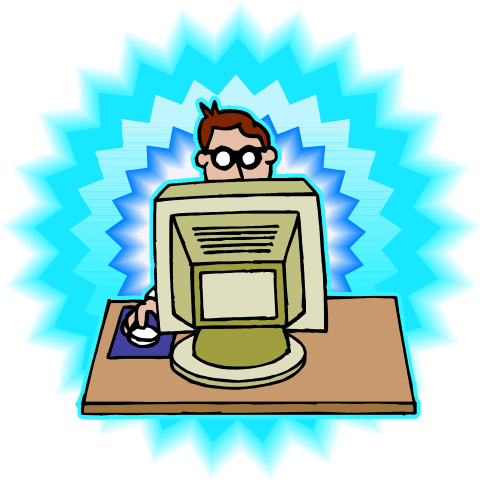
COMP 248

ASSIGNMENT 2



DESCRIPTION: In this assignment, you will have to write an application that uses the things that you learned in Chapters 3 and 4 (control flow statements and basic class definitions).

Your application will be used to provide the kind of information that one might see at an insurance company (a really bad one!). The idea is that your program will display a simple menu system that, in turn, allows the user to add new information and examine what has already been entered. (In our case, we can't actually store anything so your data disappears once the program ends. That's what databases are for).

In our environment, we will have two different types of insurance policies: car and home. The idea is that your program will record information for one car and one house plan. As you can probably guess, each of these plans should be associated with its own class. When a new plan is created, a new object will be generated and will then store information as specified by the user.

So how you do you interact with the system? In addition to the two insurance classes, there will be a "driver" application called **Insurance.java**. Basically, this is where the *main* method is located. Its job is to provide a menu system that allows one to add and query the objects. Like any good menu system, you should be able to move up and down through levels and keep circulating until you want to quit. Yes/no (or y/n) prompts should be provided as necessary.

In terms of the levels, the first menu should allow the user to select the appropriate option:

The Great Big Insurance Scam Company

a. Auto
b. Home

Select the insurance category (type q to quit):

The first screen will look something like this. If the user enters "q", then you should ask them to confirm with a y/n response. If yes, say "Thank you" and end. If no, the menu should be displayed again, with a new prompt. Note that in a real application, the screen would be cleared each time. However, you don't have to do that here. All menus can just scroll down the screen as necessary.

Depending on the option chosen, the user will be taken to either the Auto or Home insurance screen. For the Auto option, the following menu will appear:

a. Create new Auto policy
b. Display current policy
c. Update current policy

Select option (type m to return to main menu or q to quit)

The options should be fairly obvious in intent. If option **a** is selected, a new plan object will be created. It is basically empty at the time of creation, though default values may be set (discussed later). Option **b** will display all values for the policy in a neatly formatted manner. Please note that basic error checking is required here. If, for example, the user asks to display or update the policy but none has been created, then this information should be displayed on the screen. In addition, if a user selects an option in this screen (or any other screen) that is invalid, then again this should be captured and displayed to the screen.

If option **c** is chosen, then a third menu will be displayed. It will look like this:

a. Driver's Name
b. Driver's age
c. Car type
d. Number of driving citations

Select option (type m to return to main menu, u to go up one level, or q to quit)

Here, we enter basic information about the car and driver. Essentially, we are trying to calculate the risk for this person. Age and number of citations are both integers. They must be values that make sense (i.e., age is b/w 16 and 130, and citations are b/w 0 and 100). Car type can either be "sport", "sedan" or "truck". The policy will also store a "risk" value. It is calculated as a multiple of several values. Essentially, we multiply (number of citations + 1) x car type (sedan = 1, truck = 2, sport = 4) x age category (<20 = 5, b/w 20 and 50 = 2, and > 50 = 1). This value should be displayed with the others when the display option is chosen in the previous screen. In addition to the risk value itself, we should also see a risk level in the display: low (< 10), medium (10 to 100), and high (over 100).

When the given entry option is chosen, the user will enter an appropriate value and then the menu will be displayed again so that another option can be chosen. Again, error checking must be provided so that only reasonable values can be used. Any updates will replace existing values.

If the Home option is chosen, then the Create/Display/Update options will also appear in the second menu. Create and Display have the same meaning as they did for the Auto policy. If the Update option is chosen, then the following menu will appear:

a. Address
b. Room count
c. Fireplace?

d. Square footage

Room count and square footage are integers. Fireplace is a Boolean indicating whether the home has a fireplace. Again, we want to calculate (and later display) the risk. In this case, the calculation is as follows. We divide the square footage by the room count and multiply by the fireplace result (false = 1, true = 2). Risk can be low (<20), medium (20 to 200), and high (> 200).

What about address? Well, address is a little more complicated as it has several parts. It will consist of Street (String), Street number (int), City (String), and Country (String). While we could record all of these things as separate values within the Home policy class, that's messy. Instead, you will create an Address class that manages these things. An address object will then be stored within the policy just like any other variable.

THINGS TO KEEP IN MIND: You have now created an insurance policy application. In terms of the code, however, there are a number of things to keep in mind. First, structure is important. You should not be writing spaghetti code that jumps around like crazy. This is especially important with menus. It should be easy to follow the flow of control through various loops. Just because Java provides syntax to do certain things, that doesn't mean you should do it. In short, you should have obvious entry and exit points in your code. It should be easy for someone to look at your main driver application and basically see what is happening. Comments are also useful.

All classes should have their logic neatly divided into methods. In other words, you should not have one giant method that tries to do everything.

It should not be possible to directly access internal variables for the classes. There should be appropriate access methods for values that need to be displayed and set.

All classes should use a constructor. Constructors should either set default values (fireplace = false, Country = Canada) or simply provide explicitly initialization (usually to 0).

Each class should have a toString() method that provides a String representation of the object for display purposes.

DELIVERABLES: You will submit four source files: **Insurance.java**, **Auto.java**, **House.java**, and **Address.java**. No other files should be submitted! Each file will contain one class, with the same name as the file. Once you are finished writing your code, you will combine the four source files into a single zip file called:

A2_name_ID.zip

where "name" is your name and "ID" is your student ID. So if you your name is John Smith and your Student ID is 12345, then you would create a zip file called

A2_John_Smith_12345.zip

The file should be uploaded to the Moodle server via the Assignment 2 link on the web page. You are free to submit multiple versions (new versions just overwrite the last one) until the deadline. You may still submit after the deadline (up to two days late) but the server will record your submission time and an appropriate penalty will be assessed (as per the syllabus). Absolutely NO submission will be accepted by email.

As always, it is your responsibility to verify that your code has been properly submitted.

Good Luck!