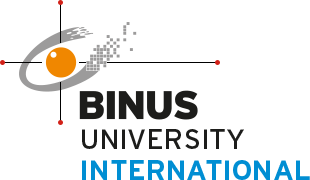
OOP Final Project Report



Kenny Lukito

(2602195306)

Table of Contents

Project Specification —----- —---------3

Solution Design —---------- —---------4

Working Mechanism —---- —---------6

Evidence of Working Program —---------8

3

Project Specification

In this report, there will be the description of the project, containing the project’s purpose, the problem it solves, data structures, algorithms, and the outcomes of the project. In addition, it will also contain the solution diagrams, the working mechanisms, and the demonstration of the code.

The project is a handyman hiring application, which is used to hire handymen provided by the app through mobile gadgets so people can find help without having to find a handyman by themselves. The purpose of the app is to make finding handymen more convenient for its users. Usually, people have to look up on the internet, looking for billboards, or asking for a handyman’s recommendations. The hassle will be majorly eased with the final project application.

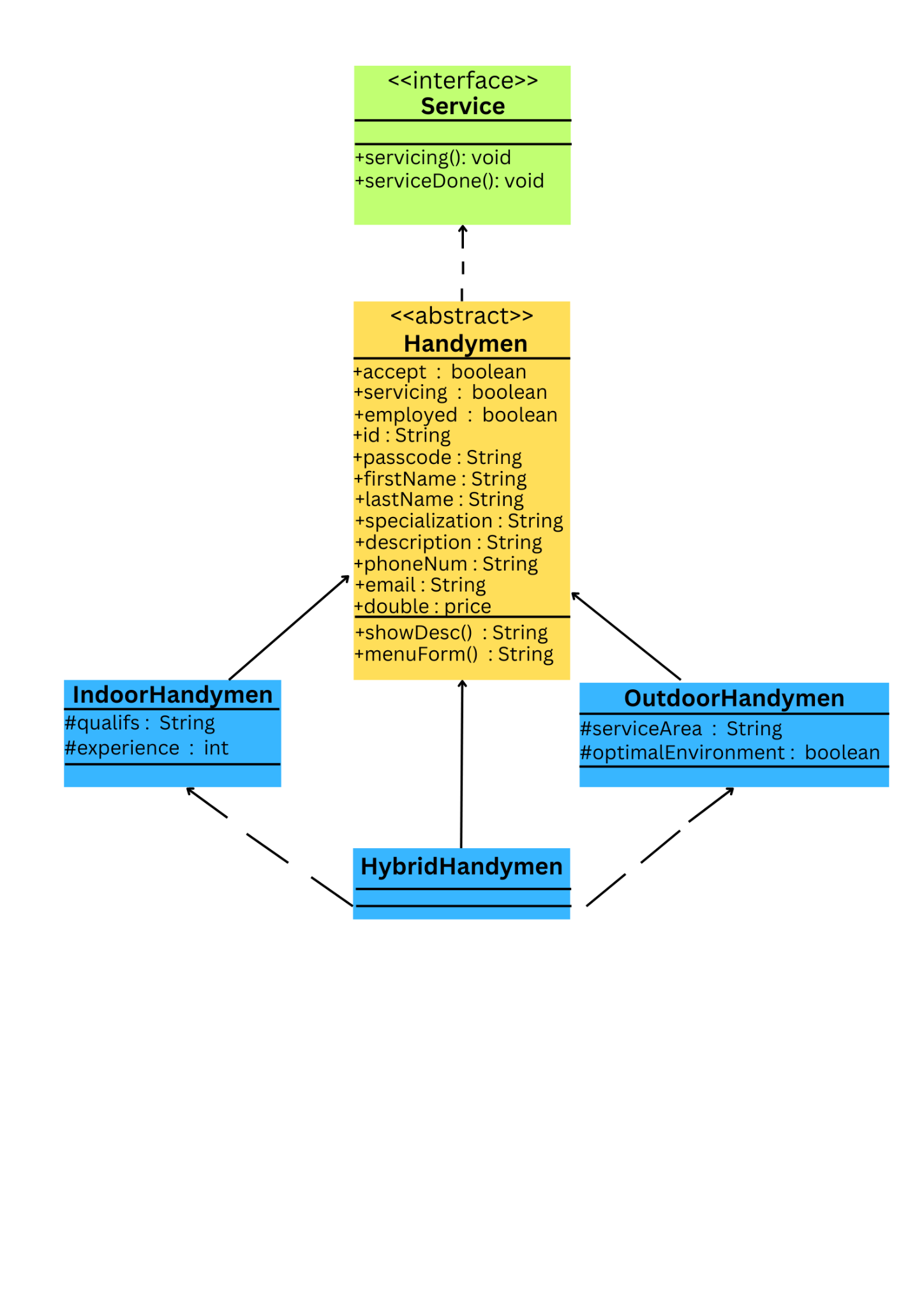
This project has main features such as hiring recommended handymen, searching for specific handymen and hiring them, checking handymen status, confirming a handyman’s service progress, and paying the handyman’s service cost.

It is noticeable that the application is run in the terminal instead of having a UI. Due to my lack of experience and confidence in creating and operating UIs, I chose to use terminal as the user interface.

4

Solution Design

To visualise the workings of the program, a class diagram will be shown below:



As seen on the diagram above, the interface “Service” provides the mandatory standard methods for each class. The abstract method servicing() serves as a basic method for related classes to set the “servicing” attribute of the implementor class to true. In addition, the serviceDone() method is used to set both “servicing” and “employed” attributes to false, representing that the handymen is dismissed, which applies to classes that implement the “Service” interface. Now that the abstract class “Handymen” implements the Service, it includes the methods found in the interface. Also, the “Handymen” abstract class also provides the attributes and its own methods. Some methods in the “Handymen” class are showDesc() that returns a string showing the details of a handyman, and menuForm() that returns the shortened version of showDesc() designed to fit in the menu screen. After that, 3 classes named “IndoorHandymen”, “OutdoorHandymen”, and “HybridHandymen” extend from the abstract “Handymen” class. The “IndoorHandymen” class inherits from the “Handymen” abstract class with “qualifs (String)” and “experience(int)” as its own exclusive attribute. For the “OutdoorHandymen”, it also contains the exclusive attributes “serviceArea(String)” and “optimalEnvironment(boolean)”. While for “HybridHandymen”, it uses the combined attributes of “IndoorHandymen” and “OutdoorHandymen” objects. Its constructor involves making an instance of each parent class. The “HybridHandymen” class does not have their exclusive features.

6

Discussion

Algorithms used:

* Hire(Scanner scn, Handyman handyman)
* removeTagDuplicates(ArrayList<String> listB)
* employedHandymen(ArrayList<Handymen> listC)
* invalidInput(Scanner scn)
* tagNotExisting(Scanner scn)
* handymenAlreadyHired(Scanner scn)
* invalidID(Scanner scn)
* If-else statements
* For loops
* While loops
* Methods

The Hire(Scanner scn, Handyman handyman) method takes 2 parameters, which is the scanner and a handyman instance. It shows a chosen handyman’s description with a prompt to proceed to requesting service to the handymen. Once the user requests the service, the handymen will choose whether they approve or decline the user’s request. If approved, the user will be given handymen’s contact details along with the terms and conditions, and finally a prompt to hire the handymen. If the user agrees to hire him/her, the method sets the selected handyman’s “employed” attribute to true and moves on to the next screen.

For removeTagDuplicates(ArrayList<String> listB), it works by taking a parameter of an ArrayList of String representing handyman’s specialisations and filters them such that there are no duplicates in the specialisation lists of all handymen. It is utilised to display unique specialisations for the user to search.

While employedHandymen(ArrayList<Handymen> listC) works by filtering an ArrayList of Handymen instances so that it returns an ArrayList of only hired Handymen instances (i.e. “employed” attribute is true). The method is used in the handymen hiring process for checking whether the selected handymen is already hired, and returns a message saying that the handymen is already hired if so.

The rest of the methods return messages upon encountering a problem. For example, invalidInput(Scanner scn) returning error message for invalid inputs, tagNotExisting(Scanner scn) returning message that the specialisation tag is nonexistent when the user searches for specialisations not available, handymenAlreadyHired(Scanner scn) saying that the selected handyman is already hired when the user is trying to hire those who are already hired, and invalidID(Scanner scn) saying that the ID is invalid during the ID selection prompt where the user enters a nonexistent ID.

If-else statements are used as a prompt to navigate through the app (selecting options and returning to previous screen), handle already-hired handymen, and detect matching handymen ID.

For loops serve as an iterator to find an ID or other attributes in an ArrayList of instances or specialisation tags of handymen.

While loops keeps the user in each screen and lets the user exit the screen with a specific prompt stated in each screen’s prompting text.

Methods have a purpose to filter lists of handymen instances according to each list’s requirements.

Solution scheme:

The main class carries out the basic application navigations and features, from hiring a handyman to making payments with the handymen.

The OOP elements involved are the interface "Service" as the basic features of the "Handymen" abstract class that implement it. Then there are regular classes "indoorHandymen", "outdoorHandymen", and "hybridHandymen" where each of them inherit from the "Handymen" abstract class. Each class has its own exclusive attributes and their methods are all inherited from the abstract "Handymen" class which are also edited to suit each of them.

The “Handymen” being abstract makes it support polymorphism, since abstract classes make its inheritors be considered or treated as their abstract superclass, while being able to have their own exclusive characteristics.

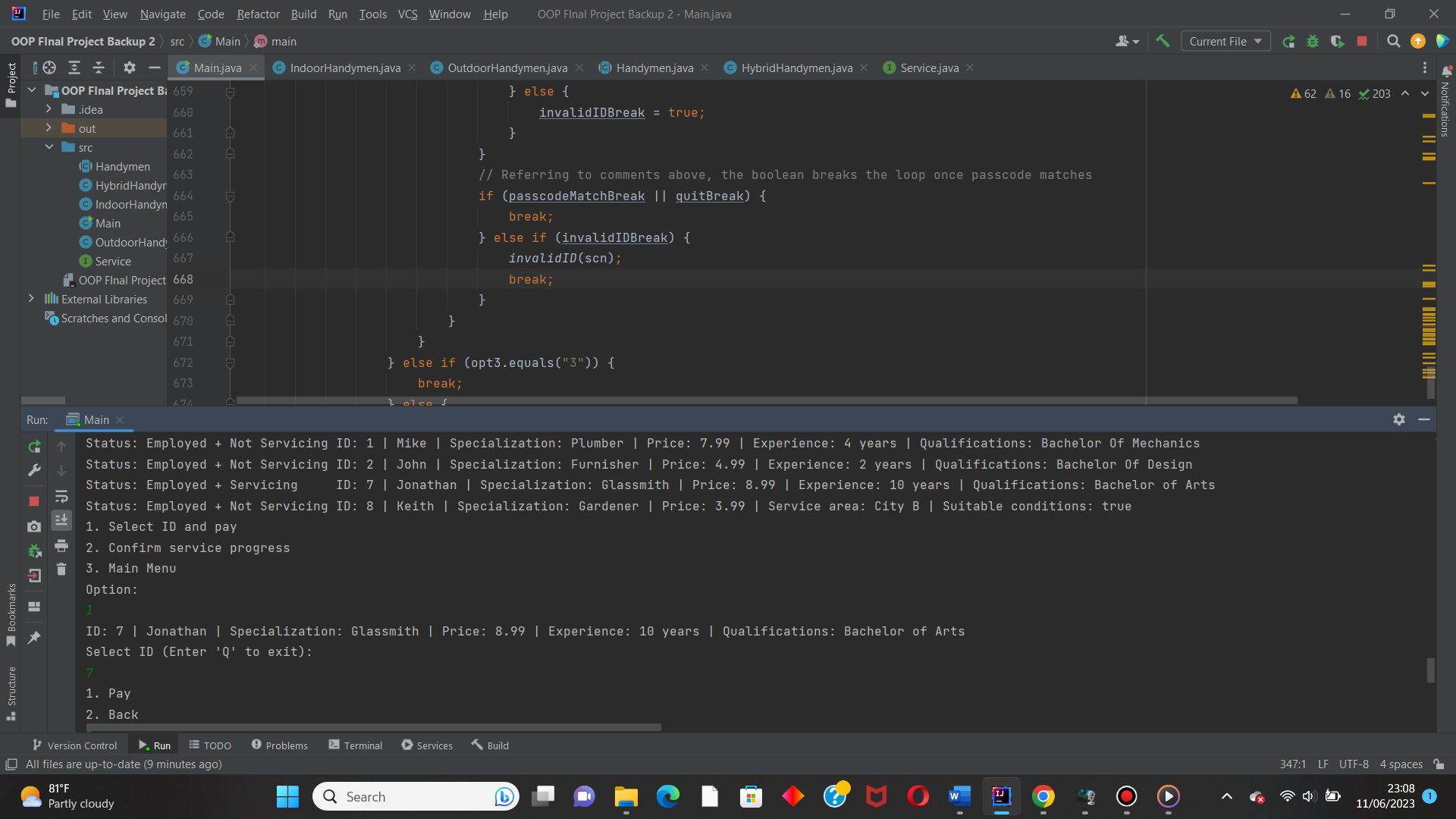
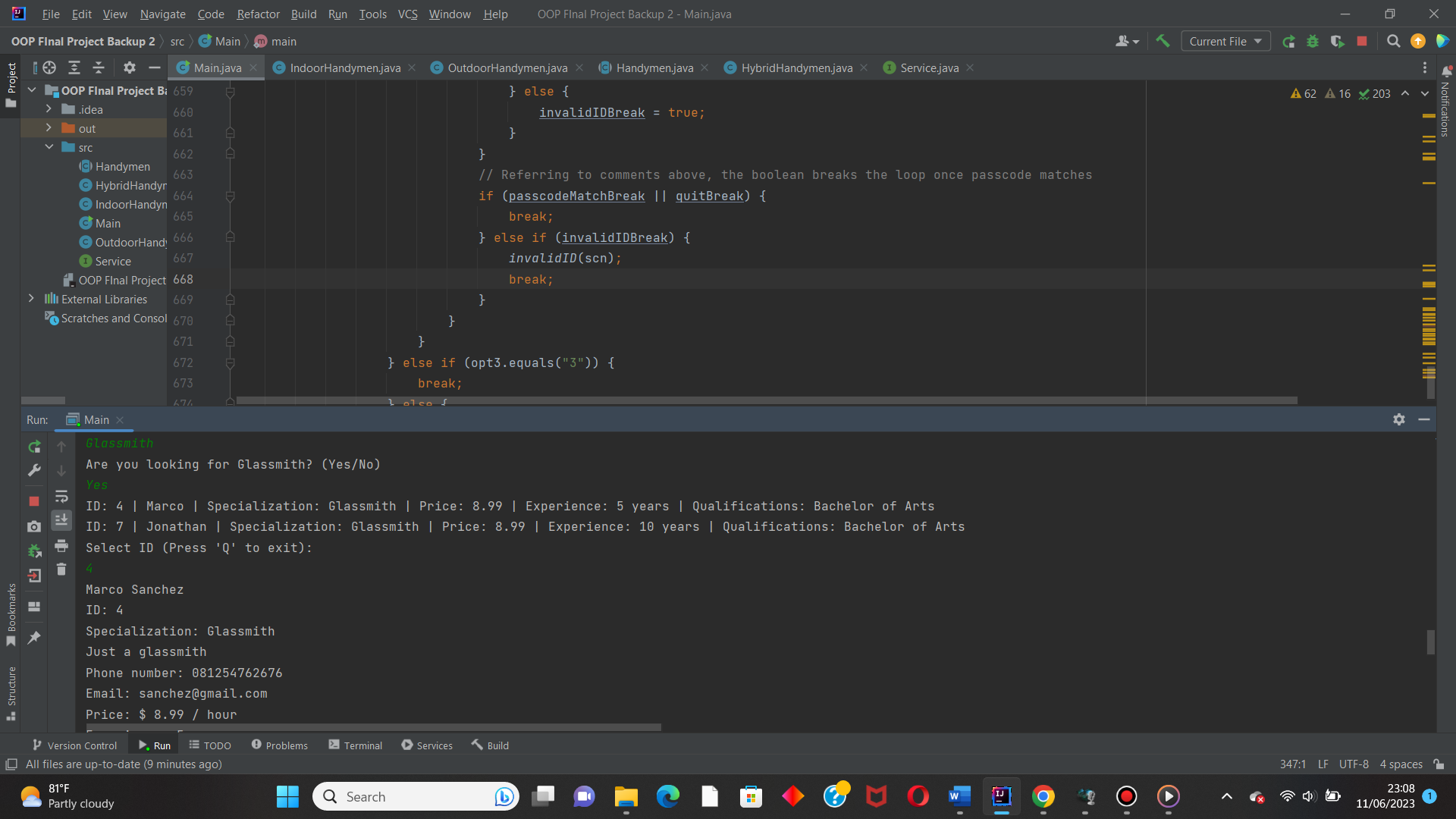
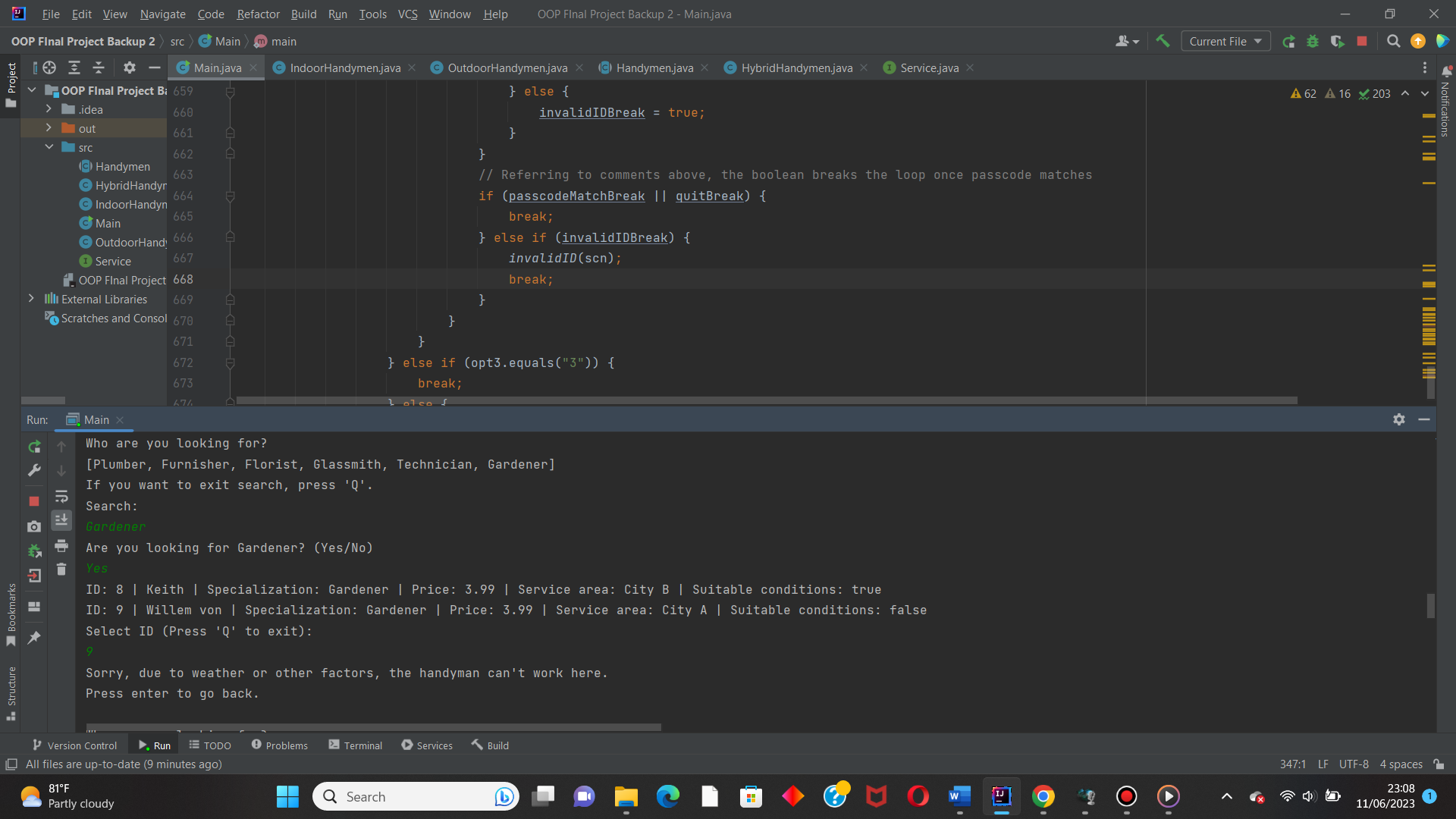
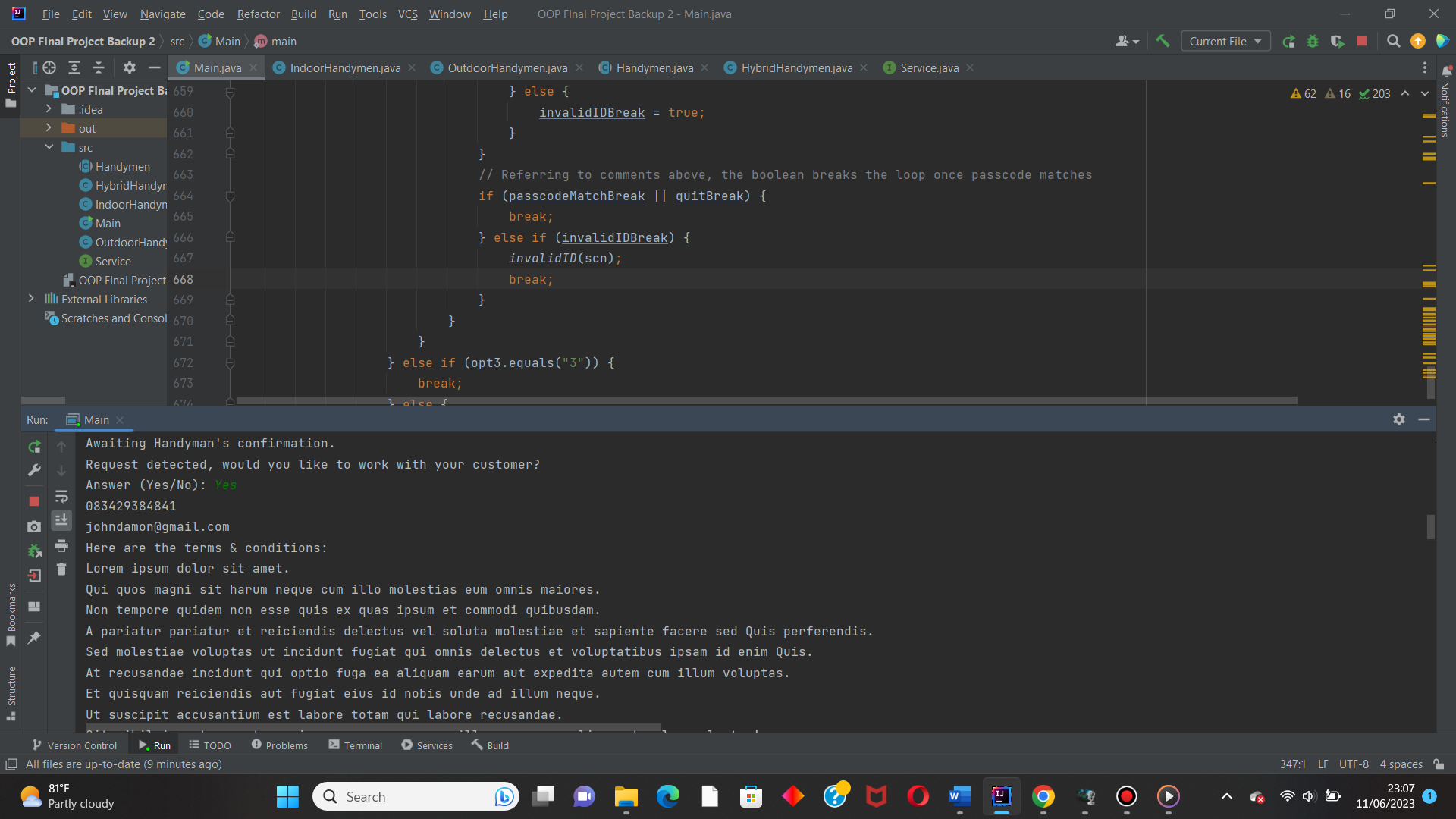
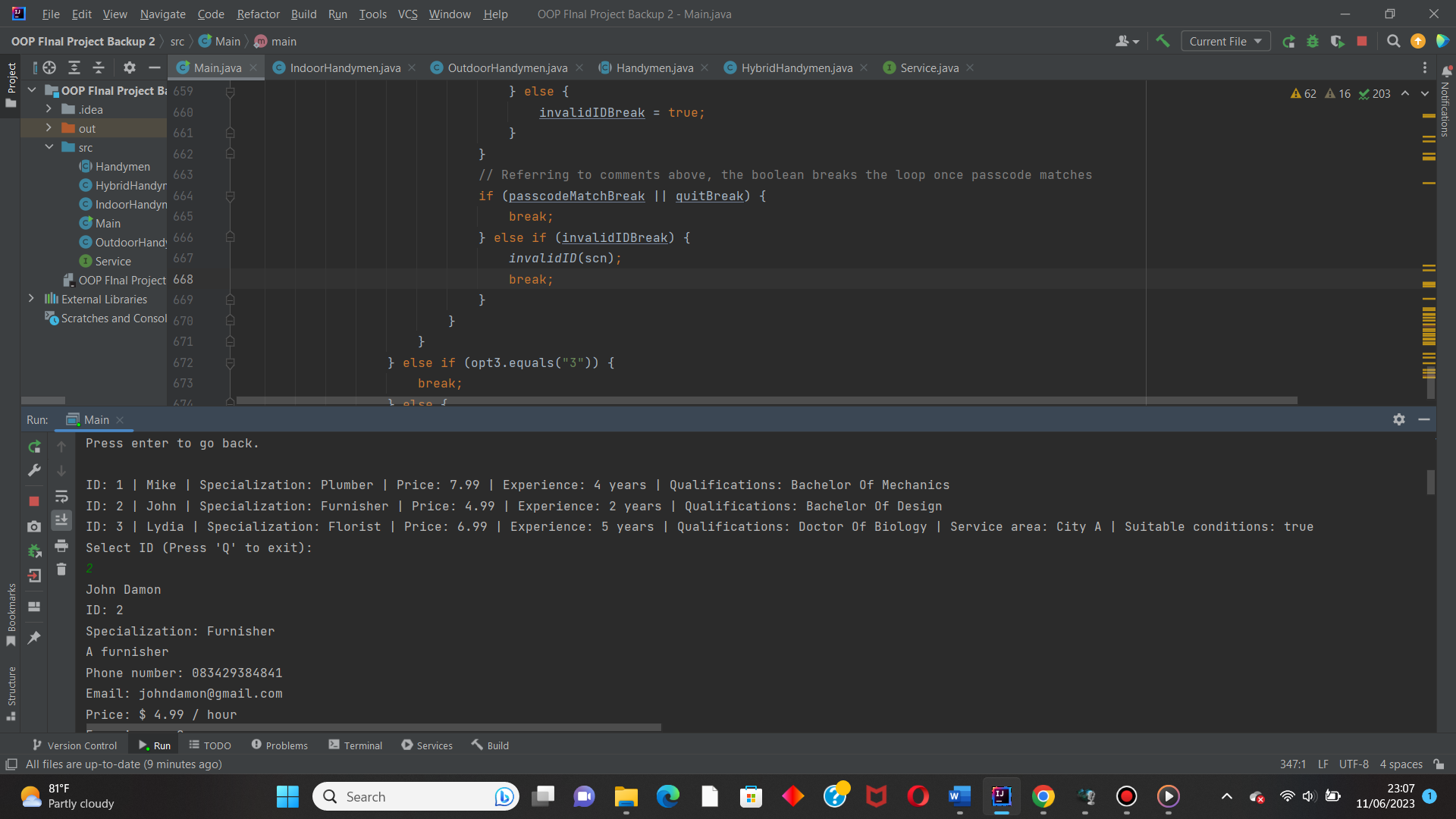
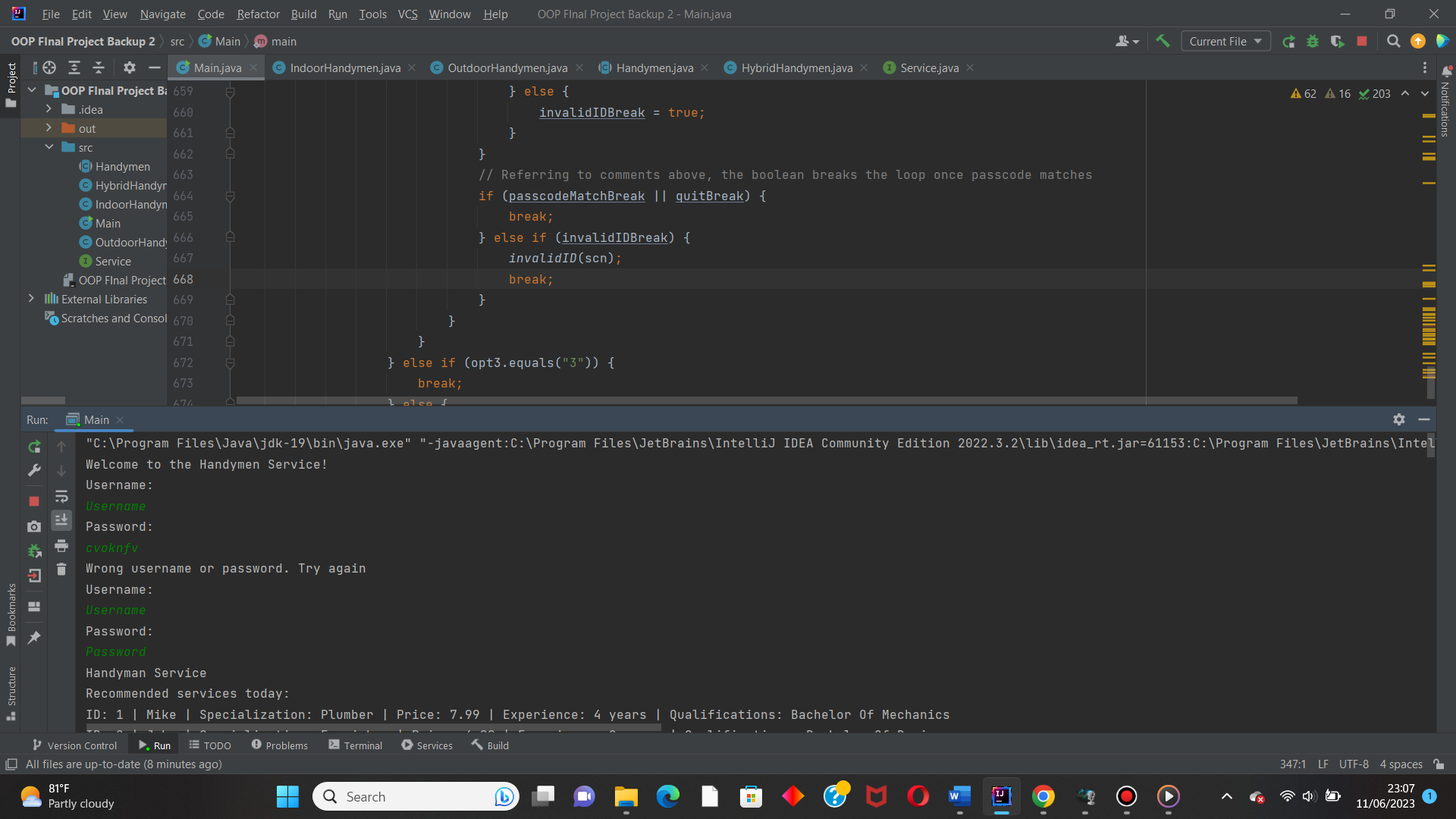
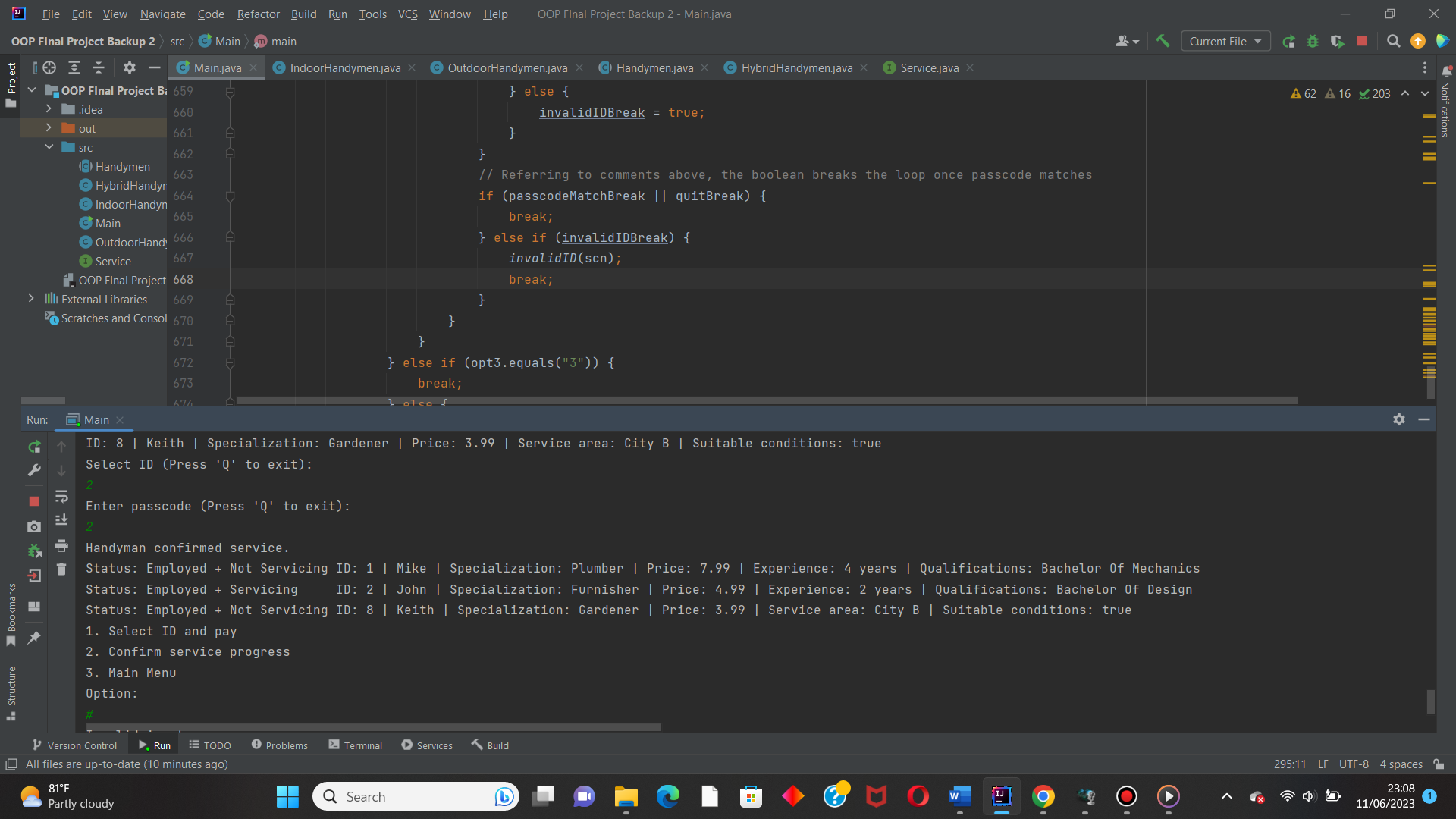
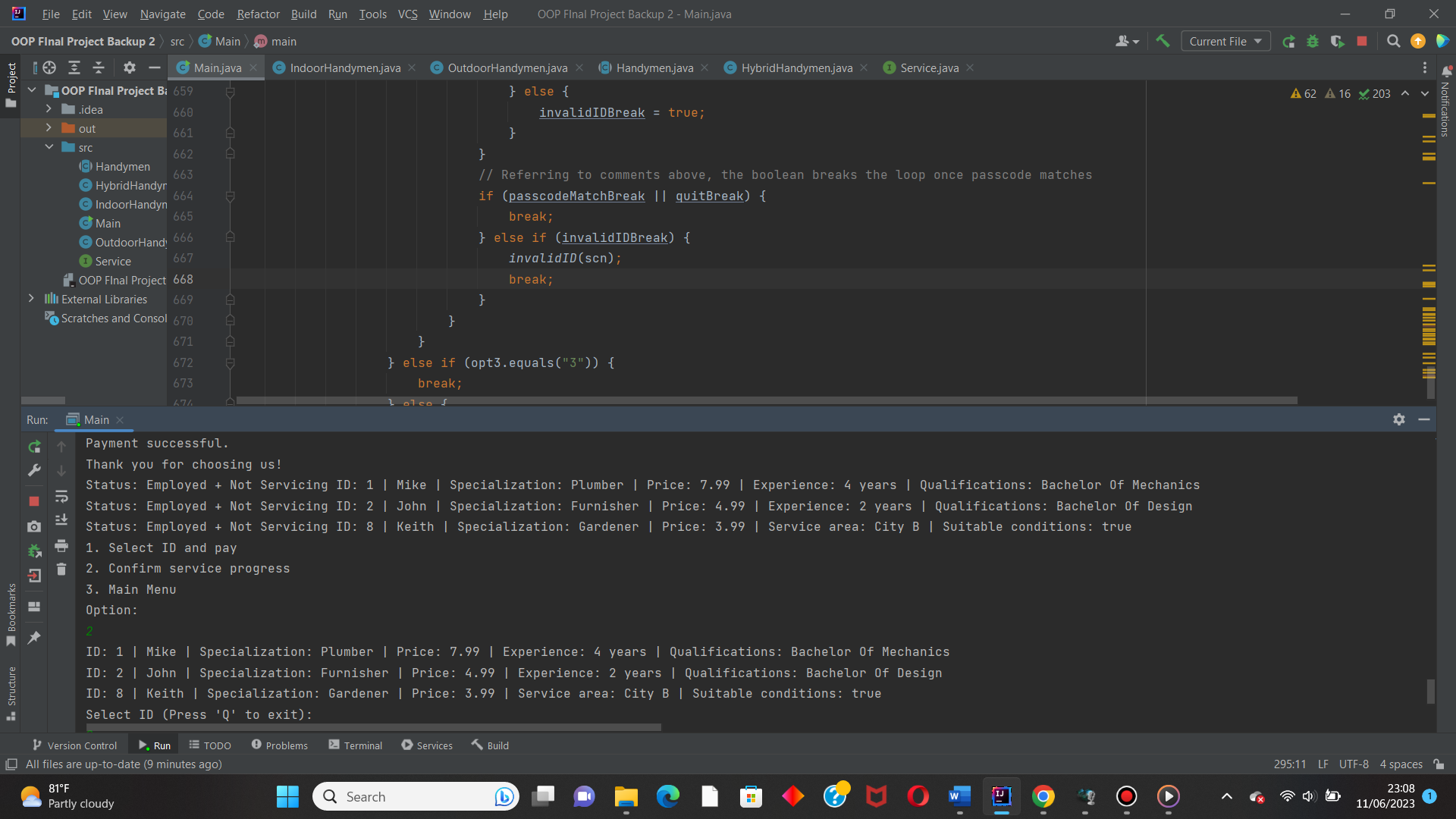
Data structures used:

The only data structures involved are ArrayList arrays. It is used to store instances representing handymen. There are numerous separate arrays for operating specific handymen, such as handymen specialisations, handymen employed, and handymen featured.

The project’s outcome is being an app that can handle handymen hiring processes through computer devices. They are expected to operate the login screen, main menu, featured handymen, search option, viewing appointment process, hiring handymen, and payments. It is also expected to operate the handymen data according to the options chosen.

8

Evidence of Working Program

Here are the screenshots that represent the working program: s\

Conclusion

So far, we can conclude that the handyman hiring app is used to help people to hire handymen within their computer devices with a friendlier interface and features. Surprisingly, it only took 1 type of data structure, which is array lists. The real challenge is in handling OOP related mechanisms and making and correcting the algorithm over and over.Although the project was challenging, it is a valuable experience that I am grateful for. As always, thank you for sparing some time to read this report.