Assignment 2

ippo 2021-2022

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 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.

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. You will then implement that design in part B.

1 The Application

For this assignment, you will develop an application which allows the user to move around and interact with objects in a very simple 3-dimensional "virtual world". The images of this world 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.
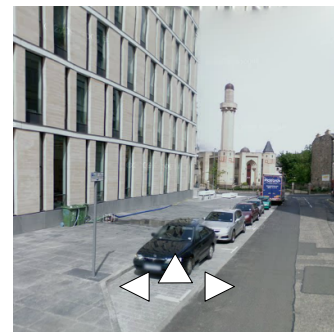
1.1 Locations

✎ [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 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.

---

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

- 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, they should be able to turn 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. Will the user navigate by typing commands, pressing buttons or using 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. The rabbit 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.
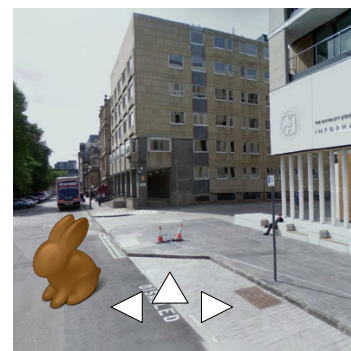
✎ [4] Extend your interface design so that it displays any items which are present in the current location. You might want to display your items overlaid on the main image, or in a separate section of the display.
You will also want to consider whether the items should be visible regardless of the direction 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.

---

[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

✐ [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.

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.

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 maintain (especially important for large companies). You may want to look carefully at the classes from the first assignment and how they are structured – you should notice some useful similarities.

Make sure that your model is capable of supporting an arbitrary number of items, even if it will be limited in practice by the number of items that a particular view is able to display. In particular, make sure that adding an extra item does not require a lot of extra code – compare this with the task of adding extra Munros to the first assignemnt.

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 alternative - perhaps using a different library (e.g. Android or Swing), or a web-based 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. This is a good test for the quality of your design!

3 Submitting the Design (Part A)

For Part A of the assignment, you will need to submit a short document describing your design. There are various ways of describing 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.

✎ [7] Create your design document as a PDF file called `design.pdf` including:
- A clear and concise description of your design.

- A *brief* description of one or two significant choices that you had to make in the model design with an explanation of <u>why</u> you chose your particular design over some plausible alternative.

You should aim to be extremely clear: it is best to avoid lengthy text, and instead focus on using precise 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 two pages (markers will not read further!).

As for the previous assignments, you will need to create and link a GitHub repository. Please take care with this process because it can be very confusing if the repository is wrongly configured, or your files are not in the expected location:

✎ [8] Accept the Assignment invitation[5] and make a note of your repository URL.

✎ [9] Create a new empty directory (eg. `ippo-assignment2`) to hold your assignment files, copy in your design document, and change into this directory.

✎ [10] Add your design document:

```
➜ git init
➜ git add design.pdf
➜ git commit -m "design document"
```

✎ [11] Push your repository to GitHub:

```
➜ git branch -M main
➜ git remote add origin url-of-your-repository
➜ git push -u origin main
```

You will probably also want to add any source files (eg. Latex or Word) which you used to generate the PDF.

Unless you intend to create separate branches, you can add and push updated files very simply. For example:

---

[4]For example, UML: https://en.wikipedia.org/wiki/Unified_Modeling_Language
[5]https://classroom.github.com/a/2kFSsiyi

```
➔ git add my-file
➔ git commit my-comment
➔ git push
```

4 Implementation (Part B)

Starting from a completely "blank page" always involves more work. You may find it tempting to use or modify code from elsewhere. This is only allowed if you clearly cite the source you are copying from via comments in the code and again in the worksheet. Even then, we can only give you marks for your own work. For this reason and for your own learning, we strongly recommend writing all code yourself.

Similarly, the use of external libraries (other that the ones mentioned) is not recommended. You will not get credit for the library code and you must also ensure that your code runs on all platforms (which gets harder as you add dependencies). If you really must, please make sure that the additional libraries are included in your Git submission and test your code on different platforms.

Your first goal should be to create a minimal application that does very little, but can be run without errors. This provides a basis to build on.

We suggest that you start by implementing (only) the basic model classes which are necessary to create a minimal 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.

It will be helpful if everyone uses the same size for the main images (512 x 384) and the item images (64 x 64). 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. As a benchmark, the entire samples solution with code and images is less than 0.5 mb. You are allowed a maximum of 10 times that i.e. 5mb. If you exceed this limit, you may lose marks as unnecessarily large image files will cause us problems when we pull your repository.

✎ [12] Implement the basic model classes.

> ℹ️ 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.

✎ [13] 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 Learn page offers some videos to help you get started with SceneBuilder and JavaFx.

✎ [14] Implement the controller.

You should now have a basic working application. It is a good idea to preserve this in some way, so that you can easily return to it (for example, to submit it) if you break things while attempting the following

sections. One good way to do this is to add a Tag[6] to the Git version - this allows you to provide a name for a specific revision that you can use to access that version in the future. You will also need to push the tag to the server if you want it to be stored in the remote repository:

```
➜ git tag my-tag-name
➜ git push origin my-tag-name
```

✐ [15] 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. Remember to tag your existing version so that you can easily return to it.

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[7] 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[8]. 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.

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

✐ [17] 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

---

[6]https://git-scm.com/book/en/v2/Git-Basics-Tagging
[7]http://www.json.org/, https://en.wikipedia.org/wiki/JSON
[8]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.

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 different models) - perhaps from a menu, or from a prompt when the application starts. We recommend keeping your image files in the same directory as the JSON file.

6 A Video

Writing portable code is not easy, and it is possible that your submitted code will not run on our systems, even though "it works for you". As an insurance policy, we would like you to ...

✎ [18] Create a very short screen capture video demonstrating the features of your assignment, and explaining how to use it.

You will be asked to provide a URL (in your worksheet) where we can view the video. You may create and publish this however you like, but we recommend using the University MediaHopper service which has a simple application for creating, uploading and publishing the video. Instructions can be found on LEARN 》 Resources 》 MediaHopper .

Don't make your video too long - around one minute is a good length.

⚠ Do not check the video in to your Git repository! The large size will make it very difficult for us to check them out.

7 The Worksheet

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

1. The following declaration: I declare that this assignment was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

2. Cite any external resources from which you took code or other materials.

3. Provide a link to your video.

4. Any extra comments that you would like to make about your solution, or the assignment in general.

Make sure that your worksheet is concise - we would not expect this to be more than a page, and in many cases it may just be a short paragraph.

✐ [19] Create your worksheet as PDF file `worksheet.pdf` at the top level of your Git repository.

8 Submitting the Assignment (Part B)

Packaging a real application for distribution is difficult to do well. JavaFx, for example, is *platform-specific* meaning that e.g. a jar file 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. So, we will simply check your application source code out of the Git repository and compile and run it on our own systems. Please do <u>not</u> submit platform specific libraries or even whole SDKs!

✐ [20] Before the submission deadline ...

1. If you have access to other platforms (eg. DICE as well as Windows) you should check out your application and attempt to run it on a different platform. This will identify any portability problems and increase the chances of us being able to run the submitted version.

2. Look at LEARN ≫ Resources ≫ CodeReadability.pdf and use this to review the quality of your code.

If we have any difficulties understanding your submission, or any other queries, you may be asked to answer questions by email, or to discuss and demonstrate your application online.

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[9]. 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 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".

---

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

---