**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 the course modules**.
  + **Submission will not be accepted when using programming concepts that are not covered in course modules.**
* **Late submission will not be accepted**
* **Submission will not be accepted if a group does not complete peer assessment and demonstration video.**

## 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.

## Equipment and Materials

For this assignment, you will need:

* Python IDE
* Github repository

## Instructions

This assignment consists of four sections, all completed outside of class time. See the course outline and Brightspace for exact dates.

**Github Training LinkedIn Learning (5%)**

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](https://www.linkedin.com/learning/github-essential-training/version-control-and-collaboration-with-github?autoplay=true&u=2245281) [2 h 48 m] (https://www.linkedin.com/learning/github-essential-training/version-control-and-collaboration-with-github)
* [Git Essential Training: The Basics](https://www.linkedin.com/learning/git-essential-training-the-basics/use-git-version-control-software-to-manage-project-code) [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

1. Submit a copy of your certificate of completion or other evidence of completion, as approved by your instructor.
2. Make sure that GitHub repository is private and add your instructor as collaborator.
3. A separate branch 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.
4. GitHub must be effectively used for group collaboration.
   1. GitHub should be used for code reviewing
5. Ensure all group members push their parts to their respective Github branches.
6. When your group is ready, submit the main Github link to Brightspace. Only one copy is required per group on the main or master branch.
7. The final project code must be pushed on Github and submit the link to it on Brightspace.

**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.

**Peer Assessment and demonstration Video (5%)**

**Peer Assessment (1%)**

Each student must also complete a peer assessment of their group members.

* In the peer assessment, each group member will give other members a point out of 10.
* The peer assessment MUST also include a contribution information which summarizes what tasks are completed by each member

**Demonstration Video (4%)**

The group should prepare up to 15 mins video in which:

* The group should demonstrate that the program works as requested and the output is exactly same as the provided sample project output.
* Each group member MUST demonstrate the code s/he developed
* Overall video time should be equally divided between students

**You MUST complete both parts to earn 5%. Missing the peer assessment or demonstration video will lead to ZERO in the whole assessment item.**

**Project Submission (90%)**

1. Check your solution against the detailed marking criteria at the end of this document.
2. Submit the following files to BrightSpace:

* The code of the program that you implemented (.py file).
* A copy of the test output (.txt file)
* Link of the main GitHub branch
* Peer-assessment document
* Demonstration video
  + - Please upload the video to Team (for example) and share the link in BrightSpace

## 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Needs Improvement (0–50%)** | **Good (51–75%)** | **Excellent (76–100%)** | **Marks** |
| **Working code** | * The project doesn’t run in all scenarios * Input requests work but don’t match the scenario * No conversion of data types * Syntax of if/else statements has mistakes * Use of classes is poor * Use of functions is poor * Output works but doesn’t match the scenario * Unable to read/write to files, or with many mistakes | * The project runs in all scenarios * Input requests work but don’t match the scenario * Some functions or methods are missing * Correct use of if/else statements * Correct use of classes * Output works but doesn’t completely match the scenario * Able to read/write to files but with a few mistakes | * The project runs in all scenarios * Input requests match the scenario exactly * Correct functions and methods developed * Correct use of if/else statements * Correct use of classes * Output matches the scenario * Able to read and write to files with no mistakes | **/60** |
| **Style** | * Indentation – not consistent * Readability – poor variable names * Documentation * No comments are included at the top. * No comments indicating major code sections or what they do | * Indentation – some parts are consistent and some are not * Readability – some variable names are not ideal * Documentation * Comments at the top are missing or incomplete. * Comments indicating major code sections and what they do are incomplete | * Indentation – consistent * Readability – good variable names * Documentation * Comments at the top are complete and include name, date, program description including details on inputs, processing and outputs  (4–5 sentences minimum). * Comments indicate major code sections and what they do | **/15** |
| **Testing** | * Sample output doesn’t match the provided sample run * Output is not formatted according to the specification (sample run) | * Parts of the sample output don’t exactly match the provided sample run * Output formatted according to the specification (sample run) | * Sample output exactly matches the provided sample run * Output formatted according to the specification (sample run) | **/15** |
| **Peer assessment and demonstration video** | * Review the above peer assessment and demonstration video section | | | **/5** |
| **Version control (evaluated in Github)** | * No evidence that group members practiced version control best practices | * Some evidence that some group members adhered to version control best practices. | * It is evident that all group members are consistently adhering to version control best practices. | **/5** |
| **Total** | | | | **/100** |