

EXPLICAÇÃO SOBRE O LABORATÓRIO 8

O CÓDIGO:

```
private int leit, escr, aux;

// Construtor
LE() {
    this.leit = 0;
    this.escr = 0;
    this.aux = 0; //variável do LABORATÓRIO 8
}
```

A variável *aux* é a variável principal do laboratório, a que será modificada pelas threads. Declarei ela no monitor *LE*, visto que várias threads irão acessá-la, usando o *synchronized*.

```
// Entrada para leitores
public synchronized void EntraLeitor (int id, int identificadorUnico) {
    try {
        while (this.escr > 0) {
            if(identificadorUnico == 102) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SE
                System.out.println ("le.(T2)Bloqueado("+id+"");
            }else if(identificadorUnico == 102) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ES
                System.out.println ("le.(T3)Bloqueado("+id+"");
            }
            wait();
        }
        this.leit++;

        if(identificadorUnico == 102) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO
            System.out.println ("le.(T2)Lendo("+id+"");
            if(this.aux % 2 == 0) { //VERIFICA SE É PAR OU ÍMPAR
                System.out.println ("le.(T2)Valor da variável é "+this.aux+" e é PAR("+ id +")");
            }else {
                System.out.println ("le.(T2)Valor da variável é "+this.aux+" e é ÍMPAR("+ id +")");
            }
        }else if(identificadorUnico == 103) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ
            System.out.println ("le.(T3)Lendo("+id+"");
            System.out.println ("le.(T3)Valor da variável é "+this.aux+" (" + id +")");
        }
    } catch (InterruptedException e) { }
}
```

No **EntraLeitor** teremos duas threads que irá acessar, a thread dois e três, para facilitar na visualização, coloquei uma variável vindo como parâmetro chamada **identificadorUnico**, para quando for imprimir, sabermos qual é a thread que está sendo executada, assim saberemos o que está acontecendo.

Então, temos *THREAD 1 = 101*, *THREAD 2 = 102* e *THREAD 3 = 103*.

Fazemos uma pequena verificação de par ou ímpar, caso for a THREAD 2. E irá imprimir a informação.

Se caso for a THREAD 3, irá apenas imprimir o valor da variável.

```
// Saida para leitores
public synchronized void SaiLeitor (int id, int identificadorUnico) {
    this.leit--;
    if (this.leit == 0)
        this.notify();
    if(identificadorUnico == 102) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE
        System.out.println ("le.(T2)Saindo("+id+"");
    }else if(identificadorUnico == 103) { // PARA AJUDAR NA VISUALIZAÇÃO
        System.out.println ("le.(T3)Saindo("+id+"");
    }
}
```

Aqui seguimos a mesma lógica já apresentada, a única diferença é que serve para chamar a saída dos leitores, então THREAD 2 e 3 que está lendo, irá sair...

```
// Entrada para escritores
public synchronized void EntraEscrivor (int id, int identificadorUnico) {
    try {
        while ((this.leit > 0) || (this.escr > 0)) {

            if(identificadorUnico == 101) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO BLOQUEADO
                System.out.println ("le.(T1)Bloqueado("+id+"");
            }else if(identificadorUnico == 103) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD 1
                System.out.println ("le.(T3)Bloqueado("+id+"");
            }
            wait();
        }
        this.escr++;
        if(identificadorUnico == 101) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO BLOQUEADO
            System.out.println ("le.(T1)Escrevendo("+id+"");
            this.aux++;
        }else if(identificadorUnico == 103) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO BLOQUEADO
            System.out.println ("le.(T3)Escrevendo("+id+"");
            this.aux = id;
        }
    } catch (InterruptedException e) { }
}

// Saida para escritores
public synchronized void SaiEscrivor (int id, int identificadorUnico) {
    this.escr--;
    notifyAll();
    if(identificadorUnico == 101) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO BLOQUEADO
        System.out.println ("le.(T1)Saindo("+id+"");
    }else if(identificadorUnico == 103) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO BLOQUEADO
        System.out.println ("le.(T3)Saindo("+id+"");
    }
}
```

Seguimos a mesma lógica, mas nesse caso a *THREAD 1* irá utilizar esse método também. Se caso for a *THREAD 1*, irá incrementar a *aux*. Já a *THREAD 3* irá colocar em *aux* o seu valor de *id*...

```
// Saída para escritores
public synchronized void SaiEscrivor (int id, int identificadorUnico) {
    this.escr--;
    notifyAll();
    if(identificadorUnico == 101) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ SENDO
        System.out.println ("le.(T1)Saindo("+id+"");
    } else if(identificadorUnico == 103) { // PARA AJUDAR NA VISUALIZAÇÃO DE QUE THREAD ESTÁ
        System.out.println ("le.(T3)Saindo("+id+"");
    }
}
```

saída de escritores, mesma lógica de saída de leitores.

```
// T1
class T1 extends Thread {
    int id; //identificador da thread
    int delay; //atraso bobo
    int identificadorUnico; // PARA AJUDAR NA IDENTIFICAÇÃO
    LE monitor; //objeto monitor para coordenar a lógica de execução das threads

    // Construtor
    T1 (int id, int delayTime, LE m) {
        this.id = id;
        this.delay = delayTime;
        this.monitor = m;
        this.identificadorUnico = 101;
    }

    // Método executado pela thread
    public void run () {
        double j=777777777.7, i;
        try {
            for (;;) {
                this.monitor.EntraEscrivor(this.id, this.identificadorUnico);
                for (i=0; i<1000000000; i++) {j=j/2;} //...loop bobo para simbolizar o tempo de ES
                this.monitor.SaiEscrivor(this.id, this.identificadorUnico);
                sleep(this.delay); //atraso bobo...
            }
        } catch (InterruptedException e) { return; }
    }
}
```

A classe da *THREAD 1*, nada muito diferente do normal.. Colocamos um *for* para ter um tempinho... Como iremos apenas escrever, teremos um ***EntraEscrivor*** e ***SaiEscrivor***

```

//T2
class T2 extends Thread {
    int id; //identificador da thread
    int delay; //atraso bobo...
    int identificadorUnico; // PARA AJUDAR NA IDENTIFICAÇÃO
    LE monitor; //objeto monitor para coordenar a lógica de execução das threads

    // Construtor
    T2 (int id, int delayTime, LE m) {
        this.id = id;
        this.delay = delayTime;
        this.monitor = m;
        this.identificadorUnico = 102;
    }

    // Método executado pela thread
    public void run () {
        double j=777777777.7, i;
        try {
            for (;;) {
                this.monitor.EntraLeitor(this.id, this.identificadorUnico);
                for (i=0; i<1000000000; i++) {j=j/2;} //...loop bobo para simbolizar o tempo de LEIT
                this.monitor.SaiLeitor(this.id, this.identificadorUnico);
                sleep(this.delay); //atraso bobo...
            }
        } catch (InterruptedException e) { return; }
    }
}

```

Mesma coisa, como apenas faz leitura, então terá apenas ***EntraLeitor*** e ***SaiLeitor***.

```

//T3
class T3 extends Thread {
    int id; //identificador da thread
    int delay; //atraso bobo...
    int identificadorUnico; // PARA AJUDAR NA IDENTIFICAÇÃO
    LE monitor; //objeto monitor para coordenar a lógica de execução das threads

    // Construtor
    T3 (int id, int delayTime, LE m) {
        this.id = id;
        this.delay = delayTime;
        this.monitor = m;
        this.identificadorUnico = 103;
    }

    // Método executado pela thread
    public void run () {
        double j=777777777.7, i;
        try {
            for (;;) {
                this.monitor.EntraLeitor(this.id, this.identificadorUnico);
                for (i=0; i<1000000000; i++) {j=j/2;} //...loop bobo para simbolizar o tempo
                this.monitor.SaiLeitor(this.id, this.identificadorUnico);
                for (i=0; i<1000000000; i++) {j=j/2;} //...loop bobo para simbolizar o tempo
                this.monitor.EntraEscritor(this.id, this.identificadorUnico);
                for (i=0; i<1000000000; i++) {j=j/2;} //...loop bobo para simbolizar o tempo
                this.monitor.SaiEscritor(this.id, this.identificadorUnico);
                sleep(this.delay); //atraso bobo...
            }
        } catch (InterruptedException e) { return; }
    }
}

```

Na THREAD 3 temos uma diferença, a thread utiliza tanto o leitor quanto o escritor, então chamamos os dois e colocamos um for entre para ter um tempinho como pedido no laboratório.

```

// Classe principal
class LeitorEscritor {
    static final int UM = 1;
    static final int DOIS = 1;
    static final int TRES = 1;

    public static void main (String[] args) {
        int i;
        LE monitor = new LE();
        T1[] a = new T1[UM];
        T2[] b = new T2[DOIS];
        T3[] c = new T3[TRES];

        //inicia o log de saida
        System.out.println ("import verificaLE");
        System.out.println ("le = verificaLE.LE()");

        for (i=0; i<UM; i++) {
            a[i] = new T1(i+1, (i+1)*500, monitor);
            a[i].start();
        }
        for (i=0; i<DOIS; i++) {
            b[i] = new T2(i+1, (i+1)*500, monitor);
            b[i].start();
        }


        for (i=0; i<TRES; i++) {
            c[i] = new T3(i+1, (i+1)*500, monitor);
            c[i].start();
        }
    }
}

```

Por último, a classe principal, que não tem nada mais que o normal.

VERIFICANDO AS SAÍDAS E TESTANDO POSSIBILIDADES.

Threads 1, 2, 3 = 1;



```
<terminated> LeitorEscritor [Java Application] /home/victor/.p2/pool/plugins/org.eclipse.justj.op
import verificaLE
le = verificaLE.LE()
le.(T1)Escrevendo(1)
le.(T2)Bloqueado(1)
le.(T1)Saindo(1)
le.(T2)Lendo(1)
le.(T2)Valor da variável é 1 e é ÍMPAR(1)
le.(T3)Lendo(1)
le.(T3)Valor da variável é 1 (1)
le.(T3)Saindo(1)
le.(T2)Saindo(1)
le.(T1)Escrevendo(1)
le.(T3)Bloqueado(1)
le.(T1)Saindo(1)
le.(T3)Escrevendo(1)
le.(T3)Bloqueado(1)
```

1. Começamos com a T1 escrevendo, logo a variável auxiliar está sendo incrementada. (Sua principal função).
2. Depois a T2 tenta ler, porém, não pode pois T1 está escrevendo.
3. Então, T1 acaba de escrever.
4. Agora, T2 consegue entrar depois de T1 ter acabado de escrever. T2 Lê a variável.
5. T2 diz que a variável é ímpar. (Sua principal função).
6. T3 começa lendo, como não há impedimento de vários leitores lerem ao mesmo tempo ela consegue ler, mesmo que a T2 ainda não tenha acabado.
7. T3 diz que o valor da variável é 1. (Uma das suas funções é ler a variável e imprimir)
8. T3 sai.
9. T2 sai.
10. T1 começa a escrever.
11. T3 quer escrever e bloqueia
12. T1 sai (aux = 2);
13. T3 escreve(aux = id, como o id é 1, então aux = 1)

```

le.(T1)Saindo(1)
le.(T3)Escrevendo(1)
le.(T2)Bloqueado(1)
le.(T3)Saindo(1)
le.(T2)Lendo(1)
le.(T2)Valor da variável é 1 e é ÍMPAR(1)
le.(T2)Saindo(1)
le.(T1)Escrevendo(1)
le.(T1)Saindo(1)
le.(T3)Lendo(1)
le.(T3)Valor da variável é 2 (1)
le.(T3)Saindo(1)

```

1. *T2 tenta ler, porém T3 está escrevendo. (ENQUANTO ler não pode ter alguém escrevendo)*
2. *T3 sai e termina. (aux = 1, como já mencionado)*
3. *T2 ler, como aux é igual a 1, dirá que é ímpar.*
4. *T2 sai.*
5. *T1 entra pra escrever. (aux = 2)*
6. *T1 sai de escrever.*
7. *T3 ler*
8. *T3 imprime o valor da variável.*

Nesses exemplos, conseguimos perceber que as regras são seguidas conforme foi solicitado no laboratório, seguindo a regra básica do LE.

Vamos modificar a quantidade de threads só para visualizar...

THREAD 1 = 2

THREAD 2 = 2

THREAD 3 = 4


```

<terminated> LeitorEscrutor [Java Application] /home/victor/.p2/pool/plugins/org.eclipse
import verificaLE
le = verificaLE.LE()
le.(T1)Escrevendo(1)
le.(T1)Bloqueado(2)
le.(T2)Bloqueado(1)
le.(T2)Bloqueado(2)
le.(T1)Saindo(1)
le.(T1)Escrevendo(2)
le.(T2)Bloqueado(2)
le.(T2)Bloqueado(1)
le.(T1)Saindo(2)
le.(T3)Lendo(4)
le.(T3)Valor da variável é 2 (4)
le.(T2)Lendo(1)
le.(T2)Valor da variável é 2 e é PAR(1)
le.(T2)Lendo(2)
le.(T2)Valor da variável é 2 e é PAR(2)
le.(T3)Lendo(1)
le.(T3)Valor da variável é 2 (1)
le.(T3)Lendo(2)
le.(T3)Valor da variável é 2 (2)
le.(T3)Lendo(3)
le.(T3)Valor da variável é 2 (3)
le.(T1)Bloqueado(1)
le.(T2)Saindo(1)
le.(T3)Saindo(3)
le.(T3)Saindo(4)
le.(T3)Saindo(2)
le.(T2)Saindo(2)
le.(T3)Saindo(1)
le.(T1)Escrevendo(1)
le.(T3)Bloqueado(3)
le.(T3)Bloqueado(4)
le.(T3)Bloqueado(2)
le.(T2)Bloqueado(1)
le.(T1)Saindo(1)
le.(T3)Escrevendo(3)

```

Podemos verificar que a regra se satisfaz também. Seguindo o mesmo procedimento da primeira exemplificação.

*Ao modificar os valores das threads, percebemos que o maior impacto acontece quando mudamos da thread um e três, visto que elas estão modificando diretamente aux. (Influenciando também no número de blocks, pois são threads que estão diretamente ligadas com a parte de escrever, onde **NÃO PODE HAVER NEM LEITORES E NEM ESCRITORES**, então há uma ocorrência maior de block, como podemos ver neste código em comparação com o outro onde a quantidade de threads era 1)*