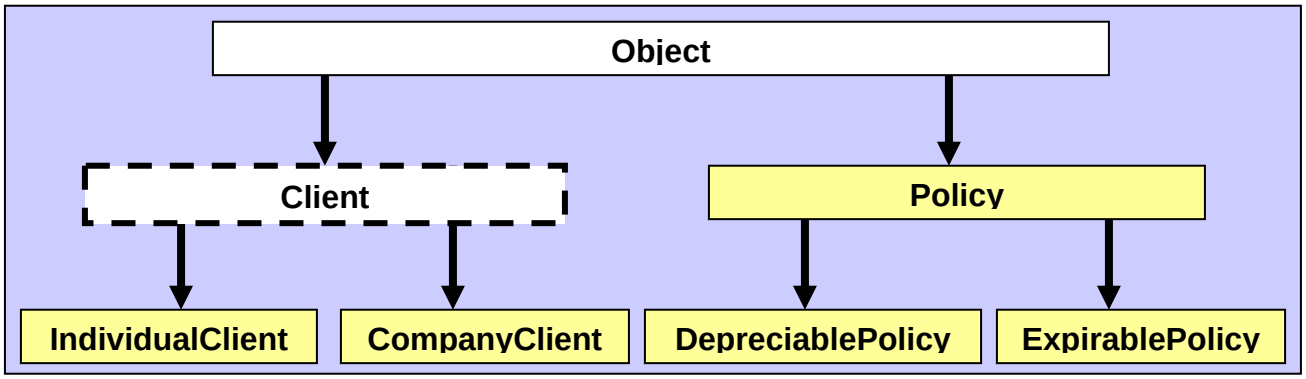
COMP1406 - Assignment #3

(Due: **Thursday, February 7th @ 9:00pm)**

In this assignment, you will begin to build a very simple insurance company system. You will gain practice with inheritance and overloading methods. The insurance system will keep track of the insurance company's clients. The hierarchy that we will use is shown here, although we will only look at the Policy class with its subclasses and the Client class. We will continue coding the system in the next assignment.



(1) The **Policy** Class

Define a class called **Policy** which has an **amount** and a **policyNumber**. Use a static variable as a counter so that the first policy created has policyNumber 1, the second has number 2, etc... Create a method called **isExpired()** which always returns **false**. Create a **print()** method that prints the policy with the following format, but does not print any carriage returns:

```
Policy #1 with amount: $320.0
```

Now test your code with this program:

```
public class PolicyTestProgram1 {
      public static void main(String args[]) {
            Policy p1 = new Policy(320);
            Policy p2 = new Policy(500.14f);
            Policy p3 = new Policy(0);
            p1.print(); System.out.println();
            p2.print(); System.out.println();
            p3.print(); System.out.println();
            System.out.println(p1.isExpired());
      }
}
```

Here is the expected output ... make sure it works before continuing:

```
Policy #1 with amount: $320.0
Policy #2 with amount: $500.14
Policy #3 with amount: $0.0
false
```

(2) The **DepreciablePolicy and ExpirablePolicy** Classes

Now we will make some subclasses to represent different types of policies. Define a class called **DepreciablePolicy** as a subclass of **Policy** which has an additional attribute called rate of type **float** which represents the rate at which the policy depreciates each time a claim is made. For example, a rate of 0.10 means that the value of the policy (i.e., the amount) depreciates 10% each time a claim is made (we will discuss the making of claims in the next section). Implement these:

- a constructor which takes a **float** representing the amount and another **float** representing the rate. This constructor should use inheritance by calling the constructor from the superclass to set the amount and policyNumber.

- a method called **isExpired()** which returns **true** if the amount of this policy has depreciated to 0, otherwise it should return **false**.

- a method called **depreciate()** which reduces the amount of the policy by the rate percentage. For example, if the amount is $100 and the rate is 0.10, then the amount after this method is called once should be $90.

- a **print()** method that prints the policy with the following format:

```
DepreciablePolicy #1 with amount: $320.0 rate: 10.0%
```

  You MUST make use of inheritance by calling the **print()** from the superclass.

Define a **ExpirablePolicy** class as a subclass of **Policy** which has an attribute called expiryDate of type **Date** that contains the date after which the policy is invalid. The **Date** class is in the **java.util** package, so import **java.util.*** at the top of your code. Implement these:

- a constructor which takes a **float** representing the amount and a **Date** representing the expiryDate. This constructor should use inheritance by calling the constructor from the superclass to set the amount and policyNumber.

- a constructor which takes a **float** representing the amount. This constructor should use inheritance by calling the constructor from the superclass to set the amount and policyNumber. The expiryDate should then be set to exactly one year after the policy was created. Here is how you can do this (dates are discussed at the end of the course):

```
GregorianCalendar aCalendar = new GregorianCalendar();  // makes today's date
aCalendar.add(Calendar.YEAR,1);
expiryDate = aCalendar.getTime();
```

- a method called **isExpired()** which returns **true** if the computer's current date is AFTER the expiryDate and **false** otherwise. (Hint: the **Date** class has methods **before(Date d)** and **after(Date d)**). Also, **new Date()** returns today's date.

- a **print()** method that prints the policy with the following format:

```
ExpirablePolicy #2 with amount: $2000.0 expires: Thu Jan 02 00:00:00 EST 2025
```

  You MUST make use of inheritance by calling the **print()** from the superclass.

Now test your new classes with this test program:

```
import java.util.*;

public class PolicyTestProgram2 {
   public static void main(String args[]) {
      DepreciablePolicy    p1;
      ExpirablePolicy      p2, p3;

      // Make a DepreciablePolicy
      p1 = new DepreciablePolicy(500.1f, 0.1f);
      p1.print(); System.out.println();

      // Now depreciate it a little
      p1.depreciate();
      p1.print(); System.out.println();

      // Make another Expirable Policy
      p2 = new ExpirablePolicy(2000, new GregorianCalendar(2025, 0, 2).getTime());
      p2.print(); System.out.println();
      System.out.println(p2.isExpired());

      // Make yet another Expirable Policy
      p3 = new ExpirablePolicy(2500, new GregorianCalendar(1997, 3, 1).getTime());
      p3.print(); System.out.println();
      System.out.println(p3.isExpired());
   }
}
```

Here is the output:

```
DepreciablePolicy #1 with amount: $500.1 rate: 10.0%
DepreciablePolicy #1 with amount: $450.09 rate: 10.0%
ExpirablePolicy #2 with amount: $2000.0 expires: Thu Jan 02 00:00:00 EST 2025
false
ExpirablePolicy #3 with amount: $2500.0 expires: Tue Apr 01 00:00:00 EST 1997
true
```

---

(3) The **Client** class

Define a class called **Client** which has a name and an id. Use a static variable as a counter so that the first client created has id 1, the second has id 2, etc... Add an attribute called policies that represents an array (maximum size 10) of all of a client's **Policy** objects, also add a numPolicies attribute to represent the number of policies that the client currently has. Implement the following in the **Client** class:

- a constructor which takes a single String parameter and uses it to set the name variable. The constructor sets the id such that each created client has a unique number and also updates your static variable appropriately. It should also set the policies list to be a new empty array of the appropriate size. The current number of policies should also be set accordingly.

- a method called **totalCoverage()** which returns a **float** containing the total amount of coverage of all the client's policies.

- a method called **openPolicyFor(**float **amt)** which creates a new **Policy** for the amount specified in the parameter, adds it to the list of the client's policies (if not exceeding the maximum) and returns the new **Policy** object (or **null** if the maximum was reached).

- a method called **openPolicyFor(**float **amt,** float **rate)** which creates a new **DepreciablePolicy** for the amount and rate specified in the parameters, adds it to the list of the client's policies and returns the new **DepreciablePolicy** object (or **null** if the maximum was reached).

- a method called **openPolicyFor(**float **amt,** Date **expire)** which creates a new **ExpirablePolicy** for the amount and expiry date specified in the parameters, adds it to the list of the client's policies and returns the new **ExpirablePolicy** object (or **null** if the maximum was reached). Notice that these 3 open methods are examples of *overloading* because they use the same method name.

- Create a **print()** method that prints the client with the following format, but does not print any carriage returns:

```
Client: Bob B. Pins with 2 policies
```

Now test what you have done so far with this program:

```
import java.util.*;

public class ClientTestProgram {
   public static void main(String args[]) {
      Client c = new Client("Bob B. Pins");
      c.print(); System.out.println();
      c.openPolicyFor(100);
      c.openPolicyFor(200, 0.10f);
      c.openPolicyFor(300, new GregorianCalendar(2005, 0, 2).getTime());
      c.print(); System.out.println();

      System.out.println("Policies:");
      for (int i=0; i<c.numPolicies; i++) {
         System.out.print("  ");
         c.policies[i].print();
         System.out.println();
      }
      System.out.println("Total Coverage: " + c.totalCoverage());
   }
}
```

Here is the expected output:

```
Client: Bob B. Pins with 0 policies
Client: Bob B. Pins with 3 policies
Policies:
  Policy #1 with amount: $100.0
  DepreciablePolicy #2 with amount: $200.0 rate: 10.0%
  ExpirablePolicy #3 with amount: $300.0 expires: Sun Jan 02 00:00:00 EST 2005
Total Coverage: 600.0
```

NOTE: Submit all **.java** and **.class** files needed to run. You **MUST NOT use packages** in your code, **nor projects**. Submit ALL of your files in one folder such that they can be opened and compiled individually in JCreator. Some IDEs may create packages and/or projects automatically. You **MUST** export the .java files and remove the package code at the top if it is there. Do NOT submit JCreator projects either. **JUST SUBMIT the JAVA and CLASS FILES**. Note that if your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

Please NOTE that you WILL lose marks on this assignment if any of your files are missing. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.
