Sincronização de Threads em Java - Sincronizando um Bloco de Código

STATUS **DESENVOLVIDA**

Objetivo do Projeto

Este documento apresenta a comparação entre dois cenários de manipulação concorrente de saldo em uma conta bancária usando threads em Java: um sem sincronização e outro com sincronização. O objetivo é demonstrar os efeitos da concorrência sem controle adequado e os benefícios da sincronização para evitar condições de corrida.

🗒️ Cenários teste

Cenário 1: Sem Sincronização

Descrição

Os métodos addAmount e subtractAmount da classe Account não são sincronizados, permitindo que múltiplas threads modifiquem o saldo simultaneamente. Isso pode resultar em inconsistências devido à interleaving das operações.

Resultado Esperado

Devido à falta de sincronização, as threads podem acessar e modificar o saldo simultaneamente, causando condições de corrida e inconsistências no saldo final.

Conclusão

O saldo final difere do esperado (1000), demonstrando a necessidade de sincronização para evitar concorrência descontrolada.

=======

Cenário 2: Com Sincronização

Descrição

Os métodos addAmount e subtractAmount agora são sincronizados, garantindo que apenas uma thread por vez possa modificar o saldo da conta.

Resultado Esperado

Como os métodos são sincronizados, as threads executam as operações de forma ordenada, evitando condições de corrida.

Conclusão

A sincronização garante a consistência do saldo, evitando problemas de concorrência.

🔧 Conclusão:

🔗

Conclusão Geral

Sem sincronização: Possíveis condições de corrida levam a um saldo final inconsistente.

Com sincronização: O saldo é mantido corretamente, garantindo integridade dos dados.

A sincronização é essencial quando múltiplas threads acessam e modificam dados compartilhados, prevenindo resultados imprevisíveis e inconsistências. Isso é especialmente crítico em sistemas bancários, onde a precisão dos saldos é fundamental.

💻 Desenvolvedor

🔗

Marcio Fonseca
👨‍💻

=======