




Programação Thread

Anthony Ferreira La Marca

anthony@computacao.cua.ufmt.br



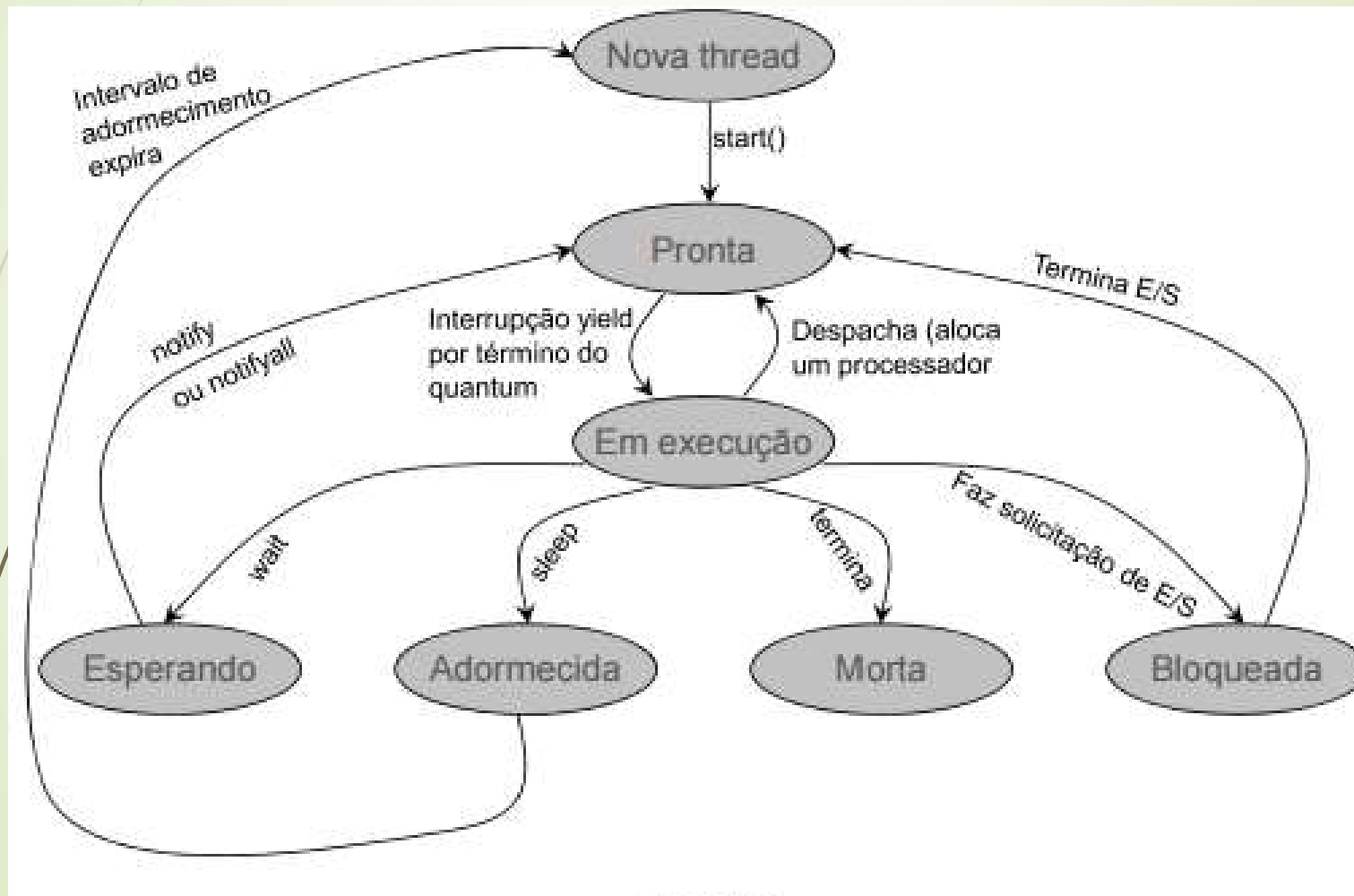
Estados de uma Thread Java

- 
- O diagrama seguinte mostra os estados nos quais uma thread Java pode estar e alguns métodos que podem ser usados para mudar de um estado para outro.

Ciclo de Vida de Java Threads



Diagrama de Estado

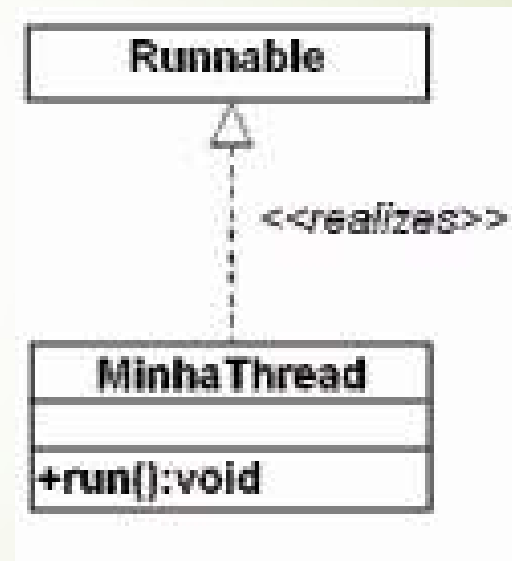
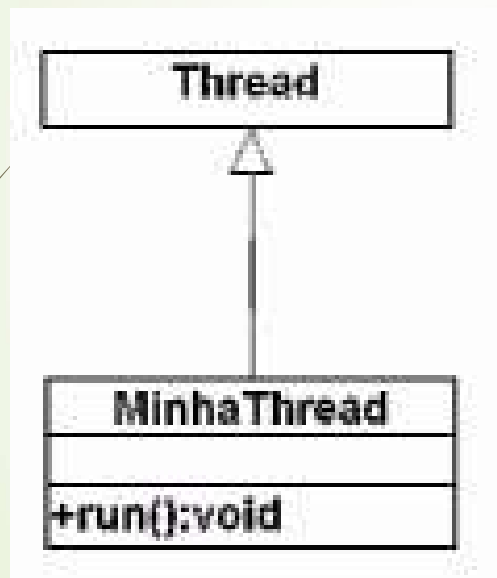




Thread

- Em java usamos a classe Thread do pacote java.lang
- Há duas maneiras de implementar threads
 - Extends threads
 - Implements runnable

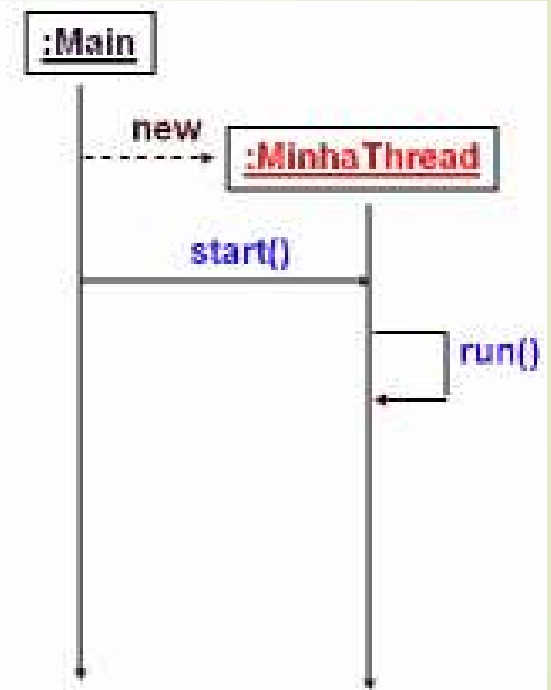
Thread



Thread

```
public class MinhaThread extends Thread {  
    public void run() {  
        System.out.println("Ola! :D");  
    }  
}
```

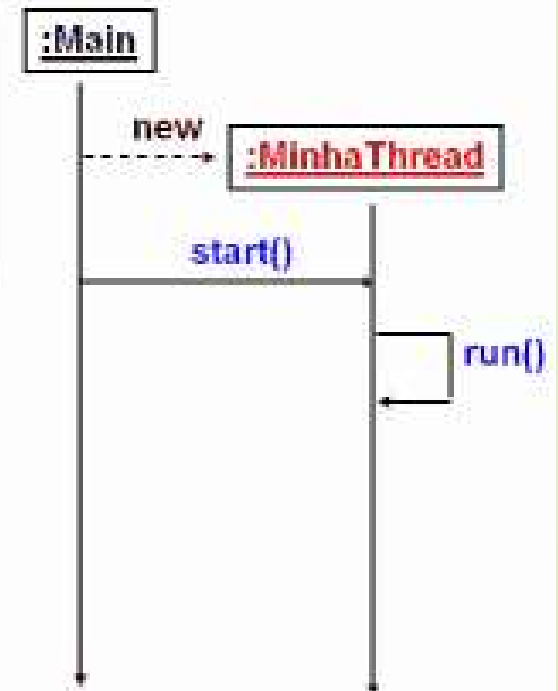
```
public class Main {  
    public static void main(String[] args) {  
        MinhaThread t = new MinhaThread();  
        t.start();  
    }  
}
```




Thread

```
public class MinhaThread implements Runnable {  
    public void run() {  
        System.out.println("ola! ");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        MinhaThread t = new MinhaThread();  
        Thread t2 = new Thread(t);  
        t2.start();  
    }  
}
```





```
public class Programa implements Runnable {

    private int id;
    // colocar getter e setter pro atributo id

    public void run () {
        for (int i = 0; i < 10000; i++) {
            System.out.println("Programa " + id + " valor: " + i);
        }
    }
}
```

```
public class Teste {
    public static void main(String[] args) {

        Programa p1 = new Programa();
        p1.setId(1);

        Thread t1 = new Thread(p1);
        t1.start();

        Programa p2 = new Programa();
        p2.setId(2);

        Thread t2 = new Thread(p2);
        t2.start();

    }
}
```



Thread

- Se rodarmos esse programa qual será a saída?
- Na verdade não saberemos. Rode ele várias vezes e observe
- Em cada execução a saída é um pouco diferente
- Quem faz a troca de contexto?
- Em sistemas multi-core há a necessidade da troca de contexto devido a quantidade de processos/thread serem muito maiores que a quantidade de núcleos



Métodos de Gerenciamento de Threads

- `Thread(ThreadGroup group, Runnable target, String name)`
- `setPriority(int newPriority)`
- `getPriority()`
- `run()`



Métodos de Gerenciamento de Threads

- `Void start()`
- `Static void sleep(int millisecs)`
- `Static void yield()`
- `Void destroy()`
- `Boolean isAlive()`



Métodos de Gerenciamento de Threads

- `Boolean isinterrupted()`
- `Static boolean interrupted()`
- `Void Stop()`
- `Void Suspend()`
- `Void resume()`



Métodos de Sincronização de Threads

- `wait(long millisecs, int nanosecs)`
- `notify(), notifyAll()`
- `join(int millisecs)`
- `interrupt()`



ThreadGroup

- Cada Thread pertence a um grupo de threads
- Caso aplique uma operação sobre o grupo, todas as threads daquele grupo agiram com base naquela operação
- Se não for definido, toda thread criada pertence ao grupo 'main'



Grupos de Threads

- Criando um grupo de threads
 - `ThreadGroup(String name);`
 - `ThreadGroup(ThreadGroup parent, String name);`
- Criando uma thread dentro de um threadgroup específico
 - `ThreadGroup meu = new ThreadGroup("Grupo A");`
 - `Thread n = new Thread(meu, Runnable, "1A");`



Grupos Threads

- Para saber qual grupo uma thread pertence
 - `getThreadGroup`
- E depois
 - `getName()`
- Exemplo de aplicar uma operação no grupo “meu”
 - `meu.interrupt()`
 - Interrompe todas as threads do grupo
- Veremos com mais detalhes em pool de threads



Prioridade

- Thread.MIN_PRIORITY = 1
- Thread.MAX_PRIORITY = 10
- Thread.NORM_PRIORITY = 5
- Uma thread herda a prioridade da thread que a criou
- A thread de maior prioridade preempta as outras threads de menor prioridade
- Se todas as threads tiverem a mesma prioridade, a CPU é alocada para todos, um de cada vez, em modo round-robin.

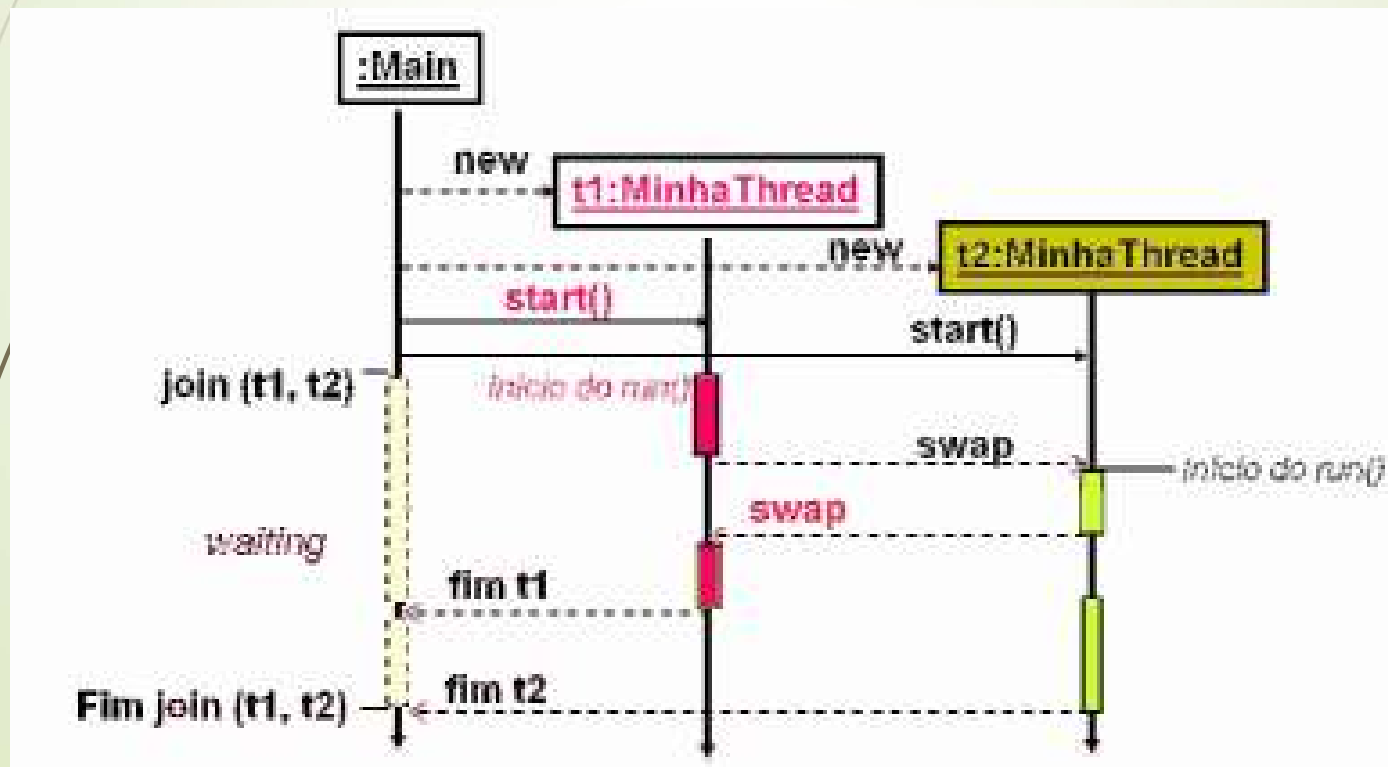


Exercício

- Faça a simulação de um cruzamento controlado por dois semáforos utilizando apenas **sleep** para controlar os tempos verde, amarelo e vermelho.
- Observe que o sleep fará sinaleiros **não sincronizados**

JOIN

- Permite que uma Thread espere pelo término de duas ou mais threads. Exemplo_2_Thread





Exercício

- ▶ Considere um simulador de corridas de F1 que simula a disputa entre dois pilotos: Massa e Hamilton
 - ▶ Cada carro funciona de forma independente
 - ▶ O tempo de cada volta é dado por um valor randômico. O programa deve esperar por este tempo sem fazer nada para então iniciar a próxima volta
 - ▶ Ao final da corrida (quando os dois completaram 5 voltas). O simulador mostra o tempo acumulado para cada um dos pilotos e aponta o vencedor ou empate



Exercício

- Que comandos da linguagem JAVA você usaria para resolver cada um dos itens acima?
- Observação
 - Mesmo dando maior prioridade à thread do Massa ele pode chegar após o Hamilton
 - Pois o tempo de cada uma das voltas é atribuído aleatoriamente
 - Logo, o tempo total da corrida do Massa pode ser maior ao do Hamilton, mesmo que a thread do Massa tenha acabado antes



Pool de Thread próxima aula