

INE 5410 - Laboratório

AULA 10 – JAVA: SINCRONIZANDO THREADS

1. INTRODUÇÃO

Em qualquer linguagem onde a concorrência é suportada, enquanto se tem a grande vantagem de se poder acelerar o processo de execução das diversas tarefas e de se associar um diferente grau de importância a cada tarefa de um programa (o conceito de prioridades, visto na aula anterior), deve-se tomar o cuidado com a sincronização das tarefas, uma vez que estas devem concorrer pelos recursos compartilhados, cujo acesso deve ser gerenciado com eficácia.

2. OBJETIVO DA AULA

O objetivo desta aula de laboratório experimental, em diversos casos, o uso dos mecanismos de controle de concorrência associados às threads da linguagem Java. Através de diversos exemplos, será possível observar a maneira como se pode gerenciar o acesso a estes recursos.

3. IMPLEMENTANDO MONITORES

Conforme descrito na parte teórica, o uso do atributo `synchronized` associado aos métodos de uma classe, assim como o uso dos métodos `wait()`, `notify()` e `notifyAll` permitem, de uma maneira relativamente simples, transformar uma classe num monitor no estilo do proposto por Hoare.

3.1 EXEMPLO 1 – CONTROLE DE ACESSO AO ESTACIONAMENTO

O exemplo a seguir ilustra o uso destes mecanismos para gerenciar a concorrência de diversos processos no acesso a recursos de uma dada aplicação.

O exemplo, já conhecido, é aquele do controle de um estacionamento, onde deve-se estabelecer prioridade de acesso às vagas obedecendo à ordem professor – funcionário - aluno. A solução encontrada foi a definição da classe **Parking**, que, como se pode observar, implementa um monitor que permite o acesso concorrente a dois métodos denominados `in` e `out`, para dar acesso ao uso de uma vaga no estacionamento e liberar uma vaga respectivamente.

Estes métodos são acessados pelas instâncias da classe **Pessoa**, que foi definida como subclasse de `Thread` para constituir os elementos ativos que irão concorrer ao acesso às vagas do estacionamento. A classe `Pessoa` permite representar todos os elementos concorrentes destas vagas, sendo que o atributo classe é o que define a função da pessoa na instituição: professor, funcionário ou aluno.

Execute o programa e observe o resultado da saída do programa.

```

public class Parking {
    private int vaga, nprof, nfunc, naluno;

    public Parking ( int vaga ){
        this.vaga = vaga;
        nprof      = 0;
        nfunc       = 0;
        naluno      = 0;
    }
    public synchronized void in( String classe, int pid ){
        try{
            if ( vaga == 0 ) {
                System.out.println( classe + "  NUM: " + pid + " ###  FILA " );
                if ( classe == "PROFESSOR"){
                    nprof++;
                    while ( vaga == 0 ) wait();
                    nprof--;
                }
            }
            else {
                if ( classe == "FUNCIONARIO"){
                    nfunc++;
                    while ( ( vaga == 0 ) || ( nprof > 0 ) ) wait();
                    nfunc--;
                }
            }
            else {
                naluno++;
                while ( ( vaga == 0 ) || ( nprof > 0 ) || ( nfunc > 0 ) ) wait();
                naluno--;
            }
        }
        }
        vaga--;
        System.out.println( classe + "  NUM: " + pid + ">>>  ENTRANDO " );
    } catch (Exception e ){}
}

public synchronized void out ( String classe, int pid ){
    System.out.println( classe + "  NUM: " + pid + "<<<  SAINDO " );
    vaga++;
    notifyAll();
}
}

public class Pessoa extends Thread{
    String classe;
    int pid;
    Parking parking;
    public Pessoa ( String classe , int pid, Parking parking ){
        this.classe = classe;
        this.pid     = pid;
        this.parking = parking;
    }
    public void run(){
        try{
            sleep( ( int )(Math.random()*9000 ) );
            parking.in( classe , pid);
            sleep( ( int )(Math.random()*7000 ) );
            parking.out( classe , pid);
        }catch(Exception e ){}
    }
}

```

```

public class Main {
    public static void main ( String [] s ){
        Parking parking = new Parking ( 10 );
        Pessoa [] prof = new Pessoa [ 10 ];
        Pessoa [] func = new Pessoa [ 15 ];
        Pessoa [] aluno = new Pessoa [ 30 ];
        for ( int i = 0 ; i < 10 ; i ++ ){
            prof[i] = new Pessoa("PROFESSOR", i, parking );
            prof[i].start();
        }
        for ( int i = 0 ; i < 15 ; i ++ ){
            func[i] = new Pessoa("FUNCIONARIO", i, parking );
            func[i].start();
        }
        for ( int i = 0 ; i < 30 ; i ++ ){
            aluno[i] = new Pessoa("ALUNO", i, parking );
            aluno[i].start();
        }
    }
}

```

3.2 EXEMPLO 2 – FUMANTES

Três fumantes inveterados, f1, f2 e f3 estão sentados em uma mesa com um agente. Todos querem fumar desesperadamente, porém, são necessários 3 ingredientes para que um fumante faça seu cigarro e possa fumar: fumo, palha, fósforo. Cada fumante possui apenas dois ingredientes, por exemplo, o f1 possui fumo e fósforo, o f2 fumo e palha e o f3 palha e fósforo. O agente possui os 3 ingredientes e fornece sempre um deles escolhido aleatoriamente. Para facilitar a implementação, suponha que o agente joga sobre a mesa uma pedra numerada: se o ingrediente sorteado for fumo ele joga a pedra 3, pois é o fumante 3 que poderá fumar, se for fósforo joga a pedra 2 e se for palha a pedra 1. O agente joga a pedra e para, a espera do fumante contemplado fazer o seu cigarro e fumar. Sempre que uma pedra é jogada sobre a mesa os 3 fumantes tentam consultar. Se a pedra for igual ao seu número ele recebe do agente o ingrediente, faz seu cigarrinho, fuma e em seguida acorda o agente para uma nova rodada.

```

public class Mesa {
    private int pedra;
    public Mesa (){
        pedra = 0;
    }
    public synchronized int consulta( int fid ){
        int aux;
        aux = pedra;
        if ( pedra == fid ) pedra = 0;
        return aux;
    }
    public synchronized void atualiza ( int novapedra ){
        try{
            pedra = novapedra;
            wait();
        } catch( Exception e ){}
    }
    public synchronized void wakeUpAg( int fid ){
        System.out.println( " FUMANTE: "+ fid+ " ACORDANDO AGENTE " );
        notify();
    }
}

```

```

public class Fumante extends Thread {

    int pedra, fid;
    Mesa mesa;
    public Fumante ( int fid, Mesa m ){
        this.fid = fid;
        mesa      = m;
    }
    public void run(){
        while ( true ){
            try{
                pedra = mesa.consulta ( fid );
                while ( pedra != fid ){
                    yield(); // libera a CPU
                    pedra = mesa.consulta ( fid );
                }
                System.out.println( " FUMANTE: + fid+ " FUMANNNNDO " );
                sleep ( ( int ) (Math.random()* 1000 ) );
                mesa.wakeUpAg(fid);
            }catch ( Exception e ){System.out.println(e);};
        }
    }
}

public class Agente extends Thread{
    int pedra;
    Mesa mesa;
    public Agente ( Mesa m ){
        mesa = m;
    }
    public void run(){
        while ( true ){
            try{
                pedra = ( int ) ( 1 + Math.random()*3 );
                System.out.println( " PEDRA SORTEADA: "+ pedra );
                sleep ( ( int )( Math.random() * 1000 ) );
            } catch (Exception e) {System.out.println(e);};
            mesa.atualiza(pedra );
        }
    }
}

public class Main {

    public static void main(String[] args) {
        Fumante f1, f2, f3;
        Agente ag;
        Mesa mesa = new Mesa ();
        f1 = new Fumante ( 1, mesa );
        f2 = new Fumante ( 2, mesa );
        f3 = new Fumante ( 3, mesa );
        ag = new Agente ( mesa );

        f1.start();
        f2.start();
        f3.start();
        ag.start();

    }
}

```

4. EXERCÍCIOS

4.1. POMBOS CORREIO

Considere o problema em que diversos pombos correio transportam cartas de uma cidade A para uma cidade B. As pessoas da cidade A depositam continuamente cartas na Caixa Postal. A cada vez que uma pessoa deposita uma carta, verifica se gerou um número de cartas que seja múltiplo de 3. Em caso positivo, acorda um pombo para que ele leve 3 cartas para a cidade B. Chegando na cidade B, o pombo entra numa gaiola e lá fica retido. Cada pombo que entrar na gaiola, verifica se a quantidade de pombos corresponde a um múltiplo de 5... se isto for confirmado, este libera os demais pombos para que retornem à cidade A.

4.2. JANTAR DOS FILÓSOFOS

O problema resume-se em garantir que os cinco filósofos sentados em volta da mesa possam pensar e, quando estiverem com fome, alimentarem-se com o auxílio de dois garfos posicionados dos dois lados de seu prato. Um filósofo, ao pegar um dos garfos, inviabiliza a refeição de um dos filósofos ao seu lado, de modo que, quando um filósofo está comendo, os outros dois imediatamente ao seu lado poderão alimentar-se. Pode-se ter mais de um filósofo comendo ao mesmo tempo, desde que estes não estejam lado a lado. Isto que significa que apenas dois filósofos poderão estar comendo simultaneamente. Além disso, deve-se tomar cuidado para que vários filósofos não peguem o mesmo garfo ao mesmo tempo (todos pegam o garfo à sua direita ou à sua esquerda). Se isto ocorrer, o resultado é bloqueio mortal.

4.3. CONTA BANCÁRIA

Um pai possui uma conta corrente num banco, da qual 10 filhos compartilham. Cada um dos filhos vai, à medida que tem necessidade, realizando saques de valor aleatório sobre esta conta. Quando não existir saldo suficiente para o saque, o filho acorda o pai que, então, realiza o depósito de um valor aleatório nesta conta.

Sugestão:

Para os saques, considere as seguintes operações:

```
saque := random(saldo-1) + 1;  
saldo := saldo - saque;
```

Para o depósito, considere:

```
saldo := random(150) + 20;.
```