COURSEWORK ASSIGNMENT **UNIVERSITY OF EAST ANGLIA**
School of Computing Sciences

MODULE :	CMP-4008Y Programming 1
ASSIGNMENT TITLE :	Module Assessment Exercises 2

DATE SET	: 12/02/16	
DATE OF SUBMISSION	: 10/03/16	
RETURN DATE	: 13/04/16	
ASSIGNMENT VALUE	: 30%	
SET BY	: Geoff McKeown	**SIGNED:**
CHECKED BY	:	**SIGNED:**

Aim:

Subject specific

Testing of the principles of Java programming. Implementation of a Java application involving file input and output.

Transferable skills

Computer programming; software testing.

Learning outcomes:

Subject specific

Development of basic competence in object oriented/algorithmic programming in Java, familiarity with the associated concepts and terminology.

Assessment criteria:

Part marks are shown on the exercise sheet.

Description of the Assignment

The purpose of this coursework is to develop a Java application that models certain aspects of an Insurance company. The company has a number of clients, to each of whom it has issued one or more insurance policies. The company offers several different kinds of policy, depending on what is insured:

- a building (typically, the policy holder's home);
- the contents of a property;
- a car; or
- the life of the policy holder.

You are required to write Java code for the following classes:

- `Name`
- `Address`
- `Policy`
- `BuildingPolicy`
- `ContentsPolicy`
- `CarPolicy`
- `LifePolicy`
- `PolicyList`
- `ClientDetails`
- `ClientDetailsList`
- `InputData`
- `InsuranceCoDemo`

1. The `Policy` class is to be an abstract class. It has two fields, one for a policy no. and the other for the year of issue of the policy.

   Each policy no. is a string of 9 characters, the first character of which indicates the type of policy:

   - B for a building policy,
   - C for a contents policy,
   - L for a life policy,
   - V for a car (vehicle) policy.

   Each of the remaining 8 characters of a policy no. is a decimal digit, 0 .. 9.

The company has no policy issued before 1990. A single constructor should be given with a parameter for each field. Accessor (get) methods should be given for both fields and a `toString` method should be defined. In addition, an abstract method called `getPremium()` should be given. This method has return type `double`, representing the annual cost (premium) of the policy. The constructor should throw an `IllegalPolicyException` if the parameter for the year of issue is not within 1990 - 2013 or if the policy no. parameter does not have the required format (see above).

The `IllegalPolicyException` is defined by the following class, a copy of which is on BlackBoard. You must add this class to your NetBeans project.

```
public class IllegalPolicyException extends Exception
{

    /** Creates an IllegalPolicyException with a given message */
    public IllegalPolicyException(String message) {
        super(message);
    }
}
```

[15 marks]

2. `BuildingPolicy`, `ContentPolicy`, `CarPolicy` and `LifePolicy` are all sub-classes of `Policy`. Each of these classes is to have its own implementation of the `getPremium` method defined as abstract in `Policy`. In addition, all four classes will have constructors, appropriate `get` (accessor) and `set` (mutator) methods for each field and a `toString()` method. Constructors and `set` methods may throw
`IllegalPolicyExceptions` with suitable messages. Use the `StringBuilder` class in your `toString` methods.

   - In addition to the inherited fields, `BuildingPolicy` has two further fields, both of type `double`, the first of which is for the estimated cost of re-building the property being insured. The second field represents the risk of insuring the property (e.g. risk from flood damage etc) and should be a value between 0 and 1, inclusive. Here, 0 is lowest risk and 1 is highest risk. For a building with estimated cost of re-building, `c`, and risk, `r`, the insurance premium is calculated by the formula

     $$\texttt{premium} = \texttt{c} \times \texttt{REBUILD\_FACTOR} \times (1 + \texttt{r}),$$

     where `REBUILD_FACTOR` is an appropriately set class constant (a typical value for this constant might be 0.001). [10 marks]

   - `ContentPolicy` also has two fields of type `double` in addition to the inherited fields: the first additional field is for the insured value of the contents and the second represents the risk of insuring the the contents (e.g. risk from burglary). This should again be a value between 0 and 1, inclusive, with, 0 representing the lowest risk and 1 the highest risk.

     The insurance premium for a contents policy with contents value, `c`, and risk, `r`, is calculated by the formula

     $$\texttt{premium} = \texttt{c} \times \texttt{CONTENTS\_FACTOR} \times (1 + \texttt{r}),$$

where CONTENTS_FACTOR is an appropriately set class constant (a typical value for this constant might be 0.01). [10 marks]

- CarPolicy has four fields in addition to the inherited fields: a field of type double for the value of the vehicle, a field of type int for the driver's age, another field of type int representing the number of years no claim has been made on the policy, and a boolean field to indicate whether or not the policy is fully comprehensive.

  The company will not issue a car policy to anyone less than 17 years or greater than 99 years of age. The premium for a car insurance policy depends on whether or not the policy is a fully comprehensive policy, on whether or not the driver is a "young driver" (a young driver is anyone between between 17 and 25 inclusive), and on the number of years "no claims bonus". For each year no claim has been made up to and including the fifth year, the company gives a discount of 7.5% off the premium. Before adjusting the premium for these three factors, the company uses the following formula to set the initial premium:

  initial premium =

  max(BASIC_COVER, company fixed percentage of vehicle value),

  where BASIC_COVER is an appropriately set class constant (a typical value for this constant might be 100). The company currently uses 10% of vehicle value in the above formula.

  If the policy is fully comprehensive, the initial premium is increased by a factor of 50%. The premium is increased by a further factor of 50% for a young driver. Finally, any no claims discount is applied to get the final premium. [10 marks]

- LifePolicy has three fields in addition to the inherited fields, the first two of type int and the third of type double. The first two of these fields are for the age of the policy-holder and the amount for which the life is covered. The final field is to represent the health risk of the policy-holder and is a value between 0 (lowest risk) and 5 (highest risk). The company charges a basic amount per 1000 or part 1000 of life cover. This is then multiplied by
  1 + the health risk value, to give a value p, say. If the policy-holder's age does not exceed a company age threshold, p is the premium. Otherwise, the premium is obtained by multiplying p by the policy-holder's age divided by the age threshold. [10 marks]

3. The PolicyList class is used to represent a collection of Policy objects. Your class should include a method to add a Policy to the collection, a method to give the number of Policy objects in the collection and a toString method. [5 marks]

4. Name should have three fields, for title, initials and surname, respectively. In addition to accessor methods and methods to update the field values, a toString method should be provided. [5 marks]

5. The Address class should implement the Comparable interface. It is to have three fields, for street, town or city and postcode, respectively. Appropriate get and set methods should be defined together with a toString method. [5 marks]

6. The ClientDetails class is to represent details of an individual client. It should include a field for each of the following:

   - the client ID (of type String),
   - the full name of the client (of type Name),
   - the full address of the client (of type Address),
   - the list of policies held by the client (of type PolicyList).

   No client has more than one policy of each kind.

   Include appropriate methods in ClientDetails. [5 marks]

7. The ClientDetailsList class is used to represent a collection of ClientDetails. In addition to methods for addClient and numberOfClients and a toString method, implementations of the methods specified as follows should be given:

```
 /**
  * A method to determine whether or not a given person, identified by a
  * surname and a postcode is a client of the Insurance company.
  * If so, the client's ID should be returned.
  * @param lastName  the surname of the person to be searched for.
  * @param code      the postcode of the address of the person to be
  *                  searched for.
  * @return          the Client ID if the person has at least one policy
  *                  with the company, null otherwise.
  */
 public String findClient(String lastName, String code){
     // YOUR CODE HERE
}

 /**
  * A method to get the client details corresponding to a given client ID
  *
  * @param clientID     the client ID whose details are required.
  *
  * @return             the required ClientDetails  if found,
  *                     null otherwise.
```

```
     */
   public ClientDetails getClientDetails(String givenID){
         // YOUR CODE HERE
   }

    /**
     * A method to determine  another client who has the same address as
     * the client whose details are given.
     *
     * @param cDetails      the client details whose address is to be
     *                      searched for.
     * @return              the ClientDetails of a client with  the same
     *                      address if there is one, null otherwise.
     */
   public ClientDetails sameAddressCheck(ClientDetails cDetails){
       // YOUR CODE HERE
   }
```

[15 marks]

8. Class `InsuranceCoDemo` should be your `main` method class. Use this class to test your methods as you develop the other classes. Do this as you go along, testing the application bit by bit. When you are satisfied with your classes do the following.

   Put a copy of the file `ClientDetailsInput` which you can find on BlackBoard into the folder for your NetBeans project. The format of this file is as follows.

   The data for successive clients are separated by a # character. Individual items of data for a client are separated by a / character. After a token for the client's ID, tokens are given for the name and then the address of the client. Finally the data is given for each of the policies held by the client. The data for a policy consists of the year of issue, followed by the policy no. string. Finally, the data for the policy-dependent fields are given. The following are typical sets of data:

```
IC-x00042W/Ms/LQ/Bethea/205, Willis Road/Bolton/BO5 1DQ/2007/C02000007/10000/0.5/

2008/B27100037/150000/0.3/2011/V30200319/2000/21/1/0#

IC-x00018O/Ms/NAP/Wallis/244, Grubb Lane/Durham/DU4 4ZX/2007/B23000006/110000/0.4/

2010/C18100012/8000/0.4/2008/L43200805/85000/33/1#
```

   The class `InputData` is to contain a single `static` method with the header

```
public static ClientDetailsList readFile( File inputFile )
                                throws IOException, IllegalPolicyException
```

   The purpose of this method is to read the data from the specified file and to create the corresponding `ClientDetailsList`.

[10 marks]

Submission instructions overleaf.

Submission Procedure

To the hub: Hard-copy listings for all of your classes. Please submit class listings in the order given in the preamble to the coursework. You should also submit printouts for your test runs. All code should be adequately documented with comments, and structured using appropriate indentation. You should also submit printouts from NetBeans of each of your java classes, together with appropriate printouts of test results from your program.

via blackboard: a zipped file containing your Netbeans project.

Plagiarism: Plagiarism is the copying of close paraphrasing of published or unpublished work, including the work of another student; without due acknowledgement. Plagiarism is regarded a serious offence by the University, and all cases will be investigated. Possible consequences of plagiarism include deduction of marks and disciplinary action, as detailed by UEA's Policy on Plagiarism and Collusion.