DPIT121 - Object Oriented Design and programming

Assignment 2 (due in week 9)

Start from your lab 6 solution or modify the sample assignment 1 solution in course webpage as the base for assignment 2. Watch video lecture 6-1 for hints

This assignment has three levels: advanced (max mark is 12 out of 12), standard (max mark is 10 out of 12), or core (maximum 8 out of 12).

Core Level: maximum 8% if completed in week 9

Standard Level: maximum 10% if completed in week 9

Advanced Level: maximum 12% if completed in week 9

Core Level

Perquisite Requirements from Lab 4-6: (4 marks)

Double check that all the requirements from lab 4, 5, and 6 are satisfied. In particular if you missed any of these requirements you can do it now: ✓

1- All classes have copy constructor and have implemented Cloneable with proper clone() methods. ✓

2) ArrayList of policies in the User class and ArrayList of users in the InsuranceCompany are replaced with HashMaps and all methods are modified to work with HashMap as lab 5 specifications required. ✓ ✓

3- All 20 methods for shallow and deep copy of ArrayLists and HashMaps are done in InsurancePolicy, User, and InsuranceCompany classes:

Inside InsurancePolicy:

static ArrayList< InsurancePolicy> shallowCopy(ArrayList< InsurancePolicy > policies) ✓

static ArrayList< InsurancePolicy> deepCopy(ArrayList< InsurancePolicy> policies) ✓

static ArrayList< InsurancePolicy> shallowCopy (HashMap< Integer, InsurancePolicy > policies) ✓

static ArrayList< InsurancePolicy> deepCopy (HashMap< Integer, InsurancePolicy > policies) ✓

static HashMap< Integer, InsurancePolicy > shallowCopyHashMap (HashMap< Integer, InsurancePolicy > policies) ✓

static HashMap< Integer,InsurancePolicy > deepCopyHashMap(HashMap< Integer, InsurancePolicy > policies) ✓

Inside User:

static ArrayList< User> shallowCopy(ArrayList< User > users) ✓

static ArrayList< User> deepCopy(ArrayList < User> users) ✓

static ArrayList< User> shallowCopy (HashMap < Integer, User > users) ✓

static ArrayList< User> deepCopy (HashMap < Integer, User > users) ✓

static HashMap< Integer,User > shallowCopyHashMap(HashMap < Integer, User > users) ✓

static HashMap< Integer,User > deepCopyHashMap(HashMap < Integer, User > users) ✓

ArrayList<InsurancePolicy> deepCopyPolicies() // note that these methods are not static ✓

ArrayList<InsurancePolicy> shallowCopyPolicies() ✓

HashMap< Integer,InsurancePolicy > deepCopyPoliciesHashMap() ✓

HashMap< Integer,InsurancePolicy > shallowCopyPoliciesHashMap() ✓

Inside InsuranceCompany:

ArrayList <User> deepCopyUsers() // note that these methods are not static ✓

ArrayList <User> shallowCopyUsers() ✓

HashMap < Integer,User > deepCopyUsersHashMap() ✓

HashMap < Integer,User > shallowCopyUsersHashMap() ✓

4) MyDate, InsurancePolicy, and User have implemented Comparable interface with proper compareTo() methods. User also has a compareTo1() method to compare users based on the total premium payments. ✓

5) These sort methods are implemented in User and InsuranceCompany classes based on lab 4:

In User class : ArrayList<InsurancePolicy> sortPoliciesByDate() ✓

In InsuranceCompany: ArrayList<User> sortUsers() ✓

6) Data aggregation methods are done and tested properly based on lab 5: ✓

In User class :

HashMap<String, Integer> getTotalCountPerCarModel () ✓

HashMap<String,Double> getTotalPremiumPerCarModel () ✓

void reportPremiumCarModel (HashMap<String,Integer> counts, HashMap<String,Double> totals) ✓

In InsuranceCompany:

HashMap<String, Double> getTotalPremiumPerCity() ✓

HashMap<String, Integer> getTotalCountPerCarModel () ✓

HashMap<String,Double> getTotalPremiumPerCarModel() ✓

void reportPremiumCarModel(HashMap<String,Integer> counts,HashMap<String,Double> totals) ✓

void reportPremiumPerCity(HashMap< HashMap<String,Double> totals) ✓

7) Proper Exception Handling has been added to your code to catch **InputMismatchException, PolicyException, IOException, etc.** The program should not be crashed if any of these exceptions occur. ✓

8) All of the classes has implemented Serializable interface and all of them have proper toDelimitedString() method. Proper methods to save and load data in/from both binary files and text files are implemented as below: ✓

Inside InsurancePolicy:

static hashMap<Integer, InsurancePolicy> load (String fileName) //binary file using serializable ✓

static boolean save (HashMap<Integer, InsurancePolicy>, String fileName) ✓

static HashMap<Integer,InsurancePolicy> loadTextFile (String fileName)//using toDelimitedString ✓

static boolean saveTextFile (HashMap<Integer, InsurancePolicy>, String fileName) ✓

Inside User:

static hashMap <Integer, User> load (String fileName) //binary file using serializable ✓

static boolean save (HashMap<Integer, User>, String fileName) ✓

static HashMap<Integer,User> loadTextFile (String fileName)//using toDelimitedString ✓

static boolean saveTextFile (HashMap<Integer, User>, String fileName) ✓

Inside InsuranceCompany:

boolean load (String fileName) //load the InsuranceCompany including all policies/users from a binary file ✓

✓

boolean save (String fileName) ✓

✓

boolean loadTextFile (String fileName) ✓

boolean saveTextFile (String fileName)

***Test your filing with the code in filing_test.java located in “assignment II test materials” folder by adding it to your test code. You need to submit six data files (3 binary and 3 text files) with your submission.**

New Requirements (4 marks) (0.5 for each item)

1) Comment compareTo1() method in User class and add a proper Comparator class to compare User objects based on the total premium payments. It can be an inner class inside class InsuranceCompany which already has flatRate to ✓

calculate the total payment. Find resources for Comparator class and how to use it in Collections.sort() in Internet and study them. ✓

2) Add a method ArrayList<User> sortUsersByPremium() to the InsuranceCompany to sort all the users by using the new Comparator. ✓

3) ID generation for the User to be automatic by using a static count and increment it. ✓

4) Use regular expression inside InsurancePolicy constructor to check if the given policyHolderName has this pattern: First name starting with capital letter and minimum 2 alphabets following by space and last name (again starting with capital letter and minimum 2 alphabets), if not throw a user defined exception. Hint: Use java.util.regex package. You may find a tutorial here: https://www.javatpoint.com/java-regex. Note that you also has a PolicyException class for policyID. You need another exception for policyHolderName now. ✓ ✓

5) Add the following method to the class InsuranceCompany:

HashMap<String, ArrayList<User>> getUsersPerCity() // returns a hashmap with the city name as key and a list of all the users from that city as the value. ✓

6) Add a method to InsuranceCompany to get a date as input and returns a HashMap of user full name (The name of the user and not the policy's policyHolderName) as the key and for each name all the policies owned by that user and expired by that given date: ✓

HashMap <String,ArrayList<InsurancePolicy>> filterPoliciesByExpiryDate (MyDate expiryDate) ✓

7) Modify your test code from lab 4 to 6 to check all the new methods including shallow and deep copy of policies, users, and insurance company, sorting policies and users by calling the methods (both Comparable and Comparator), saving and loading the insurance company to/from a binary file, and text file, data aggregation methods, etc.

8) Modify your UI to include options for new functionalities of your software. You should have additional options for deep copy, shallow copy, sorting, data aggregations, saving/loading, etc

Standard Level (10 marks: extra 2 marks from Core)

1- Finish all the requirements for Core first. You need to have the test code to demonstrate everything is working before showing the demo of your UI.

2- You need to use proxy design pattern in your UI which means your class UI only has access to one object from InsuranceCompany class with the minimum or no dependency to other classes i.e., you don't create objects or directly call methods from InsurancePolicy and User in your UI. Use Exception handling to handle any errors resulting from the user inputs. All the methods with boolean return value should be checked in the UI with user friendly promoting. ✓ ✓

3- Add password to User class if you have not done it in assignment I. Add validation methods e.g., boolean userVaildate (String username, String password) if you haven't done this in assignment I. ✓

4- To be sure your design maintains the high level of security; all the methods inside InsuranceCompany and User should apply validation/authentication prior to do the given task. Hence you send the username and password as parameters ✓

to all methods for validation which returns false if validation is not successful e.g.:

boolean User.addPolicy (String username, String password, InsurancePolicy policy) ✓

double User. calcTotalPayment (String username, String password) ✓

boolean InsuranceCompany.addPolicy (String adminUsername, String adminPassword, int userID, InsurancePolicy policy) ✓

double InsuranceCompany.calcTotalPayment (String adminUsername, String adminPassword, int userID) ✓

Note that InsuranceCompany has access to all users and can call getUsername() and getPassword() to be able to access User methods if admin login details are provided. UI also stores the username and password for a session and send them to the methods when required.

Advanced Level (12 marks: extra 2 marks from Standard)

1- Study the Bank example from week 2 to understand that how a balance range is stored as an array and how the number of accounts per each range is reported. Add following methods to your code: ✓

User: int [] policyCount (String username, string password, int [] ranges) ✓
// gets a range array as input and returns the number of policies the user owns per range as output if user validation is successful (username and password is sent to the method for validation). Range array is the range of premium payment for policies in all following methods.

InsuranceCompany: int [] policyCount (String adminUsername, String adminPassword, int [] ranges) // gets a range array as input and returns the number of policies in the company per rang as output if validation is successful. ✓

InsuranceCompany: HashMap <String, Integer []> policyCityCount(String adminUsername, String adminPassword, int [] ranges) // gets a range array as input and returns the number of policies in the company per range per city. City is the HashMap key. ✓

InsuranceCompany: int[] userCount (String adminUsername, String adminPassword, int [] ranges) // gets a range array as input and returns the number of users in the company per range. ✓

InsuranceCompany: HashMap <String, Integer []> userCarModelCount (String adminUsername, String adminPassword, int [] ranges) // gets a range array as input and returns the number of users in the company per range per car model. Car model is the HashMap key. ✓

User: HashMap <String, Integer []> policyCarModelCount (String username, string password, int [] ranges) // gets a range array as input and returns the number of policies owned by this user per range per car model. Car model is the HashMap key. ✓

InsuranceCompany: HashMap <String, Integer []> policyCarModelCount (String adminUsername, String adminPassword, int [] ranges) // gets a range array as input and returns the number of policies in the company per range per car model. Car model is the HashMap key. You should call the corresponding method in User to do it for the InsuranceCompany. ✓

Check the test scenario (located in Assignment II test materials folder) to see how to test and validate these methods.