UNIVERSIDADE ABERTA
www.uab.pt

Ministry of Education and Science

e-Folio

U.C. 21077

Programming Languages e-Folio B - Prolog Language

-- INSTRUCTIONS –

1) The e-folio has a value of 4.

2) Any attempt at plagiarism will result in a final mark of zero.

3) This e-folio should be solved using the *Prolog language*.

4) A compressed file (ZIP or RAR) with the student's name and number must be submitted:

a) Programme code;
b) Readme.txt file with the information needed to compile and run the programme;
c) Report of up to 4 pages describing the solution presented and the tests carried out.

5) The following assessment criteria will be considered:

a) 40% C1 - Problem solving
b) 40% C2 - Code Quality
c) 20% C3 -Documentation (Report + Readme)

E-folio B

As a recent hire in the IT sector of a bank, you have been assigned the task of creating a bank management system for internal use by account managers. Your IT Director insists on using the *Prolog* language for the back-end as he considers it to be faster at processing information, and has instructed that the front-end of the system will be in console mode in *Java*. After meeting with some future users and the system architects, the following requirements were raised:

Currently, Account Managers use tables to control accounts, as follows: Customer information:

Customer number	Name	Agency	City	Opening Date
123	Alice	NYC-123	New York	01-01-2020
456	Bob	CHI-456	Chicago	02-02-2020
789	Charlie	LA-789	Los Angeles	03-03-2020
752	John	CHI-456	Chicago	03-03-2020

Total account balance: (Represents the total value of the current balance plus the available credit value)

Customer Number	Balance Sheet (Currente+Credit) Total
123	2500
456	500
789	50
752	5000

Credit balance:

Customer Number	Credit balance
123	0
456	5000
789	2000

Movements:

Customer Number	Value(+-)	Date of Movement
123	-100	01-01-2020
456	200	02-02-2020
789	-10	03-03-2020
789	-20	04-03-2020

For your convenience, the system only takes integers into account in its values.

1. The system architects have suggested that you keep this structure when creating your facts, as the user is already used to it. Create the facts you need for your programme.

2. Get all your customers.
   a. Create a predicate that retrieves all your customers' data in Prolog.
   b. Create a Customer class with the necessary attributes of a customer Customer Number, Name, Branch, City and Opening Date.

   c. Create a SistemaBancario class that stores a list, vector or matrix of the Customer class type. To save its customers, the constructor of the class must obtain the list of customers from Prolog and save each one in the list, vector or matrix and sort the list in ascending order of Customer number.
   d. Create a method in the SistemaBancario class to print the list of customers with the data for each customer.
   e. Create a method in the SistemaBancario class that displays a menu of options where the first option is to show the list of clients and the last is to exit the programme.

3. Get customers based on the city.
   a. Create a predicate in Prolog that obtains the customer number and customer name based on a city.
   b. Create a method in the SistemaBancario class that gets the list of customers for a given city from Prolog and prints this list with the customer number and name.
   c. Add the option to view customers from a specific city to the menu you created previously.

4. Checking customers eligible for credit. A customer is eligible for credit if their balance on the total balance sheet is more than 100 currency units and their credit is 0 or non-existent.
   a. Create a predicate in prolog that gets all the customers eligible for credit, it should return all the customer information data, Customer Number, Name, Agency, City and Opening Date.
   b. Create a method in the SistemaBancario class that gets all the customers eligible for credit from Prolog and prints out all their customer information.
   c. Add the option to view credit-eligible customers to the menu previously created.

5. Access customer operations.
   a. Create a method in the Customer class that displays a customer menu, initially with only the option to return to the previous menu.
   b. Create a method in the SistemaBancario class where you can select a customer from your list, vector or matrix of customers, based on a given customer number, and call the menu method for that customer (Customer class).
   c. Add the previously created method to the menu (of the SistemaBancario Class), i.e. the option to select a particular customer by their number and view their customer menu.

6. Check the customer's actual balance. The customer's actual balance is the difference between the total balance and the credit balance.
   a. Create a predicate in Prolog that obtains the real balance of a given customer.
   b. Create a method in the Customer class that uses Prolog to obtain the customer's real balance and print it.
   c. Add the option to view a customer's balance to the previously created customer menu.

7. Check the customer's credit balance.
   a. Create a predicate in Prolog that obtains the credit balance of a given customer.
   b. Create a method in the Customer class that uses Prolog to obtain the customer's credit balance and print it.
   c. Add the option to view the credit balance to the previously created customer menu.

8. See customer movements.
   a. Create a predicate in Prolog that obtains the movements of a certain customer and returns the movements with the value and date.
   b. Create a method in the Client class that uses Prolog to obtain the movements and prints them with their value and date.

c. Add the option to view customer movements to the previously created customer menu.

9. Making deposits and withdrawals. A withdrawal can only be made if there is a sufficient balance on the customer's balance sheet. For each withdrawal and deposit, a new movement is recorded with its respective date (current system date at the time of registration), the balance value must be updated according to the transaction to be made.

   a. Create a predicate to make a deposit with a particular customer.
   b. Create a predicate to carry out a survey on a particular customer.
   c. Create a method in the Customer class that allows you to make a <u>deposit of a certain amount.</u>
   d. Create a method in the Customer class that allows you to <u>withdraw a certain amount.</u>
   e. Add both options to the customer menu you have already created.

10. Credit eligibility check and credit design. A customer is eligible for credit if their balance in the total balance is more than 100 monetary units and their credit is 0 or non-existent. When the credit is granted, the value of the total balance is updated by adding the balance to the value of the credit.
    a. Create a predicate in Prolog that checks whether a particular customer is eligible for credit.
    b. Create a predicate in Prolog that grants <u>a certain amount of</u> credit to a certain customer.
    c. Create a method in the Customer class that uses Prolog and checks whether the customer is eligible for credit and prints a positive or negative message.
    d. Create a method in the Customer class using Prolog that grants a credit of a certain amount to the customer and prints the success or failure message.
    e. Add the option to check credit eligibility and grant credit to the customer menu.

Notes:

- You are free to change the names of the classes to suit you.
- You don't need to create anything other than the requirements, just spend your time on what is asked of you.
- To help you get the current date, here's a predicate:

```
getCurrentDate(Date) :-
 get_time(Timestamp),
 stamp_date_time(Timestamp, DateTime, 'UTC'),
 date_time_value(year, DateTime, Year),
 date_time_value(month, DateTime,
 Month), date_time_value(day, DateTime,
 Day),
 format(atom(Date), '~|~`0t~d~2+~|`0t~d~2+~|~d', [Day, Month, Year]).
```

- The report emphasises the importance of explaining the choices made, i.e. an explanation with short code excerpts of how he made his decisions, with a debate between pros and cons. cons. Just consider the relevant decisions, you don't need to describe your entire code. Give some examples of how your programme works in a variety of functionalities.