Sistemas Operacionais

Aula 05 - Threads em Java: Introdução

UFOP

Prof. Samuel Souza Brito

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Aplicadas – ICEA
Departamento de Computação e Sistemas – DECSI

João Monlevade-MG
2024/2

Introdução

- As threads são um modelo fundamental de execução em Java.
  - Rico conjunto de aspectos da criação e gerenciamento de threads.
- Todos os programas Java incluem pelo menos uma única thread.
  - Até mesmo um programa simples.
  - Método `main()` é executado como uma thread na JVM.

Introdução

- Quando uma aplicação Java é executada:
  - A JVM cria um objeto do tipo *Thread* cuja tarefa a ser executada é descrita pelo método `main()`.
  - *Thread* é iniciada automaticamente.
  - Os comandos descritos pelo método `main()` são executados sequencialmente até que o método termine e a thread se encerre.

Criação de Threads

- Existem duas formas para criação de threads em Java:
  1. Estendendo a classe *Thread* e instanciando um objeto desta nova classe.
  2. Implementando a interface Runnable, e passando um objeto desta nova classe como argumento do construtor da classe *Thread*.
- Nos dois casos, a tarefa a ser executada pela thread deverá ser descrita pelo método `run()`.
  - Código contido nesse método é o que é executado como uma thread separada.

Estendendo a classe Thread

```
public class ThreadExtends extends Thread {
    private String nome;

    public ThreadExtends(String n) {
        nome = n;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i + " - " + nome);
        }
        System.out.println(nome + " finalizou!");
    }
}
```

Estendendo a classe Thread

Para iniciar uma thread é necessário instanciar um objeto da nova classe e executar o método `start()`:

```
public class Main {

    public static void main(String[] args) {
        ThreadExtends t1 = new ThreadExtends("Thread 1");
        ThreadExtends t2 = new ThreadExtends("Thread 2");

        t1.start();
        t2.start();
    }
}
```

Interface Runnable

- Limitações ao estender a classe Thread?

Interface Runnable

- Limitações ao estender a classe Thread?
  - Como Java não permite herança múltipla, a necessidade de estender a classe Thread restringe a criação de subclasses a partir de outras classes genéricas.
- Por meio da utilização da interface *Runnable* é possível criar classes que representem um thread sem precisar estender a classe *Thread*.
  - Pode-se, então, estender outras classes.
- A criação de uma nova thread é feita por meio da instanciação de um objeto *Thread* passando como parâmetro um objeto da classe que implementa a interface *Runnable*.

Interface Runnable

```
public class ThreadImplements implements Runnable {
    private String nome;

    public ThreadImplements(String n) {
        nome = n;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i + " - " + nome);
        }
        System.out.println(nome + " finalizou!");
    }
}
```

Interface Runnable

```
public class Main {

    public static void main(String[] args) {
        ThreadImplements t1 = new ThreadImplements("Thread 1");
        ThreadImplements t2 = new ThreadImplements("Thread 2");

        Thread thread1 = new Thread(t1);
        Thread thread2 = new Thread(t2);

        thread1.start();
        thread2.start();
    }
}
```

Criação de Threads

- A criação de um objeto *Thread* não cria especificamente uma nova thread.
  - O método `start()` que realmente cria a nova thread.
- A chamada do método `start()` realiza duas tarefas:
  - Aloca memória e inicializa uma nova thread na JVM.
  - Executa o método `run()`, tornando a thread elegível para ser executada pela JVM.
- Método `run()` não é chamado diretamente!
  - Experimente substituir as chamadas de `start()` no código anterior por `run()`. O que acontece?

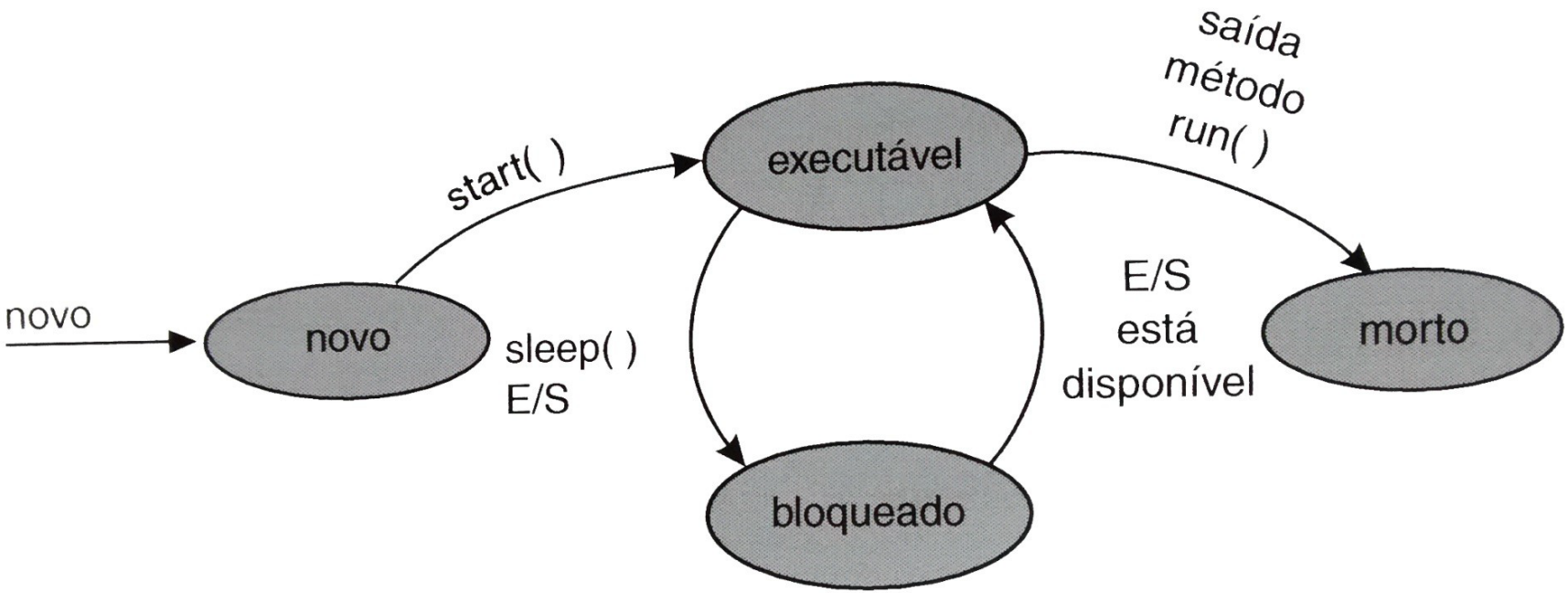
Estados da Thread em Java

- **Novo**:
  - Uma thread está nesse estado quando um objeto *Thread* é instanciado (instrução *new*).
- **Executável**:
  - A chamada do método `start()` aloca memória para a nova thread na JVM e executa o método `run()`.
  - Quando o método `run()` está sendo executado, a thread passa do estado **novo** para o estado **executável**.
  - Uma thread no estado **executável** é elegível para ser executada na JVM.
  - Uma thread sendo executada também está no estado **executável**.

Estados da Thread em Java

- **Bloqueado**:
  - Uma thread torna-se bloqueada se executar uma instrução de bloqueio (realizando E/S, por exemplo) ou se invocar certos métodos de *Thread*, como `sleep()`.
- **Morto**:
  - Uma thread passa para o estado morto quando o método `run()` termina.

Estados da Thread em Java



Exemplo

- Crie um programa em Java que calcule o somatório dos elementos de um vetor utilizando duas threads. Cada thread deve calcular uma parte do somatório e ao final o resultado completo deve ser exibido na tela.

Join

- O que acontece se as operações de algumas threads forem mais rápidas do que as operações de outras threads?
- Voltando ao exemplo anterior...
  - O que acontece quando executamos o código? O resultado do somatório está correto?
    - Execute diversas vezes e analise a saída.

Join

- O que acontece se as operações de algumas threads forem mais rápidas do que as operações de outras threads?
- Voltando ao exemplo anterior...
  - O que acontece quando executamos o código? O resultado do somatório está correto?
    - Execute diversas vezes e analise a saída.
- **Problema de sincronismo!**

Join

- Como calcular e exibir o resultado do somatório somente após o término das duas threads que fazem os cálculos parciais?
- Solução:
  - Usar o método `join()`.
  - Esse método faz com a thread que está executando (*Main*, por exemplo) aguarde por aquela que fez a chamada.
  - Todo o código abaixo da chamada do `join()` só será executado quando a thread que fez essa chamada finalizar.

Join

```
public static void main(String[] args) {
    ThreadImplements t1 = new ThreadImplements("Thread 1");
    ThreadImplements t2 = new ThreadImplements("Thread 2");

    Thread thread1 = new Thread(t1);
    Thread thread2 = new Thread(t2);

    thread1.start();
    thread2.start();

    try {
        thread1.join();
        thread2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("Main está finalizando!");
}
```

Controlando a Execução

- `start()`
  - Inicia a execução da thread.
  - Só pode ser invocado uma vez.
- `join()`:
  - Condiciona a continuação da execução de uma thread ao término de uma outra.
- `sleep(t)`:
  - Faz com que a thread fique suspensa por t milissegundos.

Material Complementar

- Vídeos sobre Java:
  - ⟨https://loiane.training/curso/java-basico⟩
  - ⟨https://loiane.training/curso/estrutura-de-dados⟩
  - ⟨https://loiane.training/curso/java-intermediario⟩
- Vídeos sobre Threads em Java:
  - ⟨https://youtu.be/v5l30QMKv6c⟩
  - ⟨https://youtu.be/oWoU0uTEaA0⟩
  - ⟨https://youtu.be/f8wwWSwi6bo⟩

Exercícios - Entrega via Moodle

1. Crie um programa que calcule a **média aritmética simples** e a **mediana** dos elementos de um vetor. Cada uma dessas operações deve ser executada por uma thread diferente.

2. Crie uma classe `Contador` contendo um atributo chamado `valor`, que é uma variável do tipo `int`. Em seguida, implemente:

   a. um construtor que inicia `valor` com zero;
   b. um método que incrementa `valor` em uma unidade;
   c. um método que decrementa `valor` em uma unidade;
   d. um método *get* que retorne `valor`.

   Crie duas threads que compartilham um mesmo objeto da classe *Contador*. Uma das threads deve executar 100.000 vezes o método que incrementa o contador e a outra thread deve 100.000 vezes o método que o decrementa. Execute o programa várias vezes e analise os resultados, consultando o valor final do contador.

Threads e Desempenho

- Crie um programa que calcule a multiplicação de duas matrizes utilizando threads. Cada célula da matriz resultante deve ser calculada por uma thread. Implemente também uma versão sequencial (única thread) desse algoritmo e compare os tempos de execução.
  - O que podemos interpretar a partir dessa comparação?

Dúvidas?