SCC0204 - Programação Orientada a Objetos

Java Threads

Prof. Jose Fernando Rodrigues **Junior**

http://www.icmc.usp.br/~junio

junio@icmc.usp.br

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO - USP

Introdução

- **Time slicing**: divisão do tempo de processamento entre múltiplas linhas de execução
- **Processo**: instância de um programa que possui recursos básicos para ser executado, em especial, seu **próprio espaço de memória**
- Uma **solução natural** da computação é a quebra de um problema em **múltiplos processos**, os quais interagem por meio de suporte operacional para **comunicação entre processos**
- Alternativamente, a computação contemporânea provê **múltiplas linhas de execução em um mesmo processo (threads)** o que permite quebrar um problema sem o ônus dos sistemas de comunicação entre processos

Introdução e definição

- Threads, também chamadas de **lightweight processes**, requerem menos recursos computacionais (conjunto de estados de processamento) do que processos

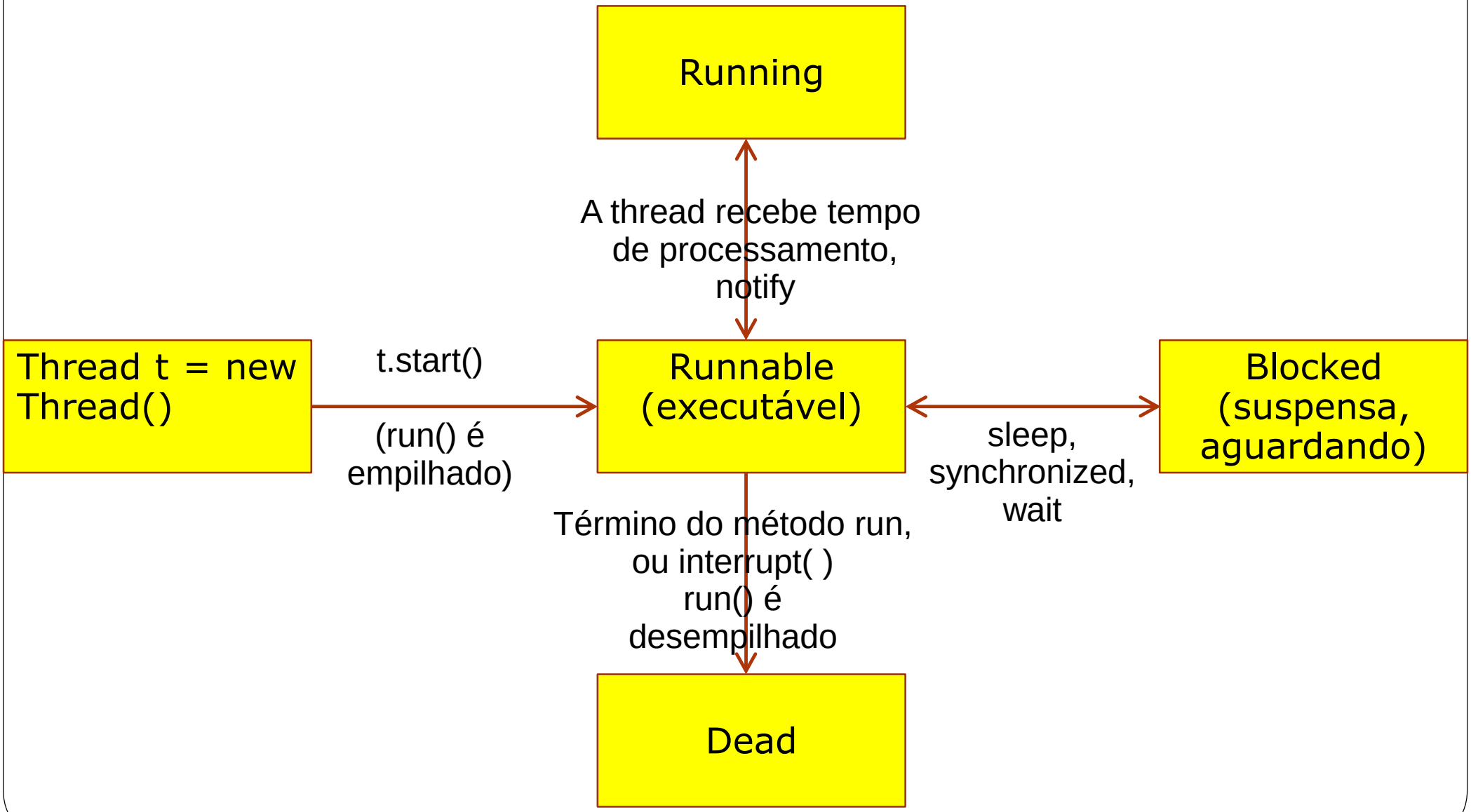
- Em Java, a execução de processos foi projetada para o suporte a threads, de maneira que a primeira e principal thread é iniciada no **método main**

- Em Java há **duas maneiras** principais de se definir threads
  - extends Thread
  - implements Runnable

➔ **Exemplo de definição e criação de threads**

Introdução e definição

- Estados de uma Thread



Execução e concorrência

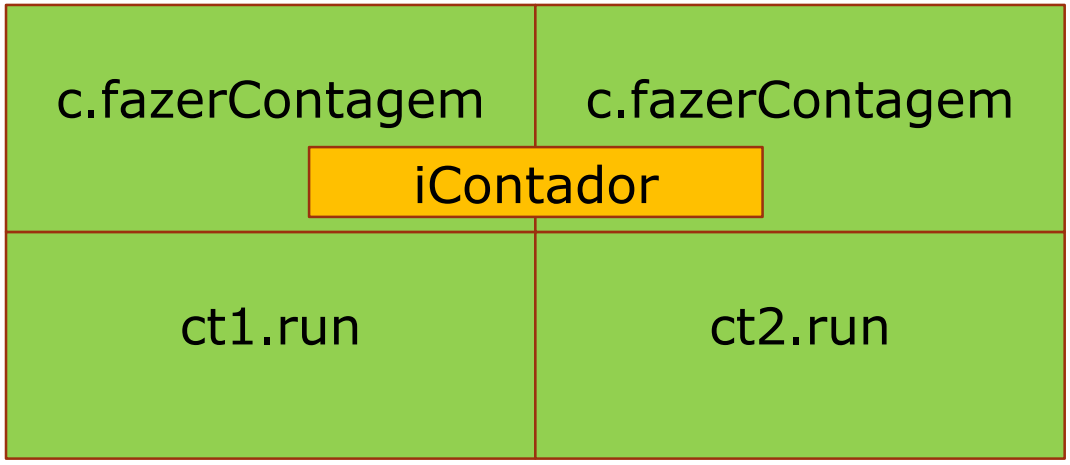
- O método principal de uma thread é o *run()*, evocado indiretamente por meio do método *start()*

➔ **Exemplo com threads se alternando**

Execução e concorrência

- No entanto, este fato causa problemas → **concorrência**

➔ **Exemplo de problema de concorrência com a classe Counter**



Synchronized

- O problema de concorrência é tratado por meio do **recurso synchronized**, um modificador que indica que um método não pode ter sua execução suspensa:
  - **Se uma thread está executando um método synchronized, e outra thread evoca um (o mesmo ou qualquer outro) método synchronized da mesma instância de objeto, então ela é suspensa até que o método termine**
  - As chamadas ao método synchronized são enfileiradas, de maneira que a **ordem de evocação** por parte das diferentes threads é respeitada
  - Métodos synchronized podem evocar outros métodos synchronized ou não – o desbloqueio só ocorre quando o método synchronized que gerou a suspensão das outras threads terminar
- Para evitar problemas de concorrência, **todos os métodos que alteram o estado** de um objeto comum a mais de uma thread, devem ser synchronized, o que garante que o dado não será alterado de maneira imprevisível

Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

```
class Amigo{
...
   public synchronized void curva_se_a(Amigo umAmigo) {
        System.out.println(this.name + ": " + umAmigo.name + " curvou-se para mim!");
        umAmigo.descurva_se(this);
    }
    public synchronized void descurva_se(Amigo umAmigo) {
        System.out.println(this.name + ": " + umAmigo.name + " descurvou-se para mim!");
    }
}
```

```
class class DeadlockThreading extends Thread {
...
 public void run(){
        amigo1.curva_se_a(amigo2);
    }
}
```

```
Public static void main(String args[]){
        Amigo jose = new Amigo("Jose");
        Amigo joao = new Amigo("Joao");
        DeadlockThreading dtTemp1 = new DeadlockThreading(jose, joao);
        DeadlockThreading dtTemp2 = new DeadlockThreading(joao, jose);
        dtTemp1.start();
        dtTemp2.start();
}
```

Synchronized-deadlock

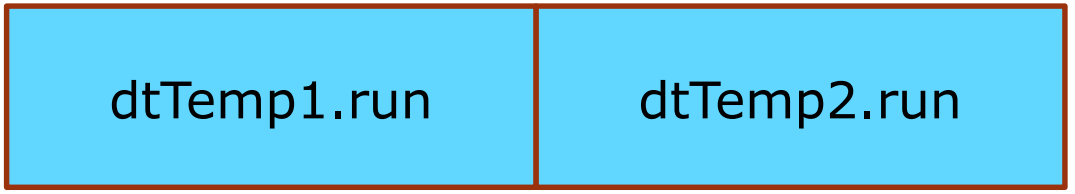
- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

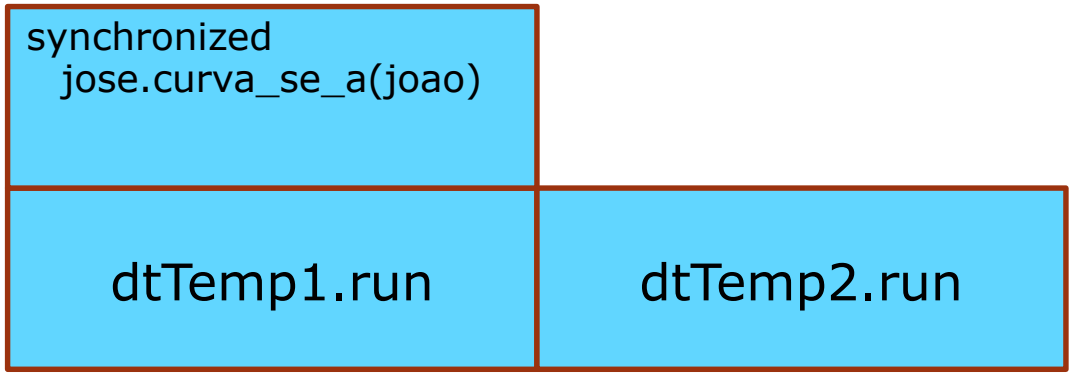


Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

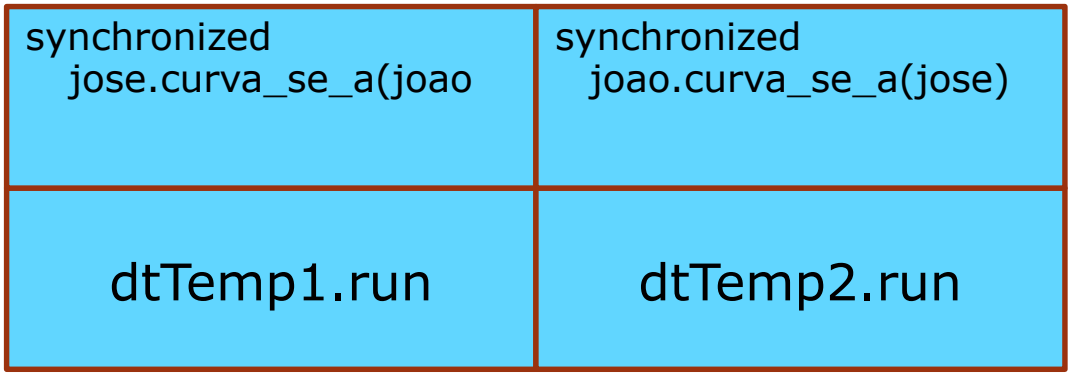
O método synchronized bloqueia outras threads que tentem executar métodos synchonized de jose



Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo



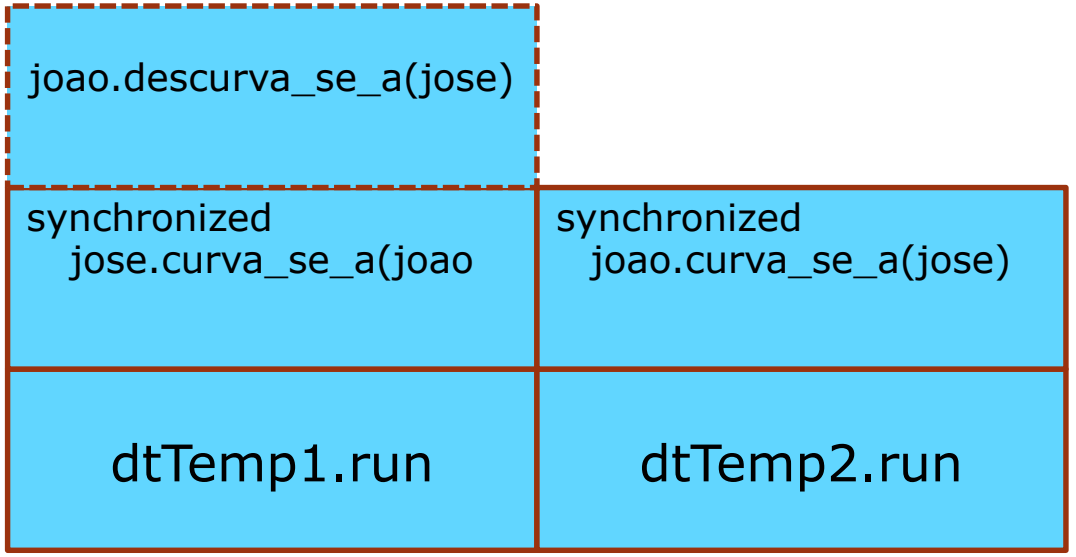
O método synchronized bloqueia outras threads que tentem executar métodos synchonized de joao

Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

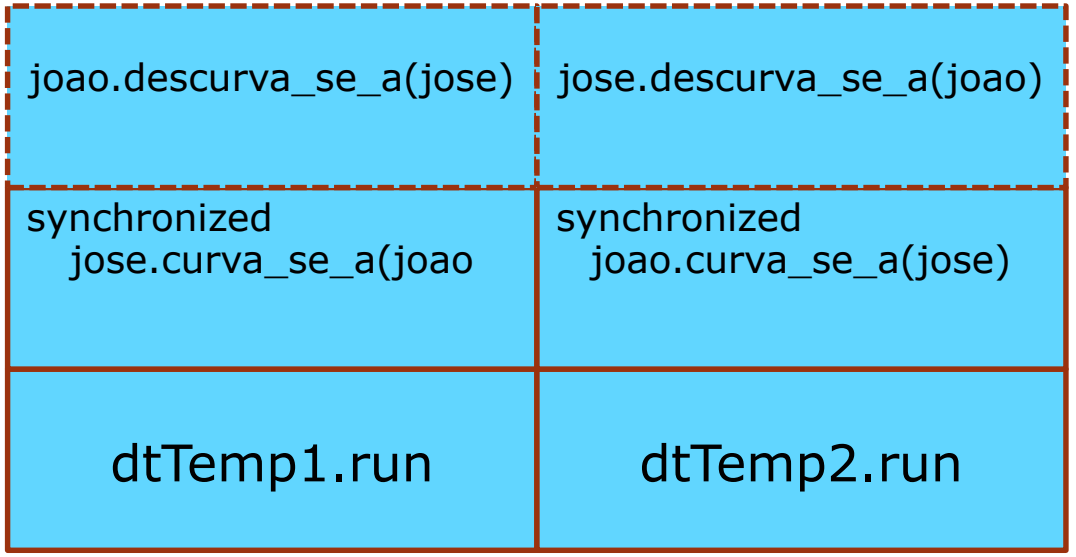
dtTemp1 tenta evocar o método descurva_se_a de joao, e é suspensa



Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo



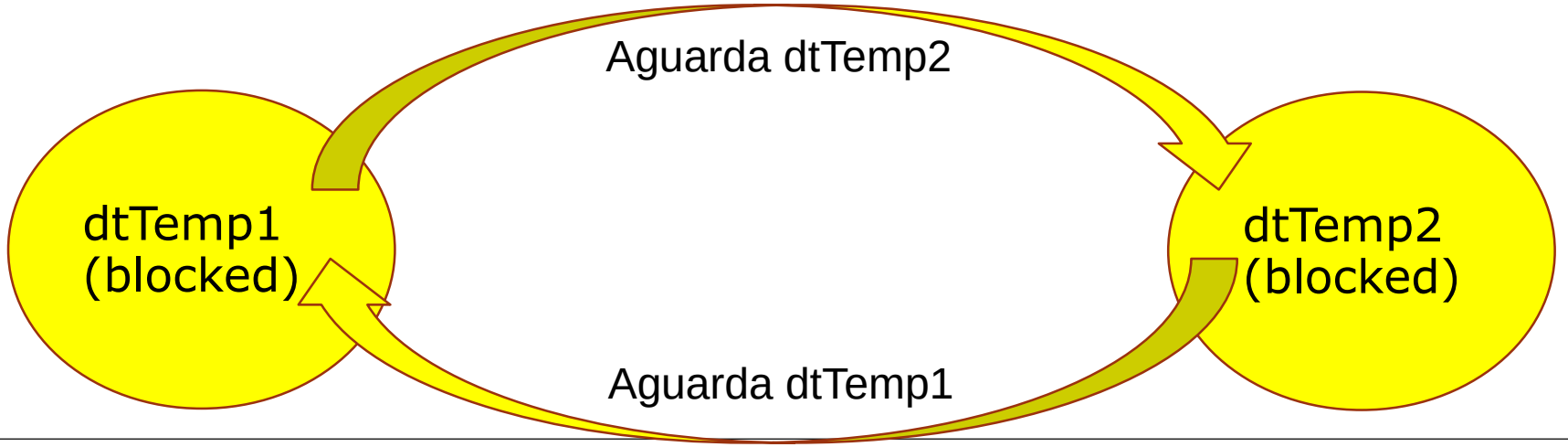
dtTemp2 tenta evocar o método descurva_se_a de jose, e é suspensa

Synchronized-deadlock

- O uso de synchronized, no entanto, pode levar a situações de **deadlock**

➔ Exemplo

joao.descurva_se_a(jose)	jose.descurva_se_a(joao)
synchronized jose.curva_se_a(joao	synchronized joao.curva_se_a(jose)
dtTemp1.run	dtTemp2.run



Synchronized

- O sincronismo de métodos se baseia em **locks (travas)**. Cada **instância de objeto** possui um lock implícito associado, o qual **impede** que um método synchronized seja executado **mais que uma vez** (alternadamente) **por diferentes threads**

- Note que os **locks são associados a instâncias de classes**, e não a classes, **um mesmo método syncronizado pode ser executado mais de uma vez, mas cada execução se refere a uma instância diferente**

- Métodos synchronized são **o recurso mais simples e direto** de controle de concorrência do Java, no entanto, **há outros**

Synchronized

- O sincronismo de métodos se baseia em **locks (travas)**. Cada **ins** o qual **im** **is que um**
- No **sses**, e não **de ser exe efere a um**
- Mé **reto** de cor

Quando uma classe usa o recurso synchronized de maneira que Threads possam concorrer por instâncias desta classe, dizemos que esta classe é **"Thread safe"**, isto é, instâncias dela podem ser compartilhadas por várias threads sem riscos de concorrência.