DPIT121 - Object Oriented Design and Programming

Lab 8 (due in week 11 Lab)

The purpose of this lab is to practice **functional programming, functional interface, and Lambda expression** in Java. You also have time to continue on the requirements of lab 7 GUI Design if you have not finished it last week. You need this for your assignment III when you add Admin Login in addition to user Login. Study Week 10 lecture codes in particular from A to Employee example before attempting the lab.

1) Be sure that your GUI from last week has all the tabs and menus working. You need to finish this before attempting Lambda and functional programming for lab 8.

2) You need to update following methods in InsurancePolicy class to use Stream, Lambda expression, and built-in functional interface. Comment the old code and add the new code. The code for some methods are provided to give you hints. Ask your tutor if you have any questions from these methods or lecture codes: (0.8 mark. 0.2 for each method highlighted in yellow)

```
public static void printPolicies(ArrayList<InsurancePolicy> policies)
{
                    policies.forEach(System.out::println);
        // or this: policies.forEach(x->x.print());
        // or this: policies.forEach(x->System.out.println(x));
}

public static double calcTotalPremiums(ArrayList<InsurancePolicy> policies, double flatRate)
{
                    return policies.stream()
                .map(x->x.calcPremium(flatRate))
                .reduce(0.0,(x,y)->x+y);
                    // or this :
                    return policies.stream()
                .mapToDouble(x->x.calcPremium(flatRate))
                .sum();
}
```

```
public static ArrayList<InsurancePolicy> filterByCarModel (ArrayList<InsurancePolicy> policies, String carModel)
{
                    return (ArrayList<InsurancePolicy>)(policies.stream()
        .filter(x->x.getCarModel().contains(carModel))
        .collect(Collectors.toList()));
}
```

Now convert the code for following methods to Lambda:

a) public static void carPriceRiseAll(ArrayList<InsurancePolicy> policies, double risePercent)

b) public static void ArrayList<InsurancePolicy> shallowCopy(ArrayList<InsurancePolicy> policies)

c) public static void ArrayList<InsurancePolicy> deepCopy(ArrayList<InsurancePolicy> policies)

d) public static ArrayList<InsurancePolicy> filterByExpiryDate (ArrayList<InsurancePolicy> policies, MyDate date)

It is optional to change the methods for HashMaps as they are very similar. The only change is to write hashmap.values().stream() to convert hashmap to stream and also to collect the streas as map and not list. Do it if you have times

3) Update these methods in the User class to use Lambda: (0.6 mark. 0.2 for each method)

a) public void printPolicies(double flatRate)

b) public static void ArrayList<User> shallowCopy(ArrayList<User> users)

c) public static void ArrayList<User> deepCopy(ArrayList<User> users)

4) Update these methods in the InsuranceCompany to use Lambda: (0.6 mark. 0.2 for each method)

a) public double calcTotalPremiums()

b) public void carPriceRise(double risePercent)

c) public ArrayList<InsurancePolicy> filterByCarModel(String carModel)

5) For data aggregation reports, in assignment I you used ArrayLists to generate the distinct list of your Item collections (e.g., list of car models or cities) and then aggregate the count or total per item. You modified your methods to use hashmaps for data aggregations in lab 5 and assignment II which was a more efficient method to aggregate the data. In this lab and later in assignment III you change your data aggregation methods to use functional programming and Lambda which uses multiple cores and SSE technology for parallel processing of the list which is even more efficient. Study the data aggregation reports in Employee example and then change the following method by commenting the old code and adding the data aggregation by using
Lambda. The rest of data aggregation methods should be changed to Lambda in Assignment III:

InsuranceCompany : HashMap<String, Double> getTotalPremiumPerCity() (0.6 mark)

6) Create a list of InsurancePolicys in your main by calling InsuranceCompany.allPolicies(). Then write pieces of test code by using Stream and Lambda to do the following tasks. See the example: (2.4 marks. 0.3 for each item)

Example: Search and filter all the policies with the carPrice<10000 and sort the filtered list based on numberOfClaims and print the sorted filtered list:

```
policies.stream()
        .filter(x->x.getCarPrice()<10000)
        .sorted(Comparator.comparing(InsuranceCompany::getNumberOfClaims)
        .forEach(System.out::println)
```

- Search for the name containing “John” (or similar pattern based on your test data) in the list of policies and displays the information for all the policies with the given name pattern.

- Search for the name containing “John” (or similar pattern based on your test data) in the list of policies and displays the total premiums for all the policies with the given name pattern.
- Find the first policy with the premium between $200 to $500 and display the name, ID and premium (use filter, findFirst, and map the same as Employee example)
- Find all policies with the premium between $200 to $500, sort them by ID and display the name, ID and premium for each policy.
- Calculate the total premium for all policies with the premium between $200 to $500.
- Write the following method in your program which has a predicate as input parameter and filters a list of policies:

  ArrayList<InsurancePolicy> filterPolicies(ArrayList<InsurancePolicy> policies, Predicate<InsurancePolicy> criteria)

  Call this methods as below:

  Predicate<InsurancePolicy> c1= x -> x.getPolicyHolderName().equals(“John Smith”);

  ArrayList<InsurancePolicy> policies1= filterPolicies(policies,c1);

  InsurancePolicy.printPolicies(policies1);

  And also more packed as:

  InsurancePolicy.printPolicies (filterPolicies(policies, x->x.getExpiryDate().getYear()==2020);

  InsurancePolicy.printPolicies (filterPolicies(policies, x->x.getCarModel().contains(“Toyota”);

- Call your filterPolicy method to filter a list of policies with Car Type equal to LUX. Then sort the filtered list based on car price and finally print the sorted filtered list.
- Aggregate (by using group by) the list of policies based on expiryDate year and print the report