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

The company assigns a Client-ID to each of its clients: this is a string of letters, digits, and hyphens.

In addition, the following information is recorded for each client:

- title (Mr/Ms etc.),
- initials,
- surname,
- address information,
- the Policy No. for each insurance policy held by that client.

It is assumed that no client has more than one policy of each kind. Each address consists of

- a street location (e.g. "888 Unthank Road"),

- a town/city name,
- a postcode.

Each Policy No. is a 10-digit number:

- the first four digits represent the year in which the policy was issued;
- the fifth digit represents the kind of policy — 0 for Home, 1 for Home-Contents, 2 for Car, 3 for Life.

Input Data Format

A file called `ClientDetailsInput` containing a sample input data set is provided on Blackboard. In this file, each client is represented by a string containing, in order, the following fields, separated from each other by forward-slash characters.

- Client-ID,
- three name fields (as described above),
- three address fields (as described above),
- Policy Numbers (one for each policy held).

The end of each client's data is marked by a # character. The following is a typical line of data:

IC-x00027Q/MR/Q/Mcqueen/236, Thatcher Street/Canterbury/CT2 6NG/2005000000/2004100022#

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

- `Name`
- `Address`
- `Policy`
- `PolicyList`
- `ClientDetails`

- ◇ `ClientsDetailsList`
- ◇ `InputData`
- ◇ `InsuranceDemo`

1. `Name` should have three fields, for title, initials and surname, respectively. In addition to accessor methods, a `toString` method should be provided. **[4 marks]**

2. The `Address` class should implement the `Comparable` interface. It is to have three fields, for street, town or city and postcode, respectively. A `toString` method should be defined. **[7 marks]**

   The structure of the `Address` class is as follows.

```
public class Address implements Comparable<Address>{

   // YOUR CODE FOR FIELDS

   // YOUR CODE FOR OTHER METHODS

   public int compareTo(Address other){

       // YOUR CODE

   }

}
```

3. The `Policy` class should have just one field, for a policy No. In addition to a constructor, an accessor method and a `toString` method, you should implement the two methods specified as follows:

```
/**
 * A method to determine the policy type of this policy.
 *
 * @return      One of the strings "HOME", "HOME_CONTENTS", "CAR" or "LIFE",
 *              depending on the code included in the policy No. of this policy.
 *
 * Throws an IllegalPolicyCodeException if the code included in the
 * policy No. is not in the range 0 to 3.
 */
 public String getPolicyType() throws IllegalPolicyCodeException
 {
   // YOUR CODE HERE
 }
```

and

```
/**
* A method to determine the year of issue of this policy.
*
* @return      a 4-digit integer representing the year of issue of this policy.
*/
public int getYearOfIssue(){
    // YOUR CODE HERE
}
```

`getPolicyType` may throw an `IllegalPolicyCodeException`. This type of Exception is defined by the following class. You must add this class to your NetBeans project.

```
public class IllegalPolicyCodeException extends Exception
{

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

[10 marks]

4. 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. **[6 marks]**

5. 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).

   Include appropriate methods in ClientDetails. **[6 marks]**

6. The ClientsDetailsListclass is used to represent a collection of ClientDetails. In addition to addClient, numberOfClients and a toString methods, 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]

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

   ```
   public static ClientsDetailsList readFile( File inputFile ) throws IOException
   ```

   The purpose of this method is to read the data from the specified file and to create the corresponding `ClientsDetailsList`. **[10 marks]**

8. `InsuranceDemo` is to contain the `main` method class and is to be used to demonstrate the use of the above classes. **[10 marks]**

9. Add a static method to the `InsuranceDemo` class that creates and uses input dialog boxes to get details for a client who wishes to take out a particular policy. The type of policy should also be obtained through the use of input dialog boxes. If the client is an existing client, the new policy should be added to that client's policy list - if the client already has a policy of this type the new policy should replace it. Thoroughly test this method. **[12 marks]**

Submission process for your work:

- ◇ All code should be adequately documented with comments, and structured using appropriate indentation.
- ◇ Together with your printouts, you should also submit a written report. The report should start with a one-paragraph executive summary indicating the extent to which you have successfully completed the exercise, and outlining the main points in the report which follows.
- ◇ The main body of the report (no more than four pages) should describe briefly but clearly and accurately the design decisions and implementation strategies you have adopted, and testing regime you have followed.
- ◇ For each milestone, state to which extent you completed it. State explicitly whether you have attempted it at all, whether your implementation compiles without errors, whether it runs without runtime errors, and whether it produces correct results. **[20 marks]**