

Programação Orientada a Objetos – Aula 11

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Conceito de Multitarefa

- **Processo** é um programa que está sendo executado em um sistema operacional
- **Thread** é a menor unidade de código que poder ser executada, ou seja, um programa pode executar 2 ou mais tarefas ao mesmo tempo
- Exemplo
 - Editor de texto é um programa que você pode formatar um texto (ao clicar em um botão) ao mesmo tempo que envia ele para impressão em uma impressora

Threads

- Uma thread é uma linha de execução, um fluxo de execução, um segmento de programa executando dentro da CPU
- Programação *multithreading* consiste em pensar e escrever as aplicações de forma que ações sejam executadas de forma concorrente
- Apenas computadores com múltiplos processadores podem de fato executar instruções em paralelo, porém, sistemas operacionais modernos oferecem recursos para criar uma “ilusão” de paralelismo

Programação Concorrente & Java

- Java disponibiliza a concorrência de forma nativa por meio da linguagem e de seu *framework*
- Ao especificar uma *thread* estamos criando um fluxo de execução independente, com sua própria pilha de chamadas de métodos e contador de programa
- Java inclui primitivas de *multithreading* de forma integrada à linguagem de programação, não sendo necessário chamar primitivas específicas do sistema operacional
- A classe Thread representa um fluxo de execução independente para a linguagem Java

Ciclo de vida de uma thread



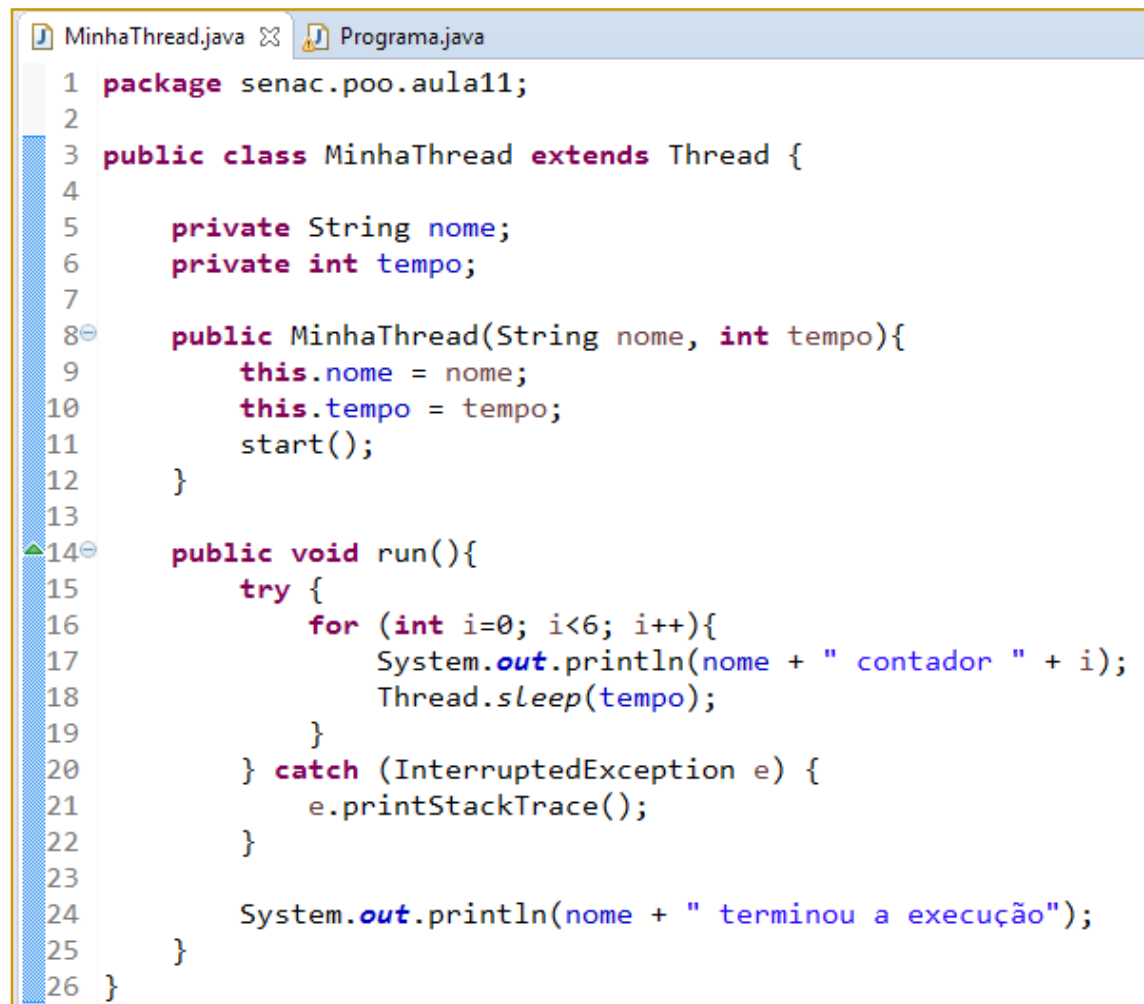
Criando uma thread

- Estender a classe Thread
- Implementar a interface Runnable

Alguns métodos da classe thread

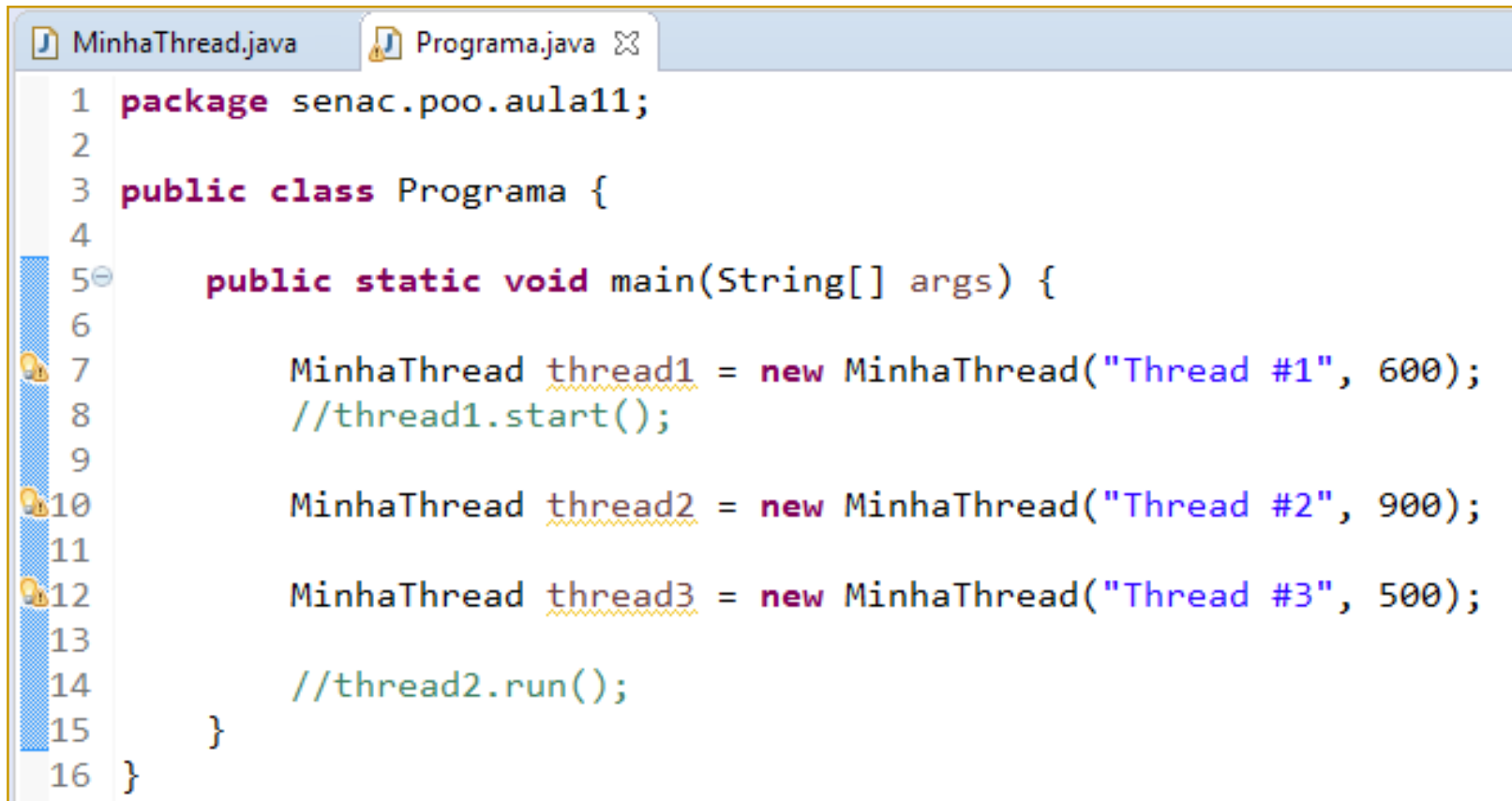
- start
 - Registra a thread no thread scheduler
- run
 - Executa a tarefa da thread, deve estar presente em todas as threads
- sleep
 - Faz com que a thread fique em estado de espera uma quantidade mínima de tempo, em ms, possibilitando a CPU executar outras threads
- getName/setName
 - Retorna ou atribui o nome de uma thread, por default as threads são nomeadas numericamente

Exemplo com herança



```
1 package senac.poo.aula11;
2
3 public class MinhaThread extends Thread {
4
5     private String nome;
6     private int tempo;
7
8     public MinhaThread(String nome, int tempo){
9         this.nome = nome;
10        this.tempo = tempo;
11        start();
12    }
13
14    public void run(){
15        try {
16            for (int i=0; i<6; i++){
17                System.out.println(nome + " contador " + i);
18                Thread.sleep(tempo);
19            }
20        } catch (InterruptedException e) {
21            e.printStackTrace();
22        }
23
24        System.out.println(nome + " terminou a execução");
25    }
26 }
```


Exemplo com herança



```
1 package senac.poo.aula11;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         MinhaThread thread1 = new MinhaThread("Thread #1", 600);
8         //thread1.start();
9
10        MinhaThread thread2 = new MinhaThread("Thread #2", 900);
11
12        MinhaThread thread3 = new MinhaThread("Thread #3", 500);
13
14        //thread2.run();
15    }
16 }
```

Exemplo com herança

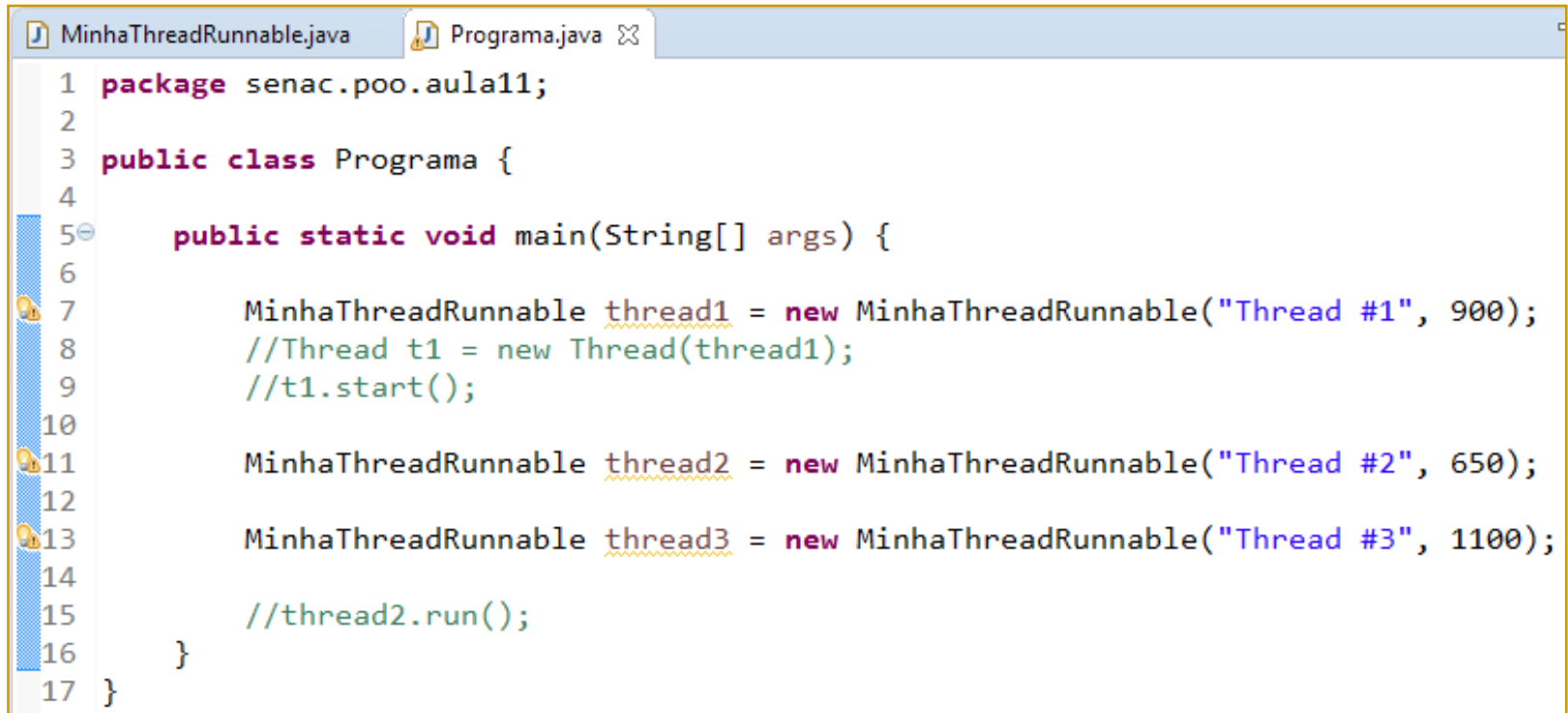
```
Thread #1 contador 0
Thread #3 contador 0
Thread #2 contador 0
Thread #3 contador 1
Thread #1 contador 1
Thread #2 contador 1
Thread #3 contador 2
Thread #1 contador 2
Thread #3 contador 3
Thread #1 contador 3
Thread #2 contador 2
Thread #3 contador 4
Thread #1 contador 4
Thread #3 contador 5
Thread #2 contador 3
Thread #3 terminou a execução
Thread #1 contador 5
Thread #1 terminou a execução
Thread #2 contador 4
Thread #2 contador 5
Thread #2 terminou a execução
```

```
Thread #2 contador 0
Thread #3 contador 0
Thread #1 contador 0
Thread #3 contador 1
Thread #1 contador 1
Thread #2 contador 1
Thread #3 contador 2
Thread #1 contador 2
Thread #3 contador 3
Thread #1 contador 3
Thread #2 contador 2
Thread #3 contador 4
Thread #1 contador 4
Thread #3 contador 5
Thread #2 contador 3
Thread #1 contador 5
Thread #3 terminou a execução
Thread #1 terminou a execução
Thread #2 contador 4
Thread #2 contador 5
Thread #2 terminou a execução
```

Exemplo com interface

```
MinhaThreadRunnable.java Programa.java
1 package senac.poo.aula11;
2
3 public class MinhaThreadRunnable implements Runnable {
4
5     private String nome;
6     private int tempo;
7
8     public MinhaThreadRunnable(String nome, int tempo){
9         this.nome = nome;
10        this.tempo = tempo;
11        Thread t = new Thread(this);
12        t.start();
13    }
14    @Override
15    public void run() {
16        try {
17            for (int i=0; i<6; i++){
18                System.out.println(nome + " contador " + i);
19                Thread.sleep(tempo);
20            }
21        } catch (InterruptedException e) {
22            e.printStackTrace();
23        }
24        System.out.println(nome + " terminou a execução");
25    }
26 }
```

Exemplo com interface



```
1 package senac.poo.aula11;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         MinhaThreadRunnable thread1 = new MinhaThreadRunnable("Thread #1", 900);
8         //Thread t1 = new Thread(thread1);
9         //t1.start();
10
11        MinhaThreadRunnable thread2 = new MinhaThreadRunnable("Thread #2", 650);
12
13        MinhaThreadRunnable thread3 = new MinhaThreadRunnable("Thread #3", 1100);
14
15        //thread2.run();
16    }
17 }
```

Métodos `isAlive()` e `join()`

- `isAlive`


- Retorna `true` se a thread em que foi chamada ainda estiver em execução, caso contrário retorna `false`

- `join`

- Método que espera o término da thread para qual foi enviada a mensagem para ser liberada

Métodos isAlive() e join()

```
MinhaThreadRunnable.java x Programa.java
1 package senac.poo.aula11;
2
3 public class MinhaThreadRunnable implements Runnable {
4
5     private String nome;
6     private int tempo;
7
8     public MinhaThreadRunnable(String nome, int tempo){
9         this.nome = nome;
10        this.tempo = tempo;
11        //Thread t = new Thread(this);
12        //t.start();
13    }
14    @Override
15    public void run() {
16        try {
17            for (int i=0; i<6; i++){
18                System.out.println(nome + " contador " + i);
19                Thread.sleep(tempo);
20            }
21        } catch (InterruptedException e) {
22            e.printStackTrace();
23        }
24        System.out.println(nome + " terminou a execução");
25    }
26 }
```



Métodos isAlive() e join()

```
MinhaThreadRunnable.java  Programa.java ✕
1 package senac.poo.aula11;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         MinhaThreadRunnable thread1 = new MinhaThreadRunnable("Thread #1", 500);
8         MinhaThreadRunnable thread2 = new MinhaThreadRunnable("Thread #2", 500);
9         MinhaThreadRunnable thread3 = new MinhaThreadRunnable("Thread #3", 500);
10
11         Thread t1 = new Thread(thread1);
12         Thread t2 = new Thread(thread2);
13         Thread t3 = new Thread(thread3);
14
15         t1.start();
16         t2.start();
17         t3.start();
18
19         System.out.println("Programa finalizado");
20     }
21 }
```

Métodos isAlive() e join()

```
#1 contador 0
#3 contador 0
#2 contador 0
Programa finalizado
#3 contador 1
#1 contador 1
#2 contador 1
#1 contador 2
#2 contador 2
#3 contador 2
#1 contador 3
#2 contador 3
#3 contador 3
#2 contador 4
#1 contador 4
#3 contador 4
#1 contador 5
#2 contador 5
#3 contador 5
#1 terminou a execução
#3 terminou a execução
#2 terminou a execução
```




Métodos `isAlive()` e `join()`

```
MinhaThreadRunnable.java  Programa.java ✕
1 package senac.poo.aula11;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         MinhaThreadRunnable thread1 = new MinhaThreadRunnable("Thread #1", 500);
8         MinhaThreadRunnable thread2 = new MinhaThreadRunnable("Thread #2", 500);
9         MinhaThreadRunnable thread3 = new MinhaThreadRunnable("Thread #3", 500);
10
11         Thread t1 = new Thread(thread1);
12         Thread t2 = new Thread(thread2);
13         Thread t3 = new Thread(thread3);
14
15         t1.start();
16         t2.start();
17         t3.start();
18
19         while(t1.isAlive() || t2.isAlive() || t3.isAlive()){ }
20
21         System.out.println("Programa finalizado");
22     }
23 }
```

Métodos isAlive() e join()

```
Thread #1 contador 0
Thread #2 contador 0
Thread #3 contador 0
Thread #2 contador 1
Thread #1 contador 1
Thread #3 contador 1
Thread #2 contador 2
Thread #3 contador 2
Thread #1 contador 2
Thread #2 contador 3
Thread #1 contador 3
Thread #3 contador 3
Thread #1 contador 4
Thread #3 contador 4
Thread #2 contador 4
Thread #1 contador 5
Thread #2 contador 5
Thread #3 contador 5
Thread #3 terminou a execução
Thread #1 terminou a execução
Thread #2 terminou a execução
Programa finalizado
```




Métodos isAlive() e join()

```
MinhaThreadRunnable.java  Programa.java ✕
5 public static void main(String[] args) {
6
7     MinhaThreadRunnable thread1 = new MinhaThreadRunnable("Thread #1", 500);
8     MinhaThreadRunnable thread2 = new MinhaThreadRunnable("Thread #2", 700);
9     MinhaThreadRunnable thread3 = new MinhaThreadRunnable("Thread #3", 800);
10
11     Thread t1 = new Thread(thread1);
12     Thread t2 = new Thread(thread2);
13     Thread t3 = new Thread(thread3);
14
15     t1.start();
16     t2.start();
17     t3.start();
18
19     try {
20         t1.join();
21         t2.join();
22         t3.join();
23     } catch (InterruptedException e) {
24         e.printStackTrace();
25     }
26
27     System.out.println("Programa finalizado");
28 }
29 }
```

Métodos isAlive() e join()

```
Thread #2 contador 0
Thread #1 contador 0
Thread #3 contador 0
Thread #1 contador 1
Thread #2 contador 1
Thread #3 contador 1
Thread #1 contador 2
Thread #2 contador 2
Thread #1 contador 3
Thread #3 contador 2
Thread #1 contador 4
Thread #2 contador 3
Thread #3 contador 3
Thread #1 contador 5
Thread #2 contador 4
Thread #1 terminou a execução
Thread #3 contador 4
Thread #2 contador 5
Thread #3 contador 5
Thread #2 terminou a execução
Thread #3 terminou a execução
Programa finalizado
```



Método `setPriority()`

- **`setPriority`**

- É utilizado para indicar qual é a prioridade da thread

- Existem 3 prioridades que podem ser utilizadas

- `Thread.MIN_PRIORITY` (prioridade baixa)
- `Thread.MAX_PRIORITY` (prioridade alta)
- `Thread.NORM_PRIORITY` (prioridade normal)

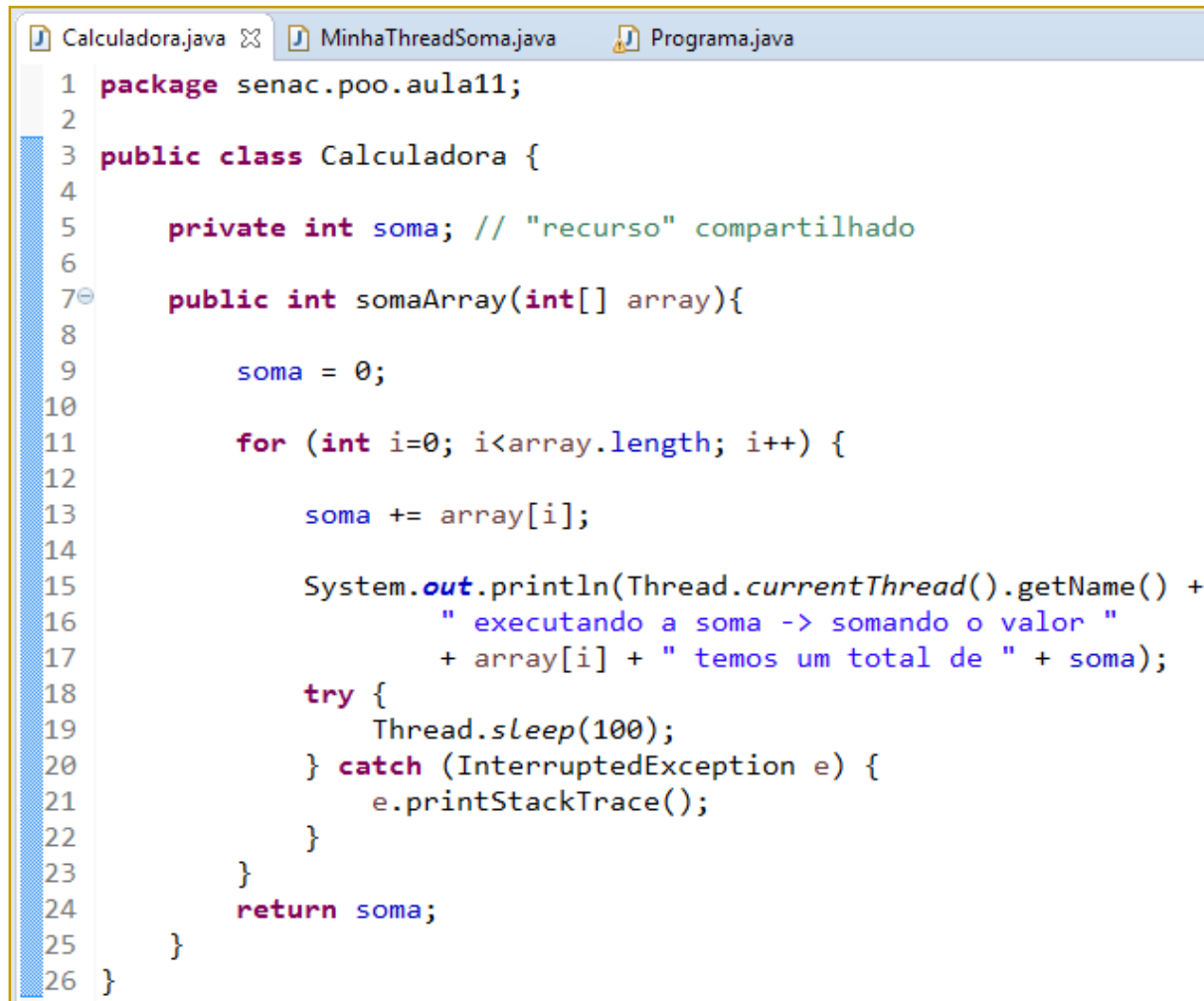
Método setPriority()

```
MinhaThreadRunnable.java Programa.java ✖
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         MinhaThreadRunnable thred1 = new MinhaThreadRunnable("#1", 500);
8         MinhaThreadRunnable thred2 = new MinhaThreadRunnable("#2", 500);
9         MinhaThreadRunnable thred3 = new MinhaThreadRunnable("#3", 500);
10
11         Thread t1 = new Thread(thred1);
12         Thread t2 = new Thread(thred2);
13         Thread t3 = new Thread(thred3);
14
15         t1.setPriority(Thread.MAX_PRIORITY);
16         t2.setPriority(Thread.NORM_PRIORITY);
17         t3.setPriority(Thread.MIN_PRIORITY);
18
19         //t1.setPriority(10);
20         //t2.setPriority(6);
21         //t3.setPriority(1);
22
23         t1.start();
24         t2.start();
25         t3.start();
26     }
27 }
```

Métodos e blocos sincronizados

- **Sincronização** é o ato de coordenar as tarefas de 2 ou mais threads
- Motivo: muitas vezes 2 ou mais threads precisam acessar um recurso compartilhado, ou apenas 1 thread possa acessar o recurso por vez
- No Java usamos a palavra chave **synchronized** em métodos (assinatura) ou em um bloco de código

Métodos e blocos sincronizados

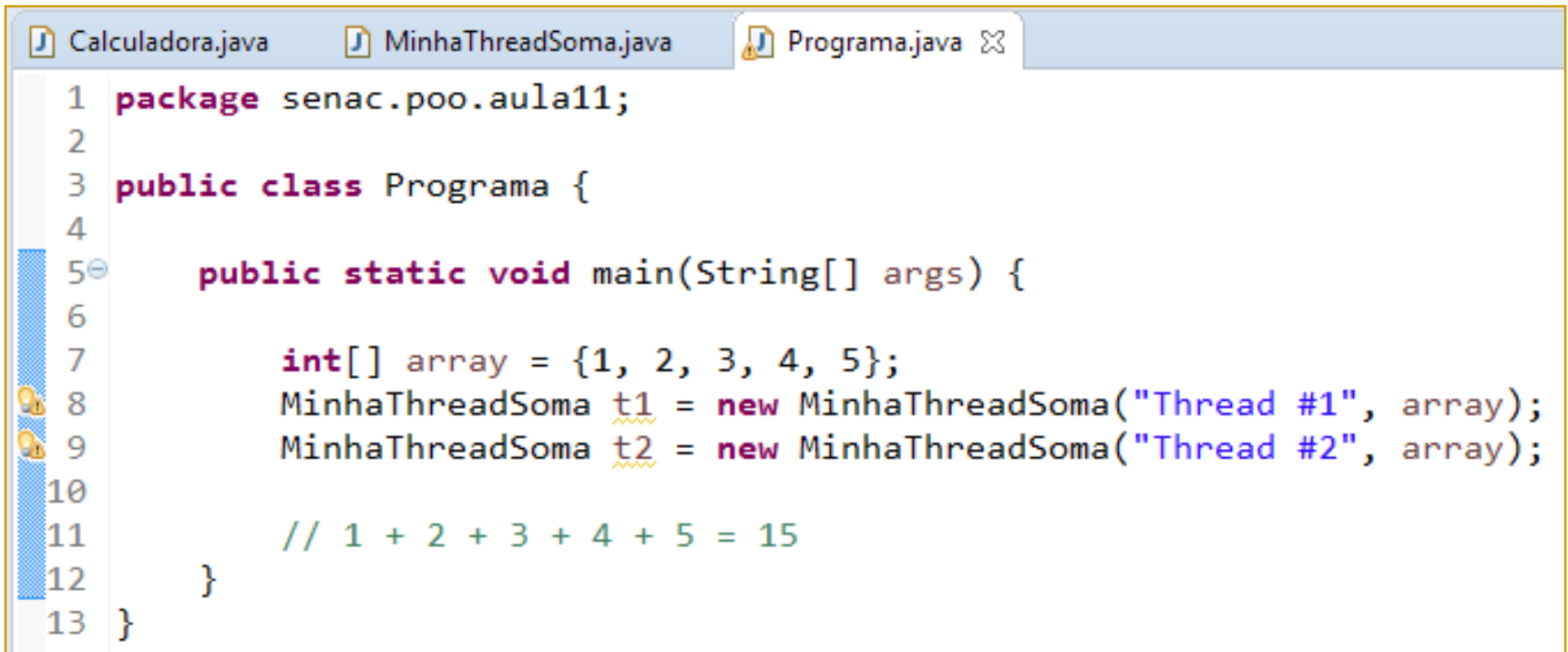


```
1 package senac.poo.aula11;
2
3 public class Calculadora {
4
5     private int soma; // "recurso" compartilhado
6
7     public int somaArray(int[] array){
8
9         soma = 0;
10
11         for (int i=0; i<array.length; i++) {
12
13             soma += array[i];
14
15             System.out.println(Thread.currentThread().getName() +
16                 " executando a soma -> somando o valor "
17                 + array[i] + " temos um total de " + soma);
18             try {
19                 Thread.sleep(100);
20             } catch (InterruptedException e) {
21                 e.printStackTrace();
22             }
23         }
24         return soma;
25     }
26 }
```


Métodos e blocos sincronizados

```
Calculadora.java  MinhaThreadSoma.java  Programa.java
1 package senac.poo.aula11;
2
3 public class MinhaThreadSoma implements Runnable {
4
5     private String nome;
6     private int[] numeros;
7     private static Calculadora calc = new Calculadora();
8
9     public MinhaThreadSoma(String nome, int[] numeros){
10         this.nome = nome;
11         this.numeros = numeros;
12         new Thread(this, nome).start();
13         //Thread t = new Thread(this, nome);
14         //t.start();
15     }
16
17     @Override
18     public void run() {
19
20         System.out.println(this.nome + " iniciou");
21         int soma = calc.somaArray(this.numeros);
22         System.out.println("Resultado da soma para a " + this.nome + " é: " + soma);
23         System.out.println(this.nome + " terminou!");
24     }
25 }
26 }
```

Métodos e blocos sincronizados



```
1 package senac.poo.aula11;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         int[] array = {1, 2, 3, 4, 5};
8         MinhaThreadSoma t1 = new MinhaThreadSoma("Thread #1", array);
9         MinhaThreadSoma t2 = new MinhaThreadSoma("Thread #2", array);
10
11         // 1 + 2 + 3 + 4 + 5 = 15
12     }
13 }
```

Métodos e blocos sincronizados

```
Thread #2 iniciou
Thread #1 iniciou
Thread #2 executando a soma -> somando o valor 1 temos um total de 1
Thread #1 executando a soma -> somando o valor 1 temos um total de 1
Thread #1 executando a soma -> somando o valor 2 temos um total de 5
Thread #2 executando a soma -> somando o valor 2 temos um total de 5
Thread #1 executando a soma -> somando o valor 3 temos um total de 11
Thread #2 executando a soma -> somando o valor 3 temos um total de 11
Thread #2 executando a soma -> somando o valor 4 temos um total de 19
Thread #1 executando a soma -> somando o valor 4 temos um total de 19
Thread #2 executando a soma -> somando o valor 5 temos um total de 29
Thread #1 executando a soma -> somando o valor 5 temos um total de 29
Resultado da soma para a Thread #1 é: 29
Resultado da soma para a Thread #2 é: 29
Thread #2 terminou!
Thread #1 terminou!
```

Métodos e blocos sincronizados

```
Calculadora.java  MinhaThreadSoma.java  Programa.java
1 package senac.poo.aula11;
2
3 public class Calculadora {
4
5     private int soma; // "recurso" compartilhado
6
7     public synchronized int somaArray(int[] array){
8
9         soma = 0;
10
11         for (int i=0; i<array.length; i++) {
12
13             soma += array[i];
14
15             System.out.println(Thread.currentThread().getName() +
16                 " executando a soma -> somando o valor "
17                 + array[i] + " temos um total de " + soma);
18             try {
19                 Thread.sleep(100);
20             } catch (InterruptedException e) {
21                 e.printStackTrace();
22             }
23         }
24         return soma;
25     }
26 }
```

Métodos e blocos sincronizados

```
Thread #2 iniciou
Thread #1 iniciou
Thread #2 executando a soma -> somando o valor 1 temos um total de 1
Thread #2 executando a soma -> somando o valor 2 temos um total de 3
Thread #2 executando a soma -> somando o valor 3 temos um total de 6
Thread #2 executando a soma -> somando o valor 4 temos um total de 10
Thread #2 executando a soma -> somando o valor 5 temos um total de 15
Thread #1 executando a soma -> somando o valor 1 temos um total de 1
Resultado da soma para a Thread #2 é: 15
Thread #2 terminou!
Thread #1 executando a soma -> somando o valor 2 temos um total de 3
Thread #1 executando a soma -> somando o valor 3 temos um total de 6
Thread #1 executando a soma -> somando o valor 4 temos um total de 10
Thread #1 executando a soma -> somando o valor 5 temos um total de 15
Resultado da soma para a Thread #1 é: 15
Thread #1 terminou!
```

Métodos wait, notify e notifyAll

- Uma thread A está sendo executada dentro de um método sincronizado e precisa de acesso a um recurso R que no momento está indisponível
- Se a thread A ficar esperando o recurso R irá bloquear o objeto, impedindo que outras threads acessem o mesmo
- Nesse caso a melhor solução para não causar problemas é liberar temporariamente o controle do objeto, permitindo que outra thread seja executada

Métodos wait, notify e notifyAll

■ wait

- Bloqueia a execução da thread temporariamente, ou seja, coloca a thread em modo de espera até que ela seja notificada

■ notify

- Notifica uma thread que estava esperando, ou seja, retorna a execução da thread

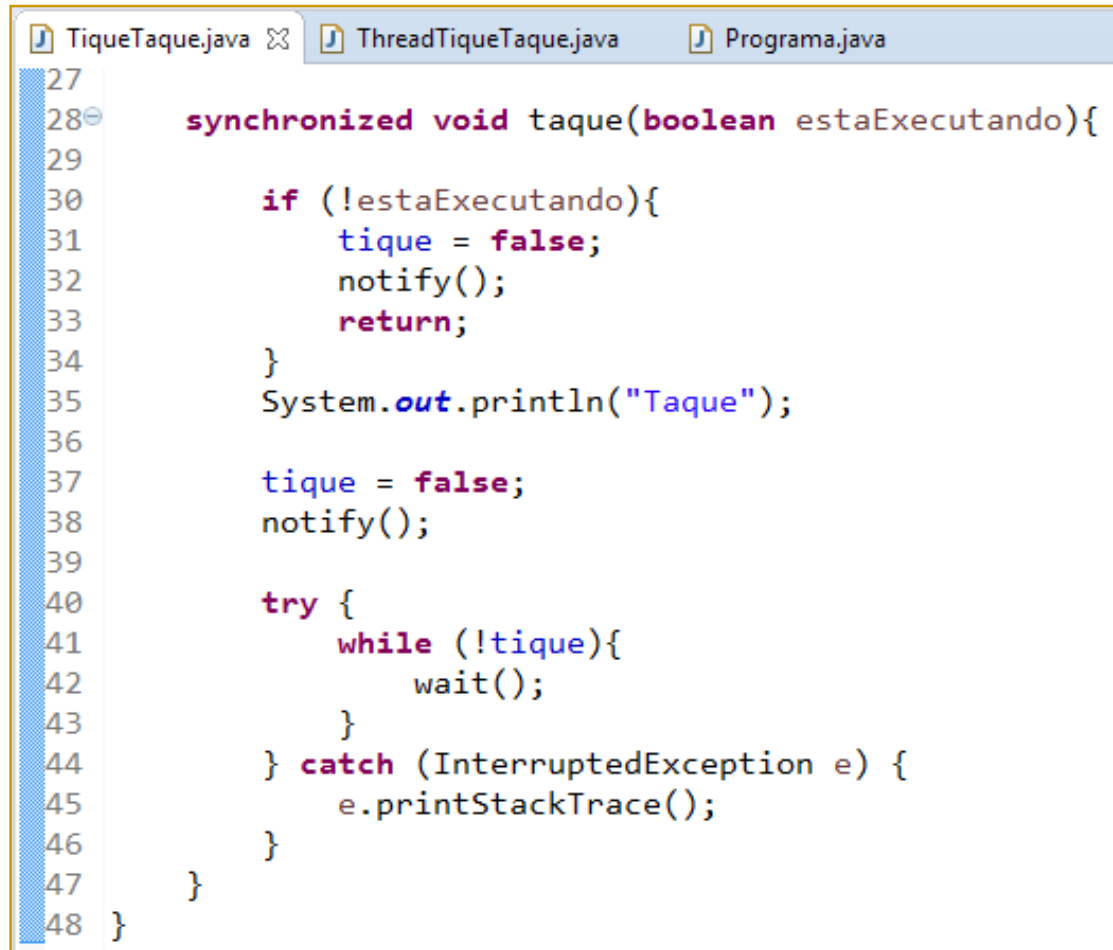
■ notifyAll

- Notifica todas as thread, e a que tem prioridade mais alta ganha o acesso ao objeto

Métodos wait, notify e notifyAll

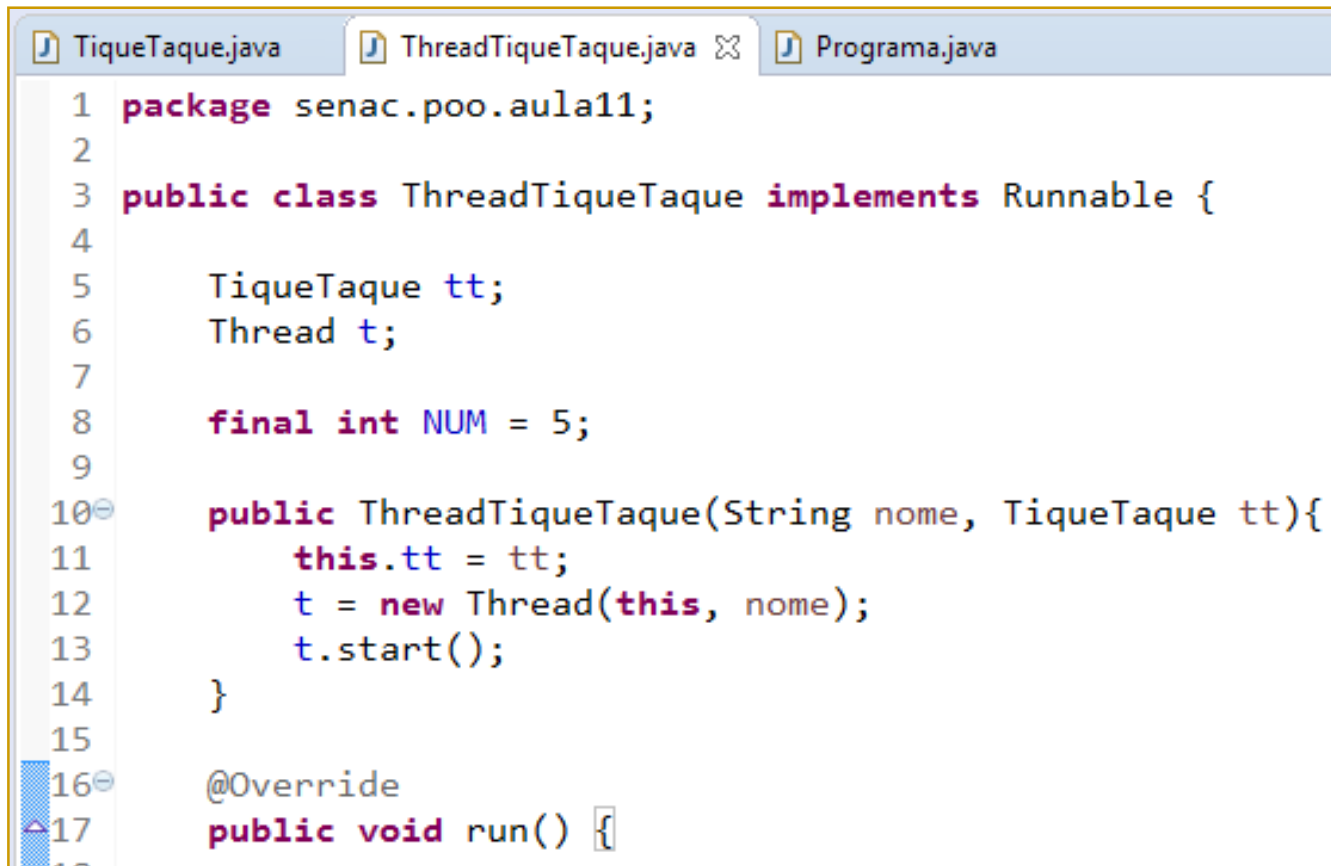
```
TiqueTaque.java ThreadTiqueTaque.java Programa.java
1 package senac.poo.aula11;
2
3 public class TiqueTaque {
4
5     boolean tique;
6
7     synchronized void tique(boolean estaExecutando){
8
9         if (!estaExecutando){
10             tique = true;
11             notify();
12             return;
13         }
14         System.out.print("Tique ");
15
16         tique = true;
17         notify();
18
19         try {
20             while (tique){
21                 wait();
22             }
23         } catch (InterruptedException e) {
24             e.printStackTrace();
25         }
26     }
```


Métodos wait, notify e notifyAll



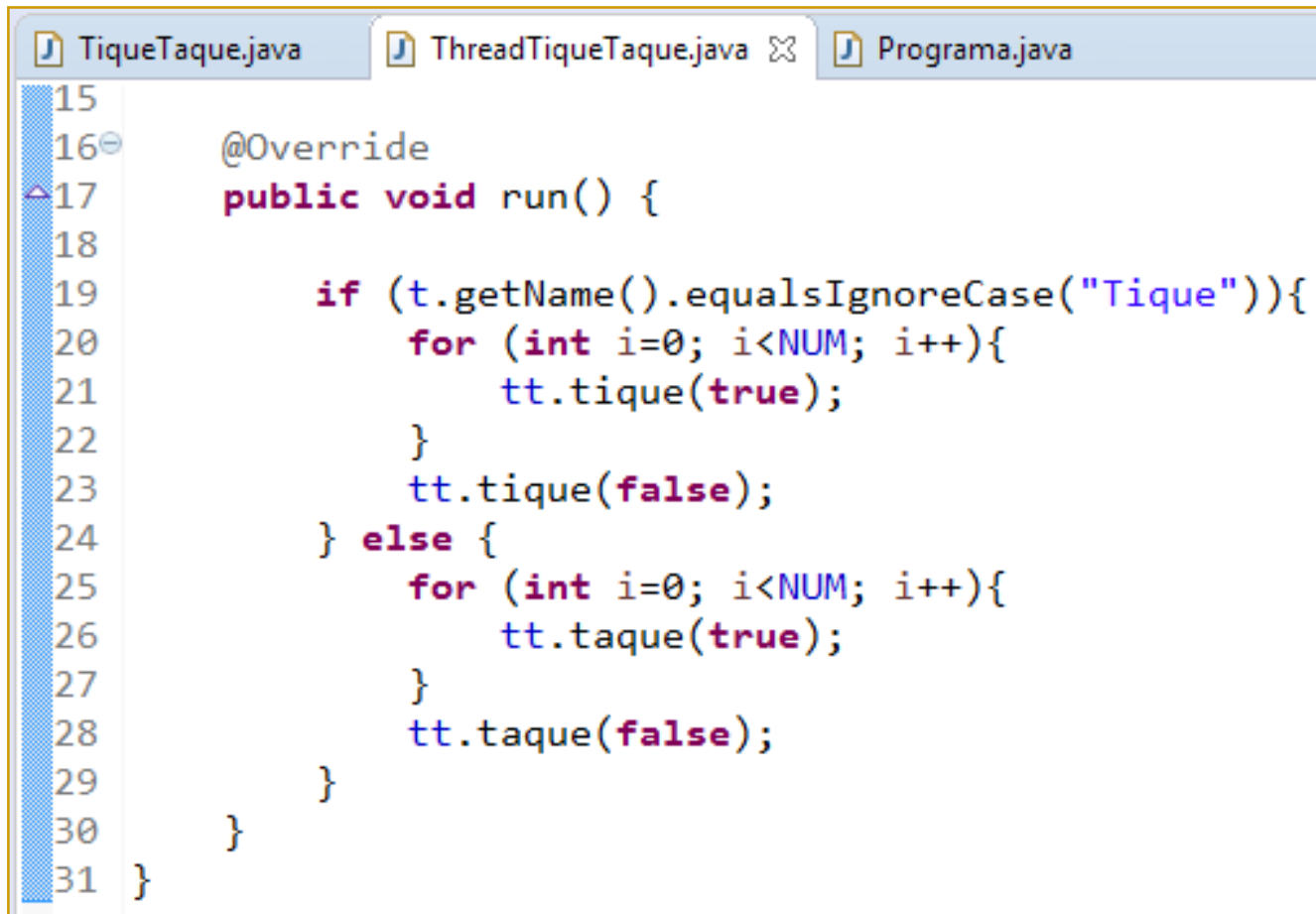
```
TiqueTaque.java  ThreadTiqueTaque.java  Programa.java
27
28 synchronized void taque(boolean estaExecutando){
29
30     if (!estaExecutando){
31         tique = false;
32         notify();
33         return;
34     }
35     System.out.println("Taque");
36
37     tique = false;
38     notify();
39
40     try {
41         while (!tique){
42             wait();
43         }
44     } catch (InterruptedException e) {
45         e.printStackTrace();
46     }
47 }
48 }
```

Métodos wait, notify e notifyAll



```
1 package senac.poo.aula11;
2
3 public class ThreadTiqueTaque implements Runnable {
4
5     TiqueTaque tt;
6     Thread t;
7
8     final int NUM = 5;
9
10    public ThreadTiqueTaque(String nome, TiqueTaque tt){
11        this.tt = tt;
12        t = new Thread(this, nome);
13        t.start();
14    }
15
16    @Override
17    public void run() {
```

Métodos wait, notify e notifyAll



The screenshot shows a Java IDE with three tabs: TiqueTaque.java, ThreadTiqueTaque.java, and Programa.java. The TiqueTaque.java tab is active, displaying the following code:

```
15
16 @Override
17 public void run() {
18
19     if (t.getName().equalsIgnoreCase("Tique")){
20         for (int i=0; i<NUM; i++){
21             tt.tique(true);
22         }
23         tt.tique(false);
24     } else {
25         for (int i=0; i<NUM; i++){
26             tt.taque(true);
27         }
28         tt.taque(false);
29     }
30 }
31 }
```

Métodos wait, notify e notifyAll

```
TiqueTaque.java  ThreadTiqueTaque.java  Programa.java ✕
1  package senac.poo.aula11;
2
3  public class Programa {
4
5      public static void main(String[] args) {
6
7          TiqueTaque tt = new TiqueTaque();
8          ThreadTiqueTaque tique = new ThreadTiqueTaque("Tique", tt);
9          ThreadTiqueTaque taque = new ThreadTiqueTaque("Taque", tt);
10
11         try {
12             tique.t.join();
13             taque.t.join();
14         } catch (InterruptedException e) {
15             e.printStackTrace();
16         }
17     }
18 }
```

Métodos wait, notify e notifyAll

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

Métodos *suspend*, *resume* e *stop*

- Até o Java 2 existiam os métodos *suspend*, *resume* e *stop*
- O método *suspend* foi substituído no Java 2 pelo fato de poder causar problema de **deadlock**
- Como o *resume* não funciona sem o *suspend*, também foi removido
- O método *stop* também foi substituído no Java 2 e agora devemos usar o método *interrupt* no lugar dele