**Package: principal**

**Class: Main**

Método Main responsável por iniciar o sistema.

**Package: telas**

**Class: TelaInicial**

Nessa classe iniciamos com o método chamarTelaMenuInicial que contém todos os atributos necessários para a composição da tela, como JFrame, JPanel, JLabel, JTextField e JButton, além de um GridLayout, BufferedImage, ImageIcon e BorderLayout para atribuir um melhor design à interface.   
Após essas configurações o frame foi adicionado ao painel e a classe controllerTelaInicial foi chamada.

Essa tela é responsável por receber a opção desejada pelo usuário para realizar um cadastro de correntista.

**Package: controller**

**Class: ControllerTelaInicial**

No controlador da tela foram adicionados os atributos JFrame, JTextField e JButton para a criação do construtor. Após isso foi iniciado um método Override actionPerformed para dar funcionalidade de acordo com a opção inputada pelo usuário, usando if/else e switch/case.

Após escolher, o usuário será direcionado para um menu secundário ou poderá escolher encerrar a execução do programa.

**Package: telas**

**Class: TelaSecundaria**

Nessa classe iniciamos com o método chamarTelaMenuSecundario que contém todos os atributos necessários para a composição da tela, como JFrame, JPanel, JLabel, JTextField e JButton, além de um GridLayout, BufferedImage para atribuir um melhor design à interface.   
Após essas configurações o frame foi adicionado ao painel e a classe controllerTelaSecundaria foi chamada.

Essa tela é responsável por dar ao usuário as opções de Cadastrar, Listar, Editar e Deletar correntista, além de também oferecer a opção de encerrar a execução do programa e um botão para voltar à tela anterior.

**Package: controller**

**Class: ControllerTelaSecundaria**

No controlador da tela foram adicionados os atributos JFrame, JTextField e JButton para a criação do construtor. Após isso foi iniciado um método Override actionPerformed para dar funcionalidade de acordo com a opção inputada pelo usuário, usando if/else, switch/case e try/catch.

A opção 1 direciona o usuário para a tela de cadastro, 2 direciona para a lista de correntistas cadastrados, 3 direciona para uma tela onde o usuário deverá informar o CPF do correntista que deseja editar e após isso direciona para a tela de edição, 4 direciona para uma tela que perguntará o CPF do correntista a ser deletado seguido de uma caixa de confirmação e o número 5 encerra a execução do programa.

**Package: telas**

**Class: TelaCadastroCorrentista**

Nessa classe iniciamos com o método chamarTelaCadastroCorrentista que contém todos os atributos necessários para a composição da tela, como JFrame, JPanel, JLabel, JTextField e JButton, além de um GridLayout e BufferedImage para atribuir um melhor design à interface.   
Após essas configurações o frame foi adicionado ao painel e a classe controllerTelaCadastroCorrentista foi chamada.

Foi adicionado um método Override actionPerformed para que ao apertar o botão ‘Enter’ após digitar o CEP todos os outros campos relacionados ao endereço sejam automaticamente preenchidos.

Essa tela é responsável por receber todos os dados fornecidos pelo usuário no momento do cadastro e possui um botão para voltar à tela anterior.

**Package: controller**

**Class: ControllerTelaCadastroCorrentista**

No controlador da tela foram adicionados os atributos JFrame, JTextField e JButton para a criação do construtor. Após isso foi iniciado um método Override actionPerformed para salvar no banco de dados as informações inputadas pelo usuário ao clicar no botão ‘Enviar’, usando if/else.

Se o cadastro for devidamente realizado abrirá um JOptionPane informando ao usuário que o cadastro foi realizado com sucesso e ao clicar em ‘Ok’ todos os campos serão limpos utilizando o método limparCampos para que o usuário possa cadastrar outro correntista se assim desejar.

**Package: telas**

**Class: TelaEditarCorrentista**

Nessa classe iniciamos com o método ControllerTelaEditarCorrentista que contém todos os atributos necessários para a composição da tela, como JFrame, JPanel, JLabel, JTextField e JButton, além de um GridLayout e BufferedImage para atribuir um melhor design à interface.   
Após essas configurações o frame foi adicionado ao painel e a classe controllerTelaEditarCorrentista foi chamada.

Foi adicionado um método Override actionPerformed para que ao apertar o botão ‘Enter’ após digitar o CEP todos os outros campos relacionados ao endereço sejam automaticamente preenchidos.

Essa tela é responsável por receber todos os novos dados fornecidos pelo usuário e possui um botão para voltar à tela anterior.

**Package: controller**

**Class: ControllerTelaEditarCorrentista**

No controlador da tela foram adicionados os atributos JFrame, JTextField e JButton para a criação do construtor. Após isso foi iniciado um método Override actionPerformed para salvar no banco de dados as novas informações inputadas pelo usuário ao clicar no botão ‘Enviar’, usando if/else.

Se o cadastro for devidamente realizado abrirá um JOptionPane informando ao usuário que o cadastro foi atualizado com sucesso e ao clicar em ‘Ok’ a tela de edição se fecha.

**Package: entidade**

**Class: Correntista**

Correntista é uma classe abstrata que possui todos os atributos necessários para cadastro, tais como Nome, CPF, CEP, Logradouro, Bairro, Cidade, UF, Email, Quantidade de transações, Valor de anuidade e também possui os métodos para calcular limite de saque e limite de crédito, dependendo do tipo de correntista escolhido.

A classe contém métodos getter e setter para cada um dos atributos. Esses métodos são usados para acessar e modificar seus valores.

**Package: entidade**

**Class: CorrentistaBasico | CorrentistaPremium**

As classes CorrentistaBasico e CorrentistaPremium herdam de Correntista, reutilizando todos os atributos e métodos da classe base e são responsáveis por realizar o cálculo de Limite de Saque (correntista básico) e Limite de Crédito (correntista premium).

**Package: entidade**

**Class: Cep**

A classe Cep representa um endereço no formato do código postal brasileiro (CEP). Ela contém diversos atributos relacionados ao endereço, além de métodos getter e setter para acessar e modificar esses atributos.

A classe Cep organiza os dados de um endereço de forma clara e estruturada, facilitando a manipulação e o acesso a esses dados em outras partes do programa.

**Package: service**

**Class: BuscarCep**

A classe BuscarCep permite buscar informações de endereço a partir de um CEP, consultando um serviço web (viacep.com.br), e converte a resposta JSON em um objeto Cep. Essa funcionalidade é útil para trazer informações detalhadas de endereço a partir de um CEP fornecido pelo usuário.

**Package: persistencia**

**Class: FabricaConexao**

A classe FabricaConexao é responsável por estabelecer uma conexão com um banco de dados MySQL. Essa classe encapsula a lógica de conexão, incluindo o tratamento de exceções e a gestão das credenciais e URL do banco de dados.

**Package: persistencia**

**Class: DaoCorrentista**

A classe DaoCorrentista é responsável por realizar operações de CRUD (Create, Read, Update, Delete) no banco de dados para a entidade Correntista. Ela utiliza a FabricaConexao para gerenciar conexões e prepara comandos SQL para buscar, atualizar e deletar correntistas. A classe também trata exceções e garante que os recursos de conexão e comando SQL sejam fechados adequadamente.

**Package: persistencia**

**Class: DaoCorrentistaBasico | DaoCorrentistaPremium**

As classes DaoCorrentistaBasico e DaoCorrentistaPremium realiza operações de salvamento e recuperação de correntistas no banco de dados. Elas usam a FabricaConexao para gerenciar as conexões e executa comandos SQL para inserir e buscar correntistas. A classe trata exceções e garante que os recursos de conexão e comandos SQL sejam fechados adequadamente após a execução.

**Package: repositorio**

**Class: CorrentistaBasicoReopositorio**

A interface CorrentistaBasicoRepositorio define um contrato para classes que implementarão operações de persistência de correntistas básicos. Ela declara dois métodos: um para salvar um correntista básico e outro para listar todos os correntistas básicos do repositório. Classes que implementarem esta interface deverão fornecer a lógica específica para essas operações, garantindo que o contrato estabelecido pela interface seja respeitado.

**Package: repositorio**

**Class: CorrentistaBasicoRepositorioImp**

A classe CorrentistaBasicoRepositorioImp é uma implementação concreta da interface CorrentistaBasicoRepositorio. Essa classe deve fornecer a lógica específica para as operações de persistência definidas pela interface.

**Package: repositorio**

**Class: CorrentistaPremiumRepositorio**

A interface CorrentistaPremiumRepositorio define um contrato para operações de persistência e recuperação de dados relacionadas a correntistas premium. Qualquer classe que implementar esta interface deve fornecer a lógica específica para salvar um CorrentistaPremium no repositório e listar todos os CorrentistaPremium presentes no repositório. Isso permite que diferentes implementações de repositórios sigam um padrão consistente, facilitando a troca de implementações sem alterar o código que depende dessas operações.

**Package: repositorio**

**Class: CorrentistaPremiumRepositorioImp**

A classe CorrentistaPremiumRepositorioImp implementa a interface CorrentistaPremiumRepositorio, assumindo a responsabilidade de implementar os métodos definidos nessa interface para lidar com operações relacionadas aos correntistas premium.

**Package: arquivo**

**Class: ManipuladorArquivo**

Esta classe encapsula a lógica para criar um arquivo PDF detalhando informações de correntistas, oferecendo uma solução robusta para geração e visualização de relatórios no formato PDF.

**Package: arquivo**

**Class: RodapeTemplate**

A classe RodapeTemplate é uma classe auxiliar que estende PdfPageEventHelper do iText, uma biblioteca Java para manipulação de arquivos PDF. Ela é utilizada para adicionar um rodapé padronizado em todas as páginas do documento PDF gerado.

Esta classe é essencial para garantir que todas as páginas do PDF gerado tenham um rodapé consistente e informativo, adicionando detalhes como a localização (exemplo: "São Paulo") e a data de geração do documento.