ESCOLA POLITÉCNICA
PUCRS

Programa IT Academy – Processo Seletivo – Edição #19

Nome Completo: Ana Carolina de Oliveira Xavier
E-mail: anacarolina.oxavier@gmail.com

Etapa 1 – Questões de lógica

Esta seleção possui 15 questões de lógica de caráter eliminatório. As questões são apresentadas no formulário de Exercício Técnico e devem ser respondidas no próprio formulário online, que deverá ser acessado através do link a seguir: https://forms.gle/ZMEqSbdpbuGNFMwK7

Etapa 2

RESUMO DA SOLUÇÃO

Para a realização do teste técnico deste processo seletivo, utilizei a linguagem de programação Java, na ferramenta de desenvolvimento (IDE) IntelliJ, e para interface do programa optei por textual/console.

O desenvolvimento do problema proposto foi realizado a partir de seis (6) classes, sendo elas:

Main(): A classe Main é responsável pela interação com o usuário com o menu principal e os menus de opção. A partir das escolhas do usuário nos menus, são realizadas decisões para chamadas de métodos na classe Painel.

Painel(): A classe Painel é responsável pelo contato entre a classe Main e a classe ContaManager. Nesta classe é realizado as chamadas de métodos da classe ContaManager.

ContaManager(): A classe ContaManager é responsável pelo funcionamento do sistema. Nesta classe foi desenvolvida todos os métodos que solucionam o problema proposto.

Conta(): Nesta classe temos as informações, por conta, do usuário. Esta classe possuí todos os Getters e Setters necessários para os acessos/modificações das informações da conta.

Transacao(): Nesta classe temos as informações, por transação, das transações das contas. Esta classe possuí todos os Getters e Setters necessários para os acessos/modificações das informações da transação.

tTransacao(): Classe "enum" armazenando os tipos de transação possíveis (tipo RECEITA e tipo DESPESA).

Portanto, a classe ContaManager é a classe que faz com que o sistema seja funcional. Nesta classe temos 15 métodos, sendo 11 métodos satisfazendo as 11 opções do sistema, 2 métodos auxiliares, 1 método construtor e 1 método para instaciar as contas e transações iniciais do sistema.

Na seção de Testes, a seguir, será explicado mais detalhadamente o funcionamento dos métodos desenvolvidos. O código acompanha comentários visando facilitar o entendimento do desenvolvimento.

TESTES (aqui você deverá colar capturas de tela de todas as funcionalidades desenvolvidas e realizar comentários, use o espaço que julgar necessário)

Ao iniciar o sistema, é possível visualizar o seguinte menu de opções:

```
Gerenciamento de despesas pessoais!
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
    Menu de opções! :)
   1 - Gerenciar Contas
   2 - Gerenciar Transações
   3 -  Painel Geral
   0 -  Sair do Sistema
Opção desejada: (número referente a opção):
```

As opções do menu principal foram separadas de acordo com o número de funcionalidades. Ou seja, na opção 1 é possível acessar as opções de gerenciamento de contas, na opção 2 é possível acessar as opções de gerenciamento de transações e na opção 3 é possível visualizar as opções do painel geral referente a funcionalidade 3. A opção 0 é para finalizar o sistema.

FUNCIONALIDADE 1 – Gerenciar Contas

Menu da primeira funcionalidade:

```
 Menu: Gerenciar Contas :)
   1 - Cadastrar Conta
   2 - Remover Conta
   3 - Mesclar Contas
   0 - Sair do gerenciamento de contas
Opção desejada: (número referente a opção):
```

a. Cadastrar Conta

Funcionamento do cadastro de conta:

```
 == Menu: Cadastrar conta :)
Digite o número da conta: 9875
Digite a agência da conta: Banrisul
O cadastro da conta foi realizado com sucesso :)
 == Informações da conta: Conta: número -> 9875, agência -> Banrisul, saldo -> 0.0
```

Na funcionalidade de cadastro de conta, é solicitado ao usuário que este informe o número da conta e sua agência. Toda conta nova cadastrada é inicializada com saldo 0 (zero).

```
public void cadastrarConta() {
    Scanner entrada = new Scanner(System.in);

    // --> Solicita as informações necessárias para o cadastro da conta.
    System.out.println(" == Menu: Cadastrar conta :)");
    System.out.print("Digite o número da conta: ");
    int nConta = entrada.nextInt();
    entrada.nextLine();

    // --> Verifica se a conta já existe.
    for (Conta conta : continhas) {
        if (conta.getNumeroConta() == nConta) {
            System.out.println("Essa conta já existe, seu cadastro será cancelado :(");
            return;
        }
    }

    System.out.print("Digite a agência da conta: ");
    String aConta = entrada.nextLine();

    // --> Cria nova conta e armazena na lista.
    Conta continha = new Conta(nConta, aConta);
    continhas.add(continha);

    System.out.println("O cadastro da conta foi realizado com sucesso :)");
    System.out.println(" == Informações da conta: " + continha);
}
```

No método para cadastrar uma conta, após o usuário informar o número da conta, antes de continuar o cadastro, é realizado a verificação de que essa conta ainda não exista. Se o usuário informar um número de conta que já exista, este receberá uma mensagem informando que o cadastro será cancelado. Após a criação da conta, esta é armazenada em uma lista de contas chamada “continhas”.

b. Remover Conta

```
 == Menu: Remover conta :)
Digite o número da conta a ser removida: 9875
 == ALERTA: Todos os dados da conta 9875 serão apagados
    == Deseja continuar (S ou N)? s
A conta 9875 foi removida com sucesso :)
```

Na funcionalidade de remover conta, é solicitado ao usuário o número da conta que este deseja apagar, e logo em seguida é informado um alerta de que todos os dados da conta serão apagados e solicitado se a pessoa deseja continuar ou não.

```java
public void removerConta() {
    Scanner entrada = new Scanner(System.in);
    // --> Solicita as informações necessárias para remover conta.
    System.out.println(" == Menu: Remover conta :)");
    System.out.print("Digite o número da conta a ser removida: ");
    int nConta = entrada.nextInt();
    entrada.nextLine();
    Conta cEncontrada = null;      // --> Encontra a conta.
    for (Conta conta : continhas) {
        if (conta.getNumeroConta() == nConta) {
            cEncontrada = conta;
            break;
        }
    }
    // --> Se encontrar a conta.
    if (cEncontrada != null) {
        System.out.println(" == ALERTA: Todos os dados da conta " + cEncontrada.getNumeroConta() + " serão apagados");
        System.out.print("    == Deseja continuar (S ou N)? ");
        String opcao = entrada.nextLine().toUpperCase();
        // --> Realiza a operação de acordo com a opção do usuário.
        if (opcao.equals("S")) {
            continhas.remove(cEncontrada);
            System.out.println("A conta " + cEncontrada.getNumeroConta() + " foi removida com sucesso :)");
        } else if (opcao.equals("N")) {
            System.out.println("A operação foi cancelada!");
        }
        // --> Se não encontrar a conta.
    } else {
        System.out.println("A conta " + cEncontrada.getNumeroConta() + " não foi encontrada. Operação cancelada ):");
    }
}
```

No método para remover uma conta, após o usuário informar o número da conta que este deseja remover e a conta ser encontrada, é verificado se a conta existe para seguir com a remoção. Antes de ser realizada a remoção de fato da conta, é alertado ao usuário de que todos os dados da conta serão apagados e se ele deseja continuar com a ação, podendo responder entre “S” para sim e “N” para não.

c. Mesclar Contas

```
 == Menu: Mesclar contas :)
ALERTA: A PRIMEIRA conta será mantida!
Digite o número da primeira conta: 1234
Digite o número da segunda conta: 5678
As contas foram mescladas com sucesso :)
```

Na funcionalidade de mesclar contas, inicialmente é informado ao usuário que a primeira conta que ele informar será a mantida. E em seguida é solicitado o número das contas que este deseja mesclar.

```java
public void mesclarConta() {
    Scanner entrada = new Scanner(System.in);
    Conta conta1 = null;
    Conta conta2 = null;

    // --> Solicita as informações necessárias para mesclar contas.
    System.out.println(" == Menu: Mesclar contas :)");
    System.out.println("ALERTA: A PRIMEIRA conta será mantida! ");

    System.out.print("Digite o número da primeira conta: ");
    int nConta1 = entrada.nextInt();
    entrada.nextLine();

    System.out.print("Digite o número da segunda conta: ");
    int nConta2 = entrada.nextInt();
    entrada.nextLine();

    // --> Encontra as contas.
    for (Conta conta : continhas) {
        if (conta.getNumeroConta() == nConta1) {
            conta1 = conta;
        } else if (conta.getNumeroConta() == nConta2) {
            conta2 = conta;
        }
    }
```

No método de mesclar contas, é solicitado ao usuário que este informe dois números de contas, sendo informado previamente que somente a primeira conta será mantida. Após o usuário informar as contas, então inicia-se uma iteração na lista de contas para buscar ambas as contas.

```
// --> Se encontrar as contas.
if (conta1 != null && conta2 != null) {
    // --> Realiza uma lista com as transações da conta 1.
    List<Transacao> transacoes = new ArrayList<>(conta1.getTransacoes());
    // --> Adiciona todas as transações da conta 2.
    transacoes.addAll(conta2.getTransacoes());

    // --> Filtra todas as transações e as ordena por data.
    transacoes = transacoes.stream()
            .filter(transacao -> transacao.getData() != null)
            .collect(Collectors.toList());
    Collections.sort(transacoes, Comparator.comparing(Transacao::getData));

    // --> Seta as transações da conta 1 com todas as transações filtradas e mescladas.
    conta1.setTransacoes(transacoes);
    // --> Altera o saldo da conta 1.
    conta1.setSaldo(conta1.getSaldo() + conta2.getSaldo());
    // --> Remove a conta 2.
    continhas.remove(conta2);

    System.out.println("As contas foram mescladas com sucesso :)");
    // --> Se não encontrar as contas.
}else{
    System.out.println("Não foi possível encontrar uma, ou ambas, contas. Operação cancelada ):");
}
```

Encontrando as contas, é criada uma lista de transações auxiliar com as transações da primeira conta e logo em seguida são adicionadas todas as transações da conta 2 nessa mesma lista.

O próximo passo é, a partir de um fluxo (stream) na lista, seja possível ordená-las. Então, é realizada a filtragem de transações que tenham a data diferente de nulo. Em seguida, após filtrar as transações, é realizado a coleta destes elementos formando uma nova lista. A linha "Collections.sort(transacoes, Comparator.comparing(Transacao::getData))" realiza a ordenação da lista "transacoes" com base em um critério específico. O método "sort" da classe "Collections" é utilizado para essa finalidade. O comparador é definido pela expressão "Comparator.comparing(Transacao::getData)", que faz a comparação das transações usando o método "getData" da classe "Transacao". Dessa forma, a lista será ordenada de acordo com as datas das transações.

Com a lista ordenada, então, a conta 1 recebe todas as transações filtradas e ordenadas, e seu saldo é modificado com a soma das duas contas. Enquanto a conta 2 é removida.

FUNCIONALIDADE 2 – Gerenciar Transações

Menu da segunda funcionalidade:

```
-------------------------------------------
 Menu: Gerenciar Transações :)
   1 - Extrato da Conta
   2 - Incluir Transação
   3 - Editar última transação
   4 - Transferir fundos
   0 - Sair do gerenciamento de transações
Opção desejada: (número referente a opção):
```

Para as próximas funcionalidades foi desenvolvido o método auxiliar “contaSelecionada” onde seu objetivo é diminuir o tamanho dos demais métodos toda vez que é necessário selecionar uma conta.

```java
// --> Método auxiliar. Selecionar conta desejada.
4 usages
public Conta contaSelecionada() {
    Scanner entrada = new Scanner(System.in);

    System.out.println(" == Menu: Gerenciar transações :)");
    System.out.print("Digite o número da conta que deseja: ");
    int nConta = entrada.nextInt();
    entrada.nextLine();

    // --> Encontra conta.
    Conta cEncontrada = null;
    for (Conta conta : continhas) {
        if (conta.getNumeroConta() == nConta) {
            cEncontrada = conta;
            break;
        }
    }
    return cEncontrada;
}
```

Então, este método recebe o número da conta que o usuário deseja e tenta busca-lá.

a. Extrato da Conta

```
-----------------------------------------
 == Menu: Gerenciar transações :)
Digite o número da conta que deseja: 1234
 == Menu: Extrato da Conta :)
--------------------------------------------------------------------------------
 Extrato da conta: 1234
--------------------------------------------------------------------------------
  Data      |  Tipo      | Categoria    |  Descrição             |  Valor
--------------------------------------------------------------------------------
2022-12-02 | RECEITA    |  Salário     | Pagamento ref. Nov/23  | 2700,00
2022-12-18 | DESPESA    |  Assinatura  | Taxa de assinatura     | -67,90
2023-01-10 | RECEITA    |  Salário     | Pagamento ref. Dez/23  | 2700,00
2023-02-16 | DESPESA    |  Lanche      | Alimentação            | -9,76
2023-03-08 | DESPESA    |  Lanche      | Alimentação            | -10,70
2023-03-10 | RECEITA    |  Salário     | Pagamento ref. Fev/23  | 2700,00
2023-03-31 | DESPESA    |  Lanche      | Alimentação            | -43,90
2023-04-28 | DESPESA    |  Contas      | Água                   | -70,89
2023-05-29 | DESPESA    |  Mercado     | Alimentação            | -679,90
2023-06-05 | DESPESA    |  Lazer       | Cinema                 | -34,98
--------------------------------------------------------------------------------
 Saldo total: R$ 7181,97
--------------------------------------------------------------------------------
```

Para o usuário visualizar o extrato da conta, é necessário que este informe o número da conta desejada.

```java
public void extratoDaConta() {
    Conta cEncontrada = contaSelecionada();

    // --> Extrato bonitinho :)
    System.out.println(" == Menu: Extrato da Conta :)");

    if (cEncontrada != null) {
        List<Transacao> transacaos = cEncontrada.getTransacoes();
        Collections.sort(transacaos, Comparator.comparing(Transacao::getData));
        System.out.println("--------------------------------------------------------------------------------");
        System.out.println(" Extrato da conta: " + cEncontrada.getNumeroConta());
        System.out.println("--------------------------------------------------------------------------------");
        System.out.println("  Data      |  Tipo      | Categoria    |  Descrição             |  Valor  ");
        System.out.println("--------------------------------------------------------------------------------");
        for (Transacao t : transacaos) {
            System.out.printf("%s | %-9s |  %-10s | %-21s  | %.2f\n", t.getData(), t.getTipo(), t.getCategoria(), t.getDescricao(), t.getValor());
        }
        System.out.println("--------------------------------------------------------------------------------");
        System.out.printf(" Saldo total: R$ %.2f\n", cEncontrada.getSaldo());
        System.out.println("--------------------------------------------------------------------------------");

    } else {
        System.out.println("A conta não foi encontrada. Operação cancelada :(");
    }
}
```

No método de extrato de conta, após o usuário informar o número da conta, caso tenha sido possível encontrá-la, então é realizada uma iteração na lista de contas imprimindo para o usuário todos as informações sobre as transações da conta selecionada.

b. Incluir Transação

```
 == Menu: Gerenciar transações :)
Digite o número da conta que deseja: 1234
 == Menu: Incluir Transação :)
 = Data da transação (dd/mm/aaaa): 13/06/2023
 = Valor da transação: 100,00
 = Tipo de transação (receita ou despesa): despesa
 = Categoria da transação: Alimentação
 = Descrição da transação: Lanche
A transação para a conta 1234 no valor de R$100,00 foi realizada! Novo saldo da conta: R$7081,97
```

Na funcionalidade de incluir transação, o usuário deve informar todos os dados necessários sobre a transação seguindo as instruções como por exemplo, utilizar “dd/mm/aaaa” como formatação para informar a data.

```java
public void incluirTransacao() {
    Scanner entrada = new Scanner(System.in);
    Conta cEncontrada = contaSelecionada();
    System.out.println(" == Menu: Incluir Transação :)");
    // --> Se conta for encontrada.
    if (cEncontrada != null) {
        // --> Solicita as informação da transação.
        System.out.print(" = Data da transação (dd/mm/aaaa): ");
        String data = entrada.nextLine();

        System.out.print(" = Valor da transação: ");
        double valor = entrada.nextDouble();
        entrada.nextLine();

        System.out.print(" = Tipo de transação (receita ou despesa): ");
        String tipoUser = entrada.nextLine().toUpperCase();

        // --> Verifica o tipo da transação.
        tTransacao tipo;
        if (tipoUser.equals("RECEITA")) {
            tipo = tTransacao.RECEITA;
        } else if (tipoUser.equals("DESPESA")) {
            tipo = tTransacao.DESPESA;
        } else {
            System.out.println("Tipo de transação inválida. A operaçao será cancelada :(");
            return;
        }
```

```java
System.out.print(" = Categoria da transação: ");
String cat = entrada.nextLine();

System.out.print(" = Descrição da transação: ");
String desc = entrada.nextLine();
```

O método para a inclusão de transação inicia-se solicitando ao usuário que este informe todas as informações sobre a transação. No momento de informar o tipo de transação, é realizada a verificação de que a transação é do tipo receita ou despesa, caso contrário será informado que o tipo de transação é inválido.

```
    // --> Caso a transação for uma despesa, o valor será subtraído do saldo da conta.
    if (tipo == tTransacao.DESPESA) {
        Transacao transacao = new Transacao(-valor, data, tipo, cat, desc);
        cEncontrada.adicionarTransacao(transacao);
    // --> Senão, o valor será somado ao saldo da conta.
    } else {
        Transacao transacao = new Transacao(valor, data, tipo, cat, desc);
        cEncontrada.adicionarTransacao(transacao);
    }

    System.out.printf("A transação para a conta %s no valor de R$%.2f foi realizada! Novo saldo da conta: R$%.2f%n",
            cEncontrada.getNumeroConta(), valor, cEncontrada.getSaldo());
// --> Se conta não for encontrada, cancela operação.
} else {
    System.out.println("Não foi possível encontrar a conta mencionada. Operação cancelada :(");
}
```

Após receber todas as informações, o método então verifica se a transação é uma despesa: caso sim, então é criada uma transação com o valor “negativo”, ou seja, que vai subtrair do saldo da conta daquela transação; caso não, então é uma receita e esta será uma nova transação que somará ao saldo da conta.

c. Editar última transação

```
 == Menu: Gerenciar transações :)
Digite o número da conta que deseja: 1234
 == Menu: Editar última transação :)
Valor e tipo da última transação: R$-100,00, DESPESA
Novo tipo de transação (receita ou despesa): receita
Novo valor da transação: 50,0
Nova descrição da transação: Novo lanche
A edição da útima transação da conta 1234 no valor de R$50,00 foi realizada! Novo saldo da conta: R$7131,97
```

Na funcionalidade de editar última transação, o usuário tem a possibilidade de escolher uma conta e modificar o valor, o tipo e a descrição da última transação realizada na conta selecionada. Antes de realizar a edição, é informado ao usuário qual foi a última transação realizada na conta selecionada.

```java
public void editarUltimaTransacao() {
    Scanner entrada = new Scanner(System.in);

    Conta cEncontrada = contaSelecionada();

    System.out.println(" == Menu: Editar última transação :)");

    // --> Se a conta for encontrada.
    if (cEncontrada != null) {
        // --> Cria uma lista de transações com as tranações da conta.
        List<Transacao> transacoes = cEncontrada.getTransacoes();
        // --> Se haver transações na lista
        if (!transacoes.isEmpty()) {
            Transacao uTransacao = transacoes.get(transacoes.size() - 1);   //Identifica a última transação
            double vAntigo = uTransacao.getValor();                         //Identifica o seu valor

            // --> Caso a última transação for uma despesa
            if (uTransacao.getTipo() == tTransacao.DESPESA) {
                cEncontrada.setSaldo(cEncontrada.getSaldo() + vAntigo);     //Soma o valor subtraído anteriormente
            } else if (uTransacao.getTipo() == tTransacao.RECEITA) {
                cEncontrada.setSaldo(cEncontrada.getSaldo() - vAntigo);     //Subtraí o valor somado anteriormente
            }
```

O método inicia selecionando a conta desejada pelo usuário. Em seguida, caso encontre a conta, é criada uma lista auxiliar com as transações da conta. Se a lista não estiver vazia, então é identificado a última transação da conta e o valor desta transação. Antes de realizar qualquer contato com o usuário sobre a futura edição, então é realizada a verificação se a última transação era uma despesa ou uma receita. Caso seja uma receita: o valor da última transação será subtraído do saldo da conta; caso seja uma despesa: o valor da última transação será somado ao saldo da conta. Isto porque, como será realizado a edição da última transação, então é necessário “voltar” ao saldo original da conta antes da última transação.

```java
System.out.printf("Valor e tipo da última transação: R$%.2f, %s\n", uTransacao.getValor(), uTransacao.getTipo());
System.out.print("Novo tipo de transação (receita ou despesa): ");
String nTipo = entrada.nextLine().toUpperCase();
// --> Identifica o novo tipo de transação.
tTransacao tipo;
if (nTipo.equals("RECEITA")) {
    tipo = tTransacao.RECEITA;
} else if (nTipo.equals("DESPESA")) {
    tipo = tTransacao.DESPESA;
}else {
    System.out.println("Tipo de transação inválida. A operaçao será cancelada :(");
    return;
}
uTransacao.setTipo(tipo);
System.out.print("Novo valor da transação: ");
double valor = entrada.nextDouble();
entrada.nextLine();

// --> Calcula a diferença dos valores.
double dValorT = valor - vAntigo;
cEncontrada.setSaldo(cEncontrada.getSaldo() + dValorT);  //Modifica o saldo da conta a partir da diferença dos valores.
uTransacao.setValor(valor);                              //Modifica o valor da última transação.

System.out.print("Nova descrição da transação: ");
String descricao = entrada.nextLine();
uTransacao.setDescricao(descricao);
System.out.printf("A edição da útima transação da conta %s no valor de R$%.2f foi realizada! Novo saldo da conta: R$%.2f%n",
        cEncontrada.getNumeroConta(), valor, cEncontrada.getSaldo());
```

Em seguida, o usuário tem a oportunidade de informar o novo tipo da transação, e assim é realizada a verificação de se o tipo informado é válido, e logo é modificado o tipo da transação. Após, o usuário deve informar o novo valor da transação, sendo assim calculado a diferença do valor atual e do valor antigo. Modificando o saldo da conta, e o valor da última transação. Por fim, o usuário informa a nova descrição e esta também é modificada.

d. Transferir fundos

```
 == Menu: Transferir fundos :)
Digite o número da primeira conta: 1234
Digite o número da segunda conta: 5678
 = Valor a ser transferido: 30,0
A transferencia da conta 1234 para a conta 5678 foi realizada :)
 Novo saldo da conta 1234: R$7151,97
 Novo saldo da conta 5678: R$5044,03
```

A funcionalidade de transferir fundos permite que o usuário selecione duas contas e realizar a transferência de valor da primeira para a segunda.

```
public void transferirFundos() {
    Scanner entrada = new Scanner(System.in);
    Conta oConta = null;
    Conta dConta = null;

    // --> Solicita as informações necessárias para tranferir fundos.
    System.out.println(" == Menu: Transferir fundos :)");

    System.out.print("Digite o número da primeira conta: ");
    int nContaO = entrada.nextInt();
    entrada.nextLine();

    System.out.print("Digite o número da segunda conta: ");
    int nContaD = entrada.nextInt();
    entrada.nextLine();

    // --> Encontra ambas as contas.
    for (Conta conta : continhas) {
        if (conta.getNumeroConta() == nContaO) {
            oConta = conta;
        } else if (conta.getNumeroConta() == nContaD) {
            dConta = conta;
        }
        if (oConta == null && dConta == null) {
            System.out.println(" Não foi possível encontrar uma, ou ambas, conta :(");
            break;
        }
    }
```

O método inicia perguntando ao usuário quais serão as duas contas desejadas, e verificando a existência de ambas.

```java
// --> Se encontrar as contas.
if (oConta != null && dConta != null) {
    System.out.print(" = Valor a ser transferido: ");
    double valor = entrada.nextDouble();

    // --> Se o saldo da conta de origem for maior ou igual ao valor desejado para trasferir, continua.
    if (oConta.getSaldo() >= valor) {
        // --> Formatação de data/pega a data atual para criar a transação.
        LocalDate dAtual = LocalDate.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        String dataAtual = dAtual.format(formatter);

        // --> Cria uma transação de saque (para a conta origem) e uma transação deposito (para conta destino)
        Transacao saque = new Transacao(-valor, dataAtual, tTransacao.DESPESA,  categoria: "Transferencia",  descricao: "Transferência para conta " + dConta.getNumeroConta());
        Transacao deposito = new Transacao(valor, dataAtual, tTransacao.RECEITA,  categoria: "Transferencia",  descricao: "Transferência da conta " + oConta.getNumeroConta());

        // --> Adiciona ambas transações.
        oConta.adicionarTransacao(saque);
        dConta.adicionarTransacao(deposito);

        System.out.printf("A transferencia da conta %s para a conta %s foi realizada :)" +
                "\n Novo saldo da conta %s: R$%.2f \n Novo saldo da conta %s: R$%.2f\n", oConta.getNumeroConta(), dConta.getNumeroConta(),
                oConta.getNumeroConta(), oConta.getSaldo(), dConta.getNumeroConta(), dConta.getSaldo());
    } else {
        System.out.println("Saldo insuficiente para realizar a transferencia. Operação cancelada :(");
    }
```

Encontrando as contas, então o usuário deve informar o valor da transferência. Em seguida é verificado se o saldo da conta de origem é maior ou igual ao valor desejado para transferência, este sendo então é criada uma variável para identificar a data atual da transferência. Após, são criadas duas transações: saque e depósito. A transação de saque é realizada na conta de origem onde o valor da transferência será subtraído do saldo da conta de origem; a transação depósito é realizada na conta de destino e o valor de transferência é somado ao saldo da conta de destino. Ao fim, cada transação é adicionada a lista de transações de sua respectiva conta.

FUNCIONALIDADE 3 – Painel Geral

Menu da terceira funcionalidade:

```
-----------------------------------------------
 Menu: Painel de Estatísticas :)
   1 - Resumo das contas
   2 - Resumo de receitas e despesas do mês
   3 - Saldo geral dos últimos 6 meses
   4 - Resumo por mês e tipo
   0 - Sair do painel
Opção desejada: (número referente a opção):
```

*A funcionalidade adicional escolhida foi: “Resumo por mês e tipo”. Nesta funcionalidade o usuário informa um mês (e o ano) e qual tipo de transação que deseja visualizar, sendo as opções: apenas receitas, apenas despesas ou todas as transações.

Para os métodos da terceira funcionalidade, foi desenvolvido o método auxiliar "getTransacoes(Conta c, int m)" onde recebe por parâmetro a conta e o mês que deseja identificar as transações. Retorna a lista de transações do mês m na conta c.

```
// --> Método auxiliar. Identificar as transações a partir de uma conta e mês selecionado.
2 usages
private List<Transacao> getTransacoes(Conta c, int m) {
    List<Transacao> tMes = new ArrayList<>();
    for (Transacao t : c.getTransacoes()) {
        LocalDate dTransacao = t.getData();
        if (dTransacao.getMonthValue() == m) {
            tMes.add(t);
        }
    }
    return tMes;
}
```

a. Resumo das contas

```
-----------------------------------------
 == Menu: Resumo das contas :)
-----------------------------------
Número da conta: 1234
Saldo: R$7181,97
-----------------------------------
-----------------------------------
Número da conta: 5678
Saldo: R$5014,03
-----------------------------------
Saldo total das contas: R$12196,00
-----------------------------------
```

Na funcionalidade de resumo de contas, o usuário tem a oportunidade de visualizar o saldo que cada conta que este possuí e o saldo total considerando todas as contas.

```java
public void resumoContas() {
    System.out.println(" == Menu: Resumo das contas :)");
    double saldoT = 0;

    // --> Imprime o número e saldo de todas as contas.
    for (Conta conta : continhas) {
        System.out.println("---------------------------------");
        System.out.println("Número da conta: " + conta.getNumeroConta());
        System.out.printf("Saldo: R$%.2f\n", conta.getSaldo());
        System.out.println("---------------------------------");

        saldoT += conta.getSaldo();
    }
    // --> Informa o saldo total das contas.
    System.out.printf("Saldo total das contas: R$%.2f\n", saldoT);
    System.out.println("---------------------------------");
}
```

O método para a visualização do resumo das contas funciona a partir de uma iteração na lista de contas, onde a cada conta é impresso ao usuário a informação do número da conta e seu saldo. Ao fim, é informado o saldo total das contas.

b. Resumo de receitas e despesas do mês

```
== Menu: Resumo de receitas e despesas do mês :)
= Mês: 6
= Total de receitas do mês: R$0,00
= Total de despesas do mês: R$-37,88
```

Na funcionalidade de resumo de receitas e despesas do mês, o usuário tem a oportunidade de visualizar o resumo de entrada e saída de suas contas. Neste caso exemplificado acima, temos apenas saídas de transações, ou seja, despesas, no mês atual, que no caso é Junho (6).

```java
public void resumoReceitasDespesasMes() {
    System.out.println(" == Menu: Resumo de receitas e despesas do mês :)");
    double tReceitas = 0;  //Total receita.
    double tDespesas = 0;  //Total despesa.
    int mesAtual = LocalDate.now().getMonthValue();   //Identifica o mês atual.

    // --> Para cada conta na lista de contas
    for (Conta conta : continhas) {
        List<Transacao> tMesAtual = getTransacoes(conta, mesAtual); //Identificamos a lista de transações do mês.
        // --> Para cada transação do mês
        for (Transacao t : tMesAtual) {
            if (t.getTipo() == tTransacao.RECEITA) {
                tReceitas += t.getValor();   //Soma ao total de receitas.
            } else if (t.getTipo() == tTransacao.DESPESA) {
                tDespesas += t.getValor();   //Soma ao total de despesas.
            }
        }
    }
    System.out.println(" = Mês: " + mesAtual);
    System.out.printf(" = Total de receitas do mês: R$%.2f\n", tReceitas);
    System.out.printf(" = Total de despesas do mês: R$%.2f\n", tDespesas);
}
```

O método inicializa duas variáveis, sendo elas “tReceitas” e “tDespesas” para armazenar o valor total das receitas e despesas, e também inicializa a variável “mesAtual” que armazena qual o mês atual. Em seguida, é realiza a iteração na lista de contas, onde para cada conta é criada uma lista com as transações a partir do auxilio do método “getTransacoes(conta, mesAtual)”. Assim inicializa-se a iteração sobre esse lista de transações, onde para cada transação é identificado se esta é uma receita ou uma despesa, somando as suas respectivas variáveis.

c. Saldo geral dos últimos 6 meses

```
 == Menu: Saldo geral dos últimos 6 meses :)
 = Conta: 1234 > R$: -34,98
 = Conta: 5678 > R$: -2,90
^ Mês: 6/2023
Saldo: R$-37,88

 = Conta: 1234 > R$: -679,90
 = Conta: 5678 > R$: -54,90
 = Conta: 5678 > R$: -1,90
^ Mês: 5/2023
Saldo: R$-736,70

 = Conta: 1234 > R$: -70,89
 = Conta: 5678 > R$: -58,60
^ Mês: 4/2023
Saldo: R$-129,49

 = Conta: 1234 > R$: -43,90
 = Conta: 1234 > R$: -10,70
 = Conta: 1234 > R$: 2700,00
 = Conta: 5678 > R$: -189,89
^ Mês: 3/2023
Saldo: R$2455,51
```

```
 = Conta: 1234 > R$: -9,76
 = Conta: 5678 > R$: 3100,00
 = Conta: 5678 > R$: -10,90
^ Mês: 2/2023
Saldo: R$3079,34

 = Conta: 1234 > R$: 2700,00
 = Conta: 5678 > R$: -798,90
^ Mês: 1/2023
Saldo: R$1901,10
```

A funcionalidade de saldo geral dos últimos 6 meses permite que o usuário visualize o saldo de suas contas nos últimos 6 meses, mostrando com detalhes cada transação.

```
public void saldoUltimos6Meses() {
    System.out.println(" == Menu: Saldo geral dos últimos 6 meses :)");
    LocalDate dAtual = LocalDate.now();  //Identifica a data atual.

    for (int i = 0; i < 6; i++) {  // De 0 a 5 (6 meses).
        double saldo = 0;
        LocalDate mAtual = dAtual.minusMonths(i); //Identifica o mês a partir do índice
        int ano = mAtual.getYear();               //Identifica o ano
        int mes = mAtual.getMonthValue();         //Identifica o mês de fato

        // --> A cada conta, identifica as transações do mês e para cada transação, soma na variável.
        for (Conta conta : continhas) {
            List<Transacao> tMesAtual = getTransacoes(conta, mes);
            for (Transacao t : tMesAtual) {
                System.out.printf(" = Conta: %s > R$: %.2f\n", conta.getNumeroConta(), t.getValor());
                saldo += t.getValor();
            }
        }
        System.out.println("^ Mês: " + mes + "/" + ano);
        System.out.printf("Saldo: R$%.2f\n\n", saldo);
    }
}
```

O método inicia identificando qual é o mês atual. Assim, inicia um laço de repetição (indo de 0 a 5) para passar por 6 meses corretamente. A cada repetição, é identificado o mês e o ano atual. Em seguida inicia-se a iteração na lista de contas onde para cada conta é criada a lista de transações com auxilio do método “getTransacoes(conta, mês)”, e esta lista é iterada onde para cada transação é impresso para o usuário o numero da conta e o valor da transação, somando este valor ao saldo total que é informado ao fim da iteração das contas ao usuário.

d. Resumo por mês e tipo

```
 == Menu: Gerenciar transações :)
Digite o número da conta que deseja: 1234
 == Menu: Resumo por mês e tipo :)
 = Digite o mês (mm/aaaa): 02/2023
 = Deseja visualizar somente as receitas ou as despesas? (receitas/despesas/todas): receitas
Não há transações nesse mês e tipo!
```

```
 == Menu: Gerenciar transações :)
Digite o número da conta que deseja: 1234
 == Menu: Resumo por mês e tipo :)
 = Digite o mês (mm/aaaa): 02/2023
 = Deseja visualizar somente as receitas ou as despesas? (receitas/despesas/todas): despesas
--------------------------------------------------------------------------------
 Conta: 1234
--------------------------------------------------------------------------------
  Data      |  Tipo     | Categoria    |  Descrição                |  Valor
--------------------------------------------------------------------------------
2023-02-16 | DESPESA   |  Lanche      | Alimentação            | -9,76
```

Nesta funcionalidade o usuário tem a opção de selecionar a conta e mês que deseja visualizar o resumo, e também se deseja visualizar somente as receitas, somente as despesas ou ambas.

```
public void resumoPorMesTipo() {
    Scanner entrada = new Scanner(System.in);
    Conta cEncontrada = contaSelecionada();

    System.out.println(" == Menu: Resumo por mês e tipo :)");

    // --> Se encontrar conta.
    if (cEncontrada != null) {
        System.out.print(" = Digite o mês (mm/aaaa): ");
        String mesAno = entrada.nextLine();

        System.out.print(" = Deseja visualizar somente as receitas ou as despesas? (receitas/despesas/todas): ");
        String tExtrato = entrada.nextLine().toUpperCase();

        // --> Cria uma lista de transações.
        List<Transacao> transacaos = new ArrayList<>();
        DateTimeFormatter f = DateTimeFormatter.ofPattern("MM/yyyy"); //Formatação de data
```

O método inicia tentando encontrar a conta desejada pelo usuário. Caso a conta seja encontrada, então é solicitado ao usuário o mês que este deseja e qual o tipo de transação. Em seguida é criada uma lista de transações auxiliar e uma variável de formatação de data.

```
    // --> Para cada transação da lista de transações da conta selecionada.
    for (Transacao t : cEncontrada.getTransacoes()) {
        LocalDate dTransacao = t.getData();          //Identifica a data da trnsação.
        String maTransacao = dTransacao.format(f);   //Seleciona o mes e ano da transação.

        // --> Identificando o mes/ano, então adiciona a transação na lista a partir da escolha do usuário.
        if (maTransacao.equals(mesAno)) {
            if (tExtrato.equals("RECEITAS") && t.getTipo() == tTransacao.RECEITA) {
                transacaos.add(t);
            } else if (tExtrato.equals("DESPESAS") && t.getTipo() == tTransacao.DESPESA) {
                transacaos.add(t);
            } else if (tExtrato.equals("TODAS")) {
                transacaos.add(t);
            }
        }
    }
    if (transacaos.isEmpty()) {
        System.out.println("Não há transações nesse mês e tipo!");
    } else {
        System.out.println("-------------------------------------------------------------------------------");
        System.out.println(" Conta: " + cEncontrada.getNumeroConta());
        System.out.println("-------------------------------------------------------------------------------");
        System.out.println("  Data      |  Tipo     | Categoria    |  Descrição               |  Valor  ");
        System.out.println("-------------------------------------------------------------------------------");
        for (Transacao t : transacaos) {
            System.out.printf("%s | %-9s |  %-10s | %-21s  | %.2f\n", t.getData(), t.getTipo(), t.getCategoria(), t.getDescricao(), t.getValor());
        }
    }
} else {
    System.out.println("A conta não foi encontrada. Operação cancelada :(");
```

Então, inicializa-se uma iteração na lista de transações da conta selecionada, e para cada transação é identificada sua data. Em seguida, verifica-se se a data da transação é a mesma do mês desejado pelo usuário. Seguindo da verificação do tipo da transação. Ao fim do método, o resumo das transações identificadas da conta, mês e tipo desejados pelo usuário são impressas no sistema.

AUTOAVALIAÇÃO

Você concluiu a implementação de 100% das funcionalidades solicitadas?

(X) Sim ( ) Não

Para as 3 principais funcionalidades solicitadas, como você avalia a sua solução?

Marque um 'X'.

	Inexistente/ Insuficiente	Pouco satisfeito(a)	Satisfeito(a)	Muito satisfeito(a)
Funcionalidade 1				X
Funcionalidade 2				X
Funcionalidade 3				X

Principais dificuldades

Eu acredito que pude desenvolver um bom sistema, satisfazendo todas os requisitos propostos com êxito sem nenhum grande problema.

Desempenho Geral

Esta está sendo a segunda vez que participo do processo seletivo para uma turma do IT Academy. A primeira vez foi no inicio do ano com a turma 18, onde eu fiquei contente com meu desempenho porém não havia conseguido entregar todas as funcionalidades. Para esta edição do IT Academy, acredito que meu desempenho foi muito satisfatório. Pude me organizar calmamente para desenvolver uma boa solução para todas as funcionalidades, conseguindo entregar o código funcional e atendendo 100% das funcionalidades.

Obrigado por participar deste processo seletivo.
Salve o documento em PDF com o seu nome.