Programação Concorrente

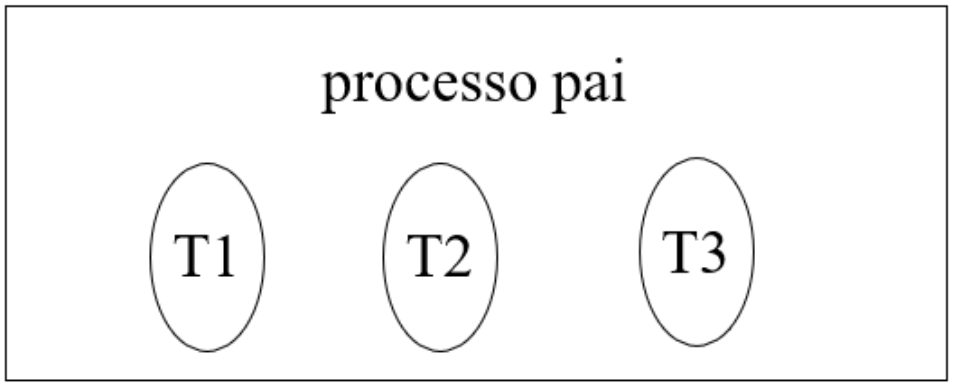
Ricardo Costa – rcosta@estg.ipp.pt

Programação Concorrente

Definição:

- "Thread" – sequência de execução independente
- As várias threads de um processo partilham o mesmo espaço de endereçamento que o processo (pai) que lhe deu origem.

4 processos em ***execução simultânea***



cria 3 threads

Programação Concorrente

Criação de Threads em Java

Hipótese 1:

subclasse de Thread

```
public class MinhaThread_1 extends Thread {

  public MinhaThread_1(){
    super(); //construtor da superclasse
    start();
  }
```

*método definido na classe Thread, invoca o método **run()***

Programação Concorrente

Ex.lo

```
public void run(){
        while (true) {
            ....
            if (isInterrupted() )
                break;
        }
    }// run
} //classe MinhaThread_1
        public class Teste{
                public static void main (String [] args){
                        MinhaThread_1 T1 = new MinhaThread_1();
                }
        }
```

define o código a ser executado pela Thread

Programação Concorrente

Hipótese 2:

```
public class MinhaThread_2 extends Thread {
  public void run () {
        ...
  }
}                                   ? onde está a chamada ao super() ?
public class Teste{
  public static void main (String [] args){
        MinhaThread_2 Ta, Tb;
        Ta =  new MinhaThread_2();
        Tb =  new MinhaThread_2();
        Ta.start();
        Tb.start();
  }}
```

← *iniciar a execução da thread na classe Teste:*

Programação Concorrente

Se quisermos aceder a variáveis do processo principal?

- Passamos como parâmetros as referências dessas variáveis(objetos)

```
public class MinhaThread extends Thread {
    ObjectoPartilhado O;
    public MinhaThread ( ObjectoPartilhado o ){
        super();
        O = o ;
        start();
    }
    public void run(){
        ...
        O....
    }
}
```

endereço do objecto partilhado

aceder ao objecto partilhado

Programação Concorrente

Hipótese 3: *- se a classe já for subclasse de outra classe?*

```
public class MinhaThread_3 implements Runable {
    public void run(){
      ...
    }
}
public class Teste{
  public static void main (String [] args){
        MinhaThread_3 Tc;
        Tc = new MinhaThread_3();
        Thread T = new Thread (Tc);
        T.start();
  }
}
```

classe que implementa o método run()

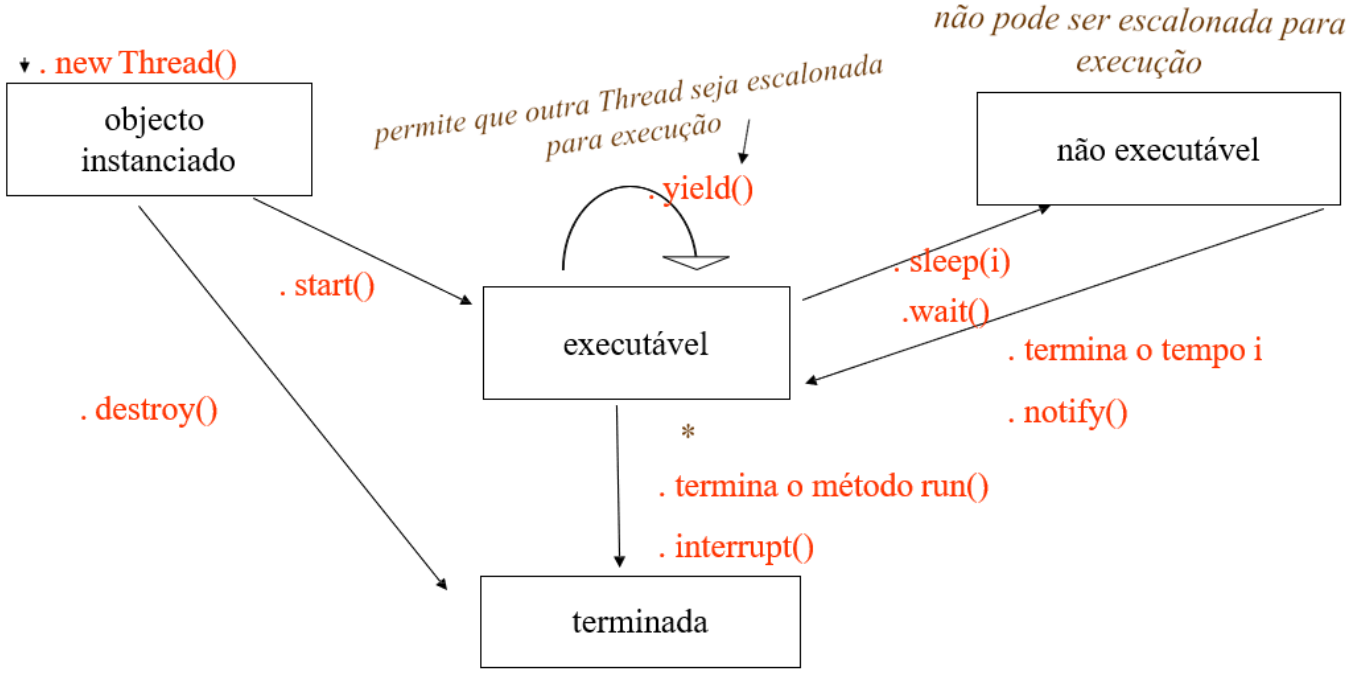
"runable object"

iniciar a execução

```
public interface Runable {
    public abstract void run();
}
```

Programação Concorrente

Diagrama de estados possíveis para uma Thread



* *uma Thread no estado executável, não está necessariamente em execução, apenas pode ser escalonada para execução*

Programação Concorrente

Sincronização de Threads

. *mecanismo baseado no conceito de monitor*

Existe,

um lock associado a cada objecto — *lock de objecto*

e

um lock associado a cada classe. — *lock de classe*

A instrução

referência para o objecto ←

synchronized (expressão)

{ instruções }

Programação Concorrente

Sincronização de threads

- a) após calcular a referência para o objecto, mas antes de executar o corpo de instruções:
  - adquire o lock associado ao objecto,
    - caso este não pertença já a alguma outra thread
  - executa o corpo de instruções
- b) Após executar o corpo de instruções
  - liberta o lock
    - se a execução do bloco de instruções termina anormalmente i. é, falha a meio, o lock é libertado

Programação Concorrente

Sincronização de threads

- Notas:
  - 1. Um método pode ser declarado como synchronized:
    - comporta-se como se estivesse contido numa instrução synchronized
  - 2. O facto de uma Thread adquirir o lock associado a um objecto, não impede que outras Threads acedam aos campos do objecto ou possam invocar métodos não sincronizados.
  - 3. Se o método sincronizado é um método de instância:
    - a Thread adquire o lock do objecto (associado a this);
    - todas as Threads que tentem executar esse mesmo método, no mesmo objecto,
    - terão que esperar, competindo pela aquisição do lock.
  - 4. Se o método sincronizado é um método de classe:
    - a Thread adquire o lock da classe;
    - todas as Threads que tentam executar esse método, em qualquer objecto da classe, terão que esperar que o lock seja libertado.

Programação Concorrente

Exemplo 1:

```
public class Exemplo{
   private int x;
   private  static int s;
   public synchronized void M1()
  {  x++;  }
   public static synchronized void M2()
   {  s++;}
}
```

É equivalente a ...

Programação Concorrente

```
public class Exemplo{
  private int x;
  private static int s;

  public void M1(){
        synchronized (this)
        { x++; }
  }

  public static void M2(){
     try { synchronized (Class.forName( "Exemplo") )
         { s++; }
     }
     catch (classNotFoundException e)  {...}
  }
}
```

Programação Concorrente

Exemplo 2:

Sejam duas Threads, T1 e T2. Supondo que
T1 invoca o método ab e
T2 invoca o método ba no mesmo objecto da classe Exemplo2, quais são os possíveis valores finais para a e b em cada caso, i) e ii) ?

```
i) public class Exemplo2{
      private int a = 1, b = 2;
      public void ab()
      {  a = b; }
       public void ba()
      { b = a; }
  }
```

a=2,b=1| a=2,b=2| a=1,b=1

Implementar e testar a resposta … !!

Programação Concorrente

ii)
```
public class Exemplo3 {
    private int a = 1, b = 2;
    public synchronized void ab()
    {  a = b; }
     public synchronized void ba()
    { b = a; }
}
```

a=2,b=2| a=1,b=1

Programação Concorrente

E nos casos iii, iv e v quais são os outputs possíveis, supondo que T1 invoca o método M1 e T2 invoca o método M2 no mesmo objeto da classe?

iii)
```
public class Exemplo4{
    private int a = 1, b = 2;
    public void M1()
    {  a = 3; b = 4; }
     public void M2()
    { System.out.println ("a=" + a + "b=" + b); }
}
```

a=3,b=4| a=3,b=2| a=1,b=2| a=1,b=4

Programação Concorrente

iv)
```
public class Exemplo5{
    private int a = 1, b = 2;
    public synchronized void M1()
    {a = 3; b = 4; }
     public void M2()
    { System.out.println ("a=" + a + "b=" + b); }
}
```

a=3,b=4| a=3,b=2| a=1,b=2| a=1,b=4

Programação Concorrente

v)
```
public class Exemplo6{
    private int a = 1, b = 2;
    public synchronized void M1()
    {a = 3; b = 4; }
    public synchronized void M2()
    { System.out.println ("a=" + a + "b=" + b); }
}
```

a=3,b=4| a=1,b=2

Programação Concorrente

"Multithreaded servers"

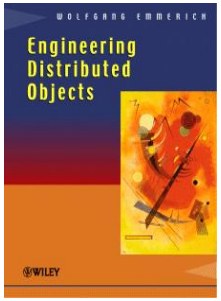
- E se quisermos que um servidor atenda vários clientes em simultâneo?
  - Em situações em que o servidor faça operações de input / output como os servidores de bases de dados ou de ficheiros, servir vários clientes em simultâneo pode melhorar significativamente o seu desempenho.
  - Isso pode ser feito, por exemplo, criando uma thread para "servir" cada cliente.
  - O mesmo é válido para o cliente: é possível melhorar o desempenho de alguns processos clientes criando várias threads para distribuir as tarefas.

Programação Concorrente

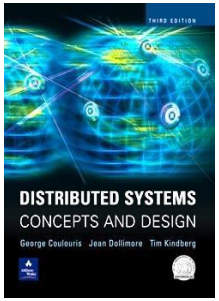
“Multithreaded servers”

- Arquitecturas possíveis:
  - 1 - Uma thread por pedido (thread-per-request)
    - É criada uma thread para cada pedido do cliente.
  - Uma thread por ligação (thread-per-connection)
    - É criada um thread por cada cliente que se liga.
  - Uma thread por objecto (thread-per-object)
    - A cada objecto remoto é associada uma thread.
  - Thread-pool
    - O servidor cria um conjunto inicial de threads.

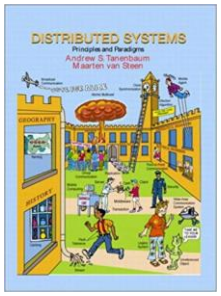
Bibliografia



From: Wolfgang Emmerich
Engineering Distributed Objects
John Wiley & Sons, Ltd 2000



From: Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design
Edition 4 © Addison-Wesley 2005



From: Andrew S., Tanembaum and Van Steen, Maarten
Distributed Systems: Principles and Paradigms
Edition 2 © Pearson 2013

Questões?