MONITORES

- Semáforos apresentam uma importante característica negativa:
  - O uso correto do semáforo está distribuído por todo o programa
  - Esquecer um *signal* após uma seção crítica pode levar a um *deadlock* difícil de solucionar.

- **Monitor** é uma primitiva de programação concorrente que centraliza todas as operações em uma função ou módulo privilegiado.

- Um **código monitor** garante que, em qualquer estado, existe **no máximo um processo** executando um código dentro do monitor.

- Quando um processo chama uma função que está em um módulo monitor e já existe um outro processo executando uma função do módulo:
  - O processo é bloqueado até que o monitor seja liberado.
- O monitor garante naturalmente a exclusão mútua das suas funções

Exemplo: Problema da Seção Crítica usando monitores.

<table>
<tr><td colspan="2">

Monitor SC;

função SC.P {
    // código
}

função SC.Q {
    // código
}

</td></tr>
<tr><td>

Processo P

Laço infinito
    Seção não crítica
    SC.P( )
Fim_laço

</td><td>

Processo Q

Laço infinito
    Seção não crítica
    SC.Q( )
Fim_laço

</td></tr>
</table>

MONITORES EM JAVA

A Linguagem JAVA permite definir métodos e trechos de códigos como monitores, usando a palavra reservada **synchronized**

Para métodos temos:

```
synchronized tiporetorno função(parâmetros){
    //código;
}
```

Para trechos de códigos:

```
Object obj = new Object();
synchronized (obj) {
  // código da seção crítica
}
```

Exemplo: O seguinte código mostra uma função monitor que começa imprimindo a mensagem "Mostra para nome_thread" e termina imprimindo a mensagem "Acabei com nome_thread"

Entre as duas mensagens, a thread é bloqueada por um período de tempo determinado. Se a função não fosse monitor, outra thread seria escalonada, intercalando outra mensagem

entre as duas.

```
class compartilha {
    public synchronized void Mostra(String texto){
        System.out.print("Mostra para "+texto);
         try {
            Thread.sleep(500);  // bloqueia monitor por 500 milisegundos
         } catch ( InterruptedException ie) {
            System.err.println( ie.toString());
         }
        System.out.println(" Acabei com "+texto);
    }
}
```

```
// classe que implementa as threads

class ImpThread extends Thread {
    int t;
    compartilha comp;
    // construtor
    public ImpThread (String nome, int tempo,compartilha c) {
       super(nome);// chama construtor da superclasse
       t=tempo; // tempo em milissegundos
       comp=c;
    }
```

```
    // metodo que executa a thread
    public void run () {
       while (true) {
        try {
            Thread.sleep(t);  // bloqueia thread pelo tempo
        } catch ( InterruptedException ie) {
             System.err.println( ie.toString());
        }
        comp.Mostra(getName());
       }
    }
}
```

```
public class monitor {// classe de teste das threads
    public static void main (String args[]) {
        compartilha comp=new compartilha();
        ImpThread th1,th2,th3;
        // cria as threads
        th1 = new ImpThread("A",100,comp);
        th2 = new ImpThread("B",200,comp);
        th3 = new ImpThread("C",300,comp);
        th1.start();   // inicia a execucao das threads
        th2.start();
        th3.start();
    }
}
```

BLOQUEANDO PROCESSOS EM MONITOR

- A linguagem Java permite bloquear uma *thread* dentro de um monitor usando o método **wait();**
- O monitor fica liberado para outras *threads*.
- Para desbloquear uma *thread* no monitor, usamos o método **notify()**
- Para desbloquear todas as *threads* em um monitor, usamos o método **notifyAll()**
- Em ambos os casos, a *thread* desbloqueada **continua no monitor** apenas quando o mesmo **estiver liberado**.

Exemplo: No programa anterior, considere a seguinte alteração

```
class compartilha {
    public synchronized void Mostra(String texto){
        System.out.print("Mostra para "+ texto);
        try {
            if (texto.compareTo("A")==0) wait();  // bloqueia A
            if (texto.compareTo("C")==0) notify(); // desbloqueia thread
            Thread.sleep(500);  // bloqueia monitor por 500 milissegundos
        } catch ( InterruptedException ie) {
            System.err.println( ie.toString());
        }
        System.out.println(" Acabei com "+ texto);
    } }
```