Descrição do Exercício de Threads e Concorrência

Objetivo:
1 – Aprender as duas maneiras de se criar *threads e* testar os mecanismos que Java disponibiliza para obter sincronização e cooperação entre *threads* de baixo nível.
2 – Aprender como usar as 3 principais configurações da classe ExecutorServcie.

Pacotes:
br.com.qualiti.java.avancado.modulo01.parte1 – Deve conter as implementações da parte 1 do exercício.
br.com.qualiti.java.avancado.modulo01.parte2 – Deve conter as implementações da parte 2 do exercício.
br.com.qualiti.java.avancado.modulo01.parte3 – Deve conter as implementações da parte 3 do exercício.

Passos para a execução do exercício

Parte 1 – Criação e execução de Threads

1. Criar uma classe, `MeuThread`, estendendo `Thread` que, quando em execução, imprima os números inteiros de 0 a 50, em sequência. Esta classe deve receber, em seu construtor, um `String` correspondendo ao nome do `Thread` e este deve ser guardado em um atributo da classe. Cada número impresso deve ser precedido pelo nome do `Thread`.

2. Criar um método `main()` nessa classe que cria duas instâncias de `MeuThread` e as inicia.

3. Rodar o método `main()` de `MeuThread`.

4. Criar uma classe, `MeuRunnable`, implementando `Runnable` e que, quando em execução, imprima os números inteiros de 0 a 50, em sequência. Esta classe deve receber, em seu construtor, um `String` correspondendo ao nome do `Runnable` e este deve ser guardado em um atributo da classe. Cada número impresso deve ser precedido pelo nome do `Runnable`.

5. Criar, nessa classe, um método `execute` responsável por iniciar um novo `Thread` a partir de `MeuRunnable`.

6. Criar, em `MeuRunnable`, um método `main()` que cria duas instâncias dela mesma e as inicia.

7. Rodar o método `main()` de `MeuRunnable`.

8. No método `run()` nas classes `MeuThread` e `MinhaExecucao`, inserir código para, uma vez que um número seja impresso, o `Thread` atual esperar pelo menos 10 milissegundos antes de imprimir o próximo.

Parte 2 – Sincronização e Cooperação entre *Threads*

9. Olhar o código das classes `Produtor` e `Consumidor`, fornecidos no arquivo JA_MOD01_STARTUP. jar

10. Rodar o método main() da classe `TesteProdutorConsumidor`. O resultado é o esperado? Uma mensagem só é escrita por um `Produtor` depois da anterior ter sido lida por um `Consumidor`? Um `Consumidor` só tenta ler uma mensagem quando alguma foi escrita por um `Produtor`?

11. Na classe `CaixaCorreio`, modificar os métodos `get()` e `put()` para que apenas um `Thread` possa estar executando cada um deles, num determinado momento.

12. O passo anterior não é suficiente para garantir o funcionamento correto do programa. É necessário garantir que o método `get()` lê uma mensagem apenas quando há alguma para ser lida e que o método `put()` coloca uma nova mensagem no *buffer* apenas quando este está vazio. Inserir, nos métodos `get()` e `put()`, o código necessário para garantir que as duas restrições acima seja obedecidas.
    a. Inserir, no métodos `get()` e `put()`, código para fazer com que *threads* tentando ler o *buffer* **esperem** enquanto este estiver vazio e sejam **notificados** quando um novo elemento for escrito.
    b. Inserir, no métodos `get()` e `put()`, código para fazer com que *threads* tentando escrever no *buffer* **esperem** enquanto este estiver cheio e sejam **notificados** quando for esvaziado (lido).

13. Rodar o método `main()` da classe `TesteProdutorConsumidor`.

Parte 3 – Observar o comportamento das 3 configurações de um ExecutorService

1. Executar as classes `TestCachedThreadPool`, `TestFixedThreadPool` e `TestSingleThreadPool`, fornecidos no arquivo JA_MOD01_STARTUP.jar.

2. Descrever o comportamento observado.