.

The project already has an `addCharacter` method, but no corresponding `removeCharacter` method. Write one!

Game (30 marks)

The Game is currently not aware where its players reside (not directly anyway). Change that by adding a field (or fields) storing that information. Moreover, the current version of the game carries information about the player which now should be taken from the Player character. Make the corresponding changes to the Game class!

We want to (potentially) move all characters in the game, each time we advance the clock (the field `time`). Write a method `void moveCharacters()` that does that, and call it in an appropriate place. This method should move every character of the game with their moveProbability to a random neighbouring room, and if a move happens subsequently exhibit their default behaviour given by their enterRoom method.

Challenge exercise (20 marks)

We want to give the player the opportunity to choose a random exit from their current position, using a `random` command. Make the corresponding changes to the project as a whole that enables this additional command.

Deadline and Submission

The deadline is on Thursday 22^{th} March 2016 at 23:55 (Week 22).

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.

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

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).

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.