Introdução às Java Threads

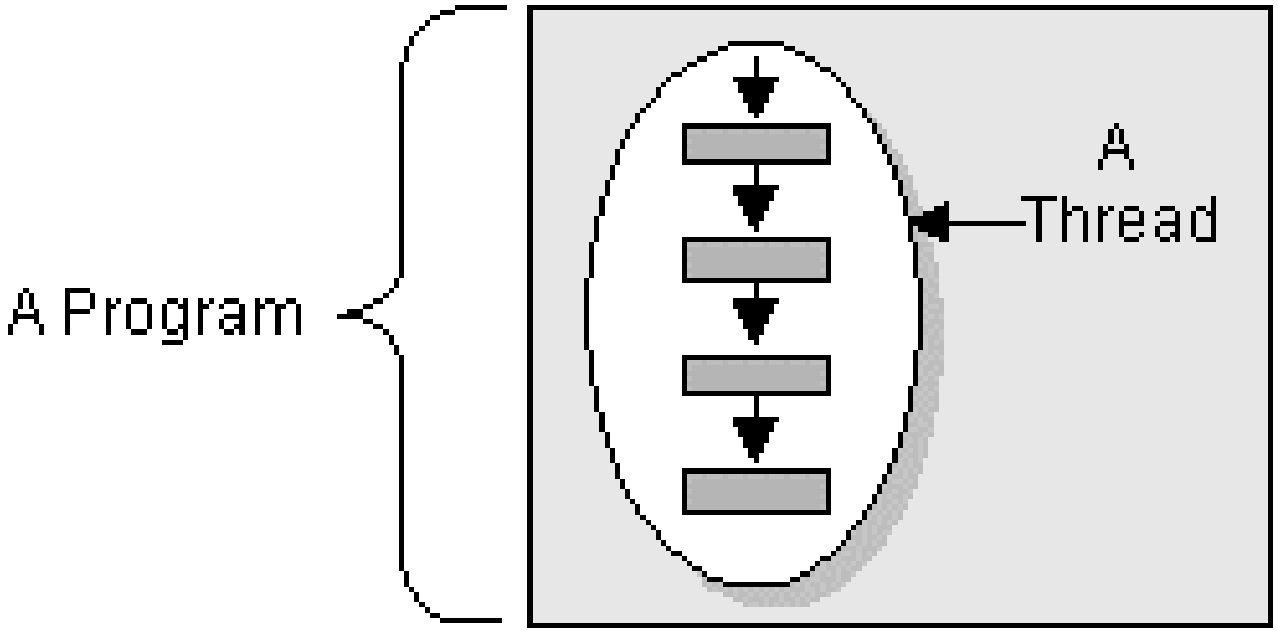
Vitor Brandi Junior

Sumário

- Introdução
- Para que servem as threads ?
- Qual a novidade ?
- O método `run()`
- Criando threads
- Implementando a interface `Runnable`
- Exemplos
- O método `sleep()`
- Estendendo a classe `Thread`
- Mais exemplos

Introdução

- O que é uma thread ?
  - Uma thread (também denominada contexto de execução ou processo peso-leve) é um fluxo sequencial único de execução dentro de um programa. Graficamente:

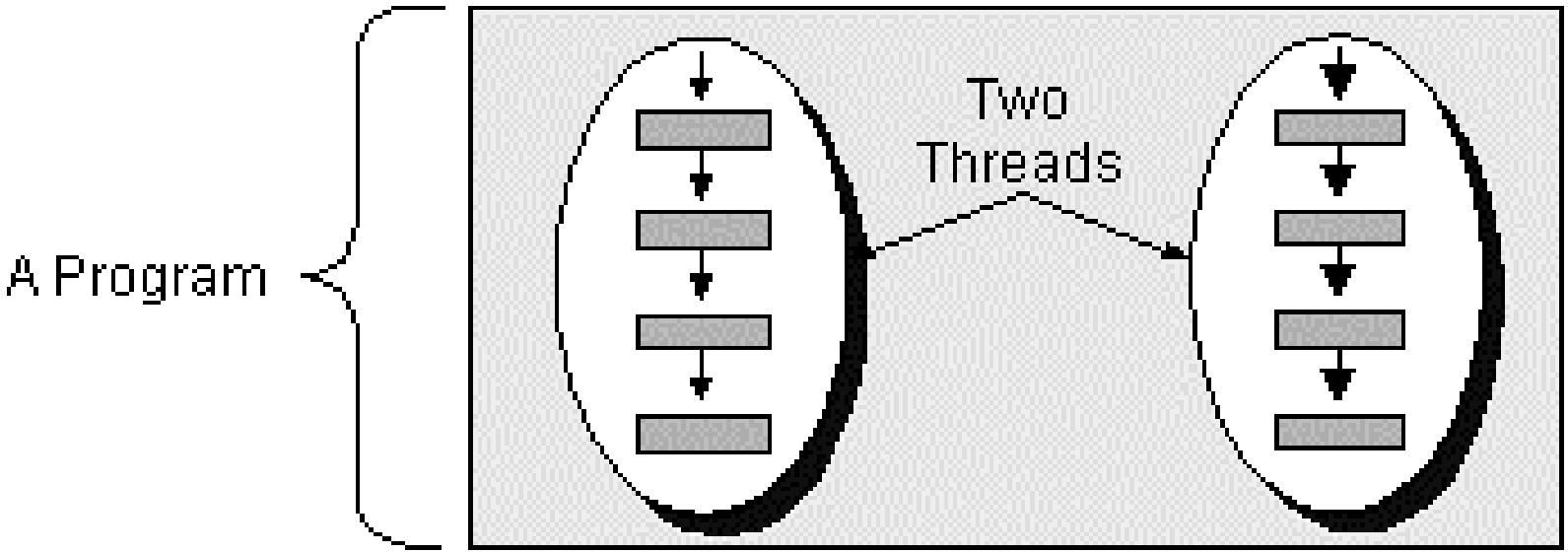


Para que ele serve ?

- Para que serve uma thread ?
  - Basicamente ela permite executar tarefas isoladamente
  - funciona de maneira semelhante aos processos do Unix, só que consumindo menos recursos, visto que utiliza-se de muitos dos recursos já previamente alocados para o programa dentro do qual ela está definida
  - claramente não existe nada de novo em relação à existência de uma única thread dentro de um programa

Então qual é a novidade ?

- Quando define-se mais de uma thread em um mesmo programa, as tarefas que elas contém podem ser executadas de maneira simultânea e independente uma da outra. Graficamente:



O método `run()`

- Este método provê alguma coisa para a thread fazer
- seu código é responsável por implementar o comportamento que a thread vai ter quando ela estiver sendo executada
- qualquer tarefa pode ser implementada dentro do método `run()`, desde que observadas algumas regras para sua definição (vide mais adiante)

Criando threads

- Existem duas maneiras básicas de um programa implementar múltiplas threads:
  - implementar a interface `Runnable`
  - ser subclasse (herdar ou estender) da classe `Thread`
- Ambas as técnicas são equivalentes, sendo a primeira apropriada para contornar a inexistência de herança múltipla em Java

A interface Runnable

- Ela é assim definida:

```
public interface Runnable
{
   abstract public void run();
}
```

- Cada thread inicia sua vida executando o método `run()` no objeto `Runnable` que foi passado ao thread
- o método `run()` pode conter qualquer código, mas precisa ser público, não usar argumentos, não ter valor de retorno e não gerar exceções

A interface `Runnable`

- Qualquer classe que contém um método `run()` com estas características pode declarar que implementa a interface `Runnable`
- uma instância desta classe é então um objeto que pode servir como destino de um novo thread
- no entanto não basta somente definir o método `run()`. É preciso invocá-lo para que a thread seja "despertada" e comece a sua execução

Como a thread se inicia

- Obrigatoriamente quem desperta a thread é invocação do método `start()`
- o método `start()` tem uma única responsabilidade, que é invocar o método `run()` definido
- uma vez chamado o método `start()`, a thread permanecerá sendo executada até que o método `run()` se encerre
- o método `start()` somente pode ser invocado uma única vez durante o tempo de vida de uma thread

Resumindo...

- Em resumo os passos necessários são:
  - declarar uma classe multithread que implementa `Runnable` e portanto, possui um método `run()`
  - instanciar o objeto multithread
  - instanciar um objeto `Thread` e passar o objeto multithread como parâmetro no construtor
  - invocar o método `start()` da classe `Thread`

Esquematicamente ...

```
class UmaClasse implements Runnable
{
  ... // declaração de atributos
  ... // e outros métodos da classe
  public void run()
  {
    while (condição)
    {
      ... // comandos que definem a
      ... // funcionalidade do thread
    }
  }
}
... // em algum outro lugar
UmaClasse umaClasse = new UmaClasse();
Thread novoThread = new Thread(umaClasse);
novoThread.start();
```

Um exemplo (meio imbecil)

```
public class TesteThread0 {
  public void poeParaQuebrar() {
    UmaClasse umaClasse = new UmaClasse();
    Thread novoThread = new Thread(umaClasse);
    novoThread.start();
  }
  public static void main(String args[]) {
    TesteThread0 t = new TesteThread0();
    t.poeParaQuebrar();
  }
}
class UmaClasse implements Runnable {
  int i = 0;
  public void run() {
    while (i < 20) {
      System.out.println(i);
      i+=2;
    }
  }
}
```

Outro exemplo (menos imbecil)

```
public class TesteThread0 {
  public void poeParaQuebrar() {
    UmaClasse umaClasse = new UmaClasse();
    Thread umThread = new Thread(umaClasse);
    OutraClasse outraClasse = new OutraClasse();
    Thread outroThread = new Thread(outraClasse);
    umThread.start();
    outroThread.start();
  }
  public static void main(String args[]) {
    TesteThread0 t = new TesteThread0();
    t.poeParaQuebrar();
  }
}
class OutraClasse implements Runnable {
  int i = 1;
  public void run() {
    while (i <= 20) {
      System.out.println(i); i+=2;
    }
  }
}
class UmaClasse implements Runnable {
  int i = 0;
  public void run() {
    while (i <= 20) {
      System.out.println(i); i+=2;
    }
  }
}
```

`sleep(tempo)`

- Pode ser necessário fazer um thread ficar ocioso (dormir) por um certo período de tempo
- enquanto ele está adormecido não consome tempo de CPU ou compete com o processamento de outros threads
- para realizar esta tarefa existe o método `sleep(tempo em milissegundos)`, que pode ser invocado como método de instância do thread ou como método estático da classe `Thread` (mais comum)

Exemplo de sleep()

```
public class TesteThread0 {
  public void poeParaQuebrar() {
    UmaClasse umaClasse = new UmaClasse();
    Thread umThread = new Thread(umaClasse);
    OutraClasse outraClasse = new OutraClasse();
    Thread outroThread = new Thread(outraClasse);
    umThread.start();
    outroThread.start();
  }
  public static void main(String args[]) {
    TesteThread0 t = new TesteThread0();
    t.poeParaQuebrar();
  }
}
class OutraClasse implements Runnable {
  int i = 1;
  public void run() {
    while (i <= 20) {
      System.out.println(i); i+=2;
      try {
        Thread.sleep(1000); //”adormece” por 1 segundo
      }
      catch (InterruptedException ie) {}
    }
  }
}
class UmaClasse implements Runnable {
  int i = 0;
  public void run() {
    while (i <= 20) {
      System.out.println(i); i+=2;
    }
  }
}
```

Estender a classe `Thread`

- É a segunda técnica de se criar um thread
- guarda muitas semelhanças com a primeira técnica, posto que a classe `Thread` também implementa a interface `Runnable`
- a única diferença é que deve-se substituir o método `run()` implementado pela classe `Thread` (uma vez que ele não faz absolutamente nada)

Resumidamente...

- Em resumo os passos necessários são:
  - declarar uma classe multithread que estenda (seja subclasse) da classe `Thread`
  - criar um objeto desta classe
  - invocar o método `start()` desta classe

Esquematicamente ...

```
class UmaClasse extends Thread
{
  ... // declaração de atributos
  ... // e outros métodos da classe
  public void run()
  {
    while (condição)
    {
      ... // comandos que definem a
      ... // funcionalidade do thread
    }
  }
}
... // em algum outro lugar
UmaClasse umaClasse = new UmaClasse();
novoThread.start();
```

Mais um Exemplo

```
public class TesteThread0 {
  public void poeParaQuebrar() {
    UmaClasse umaClasse = new UmaClasse();
    novoThread.start();
  }
  public static void main(String args[]) {
    TesteThread0 t = new TesteThread0();
    t.poeParaQuebrar();
  }
}
class UmaClasse extends Thread {
  int i = 0;
  public void run() {
    while (i < 20) {
      System.out.println(i);
      i+=2;
    }
  }
}
```

Mais outro exemplo

```
class Assobiar extends Thread {
  public void run(){
    for (int i=0; i<100; i++){
      System.out.println("Assobiando");
      try {
        Thread.sleep(100);
      }
      catch (InterruptedException e){}
    }
  }
}
class Cana implements Runnable {
  public void run() {
    for (int i=0; i<100; i++) {
      System.out.println("Chupando cana");
      try {
        Thread.sleep(100);
      }
      catch (InterruptedException e) {}
    }
  }
}
public class TesteThread {
  public static void main (String args[]) {
    Assobiar assobiar = new Assobiar();
    Thread cana = new Thread(new Cana());
    assobiar.start();
    cana.start();
  }
}
```

Último exemplo: um relógio

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
public class Teste extends JFrame implements ActionListener {
  public JLabel l1, l2;
  JButton botao;
  public Teste() {
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e){System.exit(0);}});
    setSize(300,200);
    this.getContentPane().setLayout(new BorderLayout());
    botao = new JButton("Teste");
    this.getContentPane().add("North",botao);
    botao.addActionListener(this);
    l1 = new JLabel("Um label");
    this.getContentPane().add("Center", l1);
    l2 = new JLabel("");
    this.getContentPane().add("South", l2);
  }
  public void actionPerformed(ActionEvent evt) {
    l1.setText("Atualizado pelo setText");
  }
  public void atualizaLabel() {
    Date hoje = new Date();
    l2.setText(" " +(hoje.getHours()) + ":" + hoje.getMinutes() + ":" + hoje.getSecon
  }
  public static void main(String[] args) {
    Teste f = new Teste();
    Thread t = new Thread(new Relogio());
    t.start();
  }
}
```

Último exemplo: um relógio

```
class Relogio extends Teste implements Runnable
{
  public void run()
  {
    setVisible(true);
    while (true)
    {
      atualizaLabel();
      try
      {
        Thread.sleep(1000);
      }
      catch (InterruptedException e){}
    }
  }
}
```