**Student Name: Weight: 40%**

**Student ID:** **Marks:** **/100**

# Assignment: Classes

## Scenario

Alberta Rural Patient Care (ARPC) is a new healthcare provider in Alberta. To complement the existing large-scale hospitals located in urban settings, ARPC is building a network of smaller scale mini-hospitals which target underserved rural populations. ARPC 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 schedule and Brightspace for exact dates.

**Git/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.

**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.

**Individual Submission (20%)**

1. Working individually, review the Scenario and the Management System Details sections of this document.
2. Write a program that meets the **requirements highlighted in red**.
3. When your program is complete, use the data contained in the provided Project Output PNG file to see if your program works correctly.

**Note:** Check your program against the marking criteria for individual submissions. Keep in mind that you will be completing and refining the program later as a group.

1. Push your code on Github and submit the link to it on Brightspace.

* The code of the program that you implemented (.py file)
* A copy of the output from a test run (.txt file)

**Group Submission (70%)**

1. After you’ve completed your individual submission, join a group, as directed by your instructor.
2. Share your program with your group and work together to develop one common, complete solution that meets all the requirements.
3. Check your solution against the detailed marking criteria at the end of this document.
4. Create a separate branch in Github for each group member, containing the part they work on. The branch name should include the task name and the student name.
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.

* The code of the program that you implemented (.py file).
* A copy of the test output (.txt file)

**Peer Assessment (5%)**

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

## Management System Details

Alberta Rural Patient Care (ARPC) 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. Facility
  3. Laboratory
  4. Patient
* Uses classes to create objects that interact with each other
* Uses the methods/functions listed below
* Each object has descriptive attributes/ characteristics that represent the work and actions of the class as outlined below.

### Class #1: Doctor

**Attributes**

ID, Name, Specialty, Schedule, Qualification, Room Number

**Methods**

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| formatDrInfo | Formats doctor’s information (attributes) in the same format used in the “doctors.txt” file (i.e., has underscores between values) |
| \_\_str\_\_ | Formats doctor information(attributes) – for display, see Sample Run, “Display Doctor’s List” |

**Sample data: doctors.txt** (data file provided)

### Class #2: Facility

**Attributes**

Facility Name

**Methods**

|  |  |
| --- | --- |
| **Method name** | **Description** |
| formatFacilityInfo | Formats facility information (attributes) in the same format used in the “facilities.txt” file |
| \_\_str\_\_ | Formats facility information(attributes) – for display, see Sample Run, “Display Facilities List” |

**Sample data: facilities.txt** (data file provided)

### Class #3: Laboratory

**Attributes**

Lab Name, Cost

**Methods**

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| formatLabInfo | Formats laboratory information (attributes) in the same format used in the “laboratories.txt” file (i.e., has underscores between values) |
| \_\_str\_\_ | Formats laboratory information(attributes) – for display, see Sample Run, “Display Laboratory’s List” |

**Sample data: laboratories.txt** (data file provided)

### Class #4: Patient

**Attributes**

ID, Name, Diagnosis, Gender, Age

**Methods**

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| formatPatientInfo | Formats patient information (attributes) in the same format used in the “patients.txt” file (i.e., has underscores between values) |
| \_\_str\_\_ | Formats patient information - for display, see Sample Run, “Display Patient’s List” |

**Sample data: patients.txt** (data file provided)

### Management Module

Create a separate module for your management application code which will be comprised of various functions, including a main() entry function that controls the overall processing. The functions listed below are required but you may use additional functions too.

**Doctor-related Functions**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| enterDrInfo | Asks the user to enter doctor information (attributes), then creates and returns a new Doctor object |
| readDoctorsFile | Reads “doctors.txt” file into a list of Doctor objects |
| searchDoctorById | Searches the list of Doctor objects for a specified doctor ID using the doctor ID that the user enters. Returns Doctor object or -1 if not found. |
| searchDoctorByName | Searches the list of Doctor objects for a specified doctor name using the doctor name that the user enters. Prints Doctor object if found. |
| editDoctorInfo | Finds doctor ID in the list of Doctor objects, prompts for new values and updates the remaining attributes |
| displayDoctorsList | Displays all the Doctors’ information(attributes) in doctors list |
| writeDoctorsListToFile | Writes “doctors.txt” file from the list of Doctor objects, maintaining correct formatting |
| addDrToList | Gets new Doctor object (with user-entered doctor information) and adds it to the doctors list |

**Facility-related Functions**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| addFacilityToList | Gets new Facility object (with user-entered facility information) and adds it to the facilities list |
| readFacilitiesFile | Read “facilities.txt” file into a list of Facility objects |
| displayFacilitiesList | Displays all the Facility information(attributes) in facilities list |
| writeFacilitiesListToFile | Writes “facilities.txt” file from the list of Facility objects, maintaining correct formatting |

**Laboratory-related Functions**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| addLabToList | Gets new Laboratory object (with user-entered lab information) and adds it to the laboratories list |
| displayLabsList | Displays all the Laboratory information in the laboratories list |
| enterLaboratoryInfo | Asks the user to enter lab name and cost, then creates and returns a Laboratory object |
| readLaboratoriesFile | Reads ”laboratories.txt” file into a list of Laboratory objects |
| writeLabsListToFile | Writes “laboratories.txt” file from the list of Laboratory objects, maintaining correct formatting |

**Patient-related Functions**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| enterPatientInfo | Asks the user to enter patient information (attributes), then creates and returns a new Patient object |
| readPatientsFile | Reads “patients.txt” file into a list of Patient objects |
| searchPatientById | Searches the list of Patient objects for a specified patient ID using the patient ID that the user enters. Returns Patient object or -1 if not found. |
| editPatientInfo | Finds patient ID in the list of Patient objects, prompts for new values and updates the remaining attributes |
| displayPatientsList | Displays all the Patients’ information(attributes) in patients list |
| writePatientsListToFile | Writes “patients.txt” file from the list of Patient objects, maintaining correct formatting |
| addPatientToList | Gets new Patient object (with user-entered patient information) and adds it to the patients list |

**Menu Functions**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| mainMenu | Displays main menu options, accepts and returns user’s choice |
| doctorsMenu | Displays doctors menu options, accepts and returns user’s choice |
| facilitiesMenu | Displays facilities menu options, accepts and returns user’s choice |
| laboratoriesMenu | Displays laboratories menu options, accepts and returns user’s choice |
| patientsMenu | Displays patients menu options, accepts and returns user’s choice |

Sample Run: see file *Project Output.png*.

## Marking Criteria

### Individual Submissions

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Needs Improvement (0–49%)** | **Passing (50–100%)** | **Marks** |
| **Git/Github training** | * No evidence submitted of having completed a LinkedIn Learning course. | * LinkedIn Learning Certificate of Completion submitted   OR   * Other, instructor-approved proof of completion submitted | **/5** |
| **Individual work** | * Program meets less than half of the requirements for the group submission * Incorrect file(s) submitted * Does not follow programming best practices * Inputs are not entered as required and outputs are not generated as required | * Program works in most scenarios, but could use improvement * Most inputs can be entered as required, and most outputs are generated as required * Reasonable effort has been made to format the output as required | **/20** |
| **Peer assessment** | * Not submitted or incomplete | * Completed for all group members | **/5** |

### Group Submission

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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 | **/45** |
| **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 | **/10** |
| **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) | **/10** |
| **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** |