CSE1IOO/CSE4IOO Semester 2, 2020
Programming Assignment – Part 2 (20%)

Assessment: This assignment is worth 20% of the final mark for this subject. Please read the entire specification carefully, before you start writing the code.

Due Date: Monday, 12 October 2020, at 10 am.

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime.

This is an individual Assignment. You are not permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Information Technology treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the unit guide for further information and strategies you can use to avoid a charge of academic misconduct. ***All submissions will be electronically checked for plagiarism using latcs8 server program. If you are new at La Trobe, please complete academic integrity module on LMS to understand what is regarded as an academic misconduct.***

Objectives: The general aims of this assignment are:

- To analyse a problem in an object-oriented manner, and then design and implement an object-oriented solution that conforms to given specifications
- To practise using inheritance in Java
- To practise file input and output in Java
- To make implementations more robust through mechanisms such as exception handling.

Submission Details and Marking Scheme: Instructions on how to submit electronic copies of your source code files from your latcs8 account and a marking scheme overview are given at the end. If you have not been able to complete a program that compiles and executes containing all functionality, then you should submit a program that compiles and executes with as much functionality as you have completed. (You may comment out code that does not compile.)

NOTE: While you are free to develop the code for this assignment on any operating system, your solution must run on the latcs8 system.

NOTE: You can use Arrays or ArrayList or LinkedList of the Java API. If you use arrays, assume that we never have more than 50 patients in the system.

Background

As described in the handout for Part 1, the overall aim of the assignment is to develop a prototype for a patient record system.

In Part 1, you have implemented the classes to represent patients, medical observation types, and observations. You have also implemented a PatientRecordSystem class which allows us to add patients, observation types and observations.

Building on the work that you have done for part 1, in this Part 2, you will be adding more functionalities to the classes and implementing a menu program. You are required to do the tasks (task 1, 2, and 3) described below.

Task 1

Implement **two** methods for the **`PatientRecordSystem`** class to save and load data.

****
The **first** method should have the header:

```
public void saveData() throws Exception
```

The method should write the data about observation types, patients, and observations that are currently in the **`PatientRecordSystem`** instance, in five text files as described below:

1. PRS-MeasurementObservationTypes.txt

The file will contain information about the measurement observation types. The required format of the text data in the file is shown with an example below:

```
T100; Blood Pressure; psi
T400; Height; cm
```

Each measurement observation type must be on a separate line. Each line must contain the observation type code, name, and unit, separated by semi-colons.

2. PRS-CategoryObservationTypes.txt

The file will contain information about the category observation types. The required format of the text data in the file is shown with an example below:

```
T200; Blood Type; Group A, Group B1, Group B2
T300; Stress Level; Low, Medium, High
```

Each category observation type must be on a separate line. Each line must contain the observation type code, name, and the categories. The three elements, i.e., code, name, and categories are separated by semi-colons. The different categories are separated by commas.

3. PRS-Patients.txt

The file will contain information about the patients' data without their observations. The required format of the text data in the file is shown with an example below:

```
P100; John Smith
P200; Anna Bell
```

Each patient must be on a separate line. Each line must contain the patient's id and name, separated by a semi-colon.

4. PRS-MeasurementObservations.txt

The file will contain information about the patients' measurement observations. The required format of the text data in the file is shown with an example below:

```
P100; T100; 120.0
P200; T400; 180
```

Each measurement observation must be on a separate line. Each line must contain the patient's id, the observation code and the observation value (a double). The fields are separated by semi-colons.

5. PRS-CategoryObservations.txt

The file will contain information about the patients' category observations. The required format of the text data in the file is shown with an example below:

```
P100; T200; Group A
P200; T300, Low
```

Each category observation must be on a separate line. Each line must contain the patient's id, the observation code and the observation value. The fields are separated by semi-colons.

For the `saveData()` method, refer to the lecture notes of week 5, and revise the contents related to the use of PrintWriter class.

****

The **second** method to load data from text files should have the following header:

```
public void loadData() throws Exception
```

The method will read the data from five text files and save them in a **`PatientRecordSystem`** instance by calling methods of the same class to add measurement observation types, category observation types, etc. For example, when you read details about a patient from `'PRS-Patients.txt'` file, you should call the `addPatient()` method with those details to add the patient in the **`PatientRecordSystem`**.

The data should be loaded into a **`PatientRecordSystem`** instance which is empty of data. For the `loadDate()` method, revise the lecture notes and text files under week 5 and focus on the Scanner class.

You should write a small program to test these two methods. A sample test program is given in the Appendix 1.

As for Part 1, **your implementation must be such that the sample test program can be run without any change.**

Task 2

Write the menu program, called **`PatientRecordSystemMenu`**, to provide options in the following menu:

```
====================
Patient Record System
====================

1. Add a measurement observation type
2. Add a category observation type
3. Add a patient
4. Add a measurement observation
5. Add a category observation
6. Display details of an observation type
7. Display a patient record by the patient id
8. Save data
9. Load data
D. Display all data for inspection
X. Exit
Please enter an option (1-9 or D or X):
```

The menu program must use the **`PatientRecordSystem`** class. For each option, the program should ask for the required inputs, each with a separate prompt.

For option 6, 7 and D, it is acceptable just to use the toString methods of the classes involved.

After a menu option is selected and the operation is completed, the menu should be presented to the user again. See Appendix 2 for a short sample run.

The menu program for this task is only required to satisfy the data correctness conditions. It is not required to be robust, i.e. it can crash when certain exceptions occur.

For this task, the examples in lab 3 and lab 4 might be worth looking into.

Task 3

Implement a class called **`RobustPatientRecordSystemMenu`**, which makes the previous menu program robust by including exception handling features. It is required to be robust only in the following sense: when an exception occurs in the execution of an option, the program will display some information about the exception on the screen and return to the main menu.

Electronic Submission of Your Source Code

- Submit all your Java files on latcs8 server.
- The code has to run under Unix on the latcs8 machine.
- If you want to copy a file from your local machine (laptop, desktop at home) to latcs8 server, use WinSCP (for Windows), or Cyberduck (for mac).
- While submitting the files on latcs8, make sure you are in the same directory as the files you are submitting.
- Submit each file separately using the submit command. For example, for a file called 'Patient.java', use the following command:

  ```
  submit IOO Patient.java
  ```

- After submitting the files, you can run the following command to check which files you have submitted:

  ```
  verify
  ```

- You can submit the same file as many times as you like before the assignment deadline. The previously submitted copy will be replaced by the latest one.

Marking Scheme Overview

The assignment has the total of 100 marks, which are distributed as follows:

- Implementation (Execution of code) 90 marks (Do all parts of the programs execute correctly? Note your programs must compile and run to carry out this implementation marking. So, comment out the non-working code in your submission.)
- Code Design, Layout and Documentation 10 marks (Does the program conform to specifications? Does the program solve the problem in a well-designed manner? Does the program follow good programming practices? Does the indentation and code layout follow a good, consistent standard? Are the identifiers meaningful? Are comments useful in explaining what and how the program works? (Javadoc comments are optional.)
- A detail marking rubric will be published on LMS soon.

Appendix 1 – A sample program to test methods to save and load data

```
import java.io.*;
import java.util.*;

public class PatientRecordSystemTester
{
	public static void main(String [] args) throws Exception
	{
	   testSaveData();
	   testLoadData();
	}
	public static void testSaveData() throws Exception
	{
	   // Create PatientRecordSystem
	   // Add observation types , patients and observations

	   PatientRecordSystem prs = new PatientRecordSystem();
	   prs.addMeasurementObservationType("T100", "Blood Pressure", "psi");
	   String [] categories = {"Group A", "Group B1", "Group B2"};
	   prs.addCategoryObservationType("T200", "blood type", categories);
	    String [] temp = {"low", "Medium", "high"};
	    categories = temp;
	    prs.addCategoryObservationType("T300", "stress level", categories);
	    prs.addMeasurementObservationType("T400", "height", "cm");
	    prs.addPatient("P100", "Smith");
	    prs.addPatient("P200", "Adams");
	    prs.addMeasurementObservation("P100", "T100", 120);
	    prs.addCategoryObservation("P100", "T200", "Group A");
	    // save data to file
	    prs.saveData();
	 }
	public static void testLoadData() throws Exception
	{
	    PatientRecordSystem prs = new PatientRecordSystem();
	    prs.loadData();
	    System.out.println(prs);
	}

}
```

Appendix 2 – A sample run of the menu program

```
====================
Patient Record System
====================

1. Add a measurement observation type
2. Add a category observation type
3. Add a patient
4. Add a measurement observation
5. Add a category observation
6. Display details of an observation type
7. Display a patient record by patient id
8. Save data
9. Load data
D. Display all data for inspection
X. Exit
Please enter an option (1-9 or D or X): 2
Enter observation type code: t1
Enter observation type name: stress level
Enter the number of categories: 3
Enter category 1: low
Enter category 2: medium
Enter category 3: high

====================
Patient Record System
====================

1. Add a measurement observation type
2. Add a category observation type
3. Add a patient
4. Add a measurement observation
5. Add a category observation
6. Display details of an observation type
7. Display a patient record by patient id
8. Save data
9. Load data
D. Display all data for inspection
X. Exit
Please enter an option (1-9 or D or X): 3
Enter patient ID: p1
Enter patient Name: john smith
```

```
====================
Patient Record System
====================

1. Add a measurement observation type
2. Add a category observation type
3. Add a patient
4. Add a measurement observation
5. Add a category observation
6. Display details of an observation type
7. Display a patient record by patient id
8. Save data
9. Load data
D. Display all data for inspection
X. Exit
Please enter an option (1-9 or D or X): 5
Enter patient ID: p1
Enter observation type code: t1
Enter observation type value: medium

====================
Patient Record System
====================

1. Add a measurement observation type
2. Add a category observation type
3. Add a patient
4. Add a measurement observation
5. Add a category observation
6. Display details of an observation type
7. Display a patient record by patient id
8. Save data
9. Load data
D. Display all data for inspection
X. Exit
Please enter an option (1-9 or D or X): d
-------------------------
PATIENT RECORD SYSTEM DATA
-------------------------
OBSERVATION TYPES:
-- CategoryObservationType[code: t1, name: stress level, categories: |low|m
edium|high|]

PATIENTS:
-- Patient id: p1, name: john smith Observations:
 - CategoryObservation[observationType: CategoryObservationType[code: t1, n
ame: stress level, categories: |low|medium|high|], value: medium]
```

```
====================
Patient Record System
====================

1. Add a measurement observation type
2. Add a category observation type
3. Add a patient
4. Add a measurement observation
5. Add a category observation
6. Display details of an observation type
7. Display a patient record by patient id
8. Save data
9. Load data
D. Display all data for inspection
X. Exit
Please enter an option (1-9 or D or X): x
Bye!
```