COMP1161

LAB5

PREAMBLE

This lab aims to provide experience in the use of exceptions and input/output, and also demonstrates a few concepts related to polymorphism. It specifically focuses on the following skills:

1 Writing and using try and catch blocks for specific exception classes

2 Reading input from the keyboard and from files.

3 Implementing polymorphism via overriding

THE PROBLEM

Transportation of the populace is a vital activity that is key, not only to a nation's development, but it's very survival. In acknowledgement , the Ministry of Transport in the island of Jamrock plans to prototype a system called Transporter Prime over the coming week. The system operates on three main inputs, namely,

1. A list of persons that require transportation contracts

2. A list of approvals, each with the id of the person who has been approved, and which **may** includebackground safety requirement data. An example of the information in safety requirement data is that the person cannot be in a fast moving vehicle due to health conditions

3. A list of contract batches, each with the name of the batch, the number of individuals that can be served by the batch, a preference rating for the batch, and a list of disclaimers. A disclaimer indicates the provider will adhere to a specific safety requirement. An example of a disclaimer that vehicles from a specified transport provider drive fast

Contracts are then approved over the set of waiting persons in the following order:

1. Contracts are approved in the order of the preference rating of provider.

2. While processing a batch of contracts, the approved persons are processed in reverse order of age (ie. oldest first). Anyone with a safety requirement that matches a disclaimer will not be given a contract from the affected provider.

Please note that a total of 10 marks are allocated to the lab. Lab-techs will use the 5 marks earned from the autograder as a guide to functionality, and the other 5 marks are earned from an assessment of understanding.

MARKING CRITERIA (PLEASE REVIEW BEFORE STARTING)

Criterion	Mark(s)
Explanation of the use of the exception hierarchy in software	1
Explanation of techniques for file/screen I/O	1
Explanation of why method findPerson needed a wildcard (ie. ?)	1
Explanation of why sort orderings affected logic in this system	1
Coding style and readability.	1

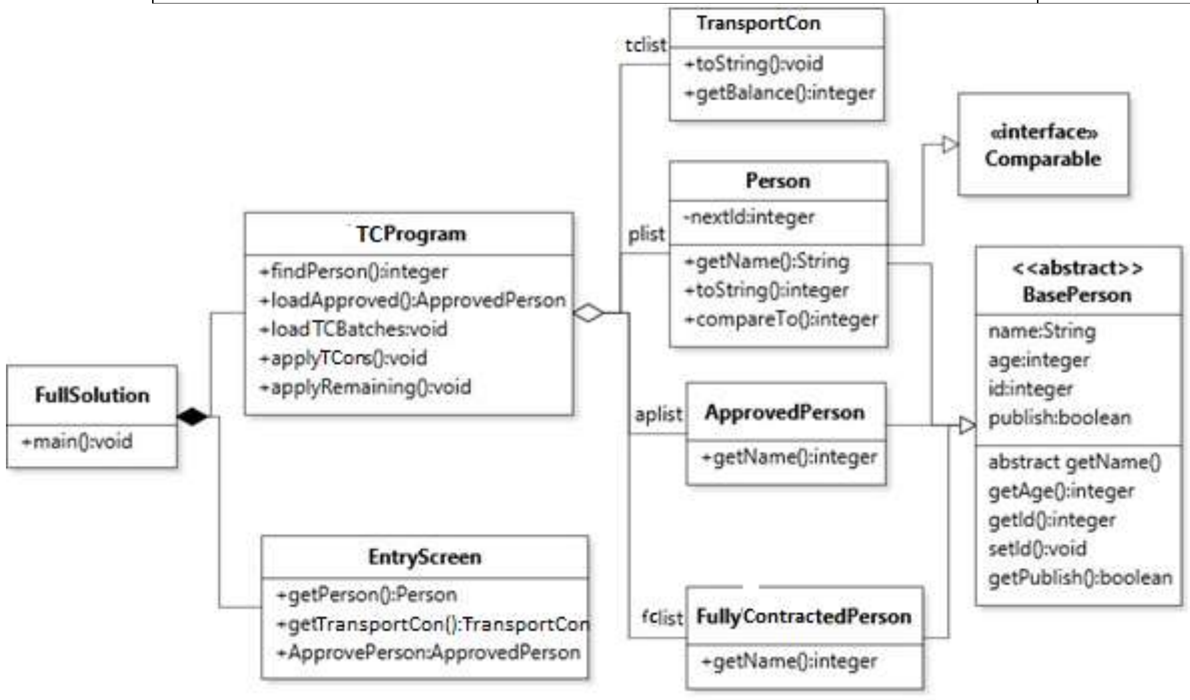

Figure 1: Simplified UML Class Diagram for Vaccinator Prime

Figure 1 depicts a simplified model of the classes that are needed to solve this problem. The starting code includes classes that will be run locally on your system, or in the REPL. The starting code implements the following functionality:

1. Allow the user to indicate whether interactive or test case execution is to be used.
   a. If running test case execution, the system will execute test cases, if a local id number has been saved.
   b. If interactive mode is selected, the program allows you to either prime the system by specifying a test case to load(six test cases have been provided), or go directly into data entry mode. Once in data entry mode, the user has to option to enter data, show the data, enter an ID number and print a report that can be read as a web page (which includes the list of transport contract batches and the list of persons who have agreed to publish)- Note the report location (which is printed by the application) allows you to see user directory for the application. .
2. Note the system includes a class named TestCase, however there is no need for you to interact with it. You should not modify it.

Your specific tasks to complete the system include :

1. Completing the menu in **FullSolution** to allow tests to be run
2. Writing the logic for the **findPerson** method in **TCProgram**
3. Completing the **loadApproved** method in **TCProgram**
4. Write logic for the **loadTCBatches** method in **TCProgram**
5. Write the **getTransportCon** method in **EntryScreen**
6. Write the **toString** methods in **ApprovedPerson** and **FullyContractedPerson**
7. Write the **publish** method in **FullyContractedPerson**
8. Ensure appropriate sort orders are in place for **TCBatch**, **ApprovedPerson** and **FullyContractedPerson**.

LAB EXERCISES

1 Allow the system to run test cases
   a. Read the code in the file **FullSolutions**.java. Verify that the logic allows a user to select whether interactive mode is required, and that if interactive mode is required, that the program loops until the user enters 'X'. Verify also that if a user selects one of a given set of characters, specific actions will be performed.
   b. Adjust the system so that if the user selects 'n' , the system will execute the method **TestCase.runTestCases**(…).(This process may be as simple as modifying comments).
   c. Modify the main method of full solution so that the interactive mode submenu also includes a submenu **Run [T]est Cases**. On entering either T or t, the system should perform the following actions:
      - Create an instance of **TCProgram**.
      - execute**TestCase.runTestCases**(…), with the instance of **TCProgram** as an argument.

2. Locate the method **findPerson** method in class **TCProgram**. Verify that it accepts an ArrayList that contains elements are instances of a derived class of **BasePerson**, as well as an integer that can represent a person's ID.
   a. Verify that the **findPerson** method currently returns the value -1.
   b. Adjust the method **findPerson** method in class **TCProgram** so that it searches through the ArrayList for the ID. If the ID is found, **findPerson** returns the position in the ArrayList at which the person was located. If not found, **findPerson** returns -1. *(Note – because you are to return the ordinal position – you may want to consider using a traditional loop rather than an iterator).*
3. Ensure approvals are operational
   a. Read the (commented) section of **loadApproved**, and verify that if, there exists a scanner object that is accessed using the name, then, on each iteration, apscan will return a string. Verify also that if the returned string is a space delimited list, then the first item is a number that is passed as an argument to **findPerson**, and the result of **findPerson** used to create an **ApprovedPerson** object. Note that apscan will indeed be used to access a file. That file can contain multiple lines where, on each line, the first item represents the ID number, and if there are any other items then each item represents a safety requirement for the **ApprovedPerson**. Examples of the file format to be read include cases/TestCase2.approved.txt and cases/TestCase4.approved.txt.
   b. The object that is to be used to access a file, **apscan**, has been declared and initialized to null. You are to initialize **apscan** to read the file that corresponds to the file name passed in as an argument to **loadApproved**. You will need to implement appropriate error handling so that the program runs properly.
   c. In **ApprovedPersons**, update the overridden method **getName** so that it returns

names in the format **lastname, firstname**
Hint… to get parts of a name, you can use syntax like- String[] nameparts = name.split(" ");

d. Complete the **toString**() method for **ApprovedPerson** so it follows the format
   getId()+"\t"+getName()+"\t\t"+getSafeReqs()

4. Allow contract batches to be properly entered
   a. Complete the method logic for the method **loadTCBatches** in class **TCProgram**. **loadTCBatches** is to use scanner object to open a file using the name passed to it as an argument. The scanner should read each line in the file until at the end of the file. Each line contains a name, a size and a preference rating. IF there are more than three items in the file, the fourth item represents a list of disclaimers. Examples of the file format can be found in cases/TestCase4.batches.txt and cases/TestCase5.batches.txt. After each line is read **loadTCBatches** should create a contract batch with the newly extracted data, and add it to an ArrayList. The ArrayList with all data encountered should be returned. If no disclaimers have been found for an contract batch, use an empty string for the disclaimer. NOTE: As a design decision, you are instructed not to include exception handling inside of **loadTCBatches**. You are to throw exceptions from that method instead.
   b. Write the toString() method of **TransportCons** to match the format:
   getName()+"\t"+getSize()+"\t"+getBalance()+"\t"+getPreference()+"\t"+getDisclaims()

5. Set ordering of contract approval delivery to meet specifications
   a. Write the **toString** method of **FullyContractedPerson** to match the template
   getId()+(getPublish()?"*":"")+"\t"+getName()+"\t\t"+getOperatorName()
   b. Verify sort orderings in the **applyTCons()** and **applyRemaining()** methods of **TCProgram** to meet the following requirements
      i. Contracts are applied in order of preference (highest first)
      ii. Contracts are given to approved persons in order of age(oldest first)
      (you can read to verify logic that expects **tclist** to be sorted by preference, and **aplist** to be sorted by age. **fclist** should be sorted in alphabetical order of name for consistent reporting.)

6. Write the **publish** method of **FullyContractedPerson** so that it returns the following code ONLY IF the associated person has agreed to publish(ie. **getPublish**() returns true):
   "<p>"+getName() + " travels with " + operatorName + "!!!</p>"