Atividade Prática - Threads em Java (2)

Em um futuro não muito distante, o Governo Federal decidiu automatizar o julgamento de processos de pequenas causas. Agora, cada Tribunal possuirá um servidor que recebe toda a documentação do processo pela rede e usa algorítimos sofisticados para emitir um veredito. A empresa licitada para implementar o novo sistema de julgamento entregou o código no final dessa página. No entanto, ao testar o sistema, técnicos observaram que ele produz julgamentos corretos, mas é pouco eficiente. Como o governo está sem recursos para um aditivo contratual, o Ministério da Justiça delegou a tarefa ao seu melhor estagiário: você. O ministério tem pressa, e já marcou uma coletiva de imprensa na segunda-feira para apresentar o sistema. Você tem até domingo para fazer os ajustes necessários no código, ou sua bolsa estará em risco.

Regras

Os computadores dos tribunais só podem executar um certo número de julgamentos em paralelo (`nJuizes`). Para que os processos sejam julgados rapidamente, há um limite de quantos processos podem estar aguardando julgamento em um Tribunal (`tamFila`). Quando um processo é recebido pelo tribunal, mas o tribunal excedeu o tamanho da fila permitida, uma exceção **`TribunalSobrecarregadoException`**deve ser lançada, para que quem criou o processo tente novamente mais tarde.

O construtor da classe **`Tribunal`** recebe dois argumentos: `nJuizes`, indicando quantos processos podem ser julgados em paralelo e `tamFila`, indicando o tamanho máximo de processos aguardando para serem julgados. Essa classe também possui um método protegido que realiza o julgamento de um processo (**`checkGuilty`**, retorna true se o réu for culpado e false caso contrário) e um método público que recebe processos e e retorna vereditos (**`julgar`**).

```
public class Tribunal implements AutoCloseable {
    public Tribunal(int nJuizes, int tamFila) {
        this.executor = Executors.newSingleThreadExecutor();
    }
    public boolean julgar(Processo p) throws TribunalSobrecarregadoException {
        return checkGuilty(processo);
    }
    protected boolean checkGuilty(Processo p);
    public void close() throws Exception;
}
```

Re-implemente o método julgar de modo que:

1. Ocorram até nJuizes julgamentos em paralelo
2. A exceção TribunalSobrecarregadoException seja lançada quando houverem `nJuizes` processos sendo julgados e `tamFila` processos já enfileirados.

Você deve usar um único executor **no atributo `executor`**. Os testes checam se o executor é desligado após o `close()`.

Dica

Use uma instância de `ThreadPoolExecutorService` em conjunto com um `ArrayListBlockingQueue`.
O `ThreadPoolExecutorService` permite definir um número máximo de threads ativas.
Um `ArrayListBlockingQueue` funciona como um *buffer* circular, e por isso possui uma capacidade máxima. Os construtores de `ThreadPoolExecutorService` recebem uma fila como argumento.
O `ThreadPoolExecutorService` oferece algumas opções sobre o que fazer quando a capacidade do *buffer* é excedida. Leia a documentação dos construtores de `ThreadPoolExecutorService` e o "*Nested Class Summary*".

Correção automática

O script de correção (**grade-threads2.sh**) está dentro do esqueleto e deve ser executado na própria pasta onde está. Você deve implementar as classes já criadas. Não crie suas soluções em novas classes ou o script corretor não as irá encontrar. Os testes estão visíveis na classe `TribunalTest`. O arquivo de testes será substituído durante a avaliação.

Caso queira usar features do Java 8+ altere o arquivo `pom.xml`.

Entrega do Exercício

Submeta um arquivo **.tar.gz (ou zip)** na **mesma estrutura do esqueleto**dado como ponto inicial nessa tarefa. O uso do esqueleto fornecido é **obrigatório**. Utilize o comando **make submission** para garantir que será gerado um arquivo conforme solicitado.

O prazo para entrega é **28 de Outubro às 23h55**.

(Atualizado em 27/10/2018 - 13:08)

- `grade-threads2.sh` mostra a saída do `mvn verify`


atividade_9.tar.gz

grade-threads2.sh