**Relatório Técnico – Sincronização de Threads em Java**

**Disciplina:** Sistemas Distribuídos

**Aluno:** Leonardo Timótio

## Introdução

Este relatório apresenta o desenvolvimento e análise de três atividades práticas voltadas à compreensão da **sincronização de threads** na linguagem Java. O objetivo foi explorar diferentes abordagens de controle concorrente, como o uso de **monitores**, **eventos** e **módulos especializados**, aplicados ao clássico problema do **Produtor-Consumidor**. As atividades foram realizadas em ambientes como Eclipse IDE e Visual Studio Code, com validação por meio de testes e comparação de saídas.

## Atividade 1: Sincronização Básica com Threads

### Objetivo

Compreender o funcionamento da sincronização de threads em Java, demonstrando como o controle de acesso a recursos compartilhados evita **condições de corrida** e garante a integridade dos dados.

### Desenvolvimento

Foi implementada a classe principal MeuDadoThreadsJava, acompanhada das classes auxiliares ProdutorThreads e ConsumidorThreads. A execução foi feita via terminal com os comandos:

```
bash
javac MeuDadoThreadsJava.java
java MeuDadoThreadsJava
```

A saída foi redirecionada para arquivos (arquivo1.txt, arquivo2.txt) e exibiu a alternância ordenada entre produtor e consumidor.

**Resultados**

Código:

Produtor: 0
Consumidor: 0
Produtor: 1
Consumidor: 1
...

A sincronização foi garantida com o uso de synchronized, wait() e notify(), assegurando que apenas uma thread acessasse a seção crítica por vez.

**Conclusão**

A atividade demonstrou com sucesso o uso de mecanismos de controle de concorrência, reforçando a importância da coordenação entre threads para evitar inconsistências em sistemas distribuídos.

**Atividade 2: Comparação de Implementações com Monitor**

**Objetivo**

Analisar diferentes abordagens de sincronização utilizando **monitores**, e verificar como variações na implementação afetam a ordem e consistência das operações.

**Desenvolvimento**

Foram criadas classes como ProdutorMonitor, ConsumidorMonitor, ProblemaMonitor e MedidorThreads. Os resultados foram redirecionados para arquivos e comparados com o comando:

bash
diff monitor/arquivo.txt problema/arquivo.txt

**Resultados**

A saída revelou diferenças na ordem de execução entre produtor e consumidor, com múltiplas chamadas de finalização e variações na alternância.

**Conclusão**

A atividade evidenciou como pequenas mudanças na lógica de sincronização podem impactar o comportamento das threads, destacando a importância de testes e validações em sistemas concorrentes.

**Atividade 3: Sincronização com Módulos de Eventos e Monitor**

**Objetivo**

Comparar abordagens de sincronização utilizando **módulos especializados**, como eventos e monitores, para entender o impacto da estrutura modular na execução concorrente.

**Desenvolvimento**

O projeto foi dividido em quatro módulos principais:

- **ModuloProblema**: contém a implementação original do problema do produtor-consumidor, com classes como MeuDadoThreadsJava, Consumidor e MeuDado. Serve como referência para comparar as abordagens mais estruturadas.

- **ModuloEventos**: com ProductorEventos e ConsumidorEventos, utilizando sinalização por eventos.

- **ModuloMonitor**: com ProductorMonitor e ConsumidorMonitor, utilizando monitores para controle de concorrência.

- **ModuloThread**: com variações diretas de execução, sem abstrações específicas, para observar o comportamento bruto das threads.

A saída dos programas foi registrada em arquivos (argjav.out) e comparada com:

bash
diff eventos/argjav.out problem/argjav.out

**Resultados**

A abordagem com eventos apresentou alternância mais previsível:

Código

Produtor usando Evento: 0
Consumidor usando Evento: 0
...

Enquanto o monitor apresentou variações na ordem e sinalização entre threads.

**Conclusão**

A atividade reforçou a importância da escolha da estratégia de sincronização conforme o contexto. A modularização permitiu observar com clareza os efeitos de cada abordagem, contribuindo para o domínio prático dos conceitos de concorrência.