Paradigmas de Linguagens de Programação – 2013.1
Aula 10 - Programação Concorrente / Prática

Demonstração 1
Thread

1. Abrir o NetBeans
2. Criar um novo projeto Java simples
3. Criar a seguinte classe:

```
public class TesteThread extends Thread {
    private String texto;
    public TesteThread(String texto) {
        this.texto = texto;
    }

    public void run() {
        for(int i=0;i<10;i++) {
            System.out.println(texto);
            try {
                Thread.sleep((int)(1000*Math.random()));
            } catch (InterruptedException ex) { }
        }
    }
}
```

4. Adicionar código de teste na classe Main

```
        TesteThread tt1 = new TesteThread("Ola ");
        TesteThread tt2 = new TesteThread("pessoal ");
        TesteThread tt3 = new TesteThread("tudo ");
        TesteThread tt4 = new TesteThread("bem? ");
        tt1.start();
        tt2.start();
        tt3.start();
        tt4.start();
```

5. Rodar
6. Trocar start() por run() e rodar

Demonstração 2
Runnable

1. Usar o mesmo projeto NetBeans da demonstração 1
2. Criar uma classe Tarefa

```
public class Tarefa {
    public double calcularImposto() {
        return 100*Math.random();
    }
}
```

3. Criar uma classe Pagamento

```
public class Pagamento extends Tarefa implements Runnable {

    public void run() {
```

```
        for(int i=0;i<100;i++) {
            System.out.println("Processando pagamento "+i);
            System.out.println("Imposto="+calcularImposto());
            try {
                Thread.sleep(500);
            } catch (InterruptedException ex) {
            }
        }
    }
}
```

4. Criar uma nova classe Main2, com o seguinte código de teste

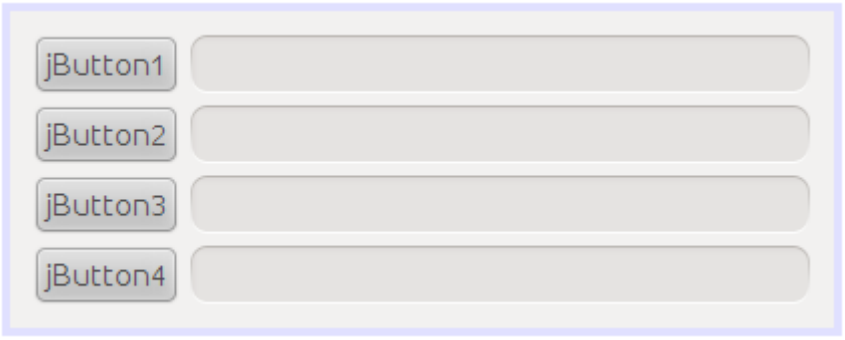
```
        Pagamento p1 = new Pagamento();
        Pagamento p2 = new Pagamento();
        Pagamento p3 = new Pagamento();
        Pagamento p4 = new Pagamento();
        Thread t1 = new Thread(p1);         Thread t2 = new Thread(p2);
        Thread t3 = new Thread(p3);         Thread t4 = new Thread(p4);
        t1.start();         t2.start();
        t3.start();         t4.start();
```

5. Testar

Demonstração 3
Prática com Threads

---

1. Criar um novo projeto Java no NetBeans
2. Criar um novo JFrame, chamado TelaPrincipal



3. Criar uma nova classe, chamada Tarefa

```
public class Tarefa {
    private JProgressBar barra;
    private int velocidade;

    // Velocidade varia de 1 a 4
    public Tarefa(JProgressBar barra, int velocidade) {
        this.barra = barra;
        this.velocidade = velocidade;
    }

    public void atualiza() {
        barra.setValue(0);
        for(int i=0;i<100;i++) {
            barra.setValue(barra.getValue()+1);
```

```
            try {
                Thread.sleep(500 - velocidade * 100);
            } catch (InterruptedException ex) {
                Logger.getLogger(Tarefa.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
    }
}
```

4. Adicionar eventos aos botões:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar1, 1);
        tar.atualiza();
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar2, 2);
        tar.atualiza();
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar3, 3);
        tar.atualiza();
    }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar4, 4);
        tar.atualiza();
    }
```

5. Adicionar o seguinte código ao main

new TelaPrincipal().setVisible(true);

6. Testar, e mostrar que não está funcionando (apertar o botão 4)
7. Modificar a classe Tarefa para representar uma nova Thread

```
public class Tarefa extends Thread {
    private JProgressBar barra;
    private int velocidade;

    @Override
    public void run() {
        atualiza();
    }
   ...
}
```

8. Modificar na classe TelaPrincipal, para iniciar as threads

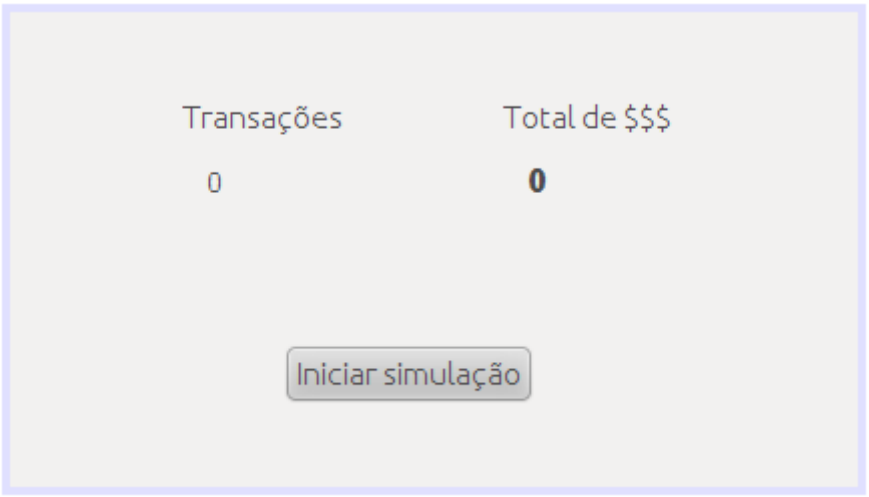
```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar1, 1);
        tar.start();
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar2, 2);
        tar.start();
```

```
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar3, 3);
        tar.start();
    }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
        Tarefa tar = new Tarefa(jProgressBar4, 4);
        tar.start();
    }
```

9. Testar novamente

Demonstração 4
Sincronização

1. Criar um novo projeto Java no NetBeans
2. Criar um novo JFrame chamado TelaPrincipal



3. Criar uma classe Conta

```
public class Conta {
    public int saldo;

    public Conta(int saldo) {
        this.saldo = saldo;
    }
}
```

4. Criar uma classe Banco

```
public class Banco {

    private Conta contas[];
    private int saldoTotal;
    private int numTransacoes;

    public Banco(int numeroContas, int saldoInicial) {
        this.contas = new Conta[numeroContas];
```

```
        for (int i = 0; i < numeroContas; i++) {
            Conta c = new Conta(saldoInicial);
            contas[i] = c;
        }
        saldoTotal = numeroContas * saldoInicial;
        numTransacoes = 0;
    }

    public int getNumeroContas() {
        return contas.length;
    }

    public void transferir(int contaSaque, int contaDeposito, int valor) {
        if(contas[contaSaque].saldo < valor) return;
        contas[contaSaque].saldo -= valor;
        contas[contaDeposito].saldo += valor;
        numTransacoes ++;
        if(numTransacoes % 10000 == 0) atualizaSaldoTotal();
    }

    public int getSaldoTotal() {
        return saldoTotal;
    }

    public int getNumTransacoes() {
        return numTransacoes;
    }

    public void atualizaSaldoTotal() {
        int novoSaldo = 0;
        for(int i=0; i<contas.length; i++)
            novoSaldo += contas[i].saldo;
        saldoTotal = novoSaldo;
    }
}
```

5. Criar uma nova classe ThreadTransferencias

```
public class ThreadTransferencias extends Thread {

    private Banco banco;
    private int contaSaque;
    private int valor = 0;

    public ThreadTransferencias(Banco b, int contaSaque, int valor) {
        this.banco = b;
        this.contaSaque = contaSaque;
        this.valor = valor;
    }

    public void run() {
        while (!interrupted()) {
            int contaDeposito = (int) (banco.getNumeroContas() *
Math.random());
            int valorTransf = (int) (this.valor * Math.random());
            this.banco.transferir(this.contaSaque, contaDeposito,
valorTransf);
        }
    }
}
```

6. Criar uma nova classe ThreadImprimeSaldo

```
public class ThreadImprimeSaldo extends Thread {
    private Banco banco;
    private JLabel labelSaldo, labelTransacoes;

    public ThreadImprimeSaldo(Banco banco, JLabel labelSaldo, JLabel
labelTransacoes) {
        this.banco = banco;
        this.labelSaldo = labelSaldo;
        this.labelTransacoes = labelTransacoes;
    }

    public void run() {
        while(!interrupted()) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
            }
            labelSaldo.setText(Integer.toString(banco.getSaldoTotal()));

labelTransacoes.setText(Integer.toString(banco.getNumTransacoes()));
        }
    }
}
```

7. Modificar a TelaPrincipal para iniciar a simulação

```
public class TelaPrincipal extends javax.swing.JFrame {

    private void iniciarSimulacao() {
        int saldoInicial = 1000;
        int numeroContas = 10;
        Banco banco = new Banco(numeroContas, saldoInicial);

        ThreadImprimeSaldo tis = new ThreadImprimeSaldo(banco,
jLabelSaldoTotal, jLabelTransacoes);
        tis.start();

        for(int j=0;j<numeroContas;j++) {
            ThreadTransferencias tt = new ThreadTransferencias(banco, j,
saldoInicial);
            tt.start();
        }
    }

   ...

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        iniciarSimulacao();
    }
    ...
}
```

8. Adicionar o seguinte código ao main()

```
    new TelaPrincipal().setVisible(true);
```

9. Testar e mostrar que há inconsistência no saldo

Demonstração 5
Corrigindo o problema da sincronização

1. Abrir o projeto da demonstração anterior
2. Adicionar synchronized no método de transferência da classe Banco
3. Mostrar que agora não tem mais inconsistência no saldo total

Demonstração 6
Guarded blocks

1. Criar um novo projeto Java no NetBeans
2. Criar uma classe Caixa

```
public class Caixa {
    private String item;
    private boolean vazia = true;

    public synchronized String pegar() {
        while (vazia) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        vazia = true;
        notifyAll();
        return item;
    }

    public synchronized void colocar(String item) {
        while (!vazia) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        vazia = false;
        this.item = item;
        notifyAll();
    }
}
```

3. Criar uma classe Colocador

```
public class Colocador implements Runnable {
    private Caixa caixa;

    public Colocador(Caixa caixa) {
        this.caixa = caixa;
    }

    public void run() {
        String coisas[] = {
            "Banana",
            "Martelo",
            "Batata",
            "Brinquedo"
```

```
        };
        Random random = new Random();

        for (int i = 0; i < coisas.length; i++) {
            caixa.colocar(coisas[i]);
            System.out.println("Item colocado: "+coisas[i]);
            try {
                Thread.sleep(random.nextInt(5000));
            } catch (InterruptedException e) {}
        }
        caixa.colocar("Livro");
    }
}
```

4. Criar uma classe Pegador

```
public class Pegador implements Runnable {

    private Caixa caixa;

    public Pegador(Caixa caixa) {
        this.caixa = caixa;
    }

    public void run() {
        Random random = new Random();
        for (String item = caixa.pegar(); !item.equals("Livro");
                item = caixa.pegar()) {
            System.out.println("Item pego: " + item);
            try {
                Thread.sleep(random.nextInt(5000));
            } catch (InterruptedException e) {
            }
        }
    }
}
```

5. Adicionar o seguinte código ao main

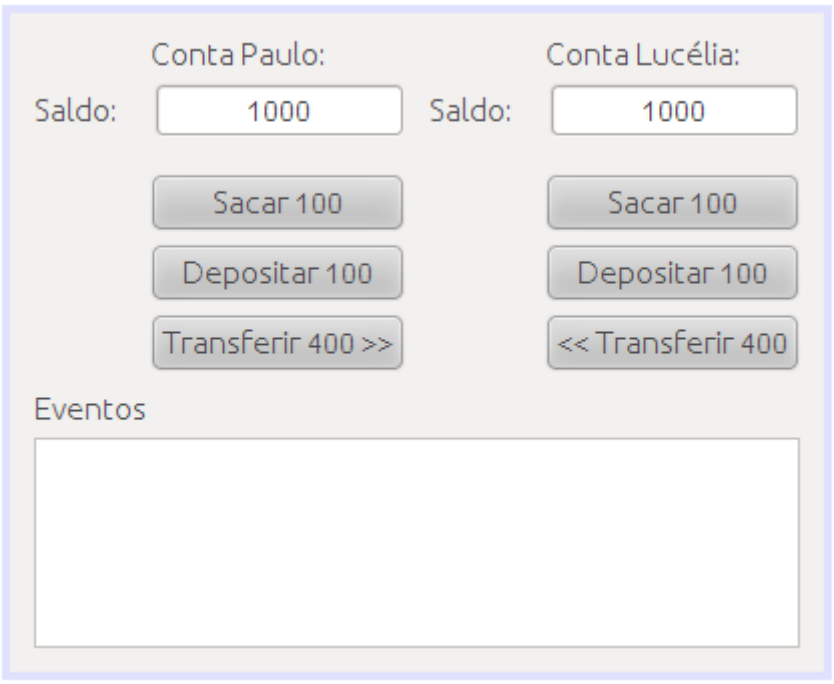
```
        Caixa caixa = new Caixa();
        (new Thread(new Colocador(caixa))).start();
        (new Thread(new Pegador(caixa))).start();
```

6. Rodar

Demonstração 7
Concorrência

---

1. Criar novo projeto Java no NetBeans
2. Criar um novo JFrame, chamado TelaPrincipal



3. Criar classe Conta

```
public class Conta {

    private int saldo;

    public Conta(int saldo) {
        this.saldo = saldo;
    }

    public void setSaldo(int saldo) {
        this.saldo = saldo;
    }

    public int getSaldo() {
        return saldo;
    }

    public void incrementaSaldo(int valor) {
        saldo += valor;
    }

    public void decrementaSaldo(int valor) {
        saldo -= valor;
    }
}
```

4. Criar classe Banco

```
public class Banco {
    public Conta[] contas;

    public Banco(Conta[] contas) {
        this.contas = contas;
    }
```

```
    public void sacar(int numConta, int valor) {
        simularDemora(100);
        if(contas[numConta].getSaldo() < valor) return;
        simularDemora(100);
        contas[numConta].decrementaSaldo(valor);
        simularDemora(100);
    }

    public void depositar(int numConta, int valor) {
        simularDemora(100);
        contas[numConta].incrementaSaldo(valor);
        simularDemora(100);
    }

    public void transferir(int numContaOrigem, int numContaDestino, int
valor) {
        simularDemora(300);
        int saldoOrigem = contas[numContaOrigem].getSaldo();
        simularDemora(300);
        int saldoDestino = contas[numContaDestino].getSaldo();
        simularDemora(300);
        if(saldoOrigem < valor) return;
        simularDemora(300);
        contas[numContaOrigem].setSaldo(saldoOrigem-valor);
        simularDemora(300);
        contas[numContaDestino].setSaldo(saldoDestino+valor);
        simularDemora(300);
    }

    private void simularDemora(long milisegundos) {
        try {
            Thread.sleep(milisegundos);
        } catch (InterruptedException ex) {
        }
    }
}
```

5. Modificar a classe TelaPrincipal para incluir os métodos de manipulação de conta

```
public class TelaPrincipal extends javax.swing.JFrame {

    Banco banco;
    Conta contaPaulo;
    Conta contaLucelia;

    private void inicializar() {
        contaPaulo = new Conta(1000);
        contaLucelia = new Conta(1000);

        Conta[] contas = new Conta[]{contaPaulo, contaLucelia};
        banco = new Banco(contas);
    }

    private void atualizarStatus() {
        saldoPaulo.setText(Integer.toString(contaPaulo.getSaldo()));
        saldoLucelia.setText(Integer.toString(contaLucelia.getSaldo()));
    }

    private void iniciarAcao(String acao) {
```

```
        jTextArea1.setText(getHora() + " Iniciado " + acao + "\n" +
jTextArea1.getText());
    }

    private void terminarAcao(String acao) {
        jTextArea1.setText(getHora() + " Concluído " + acao + "\n" +
jTextArea1.getText());
        atualizarStatus();
    }

    private String getHora() {
        SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss:SSS");
        return sdf.format(new Date());
    }

    private void iniciarThread(Runnable r) {
        new Thread(r).start();
    }

    // Paulo é conta 0
    private void sacarPaulo() {
        iniciarThread(new Runnable() {

            public void run() {
                iniciarAcao("Sacar Paulo");
                banco.sacar(0, 100);
                terminarAcao("Sacar Paulo");
            }
        });
    }

    // Paulo é conta 0
    private void depositarPaulo() {
        iniciarThread(new Runnable() {

            public void run() {
                iniciarAcao("Depositar Paulo");
                banco.depositar(0, 100);
                terminarAcao("Depositar Paulo");
            }
        });
    }

    // Paulo é conta 0
    // Lucelia é conta 1
    private void transferirPauloLucelia() {
        iniciarThread(new Runnable() {

            public void run() {
                iniciarAcao("Transferir de Paulo para Lucélia");
                banco.transferir(0, 1, 400);
                terminarAcao("Transferir de Paulo para Lucélia");
            }
        });
    }

    // Lucelia é conta 1
    private void sacarLucelia() {
        iniciarThread(new Runnable() {
```

```
            public void run() {
                iniciarAcao("Sacar Lucélia");
                banco.sacar(1, 100);
                terminarAcao("Sacar Lucélia");
            }
        });
    }

    // Lucelia é conta 1
    private void depositarLucelia() {
        iniciarThread(new Runnable() {

            public void run() {
                iniciarAcao("Depositar Lucélia");
                banco.depositar(1, 100);
                terminarAcao("Depositar Lucélia");
            }
        });
    }

    // Paulo é conta 0
    // Lucelia é conta 1
    private void transferirLuceliaPaulo() {
        iniciarThread(new Runnable() {

            public void run() {
                iniciarAcao("Transferir de Lucélia para Paulo");
                banco.transferir(1, 0, 400);
                terminarAcao("Transferir de Lucélia para Paulo");
            }
        });
    }

    /**
     * Creates new form TelaPrincipal
     */
    public TelaPrincipal() {
        initComponents();
        inicializar();
    }
   ...
}
```

6. Associar os eventos aos botões
7. Rodar e mostrar a inconsistência
   7.1. transferir e sacar/depositar da mesma conta, rapidamente, primeiro transferir, e depois sacar/depositar
   7.2. deixar com saldo de 100, e sacar duas vezes, rapidamente
8. Adicionar synchronized no método de transferência da classe Banco
9. Mostrar que agora não tem mais inconsistência no saldo total

Demonstração 8
Concorrência e o modelo requisição resposta

1. Criar novo projeto Java no NetBeans
2. Criar novo JFrame, chamado TelaPrincipal

3. Criar interface Navegador

```
public interface Navegador {
    public void respostaRecebida(String resposta);
}
```

4. Criar a classe ServicoWeb

```
public class ServicoWeb {

    // operacao: 1=soma, 2=subtracao, 3=multiplicacao
    private int operacao, arg1, arg2;
    private Navegador navegador;

    public ServicoWeb(int operacao, int arg1, int arg2, Navegador
navegador) {
        this.operacao = operacao;
        this.arg1 = arg1;
        this.arg2 = arg2;
        this.navegador = navegador;
    }

    public void tratarRequisicao() {
        // buffer é utilizado para montar a resposta
        // utilizando buffer.append("texto\n")
        StringBuffer buffer = new StringBuffer();

        simularDemora(5000);

        buffer.append("Esta é uma calculadora web!\n");
        buffer.append("Obrigado por solicitar meu serviço!\n\n");
        buffer.append("Você solicitou a operação de ");
        if (operacao == 1) {
            buffer.append("soma!\n");
            int result = arg1 + arg2;
            buffer.append(arg1 + " + " + arg2 + " = " + result + "\n");
        } else if (operacao == 2) {
            buffer.append("subtração!\n");
```

```
            int result = arg1 - arg2;
            buffer.append(arg1 + " - " + arg2 + " = " + result + "\n");
        } else if (operacao == 3) {
            buffer.append("multiplicação!\n");
            int result = arg1 * arg2;
            buffer.append(arg1 + " * " + arg2 + " = " + result + "\n");
        }

        buffer.append("\nVolte sempre!\n");

        navegador.respostaRecebida(buffer.toString());
    }

    private void simularDemora(long milisegundos) {
        try {
            Thread.sleep(milisegundos);
        } catch (InterruptedException ex) {
        }
    }
}
```

5. Criar a classe ServidorWeb

```
public class ServidorWeb {
    private static ServidorWeb instancia;

    public static ServidorWeb getInstancia() {
        if(instancia == null)
            instancia = new ServidorWeb();
        return instancia;
    }

    public synchronized void enviarRequisicao(int operacao, int arg1, int
arg2, Navegador navegador) {
        ServicoWeb sw = new ServicoWeb(operacao, arg1, arg2, navegador);
        sw.tratarRequisicao();
    }
}
```

6. Criar a classe Requisicao

```
public class Requisicao extends Thread {

    private int operacao, arg1, arg2;
    private Navegador navegador;

    public Requisicao(int operacao, int arg1, int arg2, Navegador
navegador) {
        this.operacao = operacao;
        this.arg1 = arg1;
        this.arg2 = arg2;
        this.navegador = navegador;
    }

    @Override
    public void run() {
        ServidorWeb.getInstancia().enviarRequisicao(operacao, arg1, arg2,
navegador);
    }
}
```

7. Modificar a classe TelaPrincipal para inicializar o servidor e incluir as requisições/respostas

```
public class TelaPrincipal extends javax.swing.JFrame implements Navegador
{
    private void enviarRequisicao() {
        jTextArea1.setText("Aguarde... processando...\n");

        int operacao = Integer.parseInt(jTextField1.getText());
        int arg1 = Integer.parseInt(jTextField2.getText());
        int arg2 = Integer.parseInt(jTextField3.getText());

        Requisicao req = new Requisicao(operacao, arg1, arg2, this);
        req.start();
    }

    public void respostaRecebida(String resposta) {
        jTextArea1.setText("Resposta recebida\n");
        jTextArea1.append("================\n");
        jTextArea1.append(resposta+"\n");
        jTextArea1.append("================\n");
        jTextArea1.append("Fim da resposta\n");
    }
    ...

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        enviarRequisicao();
    }
    ...
}
```

7. Incluir o seguinte código no void main()

```
        new TelaPrincipal().setVisible(true);
        new TelaPrincipal().setVisible(true);
```

8. Testar e mostrar que a cada requisição, é gerada uma nova resposta, mas há demora na atualização. Se chamar ao mesmo tempo nas duas telas, o servidor processa em sequência
9. Modificar o código do servidor/servicoWeb, para iniciar uma nova Thread para cada requisição recebida

```
public class ServicoWeb extends Thread {

    @Override
    public void run() {
        tratarRequisicao();
    }
    ...
}

public class ServidorWeb {
    ...

    public synchronized void enviarRequisicao(int operacao, int arg1, int
arg2, Navegador navegador) {
        ServicoWeb sw = new ServicoWeb(operacao, arg1, arg2, navegador);
        sw.start();
    }
}
```

10. Testar novamente