|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Submission Coversheet | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Student ID Number**  *(Do not include student name as anonymous marking is implemented)* | Mateusz Utracki-Janeta | | | | | | |
| **Programme Title** | Bsc Computing Technologies | | | | | | |
| **Module Title** | Object Oriented Programming | | | | | | |
| **Module Code** *(listed on Moodle and in LTAFP)* |  | | | | | | |
| **Module Convenor** |  | | | | | | |
| **Coursework Title** | Mechachrome Registration App | | | | | | |
| **Academic Declaration:**  *Students are reminded that the electronic copy of their essay may be checked, at any point during their degree, with Turnitin or other plagiarism detection software for plagiarised material.* | | | | | | | |
| **Word Count** | 1598 | | | **Date Submitted** | | **05/04/2019** | |
|  |  | | |  | |  | |

Table of Contents

[1. Design and development 3](#_Toc5623087)

[1.1. Person Class 4](#_Toc5623088)

[1.2. Subclasses 6](#_Toc5623089)

[1.2.1. Patient Class 6](#_Toc5623090)

[1.2.2. Doctor Class 8](#_Toc5623091)

[1.3. Appointment Class 11](#_Toc5623092)

[1.4. Armstrong GP Class (Main class) 14](#_Toc5623093)

[2. Test table 27](#_Toc5623094)

[3. References 29](#_Toc5623095)

[4. Appendices 29](#_Toc5623096)

# Design and development

To develop this application I used **Object-Oriented Programming** which is a programming paradigm based on the concept of "**objects**", which can contain data, in the form of **fields** (often known as attributes), and code, in the form of procedures (often known as **methods**).

“Java – one of the world’s most widely used computer programming languages”is an Object-Oriented programming language which is widely spread across the world and very well developed. A key goal of Java is to be able to write programs that will run on a great variety of computer systems and computer-controlled devices. (Deitel, 2015)

Below is a screenshot that shows concept model as a class diagram for Armstrong GP, which is designed in Star UML (Unified Modelling Language). (Banas, 2012)According to Deitel, UML as a graphical scheme for modelling object-oriented systems is one of the most widely used. (Deitel, 2015)

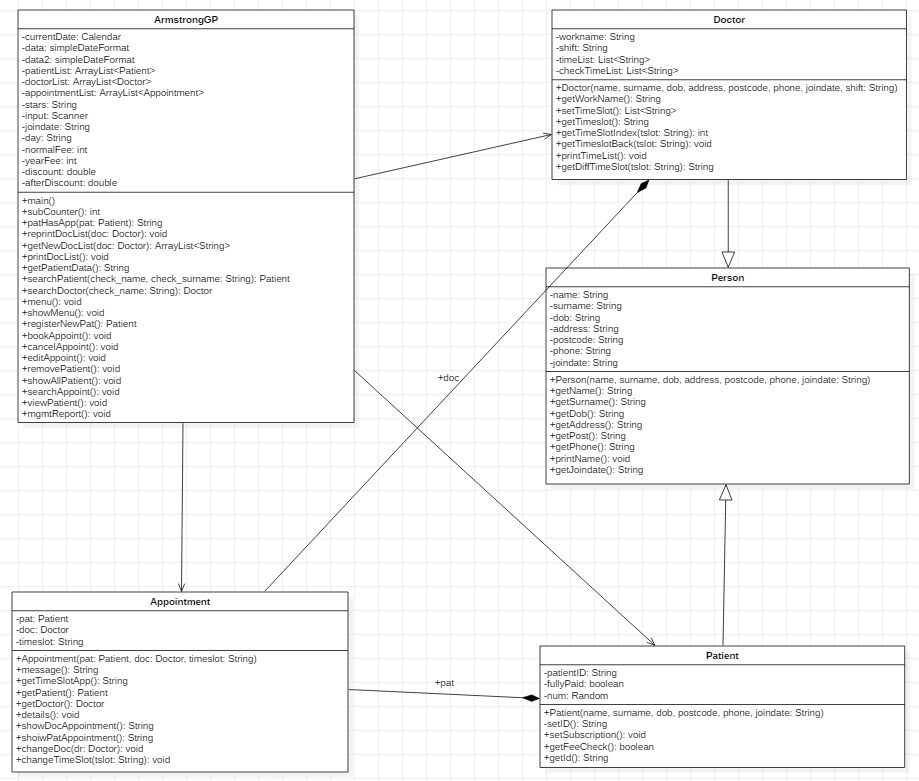


Figure 1

Figure 1 shows five classes. Every class attribiutes and methods You can see that; Patient class and Doctor class are sub-classes or children of Person class which is an abstract class. All classes are Encapsulated. Encapsulation means that variables in one class are hidden from the other classes, it is represented by “-“in class diagram and declared private in the program’s source code. A class’s (and it’s object’s) attributes and methods are intimately related. (Dietel, 2015)

I comment my application with doc comments. This is an industry standard and allows to generate Javadoc page with description of classes and it methods. Zipped folder with Javadoc is attached to this report. (Javadoc, 2004)

Next, I will discuss key features for each class with screenshots of code implementation.

## Person Class

Sometimes when can create classes that we have no intention of making instances for, they are called abstract classes (also super class or parent class). They provide a template for the subclasses, or child classes which are more specific. (Dietel, 2015) Purpose for doing that is that it helps to keep our code DRY (Don’t Repeat Yourself) and the child classes inherit all the common attributes and methods from parent classes. As an example, in Registration App I designed Person class as Patient and Doctor class have common attributes like name, surname and methods for getting name, etc. All in Figure 2.

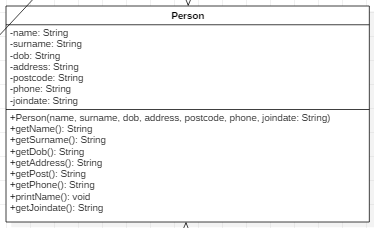


Figure 2

All variables are made private so only subclasses can access them.

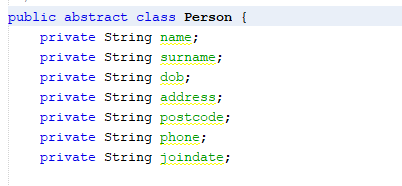


Figure 3

Figure 3 shows the source code of Person class variables; Variables are declared global because one of the requirements were to capture and store patients data(name, surname, address, etc.) For collecting variables a constructor is created.

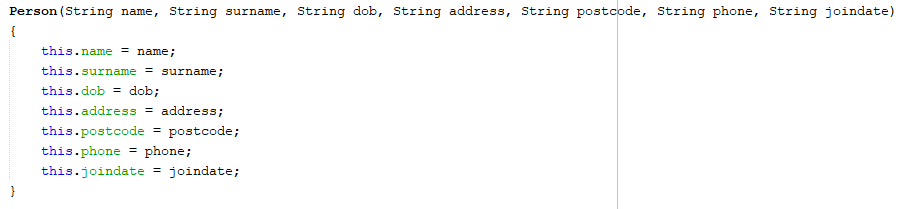


Figure 4

Because of encapsulation of the variables I declared get methods to access the attributes values.

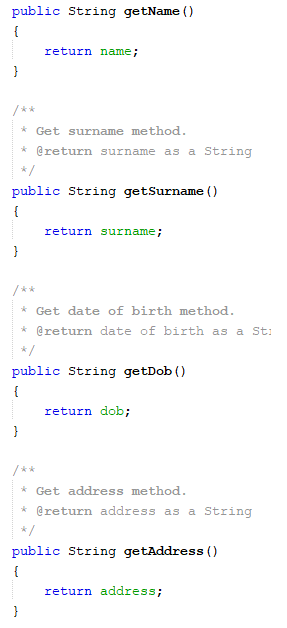
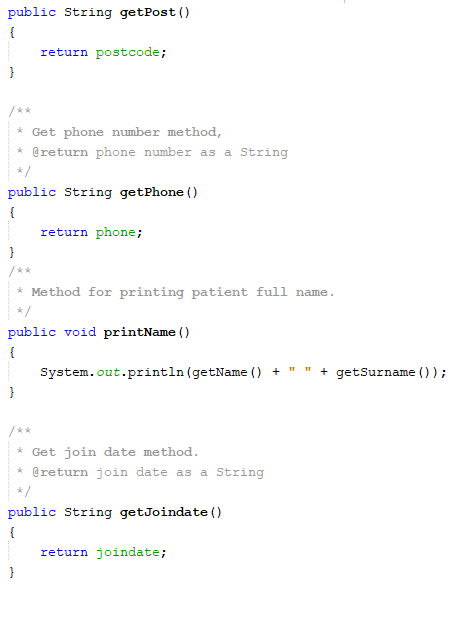
 

Figure 5 Figure 6

## Subclasses

The idea of inheritance helps you keep your code easy to read and DRY. Inheritance in simple but powerful tool that allows you to inherit all the members (fields, methods, and nested classes) of parent class.

With inheritance the polymorphism rule adapts while extending the abstract class, modifying and/or completing the appearance or behaviour of an object given it features. (Oracle, 2017)

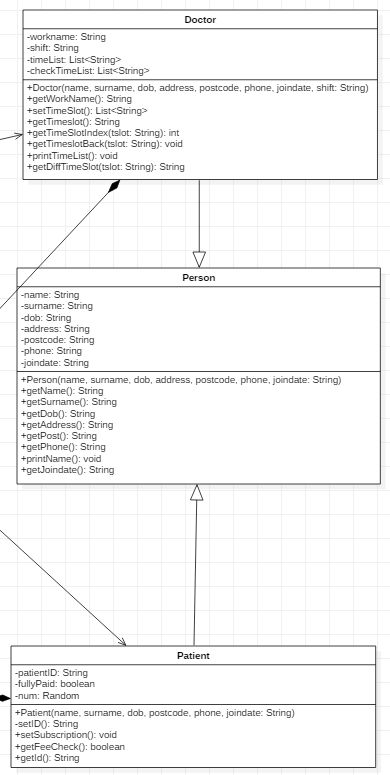


Figure 7

Figure 7 shows inheritance hierarchy where classes Patient and Doctor share parent class common data and some new features are added.

### Patient Class

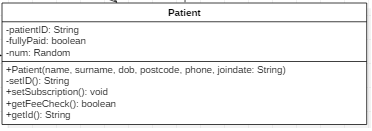


Figure 8

Patient class is a child of Person class it used to create Patient objects. Variable fullyPaid is used to check if patient is a subscriber, patientID is made final because it won’t change once it is made, Random is used to generate random numbers. Constructors cannot be inherited so there is a Patient constructor that shares the parent class variables.

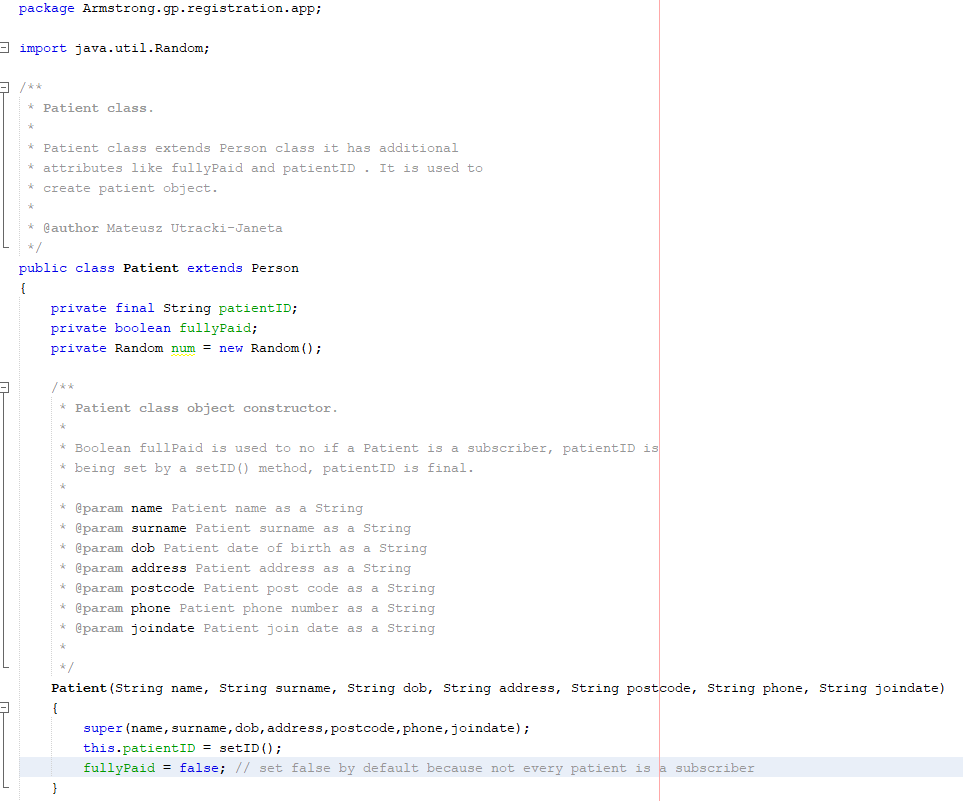


Figure 9

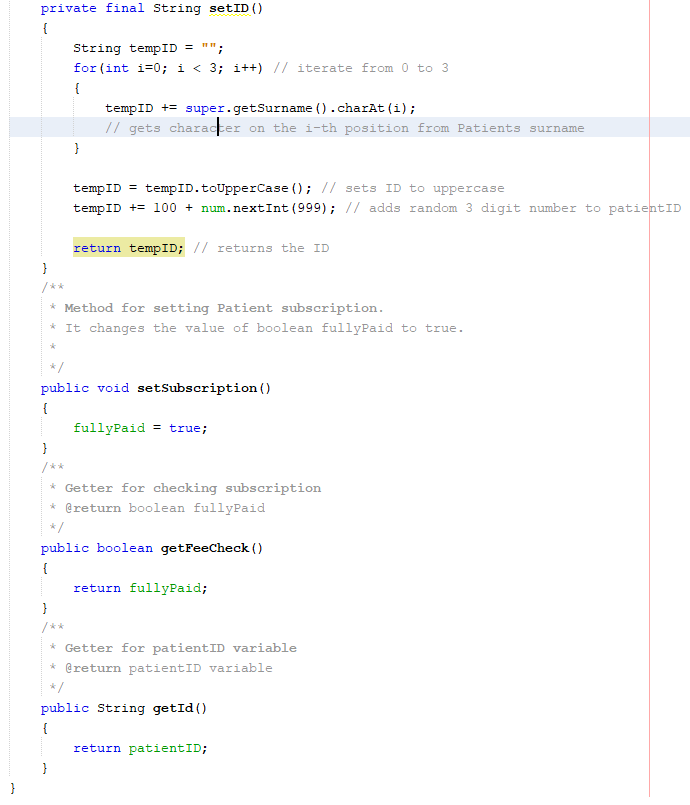


Figure 10

Patient object has additional methods:

1. **setID()** - used to make an unique ID for patient object it uses for loop to iterate throught patient surname and get first three letters, puts them to upper case and adds random number from 100 to 999.
2. **setSubscription()** - used to change variable fullyPaid to true if patient is a subscriber
3. **getFeeCheck()** - used for checking if patient is a subscriber
4. **getId()** - for getting Patient object id.

### Doctor Class

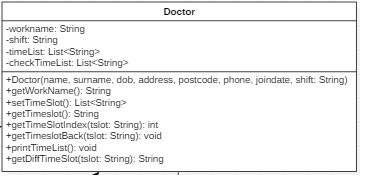


Figure 11

Doctor class is a child class of an abstract Person class. It has additional variables: workname, shift, timeList and checkTimeList. Shift is used to determine if doctor works on AM or PM shift and set appropriate available time slots for Doctor(Figure 11).



Figure 12



Figure 13

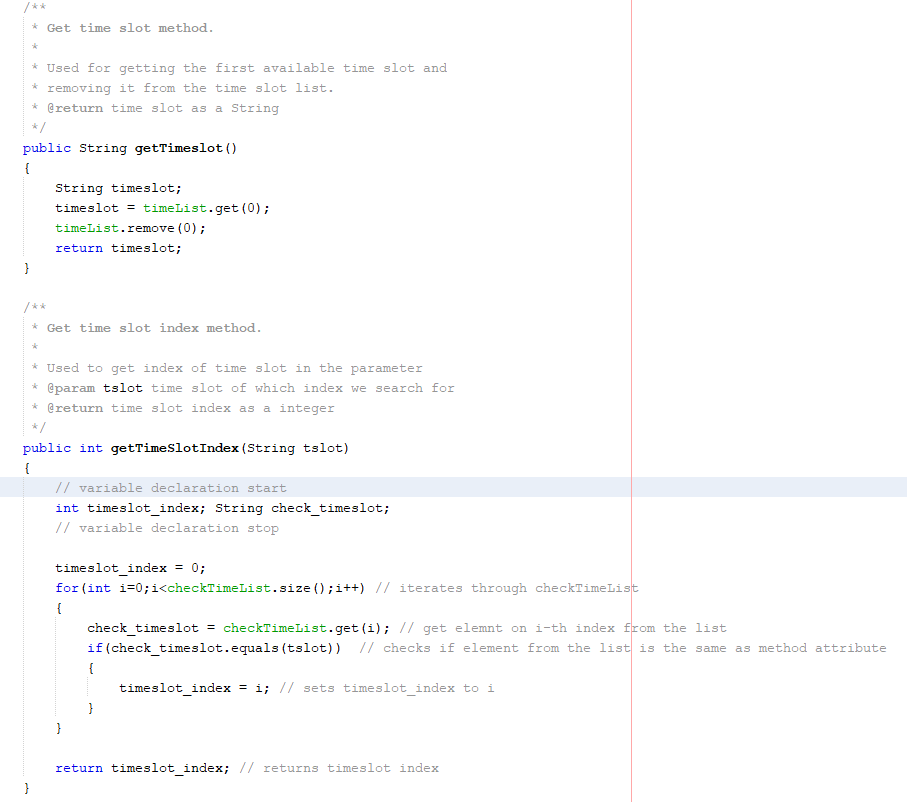


Figure 14



Figure 15



Figure 16 4

Doctor class has additional methods:

1. **getWorkName()** – gets doctor workname
2. **SetTimeSlot()** – sets the time slots according to shift(AM or PM)
3. **getTimeSlot()** – gets the time slot
4. **getTimeSlotIndex()** – gets time slot index
5. **getTimeSlotBack(String tslot)** – returns the time slot in attribute to Doctors time slot list
6. **getDiffTimeSlot(String timeslot)** – get a different time slot than this in attribute from Doctor

## Appointment Class

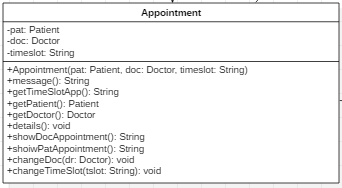


Figure 17

The appointment class is used for storing data about appointments it takes Patient and Doctor class objects as variables so without these two classes existence of this class is pointless. I decided to make Appointment a separate class because in my opinion it is easier to store each appointment and perform any operations on it i.e. changing time slot.

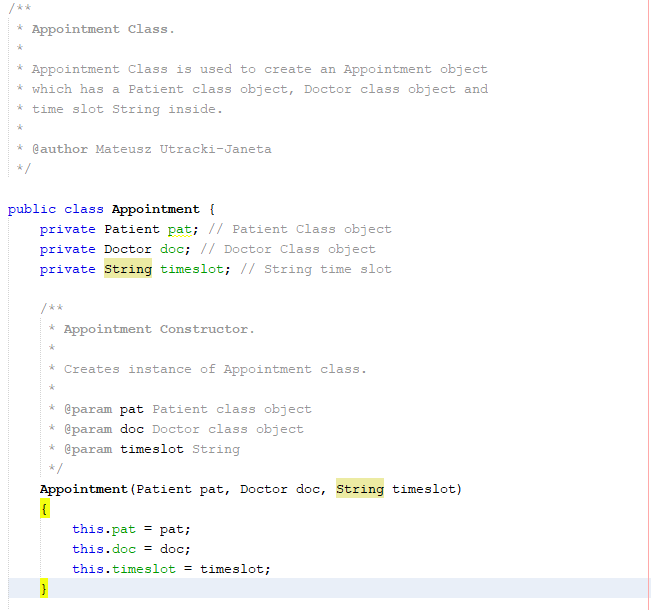


Figure 18

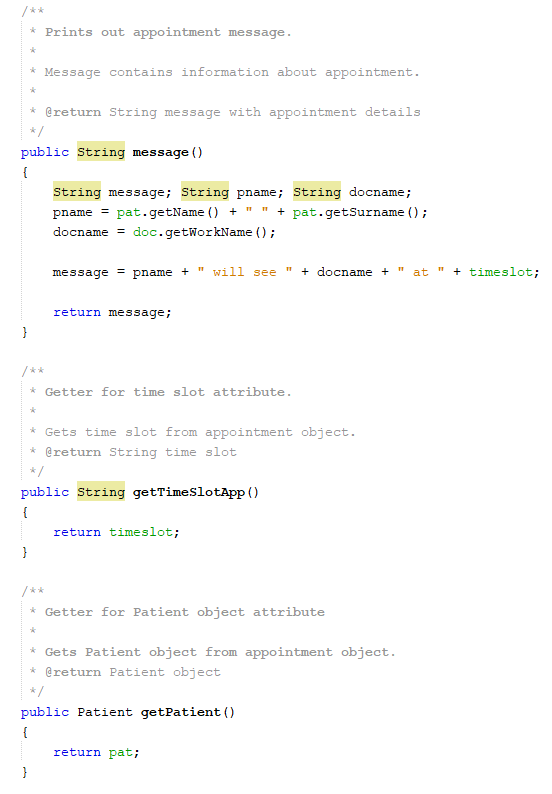


Figure 19

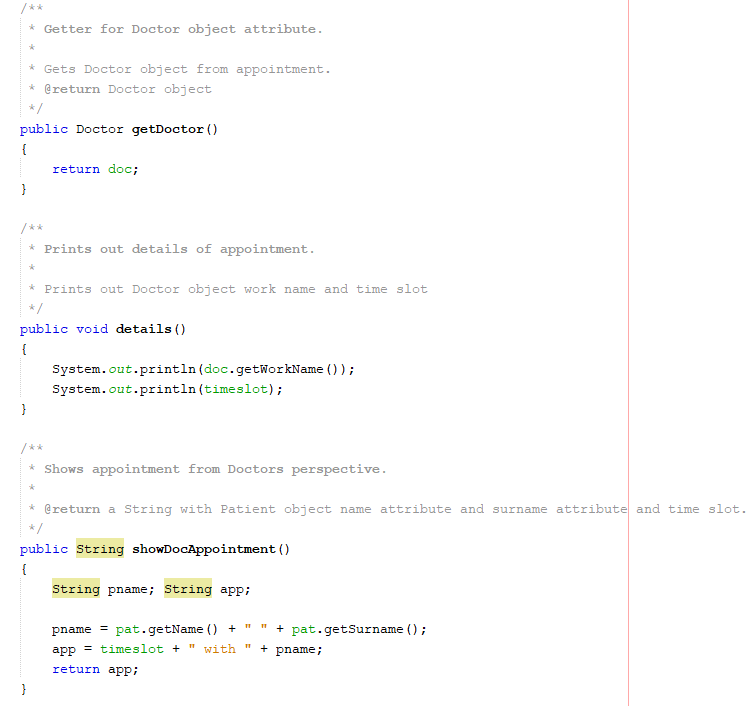


Figure 20



Figure 21

## Armstrong GP Class (Main class)

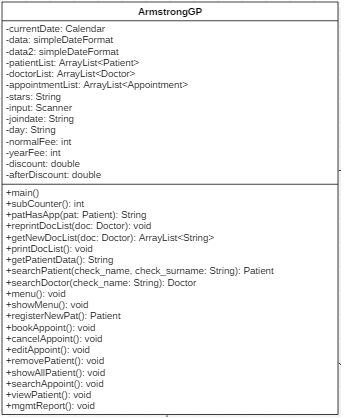


Figure 22

Armstrong GP class is the main class of this application. This class interferes with other classes and here are instances of other classes created. It uses ArrayList to store Patient, Doctor and Appointments objects. I chose ArrayList because Arrays do not automatically change their size at execution time to accommodate additional items.(Deitel, 2015) Which means Array size is determined with its creation and ArrayList can adapt their size. That solution is good because we do not know the finite number of patients, doctors or appointments that will be registered or booked with that GP Clinic.

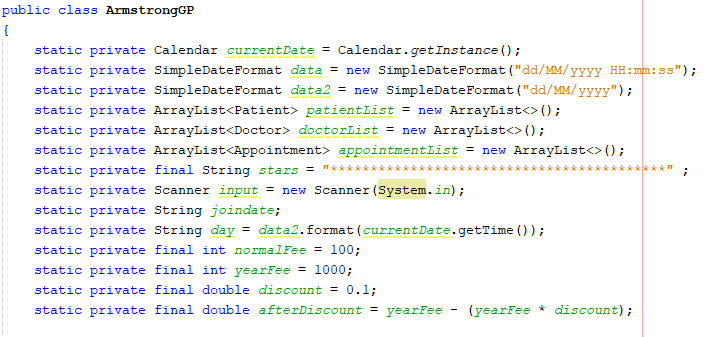


Figure 23



Figure 24

Before I describe the main functionality methods I want to present minor methods that work in the background and keep code DRY. This methods are:

1. **Void subsCounter(**) - used to count the number of subscribers. Goes throught every patient objects and increments counter when fullyPaid is True.
2. **String patHasApp(Patient pat)** – checks if Patient has an appointment returns a time slot of appointment as a String.
3. **void reprintDocList(Doctor doc)**– used to re-print the doctors list without the doctor in method attribute.
4. **ArrayList<Doctor> getNewDocList(Doctor doc)** – returns an ArrayList of Doctor objects without the Doctor in the method attribute.
5. **void printDocList()** – prints all Doctors in a form of an ordered list.
6. **String[] getPatientData()** – gets patient data from receptionist(name and surname) and returns an Array of Strings.
7. **Patient searchPatient(String check\_name, String check\_surname)** – used to search for a Patient with the method attributes (check\_name and check\_surname) – if Patient is on the list it returns Patient object if it isn’t returns null.
8. **Doctor searchDoctor(String check\_name)** – used to search for a Doctor with the method attribute (check\_name) – if Doctor is on the list it returns Doctor object if it isn’t returns null.

This methods do simple tasks as looking for patient or doctor, it saves us time because we do not have to create loops for getting this data, instead we use defined method to do the tasks. This procedure makes code very easy to read. Without these methods main program functionality methods would have much lines of code. They basically iterate through ArrayList to produce different outputs e.g. a list of Doctors without particular doctor, this is useful when we change appointment doctor.



Figure 25 Armstrong GP minor methods1

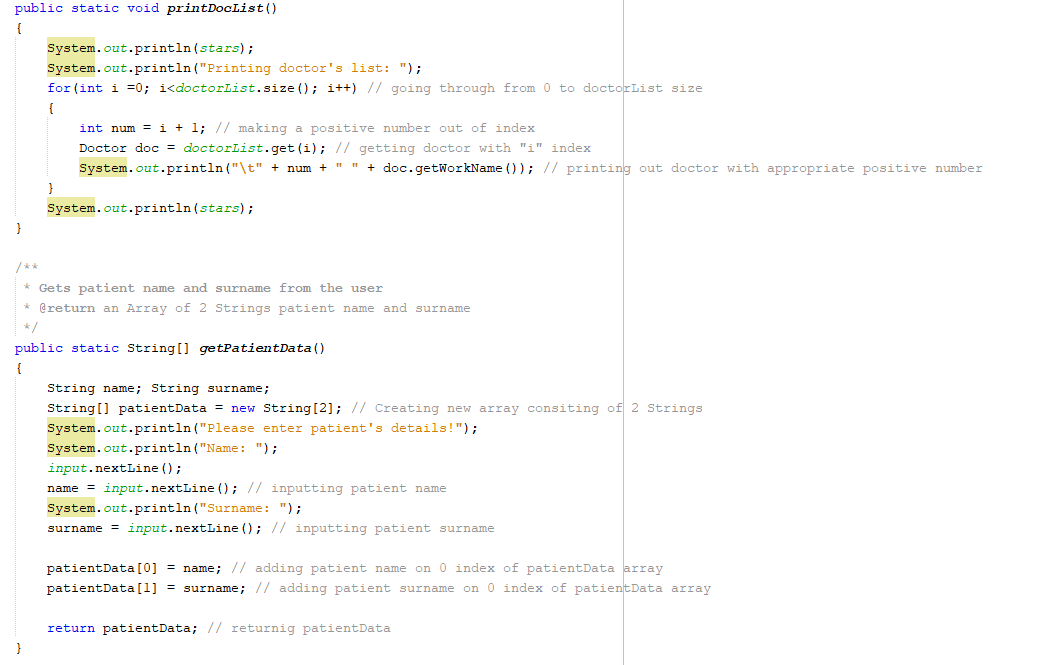


Figure 26 Armstrong GP minor methods2

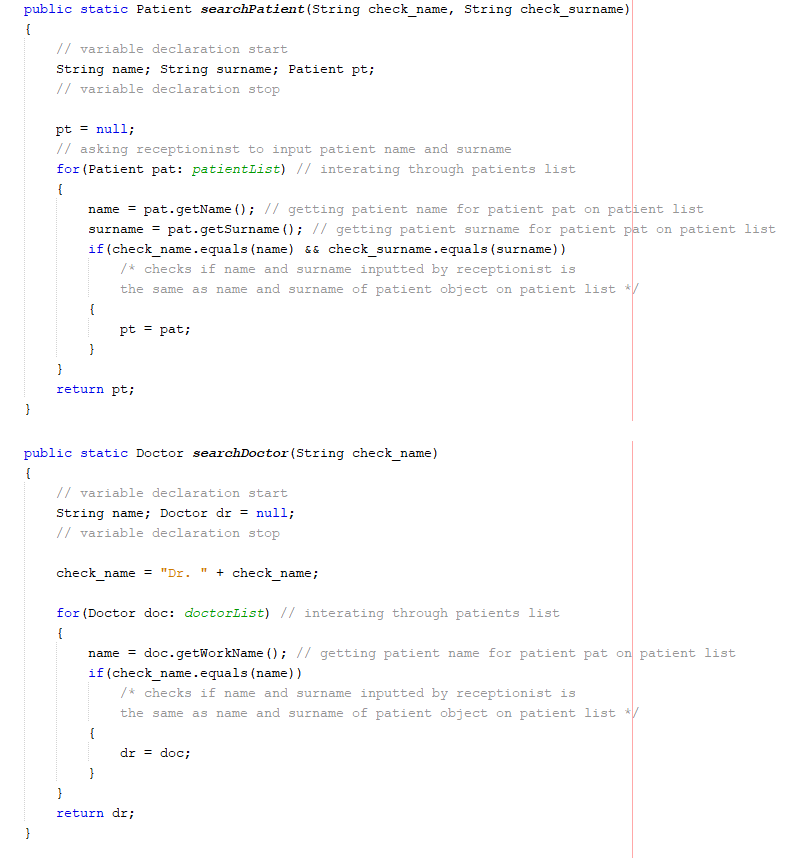


Figure 27 Armstrong GP minor methods3

The methods that appear in the menu are:

1. **Patient registerNewPat()** – this method captures data from receptionist and creates a Patient class instance, ask for subscribtion and sets it when the answer is positive. Returns a Patient object.
2. **void bookAppoint()** – this method is used to book appointment it uses minor methods to get patientData, search for Patient and print doctors list. This methods combined make booking appointment possible.
3. **void cancelAppoint()** – this method is used to cancel appointments.
4. **void editAppoint()** – this method is used to edit the appointment. You can change doctor or time slot.
5. **void removePatient()** – used for removing the Patient object from ArrayList. It iterates through patientList and compares if data captured from receptionist matches the object on the list if it matches it removes the element on the matching index.
6. **void showAllPatients()** – iterates through patients list and prints details for every patient and separates it using stars.
7. **void searchAppoint()** – used to search for appointment. U can search appointments by doctor or patient. For each option there is appropriate switch statements.
8. **void viewPatient()** – used to search for patient. Captures data from receptionist and based on it gets Patient object and print out it details.
9. **void mgmtReport()** – used to show management report where you have 3 options to choose from put in a switch statement.

Now I will present source code of methods and program behaviour.

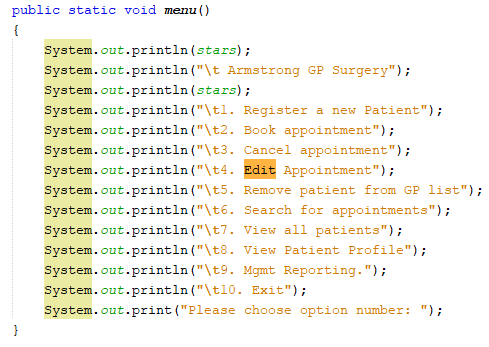


Figure 28

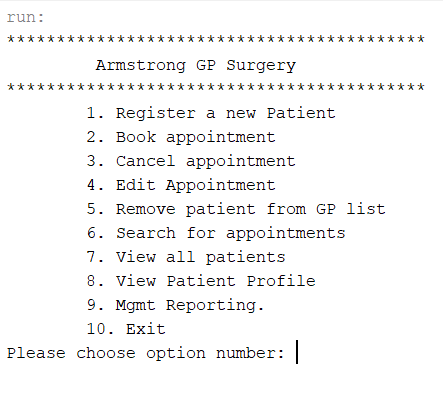


Figure 29 menu() output

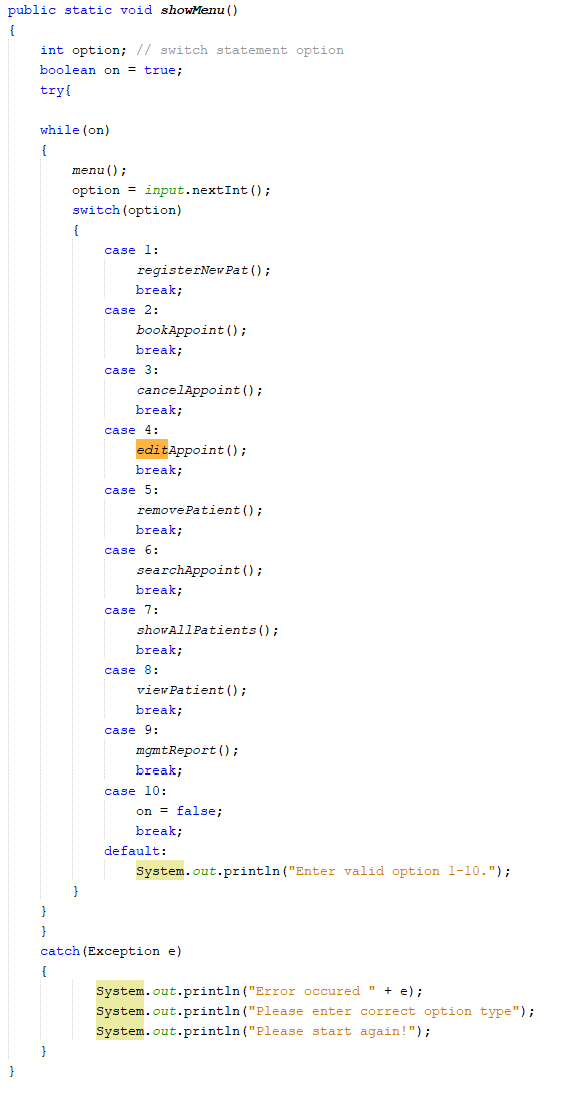


Figure 30



Figure 31 registerNePar() method and output

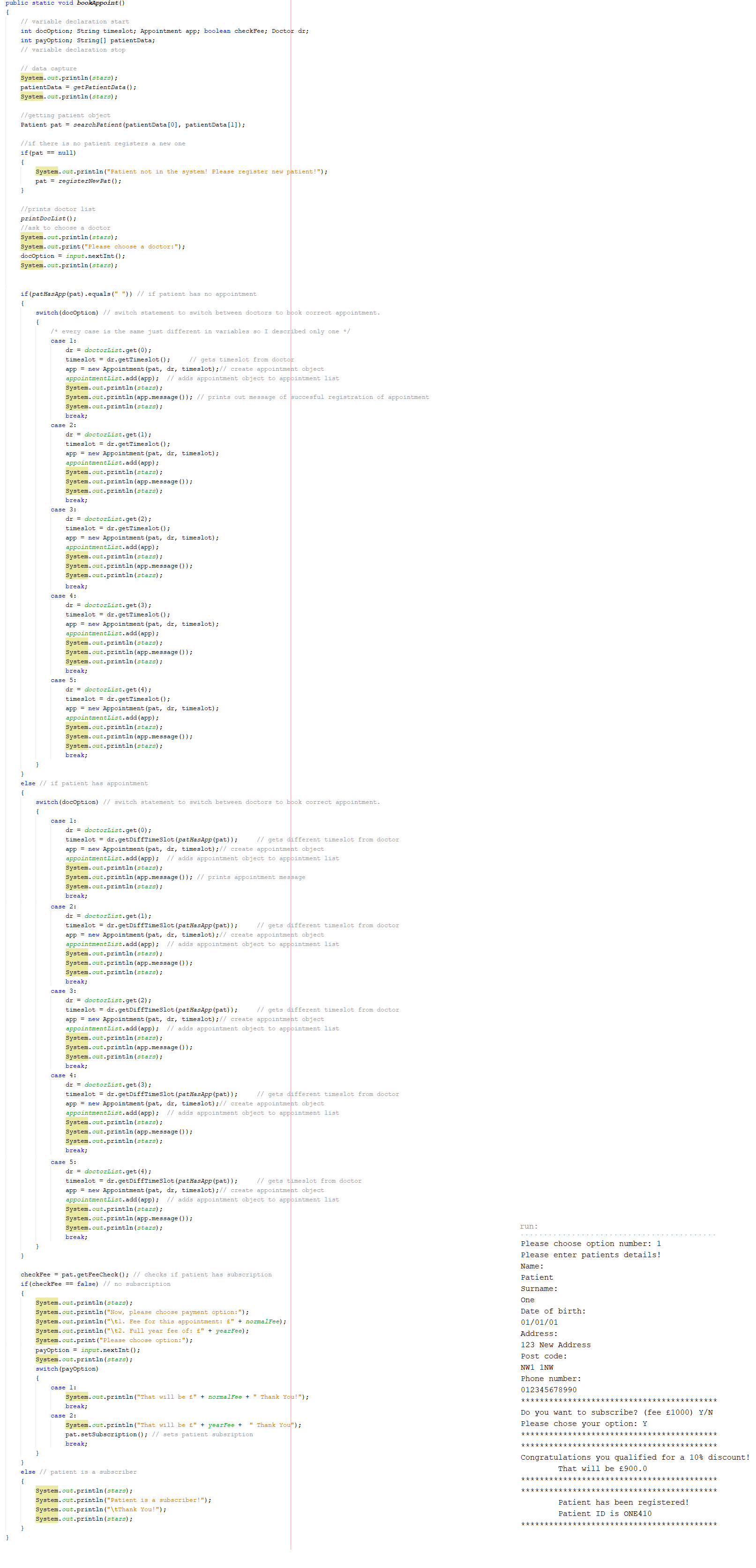


Figure 32 bookAppoiint() method and output



Figure 33 cancelAppoint() method and output

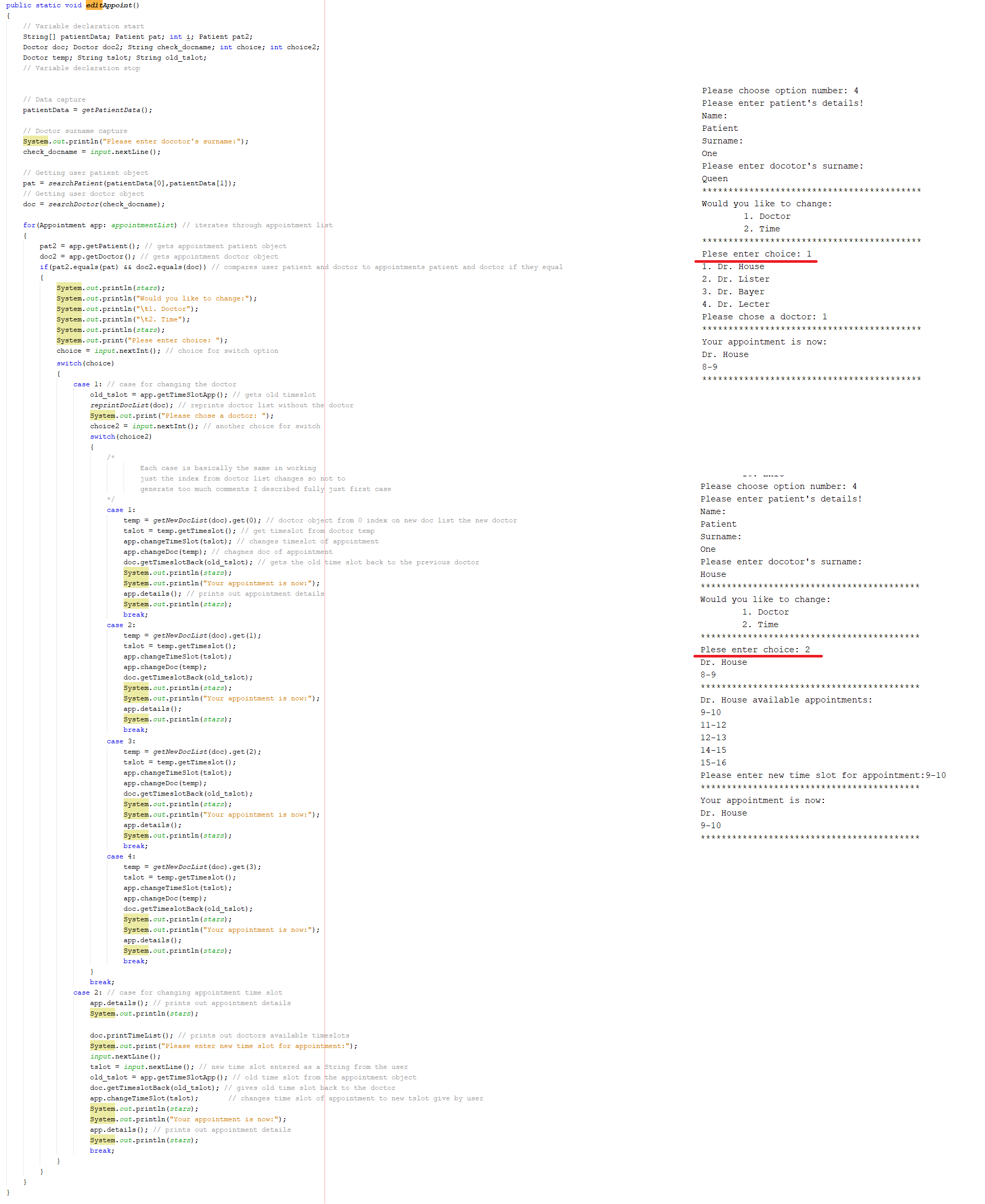


Figure 34 editAppoint() method and output

As you can see editAppoint() produces two outputs based on input from the user.

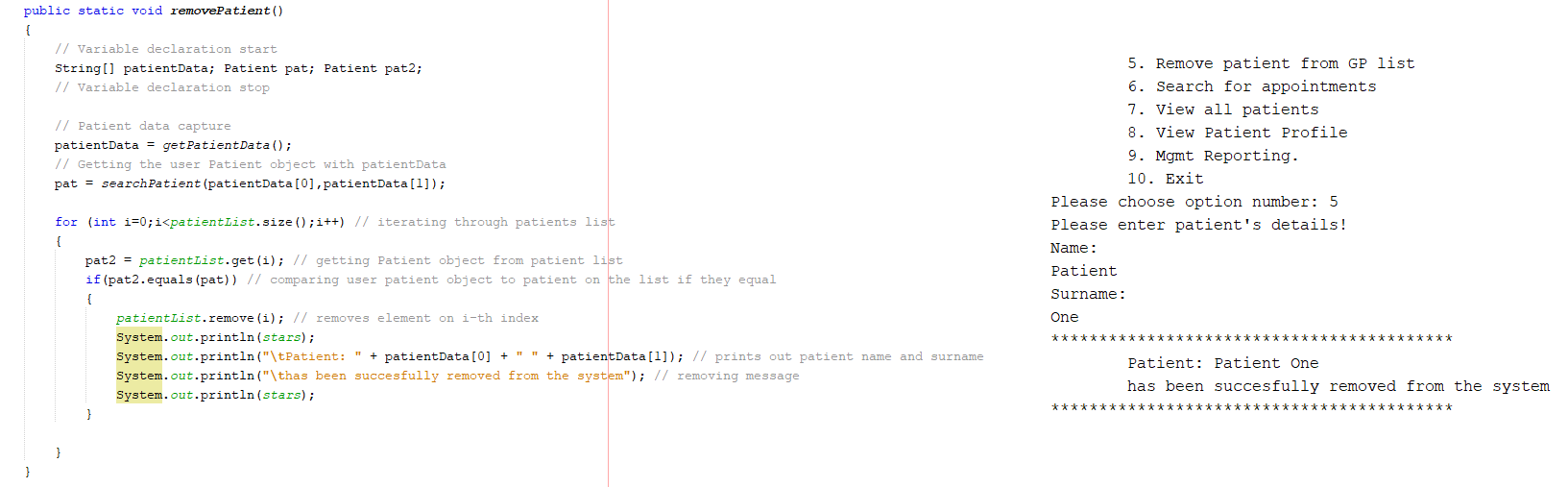


Figure 35 removePatient() method and output



Figure 36 searchAppoint() method and output

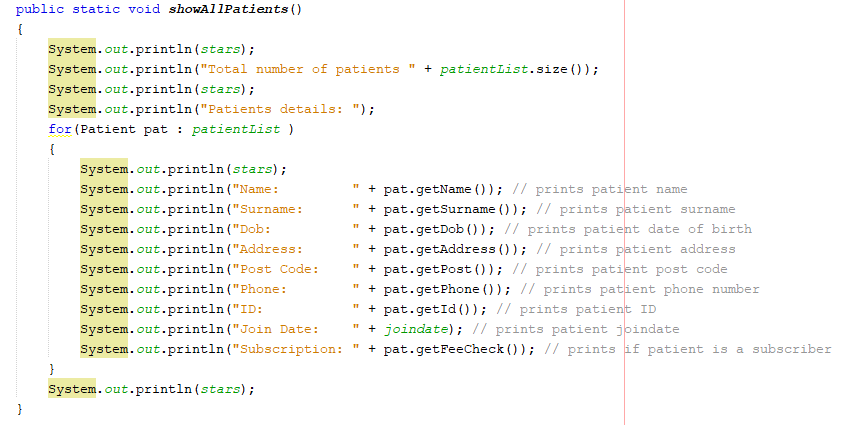


Figure 37 showAllPatients() method

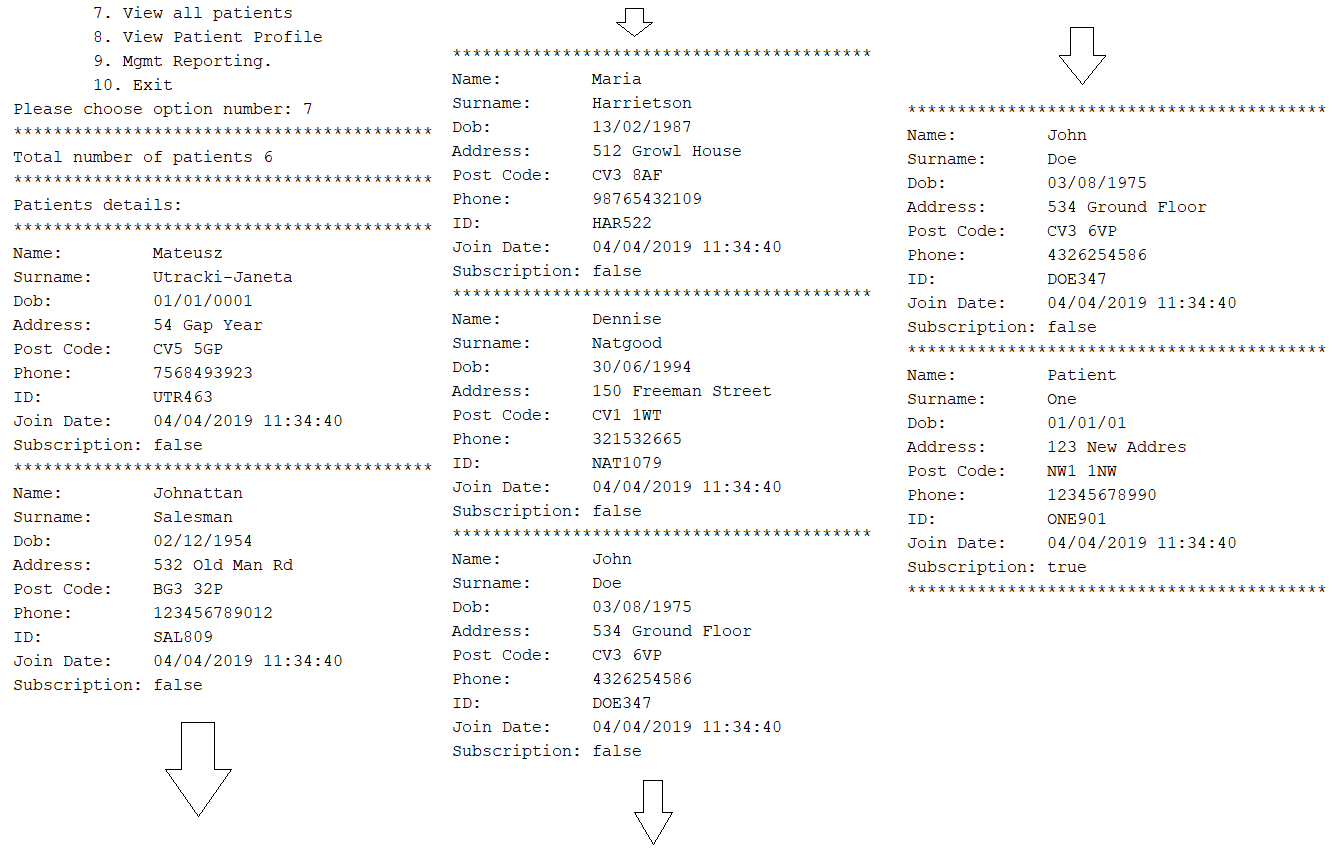


Figure 38 showAllPatients() outpus



Figure 39 viewPatient() method and output



Figure 40 mgmtReport() method and output

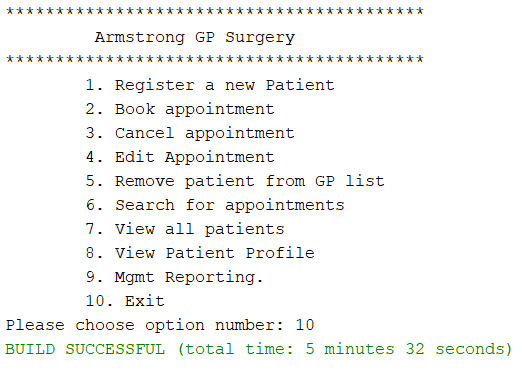


Figure 41 case 10 output

# Test table

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Input | Expected output (on screen) | Result |
| Menu(); | n/a | menu | Successful |
| showMenu(); | 1 | registerNewPat() | Successful |
| 2 | bookAppoint() | Successful |
| 3 | cancelAppoint() | Successful |
| 4 | editAppoint() | Successful |
| 5 | removePatient() | Successful |
| 6 | showAllPatients() | Successful |
| 7 | searchAppoint() | Successful |
| 8 | viewPatient() | Successful |
| 9 | mgmtReport() | Successful |
| 10 | exit | Successful |
| Any number above 10 | Enter valid option 1-10. And prints menu again | Successful |
| Any letter | Error occured java.util.InputMismatchException | Successful |
| viewPatient() | Mateusz, Utracki-Janeta | Prints Patient object with input value data | Successful |
| registerNewPat() | Personal details | Creation of new Patient object | Successful |
| Y | Changes variable fullyPaid to true | Successful |
|  | N | does nothing | Successful |
| bookAppoint() | Name, surname | Get Patient | Successful |
| 1 | Gets doctorList object on 0 index,  Gets first available timeslot,  Creates appointment,  Print success message | Successful |
| 2 | Gets doctorList object on 1 index,  Gets first available timeslot,  Creates appointment,  Print success message | Successful |
| 3 | Gets doctorList object on 2 index,  Gets first available timeslot,  Creates appointment,  Print success message | Successful |
| 4 | Gets doctorList object on 3 index,  Gets first available timeslot,  Creates appointment,  Print success message | Successful |
| 5 | Gets doctorList object on 4 index,  Gets first available timeslot,  Creates appointment,  Print success message | Successful |
| 1 | Ask for paining fee for appointment | Successful |
| 2 | Asks for paying full year fee and set fullyPaid to true | Successful |
| cancelAppoint() | Patient Name, Surname | Gets patient from patientList | Successful |
| Doctor surname | Gets doctor from doctoList | Successful |
| Doctor, patient | Searches for appointment and removes appropriate index | Successful |
| removePatient() | Patient Name, Surname | Gets patient from patientList  And deletes the index of Patient which results in deleting patient object | Successful |
| showAllPatients() | N/a | Prints all patients details separated with stars | Successful |

# References

(Deitel, 2015) - Deitel P. and Deitel H.(2015) *Java How to Program Early Objects.* (10th Edition) Essex: Pearson.

(Banas, 2012) - *UML 2.0 Class Diagrams* [YouTube]. Available at: <https://youtu.be/3cmzqZzwNDM> (Accessed: 04/04/2019).

(Javadoc, 2004) Laurie Williams, Dright Ho, and Sarah Heckman. – *Javadoc tutorial.* Avaialable at:

[https://web.archive.org/web/20170613233020/http://agile.csc.ncsu.edu/SEMaterials/tutorials/javadoc/](https://web.archive.org/web/20170613233020/http:/agile.csc.ncsu.edu/SEMaterials/tutorials/javadoc/)

(Accessed: 04/04/2019)

(Oracle, 2017) - *The Java Tutorials*. Available at: <https://docs.oracle.com/javase/tutorial/java/> (Accessed: 04/04/2019).

# Appendices

