Sistemas Operacionais
Aula 05 – Parte 03

Cléver Ricardo Guareis de Farias
DCM/FFCLRP/USP

Conteúdo Programático

- Threads em Java

Threads em Java

- São gerenciados pela máquina virtual Java e não por uma biblioteca no nível do usuário ou núcleo
- Mapeamento de threads Java para threads de núcleo depende de cada implementação da máquina virtual Java

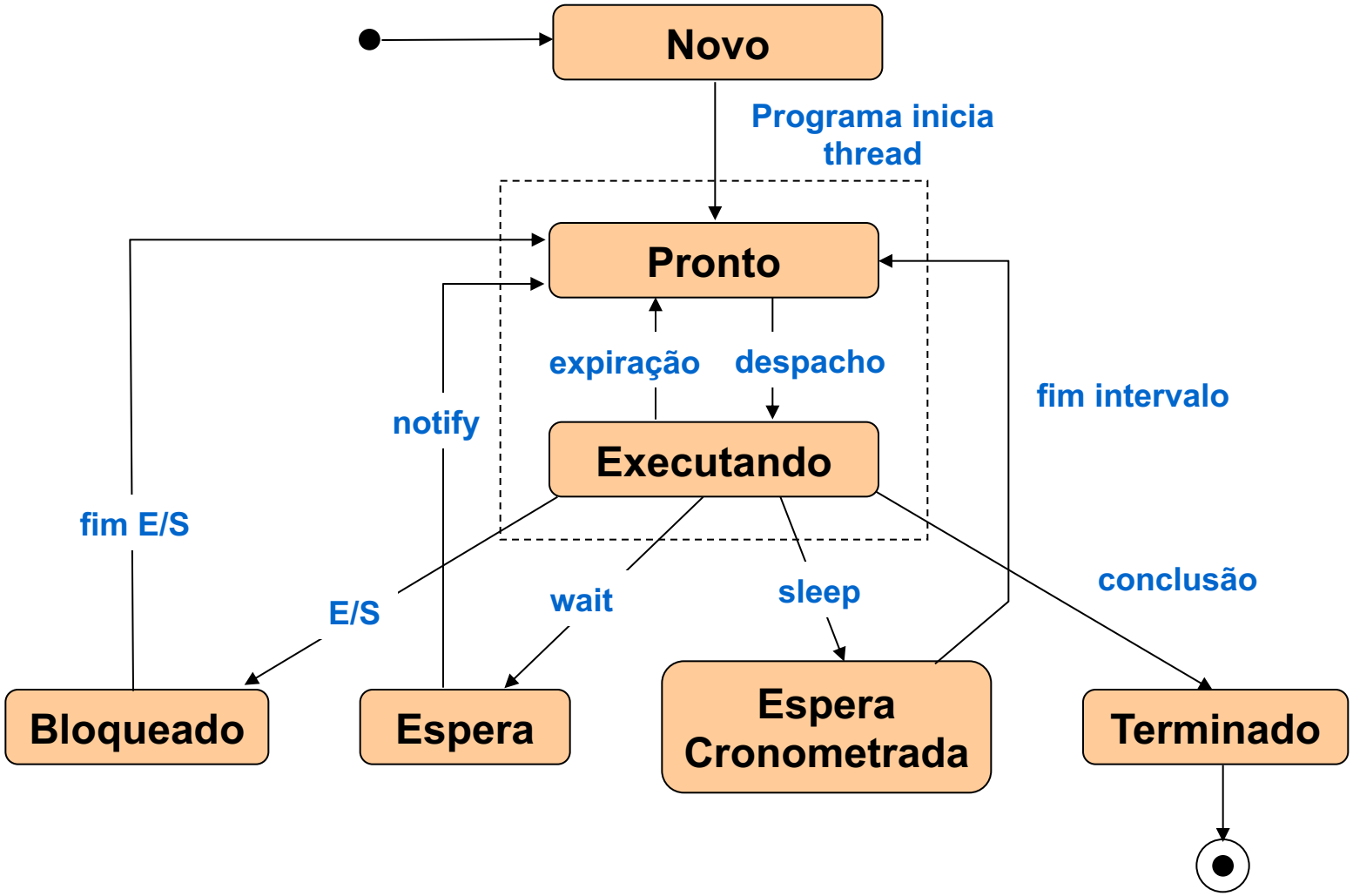
Estados do thread (1)

- Estados de um thread:
  - quando um thread é criado, este inicia seu ciclo de vida no estado Novo
  - thread permanece no estado Novo até que este inicie sua execução (método start), o que o coloca no estado executável
  - estado executável consiste de dois (sub)-estados, o estado Pronto e o estado Executando

Estados do thread (2)

- thread pode entrar no estado de Espera se este tiver que esperar por outro thread realizar uma tarefa
- thread pode entrar no estado de Espera Cronometrada por um intervalo especificado de tempo
- thread entra no estado Terminado quando este finaliza sua execução

Estados do thread (3)



Criação de threads (1)

- Duas possibilidades para a criação de um novo thread:
  - definir uma classe derivada da classe Thread
    - classe deve possuir um método chamado run, o qual define o que deve ser executado quando um thread é inicializado (método start)
    - parâmetros não podem ser passados através do método run, mas somente através do construtor do thread

Criação de threads (2)

```
public class Somador extends Thread{
    private int limiteSomatoria;
    private Soma somaLimite;
    public Somador(int limite, Soma valor){
            limiteSomatoria = limite;
            somaLimite = valor;
    }
    public void run() {
        int soma = 0;
        for (int i = 0; i <= limiteSomatoria; i++)
                soma+= i;
        somaLimite.setSoma(soma);
    }
}
```

Criação de threads (3)

```
public class SomaNroPositivo {
  public static void main(String[] args) {
        if (args.length != 1) {
              System.err.println("Nro de parametros deve ser 1");
              System.exit(0);      }
        int limite = Integer.parseInt(args[0]);
        Soma nro = new Soma();
        Somador thrd = new Somador(limite, nro);
        thrd.start();
...
        System.out.println("Soma de 0 ate " + limite + " = " + nro.getSoma());
   }
}
```

Criação de threads (4)

- definir uma classe que implementa a interface Runnable
  - classe também deve possuir um método run
  - instância da classe deve ser passada como parâmetro quando da criação de um thread (instância da classe Thread)

Criação de threads (5)

```
public class Somador implements Runnable{
    private int limiteSomatoria;
    private Soma somaLimite;
    public Somador(int limite, Soma valorInicial) {
            limiteSomatoria = limite;
            somaLimite = valorInicial;
    }
    public void run() {
        int soma = 0;
        for (int i = 0; i <= limite; i++)
                soma+= i;
        somaLimite.setSoma(soma);
    }
}
```

Criação de threads (6)

```
public class SomaNroPositivo {
  public static void main(String[] args) {
        if (args.length != 1) {
              System.err.println("Nro de parametros deve ser 1");
              System.exit(0);
       }
        int limite = Integer.parseInt(args[0]);
        Soma somaLimite = new Soma();
        Thread thrd = new Thread(new Somador(limite, somaLimite));
         thrd.start();
         ...
        System.out.println("Soma de 0 ate " + limite + " = " + somaLimite.getSoma());
  }
}
```

Discussão (3)

- Por que Java admite duas técnicas para a criação de threads? Qual técnica é mais apropriada?

Unindo threads (1)

- De maneira geral um thread é executado independentemente do “thread” que o cria
- Quando um thread deseja esperar pelo término da execução de outro thread criado, o método join deve ser utilizado
  - usado em situações onde um thread (thread que cria outro thread) só pode continuar depois do término de outro thread (thread criado)

Unindo threads (2)

- public final void join() throws InterruptedException
  - thread que invocou o método é bloqueado à espera do término da execução do thread alvo da requisição
  - exceção é lançada se um outro thread interromper a execução do thread atual

Unindo threads (3)

```
public class SomaNroPositivo {
  public static void main(String[] args) {
        ...
        Soma nro = new Soma();
        int limite = Integer.parseInt(args[0]);
        Thread thrd = new Thread(new Somador(limite, nro));
        thrd.start();
        try {
          thrd.join();
        } catch (InterruptedException ie) {}
       System.out.println("Soma de 0 ate " + limite + " = " + nro.getSoma());
  }
}
```

Interrompendo threads (1)

- Um thread pode ser interrompido por outro thread por meio do método interrupt
  - interrupção adiada
- Interrupção adiada funciona por meio da verificação periódica de um thread alvo da interrupção se este deve ser terminado

Interrompendo threads (2)

- public void interrupt() throws SecurityException
  - thread solicita a interrupção adiada definindo o status de interrupção
  - thread que estiver sendo interrompido deve verificar status e então terminar sua execução
  - se thread que solicitar a interrupção não puder interromper o thread em execução, uma exceção é gerada

Interrompendo threads (3)

- public static void sleep(long millis) throws InterruptedException
  - método que suspende a execução do thread pelo intervalo de tempo millis
- public static void yield()
  - método que interrompe a execução do thread e cede o processador para outro thread
  - só pode ser invocado pelo thread em execução

Interrompendo threads (4)

```
public class InterruptibleThread extends Thread{
    public void run(){
        while (true){
            // realiza tarefa
            ...
            // verifica se ha interrupcao
            try{
                sleep(sleepTime);
            } catch (InterruptedException ie){
                ...
            }
        }
    }
}
```

Interrompendo threads (5)

```
public static void main(String[] args) {
        ...
      Thread thrd = new Thread(new InterruptibleThread(Integer.parseInt(args[0])));
      thrd.start();
        ...
        try {
            thrd.interrupt();
         } catch (SecurityException se) { ...  }
        try {
             thrd.join();
        } catch (InterruptedException ie) { ... }
}
```

Interrompendo threads (6)

```
public class InterruptibleThread extends Thread{
    public void run(){
        while (true){
            // realiza tarefa
            ...
            // verifica se ha interrupcao
            if (Thread.currentThread().isInterrupted()){
                System.out.println("Thread interrompida!!!");
                break;
            }
        }
    }
}
```

Grupos de threads (1)

- Em algumas situações faz-se necessário monitorar um conjunto de threads
- Threads podem ser agrupados sucessivamente em grupos por meio da classe ThreadGroup
  - grupo suporta algumas operações sobre todos os threads do grupo

Grupos de threads (2)

- public ThreadGroup(String name)
  - cria um grupo com o nome especificado por name
- public Thread(ThreadGroup group, Runnable target)
  - cria uma thread e associa a thread ao grupo group

Grupos de threads (3)

```
...
ThreadGroup grupo = new ThreadGroup("Somador");
 for (int i = 0; i < args.length; i++){
     Thread thrd = new Thread(grupo, new Somador(...));
     thrd.start();
  }
while (grupo.activeCount()> 0)
   try{
       Thread.sleep(500);
     } catch (InterruptedException ie){
       System.out.println("Erro: "+ie);
     }
...
```

Exercício

- Faça um aplicativo Java utilizando threads que receba como parâmetro, via linha de comando, um conjunto de números inteiros positivos e verifique, usando um thread separado para cada número, se cada um destes é ou não primo.