UNIP
UNIVERSIDADE PAULISTA

Linguagem de Programação Orientada a Objeto

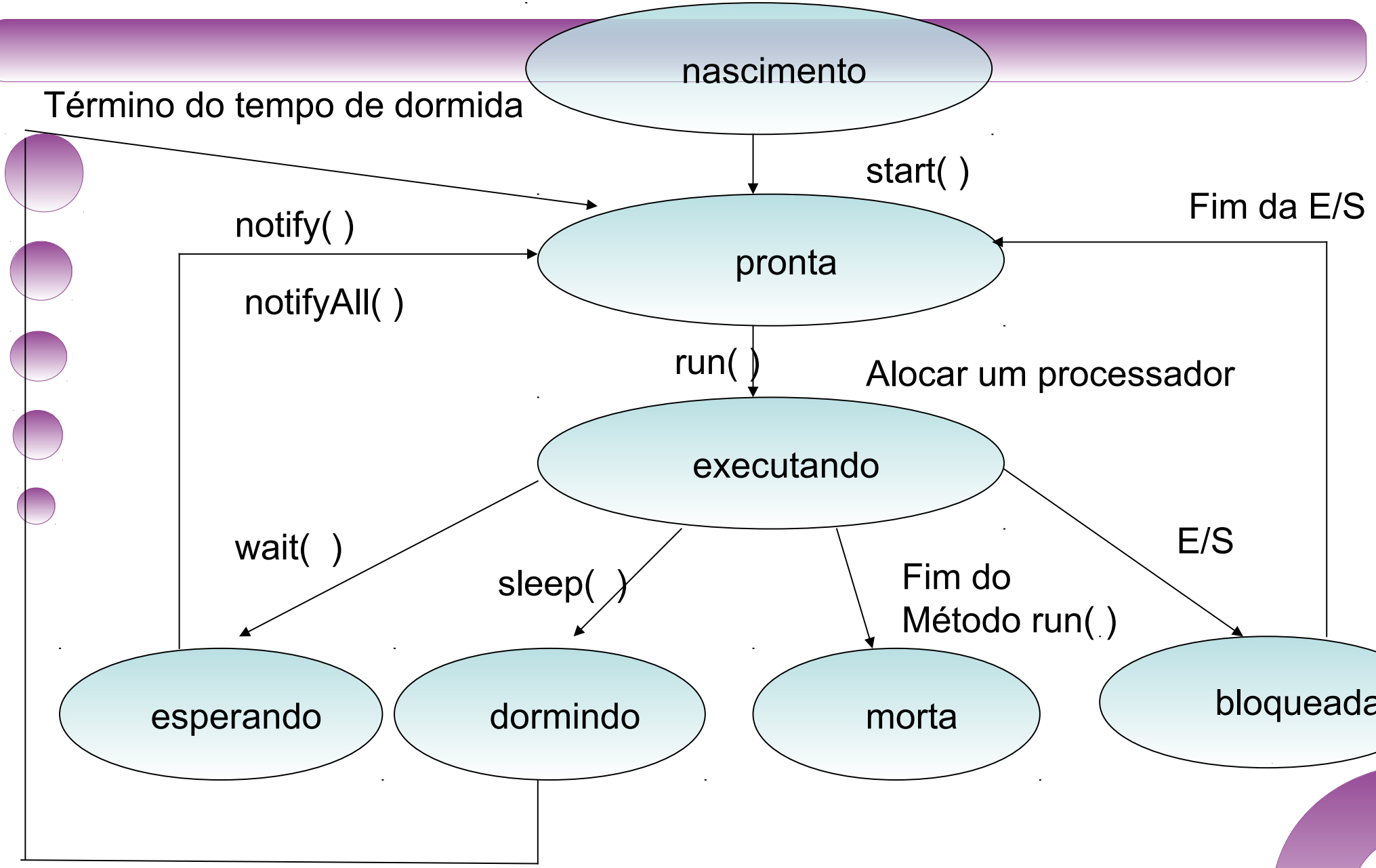
Threads

Professora Sheila Cáceres

Introdução

- Com frequência precisamos rodar mais de uma tarefa ao mesmo tempo (ex: navegar pela internet e ouvir música).
- Podemos executar tarefas em paralelo (**concorrentemente**) usando Threads (ex. Uma thread para navegar pela internet e outra para ouvir música)
- Em Java, usamos a classe **Thread** do pacote java.lang para criarmos linhas de execução paralelas

Estados de uma Thread em Java



Criação de Threads

Existem duas maneiras de criar threads em java:

Maneira 1

- Criar uma classe que herda de java.lang.Thread e nela sobrescrever o método run() contendo os comandos que a thread deverá executar.
  - O método run da classe Thread não faz nada (não é abstrato).

```
class MinhaThread extends Thread{
    public void run(){
      <instruções da thread>
    }
}
```

- A seguir, criamos instancias da classe e as iniciamos chamando o método ***start()*** que rodará o ***run()***

```
MinhaThread t = new MinhaThread();
t.start();
```

Maneira 2

- Criar uma classe que implementa a interface ***Runnable.***
- Runnable tem só um método: ***run()***.
- Nossa classe seria obrigada a implementar o método ***run*** contendo os comandos que a thread executará.

```
class MinhaClasse implements Runnable{
    public void run(){
      <instruções da thread>
    }
...outros métodos, construtores, atributos, etc
}
```

- A seguir, em outro lugar, criamos uma instância de MinhaClasse e passamos a instância como argumento do construtor da classe Thread:

```
MinhaClasse miClasse = new MinhaClasse();
Thread t = new Thread(miClasse);
t.start(); // igual que na Maneira 1
```

Criação de Threads

- Cada vez que enviar a mensagem ***start***() para uma instância de uma **thread**, uma nova linha de execução será iniciada com os comandos do método ***run***(), que rodará em paralelo com as outras threads.
- Não pode-se enviar a mensagem **run**() diretamente para um objeto Thread. Envia-se a mensagem **start**(), que criará a thread onde rodarão os comandos do método **run**().
- Como vimos, o método **run**() pode ser implementado na classe filha de Thread (maneira 1) ou numa classe que implementa Runnable (maneira 2) e é passado como parâmetro para a Thread.

Qual é a melhor maneira?

Herdar de Thread:

- Codificação mais simples;
- Herdamos muitos de métodos mas apenas precisamos usar o run.
- Nem sempre é possível herdar de threads. Se a classe que precisamos rodar como Thread já for subclasse de outra não poderia herdar de Thread, nesse caso só poderiamos implementar Runnable.

Implementar Runnable:

- Melhor design OO;
- Mais consistente;

Prefira implementar Runnable a herdar de Thread.

Principais Métodos

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

  Ex: t.sleep(3000). // thread t dormirá por 3 segundos

Principais Métodos

- join( ): método que espera o término da THREAD para qual foi enviada a mensagem para ser liberada.
- interrupt( ): método que interrompe a execução de uma THREAD.
- interrupted( ): método que testa se uma THREAD está ou não interrompida.

Exemplo

```
public class Tarefa implements Runnable{
   private int tempoADormir; // random
   private String nomeThread;
   private static Random generator = new Random();

   public Tarefa(String name){
      nomeThread = name;
      // tempo randomico entre 0 e 5 segundos
      tempoADormir = generator.nextInt( 5000 );
   }

   // codigo que será executado pelo thread
   public void run(){
      try {
         System.out.println( nomeThread+" vai dormir por "+
             tempoADormir+" milisegundos.");
         Thread.sleep( tempoADormir );
      }
      // catch será chamado se o thread é interrumpido enquanto dorme
      catch ( InterruptedException exception ){
         exception.printStackTrace();
      }
      System.out.println(nomeThread+ " acordou");
   }
}
```

Exemplo

```
public class CreadorTarefas{
   public static void main( String[] args ){
      System.out.println( "Criando as threads" );

      // Criamos threads com Runnable Tarefa
      Thread thread1 = new Thread( new Tarefa( "tarefa1" ) );
      Thread thread2 = new Thread( new Tarefa( "tarefa2" ) );
      Thread thread3 = new Thread( new Tarefa( "tarefa3" ) );

      System.out.println( "Threads creadas, start()." );

      // Rodamos os threads
      thread1.start(); // invoca metodo run da tarefa1
      thread2.start(); // invoca metodo run da tarefa2
      thread3.start(); // invoca metodo run da tarefa3

      System.out.println( "Tarefas rodando, fim do main." );
   }
}
```

```
Criando as threads
Threads criadas, start().
Tarefas rodando, fim do método main.

tarefa1 vai dormir por 2666 milisegundos.
tarefa2 vai dormir por 4275 milisegundos.
tarefa3 vai dormir por 821 milisegundos.
tarefa3 acordou
tarefa1 acordou
tarefa2 acordou
```

```
Criando as threads
Threads criadas, start().
tarefa1 vai dormir por 4136 milisegundos.
Tarefas rodando, fim do método main.

tarefa2 vai dormir por 2677 milisegundos.
tarefa3 vai dormir por 2873 milisegundos.
tarefa2 acordou
tarefa3 acordou
tarefa1 acordou
```

Outro Exemplo

```
public class ImprimidorNumeros implements Runnable {
    private int id;
    public ImprimidorNumeros(int id) {
        this.id=id;
    }

    public void run() {
        for (int i = 0; i < 1000; i++) {
            System.out.println("Imprimidor " +
                this.id + " valor: " + i);
        }
    }
}
```

```
public class TesteImprimidorNumeros {

    public static void main(String[] args) {
        ImprimidorNumeros p1 = new ImprimidorNumeros(1);
        Thread t1 = new Thread(p1);
        t1.start();

        ImprimidorNumeros p2 = new ImprimidorNumeros(2);
        Thread t2 = new Thread(p2);
        t2.start();
    }
}
```

Saida

```
Imprimidor 1 valor: 0
Imprimidor 1 valor: 1
Imprimidor 1 valor: 2
...
Imprimidor 1 valor: 13
Imprimidor 1 valor: 14
Imprimidor 2 valor: 0
Imprimidor 2 valor: 1
Imprimidor 1 valor: 15
Imprimidor 1 valor: 16
...
Imprimidor 1 valor: 29
Imprimidor 1 valor: 30
Imprimidor 1 valor: 31
Imprimidor 2 valor: 2
Imprimidor 2 valor: 3
...
Imprimidor 2 valor: 99
Imprimidor 2 valor: 100
Imprimidor 2 valor: 101
Imprimidor 1 valor: 32
Imprimidor 1 valor: 33
Imprimidor 1 valor: 34
...
```

- Ao rodar o último exemplo ele vai intercalar a saida do primeiro e do segundo thread.
- O processador só consegue fazer uma coisa de cada vez mas deseja-se que as threads rodem simultaneamente.
- O escalonador de threads (scheduler) administra o que o processador vai executar e fica alternando a execução de cada thread.
- A ideia é executar um pouco de cada thread e fazer essa troca tão rapidamente que da a impressão que as coisas estão sendo feitas ao mesmo tempo.

Bibliografia

- Java: Como programar.

  Autores: H. M. Deitel e P. J. Deitel

  Editora: Pearson – 9a Edição.
- Caelum: http://www.caelum.com.br/apostila-java-orientacao-objetos/programacao-concorrente-e-threads/

  Acessado o 11 de maio do 2014.