1. Uso do `join()` para Sincronização de Threads

- **Objetivo:** Demonstrar como utilizar o método `join()` para garantir que uma *thread* espere outra terminar antes de continuar sua execução.

2. Introdução

Na programação concorrente, pode ser necessário garantir que certas operações ocorram em sequência, mesmo quando múltiplas *threads* estão envolvidas. O método `join()` permite que uma *thread* espere outra terminar antes de continuar sua execução.

Este exemplo simula o carregamento de conexões e fontes de dados, garantindo que a configuração principal só seja carregada após a finalização dessas operações.

3. Descrição do Código Original

O código contém quatro classes principais:

CarregadorConexoes.java

- Simula o carregamento de conexões da aplicação.
- Exibe mensagens indicando o início e fim do processo.
- Usa `TimeUnit.SECONDS.sleep(4)` para simular um tempo de carregamento.

FontesDeDados.java

- Simula o carregamento de fontes de dados.
- Exibe mensagens indicando o início e fim do processo.
- Usa `TimeUnit.SECONDS.sleep(6)`, simulando um tempo de carregamento maior que o do `CarregadorConexoes`.

Principal.java

- Cria e inicia as duas *threads* (`CarregadorConexoes` e `FontesDeDados`).
- Usa `thread1.join()` para garantir que o carregamento de conexões termine antes da execução do próximo trecho de código.
- Após a finalização de `thread1`, imprime uma separação (`System.out.println("==========`
- Em seguida, usa `thread2.join()` para garantir que a *thread* de fontes de dados também finalize antes de prosseguir.

Principal2.java

- Similar à classe `Principal`, mas com uma diferença:
  - Em vez de chamar `join()` diretamente, usa `thread1.join(1000)`, o que permite que a execução continue mesmo se `thread1` ainda estiver rodando após 1 segundo.

4. Modificações Realizadas

- **Tradução:** Todos os nomes de classes, métodos e variáveis foram traduzidos para o português.
- **Ajuste nos Tempos de Espera:** Inicialmente, `CarregadorConexoes` esperava 6 segundos e `FontesDeDados` 4 segundos. Isso foi invertido para testar o impacto na sincronização.
- **Teste com `join(tempo)`:** Foi incluído um teste para ver o que acontece quando usamos `join(1000)`, permitindo que a execução continue após 1 segundo, mesmo que a *thread* ainda esteja rodando.

5. Testes Realizados e Resultados Obtidos

Cenário 1: Código Original (Principal.java)

Saída esperada:
1. O carregamento de conexões inicia e dura 4 segundos.
2. O carregamento de fontes de dados inicia e dura 6 segundos.
3. A configuração principal só é carregada depois que ambas as *threads* terminam.

Saída real (exemplo):

```
Carregador de conexoes carregando: Wed Feb 07 15:00:00 BRT 2025
Carregamento de conexoes terminou: Wed Feb 07 15:00:04 BRT 2025
========================================
Fontes de dados carregando: Wed Feb 07 15:00:04 BRT 2025
Carregamento de fontes de dados terminou: Wed Feb 07 15:00:10 BRT 2025
Principal: Configuracao carregada: Wed Feb 07 15:00:10 BRT 2025
```

O comportamento está conforme esperado: `thread1` termina primeiro e `thread2` começa logo após a separação.

Cenário 2: Código com `join(1000)` (Principal2.java)

Modificação: A *thread* `FontesDeDados` é iniciada primeiro e `CarregadorConexoes` depois.
Impacto do `join(1000)`:
- A execução continua após 1 segundo, mesmo que `FontesDeDados` ainda esteja rodando.
- `CarregadorConexoes` pode ser iniciado enquanto `FontesDeDados` ainda não terminou.

Saída real (exemplo):

```
Fontes de dados carregando: Wed Feb 07 15:00:00 BRT 2025
Carregador de conexoes carregando: Wed Feb 07 15:00:01 BRT 2025
Carregamento de conexoes terminou: Wed Feb 07 15:00:05 BRT 2025
Carregamento de fontes de dados terminou: Wed Feb 07 15:00:06 BRT 2025
```

`Principal: Configuracao carregada: Wed Feb 07 15:00:06 BRT 2025`

O comportamento mostra que `join(1000)` *permitiu a continuação da execução antes do término da* thread.

6. Conclusão

Este exemplo demonstrou: - Como `join()` pode ser usado para sincronizar a execução de *threads*. - O impacto do `join(tempo)`, que permite continuar a execução sem esperar indefinidamente. - A importância de testar diferentes tempos de espera para compreender a ordem de execução das *threads*.