DPIT121 – Assignment I

Due: Week 4 and 5 lab

In Assignment I, you will continue on another iteration for the same scenario from lab 1, 2, and 3. If you have not finished those labs, you have time to work on them and finalize your assignment I. We can consider lab 1 to 3 as internal iterations or Scrum Sprints inside the company and Assignment I as a phase of project delivery to the customer.

Assignment I has three levels of completion: Core, Standard, and Advanced. You should completely finish a level before attempting the following level.

Core Level: 6 marks to be marked in Week 4 or Week 5

Standard Level: 9 marks to be marked in Week 4 or Week 5

Advanced Level: 12 marks to be marked in Week 4 or Week 5

Core Level:

You need to finish all the requirements of lab 1 to 3 including all the classes, methods and the test code. In addition, you need to do the following tasks:

1) Comment out the scanner and user input from your test code that was required in lab 2 and 3. You need to write a proper Test Case (refer to Lectures) and modify your test code to implement the test case. Test code should call all the methods across all classes. To validate your program, write your output expectations in the test case (You may use excel) and compare the output of the test code with your expectation. Include the test case in your submission. You need to justify and convince me as your customer that the output is correct based on your test case and expectations. Note that the test case should not include any user interaction (Scanner) and has to follow a proper story. 1 mark

2) Draw a UML class diagram for your program by using UMLet or any other tools. 1 mark

3) Draw a UML Sequence Diagram for ArrayList<InsurancePolicy> filterByExpiryDate (int userID, MyDate date) in InsuranceCompany class. 1 mark

4) Design and implement a basic command based console User Interface. (See the sample code in week 3). The program asks company admin to enter the username and password to log in to the system. After successful log in the program shows 9 options: 3 marks

Option 1: Test code: To run the test code based on your test case

Option 2: Create User: To get the information from admin and creates a user

Option 3: Create ThirdParty Policy: To get the information of a policy and a user id from admin and creates a ThirdPartyPolicy for the given user. Prints an error message if not successful.

Option 4: Create Comprehensive Policy: To get the information of a policy and a user id from admin and creates a Comprehensive Policy for the given user. Prints an error message if not successful.

Option 5: Print User Information: To get a user ID from user and prints the user information (including all user policies). Prints an error message if user ID is not found.

Option 6: Filter by Car Model: To get a car model from admin and filters and prints all the policies in the system for that car model as well as the total payment for the filtered policies.

Option 7: Filter by Expiry Date: To get a date and a user id from admin and filters and prints all the policies belong to that user and expiring by the given date.

Option 8: Update Address: To get a user id and address information and updates the address of the given suer

Option 9: Log Out

Play with your UI to be sure that all options are working properly. Remember that you already created few users and policies in your test code.

Standard Level:

1- You need to add new methods to your class InsuranceCompany for a data aggregation report. The report is the total premium payments of all users per city (from User.address).

- ArrayList<String> populateDistinctCityNames() //goes through all the users and populate a list of distinct city names for all users and returns it as a list.

- double getTotalPaymentForCity(String city) // returns the total premium payment for the given city across all users.
- ArrayList<Double> getTotalPaymentPerCity(ArrayList<String> cities) // aggregates the total premium payments for each city in the list and returns it as a double list with the same order as city names. This method calls getTotalPaymentForCity (String city)
- reportPaymentPerCity(ArrayList<String> cities, ArrayList<Double> payments, // generates the report as follow:

City Name	**Total Premium Payment**
Wollongong	$10,500.00
Sydney	$45,980.00
…	

Assuming that there are 110 users with the city=”Wollongong” and the total of payments for all of their policies is $10,500

* Update your test code to test these extra methods.

2- Update the text based menu as below:

The Main Menu has 3 options:

Option 1: Admin Login. Then it prompts to get admin username and password and if login is successful it shows the Admin Menu if not it shows an error message and shows the main menu again.

Option 2: User Login. It shows the User Menu

Option 3: exit the program

Admin Menu has all 9 options the same as core. Add options to do the data aggregation reports.

User Menu has options for the methods inside User class such as *Add a Policy, print a Policy, change the address, etc*

Note: Each menu is displayed again after each report/task till user enters the option to log out and returns to the main menu. After each task a message "please press any key to continue" will be displayed and after the user presses a key the menu will be displayed again.

Advanced Level:

Add these functionalities to your code as well as UI:

1- User and admin to be able to remove a policy from a user

2- User ID to be automatically generated (incrementally). Hint: have a static int count in the class and increment it

3- Admin to be able to remove a user by providing the user ID.

4- Admin to be able to change the admin password

5- You need to add new methods to your class User and InsuranceCompany for another data aggregation report. The report is the count, total and average of premium payments per CarModel across all the polices a user holds and across all the users in the company

Add these methods to User:

- ArrayList<String> populateDistinctCarModels() // goes through all the policies for a user and populates a list of distinct car model names
- double getTotalCountForCarModel(String carModel) // returns the number of policies this user owns for the given carModel
- double getTotalPaymentForCarModel(String carModel) // returns the total payments for the given carModel across all the policies this user owns

- ArrayList<Integer> getTotalCountPerCarModel (ArrayList<String> carModels) // returns the count for each model in the carModels as a list of integers. Call getTotalCountForCarModel(String carModel) in a loop for each carModel in the list
- ArrayList<Double> getTotalPaymentPerCarModel (ArrayList<String> carModels) // returns the Total Payment for each model in the carModels as a list of doubles. Call getTotalPaymentForCarModel(String carModel) in a loop for each carModel in the list
- reportPaymentsPerCarModel (ArrayList<String> carModels, ArrayList<Integer>counts, ArrayList<Double> premiumPayments) // generates the report such as below:

Car Model	Total Premium Payment	Average Premium Payment
Toyota Camry 2018	$11,050.00	$1,050.00
Nissan dualis 2018	$4,250.00	$850.00

(Assume this user has ten policies with the Car Model of "Toyota Camry 2018" and five with Nissan dualis 2018)

Add these methods to InsuranceCompany:

- ArrayList<String> populateDistinctCarModels() // goes through all the users within the InsuranceCompany and populates a list of distinct car models. You need to call the corresponding method inside the User and aggregate them for all the users
- ArrayList<Integer> getTotalCountPerCarModel (ArrayList<String> carModels) // returns the count for each model across all the users. You need to call the corresponding method inside the User and aggregate them for all the users
- ArrayList<Double> getTotalPaymentPerCarModel (ArrayList<String> carModels) // the same as the previous one but across all users in the company

- reportPaymentsPerCarModel (ArrayList<String> carModels, ArrayList<Integer>counts, ArrayList<Double> premiumPayments) // to generate the same report as before but across all users in the system

6- Modify your test code to test all of these methods for User and InsuranceCompany.

7- Update your UI to include all of these changes for User and InsuranceCompany.