

# Linguagem de Programação Orientada a Objetos 2

Threads – Influenciando o Scheduler

Prof. Tales Viegas

<https://fb.com/ProfessorTalesViegas>

# Thread Scheduler

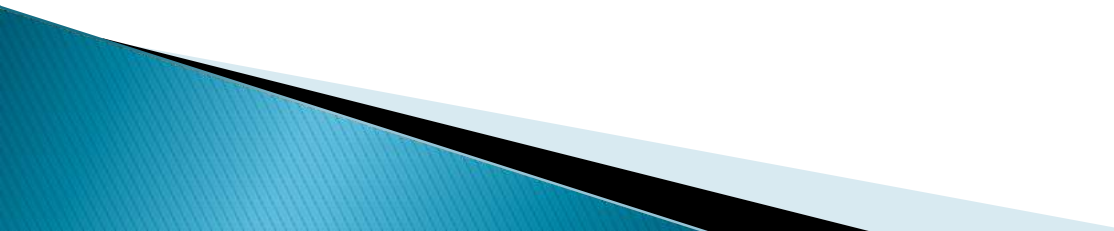
- ▶ É possível influenciar o comportamento através do métodos :
  - `sleep (long millis)`
  - `yield()`
  - `join()`
  - `setPriority(int newPriority)`
- ▶ Influência não é controle !!!!

# Sleep/Suspensão

- ▶ Usado para:
  - desacelerar um segmento
  - dar oportunidade para outros segmentos
- ▶ Pode lançar uma exceção verificada `InterruptedException`.


# Exemplo5

```
public class Exemplo5 extends Thread{
    public void run() {
        for (int i=1; i<4; i++){
            System.out.println("Thread " +
                               Thread.currentThread().getName());
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException ie){
                System.out.println(ie);
            }
        }
    }
}
```



# Testa Exemplo5

```
public class Exemplo5Main{  
    public static void main(String args[]){  
  
        Exemplo5 primeira = new Exemplo5();  
        primeira.setName("Primeira");  
  
        Exemplo5 segunda = new Exemplo5();  
        segunda.setName("Segunda");  
  
        Exemplo5 terceira = new Exemplo5();  
        terceira.setName("Terceira");  
  
        primeira.start();  
        segunda.start();  
        terceira.start();  
    }  
}
```



# Prioridades

- ▶ Os segmentos são executados com níveis de prioridade que variam de 1 a 10.
  - Thread.MIN\_PRIORITY (1)
  - Thread.NORM\_PRIORITY (5) → default
  - Thread.MAX\_PRIORITY (10)
- ▶ É possível controlar a prioridade através dos métodos:
  - setPriority
  - getPriority

# Prioridades

- ▶ O segmento que estiver sendo executado terá prioridade maior ou igual à prioridade mais alta dos segmentos do pool.
- ▶ A especificação da linguagem não dita como será o processo de escalonamento das *threads* e portanto, não determina como esse parâmetro influência na decisão do escalonador.

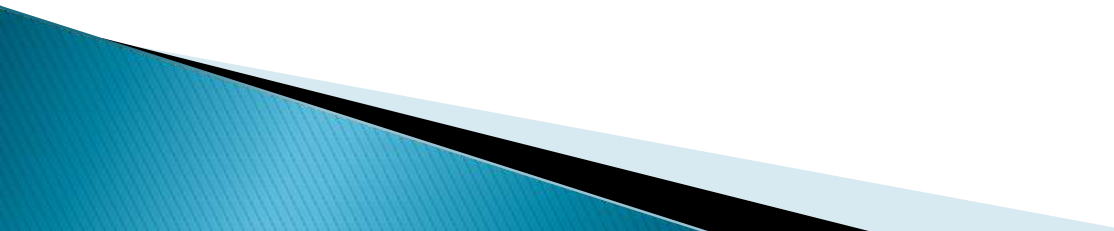
# Prioridades

- ▶ Geralmente (não é garantido!) o escalonador escolhe a *thread* que possui a maior prioridade.
- ▶ Caso haja mais de uma com a mesma prioridade, ele irá escolher uma, não necessariamente a que está esperando há mais tempo.



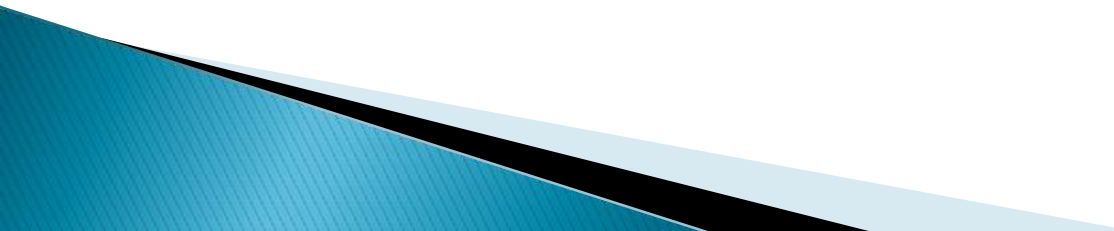
# Exemplo6

```
public class Exemplo6 extends Thread {  
  
    Exemplo6(String nome){  
        super(nome);  
    }  
  
    public void run() {  
        for (int i=1; i<11; i++){  
            System.out.println("Thread "  
                                + Thread.currentThread().getName());  
        }  
    }  
}
```



# Testa Exemplo6

```
public class Exemplo6Main{  
  
    public static void main(String args[]) {  
        Exemplo6 primeira = new Exemplo6("Primeira");  
        Exemplo6 segunda = new Exemplo6("Segunda");  
        Exemplo6 terceira = new Exemplo6("Terceira");  
  
        terceira.setPriority(Thread.MAX_PRIORITY);  
        primeira.start();  
        segunda.start();  
        terceira.start();  
    }  
}
```

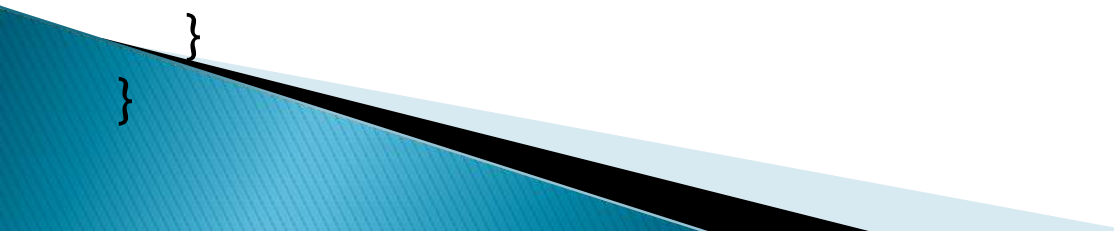


# Yield

- ▶ O método `yield()` retorna o segmento para o estado executável a fim de permitir que outros segmentos com a mesma prioridade tenham a sua oportunidade de serem processados.

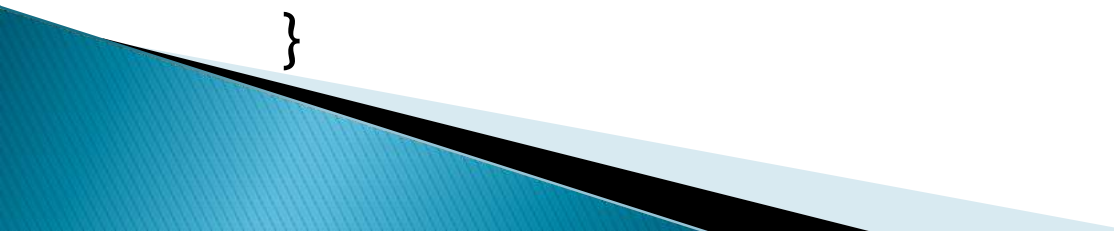
# Exemplo7

```
public class Exemplo7 extends Thread {  
  
    private int countDown = 5;  
  
    public Exemplo7(String nome){  
        super(nome);  
    }  
  
    public void run() {  
        while (true) {  
            System.out.println(this.getName() + " - " + countDown);  
            if (--countDown == 0)  
                return;  
            yield();  
        }  
    }  
}
```

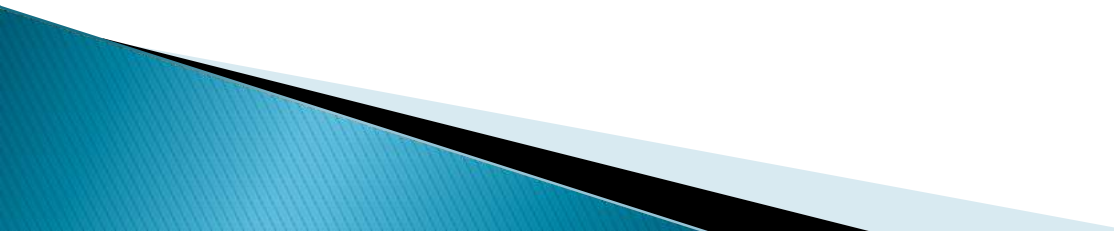


# Testa Exemplo 7

```
public class Exemplo7Main{  
  
    public static void main(String[] args) {  
  
        Thread t1 = new Exemplo7("Thread 1");  
        Thread t2 = new Exemplo7("Thread 2");  
        Thread t3 = new Exemplo7("Thread 3");  
        t1.start();  
        t2.start();  
        t3.start();  
    }  
}
```

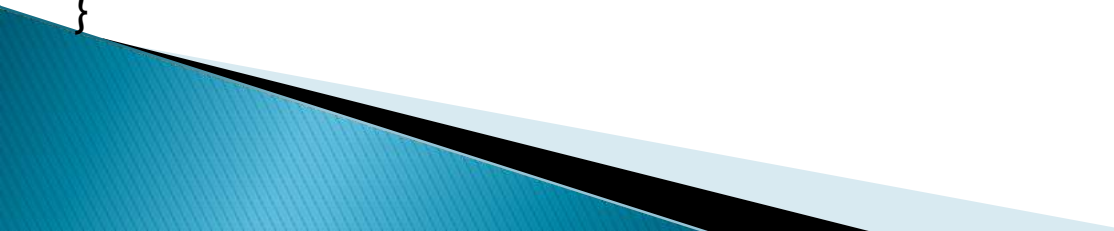


# Join – Espera por uma Thread

- ▶ Se houver um segmento B que não possa executar sua tarefa até que o segmento A tenha sido finalizado então deve-se acrescentar o segmento B ao A.
  - ▶ Assim, o segmento B só poderá ser executado após o segmento A.
  - ▶ O método `join()` é usado para esperar a Thread acabar.
- 

# Exemplo8

```
public class Exemplo8 extends Thread{  
    private String texto;  
    private int repeticoes;  
  
    public Exemplo8(String texto, int repeticoes){  
        this.texto = texto;  
        this.repeticoes = repeticoes;  
    }  
  
    public void run() {  
        for (int i=0; i< this.repeticoes; i++){  
            System.out.println(this.texto);  
        }  
    }  
}
```



# Testa Exemplo8

```
public class Exemplo8Main{

    public static void main(String[] args) {
        Exemplo8 t1 = new Exemplo8("Tales", 20);
        Exemplo8 t2 = new Exemplo8("É o melhor", 20);
        Exemplo8 t3 = new Exemplo8("E mais convencido", 20);

        t1.start();
        t2.start();
        try{
            // Aguarda a t1 acabar
            t1.join();
            System.out.println("Thread 1 acabou!");
            t3.start(); // Inicia a t3
            t2.join(); // Aguarda a t2 acabar
            System.out.println("Thread 2 acabou!");
        } catch (InterruptedException ie){
            ie.printStackTrace();
        }
    }
}
```

