


UNIP – Universidade Paulista		 UNIVERSIDADE PAULISTA
Disciplina...:	LP00 – Ling. Programação Orientada a Objetos	
Professor...:	Leandro Carlos Fernandes	

:: Lista de Exercícios – Java (Threads) ::

- 1) Faça um programa que crie 10 threads e que, cada uma delas, escreva na console uma mensagem contendo seu nome.
- 2) Defina a classe Contador como uma implementação da interface Runnable, que imprime números de 0 a 1000. Crie um programa chamado TesteContador, cujo método main deve criar e iniciar a execução de uma instância de Contador. Execute a programa e observe se o resultado produzido era o esperado.
- 3) Modifique o método main da classe TesteContador para criar dois ou mais threads Contador e execute novamente o programa. Observe o resultado produzido, comparando-o com o obtido no exercício anterior.
- 4) Suponha as seguintes classes Java:

```

public class InterruptExemplo {
    public static void main(String[] args) {
        Thread t = new Thread(new Xpto());
        t.start();
        t.interrupt();
    }
}

class Xpto implements Runnable {
    public void run() {
        try {
            Thread.currentThread().join(0);
            System.out.println(10);
        }
        catch (SecurityException e) {
            System.out.println(20);
        }
        catch (IllegalMonitorStateException e) {
            System.out.println(30);
        }
        catch (IllegalArgumentException e) {
            System.out.println(40);
        }
        catch (Exception e) {
            System.out.println(50);
        }
    }
}

```

O que será exibido no console após a execução do comando `t.interrupt()`?

- a) 10 b) 20 c) 30 d) 40 e) 50
- 5) Escreva um programa que realize o cálculo das somas dos valores das linhas de uma matriz qualquer de números inteiros e imprima o resultado na tela. Faça com que o cálculo do somatório de cada linha seja realizado em paralelo por thread.
 - 6) Crie uma classe T1 do tipo Thread com um método construtor que receba um número de identificação da Thread e um método run que fique em loop eterno imprimindo na tela a frase "Thread x executando", onde x é o número de identificação da Thread. Faça um programa que crie um vetor de 5 Threads T1 e, com um laço FOR, instancie e dispare todas as Threads.

- 7) Crie duas threads tal que: uma fica enviando notícias (textos quaisquer) a cada 5 segundos, enquanto a outra fica enviando o horário atual a cada 10 segundos. O horário deverá ser informado ao menos cinco vezes antes que o programa se encerre.
- 8) Faça um programa que cria 3 threads responsáveis, respectivamente, por escrever "A", "B" e "C" na tela. Garanta que durante a execução seja sempre escrito "ABC" na tela.
- 9) Desenvolva um programa em que duas threads escrevam números aleatórios em um vetor global de inteiros e de tamanho 100. Lembre-se que neste caso essas threads devem garantir exclusividade durante a operação para obter o valor que indica a posição a ser preenchida no vetor. Este índice deve começar com o valor -1 e cada thread deve incrementá-lo antes de usá-lo. Quando o vetor estiver preenchido a thread original deve ser acordada (não se esqueça de usar uma variável de condição).
- 10) Escreva uma classe que permita paralelizar uma pesquisa em um array de inteiros. Isso deve ser feito com o seguinte método: `public static int parallelSearch(int x, int[] A, int numThreads)`. Este método cria tantas threads quanto especificadas em numThreads, divide o array A em muitas partes e dá a cada thread parte do array para procurar sequencialmente pelo valor x. Se uma thread encontrar o valor x, então é retornado o índice i ($A[i]=x$), ao contrário -1.
- 11) Cinco lebres disputarão uma corrida. Cada lebre pode dar um salto que varia de 1 a 3 metros de distância. A distância percorrida é de 20 metros. Na corrida, cada lebre dará um salto, informar quantos metros ela pulou a cada salto realizado. Em seguida, a lebre pára para descansar (método `yield()`). Escreva um programa, utilizando threads (uma para cada lebre), que informe a lebre vencedora e a colocação de cada uma delas no final da corrida. Informar também quantos pulos cada uma delas deu.

"I dare you!"

- 12) Defina uma classe *Mailbox* que tem um atributo *message* do tipo *String*. A classe *Mailbox* deve ter dois métodos: *storeMessage* e *retrieveMessage*. O método *storeMessage* recebe um *String* e, se o mesmo tiver sido consumido (*message == null*), armazena no atributo *message* do *Mailbox*. Caso contrário, quem chamou o método *storeMessage* deve esperar até que alguém consuma a mensagem (chamando o método *retrieveMessage*). De forma similar, o método *retrieveMessage* retorna o valor da mensagem, caso ela tenha sido produzida/armazenada (*message != null*). Caso contrário quem chamou o método deve esperar até que alguém produza uma mensagem (chamando o método *storeMessage*).
 - > Crie a classe *Producer*, que é um thread e deve ter um atributo do tipo *Mailbox* e no seu método *run* deve ser definido um loop que executa o método *storeMessage* do *Mailbox*, armazenando mensagens no *Mailbox*.
 - > Defina uma classe *Consumer*, que também é um thread, e que deve consumir mensagens (chamando o método *retrieveMessage*) escritas em no seu atributo do tipo *Mailbox*.
 - > Crie uma classe de teste com um método *main* que cria um *Producer* e um *Consumer* que compartilham o mesmo *Mailbox* e iniciam a execução.
 - > Altere a classe de teste para criar mais de um produtor e/ou consumidor para o mesmo *Mailbox*.
- Dicas:
- > Utilize os métodos *wait* e *notifyAll* para implementar os métodos da classe *Mailbox*.
 - > Na última parte do exercício, onde é pedido para alterar a classe de teste, você deve adicionar um atributo de identificação (*String*) nas classes *Producer* e *Consumer*, de modo a permitir que sejam identificados tanto que está consumindo uma mensagem, quanto que esta produzindo a mesma. Para tal concatene o identificador dos objetos a mensagem a ser enviada (no caso do produtor) ou a mensagem a ser impressa (no caso do consumidor).