**DPIT121 – Lab Exercise 5**

**Due: Week 7 lab**

In lab 5, you will continue on another iteration focusing on HashMap as a Data Structure and for Data Aggregation, and Exception Handling.

**Download and study lecture codes from week 5. Also, study Lab 2 Solution from "Sample labs 4-6 and Solutions" folder which is similar to this lab.**

1) Comment the ArrayList of policies in the User class and users in the InsuranceCompany and replace it with HashMaps. The HashMap key is the **policyID** and **userID** and the value is the corresponding policy and user, respectively. Don't remove any codes and methods but comment the old code and modify your code/methods to be compatible with HashMaps as below: (0.5 marks)

- Comment the old code and modify the addPolicy, findPolicy, addUser, and findUser for Hashmap instead of ArrayList.

- Keep all the static methods: print, filter, calcTotalPayment, and carPriceRiseAll for ArrayList<InsurancePolicy> inside the **InsurancePolicy class** but add new methods with the same names but for a HashMap of policies.

- Comment the old code for print, filter, calcTotalPayment, and priceRiseAll for ArrayList inside the **User class** and modify them for the new static methods inside InsurancePolicy working with HashMap. You also need to modify toString(), copy constructor, and clone() methods.

- Comment the old code for any methods inside the **InsuranceCompany** working with ArrayList<User> and modify it for the HashMap.

- keep the old static shallow and deep copy methods of ArrayList< InsurancePolicy> inside the InsurancePolicy but add these methods to InsurancePolicy:

ArrayList< InsurancePolicy> shallowCopy(HashMap< Integer, InsurancePolicy> policies)
ArrayList< InsurancePolicy> deepCopy (HashMap< Integer, InsurancePolicy> policies)
HashMap< Integer, InsurancePolicy> shallowCopyHashMap (HashMap< Integer, InsurancePolicy> policies)
HashMap< Integer, InsurancePolicy > deepCopyHashMap (HashMap< Integer, InsurancePolicy > policies)

- Comment the old code for shallow copy and deep copy methods inside the User and InsuranceCompany and modify them to work with HashMaps. Note that these methods still should return an ArrayList of policies and users. Add new versions of these methods to return HashMaps of policies and users.

- The sort methods inside the User and InsuranceCompnay still should return an ArrayList as HashMap is not sortable. Comment the old code and modify it to make a shallow copy of Policy and User HashMaps to ArrayLists first, and then sort the ArrayList and return it.

2) For data aggregation reports by using HashMap add following methods to the class User: (0.5 marks)

HashMap<String, Integer> getTotalCountPerCarModel () // which aggregates the number of policies the user owns for any given car model as <Car Model, Number of policies with that given model> and returns the hashmap.

HashMap<String,Double> getTotalPremiumPerCarModel () // which aggregates the total premium payments of policies the user owns for any given car model as a list of pairs of <Car Model, Total Premium Payment> and returns the HashMap

- Add proper report methods to the class User to generate this report from the aggregated hashmaps:

| Car Model | Total Premium Payment | Average Premium Payment |
|---|---|---|
| Toyota Camry 2018 | $10,050.00 | $1,050.00 |
| Nissan Dualis 2017 | $4,025.00 | $850.00 |

(Assume this user has ten policies with the Car Model of "Toyota Camry 2018" and five with Nissan Dualis 2017)

3- For data aggregation reports add following methods to the class InsuranceCompany: (1.25 marks)

HashMap<String, Double> getTotalPremiumPerCity() // which aggregates the total premium payments for users from any given city within the company as <City name, Total Premium Payments for all users from the given city> e.g., we have three users from Wollongong, two from Sydney, etc

HashMap<String, Integer> getTotalCountPerCarModel () // returns a list of pairs of (Car Model, Count) as a HashMap across all the users within the company (by calling the corresponding method inside User)

HashMap<String,Double> getTotalPremiumPerCarModel () // returns a list of pairs of (Car Model, Total Premium Payments) as a HashMap across all the users within the company (by calling the corresponding method inside User)

- Add proper report methods to the class InsuranceCompany to generate these reports from the aggregated hashmaps:

| City Name | Total Premium Payments |
|---|---|
| Wollongong | $100,500.00 |
| Sydney | $450,980.00 |

Assuming that there are 110 users with the city="Wollongong" and the total premium payments for all of their policies is $100,500

And the same as 2 but across all users:

| Car Model | Total Premium Payment | Average Premium Payment |
|---|---|---|
| Toyota Camry 2018 | $100,050.00 | $1,150.00 |
| Nissan Dualis 2017 | $40,025.00 | $750.00 |

4) Add try catch to any section of the code that the program uses Scanner to read from the console and catch the **InputMismatchException** to catch any wrong input (e.g., String entry for a number) and prompt user. Keep asking user to enter the proper data. Your program should not stop/crash but to continue normally after the catch. In particular modify your UI **option 2, 3, 4, and 8**. If admin enters a wrong data (such as a string for street number or car price) prompt the admin to enter the correct data type and keep asking for the correct data without crashing (By using a while loop and a Boolean success flag the same as lecture code). (0.5 marks)

5) Let's assume that Policy ID should be 6 digits starting with 3: (0.5 marks)

- Add a new Exception Class PolicyException with one field: int ID.
- In the Policy constructor if the given ID does not follow the required pattern:
  - **Generate a random ID with the given pattern 3xxxxxx**
  - Throw a PolicyException by passing the generated ID as an argument to PolicyException constructor.
  - The toString() method in PolicyException creates an error message " The Policy ID was not valid and a new ID (the generated ID) is generated by admin and assigned for the Policy".

In your UI option 3 and 4 catch this exception and print it. Again your program should continue normally after catching this exception. You need to create the ThirdParty or Comprehensive Policy with the user information but the generated ID and add it to the user policies.

6) Modify your test code for the modified code in this lab in particular calling all the data aggregation methods. (1.0 marks)

7) Add following options to your basic UI: (0.75 marks)

- Option 9 to report Car Model, Total Premium Payment, and Average Premium Payment for a given user

- Option 10 to report City Name, Total Premium Payments across all users

- Option 11 to report Car Model, Total Premium Payment, and Average Premium Payment across all users