



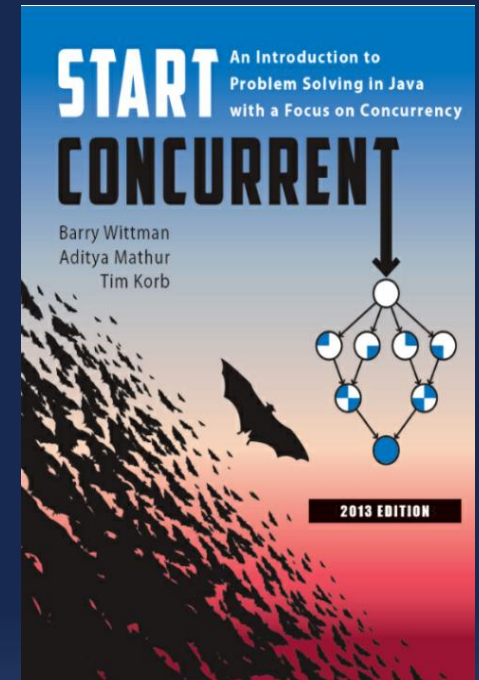
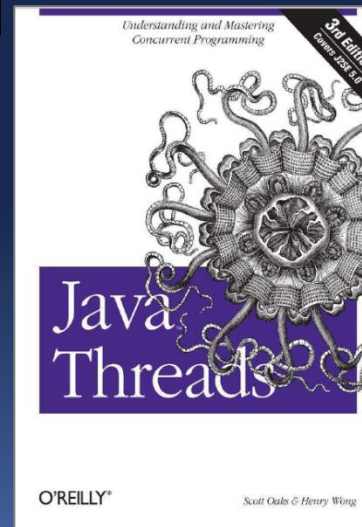
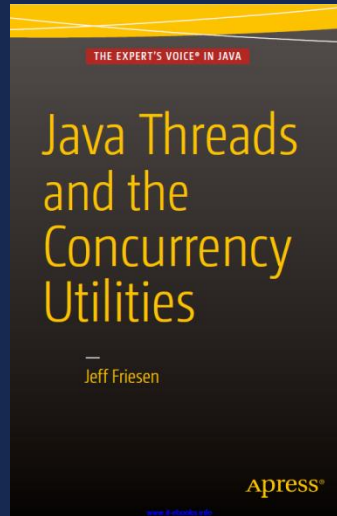
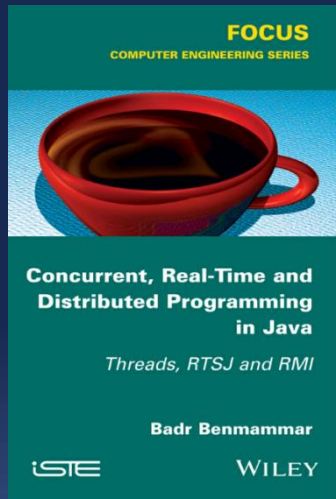
Programação Paralela e Concorrente

Unidade 4 – Aplicação Desktop com Threads



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecidovfreitas@gmail.com

Bibliografia



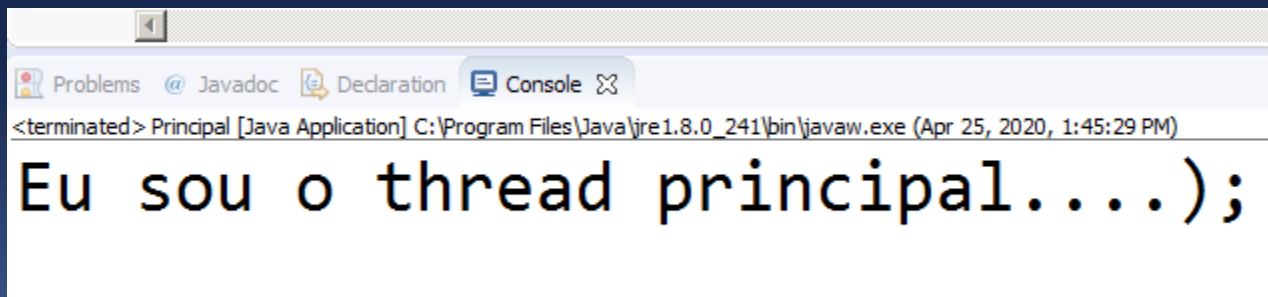
Threads em Java

- Como vimos na Unidade anterior, **threads** são **inerentes** à plataforma **Java**;
- Considere um **projeto Java** com uma classe contendo o método **main()**;
- Ao executar essa classe , a máquina virtual **automaticamente** irá criar um **thread** ou uma **nova linha de execução**;
- Assim, toda aplicação **Java** é composta pelo menos de 1 thread.



Threads em Java

```
Principal.java
1 package br.uscs;
2
3 public class Principal {
4
5     public static void main(String[] args) {
6         System.out.println("Eu sou o thread principal....");
7     }
8
9 }
10
```



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the Java application. The text in the console is "Eu sou o thread principal....);", which is the output of the `System.out.println` statement in the code above. The console also shows the command prompt path and the execution time.

```
<terminated> Principal [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Apr 25, 2020, 1:45:29 PM)
Eu sou o thread principal....);
```

Fazendo o Thread dormir....

- Podemos solicitar à Máquina Virtual que coloque o **thread** para dormir chamando a função estática **Thread.sleep()**;
- Essa função recebe como parâmetro a quantidade de **milissegundos** que o **thread** fica em estado de **"wait"**.



Fazendo o Thread dormir....

```
1 package br.uscs;
2
3 public class Principal {
4
5     public static void main(String[] args) throws InterruptedException {
6         System.out.println("Eu sou o thread principal...");
7         System.out.println("Agora, vou dormir 60 segundos....");
8         Thread.sleep(60000);
9         System.out.println("Fim do programa Principal....");
10    }
11
12 }
13
```

```
Eu sou o thread principal...
Agora, vou dormir 60 segundos....
Fim do programa Principal....
```

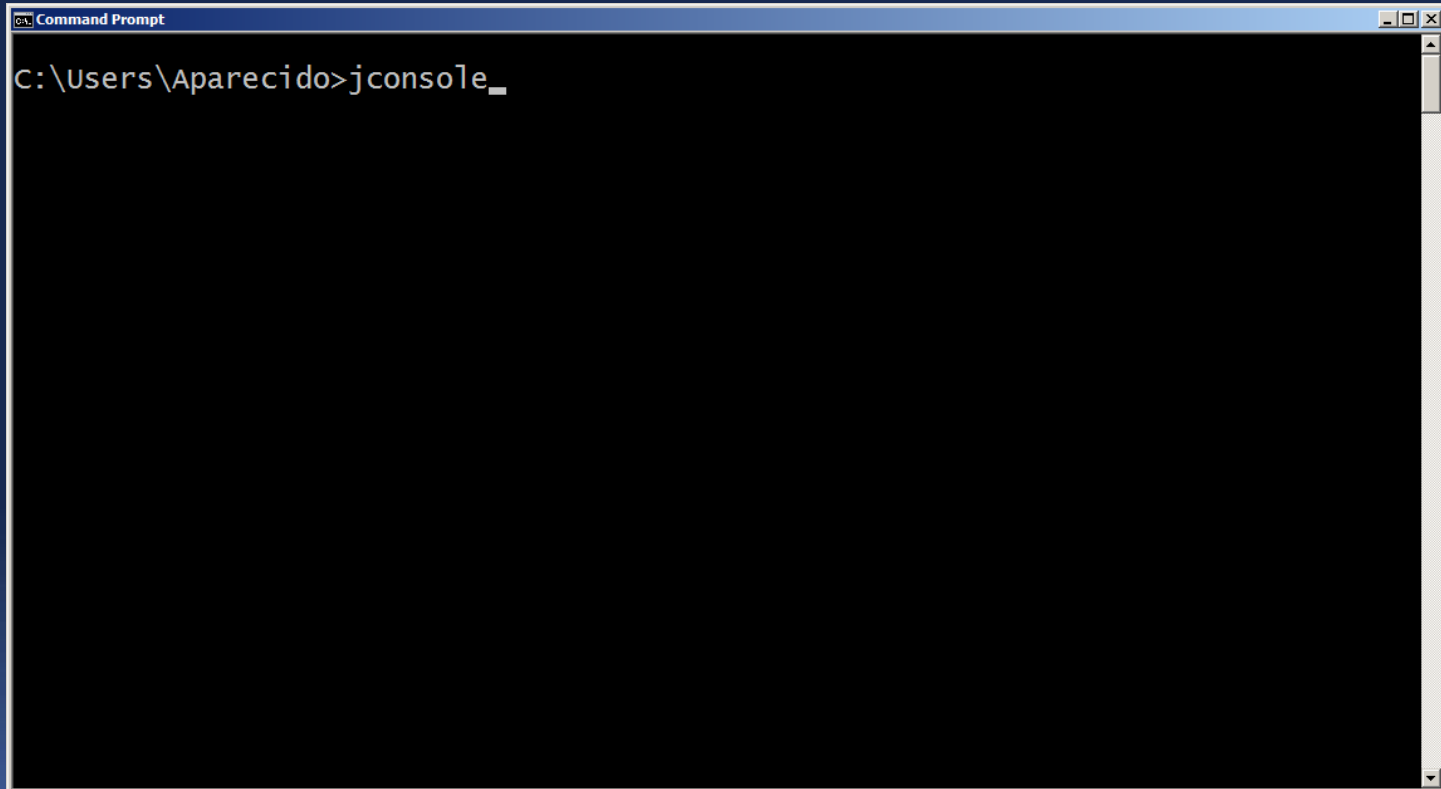
Inspecionando o programa dormir

- Vamos executar **novamente** o programa e durante a sua execução, vamos **ativar** a ferramenta **jconsole** pela linha de comandos;
- Essa ferramenta faz parte do pacote java **jdk**;
- Por meio dessa ferramenta pode-se **conectar** à aplicação e obter-se informações sobre a Máquina Virtual que está processando o nosso código.



Criação de Threads em Java – Método 1

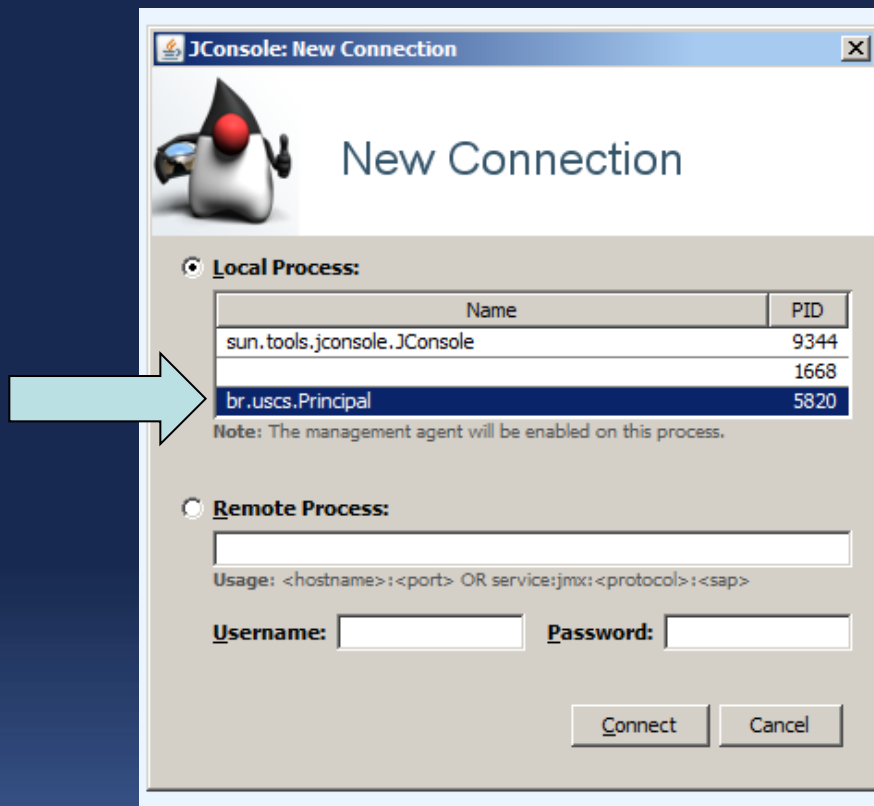
- Para ativar a ferramenta **jconsole**, entre com o comando **jconsole** no prompt de comandos.



```
Command Prompt
C:\Users\Aparecido>jconsole_
```

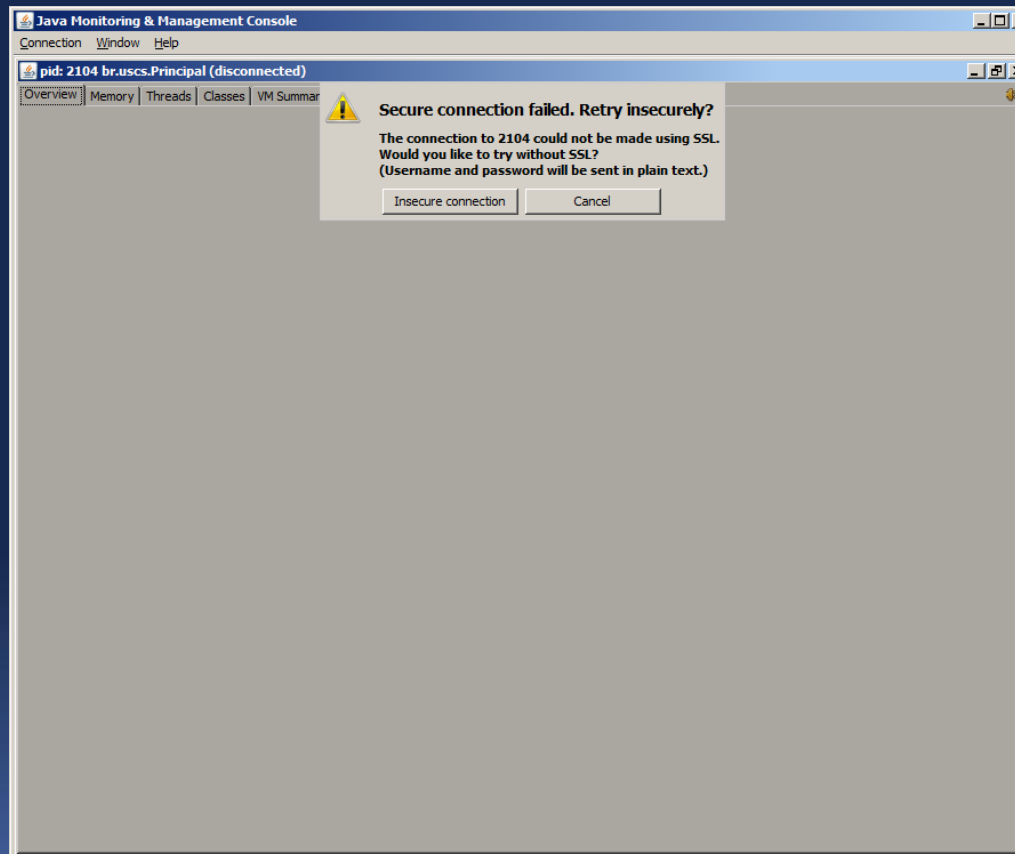

Criação de Threads em Java – Método 1

- Vamos executar o método **main()** da classe **Principal** e logo em seguida vamos nos conectar à aplicação pelo **jconsole**.



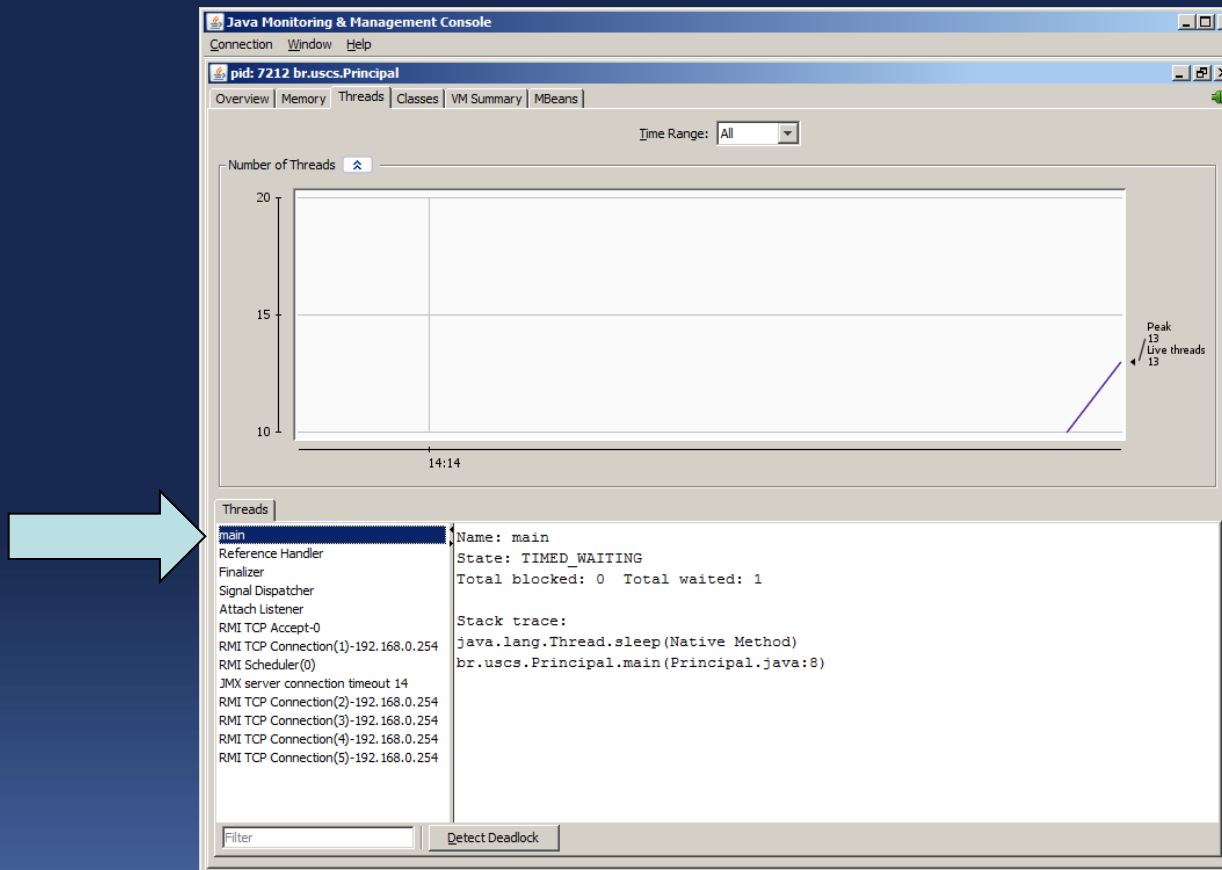
Criação de Threads em Java – Método 1

- Escolha **Insecure connection**, para confirmar a conexão.



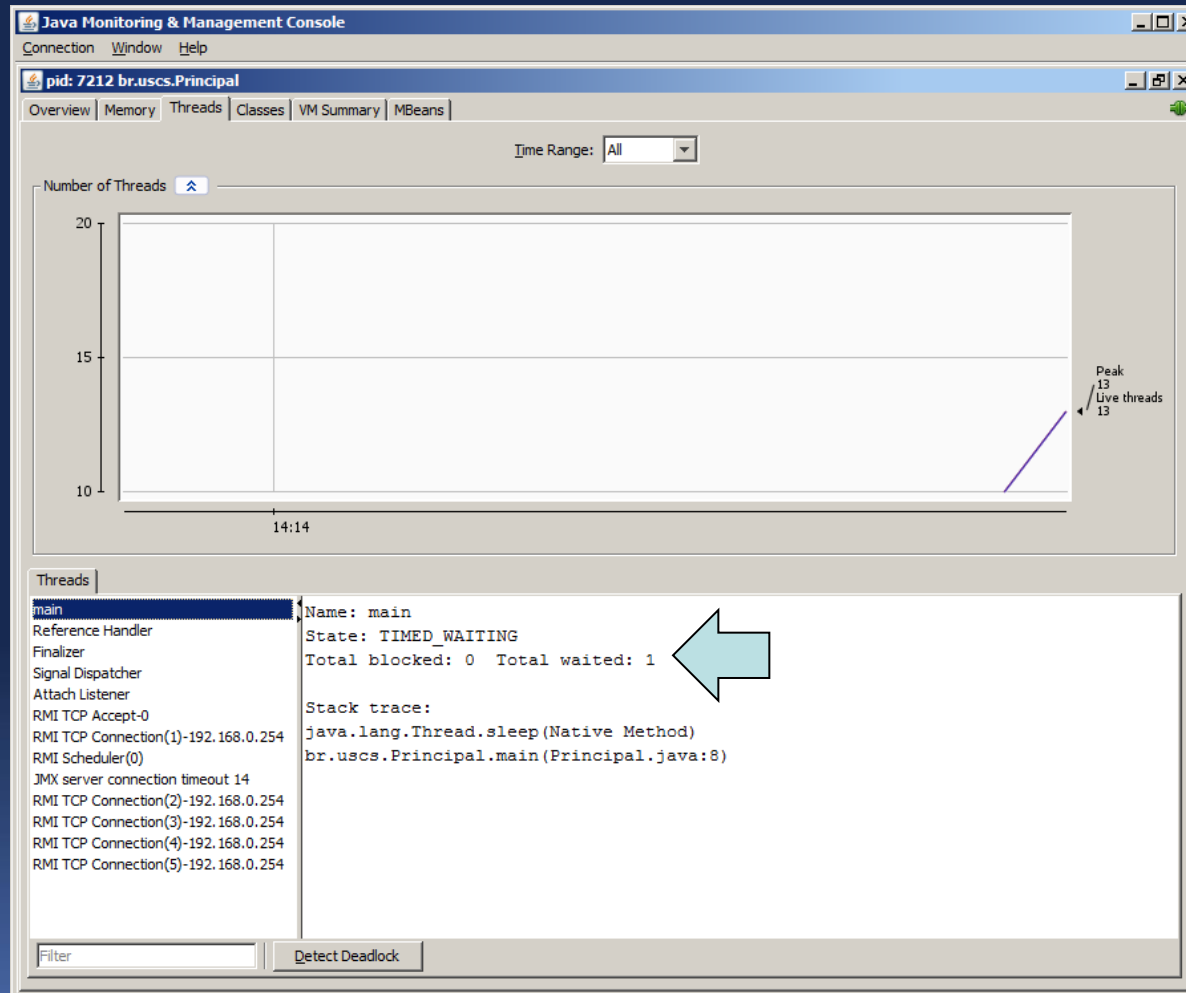
Criação de Threads em Java – Método 1

- Clique na aba **Threads** e localize o thread **main**;
- Observe que **jconsole** exibe também diversos outros **threads** que estão em execução na JVM;



Criação de Threads em Java – Método 1

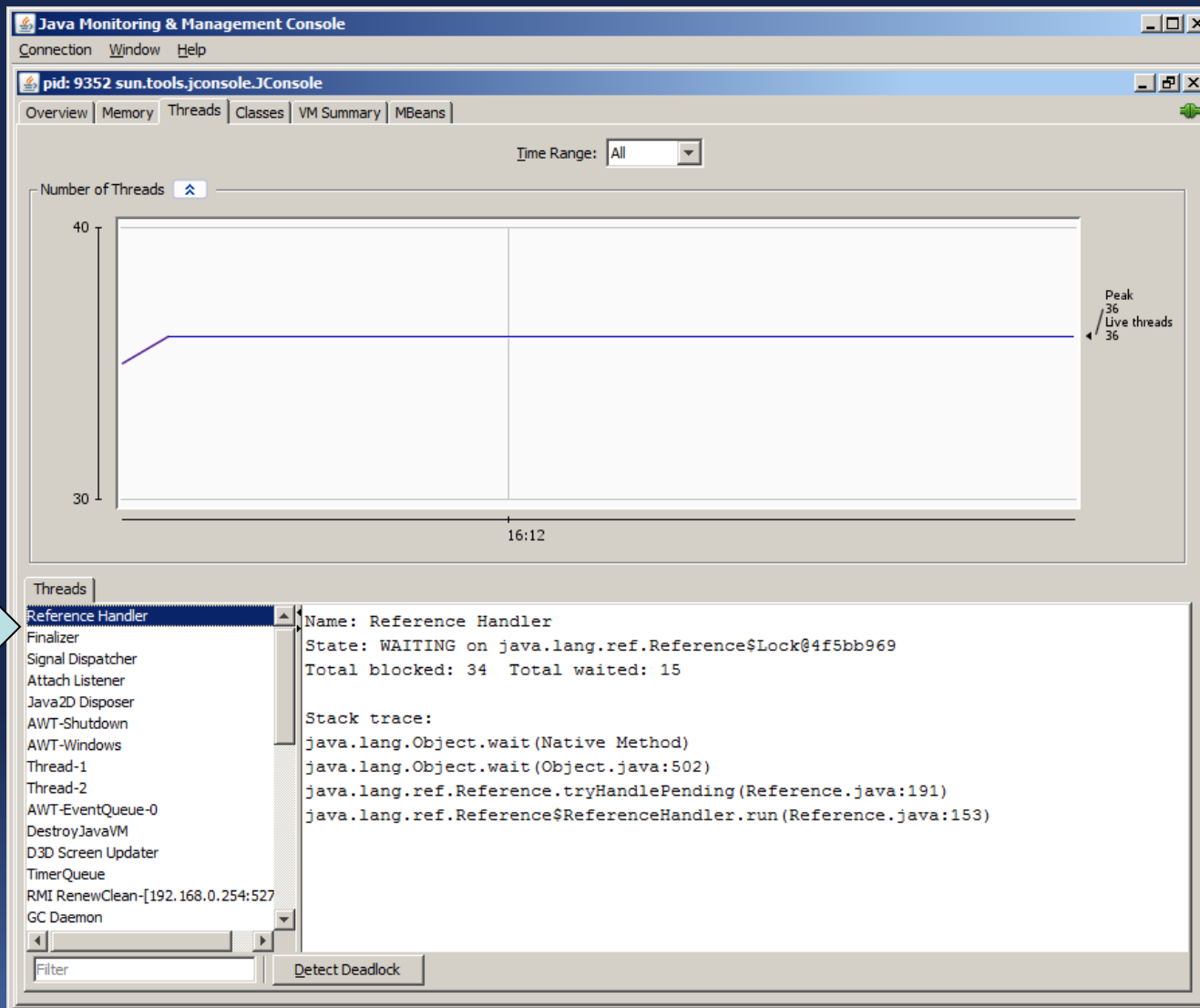
- Veja que o thread **main** está em estado de **"waiting"**;
- Observe também que a **JVM** está processando diversos outros threads.



Threads e a JVM

- A máquina virtual **Java** já **nasceu** com o emprego de Threads;
- Com o uso de Threads a JVM opera com maior desempenho;
- Um exemplo típico do uso de threads pela Máquina Virtual Java são os threads do Coletor de Lixo (**Gargage Collector**)

Threads – Garbage Collector



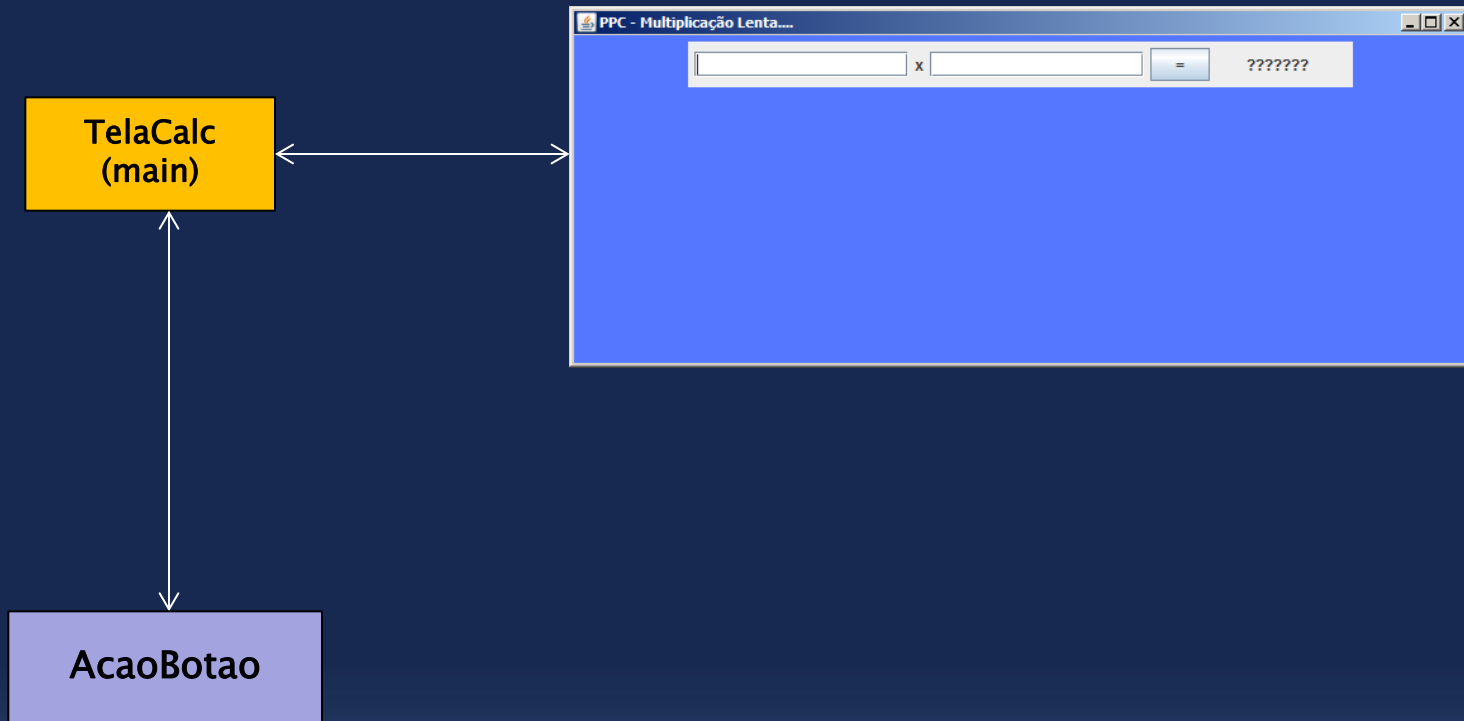
Exemplo de uma aplicação Java com o emprego de Threads



- Neste exemplo, iremos criar uma **interface visual** desktop em Java com a **API swing**;
- Iremos construir gradativamente a interface, inicialmente sem a implementação de **threads adicionais**;
- Ou seja, inicialmente a aplicação terá um **único** thread (**main**).



Estrutura da Aplicação



Código da classe Principal (TelaCalc)

- Nossa classe principal será chamada **TelaCalc** que será um Frame;
- Portanto, deveremos herdá-la da Classe **Jframe**;
- A função **main()** irá instanciar um objeto que representa a execução da classe **TelaCalc**.

```
package br.uscs;  
  
import javax.swing.JFrame;  
  
public class TelaCalc extends JFrame {  
  
    public static void main(String[] args) {  
        TelaCalc app = new TelaCalc();  
    }  
}
```

Código da classe Principal (TelaCalc)

- Vamos criar o construtor que irá criar a tela da aplicação (**JPanel**);
- Aplicaremos na interface, o organizador de layout (**FlowLayout**).

```
package br.uscs;

import java.awt.FlowLayout;
import javax.swing.JFrame;

public class TelaCalc extends JFrame {

    public TelaCalc() {
        super("PPC - Multiplicação Lenta....");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
    }

    public static void main(String[] args) {
        TelaCalc app = new TelaCalc();
    }
}
```



Código da classe Principal (TelaCalc)



- Agora, vamos definir o tamanho da Janela: **setSize(800x500);**

```
package br.uscs;

import java.awt.FlowLayout;
import javax.swing.JFrame;

public class TelaCalc extends JFrame {

    public TelaCalc() {
        super("PPC - Multiplicação Lenta....");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);
    }

    public static void main(String[] args) {
        TelaCalc app = new TelaCalc();
    }
}
```



Código da classe Principal (TelaCalc)



- Vamos agora exibir a janela para o usuário - **setVisible(true)** ;

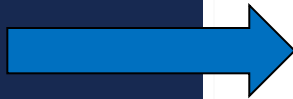
```
package br.uscs;

import java.awt.FlowLayout;
import javax.swing.JFrame;

public class TelaCalc extends JFrame {

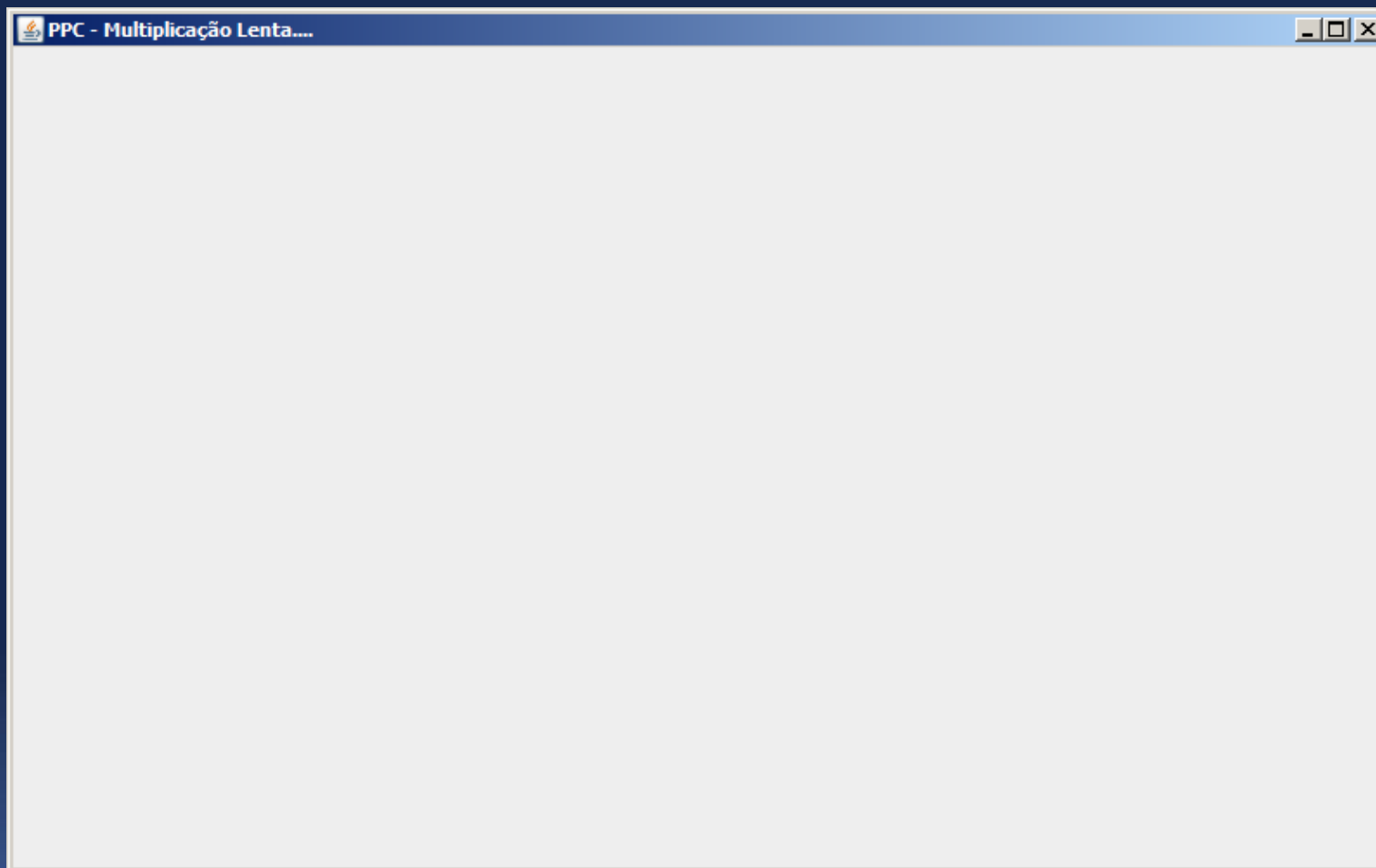
    public TelaCalc() {
        super("PPC - Multiplicação Lenta....");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        TelaCalc app = new TelaCalc();
    }
}
```



Código da classe Principal (Telacalc)

- Ao executar a aplicação, a **janela** será **exibida** ao usuário.



Código da classe Principal (TelaCalc)



- Vamos centralizar a tela - **setLocationRelative(true)**.

```
package br.uscs;

import java.awt.FlowLayout;
import javax.swing.JFrame;

public class TelaCalc extends JFrame {

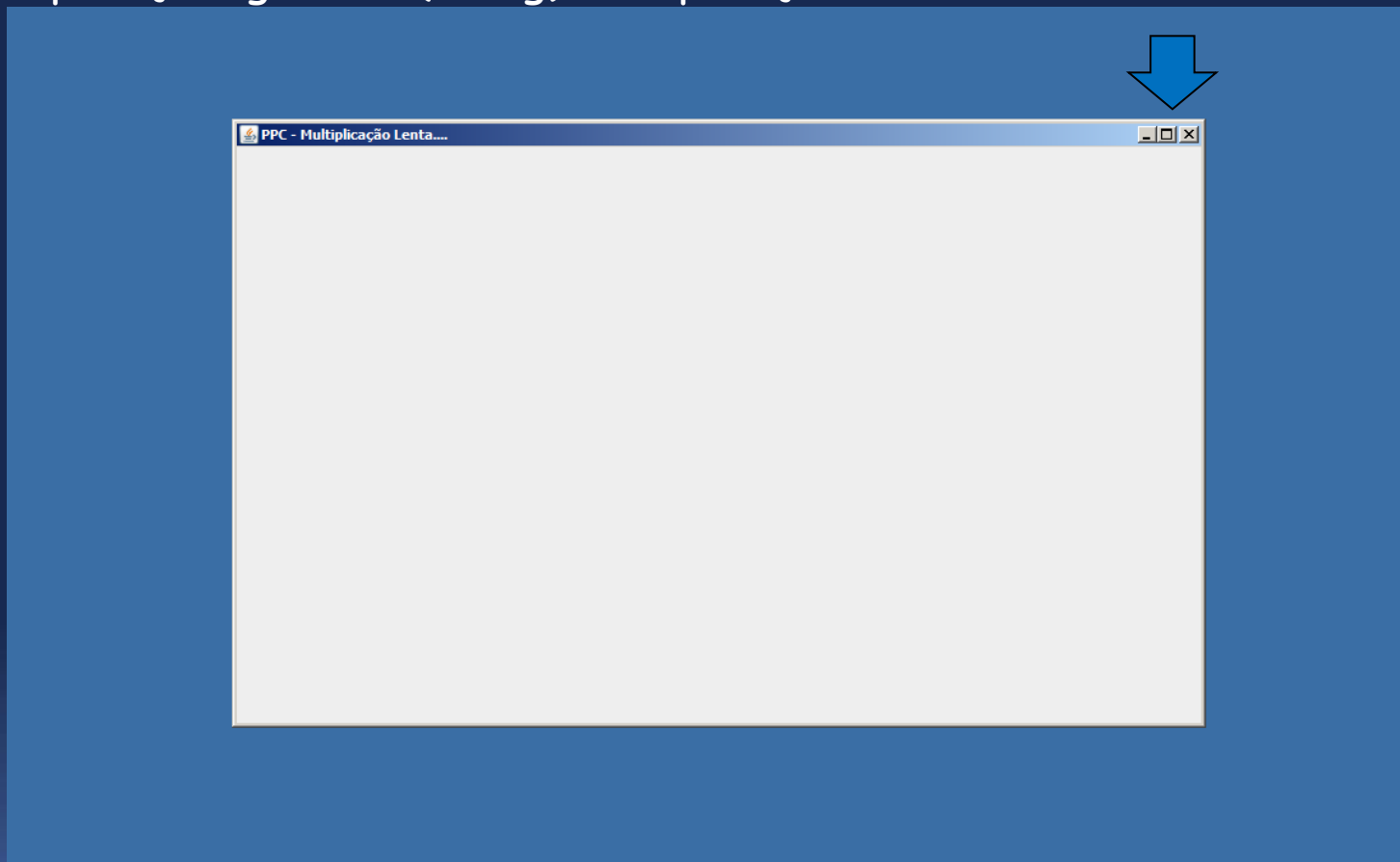
    public TelaCalc() {
        super("PPC - Multiplicação Lenta....");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);
        this.setVisible(true);
        this.setLocationRelativeTo(null);
    }

    public static void main(String[] args) {
        TelaCalc app = new TelaCalc();
    }
}
```



Código da classe Principal (Telacalc)

- Vamos centralizar a tela - **setLocationRelative(true)**.
- Observe que a interface de controle está funcionando e já temos uma aplicação gráfica (swing) em operação.



○ Vamos acrescentar uma caixa de texto à janela - JTextField



```
package br.uscs;

import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class TelaCalc extends JFrame {

    public TelaCalc() {
        super("PPC - Multiplicação Lenta...");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);

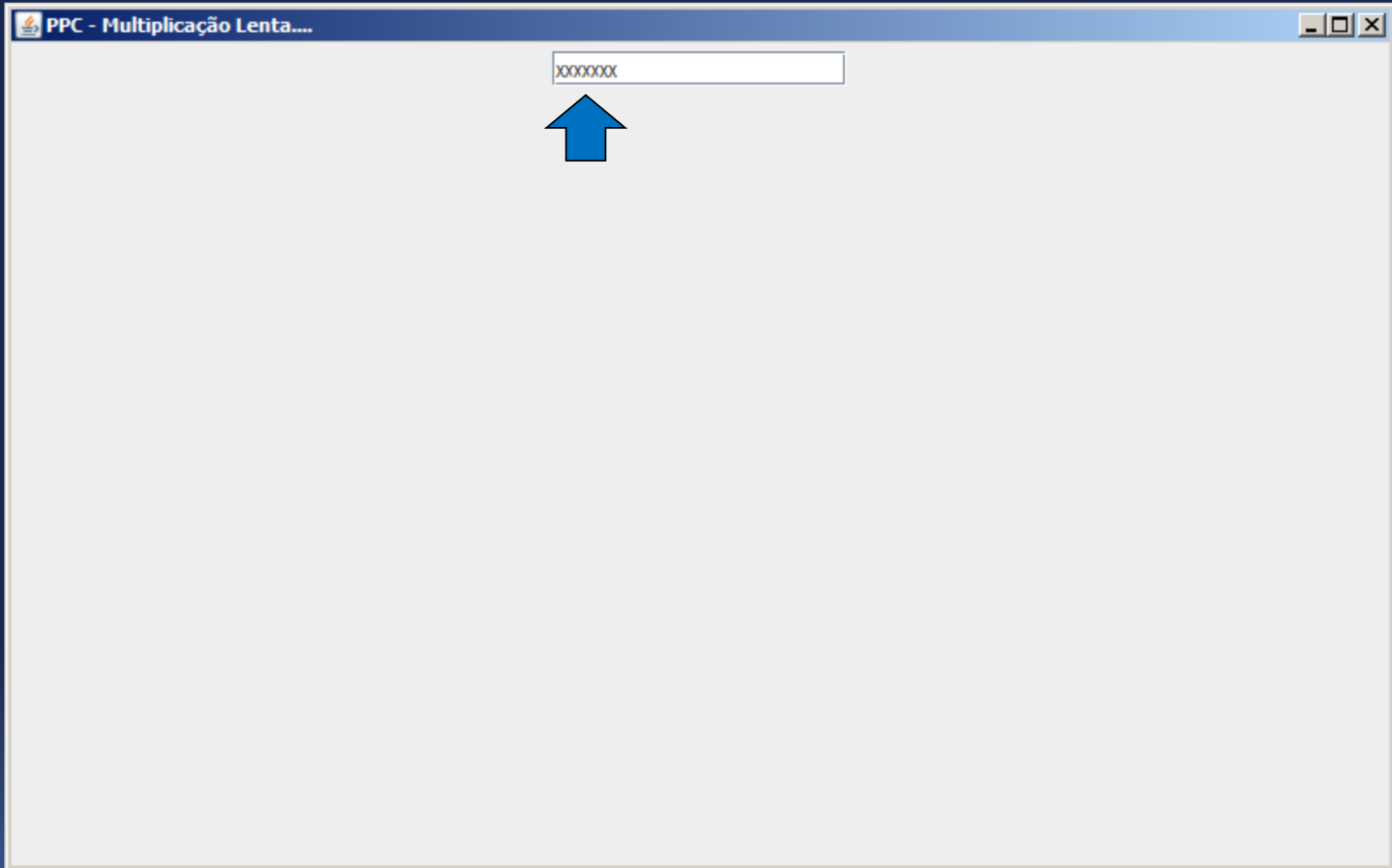
        JTextField texto1 = new JTextField(15);
        this.add(texto1);

        this.setVisible(true);
        this.setLocationRelativeTo(null);
    }

    public static void main(String[] args) {
        TelaCalc app = new TelaCalc();
    }
}
```



- Ao executar a aplicação, você poderá entrar com alguma informação na caixa de texto criada.



● Vamos acrescentar outra caixa de texto à janela - JTextField

```
public class TelaCalc extends JFrame {

    public TelaCalc() {
        super("PPC - Multiplicação Lenta....");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);

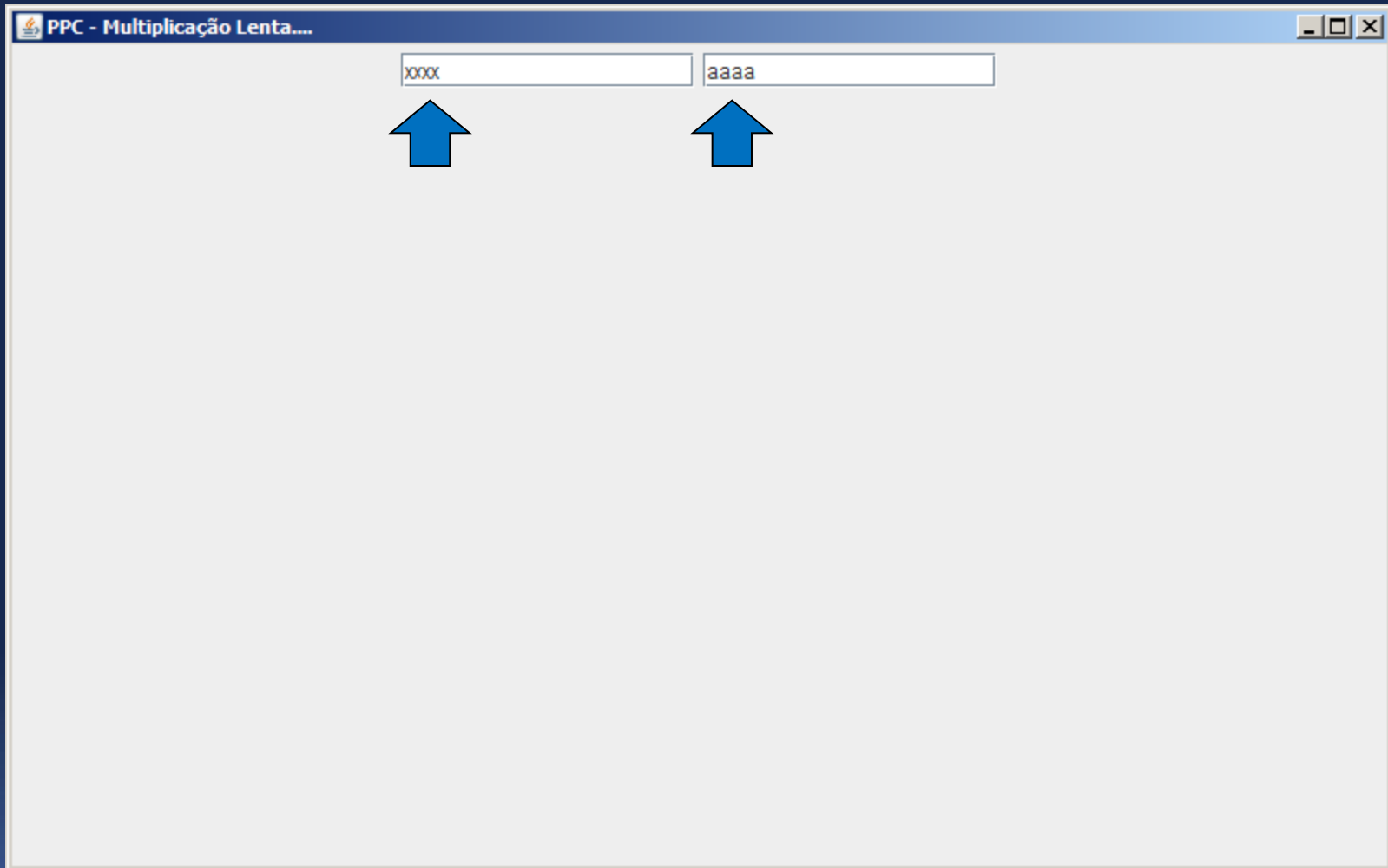
        JTextField texto1 = new JTextField(15);
        this.add(texto1);

        JTextField texto2 = new JTextField(15);
        this.add(texto2);

        this.setVisible(true);
        this.setLocationRelativeTo(null);
    }
}
```



- Ao executar a aplicação, você poderá entrar com alguma informação nas duas caixas de texto criadas.

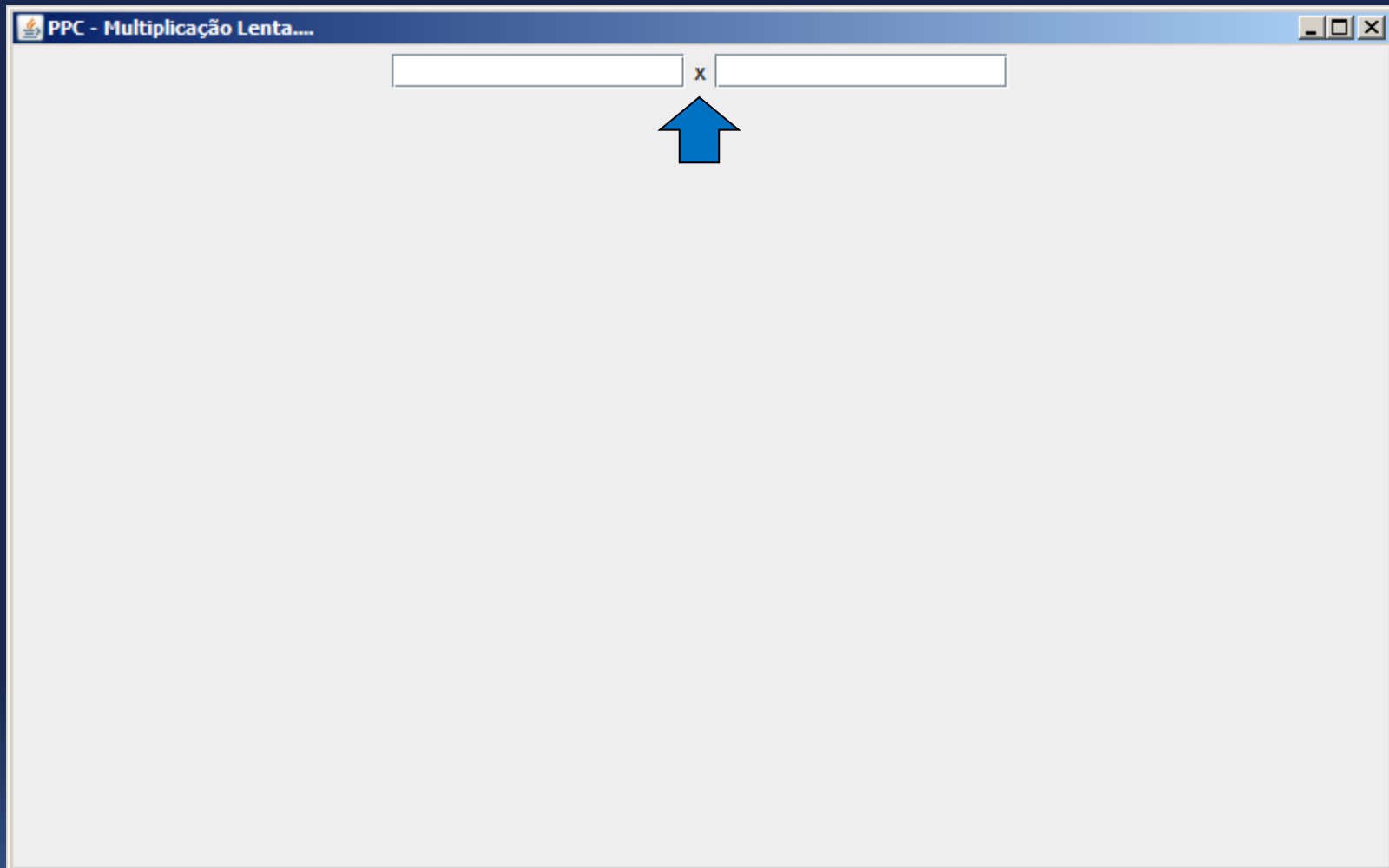


- A nossa aplicação implementará uma operação de **multiplicação** dos valores entrados em cada uma das caixas de texto criadas;
- Para tornar a operação mais legível, incluiremos **entre** as duas caixas de texto, um **Label** que indica multiplicação (*).



```
public TelaCalc() {  
    super("PPC - Multiplicação Lenta...");  
    FlowLayout layout = new FlowLayout();  
    this.setLayout(layout);  
    this.setSize(800, 500);  
  
    JTextField texto1 = new JTextField(15);  
    this.add(texto1);  
  
    JLabel simboloMultiplica = new JLabel("x");  
    this.add(simboloMultiplica);  
  
    JTextField texto2 = new JTextField(15);  
    this.add(texto2);  
  
    this.setVisible(true);  
    this.setLocationRelativeTo(null);  
}
```

- Executando a interface, para visualizar o label criado;



- Vamos acrescentar mais um **label** à **direita** da segunda caixa de texto, para indicar o resultado (**símbolo =**) ;

```
public TelaCalc() {
    super("PPC - Multiplicação Lenta...");
    FlowLayout layout = new FlowLayout();
    this.setLayout(layout);
    this.setSize(800, 500);

    JTextField texto1 = new JTextField(15);
    this.add(texto1);

    JLabel simboloMultiplica = new JLabel("x");
    this.add(simboloMultiplica);

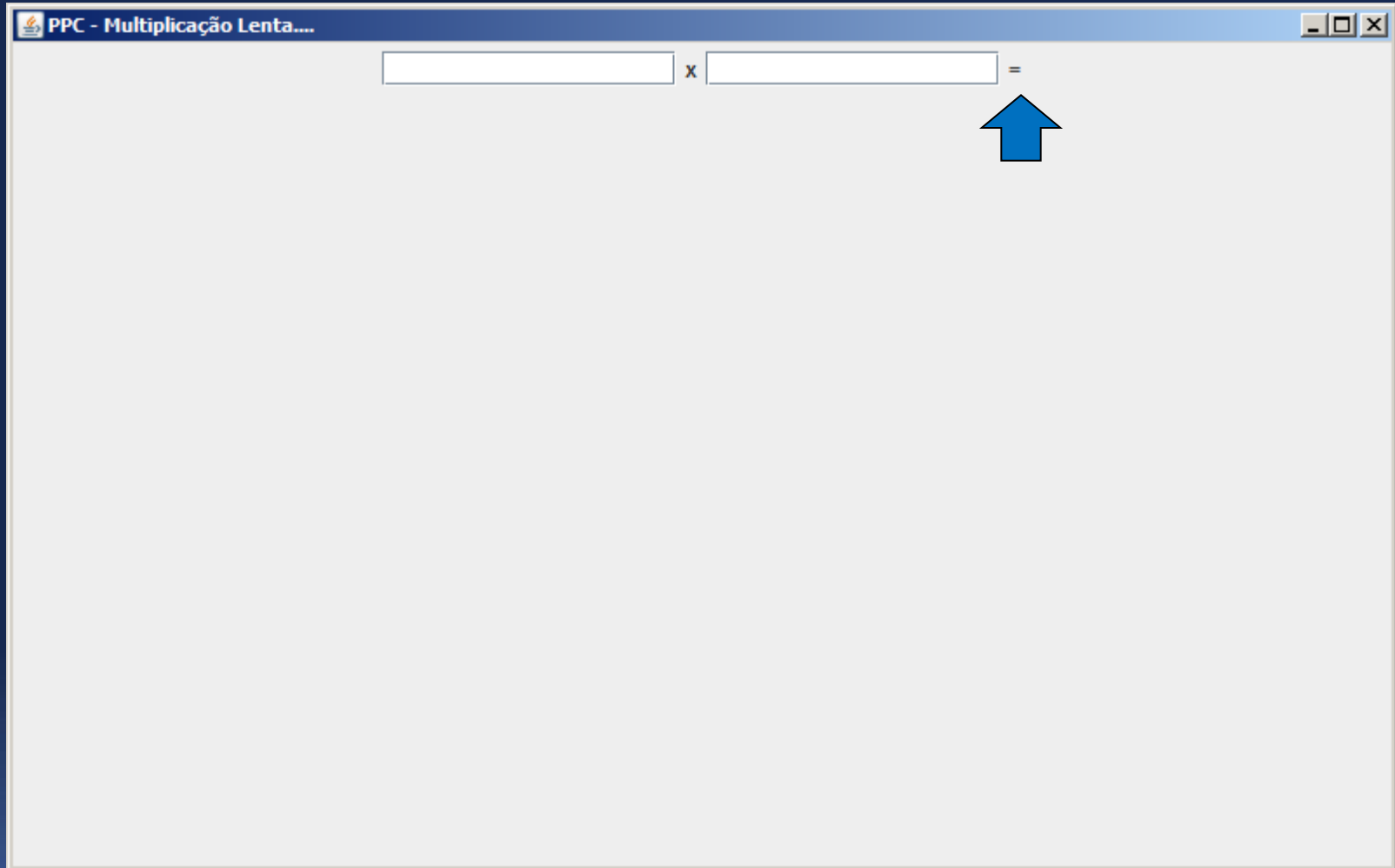
    JTextField texto2 = new JTextField(15);
    this.add(texto2);

    JLabel simboloIgual= new JLabel("=");
    this.add(simboloIgual);

    this.setVisible(true);
    this.setLocationRelativeTo(null);
}
```



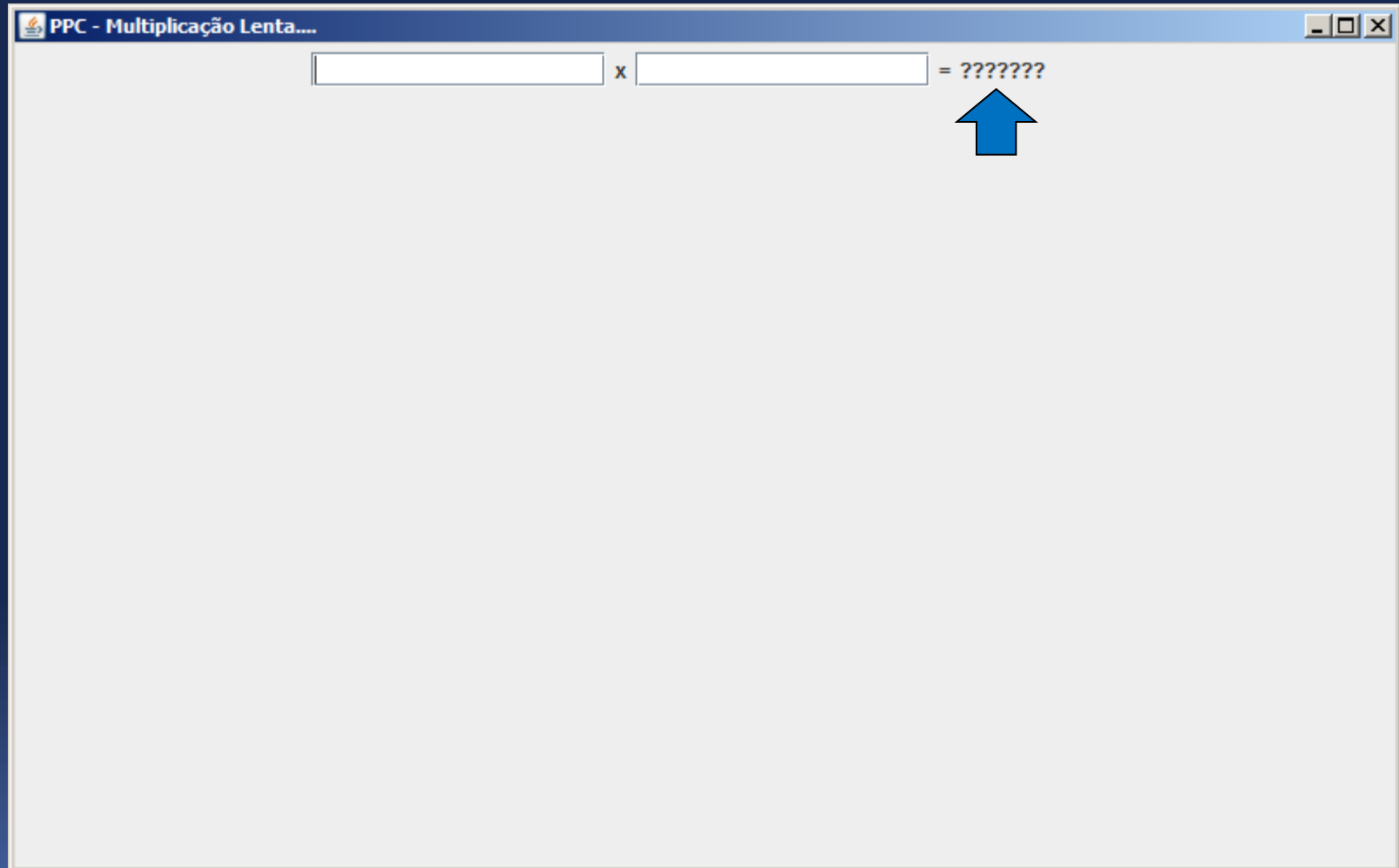
- Executando a interface, para visualizar o label criado;



- Vamos agora adicionar um **label** no qual a aplicação irá exibir ao usuário o **resultado** da multiplicação;
- Vamos incluir nesse Label os caracteres "????";

```
public TelaCalc() {  
    super("PPC - Multiplicação Lenta....");  
    FlowLayout layout = new FlowLayout();  
    this.setLayout(layout);  
    this.setSize(800, 500);  
  
    JTextField texto1 = new JTextField(15);  
    this.add(texto1);  
  
    JLabel simboloMultiplica = new JLabel("x");  
    this.add(simboloMultiplica);  
  
    JTextField texto2 = new JTextField(15);  
    this.add(texto2);  
  
    JLabel simboloIgual= new JLabel("=");  
    this.add(simboloIgual);  
  
    JLabel resultado = new JLabel("???????");  
    this.add(resultado);  
  
    this.setVisible(true);  
    this.setLocationRelativeTo(null);  
}
```

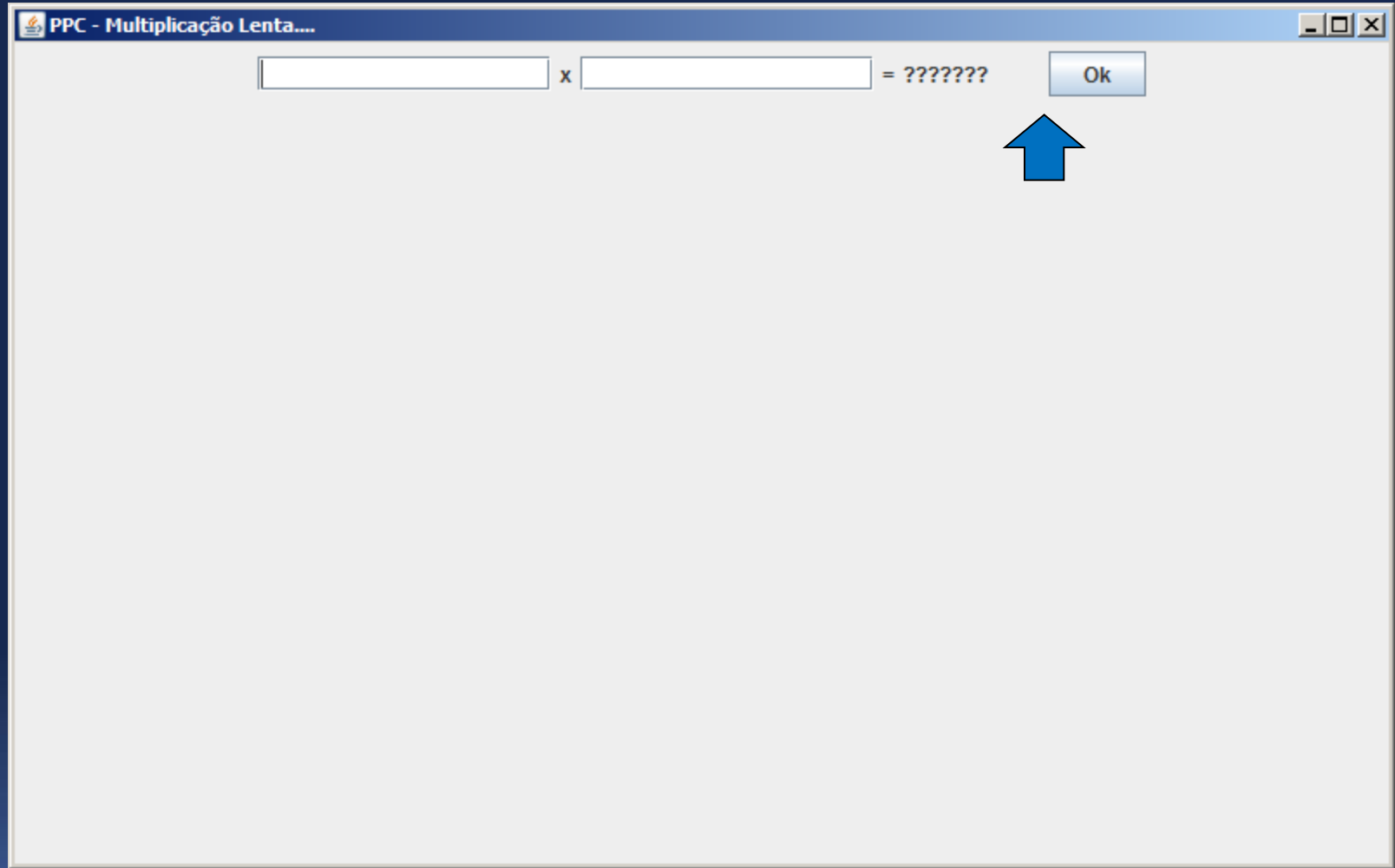

- Vamos agora adicionar um label no qual a aplicação irá exibir ao usuário o resultado da multiplicação;
- Vamos incluir nesse Label os caracteres "?????";



- Após o usuário digitar os 2 valores a serem multiplicados, será necessário clicar em algum **Botão** para enviar os dados à aplicação para processá-los;
- Vamos então inserir na interface um **Botão de Comando**.

```
public TelaCalc() {  
    super("PPC - Multiplicação Lenta...");  
    FlowLayout layout = new FlowLayout();  
    this.setLayout(layout);  
    this.setSize(800, 500);  
  
    JTextField texto1 = new JTextField(15);  
    this.add(texto1);  
  
    JLabel simboloMultiplica = new JLabel("x");  
    this.add(simboloMultiplica);  
  
    JTextField texto2 = new JTextField(15);  
    this.add(texto2);  
  
    JLabel simboloIgual= new JLabel("=");  
    this.add(simboloIgual);  
  
    JLabel resultado = new JLabel("???????");  
    this.add(resultado);  
  
    JButton botao = new JButton(" Ok ");  
    this.add(botao);  
  
    this.setVisible(true);  
    this.setLocationRelativeTo(null);  
}
```

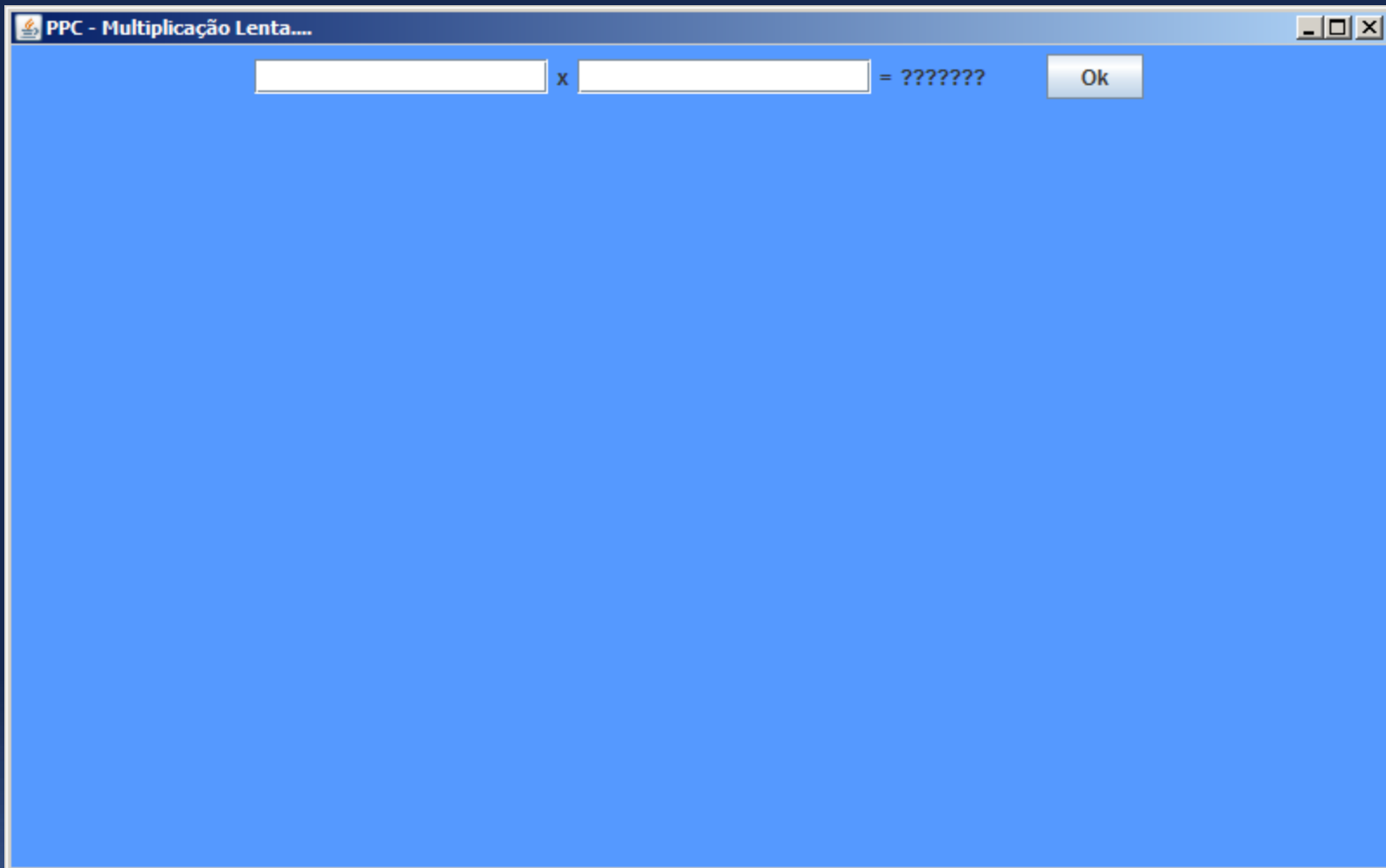
- Executando a interface, podemos visualizar o Botão.



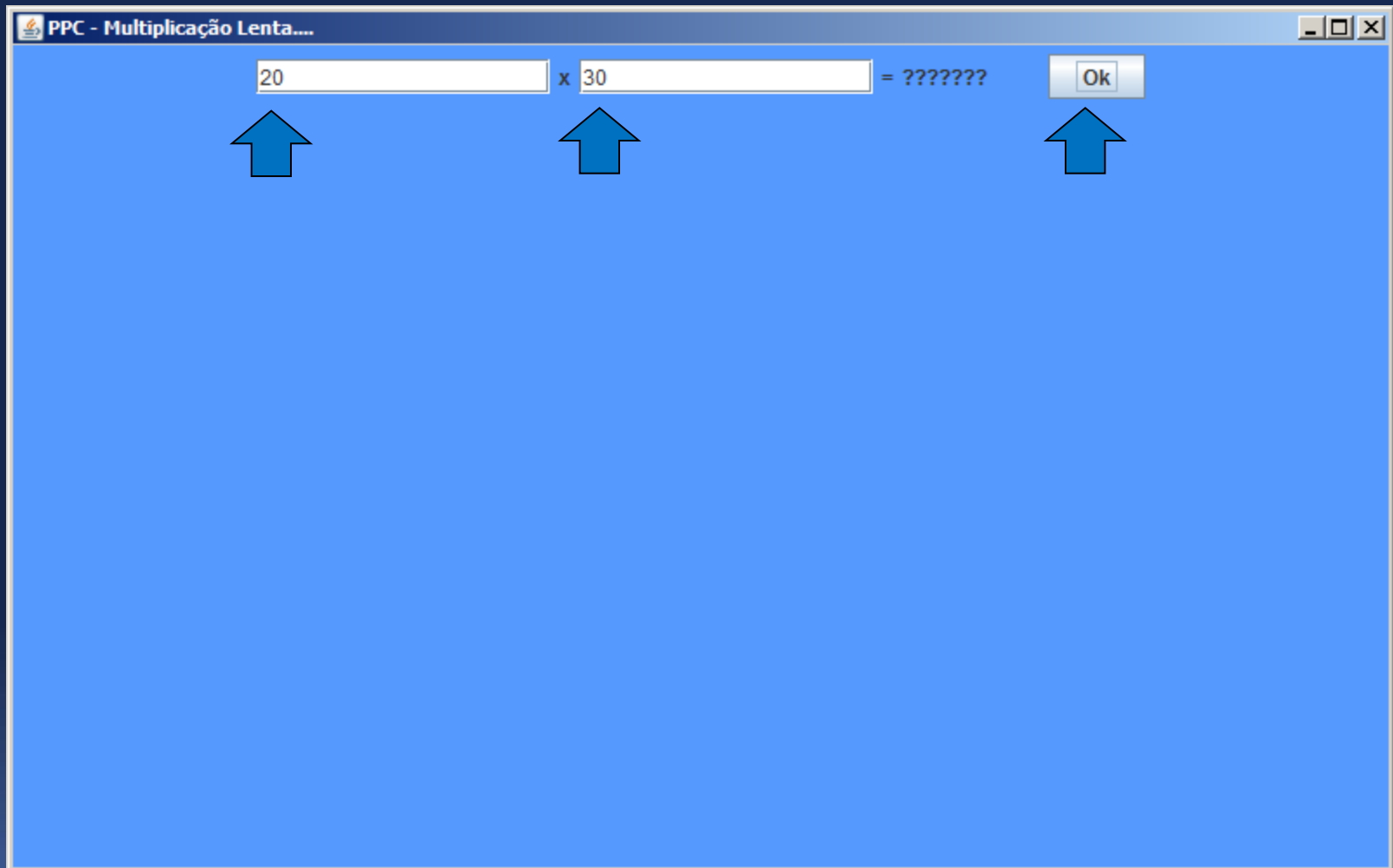
- Vamos mudar a cor da interface, para melhorar o visual.

```
public TelaCalc() {  
    super("PPC - Multiplicação Lenta....");  
    FlowLayout layout = new FlowLayout();  
    this.setLayout(layout);  
    this.setSize(800, 500);  
  
    JTextField texto1 = new JTextField(15);  
    this.add(texto1);  
  
    JLabel simboloMultiplica = new JLabel("x");  
    this.add(simboloMultiplica);  
  
    JTextField texto2 = new JTextField(15);  
    this.add(texto2);  
  
    JLabel simboloIgual= new JLabel("=");  
    this.add(simboloIgual);  
  
    JLabel resultado = new JLabel("???????");  
    this.add(resultado);  
  
    JButton botao = new JButton(" Ok ");  
    this.add(botao);  
  
    Color cor = new Color(0x5599FF);  
    this.getContentPane().setBackground(cor);  
  
    this.setVisible(true);  
    this.setLocationRelativeTo(null);  
}
```

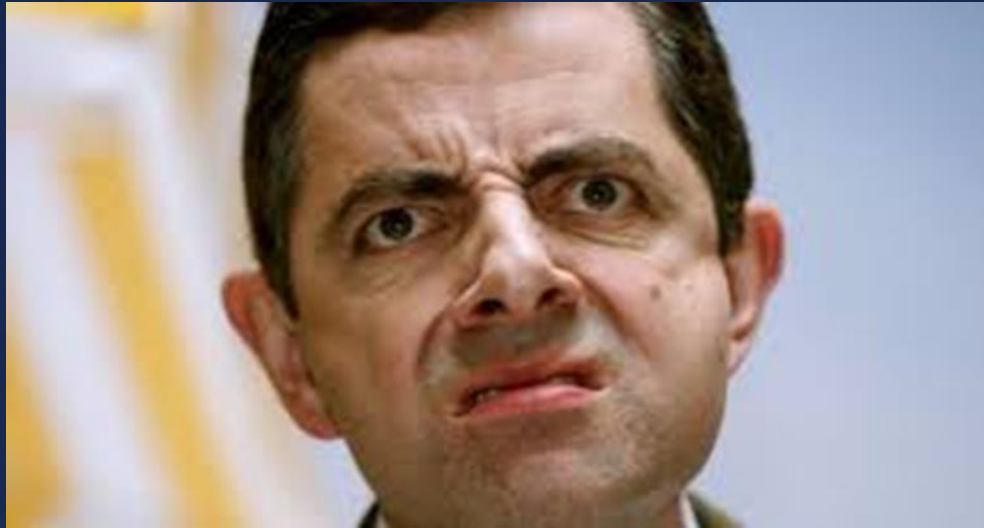
- Vamos mudar a cor da interface, para melhorar o visual.



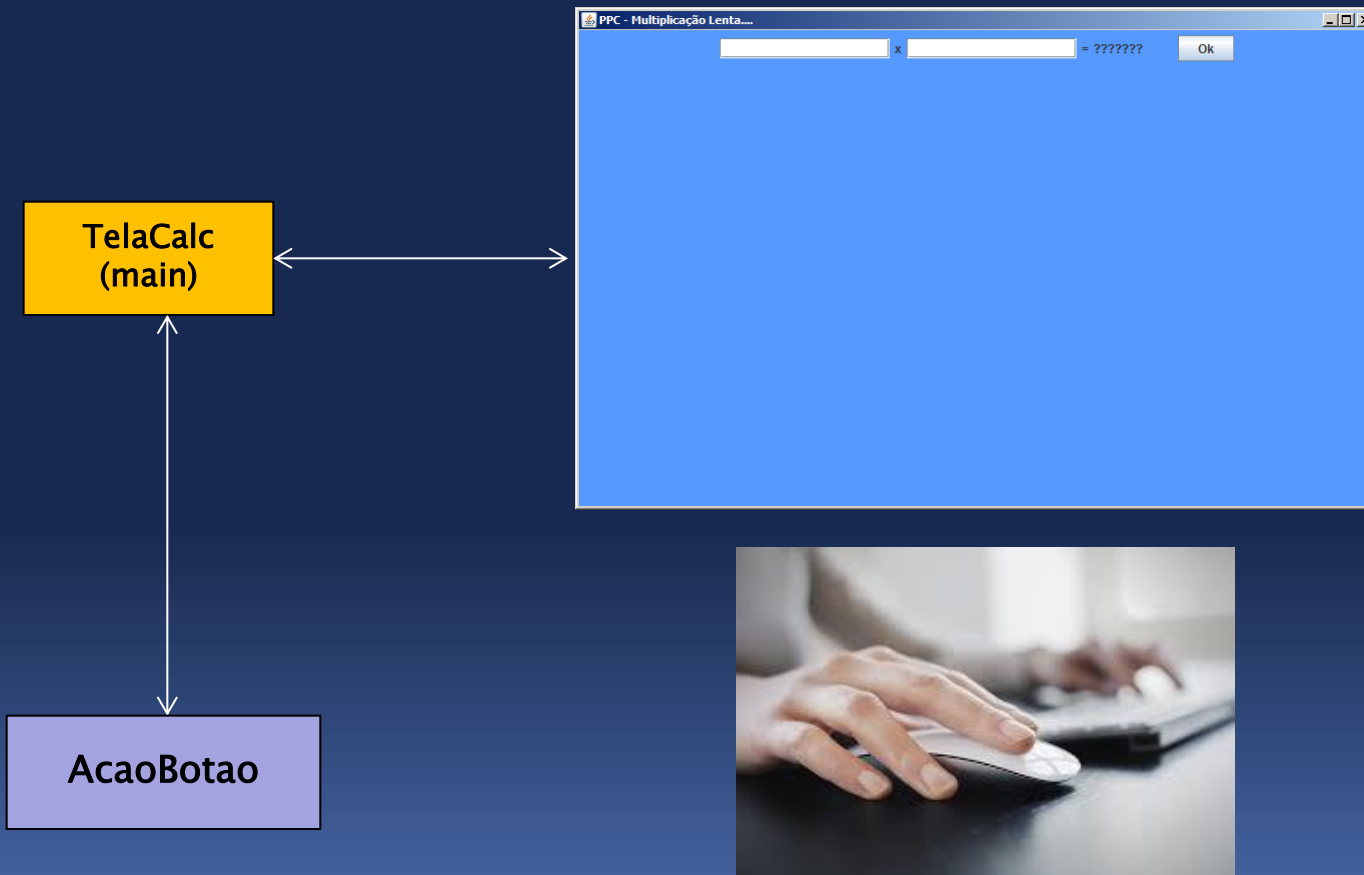
- Vamos agora entrar com dados nas caixas de texto e clicar no botão ok, para processar a multiplicação dos dados.



Ué ... Mas, ao clicar no Botão nada acontece ?????



- **Sim**, precisamos agora tratar o **evento** de Clique do Mouse;
- Vamos criar agora uma **classe** que será **responsável** pelo tratamento do **clique** do mouse (**AcaoBotao.class**);
- Essa classe deverá implementar a interface **ActionListener**.



Classe `AcaoBotao.class` para tratar clique do Mouse

- Essa classe tratará os campos da tela que serão preenchidos pelo usuário e pelo campo da tela onde será mostrado o resultado (`texto1`, `texto2` e `resultado`).

```
package br.uscs;

import java.awt.event.ActionEvent;

public class AcaoBotao implements ActionListener {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

}
```

Classe `AcaoBotao.class` para tratar clique do Mouse


- Vamos agora implementar o **construtor** para que o objeto a ser criado a partir dessa classe contenha os dados informados pelo usuário na interface (`texto1`, `texto2`), bem como o campo `resultado` que será objeto do cálculo.

```
public class AcaoBotao implements ActionListener {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

    public AcaoBotao(JTextField texto1, JTextField texto2, JLabel resultado) {

        this.texto1 = texto1;
        this.texto2 = texto2;
        this.resultado = resultado;
    }
}
```



Classe **AcaoBotao.class** para tratar clique do Mouse

- O Eclipse está sinalizando um erro pois a classe AcaoBotao implementa a interface ActionListener e, portanto, deve implementar o método **actionPerformed** para tratamento do evento de clique do mouse.

```
public class AcaoBotao implements ActionListener {
    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

    public AcaoBotao(JTextField texto1, JTextField texto2, JLabel resultado) {
        this.texto1 = texto1;
        this.texto2 = texto2;
        this.resultado = resultado;
    }
}
```

Classe `AcaoBotao.class` para tratar clique do Mouse

- O Eclipse está sinalizando um erro pois a classe `AcaoBotao` implementa a interface `ActionListener` e, portanto, deve implementar o método `actionPerformed` para tratamento do evento de clique do mouse.

```
public class AcaoBotao implements ActionListener {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

    public AcaoBotao(JTextField texto1, JTextField texto2, JLabel resultado) {

        this.texto1 = texto1;
        this.texto2 = texto2;
        this.resultado = resultado;
    }

    public void actionPerformed(ActionEvent e) {
        Long valor1 = Long.parseLong(texto1.getText());
        Long valor2 = Long.parseLong(texto2.getText());
    }
}
```

- O método **actionPerformed** receberá como parâmetro o evento **e** do clique de mouse;
- **Adicionalmente**, converterá os textos digitados nas caixas de texto de String para Long (**parseLong**).

```
package br.uscs;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JLabel;
import javax.swing.JTextField;


public class AcaoBotao implements ActionListener {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

    public void actionPerformed(ActionEvent e) {

        Long valor1 = Long.parseLong(texto1.getText());
        Long valor2 = Long.parseLong(texto2.getText());

    }
}
```



- O método **actionPerformed** poderá agora efetuar o cálculo da multiplicação dos valores entrados pelo usuário;
- Faremos um cálculo de forma lenta, para ilustrar o emprego de Threads.

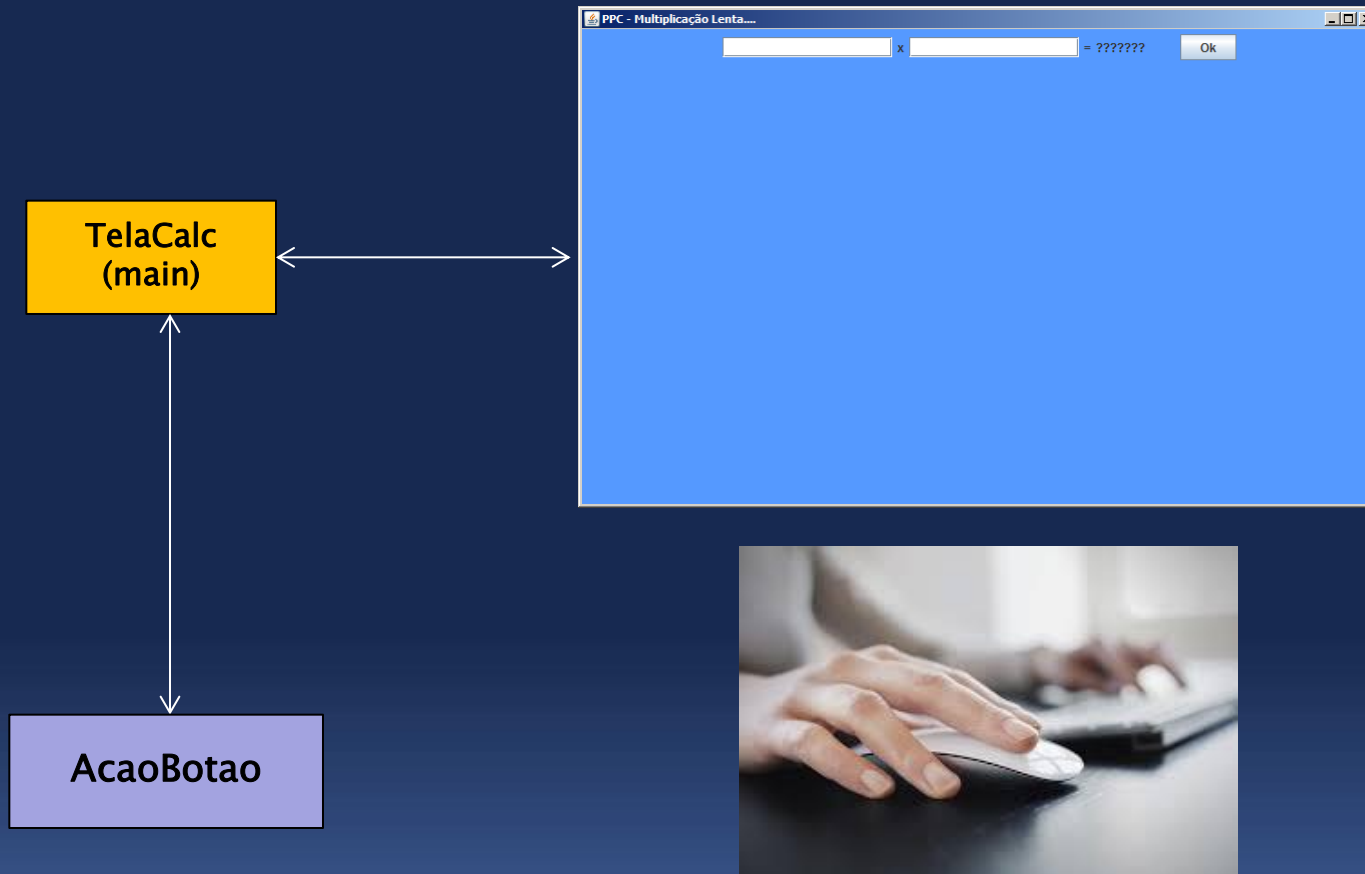
```
public void actionPerformed(ActionEvent e) {  
  
    Long valor1 = Long.parseLong(texto1.getText());  
    Long valor2 = Long.parseLong(texto2.getText());  
  
    BigInteger calculo = new BigInteger("0");  
  
    ➡ for (int i = 0; i < valor1; i++) {  
        for (int j = 0; j < valor2; j++) {  
            calculo = calculo.add(new BigInteger("1"));  
        }  
    }  
}
```

- O resultado do cálculo deverá ser postado no campo de Texto **resultado** da interface (função **setText**).

```
public void actionPerformed(ActionEvent e) {  
  
    Long valor1 = Long.parseLong(texto1.getText());  
    Long valor2 = Long.parseLong(texto2.getText());  
  
    BigInteger calculo = new BigInteger("0");  
  
    for (int i = 0; i < valor1; i++) {  
        for (int j = 0; j < valor2; j++) {  
            calculo = calculo.add(new BigInteger("1"));  
        }  
    }  
  
    resultado.setText(calculo.toString());  
}
```



- Com isso, terminamos a classe de tratamento do clique do mouse;
- Precisamos agora ligar a classe principal (**TelaCalc**) com a classe de tratamento de clique do mouse (**AcaoBotao**).



- Para fazer essa ligação, devemos voltar à classe **TelaCalc** e incluir a chamada para a classe de tratamento de evento (**AcaoBotao**);
- Para isso, vamos instanciar a classe **AcaoBotao** e associá-la ao botão.

```
public class TelaCalc extends JFrame {

    public TelaCalc() {
        super("PPC - Multiplicação Lenta...");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);

        JTextField texto1 = new JTextField(15);
        this.add(texto1);

        JLabel simboloMultiplica = new JLabel("x");
        this.add(simboloMultiplica);

        JTextField texto2 = new JTextField(15);
        this.add(texto2);

        JLabel simboloIgual= new JLabel("=");
        this.add(simboloIgual);

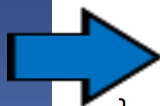
        JLabel resultado = new JLabel("???????");

        JButton botao = new JButton(" Ok ");
        this.add(botao);

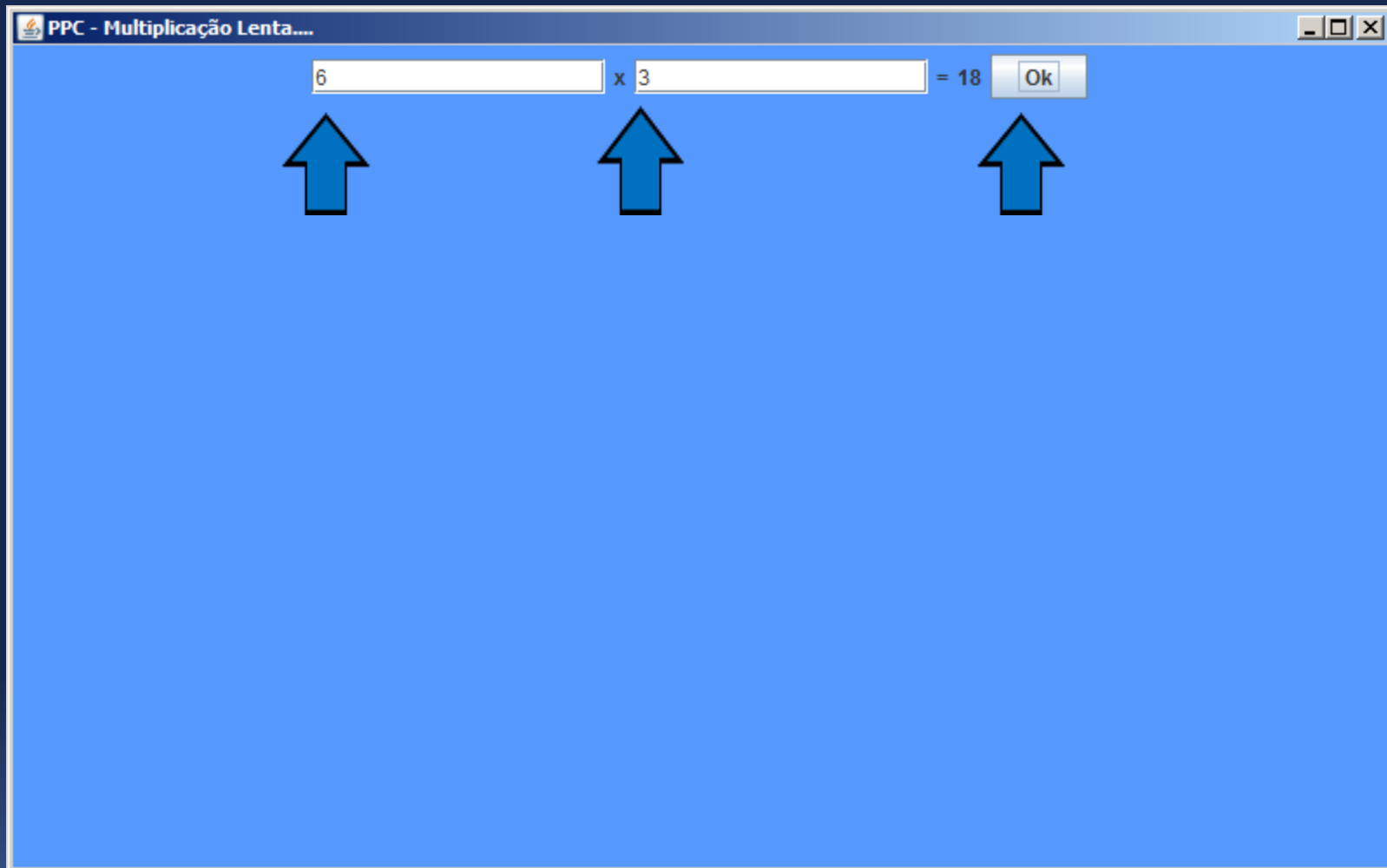
        Color cor = new Color(0x5599FF);
        this.getContentPane().setBackground(cor);

        this.setVisible(true);
        this.setLocationRelativeTo(null);

        AcaoBotao acaobotao = new AcaoBotao(texto1, texto2, resultado);
        botao.addActionListener(acaobotao);
    }
}
```



- Vamos agora rodar a aplicação e checar o resultado de um cálculo, após a entradas destes e clicando no botão ok.



- Vamos agora, programar mais um botão para encerrar a aplicação.

```
JLabel resultado = new JLabel("???????");
this.add(resultado);
```

```
JButton botao = new JButton(" Ok ");
this.add(botao);
```



```
JButton botaoFim = new JButton("Fim");
this.add(botaoFim);
```

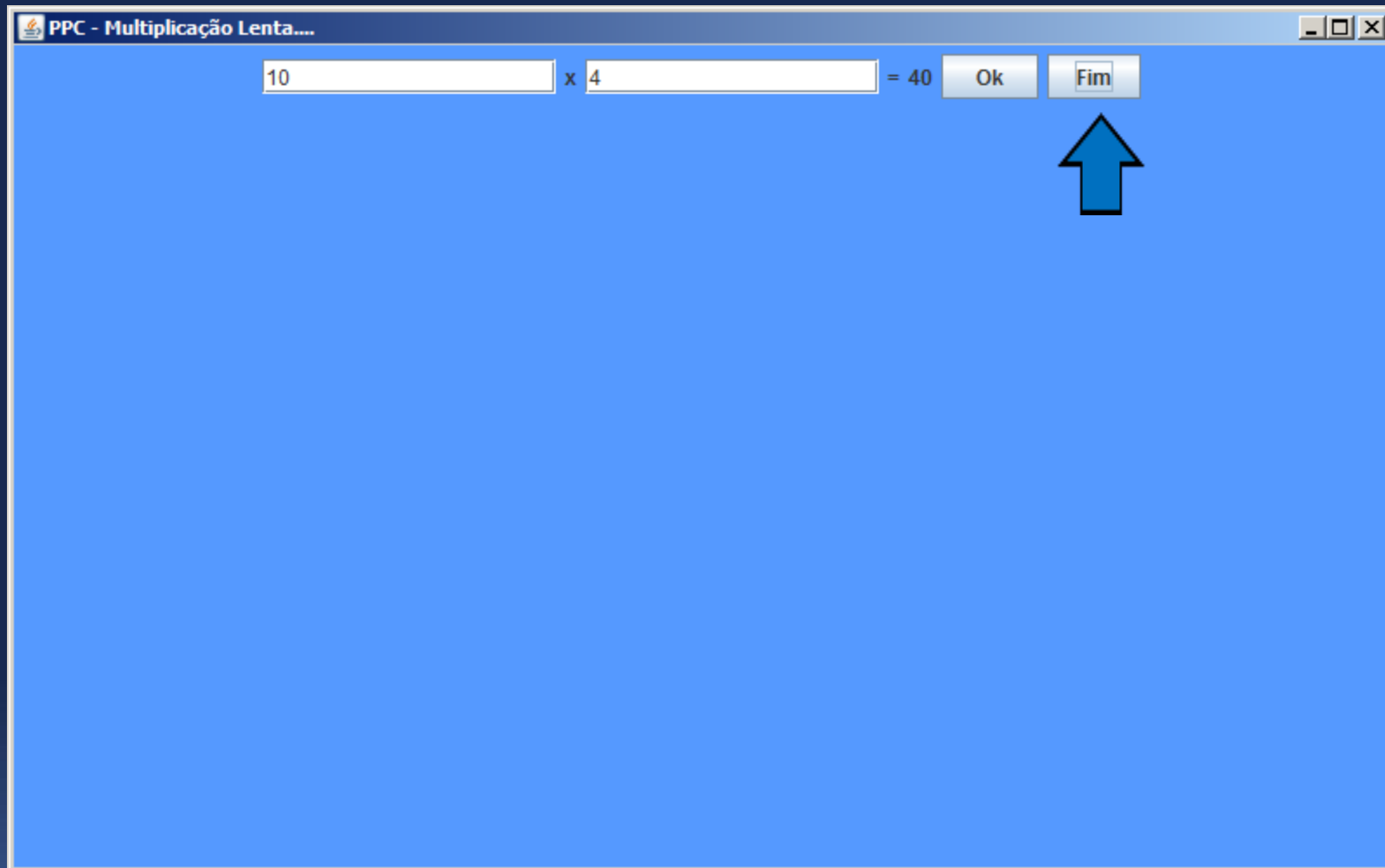
```
Color cor = new Color(0x5599FF);
this.getContentPane().setBackground(cor);
```

```
this.setVisible(true);
this.setLocationRelativeTo(null);
```

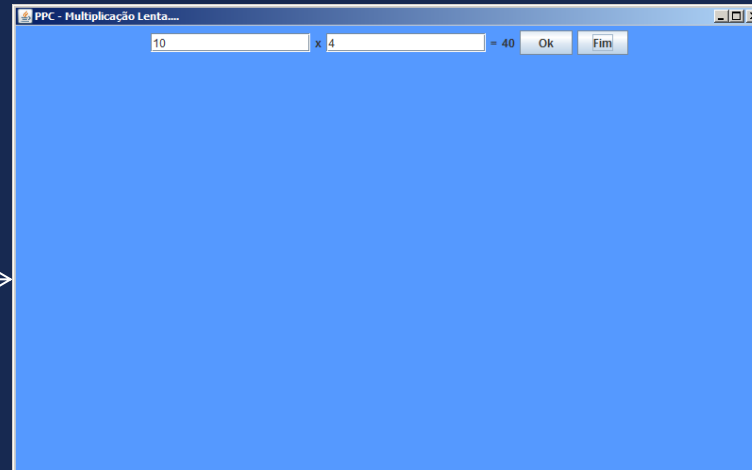
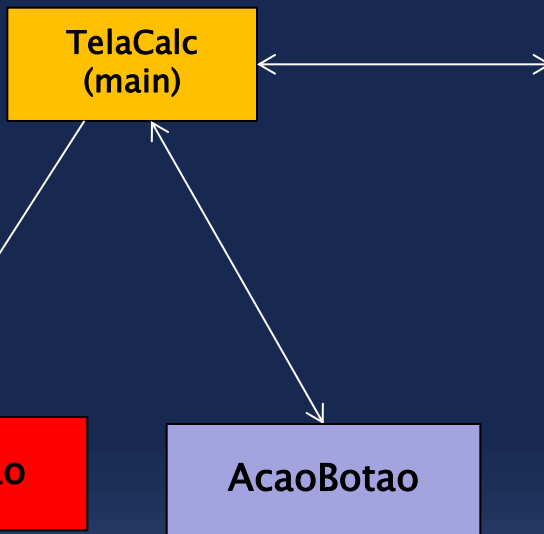
```
AcaoBotao acaobotao = new AcaoBotao(texto1, texto2, resultado);
botao.addActionListener(acaobotao);
```

```
}
```

- Processando a aplicação para visualizar o botão.



- Escrevendo o código para tratar o evento;
- Criaremos uma nova classe chamada **AcaoFim.class**.



Programando a classe **AcaoFim**



```
package br.uscs;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;

public class AcaoFim implements ActionListener {

    public void actionPerformed(ActionEvent e) {

        JOptionPane.showMessageDialog(null, "Fim de Execução ....");
        System.exit(0);
    }
}
```

Vinculando a classe **AcaoFim** com a interface principal.

```
public class TelaCalc extends JFrame {

    public TelaCalc() {
        super("PPC - Multiplicação Lenta...");
        FlowLayout layout = new FlowLayout();
        this.setLayout(layout);
        this.setSize(800, 500);

        JTextField texto1 = new JTextField(15);
        this.add(texto1);

        JLabel simboloMultiplica = new JLabel("x");
        this.add(simboloMultiplica);

        JTextField texto2 = new JTextField(15);
        this.add(texto2);

        JLabel simboloIgual= new JLabel("=");
        this.add(simboloIgual);

        JLabel resultado = new JLabel("??????");

        JButton botao = new JButton(" Ok ");
        this.add(botao);

        JButton botaoFim = new JButton("Fim");
        this.add(botaoFim);

        Color cor = new Color(0x5599FF);
        this.getContentPane().setBackground(cor);

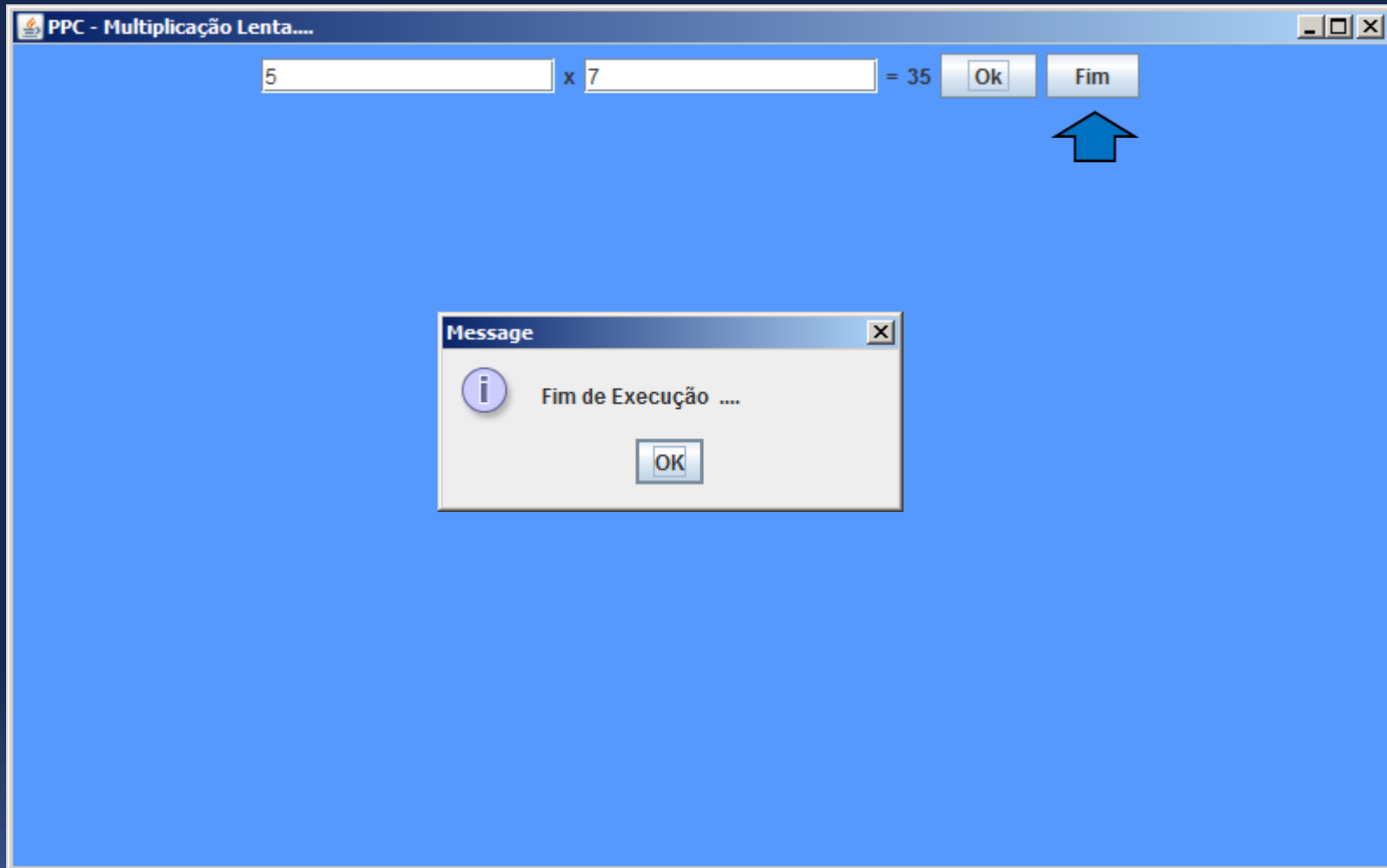
        this.setVisible(true);
        this.setLocationRelativeTo(null);

        AcaoBotao acaobotao = new AcaoBotao(texto1, texto2, resultado);
        botao.addActionListener(acaobotao);

        AcaoFim acaofim = new AcaoFim();
        botaoFim.addActionListener(acaofim);
    }
}
```



- Executando a interface e clicando no botão **Fim**.



Executando a aplicação

- A aplicação foi desenvolvida para se explorar os conceitos de **threads** existentes na Linguagem Java;
- Para tanto, o cálculo da multiplicação foi feito, propositalmente, com **baixo desempenho**;
- Para melhor visualizar o processamento da rotina de cálculo, incluiremo na função **actionPerformed** da classe **AcaoBotao**, uma mensagem para informar ao usuário o momento em que o cálculo foi finalizado.

Incluindo mensagem em actionPerformed

```
public void actionPerformed(ActionEvent e) {

    Long valor1 = Long.parseLong(texto1.getText());
    Long valor2 = Long.parseLong(texto2.getText());

    if (e.getSource() == "botao_Fim") {
        JOptionPane.showMessageDialog(null, "Fim de Execução ....");
        System.exit(0);
    }

    else {

        BigInteger calculo = new BigInteger("0");

        for (int i = 0; i < valor1; i++) {
            for (int j = 0; j < valor2; j++) {
                calculo = calculo.add(new BigInteger("1"));
            }
        }
        ➡ JOptionPane.showMessageDialog(null, "Cálculo finalizado.....Grato pela paciência ....");
        resultado.setText(calculo.toString());
    }

}
```

Processando a aplicação

- Vamos executar a aplicação para os seguintes **dados** abaixo;
- **Anote** na planilha abaixo os resultados.

Valor1	Valor2	Resultado
12345	1000	
12345	5000	
12345	10000	
12345	20000	
12345	30000	
12345	40000	
12345	50000	

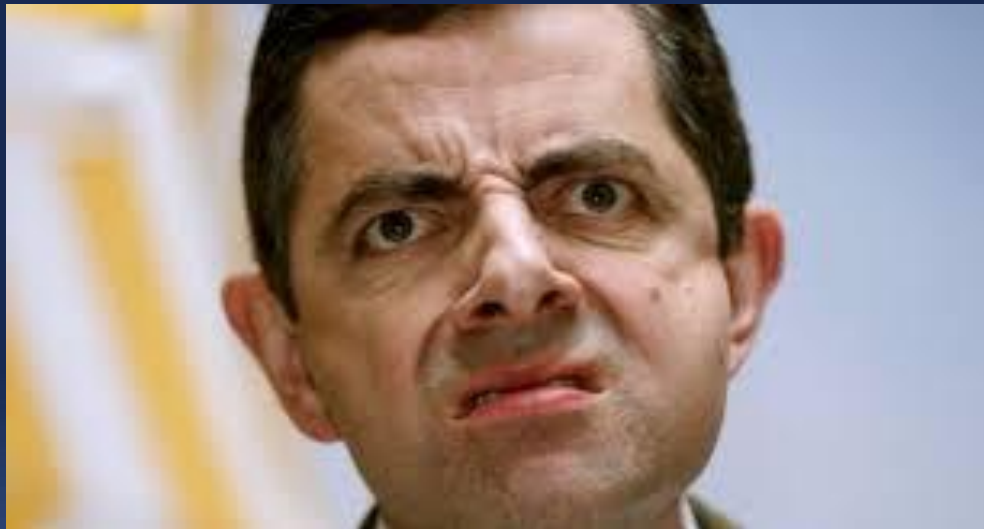
Avaliando a execução

- Para os últimos valores, (12345) e (50000) observe que a interface fica indisponível ao usuário enquanto o cálculo **não** for finalizado;
- Trata-se, portanto, de uma execução tipicamente **sequencial**.

Valor1	Valor2	Resultado
12345	1000	
12345	5000	
12345	10000	
12345	20000	
12345	30000	
12345	40000	
12345	50000	



Enquanto a aplicação faz o cálculo
como proceder para que ela nesse
tempo executasse alguma outra tarefa?



Simples, vamos programar Threads !



Aplicando Threads na aplicação

- A **função** que efetua o cálculo **poderia** ser processada por um **thread** específico para esta tarefa.
- Assim, iremos trabalhar com a função **actionPerformed()**.

```
public void actionPerformed(ActionEvent e) {

    Long valor1 = Long.parseLong(texto1.getText());
    Long valor2 = Long.parseLong(texto2.getText());

    if (e.getSource() == "botao_Fim") {
        JOptionPane.showMessageDialog(null, "Fim de Execução ....");
        System.exit(0);
    }

    else {

        BigInteger calculo = new BigInteger("0");

        for (int i = 0; i < valor1; i++) {
            for (int j = 0; j < valor2; j++) {
                calculo = calculo.add(new BigInteger("1"));
            }
        }
        JOptionPane.showMessageDialog(null, "Cálculo finalizado.....Grato pela paciência ....");
        resultado.setText(calculo.toString());

    }

}
```

Aplicando Threads na aplicação

- Para isso, vamos inicialmente criar um novo **Thread** na aplicação;
- Vimos na unidade anterior que para criarmos novo **thread** devemos escrever o código:

```
Thread threadMultiplicacao – new Thread();
```


Aplicando Threads na aplicação

- Após a criação do **Thread** devemos passar para ele a **tarefa** de efetuar a **multiplicação** ;
- Ao se observar o construtor de um Thread na classe `java.lang.Thread`, pode-se constatar que podemos passar para esse construtor um objeto da interface **Runnable** que representa algo que pode ser **executado** (algo que é **executável**).

Construtor de Thread

java.lang

Class Thread

Constructors

Constructor and Description

Thread()

Allocates a new Thread object.



Thread(Runnable target)

Allocates a new Thread object.

Thread(Runnable target, String name)

Allocates a new Thread object.

Thread(String name)

Allocates a new Thread object.

A interface Runnable

- A interface **Runnable** possui apenas um método;
- Esse método é **run()**;
- Nesse método **run()** define-se a tarefa que será **processada** pelo **thread**, no nosso caso a operação de **multiplicação**.

```
java.lang
```

```
Interface Runnable
```

```
void run()
```



When an object implementing interface Runnable is used to create a thread,

starting the thread causes the object's run method to be called in that separately executing thread.

Associando um Thread à multiplicação

- Vamos então criar uma nova classe, chamada **TaskMultiplicacao**, que implementa a **interface Runnable**;

```
package br.uscs;  
  
public class TaskMultiplicacao implements Runnable {  
    @Override  
    public void run() {  
        // Esse método deverá processar a multiplicação  
    }  
}
```



Associando um Thread à multiplicação

- Vamos agora incluir na classe `TaskMultiplicacao` o código que representa o thread associado à multiplicação;

```
package br.uscs;
import java.math.BigInteger;

public class TaskMultiplicacao implements Runnable {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

    public TaskMultiplicacao (JTextField texto1, JTextField texto2, JLabel resultado) {
        this.texto1 = texto1;
        this.texto2 = texto2;
        this.resultado = resultado;
    }

    @Override
    public void run() {
        Long valor1 = Long.parseLong(texto1.getText());
        Long valor2 = Long.parseLong(texto2.getText());

        BigInteger calculo = new BigInteger("0");
        for (int i = 0; i < valor1; i++) {
            for (int j = 0; j < valor2; j++) {
                calculo = calculo.add(new BigInteger("1"));
            }
        }
        JOptionPane.showMessageDialog(null, "Cálculo finalizado....Grato pela paciência ....");
        resultado.setText(calculo.toString());
    }
}
```



Associando um Thread à multiplicação

- Para finalizar, vamos acertar o código da classe **AcaoBotao** na função **actionPerformed()** para definir o **novo** thread;
- Para iniciar a execução do novo thread deverá ser codificada a função **start()**;

```
package br.uscs;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class AcaoBotao implements ActionListener {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

    public AcaoBotao(JTextField texto1, JTextField texto2, JLabel resultado) {

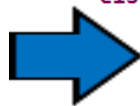
        this.texto1 = texto1;
        this.texto2 = texto2;
        this.resultado = resultado;
    }

    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == "botao_Fim") {
            JOptionPane.showMessageDialog(null, "Fim de Execução ....");
            System.exit(0);
        }

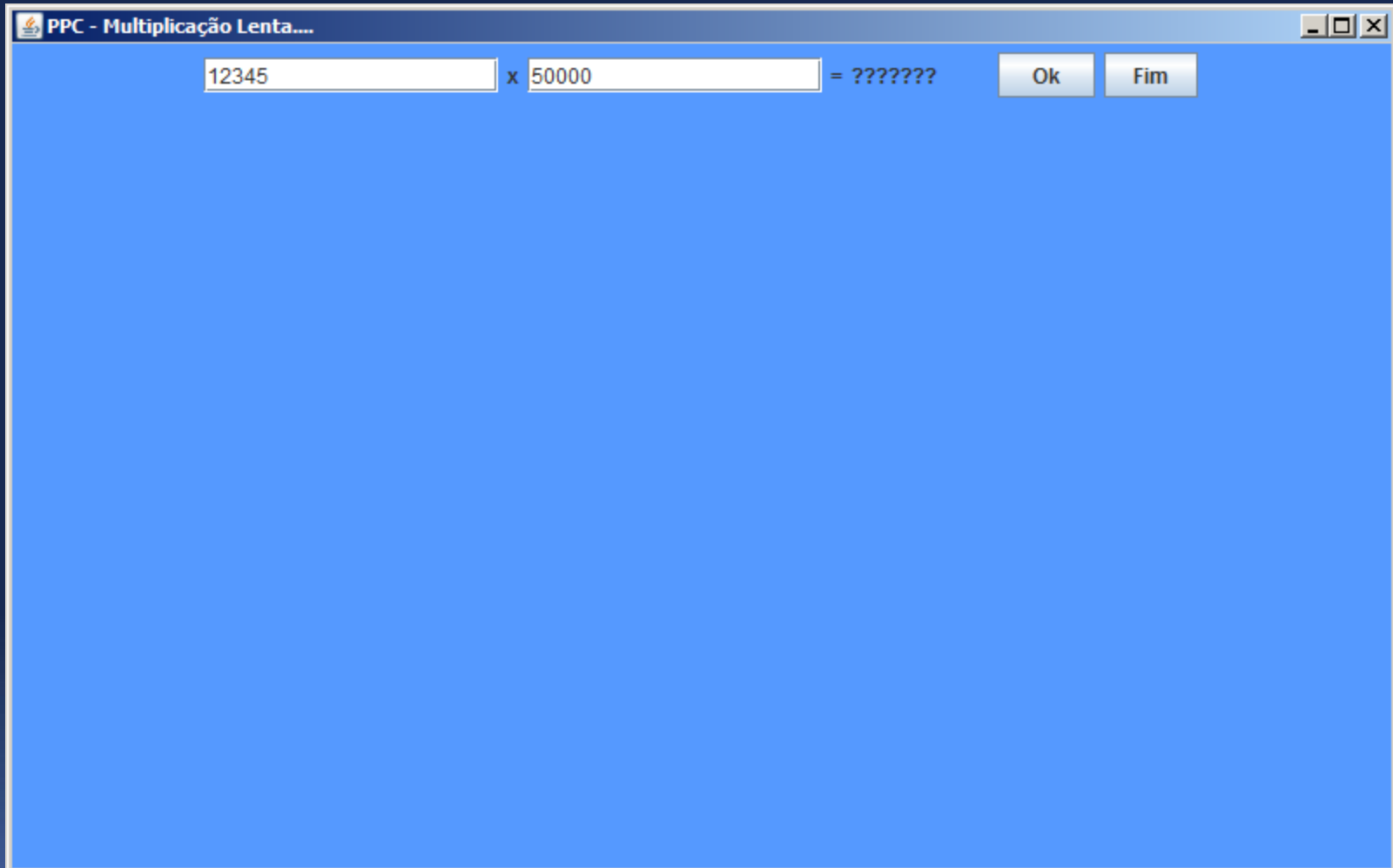
        else {
            TaskMultiplicacao task = new TaskMultiplicacao(texto1, texto2, resultado);
            Thread threadMultiplicacao = new Thread(task);

            threadMultiplicacao.start();
        }
    }
}
```



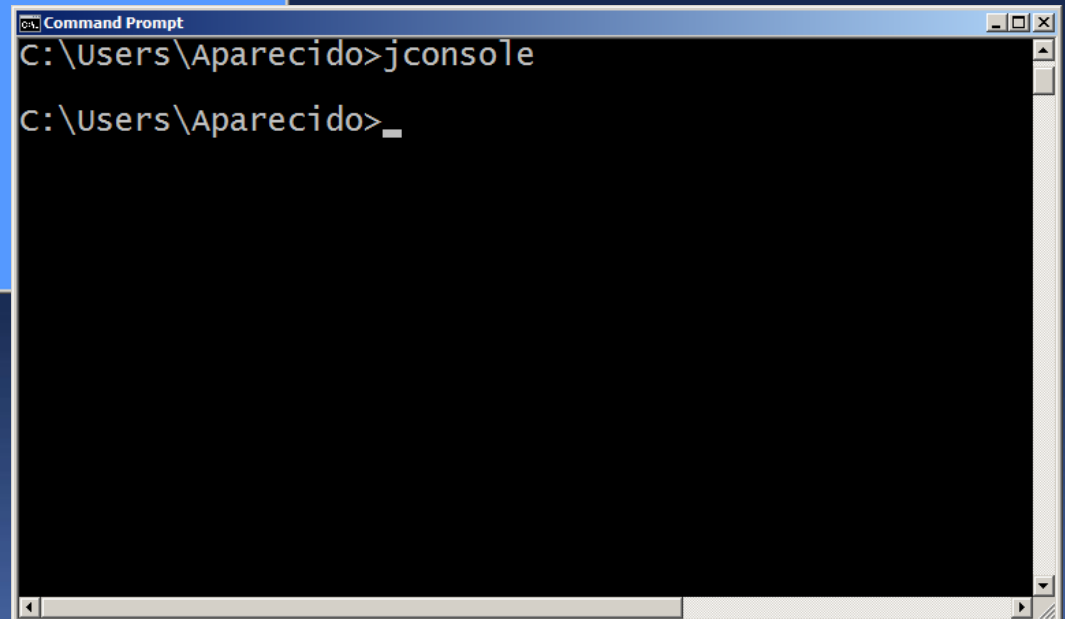
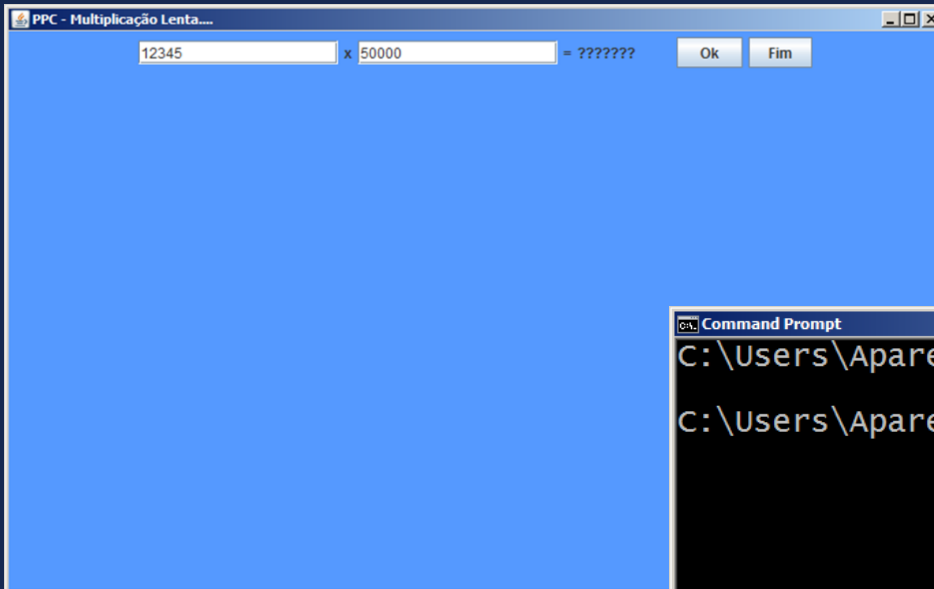
Executando a aplicação

- Reprocesse a aplicação com os valores **12345** e **50000** e verifique se as caixas de texto estão **liberadas** enquanto o **cálculo** da **multiplicação** é **feito**!

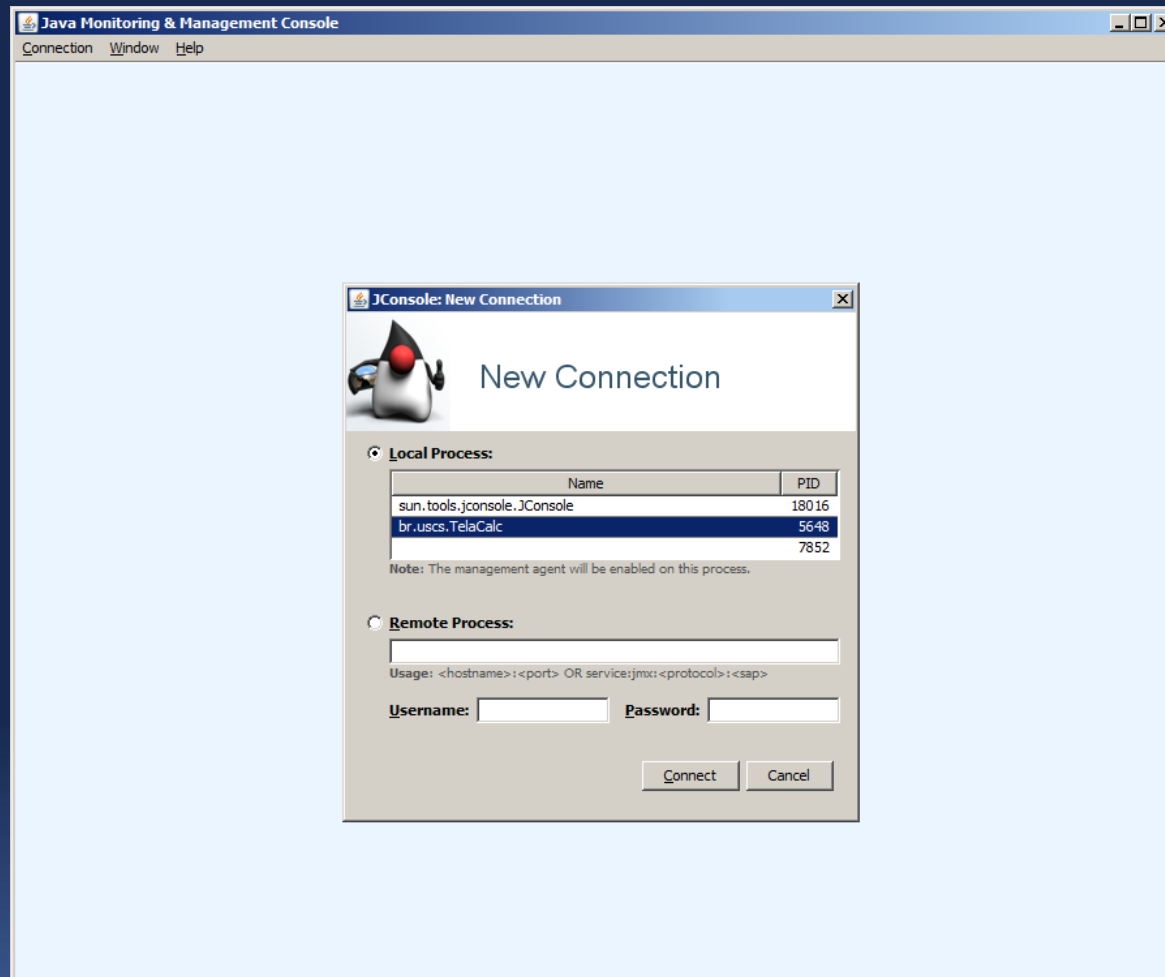


Checando a execução com JConsole

- **Reprocesse** a aplicação com os valores **12345** e **50000** e ative a ferramenta Jconsole para verificar a execução dos threads relativos à aplicação.

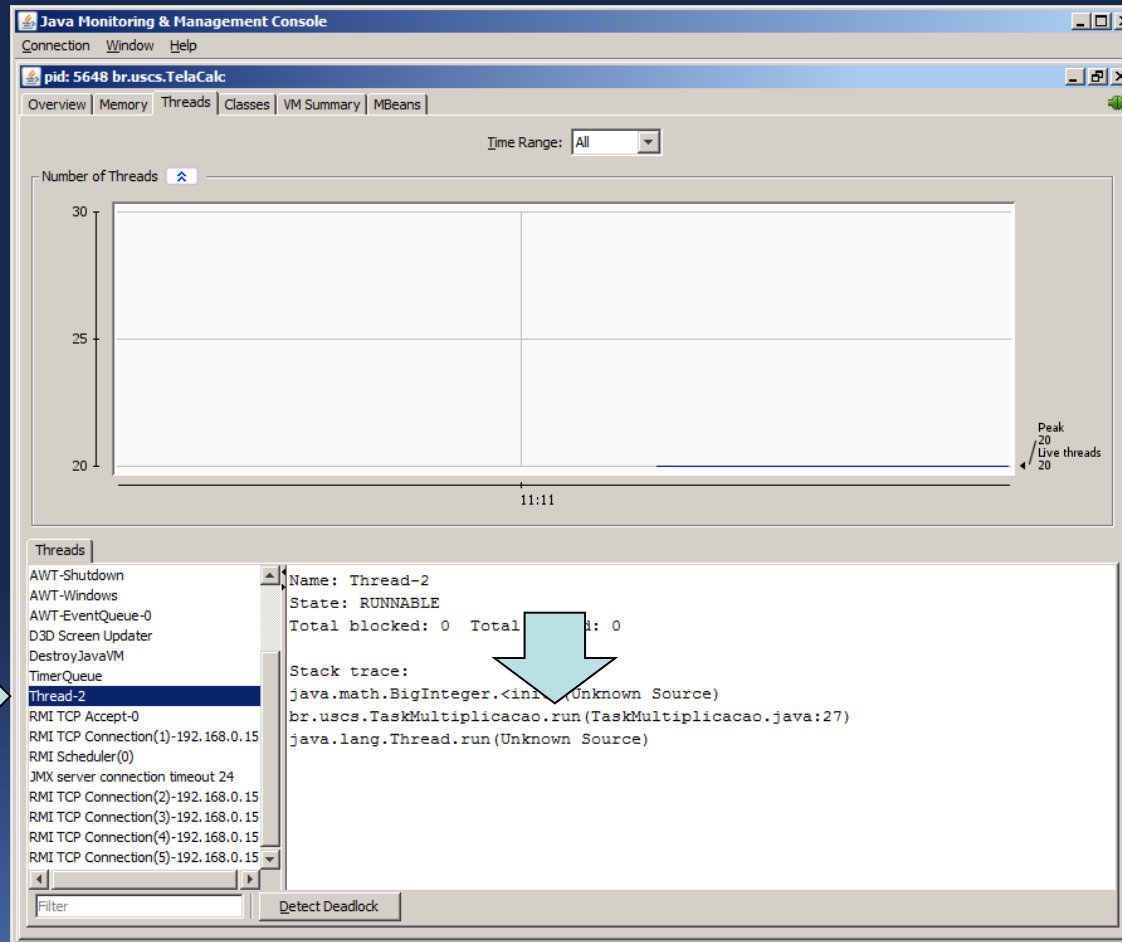


Conectando à aplicação com JConsole



Conectando à aplicação com JConsole

- O **thread** correspondente à função **main()** **não** é mais exibida da lista de Threads, uma vez que já foi processada. Mas, o **thread** correspondente à multiplicação **está sendo exibido**.
- A JVM deu um nome a esse thread, no nosso caso, **Thread-2**.



Aterando o nome do Thread

- Na classe o oThread está sendo criado, vamos incluir no construtor do Thread, um segundo parâmetro (String) que corresponde ao nome do Thread;
- Nomearemos o thread com o nome "USCS".

```
package br.uscs;
import java.awt.event.ActionEvent;

public class AcaoBotao implements ActionListener {

    private JTextField texto1;
    private JTextField texto2;
    private JLabel resultado;

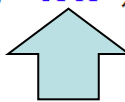
    public AcaoBotao(JTextField texto1, JTextField texto2, JLabel resultado) {

        this.texto1 = texto1;
        this.texto2 = texto2;
        this.resultado = resultado;
    }
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == "botao_Fim") {
            JOptionPane.showMessageDialog(null, "Fim de Execução ....");
            System.exit(0);
        }
        else {

            TaskMultiplicacao task = new TaskMultiplicacao(texto1, texto2, resultado);
            Thread threadMultiplicacao = new Thread(task, "USCS");

            threadMultiplicacao.start();
        }
    }
}
```



Reprocessando a Aplicação e ativando Jconsole



- Na lista de threads exibida pelo Jconsole, pode-se ver agora o thread "USCS" em execução.

