Laboratório de Programação Concorrente

Lab1

Warmup - 25.2

Objetivo

Dominar a sintaxe básica para **criar, nomear e iniciar Threads** em **Java**, observando o comportamento de concorrência simples (intercalação de execução) e aplicando as boas práticas de encapsulamento de tarefas.

Cenário Básico: O Processo de Inicialização de um Sistema

O objetivo do cenário é simular as etapas iniciais de boot de um Servidor de Aplicação Crítico que deve ser iniciado o mais rápido possível, mas precisa garantir que componentes essenciais estejam prontos antes de aceitar tráfego.

O processo de inicialização é dividido em duas grandes fases que são independentes o suficiente para serem executadas concorrentemente, mas que juntas compõem o startup completo:

1. **Inicialização de Logs:** Uma tarefa que leva um tempo fixo para configurar o sistema de *logging*.
2. **Verificação de Recursos:** Uma tarefa que executa um ciclo de verificação de recursos do sistema. Esta tarefa foca em validar que todos os componentes necessários para receber requisições estão disponíveis.

O objetivo é que ambas as tarefas sejam iniciadas concorrentemente a partir da *thread* principal.

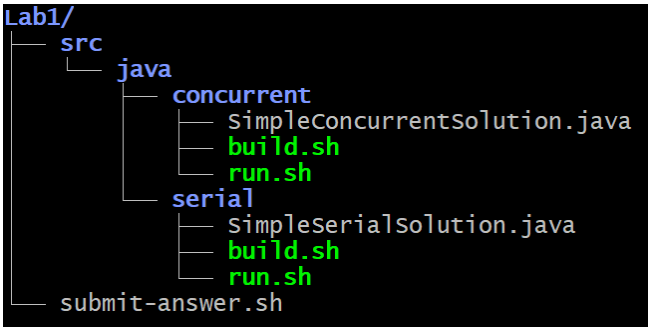
Visão geral do código base

No código base vocês encontrarão implementações serial e concorrente em Java.

A entrega, detalhada nas seções seguintes, envolverá o código fonte e análises de execução. Iremos avaliar tanto as possibilidades de plágio entre os alunos quanto a geração automática de código.

https://github.com/giovannifs/fpc/tree/master/2025.2/Lab1

O código está organizado na seguinte hierarquia:



Preparação

1. Clone o repositório do código base

   git clone [link do repositório]

2. Execute a versão serial da solução. Para isso, você deve ir até o diretório 2025.2/Lab1/src/java/serial e executar os seguintes comandos:

   bash build.sh

   bash run.sh

   Entendendo o output do script run.sh:

   - real: o tempo total decorrido
   - user: o tempo total que o processo gastou utilizando a CPU em modo usuário
   - sys: o tempo total que o processo gastou utilizando recursos do kernel

   Interpretação

   - **real: é o tempo que você veria em um cronômetro**
   - user + sys: representa o tempo efetivamente gasto pela CPU no processamento

   Se o programa usar múltiplas threads em um sistema com vários núcleos, o valor de user pode ser maior que real, já que múltiplas threads podem trabalhar simultaneamente.

Execução da solução serial

Execute algumas vezes a solução serial e verifique seu comportamento através dos logs. Houve alguma alteração na ordem de

execução das tarefas (ou sub-etapas) entre as execuções? Descreva o comportamento da ordem de execução. Além disso, através de análise do código, descreva qual seria aproximadamente o tempo de execução mínimo e máximo esperado para este código?

Crie o diretório **comments** dentro do diretório **Lab1/src,** e, dentro do novo diretório, crie o arquivo **comments1.txt** com sua análise sobre a execução da solução serial.

Desenvolvendo uma solução concorrente

O mecanismo de gerenciamento das threads tem três etapas iniciais:

1) a definição do código a ser executado pela thread;

2) a criação da thread; e, por fim,

3) a inicialização da thread.

Executando a solução concorrente

Nesta atividade, usamos o tipo Runnable para definir o fluxo que será executado por uma Thread. Em outras palavras, escrevemos o código a ser executado pela Thread através do tipo Runnable. Como temos duas tarefas diferentes a serem executadas concorrentemente,também temos duas classes definidas que implementam a interface `java.lang.Runnable`.

No código disponibilizado, implementamos a **Tarefa de Verificação de Recursos** como uma Classe Interna Não Anônima (*ResourceCheckTask*), enquanto que a tarefa de **Inicialização de Logs** foi implementada como classe anônima (*logSetupTask*).

Q1) Execute o código e analise a saída. Há algo diferente em relação à execução serial?

Ao desenvolver um programa concorrente, uma boa prática de programação é nomear as Threads criadas. Isso facilita a depuração e acompanhamento da execução da solução. Além disso, é importante sempre atribuir nomes claros e distintos às threads. Para atribuir nomes às threads, podemos usar o construtor.

```
Thread t = new Thread(Runnable, "Thread_name");
```

Altere o código disponibilizado de tal forma que tenhamos nomes significativos para as duas tarefas, e, execute novamente o código.

Q2) O que mudou em relação ao código anterior? Há diferença entre depurar a execução do código anterior e o mais atual? Se sim, qual?

Analisando especificamente o fluxo de execução da thread principal (main) através dos logs de saída, **Q3)** como ela se comporta? Quando a thread se inicia e quando é finalizada?

Avançando na análise, adicione o trecho de código abaixo imediatamente antes do último print da thread principal.

```
     try {
           tLogs.join();
           tCheck.join();
        } catch (InterruptedException e) {
               System.err.println("[" + currentThread.getName() + "] interrompida.");
           Thread.currentThread().interrupt();
           return;
     }
```

Execute o novo código algumas vezes e observe sua saída.
Q4) O que aconteceu após a inserção do código? O fluxo de execução foi alterado? Se sim, o que mudou?

No diretório **comments,** crie o arquivo **comments2.txt** com a sua análise sobre a execução da solução concorrente. Neste arquivo, responda comente sobre os questionamentos definidos em Q1, Q2, Q3 e Q4.

Evoluindo a solução concorrente

Por fim, a **ResourceCheckTask** executa 5 verificações em uma única thread. Para aproveitar melhor os múltiplos núcleos de processamento e paralelizar o trabalho interno, vamos criar uma nova versão onde cada uma das 5 verificações é executada por sua própria thread. Dentro do diretório concurrent, crie uma nova classe chamada **SimpleConcurrentSolutionV2** a partir da classe **SimpleConcurrentSolution** e implemente uma solução concorrente que

atenda este requisito. Algumas pontos importantes que você precisa pensar sobre:

- O que precisa ser alterado na classe **ResourceCheckTask** para que a tarefa verifique apenas um recurso?
- Como podemos informar a uma instância da classe **ResourceCheckTask** qual o id do recurso que ela deve processar?
- Quantas threads da classe **ResourceCheckTask** devemos iniciar?

Prazo

21/10/2025 às 16h00

Entrega

Você deve criar e manter um repositório privado no GitHub com a sua solução. No entanto, a entrega do laboratório deverá ser realizada por meio de submissão online utilizando o script submit-answer.sh, disponibilizado na estrutura de arquivos do próprio laboratório. Uma vez que você tenha concluído sua resposta, seguem as instruções:

1) Crie um arquivo lab1_matr1_matr2.tar.gz somente com o "src" do repositório que vocês trabalharam. Para isso, supondo que o diretório raiz de seu repositório privado chama-se lab1_pc, você deve executar:

   ```
   tar -cvzf lab1_matr1_matr2.tar.gz lab1_pc/src
   ```

2) Submeta o arquivo lab1_matr1_matr2.tar.gz usando o script submit-answer.sh, disponibilizado no mesmo repositório do laboratório:

   ```
   bash submit-answer.sh lab1 path/lab1_matr1_matr2.tar.gz
   ```

Lembre-se que você deve manter o seu repositório privado no GitHub para fins de comprovação em caso de problema no empacotamento ou transmissão online. Alterações no código realizadas após o prazo de entrega não serão analisadas.