

# MULTITHREADING EM JAVA

## PARTE I

Prof. Dr. Rodrigo Palácios

# MULTITHREADING

- Por que utilizar threads?

- Há muitos motivos para se usar threads. Entre eles, podemos citar:
  - Responsividade em Interfaces Gráficas: imagine se o seu navegador web parasse de carregar uma página só porque você clicou no menu “arquivo”;
  - Sistemas multiprocessados: o uso de threads permite que o SO consiga dividir as tarefas entre todos os processadores disponíveis aumentando, assim, a eficiência do processo;
  - Simplificação na Modelagem de Aplicações: suponha que você precise fazer um programa que simule a interação entre diferentes entidades. Carros em uma estrada, por exemplo. É mais fácil fazer um loop que atualiza todos os carros da simulação ou criar um objeto carro que anda sempre que tiver espaço a frente dele?

# MULTITHREADING

- Fundamentos do uso de várias threads
  - Há dois tipos distintos de multitarefa:
    - Baseada em processos
      - É o recurso que permite que o computador execute dois ou mais programas ao mesmo tempo.
      - Por exemplo, é a multitarefa baseada em processos que nos permite executar o compilador Java ao mesmo tempo em que estamos usando um editor de texto ou navegando na Internet.
    - Baseada em threads
      - A thread é a menor unidade de código que pode ser despachada.
      - Um mesmo programa pode executar duas ou mais tarefas ao mesmo tempo.
      - Por exemplo, um editor de texto pode formatar texto ao mesmo tempo em que está imprimindo, contanto que essas duas ações estejam sendo executadas por duas threads separadas.

# MULTITHREADING

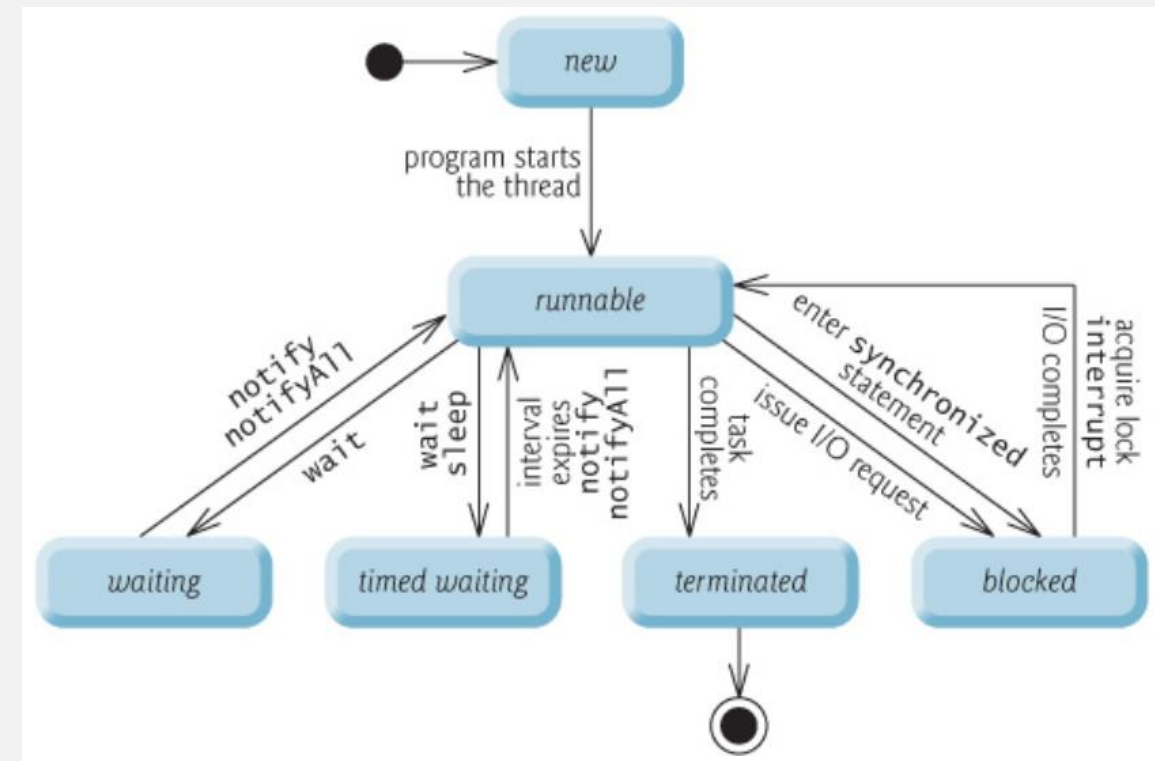
- Fundamentos do uso de várias threads
  - Vantagem importante do uso de várias threads
    - Permite a criação de programas eficientes.
    - É possível utilizar o tempo ocioso que está presente em quase todos os programas.
  - Como você deve saber, a maioria dos dispositivos de I/O, sejam portas de rede, unidades de disco ou o teclado, é muito mais lenta do que a CPU. Logo, com frequência o programa gasta grande parte de seu tempo de execução esperando receber ou enviar informações. Usando várias threads, o programa pode executar outra tarefa durante seu tempo ocioso.
  - Por exemplo, enquanto uma parte do programa está enviando um arquivo pela Internet, outra parte pode estar lendo entradas no teclado e ainda outra pode estar armazenando em buffer o próximo bloco de dados a ser enviado.

# MULTITHREADING

- Fundamentos do uso de várias threads
  - Nos últimos anos, os sistemas multiprocessadores e multicore se tornaram comum apesar de os sistemas com um único processador ainda serem muito usados.
  - É importante entender que os recursos multithread de Java funcionam nos dois tipos de sistema:
    - Em um sistema single-core, a execução concorrente de threads compartilha a CPU, com cada thread recebendo uma fração de tempo.
      - Em um sistema single-core, duas ou mais threads não são executadas realmente ao mesmo tempo, o tempo ocioso da CPU é utilizado.
    - Em sistemas multiprocessadores/multicore, é possível duas ou mais threads serem executadas simultaneamente.
      - Em muitos casos, isso pode melhorar ainda mais a eficiência do programa e aumentar a velocidade de certas operações.

# MULTITHREADING

- Uma thread pode estar em vários estados:
  - NEW:
    - Quando a thread é criada, porém não invocaram o start() na referência.
  - RUNNABLE :
    - Quando volta de algum estado, ou quando foi invocada o start() na referência.
  - BLOCKED:
    - Uma thread é considerada no estado BLOCKED quando está esperando dados.
    - Uma thread também é considerada BLOCKED quando está aguardando a Lock de outra thread.
  - WAITING:
    - Uma thread que está esperando indefinidamente por outra thread para executar uma determinada ação.
  - TIMED\_WAITING:
    - Uma thread que está esperando por outro thread para executar uma ação por até um tempo de espera especificado está neste estado.
  - TERMINATED:
    - Uma thread que saiu.



# MULTITHREADING

- A classe Thread e a interface Runnable
  - O sistema de várias threads de Java tem como base a classe Thread e a interface que a acompanha, Runnable.
  - As duas estão empacotadas em java.lang. Thread encapsula uma thread de execução. Para criar uma nova thread, o programa deve estender Thread ou implementar a interface Runnable.
  - A classe Thread define vários métodos que ajudam a gerenciar as threads. Aqui estão alguns dos mais comuns (eles serão examinados com mais detalhes quando forem usados):

Método	Significado
final String getName( )	Obtém o nome de uma thread.
final int getPriority( )	Obtém a prioridade de uma thread.
final boolean isAlive( )	Determina se uma thread ainda está em execução.
final void join( )	Espera uma thread terminar.
void run( )	Ponto de entrada da thread.
static void sleep(long <i>milissegundos</i> )	Suspende uma thread pelo período de milissegundos especificado.
void start( )	Inicia uma thread chamando seu método <b>run( )</b> .

# MULTITHREADING

- Criando uma thread
  - Instanciando um objeto de tipo Thread.
  - A classe Thread encapsula um objeto que é executável, com duas abordagens:
    - Implementar a interface Runnable.
    - Estender a classe Thread.
  - A interface Runnable concebe uma unidade de código executável. Você pode construir uma thread em qualquer objeto que implementar a interface Runnable.
  - Runnable só define um método, chamado `run()`:
    - `public void run()`
    - Dentro de `run()`, define-se o código que constitui a nova thread.
    - `run()` pode chamar outros métodos, usar outras classes e declarar variáveis da mesma forma que a thread principal. A única diferença é que `run()` estabelece o ponto de entrada de uma thread de execução concorrente dentro do programa.
    - Essa thread terminará quando `run()` retornar.





# MULTITHREADING




- Criando uma thread
  - Após ter criado uma classe que implementa Runnable, pode-se instanciar um objeto de tipo Thread em um objeto dessa classe.
  - Thread define vários construtores, exemplo:
    - Thread(Runnable obThread)
    - obThread é a instância de uma classe que implementa a interface Runnable.
    - Isso define onde a execução da thread começará.
  - Uma vez criada, a nova thread só começará a ser executada quando você chamar seu método start( ), que é declarado dentro de Thread.
    - start( ) executa uma chamada a run( ).

# MULTITHREADING

- Exemplo MyThread

```
// Cria uma thread implementando Runnable.
```

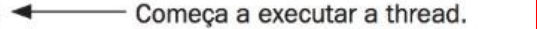
```
class MyThread implements Runnable {  Objetos de MyThread podem ser executados em suas próprias threads, porque MyThread implementa Runnable.  
    String thrdName;  
  
    MyThread(String name) {  
        thrdName = name;  
    }  
  
    // Ponto de entrada da thread.  
    public void run() {  Threads começam a ser executadas aqui.  
        System.out.println(thrdName + " starting.");  
        try {  
            for(int count=0; count < 10; count++) {  
                Thread.sleep(400);  
                System.out.println("In " + thrdName +  
                                   ", count is " + count);  
            }  
        }  
        catch (InterruptedException exc) {  
            System.out.println(thrdName + " interrupted.");  
        }  
        System.out.println(thrdName + " terminating.");  
    }  
}
```

```
class UseThreads {  
    public static void main(String args[]) {  
        System.out.println("Main thread starting.");  
  
        // Primeiro, constrói um objeto MyThread.  
        MyThread mt = new MyThread("Child #1");  Cria um objeto executável.  
  
        // Em seguida, constrói uma thread a partir desse objeto.  
        Thread newThrd = new Thread(mt);  Constrói uma thread nesse objeto.  
  
        // Para concluir, começa a execução da thread.  
        newThrd.start();  Começa a executar a thread.  
  
        for(int i=0; i<50; i++) {  
            System.out.print(".");  
            try {  
                Thread.sleep(100);  
            }  
            catch (InterruptedException exc) {  
                System.out.println("Main thread interrupted.");  
            }  
        }  
  
        System.out.println("Main thread ending.");  
    }  
}
```

# MULTITHREADING

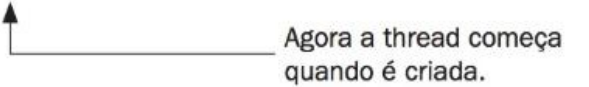
- Exemplo MyThread Melhorada

```
// MyThread melhorada.

class MyThread implements Runnable {
    Thread thrd; 
    // Constrói uma nova thread.
    MyThread(String name) {
        thrd = new Thread(this, name); 
        thrd.start(); // inicia a thread 
    }

    // Começa a execução da nova thread.
    public void run() {
        System.out.println(thrd.getName() + " starting.");
        try {
            for(int count=0; count<10; count++) {
                Thread.sleep(400);
                System.out.println("In " + thrd.getName() +
                    ", count is " + count);
            }
        } catch (InterruptedException exc) {
            System.out.println(thrd.getName() + " interrupted.");
        }
        System.out.println(thrd.getName() + " terminating.");
    }
}
```

```
class UseThreadsImproved {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");

        MyThread mt = new MyThread("Child #1"); 

        for(int i=0; i < 50; i++) {
            System.out.print(".");
            try {
                Thread.sleep(100);
            } catch (InterruptedException exc) {
                System.out.println("Main thread interrupted.");
            }
        }

        System.out.println("Main thread ending.");
    }
}
```

- Algumas melhorias simples o código ficará mais eficiente e fácil de usar.
- É possível fazer a thread começar a ser executada assim que for criada.
- No caso de MyThread, isso é feito pela instanciação de um objeto Thread dentro do construtor de MyThread.
- Em segundo lugar, não há necessidade de MyThread armazenar o nome da thread,
- O nome é atribuído a uma thread quando ela é criada.

# MULTITHREADING

- Criando várias threads

```
// Cria várias threads.

class MyThread implements Runnable {
    Thread thrd;

    // Constrói uma nova thread.
    MyThread(String name) {
        thrd = new Thread(this, name);

        thrd.start(); // inicia a thread
    }

    // Começa a execução da nova thread.

    public void run() {
        System.out.println(thrd.getName() + " starting.");
        try {
            for(int count=0; count < 10; count++) {
                Thread.sleep(400);
                System.out.println("In " + thrd.getName() +
                                   ", count is " + count);
            }
        }
        catch(InterruptedException exc) {
            System.out.println(thrd.getName() + " interrupted.");
        }
        System.out.println(thrd.getName() + " terminating.");
    }
}
```

```
class MoreThreads {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");

        MyThread mt1 = new MyThread("Child #1");
        MyThread mt2 = new MyThread("Child #2");
        MyThread mt3 = new MyThread("Child #3");

        for(int i=0; i < 50; i++) {
            System.out.print(".");
            try {
                Thread.sleep(100);
            }
            catch(InterruptedException exc) {
                System.out.println("Main thread interrupted.");
            }
        }

        System.out.println("Main thread ending.");
    }
}
```

← Cria e começa a executar três threads.

# MULTITHREADING

- Estendendo a Classe Thread
  - A implementação de Runnable é uma maneira de criar uma classe que possa instanciar objetos de thread.
  - Herdar Thread é outra.
  - Exemplo: Herança da classe Thread para o programa UseThreadsImproved.

# MULTITHREADING

- Estendendo a Classe Thread

```
class MyThread extends Thread {  
  
    // Constrói uma nova thread.  
    MyThread(String name) {  
        super(name); // nomeia a thread  
        start(); // inicia a thread  
    }  
  
    // Começa a execução da nova thread.  
    public void run() {  
        System.out.println(getName() + " starting.");  
        try {  
            for(int count=0; count < 10; count++) {  
                Thread.sleep(400);  
                System.out.println("In " + getName() +  
                    ", count is " + count);  
            }  
        }  
        catch (InterruptedException exc) {  
            System.out.println(getName() + " interrupted.");  
        }  
  
        System.out.println(getName() + " terminating.");  
    }  
}
```

```
class ExtendThread {  
    public static void main(String args[]) {  
        System.out.println("Main thread starting.");  
  
        MyThread mt = new MyThread("Child #1");  
  
        for(int i=0; i < 50; i++) {  
            System.out.print(".");  
            try {  
                Thread.sleep(100);  
            }  
            catch (InterruptedException exc) {  
                System.out.println("Main thread interrupted.");  
            }  
        }  
  
        System.out.println("Main thread ending.");  
    }  
}
```

# MULTITHREADING

- Considerações finais
  - A classe Thread define vários métodos que podem ser sobrepostos por uma classe derivada.
  - O único que deve ser sobreposto é o run(), tanto quando implementamos o Runnable ou estendemos a classe Thread.
  - Práticas remetem que só devemos estender classes quando necessitamos melhorar ou modificar.
  - Caso não necessite sobrepor nenhum dos outros métodos de Thread, é indicado simplesmente implementar Runnable.
  - Implementar Runnable também permite que a thread herde uma classe que não seja Thread.

# MULTITHREADING

- Referência
  - Schildt, Herbert. Java para Iniciantes. 6º Edição. Bookman, 2015.