**Trabalho Final de Programação Lab. Banco de Dados**

1. **Objetivo do trabalho**

Neste trabalho, você deve desenvolver uma aplicação de banco chamada *Banco Malvader* em Java, com interface gráfica utilizando **Swing** e **salvamento de dados em um banco de dados**. O sistema deve implementar funcionalidades de gerenciamento de contas bancárias, tal como apresentado anteriormente, aplicando conceitos de Programação Orientada a Objetos (POO) e persistência de dados.

O sistema em Java deve estar conectado a um banco de dados MySQL para realizar operações de cadastro, consulta e geração de relatórios.

1. **Requisitos do Sistema**

**Acesso ao Programa**

1. **Autenticação Inicial**:
   * Exibir o menu principal com três opções:
     + **1. Funcionário**: Permite acesso ao menu do funcionário.
     + **2. Cliente**: Permite acesso ao menu do cliente.
     + **3. Sair do Programa**: Encerra a aplicação.
   * Solicitar senha para acessar as funcionalidades.

**Menu Funcionário**

1. **Abertura de Conta**:
   * Permitir ao funcionário abrir novas contas bancárias.
   * Opções de tipo de conta:
     + **Conta Poupança (CP)** e **Conta Corrente (CC)**.
   * Solicitar dados específicos:
     + Poupança: agência, número da conta, nome do cliente, CPF, data de nascimento, telefone, endereço completo (incluindo CEP, local, número da casa, bairro, cidade, estado), e senha do cliente.
     + Corrente: itens acima, além de limite da conta e data de vencimento.
2. **Encerramento de Conta**:
   * Solicitar senha de administrador para encerrar contas.
   * Permitir busca por número da conta para encerramento.
   * Confirmar o encerramento ao usuário após operação bem-sucedida.
3. **Consulta de Dados**:
   * Submenu com três opções:
     + **Consultar Conta**: Exibir tipo de conta, nome, CPF, saldo, limite disponível e data de vencimento.
     + **Consultar Funcionário**: Exibir código, cargo, nome, CPF, data de nascimento, telefone, e endereço completo.
     + **Consultar Cliente**: Exibir nome, CPF, data de nascimento, telefone, e endereço completo.
   * Possibilitar retorno ao menu principal a partir desse submenu.
4. **Alteração de Dados**:
   * Submenu com três opções para editar:
     + **Conta**: Alterar limite disponível e data de vencimento.
     + **Funcionário**: Alterar código, cargo, telefone e endereço completo.
     + **Cliente**: Alterar telefone e endereço completo.
   * Solicitar senha de administrador para alterações.
5. **Cadastro de Funcionários**:
   * Solicitar senha de administrador para acesso.
   * Inserir dados completos: código do funcionário, cargo, nome, CPF, data de nascimento, telefone e endereço completo.
6. **Geração de Relatórios**:
   * Gerar um relatório geral com movimentações financeiras e exportá-lo para Excel.
   * Solicitar senha do funcionário antes de exibir o relatório.
7. **Sair**:
   * Retornar ao menu principal do sistema.

**Menu Cliente**

1. **Operações de Conta**:
   * Submenu com funcionalidades:
     + **Saldo**: Mostrar saldo ao cliente (solicitar senha antes de exibir).
     + **Depósito**: Permitir depósito de valores na conta.
     + **Saque**: Permitir saque (solicitar senha e verificar saldo).
     + **Extrato**: Exibir extrato com movimentações e permitir exportação para Excel.
     + **Consultar Limite**: Exibir limite disponível na conta (solicitar senha).
   * Opção para retornar ao menu principal.
2. **Encerrar Programa**:
   * Finalizar o sistema e encerrar a execução da aplicação.
3. **Classes do Sistema**
4. **Classe BancoMalvader**
   * **Atributos**:
     + nome: String
   * **Métodos**:
     + main(String[] args): void
     + iniciarSistema(): void
5. **Classe Usuario (Classe Abstrata)**
   * **Atributos**:
     + id: int
     + nome: String
     + cpf: String
     + dataNascimento: LocalDate
     + telefone: String
     + endereco: Endereco
   * **Métodos**:
     + login(String senha): boolean
     + logout(): void
     + consultarDados(): String
6. **Classe Funcionario (Herda de Usuario)**
   * **Atributos**:
     + codigoFuncionario: String
     + cargo: String
     + senha: String
   * **Métodos**:
     + abrirConta(Conta conta): void
     + encerrarConta(Conta conta): void
     + consultarDadosConta(int numeroConta): Conta
     + consultarDadosCliente(int idCliente): Cliente
     + alterarDadosConta(Conta conta): void
     + alterarDadosCliente(Cliente cliente): void
     + cadastrarFuncionario(Funcionario funcionario): void
     + gerarRelatorioMovimentacao(): void
7. **Classe Cliente (Herda de Usuario)**
   * **Atributos**:
     + senha: String
   * **Métodos**:
     + consultarSaldo(): double
     + depositar(double valor): void
     + sacar(double valor): boolean
     + consultarExtrato(): String
     + consultarLimite(): double
8. **Classe Endereco**
   * **Atributos**:
     + cep: String
     + local: String
     + numeroCasa: int
     + bairro: String
     + cidade: String
     + estado: String
   * **Métodos**:
     + toString(): String
9. **Classe Conta (Classe Abstrata)**
   * **Atributos**:
     + numero: int
     + agencia: String
     + saldo: double
     + cliente: Cliente
   * **Métodos**:
     + depositar(double valor): void
     + sacar(double valor): boolean
     + consultarSaldo(): double
10. **Classe ContaPoupanca (Herda de Conta)**
    * **Atributos**:
      + taxaRendimento: double
    * **Métodos**:
      + calcularRendimento(): double
11. **Classe ContaCorrente (Herda de Conta)**
    * **Atributos**:
      + limite: double
      + dataVencimento: LocalDate
    * **Métodos**:
      + consultarLimite(): double
12. **Classe Relatorio**
    * **Atributos**:
      + tipo: String
      + dataGeracao: LocalDateTime
      + dados: List<String>
    * **Métodos**:
      + gerarRelatorioGeral(): void
      + exportarParaExcel(): void
13. **Classe ConexaoBanco**
    * **Atributos**:
      + url: String
      + usuario: String
      + senha: String
    * **Métodos**:
      + conectar(): Connection
      + desconectar(): void

**Relacionamentos entre as Classes**

* Funcionario e Cliente herdam de Usuario.
* ContaPoupanca e ContaCorrente herdam de Conta.
* Cliente possui um relacionamento com Conta.
* Endereco é um atributo de composição na classe Usuario.
* Relatorio é associado ao Funcionario para a geração de relatórios.
* ConexaoBanco é utilizada para gerenciar a conexão com o banco de dados no sistema.

1. **Estrutura do Projeto Java**
2. **Pacotes do Projeto**
   * **dao:** Contém as classes de Data Access Object para interagir com o banco de dados.
   * **model:** Armazena as classes de modelo que representam as tabelas do banco de dados.
   * **view:** Abriga as telas Swing (interface gráfica) da aplicação.
   * **controller:** Contém a lógica para conectar a interface com o modelo.
   * **util:** Inclui classes utilitárias, como a conexão com o banco de dados.
3. **Classes Principais**
   * **dao**
     + ConnectionFactory: Configuração e criação da conexão com o MySQL.
     + FuncionarioDAO, ClienteDAO, etc.: Classes para acessar dados do banco e executar consultas e operações de inserção/atualização/exclusão.
   * **model**
     + Funcionario: Modelo para dados dos funcionários (código, cargo, nome, CPF, etc.).
     + Cliente: Modelo para dados dos clientes.
     + Conta: Modelo para conta bancária, com tipos “Poupança” e “Corrente”.
   * **view**
     + LoginView: Tela inicial de login com opções de "Funcionário" ou "Cliente".
     + MenuFuncionarioView, MenuClienteView: Telas de menus específicos, baseados nas funções do banco.
     + CadastroFuncionarioView, CadastroContaView, etc.: Interfaces para cadastro, consulta, e alteração de dados.
   * **controller**
     + FuncionarioController, ClienteController, ContaController: Controladores que conectam a interface gráfica com a lógica de negócios e os DAOs.
   * **util**
     + DBUtil: Classe com métodos utilitários para conectar, desconectar e manipular a base de dados MySQL.

**3. Desenvolvimento da Interface Gráfica com Swing**

**Estrutura das Telas e Funcionalidades**

A aplicação deve conter as seguintes telas, de forma modular e organizada conforme o menu do sistema:

**Tela de Login**

* **Componentes**:
  + JTextField para o campo de usuário.
  + JPasswordField para a senha.
  + Dois JButton: um para acesso como Funcionário e outro para Cliente.
* **Funcionalidade**: Autentica o usuário e redireciona para o Menu Principal.

**Menu Principal**

* **Componentes**:
  + Três JButton para acessar as opções: Funcionário, Cliente e Sair.
* **Funcionalidade**: O botão "Funcionário" abre o Menu do Funcionário, e "Cliente" abre o Menu do Cliente.

**Menu Funcionário**

* **Componentes**:
  + Sete JButton para cada operação: Abertura de Conta, Encerramento de Conta, Consulta de Dados, Alteração de Dados, Cadastro de Funcionário, Geração de Relatórios e Sair.
* **Funcionalidade**: Cada botão executa uma função específica usando métodos da classe BancoController.

**Menu Cliente**

* **Componentes**:
  + Seis JButton para operações: Saldo, Depósito, Saque, Extrato, Consultar Limite e Sair.
* **Funcionalidade**: Cada botão chama uma função específica do sistema.

**Exemplo de Tela com Swing (Tela de Login):**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Estrutura de Classes e Funcionalidades (Modelo e Controle)**

**Classe BancoController (Controle de Operações)**

* **Atributos**:
  + Uma List<Conta> para armazenar as contas.
  + Uma List<Funcionario> para armazenar os funcionários.
* **Métodos**:
  + public void abrirConta(Conta conta): Registra uma nova conta.
  + public void encerrarConta(int numeroConta): Encerra uma conta.
  + public Conta consultarConta(int numeroConta): Consulta informações de uma conta específica.
  + public void salvarDados(): Salva todos os dados em arquivos.
  + public void carregarDados(): Carrega dados dos arquivos no início da aplicação.

**Classes de Modelo**

* **Conta** (Classe Abstrata):
  + Atributos: numero, saldo, tipoConta.
  + Métodos: depositar(), sacar(), consultarSaldo().
* **ContaCorrente** e **ContaPoupanca** (Subclasses de Conta):
  + Atributos específicos, como limite para ContaCorrente.
  + Métodos específicos, como verificarLimite() para ContaCorrente.
* **Cliente** e **Funcionario**:
  + Atributos como nome, cpf, telefone, endereço.
  + Métodos de getters e setters.

**5. Persistência de Dados com Arquivos**

Os dados do banco serão salvos em arquivos para garantir que as informações persistam entre as execuções do programa.

**Criando Métodos de Salvamento e Carregamento de Dados**

Crie uma classe DataManager em utils para lidar com operações de leitura e gravação de dados em arquivos.

**Exemplo de Salvamento de Dados**:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Uso no BancoController**:

* + No método salvarDados(), chame DataManager.salvarContas(contas, "contas.dat").
  + No método carregarDados(), chame DataManager.carregarContas("contas.dat").

Esses métodos devem ser chamados ao iniciar e encerrar a aplicação, para garantir que os dados estejam sempre atualizados.

**6. Exemplo de Implementação do Menu de Abertura de Conta**

No Menu Funcionário, implemente a funcionalidade de abertura de conta:

1. **Interface**: Ao selecionar "Abertura de Conta", exiba um formulário para o funcionário preencher os dados do cliente.
2. **Classe AberturaDeContaScreen**:
   * Inclua campos como JTextField para número da conta, nome, CPF, etc.
   * JComboBox para selecionar o tipo de conta (Corrente ou Poupança).
   * JButton para confirmar a criação da conta.
3. **Exemplo de Ação de Botão para Abertura de Conta**:

Texto

Descrição gerada automaticamente

1. **Tutorial Rápido de Interface com Swing**

Para criar uma interface gráfica em Java usando **Swing**, vamos explorar alguns conceitos básicos e componentes essenciais. Swing é uma biblioteca gráfica nativa do Java que facilita a criação de janelas, botões, campos de texto e outros elementos visuais.

**1. Criando Janelas com JFrame**

Para iniciar uma interface, criamos uma janela principal usando JFrame.

Texto

Descrição gerada automaticamente

**Organizando Componentes com JPanel e Layouts**

O JPanel é usado para organizar componentes como botões e campos de texto dentro do JFrame. Usamos diferentes layouts (como FlowLayout, GridLayout, BorderLayout) para posicionar os componentes na tela.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Adicionando Componentes Básicos**

Swing possui diversos componentes, como:

* **JLabel**: Exibe texto.
* **JTextField**: Campo para entrada de texto.
* **JPasswordField**: Campo para entrada de senha.
* **JButton**: Botão para executar ações.

**Exemplo**:

Tela de computador

Descrição gerada automaticamente

**Adicionando Ações aos Botões com ActionListener**

Para responder a cliques em botões, usamos o ActionListener, que permite associar uma ação ao clique do usuário.

Tela de computador

Descrição gerada automaticamente

**Usando Caixas de Diálogo com JOptionPane**

O JOptionPane permite criar caixas de diálogo simples para exibir mensagens ou capturar entradas.

* **Mostrar uma mensagem**:

Texto

Descrição gerada automaticamente

**Mostrar uma pergunta e capturar a resposta**:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Exemplo Completo: Tela de Login com Swing**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Próximos Passos para o Projeto**

Para o projeto do Banco Malvader, vocês usarão esse conhecimento para criar menus e telas adicionais, conectando-as à lógica da aplicação. Experimente criar:

* **Menus com Botões** para as funcionalidades.
* **Formulários** de entrada para capturar dados dos usuários.
* **Caixas de Diálogo** para exibir mensagens e confirmações.

Pessoal, essas ferramentas vão permitir que vocês criem uma interface gráfica interativa para o sistema de banco, integrando funcionalidades conforme solicitado no projeto.

Vamos lá, vocês conseguem.

**F. Passo a Passo para Configuração**

**1. Configuração da Conexão Java com MySQL**

1. **Baixe o Driver JDBC MySQL**:
   1. Acesse o site oficial do MySQL e baixe o driver JDBC (arquivo .jar).
2. **Adicione o Driver ao Projeto**:
   1. No ambiente de desenvolvimento (Eclipse), vá até as configurações do projeto e adicione o .jar do driver JDBC ao classpath do projeto.
3. **Configuração da Classe de Conexão (ConnectionFactory)**:
   1. Crie a classe ConnectionFactory no pacote dao.
   2. No método de conexão, utilize o seguinte código para estabelecer a conexão:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

* 1. **Implementação da Aplicação**

1. **Desenvolvimento dos Modelos (Pacote model)**:
   * Defina classes de modelo (e.g., Funcionario, Cliente, Conta) que representem as tabelas no banco de dados.
   * Cada classe deve conter atributos correspondentes às colunas e métodos getters e setters.
2. **Implementação dos DAOs (Pacote dao)**:
   * Em cada DAO, crie métodos como save(), update(), delete(), e findById() para manipular as informações.
   * Exemplo de método de inserção em FuncionarioDAO:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

1. **Criação da Interface Gráfica (Pacote view)**:

* Implemente telas Swing para cada funcionalidade (Cadastro, Consulta, Relatórios, etc.).
* Utilize o JFrame como janela principal e adicione componentes como JButton, JTextField, e JTable conforme necessário.

1. **Desenvolvimento dos Controladores (Pacote controller)**:

* Em cada controlador, implemente a lógica de negócios que será usada para chamar métodos dos DAOs a partir da interface gráfica.
* Exemplo de método no FuncionarioController:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

1. **Interação com o Banco de Dados (Pacote util)**:

* A classe DBUtil conterá métodos para gerenciar as conexões e transações com o banco de dados.

1. **Tabelas do Banco de Dados**
2. **Tabela usuario**
   * Armazena dados comuns para usuários do sistema, tanto funcionários quanto clientes.
   * **Colunas**:
     + id\_usuario (PK, INT, AUTO\_INCREMENT): Identificador único do usuário.
     + nome (VARCHAR, 100): Nome completo do usuário.
     + cpf (VARCHAR, 11): CPF do usuário.
     + data\_nascimento (DATE): Data de nascimento.
     + telefone (VARCHAR, 15): Telefone de contato.
     + tipo\_usuario (ENUM: 'FUNCIONARIO', 'CLIENTE'): Define se o usuário é funcionário ou cliente.
     + senha (VARCHAR, 50): Senha para acesso ao sistema.
3. **Tabela funcionario**
   * Armazena dados específicos dos funcionários.
   * **Colunas**:
     + id\_funcionario (PK, INT, AUTO\_INCREMENT): Identificador único do funcionário.
     + codigo\_funcionario (VARCHAR, 20): Código interno do funcionário.
     + cargo (VARCHAR, 50): Cargo do funcionário.
     + id\_usuario (FK, INT): Relacionamento com a tabela usuario.
4. **Tabela cliente**
   * Armazena dados específicos dos clientes.
   * **Colunas**:
     + id\_cliente (PK, INT, AUTO\_INCREMENT): Identificador único do cliente.
     + id\_usuario (FK, INT): Relacionamento com a tabela usuario.
5. **Tabela endereco**
   * Armazena os endereços de usuários, vinculando clientes e funcionários aos respectivos endereços.
   * **Colunas**:
     + id\_endereco (PK, INT, AUTO\_INCREMENT): Identificador único do endereço.
     + cep (VARCHAR, 10): Código postal.
     + local (VARCHAR, 100): Logradouro.
     + numero\_casa (INT): Número da residência.
     + bairro (VARCHAR, 50): Bairro.
     + cidade (VARCHAR, 50): Cidade.
     + estado (VARCHAR, 2): Estado (sigla).
     + id\_usuario (FK, INT): Relacionamento com a tabela usuario.
6. **Tabela conta**
   * Tabela base para contas bancárias, tanto para poupança quanto para conta corrente.
   * **Colunas**:
     + id\_conta (PK, INT, AUTO\_INCREMENT): Identificador único da conta.
     + numero\_conta (VARCHAR, 20): Número da conta.
     + agencia (VARCHAR, 10): Agência onde a conta foi criada.
     + saldo (DECIMAL, 15,2): Saldo atual da conta.
     + tipo\_conta (ENUM: 'POUPANCA', 'CORRENTE'): Tipo da conta bancária.
     + id\_cliente (FK, INT): Relacionamento com a tabela cliente.
7. **Tabela conta\_corrente**
   * Armazena dados específicos de contas correntes.
   * **Colunas**:
     + id\_conta\_corrente (PK, INT, AUTO\_INCREMENT): Identificador da conta corrente.
     + limite (DECIMAL, 15,2): Limite de crédito da conta corrente.
     + data\_vencimento (DATE): Data de vencimento do limite.
     + id\_conta (FK, INT): Relacionamento com a tabela conta.
8. **Tabela conta\_poupanca**
   * Armazena dados específicos de contas poupança.
   * **Colunas**:
     + id\_conta\_poupanca (PK, INT, AUTO\_INCREMENT): Identificador da conta poupança.
     + taxa\_rendimento (DECIMAL, 5,2): Taxa de rendimento da poupança.
     + id\_conta (FK, INT): Relacionamento com a tabela conta.
9. **Tabela transacao**
   * Armazena todas as transações realizadas nas contas, como saques, depósitos e extratos.
   * **Colunas**:
     + id\_transacao (PK, INT, AUTO\_INCREMENT): Identificador único da transação.
     + tipo\_transacao (ENUM: 'DEPOSITO', 'SAQUE', 'TRANSFERENCIA'): Tipo de transação realizada.
     + valor (DECIMAL, 15,2): Valor da transação.
     + data\_hora (TIMESTAMP): Data e hora da transação.
     + id\_conta (FK, INT): Relacionamento com a conta em que a transação foi realizada.
10. **Tabela relatorio**
    * Armazena relatórios gerados pelos funcionários.
    * **Colunas**:
      + id\_relatorio (PK, INT, AUTO\_INCREMENT): Identificador único do relatório.
      + tipo\_relatorio (VARCHAR, 50): Tipo de relatório gerado.
      + data\_geracao (TIMESTAMP): Data e hora da geração do relatório.
      + conteudo (TEXT): Conteúdo do relatório.
      + id\_funcionario (FK, INT): Relacionamento com a tabela funcionario.

**Relacionamentos**

* **1**

entre usuario e funcionario / cliente (um usuário pode ser um cliente ou um funcionário).

* **1**

entre usuario e endereco (um usuário pode ter um endereço registrado).

* **1**

entre cliente e conta (um cliente pode ter várias contas).

* **1:1** entre conta e conta\_corrente / conta\_poupanca (uma conta pode ser corrente ou poupança).
* **1**

entre conta e transacao (uma conta pode ter várias transações).

* **1**

entre funcionario e relatorio (um funcionário pode gerar vários relatórios).

**Entrega e Avaliação**

1. **Entrega do Código**: Organize o código em pacotes e classes claras, com comentários explicativos.
2. **Documentação**: Inclua uma breve documentação descrevendo as classes, o fluxo do sistema e as principais funcionalidades.
3. **Apresentação**: Apresente o funcionamento do sistema, explicando a interface e o processo de persistência dos dados.

**Critérios de Avaliação**

* **Funcionalidade**: Implementação correta das operações.
* **Interface Gráfica**: Interface amigável