CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS VII - UNIDADE TIMÓTEO - Engenharia da Computação

Laboratório de Programação Concorrente
Prof. Lucas Pantuza Amorim

Uso de *threads*, Sincronização por exclusão mútua e condicional em Java

O objetivo deste Laboratório é aprender como criar programas concorrentes em Java e praticar o uso de variáveis de condição implementando problemas clássicos de concorrência. Para cada atividade, siga o roteiro proposto e responda às questões colocadas.

Atividade 1

Objetivo: Mostrar como criar um programa concorrente em Java. Em Java, a classe `java.lang.Thread` oferece métodos para criar *threads*, iniciá-las, suspendê-las e esperar pelo seu término.

O primeiro passo para criar uma aplicação concorrente em Java é criar uma classe que implementa a interface `Runnable`. Essa interface define o método `run()`, responsável pelo código que deverá ser executado pela *thread*.

O segundo passo é transformar o objeto `Runnable` em uma *thread*. Para isso, chame o construtor da classe `java.lang.Thread` com o objeto `Runnable` como argumento.

O terceiro passo é iniciar as *threads* criadas, usando o método `start()` da classe `Thread`.

Roteiro:

1. Abra o arquivo `HelloThread.java`. Leia o programa e tente entender o que ele faz.
2. Compile o programa fazendo `javac HelloThread.java` no terminal.
3. Execute o programa várias vezes (fazendo `java HelloThread`) e observe os resultados impressos na tela. Há mudanças na ordem de execução das *threads*? Por que isso ocorre?
4. Descomente as linhas 42-46 e compile o programa novamente.
5. Execute o programa várias vezes e observe os resultados impressos na tela. Qual alteração na execução da aplicação pode ser observada e por que ela ocorre?

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 2

Objetivo: Mostrar outra forma de criar *threads* em Java.

Roteiro:

1. Outra forma de criar programas concorrentes em Java é estendendo a classe `Thread`. Abra o arquivo `OlaThread.java`.
2. Primeiro, encontre as principais diferenças em relação ao programa `HelloThread.java`.
3. Compile e execute o programa várias vezes, e observe os resultados impressos. Há mudanças na ordem de execução das *threads*? Por que isso ocorre?

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 3

Objetivo: Mostrar um exemplo de aplicação com *threads* e memória compartilhada em Java.

Roteiro:

1. Abra o arquivo `TIncrementoBase.java`. Leia o programa para entender o que ele faz. Qual é a seção crítica do código? Qual saída é esperada para o programa (valor final de `s`)?
2. Compile o programa, execute-o várias vezes e observe os resultados impressos na tela. Os valores impressos foram sempre o valor esperado? Por que?

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 4

Objetivo: Mostrar como implementar exclusão mútua em Java.

Roteiro:

1. Ainda no arquivo `TIncrementoBase.java`. Comente as linhas 15-17 e 28-30; e descomente as linhas 19-23 e 32-36.
2. Acompanhe a explanação sobre ouso de `synchronized` em Java.
3. Compile o programa, execute-o várias vezes e observe os resultados impressos na tela. Os valores impressos foram sempre o valor esperado? Por que?

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 5

Objetivo: Implementar um programa concorrente, com M *threads* (além da *thread* principal), para incrementar de 10 cada elemento de um vetor de $N(10 < N < 100)$ elementos.

Roteiro:

1. Projete uma classe em Java para conter o vetor e os métodos de acesso a ele (construtor, incremento de uma posição, impressão, tamanho do vetor, etc.).
2. Adote a estratégia de cada *thread* incrementar posições alternadas do vetor.
3. Qual(is) argumento(s) deverá(ão) ser passado(s) para cada *thread*?
4. Na *thread* `main` crie uma instância da classe vetor, imprima seus valores iniciais, crie e dispare as *threads*, aguarde todas as *threads* terminarem e imprima os valores finais do vetor.
5. Teste seu programa.

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 6

Objetivo: Reapresenta o problema do produtor/consumidor.

Roteiro:

1. Abra o arquivo `PC.java`. Complete a implementação dos métodos `Insere` e `Remove` da classe `Buffer`.
2. Inclua código adicional para geração de *log* da execução de modo que seja possível verificar a sua corretude.
3. Execute o programa várias vezes e verifique se a execução está correta.
4. Varie o número de *threads* consumidoras e produtoras, fazendo:
   - um produtor e um consumidor;
   - um produtor e vários consumidores;
   - vários produtores e um consumidor;
   - vários produtores e vários consumidores.
5. Verifique se a execução do programa está sempre correta.
6. Em todas as execuções, aponte quem foi o "produtor vencedor" (o que conseguiu inserir o maior número de elementos no *buffer*) e o "consumidor vencedor" (o que conseguiu retirar o maior número de elementos do *buffer*).

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 7

Objetivo: Propõe uma variação na implementação do problema produtor/consumidor.

Roteiro: Implemente a seguinte variação do problema produtor/consumidor: a cada execução de um consumidor, ele consome o *buffer* inteiro, e não apenas um único item (para isso ele deve esperar o *buffer* ficar completamente cheio). O produtor continua com a mesma lógica, i.e., insere um item de cada vez. Varie o número de *threads* consumidoras e produtoras, fazendo:

- um produtor e um consumidor;
- um produtor e vários consumidores;
- vários produtores e um consumidor;
- vários produtores e vários consumidores.

Inclua código adicional para geração de log da execução de modo que seja possível verificar a sua corretude.

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.