IJP Assignment 2

Introduction to Java Programming: 2015-2016

In the first assignment, you started from a “skeleton” application which we had provided. In this assignment, you will complete an entire application yourself - including the design of the object model, and the user interface. We would expect a good solution to be possible with about three days work (24 hours), although you will probably need to spend longer if you are aiming for a particularly good mark, or if you do not have a lot of previous programming experience. The course schedule allows for this by covering no new topics in weeks ten and eleven, but you will almost certainly want to start earlier and spread the work - this will allow time for you to get feedback from the demonstrators on your design and approach, before starting on the code. If you are taking courses which have exams, you should probably aim to finish the assignment well before the final deadline, to allow time for revision. The assignment will be published in week seven and you should be able to start thinking immediately about the design issues in section 1.

Tasks that you will need to do are marked with a ✐. *You should not spend an excessive amount of time struggling with the more advanced tasks* - especially if you do not have a lot of previous experience - it is still possible to obtain a very good mark even by omitting these. *Read the marking scheme carefully to understand how these will be assessed, and which tasks are essential for a good grade.*

1 The Application Design

For this practical, you will develop an application which allows the user to move around and manipulate objects in a very simple “virtual world”. The images will be two-dimensional, but the user should be able to move from one location to another and to look in different directions - similar to Google “Street View”[1], but with discrete images. In addition to the skills that you developed in the first assignment, this will give you some experience with designing object-oriented solutions, and graphical user interfaces.

✐ [1] You will need to start by choosing a “theme” for your world. This should consist of small number of connected “locations” and a set of images for each location which represent views in different directions. For example:

- ❑ The locations might be the rooms in your flat. You would need to have a set of images for each room, looking in different directions, and a “logical map” of the flat which shows how the rooms are connected.

[1] See: `http://maps.google.com/help/maps/streetview/`

- ❑ The locations might be an area around the University. Again you would need a set of photographs looking in different directions, and a logical map which shows how you can move from one location to another.
- ❑ You might create a completely fictitious environment, perhaps with hand-drawn locations.
- ❑ Be imaginative! The `Assignments` menu on the course website shows some examples of student work from previous years.

If you decide to use images from elsewhere, please be conscious of the copyright, or any restrictions on their use. Make sure that you acknowledge the source of the images, and that you do not violate any copyright restrictions.

✐ [2] Now think about the basic actions that the user will be allowed to perform - as a minimum, the user probably needs to be able to turn around (left or right) and move forward into the next location[2]. This requires some thought. What does ”turn” mean? 90 degrees? What happens if there are two doors in the same wall - how do you distinguish between the directions? What happens if the user attempts to go forward when there is no exit? Is there an ”up” and ”down” as well?

✐ [3] Now, think about the interface. It is useful to sketch on paper what you think this might look like. How will the user specify the commands? Buttons? Typing commands? Menu items? How do you prevent the user from taking impossible actions? How should the program react if they do?

✐ [4] Finally, think about the controller. What actions will it need to perform? Separating these from the GUI allows you to easily change the way in which the interface works - for example to change from a menu-based action to a button action.

2 Creating the Model

To model your world, you need to think about the design of the classes and how they would be used. For example, you may decide to have a class which models a `Location`. This would include a collection of images for the different views. What methods would be required? What are the different possibilities for storing the collection of images? Similarly, you may want to have a class which models the `World`. What methods would be required? How would this relate to the locations?

You will probably find it useful to think about several possible alternatives - consider how easy it would be to implement each of the actions you identified in task [2] using various different models. You might want to think about any extra operations that you would need for section 4 as well before you commit to a particular implementation. You may also want to consider how object-oriented features such as inheritance may be exploited in your design.

The design of the model is important – your description (and justification) of the classes you have chosen is a crucial part of the assessment. In practice, having a good design will make the coding much easier; a bad design will make it much harder.

✐ [5] Implement your model objects. You may use/modify code from the first assignment (including the supplied code), or the JavaFx examples if you find this a useful starting point (but make sure that you clearly state the source of any code which is not your own).

[2]Note that “turning” and always moving forward is much clearer than allowing the user to “move left” and “move right” (in which case it might not be clear whether they are looking in the same direction that they are moving!)

If you have high-resolution images, you will need to pre-scale them to make them smaller (See the FAQ). There is a limit on the size of the assignment file which you can submit, and *you will not be able to submit your assignment if it contains unnecessarily large image files*. Large images may also cause you problems with memory allocation in your program – there are ways of dealing with this, but you probably want to avoid these extra complications.

In a real project, this would be a good time to write some unit tests - this would allow you to convince yourself that the model objects were working properly before you get involved in the complications of connecting them to the interface. Unless you are finding the assignment easy though, you may want to move on and concentrate first on the following sections.

3 The Interface & Controller

[6] Now implement the GUI that you designed in task [3]. We strongly recommend that you use JavaFx for the interface – if you have good reasons for preferring a different toolkit, then you should discuss this first with the course lecturer. If you use a graphical designer such as SceneBuilder, you should be able to check the appearance and some basic behaviour of the interface without writing any additional code. The Video page on the website has some videos and sample code to help you get started with JavaFx.

[7] Implement the controller that you designed in task [4].

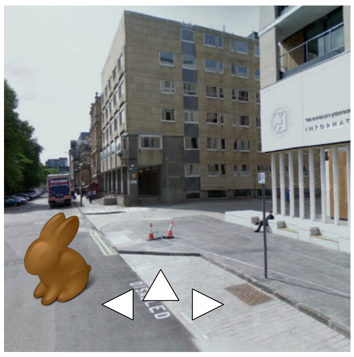
[8] You should now have a basic working application. You should probably save a copy of the code at this stage - this will give you a working copy to demonstrate for marking if you break things while attempting to extend the code in the following section.

ⓘ For a real project, you would almost certainly want to use a *Version control* tool[3]. This would allow you to save the state of your code every time that you make a change. This means that you can easily return to previous versions, and you can clearly see what was changed, and when. These tools are also particularly valuable in tracking and merging changes when multiple people are working on the same project. They do take some time to learn but if you intend to continue programming, you will almost certainly find this worthwhile at some stage.

[3]For example GIT: http://git-scm.com

4 Portable Items

Now that you are able to move about in your virtual world, let's add some items that you can carry from one place to another. For example, one location might contain a chocolate rabbit[4]. We should be able to pick up the rabbit, move to a different location and put it down again. In our ideal word, nobody would pick up the rabbit and eat it while we were away, so it should stay in the new location and be visible every time we visited that location. Of course, we should be able to pick it up again and move it somewhere else.



[4]Rabbit icon from: http://www.iconfinder.com/icondetails/67005/128/_icon

✐ [9] Extend your model to support some "portable" items. Make sure that your model is capable of supporting more than one item.

✐ [10] Extend your interface so that it displays any items which are present in the current location (whichever view we are taking). You might want to display your items overlaid on the main image, or in a separate section of the display.

✐ [11] Extend your interface so that the user can pick up and put down the items. Use a different interface element for this interaction - for example, if you used buttons to move between the locations, you might use a menu, or a text command to select the item.

✐ [12] You should now have a working application which allows the user to move items between locations.

5 Optional Extras

If you have significant previous programming experience, and are finding this exercise fairly easy, you may want to extend the application in some way. This would give you an opportunity to learn something new, and perhaps to obtain an exceptional mark. *But*: this is entirely optional – it is quite possible to obtain a distinction on the basis of the previous sections. So, you should *only do this if you are confident that you have completed the above sections well* (check the marking scheme), and you have time – you should also make sure that you have completed the worksheet (below) which is essential, before starting on anything extra.

If you are considering this, you may want to discuss your ideas with a demonstrator or the course lecturer, and you should remember to save a copy of your code before attempting to extend it.

6 Worksheet & Documentation

As well as submitting your code, you will also need to submit a worksheet with answers to a few questions. This should give you a chance to demonstrate some understanding of the design without being distracted by the details of the code. Your answers contribute significantly to the assessment (see the marking scheme), so you should give these as much attention as you have given the code. Especially if English is not your first language, *take care to express yourself clearly and concisely* - you may find the EUSA Peer Proofreading Service useful.

✐ [13] Answer the following questions on your worksheet (please number the answers clearly):

1. *Briefly* describe your class design, including the function and relationships between the important classes. You may do this using text and/or diagrams, but the result must be clear and concise.

2. *Briefly* explain why you made your final choice in preference to various alternatives - perhaps mentioning cohesion, coupling, efficiency, and/or clarity where appropriate.

3. Add any extra comments that you would like to make about your solution, or the assignment in general. In particular:

   ❑ You should note any external resources from which you took code, and any significant collaborations with others.

- ❑ It would also be useful if you could indicate here if you would be prepared to allow us to publish your screenshots – many students produce interesting interfaces and images and it is useful to be able to show these to others – for example, the course website shows some previous implementations.

You may create the worksheet using any application that you like, but please (a) Make sure that it in PDF format - the FAQ pages on the course web site explain how to generate PDF; (b) Make sure that your answers are concise – *do not be too verbose*; and (c) Make sure that your name and matriculation number are clearly visible.

✎ [14] Create a directory `javadoc` containing the HTML version of the Javadoc for your own classes. The FAQ pages on the course web site explain how to generate the Javadoc.

Make sure that there are no errors creating the Javadoc, and that you include all of the supporting files from the Javadoc directory[5]. Open the `javadoc/index.html` with a browser and read the HTML files carefully to make sure that you have not left any inappropriate comments from the template files. If your own Java installation is not in English, you should generate the Javadoc on DICE to make sure that the documentation is in English.

7 Submission

You should attempt your submission well before the deadline to allow for any problems that you may have with the submission system. If you need to update your submission (anytime before the deadline), you can simply make a new submission which will replace the previous one. *No marks will be awarded for submissions received after the deadline*[6].

Before you submit your assignment, make sure that you understand about plagiarism[7] - we have the ability run automated plagiarism checking on both worksheets and code. If you are in any doubt about any part of your submission, please discuss this with the course lecturer.

✎ [15] Generate a few screen dumps showing various screens from your application.

✎ [16] Create a ZIP archive (see the submission page on the website) containing the following files (please use these exact filenames):

- ❑ `javadoc` - a directory containing your Javadoc in HTML.
- ❑ `src` - a directory containing all of the source files necessary to compile your application.
- ❑ `screendumps` - a directory containing images of a few screen dumps from your running application.
- ❑ `worksheet.pdf` - your answers to the worksheet questions.
- ❑ `assignment2.jar` - the jar file containing your runnable application.

You may want to unpack the archive again yourself to verify that it contains the files that you expect.

✎ [17] Use the online submission system (described on the website) to submit your ZIP file.

[5] The directory should contain all of the style files and index files necessary to view your Javadoc correctly.
[6] See the School policy: `http://edin.ac/18DwiRZ`
[7] See the plagiarism page on the course website, and the School policy here: `http://edin.ac/18DwG2N`

ⓘ It should be possible to run your application simply by (double) clicking on the jar file. But it is not always easy to produce a jar file which runs correctly on different systems – there may be differences in the operating system (pathnames, for example), in the versions of available libraries, in the screen characteristics, etc. which mean that your code is not *portable* – this is why we will assess the functionality of your code mainly on the basis of your demonstration. For a real application, you would need to test your final jar file on a range of different systems.

8 That's It!

During the lab session in week twelve, you will be asked to show your solution to the demonstrator and a small group of students. This is a valuable opportunity to get feedback and to see potentially different approaches to tackling the same problem. The demonstrator will also be asked to check that your code runs as expected, and give you chance to explain the code, and any problems that it may have.

✎ [18] Make sure that you attend this lab session (on time!), and that *you are able to run the same version of the code as the one that you submitted*.

We hope that you found this a useful and realistic exercise. Please do let us know what you think. Ask the lab demonstrators if you would like any further feedback, or use the forum to discuss any issues relating to this exercise.