Sistemas Distribuídos

Universidade do Minho

2011/2012

lock/ monitor Classe

lock/ monitor Instância

```
public class Contador {
    private long i=0;
    private static long i2=0;

    public long get() return i;
    public synchronized void inc() i++;

    public synchronized static void incStatic() i2++;
    public static long getStatic() return i2;
}
```

```
C = new Contador();
Thread1(C);
Thread2(C);
```

```
public void run(){
        for(int i=0; i<1000000; i++){
            c.incStatic();
            this.c.inc();
        }
    }
```

- Exemplo:

```
public synchronized void metodo() {
        i++;
}
```

```
public void metodo(){
...
   synchronized (objecto){
          i++;
   }
...
}
```

Exclusão Mútua

```
synchronized void metodo(){
...
}
```

Exclusão Mútua explícita

```
import
java.util.concurrent.locks.Reentran
tLock;

lock = new ReentrantLock();

void metodo(){
    this.lock.lock();
    try{
        ...
    } catch (InterruptedException e) {
        e.printStackTrace();
        }
    finally{
        this.lock.unlock();
        }
}
```

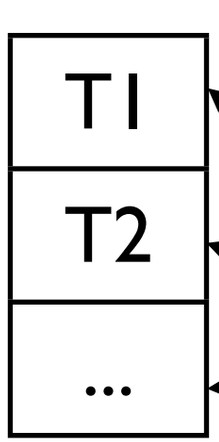
- Variáveis de condição:
  - suspensão/retoma de execução dentro de zona crítica;
  - mecanismo intrínseco de variável de condição em todos os objectos;
  - podem ser usadas dentro de uma zona crítica;
  - métodos wait(), notify(), notifyAll();

- Exemplo:

```
synchronized void metodo(){
...
while(condição){
   this.wait();
   ...
}
```

```
synchronized void metodo2(){
   this.notify();
}
```

Esta thread T2 em espera




```
synchronized void metodo3(){
   this.notifyAll();
}
```

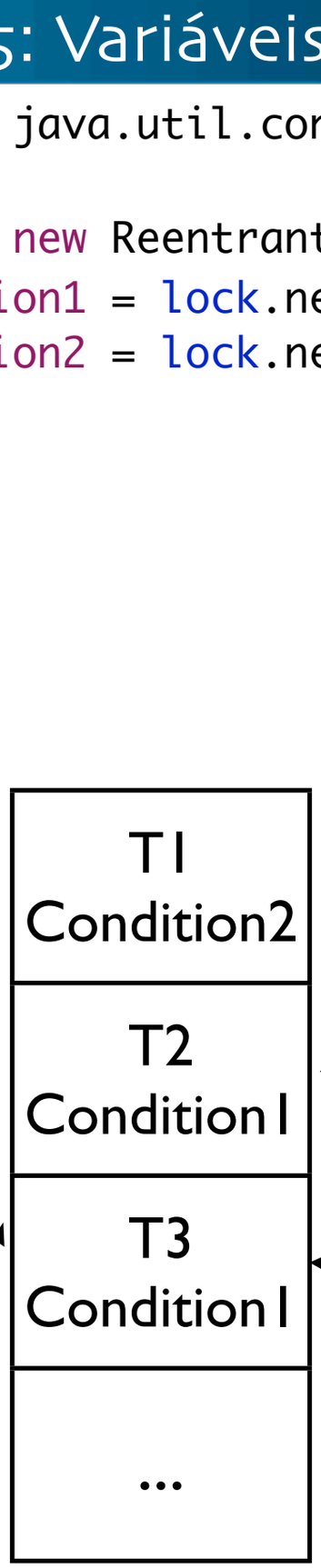
- Variáveis de condição:
  - suspensão/retoma de execução dentro de zona crítica;
  - usada em conjugação com o ReentrantLock();
  - java.util.concurrent.locks.ReentrantLock
    – métodos lock(), unlock(), newCondition();
  - java.util.concurrent.locks.Condition
    – métodos relevantes: await(), signal(), signalAll()

```
import java.util.concurrent.locks.Condition;

lock = new ReentrantLock();
condition1 = lock.newCondition();
condition2 = lock.newCondition();
```

- Exemplo:

```
void metodo(){

lock.lock();
try{
while(condição){
    condition1.await();
    ...
}
finally{
 lock.unlock();
}
```

T3 em espera na condição 1



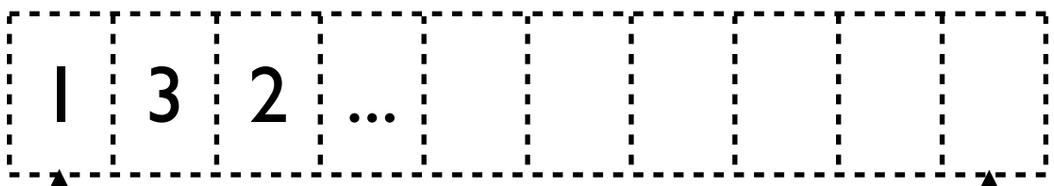

```
void metodo2(){
   lock.lock()
    try{
    ...
    condition2.signal();
    finally{
     lock.unlock();
    }
  }

void metodo3(){
   lock.lock();
   try{
   ...
   condition1.signalAll();
   finally{
    lock.unlock();
   }
}
```

Aula 5: Exercicio 1

- Exercicio aula anterior: Implemente uma classe BoundedBuffer que ofereça as operações void put(int v) e int get() sobre um array cujo tamanho é definido no momento da construção de uma instância. O método put() deverá bloquear enquanto o array estiver cheio e o método get() deverá bloquear enquanto o array estiver vazio. Os métodos oferecidos podem estar sujeitas a invocações de threads concorrentes sobre uma instância partilhada. A classe BoundedBuffer deverá garantir a correcta execução em cenário multi-thread.

- Reimplemente a classe BoundedBuffer de modo a distinguir as situações de bloqueio pelo array estar vazio e estar cheio.

Aula 5: Exercicio 1



BoundedBuffer

por exemplo: LinkedList;
int size;

- void put(int v);
- int get();

Produtor
implements Runnable

Consumidor
implements Runnable

Condições a respeitar:

- put(v) -> Bloquear enquanto o array estiver cheio;
- get() -> Bloquear enquanto o array estiver vazio.

Usando variáveis de condição e exclusão mútua;

Aula 5: Exercicio 2

- Implemente a classe RWLock com os métodos readLock(), readUnlock(), writeLock() e writeUnlock() de modo a permitir o acesso simultâneo de múltiplos leitores a uma dada região crítica, ou em alternativa, o acesso de um único escritor.

- Usando ReetrantLock e respectivas variáveis de condição.