Relatório: Execução de Tarefas com Threads

1. Objetivo

Demonstrar o uso de threads e grupos de threads em Java para executar tarefas de forma concorrente. O código simula um processo de pesquisa, onde cada thread executa uma tarefa que gera um tempo de espera aleatório e armazena seu nome em um objeto compartilhado.

2. Estrutura do Código

2.1. Classe `Resultado`

- **Descrição:**
  - Classe simples que contém um atributo `nome` para armazenar o resultado da execução de uma thread.
- **Métodos:**
  - `getNome()`: Retorna o valor do atributo `nome`.
  - `setNome(String nome)`: Define o valor do atributo `nome`.

2.2. Classe `ProcurarTarefas`

- **Descrição:**
  - Implementa a interface `Runnable` e executa uma tarefa que simula um tempo de execução aleatório.
- **Funcionamento:**
  - Obtém o nome da thread atual e imprime uma mensagem de início.
  - Executa a tarefa simulada, que gera um valor aleatório e faz a thread dormir pelo tempo correspondente.
  - Armazena o nome da thread no objeto `Resultado`.
  - Em caso de interrupção, imprime uma mensagem de interrupção e encerra a execução.
  - Imprime uma mensagem de finalização após a conclusão da tarefa.

2.3. Classe `Principal`

- **Descrição:**
  - Orquestra a execução das tarefas utilizando um grupo de threads.
- **Funcionamento:**
  - Cria um grupo de threads denominado “Pesquisador” e um objeto `Resultado` compartilhado.
  - Inicia 5 threads que executam a tarefa de pesquisa (`ProcurarTarefas`).
  - Exibe o número de threads ativas no grupo e suas informações (nome e estado).
  - Aguarda a finalização das threads conforme um critério definido e, em seguida, interrompe todas as threads do grupo.

3. Conclusão

O código ilustra como: - Utilizar a interface `Runnable` para definir tarefas concorrentes. - Organizar threads em um grupo (`ThreadGroup`) para facilitar o gerenciamento e o monitoramento das threads. - Simular operações com tempo de execução variável por meio de pausas aleatórias. - Controlar e interromper a execução das threads de forma coordenada.

Esta abordagem é útil para aplicações que necessitam executar múltiplas tarefas de forma paralela, permitindo um melhor controle sobre o fluxo e o término dos processos em execução.