4CCS1PPA Programming Practice and Applications

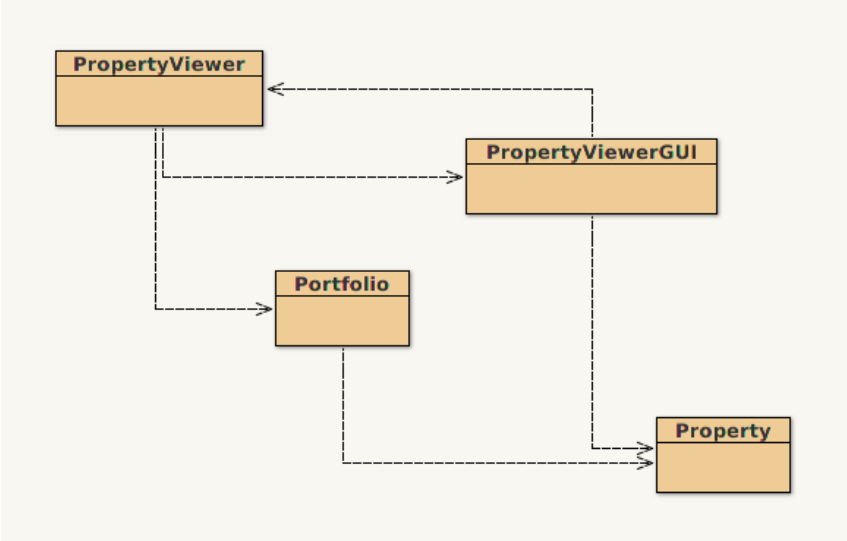
Coursework 1: Property Viewer

Michael Kölling and Jeffery Raphael
Questions? `programming@kcl.ac.uk`

The goal of this assignment is to implement an application that allows the user to view properties loaded from a spreadsheet. The application is partially implemented, but you need to finish implementing some of the methods, and fix one that is currently not working as intended.

Getting started

- You will need to download the 'Property' project from the KEATS page. This is the partially implemented project.
- When you open it, you will see the class structure depicted below.
- Three of the four classes are fully implemented so you do not need to modify them. One class, `PropertyViewer`, has been left unfinished, and it is your job to finish the implementation of this class.



Project Overview

Here is a quick overview of the existing classes:

- `Property`
  - This class represents a single property. You should take a look inside this class.
  - It has methods you will need to use, such as `getID`, `getLatitude`, and `toggleFavourite`.
- `Portfolio`
  - This represents a collection of properties.
  - A portfolio is built by specifying a spreadsheet on disk with the data on some properties in it (this is by default the file called `airbnb-london.csv` — this is real-world data are some of the actual AirBnB listings).
  - The portfolio will automatically load all the properties that it finds in that spreadsheet. Feel free to add more properties to the spreadsheet if you like!
- `PropertyViewerGUI`
  - This class presents the GUI (Graphical User Interface) of the application. That is: it draws the main window, the buttons, and all the other things you see on the screen.
  - The class does two additional things: (1) if the user clicks a button, that call is passed onto the relevant `PropertyViewer` method; (2) the `PropertyViewer` class may call this one to display a `Property` or `String` in the interface.
- `PropertyViewer`
  - This class implements the logic of the property viewer.
  - This is where you have to do your work.
  - This is also the class that you instantiate to start this application.
  - This is also the class that you instantiate to run the application.

Base Tasks

To complete the assignment you have to do the following tasks.

- When the application is started, the first property in the portfolio (index 0) should be automatically displayed. **(6 pts)**
- With any property that is displayed, the ID of the property should be shown near the top of the window. The GUI class has a method to do this. **(10 pts)**
- When the *Toggle Favourite* button is pressed, the `isFavourite` field of that property should be updated. There is a method in the `Property` class to do this. **(15 pts)**
- The bar at the bottom of the window should show whether the property has been marked by the user as one of their favourites. **(10 pts)**
- When the *Next* button is pressed, the next property should be displayed, with the correct data. Furthermore, the ID at the top should be updated correctly, as well as whether the property is one of the user's favourites. The application should return to the first property if the *Next* button is pressed while on the last property. **(12 pts)**

- When the *Previous* button is pressed, the previous property should be displayed, with the correct data. Furthermore, the ID at the top should be updated correctly, as well as whether the property is one of the user's favourites. The application should go to the last property if the *Previous* button is pressed while on the first property. **(12 pts)**
- In the PropertyViewer class, several methods are undocumented. Provide appropriate method level comments. **(5 pts)**

Challenge Tasks

These are tasks you should complete only when you have completed the base tasks. Note, challenge tasks may require knowledge far beyond that which we have covered so far in the course.

- Implement a method named `getNumberOfPropertiesViewed` that returns the number of properties that have been viewed since the application was started. The return type should be `int`. Viewing the same property twice counts as two views. **(5 pts)**
- Implement a method named `averagePropertyPrice` that returns the average price of the properties viewed so far. The return type should be `int`. **(10 pts)**
  - Example: viewing property A (£50), then property B (£20), and then property A again counts as 3 views, and the average price should be $\frac{50+20+50}{3} = 40$.
- The *View Property on Map* button is currently broken, in that it only every displays Bush House on the map. Fix this functionality so that it instead displays the location of the property on the map. **(5 pts)**
- Add a new *Statistics* button to the application. When clicked, a new window should open that displays the statistics information from the two new methods from the Challenge Tasks. **(10 pts)**

Submission and Deadline

- You must submit the following on KEATS by **Friday, Oct. 28th, 16:00 (4pm)**:
  1. A Jar file of your BlueJ project.
     - You can create a Jar from within BlueJ by going to Project, and then "Create Jar File...".
     - You do not need to change any of the default options, and so you should just click the "Continue" button.
- **The Jar file must contain your source code, i.e., the *.java files, and it must run on BlueJ.**
- Late Submissions. If you submit late, but within 24 hours of the deadline, the work will be marked, and 10 raw marks will be deducted. If this deduction brings your mark below the pass mark (40%), your mark will be capped at 40%.

Marking

Applications are marked based on four categories:

1. Program Correctness —The application meets all of the program specifications, i.e., the student has completed all of the base tasks including following submission instructions (e.g., the student submitted a Jar file of their BlueJ project).
2. Code Elegance —The application is written in such a way that the code is reusable and efficient (i.e., memory usage and complexity). The application appropriately uses loops and functions to reduce code complexity and/or repeated code. The application does not have hard-coded solutions or poorly designed solutions. A poorly designed solution is overly complicated, utilises excessive amounts of memory or utilises a slower approach to a problem.
3. Documentation —The application is sufficiently documented. Good documentation/comments should explain what the code does and how it does it. Comments can also be used to highlight nuances in your solution, e.g., a segment of code that only works under certain conditions.
4. Readability —The application is easy to understand and uses good programming practices.