E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Linguagens de Programação

CÓDIGO: 21077

DOCENTE: Ricardo Baptista e Rúdi Gualter Oliveira

A preencher pelo estudante

NOME: Pedro Pereira Santos

N.º DE ESTUDANTE: 2000809

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 15/05/2023

TRABALHO / RESOLUÇÃO:

Para este trabalho, foram utilizadas a linguagens de Java e Prolog. Para desenvolver o código utilizei o IDE Visual Studio Code, com recurso ao WSL (Windows SubSystem for Linux):

```
pedro@Santini:~/efolioB$ cd /home/pedro/efolioB ; /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java @/tmp/cp_2zhxdh7e38fzn7v98mye5n2ms.argfile App
Escolha uma opcao, pressionando o numero em frente:
Mostrar Lista de Clientes - 1
Mostrar Clientes de uma Cidade - 2
Mostrar Clientes elegiveis para credito - 3
Mostrar Cliente por Num. de Cliente - 4
Sair - 0
```

uma vez que com Windows obtia este erro, mesmo configurando a Path das variáveis de sistema:

```
App.java:1: error: package org.jpl7 does not exist
import org.jpl7.*;
^
```

Foram criadas 3 classes em Java, App, Cliente e SistemaBancario.

A Classe App serve para começar o programa, contendo o Main(), estabelecendo a conexão entre o Prolog e o Java. Para isto utilizamos o primeiro Query do programa com o predicado de consult("src/DB.pl"), sendo "src/DB.pl" a localização do ficheiro Prolog, como mostra a seguinte imagem:

```
//Esta pesquisa vai verificar se o ficheiro do prolog abre com sucesso
Query Conexao =
    new Query(
        "consult",
        new Term[] {new Atom("src/DB.pl")}
    );
```

De seguida passamos para a Classe SistemaBancario, onde são aplicadas os métodos gerais, para todos os clientes.

Para começar, o construtor deste Classe busca, como pedido, todos os clientes inseridos na base de dados do Prolog, segundo um predicado feito em Prolog. Para obter as informações dos clientes, criei um predicado getInformacoes/5 , onde se insere 5 variáveis, retornando nessas variáveis os dados gerais dos clientes, tendo de seguida o exemplo do código:

```
%Ex 2.a.
getInformacoes(ID,Nome,Posto,Cidade,Data):-
    infocliente(
        ID,
        Nome,
        Posto,
        Cidade,
        Data
    ).
```

Para guardar a informação dos clientes em Java, decidi utilizar um vetor do tipo da classe Cliente, pois é o estilo de estrutura que mais conheço e tenho facilidade em utilizar.

Para fazer as Queries, de modo geral, decidi que, quando se utiliza variáveis para buscar valores nas pesquisas, era melhor criar variáveis facilitando a escrita e compreensão destas pesquisas. Podemos ver isto neste exemplo:

```
//Cria variaveis para usar nas pesquisas
Variable ID = new Variable("ID");
Variable Nome = new Variable("Nome");
Variable Posto = new Variable("Posto");
Variable Cidade = new Variable("Cidade");
Variable Data = new Variable("Data");
```

Para o método de impressão da lista de clientes, o programa buscar as informações guardadas no vetor da lista de clientes, imprimindo as informações de um cliente à vez. Este método só utiliza a parte de Java.

Para o método de obter os clientes de uma determinada cidade, recebe-se o nome da cidade como input do utilizador. É preciso então formatar o input para igualar o modo como os nomes das cidades estão guardadas na base de dados em Prolog. Temos então:

```
String input = sc.nextLine().toLowerCase();//recebe a cidade em letras minusculas para igualar a base de dados

//usa-se um tokenizer para igualar o nome da cidade ao formato da base de dados
String Cidade = new String();
StringTokenizer st = new StringTokenizer(input," ");

while(st.hasMoreTokens())//ciclo para formatar nome da cidade
{
    Cidade+=st.nextToken();
    Cidade+="_";
}

Cidade=Cidade.substring(0,Cidade.length()-1);//retira-se o ultimo "_" adicionado, uma vez que esta a mais
```

Podemos ver que o input é convertido para letra minúsculas, pois é como o Prolog guarda infomações, adicionando o “_”, que substitui o espaço entre palavras. Visto que o ciclo adiciona um “_” extra, é necessário retirá-lo. Igualamos assim o input recebido ao formato da base de dados. Após isto, faz-se uma pesquisa parecida ao de obtenção de cliente, mas neste caso, só

precisamos de obter o Numero de Cliente e Nome do Cliente. Imprime-se, então, o resultado da pesquisa

Para o método de obter os clientes elegíveis a crédito, faz-se uma procura com um predicado que insere todas as variáveis de informação de cliente, retornando as informações dos clientes elegíveis. Após fazer a pesquisa, imprime os clientes elegíveis.

Para saber quais são os clientes elegíveis, necessitamos de fazer um predicado composto onde se usa a condição “OR” para verificar se o cliente, caso tenha balanço de crédito, este é maior a 100 U.M., sendo que neste caso, não ter balanço de crédito é um ponto favorável. Verifica-se, também, se o valor de Balanço Total é superior a 100 U.M. Sendo os pontos favoráveis, os dados dos clientes elegíveis são retornados para a solução. Podemos verificar o uso dos condicionais no predicado:

```
%Ex 4.a
getElegiveis(ID,Nome,Posto,Cidade,Data):-
    getInformacoes(ID,Nome,Posto,Cidade,Data),
    ((balancocredito(ID,Val),
    Val =:= 0);
    \+balancocredito(ID,_)),
    balancototal(ID,BalTot),
    BalTot > 100.
```

Para o próximo requisito, cria-se um método, para imprimir um Menu de Operações de Cliente, na Classe Cliente. Este menu funciona parecido ao menu da Classe SistemaBancario, tendo opções diferentes. Chama-se, então, o

```
for(int i = 0; i < ListaClientes.size(); i++)
{
    aux = ListaClientes.get(i);

    if(ID == aux.NumeroDeCliente)//Após confirmar que o cliente existe
        break;
}

aux.MenuCliente();//Abrir o menu de cliente para o cliente inserido
```

método do MenuCliente da Classe clientes para o cliente escolhido, como mostra abaixo, após confirmar que o cliente existe na lista de clientes.

De seguida passamos para a última Classe, a Classe Cliente, onde são implementados os métodos a usar apenas num cliente especifico, sendo este identificado pelo seu Número de Cliente.

Para os métodos seguintes, de obtenção do saldo real e do saldo de crédito, utilizam-se métodos parecidos. Em ambos recorremos aos balanços de crédito dos clientes, com a diferença que no método do saldo real, tem de se obter o saldo total e subtrair o saldo de crédito ao saldo total, obtendo assim o saldo real. Podemos ver a parecença dos métodos abaixo:

```
%Ex 6.a
getSaldoReal(ID,Saldo):-
    balancototal(ID,SaldoTot),
    (balancocredito(ID,SaldoCred) ->
        Saldo is SaldoTot - SaldoCred;
        Saldo is SaldoTot).
```

```
%Ex 7.a
getSaldoCredito(ID,Saldo):-
    balancocredito(ID,Saldo) ->
    Saldo is Saldo;
    Saldo is 0.
```

Para o próximo método, de obtenção de movimentos, cria-se um predicado simples, que chama os factos de movimentos/3. Podemos ver na imagem abaixo:

```
%Ex 8.a
getMovimentos(ID,Saldo,Data):-
    movimentos(ID,Saldo,Data).
```

Para os métodos de depósitos e levantamentos, foi onde tive mais problemas. Para conseguir executar estes passos com sucesso, tive de inserir a keyword dynamic para os predicados movimentos/3, balancototal/2 e balancocredito/2, como mostra a imagem.

```
%Variaveis dinamicas para alterar durante execucao do programa.
:- dynamic movimentos/3 ,balancototal/2, balancocredito/2.
```

Após várias tentativas, sem sucesso, de utilizar um predicado que alterasse o ficheiro de Prolog permanentemente com os novos movimentos às contas, tive de abandonar este método e utilizar a keyword dynamic. As tentativas incluíram a utilização de:

1. tell/1 , abrindo o ficheiro de Prolog.
2. Retract/1 , eliminando o facto a ser atualizado.
3. assertz/1 , criando os novos facto, dependendo do método.
4. listing/0 , de modo a reescrever todos os factos incluídos (pois usando listing/1, apenas o facto incluído era inserido apagando o resto do ficheiro)
5. told/0 ,fechando o ficheiro.

Um predicado a utilizar estes predicados, pela ordem numérica, criaria um ficheiro muito confuso, onde não existem muitas ajudas na internet, e de modo a não estragar a base de dados, preferi abandonar este método. Com isto, os dados são criados dinamicamente, enquanto o programa funciona, podendo ter acesso a eles enquanto o programa correr, mas, são apagados mal ele encerra.

Estes dois métodos são parecidos, criando novos factos após serem sucedidos, sendo que no de levantamento é necessário ter condições para verificar se o saldo no balanço do cliente for suficiente. Podemos ver as parecenças dos predicados abaixo.

```
%Ex 9.a
deposito(ID,Valor):-
    balancototal(ID,Saldo),
    NewSaldo is Saldo+Valor,
    getCurrentDate(Data),
    assertz(movimentos(ID,Valor,Data)),
    retract(balancototal(ID,Saldo)),
    assertz(balancototal(ID,NewSaldo)).
```

```
%Ex 9.b
levantamento(ID,Valor):-
    balancototal(ID,Saldo),
    NewSaldo is Saldo-Valor,
    NewSaldo >= 0,
    getCurrentDate(Data),
    ValorLevantamento is - Valor,
    assertz(movimentos(ID,ValorLevantamento,Data)),
    retract(balancototal(ID,Saldo)),
    assertz(balancototal(ID,NewSaldo)).
```

Para receber a data, utilizei a função auxiliar fornecida, sendo que me foi necessário modificar, por uma função própria, o formato da data em si. Esta função é a seguinte:

Este predicado retorna data no formato de DD-MM-AAAA, o formato utilizado por toda a base de dados, sendo a linha de

```
%Preficado para formatar data
formatar_data(Day, Month, Year, FormattedDate) :-
    atom_number(DayAtom, Day),
    atom_number(MonthAtom, Month),
    atom_number(YearAtom, Year),
    atom_concat(DayAtom, '-', Temp1),
    atom_concat(Temp1, MonthAtom, Temp2),
    atom_concat(Temp2, '-', Temp3),
    atom_concat(Temp3, YearAtom, FormattedDate).
```

“format(atom(Date), '~|~`0t~d~2+~|~`0t~d~2+~|~d', [Day, Month, Year]”

fornecida no predicado de getCurrentDate/1, alterada por

“formatar_data(Day,Month,Year,Date).”

Na parte do Java, após efetuar as pesquisas de depósito ou levantamento, verifica-se se existe solução, onde em caso afirmativo, o movimento foi executado com sucesso. O sucesso ou insucesso da operação é impresso para o utilizador.

Para o método de verificação de elegibilidade do cliente em questão, utilizei um predicado idêntico ao do método de obtenção dos clientes elegíveis, sendo que este retorna apenas “True” ou “False” em vez dos dados do cliente. Este estado booleano, é guardado na variável “EligibilidadeCredito” da Classe Cliente, para ser verificado no método seguinte, de conceção de crédito. Este último método, concede um valor de crédito ao cliente em questão, caso ele

esteja elegível para crédito, ou seja, caso o outro método tenha resultado em “True”.

Podemos ver na seguinte imagem o funcionamento da avaliação do método de verificação para guardar na variável “EligibilidadeCredito” ,

```
//Consoante o resultado da pesquisa, imprime o sucesso ou insucesso da operacao
if(ClienteElegivel.hasSolution()){
    System.out.println("O cliente e elegivel para credito!\n");
    EligibilidadeCredito = true;
}
else{
    System.out.println("O cliente nao e elegivel para credito!\n");
    EligibilidadeCredito = false;
}
```

sendo esta variável avaliada na realização do método de ConcederCredito, imprimindo uma mensagem de Erro e retornando da função, como podemos observar:

```
//Utiliza a elegibilidade a credito para saber se concede ou nao credito,
//uma vez que ja foi utilizado o Prolog para a obter
if(!EligibilidadeCredito){//Se o cliente nao for elegivel para credito
    System.out.println("Erro. O Cliente nao e elegivel a credito!\n");//Devolve se mensagem de erro
    return;//e retorna da função
}
```

Posto isto, temos todos os requisitos criados, com mensagens de erro em caso de impossibilidade de realizar alguma ação, informando sempre o utilizador do resultado dos métodos.

Para a realização deste trabalho, utilizei a documentação fornecida na UC e o ChatGPT para me explicar e ajudar a saber como funcionam algumas funções dentro do Prolog.

Após a conclusão do programa, testei todas as opção, verificando, até agora, que não existem bugs, a não ser, o input de dados que não sejam do tipo especifico em cada função.

O código encontra-se na pasta com nome src. Deixei este nome na pasta, pois o path para aceder ao ficheiro de prolog inclui o prefixo "src/".

O relatório inclui mais de 4 páginas pois inclui imagens do código.