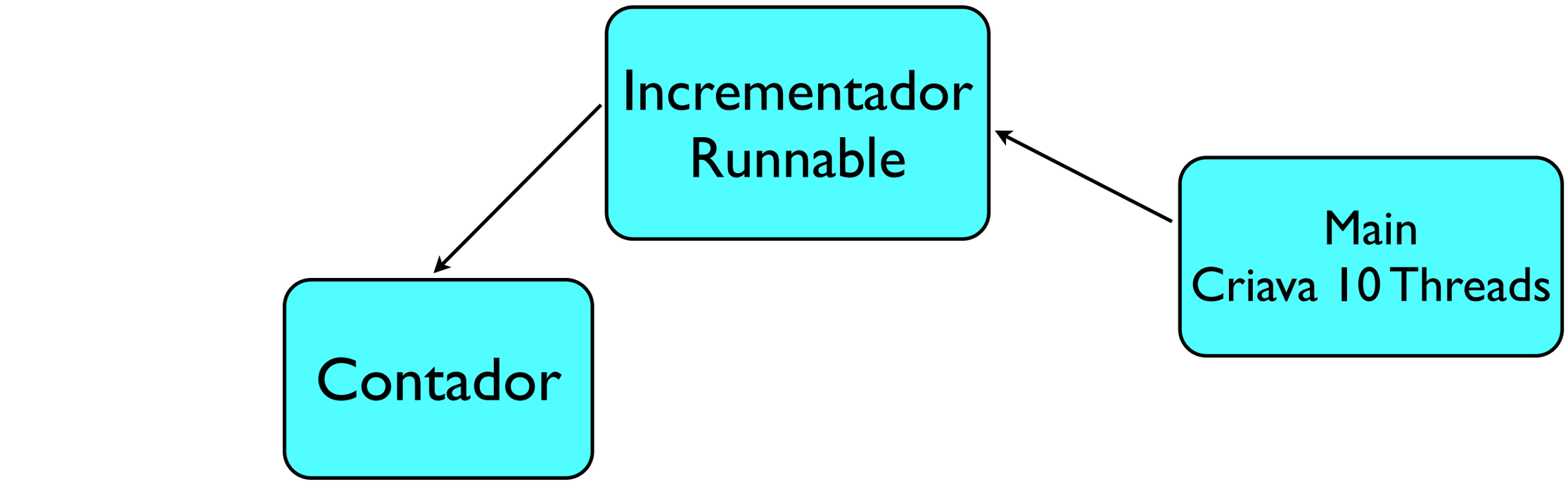
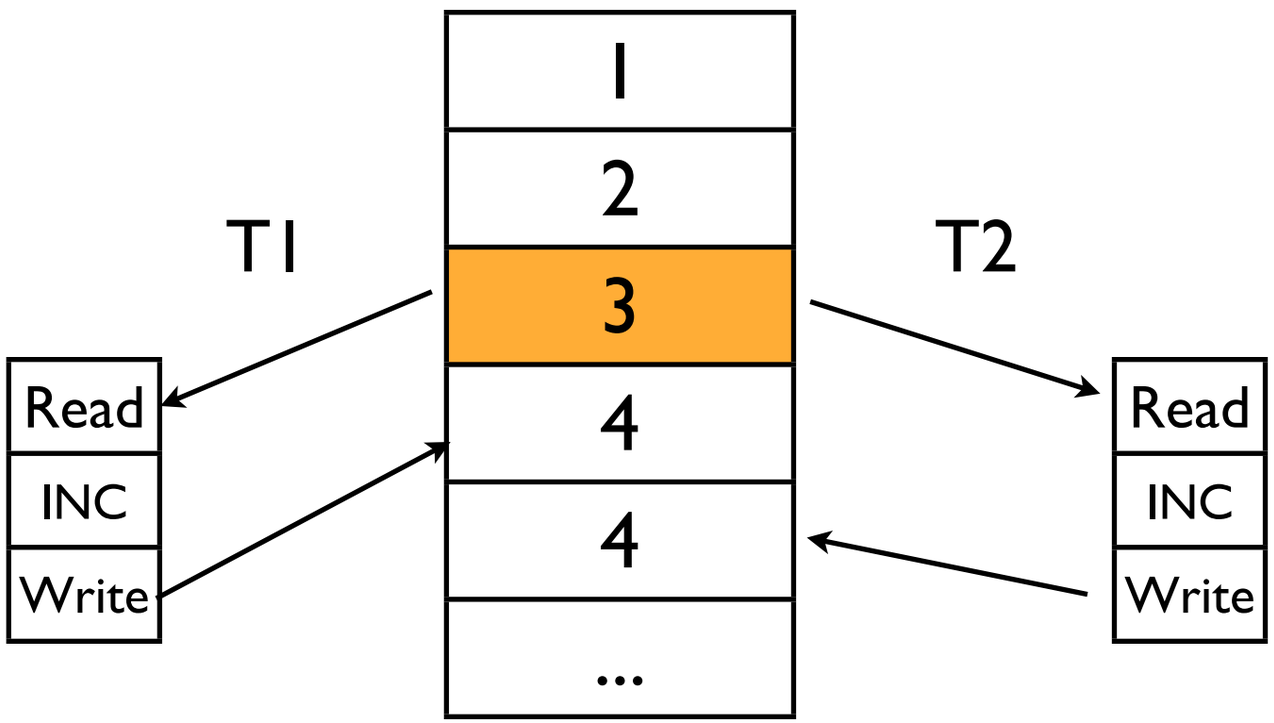
Sistemas Distribuídos

Universidade do Minho

2011/2012

- java.lang.Runnable
- java.lang.Thread
- classes que extendam Thread ou implementem Runnable devem reimplmentar o método run()
- métodos relevantes: Thread(), start(), sleep(), join();





Output variável!

- Exclusão mútua: garantir que dois processos ou threads não tenham acesso simultaneamente a um recurso partilhado.
- Proteger secções críticas do código.

- Em JAVA:
  - Monitor locks para garantir exclusão mútua;
  - mecanismo intrínseco de exclusão mútua disponível em todos os objectos

- synchronized metodo (argumentos) { statements }
- synchronized (objecto) { statements }

- Exemplo:

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

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

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();
        }
    }
```

Aula 3: Exclusão Mútua

Main

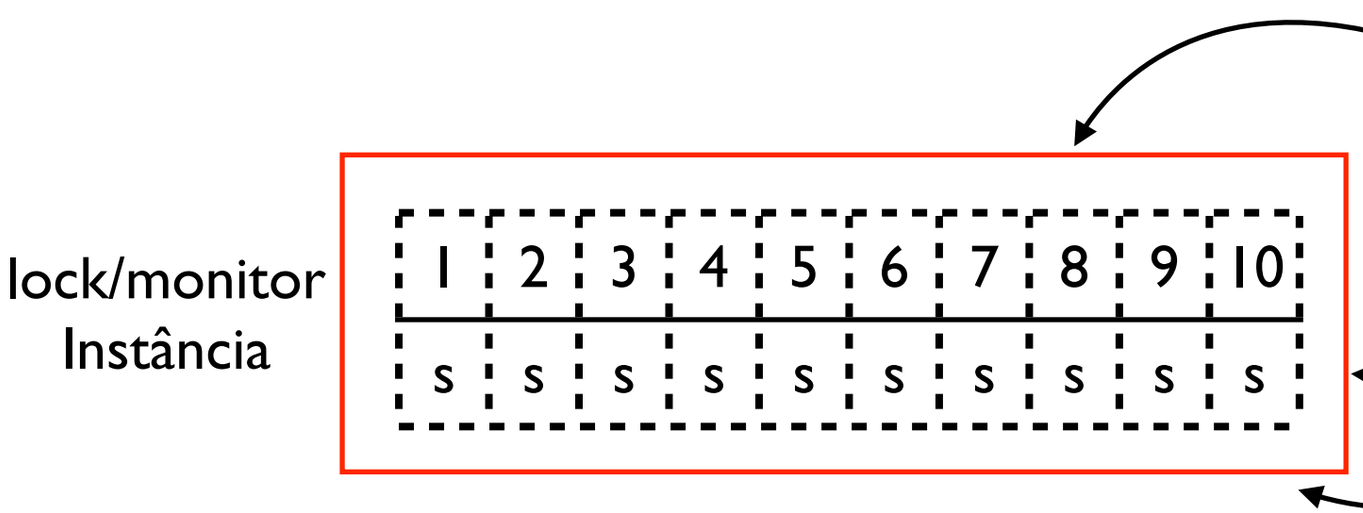
```
Banco banco = new Banco(10);
Thread t[] = new Thread[10];
        for(int i = 0; i < nThreads; i++)
        {
            t[i] = new Thread(new Acesso(banco));
            t[i].start();
        }
```

Acesso implements Runnable

```
public void run() {
        for(int i=0;i<nOps;i++)
        {
            for(int conta=0; conta<nContas; conta++)
                banco.credita(conta,10);
            for(int conta=0; conta<nContas; conta++)
                banco.debita(conta,5);
        }
```

Banco

```
private double[] contas
synchronized void credita(int conta,double
valor){
        contas[conta]+=valor; }
synchronized void debita(int conta,double
valor){
        contas[conta]-=valor; }
void transfere(int contaOrigem,int
contaDestino,double valor){
        this.debita(contaOrigem, valor);
        this.credita(contaDestino, valor);
    }
```



4. Reimplemente a classe Banco utilizando exclusão mútua ao nível das contas individuais. A classe Banco tem que disponibilizar os seguintes métodos:

- void credita(int conta,double valor)
- void debita(int conta,double valor)
- double getSaldoTotal(int conta)
- double getSaldo(int conta)
- void transfere(int contaOrigem,int contaDestino,double valor)

Dica: Adicionar ao exercício anterior uma classe Conta, que disponibilize os métodos:

- void credita(double valor)
- void debita(double valor)
- double getSaldo()

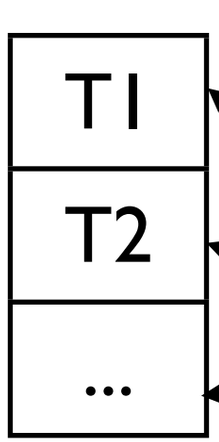
- Variáveis de condição:
  - suspensão/retoma de execução dentro de zona crítica;
  - mecanismo intrínseco de variável de condição em todos os objectos;
  - métodos wait(), notify(), notifyAll();

- Exemplo:

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

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

Esta thread T2 em espera




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

Aula 3: Exercicio 2

- 2. Reimplemente a classe Banco de modo a bloquear as operações que conduzam a saldos negativos.