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

UNIDADE CURRICULAR: Linguagens de Programação

CÓDIGO: 21077

DOCENTE: Ricardo José Vieira Baptista

A preencher pelo estudante

NOME: Ivo Vieira Baptista

N.º DE ESTUDANTE: 2100927

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 15 de Maio de 2023

TRABALHO / RESOLUÇÃO:

RELATORIO:
Quero pedir desculpas porque o meu relatório tem mais de 4 páginas, mas no efolioA fui penalizado por não ter trechos de código, e eu tinha colocado e deu 15 páginas, mas como estava extenso, decidi tirar e comentar linha a linha o programa Ocaml, diretamente no código, neste relatório vou passar as 4 paginas, arriscando ser penalizado novamente, mas optei por a versão mais alargada ☺.

Agora falando sobre o programa, fiquei mais confortável na parte de Java e fiz uma versão mais java, link na Webgrafia, e com Prolog foi com ajuda do vídeo do Professor Rudi, e do chat GPT, que me ajudou a otimizar o meu código de simples, para mais robusto, até conseguir a solução pretendida, tive 2 dias a tentar configurar o ambiente no MacOS, e acabei por instalar no Windows, depois voltei para o MacOS e descobri que compilar without debugging, trabalha perfeitamente e consegui criar o meu programa.

Relatório do Sistema Bancário

Introdução:

O Sistema Bancário é uma aplicação desenvolvida em Java que utiliza a biblioteca jpl7 para consultar informações em um arquivo Prolog (banco.pl), mantendo Backend com Prolog no ficheiro banco.pl e o Frontend com Java nos ficheiros Main.java, SistemaBancario.java e Cliente.java. A aplicação permite gerir clientes e suas informações, como nome, agência, cidade e data de abertura, além de operações como mostrar saldo real, balanço de crédito, clientes elegíveis a crédito e filtragem por cidade.

No ficheiro Main.java chamamos o menu principal, depois no ficheiro SistemaBancario.java:

1. Obter clientes do Prolog:

O código começa com a classe SistemaBancario, que possui um construtor e várias funções. No construtor, são chamadas as funções obterClientesDoProlog() e ordenarClientesPorNumero().

```
public SistemaBancario() {
    clientes = new ArrayList<>();
    obterClientesDoProlog();
    ordenarClientesPorNumero();
}
```

A função obterClientesDoProlog() consulta o arquivo Prolog e cria objetos Cliente para cada entrada de cliente encontrada.

```
private void obterClientesDoProlog() {
    Query query = new Query("consult('src/banco.pl')");
    System.out.println("accesso ao ficheiro banco.pl: " + (query.hasSolution() ? "com
successo" : "falhou ligação"));
    Query queryClientes = new Query("cliente(NumeroCliente, _, _, _, _)");
    while (queryClientes.hasMoreSolutions()) {
        int numeroCliente =
Integer.parseInt(queryClientes.nextSolution().get("NumeroCliente").toString());
        clientes.add(new Cliente(numeroCliente));
    }
```

2. Ordenar clientes por número:

A função ordenarClientesPorNumero() ordena a lista de clientes com base no número do cliente. Essa decisão foi tomada para facilitar a busca e exibição de informações dos clientes, garantindo que a lista esteja sempre ordenada.

```
private void ordenarClientesPorNumero() {
    Collections.sort(clientes, (cliente1, cliente2) -> {
        if (cliente1.getNumeroCliente() < cliente2.getNumeroCliente()) {
            return -1;
        } else if (cliente1.getNumeroCliente() > cliente2.getNumeroCliente()) {
            return 1;
        } else {
            return 0;
        }
    });
}
```

3. Funções principais:

O menu principal do Sistema Bancário é gerido pela função exibirMenu(), que exibe as opções disponíveis e chama as funções correspondentes, como imprimirListaClientes(), mostrarSaldoRealCliente(), mostrarBalancoCreditoCliente(), mostrarClientesPorCidade(), mostrarClientesElegiveisCredito() e selecionarCliente(). Um exemplo de funcionamento é a função mostrarSaldoRealCliente(), que obtém o saldo real de um cliente consultando o arquivo Prolog.

```
private void mostrarSaldoRealCliente() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Digite o número do cliente: ");
    int numeroCliente = scanner.nextInt();
    Cliente cliente = buscarCliente(numeroCliente);
    if (cliente != null) {
        System.out.println("Saldo Real do Cliente " + cliente.getNome() + ":");
        Query query = new Query("saldo_real(" + numeroCliente + ", RealBalance)");
        if (query.hasSolution()) {
            int saldoReal =
Integer.parseInt(query.oneSolution().get("RealBalance").toString());
            System.out.println("Saldo Real: " + saldoReal);
        } else {
            System.out.println("Não foi possível obter o saldo real do cliente.");
        }
    } else {
    System.out.println("Cliente não encontrado.");
    }
}
```

4. Filtrar clientes por cidade:

A função `mostrarClientesPorCidade()` permite ao usuário filtrar clientes com base na cidade. O usuário insere o nome da cidade e a aplicação consulta o arquivo Prolog para obter os clientes dessa cidade.

```
private void mostrarClientesPorCidade() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Digite a cidade: ");
    String cidade = scanner.nextLine();
    System.out.println("Clientes na cidade " + cidade + ":");
    Query query = new Query("clientes_por_cidade('" + cidade + "', NumeroCliente,
Nome)");
    while (query.hasMoreSolutions()) {
        java.util.Map<String, Term> solution = query.nextSolution();
        int numeroCliente = Integer.parseInt(solution.get("NumeroCliente").toString());
        String nome = solution.get("Nome").toString();
        System.out.println("Número de Cliente: " + numeroCliente);
        System.out.println("Nome: " + nome);
        System.out.println("------------------------");
    }
}
```

5. Verificar clientes elegíveis a crédito:

A função mostrarClientesElegiveisCredito() consulta o arquivo Prolog para identificar clientes que atendem a certos critérios e são elegíveis a crédito.

```
private void mostrarClientesElegiveisCredito() {
    System.out.println("Clientes elegíveis a crédito:");
    Query query = new Query("clientes_elegiveis_credito(NumeroCliente, Nome, Agencia,
Cidade, DataAbertura)");
    while (query.hasMoreSolutions()) {
        java.util.Map<String, Term> solution = query.nextSolution();
        int numeroCliente = Integer.parseInt(solution.get("NumeroCliente").toString());
        String nome = solution.get("Nome").toString();
        String agencia = solution.get("Agencia").toString();
        String cidade = solution.get("Cidade").toString();
        String dataAbertura = solution.get("DataAbertura").toString();
        System.out.println("Número de Cliente: " + numeroCliente);
        System.out.println("Nome: " + nome);
        System.out.println("Agência: " + agencia);
        System.out.println("Cidade: " + cidade);
        System.out.println("Data de Abertura: " + dataAbertura);
        System.out.println("------------------------");
    }
}
```

6. Mostra o balanço de crédito do cliente

A função mostrarBalancoCreditoCliente() consulta o arquivo Prolog para identificar clientes e mostrar o valor em credito.

```
private void mostrarBalancoCreditoCliente() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite o número do cliente: ");
        int numeroCliente = scanner.nextInt();
        //
        Cliente cliente = buscarCliente(numeroCliente);
        if (cliente != null) {
            System.out.println("Balanço de Crédito do Cliente " + cliente.getNome() +
":");
            Query query = new Query("saldo_credito(" + numeroCliente + ",
CreditBalance)"); // Use saldo_credito instead of credit_balance
            if (query.hasSolution()) {
                Term creditBalanceTerm = query.oneSolution().get("CreditBalance");
                if (creditBalanceTerm.isInteger()) {
                    int balancoCredito = ((org.jpl7.Integer)
creditBalanceTerm).intValue();
                    System.out.println("Balanço de Crédito: " + balancoCredito);
                }
            } else {
                System.out.println("Não foi possível obter o balanço de crédito do
cliente.");
            }
        } else {
            System.out.println("Cliente não encontrado.");
        }
    }
```

Prefiro programar em inglês, mas no SistemaBancario.java, optei por deixar algumas funções em português, no Cliente.java deixei algumas funções em inglês.

No ficheiro Cliente.java é onde esta a classe Cliente, é a principal classe e realiza várias operações relacionadas a contas bancárias, como verificar saldos, movimentos, depósitos e levantamentos.

A classe Cliente possui os seguintes atributos:

numeroCliente: número do cliente (int)
nome: nome do cliente (String)
agencia: agência bancária do cliente (String)
cidade: cidade do cliente (String)
dataAbertura: data de abertura da conta do cliente (String)
A classe Cliente possui um construtor que aceita o número do cliente como argumento. Ele usa uma consulta Prolog para buscar as informações do cliente e atribui aos atributos da classe.

```
    public Cliente(int numeroCliente) {
        this.numeroCliente = numeroCliente;
        Query query = new Query("cliente(" + numeroCliente + ", Nome, Agencia, Cidade,
DataAbertura)");
        if (query.hasSolution()) {
            nome = query.oneSolution().get("Nome").toString();
            agencia = query.oneSolution().get("Agencia").toString();
            cidade = query.oneSolution().get("Cidade").toString();
            dataAbertura = query.oneSolution().get("DataAbertura").toString();
        }
    }
```

Métodos da classe Cliente:

printRealBalance(): Exibe o saldo real da conta do cliente.
printCreditBalance(): Exibe o saldo de crédito do cliente.
printMovements(): Exibe os movimentos da conta do cliente.
deposit(double amount): Deposita dinheiro na conta do cliente. Retorna true se bem-sucedido, false caso contrário.
withdraw(double amount): Levanta dinheiro da conta do cliente. Retorna true se bem-sucedido, false caso contrário.
checkCreditEligibility(): Verifica se o cliente é elegível para crédito.
grantCredit(double creditValue): Concede crédito ao cliente.
Getters para os atributos da classe: getNumeroCliente(), getNome(), getAgencia(), getCidade() e getDataAbertura().
A classe também possui um método exibirMenuCliente() que exibe um menu interativo para o usuário realizar operações na conta do cliente.

```
--** Menu do Cliente **--
1 - Ver saldo total (real + crédito))
2 - Ver saldo real (sem crédito)
3 - Ver saldo de crédito existente
4 - Ver movimentos do cliente
5 - Fazer depósito na conta
6 - Fazer levantamento da conta
7 - Verificar elegibilidade de crédito do cliente
8 - Conceder crédito ao cliente
9 - Voltar ao menu anterior

Escolha uma opção:
```

Ao escolher uma opção, o usuário pode realizar a operação correspondente na conta do cliente. O menu é executado em loop até que o usuário escolha a opção "Voltar ao menu anterior" (opção 8).

No ficheiro banco.pl temos:

Um programa em Prolog que gerencia informações de clientes em um sistema bancário simples. Ele armazena informações sobre clientes, seus saldos e movimentações, e permite efetuar consultas e realizar operações financeiras, como depósitos, levantamentos e concessão de crédito.

Conteúdo de bancos.pl:

1. Predicados dinâmicos:

balanco_total/2: Armazena o balanço total da conta de um cliente.
balanco_credito/2: Armazena o balanço de crédito de um cliente.
movimento/3: Armazena os movimentos de um cliente (valor e data).

2. Informações de cliente:

São representadas pelos fatos do predicado **cliente/5.**
Cada fato contém o número do cliente, nome, agência, cidade e data de abertura da conta.

3. Predicados para obter os dados dos clientes:

todos_clientes/5: Obtém todos os dados dos clientes.
clientes_por_cidade/3: Obtém o número e nome do cliente com base na cidade.
clientes_elegiveis_credito/5: Retorna os clientes elegíveis a crédito.
saldo_real/2: Obtém o saldo real de um cliente.
saldo_total/2: Função para obter o saldo total de um cliente.
saldo_total_cliente/1: Predicado para obter o saldo total de um cliente a partir do número do cliente.
saldo_credito/2: Obtém o balanço de crédito de um cliente.
saldo_credito_existente/2: Obtém o balanço de crédito de um cliente, atribuindo 0 se inexistente.
movimentos_cliente/3: Obtém os movimentos de um cliente.
sem_movimentos/1: Verifica se um cliente não possui movimentos.

4. Função auxiliar:

getCurrentDate/1: Obtém a data atual.

5. Predicados para efetuar operações:

deposito/2: Realiza um depósito em um cliente.
levantamento/2: Realiza um levantamento (retirada) em um cliente.
elegivel_credito/1: Verifica se um cliente é elegível a crédito.
conceder_credito/3: Concede crédito a um cliente.

Conclusão:

O arquivo banco.pl apresenta um programa em Prolog que gerencia um sistema bancário simples, fornecendo informações sobre clientes e permitindo realizar operações financeiras. O programa utiliza predicados dinâmicos para armazenar os dados dos clientes e oferece uma série de predicados para efetuar consultas e operações, como depósitos, levantamentos e concessão de crédito
A combinação desses arquivos fornece a base para um sistema bancário simples, permitindo realizar operações básicas, como depósitos, levantamentos e concessão de crédito.

Resumo e Testes:

O Sistema Bancário apresentado é um sistema em Java que utiliza a biblioteca JPL para conectar e interagir com o código Prolog para gerenciar informações de clientes e suas contas bancárias. As principais funcionalidades incluem listar clientes, filtrar clientes por cidade, verificar o saldo real e o balanço de crédito de um cliente e identificar clientes elegíveis a crédito. A aplicação utiliza um menu principal para facilitar a navegação e a execução das funcionalidades.
A classe Cliente é a principal classe e realiza várias operações relacionadas a contas bancárias, como verificar saldos, movimentos, depósitos e levantamentos.

Testes realizados Menu Cliente(neste exemplo mostramos o cliente 123):
Verificar saldo Total:
Testamos o método printTotalBalance() para verificar se o salto total da conta do cliente é exibido corretamente, este saldo é o total real + credito.

```
Escolha uma opção: 1
O saldo total do cliente 123 é 3500
```

Verificar saldo real:
Testamos o método printRealBalance() para verificar se o saldo real da conta do cliente é exibido corretamente, este saldo reflete o valor na conta e não mostra o credito do cliente, o que tem logica, pois nas contas bancarias, temos o saldo e não vemos o valor dos créditos no nosso saldo, só quando verificamos um balance é que conseguimos ver, por isso separei o balanceTotal de saldoReal.

```
Escolha uma opção: 2
O saldo real do cliente 123 é 2500
```

Verificar saldo de crédito:
Testamos o método printCreditBalance() para verificar se o saldo de crédito do cliente é exibido corretamente.

```
Escolha uma opção: 3
O balanço de crédito do cliente 123 é 1000
```

Verificar movimentos:
Testamos o método printMovements() para verificar se os movimentos da conta do cliente são exibidos corretamente.

```
Escolha uma opção: 4
Movimentos do Cliente:
Valor retirado: -100 | Data: '01-01-2020'
Valor depositado: 30 | Data: '15052023'
Valor depositado: 50 | Data: '15052023'
```

Realizar depósito:
Testamos o método deposit, com diferentes valores de depósito para verificar se os depósitos são realizados com sucesso e se os saldos são atualizados corretamente.

```
Escolha uma opção: 5
Insira a quantia a depositar: 50
Depósito realizado com sucesso.
```

Realizar levantamento:
Testamos o método withdraw, com diferentes valores de levantamento para verificar se os levantamentos são realizados com sucesso e se os saldos são atualizados corretamente.

```
Escolha uma opção: 6
Insira a quantia a levantar: 500
Levantamento realizado com sucesso.
```

Verificar elegibilidade de crédito:
Testamos o método checkCreditEligibility() para verificar se a elegibilidade de crédito do cliente é determinada corretamente.

```
Escolha uma opção: 6
O cliente não é elegível para crédito.
```

Conceder crédito:
Testamos o método grantCredit, com diferentes valores de crédito para verificar se o crédito é concedido com sucesso e se os saldos são atualizados, se o cliente já tem credito retorna que já não é elegível.

```
Escolha uma opção: 8
Insira o valor do crédito a conceder: 1000
O cliente não é elegível para crédito.
```

Interação com o menu do cliente:
Testamos o método exibirMenuCliente() para verificar se o menu interativo funciona corretamente e se as opções selecionadas executam as operações correspondentes.

Com base nos testes realizados, a solução apresentada funcionou corretamente e atendeu aos requisitos propostos, permitindo ao usuário interagir com as contas bancárias e realizar operações de forma eficiente. Os erros que este programa tem está nas validações, por exemplo, não valida se não colocar uma cidade, a meu ver falta validar e otimizar mais o programa, mas por falta de tempo, ficou nesta versão.

Testes no menu principal também com sucesso.

Resultado ao executar o programa:

```
--** Menu do Sistema Bancário **--
1 - Mostrar Lista de Clientes e suas informações
2 - Mostrar Saldo Real de um Cliente (com crédito)
3 - Mostrar Balanço de Crédito de um Cliente
4 - Mostrar clientes por cidade
5 - Verificar clientes elegíveis a crédito
6 - Selecionar cliente pelo número
7 - Sair

Escolha uma opção: 6
Digite o número do cliente: 123

--** Menu do Cliente **--
1 - Ver saldo total (real + crédito))
2 - Ver saldo real (sem crédito)
3 - Ver saldo de crédito existente
4 - Ver movimentos do cliente
5 - Fazer depósito na conta
6 - Fazer levantamento da conta
7 - Verificar elegibilidade de crédito do cliente
8 - Conceder crédito ao cliente
9 - Voltar ao menu anterior

Escolha uma opção:
```

BIOGRAFIA:

F. Mário Martins, Java 8 - POO + Construções Funcionais, FCA

Introduction to the Objective Caml Programming Language - Jason Hickey

The Art of Prolog - Leon Sterling and Ehud Shapiro

WEBGRAFIA:

https://github.com/StudentUAb/EfolioB_LP_Prolog

https://github.com/StudentUAb/EfolioB_LP_Java

https://www.youtube.com/watch?v=3wEx_B_0qng

https://highlight.hohli.com

https://codebotic.blogspot.com/2015/08/conectar-prolog-con-java-usando-jpl.html

https://jpl7.org/DeploymentMacos
percebi depois de 2 dias que no Mac o mais recomendado é compilar without debugging

https://www.swi-prolog.org/build/macos.html

https://github.com/SWI-Prolog/packages-jpl/issues/2

https://arxiv.org/pdf/2203.17134.pdf

https://www.youtube.com/watch?v=x_ahRnd1gTI&list=PLZ-Bk6jzsb-OScKa7vhpcQXoU2uxYGaFx

https://www.youtube.com/watch?v=RRubcjpTkks