Student Name: **Weight: 30%**

Student ID: **Marks: /100**

Assignment: Classes

Type: Group Assignment

Needed Modules: 1 to 4 ONLY

- Students should **ONLY use** programming constructs covered in modules 1, 2, 3 and 4.
- **Submission will not be accepted when using programming concepts that are not covered in modules 1, 2, 3 and 4.**
- **Late submission, incorrect submission format, Public Github Repo, or submissions without a peer assessment will not be accepted.**
- Submit the following files to D2L:
    - The .py file (project_g<group_no>.py) with your code.
    - A pdf (assign3_g<group_no>.pdf) with a screenshot of your test outputs and Peer-assessment as well as a link to your main GitHub branch in your GitHub Repo (project216_g<group_no>). You must Add Hazem (https://github.com/hazemwork) to your Private Repo before the due date.
    - PDFs with your individual GitHub LinkedIn certificates.
- **Peer Assessment:** Each student must also complete a peer assessment of their group members. The peer assessment MUST also include a contribution information which summarizes what tasks are completed by each member.
- **GitHub Usage:**
    - ***Make sure that GitHub repository is Private.***
    - A separate branch(s) in Github must be created for each group member, containing the part they work on. The branch name should include the task name and the student name.
    - Ensure all group members push their parts to their respective Github branches.
    - When your group is ready, submit the main Github link to D2L. Only one copy is required per group on the main or master branch.
    - GitHub can be used for code reviewing

Github Training LinkedIn Learning:

Learning how to work efficiently with a team in a hosting platform such as Github is an essential skill for programmers. A group coding project such as this one provides the perfect opportunity to learn about and then practice this essential skill.

1. Complete one of the following LinkedIn Learning courses:
   - GitHub Essential Training [2 h 48 m] (https://www.linkedin.com/learning/github-essential-training/version-control-and-collaboration-with-github)
   - Git Essential Training: The Basics [2 h 55 m] (https://www.linkedin.com/learning/git-essential-training-the-basics/use-git-version-control-software-to-manage-project-code)
   - Any other course that is pre-approved by your instructor
2. Submit a copy of your certificate of completion or other evidence of completion, as approved by your instructor.

   Note: There should be no out of pocket expenses for the LinkedIn Learning course. As a SAIT student, you have free access to thousands of professional development courses through LinkedIn Learning. Ask your instructor if you run into issues accessing the courses.

Scenario

Alberta Hospital (AH) is a new healthcare provider in Alberta. To complement the existing large-scale hospitals located in urban settings, AH is building a network of smaller scale mini-hospitals which target underserved rural populations. AH has hired your company to create a management system which is customized to meet their unique operational needs.

Management System Details

Alberta Hospital (AH) requires that their management system application meets the following criteria.

- Supports data entry as well as report generation
- Uses the following classes throughout the application:
  - #1 - Doctor
  - #2 - DoctorManager
  - #3 - Patient
  - #4 - PatientManager
  - #5 - Management
- Uses classes to create objects that interact with each other
- Uses the properties and methods/functions listed below for each class.

Class #1: Doctor

Properties

Doctor ID, Name, Specialization, Working Time, Qualification, Room Number

Methods

Method Name	Description
Constructor	• __init__() should initialize the doctor object properties. • The constructor should allow creating a doctor object without passing values to the constructor ○ Hint: Use keyword arguments in the constructor
Getters	• Implement one getter function for each Doctor property. The getter function should return the value of the property. ○ Example is get_doctor_id(self)
Setters	• Implement one setter function for each Doctor property. The setter function should set the property to a new value. ○ Example is set_doctor_id(self, new_id)
__str__()	• It returns the string representation of a doctor object. • This representation should include all doctor properties separated by underscore (_)

Class #2: DoctorManager

Methods

Method Name	Description
Constructor	• It creates an empty list of doctors. • It calls read_doctors_file() to load doctors data from doctorss.txt into this list.
format_dr_info	• It receives a doctor object. • It formats doctor object information similarly to the format used in doctors TXT file (i.e., properties separated by underscore).
enter_dr_info	• Asks the user to enter the doctor info (id, name, etc.). • Creates a doctor object using the entered information. • Returns the created doctor object.
read_doctors_file	• Reads doctors data from file doctors.txt. • Create an object for each doctor record. • Append doctor objects to the doctors list.
search_doctor_by_id	• Searches for a doctor using their ID. • Accepts doctor ID from the user. • Iterates through the doctors list to check if a doctor with the entered id exists or not.

	• If the doctor exists, it displays the doctor information formatted as in the project output file. • Otherwise, it displays "Can't find the doctor….".
search_doctor_by_name	• Searches for a doctor using their name. • It accepts doctor name from the user. • Iterates through the doctors list to check if a doctor with the entered name exists or not. • If the doctor exists, it displays the doctor information formatted as in the project output file. • Otherwise, it displays "Can't find the doctor….".
display_doctor_info	• It takes a doctor object and displays doctor info as in the project output file.
edit_doctor_info	• Asks the user to enter the doctor id which the user wants to edit. • Searches the doctors list to find the doctor who has the entered id. • If the doctor exists, get the new values for name, speciality, timing, qualification and room number from the user. ○ Updates this information in the list. ○ Writes the updated doctors list to doctors.txt. ○ Confirms that the doctor has been edited • If the doctor does not exist, it displays "Cannot find the doctor …..".
display_doctors_list	• Iterates through the doctors list and display doctor's information as shown in the project output file.
Write_list_of_doctors_to_file	• Writes a list of doctors into the doctorss.txt file. ○ Iterates through doctors list. ○ Each doctor information must be formatted using format_dr_info() before writing it in the doctors.txt file.
add_dr_to_file	• It asks the user to enter the new doctor information such as id, name, speciality, qualification, and room number. ○ Hint, use enter_dr_info() to get the doctor information from the user • Appends the new doctor object to doctors list. • Formats this information to match the doctors.txt format. • Appends the new doctor to doctors file. • Confirms that a new doctor has been added

Sample data: doctors.txt (data file provided)

Class #3: Patient

Properties

pid, name, disease, gender, age

Methods

Method Name	Description
Constructor	• __init__() should initialize the patient object properties. • The constructor should allow creating a patient object without passing values to the constructor ○ Hint: Use keyword arguments in the constructor
Getters	• Implement one getter function for each Patient property. • The getter function should return the value of the property. ○ Example is get_pid(self)
Setters	• Implement one setter function for each Patient property. • The setter function should set the property to a new value. ○ Example is set_doctor_id(self, new_id)
__str__()	• It returns the string representation of a patient object. • This representation should include all doctor properties separated by underscore (_)

Class #4: PatientManager

Methods

Method Name	Description
Constructor	• It creates an empty list of patients. • It calls read_patients_file() to load patient data from patients.txt into this list
format_patient_Info_for_file	• It receives a patient object. • It formats patient object information similarly to the format used in patients file (i.e., properties separated by underscore)
enter_patient_iInfo	• Asks the user to enter the patient info (id, name, etc.) • Creates a patient object using the entered information • Returns the created patient object
read_patients_file	• Reads patients data from file patients.txt • Create an object for each patient record • Append patient objects to the patients list

search_patient_by_Id	• Searches for a patient using their ID • It accepts patient ID from the user • Iterates through the patients list to check if a patient with the entered id exists or not • If the patient exists, it displays the patient information formatted as in the project output file. • Otherwise, it displays "Can't find the patient…."
display_patient_info	• It takes a patient object and displays patient info as in the project output file
edit_patient_info_by_id	• Asks the user to enter the patient id which the user wants to edit. • Searches the patients list to find this patient. • If the patient exists, get the new values for name, disease, gender, and age from the user. ○ Updates this patient information in the list. ○ Writes the updated patients list to patients.txt. ○ Confirms that the doctor has been edited • If the patient does not exist, it displays "Cannot find the patient …..".
display_patients_list	• Iterates through the patients list and display patients information as shown in the project output file.
write_list_of_patients_to_file	• Writes a list of patients into the patients.txt file. ○ The patient information must be formatted using format_patient_info_for_file() before writing it in the patients.txt file.
add_patient_to_file	• It asks the user to enter the new patient information such as id, name, disease, etc. ○ Hint, use enter_patient_info() to get the patient information from the user. • Appends the new patient object to patients list. • Formats this information to match the patients.txt format. • Appends the new patient to patients file. • Confirms that a new doctor has been added

Sample data: patients.txt (data file provided)

Class #5: Management: Methods

Method Name	Description
display_menu	• It displays the main menu which has 3 options (1 for Doctors submenu, 2 for Patients submenu, and 3 for exiting the program. ○ The program should continue displaying the main menu until the user enters 3. • When the user selects option 1, Doctors submenu will be displayed to allow user working with doctors. ○ Doctors menu has 6 options. ○ The first 5 options allow a variety of manipulation (displaying doctors list, searching by id or name, adding a new doctor, and editing existing doctor information) of doctors. ○ Option 6 allows returning to the main menu. ▪ The program should continue displaying the doctors menu until the user enters 6. • When the user selects option 2, Patients submenu will be displayed to allow user working with patients. ○ Patients menu has 5 options. ○ The first 4 options allow a variety of manipulation (displaying patients list, searching by id, adding a new patient, and editing existing patient information) of patients. ○ Option 5 allows returning to the main menu. ▪ The program should continue displaying patients menu until the user enters 5.

Marking Criteria

	Criteria	Marks
Working code	• Code is DRY • The project runs in all scenarios – correct logic • Input requests match the scenario exactly • Correct use of if/else statements • Correct use of classes • Correct use of functions • Correct file manipulation • Output matches the scenario • No redundant code – No unnecessary calculations	**/75**
Style	○ Indentation – consistent ○ Readability – good variable names – good use of white spaces – good use of comments ○ No redundant syntax	**/20**
Version control (evaluated in Github)	○ group members adhered to version control best practices	**5**