Assignment 2

ippo 2019-2020

Please read this entire document so that you have a clear idea of what is involved in the various stages and what you should do when, before starting work on any of the tasks.

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 class 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 don't have a lot of previous experience.

Don't be intimidated if you don't have much previous programming experience – you don't have to complete all of the sections, and you may be surprised at how much you can achieve by working steadily and following the instructions carefully. As with the first assignment, you should pay careful attention to the marking criteria (appendix A). A small, simple, clear design is much better than an over-complex one.

Especially if you have programmed before in some other language, *do not underestimate the importance of having a good, clear model before starting your implementation* – if the model is good, the implementation should be straightforward; if not, the implementation is likely to be awkward and difficult - even if it appears to "work". This is the single most important point of the course, so it won't be possible to pass by ignoring this - solutions which are not well-designed are not useful and will not pass, regardless of how well they appear to run.

To help you create a good design, the assignment is split into two stages: in the first stage (part A), you will need to study the book chapters and create an initial design of your own. I will then talk about this in the third lecture, and you will have an opportunity to re-think your design and discuss it with others before revising and implementing your final design (part B). It is important that you spend time thinking about the initial design yourself, so this will contribute to the final mark, as well as the final design.

The two parts have separate deadlines (see the schedule for submission A and submission B).

1 The Application

For this assignment, 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, using graphical user interfaces, and working with large libraries of external code.

1.1 Locations

[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:

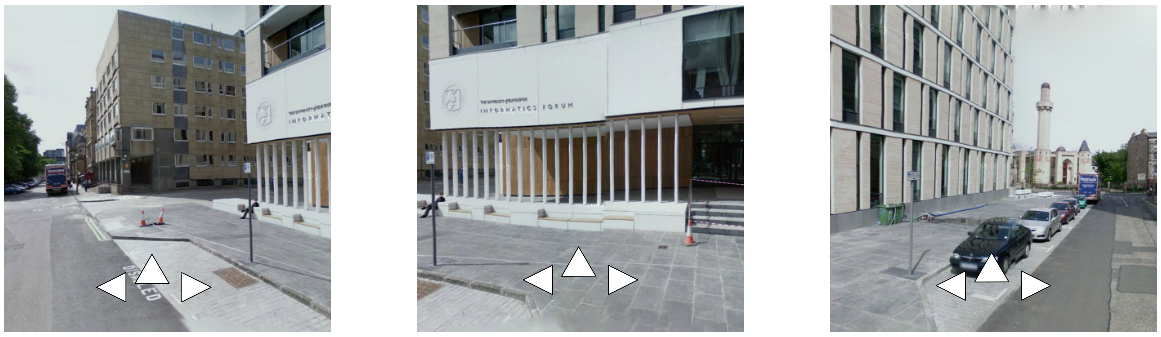
- The locations might be the rooms in your flat. You would need to have a set of images for each

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

room, looking in different directions, and a "logical map" of the flat which shows how the rooms are connected.

- 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.
- You won't get extra marks for "imagination", but creative scenarios are welcome!

If you decide to use images from elsewhere, make sure that you acknowledge the source of the images, and that you do not violate any copyright restrictions.

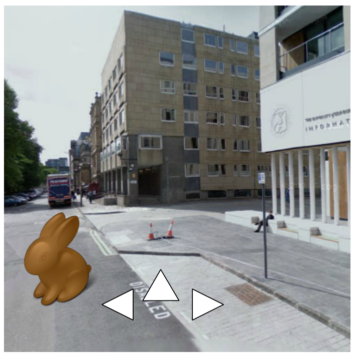


✐ [2] Now think about the basic actions that the user will be allowed to perform - as a minimum, the user probably needs to be able to turn around (left or right) and move forward into the next location[2]. This requires some thought. What does "turn" mean? 90 degrees? What happens if there are two doors 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?

1.2 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[3]. We should be able to pick up the rabbit, move to a different location and put it down again. Providing that the rabbit doesn't melt or get eaten, it should stay in the new location and be visible every time we return there. Of course, we should be able to pick it up again and move it somewhere else.



✐ [4] Extend your interface design so that it displays any items which are present in the current location. You might want to display your items overlaid on the main image, or in a separate section of the display, for example.

[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!)

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

You will also want to consider whether the items should be visible regardless of the direction in which you are facing, or only be visible in one direction. This is your choice, but these kind of decisions may have a significant effect on your design.

[5] Think about the additional actions that the user will need to perform, such as picking up and putting down items.

[6] Extend your interface design to support these actions. 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, or you may simply allow the user to click on the item.

Make sure that your design is capable of supporting an arbitrary number of items.

2 Designing the Model (Part A)

The design of the class model is the most important part of this assignment – submission of a design document describing a good, well-reasoned choice of classes and methods is essential in order to pass. Your design will be assessed both on your initial design (part A), and on any changes that you decide to make for the final implementation (part B).

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?

In practice, having a good design will make the coding much easier; a bad design will make it much harder. So, you will want to think carefully about the possible alternatives - consider how easy it would be to implement each of the actions you identified in tasks [2] and [5] using various different models. *Read the appropriate book chapters*. Think about *nouns* and *verbs*. Consider properties such as *cohesion* and *coupling* – how easy would it be for other people to use your classes in their own application?

In addition to the more concrete entities, such as locations and portable items, you will need to think carefully about the controller and the GUI. What methods should they support? Separating these clearly 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. It also makes the system easier to test, and for multiple people to work on the same application. You may want to look carefully at the classes from the first assignment and how they are structured – you should notice some useful similarities.

Thinking about the following questions may help you to evaluate your own design - *you do not actually have to do any of these tasks* - just think about how easy, or hard they may be:

❏ How easy would it be to replace the JavaFx with some other graphical interface? What proportion of your classes depend on JavaFx? Could you replace the graphical interface with a text-based one, without changing most of your classes?

❏ How easy would it be to test your classes? Would it be possible to write a test to show that all of the locations are accessible from the starting point (possibly via other locations)?

❏ How easy would it be for someone else to add a different kind of “item” to your application (perhaps one which included sound). How many of your classes would be affected by this change?

❏ Rather than retrieving the images from the local filesystem, could you fetch them from the web? Could you make use of any of the services from the first assignment to do this? You probably don’t use the same `Picture` class as the services, but could you create an "adaptor" class which translated the `Picture` objects into the format that you use?

- ❏ If you were fetching the pictures from the web, would any of the proxy classes (`CacheProxy`, `RetryProxy`?) from the first assignment be helpful?
- ❏ How easy would it be for someone to create a game out of your application – for example, displaying a score to show how many items had been collected?

Such extensions should not require big changes to your code, and should only involve a small percentage of your classes.

3 Submitting the Design (Part A)

For Part A of the assignment, you should submit a short document describing your design. There are various ways of doing this formally[4], but the main aim is to show what classes you have, what their responsibilities are, and how they relate. So a simple list of classes and methods, with brief descriptions, and perhaps an informal diagram, is sufficient in this case. You should also *briefly* describe one or two significant choices that you had to make in the model design and explain why you chose your particular design over some plausible alternative. You should aim to be extremely clear: it is probably best to avoid lengthy text descriptions, and take special care if English is not your first language. You may create the design document using any application that you like, but please make sure that the submitted copy is in PDF format, and that it has no more than about two pages.

✐ [7] Submit your assignment:

1. Create your design document as `design.pdf`.
2. Make sure that your name and student number are clearly visible at the top of the document.
3. Create a ZIP archive containing this single document. You may want to unpack the archive again yourself to verify that it contains the expected file.
4. Use the Submission Form to submit your ZIP file.

As for the first assignment, it is a good idea to 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. Similarly, *late submissions will not be accepted* because we will be discussing the designs as soon as they have been submitted.

✐ [8] Attend the third lecture, and revise your design to take into account anything new which you may have learned.

4 Implementation (Part B)

⚠ Most people will want to modify their design to take account of the third lecture. If you start implementation before then, you should be prepared to change significant parts of your code if necessary to improve the design.

To help you get started, we have provided a Gradle build file and a basic `Main` class. We recommend that you use this build file because it includes a task to package your solution for submission:

✐ [9] Download the Zip file containing the template project, and import this into a new IntelliJ project.

[4]For example, UML: `https://en.wikipedia.org/wiki/Unified_Modeling_Language`

The template assumes that you have a an application class called `App`. You may find it useful to create a temporary `App` class (perhaps based on the Getting Started example) to check the basic project structure is correct.

We suggest that you start by implementing (only) those classes which are necessary to create the basic application (moving between locations), and leave the portable items until the basics are working. If you don't have a lot of experience, you may find it difficult to implement the portable items, and it is possible to pass without doing this, providing that you have a sound design, and a working implementation of the basic locations.

In your implementation, you may find it useful to use or modify code from elsewhere. This is fine - but you will only be credited for code which you have written yourself, and you must make sure that you clearly state the source of any code which is not your own - you should do this both in the code (comments), and on the submitted worksheet. If you do use external libraries, make sure that these are included in your submitted jar file.

You will find it helpful to adopt a standard size for the main images (say 512 x 384) and the item images (say 64 x 64). This will make it much easier to share your model with others (see section 5). There is also 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*. In addition, large images may also cause you problems with memory allocation in your program – there are ways of dealing with this, but keeping the images sizes small will avoid these problems.

✐ [10] Implement the basic model classes.

> 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 probably want to move on and concentrate first on the following sections.

✐ [11] Implement the GUI.

You must use JavaFx for the interface, and create your design using SceneBuilder. This is probably new to most of you. This is intentional. It is deliberately intended to give you a realistic experience of working with an unfamiliar library using only the public documentation and support available on the web. The Scenebuilder application will allow you to layout your interface graphically, and check the appearance and some basic functionality without writing any code.

The course website includes some Code and an associated note Note for a few simple JavaFx examples to get you started.

✐ [12] Implement the controller.

You should now have a basic working application. You should save a copy of the code at this stage – this will give you a working copy to submit for marking if you break things while attempting the following sections!

> For a real project, you would almost certainly want to use a *Version control* tool so 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. See the description of Git in the Development Tools document.

✐ [13] Once you have a working application which allows you to move between locations, you should extend this to include the portable items.

5 An Advanced (Optional) Task: Loading the Model

This section is intended for those of you with more programming experience who would like to learn a little more about using Java in practical applications - specifically locating and using external libraries, and reading structured data in JSON format.

> ⚠ It is possible to obtain a very good mark without attempting this section, and if it is implemented badly, it can spoil a good design of the basic program which may actually result in a *lower* mark! So, only attempt this if you are confident that you have a solid solution to all of the other parts of the assignment, and you are willing to spend additional time. You should save a copy of the code at this stage, so that you have a working copy to submit if you later decide that the extensions have resulted in a poorer class design, for example.

You will hopefully have realised from the first assignment, that it is useful to separate the "content" from the code: rather than "hard-wiring" the Munro names into the controller code, they could be stored in a separate property file. This allowed other people to easily change the data without understanding, or re-compiling your code. It also allowed the same data to be used with different implementations, simply by copying the property file.

In this case, our model is a little more complicated. It *would* be possible to represent this in a way which could be stored in a property file, but this would be rather awkward to deal with, and difficult for other applications to read. JSON[5] is a standard way of representing structured data in plain text files. JSON libraries are available for most languages, so this makes it a convenient format to represent our model.

Figure 1 shows how a simple model might be represented in JSON[6]. This contains a number of rooms (`bedroom`, `hall`, ...). Each room specifies a collection of image files representing the view in different directions, from inside that room. It also has a number of exits (in different directions) leading to other rooms. And it has a list of items initially present in that room. The portable item records just define the image file representing each item.

✐ [14] Explore some of the JSON libraries which are available for Java and choose an appropriate one for your application. For example, simplicity is probably a useful criteria in this case, but high performance is probably not.

✐ [15] Modify your application so that it reads the model from a JSON file in the same format as the one in figure 1.

If your application design is good, it should be possible to do this without major changes to the class structure. You may find the following hints helpful:

❑ The JSON entries (for example the exits) reference the rooms by their names. You will need some way of looking up the names to find the corresponding objects. And you will need to think whether it is better to store the names of the rooms (and look up the objects when you need them), or store the references to the objects themselves.

❑ Delegate the responsibility for interpreting the various parts of the JSON to the appropriate classes.

Your application should provide some method for the user to select different JSON files (and hence

[5] `http://www.json.org/`, `https://en.wikipedia.org/wiki/JSON`

[6] There are, of course, many other possible representations

```
{
  "rooms": {
    "bedroom": {
      "views": {
        "north": "bedroom-north.jpg",
        "west":  "bedroom-west.jpg",
        ...
      },
      "exits": {
        "north": "bathroom",
        "south": "hall"
      },
      "contents": [ "cat", "dog" ]
    },
    "hall": {
      ...
    },
    ...
  },
  "items": {
    "cat": { "image": "cat.jpg" },
    "dog": { "image": "dog.jpg" }
  }
}
```

Figure 1: Representing the model in JSON.

different models) - perhaps from a menu, or from a prompt when the application starts. If this is to be useful for sharing models, you will need to be careful about the location of the image files - you should assume that these will be in the same directory as the JSON file.

✐ [16] See if you can find someone else who is attempting this section (try the Piazza forum), and exchange your model with theirs (do no exchange any code). Your applications should be able to display each other's model.

You might find it fun to try and merge two models: you should easily be able to create a composite JSON file which allows to you to move between different worlds (via specific exits) in the same application!

6 Submitting the Implementation (Part B)

Packaging a real application for distribution is difficult to do well. JavaFx, for example, is *platform-specific* and the jar file created by the simple Gradle build task will run only on the platform on which it was created - if you have developed on Windows, we will not be able to run the jar directly on a Mac or on Linux. Building a good cross-platform binary distribution takes a significant amount of effort and is beyond the scope of this assignment.

However, the Gradle build file includes a `submission` task which will package the *source code* for your application as a zip file. We will then unpack this and compile and run the code on our own systems using `gradle run`.

✐ [17] As well as submitting your code, you will also need to submit a worksheet containing the following:

1. A *brief* description of any changes you made to your initial design. If you did not make any changes, then you should write a brief justification of this.
2. 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.

As before, you should probably attempt your submission before the deadline to allow for any problems that you may have with the submission system, and *late submissions will not be accepted* because answers to assignment questions and feedback will be made available during the demonstration sessions in the following week.

[18] Before you submit your assignment ...

1. Create a PDF version of your worksheet in `doc/worksheet.pdf` in the project directory.
2. Look at the notes on code Readability and use this to review the readability of your code.
3. Please make sure that you understand about good scholarly practice – see the Course Handbook for further details. We have automated tools for checking both worksheets and code for similarity with other submissions (including previous years). If you are in any doubt about any part of your submission, please discuss this with the course lecturer.
4. Generate a few screen shots showing various screens from your application. These will show the marker what your application looks like if (s)he is unable to run your code. Put these in the `doc` subdirectory of the project directory together with your worksheet.
5. Use the Gradle `submission` task to create a zip file. You will probably want to unpack this zip file to ensure that it contains all of the files that you would expect.

[19] Submit the zip file using the Submission Form on the course web page.

7 That's (almost) It

In the lab session following the deadline, on-campus students will be asked to show their running tests and assignment code to a demonstrator. This will give you an opportunity to explain any problems that you had with the code and to get some feedback on your solution. This is difficult to schedule, so . . .

[20] Please make sure that you attend this lab session on time, and that you have all of the necessary files to run and demonstrate the same version of your code as the one you submitted (we will check this).

The demonstrator will *not* be marking your assignment, but we will refer to the demonstrator's notes if, for example, we cannot run your submitted assignment code on our systems. If you do not attend the demonstration, code which we are unable to compile and run (for any reason) will not be awarded any marks.

If you are taking the distance version of this course, and we have trouble running your code, the tutor will contact you to discuss this, and possible arrange an online demonstration.

8 That's (really) It!

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.

Appendix A Marking Criteria

When assessing the assignment submission, we will consider the following criteria:

Class Design: The class design is the most important criteria. The initial design must be plausible and the final design must demonstrate good design principles (see section 2).

User Interface: The user interface should be clear and simple to use.

Completion: A good solution will require a working implementation which supports locations and portable items. However it should be possible to pass without a working implementation of the portable items, providing that the design is good. The advanced task (section 5) will only be taken into account if all of the other criteria are good.

Additional Criteria: As for the first assignment, the following will also be taken into account:

- Readability & code structure; Correctness & robustness; Use of the Java language

Marks will be assigned according to the University Common Marking Scheme[7]. The following table shows the requirements for each grade. All of the requirements specified for each grade must normally be satisfied in order to obtain that grade.

Pass (Diploma only): 40-49%:

- Submit a final design for the application which shows some awareness of good design principles.
- Submit some plausible code for a significant part of the assignment, even if it does not work correctly.

Good: 50-59%: *in addition ...*

- Submit a plausible initial design.
- Submit working code for an application which supports movement between locations, even if there are small bugs or omissions.
- Submit code which is sufficiently well-structured and documented to be comprehensible.

Very Good: 60-69%: *in addition ...*

- Submit a final design which incorporates some good design principles.
- Submit working code for an application which supports portable items, even if there are small bugs or omissions.
- Submit code which is well-structured and documented.

Excellent: 70-79%: *in addition ...*

- Submit a final design which is clearly based on good design principles.
- Submit and demonstrate working code for all parts of the assignment, with no significant bugs.

Excellent: 80-89%: *in addition ...*

- Marks in this range are uncommon. This requires faultless, professional-quality design and implementation, in addition to well-reasoned and presented description of the design on the worksheet.

Outstanding: 90-100%:

- Marks in this range are exceptionally rare. This requires work “well beyond that expected”.

[7] `https://web.inf.ed.ac.uk/infweb/student-services/ito/students/common-marking-scheme`