Um problema que possa ser resolvido com a programação:

Gerenciar finanças: Criar um aplicativo para controlar seus gastos, acompanhar investimentos ou criar orçamentos. Isso pode ajudar a economizar dinheiro e alcançar seus objetivos financeiros.

Funcionalidades essenciais para um sistema de gerenciamento financeiro completo:

1. Rastreamento de transações:

- **Entrada manual de transações:** O sistema deve permitir a entrada manual de transações, incluindo data, valor, categoria, descrição e conta bancária.
- **Importação de extratos bancários:** O sistema deve ser capaz de importar automaticamente extratos bancários em formatos comuns (como CSV, OFX ou Quicken).
- **Categorização automática:** O sistema deve categorizar automaticamente as transações com base em regras personalizáveis ou aprendizado de máquina.

2. Monitoramento de contas:

- **Visão geral do saldo:** O sistema deve fornecer uma visão geral atualizada do saldo de todas as contas bancárias e investimentos.
- **Histórico de transações:** O sistema deve fornecer um histórico detalhado de todas as transações, com filtros por data, conta, categoria e valor.
- **Alertas de saldo:** O sistema deve configurar alertas para notificá-lo quando o saldo de uma conta atingir um determinado nível ou quando uma transação incomum for realizada.

3. Criação de orçamentos:

- **Definição de metas de gastos:** O sistema deve permitir que você defina metas de gastos mensais ou anuais para diferentes categorias.
- **Acompanhamento do orçamento:** O sistema deve comparar seus gastos reais com suas metas de orçamento e fornecer relatórios sobre o desempenho do orçamento.

- **Alertas de orçamento:** O sistema deve configurar alertas para notificá-lo quando você estiver se aproximando de uma meta de orçamento ou quando exceder seus limites de gastos.

4. Planejamento financeiro:

- **Definição de metas financeiras:** O sistema deve permitir que você defina metas financeiras de longo prazo, como comprar uma casa ou se aposentar.
- **Calculadoras financeiras:** O sistema deve fornecer calculadoras financeiras para ajudá-lo a estimar o custo de grandes compras, calcular juros e avaliar investimentos.
- **Simulações financeiras:** O sistema deve permitir que você execute simulações financeiras para ver como diferentes decisões financeiras afetarão seus objetivos.

5. Relatórios e análises:

- **Relatórios de gastos:** O sistema deve gerar relatórios detalhados sobre seus gastos, por categoria, conta bancária e período de tempo.
- **Análise de tendências:** O sistema deve analisar seus dados financeiros para identificar tendências e padrões de gastos.
- **Relatórios de patrimônio líquido:** O sistema deve calcular e acompanhar seu patrimônio líquido ao longo do tempo

A Eficácia da IA na Identificação de Requisitos Funcionais para Gerenciar Finanças:

A inteligência artificial (IA) tem demonstrado grande potencial para auxiliar em diversas áreas, incluindo o gerenciamento de finanças pessoais. A capacidade da IA de processar grandes volumes de dados, identificar padrões e realizar análises complexas abre caminho para o desenvolvimento de ferramentas inovadoras que podem auxiliar os usuários na tomada de decisões financeiras mais conscientes e eficientes.

No entanto, a questão da **eficácia da IA na identificação dos requisitos funcionais** específicos para gerenciar finanças ainda é um tema em debate. É crucial avaliar se a IA pode realmente compreender as necessidades e objetivos individuais de cada

usuário, considerando a complexa natureza das finanças pessoais e as diversas variáveis envolvidas.

Para embasar essa análise, recorreremos aos livros da bibliografia do curso que abordam os princípios da engenharia de software e a elicitação de requisitos:

1. "Análise e Especificação de Requisitos de Software" (Sommerville & Kolak);

2. "Engenharia de Software: Uma Abordagem Moderna" (Pressman & Bruce);

3. "Requisitos Funcionais e Não Funcionais" (Goguen & Linde);

Classes:

1. **Conta:**
    - **Propriedades:**
        - Nome: Identifica o nome da conta (corrente, poupança, investimento, etc.).
        - Saldo: Representa o valor monetário disponível na conta.
        - Tipo: Classifica a conta (corrente, poupança, investimento, etc.).
        - Instituição: Indica a instituição financeira à qual a conta pertence.
    - **Métodos:**
        - Depositar(valor): Adiciona um valor ao saldo da conta.
        - Sacar(valor): Remove um valor do saldo da conta, se houver saldo suficiente.
        - Transferir(valor, contaDestino): Transfere um valor para outra conta.
        - ObterExtrato(): Retorna o histórico de transações da conta.
2. **Transação:**
    - **Propriedades:**
        - Data: Registra a data da transação.

        - Valor: Representa o valor monetário da transação.
        - Descrição: Fornece uma breve descrição da transação.
        - Categoria: Classifica a transação por tipo de gasto (alimentação, transporte, etc.).
        - Conta: Indica a conta na qual a transação foi realizada.
    - **Métodos:**
        - EditarDescrição(novaDescrição): Permite modificar a descrição da transação.
        - EditarCategoria(novaCategoria): Permite reclassificar a transação por categoria.
3. **Categoria:**
    - **Propriedades:**
        - Nome: Identifica o nome da categoria (alimentação, transporte, etc.).
        - Orçamento Mensal: Define um limite de gastos mensais para a categoria.
    - **Métodos:**
        - EditarOrcamento(novoOrcamento): Permite modificar o limite de gastos mensais.
4. **Relatório:**
    - **Propriedades:**
        - Tipo: Indica o tipo de relatório (resumo de gastos, análise de categorias, etc.).
        - Período: Define o intervalo de tempo abrangido pelo relatório.
    - **Métodos:**
        - GerarRelatorio(): Cria e formata o relatório de acordo com o tipo e período selecionados.

Relacionamentos entre Classes:

- Uma conta pode ter várias transações.
- Uma transação pertence a uma conta e a uma categoria.
- Uma categoria pode ter várias transações.
- Um relatório pode ser gerado a partir de várias transações e categorias.

Resolução de Problemas Financeiros:

O sistema gerencia finanças através da interação entre as classes:

- **Rastreamento de Gastos:** As transações registram cada compra ou movimentação financeira, categorizando-as para análise posterior.
- **Controle de Orçamento:** O sistema compara os gastos por categoria com os orçamentos definidos, alertando o usuário sobre possíveis excedentes.
- **Geração de Relatórios:** Relatórios personalizados fornecem insights sobre o comportamento financeiro, auxiliando na identificação de áreas de economia e na tomada de decisões conscientes.
- **Planejamento Financeiro:** O sistema permite definir metas financeiras e acompanhar o progresso ao longo do tempo, auxiliando na realização de objetivos.

Conclusão:

O sistema de gerenciamento financeiro, composto pelas classes, propriedades e métodos descritos, fornece uma ferramenta completa para organizar, analisar e controlar as finanças pessoais, promovendo a saúde financeira e a realização de objetivos.

Estratégia de Programação com IA:

Exercícios para praticar:

Nível Iniciante:

1. Rastreamento Manual de Gastos:

- **Objetivo:** Registrar todas as suas despesas por um período de uma semana, categorizando-as por tipo de gasto (alimentação, transporte, lazer, etc.).
- **Atividade:**
    - Utilize um caderno, planilha ou aplicativo de notas para anotar cada compra, incluindo data, valor, local e categoria.
    - No final da semana, calcule o total gasto em cada categoria e identifique as áreas que mais impactam seu orçamento.

2. Criação de um Orçamento Simples:

- **Objetivo:** Definir um orçamento mensal para suas despesas fixas e variáveis.
- **Atividade:**
    - Liste todas as suas despesas fixas (aluguel, contas de luz, água, internet, etc.) e variáveis (alimentação, transporte, lazer, etc.).
    - Determine o valor total da sua renda mensal.
    - Subtraia o total de despesas fixas da sua renda mensal para obter o valor disponível para despesas variáveis.
    - Aloque o valor disponível para despesas variáveis entre as categorias, definindo um limite máximo para cada uma.

5. **Codificação do Programa:**

import javax.swing.*; import java.awt.*; import java.awt.event.ActionEvent; import java.awt.event.ActionListener; import java.text.DecimalFormat; import java.util.ArrayList; import java.util.List; // Classe principal que inicia o aplicativo public class FinancialApp { public static void main(String[] args) { SwingUtilities.invokeLater(() -> { FinancialFrame frame = new FinancialFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); }); } } // Frame principal que contém a interface gráfica do aplicativo class FinancialFrame extends JFrame { private JTabbedPane tabbedPane; private ExpensePanel expensePanel; private InvestmentPanel investmentPanel; private BudgetPanel budgetPanel; public FinancialFrame() { setTitle("Financial Manager"); setSize(600, 400); setLocationRelativeTo(null); tabbedPane = new JTabbedPane(); expensePanel = new ExpensePanel(); investmentPanel = new InvestmentPanel(); budgetPanel = new BudgetPanel(); tabbedPane.addTab("Expenses", expensePanel); tabbedPane.addTab("Investments", investmentPanel); tabbedPane.addTab("Budgets", budgetPanel); getContentPane().add(tabbedPane); } } // Panel para controle de despesas class ExpensePanel extends JPanel { private DefaultListModel<Expense> expenseListModel; private JList<Expense> expenseList; private JTextField dateField, categoryField, amountField; public ExpensePanel() { setLayout(new BorderLayout()); expenseListModel = new DefaultListModel<>(); expenseList = new JList<>(expenseListModel); JScrollPane scrollPane = new JScrollPane(expenseList); add(scrollPane, BorderLayout.CENTER); JPanel inputPanel = new JPanel(new GridLayout(4, 2)); inputPanel.add(new JLabel("Date (yyyy-mm-dd):")); dateField = new JTextField(); inputPanel.add(dateField); inputPanel.add(new JLabel("Category:"));

```
categoryField = new JTextField(); inputPanel.add(categoryField); inputPanel.add(new JLabel("Amount ($):")); amountField = new JTextField(); inputPanel.add(amountField); JButton addButton = new JButton("Add Expense"); addButton.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent e) { try { String date = dateField.getText(); String category = categoryField.getText(); double amount = Double.parseDouble(amountField.getText()); Expense expense = new Expense(date, category, amount); expenseListModel.addElement(expense); dateField.setText(""); categoryField.setText(""); amountField.setText(""); } catch (NumberFormatException ex) { JOptionPane.showMessageDialog(ExpensePanel.this, "Invalid amount. Please enter a valid number.", "Error", JOptionPane.ERROR_MESSAGE); } } }); inputPanel.add(addButton); add(inputPanel, BorderLayout.SOUTH); } } // Panel para controle de investimentos class InvestmentPanel extends JPanel { private DefaultListModel<Investment> investmentListModel; private JList<Investment> investmentList; private JTextField investmentNameField, investmentAmountField; public InvestmentPanel() { setLayout(new BorderLayout()); investmentListModel = new DefaultListModel<>(); investmentList = new JList<>(investmentListModel); JScrollPane scrollPane = new JScrollPane(investmentList); add(scrollPane, BorderLayout.CENTER); JPanel inputPanel = new JPanel(new GridLayout(3, 2)); inputPanel.add(new JLabel("Investment Name:")); investmentNameField = new JTextField(); inputPanel.add(investmentNameField); inputPanel.add(new JLabel("Initial Amount ($):")); investmentAmountField = new JTextField(); inputPanel.add(investmentAmountField); JButton addButton = new JButton("Add Investment"); addButton.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent e) { try { String name = investmentNameField.getText(); double amount = Double.parseDouble(investmentAmountField.getText()); Investment investment = new Investment(name, amount); investmentListModel.addElement(investment); investmentNameField.setText(""); investmentAmountField.setText(""); } catch (NumberFormatException ex) { JOptionPane.showMessageDialog(InvestmentPanel.this, "Invalid amount. Please enter a valid number.", "Error", JOptionPane.ERROR_MESSAGE); } } }); inputPanel.add(addButton); add(inputPanel, BorderLayout.SOUTH); } } // Panel para controle de orçamentos class BudgetPanel extends JPanel { private List<Budget> budgets; private JTextField categoryField, budgetAmountField; private JTextArea budgetTextArea; public BudgetPanel() { setLayout(new BorderLayout()); budgets = new ArrayList<>(); JPanel inputPanel = new JPanel(new GridLayout(3, 2)); inputPanel.add(new JLabel("Category:")); categoryField = new JTextField();
```

```
inputPanel.add(categoryField); inputPanel.add(new JLabel("Budget Amount ($):")); budgetAmountField = new JTextField(); inputPanel.add(budgetAmountField); JButton addButton = new JButton("Add Budget"); addButton.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent e) { try { String category = categoryField.getText(); double amount = Double.parseDouble(budgetAmountField.getText()); Budget budget = new Budget(category, amount); budgets.add(budget); updateBudgetTextArea(); categoryField.setText(""); budgetAmountField.setText(""); } catch (NumberFormatException ex) { JOptionPane.showMessageDialog(BudgetPanel.this, "Invalid amount. Please enter a valid number.", "Error", JOptionPane.ERROR_MESSAGE); } } }); inputPanel.add(addButton); add(inputPanel, BorderLayout.NORTH); budgetTextArea = new JTextArea(10, 30); budgetTextArea.setEditable(false); JScrollPane scrollPane = new JScrollPane(budgetTextArea); add(scrollPane, BorderLayout.CENTER); } private void updateBudgetTextArea() { DecimalFormat df = new DecimalFormat("#.##"); StringBuilder sb = new StringBuilder(); sb.append("Category\t\tBudget Amount ($)\n"); sb.append("---------------------------------------\n"); for (Budget budget : budgets) { sb.append(budget.getCategory()).append("\t\t\t"); sb.append(df.format(budget.getAmount())).append("\n"); } budgetTextArea.setText(sb.toString()); } } // Classe para representar uma despesa class Expense { private String date; private String category; private double amount; public Expense(String date, String category, double amount) { this.date = date; this.category = category; this.amount = amount; } @Override public String toString() { return "[" + date + "] " + category + ": $" + amount; } } // Classe para representar um investimento class Investment { private String name; private double amount; public Investment(String name, double amount) { this.name = name; this.amount = amount; } @Override public String toString() { return name + ": $" + amount; } } // Classe para representar um orçamento class Budget { private String category; private double amount; public Budget(String category, double amount) { this.category = category; this.amount = amount; } public String getCategory() { return category; } public double getAmount() { return amount; }
```