Conteúdo da Aula

1. **Conceitos Abordados**
   - **Threads**: Execução paralela de tarefas em ambientes multithreaded.
   - **Concorrência**: Gerenciamento de múltiplas threads trabalhando de forma simultânea.
   - **Sincronização**: Uso de técnicas para evitar condições de corrida (*race conditions*).
   - **Filas**: Estrutura para armazenamento e processamento organizado de dados em um ambiente concorrente.

---

2. **Código Estudado**
   - **Classe `Fila`**:
     - Implementa um vetor para armazenar valores adicionados por diferentes threads.
     - Garante sincronização durante operações de escrita para evitar conflitos entre threads.
   - **Classe `Escrever`**:
     - Implementa a interface `Runnable` para definir a lógica executada por uma thread.
     - Adiciona valores a uma fila em intervalos definidos.
   - **Classe `Main2`**:
     - Configura e executa múltiplas threads usando `ExecutorService` para gerenciamento.
     - Trabalha com objetos da classe `Fila` e `Escrever` para demonstrar concorrência.

Fluxo de Execução

1. **Configuração Inicial**

   - Uma `Fila` com capacidade definida é criada para armazenar os valores.
   - Duas tarefas (`pares` e `impares`) são instanciadas para adicionar números pares e ímpares à fila.

PROF

2. **Execução das Tarefas**

   - As tarefas são submetidas a um `ExecutorService`, que executa cada uma em threads separadas.
   - A execução é realizada de forma concorrente, com atrasos definidos para simular operações demoradas.

3. **Finalização**

   - O programa aguarda a conclusão das threads usando `Thread.sleep` na thread principal.
   - A fila é exibida no início (vazia) e no final (preenchida) para verificar o resultado.

Pontos-Chave

Sincronização

- O método `adicionar` da classe `Fila` utiliza `synchronized` para evitar condições de corrida, garantindo que apenas uma thread possa alterar o vetor de cada vez.

ExecutorService

- Gerencia as threads de forma eficiente, permitindo reutilização e controle sobre a execução das tarefas.

Condição de Concorrência

- Demonstrada quando múltiplas threads tentam adicionar valores à mesma estrutura de dados simultaneamente. A sincronização foi usada para resolver os conflitos.

Exemplo de Saída Esperada

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  // Fila inicial
2 adicionado.
4 adicionado.
1 adicionado.
6 adicionado.
8 adicionado.
10 adicionado.
3 adicionado.
5 adicionado.
7 adicionado.
9 adicionado.
[1, 4, 6, 3, 8, 10, 0, 5, 7, 9] // Fila final
```

Tópicos Extras

PROF

- Como ajustar o delay das tarefas para diferentes cenários.
- Impacto da sincronização no desempenho do programa.
- Alternativas ao `ExecutorService`, como criação manual de threads (`Thread`).