



## Lista 11 - Threads

### 1) Produtor-Consumidor

- a) Altere o programa visto em aula que exemplifica o modelo produtor consumidor para que ele funcione com a classe Stack (pilha) do Java (pesquise como usá-la).

Usando-se uma pilha, a condição de wait tanto para o produtor quanto para o consumidor deverá ser alterada; o produtor não poderá empilhar mais do que 10 itens, e o consumidor deverá esperar caso a pilha esteja vazia, as variáveis disponibilidade e conteúdo não serão mais necessárias.

- b) Agora faça com que seu programa utilize múltiplos consumidores, os quais serão notificados pelo comando *notifyAll*. Faça com que seu produtor produza um número finito de itens e após todos eles serem consumidos, indique o consumidor que mais itens consumiu.

### 2) Threads sob demanda

- a) Escreva um programa que leia a entrada do teclado continuamente, como no exemplo feito em aula:

- caso o usuário digite “exit”, o programa se encerra com `System.exit(0)`;
- caso o usuário digite “new”, uma nova thread é criada.

As threads criadas deverão criar uma janela com um label, segundo o código:

```
JFrame jfTela = new JFrame();  
jfTela.setSize(new Dimension(200, 200));  
jfTela.setLayout(new BorderLayout());  
jfTela.setVisible(true);  
JLabel jlTexto = new JLabel();  
jfTela.add(jlTexto);
```

As threads deverão escrever uma mensagem no label a cada 1000 ms, por tempo indefinido (`while(true)`). Pense em alguma mensagem que se altere ao longo do tempo – um contador por exemplo.

- b) Faça com que as threads criadas fiquem armazenadas em um array. Quando o usuário digitar:

- “reset”, todas as threads criadas deverão retornar ao estado inicial de suas mensagens;
- “bold”, todas as threads deverão alternar o negrito de seu label com o comando `jlTexto.setFont(jlTexto.getFont().deriveFont(jlTexto.getFont().getStyle() ^ Font.BOLD))`;

- “close”, todas as threads serão fechadas.

### 3) Concorrência

- a) Escreva um programa que simule uma conta bancária de um dado cliente. O sistema terá dois métodos:

- `void deposita(int iValor)`: acrescenta valor ao saldo;
- `int saca()`: decrementa valor do saldo.

Escreva duas threads (chamadas `userThread` e `bankThread`), cada uma irá fazer 5.000 depósitos e 5.000 saques na mesma conta (concorrência). Em cada operação de saque, ou de depósito, os métodos irão seguir a sequência:

1. Lê saldo
2. Calcula novo saldo em uma variável temporária
3. Imprime saldo temporário na tela
4. Atribui ao saldo da conta o valor armazenado na variável temporária

Demonstre o problema de concorrência de dados com este programa. Use valores de saldo, depósito, e saque que possam ser verificados; e, se necessário, varie a quantidade de operações.

- b) resolva o problema usando o modificador `synchronized`, e verifique que o problema de concorrência desapareceu

**4)** Imagine que se tem uma determinada carga de processamento composta por  $10^7$  de cálculos de raiz quadrada de números de ponto flutuante com 64 bits (double).

Para este exercício, vamos simular a carga de processamento sorteando-se números aleatórios positivos que serão usados para se calcular `Math.sqrt()`;

➔ Atenção: não usar `Math.random()` ou qualquer outro método sincronizado; do contrário, não se terá paralelismo, apenas overhead de threads. Use a classe `java.util.Random`.

**a)** Calcule a raiz quadrada de  $10^7$  de números sem usar threads – meça o tempo total;

**b)** Identifique quantos núcleos de processamento tem seu computador (`int nCores = Runtime.getRuntime().availableProcessors();`); crie `nCores` threads e faça com que cada thread realize  $10^7/nCores$  cálculos de raiz quadrada – meça o tempo total.

**c)** Faça com que o número de cálculos seja um parâmetro a ser passado para o seu programa; em seguida realize os passos a) e b) para  $10^7$ ,  $10^8$ ,  $10^9$ ,  $10^{10}$ , e  $10^{11}$ , (e mais, se necessário, até ter tempo da ordem de minutos).

Usando um software de planilha de cálculo, plot um gráfico  $\log(\text{número\_de\_cálculos}) \times \log(\text{tempo\_total})$ ; comente a curva descrita no gráfico apontando de quanto foi o ganho (ou a perda) de desempenho com o uso de threads.

**d)** Realize os passos a), b), e c) para uma operação mais simples do que a raiz quadrada, faça apenas uma contagem (+1), por exemplo.

**e)** Realize os passos a), b), e c) para um conjunto de operações mais custoso do que uma raiz quadrada, por exemplo uma raiz quadrada, mais uma potência (`Math.pow`), e mais um cálculo de cosseno (`Math.cos`).