VARIÁVEIS DE CONDIÇÃO

- São usadas para bloquear processos em uma fila específica, dentro de um bloco de monitor
- Duas operações:

waitC(cond)	**signalC(cond)**
Coloca p na fila de cond p.estado = bloqueado monitor.trava = liberado	Se (cond <> vazia) Retira q da fila cond q.estado = pronto

- Obs.: Em alguma linguagens podemos ter variáveis de condição fora de bloco monitor

VARIÁVEIS DE CONDIÇÃO EM JAVA

- Java implementa variáveis de condição fora de um bloco monitor, usando apenas uma trava de exclusão mútua
- A trava deve ser obtida usando uma classe Lock, na forma:

```
Lock trava = new ReentrantLock();
```

- Um bloco de exclusão mútua por trava deve ser iniciado com uma chamada ao método lock() e liberado com o método unlock();

```
trava.lock();
// bloco de exclusão mútua de trava
trava.unlock()
```

- Usando o método newCondition() da classe **Lock**, são obtidas as variáveis de condição:

```
Condition A  = trava.newCondition();
```

- Um processo é bloqueado na variável de condição usando o método await(), na forma A.await().
- Um processo é desbloqueado quando o método signal() é chamado, na forma A.signal().

- Todos os processos podem ser desbloqueados usando o método signalAll() na forma A.signalAll().

Exemplo 1: Considere o seguinte programa em Java sem variáveis de condição:

```
class compartilha {
    public synchronized void Mostra(String texto){
        System.out.print("Mostra para "+texto);
        try {
            Thread.sleep(50);  // bloqueia por 50 milisegundos
        } catch ( InterruptedException ie) {
            System.err.println( ie.toString());
        }
        System.out.println(" Acabei para "+texto);
    } }
```

```
class ImpThread extends Thread { // implementa as threads
    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 varcond {
    public static void main (String args[]) {
        compartilha comp = new compartilha();
        ImpThread th1,th2;
        // cria as threads
        th1 = new ImpThread("A",10,comp);
        th2 = new ImpThread("B",500,comp);
        // inicia a execucao das threads
        th1.start();
        th2.start();
    }
}
```

Produz como resultado algo parecido a:

Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para B Acabei para B
Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para A Acabei para A
Mostra para A Acabei para A

A *thread* A consegue imprimir muitas vezes mais que a *thread* B.

Exemplo 2: Igual ao exemplo 1 usando bloco de trava:

```
class compartilha {
   final Lock trava = new ReentrantLock();
    public void Mostra(String texto){
        trava.lock();
        System.out.print("Mostra para "+texto);
         try {
            Thread.sleep(50);  // bloqueia por 50 milisegundos
         } catch ( InterruptedException ie) {
            System.err.println( ie.toString());          }
         System.out.println(" Acabei para "+texto);
         trava.unlock();
   } }
```

Exemplo 3: Usando variáveis de condição para sincronizar as threads.

```
class compartilha {
   final Lock trava = new ReentrantLock();
   final Condition A  = trava.newCondition();
   final Condition B = trava.newCondition();
   int libera = 0;  // variável de controle

   public void Mostra(String texto){
        trava.lock();
        try {
          if ((libera ! = 0) && (texto.compareTo("A") = = 0)) A.await();
          if ((libera ! = 1) && (texto.compareTo("B") = = 0)) B.await();
          System.out.print("Mostra para "+texto);
```

```
            Thread.sleep(50);  // bloqueia por 50 milisegundos
        } catch ( InterruptedException ie) {
            System.err.println( ie.toString());
        }
        System.out.println(" Acabei para "+texto);
        libera = (libera + 1) % 2;
        if (texto.compareTo("A") = = 0) B.signal();
        if (texto.compareTo("B") = = 0) A.signal();
        trava.unlock();
    }
}
```

Produz como resultado:

Mostra para A Acabei para A
Mostra para B Acabei para B
Mostra para A Acabei para A
Mostra para B Acabei para B
Mostra para A Acabei para A
Mostra para B Acabei para B
Mostra para A Acabei para A
Mostra para B Acabei para B

Realizando a sincronização com a exclusão mútua.

VARIÁVEIS DE CONDIÇÃO EM MONITORES JAVA

- Variáveis de condição em geral são usadas na forma:

```
if (condicional) varcond.wait();
```

sendo condicional o teste realizado e

varcond o identificador da variável de condição

dentro de um bloco de exclusão mútua por trava.

- Para usar monitor e não trava, basta substituir o comando acima por:

```
while (condicional) wait();
```

  e o(s) comando(s) signal() por notifyAll().

- Neste caso, todos os processos são desbloqueados e aqueles que não satisfazerem a condição são novamente bloqueados.

Exemplo 4:

```
class compartilha {
    int libera = 0;  // variável de controle
   public synchronized void Mostra(String texto){
       try {
          if (texto.compareTo("A") = = 0)
             while (libera ! = 0) wait();
          if (texto.compareTo("B") = = 0)
             while (libera ! = 1) wait();
           System.out.print("Mostra para "+texto);
           Thread.sleep(50);  // bloqueia por 50 milisegundos
```

```
        } catch ( InterruptedException ie) {
            System.err.println( ie.toString());
        }
        System.out.println(" Acabei para "+texto);
        libera = (libera + 1) % 2;
        notifyAll();
    }
}
```

As demais classes são idênticas aos exemplos anteriores.