Component 3 – Assessment Support Sheet

CMP4266 – Computer Programming

2022

Table of contents

Basic design of the Hospital Management System **3**
The Patient Class **4**
The Doctor class **6**
The Admin class **8**
Main.py **12**
Examples to start **15**
Possible extensions for extra marks – Inheritance **16**

List of tables

Table 1- Methods of the Patient Class 5
Table 2- Methods of the Doctor Class 7
Table 3- Methods of the Admin Class 8
Table 4- Main Function 14

Basic design of the Hospital Management System

According to the assessment brief for this module (CMP4266 Computer Programming), the students need to design, develop and test a software that implements the basic functionalities of a hospital management system. A partial implementation will be available under the Assessment Section on Moodle.

The released partial implementation consists of the following classes: Patient, Admin and Doctor and the main file. The following sections will briefly discuss each class by explaining it's important attributes and the class role in the system.

The partial implementation provided in the Moodle has some place markers **(#ToDo pass)** in where you need to add code to proceed with the implementation of the Hospital Management System.

The Patient Class

```
class Patient:
    """Patient class"""

    def __init__(self, first_name, surname, age, mobile, postcode):
        """
        Args:
            first_name (string): First name
            surname (string): Surname
            age (int): Age
            mobile (string): the mobile number
            address (string): address
        """

        #ToDo1
        pass
        self.__doctor = 'None'



    def full_name(self) :
        """full name is first_name and surname"""
        #ToDo2
        pass


    def get_doctor(self) :
        #ToDo3
        pass

    def link(self, doctor):
        """Args: doctor(string): the doctor full name"""
        self.__doctor = doctor

    def print_symptoms(self):
        """prints all the symptoms"""
        #ToDo4
        pass

    def __str__(self):
        return
f'{self.full_name():^30}|{self.__doctor:^30}|{self.__age:^5}|{self.__mobile:^15}|{self
.__postcode:^10}'
```

The Patient class is a template used to create patient instances. It includes attributes such as patients' first and last names, age, address, symptoms, mobile etc. There are different methods in this class as follows:

Table 1- Methods of the Patient Class

Methods	Parameters	What should it do?	ToDo
__init__	first_name, surname, age, mobile, postcode	Initializes the attributes of the class	#ToDo1
full_name	self	Returns full name (fist name and surname of a patient)	#ToDo2
get_doctor	self	Returns the doctor of the patient	#ToDo3
link	self, doctor	Set the doctor of the patient	
print_symptoms	self	Prints all the symptoms	#ToDo4
__str__	self	Returns the string representation of the object	

The Doctor class

The Doctor class should contain the common attributes to store common information about the doctors such as first name, last name, speciality, list of patients and list of appointments.

```
class Doctor:
    """A class that deals with the Doctor operations"""

    def __init__(self, first_name, surname, speciality):
        """
        Args:
            first_name (string): First name
            surname (string): Surname
            speciality (string): Doctor`s speciality
        """

        self.__first_name = first_name
        self.__surname = surname
        self.__speciality = speciality
        self.__patients = []
        self.__appointments = []


    def full_name(self) :
        #ToDo1
        pass

    def get_first_name(self) :
        #ToDo2
        pass

    def set_first_name(self, new_first_name):
        #ToDo3
        pass

    def get_surname(self) :
        #ToDo4
        pass

    def set_surname(self, new_surname):
        #ToDo5
        pass

    def get_speciality(self) :
        #ToDo6
        pass

    def set_speciality(self, new_speciality):
        #ToDo7
        pass
```

```
def add_patient(self, patient):
    self.patients.append(patient)


def __str__(self) :
    return f'{self.full_name():^30}|{self.__speciality:^15}'
```

Looking at the `__init__` method of the Doctor class, it is clear that a doctor object cannot be created without the parameters, first name, surname, and speciality. This class contains the following methods:

Table 2- Methods of the Doctor Class

Methods	Parameters	What should it do?	ToDo
__init__	self, first_name, surname, speciality	Initializes the attributes of the class	
full_name	self	Returns full name (fist name and surname of a doctor)	#ToDo1
get_first_name	self	Returns the name of a doctor	#ToDo2
set_first_name	self, new_first_name	Updates the name of a doctor	#ToDo3
get_surname	self	Returns the surname of a doctor	#ToDo4
set_surname	self, new_surname:	Updates the surname of a doctors	#ToDo5
get_speciality	self	Returns the speciality of a doctor	#ToDo6
set_speciality	self, new_speciality	Updates the speciality of a doctor	#ToDo7
add_patient	self,patient	Add a new patient to the patients list	
__str__	self	Returns the string representation of the object	

The Admin class

The Admin class is the main class that handles most of the core functionalities of the system. It is the container that holds and deals with Admin, and this is why the class imports the Doctor class as depicted in the beginning of the source code of the Admin class as shown below.

Use the table below and the assessment brief to write the methods in the table below so that they do as shown in the following table.

```
from Doctor import Doctor
```

Looking at the `__init__` method of the Admin class, it is clear that an admin object cannot be created without the parameters, username, password, and address. The class contains the following methods:

Table 3- Methods of the Admin Class

Methods	Parameters	What does it do?	ToDo
__init__	self, username, password, address = ''	Initializes the attributes of the class	
view	self,a_list	Print the element of a list	
login	self	A method that deals with the Admin login, check if the username and password match the registered ones.	#ToDo1
find_index	self,index, doctors	Check if a doctor id exists. *Hint: returns True if the doctor ID exists and False if the doctor ID does not exist*	
get_doctor_details	self	Get the details of a doctor, first name, surname and the speciality and returns them. *Hint: The user will be asked to provide the first name, surname and the speciality of the doctor*	#ToDo2

doctor_management	self, doctors	To do this, you will need to write a large if statement that calls the different methods from the Admin and Doctor classes. You did something similar in the contacts exercise for Lab 6. Therefore, please refer to your submission in that Lab.
		
		A method that deals with registering, viewing, updating, deleting doctors
		
		`-----Doctor Management-----`
		`Choose the operation:`
		`1 - Register`
		`2 - View`
		`3 - Update`
		`4 - Delete`
		`Input:`
			#ToDo3

REGISTER

#ToDo4
#ToDo5
#ToDo6

If the user enters 1, details about the doctor will be asked (first name,surname and speciality).

```
-----Register-----
Enter the doctor's details:
Enter the first name: Adela
Enter the surname: Duka
Enter the speciality: Pediatrics
Doctor registered.
```

The program will check for the name. If the name provided does not exist in the doctor list, the doctor will be added to the doctor list and a message will be printed "Doctor registered".
If the name provided exist the doctor will not be added and a message will be printed "Name already exists".

VIEW

If the user enters 2, a list of doctors will be printed.

#ToDo7

```
Input: 2
-----List of Doctors-----
ID |          Full name          |  Speciality
  1|          John Smith         | Internal Med.
  2|          Jone Smith         |  Pediatrics
  3|         Jone Carlos         |  Cardiology
  4|          Adela Duka         |  Pediatrics
```

You can use the view method to print the list.

UPDATE

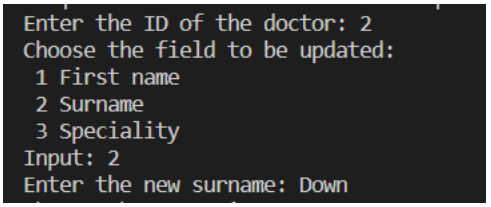
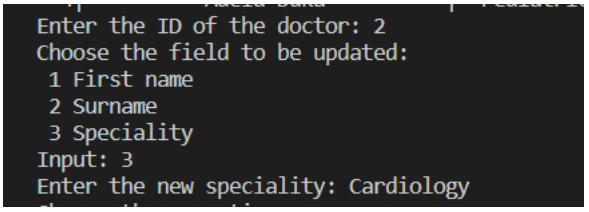
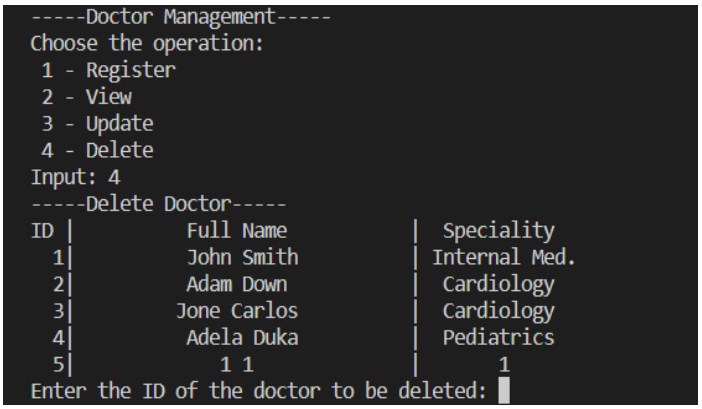
#ToDo8

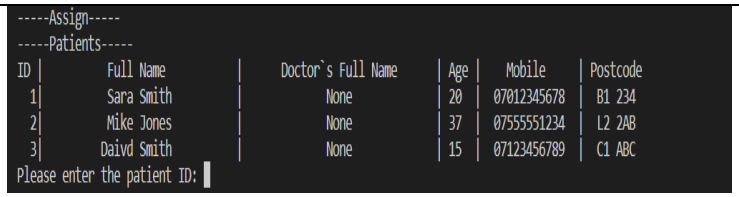
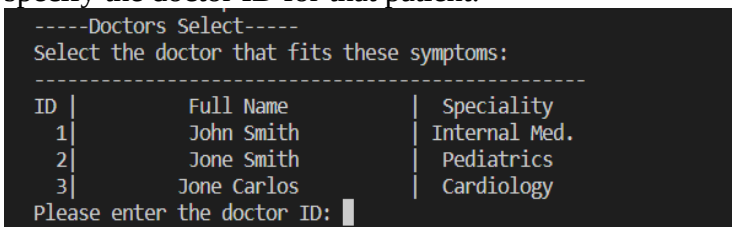
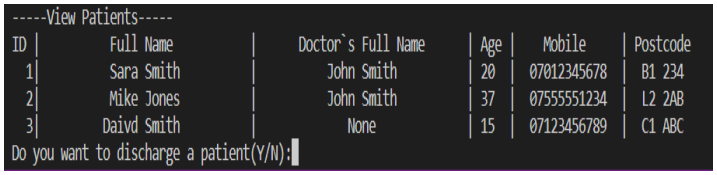
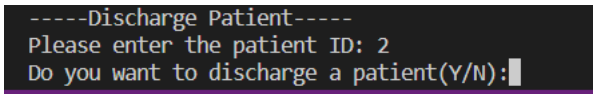
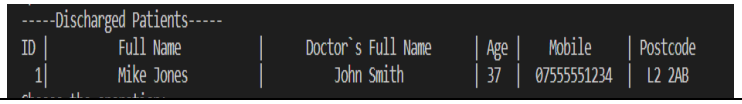
If the user enters 3, first a list of the doctors will be printed and the user will be asked to enter the ID of the doctor to update his data.

```
Input: 3
-----Update Doctor`s Details-----
ID |          Full name          |  Speciality
  1|          John Smith         | Internal Med.
  2|          Jone Smith         |  Pediatrics
  3|         Jone Carlos         |  Cardiology
  4|          Adela Duka         |  Pediatrics
Enter the ID of the doctor: 2
Choose the field to be updated:
 1 First name
 2 Surname
 3 Speciality
```

After a valid ID is entered the user will be asked to select the field he wants to update.
If the user selects 1, he will be asked for the new first name of the doctor.

```
Choose the field to be updated:
 1 First name
 2 Surname
 3 Speciality
Input: 1
Enter the new first name: Adam
```

		If the user selects 2, he will be asked for the new surname of the doctor.  If the user selects 3, he will be asked for the new speciality.  The doctor attributes will be updated calling the set_first_name(), set_surname() and set_speciality () methods of Doctor class. DELETE If the user enters 4, the list of the doctors will be printed and the user will be asked for the ID of the doctor to be deleted.  If the user provides a valid ID, the doctor with the entered ID will be deleted from the doctor list and a message “Doctor deleted” will be printed. If a doctor with that ID does not exist, a message “ The id entered was not found” will be printed.	#ToDo9
view_patient	self,patients	Print the list of patients	#ToDo10
assign_doctor_to_patient	self, patients, doctors	Allow the admin to assign a doctor to a patient. First, it prints a list of all the patients and ask the user for the ID of the patient he wants to assign to a doctor	#ToDo11

		 If the user enters an existing ID (ex.2) a list of the doctors will be printed and the user will be asked to specify the doctor ID for that patient.  If the user enters a valid doctor ID, using the **link() method of Class Patient and add_patient () of class Doctor**, the patients will be linked to the doctor and vice versa. A message will be printed "The patient is now assigned to the doctor."	
discharge	self, patients, discharge_ patients	Allow the admin to discharge a patient. It prints a list of all the patients and ask the user if he wants to discharge a patient.  If the user enters Y, it will be asked for the patient ID.  **If the patient ID exists, the patient will be removed from the list of patients and it will be added to the discharged list.** The user will be asked for discharging a patient until he enters N.	#ToDo12
view_discharg e	self, discharged _patients	Prints the list of all discharged patients 	#ToDo13
update_detail s	self	Allows the user to update the Admin data, changing username, password and address	#ToDo14 #ToDo15 #ToDo16

Main.py

1 admin instance has been created. To instantiate an admin instance from the Admin class 3 argument values should be passed to the Admin's class __init__ method . These are username, password and address.
Doctors and patients' objects are instantiated and stored in lists, doctors = [] and patients = [] respectively.

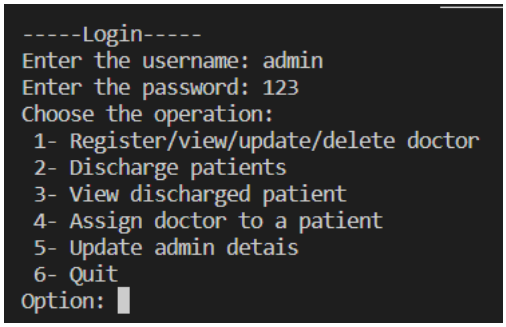
```
# Imports
from Admin import Admin
from Doctor import Doctor
from Patient import Patient

def main():
    """
    the main function to be ran when the program runs
    """

    # Initialising the actors
    admin = Admin('admin','123','B1 1AB') # username is 'admin', password is '123'
    doctors = [Doctor('John','Smith','Internal Med.'),
Doctor('Jone','Smith','Pediatrics'), Doctor('Jone','Carlos','Cardiology')]
    patients = [Patient('Sara','Smith', 20, '07012345678','B1 234'),
Patient('Mike','Jones', 37,'07555551234','L2 2AB'), Patient('Daivd','Smith', 15,
'07123456789','C1 ABC')]
    discharged_patients = []

    # keep trying to login tell the login details are correct
    while True:
        if admin.login():
            running = True # allow the program to run
            break
        else:
            print('Incorrect username or password.')

    while running:
        # print the menu
        print('Choose the operation:')
        print(' 1- Register/view/update/delete doctor')
        print(' 2- Discharge patients')
        print(' 3- View discharged patient')
        print(' 4- Assign doctor to a patient')
        print(' 5- Update admin detais')
        print(' 6- Quit')

        # get the option
```

```
        op = input('Option: ')

        if op == '1':
            # 1- Register/view/update/delete doctor
         #ToDo1
          pass

        elif op == '2':
            # 2- View or discharge patients
            #ToDo2
            pass

            while True:
                op = input('Do you want to discharge a patient(Y/N):').lower()

                if op == 'yes' or op == 'y':
                    #ToDo3
                    pass

                elif op == 'no' or op == 'n':
                    break

                # unexpected entry
                else:
                    print('Please answer by yes or no.')

        elif op == '3':
            # 3 - view discharged patients
            #ToDo4
            pass

        elif op == '4':
            # 4- Assign doctor to a patient
            admin.assign_doctor_to_patient(patients, doctors)

        elif op == '5':
            # 5- Update admin detais
            admin.update_details()

        elif op == '6':
            # 6 - Quit
            #ToDo5
            pass

        else:
            # the user did not enter an option that exists in the menu
            print('Invalid option. Try again')

if __name__ == '__main__':
    main(
```

In this file it is the main function, which runs once we run the program.

The login method of the admin is called to check if the username and password match the registered ones. If the username and password provided are correct, the following menu will be printed:

Table 4- Main Function

Function	Parameters	What should it do?	ToDo
main	-	• Main function, which runs once we run the program. • The login method of the admin is called to check if the username and password match the registered ones. If the username and password provided are correct, the following menu will be printed:  1. If the user enters 1 the **doctor_management()** method (inside the Admin class) should be called. 2. If the user enters 2 the **discharge ()** method (inside the Admin class) should be called. 3. If the user enters 3 the **view_discharge()** method (inside the Admin class) should be called. 4. If the user enters 4 the **assign_doctor_to_patient()** method (inside the Admin class) should be called. 5. If the user enters 5 the **update_details()** method (inside the Admin class) should be called. 6. If the user enters 6 the program terminates.	#ToDo1 #ToDo2 #ToDo3 #ToDo4 #ToDo5

Examples to start

1. Admin login function – ToDo1

First, let us fill in the method login**(self)** located inside the Admin class . Write the following code inside the method login(self). Remember to delete the statement **#ToDo1** and **pass** before placing your code. Make sure the indentation is kept the same as shown in the code below, because pasting text to Spyder IDE may corrupt the spacing and indentation of the source code.

```
return self.__username in username and self.__password == password
```

2. get_doctor_details function- ToDo2

Let us fill in the method **get_doctor_details(self)** located inside the Admin class. Write the following code inside the method get_doctor_details(self). Remember to delete the statement **#ToDo2** and **pass** before placing your code. Make sure the indentation is kept the same as shown in the code below, because pasting text to Spyder IDE may corrupt the spacing and indentation of the source code.

```
        first_name = input('Enter the first name: ')
        surname = input('Enter the surname: ')
        speciality = input('Enter the speciality: ')

        return first_name, surname, speciality
```

3. doctor_management function- ToDo3

Let us fill in the method **doctor_management (self,doctors)** located inside the Admin class. Delete the statement **#ToDo3** and **pass** and write the following code. Make sure the indentation is kept the same as shown in the code below, because pasting text to Spyder IDE may corrupt the spacing and indentation of the source code.

```
 op = input('Input: ')
```

1. 4. doctor_management function- ToDo4

Let us fill in the method **doctor_management (self,doctors)** located inside the Admin class. Delete the statement **#ToDo4** and **pass** and write the following code. Make sure the indentation is kept the same as shown in the code below, because pasting text to Spyder IDE may corrupt the spacing and indentation of the source code.

```
first_name, surname, speciality = self.get_doctor_details()
```

Continue in the same way to replace all the ToDo statements with your code.

Possible extensions for extra marks – Inheritance

Looking at the code provided we can see clearly that there are attributes and methods that are common/similar in both classes Patient and Doctor. It is more efficient to create a parent class Person that includes all the common attributes and methods between the Patient class and Doctor class. For this, the Patient class and the Doctor class may become subclasses of the Person class. As a result, they will inherit all the common attributes and methods included in the super class Person.

Please note the following:

❖ You have to use the provided partial implementation of the hospital management system as a starting point to complete this assessment.

❖ Exception handling should be implemented in the code; hence, the system should not crash if the user inserts unexpected input. For example, if a user input a string instead of an integer for the selection of a menu option. Students must write robust code.

❖ The source code may contain errors or logical issues, and it is the student responsibility to identify and fix these errors if they exist.