Programação Concorrente com Thread Java

Adaptado de Luiz Affonso Guedes

Sistemas Distribuidos

Definições Básicas

- Threads são sub-procesos no sistema operacional.
- É menos custoso gerenciar threads do que processos.
  - Para ambos: SO e programador
- A linguagem Java possui suportes a threads na própria estrutura da linguagem.
- C e C++ necessitam de bibliotecas especificas para processamento multi threads
  - POSIX Threads

Threads em Java

- Em Java, threads são implementadas como objetos
  - Pacote java.lang
  - São extensões da classe Thread
  - Contrutores:
    - public Thread (String nome_da_thread);
    - public Thread ( ); // o nome sera Thread-#
      - » Thread-1, Thread-2,...

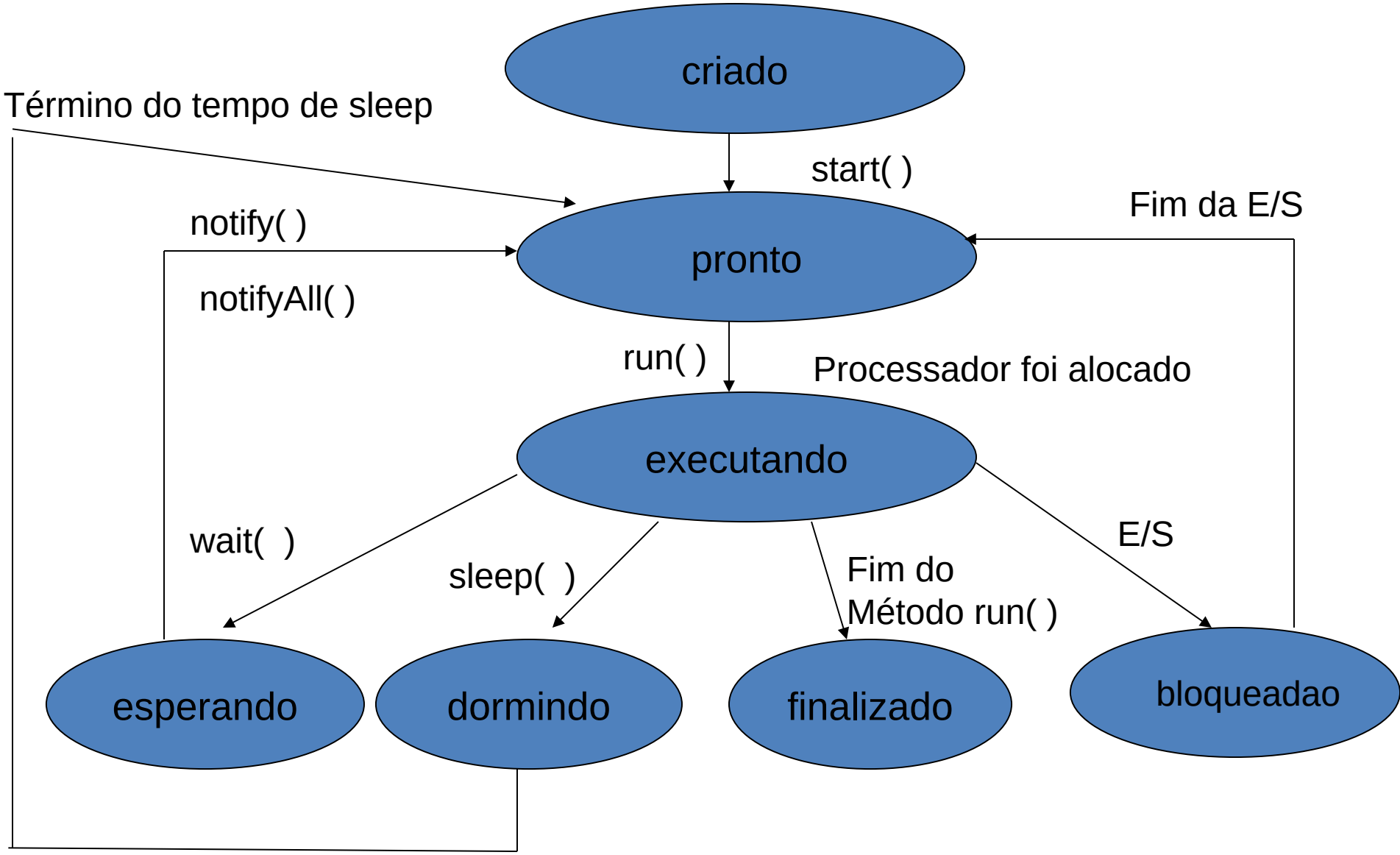
Principais Métodos

- run(): é o método que executa as atividades de um Thread. Quando este método finaliza, a THREAD também termina.
- start(): método que dispara a execução de um Thread. Este método chama o método run( ).
- sleep(int x): método que coloca a THREAD para dormir por x milisegundos.

Principais Métodos

- join(): método que espera o término do Thread para qual foi enviada a mensagem para ser liberada.
- interrupt( ): método que interrompe a execução de um Thread.
- interrupted( ): método que testa se um Thread está ou não interrompido.

Estados de uma Thread em Java



Exemplo: Threads em Java

```
class Inicio {
    public static int NUM_THREADS = 10; // constante
    public static void main( String args[] ) {
        int i;
        Thread[] threads = new Thread[Inicio.NUM_THREADS];
        ClasseTeste tmpobj = new ClasseTeste();
        System.out.println( "Inicio do programa" );
        for ( i = 0; i < 10; i++ ) {
            threads[i] = new Thread( tmpobj );
            threads[i].start();
        }
    }
}
class ClasseTeste implements Runnable {
    public void run() {
        System.out.println("Teste");
    }
}
```

Primeiro exemplo:
Uso de threads em Java

Exemplo: Threads em Java (2)

- Segundo exemplo: Variável `contador` do objeto tmpobj será incrementada por 10 threads

- Contador é o recurso compartilhado pelos 10 threads
  - Todos atualizarão contador e mostrarão seu resultado na tela

Monitores e Threads em Java (3)

```
class Inicio2 {
    public static int NUM_THREADS = 10; // constante
    public static void main( String args[] ) {
        int i;
        Thread[] threads = new Thread[Inicio.NUM_THREADS];
        ClasseTeste2 tmpobj = new ClasseTeste2();
        System.out.println( "Inicio do programa" );
        for ( i = 0; i < 10; i++ ) {
            threads[i] = new Thread( tmpobj );
            threads[i].start();
        }
    }
}
class ClasseTeste2 implements Runnable {
    private int contador;
    public ClasseTeste2() {
        this.contador = 0;
    }
    public void run() {
        this.contador++;
        System.out.println( "Contador: " + this.contador );
    }
}
```

- Recurso compartilhado
- Método executado pelos threads

Monitores e Threads em Java (4)

- Saída possível:

```
Inicio do programa
Contador: 4
Contador: 4
Contador: 4
Contador: 4
Contador: 5
Contador: 6
Contador: 7
Contador: 8
Contador: 9
Contador: 10
```

- Problema ocorreu pois o não se aplicou a exclusão mútua no recurso compartilhado: `contador`

Monitores e Threads em Java (5)

- Utilizamos o synchronized para proteger uma região crítica

- Podemos usar o monitor do próprio objeto
- Ex:

```
synchronized {
}
```

- ou de um outro objeto

```
synchronized(contador) {
}
```

Monitores e Threads em Java (6)

```
class ClasseTeste2 implements Runnable {
    private  Integer contador;
    public ClasseTeste2() {
        contador = 0;
    }
    public void run() {
        synchronized (contador) {
            System.out.println( "Contador: " + contador );
            contador++;
        }
    }
}
```

Exercício 01

- Criar um programa que inicialize um vetor com 1000000 de elementos, cada elemento contendo o valor 1 e que dispare 4 threads que efetue a soma total do vetor e exiba no metodo main.
- Crie uma classe chamada Somadora que implemente a interface Runnable
  - Essa classe deverá ter um atributo privado chamado nt (numero do thread).
  - Crie um construtor de 1 argumento int que receba o nt e salve no nt do objeto

Exercício 01

- Crie uma classe chamada Somadora que implemente a interface Runnable (continuação)
  - Crie um atributo private static Long somaTotal = 0; // na classe Somadora e inicialize ele com 0
    - No método run, faça um for que gere a soma parcial de cada thread, começando em nt e incrementando o contador sempre por nt (for i = nt; i < 10000000; i = i + nt)
    - Faça um trecho synchronized para atualizar a variável total da classe Somadora
    - Crie um método getSomaTotal na classe Somadora que retorne a somaTotal para o método main

A interface Runnable

- Para utilizar multithreads em Java é necessário instanciar um objeto de uma classe que estende a classe básicaThread, certo?
- Uma vez que Java não possui herança múltipla, como eu posso utilizar um objeto, cuja classe já é derivada, como no caso da ClasseThread?
  - public class Filho extends Pai extends Thread {

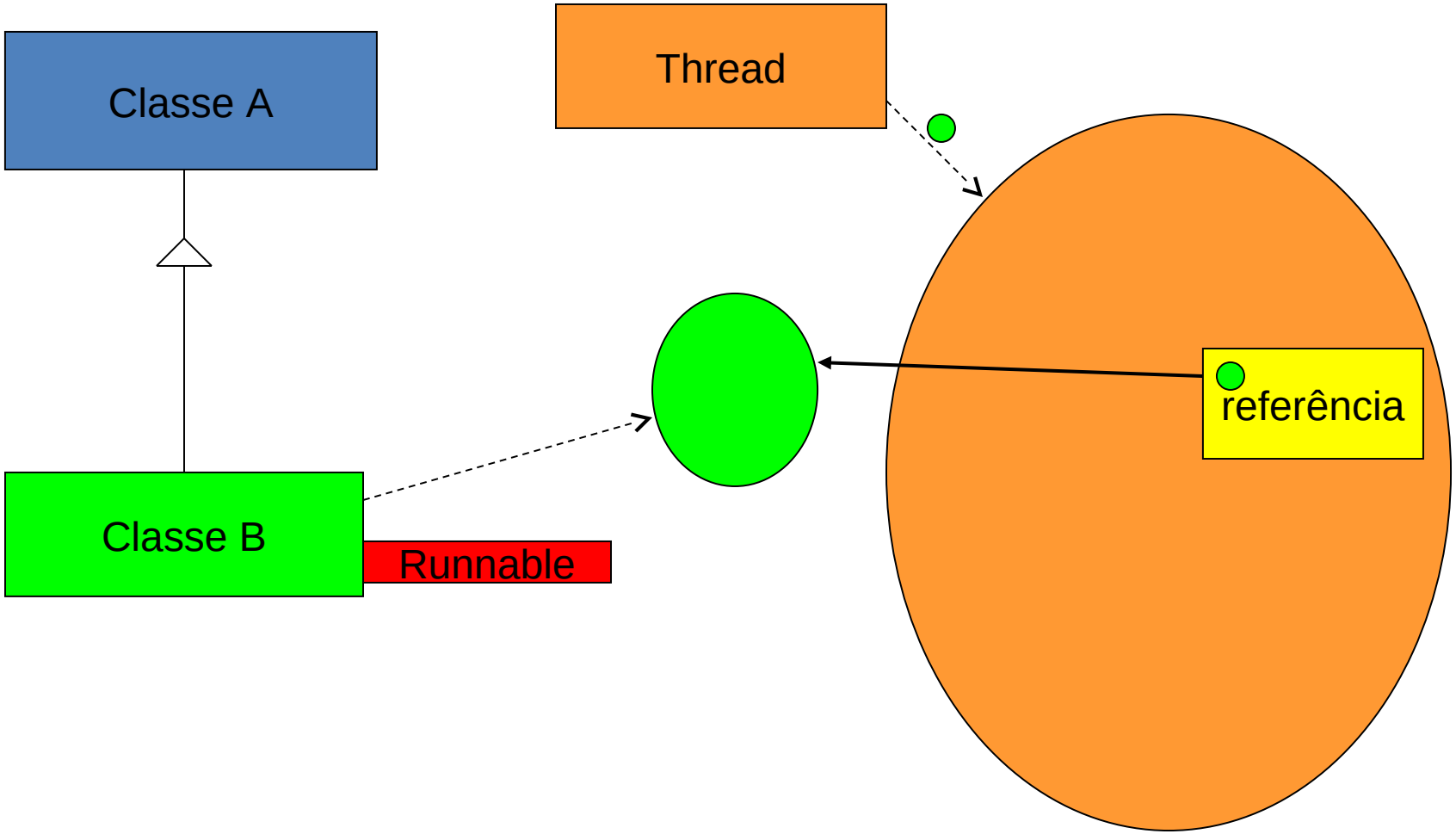
    .....

    } // isto nao eh possivel em Java

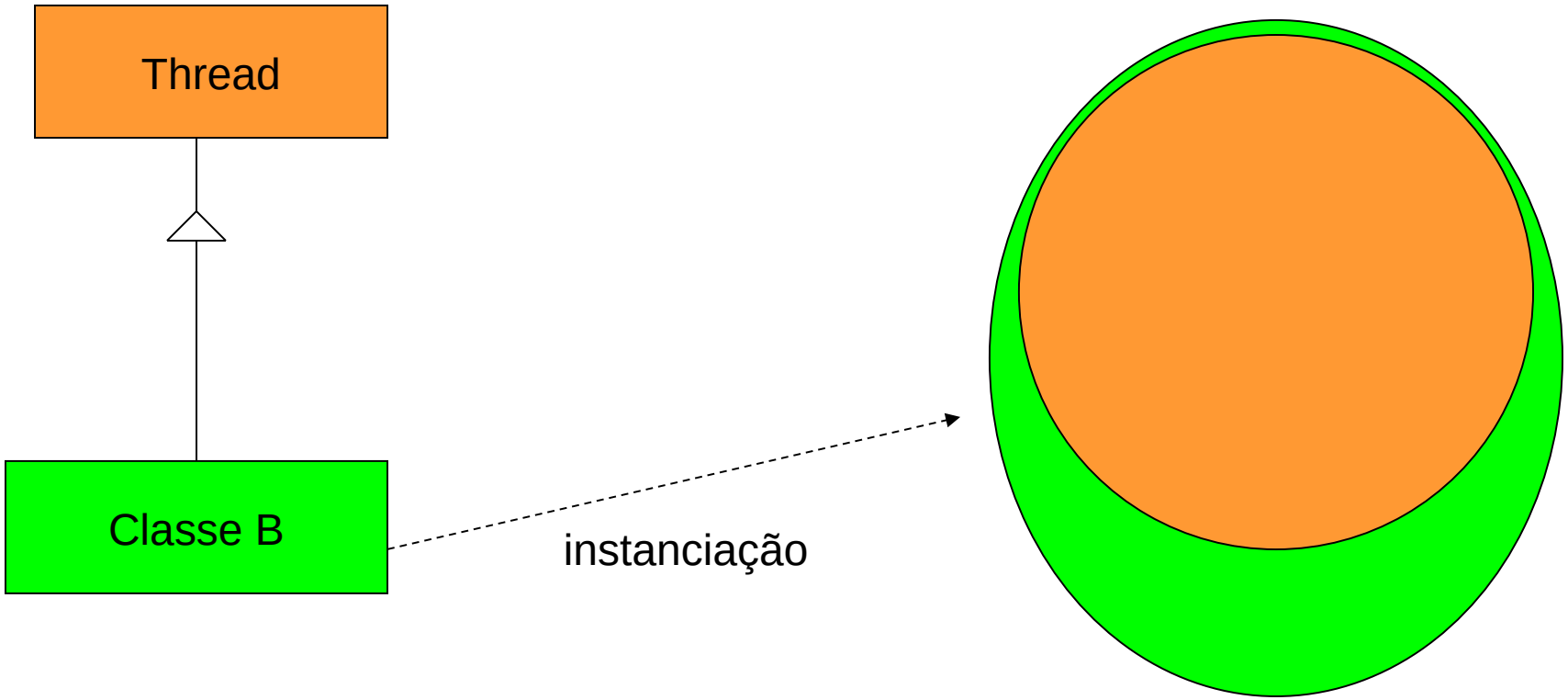
A interface Runnable

- A solução encontrada em Java foi a utilização de uma interface: Runnable
  - No caso, tem-se de implementar esta interface, que possui o método run( ).
  - public class Filho extends Pai implements Runnable
  - Ao implementar uma interface, a classe se capacita a ser tratada como se fosse um objeto do tipo da inteface implementada.
    - Se a classe Filho implementar a interface Runnable, ela pode ser tratada como tal.

Solução Baseada na Interface Runnable



Solução Baseada em Herança da Classe Thread



Exercício

- Converta o programa abaixo de forma que ele use a Interface Runnable ao invés de estender Thread.

```
class MyTask extends Thread {
  public MyTask(String name){
    super(name);
  }
  public void run(){
      for (int i=10; i!=0; i--){
      try{
        this.sleep(200 + (int)
(Math.random()*100));

System.out.println(this.getName());
      }catch (InterruptedException ie){
        System.out.println(ie);
      }
    }
  }
}
```

Problemas de Concorrência

- Ocorrem quando mais que um thread tenta acessar algum recurso compartilhado simultâneamente
- Definições
  - Recurso Compartilhado
  - Região Crítica
  - Exclusão Mútua
- Realizar Exclusão Mútua
  - Métodos e blocos synchronized

Problema: Depósito de Caixas

```
public class Deposito {
private int items=0;
private final int capacidade=10;
public int retirar() {
if (items>0) {
items--;
System.out.println("Caixa retirada: Sobram "+items+" caixas");
return 1; }
return 0;
}
public int colocar () {
if (items<capacidade) {
items++;
System.out.println("Caixa armazenada: Passaram a ser "+items+"
caixas");
return 1; }
return 0;
}
public static void main(String[] args) {
Deposito dep = new Deposito();
Produtor p = new Produtor(d, 2);
Consumidor c = new Consumidor(d, 1);
//arrancar o produtor
//...
//arrancar o consumidor
//...
System.out.println("Execucao do main da classe Deposito terminada");
}
```

Sincronização em Java