

Telas e Implementação das Classes

Bruno Eduardo [REDACTED]
[REDACTED]
[REDACTED]

¹Departamento de Ciência da Computação – Nome da Universidade
Técnicas de programação 1
16 de fevereiro de 2025

[REDACTED]@aluno.unb.br, [REDACTED]@aluno.unb.br, [REDACTED]@aluno.unb.br

Sumário

1	Descrição do problema	2
1.1	Objetivos	2
2	Definição das regras de negócio	3
2.1	Cadastro de Transações	3
2.2	Definição de Limite Mensal	3
2.3	Categorias Personalizadas	3
2.4	Geração de Relatórios	3
2.5	Funcionalidades novas	3
2.6	Funcionalidades descontinuadas	3
3	Distribuição das Tarefas	4
4	Diagrama de Classes	6
5	Telas	8
5.1	Cadastro de Usuário	8
5.2	Login de Usuário	9
5.3	Tela Principal e mini telas(Bruno)	10
5.3.1	Menu de Navegação do Sistema	11
5.4	Tela: Contas ([REDACTED])	12
5.5	Tela: Categorias ([REDACTED])	13
5.6	Tela: Histórico ([REDACTED])	15
5.7	Tela: Gráfico ([REDACTED])	16
6	Classes	17

1. Descrição do problema

O aumento do endividamento das famílias brasileiras, que chegou a 78,8% em maio de 2024, como revelou um estudo da Confederação Nacional do Comércio, destaca uma dificuldade comum enfrentada por muitas famílias: o controle financeiro. Esse cenário tem sido agravado pela falta de planejamento adequado, levando muitas pessoas a gastarem mais do que podem, o que resulta no acúmulo de dívidas e, consequentemente, na inadimplência. Para entender melhor as necessidades e os desafios enfrentados pelas pessoas, realizamos uma pesquisa com 37 participantes, abordando quatro temas: Gerenciamento de Finanças, Gerenciamento de Vendas, Gerenciamento de Leitura e Gerenciamento de Viagens. O gráfico abaixo mostra a distribuição das preferências dos participantes em relação aos temas pesquisados:

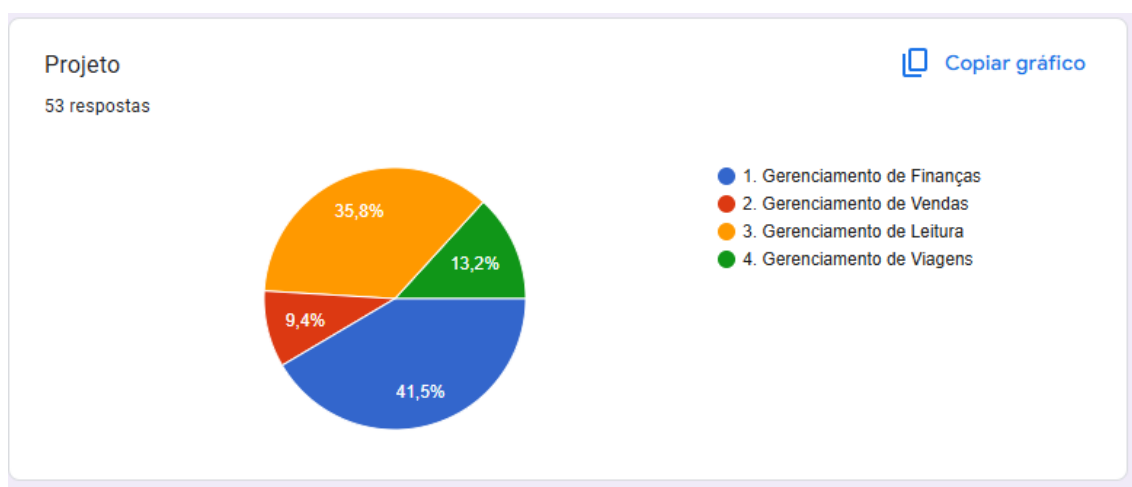


Figura 1. Pesquisa de projetos

Como é possível ver, 41,5% dos participantes escolheram Gerenciamento de Finanças Pessoais e Familiares como a maior prioridade, o que enfatiza a necessidade da relevância do assunto. Com base nesse dado, nosso projeto tem como objetivo criar um sistema simples e acessível que ajude as pessoas a monitorar seus gastos, planejar suas economias e tomar decisões financeiras mais inteligentes. A ideia é oferecer uma solução prática, que se encaixe na rotina de cada usuário, trazendo mais clareza e controle sobre a gestão do seu dinheiro.

1.1. Objetivos

O principal objetivo deste projeto é desenvolver um aplicativo desktop que auxilie os usuários na gestão de suas finanças pessoais. A ferramenta permite organizar receitas e despesas de forma prática, possibilitando um controle mais eficiente do orçamento.

Para aprimorar a experiência do usuário, o aplicativo gera relatórios, facilitando a análise do uso do orçamento e proporcionando maior clareza sobre os hábitos financeiros. A interface foi projetada para ser funcional, garantindo que mesmo usuários sem conhecimento avançado em finanças possam utilizar a ferramenta de maneira eficaz.

O desenvolvimento do projeto foi realizado de forma colaborativa e pode ser acessado no repositório oficial no GitHub: <https://github.com/brnduol/projeto-gerenciador-financeiro>.

2. Definição das regras de negócio

As regras de negócio deste projeto foram desenvolvidas com o objetivo de criar um sistema eficiente e intuitivo para o gerenciamento financeiro pessoal. Focado em simplicidade e acessibilidade, o sistema permitirá aos usuários organizarem suas finanças de forma prática, auxiliando no controle de receitas, despesas e cartões de crédito. De acordo com a última entrega, fizemos algumas alterações que vimos que faria mais sentido para o sistema.

2.1. Cadastro de Transações

O sistema permitirá o registro de receitas e despesas, e foi removida a de transação. Esses registros terão que ter as informações como data, valor e categoria ou origem da renda, no caso de receita. Não será possível salvar uma transação com informações incompletas.

2.2. Definição de Limite Mensal

A funcionalidade de limite mensal foi removida por não vemos mais a necessidade dessa funcionalidade, distoando do objetivo principal.

2.3. Categorias Personalizadas

As categorias também terão como ser adicionadas. Além das categorias, terá a classe `OrigemRenda`, onde servirá para categorizar as entradas de dinheiro.

2.4. Geração de Relatórios

A aplicação terá funcionalidade de criação de gráficos visuais, como o gráfico de pizza, para o usuário ter noção dos seus gastos, por categoria, do mês escolhido.

2.5. Funcionalidades novas

- **Histórico de Transação:** A aplicação contará com histórico de transação, onde poderá ser visualizado todas as transações feitas. Esse histórico mostra todas, independente da data.
- **Origem de Renda:** Adicionamos a classe `OrigemRenda` para detalhar e categorizar as fontes da receita de forma mais clara.
- **Transação de Despesa e Transação de Receita:** Incluímos as classes `TransaçãoDespesa` e `TransaçãoReceita` para organizar melhor o controle do fluxo de entrada e saída.

2.6. Funcionalidades descontinuadas

- **Cartão de crédito:** Optamos por descontinuar a funcionalidade de cartão de crédito. Passamos com algumas dificuldades na implementação da lógica, o que fez com que achamos melhor descontinuar para focar no objetivo principal, de receitas e despesas.
- **Tipo de Transação:** Retiramos o tipo de transação porque já não fazia mais tanto sentido, já que acrescentamos outras funcionalidades para o tipo transação.
- **Transação Transferência:** Removemos a classe transação de transferência devido às dificuldades em implementá-la de forma eficiente, e para evitar complicações desnecessárias.
- **Limite Mensal:** Resolvemos remover pelo mesmo motivo da transação transferência, já que ele não era mais necessário para o sistema.

3. Distribuição das Tarefas

Para garantir um desenvolvimento organizado e eficiente do sistema, as atividades foram distribuídas entre os membros do grupo, considerando tanto a implementação das classes quanto o desenvolvimento das telas. Apesar da divisão, o trabalho foi conduzido de maneira colaborativa, com cada integrante contribuindo para o suporte mútuo e a resolução de desafios ao longo do projeto.

Implementação das Classes

O desenvolvimento das classes foi distribuído de forma a equilibrar a carga de trabalho e aproveitar as habilidades individuais de cada membro. A Tabela 1 apresenta a divisão de responsabilidades:

Responsável	Classes Implementadas
██████	Atualizável (Interface) Transação TransaçãoDespesa (Herda Transação) TransaçãoReceita (Herda Transação)
Bruno	Carteira Usuário OrigemRenda ContasUsuários ControleTelas
██████	Categoria Balanço (Classe utilitária)

Tabela 1. Distribuição das classes por responsável.

Desenvolvimento das Telas

Além da implementação das classes, cada integrante também ficou responsável pelo desenvolvimento de interfaces específicas do sistema. A Tabela 2 detalha essa divisão:

Responsável	Telas Implementadas
Bruno	Tela de Cadastro Tela de Login Tela Principal MiniTela Detalhar MiniTela Despesa MiniTela Receita
██████	Tela de Categoria Tela de Contas
██████	Tela Gráfico Tela Histórico

Tabela 2. Distribuição das telas por responsável.

Colaboração e Revisão

Embora cada membro tenha assumido uma parte específica do projeto, a colaboração foi um elemento essencial para o sucesso do desenvolvimento. Durante a implementação, o grupo trabalhou de forma integrada, auxiliando-se mutuamente na solução de desafios técnicos.

Na parte da escrita, cada integrante foi responsável por documentar as classes e telas que implementou, garantindo uma explicação detalhada do funcionamento de cada componente. Além disso, a criação do diagrama UML ficou sob a responsabilidade de Bruno, enquanto a revisão geral do projeto foi realizada coletivamente.

4. Diagrama de Classes

O sistema de gerenciamento financeiro foi desenvolvido com foco na organização e integração das informações relacionadas a usuários, transações, categorias de despesas e receitas, carteiras e controle de telas da interface gráfica. O diagrama, ilustrado na Figura 2, representa as principais classes do sistema e seus relacionamentos.

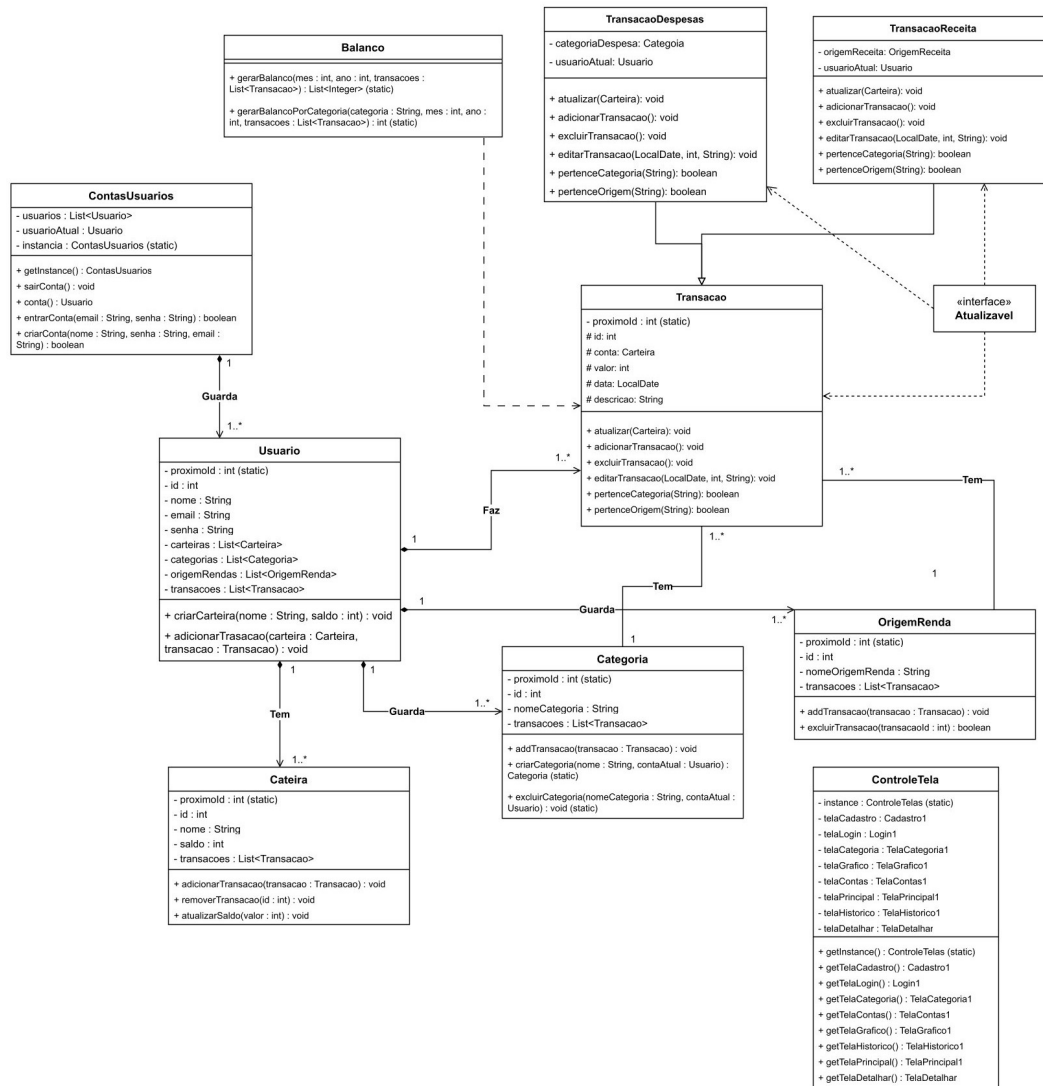


Figura 2. Diagrama de Classes

A classe central é **ContasUsuarios**, que adota o padrão *singleton* para gerenciar os usuários cadastrados no sistema. Ela também armazena o usuário atualmente autenticado, servindo como ponto de acesso para as operações de login e controle das informações de cada usuário.

Cada **Usuario** possui informações pessoais, como nome, e-mail e senha, além de manter relações diretas com entidades como **Carteira**, **Categoria**, **OrigemRenda** e

Transacao. A classe oferece métodos que permitem criar carteiras e adicionar transações, conectando o usuário às suas operações financeiras de maneira prática e eficiente.

As carteiras financeiras são representadas pela classe **Carteira**, que armazena o saldo disponível e uma lista de transações associadas. Essa classe também permite gerenciar as transações vinculadas a cada carteira, oferecendo métodos para adicionar, remover e atualizar registros, garantindo que o saldo esteja sempre correto.

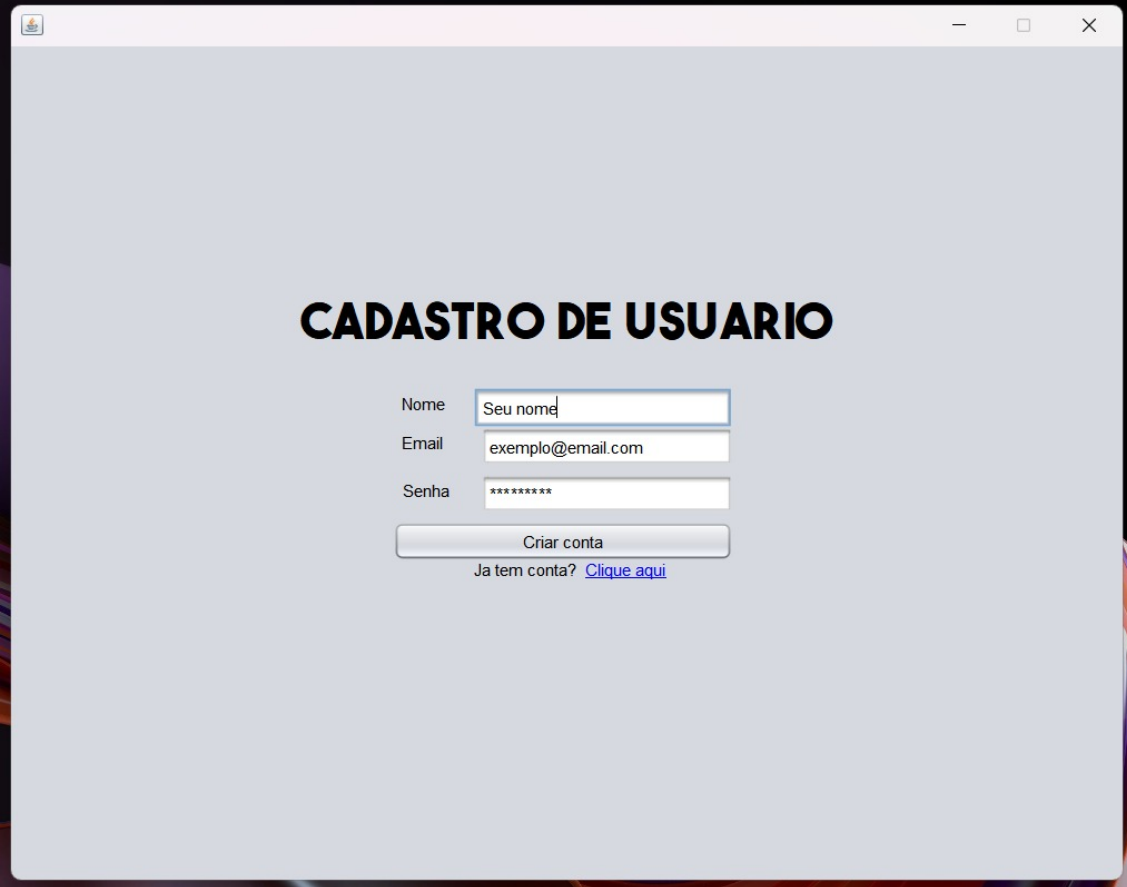
Para modelar as operações financeiras, a classe abstrata **Transacao** serve como base para as especializações **TransacaoReceita** e **TransacaoDespesas**. Essas subclasses representam, respectivamente, as receitas e despesas de cada usuário. Além de implementar métodos específicos para editar, excluir e atualizar transações, elas também utilizam a interface **Atualizavel**, que organiza as operações relacionadas às categorias e origens.

A organização das transações financeiras é feita pelas classes **Categoria** e **OrigemRenda**. A primeira representa os diferentes tipos de despesas, como alimentação e transporte, enquanto a segunda modela as fontes de receitas, como salários ou aluguéis. Ambas possuem listas de transações associadas e métodos para adicionar ou remover registros, promovendo uma categorização clara e acessível.

Por fim, a classe **ControleTelas** desempenha um papel crucial na navegação do sistema. Responsável por gerenciar as telas da interface gráfica, ela adota o padrão *singleton* para criar e manipular objetos das diferentes telas, como login, cadastro e visualizações de categorias, carteiras e transações. Essa abordagem otimiza o desempenho do sistema ao evitar a criação repetida de objetos, além de facilitar a atualização de dados exibidos em tabelas e outros componentes visuais.

5. Telas

5.1. Cadastro de Usuário



A imagem mostra uma interface web para o cadastro de usuário. No topo, há uma barra de título com ícones de minimizar, maximizar e fechar. O conteúdo principal tem um fundo cinza claro. No centro, o título "CADASTRO DE USUARIO" está em negrito. Abaixo dele, há três campos de entrada: "Nome" com o placeholder "Seu nome", "Email" com "exemplo@email.com", e "Senha" com caracteres ocultos por asteriscos. Um botão "Criar conta" está abaixo dos campos. Na base, há o texto "Ja tem conta?" seguido por um link azul "Clique aqui".

Figura 3. Tela de Cadastro de Usuário

A tela de cadastro de usuário foi projetada para garantir um processo de registro intuitivo. O usuário deve fornecer seu nome, e-mail e senha nos respectivos campos, garantindo a criação de uma conta personalizada no sistema. Para concluir o cadastro, basta pressionar o botão "Criar Conta", momento em que o sistema verificará se as informações estão corretas e se o e-mail informado já está cadastrado. Caso o e-mail já esteja em uso, um aviso será exibido, impedindo o registro de contas duplicadas.

Além do cadastro, a tela também oferece uma opção para usuários que já possuem uma conta. Caso o usuário perceba que já tem um registro, ele pode ser redirecionado para a tela de login com um simples clique.

5.2. Login de Usuário

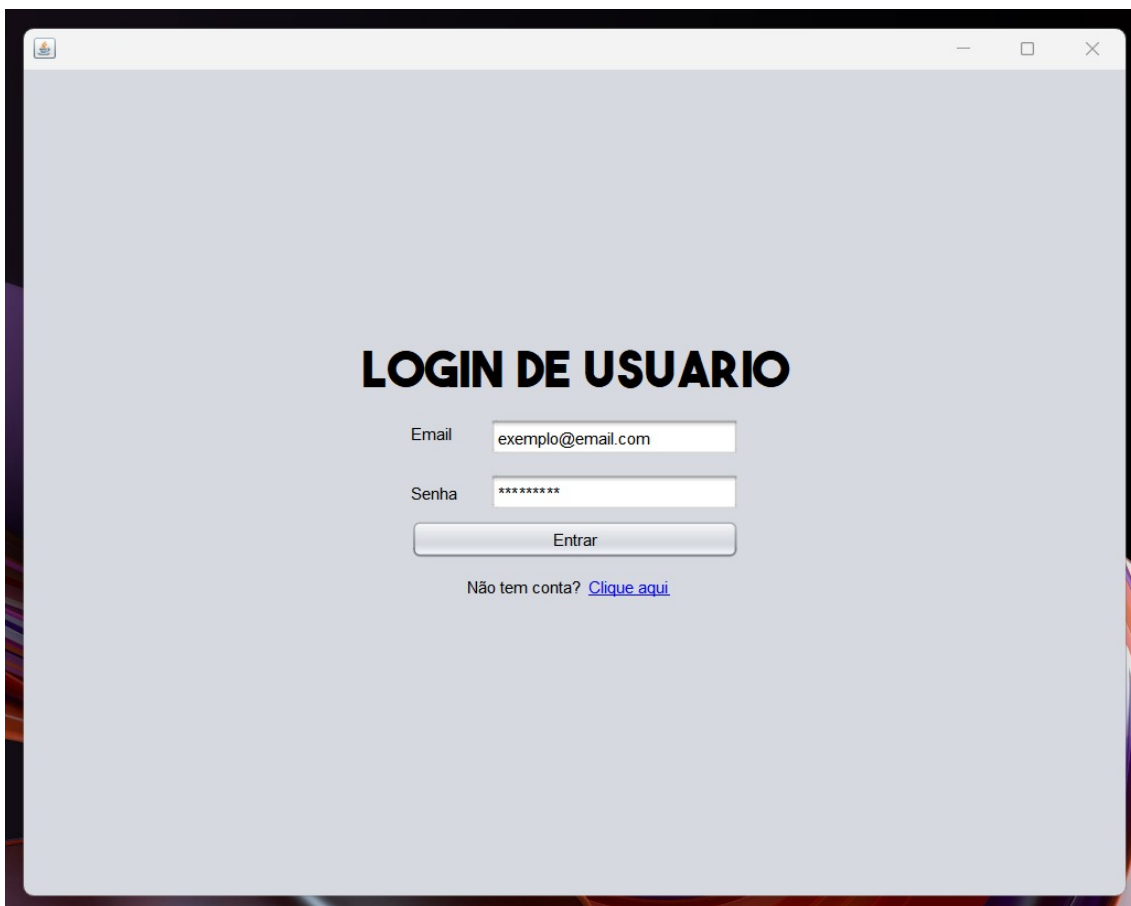
The image shows a web browser window with a light gray background. At the top center, the title "LOGIN DE USUARIO" is displayed in a large, bold, black font. Below the title, there are two input fields. The first is labeled "Email" and contains the text "exemplo@email.com". The second is labeled "Senha" and contains a series of asterisks "*****". Below these fields is a button labeled "Entrar". At the bottom of the form, there is a link that says "Não tem conta? [Clique aqui](#)". The browser window has a standard title bar with a small icon on the left and minimize, maximize, and close buttons on the right.

Figura 4. Tela de Login de Usuário

A tela de login permite que o usuário acesse o sistema de forma rápida. Para isso, ele deve inserir suas credenciais nos campos de e-mail e senha e pressionar o botão "Entrar". O sistema validará os dados informados e permitirá o acesso apenas se ambos estiverem corretos. Em caso de erro, um aviso será exibido informando que as credenciais são inválidas, garantindo que apenas usuários cadastrados possam entrar.

5.3. Tela Principal e mini telas(Bruno)

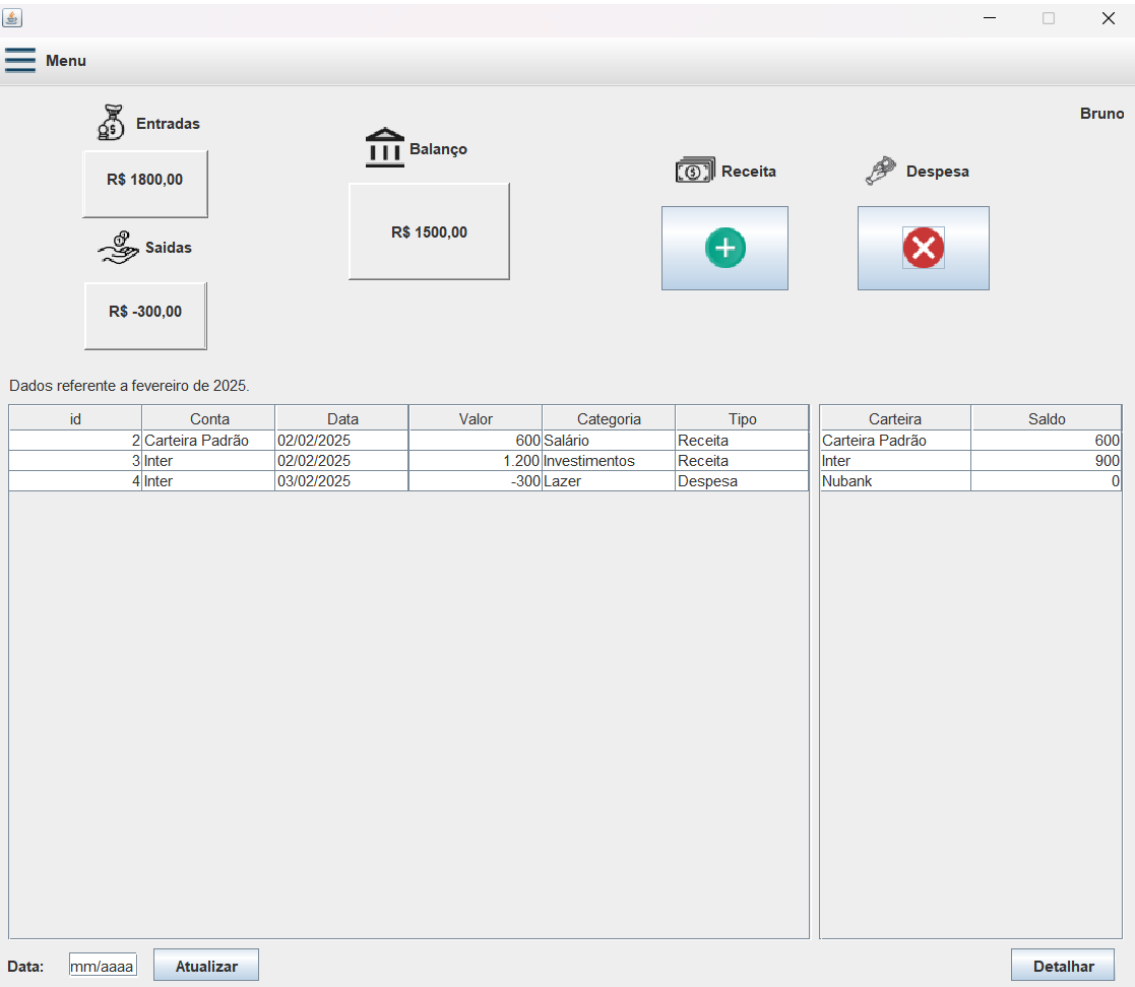


Figura 5. Tela: Principal

A tela principal do sistema oferece uma visão geral das operações financeiras do usuário, centralizando as informações em diferentes seções para facilitar o acesso e o acompanhamento. Na parte superior, são apresentados os valores totais referentes ao mês atual. A seção **Entradas** exibe o valor total de receitas registradas, enquanto a seção **Saídas** apresenta o valor total de despesas. Entre elas, encontra-se a seção **Balanço**, que calcula automaticamente o saldo atual com base nas entradas e saídas do mês.

Logo abaixo, encontram-se dois botões principais: **Receita** (Figura 6) e **Despesa** (Figura 7). Esses botões permitem que o usuário adicione novas transações financeiras. Ao clicar em qualquer um deles, uma nova janela é aberta, permitindo o registro de informações detalhadas sobre receitas ou despesas, como o valor, a data, a categoria e conta de onde foi debitada.

Figura 6.
Cadastro de receitas

Figura 7.
Cadastro de despesas

Figura 8.
Detalhes das despesas

A área central da tela é ocupada por uma tabela que lista as transações realizadas no mês atual. Essa tabela apresenta informações como a conta associada à transação, a data em que foi realizada, o valor e a categoria. Para cada transação, o usuário pode selecionar uma linha e clicar no botão **Detalhar** (Figura 8), que exibe informações adicionais, como a descrição da operação.

Por fim, no canto inferior direito, é exibido um resumo das carteiras financeiras do usuário, com os respectivos saldos atuais. O objetivo principal da tela é centralizar as informações financeiras do usuário, proporcionando um ambiente intuitivo e eficiente para consulta e análise das movimentações financeiras.

5.3.1. Menu de Navegação do Sistema

O menu principal permite que o usuário navegue entre diferentes telas, oferecendo acesso às principais funcionalidades do sistema.



Figura 9. Menu de Navegação do Sistema

5.4. Tela: Contas ()

Menu

Conta

Saldo

Data

Criar Editar Excluir Salvar Pesquisar

Conta	Saldo	Data
Inter	150.0	21/01/2025

Figura 10. Tela: Contas

A tela **Contas** permite que o usuário gerencie suas contas financeiras com diversas funcionalidades:

Para criar uma nova conta, basta clicar em **Criar Nova Conta**, onde o usuário poderá definir um nome e um saldo inicial para a conta. O nome da conta se refere ao banco ou instituição financeira, como, por exemplo, "Inter", "Nubank", entre outros. Uma vez criada, a conta será registrada e estará disponível para o gerenciamento.

Caso deseje modificar os dados de uma conta existente, o usuário pode selecionar a opção **Editar Conta**, permitindo que ele altere o nome, o saldo ou a data de criação da conta. Após a edição, o usuário pode confirmar as mudanças clicando em **Salvar Conta**, garantindo que os dados sejam atualizados corretamente.

Se for necessário excluir uma conta, a opção **Excluir Conta** está disponível. No entanto, uma confirmação será solicitada para evitar a exclusão acidental de uma conta.

Além disso, o usuário pode utilizar a opção **Pesquisar Conta** para localizar contas na tabela com base no nome, facilitando o gerenciamento e a busca de informações específicas.

Objetivo da Tela: Facilitar o gerenciamento de múltiplas contas financeiras, garantindo a precisão do saldo e a natureza das transações.

5.5. Tela: Categorias (■■■■■)

Nome:

Tipo:

Data:

Existem categorias padrão já criadas para facilitar a organização. As receitas incluem: Salário, Freelance, Investimentos e Aluguel. Já as despesas são categorizadas como Alimentação, Transporte, Saúde e Lazer.

Nome	Tipo	Data

Figura 11. Tela: Categorias

Nesta tela, o usuário pode gerenciar categorias de forma simples:

Para criar uma nova categoria, basta clicar em **Criar Nova Categoria**, onde será solicitado que o nome desejado seja inserido. A categoria será registrada na tabela e adicionada à lista de categorias dentro da conta. Após a criação, uma mensagem de confirmação aparecerá, informando que a categoria foi criada com sucesso.


Caso queira excluir uma categoria, é necessário selecionar a linha correspondente na tabela e, em seguida, clicar em **Excluir Categoria**. Uma confirmação será solicitada para garantir que a exclusão não seja acidental.

Se o objetivo for editar o nome de uma categoria, basta selecionar a opção **Editar Categoria**, fazer a alteração desejada e, ao clicar em **Salvar Categoria**, a mudança será aplicada.

Para pesquisar categorias, é possível usar o **Pesquisar Categoria**, que realiza a busca com base na data de criação. Caso nenhuma data seja informada, uma mensagem de erro será exibida, informando que o campo de data é obrigatório para a consulta.

Por fim, ao editar o nome de uma categoria, o botão **Salvar Categoria** deve ser pressionado para confirmar e efetivar as alterações.

5.6. Tela: Histórico (██████)

 Menu

Conta	Data	Valor	Tipo	Categoria
1	21/02/2025	-300.0	Despesa	alimentação
1	12/02/2025	-50.0	Despesa	saúde
1	26/02/2025	-130.0	Despesa	lazer
1	10/02/2025	500.0	Receita	salário
1	28/02/2025	1000.0	Receita	freelance
1	23/02/2025	69.0	Receita	investimentos

Data: 02 / 2025

Figura 12. Tela: Histórico

A tela **Historico** é dedicada a exibir uma lista detalhada de todas as transações do usuário, organizadas em uma tabela que contém colunas como "Conta", "Data", "Valor", "Tipo" e "Categoria/Origem". Essa tela oferece ao usuário a flexibilidade de filtrar as transações de três maneiras: por **conta ou categoria/origem**, por **mês** e/ou por **ano**.

Os filtros podem ser usados separadamente ou em conjunto. Por exemplo, se o usuário digitar 2023 no campo do ano e clicar em **Pesquisar**, a tabela será atualizada para exibir apenas as transações daquele ano. Da mesma forma, se ele digitar 10 no campo do mês, a tabela mostrará apenas as transações de outubro, independentemente do ano. Se ele usar esses dois filtros e adicionar ainda o número de uma conta ou o nome de uma categoria/origem, a tabela mostrará apenas as transações daquele filtro no mês de outubro de 2023.

A tabela é dinâmica e atualizada automaticamente conforme o usuário digita no campo de texto, seja para filtrar por conta ou por categoria. Isso significa que, ao digitar "Alimentação" no campo de texto, a tabela já começa a exibir apenas as transações relacionadas à categoria "Alimentação", sem a necessidade de clicar em "Pesquisar". Essa funcionalidade é possível graças a um **DocumentListener**, que monitora as alterações nos campos de texto em tempo real.

Além disso, há um botão **Limpar** que remove todos os filtros aplicados.

5.7. Tela: Gráfico ()

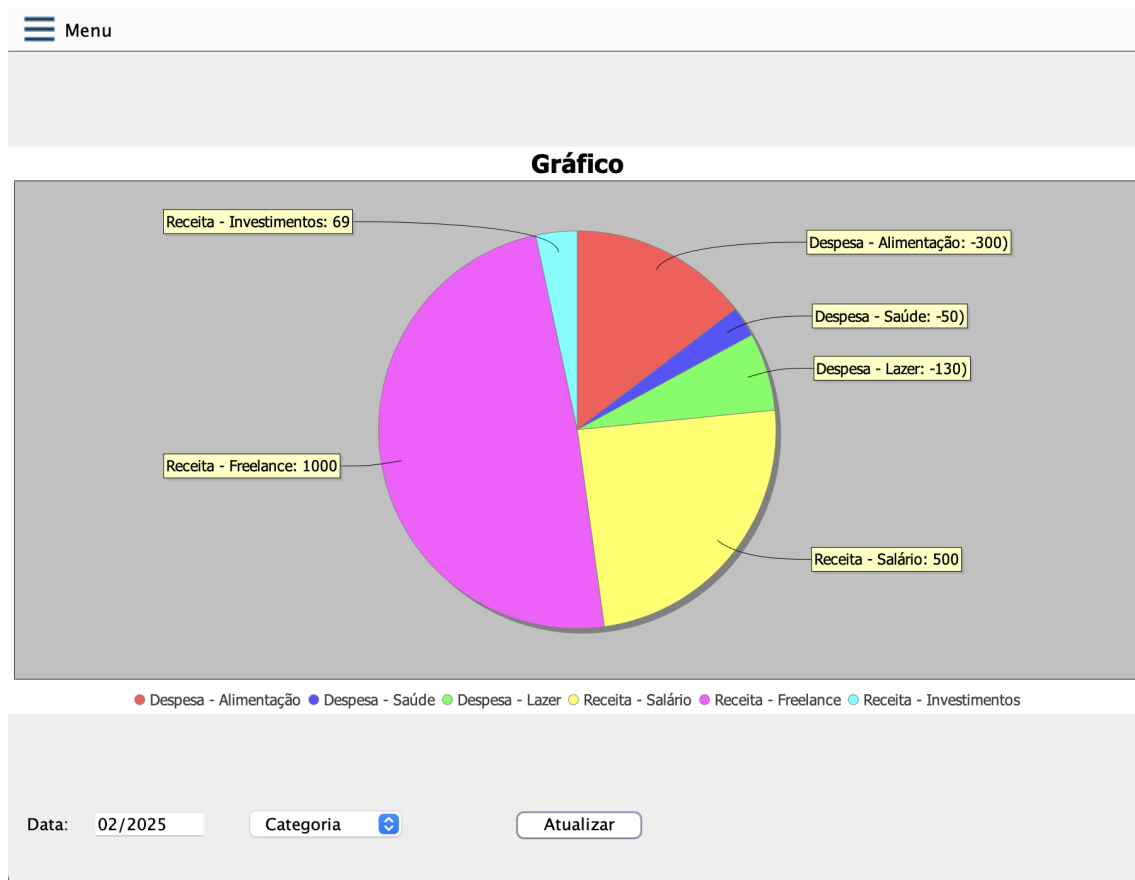


Figura 13. Tela: Gráfico

A tela do **Gráfico** tem como objetivo principal fornecer uma representação visual das transações financeiras do usuário, permitindo uma análise rápida e intuitiva de padrões.

Quando o usuário abre essa tela, o sistema carrega automaticamente um gráfico de pizza que exibe as transações separadas por receitas e despesas do mês e ano atual registrados no sistema. O gráfico é gerado usando a biblioteca **JFreeChart**, que permite a criação de gráficos dinâmicos e interativos.

Ao clicar em **Atualizar** o usuário pode filtrar os dados por mês e ano, além de escolher entre visualizar as transações por categoria ou por tipo (receitas e despesas). Por exemplo, se o usuário selecionar o mês de outubro de 2023 e o tipo "Categoria", o gráfico mostrará a proporção entre as categorias daquele período.

As despesas são exibidas com valores negativos, facilitando a distinção entre os dois tipos de transações.

Além disso, ao passar o mouse sobre as seções do gráfico, o usuário pode ver detalhes como o valor e a porcentagem de cada categoria ou tipo de transação.

6. Classes

Classes Desenvolvidas por Bruno

Listing 1. Classe Carteira

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package classesGerenciador;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * A classe `Carteira` representa uma carteira financeira no sistema.
12  * Cada carteira possui um identificador único, nome, saldo atual e uma lista de transações associadas.
13  * Atribui IDs automaticamente para cada instância, usando uma variável estática.
14  *
15  * @author Bruno Eduardo <https://github.com/brnduol>
16  */
17 public class Carteira {
18     private static int proximoId = 1; // Variável estática para controlar o próximo ID único a ser atribuído.
19     private int id; // Identificador único de cada carteira.
20     private String nome; // Nome da carteira.
21     private int saldo; // Saldo atual da carteira.
22     private List<Transacao> transacoes; // Lista de transações associadas à carteira.
23
24     // Construtor vazio, necessário para casos onde a carteira será inicializada posteriormente.
25     public Carteira() {}
26
27     // Construtor principal, inicializa a carteira com nome e saldo. Gera automaticamente o ID único.
28     public Carteira(String nome, int saldo) {
29         this.id = proximoId++; // Atribui o próximo ID e incrementa para a próxima instância.
30         this.nome = nome;
31         this.saldo = saldo;
32         this.transacoes = new ArrayList<>(); // Inicializa a lista de transações vazia.
33     }
34
35     // Métodos estáticos para obter e configurar o próximo ID único.
36     public static int getProximoId() {
37         return proximoId;
38     }
39
40     public static void setProximoId(int proximoId) {
41         Carteira.proximoId = proximoId; // Permite redefinir o próximo ID, caso necessário.
42     }
43
44     // Métodos getters e setters para os atributos principais da classe.
45
46     public int getId() {
47         return id; // Retorna o identificador único da carteira.
48     }
49
50     public void setId(int id) {
51         this.id = id; // Permite configurar manualmente o ID da carteira.
52     }
53
54     public String getNome() {
55         return nome; // Retorna o nome da carteira.
56     }
57
58     public void setNome(String nome) {
59         this.nome = nome; // Atualiza o nome da carteira.
60     }
61
62     public int getSaldo() {
63         return saldo; // Retorna o saldo atual da carteira.
64     }
65
66     public void setSaldo(int saldo) {
67         this.saldo = saldo; // Permite atualizar diretamente o saldo da carteira.
68     }
69
70     // Adiciona uma transação à carteira. A transação também atualiza o saldo da carteira automaticamente.
71     public void adicionarTransacao(Transacao transacao) {
72         transacao.atualizar(this); // Atualiza o saldo da carteira com base na transação.
73         transacoes.add(transacao); // Adiciona a transação à lista de transações.
74     }
75
76     // Remove uma transação da carteira com base no ID da transação fornecido.
77     public void removerTransacao(int id) {
78         for (int i = 0; i < transacoes.size(); i++) {
79             if (transacoes.get(i).getId() == id) { // Verifica se o ID da transação corresponde ao informado.
80                 transacoes.remove(i); // Remove a transação encontrada.
81                 break; // Interrompe o loop após remover a transação.
82             }
83         }
84     }
85
86     // Atualiza o saldo da carteira somando ou subtraindo o valor fornecido.
87     public void atualizarSaldo(int valor) {
```

```

88         this.saldo += valor; // Ajusta o saldo da carteira.
89     }
90 }

```

Listing 2. Classe Usuário

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package classesGerenciador;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * A classe `Usuario` representa um usuário do sistema.
12  * Cada usuário possui um identificador único, nome, e-mail, senha, e diversas listas relacionadas às suas operações
13  * financeiras:
14  * - carteiras, categorias de despesas, origens de renda e transações.
15  * - A classe inclui inicialização padrão de carteiras, categorias e origens de renda para facilitar a experiência do
16  *   usuário.
17  *
18  * @author Bruno Eduardo <https://github.com/brnduol>
19  */
20 public class Usuario {
21     private static int proximoId = 1; // Controle estático para geração de IDs únicos para cada usuário.
22     private int id; // Identificador único de cada usuário.
23     private String nome; // Nome do usuário.
24     private String email; // E-mail do usuário.
25     private String senha; // Senha do usuário.
26     private List<Carteira> carteiras; // Lista de carteiras do usuário.
27     private List<Categoria> categorias; // Lista de categorias de despesas padrão associadas ao usuário.
28     private List<OrigemRenda> origemRendas; // Lista de origens de renda padrão associadas ao usuário.
29     private List<Transacao> transacoes; // Lista de transações realizadas pelo usuário.
30
31     // Construtor vazio, necessário para instâncias criadas sem parâmetros.
32     public Usuario() {}
33
34     // Construtor principal, inicializa o usuário com nome, senha e e-mail.
35     public Usuario(String nome, String senha, String email) {
36         this.id = proximoId++; // Atribui um ID único ao usuário.
37         this.nome = nome;
38         this.senha = senha;
39         this.email = email;
40
41         // Inicialização das listas que armazenam objetos relacionados ao usuário.
42         this.carteiras = new ArrayList<>();
43         this.categorias = new ArrayList<>();
44         this.origemRendas = new ArrayList<>();
45         this.transacoes = new ArrayList<>();
46
47         // Adiciona uma carteira padrão ao usuário, com saldo inicial de zero.
48         Carteira carteiraFixa = new Carteira("Carteira_Padrão", 0);
49         this.carteiras.add(carteiraFixa);
50
51         // Adiciona categorias padrão associadas ao usuário.
52         categorias.add(new Categoria("Alimentação"));
53         categorias.add(new Categoria("Transporte"));
54         categorias.add(new Categoria("Saúde"));
55         categorias.add(new Categoria("Lazer"));
56         System.out.println("Categorias_criadas:");
57         for (Categoria categoria : categorias) {
58             System.out.println(categoria.getNomeCategoria());
59         }
60
61         // Adiciona origens de renda padrão associadas ao usuário.
62         origemRendas.add(new OrigemRenda("Salário"));
63         origemRendas.add(new OrigemRenda("Freelance"));
64         origemRendas.add(new OrigemRenda("Investimentos"));
65         origemRendas.add(new OrigemRenda("Aluguel"));
66         System.out.println("Origens_de_renda_criadas:");
67         for (OrigemRenda origemRenda : origemRendas) {
68             System.out.println(origemRenda.getNomeOrigemRenda());
69         }
70     }
71
72     // Métodos getters para acessar os atributos do usuário.
73     public int getId() {
74         return id;
75     }
76
77     public String getNome() {
78         return nome;
79     }
80
81     public String getEmail() {
82         return email;
83     }
84
85     public String getSenha() {
86         return senha;
87     }
88 }

```

```

87     public List<Carteira> getCarteiras() {
88         return carteiras;
89     }
90
91     public List<OrigemRenda> getOrigemRendas() {
92         return origemRendas;
93     }
94
95     public List<Categoria> getCategorias() {
96         return categorias;
97     }
98
99     public List<Transacao> getTransacoes() {
100         return transacoes;
101     }
102
103     // Métodos setters para modificar os atributos do usuário.
104     public void setId(int id) {
105         this.id = id;
106     }
107
108     public void setNome(String nome) {
109         this.nome = nome;
110     }
111
112     public void setEmail(String email) {
113         this.email = email;
114     }
115
116     public void setSenha(String senha) {
117         this.senha = senha;
118     }
119
120     public void setCarteiras(List<Carteira> carteiras) {
121         this.carteiras = carteiras;
122     }
123
124     // Método para criar e adicionar uma nova carteira à lista de carteiras do usuário.
125     public void criarCarteira(String nome, int saldo) {
126         Carteira carteira = new Carteira(nome, saldo);
127         carteiras.add(carteira);
128     }
129
130     // Método para adicionar uma transação a uma carteira específica.
131     public void adicionarTransacao(Carteira carteira, Transacao transacao) {
132         carteira.adicionarTransacao(transacao);
133     }
134 }

```

Listing 3. Classe OrigemRenda

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package classesGerenciador;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * A classe `OrigemRenda` representa uma fonte de renda no sistema financeiro.
12  * Cada origem de renda possui um identificador único, um nome, e uma lista de transações associadas.
13  * Essa classe permite gerenciar as transações vinculadas à origem de renda, oferecendo métodos para
14  * adicionar ou excluir transações.
15  *
16  * @author Bruno Eduardo <https://github.com/brnduol>
17  */
18 public class OrigemRenda {
19     private static int proximoId = 1; // Variável estática para gerar IDs únicos para cada origem de renda.
20     private int id; // Identificador único para cada instância de OrigemRenda.
21     private String nomeOrigemRenda; // Nome da origem de renda (ex.: Salário, Freelance, etc.).
22     private List<Transacao> transacoes; // Lista de transações associadas a esta origem de renda.
23
24     // Construtor que inicializa a origem de renda com um nome e um ID único.
25     // Também inicializa a lista de transações como vazia.
26     public OrigemRenda(String nome) {
27         this.id = proximoId++; // Atribui o próximo ID disponível e incrementa o contador.
28         this.nomeOrigemRenda = nome;
29         this.transacoes = new ArrayList<>(); // Inicializa a lista de transações.
30     }
31
32     // Retorna o identificador único desta origem de renda.
33     public int getId() {
34         return id;
35     }
36
37     // Retorna a lista de transações associadas a esta origem de renda.
38     public List<Transacao> getTransacoes() {
39         return transacoes;
40     }
41
42     // Adiciona uma transação à lista de transações desta origem de renda.
43     public void addTransacao(Transacao transacao) {

```

```

44         transacoes.add(transacao); // Adiciona a transação à lista.
45     }
46
47     // Remove uma transação da lista com base no identificador (ID) da transação.
48     // Retorna `true` se a transação foi encontrada e removida, ou `false` caso contrário.
49     public boolean excluirTransacao(int transacaoId) {
50         return transacoes.removeIf(transacao -> transacao.getId() == transacaoId); // Remove a transação com ID
51         correspondente.
52     }
53
54     // Retorna o nome da origem de renda.
55     public String getNomeOrigemRenda() {
56         return nomeOrigemRenda;
57     }
58
59     // Atualiza o nome desta origem de renda.
60     public void setNomeOrigemRenda(String nomeOrigemRenda) {
61         this.nomeOrigemRenda = nomeOrigemRenda;
62     }
63 }

```

Listing 4. Classe ContasUsuarios

```

1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package classesGerenciador;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * A classe `ContasUsuarios` gerencia a lista de usuários cadastrados no sistema,
12  * permitindo a criação, autenticação e controle de sessão de usuários.
13  * Implementa o padrão Singleton, garantindo que apenas uma instância dessa classe
14  * exista durante a execução do programa.
15  *
16  * @author Bruno Eduardo <https://github.com/brnduol>
17  */
18 public class ContasUsuarios {
19     private List<Usuario> usuarios; // Lista de usuários cadastrados no sistema.
20     private Usuario usuarioAtual; // Usuário atualmente autenticado.
21
22     private static ContasUsuarios instancia; // Instância única da classe (Singleton).
23
24     // Construtor privado para impedir a criação de múltiplas instâncias.
25     private ContasUsuarios() {
26         usuarios = new ArrayList<>();
27     }
28
29     // Método Singleton para obter a instância única de `ContasUsuarios`.
30     public static ContasUsuarios getInstance() {
31         if (instancia == null) {
32             instancia = new ContasUsuarios(); // Cria a instância única se ainda não existir.
33         }
34         return instancia;
35     }
36
37     // Método para criar um novo usuário no sistema.
38     // Retorna `true` se o cadastro for bem-sucedido e `false` caso o e-mail já esteja cadastrado.
39     public boolean criarConta(String nome, String senha, String email) {
40         for (Usuario usuario : usuarios) {
41             if (usuario.getEmail().equals(email)) { // Verifica se o e-mail já está cadastrado.
42                 System.out.println("E-mail_já_cadastrado!");
43                 return false;
44             }
45         }
46
47         Usuario novoUsuario = new Usuario(nome, senha, email);
48         usuarios.add(novoUsuario); // Adiciona o novo usuário à lista de usuários.
49         System.out.println("Conta_criada_com_sucesso!");
50         return true;
51     }
52
53     // Método para autenticar um usuário no sistema.
54     // Retorna `true` se o login for bem-sucedido e `false` caso contrário.
55     public boolean entrarConta(String email, String senha) {
56         for (Usuario usuario : usuarios) {
57             if (usuario.getEmail().equals(email) && usuario.getSenha().equals(senha)) {
58                 System.out.println("Login_bem-sucedido!");
59                 usuarioAtual = usuario; // Define o usuário autenticado.
60                 return true;
61             }
62         }
63         System.out.println("E-mail_ou_senha_incorretos.");
64         return false;
65     }
66
67     // Retorna o usuário atualmente autenticado no sistema.
68     public Usuario conta() {
69         return usuarioAtual;
70     }
71 }

```

```

72 // Método para encerrar a sessão do usuário, removendo o usuário autenticado.
73 public void sairConta() {
74     usuarioAtual = null; // Define o usuário atual como `null`, efetivamente realizando o logout.
75 }
76 }

```

Listing 5. Classe ControleTelas

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license/default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package projeto.gerenciador.financeiro;
6
7  import telas.Cadastro1;
8  import telas.Login1;
9  import telas.TelaCategorial;
10 import telas.TelaContas1;
11 import telas.TelaGrafico1;
12 import telas.TelaHistorico1;
13 import telas.TelaPrincipal1;
14 import telas.mini.TelaDetalhar;
15
16 /**
17  * A classe `ControleTelas` gerencia todas as telas do sistema,
18  * garantindo que cada uma seja instanciada apenas uma vez através do padrão Singleton.
19  * Esse gerenciamento evita a criação repetitiva de objetos e melhora o desempenho da aplicação.
20  *
21  * @author Bruno Eduardo <https://github.com/brnduol>
22  */
23 public class ControleTelas {
24     private static ControleTelas instance; // Instância única da classe (Singleton).
25
26     private Cadastro1 telaCadastro;
27     private Login1 telaLogin;
28     private TelaCategorial telaCategoria;
29     private TelaContas1 telaContas;
30     private TelaGrafico1 telaGrafico;
31     private TelaHistorico1 telaHistorico;
32     private TelaPrincipal1 telaPrincipal;
33     private TelaDetalhar telaDetalhar;
34
35     // Construtor privado para impedir a criação de múltiplas instâncias da classe.
36     private ControleTelas() {}
37
38     // Método Singleton para obter a instância única da classe.
39     public static ControleTelas getInstance() {
40         if (instance == null) {
41             instance = new ControleTelas(); // Cria a instância única se ainda não existir.
42         }
43         return instance;
44     }
45
46     // Métodos que retornam as telas do sistema, criando-as apenas se necessário.
47
48     // Retorna a tela de cadastro, instanciando-a caso ainda não tenha sido criada.
49
50     public Cadastro1 getTelaCadastro() {
51         if (telaCadastro == null) {
52             telaCadastro = new Cadastro1();
53         }
54         return telaCadastro;
55     }
56 }
57
58 // Retorna a tela de login, instanciando-a caso ainda não tenha sido criada.
59
60 public Login1 getTelaLogin() {
61     if (telaLogin == null) {
62         telaLogin = new Login1();
63     }
64     return telaLogin;
65 }
66
67 // Retorna a tela de categorias, instanciando-a caso ainda não tenha sido criada.
68
69 public TelaCategorial getTelaCategoria() {
70     if (telaCategoria == null) {
71         telaCategoria = new TelaCategorial();
72     }
73     return telaCategoria;
74 }
75
76 // Retorna a tela de contas, instanciando-a caso ainda não tenha sido criada.
77
78 public TelaContas1 getTelaContas() {
79     if (telaContas == null) {
80         telaContas = new TelaContas1();
81     }
82     return telaContas;
83 }
84
85 // Retorna a tela de gráficos, instanciando-a caso ainda não tenha sido criada.
86

```

```
87
88 public TelaGrafico1 getTelaGrafico() {
89     if (telaGrafico == null) {
90         telaGrafico = new TelaGrafico1();
91     }
92     return telaGrafico;
93 }
94
95 // Retorna a tela de histórico de transações, instanciando-a caso ainda não tenha sido criada.
96 public TelaHistorico1 getTelaHistorico() {
97     if (telaHistorico == null) {
98         telaHistorico = new TelaHistorico1();
99     }
100    return telaHistorico;
101 }
102
103 // Retorna a tela principal do sistema, instanciando-a caso ainda não tenha sido criada.
104
105 public TelaPrincipal1 getTelaPrincipal() {
106     if (telaPrincipal == null) {
107         telaPrincipal = new TelaPrincipal1();
108     }
109     return telaPrincipal;
110 }
111
112
113 // Retorna a tela de detalhes, instanciando-a caso ainda não tenha sido criada.
114
115 public TelaDetalhar getTelaDetalhar() {
116     if (telaDetalhar == null) {
117         telaDetalhar = new TelaDetalhar();
118     }
119     return telaDetalhar;
120 }
121 }
```