

Programação Orientada a Objetos

Professor: Anderson Elias



Estácio

1


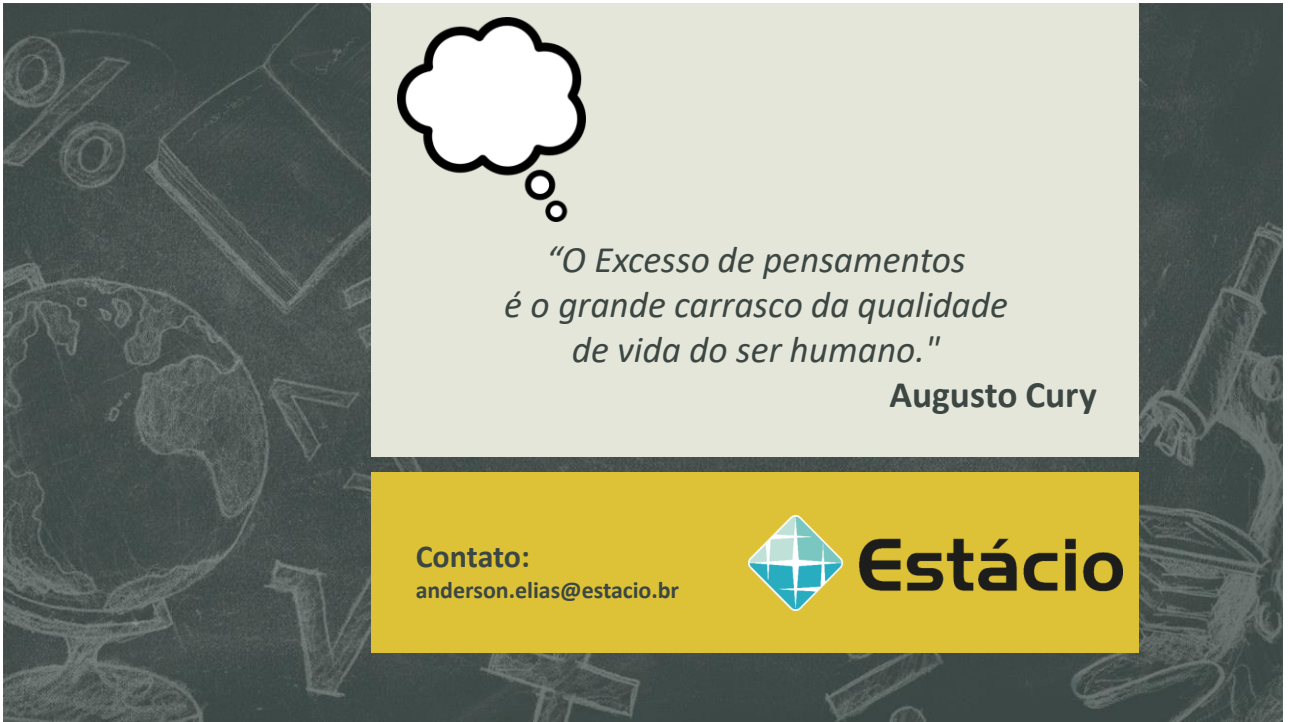
Java e Paralelismo

Contato:
anderson.elias@estacio.br



Estácio


2



*"O Excesso de pensamentos
é o grande carrasco da qualidade
de vida do ser humano."*

Augusto Cury

Contato:
anderson.elias@estacio.br



Estácio

3

Roteiro – Conceitos Básicos

- Java e Paralelismo;
 - Singlecore e Multicore;
 - Multitarefa/Multitasking;
 - Threads;
 - Estendendo a classe Thread;
 - Implementando a interface Runnable;

4

Java e Paralelismo

- Os processadores com mais de um núcleo viabilizaram o processamento de tarefas paralelas e concorrentes;
- O principal objetivo da programação paralela é minimizar o desperdício de recursos de hardware para obter maior desempenho do programa;
- A linguagem Java dispõe de APIs que contextualizam a implementação do paralelismo em programas.

5

Java e Paralelismo

- Java pode nos proporcionar um poder da programação paralela ainda mais expressivo com recursos computacionais com mais de um núcleo ou **multicore**.
- Esta arquitetura com mais de um núcleo é fundamental para que isto seja possível.
- Diferente das arquiteturas **singlecore**, as **multicore** permitem a execução de mais de uma tarefa simultaneamente.

6

Java e Paralelismo

- Uma arquitetura **quadcore** de clock em 1.5Ghz é aquela que consegue realizar a execução de quatro tarefas ao mesmo tempo onde cada núcleo tem a velocidade de 1.5Gz e não somar cada clock dos núcleos para obter 6.0Ghz de velocidade.
- Os Sistemas Operacionais já se beneficiam da arquitetura dos recursos **multicore** usando o conceito de **multitarefa**.

7

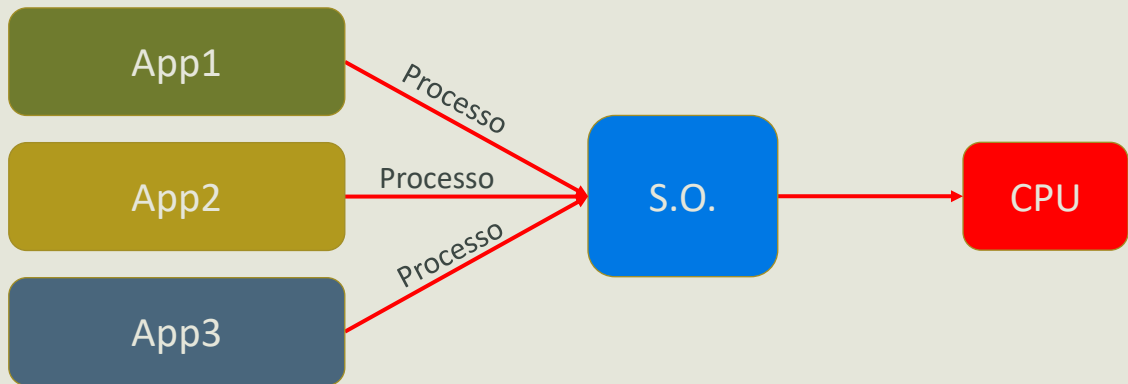
Java e Paralelismo

- A capacidade que um sistema tem de executar várias tarefas simultaneamente é chamado de **multitarefa** ou **multitasking**.
- Este processo se utiliza do compartilhamento do núcleo de processamento e da memória.
- Se chama de **escalonamento de processos** onde há um gerenciamento das tarefas que serão executadas através de uma fila no núcleo.

8

Java e Paralelismo

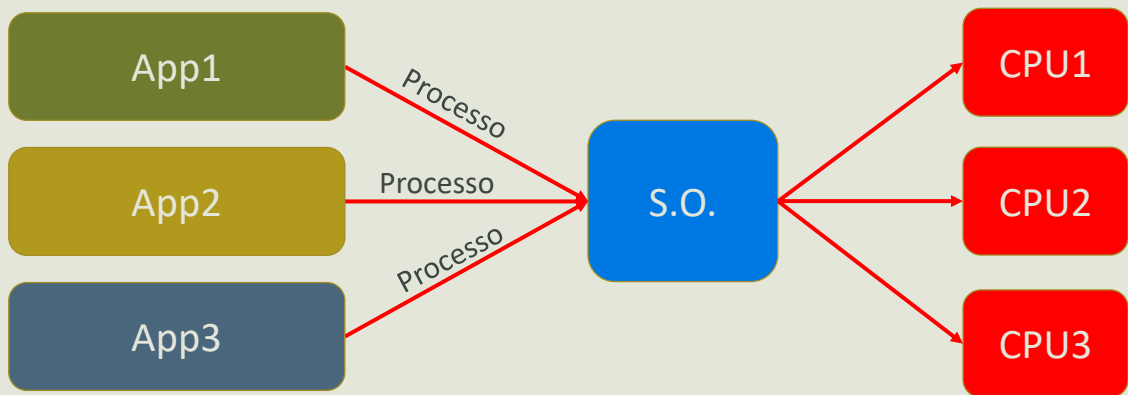
- Exemplo: aplicação **singlecore**



9

Java e Paralelismo

- Exemplo: aplicação **multicore**



10

Java e Paralelismo

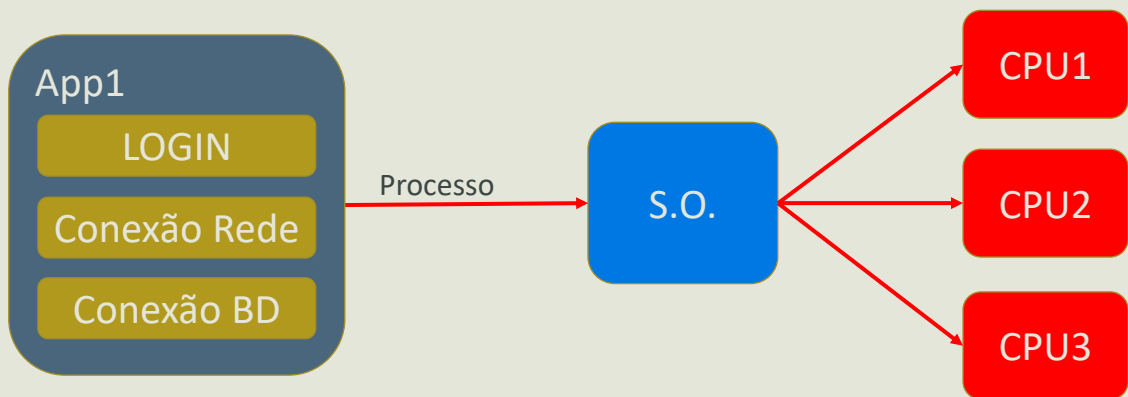
Não adianta ter muitos núcleos de processamento se o software em questão não utiliza a programação paralela ou concorrente.

- **multitarefa** ou **multitasking** foi evoluído para **multithreading** que foca no nível de processo.
- **Multithreading** prover a subdivisão das tarefas do software em questão em partes de códigos independentes para serem executados em paralelo. processo se utiliza do compartilhamento do núcleo de processamento e da memória.

11

Java e Paralelismo

- Exemplo: aplicação **multithreading**



12

Java e Paralelismo

- Através destes novos recursos, o ganho de performance é o mais desejado de se alcançar.
- Além do controle mais eficiente no balanceamento de carga sobre os núcleos de processamento.

13

Java e Paralelismo - Threads

- Threads são fluxos de controles sequenciais isolados dentro de um mesmo software.
- Em java são um mecanismo de prover concorrência suportada.
- Diferente dos softwares em que não dividem o mesmo espaço em memória, as threads conseguem e isto permite a elas compartilhar dados dentro do contexto do programa.

14

Java e Paralelismo - Threads

- Cada objeto instanciado de Thread possui:
 - Identificador único que não pode ser alterado;
 - Nome;
 - Prioridade;
 - Estado;
 - Gerenciador de exceções;
 - Espaço para armazenamento local;
 - E uma série de estruturas utilizadas pela JVM e pelo SO.

15

Java e Paralelismo - Threads

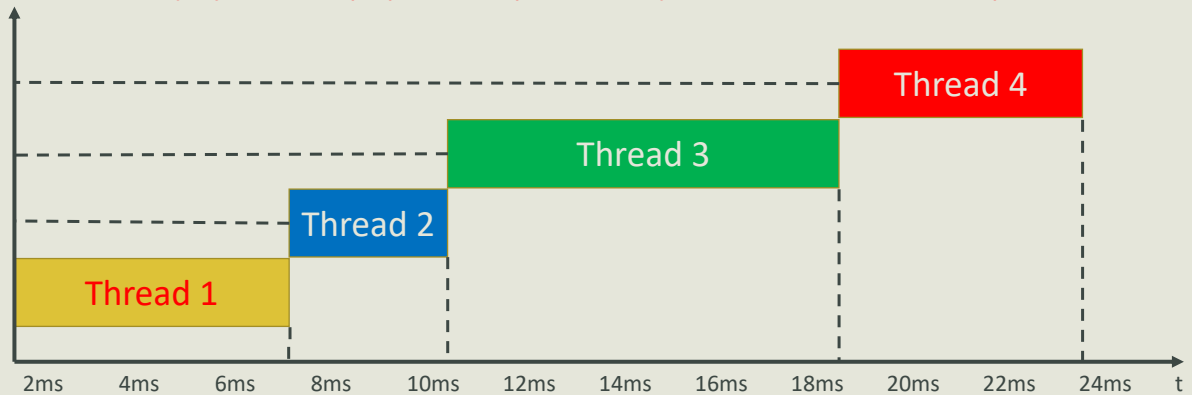
- Mesmo utilizando Threads em apenas um núcleo de processamento o software implementa um sincronismo de threads que não as permite executar em paralelo ou quando o sistema não faz uso de threads.
- O escalonador da JVM pode pausar e dar espaço e tempo para outra thread ser executada.

16

Java e Paralelismo - Threads

Escalonamento de threads, modo round-robin

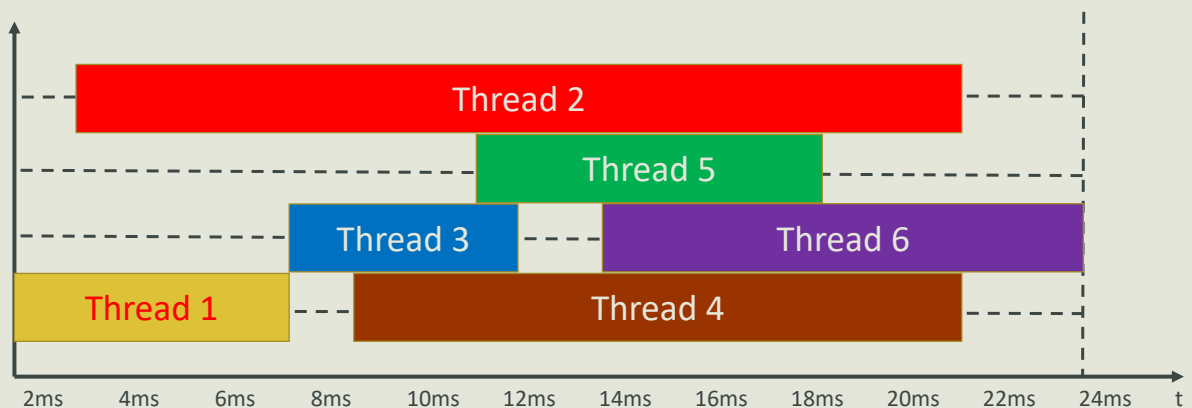
São algoritmos simples para agendamento de processos em um sistema operacional, nele se atribui frações de tempo para cada processo de forma circular onde não há prioridades.



17

Java e Paralelismo - Threads

- Várias threads executando paralelamente e otimizando o uso da CPU.



18

Java e Paralelismo - Threads

A partir da versão 5, Java incluiu APIs de alto nível para fornecer mais recursos na implementação de tarefas paralelas.

Estes recursos podem ser encontrados no pacote:

```
import java.util.concurrent.*;
```

19

Java e Paralelismo - Threads

Todo programa desenvolvido em java, no mínimo já tem uma thread que é criada e iniciada pela JVM quando iniciamos o programa que tem como tarefa executar o método **main()** da classe principal.

Será executado sequencialmente os códigos contidos neste método até que termine e encerra a thread e seu processamento, em seguida a aplicação poderá ser finalizada.

20

Java e Paralelismo - Threads

Podemos utilizar na linguagem Java dois modos para criar nossas Threads:

- Estendendo a classe Thread.

```
import java.lang.Thread;
```

- E implementando a interface Runnable.

```
import java.lang.Runnable;
```

21

Java e Paralelismo - Threads

- A classe Thread é usada quando a classe a ser executada concorrentemente não deriva de outra classe;
- Contém métodos para controlar a execução;
- Criação e execução:
 - Declarar uma nova classe que seja subclasse da classe Thread;
 - Sobrescrever o método **run()** com o código que será executado pela thread;
 - Instanciar a nova classe;
 - Invocar seu método **start()**. Este método irá rodar em seu próprio thread de controle.

22

Java e Paralelismo - Threads

- Contexto da classe Thread:

java.lang.Object  java.lang.Thread

```
public class Thread extends Object implements Runnable
```

23

Java e Paralelismo - Threads

- Alguns métodos da classe Thread:

- **start()**: inicia a execução do Thread;
- **sleep()**: suspende a execução por um determinado tempo (*especificado em milisegundos*) e automaticamente recomeça a execução;
- **join()**: congela a execução da thread corrente e aguarda a conclusão da thread na qual esse método foi invocado;
- **wait()**: faz a thread aguardar até que outra invoque o método **notify()** ou **notifyAll()**;
- **interrupt()**: acorda uma thread que está dormindo devido a uma operação de **sleep()** ou **wait()**, ou foi bloqueada por causa de um processamento longo de I/O
- **destroy()**: destrói esta thread.

24

Java e Paralelismo - Threads

• **Exemplo1:**

```
public class Corrida extends Thread {
    private String nome;
    public Corrida(String nome) {
        this.nome = nome;
        start();
    }
    public void run() {
        System.out.println(nome + " Começou...");
        for (int i = 0; i < 5; i++) {
            System.out.println(nome + " já percorreu " + i + " voltas.");
        }
        System.out.println(nome + "Terminou");
    }
}
```

25

Java e Paralelismo - Threads

• **Exemplo1 Continuação:**

```
public class CorridaStart {
    public static void main(String[] args){
        Corrida t1 = new Corrida("José");
        Corrida t2 = new Corrida("Maria");
        Corrida t3 = new Corrida("Pedro");
        Corrida t4 = new Corrida("Elaine");
        Corrida t5 = new Corrida("Carla");
        Corrida t6 = new Corrida("Manoel");
        Corrida t7 = new Corrida("Paulo");
    }
}
```

26

Java e Paralelismo - Threads

• Resultado:

José Começou...	Todos Terminaram	Elaine já percorreu 4 voltas.
José já percorreu 0 voltas.	Manoel Começou...	Elaine foi interrompida.
Maria Começou...	Manoel já percorreu 0 voltas.	ElaineTerminou
Maria já percorreu 0 voltas.	Manoel já percorreu 1 voltas.	José foi interrompida.
Maria já percorreu 1 voltas.	Manoel já percorreu 2 voltas.	JoséTerminou
Elaine Começou...	Manoel já percorreu 3 voltas.	Paulo Começou...
Carla Começou...	Manoel já percorreu 4 voltas.	Pedro já percorreu 0 voltas.
Carla já percorreu 0 voltas.	Manoel foi interrompida.	Pedro já percorreu 1 voltas.
José já percorreu 1 voltas.	ManoelTerminou	Pedro já percorreu 2 voltas.
José já percorreu 2 voltas.	Elaine já percorreu 0 voltas.	Pedro já percorreu 3 voltas.
José já percorreu 3 voltas.	Elaine já percorreu 1 voltas.	Pedro já percorreu 4 voltas.
José já percorreu 4 voltas.	Elaine já percorreu 2 voltas.	Paulo já percorreu 0 voltas.
Carla já percorreu 1 voltas.	Elaine já percorreu 3 voltas.	Pedro foi interrompida.
Carla já percorreu 2 voltas.	Maria já percorreu 2 voltas.	PedroTerminou
Carla já percorreu 3 voltas.	Maria já percorreu 3 voltas.	Paulo já percorreu 1 voltas.
Carla já percorreu 4 voltas.	Maria já percorreu 4 voltas.	Paulo já percorreu 2 voltas.
Carla foi interrompida.	Maria foi interrompida.	Paulo já percorreu 3 voltas.
CarlaTerminou	MariaTerminou	Paulo já percorreu 4 voltas.
	Pedro Começou...	Paulo foi interrompida.
	Elaine já percorreu 4 voltas.	PauloTerminou

27

Java e Paralelismo – Threads (Runnable)

- A interface **Runnable** é usada quando não se pode herdar da classe **Thread**, pois há necessidade de herança de alguma outra classe;
 - possui apenas o método **run()**.
- Se cria um objeto da classe base **Thread**, mas o código a ser executado está descrito na classe do usuário;
- Criação e execução:
 - Declara uma nova classe que implementa a interface **Runnable**;
 - Sobrescrever o método **run()**;
 - Criar um objeto da classe **Thread**;
 - Exemplo: **Thread** um = new **Thread(this)**.

28

Java e Paralelismo - Threads

• Exemplo2:

```
public class CorridaRun implements Runnable {
    private String nome;
    public CorridaRun(String nome) {
        this.nome = nome;
    }
    public void run() {
        System.out.println(nome + " Começou...");
        for (int i = 0; i < 5; i++) {
            System.out.println(nome + " já percorreu " + i + " voltas.");
        }
        System.out.println(nome + "Terminou");
    }
}
```

29

Java e Paralelismo - Threads

• Exemplo2 Continuação:

```
public class CorridaRunStart {
    public static void main(String[] args){
        CorridaRun t1 = new CorridaRun("José");
        CorridaRun t2 = new CorridaRun("Maria");
        CorridaRun t3 = new CorridaRun("Pedro");
        CorridaRun t4 = new CorridaRun("Elaine");
        CorridaRun t5 = new CorridaRun("Carla");
        CorridaRun t6 = new CorridaRun("Manoel");
        CorridaRun t7 = new CorridaRun("Paulo");
        new Thread(t1).start();
        new Thread(t2).start();
        new Thread(t3).start();
        new Thread(t4).start();
        new Thread(t5).start();
        new Thread(t6).start();
        new Thread(t7).start();
    }
}
```

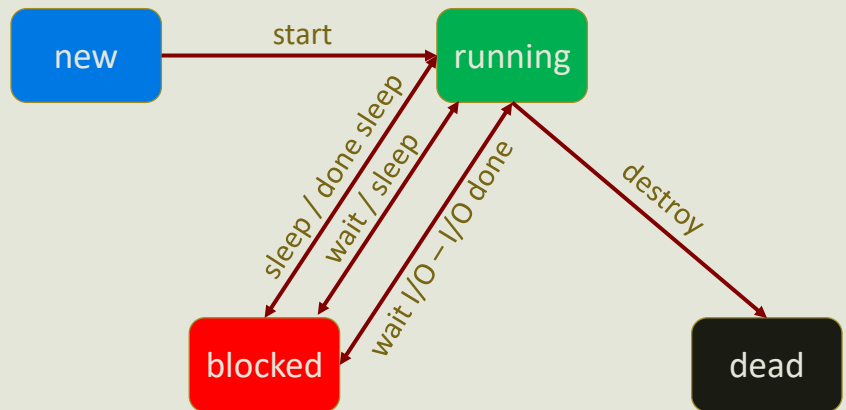
Iniciando através das Threads →

30

Java e Paralelismo – Estado das Threads

No decorrer da execução do programa, as Threads podem alterar seu estado para:

- Ativo;
- Inativo;
- Encerrado.



31

Java e P

- Exemplo3:
- sleep

```

public class Corrida extends Thread {
    private String nome;
    private int tempoHidratacao;
    public Corrida(String nome, int tempoHidratacao) {
        this.nome = nome;
        this.tempoHidratacao = tempoHidratacao;
        start();
    }
    public void run() {
        System.out.println(nome + " Começou...");
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println(nome + " já percorreu " + i + " voltas.");
                Thread.sleep(tempoHidratacao);
            }
        } catch (InterruptedException e) {
            System.out.println(nome + " foi interrompida.");
        }
        System.out.println(nome + " Terminou");
    }
}
  
```

32

Java e Paralelismo – Threads (Sleep)

- Exemplo3
- Continuação:
- **sleep**

```
public class CorridaStart {
    public static void main(String[] args){
        Corrida t1 = new Corrida("José", 1200);
        Corrida t2 = new Corrida("Maria", 900);
        Corrida t3 = new Corrida("Pedro", 500);
        Corrida t4 = new Corrida("Elaine", 870);
        Corrida t5 = new Corrida("Carla", 1180);
        Corrida t6 = new Corrida("Manoel", 1525);
        Corrida t7 = new Corrida("Paulo", 720);
        System.out.println( "Todos Terminaram");
    }
}
```

33

Java e Paralelismo – Threads (Sleep)

- Exemplo3 Resultado: **sleep**

Identificando vários núcleos disponíveis

```
System.out.println("Núcleos Livres neste momento:
    "+Runtime.getRuntime().availableProcessors()+ " núcleos");
```

34

Java e Paralelismo – Threads (Sleep)

• Exemplo3 Resultado: sleep

```
public class CorridaStart {
    public static void main(String[] args){
        System.out.println("Núcleos Livres neste momento: "
            +Runtime.getRuntime().availableProcessors()+ "
            núcleos");
        Corrida t1 = new Corrida("José", 1200);
        Corrida t2 = new Corrida("Maria", 900);
        Corrida t3 = new Corrida("Pedro", 500);
        Corrida t4 = new Corrida("Elaine", 870);
        Corrida t5 = new Corrida("Carla", 1180);
        Corrida t6 = new Corrida("Manoel", 1525);
        Corrida t7 = new Corrida("Paulo", 720);
        System.out.println( "Todos Terminaram");
    }
}
```

35

Java e Paralelismo – Saída

RESULTADO.

Núcleos Livres neste momento: 12 núcleos

José Começou...
 José já percorreu 0 voltas.
 Pedro Começou...
 Maria Começou...
 Pedro já percorreu 0 voltas.
 Elaine Começou...
 Elaine já percorreu 0 voltas.
 Pedro já percorreu 1 voltas.
 Elaine já percorreu 1 voltas.
 Maria já percorreu 1 voltas.
 Pedro já percorreu 2 voltas.
 José já percorreu 1 voltas.
 Pedro já percorreu 3 voltas.
 Elaine já percorreu 2 voltas.

Maria já percorreu 2 voltas.
 Pedro já percorreu 4 voltas.
 José já percorreu 2 voltas.
 Pedro já percorreu 5 voltas.
 Elaine já percorreu 3 voltas.
 Maria já percorreu 3 voltas.
 PedroTerminou
 Elaine já percorreu 4 voltas.
 José já percorreu 3 voltas.
 Maria já percorreu 4 voltas.
 Elaine já percorreu 5 voltas.
 Maria já percorreu 5 voltas.
 José já percorreu 4 voltas.
 ElaineTerminou
 MariaTerminou
 José já percorreu 5 voltas.
 JoséTerminou

36

Java e Paralelismo

```
public class CorridaRun implements Runnable {
    private String nome;
    private int tempoHidratacao;
    public CorridaRun(String nome, int tempoHidratacao) {
        this.nome = nome;
        this.tempoHidratacao = tempoHidratacao;
    }
    public void run() {
        System.out.println(nome + " Começou...");
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println(nome + " já percorreu " + i + " voltas.");
                Thread.sleep(tempoHidratacao);
            }
        } catch (InterruptedException e) {
            System.out.println(nome + " foi interrompida.");
        }
        System.out.println(nome + "Terminou");
    }
}
```

37

Java e Paralelismo –

```
public class CorridaRunStart {
    public static void main(String[] args) {
        CorridaRun t1 = new CorridaRun("José", 1200);
        CorridaRun t2 = new CorridaRun("Maria", 900);
        CorridaRun t3 = new CorridaRun("Pedro", 500);
        CorridaRun t4 = new CorridaRun("Elaine", 870);
        CorridaRun t5 = new CorridaRun("Carla", 1180);
        CorridaRun t6 = new CorridaRun("Manoel", 1525);
        CorridaRun t7 = new CorridaRun("Paulo", 720);
        Thread t1_1 = new Thread(t1);
        Thread t2_2 = new Thread(t2);
        Thread t3_3 = new Thread(t3);
        Thread t4_4 = new Thread(t4);
        Thread t5_5 = new Thread(t5);
        Thread t6_6 = new Thread(t6);
        Thread t7_7 = new Thread(t7);
    }
}
```

38

Java e Paralelismo –

```
t1_1.start();
t2_2.start();
t3_3.start();
t4_4.start();
t5_5.start();
t6_6.start();
t7_7.start();
}
```

```
José Começou...
Elaine Começou...
Elaine já percorreu 0 voltas.
Maria Começou...
Maria já percorreu 0 voltas.
Pedro Começou...
Pedro já percorreu 0 voltas.
Paulo Começou...
Paulo já percorreu 0 voltas.
Manoel Começou...
Carla Começou...
Carla já percorreu 0 voltas.
José já percorreu 0 voltas.
Manoel já percorreu 0 voltas.
Pedro já percorreu 1 voltas.
Paulo já percorreu 1 voltas.
Elaine já percorreu 1 voltas.
Maria já percorreu 1 voltas.
Pedro já percorreu 2 voltas.
Carla já percorreu 1 voltas.
```

39

Java e Paralelismo –

```
t1_1.start();
t2_2.start();
t3_3.start();
t4_4.start();
t5_5.start();
t6_6.start();
t7_7.start();

System.out.println("=====A CORRIDA FINALIZOU=====");
}
```

**PROBLEMA, A THREAD
MAIND TERMINOU ANTES
DAS THREADS**

```
José Começou...
Elaine Começou...
Elaine já percorreu 0 voltas.
Maria Começou...
Maria já percorreu 0 voltas.
Pedro Começou...
Pedro já percorreu 0 voltas.
Paulo Começou...
Paulo já percorreu 0 voltas.
Manoel Começou...
=====A CORRIDA FINALIZOU=====
Carla Começou...
Carla já percorreu 0 voltas.
José já percorreu 0 voltas.
Manoel já percorreu 0 voltas.
Pedro já percorreu 1 voltas.
Paulo já percorreu 1 voltas.
Elaine já percorreu 1 voltas.
Maria já percorreu 1 voltas.
Pedro já percorreu 2 voltas.
Carla já percorreu 1 voltas.
```

40

Java e Paralelismo – Threads

- A partir de agora iremos utilizar dois recursos para resolver um problema que geralmente encontramos nas Threads.
- Quando se tem várias instancias de Thread sendo executadas simultaneamente, sabemos que o fluxo do código continua normalmente. Pois os processos das Threads após iniciadas, são inerentes ao restante do fluxo. (São fluxos paralelos).

41

Java e Paralelismo – Threads

- *Usaremos a resposta do exercício anterior:*
- *Podemos incluir este trecho de código logo depois de iniciar os objetos de Threads.*

```
for (int i = 0; i < 5; i++) {
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {System.out.println("foi interrompida.");}
}
System.out.println("=====A CORRIDA FINALIZOU=====");
```

42


Java e Paralelismo – Exercícios **RESPOSTA**

RESULTADO.

```
Pedro Começou...
Pedro já percorreu 0 voltas.
Elaine Começou...
Elaine já percorreu 0 voltas.
Pedro Começou...
Pedro já percorreu 0 voltas.
José Começou...
José já percorreu 0 voltas.
José Começou...
José já percorreu 0 voltas.
Elaine Começou...
Elaine já percorreu 0 voltas.
Maria Começou...
Pedro já percorreu 1 voltas.
=====TRHEAD MAIN INTERROMPIDA=====
Elaine já percorreu 1 voltas.
Elaine já percorreu 1 voltas.
```

```
Maria já percorreu 1 voltas.
Maria já percorreu 1 voltas.
Pedro já percorreu 2 voltas.
Pedro já percorreu 2 voltas.
José já percorreu 1 voltas.
```

**CONTINUA O PROBLEMA,
TERMINOU ANTES DAS
THREADS**



```
Maria já percorreu 2 voltas.
Pedro já percorreu 4 voltas.
Pedro já percorreu 4 voltas.
José já percorreu 2 voltas.
José já percorreu 2 voltas.
PedroTerminou
```

43

Java e Paralelismo – Threads (**IsAlive**)

- Para evitar que uma parte do código seja processada antes das Threads terminarem, podemos utilizar o método **IsAlive()**:
- Ele retorna True se a Thread ainda estiver em execução. Portanto, faremos uma pequena e significativa alteração:

```
//MUDANÇA PARA QUE A THREAD MAIN NÃO TERMINE APÓS AS OUTRAS
//for (int i = 0; i < 5; i++) {
//}
```

44

Java e Paralelismo – Threads (**IsAlave**)

```

t1_1.start();
t2_2.start();
t3_3.start();
t4_4.start();
t5_5.start();
t6_6.start();
t7_7.start();
do{
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        System.out.println("foi interrompida.");
    }
} while(t1_1.isAlive() || t2_2.isAlive() || t3_3.isAlive() || t4_4.isAlive() ||
        t5_5.isAlive() || t6_6.isAlive() || t7_7.isAlive());
    System.out.println("=====A CORRIDA FINALIZOU=====");
}
}

```

45

Java e Paralelismo – Threads (**Prioridade**)

Podemos utilizar um recurso muito útil com as Threads que é criar prioridade de execução de cada processo.

Desta forma faremos uma ordenação entre as Threads para informar ao compilador qual Thread tem prioridade de execução sobre a outra.

Utilizaremos o método **setPriority()**: Passaremos qual é o grau de prioridade através de números inteiros.

46

Java e Paralelismo – Threads (Prioridade)

O intervalo de prioridade de execução das Threads são entre 1 e 10.

Por padrão quando não informada, todas as prioridades de execução de uma Thread é de valor 5.

Podemos também definir o grau de prioridade através das constantes: *Min_Priority*, *Max_Priority* e *Norm_Priority*

47

Java e Paralelismo – Threads (Prioridade)

Exemplo:

```
public class ExemploSleepRun implements Runnable{
    private String nome;
    private int tempoHidratacao;
    public ExemploSleepRun (String nome, int tempoHidratacao){
        this.nome = nome;
        this.tempoHidratacao = tempoHidratacao;
    }
    public void run(){
        System.out.println(nome+" Começou...");
        try{
            for(int i=0;i<5; i++){
                System.out.println(nome+ " já percorreu "+ i + " voltas.");
                Thread.sleep(tempoHidratacao);
            }
        }catch(InterruptedException e){
            System.out.println(nome+ " foi interrompida.");
        } System.out.println(nome+ "Terminou");
    }
}
```

Continua...

48

Java e Paralelismo – Threads (Prioridade)

Exemplo:

```
public static void main(String[] args){
    ExemploSleepRun t1 = new ExemploSleepRun("José", 1200);
    ExemploSleepRun t2 = new ExemploSleepRun("Maria", 900);
    ExemploSleepRun t3 = new ExemploSleepRun("Pedro", 500);
    ExemploSleepRun t4 = new ExemploSleepRun("Elaine", 870);
    Thread t1_1 = new Thread(t1);
    Thread t2_2 = new Thread(t2);
    Thread t3_3 = new Thread(t3);
    Thread t4_4 = new Thread(t4);
    t1_1.setPriority(5);
    t2_2.setPriority(3);
    t3_3.setPriority(1);
    t4_4.setPriority(2);
    t1_1.start();
    t2_2.start();
    t3_3.start();
    t4_4.start();
} }
```

Ou se usa as Constantes...

```
t1_1.setPriority(Thread.NORM_PRIORITY);
```

49

Java e Paralelismo – Threads (Sincronizar)

Programas multi-threaded muitas vezes podem chegar a uma situação em que vários segmentos tentam acessar os mesmos recursos e, finalmente, produzir resultados errados e imprevistos.

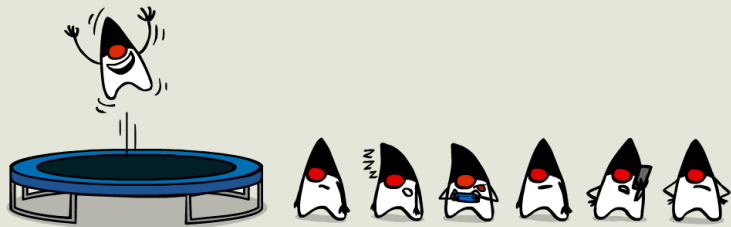
Portanto, é necessário garantir que, por meio de algum método de sincronização, apenas um thread possa acessar o recurso em um determinado momento.

50

Java e Paralelismo – Threads (**Sincronizar**)

Java fornece uma maneira de criar threads e sincronizar suas tarefas usando blocos sincronizados.

Blocos sincronizados em Java são identificados pela palavra-chave *synchronized*.



51

Java e Paralelismo – Threads (**Sincronizar**)

Um bloco sincronizado em Java é sincronizado em algum objeto.

Todos os blocos sincronizados no mesmo objeto podem ter apenas uma thread executando dentro deles por vez.

Todos os outros threads que tentam entrar no bloco sincronizado são bloqueados até que o thread dentro do bloco sincronizado saia do bloco.

52

Java e Paralelismo – Threads (Sincronizar)

RECAPITULANDO...

Sincronização é o ato de coordenar as atividades de duas ou mais Threads.

Motivo: Quando duas ou mais Threads precisam acessar um recurso compartilhado, ou somente uma Thread pode acessar o recurso por vez.

Em Java se usa a palavra chave **Synchronized** em métodos (Assinatura) ou em bloco de códigos.

53

Java e Paralelismo – Threads (Sincronizar)

Exemplo:

```
public class Calculadora {
    private int soma;
    synchronized int somaArray(int[] nums) {
        soma = 0;
        for (int i = 0; i < nums.length; i++) {
            soma += nums[i];
            System.out.println("Executando Thread " +
                Thread.currentThread().getName() +
                " somando valor " + nums[i] + " e a soma total é " + soma);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return soma;
    }
}
```

Método para
prover o sincronismo

Continua...

54

Java e Paralelismo – Threads (Sincronizar)

Exemplo:

```

public class MinhaThreadSoma implements Runnable{
    private String nome;
    private int[] nums;
    private static Calculadora calc = new Calculadora();
    private Thread thread;
    public MinhaThreadSoma(String nome, int[]nums){
        this.nome = nome;
        this.nums = nums;
        thread = new Thread(this, nome);
        thread.start();
    }
    public void run() {
        System.out.println(this.nome+" Início");
        int soma = calc.somaArray(nums);
        System.out.println("Resultado para "+ nome + " é " + soma);
        System.out.println(this.nome+" FIM");
    }
}

```

Continua...

55

Java e Paralelismo – Threads (Sincronizar)

Exemplo:

```

public class TestarCalculadora {
    public static void main(String[] arg) {
        int [] nums = {1, 2, 3, 4, 5};
        MinhaThreadSoma t1 = new MinhaThreadSoma("Thread 1", nums);
        MinhaThreadSoma t2 = new MinhaThreadSoma("Thread 2", nums);
    }
}

```

56

Java e Paralelismo – Exemplo

Resposta:

```
public class Enviar {
    public void envia(String msg)
    { //(\t) tabula
        System.out.println("Enviando\t" + msg );
        try {
            Thread.sleep(1000);
        }
        catch (Exception e){
            System.out.println("Thread interrompida.");
        }
        System.out.println("\n" + msg + "Enviada.");
    }
}
```

Continua...

57

Java e Paralelismo – Exemplo

Resposta:

```
public class EnviarThread extends Thread {
    private String msg;
    Enviar enviar;
    // Recebe uma string e um objeto msg para ser enviada.
    EnviarThread(String m, Enviar obj){
        msg = m;
        enviar = obj;
    }
    public void run() {
        // Apenas uma Thread pode enviar uma mensagem por vez.
        synchronized(enviar){
            // sincroniza o objeto enviar.
            enviar.envia(msg);
        }
    }
}
```

Continua...

58

Java e Paralelismo – Exemplo

Resposta:

```
public class EnviarMsg {
    public static void main(String args[]){
        Enviar enviar = new Enviar();
        EnviarThread S1 = new EnviarThread( " Olá " , enviar );
        EnviarThread S2 = new EnviarThread( " Xau " , enviar );
        // Iniciando duas Threads do tipo EnviarThread
        S1.start();
        S2.start();
        // Configurando para que aguardem pelo encerramento das threads
        try{
            S1.join();
            S2.join();
        }
        catch(Exception e){ //Caso algum problema ocorra.
            System.out.println("Thread Interrompida.");
        }
    }
}
```

FIM.

59

Java e Paralelismo – Threads (Sincronizar)

Nem sempre temos que sincronizar um método inteiro.

Às vezes é preferível sincronizar apenas parte de um método.

Portanto, blocos sincronizados dentro de métodos tornam isso possível.

60

Java e Paralelismo – Threads (**Sincronizar**)

Melhorando o código...

Como sincronizamos o objeto **Enviar** dentro do método `run ()` da classe **EnviarThread**, a saída será sempre a mesma.

Então, como alternativa, podemos definir todo o bloco `Enviar()` como `synchronized` e produzir o mesmo resultado. Desta forma, não precisamos sincronizar o objeto da mensagem dentro do método `run()` na classe `EnviarThread`.

Altere sua a solução para melhor utilizar o `synchronized` neste contexto.

61

Java e Paralelismo – Exemplo

Alternativa 1:

```
public class Enviar {
    public synchronized void envia(String msg) {
        System.out.println("Enviando\t" + msg );
        try {
            Thread.sleep(1000);
        } catch (Exception e){
            System.out.println("Thread interrompida.");
        }
        System.out.println("\n" + msg + "Enviada.");
    }
}
```

Uma implementação alternativa para demonstrar que podemos usar `synchronized` com o método também.

62

Java e Paralelismo – Exemplo

Alternativa 2:

```
public class Enviar {
    public void envia(String msg {
        synchronized(this){
            System.out.println("Enviando\t" + msg );
            try {
                Thread.sleep(1000);
            } catch (Exception e){
                System.out.println("Thread interrompida.");
            }
            System.out.println("\n" + msg + "Enviada.");
        }
    }
}
```

Mais uma implementação alternativa para demonstrar que sincronizado pode ser usado com apenas uma parte método.

63

Java e Paralelismo – Threads (**Recursos**)

Nesse caso a melhor solução para não causar problemas é liberar temporariamente o controle do objeto permitindo que outra Thread seja executada.

Para isto, usaremos três métodos:

- **Wait()**
- **Notify()**
- **NotifyAll()**

64

Java e Paralelismo – Threads (Recursos)

Para isto, usaremos três métodos:

- **Wait():** Este método bloqueia a execução da Thread temporariamente, ou seja, coloca a Thread em modo de espera.
- **Notify():** Este método notifica uma Thread que estava esperando, ou seja, retorna a execução da Thread.
- **NotifyAll():** Este método notifica todas as Threads, e a que tem prioridade mais alta ganha acesso ao objeto.

65

Java e Paralelismo – Threads (Recursos)

Exemplo:

```

public class TiqueTaque {
    boolean tique;
    synchronized void tique(boolean Executando){
        if (!Executando){
            tique = true;
            notify();//Notifica qualquer Thread que estiver executando.
            return;
        }
        System.out.println("Tique ");
        tique = true;
        notify();
        try{
            while (tique){
                wait(); //Aguarda Tique executar.
            }
        }catch(InterruptedException e){
            System.out.println("A Thread foi Interrompida.");
        }
    }
}

```

Continua...

66

Java e Paralelismo – Threads (Recursos)

Exemplo:

```
boolean taque;
synchronized void taque(boolean Executando){
    if (!Executando){
        tique = false;
        notify();//Notifica qualquer Thread que estiver executando.
        return;
    }
    System.out.println("Taque ");
    tique = false;
    notify();
    try{
        while (!tique){
            wait(); //Aguarda Tique executar.
        }
    }catch(InterruptedException e){
        System.out.println("A Thread foi Interrompida.");
    }
}
```

Continua...

67

Java e Paralelismo – Threads (Recursos)

Exemplo:

```
public class ThreadTiqueTaque implements Runnable {
    Thread t1;
    TiqueTaque tt;
    final int QTD = 5;
    public ThreadTiqueTaque(String nome, TiqueTaque tt) {
        t1 = new Thread(this, nome);
        this.tt = tt;
        t1.start();
    }
}
```

Continua...

68

Java e Paralelismo – Threads (Recursos)

Exemplo ainda no mesmo método :

```
public void run() {
    if (t1.getName().equalsIgnoreCase("Tique")) {
        for (int i = 0; i < QTD; i++) {
            tt.tique(true);
        }
        tt.taue(false);
    } else {
        for (int i = 0; i < QTD; i++) {
            tt.taue(true);
        }
        tt.tique(false);
    }
}
```

Continua...

69

Java e Paralelismo – Threads (Recursos)

Exemplo ainda no mesmo método :

```
public static void main(String[] args) {
    TiqueTaue tt = new TiqueTaue();
    ThreadTiqueTaue tique = new ThreadTiqueTaue("Tique", tt);
    ThreadTiqueTaue taue = new ThreadTiqueTaue("Taue", tt);
    try {
        tique.t1.join();
        taue.t1.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

70

Java e Paralelismo – Threads (**Recursos**)

Como resultado teremos uma alternância entre “Tique” e “Taque”

Cada um será executado **5 vezes**:

```
Tique  
Taque  
Tique  
Taque  
Tique  
Taque  
Tique  
Taque  
Tique  
Taque
```

71

Dúvidas?



Estácio



72