CST2110 Resit Assessment Part 2

Deadline for submission **16:00, Friday 8th July, 2022**

Please read all the assignment specification carefully first, before asking your tutor any questions.

General information

You are required to submit your work via the dedicated resit assignment link in the module myUnihub space by the specified deadline. This link will 'timeout' at the submission deadline. Your work will not be accepted as an email attachment if you miss this deadline. Therefore, you are strongly advised to allow plenty of time to upload your work prior to the deadline.

Your submission (ZIP file) must also include a completed declaration of authenticity using the form provided in the module myUnihub space. Your work will not be marked if you do not complete and submit the declaration.

Submission should comprise a single 'ZIP' file. This file should contain a separate, cleaned[1], NetBeans project for each of the three tasks described below. The work will be compiled and run in a Windows environment, so it is strongly advised that you test your work in a Windows environment prior to cleaning and submission.

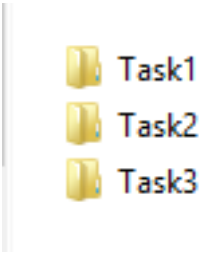

Figure 1: When the ZIP file is extracted there should be three folders named Task1, Task2 and Task3

Accordingly, when loaded into NetBeans, each task must be a separate project as illustrated by Figure 2 below.

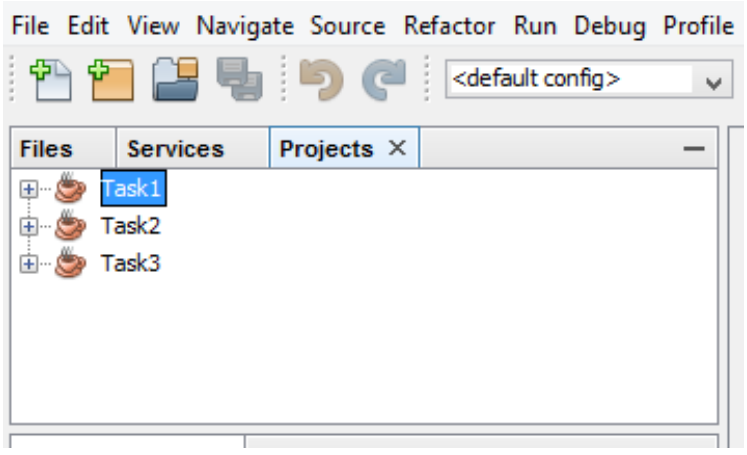

Figure 2: Each task must be a separate NetBeans Project

To make this easier, a template NetBeans project structure is provided for you to download.

[1] In the NetBeans project navigator window, right-click on the project and select 'clean' from the drop-down menu. This will remove .class files and reduce the project file size for submission.

Task 1 (20 Marks)

Create a NetBeans project for this task, named *Task1*.

You are required to write a Java 8 program that opens and reads a delimited data file that is located relative to the NetBeans project root folder. The delimited data file contains information about (fake) insurance companies. The data file is called *insurance-data.txt*. The data file must not be altered and should be considered as a *read-only* data file.

The data file is delimited in a consistent structure and contains entries relating to twenty (fake) insurance companies (with fake details). Each entry contains a series of data fields representing the following information: the name of the company, a telephone number, a web address, a list of the types of insurance cover offered by the company, a broker percentage, and a general description of the company.

You are required to implement a Java class to represent the insurance company information with respect to this data set. The program should parse the data file and create and store a collection of objects for each entity. Figure 3 provides a partial UML class representation of the class that you will need to implement. The class model indicates the data members and accessor (i.e., *getter*) methods that map to those data members, and a *toString()* method for the *InsuranceCompany* class. It is left to you to determine how the objects should be initialised.

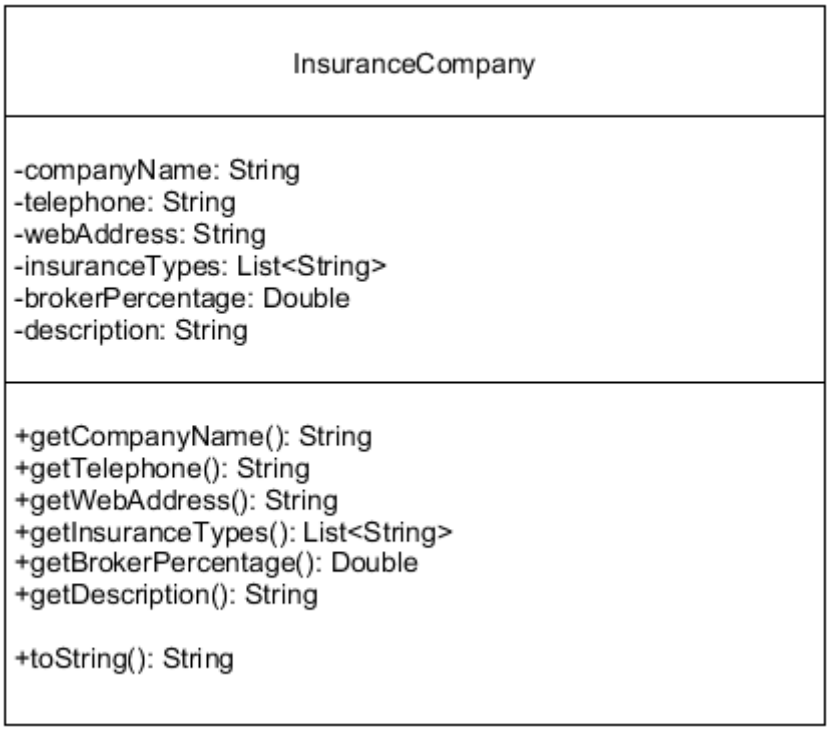

Figure 3: UML class specification for *InsuranceCompany* class

Once all the objects are loaded into the collection, the program should present the User with a console-based menu to interact with the data set. This menu should loop until the User enters a character to exit the menu (e.g., zero as illustrated below). In addition to an exit option, the menu should offer three other options: list all insurance companies, select a single insurance company (to display its details), and perform a search on company insurance cover.

On starting the program, the following menu should be displayed to the console:

```
List insurance companies.......1
Select insurance company.......2
Search cover...................3
Exit...........................0

Enter choice:>
```

The User can simply exit the program by entering zero. The other menu options allow the User to inspect the information in the data set (note again that this program is entirely *read-only* and there is no requirement to add, update or delete any part of the data set). The necessary interaction of the program with respect to these three user options (i.e., list, select and search) is illustrated in Appendix A.

Note that console output should be neatly formatted, and some consideration will be given to formatting when the program is assessed. In particular, when the option to view a single insurance company details is selected, it should result in the invocation of the *toString()* method for that particular *InsuranceCompany* object. You are encouraged to explore and utilise a *StringBuilder* object when implementing the *toString()* method for the *InsuranceCompany* class.

Task 2 (40 Marks)

Create a new NetBeans project, called Task2.

For this task you are required to write a Java 8 program that incorporates a series of Java classes that represent the core logic of an application and provides a NetBeans console user interface. The required Java application relates to a proposed software system to manage some aspects of administration for a small company that owns several stores.

System domain

The company owns two kinds of store: local newsagents and small supermarkets. Each store has a unique name. The company keeps records of every customer registered in their customer rewards scheme. Every customer has a unique customer identifier (alphanumeric) and is given one reward point each time they make a shopping trip to one of the company's stores. The products bought during each shopping trip are recorded but no record is kept of the quantity of each product bought.

Each product has a unique name and is categorised as a type of product (e.g., fruit, vegetable, general household etc.). Supermarkets stock a full range of products, whereas newsagent's stock only stationary and confectionary products. Every supermarket is partitioned into aisles, each containing products pertaining to the category (type) of product e.g., there will be an aisle that contains all the fruit products, and aisle that contains all the meat products etc.

Use cases

Use case 1: List Store's Shopping Trips. The system allows the User (i.e., the administrator) to select a store from a list of stores owned by the company which are displayed to the console. Following the User selection, the system displays the name of each registered customer who has made a shopping trip to that store, together with their customer number. For each such customer, the system also displays the date of each shopping trip to the store made by the customer and the total amount paid for the products bought during that shopping trip. The list of trips presented to the user should be ordered by the date of the shopping trip.

Use case 2: List Customer's Shopping Trips. The system allows the User (i.e., the administrator) to select a customer from a list of registered customers which is displayed to the console. Following the User selection, the system displays the number of reward points gained by the customer and, for each shopping trip made by the customer, the date of the shopping trip, the name of the store visited, the name of each product bought during that trip and the description of its type. The list presented to the user should be ordered by the date of the shopping trip.

Use case 3: Record Customer's Shopping Trip. The software system prompts the User (i.e., the administrator) to select a customer from a list of registered customers which is displayed to the console. The User is then prompted to enter a store from a list of stores displayed to the console. If the selected store is a newsagent, then the User is presented with a list of all confectionary and stationary products. The system allows the User to specify a list of products (from those listed) for that shopping trip. Once all required items are selected, the system records the shopping trip and displays a message to the console confirming the details of the shopping trip. If the selected store is a supermarket, the User is prompted to first select an aisle. On selection of an aisle, the User is presented with a list of all products contained within that aisle and prompted to select a product to be added to the current shopping basket. The User should be allowed to continue shopping from any aisle until the User indicates that there are no further products required, in which case the system records the shopping trip and displays a message to the console confirming the details of the shopping trip.

Your task is to implement a Java 8 program for the store administration system described above, which provides a console-based (text I/O) user interface that facilitates the use cases listed above. An initial analysis of the domain has already been completed, and a class model has been designed as shown in Figure 4.

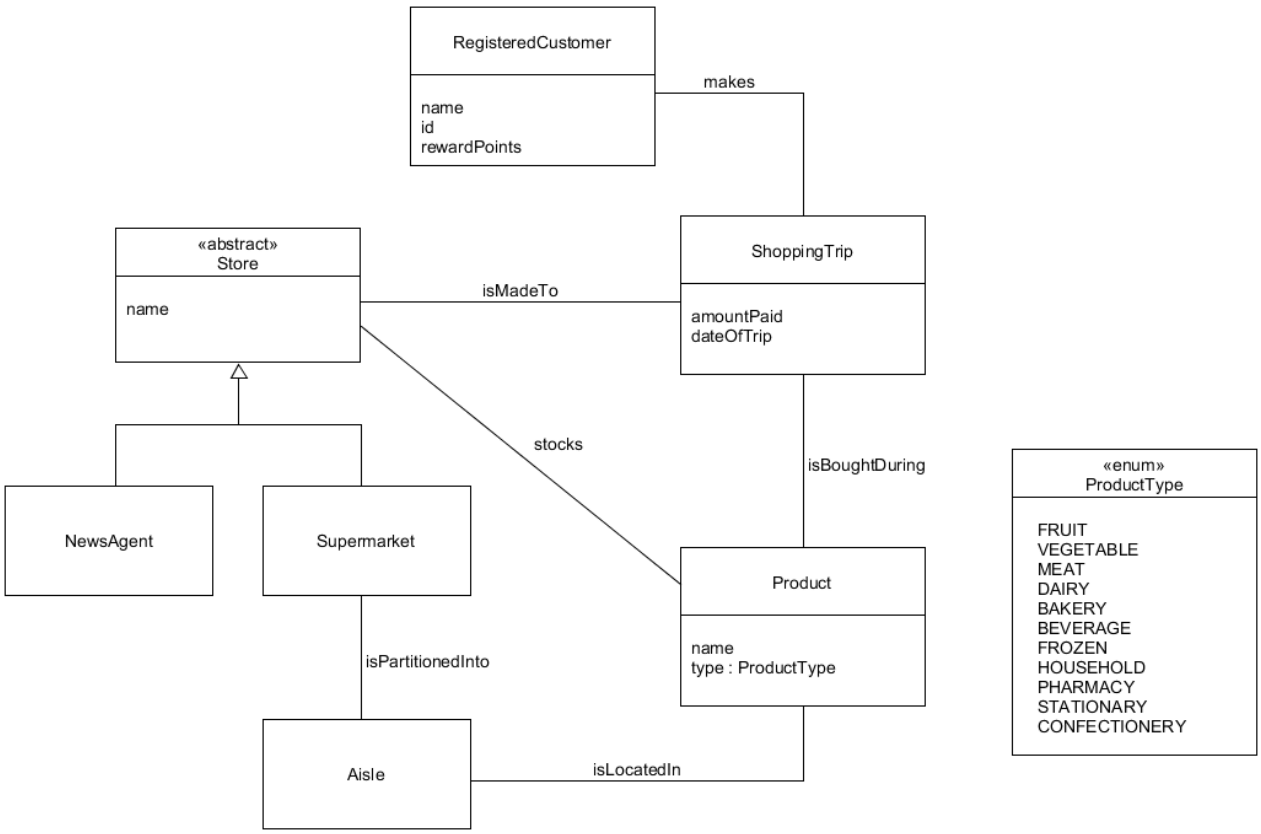

Figure 4: Class model of the store domain.

The class diagram above is provided as a design for your Java implementation, i.e., your application must include Java classes for the above class model as a minimum. You can extend/enhance the model if you wish to, but the above model structure must be implemented as specified. You are not required to submit any UML for this task.

So that the three use cases specified above can be tested, your program must provide a console-based User interface menu, with a selectable menu item for each of the three use cases. Your application should pre-load some 'dummy' data into the application, but also allow information to be input (use case 3) during runtime. Pre-loaded data should be achieved by coding data directly within the Java program. It should **not** use an external database link (e.g., such as an SQL database). All pre-loaded data should be *read-only*. Any data that is input during the runtime of the Java program should be volatile i.e., it should exist only for the time the program is executing and should not be stored to non-volatile location such as an external file or a database.

Task 3 (15 Marks)

Create a new NetBeans project, called Task3.

For this task you are required to program a JavaFX Graphical User Interface (GUI), with a focus on layout and some event handling.

This program must be coded as a JavaFX program following the approach described in the lecture series and Chapters 14 to 16 of the *kortext*. It must not be created using a visual drag-and-drop tool, and must not be an FXML program, neither of which have been taught on the module. If the submission is either a drag-and-drop application, or a FXML application, then it will simply be awarded zero marks.

The GUI application that you need to create is a smart home controller simulation. The required design is illustrated in Figure 5a below.

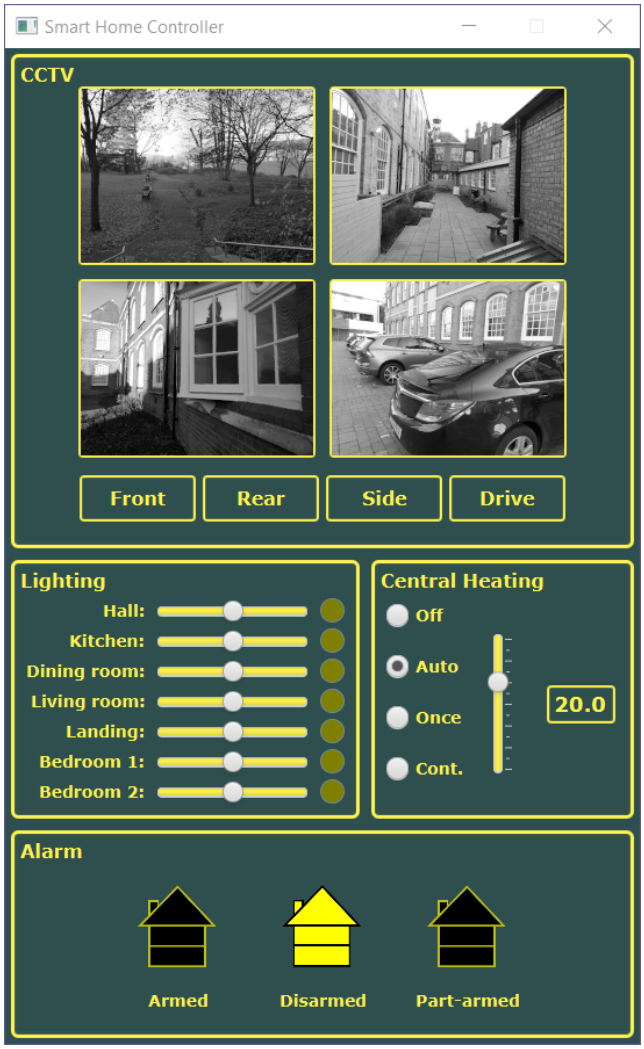

Figure 5a: Smart home controller simulation user interface

The application comprises several areas representing controls for CCTV, lighting, central heating, and a house alarm. Each area on screen comprises JavaFX nodes and controls (e.g., Labels, ImageViews, Buttons, TextFields, and Sliders).

There are several interactive aspects to the GUI:

- For the CCTV area, each of the four ImageViews should be mapped to one of the four ToggleButtons. Clicking on any of the four ToggleButtons should result in the associated image being 'blacked out' (clicking again would reveal the image again).
- For the lighting area, the colour of each Circle object should change in response to the associated horizontal slider being dragged. When the Slider is dragged to the left the colour should 'darken', and when dragged to the right, the colour should 'lighten'.
- For the central heating area, the RadioButtons should operate correctly in that only one RadioButton can be selected at a time (i.e., they are mutually exclusive in terms of selection). The Slider must be vertical and when dragged up or down the value in the TextField must be updated to reflect the value on the Slider.

The results of these interactions are illustrated in Figure 5b below

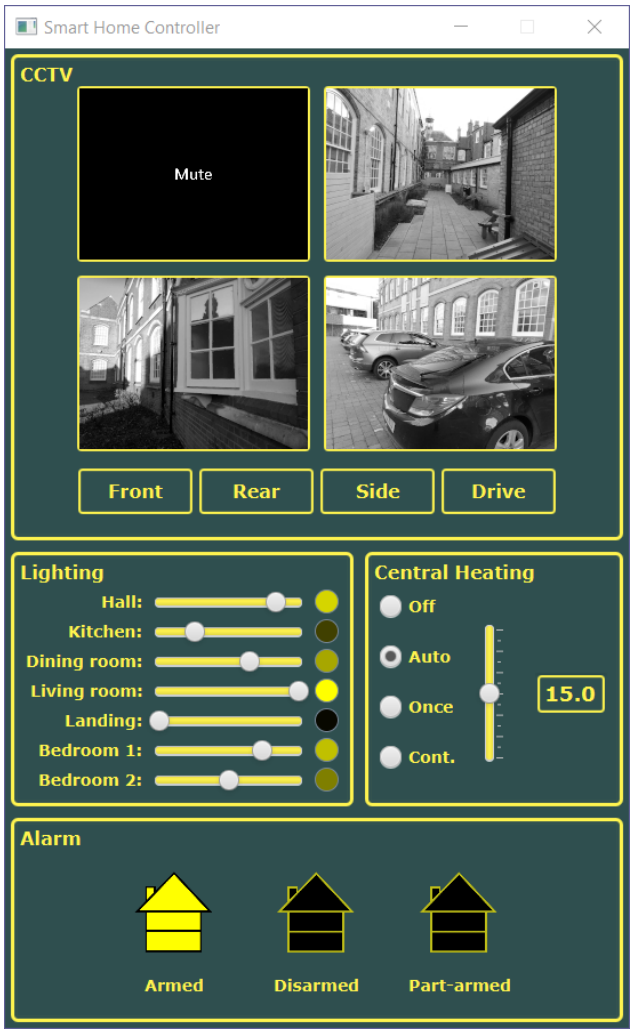

Figure 5b: Results of event interaction with the smart home controller simulation

The alarm section should allow the User to select one of three modes (armed, disarmed, or part-armed) by clicking with the mouse. They should be highlighted accordingly, with only one option selected at a time (i.e., they should be mutually exclusive such that when one is switched on, the other two options are switched off). Figure 5c (below) illustrates the selection of the 'part-armed' option:

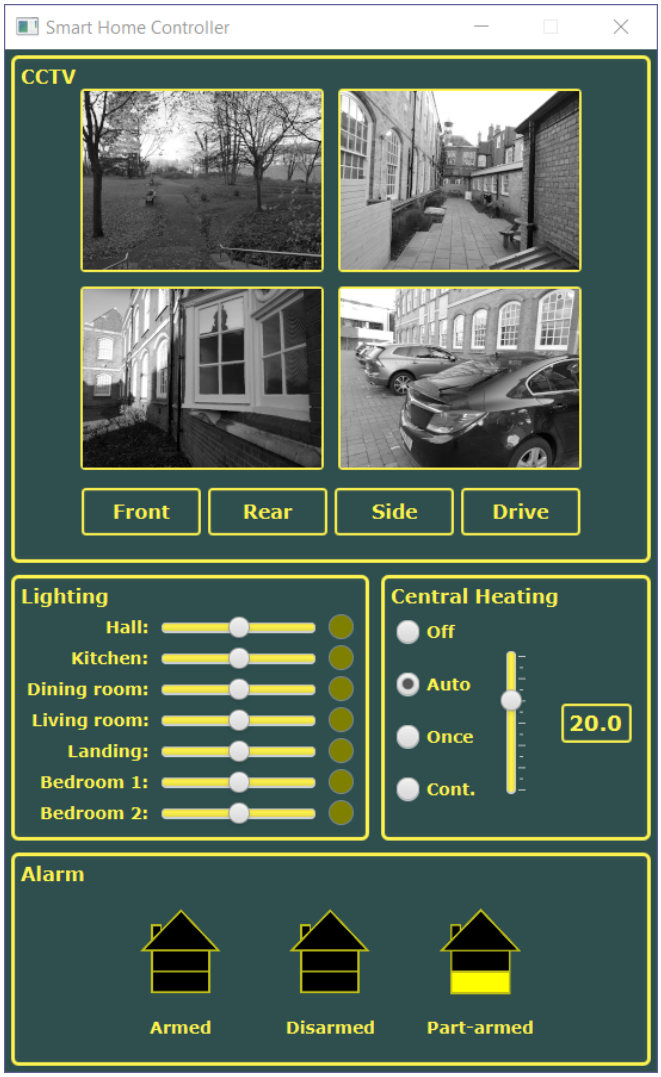

Figure 5c: Result of clicking the 'part-armed' option with the mouse

To assist in explaining how this JavaFX simulation GUI should operate, a video demonstration of a prototype will be made available to view in the module myUnihub space.

Your task is to write a JavaFX program that replicates (as closely as possible) the GUI layout and design as illustrated in Figures 5a, 5b and 5c. Styling of widgets must be achieved programmatically using node and control property setting directly (e.g., by using the relevant 'setStyle' method for the respective Node objects). In other words, all layout and styling should be contained within the Java code, use standard JavaFX components, and not by use of a separate CSS file or external custom JavaFX GUI libraries. It is advised that you focus on layout aspects first with regards to the main screen, and when you have a working layout then focus on the interactive and dynamic aspects of the application.

Appendix A

Option 1: list insurance companies

On selecting option 1, the User should be presented with a neatly formatted listing of the insurance companies. The data displayed should be the company name and the insurance cover types offered by the respective company. Most importantly, the listing should also provide a unique identifier in order that the User can use this to select a single insurance company to view all the details of that company (i.e., option 2).

```
List insurance companies .......1
Select insurance company........2
Search cover....................3
Exit............................0

Enter choice:> 1
```

ID	Company name	Cover Types							
1	Umbrella Insurance	Motorcar	Buildings	Contents	Travel	Boiler			
2	GoStudent	Motorcycle	Travel	Flights	Pet	Bicycle	Student		
3	CompareThem	Buildings	Emergency	Life	Accidental	Pet	Bicycle		
4	Geneva	Travel	Flights	Life	Motorcar	Van	Buildings	Contents	
5	Dunhill	Motorcar	Medical	Travel	Holidays	Pet	Boiler	Bicycle	
6	Legal and Accidental	Legal	Accidental	Travel	Flights	Life	Buildings	Contents	
7	Direct Insurance	Buildings	Contents	Legal	Motorcar	Life	Emergency		
8	Meerkat & Co	Pet	Van	Motorcar	Motorcycle	Student	Contents	Accidental	
9	Red Telephone Direct	Motorcar	Bicycle	Van	Buildings	Contents	Life		
10	Triple A Insurance	Travel	Holidays	Buildings	Life	Emergency	Accidental	Legal	
11	Endersley Insurance	Student	Travel	Bicycle	Contents	Emergency			
12	Salamander	Buildings	Contents	Life	Emergency	Boiler	Accidental	Plumbing	
13	Melton & Co	Travel	Life	Motorcar	Bicycle	Accidental	Pet		
14	Less Than	Life	Buildings	Contents	Travel	Pet	Motorcar		
15	Greenwich Victoria	Van	Motorcar	Motorcycle	Bicycle	Accidental	Legal		
16	Winston Insurance	Buildings	Contents	Legal	Life	Accidental	Motorcar		
17	Staycertain travel	Travel	Holidays	Medical	Luggage	Sports	Flights		
18	Ideal Insurance	Boiler	Heating	Emergency	Plumbing	Accidental	Contents	Building	
19	Hendon Insurance	Life	Student	Bicycle	Travel	Holidays	Luggage		
20	Muffield Health	Health	Medical	Life	Legal	Accidental			

Option 2: select insurance company

On selecting option 2, the User should be prompted to enter the unique identifier of a listed insurance company (that was displayed when option 1 was chosen). Following this, all the details of that company should be displayed, as illustrated below. This should be achieved by invocation of the relevant *InsuranceCompany* object's *toString()* method. Console output should be neatly formatted.

```
List insurance companies .......1
Select insurance company........2
Search cover....................3
Exit............................0

Enter choice:> 2

Enter company ID from list [1 - 20] :> 12

-------------------------------------------------------------------------
| Salamander            | General high-street provider.                 |
-------------------------------------------------------------------------
| Covers                | Buildings                                     |
|                       | Contents                                      |
|                       | Life                                          |
|                       | Emergency                                     |
|                       | Boiler                                        |
|                       | Accidental                                    |
|                       | Plumbing                                      |
-------------------------------------------------------------------------
| %                     | 7.4                                           |
-------------------------------------------------------------------------
| Tel.                  | 0852 896120                                   |
-------------------------------------------------------------------------
| Web                   | www.salamander.com                            |
-------------------------------------------------------------------------


List insurance companies .......1
Select insurance company........2
Search cover....................3
Exit............................0
```

Option 3: search cover

On selecting option 3, the User should be prompted to enter a 'search string' to match against the *insuranceTypes* field of the *InsuranceCompany* objects. The search string entered by the User should be on single line (followed by a return to enter the search parameters). The structure of the search string should be a list of insurance cover types separated by whitespace, followed by either the string 'AND' or the string 'OR' to specify the Boolean combination of the search. That is to say, if the User enters two insurance cover types followed by the string 'AND' then the search is for insurance companies that offer both types of cover. If the User enters two cover types followed by the string 'OR' then the search should return insurance companies that offer either of the cover types (but not necessarily both). The program should return a listing of trading companies that match using the same neat formatting as the option to list all trading companies, as illustrated below:

```
List insurance companies .......1
Select insurance company........2
Search cover....................3
Exit............................0

Enter choice:> 3

Enter search parameters [ cover1 cover2 cover3 AND/OR ] > Health Medical AND

The following companies offer the cover you need:

-------------------------------------------------------------------------------------------------------------------
| ID | Company name          | Cover Types                                                                        |
-------------------------------------------------------------------------------------------------------------------
| 20 | Muffield Health       |       HEALTH      MEDICAL         Life        Legal   Accidental                    |
-------------------------------------------------------------------------------------------------------------------


List insurance companies .......1
Select insurance company........2
Search cover....................3
Exit............................0

Enter choice:> 3

Enter search parameters [ cover1 cover2 cover3 AND/OR ] > Health Medical OR

The following companies offer the cover you need:

-------------------------------------------------------------------------------------------------------------------
| ID | Company name          | Cover Types                                                                        |
```

```
-------------------------------------------------------------------------------------------------------------------------
|  5 | Dunhill               |     Motorcar      MEDICAL       Travel     Holidays          Pet       Boiler      Bicycle |
| 17 | Staycertain travel    |       Travel     Holidays      MEDICAL      Luggage       Sports      Flights              |
| 20 | Muffield Health       |       HEALTH      MEDICAL         Life        Legal   Accidental                           |
-------------------------------------------------------------------------------------------------------------------------


List insurance companies .......1
Select insurance company........2
Search cover....................3
Exit............................0
```