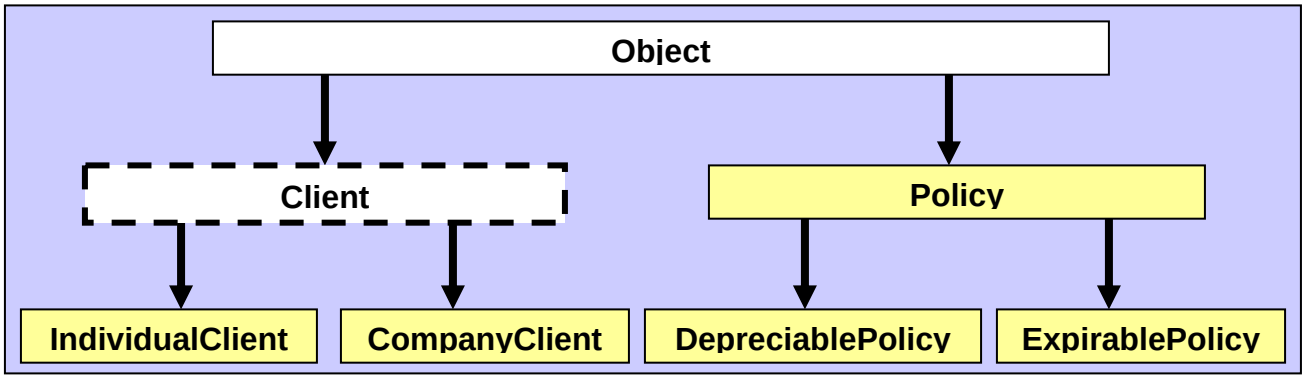
COMP1406 - Assignment #4

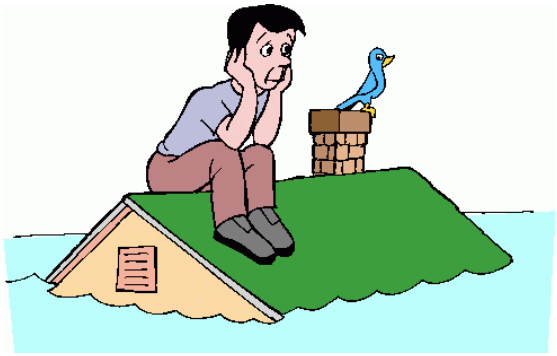
(Due: **Thursday, February 14th @ 9:00pm)**

In this assignment, you will continue your work from the previous assignment by adding two subclasses of Client. You will get practice using an *abstract* class with *abstract methods* and also practice writing *overriding* methods. Finally, we will make a window with components on it, but you WILL NOT get the window to work at this time … we are only interested in designing a layout for the window.



(1) The **IndividualClient** and **CompanyClient** Subclasses

Make the **Client** class **abstract**. Now create two subclasses of **Client**, one called **IndividualClient** and the other called **CompanyClient**. Try compiling them. The code will not compile because these subclasses do not inherit constructors and java requires a constructor to be available. You will have to create a constructor (taking a single String argument representing the client name) for each of these subclasses. The constructors should call the superclass constructor.



Alter your **ClientTestProgram** by changing the first line to be:

```
CompanyClient c = new CompanyClient("Bob B. Pins");
```

Re-run the test. As you did with the **Policy** subclasses, write **print()** methods in the subclasses to display the type of client ... for example: `CompanyClient: Bob B. Pins with 0 policies`

- Now define an **abstract** method in the **Client** class called **makeClaim(**int **polNum)** that returns a **float**. After re-compiling the **Client** class, try recompiling the **IndividualClient** and **CompanyClient** classes. Do you understand the error message ?

- Write a **getPolicy(int policyNumber)** method in the **Client** class that returns the **Policy** object that has the given **policyNumber**. If the **Client** does not have a policy with this number, return **null**.

- Implement the **makeClaim(**int **polNum)** method in the **IndividualClient** class as follows. First check to make sure that the policy with the given number has not expired. If it has, then return 0. If the claim is being made for a **DepreciablePolicy**, then you must make sure to depreciate the policy. You should depreciate the policy BEFORE returning the claim amount. This method should return the amount of the policy if the policy was indeed valid, otherwise 0 is returned.

We will now write a similar method in the **CompanyClient** class. However, we will make use of the double dispatching technique. Before we do this, we will have to go write a method called **handleClaim()** in each of the policy classes so that we can make use of them here.

- In the **Policy** class, implement **handleClaim()** so that it returns the amount of the claim.

- In the **DepreciablePolicy** class, implement **handleClaim()** so that it returns the amount of the claim but also depreciates the claim. Note that the amount returned is BEFORE depreciating (note that this is the opposite from IndividualClients which depreciated before the amount was returned). This method *overrides* the one in **Policy**.

- In the **ExpirablePolicy** class, implement **handleClaim()** so that it returns the amount of the claim as long as the policy has not yet expired, otherwise it returns 0. This method *overrides* the one in **Policy**.

Now go back to the **CompanyClient** class and implement the **makeClaim(**int **polNum)** method so that it first checks to make sure the policy number is valid and then uses the double-dispatching technique by calling the **handleClaim()** method. Note that you MUST NOT use any IF statements to determine the policy type ... as that would undo the advantages of double-dispatching.

Test your code with the following program:

```
import java.util.*;

public class ClientTestProgram2 {
	public static void main(String args[]) {
		// Create an individual client, open some policies and then make some claims
		IndividualClient ic = new IndividualClient("Bob B. Pins");
		ic.openPolicyFor(100);
		ic.openPolicyFor(200, 0.10f);
		ic.openPolicyFor(300, new GregorianCalendar(2025, 0, 2).getTime());
		ic.openPolicyFor(400, new GregorianCalendar(1999, 5, 4).getTime());
		Policy p = new Policy(500);
		System.out.println("Here are the Individual Client's policies:");
		System.out.println("Policies:");
		for (int i=0; i<ic.numPolicies; i++) {
			System.out.print("  ");
			ic.policies[i].print();
			System.out.println();
		}
		System.out.println("Making claims:");
		System.out.println("Claim for policy 0001: " + ic.makeClaim(1));
		System.out.println("Claim for policy 0002: " + ic.makeClaim(2));
		System.out.println("Claim for policy 0003: " + ic.makeClaim(3));
		System.out.println("Claim for policy 0004: " + ic.makeClaim(4));
		System.out.println("Claim for policy 0005: " + ic.makeClaim(5));
		System.out.println("Here are the Individual Client's policies after claims:");
		System.out.println("Policies:");
		for (int i=0; i<ic.numPolicies; i++) {
			System.out.print("  ");
			ic.policies[i].print();
			System.out.println();
		}

		// Create a company client, open some policies and then make some claims
		CompanyClient cc = new CompanyClient("The Pillow Factory");
		cc.openPolicyFor(1000);
		cc.openPolicyFor(2000, 0.10f);
		cc.openPolicyFor(3000, new GregorianCalendar(2025, 0, 2).getTime());
		cc.openPolicyFor(4000, new GregorianCalendar(1999, 5, 4).getTime());
```

```
            System.out.println("\nHere are the Company Client's policies:");
            System.out.println("Policies:");
            for (int i=0; i<cc.numPolicies; i++) {
                  System.out.print("  ");
                  cc.policies[i].print();
                  System.out.println();
            }
            System.out.println("Making claims:");
            System.out.println("Claim for policy 0006: " + cc.makeClaim(6));
            System.out.println("Claim for policy 0007: " + cc.makeClaim(7));
            System.out.println("Claim for policy 0008: " + cc.makeClaim(8));
            System.out.println("Claim for policy 0009: " + cc.makeClaim(9));
            System.out.println("Claim for policy 0005: " + cc.makeClaim(5));
            System.out.println("Here are the Company Client's policies after claims:");
            System.out.println("Policies:");
            for (int i=0; i<cc.numPolicies; i++) {
                  System.out.print("  ");
                  cc.policies[i].print();
                  System.out.println();
            }
      }
}
```

Here is the expected output:

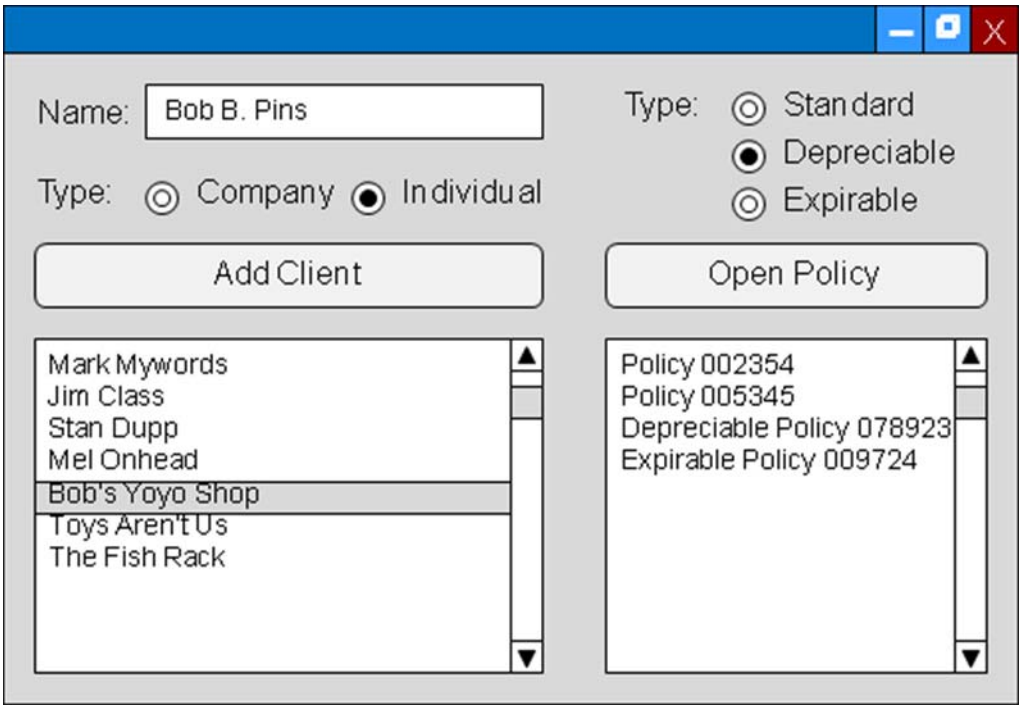
```
Here are the Individual Client's policies:
Policies:
  Policy #1 with amount: $100.0
  DepreciablePolicy #2 with amount: $200.0 rate: 10.0%
  ExpirablePolicy #3 with amount: $300.0 expires: Thu Jan 02 00:00:00 EST 2025
  ExpirablePolicy #4 with amount: $400.0 expires: Fri Jun 04 00:00:00 EDT 1999
Making claims:
Claim for policy 0001: 100.0
Claim for policy 0002: 180.0
Claim for policy 0003: 300.0
Claim for policy 0004: 0.0
Claim for policy 0005: 0.0
Here are the Individual Client's policies after claims:
Policies:
  Policy #1 with amount: $100.0
  DepreciablePolicy #2 with amount: $180.0 rate: 10.0%
  ExpirablePolicy #3 with amount: $300.0 expires: Thu Jan 02 00:00:00 EST 2025
  ExpirablePolicy #4 with amount: $400.0 expires: Fri Jun 04 00:00:00 EDT 1999

Here are the Company Client's policies:
Policies:
  Policy #6 with amount: $1000.0
  DepreciablePolicy #7 with amount: $2000.0 rate: 10.0%
  ExpirablePolicy #8 with amount: $3000.0 expires: Thu Jan 02 00:00:00 EST 2025
  ExpirablePolicy #9 with amount: $4000.0 expires: Fri Jun 04 00:00:00 EDT 1999
Making claims:
Claim for policy 0006: 1000.0
Claim for policy 0007: 2000.0
Claim for policy 0008: 3000.0
Claim for policy 0009: 0.0
Claim for policy 0005: 0.0
Here are the Company Client's policies after claims:
Policies:
  Policy #6 with amount: $1000.0
  DepreciablePolicy #7 with amount: $1800.0 rate: 10.0%
  ExpirablePolicy #8 with amount: $3000.0 expires: Thu Jan 02 00:00:00 EST 2025
  ExpirablePolicy #9 with amount: $4000.0 expires: Fri Jun 04 00:00:00 EDT 1999
```

(2) An Interface

Now you will create the "view (or "look") of a user interface, but you will not make the interface work at this time. The main purpose is to help you understand how to arrange components on a window.

Create a class called **InsurancePolicyApp** which creates and displays the 800 x 600 pixel window shown in the drawing on the next page. The window has two JLists in JScrollPanes which display a list of clients and a list of policies. When the window first opens, these lists should have the information as shown. There are also two JButtons ... **Add Client** and **Open Policy**. There are also two ButtonGroups of JRadioButtons. Ensure that only one button is selectable in each group. There is one JTextField that allows the user to enter a name. Finally, there are three JLabels ("Name:", "Type:" and "Type:") which help to explain what the components do. You will be marked on how "**nice**" your window looks, so make sure to line up everything neatly. NOTE: YOU DO NOT HAVE TO MAKE THE WINDOW WORK ... AT THIS POINT WE ARE JUST INTERESTED IN HAVING THE WINDOW APPEAR WITH THE PROPER COMPONENT ARRANGEMENT.



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.
