Group Assignment 2

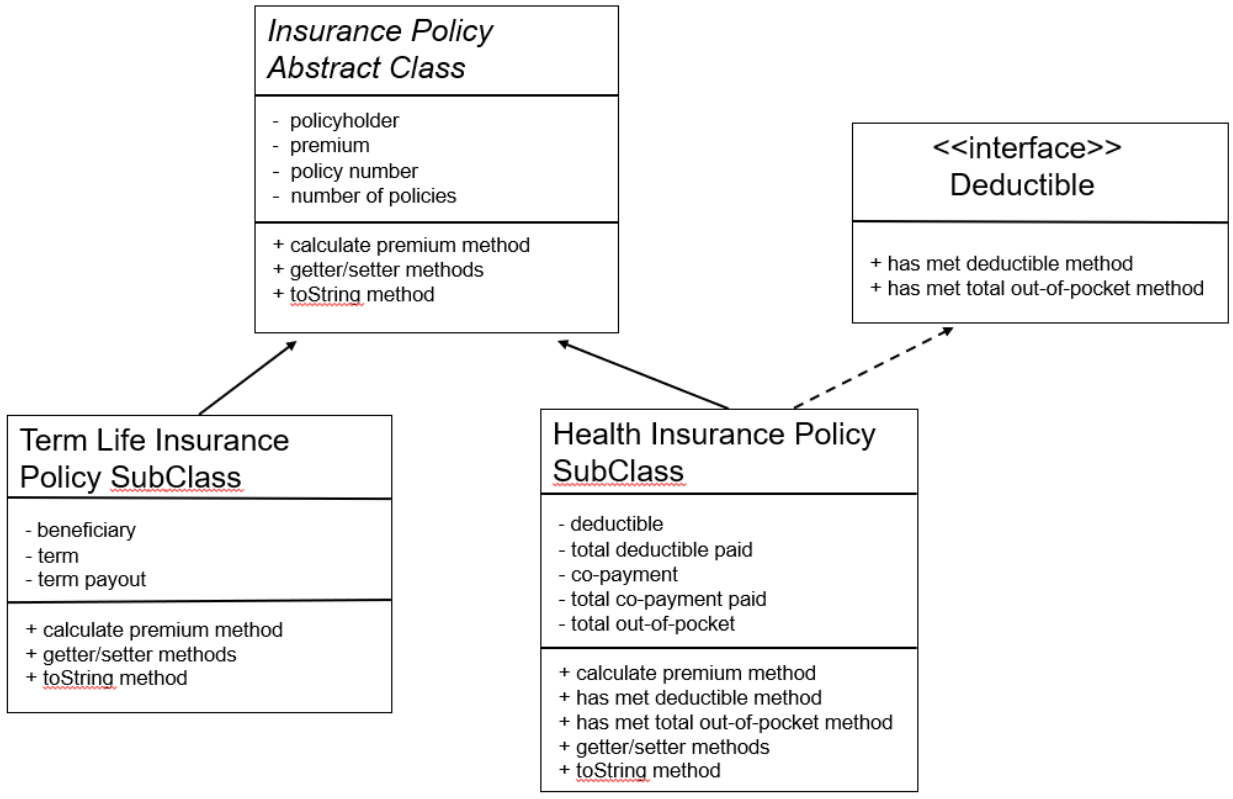
Background

Topics

- Interface
- Abstract Class
- Polymorphism
- Casting of objects
- Super/Base
- ArraysList
- Static variable/method
- Method Overriding
- ToString
- Insurance products/terminology

Description:

For this assignment, you will code a basic insurance policy management system, which is depicted in the following UML (unified modeling language) diagram:



Unified Modeling Language (UML) Diagram

Group Assignment 2

The assignment is designed to emphasize the topics covered in the previous week's classes and attempts to bring everything together. Further, you are expected to research insurance products to understand the necessary industry terminology and how insurance products work overall.

Instructions

For this assignment, you will implement an interface, an abstract class, two subclasses, and a driver class, utilizing the UML diagram, the fundamentals of inheritance, polymorphism, casting, and ArrayLists.

Define the following:

1) an interface that represents an insurance deductible
    a) must have an abstract method to determine if the deductible has been met, which take no parameters and returns a boolean value indicating whether the deductible has been met
    b) must have an abstract method to determine if the total out-of-pocket amount has been met, which take no parameters and returns a boolean indicating whether the total out-of-pocket amount has been met

2) an abstract class that represents an insurance policy
    a) must have the following attributes:
        - string, private, policyholder's full name
        - int, private, unique policy number (created using **static** attributed below. Hint: increment the static attribute first and then assign its value to the policy number)
        - int, **static**, private, initialized with zero (0), total number of policies created so far (Hint: increment the static attribute first and then assign its value to the policy number)
        - double, private, initialized with zero (0), premium amount
    b) constructor which:
        - takes in a string and assigns its value to the policyholder attribute
        - increments the total number of policies attribute by one (1) and then assigns its value to the policy number attribute (i.e., the creation of each object increments the total number of policies attribute, which is then used as the policy number)
    c) must have getter/setter methods for all attributes
    d) must have an abstract method that calculates the premium, which takes no parameters and returns no value

e) must override the tostring method to return a string which includes a label and value for all attributes

3) a sub-class that represents a term-life insurance policy
   a) must inherit the abstract class
   b) must have the following attributes:
      - string, private, beneficiary full name
      - int, private, policy term
      - double, private, term payout
   c) constructor which:
      - takes in a string and assigns it to the policyholder attribute via a call to the parent constructor
      - takes in a string and assigns it to beneficiary attribute
      - takes in an int and assigns it to term attribute
      - takes in a double and assigns it to the term payout attribute
   d) must have getter/setter methods for all attributes
   e) must override the abstract method, which takes no parameters, returns no value, and calculates the premium value by dividing the amount of the term payout by (twelve (12) times the term)
   f) must override the tostring method to return a string which includes a label and value for all attributes

4) a sub-class that represents a health insurance policy
   a) must inherit the abstract class and the interface
   b) must have the following attributes:
      - double, private, amount of deductible
      - double, private, initialized with zero (0), total deductible paid so far
      - double, private, amount of co-payment
      - double, private, initialized with zero (0), total co-payment paid so far
      - double, private, total out-of-pocket amount
   c) constructor which:
      - takes in a string and assigns it to the policyholder attribute via a call to the parent constructor
      - takes in a double and assigns it to the deductible attribute
      - takes in a double and assigns it to the co-payment attribute
      - takes in a double and assigns it to the total out-of-pocket attribute
   d) must have getter/setter methods for all attributes
   e) must override the abstract method that calculates the premium, which takes no parameters, returns no value, and calculates the premium by dividing the deductible by twelve (12)

f) must override the abstract method that determines if the deductible has been met, which takes no parameters, returns a boolean value, and determines whether the deductible has been met by the following:
   - if the total deductible paid so far is greater than or equal to the deductible amount, the policyholder has met the deductible. Otherwise, the policyholder has not met the deductible.

g) must override the abstract method that determines if the total out-of-pocket amount has been met, which takes no parameters, returns a boolean value, and determines whether the total out-of-pocket amount has been met by the following:
   - if the total deductible amount paid so far plus the total co-payment amount paid so far is greater than or equal to the total out-of-pocket amount, the policy holder has met the total out-of-pocket amount. Otherwise, the policy holder has not met the total out-of-pocket amount

h) must override the tostring method to return a string which includes a label and value for all attributes

Define the following driver class:

- create a list of customer insurance policies (i.e., an ArrayList of insurance objects) *Note: you must create **only one** ArrayList, which will contain both types of insurance objects, i.e., term-life and health insurance objects.*

- using a loop, prompt the user with the following menu, read in the user's input, and populate the insurance list:

```
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
```

  - if the user enters 1, prompt the user and read in the data needed to create a health insurance policy object, and add the object to the insurance policy list
  - if the user enters 2, prompt the user and read in the data needed to create a term-life insurance policy object, and add the object to the insurance policy list
  - if the user enters 3, terminate the loop
  - if the user enters **any numeric or non-numeric character** other than a 1, 2, or 3, the following error message should display: Error: Please enter valid input, and the user

should be allowed to reenter a valid choice. (Hint: your code must test for numeric and non-numeric input)

- After the above loop ends, print the contents of the insurance list (dollar amounts must be displayed using two decimal points):
  - traverse the entire list
  - for all objects in the list:
    - calculate the policy premium
    - print the policyholder, policy number, and premium
    - if the object is a health insurance policy, also print whether the deductible has been met and whether the total out-of-pocket amount has been met

- **See Example Screenshots below for additional information**

Submission Instructions

All five classes/subclasses below must be created, used, and submitted for a grade. Your .java files should be saved as .txt files and uploaded to the D2L dropbox before the deadline.

1. deductable.txt
2. insurance.txt
3. termLife.txt
4. health.txt
5. InsuranceTest.txt

Note: points will be deducted if all five files are not submitted or if multiple files are combined in one document.

Grading Criteria

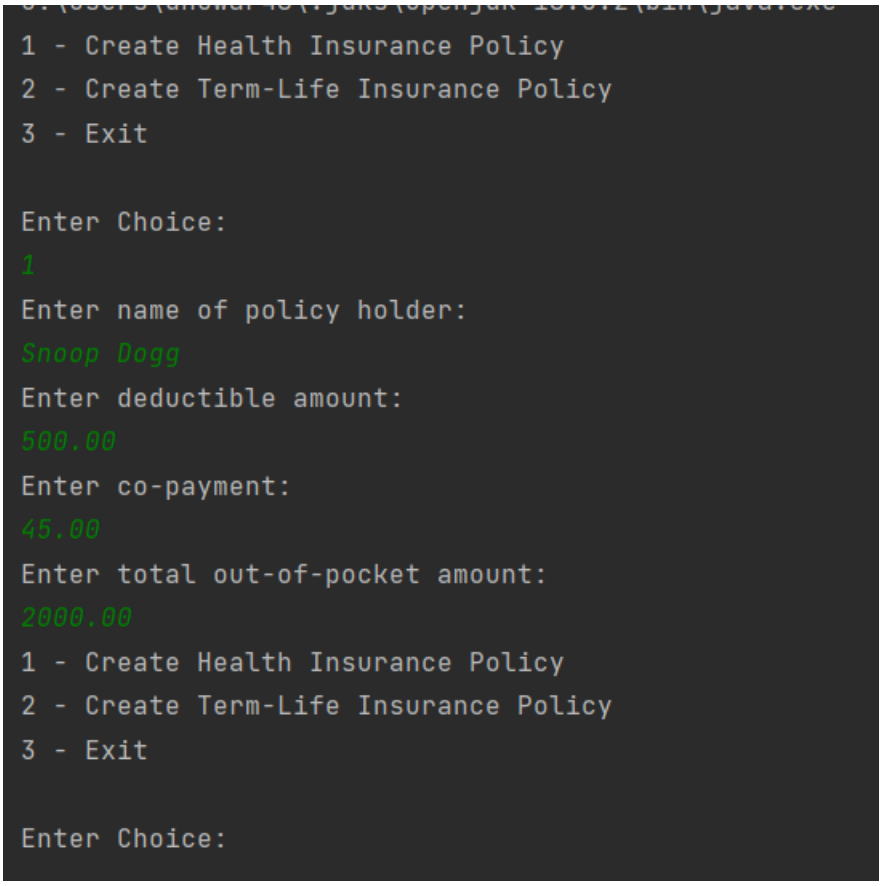
- Successfully compiles (5%)
- Indentation/spacing, comments, and naming – (5%)
- Deductible Interface (10%)
- Insurance Policy Abstract Class (20%)
- Term Life Insurance Policy SubClass (20%)
- Health Insurance Policy SubClass (20%)
- Driver Class (20%)

Example Screenshots

Screenshot 1 – User entered invalid numeric and non-numeric data

```
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
k
Error: Please enter valid input

1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
9
Error: Please enter valid input

1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
```

Screenshot 2 – Create a health insurance policy



Screenshot 3 – Create a term-life insurance policy

```
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
2
Enter name of policy holder:
Dr. Dre
Enter name of beneficiary:
Mary J
Enter number of years in term:
30
Enter amount of payout:
50000.00
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
```

Screenshot 4 – Create a term-life insurance policy

```
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
2
Enter name of policy holder:
Don Duck
Enter name of beneficiary:
Sylvester D. Cat
Enter number of years in term:
25
Enter amount of payout:
30000.00
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
```

Screenshot 5 – Create a health insurance policy

```
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
1
Enter name of policy holder:
Captain America
Enter deductible amount:
1000.00
Enter co-payment:
30.00
Enter total out-of-pocket amount:
5000.00
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
```

Screenshot 6 – Exit the menu, Print the report, and Terminate the application

```
1 - Create Health Insurance Policy
2 - Create Term-Life Insurance Policy
3 - Exit

Enter Choice:
3
Policyholder: Snoop Dogg
Policy Number: 0
Premium: 41.67
Met Deductible: false
Met Total Out-Of-Pocket: false

Policyholder: Dr. Dre
Policy Number: 1
Premium: 138.89

Policyholder: Don Duck
Policy Number: 2
Premium: 100.00

Policyholder: Captain America
Policy Number: 3
Premium: 83.33
Met Deductible: false
Met Total Out-Of-Pocket: false


Process finished with exit code 0
```