INSTITUTO MAUÁ DE TECNOLOGIA

Linguagens I

Threads e o Java

Profº. Tiago Sanches da Silva

Introdução a Threads

Introdução

Em várias situações, precisamos "**rodar duas coisas ao mesmo tempo**".

- Imagine um programa que gera um relatório muito grande em PDF. É um processo demorado e, para dar alguma satisfação para o usuário, queremos mostrar uma barra de progresso. Queremos então gerar o PDF e ao mesmo tempo atualizar a barrinha.

- Ou continuar executando partes do código enquanto espera uma entrada de dados, ou conexão.

Introdução

A necessidade de se fazer várias coisas simultaneamente, paralelamente, aparece frequentemente na computação.

Para vários programas distintos, normalmente o próprio sistema operacional gerencia isso através de vários **processos** em paralelo.

Em um programa só (um processo só), se queremos executar coisas em paralelo, normalmente falamos de **Threads**.

Processos

"Um processo é basicamente um programa em execução, sendo constituído do código executável, dos dados referentes ao código, da pilha de execução, do valor do contador de programa (**registrador PC**), do valor do apontador do apontador de pilha (**registrador SP**), dos valores dos demais registradores do hardware, além de um conjunto de outras informações necessárias à execução dos programas."

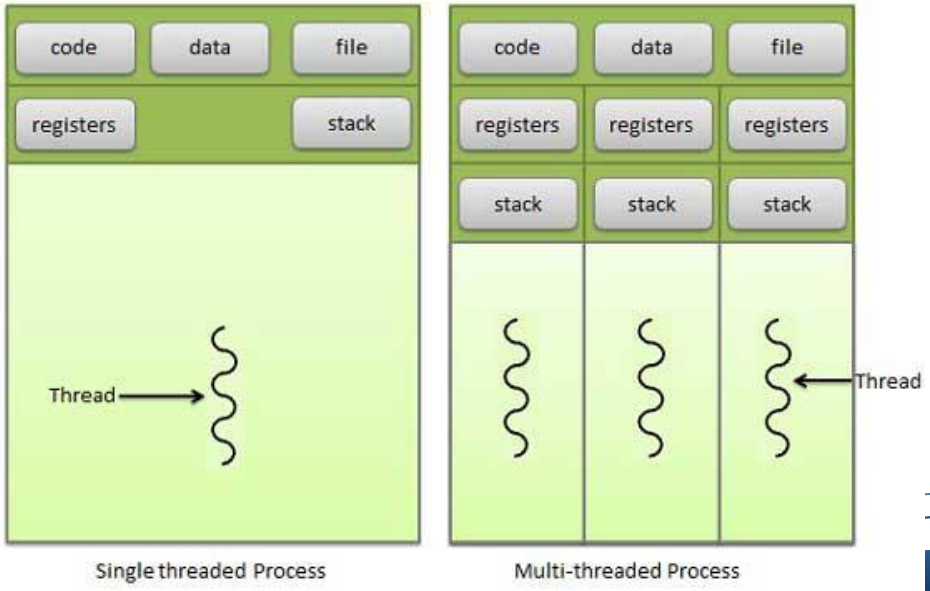
Tanenbaum

Thread

Thread são estruturas de execução pertencentes a um **processo** e assim compartilham os segmentos de código e dados e os recursos alocados ao sistema operacional pelo processo.

A troca de contexto entre as threads exige um esforço bem menor.

Sendo que ainda assim, ocorrerá o escalonamento de processos, pois outros processos poderão estar sendo executado paralelamente ao processo que possui as threads.



Thread no Java

Thread no Java

Em Java, usamos a classe **Thread** do pacote *java.lang* para criarmos linhas de execução paralelas. A classe **Thread** recebe como argumento um objeto com o código que desejamos executar. Por exemplo, no programa de PDF e barra de progresso:

```
public class GeraPDF {
  public void rodar () {
    // lógica para gerar o pdf...
  }
}

public class BarraDeProgresso {
  public void rodar () {
    // mostra barra de progresso e vai atualizando ela...
  }
}
```

... Na verdade o método é o **run**(), rodar é apenas para exemplificar.

Executando threads

E, no método **main**, criamos os objetos e passamos para a classe **Thread**.

O método **start** é responsável por iniciar a execução da **Thread**:

```
public class MeuPrograma {
  public static void main (String[] args) {
      GeraPDF ger
      Thread
      thr
      Bar                                          rogresso();
      Th                                           gresso);
      thre

  }
}
```

A classe **Thread** recebe como argumento um objeto com o código que desejamos executar.

Interface Runnable

Para que o código anterior compile é necessário existir um contrato entre as nossas classes a serem executadas e a classe **Thread.**

Esse contrato existe e é feito pela interface **Runnable:** devemos dizer que nossa classe é "**executável**" e que segue esse contrato.

Na interface **Runnable,** há apenas um método chamado **run**. Basta implementá-lo, "assinar" o contrato e a classe Thread já saberá executar nossa classe.

Interface Runnable

```
public class GeraPDF implements Runnable {
  public void run () {
    // lógica para gerar o pdf...
  }
}

public class BarraDeProgresso implements Runnable {
  public void run () {
    // mostra barra de progresso e vai atualizando ela...
  }
}
```

Interface Runnable

A classe Thread recebe no construtor um objeto que é um **Runnable**, e seu método **start** chama o método **run** da nossa classe.

Repare que a classe **Thread** não sabe qual é o tipo específico da nossa classe; para ela, basta saber que a classe segue o contrato estabelecido e possui o método **run**.

```
GeraPDF gerapdf = new GeraPDF();
Thread threadDoPdf = new Thread(gerapdf);
threadDoPdf.start();
```

Estendendo a classe Thread

A classe **Thread** implementa ***Runnable***. Então, você pode criar uma subclasse dela e reescrever o *run* que, na classe Thread, não faz nada:

```
public class GeraPDF extends Thread {
  public void run () {
    // ...
  }
}
```

E, como nossa classe é uma **Thread**, podemos usar o start diretamente:

```
GeraPDF gera = new GeraPDF();
gera.start();
```

Criando Threads

Resumindo, existem duas formas de criar explicitamente um thread em Java:

- **Implementando** a interface **Runnable**, e passando um objeto desta nova classe como argumento do construtor da classe Thread.
- **Estendendo** a classe **Thread** e instanciando um objeto desta nova classe.

Nos dois casos a tarefa a ser executado pelo thread deverá ser descrita pelo método **run().**

Discussão em sala: **Qual usar? Tanto faz?**

Vamos a um exemplo

Contador simples

```
public class Programa1 implements Runnable {

    private int id = 1;

public class Programa2 implements Runnable {

    private int id = 2;

    public void run () {
        for (int i = 0; i < 20; i++) {
            System.out.println("Programa " + id + " valor: " + i);
        }
    }
}

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

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

        Programa p2 = new Programa();
        Thread t2 = new Thread(p2);
        t2.start();
    }
}
```

Discussão em sala

Discuta com o professor sobre os resultados.

As threads foram perfeitamente intercaladas?

Documentação

Oracle – Processos e threads

https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

Oracle – Classe Thread

https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

Um pouco mais de informação

Controlando a Execução da thread

start()

- Inicia a execução do thread (só pode ser invocado uma vez).

yield()

- Faz com que a execução do thread corrente seja imediatamente suspensa, e outro thread seja escalonado.

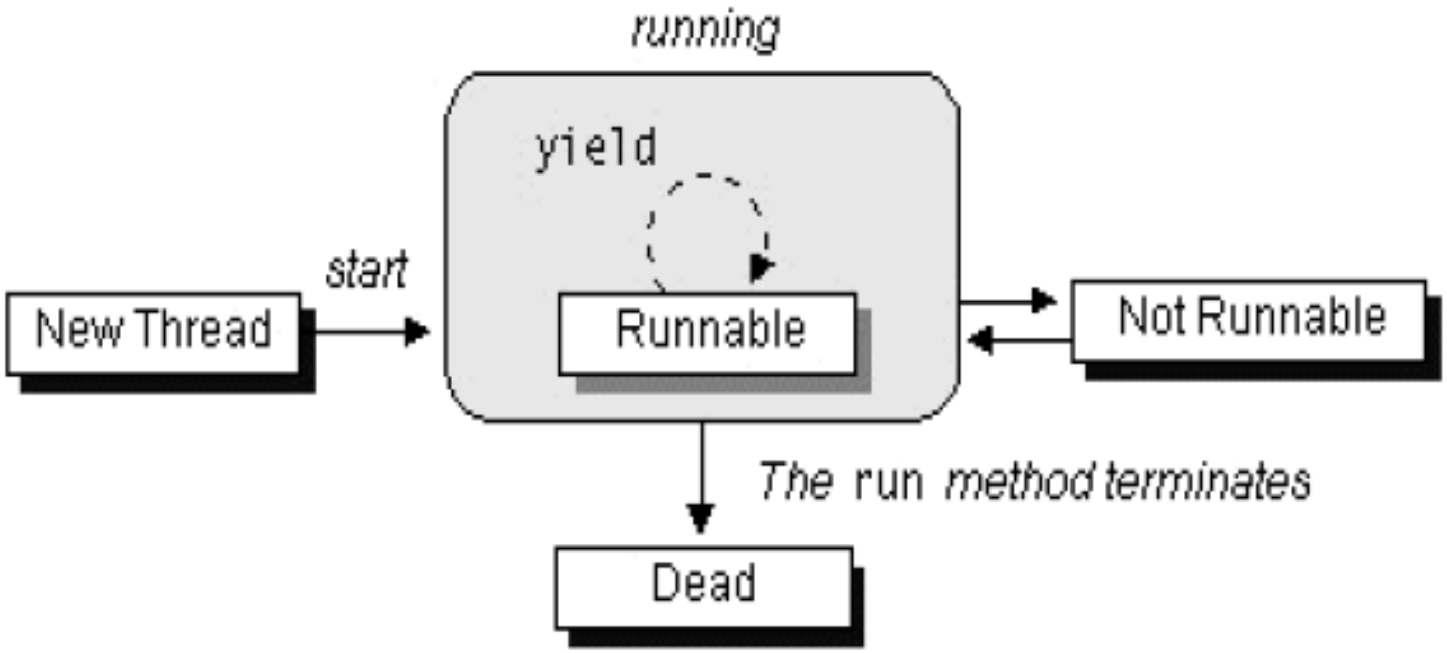
sleep(t)

- Faz com que o thread fique suspenso por t segundos.

wait()

- Faz com que o thread fique suspenso até que seja explicitamente reativado por um outro thread.

Ciclo de vida de uma thread



Levando programação multi-thread a serio

Nesta aula foram apresentados apenas conceitos básicos para trabalhar com threads.

A área de programação paralela é muito ampla e necessita de maior estudo e profundidade.

Para continuar os estudos é de extrema importância que você aprenda mais sobre programação concorrente e gerenciamento de recursos em aplicações concorrentes.

Bom estudo!

Inner Class no Java

Documentação – Inner Class

Oracle
https://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html

Tutorial Points
https://www.tutorialspoint.com/java/java_innerclasses.htm

Perguntas?

Referências

- Oracle
- Caelum
- http://www.inf.puc-rio.br/~inf1621/java2.pdf