

# Aula 1

Na primeira aula é feita uma visão geral sobre JEE, definição, histórico, componentes e servidores.

## Java EE

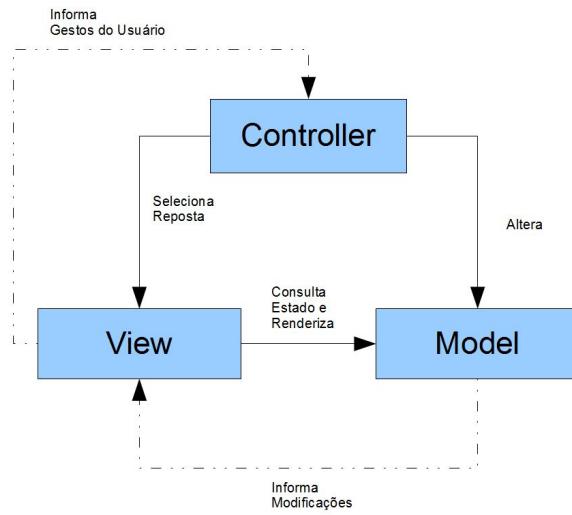
Java Enterprise Edition é um conjunto de especificações que começou a ser desenvolvido em 1998. Essas especificações podem ser executadas de forma individual sendo introduzidas ao projeto a medida que são necessárias, um programa Java EE sempre irá rodar em um *application server*. A Figura 1 mostra os servidores certificados pela Oracle totalmente compatíveis com Java EE 7.



Durante a aula 1 é estabelecido que dentre as implementações do Java EE serão utilizadas no decorrer do curso o JSF 2.2, EL 3.0 e CDI Extensions.

## JSF 2.2

JSF é um framework web baseado em componente que utiliza o padrão MVC (Model, View, Controller), Figura 2 ilustra o padrão MVC.



Entre os componentes que estão presentes no JSF a aula 1 apresenta o ICEfaces (<http://icefaces-showcase.icesoft.org/showcase.jsf>), o BootFaces (<https://showcase.bootsfaces.net/>) e o PrimeFaces (<https://www.primefaces.org/showcase/>).

## Aula 2

Aula 2 é feita a instalação e configuração das ferramentas necessárias para desenvolver as atividades do curso.

### Ferramentas Utilizadas

As ferramentas baixada durante a aula 2 foram: JDK 8, IntelliJ IDEA, Maven e WildFly. Após o download das ferramentas foi estabelecida as variaveis de ambiente JAVA HOME e M2. A Figura 3 mostra as versões de Java e Maven no terminal do Windows.

```
Microsoft Windows [versão 10.0.18362.720]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

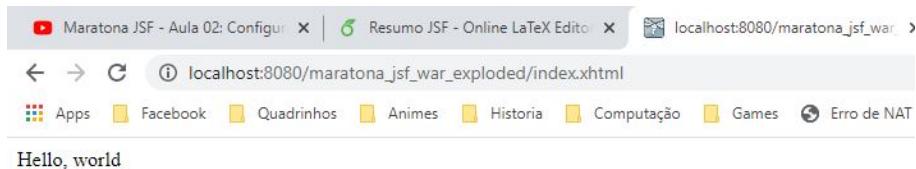
C:\Users\jaime>java -version
java version "1.8.0_241"
Java(TM) SE Runtime Environment (build 1.8.0_241-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.241-b07, mixed mode)

C:\Users\jaime>mvn -v
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\Users\jaime\Downloads\Curso JSF\apache-maven-3.6.3-bin\apache-maven-3.6.3\bin\
Java version: 1.8.0_241, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_241\jre
Default locale: pt_BR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\jaime>
```

### Hello World

Após a instalação e configuração do ambiente um pequeno projeto foi iniciado com um Hello World sendo exibido na porta 8080 ([http://localhost:8080/maratona\\_jsf\\_war\\_exploded/index.xhtml](http://localhost:8080/maratona_jsf_war_exploded/index.xhtml)). A Figura 4 mostra o Hello World.



## Aula 3

A aula 3 explica os contextos de uma aplicação web e como configurá-lo também foi apresentado o hot deploy para atualizar uma pagina web sem precisar reiniciar o servidor.

### Corrigindo Erro da Aula 2

Após iniciar o projeto na aula o link `http://localhost:8080/maratona_jsf_war_exploded/` informava a mensagem "Forbbiden" ao invés de "Hello World", para fazer funcionar foi necessário modificar a URL para `http://localhost:8080/maratona_jsf_war_exploded/index.xhtml`. Na aula 3 esse problema é corrigido ao passar a informação no arquivo web.xml. Figura 1 mostra a modificação feita no arquivo web.xml.

```
<welcome-file-list>
    <welcome-file>index.xhtml</welcome-file>
</welcome-file-list>
```

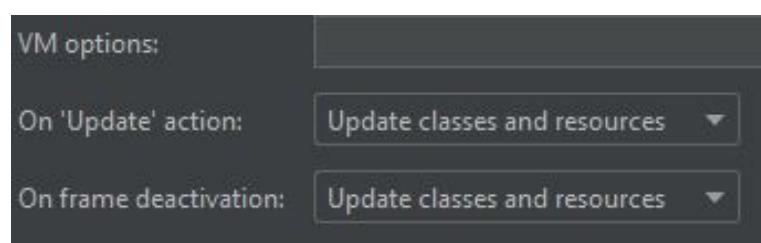
### Adição de Dependências

Na aula 3 ocorreu a adição da primeira dependência, através de uma adição no arquivo pom.xml. Figura 2 mostra a dependência.

```
<dependencies>
    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-api</artifactId>
        <version>2.2.13</version>
    </dependency>
</dependencies>
```

### Hot Deploy

Configurando o wildfly para atualizar sem precisar reiniciar o servidor é possível alterar a mesagem. Figura 3 mostra a configuração pra fazer o hot deploy funcionar e a Figura 4 a nova mensagem exibida.





## Aula 4

A aula 4 inicia a parte de Expression Language, seus conceitos e usos.

### Criação de Pacote e Classe

Foi criado o pacote bean.estudante e a classe EstudanteRegistrarBean. Figura 5 mostra o código da classe criada.

```
package bean.estudante;

import javax.inject.Named;
import java.io.Serializable;

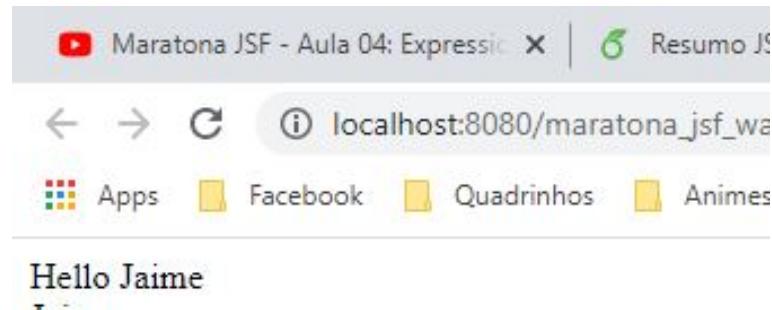
@Named
public class EstudanteRegistrarBean implements Serializable {
    private String nome = "Jaime";
    private String sobrenome = "Eduardo";
    private double nota1;
    private double nota2;
    private double nota3 = 10.0;

    public String getName() {
```

### Expression Language

Utilizando expression language foi possível exibir o nome da classe na porta 8080. Figura 6 mostra a modificação no arquivo utilizando expression language e a Figura 7 mostra o nome sendo exibido no [http://localhost:8080/maratona\\_jsf\\_war\\_exploded/](http://localhost:8080/maratona_jsf_war_exploded/).

```
<f:view>
    <h:outputLabel value="Hello Jaime"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.nome}"/><br/>
</f:view>
```



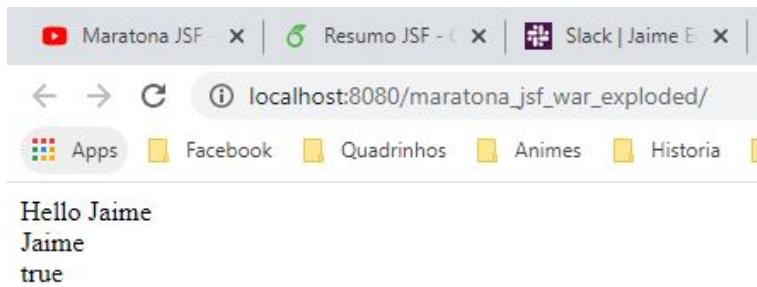
# Aula 5

A aula 5 consiste de realizar operações lógicas através de expression language.

## Operações Logicas

Durante a aula 5 foram feitos testes utilizando operadores lógicos para exibir mensagens na tela sem reiniciar o servidor. O primeiro exemplo foi o uso do operador 'eq' para verificar se as notas da classe 'EstudanteRegistrarBean' eram iguais. Figura 1 exibi o código enquanto a Figura 2 exibi mensagem exibida.

```
<h:outputLabel value="Hello Jaime"/><br/>
<h:outputLabel value="#{estudanteRegistrarBean.nome}"/><br/>
<h:outputLabel value="#{estudanteRegistrarBean.nota1 eq estudanteRegistrarBean.nota2}"/>
```



## Palavras Reservadas

Ao final da aula foi apresentado um conjunto de palavras reservadas que podem ser utilizadas no JSF.

- and
- or
- not
- eq
- ne
- lt
- le
- gt
- ge
- true
- false
- null
- instanceof
- empty
- div
- mod

## Aula 6

A aula 6 introduziu o uso de enums para adicionar mais uma informação a classe 'Estudante'

### Reorganização dos Pacotes

Antes da aula começar os pacotes foram alterados e a classe 'EstudanteRegistrarBean' passou a receber os dados de uma nova classe 'Estudante'. Figura 3 mostra a classe 'Estudante' e Figura 4 mostra a classe 'EstudanteRegistrarBean'.

```
package br.com.maratonajsf.model;

import br.com.maratonajsf.model.enums.Turno;

public class Estudante {
    private String nome = "Jaime";
    private String sobrenome = "Eduardo";
    private double nota1;
    private double nota2;
    private double nota3 = 10;
    private Turno turno = Turno.MATUTINO;
```

```
import br.com.maratonajsf.model.Estudante;

import javax.inject.Named;
import java.io.Serializable;

@Named
public class EstudanteRegistrarBean implements Serializable {

    Estudante estudante = new Estudante();
```

### Enum

Foi criado um tipo enum chamado 'Turno' com os valores possíveis sendo, MATUTINO, VESPERTINO, NOTURNO. Figura 5 mostra o código de 'Turno'.

```
package br.com.maratonajsf.model.enums;

public enum Turno {
    MATUTINO, VESPERTINO, NOTURNO
}
```

### PrimeFaces

Para poder fazer uma operação de comparação entre dois valores de 'Turno' foi importado um pacote do PrimeFaces. Figura 6 mostra o código importando o PrimeFaces e a Figura 7 mostra o resultado da comparação na porta 8080.

```
>
<p:importEnum type="br.com.maratonajsf.model.enums.Turno" var="Turno" allSuffix="ALL" />
<f:view>
    <h:outputLabel value="Hello Jaime"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.nome}"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.notal eq estudanteRegistrarBean.estudante.notal ? 'true' : 'false'}"/><br/>
    <h:outputText value="Comparacao"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.notal le estudanteRegistrarBean.estudante.notal ? 'true' : 'false'}"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.notal eq 0 ? 'ZERO' : 'NOTZERO'}"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.notal = 20}"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.notal3}"/><br/>
    <h:outputText value="ENUM"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.turno}"/><br/>
    <h:outputLabel value="#{estudanteRegistrarBean.estudante.turno eq Turno.MATUTINO ? 'true' : 'false'}"/>
```

localhost:8080/r

Hello Jaime  
Jaime  
true  
Comparacao  
true  
ZERO  
20  
10.0  
ENUM  
MATUTINO  
true

# Aula 7

Na aula 7 foram utilizadas diversas formas de coleções.

## Coleções

Ao longo da aula se utilizou array, list, set e map como formas de coleção todas elas criadas na classe 'EstudanteRegistrarBean'. Figura 1 mostra as alterações no código da classe.

```
@Named  
public class EstudanteRegistrarBean implements Serializable {  
  
    private Estudante estudante = new Estudante();  
    private String[] nomesArray = {"DevDojo", "Curso", "JSF"};  
    private List<String> nomesLista = asList("Jaime", "Eduardo", "Padilla");  
    private Set<String> nomesSet = new HashSet<>(asList("Java", "Cpp", "Python"));  
    private Map<String, String> nomesMap = new HashMap<>();  
  
    {  
        nomesMap.put("Calculo", "Departamento de Matemática");  
        nomesMap.put("PDI", "Departamento de Informática");  
        nomesMap.put("Filosofia", "Departamento de Filosofia");  
    }  
}
```

Em seguida o index.xhtml também sofreu alterações com o objetivo de exibir as coleções criadas. Figura exibi as adições ao código e a Figura 3 exibi o resultado na porta 8080.

```
<h:outputText value="ARRAY"/><br/>  
<h:outputLabel value="#{estudanteRegistrarBean.nomesArray[0]}"/>  
<h:outputLabel value="#{estudanteRegistrarBean.nomesArray[1]}"/>  
<h:outputLabel value="#{estudanteRegistrarBean.nomesArray[2]}"/><br/>  
<ui:repeat value="#{estudanteRegistrarBean.nomesArray}" var="nome">  
    <h:outputLabel value="#{nome}"/><br/>  
</ui:repeat>  
<h:outputText value="LISTA"/><br/>  
<h:outputLabel value="#{estudanteRegistrarBean.nomesLista.get(0)} #{estudanteRegistrarBean.nomesLista.get(1)}"/><br/>  
<ui:repeat value="#{estudanteRegistrarBean.nomesLista}" var="nome">  
    <h:outputLabel value="#{nome}"/><br/>  
</ui:repeat>  
<h:outputText value="SET"/><br/>  
<ui:repeat value="#{estudanteRegistrarBean.nomesSet}" var="nome">  
    <h:outputLabel value="#{nome}"/><br/>  
</ui:repeat>  
<h:outputText value="MAP"/><br/>  
<h:outputLabel value="#{estudanteRegistrarBean.nomesMap.get('Filosofia')}"/><br/>  
<h:outputLabel value="#{estudanteRegistrarBean.nomesMap}"/><br/>  
<ui:repeat value="#{estudanteRegistrarBean.nomesMap}" var="entry">  
    <h:outputLabel value="#{entry}"/><br/>  
</ui:repeat>
```

```
ARRAY  
DevDojoCursoJSF  
DevDojo  
Curso  
JSF  
LISTA  
Jaime  
Eduardo  
Padilla  
SET  
Java  
Cpp  
Python  
MAP  
Departamento de Filosofia  
{PDI}=Departamento de Informática, Filosofia=Departamento de Filosofia, Calculo=Departamento de Matematica}  
PDI=Departamento de Informática  
Filosofia=Departamento de Filosofia  
Calculo=Departamento de Matematica
```

# Aula 8

A aula trabalha com o uso de métodos através de expression language.

## Metodos

Ao longo da aula 8 foram criados 4 métodos na classe 'EstudanteRegistrarBean'. O primeiro métodos exibi três linhas de texto no terminal, o segundo executa a mesma ação com a adição de um parametro, o terceiro método retorna uma string na porta 8080 e o ultimo método retorna uma string index2 que será usado para ir para uma nova página. A Figura 4 mostra as adições ao código e a Figura 5 e 6 mostram os resultados no terminal.

```
public void executar(){
    System.out.println("Fazendo busca no BD");
    System.out.println("Processando os dados");
    System.out.println("Exibindo os dados");
}

public void executar(String param){
    System.out.println("Fazendo busca no BD com o parametro: " +param);
    System.out.println("Processando os dados");
    System.out.println("Exibindo os dados");
}

public String executarRetorno(String param){
    return "Curso online sobre: " +param;
}

public String irParaIndex2(){
    return "index2";
}
```

```
10:59:09,372 INFO [stdout] (default task-1) Fazendo busca no BD
10:59:09,373 INFO [stdout] (default task-1) Processando os dados
10:59:09,373 INFO [stdout] (default task-1) Exibindo os dados
```

```
11:11:12,400 INFO [stdout] (default task-1) Fazendo busca no BD com o parametro: Jaime
11:11:12,400 INFO [stdout] (default task-1) Processando os dados
11:11:12,401 INFO [stdout] (default task-1) Exibindo os dados
```

No arquivo 'index.xhtml' foi criado um button que recebe a string index2 e abre uma nova página mostrando o conteúdo do arquivo 'index2.xhtml'. Figura 7 mostra as adições ao index e figura 8 e 9 mostram os resultados na porta 8080.

```
#{estudanteRegistrarBean.executar()}
#{estudanteRegistrarBean.executar(estudanteRegistrarBean.estudante.nome)}
#{estudanteRegistrarBean.executarRetorno('JSF')}<br/>
<h:form>
    <h:commandButton value="Ir para proxima Pagina"
                      action="#{estudanteRegistrarBean.irParaIndex2()}" />
</h:form>
```

Curso online sobre: JSF  
Ir para proxima Pagina



## Aula 9

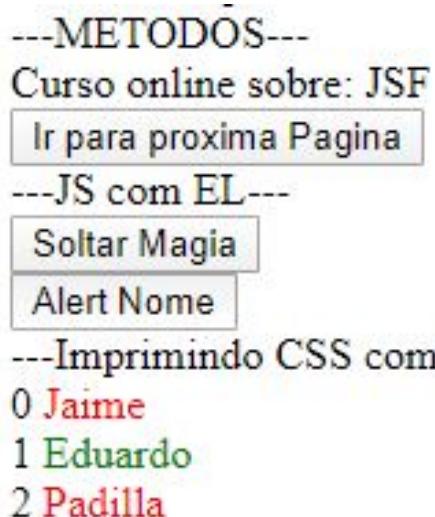
Na aula 9 foi utilizado JS e CSS através de expression language.

### CSS

O CSS foi utilizado para exibir os nomes presentes em lista em cores diferentes. Figura 1 mostra o código tanto pro CSS quanto pras funções em javaScript falados na proxima seção e a Figura 2 mostra o resultado exibido na porta 8080.

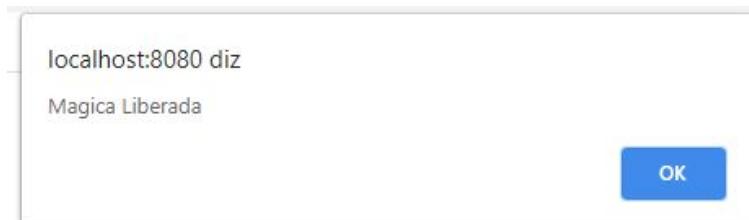
```
<h:outputText value="---JS com EL---"/><br/>
<h:commandButton value="Soltar Magia" onclick="fazMagica('Magica Liberada')"/><br/>
<h:commandButton value="Alert Nome" onclick="alertNome('#{estudanteRegistrarBean.estudante.nome}')"/>
</h:form>
<h:outputText value="---Imprimindo CSS com EL---"/><br/>
<ui:repeat value="#{estudanteRegistrarBean.nomesLista}" var="nome" varStatus="status">
    #{status.index}
    <h:outputLabel value="#{nome}" style="color: #{status.index mod 2 == 0 ? 'red' : 'green'}"/><br/>
</ui:repeat>
</f:view>
</h:body>
<script type="text/javascript">
    function fazMagica(magica){
        alert(magica);
    }

    function alertNome(nome){
        alert(nome);
    }
</script>
```



### JS

Foram criadas duas funções em javaScript para emitir um alerta, primeiro exibi uma string e o segundo exibi o nome contido na classe. Figura 3 mostra a janela com a string e a Figura 4 mostra a janela com o nome da classe 'estudante'.



localhost:8080 diz

Jaime

OK

## Aula 10

Na aula 10 foi utilizado o rendered e o ajax para exibir e esconder informação apertando um botão.

### Mostrar Notas e Link

Foram criados 4 botões, os dois primeiros tinham o objetivo de exibir ou esconder as notas da classe 'estudante' para isso funcionar os botões ativam uma variável booleana criada na classe 'estudanteRegistrarBean'. Os outros dois botões exibem ou escondem um link que abre uma nova aba para o site do google. Figura 5 mostra o código da classe, Figura 6 mostra o código no arquivo 'index.xhtml', e a Figura 7 mostra o resultado na porta 8080.

---JS com EL---

Soltar Magia  
Alert Nome

---Imprimindo CSS com EL---

0 Jaime  
1 Eduardo  
2 Padilla

Exibir Notas  
Esconder Notas

Jaime  
Eduardo  
MATUTINO

Exibir Link  
Esconder Link

[www.google.com.br](http://www.google.com.br)

```
public void exibirNotas() { this.mostrarNotas = true; }

public void esconderNotas() { this.mostrarNotas = false; }

public void exibirLink() { this.mostrarLink = true; }

public void esconderLink() { this.mostrarLink = false; }
```

```
<h:commandButton value="Exibir Notas">
    <f:ajax render="notasGrid" listener="#{estudanteRegistrarBean.exibirNotas()}" /><br/>
</h:commandButton>
<h:commandButton value="Esconder Notas">
    <f:ajax render="notasGrid" listener="#{estudanteRegistrarBean.esconderNotas()}" /><br/>
</h:commandButton>
<h:panelGrid id="notasGrid">
    <h:outputText value="#{estudanteRegistrarBean.estudante.nome}" />
    <h:outputText value="#{estudanteRegistrarBean.estudante.sobrenome}" />
    <h:outputText value="#{estudanteRegistrarBean.estudante.turno}" />
    <h:outputText value="#{estudanteRegistrarBean.estudante.nota1}" rendered="#{estudanteRegistrarBean.mostrarNotas1}" />
    <h:outputText value="#{estudanteRegistrarBean.estudante.nota2}" rendered="#{estudanteRegistrarBean.mostrarNotas2}" />
    <h:outputText value="#{estudanteRegistrarBean.estudante.nota3}" rendered="#{estudanteRegistrarBean.mostrarNotas3}" />
</h:panelGrid><br/>
<h:commandButton value="Exibir Link">
    <f:ajax render="pgLink" listener="#{estudanteRegistrarBean.exibirLink()}" /><br/>
</h:commandButton>
<h:commandButton value="Esconder Link">
    <f:ajax render="pgLink" listener="#{estudanteRegistrarBean.esconderLink()}" /><br/>
</h:commandButton>
<h:panelGroup id="pgLink">
    <ui:fragment id="fragmentLink" rendered="#{estudanteRegistrarBean.mostrarLink}">
        <a href="http://www.google.com.br" target="_blank">www.google.com.br</a>
    </ui:fragment>
```

# Aula 11

Na aula 11 foi utilizado lambda expressions e streams, essa é a ultima aula de expression language do curso.

## Lambda

Foi utilizado expressões lambda para realizar operações matemáticas e exibi-las na porta 8080 assim como criar alguns tipos de coleções. A Figura 1 mostra o código e a Figura 2 mostram o resultado na tela.

```
<h:outputText value="---Lambda---"/><br/>
#{(soma -> soma + num) (10)}<br/>
#{((v1,v2,v3) -> v1*v2*v3) (1,5,10)}<br/>
#{{(x -> x*x*x) (2)}<br/>
#{cubo=(x -> x*x*x); cubo(10)}
<h:form>
    <h:commandButton value="Calcular Cubo" actionListener="#{estudanteRegistrarBean.calcularCubo((x->x*x*x),10)}"/>
</h:form>
<h:outputText value="---Listas---"/><br/>
<ui:repeat value="#[['Lista', 'de', 'teste']]" var="listaTeste">
    #{listaTeste}
</ui:repeat>
<br/>
<h:outputText value="---Set---"/><br/>
<ui:repeat value="#[['Lista', 'de', 'teste', 'teste']]" var="setTeste">
    #{setTeste}
</ui:repeat>
<br/>
<h:outputText value="---Map---"/><br/>
<ui:repeat value="#[['Curso' : 'JSF', 'Aula' : 'Decima Primeira']]" var="mapTeste">
    #{mapTeste}
</ui:repeat>
```

---Lambda---  
10  
50  
8  
1000  
**Calcular Cubo**  
---Listas---  
Lista de teste  
---Set---  
de teste Lista  
---Map---  
Aula=Decima Primeira Curso=JSF

## Stream

Com o uso de stream foi possível filtrar os números exibidos de uma lista através de um filtro, no exemplo da Figura 3 só exibido os números acima de 5. Também se usa stream para obter a média dos números da lista assim como a soma de todos os valores. Figura 4 mostra o código.

6 7 8 9 10 11 12  
6 7 8 9  
5.0  
45  
30

```
<ui:repeat value="#{[1,2,3,4,5,6,7,8,9,10,11,12].stream().filter(v-> v>5).toList()}" var="resultado">  
| #{resultado}  
<br/>  
<ui:repeat value="#{valuesList.stream().filter(v-> v>5).toList()}" var="resultado">  
| #{resultado}  
</ui:repeat>  
<br/>  
#{valuesList.stream().average().get()}<br/>  
#{valuesList.stream().sum()}<br/>  
#{valuesList.stream().filter(v-> v>5).sum()}<br/>
```

## Aula 12

Na aula 12 se iniciou o trabalho com diferentes tipos de scopes, nessa aula em específico se trabalhou com o RequestScope.

### Nova Classe

Para testar o RequestScope foi criada uma nova classe chamada de 'TesteRequestBean', a Figura 5 mostra o código.

```
@Named  
@RequestScoped  
public class TesteRequestBean implements Serializable {  
    private List<String> personagens = asList("Bruce Wayne", "Peter Parker", "Diana Prince");  
    private List<String> personagemSelecionado = new ArrayList<>();  
  
    public void selecionarPersonagem(){  
        int index = ThreadLocalRandom.current().nextInt( bound: 3);  
        String personagem = personagens.get(index);  
        personagemSelecionado.add(personagem);  
    }  
  
    public List<String> getPersonagens() {  
        return personagens;  
    }  
}
```

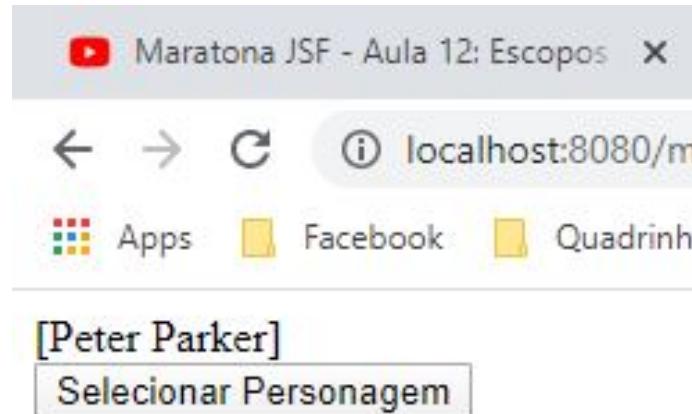
### Nova Página

Foi criada uma nova página chamada 'request.xhtml' para exibir os resultados dos testes com RequestScope. Figura 6 mostra o código desse novo arquivo.

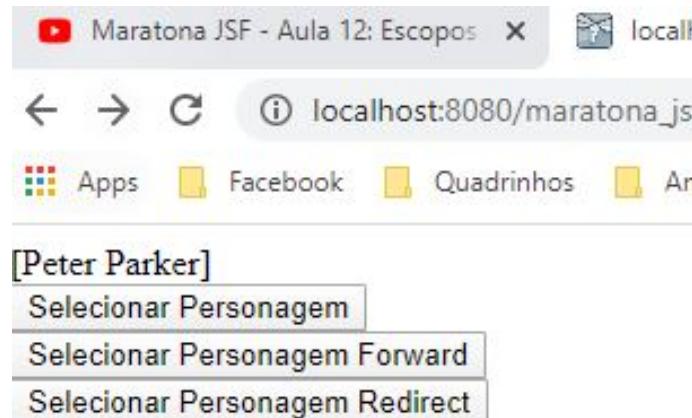
```
<h:head>  
    <h:body>  
        <h:form>  
            <h:outputText value="#{testeRequestBean.personagemSelecionado}" /><br/>  
            <h:commandButton actionListener="#{testeRequestBean.selecionarPersonagem()}" value="Selecionar Personagem" /><br/>  
            <h:commandButton actionListener="#{testeRequestBean.selecionarPersonagem()}" value="Selecionar Personagem Forward" />  
            <h:commandButton actionListener="#{testeRequestBean.selecionarPersonagem()}" value="Selecionar Personagem Redirec  
        </h:form>  
    </h:body>  
</h:head>
```

### Resultados

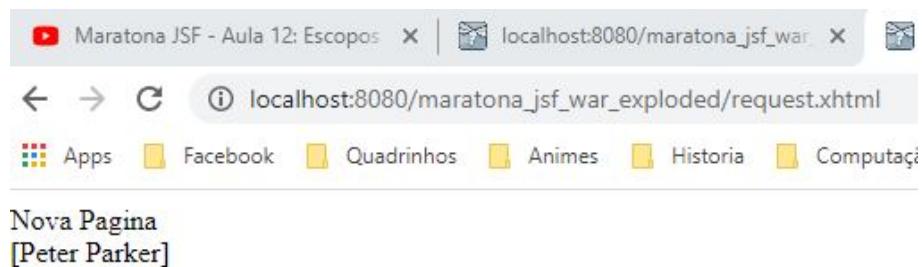
Na classe de teste existe uma lista de personagens e uma lista que recebe um dos personagens aleatoriamente através de um método. A Figura 7 mostra o resultado na porta 8080.



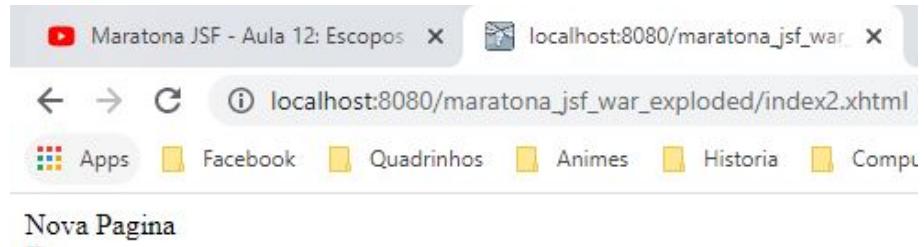
Em seguida foi criados mais dois botões, um para forward e outro para redirect. Figura 8 mostra o resultado.



Ao clicar no botão forward é carregada a página 'index2' com o nome do personagem selecionado aleatoriamente, mas no link da página ainda consta como 'request', isso ocorre porque a requisição da página é feita internamente. Figura 9 mostra o resultado.



Ao clicar no botão redirect a página é carregada com o link correto, mas o nome do personagem é perdido na segunda requisição. A Figura 10 mostra o resultado.

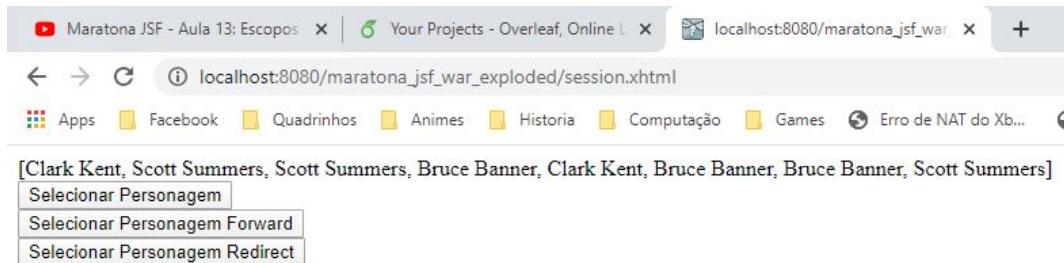


# Aula 13

A aula 13 trabalha com o uso do SessionScope.

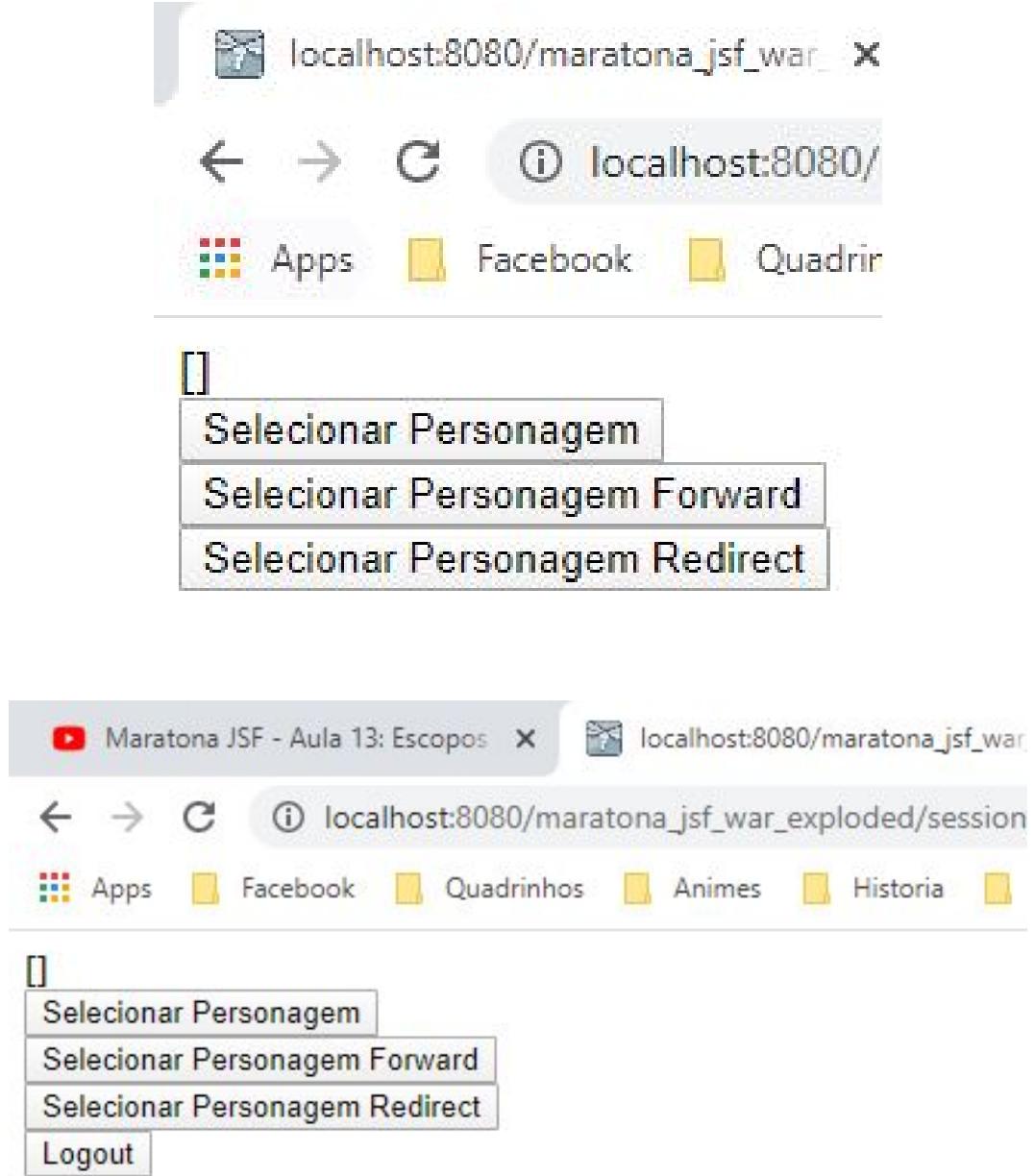
## SessionScope

O session scope mantém os dados enquanto o navegador estiver aberto. Para fazer o teste foi criado um arquivo 'TesteScopeBean' que assim como os outros arquivos recebem uma lista de personagens. A Figura 1 mostra os resultados na porta 8080.



Estando no SessionScope mesmo recarregando a página ou mudando de aba os dados não são perdidos. As Figuras 2 e 3 mostram que mesmo usando o forward e o redirect os dados são preservados.

Para perder os dados é preciso fechar o navegador, criar uma função de logout ou estabelecer nas configurações um tempo limite. Figura 4 mostra os dados resetados após o encerramento do navegador e a Figura 5 mostra um botão com a função de logout.



## PostConstruct

Foi adicionado ao código do 'TesteSessionBean' que inicia o conteúdo da lista, com ele da pra comparar o SessionScope com o RequestScope. Figura 6 mostra o resultado no log.

```
09:52:34,373 INFO [stdout] (default task-1) Entrou no PostConstruct do Session
09:53:02,613 INFO [stdout] (default task-1) Entrou no PostConstruct do Request
09:53:04,414 INFO [stdout] (default task-1) Entrou no PostConstruct do Request
09:53:05,198 INFO [stdout] (default task-1) Entrou no PostConstruct do Request
09:53:05,949 INFO [stdout] (default task-1) Entrou no PostConstruct do Request
09:53:06,646 INFO [stdout] (default task-1) Entrou no PostConstruct do Request
```

Na imagem acima é possível ver que enquanto o session scope só inicia os dados uma vez o request scope os inicia toda vez que a página é recarregada.

## Logout

A função de logout foi criada pra apagar os dados do session scope. A Figura 7 mostra o código dessa função.

```
@PostConstruct
public void init(){
    System.out.println("Entrou no PostConstruct do Session");
    personagens = asList("Clark Kent", "Scott Summers", "Bruce Banner");
}

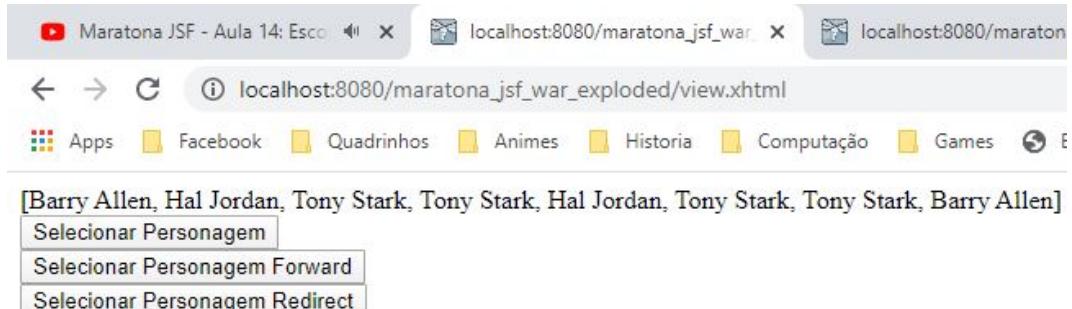
public String logout(){
    FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
    return "session?faces-redirect=true";
}
```

## Aula 14

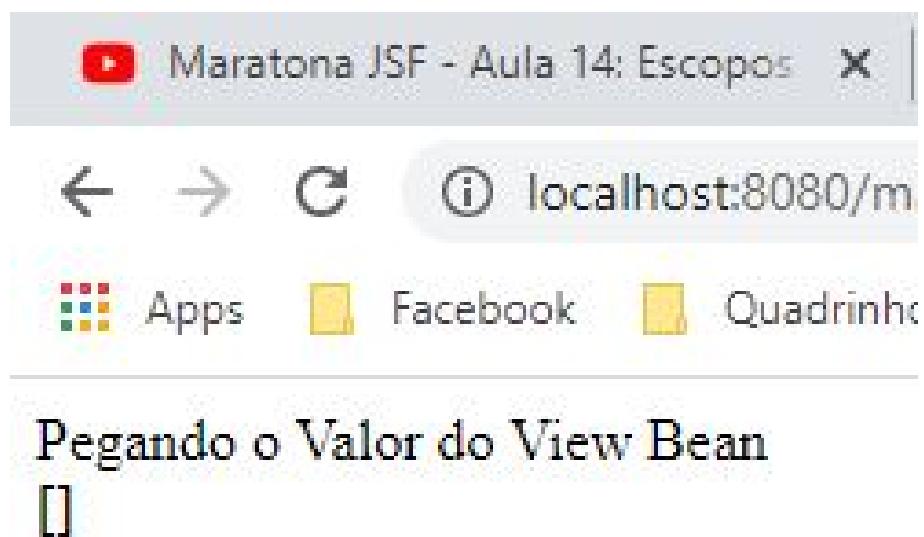
A aula 14 trabalha com o uso do ViewScope.

### ViewScope

O view scope mantém os dados enquanto a aba estiver aberta. Para realizar os testes com o view scope foi criado um arquivo 'TesteViewBean' que possui uma lista de personagens. A Figura 8 mostra o resultado na porta 8080.



Toda vez que uma nova aba é aberta os dados do view scope são perdidos. A Figura 9 mostra que quando o botão forward é pressionado os dados não estão presentes.



## Aula 15

Na aula 15 é utilizado o Application Scope.

### Application Scope

O application scope está ativo enquanto a aplicação está ativa, portanto ele pode ser acessado a qualquer momento. A Figura 1 mostra o código do 'TesteApplicationBean'.

```
@ApplicationScoped
public class TesteApplicationBean implements Serializable {
    private List<String> categoriaList;

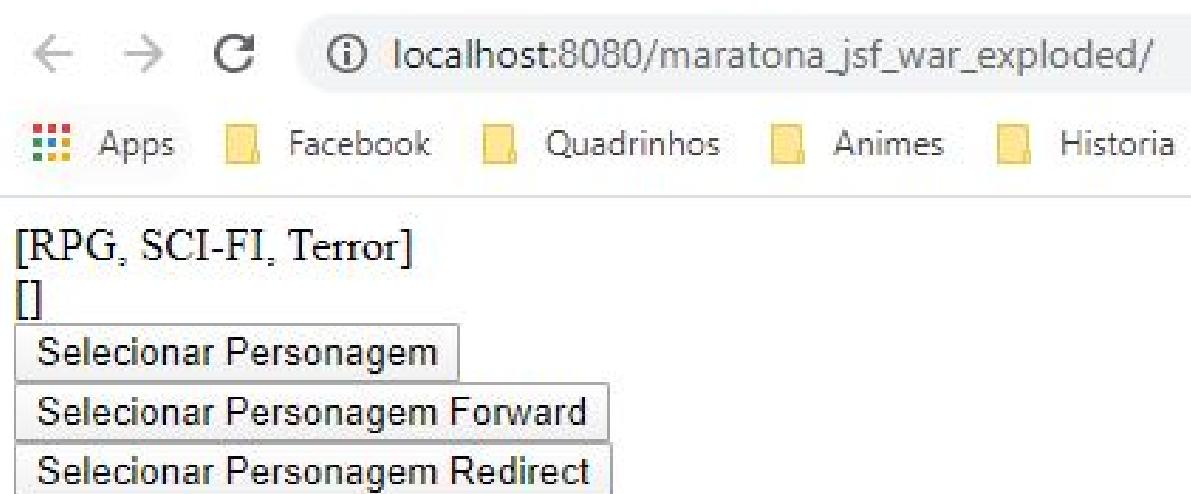
    @PostConstruct
    public void init(){
        System.out.println("Entrou no PostConstruct da Application");
        categoriaList = asList("RPG", "SCI-FI", "Terror");
    }

    public void mudarLista() { categoriaList = asList("RPG", "SCI-FI", "Terror", "Medieval"); }

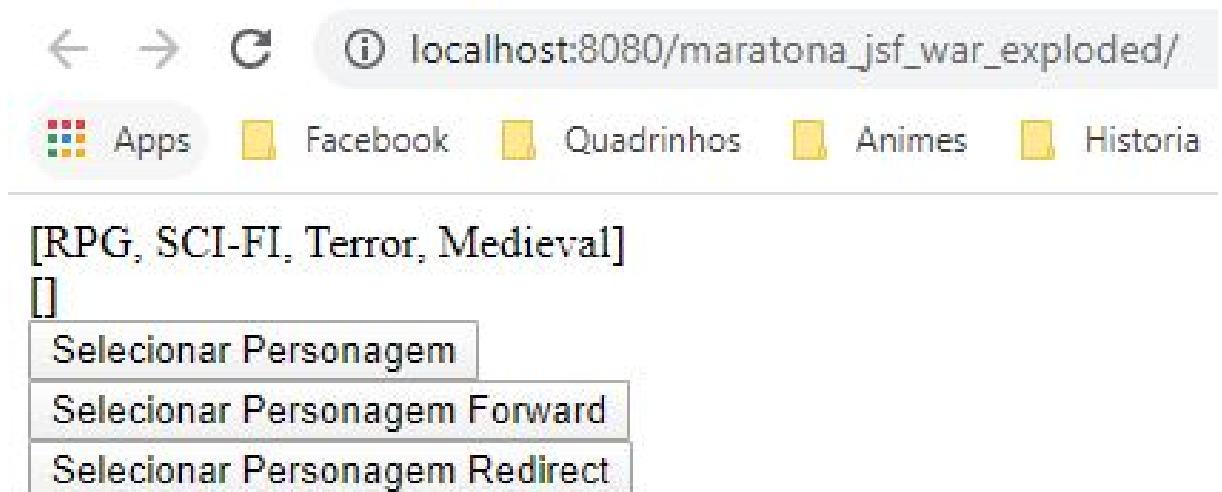
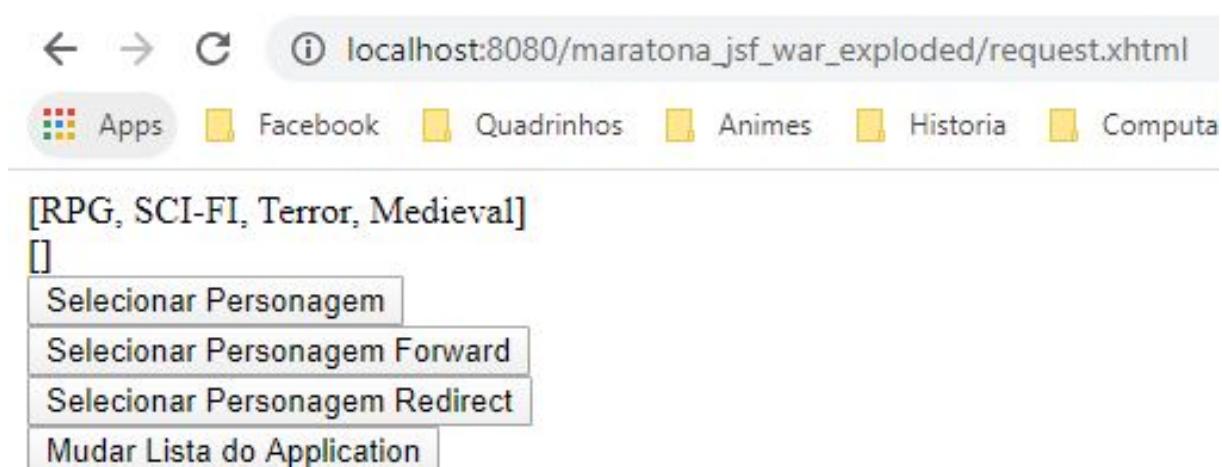
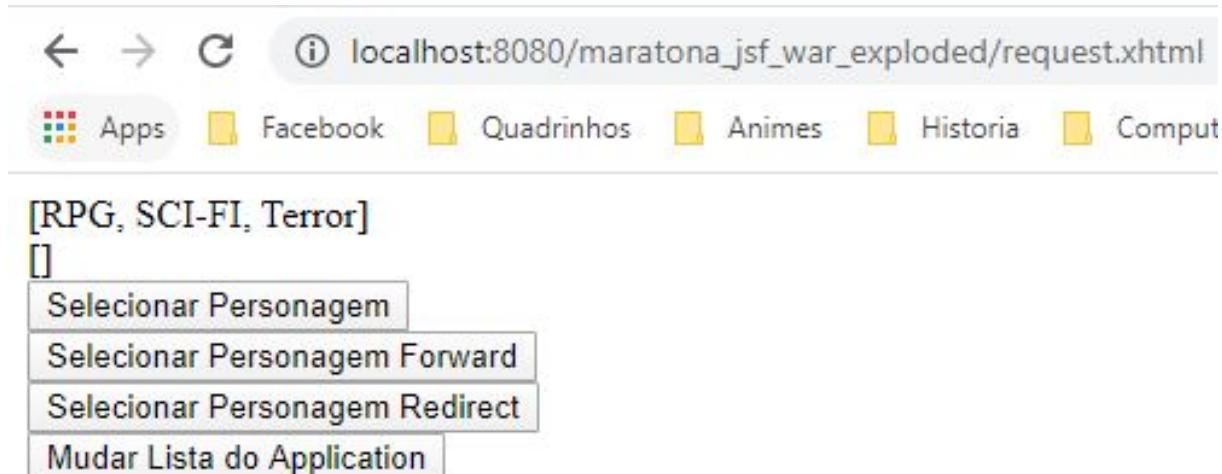
    public List<String> getCategoriaList() { return categoriaList; }

    public void setCategoriaList(List<String> categoriaList) { this.categoriaList = categoriaList; }
}
```

Para testar seu funcionamento sua lista de personagens foi exibida em todas as outras página, session, view e redirect. Figura 2 mostra o resultado na página 'view.xhtml'.



Na página 'request.xhtml' foi criado um botão que adiciona mais um nome a lista do application scope e é possível ver essa mudança em todas as páginas. Figura 3 mostra o botão de mudar lista e Figura 4 e 5 mostram as mudanças nas páginas 'view.xhtml' e 'request.xhtml'.



A Figura 6 mostra as alterações no código de arquivo 'request.xhtml' para poder adicionar o botão de mudar lista e exibir a lista do application scope, essa mudança foi aplicada aos outros arquivos.

```
<h:outputText value="#{testeApplicationBean.categoriaList}"/><br/>
<h:outputText value="#{testeRequestBean.personagemSelecionado}"/><br/>
<h:commandButton actionListener="#{testeRequestBean.selecionarPersonagem()}" value="Selecionar Personagem"/><br/>
<h:commandButton actionListener="#{testeRequestBean.selecionarPersonagem()}" value="Selecionar Personagem Forward">
<h:commandButton actionListener="#{testeRequestBean.selecionarPersonagem()}" value="Selecionar Personagem Redirect">
<h:commandButton actionListener="#{testeApplicationBean.mudarLista()}" value="Mudar Lista do Application"/><br/>
```

# Aula 16

Na aula 16 foi trabalhado o Conversation Scope.

## Conversation Scope

O conversation scope possui dois estados, o primeiro funciona de forma semelhante ao request e o segundo é capaz de manter os dados enquanto permanecer nesse estado. Para fazer a troca de estado foi criado um método na classe 'TesteConversationBean'. A Figura 7 mostra o código.

```
@Inject
private Conversation conversation;

public void init(){
    System.out.println("Entrou no PostConstruct do Conversation");
    personagens = asList("Logan", "Wally West", "Dick Greyson");
    if(conversation.isTransient()){
        conversation.begin();
        System.out.println("Iniciando Conversation, ID:"+conversation.getId());
    }
}

public String conversationEnd(){
    if(!conversation.isTransient()){
        conversation.end();
    }
    return "conversation?face-redirect=true";
}
```

Para testar o conversation scope foi criado um arquivo 'conversation.xhtml'. A Figura 8 mostra o código desse arquivo.

```
<metadata>
<f:viewAction action="#{testeConversationBean.init()}"/>
</metadata>
<body>
<h:form>
    <h:outputText value="#{testeApplicationBean.categoriaList}" /><br/>
    <h:outputText value="#{testeConversationBean.personagemSelecionado}" /><br/>
    <h:commandButton actionListener="#{testeConversationBean.selecionarPersonagem()}" value="Selecionar Personag" />
    <h:commandButton actionListener="#{testeConversationBean.selecionarPersonagem()}" value="Selecionar Personag" />
    <h:commandButton actionListener="#{testeConversationBean.selecionarPersonagem()}" value="Selecionar Personag" />
    <h:link outcome="conversation2" value="Vai para o Conversetion2" rendered="#{!testeConversationBean.conversa" />
        <f:param name="cid" value="#{testeConversationBean.conversation.id}" />
    </h:link><br/>
    <h:commandButton action="#{testeConversationBean.conversationEnd()}" value="Finalizar Conversation"/><br/>
</h:form>
</body>
</>
```

Após o estado ser alterado um ID é criado e os dados da aplicação permanecem enquanto o estado permanecer inalterado. A Figura 9 mostra os resultados na porta 8080.

The screenshot shows a web browser window with two tabs: "Maratona JSF - Aula 16: Escopos" and "localhost:8080/maratona\_jsf\_war\_exploded/conversation.xhtml?cid=1". The main content area displays "[RPG, SCI-FI, Terror]" and "[Dick Greyson, Logan, Logan, Dick Greyson, Logan]". Below this are several buttons: "Selecionar Personagem" (highlighted), "Selecionar Personagem Forward", "Selecionar Personagem Redirect", and "Finalizar Conversation".

Caso tente acessar a mesma página com o mesmo ID em um outro navegador irá ocorrer um erro, pois o ID não será reconhecido. Figura 10 mostra o erro gerado.

The screenshot shows a browser window titled "ERROR" with the URL "localhost:8080/maratona\_jsf\_war\_exploded/conversation.xhtml?cid=1". The main content area has a red minus sign icon and the text "Error processing request". Below this, it lists request details and a stack trace:

**Context Path:** /maratona\_jsf\_war\_exploded  
**Servlet Path:** /conversation.xhtml  
**Path Info:** null  
**Query String:** cid=1  
**Stack Trace:**

```
javax.servlet.ServletException: WELD-000321: No conversation found to restore for id 1
```

Para finalizar a aplicação foi criado um botão onde novamente é realizada uma mudança de estado, dessa forma o id gerado é perdido. Para reiniciar basta atualizar a página e um novo ID será gerado. A Figura 11 mostra o resultado após o botão de finalizar é clicado.

Maratona JSF - Aula 16: Escopos X localhost:8080/maratona\_jsf\_war\_Exploded/conversatio X

localhost:8080/maratona\_jsf\_war\_exploded/conversatio

Apps Facebook Quadrinhos Animes Historia Com

[RPG, SCI-FI, Terror]

Selecionar Personagem

Selecionar Personagem Forward

Selecionar Personagem Redirect

Finalizar Conversation

### Mensagem de Erro

Após o ID ser criado é possível ir na url e alterar o valor dele, ao fazer uma mensagem de erro irá aparecer, é possível configurar a aplicação para caso esse erro ocorra uma nova página, chamada 'erro.xhtml' é aberta. Figura 12 mostra o código para redirecionar a página e Figura 13 mostra o resultado.

```
<error-page>
    <location>/error.xhtml</location>
</error-page>
```

Maratona JSF - Aula 16: Escopos X localhost:8080/maratona\_jsf\_war\_Exploded/conversatio X

localhost:8080/maratona\_jsf\_war\_exploded/conversatio

Apps Facebook Quadrinhos Animes Historia Com

ID Inválido

## Aula 17

Na aula 17 se dá inicio ao uso de flow scoped.

### Páginas do Flow Scoped

Para o uso do flow scoped foram criadas multiplas páginas que armazenam a informação e seguem para a proxima página, podendo também voltar para a página anterior, ao final do processo as informações são exibidas no terminal. Figura 1 mostra o código do arquivo que configura a página inicial do flow.

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <flow-definition id="registration">
        <flow-return id="exitToInicio">
            <from-outcome>/inicioflow.xhtml</from-outcome>
        </flow-return>
    </flow-definition>
```

As páginas seguem o caminho do ponto inicial e segue por três páginas onde onde serão inseridos nome, sobrenome e endereço, ao final é feito o retorno a página inicial e os dados são exibidos no terminal. As Figuras 2, 3, 4 e 5 mostram os códigos das páginas do flow.

```
<h:head>
    <h:body>
        <h2>Ponto de entrada do FlowScoped</h2>
        <h:form>
            <h:commandButton value="Entrar no Flow" action="registration"/>
        </h:form>
    </h:body>
</h:head>
```

```
<h:head>
    <h:body>
        <h:form>
            <h2>Primeira pagina de registro FlowScoped</h2>
            <h:panelGrid columns="2">
                <h:outputLabel value="Nome"/>
                <h:inputText value="#{testeFlowBean.nome}" />
                <h:outputLabel value="Sobrenome"/>
                <h:inputText value="#{testeFlowBean.sobrenome}" />
                <h:commandButton value="Next" action="registration2?faces-redirect=true"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</h:head>
```

```

<h:head>
    <h:body>
        <h:form>
            <h2>Segunda pagina de registro FlowScoped</h2>
            <h:panelGrid columns="2">
                <h:outputLabel value="Endereco"/>
                <h:inputText value="#{testeFlowBean.endereco}" />
                <h:commandButton value="Back" action="registration?faces-redirect=true"/>
                <h:commandButton value="Next" action="registration3?faces-redirect=true"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</h:head>

```

```

<h:head>
    <h:body>
        <h:form>
            <h2>Terceira pagina de registro FlowScoped</h2>
            <h:panelGrid columns="2">
                <h:outputLabel value="Nome"/>
                <h:outputLabel value="#{testeFlowBean.nome}" />
                <h:outputLabel value="Sobrenome"/>
                <h:outputLabel value="#{testeFlowBean.sobrenome}" />
                <h:outputLabel value="Endereco"/>
                <h:outputLabel value="#{testeFlowBean.endereco}" />
                <h:commandButton value="Back" action="registration2?faces-redirect=true"/>
                <h:commandButton value="Salvar" action="#{testeFlowBean.salvar()}" />
            </h:panelGrid>
        </h:form>
    </h:body>
</h:head>

```

Para realizar os testes foi criado uma classe chamada 'TesteFlowBean' que possui um nome, sobrenome e endereço. A Figura 6 mostra o código dessa classe.

```

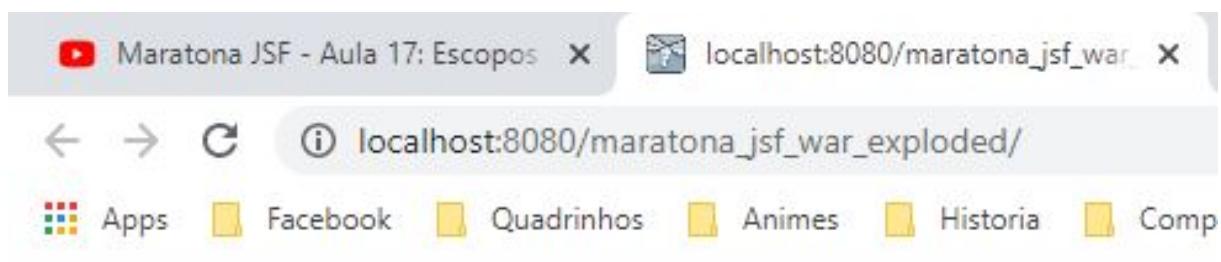
@Named
@FlowScoped(value = "registration")
public class TesteFlowBean implements Serializable {
    private String nome;
    private String sobrenome;
    private String endereco;

    public String salvar(){
        System.out.println("Salvando no Banco");
        System.out.println(nome);
        System.out.println(sobrenome);
        System.out.println(endereco);
        return "exitToInicio";
    }
}

```

## Resultados

Após entrar no flow a primeira página vai pedir pra inserir um nome e sobrenome pra classe, na segunda página o campo endereço pode ser preenchido e na terceira página é exibido os dados preenchidos nas páginas anteriores. As Figuras 7, 8, 9 e 10 mostram os resultados na porta 8080 e a Figura 11 mostra o resultado no terminal.



## Ponto de entrada do FlowScoped

[Entrar no Flow](#)



← → C ⓘ localhost:8080/maratona\_jsf\_war\_exploded/registration/re

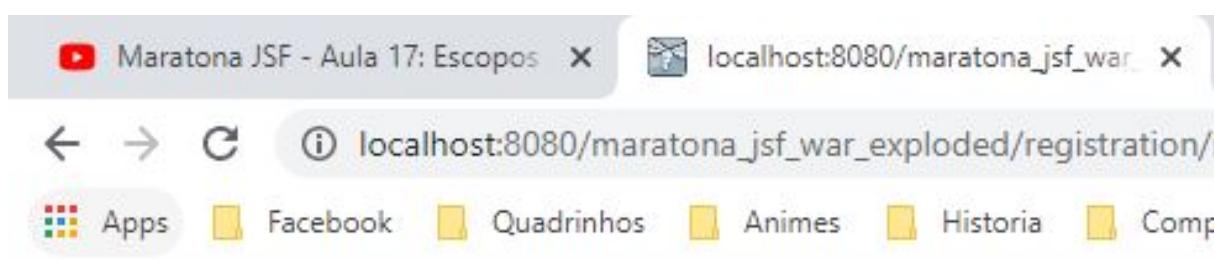
Apps Facebook Quadrinhos Animes Historia Comput

---

## Primeira pagina de registro FlowScoped

Nome

Sobrenome



## Segunda pagina de registro FlowScoped

Endereço

The screenshot shows a web browser window with two tabs. The active tab is titled 'localhost:8080/maratona\_jsf\_war\_exploded/registration/re' and displays a registration form. The form has fields for Nome ('Jaime'), Sobrenome ('Eduardo'), and Endereco ('Rua Sebastiao M Ramalho'). Below the form are two buttons: 'Back' and 'Salvar'. The browser's address bar also shows the URL. The top of the browser window has a navigation bar with icons for back, forward, and search.

Nome      Jaime  
Sobrenome      Eduardo  
Endereco      Rua Sebastiao M Ramalho

Back      Salvar

```
09:24:11,040 INFO [stdout] (default task-1) Salvando no Banco
09:24:11,040 INFO [stdout] (default task-1) Jaime
09:24:11,040 INFO [stdout] (default task-1) Eduardo
09:24:11,043 INFO [stdout] (default task-1) Rua Sebastiao M Ramalho
```

## Aula 18

A aula 18 da continuidade ao uso do flow scoped, dando um foco na parte de configuração.

### Finalizer

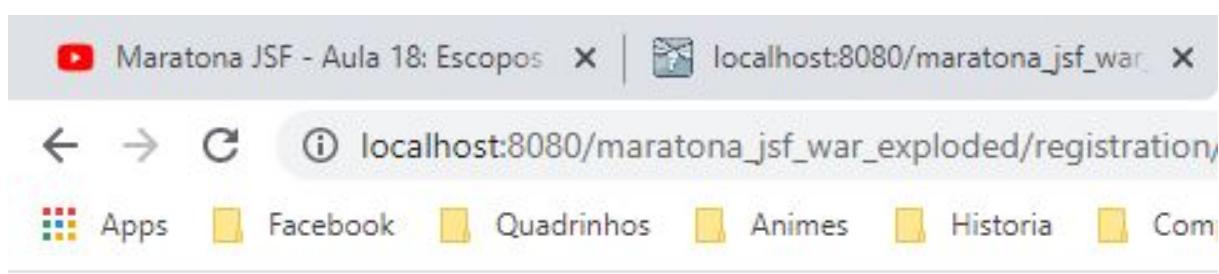
A primeira configuração foi para mudar a forma de finalizar o flow, chamando a função de salvar os dados no terminal através dessa configuração ao invés de chamar na página três do flow. Figura 12 mostra o código.

```
<finalizer>#{testeFlowBean.salvar()}</finalizer>
```

### Campo Obrigatório

Uma alteração foi feita na página onde o campo nome passou a ser obrigatório e caso não fosse preenchido uma mensagem seria exibida. A Figura 13 mostra o código e a Figura 14 mostra o resultado.

```
<h:messages/>
<h:form>
    <h2>Primeira pagina de registro FlowScoped</h2>
    <h:panelGrid columns="2">
        <h:outputLabel value="Nome"/>
        <h:inputText value="#{testeFlowBean.nome}" required="true" requiredMessage="nome obrigatorio"/>
    </h:panelGrid>
</h:form>
```



## Primeira pagina de registro FlowScoped

Nome	<input type="text"/>
Sobrenome	Eduardo
<input type="button" value="Next"/>	

Outra forma de tornar os campos obrigatórios foi apresentada nessa aula, no arquivo de configuração foi adicionado um switch-case onde se os campos nome e sobrenome não estiverem preenchidos ele permanece na mesma página. A Figura 15 mostra o código.

```
<flow-definition id="registration">
    <start-node>registrationInicio</start-node>
    <view id="registrationInicio">
        <vdl-document>/registration/registration1.xhtml</vdl-document>
    </view>
    <switch id="registrationPage2">
        <case>
            <if>#{not empty testeFlowBean.nome and not empty testeFlowBean.sobrenome}</if>
                <from-outcome>registration2</from-outcome>
            </case>
            <default-outcome>registrationInicio</default-outcome>
        </switch>
        <flow-return id="exitToInicio">
            <from-outcome>/inicioflow.xhtml</from-outcome>
        </flow-return>
        <finalizer>#[testeFlowBean.salvar()]</finalizer>
    </flow-definition>
```

## Aula 19

Na aula 19 é utilizado o conceito de nested flow scope.

### Nested Flow Scope

Nested flow é uma sub etapa que precisa ser acessada antes do flow principal seguir para a próxima página. Na aula foi criada uma página de validação para os dados do nome e sobrenome inseridos. Figura 1 mostra o código de configuração e Figura 2 mostra o código da classe criada para realizar a validação.

```
<flow-definition id="pendencies">
    <flow-return id="proceedToRegistration3">
        <from-outcome>/registration/registration3</from-outcome>
    </flow-return>
    <flow-return id="exitToInicio">
        <from-outcome>registrationInicid</from-outcome>
    </flow-return>
    <inbound-parameter>
        <name>userName</name>
        <value>#{testeFlowNestedBean.userName}</value>
    </inbound-parameter>
    <inbound-parameter>
        <name>userSurname</name>
        <value>#{testeFlowNestedBean.userSurname}</value>
    </inbound-parameter>
</flow-definition>
```

```
@Named
@FlowScoped(value = "pendencies")
public class TesteFlowNestedBean implements Serializable {
    private String userName;
    private String userSurname;

    public String validarUser(){
        System.out.println("Fazendo Consulta no Banco de Dados");
        System.out.println("Obtendo Permissao");
        System.out.println("Validacao obtida com sucesso");
        if (true) {
            FacesContext context = FacesContext.getCurrentInstance();
            context.addMessage( s: null, new FacesMessage(FacesMessage.SEVERITY_ERROR,
                summary: "Usuario nao valido", detail: "nao encontrado no banco"));
            return null;
        }
        return "proceedToRegistration3";
    }
}
```

Com a configuração e a classe prontas, foi criado um arquivo xhtml que gera a página a ser exibida na porta 8080 quando o programa for executado. Figura 3 mostra o código do arquivo e Figura 4 mostra o resultado.

```
<h:head>
    <h:body>
        <h:form>
            <p:messages autoUpdate="true"/>
            <h2>Validando dados do usuario</h2>
            <h:panelGrid columns="2">
                <h:outputLabel value="#{testeFlowNestedBean.userName}"/>
                <h:outputLabel value="#{testeFlowNestedBean.userSurname}"/>
                <h:commandButton value="Validar Usuario" action="#{testeFlowNestedBean.validarUser()}" />
                <h:commandButton value="Volta Inicio" action="exitToInicio"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</h:head>
```



## Validando dados do usuario

Jaime

Eduardo

**Validar Usuario**

### Validação

Foram testados dois casos, o primeira em que a validação é bem sucedida e mensagens são exibidas no terminal, o segundo em que a validação falha e uma mensagem é exibida. Figura 5 mostra o resultado no terminal em caso de sucesso e a Figura 6 mostra a mensagem de erro em caso de falha.

```
09:18:21,455 INFO [stdout] (default task-1) Fazendo Consulta no Banco de Dados
09:18:21,455 INFO [stdout] (default task-1) Obtendo Permissao
09:18:21,455 INFO [stdout] (default task-1) Validacao obtida com sucesso
```



Apps



Facebook



Quadrinhos



Animais

**Usuario nao validado**

## Validando dados do usuario

Jaime

Eduardo

[Validar Usuario](#)[Volta Inicio](#)

## Aula 20

Na aula 20 foi mostrada uma outra forma de configurar o flow, ao invés de usar arquivos xml é criada uma classe em java.

### Configuração em Java

Para fazer a configuração do flow através de java foram criadas duas novas classes uma pra configurar as etapas principais do flow e outra pra sub etapa de validação. Figura 7 e 8 mostram os códigos de configuração.

```
@Produces
@FlowDefinition
public Flow defineFlow(@FlowBuilderParameter FlowBuilder flowBuilder){
    String flowId = "newregistration";
    flowBuilder.id( s: "", flowId);
    flowBuilder.viewNode(flowId, s1: "/newregistration/newregistration.xhtml").markAsStartNode();
    flowBuilder.viewNode(flowId, s1: "/newregistration/newregistration2.xhtml");
    flowBuilder.viewNode(flowId, s1: "/newregistration/newregistration3.xhtml");
    flowBuilder.switchNode( s: "newRegistrationPage2") SwitchBuilder
        .defaultOutcome(flowId) SwitchCaseBuilder
        .switchCase()
        .condition("#{not empty testeFlowBuilderBean.nome and not empty testeFlowBuilderBean.sobrenome}")
        .fromOutcome("newregistration2");
    flowBuilder.flowCallNode( s: "callNewPendencies")
        .flowReference( s: "", s1: "newpendencies")
        .outboundParameter( s: "userName", s1: "#{testeFlowBuilderBean.nome}")
        .outboundParameter( s: "userSurname", s1: "#{testeFlowBuilderBean.sobrenome}");
    flowBuilder.returnNode( s: "exitToInicio").fromOutcome("/inicioflow.xhtml");
    flowBuilder.returnNode( s: "exitToIndex").fromOutcome("/index.xhtml");
    flowBuilder.finalizer("#{testeFlowBuilderBean.salvar()}");
    return flowBuilder.getFlow();
```

```
public class NewPendenciesFlowBuilder implements Serializable {
    @Produces
    @FlowDefinition
    public Flow defineFlow(@FlowBuilderParameter FlowBuilder flowBuilder) {
        String flowId = "newpendencies";
        flowBuilder.id( s: "", flowId);
        flowBuilder.viewNode(flowId, s1: "/newpendencies/newpendencies.xhtml").markAsStartNode();
        flowBuilder.returnNode( s: "proceedToNewRegistration3")
            .fromOutcome("/newregistration/newregistration3.xhtml");
        flowBuilder.returnNode( s: "exitToNewInicio")
            .fromOutcome("/newregistration/newregistration.xhtml");
        flowBuilder.inboundParameter( s: "userName", s1: "#{testeFlowBuilderNestedBean.userName}");
        flowBuilder.inboundParameter( s: "userSurname", s1: "#{testeFlowBuilderNestedBean.userSurname}");
        return flowBuilder.getFlow();
    }
}
```

Após a configuração ser concluída é possível obter os mesmos resultados de aulas anteriores. Figura 9 mostra os resultados na porta 8080 e Figura 10 mostra o resultado no terminal.

The screenshot shows a web browser window with two tabs. The active tab is titled 'localhost:8080/maratona\_jsf\_war\_exploded/newpendencies'. The page content displays a registration form with three input fields: 'Nome' (Nome: Jaime), 'Sobrenome' (Sobrenome: Eduardo), and 'Endereco' (Endereço: Rua Sebastiao M Ramalho). Below the form are two buttons: 'Back' and 'Salvar'.

Nome      Jaime  
Sobrenome      Eduardo  
Endereco      Rua Sebastiao M Ramalho

Back      Salvar

```
11:32:33,930 INFO [stdout] (default task-1) Salvando no Banco do Builder
11:32:33,931 INFO [stdout] (default task-1) Jaime
11:32:33,931 INFO [stdout] (default task-1) Eduardo
11:32:33,931 INFO [stdout] (default task-1) Rua Sebastiao M Ramalho
```

## Aula 21

Na aula 21 é utilizado o dependent scope.

### Dependent Scope

O dependent scope mantém os dados vivos de acordo com o escopo em que ele foi inserido. Figura 1 mostra o código do dependent scope sendo inserido via inject no view scope.

```
private final TesteDependentBean dependentBean;

@Inject
public TesteViewBean(TesteDependentBean dependentBean){
    this.dependentBean = dependentBean;
}

@PostConstruct
public void init(){
    System.out.println("Entrou no PostConstruct do View");
    personagens = asList("Tony Stark", "Barry Allen", "Hal Jordan");
}

public void selecionarPersonagem(){
    int index = ThreadLocalRandom.current().nextInt( bound: 3);
    String personagem = personagens.get(index);
    personagemSelecionado.add(personagem);
    dependentBean.getPersonagemSelecionado().add(personagem);
}
```

### Resultados

No view.xhtml o personagem selecionado foi chamado via view e dependent scope, com isso é possível observar o dependent scope tendo o mesmo comportamento do view. Figura 2 mostra o código em xhtml e a Figura 3 mostra o resultado na porta 8080.

```
<h:head>
<h:body>
    <h:form>
        <h:outputText value="#{testeApplicationBean.categoriaList}" /><br/>
        <h:outputText value="#{testeViewBean.personagemSelecionado}" /><br/>
        <h:outputText value="Dependent: #{testeViewBean.dependentBean.personagemSelecionado}" /><br/>
        <h:commandButton actionListener="#{testeViewBean.selecionarPersonagem()}" value="Selecionar" />
        <h:commandButton actionListener="#{testeViewBean.selecionarPersonagem()}" value="Selecionar" />
        <h:commandButton actionListener="#{testeViewBean.selecionarPersonagem()}" value="Selecionar" />
    </h:form>
```

The screenshot shows a browser window with two tabs: "Maratona JSF - Aula 21: Escopos" and "localhost:8080/maratona\_jsf\_war\_exploded/view.xhtml?jfwid=...". The main content area displays the following text:  
[RPG, SCI-FI, Terror]  
[Tony Stark, Hal Jordan, Barry Allen, Barry Allen, Barry Allen]  
Dependent: [Tony Stark, Hal Jordan, Barry Allen, Barry Allen, Barry Allen]  
Three buttons are visible:  
- Seleccionar Personagem (highlighted)  
- Seleccionar Personagem Forward  
- Seleccionar Personagem Redirect

### Dependent sem ser Inserido

Caso o dependent scope seja chamado sem estar inserido em um contexto o dado será criado e destruído automaticamente. Na Figura 4 mostra o resultado na página dependent.xhtml, nessa página caso o botão selecionar personagem seja precionado nenhum personagem irá aparecer.

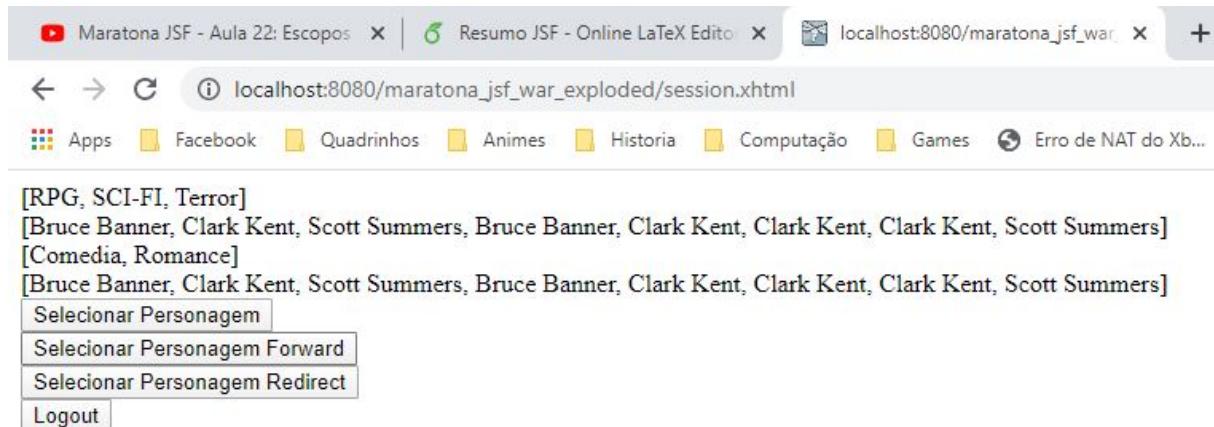
The screenshot shows a browser window with two tabs: "Maratona JSF - Aula 21: Escopos" and "localhost:8080/maratona\_jsf\_war\_exploded/view.xhtml?jfwid=...". The main content area displays the following text:  
[RPG, SCI-FI, Terror]  
[]  
Three buttons are visible:  
- Seleccionar Personagem (highlighted)  
- Seleccionar Personagem Forward  
- Seleccionar Personagem Redirect

## Aula 22

Na aula 22 é trabalhado em mais detalhes o conceito injection.

### Injection

O exemplo de injeção utilizado na aula 22 foi um injeção do session scope na application scope onde o personagem selecionado do session aparece na página session.xhtml através do session scope e da application scope e podemos observar na Figura 5 que os valores são iguais.



### Session recebe Estudante

O session scope recebe a classe estudante e exibi o nome na página view.xhtml através da application scope e do session scope. Figura 6 mostra o código em java, Figura 7 mostra o código em xhtml e Figura 8 mostra o resultado na porta 8080.

```
@SessionScoped
public class TesteSessionBean implements Serializable {
    private List<String> personagens;
    private List<String> personagemSelecionado = new ArrayList<>();
    private Estudante estudante;

    @PostConstruct
    public void init(){
        System.out.println("Entrou no PostConstruct do Session");
        personagens = asList("Clark Kent", "Scott Summers", "Bruce Banner");
        logar();
    }

    public void logar(){
        estudante = new Estudante();
    }
}
```

```
<h:outputText value="#{testeApplicationBean.categoriaList}" /><br/>
<h:outputText value="#{testeApplicationBean.dependentBean.categoriaList}" /><br/>
<h:outputText value="Estudante Application: #{testeApplicationBean.sessionBean.estudante.nome}" />
<h:outputText value="Estudante Session: #{testeSessionBean.estudante.nome}" /><br/>
```

Maratona JSF - Aula 22: Escopos

localhost:8080/marato

Apps Facebook Quadrinhos

[RPG, SCI-FI, Terror]  
[Comedia, Romance]

Estudante Application: Jaime  
Estudante Session: Jaime

[]

Dependente: []

Selecionar Personagem

Selecionar Personagem Forward

Selecionar Personagem Redirect

## Aula 23

Na aula 23 foi implementado um login e senha para acessar uma nova página.

### Login

Foi criada uma classe login com nome e senha como atributos e métodos logar e logout. Figura 1 mostra o código da classe login.

```
public class LoginBean implements Serializable {
    private String nome;
    private String senha;
    private Estudante estudante;

    public String logar(){
        if(nome.equals("J") && senha.equals("1")){
            estudante = new Estudante();
            return "/restricted/iniciosistema.xhtml?faces-redirect=true";
        }
        FacesContext context = FacesContext.getCurrentInstance();
        context.addMessage( null, new FacesMessage(FacesMessage.SEVERITY_ERROR,
                summary: "Usuario e/ou senha invalidos", detail: ""));
        return null;
    }

    public String logout(){
        FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
        estudante = null;
        return "/login?faces-redirect=true";
    }
}
```

Na página login.xhtml existem dois campos obrigatórios e uma vez que sejam preenchidos é aberta a página do sistema que exibi o nome de usuário e possui a opção de logout. Figura 2 e 3 mostram os códigos em xhtml das páginas de login e de usuário.

```
<h:head>
<h:body>
    <h:form>
        <h:panelGrid columns="1">
            <p:messages autoUpdate="true"/>
            <h:outputLabel value="Nome"/>
            <h:inputText value="#{loginBean.nome}" required="true" requiredMessage="Nome Obrigatorio"/>
            <h:outputLabel value="Senha"/>
            <h:inputSecret value="#{loginBean.senha}" required="true" requiredMessage="Senha Obrigatoria"/>
            <h:commandButton value="Logar" action="#{loginBean.logar()}" />
        </h:panelGrid>
    </h:form>
</h:body>
```

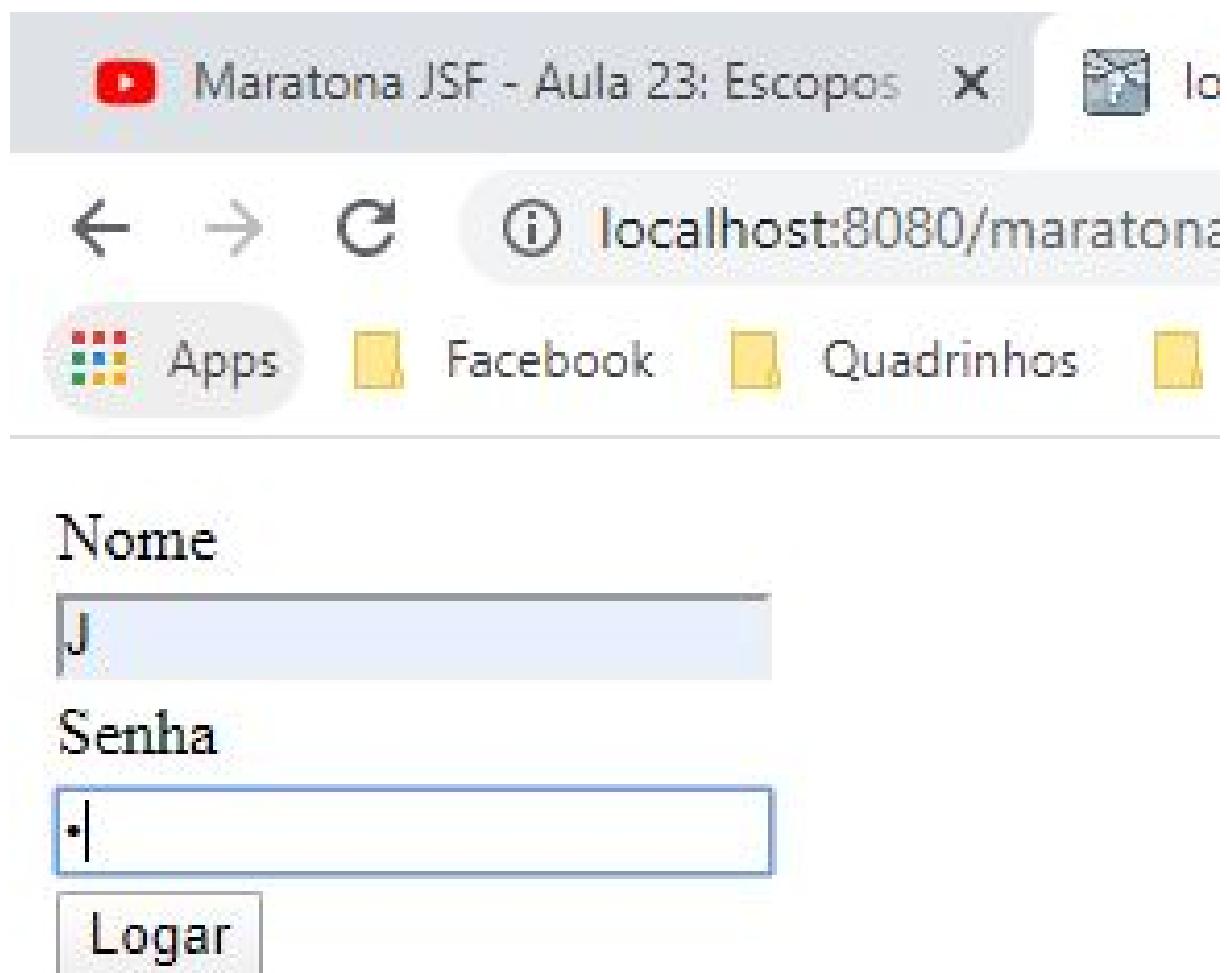
```

<h:head>
    <h:body>
        <h:form>
            <h1>Usuário Logado: #{loginBean.estudante.nome}</h1>
            <h:commandButton value="Logout" action="#{loginBean.logout()}" />
        </h:form>
    </h:body>
</h:head>

```

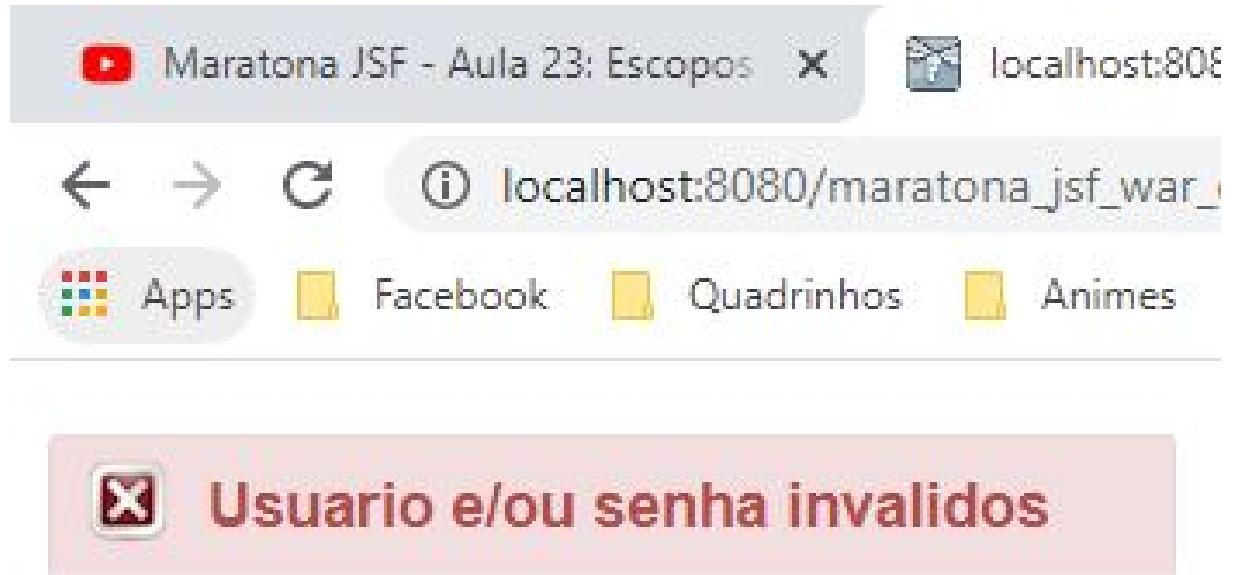
## Resultados

Uma vez que os dados corretos são preenchidos a página de acesso exibi o nome do usuário e a opção de logout, porém caso seja feita uma alteração na url é possível acessar a página de usuário sem ter efetuado login. Figura 4 mostra a página de login, Figura 5 mostra a página do usuário, Figura 6 mostra a página de usuário sem login efetuado e a Figura 7 mostra a página de login com notificação de erro caso o nome ou a senha estejam errados.



A screenshot of a web browser window. The title bar says "Maratona JSF - Aula 23: Escopos" and the address bar says "localhost:8080/maratona\_jsf\_war\_exploded/rest". Below the address bar are five navigation links: "Apps" (with a grid icon), "Facebook" (with a yellow square icon), "Quadrinhos" (with a yellow square icon), "Animes" (with a yellow square icon), and "Historia" (with a yellow square icon). The main content area displays the text "Usuário Logado: Jaime" and a "Logout" button.

A screenshot of a web browser window. The title bar says "Maratona JSF - Aula 23: Escopos" and the address bar says "localhost:8080/maratona\_jsf\_war\_exploded/rest". Below the address bar are five navigation links: "Apps" (with a grid icon, highlighted in blue), "Facebook" (with a yellow square icon), "Quadrinhos" (with a yellow square icon), "Animes" (with a yellow square icon), and "Historia" (with a yellow square icon). The main content area displays the text "Usuário Logado:" and a "Logout" button.



## Filtro

Para impedir que seja possível acessar a página de usuário sem ter o login efetuado foi criada uma classe de filtro que caso a página de usuário tente ser acessado sem login será feito um redirecionamento para a página de login. Figura 8 mostra o código da classe Filtro e Figura 9 mostra a configuração no arquivo web.xml.

```
public class LoginFilter implements Filter {
    @Inject
    private LoginBean loginBean;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
        HttpServletRequest req = (HttpServletRequest) servletRequest;
        HttpServletResponse res = (HttpServletResponse) servletResponse;
        String url = req.getRequestURL().toString();
        if(url.contains("/restricted") && loginBean.getEstudante() == null){
            res.sendRedirect( req.getServletContext().getContextPath()+"/login.xhtml");
        }else{
            filterChain.doFilter(servletRequest,servletResponse);
        }
    }
}
```

```
<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>br.com.maratonajsf.filter.LoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/restricted/*</url-pattern>
</filter-mapping>
```

## Aula 24

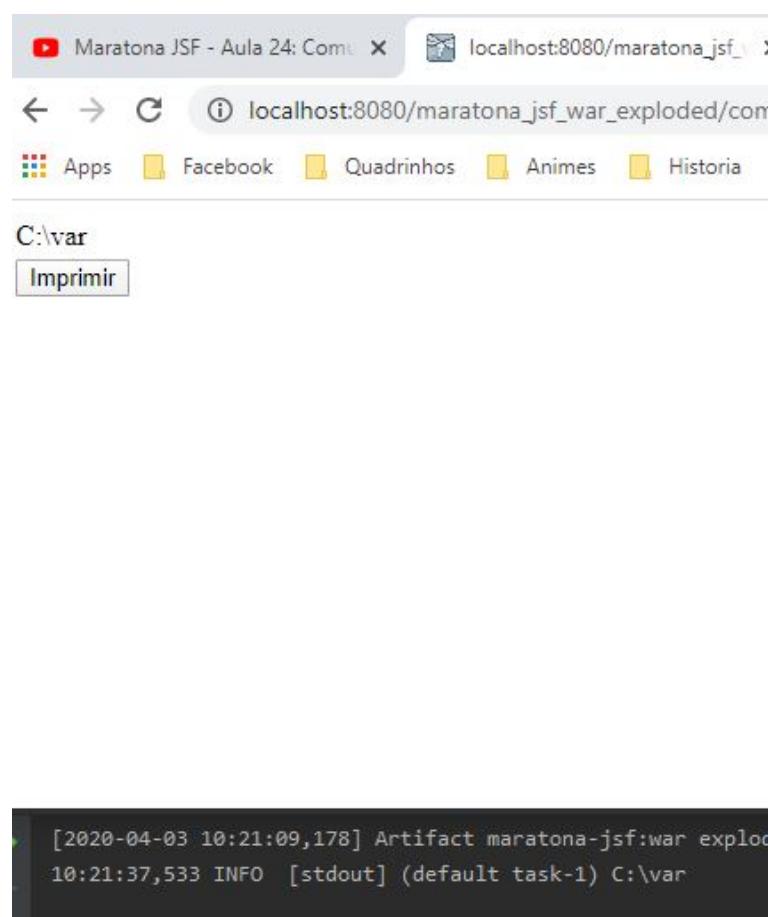
Na aula 24 foi trabalhado o conceito de criar variáveis que podem ser acessadas por todo o sistema.

### Variável no web.xml

A primeira forma de criar esse tipo de variável é definindo no arquivo web.xml. Figura 10 mostra o código.

```
<context-param>
    <param-name>images.location</param-name>
    <param-value>C:\var</param-value>
</context-param>
```

Após a definição da variável foi criado uma página que exibi essa informação e possui um botão que permite exibir essa informação no terminal. Figura 11 mostra o resultado.



### Atributos via Return

Uma classe de teste foi criada onde o nome e sobrenome são enviado para uma outra classe através do return de um dos métodos da primeira classe. Uma página xhtml também foi criada para a realização desse teste. A Figura 12 mostra o código da classe que envia os dado, Figura 13 mostra o código que recebe os dados, Figura 14 mostra o código da página xhtml e Figura 15 mostra o código da segunda página em xhtml.

```

@ViewScoped
public class ComunicacaoTeste1Bean implements Serializable {
    private String nome;
    private String sobrenome;

    public void imprimirAtributos(){
        String initParameter = FacesContext.getCurrentInstance()
            .getExternalContext()
            .getInitParameter( s: "images.location");
        System.out.println(initParameter);
    }

    public String save(){
        System.out.println(nome);
        System.out.println(sobrenome);
        return "comunicacao2?faces-redirect=true&nome="+nome+"&sobrenome="+sobrenome;
    }
}

```

```

class ComunicacaoTeste2Bean implements Serializable {
private String nome;
private String sobrenome;

stConstruct
lic void init(){
    System.out.println("criou comunicacao 2");
    Map<String, String> paramsMap = FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap();
    nome = paramsMap.get("nome");
    sobrenome = paramsMap.get("sobrenome");
    System.out.println(nome);
    System.out.println(sobrenome);
}

```

```

<h:panelGrid columns="1">
    <h:outputText value="#{facesContext.externalContext.getInitParameter('images.location')}" />
    <h:commandButton value="Imprimir" actionListener="#{comunicacaoTeste1Bean.imprimirAtributos()}" />
</h:panelGrid>
/h:form>
n:form>
<h:panelGrid columns="1">
    <h:inputText value="#{comunicacaoTeste1Bean.nome}" />
    <h:inputText value="#{comunicacaoTeste1Bean.sobrenome}" />
    <h:commandButton value="Save" action="#{comunicacaoTeste1Bean.save()}" />
    <h:button value="Save 2" outcome="comunicacao2?faces-redirect=true">
        <f:param name="nome" value="Padilla"/>
        <f:param name="sobrenome" value="Aranha"/>
    </h:button>
    <h:link value="Vai Para Comunicacao 2" outcome="comunicacao2">
        <f:param name="nome" value="Eduardo"/>
        <f:param name="sobrenome" value="Padilla"/>
    </h:link>
</h:panelGrid>

```

```

<h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="1">
                <h:inputText value="#{comunicacaoTeste2Bean.nome}" />
                <h:inputText value="#{comunicacaoTeste2Bean.sobrenome}" />
            </h:panelGrid>
        </h:form>
    </h:body>
</h:head>

```

## Resultados

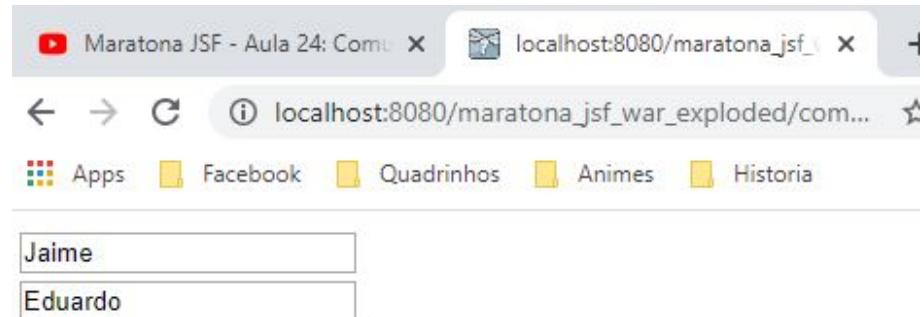
Inicialmente a página de teste após receber os nomes exibia ambos no terminal, após o teste inicial foi criada uma classe que recebe esses valores e exibe ambos em uma outra página. Figura 16 mostra o resultado inicial, a Figura 17 mostra o resultado exibido na segunda página e no terminal.

Jaime
Eduardo
<input type="button" value="Save"/>

```

10:38:48,816 INFO [stdout] (default task-1) Jaime
10:38:48,816 INFO [stdout] (default task-1) Eduardo

```



```
11:02:47,208 INFO [stdout] (default task-1) Jaime
11:02:47,209 INFO [stdout] (default task-1) Eduardo
11:02:47,394 INFO [stdout] (default task-1) criou comunicacao 2
11:02:47,394 INFO [stdout] (default task-1) Jaime
11:02:47,395 INFO [stdout] (default task-1) Eduardo
```

## Atributos via Parametro

Definindo os dados como parametros através de um botão ou link é possível enviar esses dados sem a necessidade do return no método. A Figura 18 mostra a página com o novo botão e um link, Figura 19 mostra os resultados ao pressionar o botão e a Figura 20 mostra os resultados ao pressionar o link.

Maratona JSF - Aula 24: Comunicando entre componentes

localhost:8080/maratona\_jsf\_war\_expl

← → C ⓘ localhost:8080/maratona\_jsf\_wa

Apps Facebook Quadrinhos Animes

C:\var

[Vai Para Comunicacao 2](#)

Maratona JSF - Aula 24: Comunicando entre componentes

localhost:8080/maratona\_jsf\_war\_expl

← → C ⓘ localhost:8080/maratona\_jsf\_war\_expl

Apps Facebook Quadrinhos Animes

Maratona JSF - Aula 24: Comunicando entre componentes

localhost:8080/maratona\_jsf\_war\_expl

← → C ⓘ localhost:8080/maratona\_jsf\_war\_expl

Apps Facebook Quadrinhos Animes

## Omnifaces

Para fazer a classe que 'ComunicacaoTest2Bean' receber o nome e sobrenome da classe 'ComunicacaoTeste1Bean' foi utilizado inicialmente um mapping, mas também é possível fazer utilizando o Omnifaces, pra isso foi feito o download da dependencia no arquivo pom.xml. Figura 21 mostra o código e a Figura 22 mostra as alterações feitas na classe.

```
<!-- https://mvnrepository.com/artifact/org.omnifaces/omnifaces -->
<dependency>
    <groupId>org.omnifaces</groupId>
    <artifactId>omnifaces</artifactId>
    <version>2.6.2</version>
</dependency>
```

```
@Inject @Param(name = "nome")
private String nome;
@Inject @Param(name = "sobrenome")
private String sobrenome;

@PostConstruct
public void init(){
    System.out.println("criou comunicacao 2");
    Map<String, String> paramsMap = FacesContext.getCurrentInstance().getExternalContext().getParameters();
    nome = paramsMap.get("nome");
    sobrenome = paramsMap.get("sobrenome");
    System.out.println(nome);
    System.out.println(sobrenome);
}
```

Com essas mudanças é possível obter os mesmos resultados sem utilizar do maping na função init.

## Aula 25

Na aula 25 foi utilizado o view param.

### Nova Página

Para utilizar o view param foi criada uma nova página xhtml chamada 'comunicacao3' e uma nova classe em java chamada 'ComunicacaoTeste3Bean'. O view param é utilizado para receber os valores, exibi-los na tela e carregá-los as variaveis da classe. Figura 1 mostra o código da classe e Figura 2 mostra o código da página xhtml.

```
@ViewScoped
public class ComunicacaoTeste3Bean implements Serializable {
    private String nome;
    private String sobrenome;

    @PostConstruct
    public void init(){
        System.out.println("criou comunicacao 3");
        System.out.println(nome);
        System.out.println(sobrenome);
    }

    public String getNome() { return nome; }

    public void setNome(String nome) { this.nome = nome; }

    public String getSobrenome() { return sobrenome; }

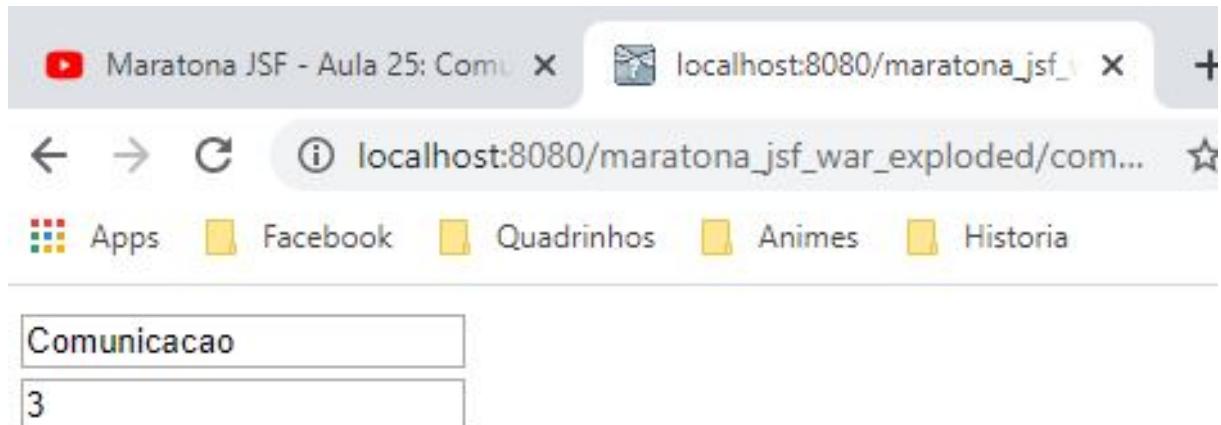
    public void setSobrenome(String sobrenome) { this.sobrenome = sobrenome; }
}
```

```
<h:head>

</h:head>
<f:metadata>
    <f:viewParam name="nome" value="#{comunicacaoTeste3Bean.nome}" />
    <f:viewParam name="sobrenome" value="#{comunicacaoTeste3Bean.sobrenome}" />
</f:metadata>
<h:body>
    <h:form>
        <h:panelGrid columns="1">
            <h:inputText value="#{comunicacaoTeste3Bean.nome}" />
            <h:inputText value="#{comunicacaoTeste3Bean.sobrenome}" />
        </h:panelGrid>
    </h:form>
</h:body>
```

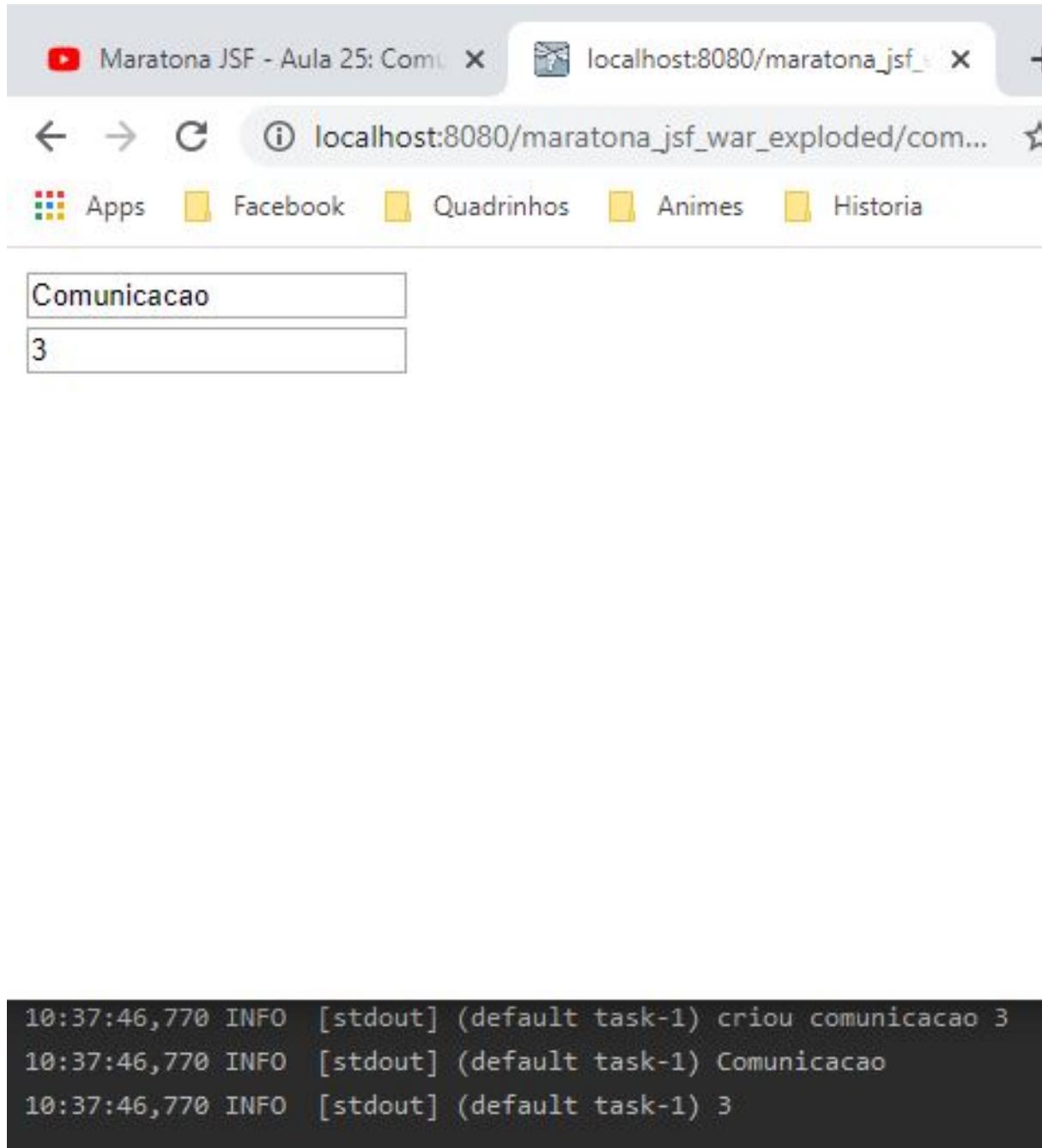
## Carregando os Valores na Classe

Utilizando apenas o view param é possível exibir os resultados na pagina 'comunicacao3' porem os valores não são carregados na classe, isso acontece porque os valores são passados antes do post construct da classe. Figura 3 mostra o resultado inicial.



```
10:32:50,748 INFO [stdout] (default task-1) criou comunicacao 3
10:32:50,749 INFO [stdout] (default task-1) null
10:32:50,750 INFO [stdout] (default task-1) null
```

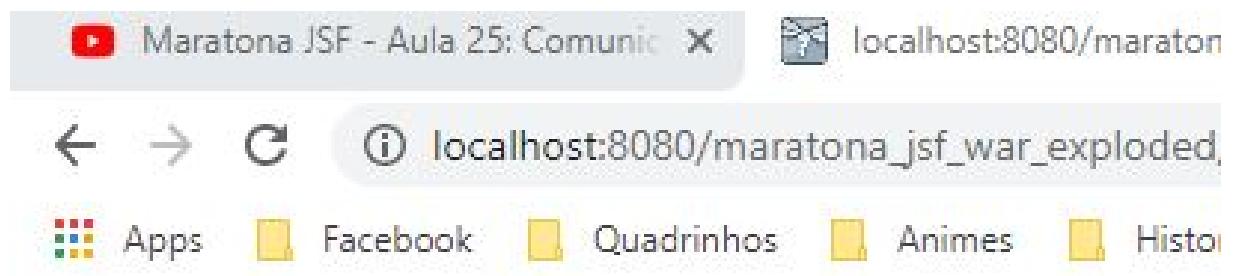
Para fazer os dados passarem para a classe 'ComunicacaoTeste3Bean' foi preciso informar ao view param que antes da renderização da página é preciso chamar a função init da classe. Figura 4 mostra o resultado.



## Passando os Parâmetros para um Nova Página

Na classe 'ComunicacaoTeste3Bean' foi criada uma função chamada save que retorna uma string que inclui os parâmetros do view a uma outra página chamada 'resultado'. Figura 5 mostra o código da função save e Figura 6 mostra o resultado exibido na página.

```
public String save(){
    System.out.println("Salvando");
    return "resultado?faces-redirect=true&includeViewParams=true";
}
```

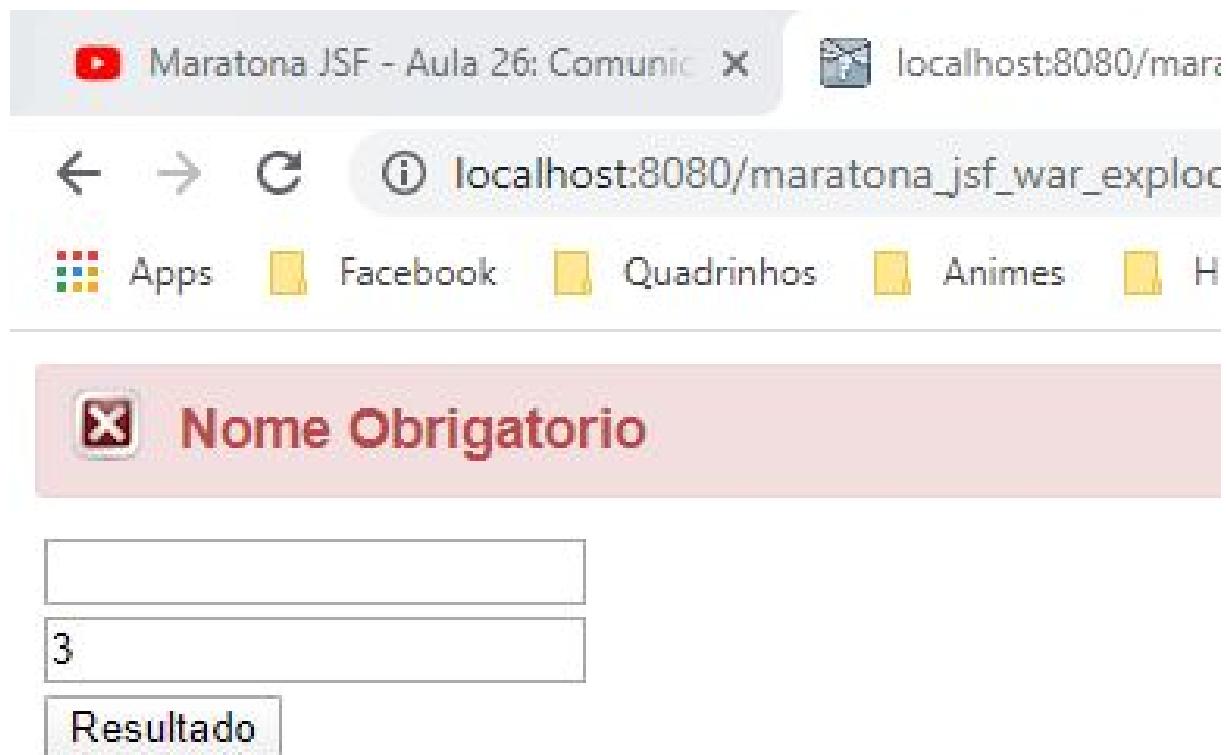


## Aula 26

A aula 26 dá continuidade ao uso de view param.

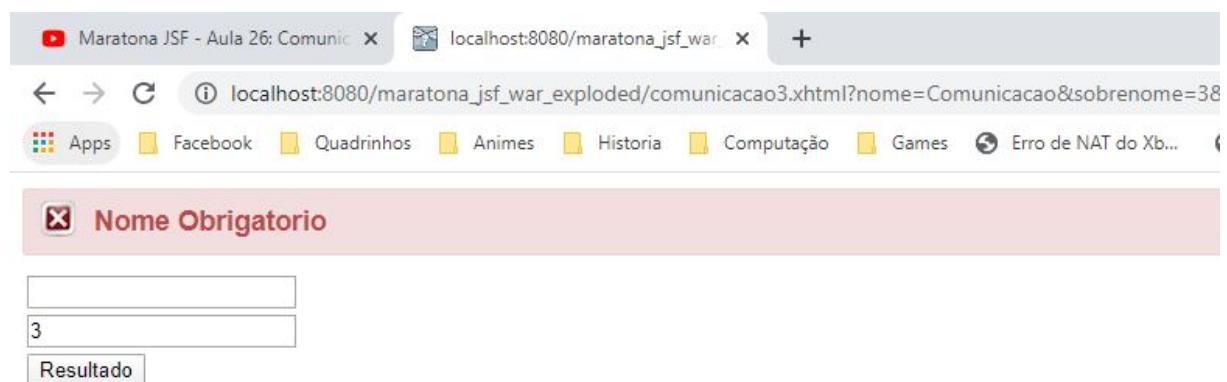
### Campo Obrigatório

O campo nome da página 'comunicacao3' foi modificado para que seja obrigatório e caso esteja vazio uma mensagem de erro é exibida, porém quando a mensagem é exibida os dados na url são perdidos. Para impedir essa perda foi utilizado o omnifaces para preservar os dados. Figura 7 mostra o resultado com perda de dados e Figura 8 mostra o resultado mantendo os dados.



The screenshot shows a web browser window with the following details:

- Address Bar:** Maratona JSF - Aula 26: Comunicacao3.xhtml → localhost:8080/maratona\_jsf\_war\_exploded/comunicacao3.xhtml
- Toolbar:** Back, Forward, Stop, Home, Refresh, Info icon, Address bar with URL, and several icons for apps like Facebook, Quadrinhos, Animes, Historia, Computação, Games, and Erro de NAT do Xb...
- Content Area:**
  - Title:** Nome Obrigatorio
  - Inputs:** Two empty input fields stacked vertically.
  - Button:** A large blue button labeled "Resultado".



The screenshot shows a web browser window with the following details:

- Address Bar:** Maratona JSF - Aula 26: Comunicacao3.xhtml → localhost:8080/maratona\_jsf\_war\_exploded/comunicacao3.xhtml?nome=Comunicacao&sobrenome=38
- Toolbar:** Back, Forward, Stop, Home, Refresh, Info icon, Address bar with URL, and several icons for apps like Facebook, Quadrinhos, Animes, Historia, Computação, Games, and Erro de NAT do Xb...
- Content Area:**
  - Title:** Nome Obrigatorio
  - Inputs:** Two input fields. The top one is empty, and the bottom one contains the value "3".
  - Button:** A large blue button labeled "Resultado".

### Requisição Ajax

Para fazer com que a mensagem de nome obrigatório seja exibida sem precisar clicar no botão para ir para a próxima página foi utilizada uma requisição ajax, porém toda vez que a mensagem é exibida a função init da classe 'ComunicacaoTeste3Bean' é executada para evitar isso foi utilizado o view action. Figura 9 mostra o código.

```
<f:viewAction action="#{comunicacaoTeste3Bean.init()}" />
</f:metadata>
<h:body>
    <o:form includeRequestParams="true">
        <p:messages autoUpdate="true"/>
        <h:panelGrid columns="1">
            <h:inputText value="#{comunicacaoTeste3Bean.nome}" required="true" re
                <f:ajax event="blur"/>
            </h:inputText>
        </h:panelGrid>
    </o:form>
</h:body>
```

## Aula 27

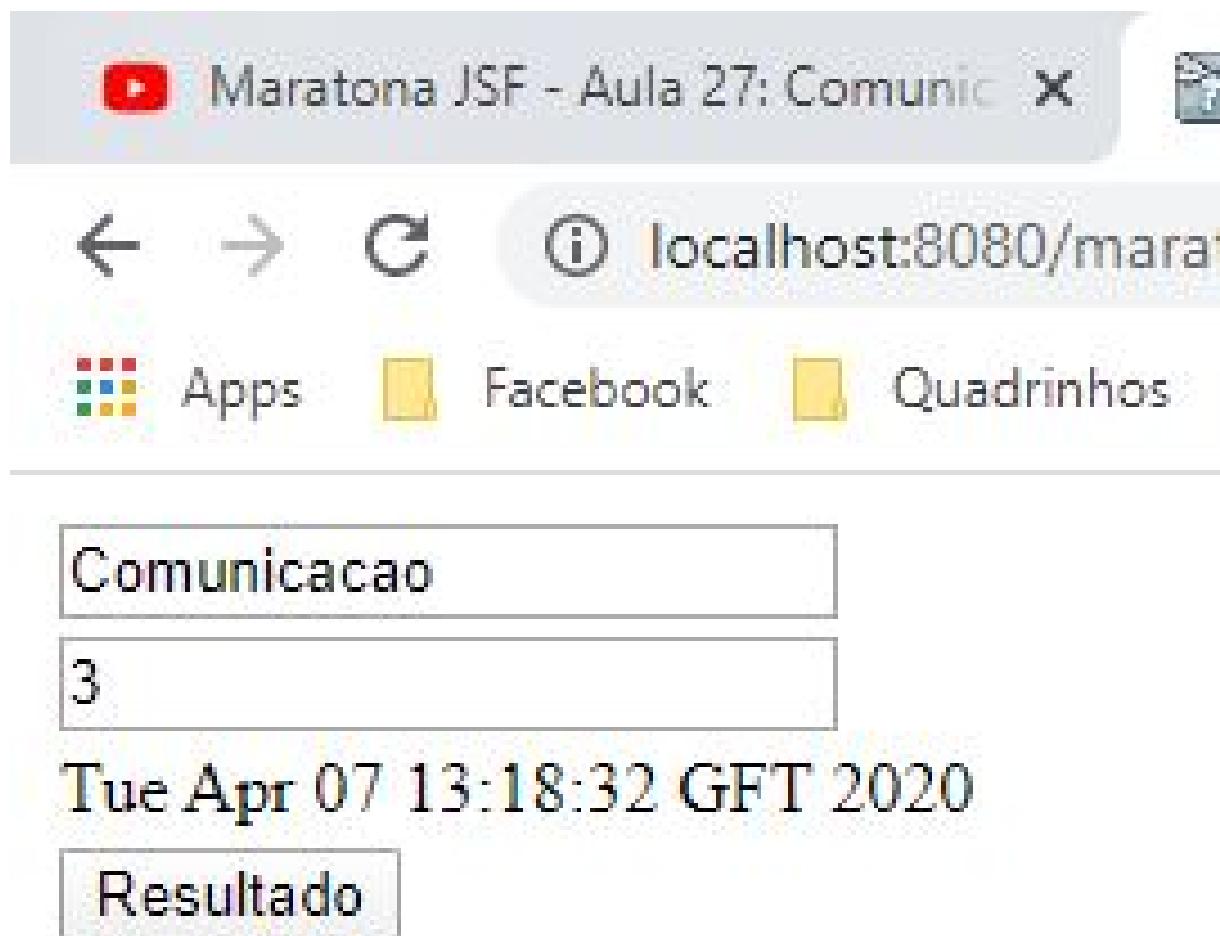
A aula 27 foi dado continuidade ao uso do view param.

### Data

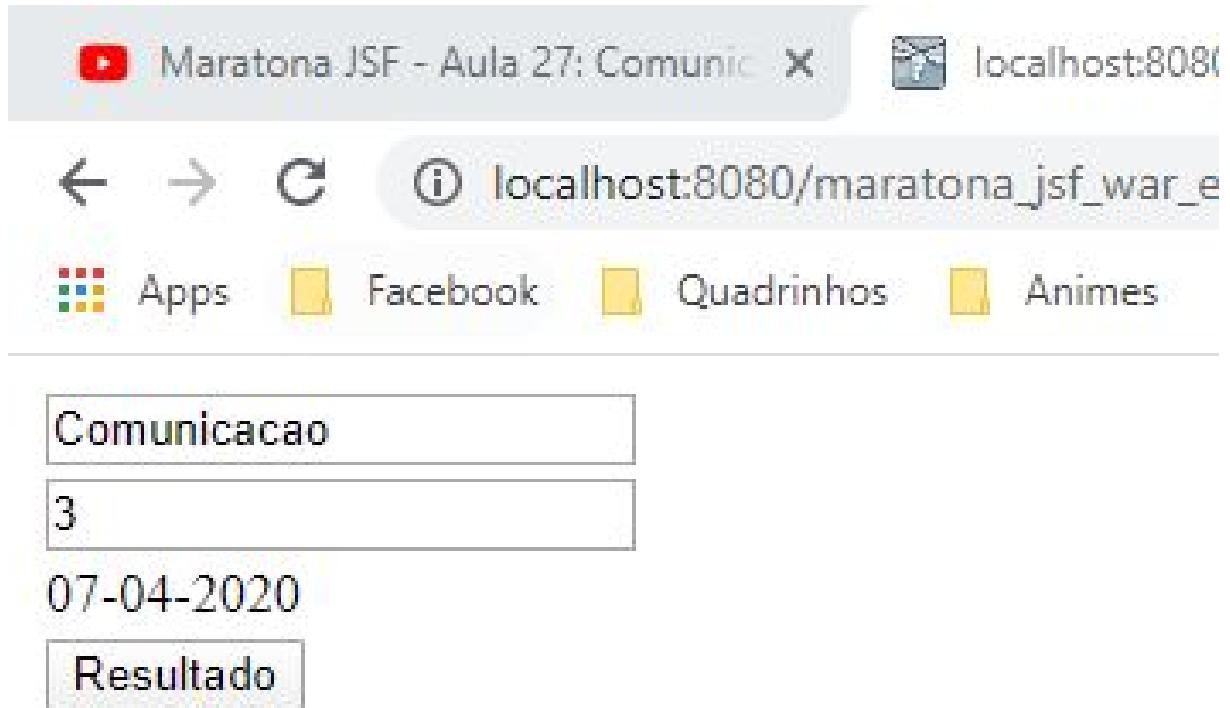
Na página 'comunicacao.xhtml' foi passado mais um parametro através do view param, dessa vez do tipo Date, para a página 'comunicacao3.xhtml'. Figura 1 mostra o código.

```
<f:param name="data" value="#{of:formatDate[comunicacaoTeste1Bean.data,'dd-MM-yyyy']}"/>
```

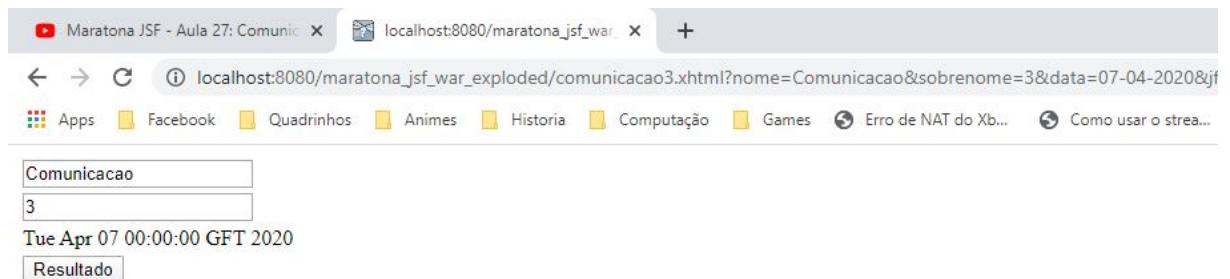
Após passar o dado ele é exibido na página como uma string, a Figura 2 mostra o resultado.



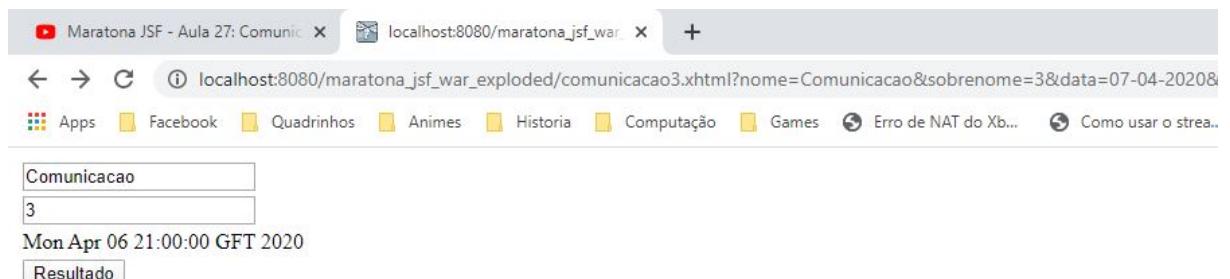
Para transformar a forma como o dado é exibido foi utilizado o SimpleDateFormat para exibir a data no padrão dia, mês e ano. Figura 3 mostra o resultado.



Através do parsing foi possível manter os dois tipos de dados, com a data no padrão dd-MM-yyyy sendo exibido na url e a data com informações mais detalhadas sendo exibida na página. Figura 4 mostra o resultado



Também é possível realizar essa tarefa utilizando o omnifaces que obtém o mesmo resultado com menos linhas de código. Figura 5 mostra o resultado.



## Aula 28

Na aula 28 foi utilizado o attribute para passar dados para a classe.

### Nova Classe e Página

Para a aula 28 foi criada uma nova classe chamada 'ComunicacaoTeste4Bean' e uma nova página chamada 'comunicacao4'. Figura 6 mostra o código da classe e Figura 7 mostra o código da página xhtml.

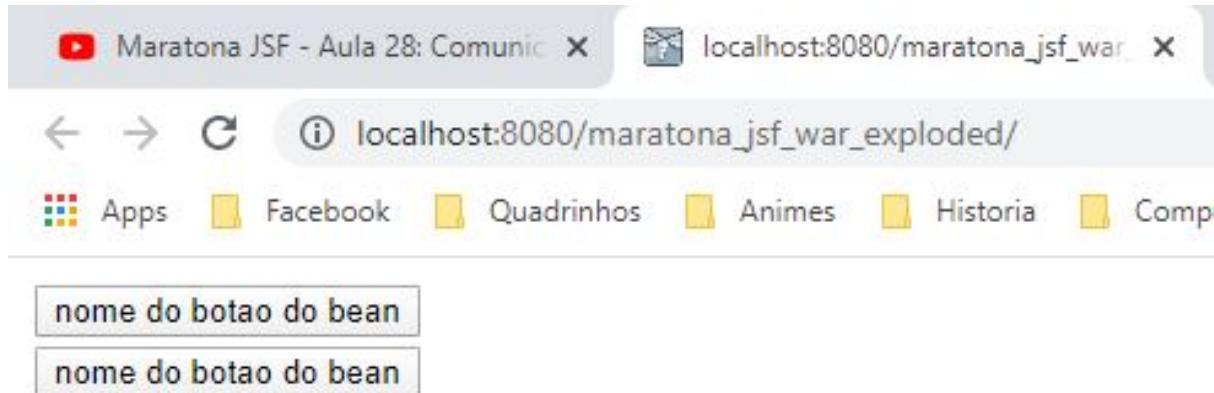
```
import javax.faces.view.ViewScoped;
import javax.inject.Named;
import java.io.Serializable;

@Named
@ViewScoped
public class ComunicacaoTeste4Bean implements Serializable {
    private String buttonName = "nome do botao do bean";

    public void execute(){
    }
}
```

```
<o:form includeRequestParams="true">
    <p:messages autoUpdate="true"/>
    <h:panelGrid columns="1">
        <h:commandButton actionListener="#{comunicacaoTeste4Bean.execute()}">
            <f:attribute value="#{comunicacaoTeste4Bean.buttonName}" name="value"/>
        </h:commandButton>
        <h:commandButton value="#{comunicacaoTeste4Bean.buttonName}"
                        actionListener="#{comunicacaoTeste4Bean.execute()}">
    </h:panelGrid>
</o:form>
```

Na página é criado o botão que possui o nome de uma das variáveis da classe. Figura 8 mostra o resultado.



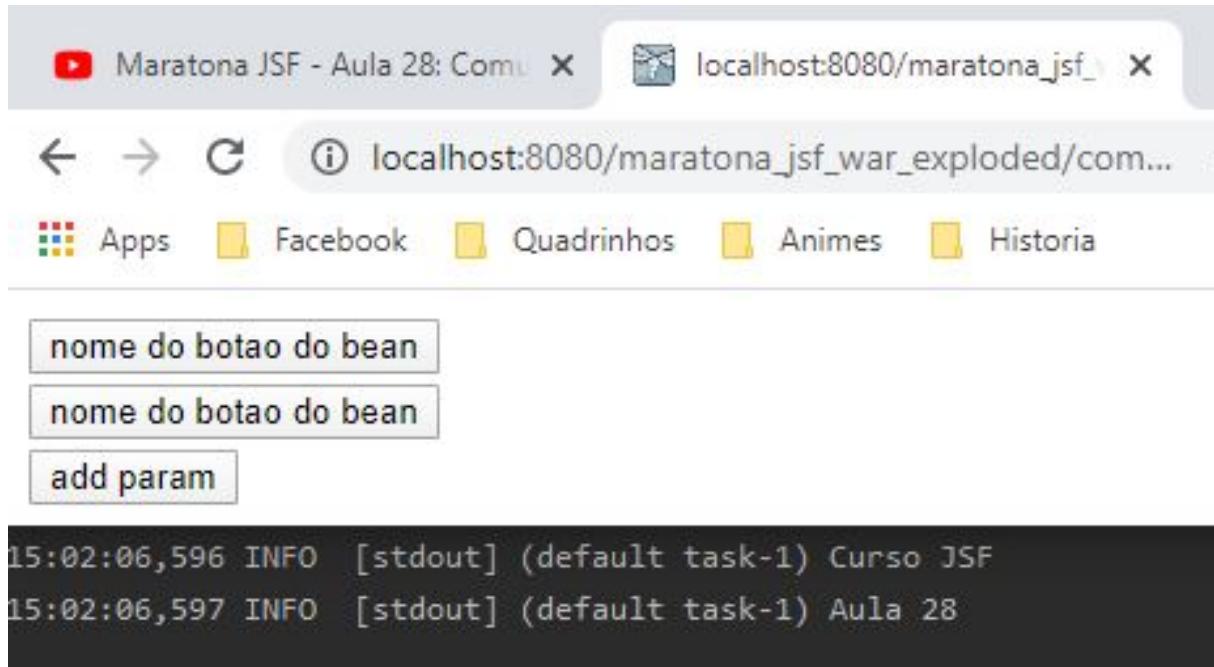
## Função execute

Para passar as variáveis da página foi utilizado attribute, que passa duas strings que são recebidas e exibidas no terminal através da função execute, para a função execute receber esses ela precisa receber um parâmetro do tipo ActionEvent. Figura 9 mostra o código da função e a Figura 10 mostra os dados que serão enviados.

```
public void execute(ActionEvent event){  
    String nome = (String) event.getComponent().getAttributes().get("Nome");  
    String aula = (String) event.getComponent().getAttributes().get("Aula");  
    System.out.println(nome);  
    System.out.println(aula);  
}
```

```
<h:commandButton value="add param" actionListener="#{comunicacaoTeste4Bean.execute}">  
    <f:attribute name="Nome" value="Curso JSF"/>  
    <f:attribute name="Aula" value="Aula 28"/>  
</h:commandButton>
```

Os dados serão passados e exibidos no terminal após o botão da página ser pressionado. Figura 11 mostra o resultado no terminal.



## Passando o Objeto Estudante

Também é possível passar valores do objeto estudante utilizando attribute. Figura 12 mostra o código adicionando o tipo estudante a função execute e a Figura 13 mostra a alteração no arquivo xhtml.

```
private Estudante estudante = new Estudante();

public void execute(ActionEvent event){
    String nome = (String) event.getComponent().getAttributes().get("Nome");
    String aula = (String) event.getComponent().getAttributes().get("Aula");
    Estudante estudante = (Estudante) event.getComponent().getAttributes().get("Estudante");
    System.out.println(nome);
    System.out.println(aula);
    System.out.println(estudante.getNome());
}

<f:attribute name="Estudante" value="#{comunicacaoTeste4Bean.estudante}" />
```

Após as alterações o servidor foi reiniciado, o botão foi mais uma vez pressionado e foi exibido no terminal o nome do estudante. Figura 14 mostra o resultado no terminal.

```
15:09:33,541 INFO [stdout] (default task-1) Curso JSF
15:09:33,542 INFO [stdout] (default task-1) Aula 28
15:09:33,542 INFO [stdout] (default task-1) Jaime
```

## Aula 29

Na aula 29 foi utilizado o setPropertyActionListener para poder passar valores pra classe.

### Nome e Sobrenome

Foi criada uma classe chamada 'ComunicacaoTeste5Bean' com dois atributos nome e sobrenome, também foi criada uma página chamada 'comunicacao5' que utiliza o setPropertyActionListener para enviar duas string para a classe. A Figura 1 mostra o código da classe, a Figura 2 mostra o código da página e a Figura 3 mostra o resultado.

```
@ViewScoped
public class ComunicacaoTeste5Bean implements Serializable {
    private String nome;
    private String sobrenome;

    public void execute(){}
}

public String getName() {
    return nome;
}

public void setName(String nome) {
    System.out.println("veio do set property action listener"+nome);
    this.nome = nome;
}
```

```
<h:commandButton value="Property Action Listener" >
    <f:setPropertyActionListener target="#{comunicacaoTeste5Bean.nome}" value="Jaime Eduardo"/>
    <f:setPropertyActionListener target="#{comunicacaoTeste5Bean.sobrenome}" value="Padilla Aranha"/>
</h:commandButton>
```

### Property Action Listener

```
task-1) veio do set property action listenerJaime Eduardo
task-1) veio do set property action listenerPadilla Aranha
```

### Função execute

Na função execute foi escrito para o nome e o sobrenome serem exibidos no terminal. A função execute é chamada através de um ActionListener. A Figura 4 mostra o código da função execute e a Figura 5 mostra o resultado no terminal.

```

public void execute(){
    System.out.println("Dentro do execute" + nome);
    System.out.println("Dentro do execute" + sobrenome);
}

```

```

13:47:48,453 INFO [stdout] (default task-1) Dentro do executenull
13:47:48,453 INFO [stdout] (default task-1) Dentro do executenull
13:47:48,456 INFO [stdout] (default task-1) veio do set property action listenerJaime Eduardo
13:47:48,457 INFO [stdout] (default task-1) veio do set property action listenerPadilla Aranha

```

É possível observar que o resultado é null, isso se deve ao fato de que os valores só são atribuídos depois do actionPerformed, para corrigir esse erro é preciso utilizar o action. Figura 6 mostra o resultado utilizando action.

```

13:49:49,010 INFO [stdout] (default task-1) veio do set property action listenerJaime Eduardo
13:49:49,011 INFO [stdout] (default task-1) veio do set property action listenerPadilla Aranha
13:49:49,012 INFO [stdout] (default task-1) Dentro do executeJaime Eduardo
13:49:49,012 INFO [stdout] (default task-1) Dentro do executePadilla Aranha

```

## Passando um Objeto

Utilizando o setPropertyActionListener também é possível passar um objeto, para realizar esse teste foram criadas dois estudantes na classe 'ComunicacaoTeste5Bean', sendo um deles vazio. A Figura 7 mostra o código da classe, a Figura 8 mostra o código na página e a Figura 9 mostra o resultado no terminal.

```

private Estudante estudante = new Estudante();
private Estudante estudante2;

public void execute(){
    System.out.println("Dentro do execute" + nome);
    System.out.println("Dentro do execute" + sobrenome);
    System.out.println("Dentro do execute" + estudante2.getTurno());
    System.out.println("Dentro do execute" + estudante2.getNome());
}

```

```
PropertyActionListener target="#{comunicacaoTeste5Bean.estudante2}" value="#{comunicacaoTeste5Bean.estudante}"
```

```

13:55:05,811 INFO [stdout] (default task-1) Dentro do executeJaime Eduardo
13:55:05,811 INFO [stdout] (default task-1) Dentro do executePadilla Aranha
13:55:05,812 INFO [stdout] (default task-1) Dentro do executeMATUTINO
13:55:05,812 INFO [stdout] (default task-1) Dentro do executeJaime

```

## Aula 30

Na aula 30 é feita a passagem de atributos entre beans através do flash scope.

### Modificando a Classe Estudante

Para o inicio da atividade foi adicionada uma lista de estudantes na classe estudante fornecendo novos dados. Figura 10 mostra o código.

```
public Estudante(String nome, String sobrenome, double nota1) {
    this.nome = nome;
    this.sobrenome = sobrenome;
    this.nota1 = nota1;
}

public static List<Estudante> estudanteList() {
    return asList(new Estudante( nome: "Jaime", sobrenome: "Eduardo", nota1: 9),
        new Estudante( nome: "Miguel", sobrenome: "Xavier", nota1: 7),
        new Estudante( nome: "Diego", sobrenome: "Alexandre", nota1: 8));
}
```

Foi criada uma página chamada 'comunicacao6' e uma classe chamada 'ComunicacaoTeste6Bean', que recebe a lista de estudantes. Na página xhtml foi criado um laço de repetição que exibi os dados contidos na lista. Figura 11 mostra o código da página, Figura 12 mostra o código da classe e Figura 13 mostra o resultado.

```
<ui:repeat value="#{comunicacaoTeste6Bean.estudanteList}"
    var="estudante" varStatus="status">
    <h:outputText value="#{status.index}"/>&nbsp;
    <h:outputText value="#{estudante.nome}"/>&nbsp;
    <h:outputText value="#{estudante.sobrenome}"/>&nbsp;
    <h:outputText value="#{estudante.nota1}"/>&nbsp;
    <h:commandLink value="editar" action="#{comunicacaoTeste6Bean.editar(status.index)}"/><br/>
</ui:repeat>
```

```
@Named
public class ComunicacaoTeste6Bean implements Serializable {
    private List<Estudante> estudanteList = Estudante.estudanteList();

    public String editar(int index){
        Estudante estudante = estudanteList.get(index);
        return "";
    }
}
```



### Passando Objeto de um Bean Para Outro

Foi criada uma classe chamada 'ComunicacaoTeste7Bean' e uma página chamada 'comunicacao7' que recebem os dados de um dos estudantes selecionados pelo link editar, isso é feito utilizando flash. Figura 14 mostra a modificação feita na classe 'ComunicacaoTeste6Bean', Figura 15 mostra a classe nova, Figura 16 mostra a página nova e Figura 17 mostra o resultado.

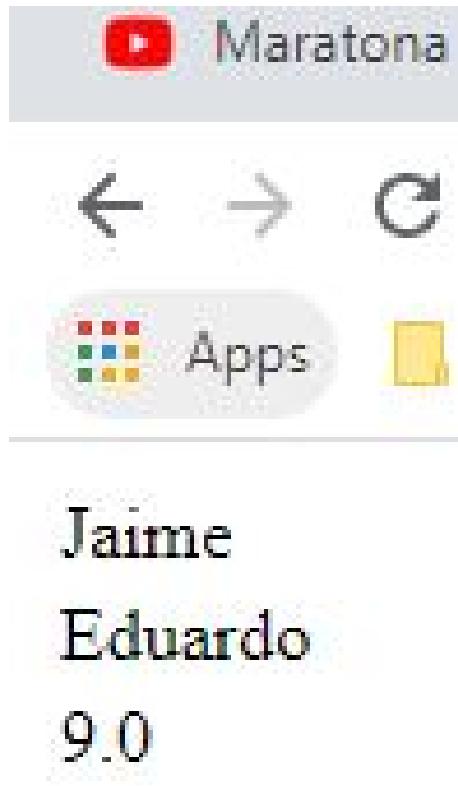
```
public String editar(int index){  
    Estudante estudante = estudanteList.get(index);  
    Flash flash = FacesContext.getCurrentInstance().getExternalContext().getFlash();  
    flash.put("estudante",estudante);  
    return "comunicacao7?faces-redirect=true";  
}
```

```
@Named  
@ViewScoped  
public class ComunicacaoTeste7Bean implements Serializable {  
    private Estudante estudante;  
  
    public void init(){  
        Flash flash = FacesContext.getCurrentInstance().getExternalContext().getFlash();  
        estudante = (Estudante) flash.get("estudante");  
    }  
}
```

```

<f:metadata>
    <f:viewAction action="#{comunicacaoTeste7Bean.init()}" />
</f:metadata>
<h:body>
    <o:form includeRequestParams="true">
        <p:messages autoUpdate="true"/>
        <h:panelGrid columns="1">
            <h:outputText value="#{comunicacaoTeste7Bean.estudante.nome}" />
            <h:outputText value="#{comunicacaoTeste7Bean.estudante.sobrenome}" />
            <h:outputText value="#{comunicacaoTeste7Bean.estudante.nota1}" />
        </h:panelGrid>
    </o:form>

```



## Mantendo os Dados

Apesar da transmissão de dados ser bem sucedida caso a página seja atualizada os dados são perdidos, para poder manter os dados foi feita uma adição no código do arquivo 'comunicacao7'. Figura 18mostra o código.

```

<h:inputHidden value="#{flash.keep}" />
<h:inputHidden value="#{flash.estudante}" />

```

Dessa forma os dados são mantidos na página independentemente dela ter sido atualizada.

# Aula 31

Na aula 31 é ensinado o uso de cookies.

## Exibindo os Cookies Existentes

Inicialmente foi criada uma classe chamada 'ComunicacaoTeste8Bean' e um arquivo xhtml chamado 'comunicacao8'. O primeiro teste busca exibir os cookies existentes no terminal. Figura 1 mostra o código da classe e Figura 2 mostra o resultado no terminal.

```
class ComunicacaoTeste8Bean implements Serializable {  
    public void init(){  
        Map<String, Object> requestCookieMap = FacesContext.getCurrentInstance().getExternalContext().getRequestCookieMap();  
        System.out.println(requestCookieMap);  
    }  
}
```

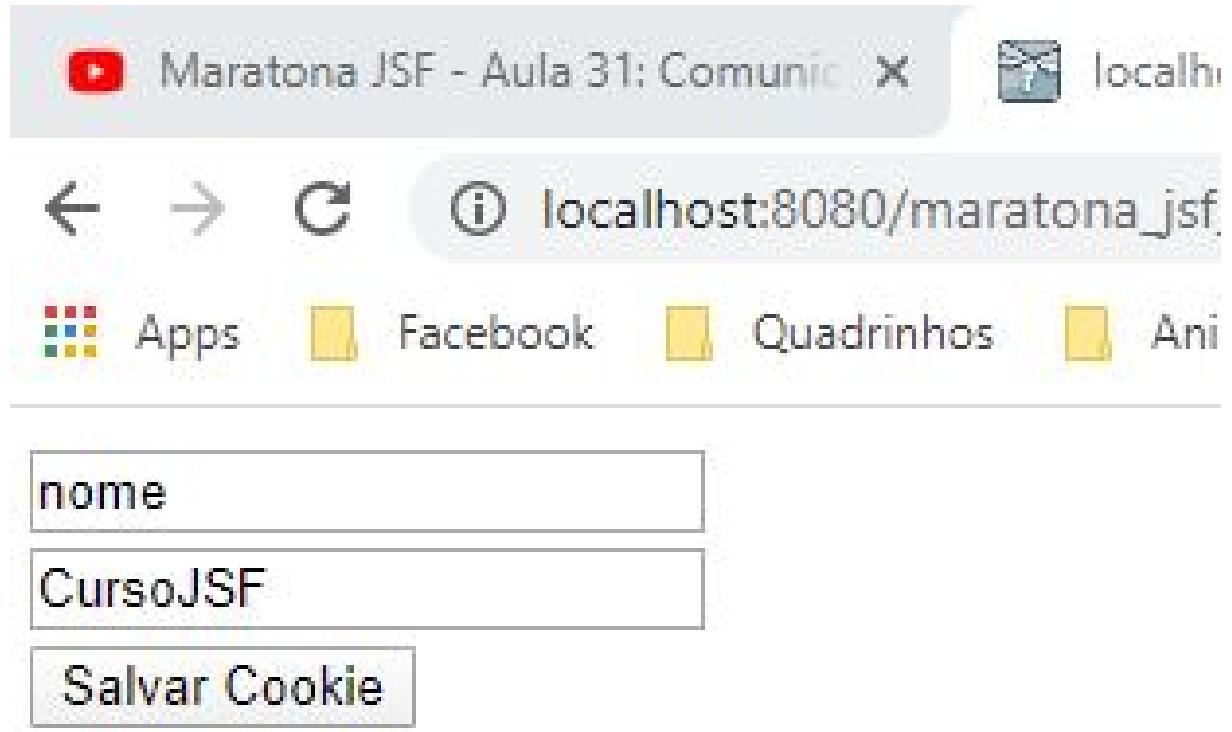
```
14:07:59,750 INFO [stdout] (default task-1) {JSESSIONID=javax.servlet.http.Cookie@6ecd0f43}
```

## Criando Cookies

Foi adicionado a classe uma função para salvar cookies, os dados a serem salvos foram inseridos na página. Figura 3 mostra o código da função, Figura 4 mostra o código da página e Figura 5 mostra o resultado na porta 8080.

```
private String key;  
private String value;  
  
public void init(){  
    Map<String, Object> requestCookieMap = FacesContext.getCurrentInstance().getRequestCookieMap();  
    System.out.println(requestCookieMap);  
}  
  
public void salvarCookie(){  
    Faces.addResponseCookie(key,value, maxAge: -1);  
}
```

```
<h:inputText required="true" value="#{comunicacaoTeste8Bean.key}" />  
<h:inputText required="true" value="#{comunicacaoTeste8Bean.value}" />  
<h:commandButton value="Salvar Cookie" actionListener="#{comunicacaoTeste8Bean.salvarCookie()}" />
```



Após pressionar o botão Salvar Cookie e reiniciar a página é possível ver um novo cookie sendo exibido no terminal. Figura 6 mostra o resultado no terminal.

```
{JSESSIONID=javax.servlet.http.Cookie@5583cde9}  
{JSESSIONID=javax.servlet.http.Cookie@34a0d393, nome(javax.servlet.http.Cookie@c46e81e)}
```

### Mostrando Conteúdo do Cookie

Para poder exibir o conteúdo do cookie foi adicionado código a função init e o resultado é exibido no terminal. Figura 7 mostra o código e Figura 8 mostra o resultado no terminal.

```
Cookie cookie = (Cookie) requestCookieMap.get("nome");  
System.out.println(cookie.getValue());
```

```
(default task-1) CursoJSF+Aula+31
```

Para poder mostrar o conteúdo de forma limpa sem estar no formato de url é preciso utilizar o URLDecoder. Figura 9 mostra o código e Figura 10 mostra o resultado.

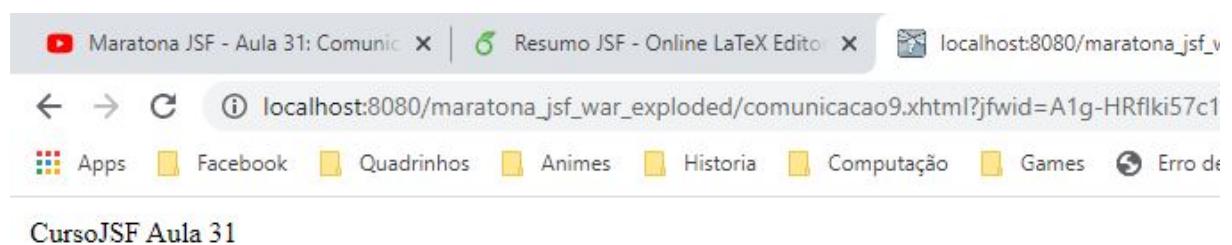
```
Cookie cookie = (Cookie) requestCookieMap.get("nome");
try {
    String decode = URLDecoder.decode(cookie.getValue(), enc: "UTF-8");
    System.out.println(decode);
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
```

```
4:47:09,855 INFO [stdout] (default task-1) CursoJSF Aula 31
```

Utilizando o omnifaces é possível obter o mesmo resultado utilizando bem menos linhas de código. Figura 11 mostra o código utilizando o omnifaces.

```
System.out.println(Faces.getRequestCookie( name: "nome" ));
```

Foi criada uma nova página para exibir o valor do cookie chamada 'comunicacao9'. Figura 12 mostra o resultado.



## Aula 32

Na aula 32 foi trabalhado como esconder o texto para senhas e como chamar funções com argumentos.

### Salvar

Para esse aula foi criada uma classe chamada 'ComunicacaoTeste10' e uma página chamada comunicacao10. Figura 13 mostra o código da classe e Figura 14 mostra o código da página para o uso da função salvar.

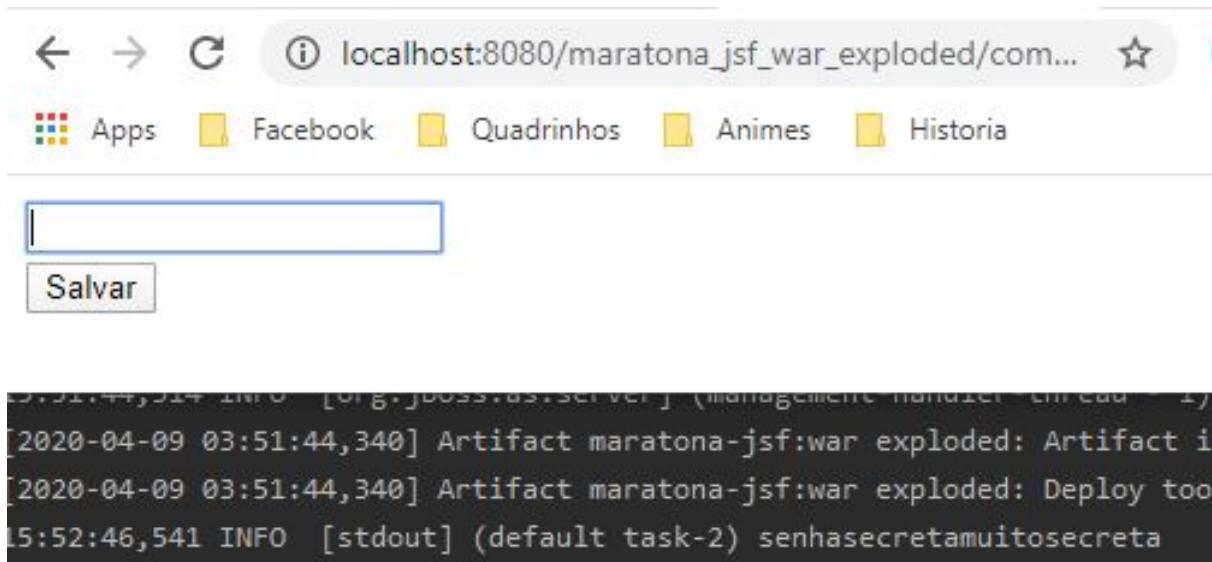
```
public class ComunicacaoTeste10Bean implements Serializable {
    private String password;
    private List<Estudante> estudanteList = Estudante.estudanteList();

    public void remover(Estudante estudante){
        estudanteList.remove(estudante);
    }

    public void salvar(){
        System.out.println(password);
    }
}
```

```
<o:form includeRequestParams="true">
    <p:messages autoUpdate="true"/>
    <h:panelGrid columns="1">
        <h:inputSecret required="true" value="#{comunicacaoTeste10Bean.password}" />
        <h:commandButton value="Salvar" actionListener="#{comunicacaoTeste10Bean.salvar()}" />
    </h:panelGrid>
```

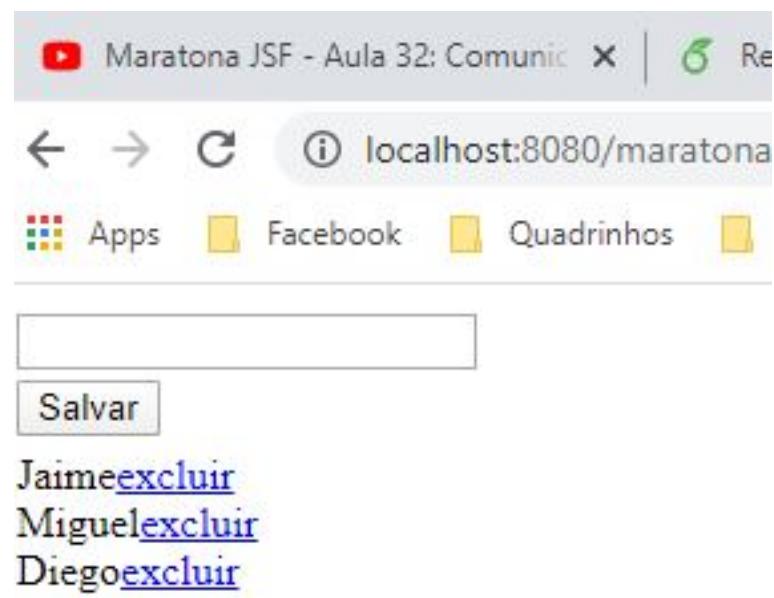
Para escrever um texto de forma secreta foi utilizado o inputSecret e ao pressionar o botão salvar é possível ver o conteúdo no terminal. Figura 15 mostra o resultado.



### Remover

Para Remover os nomes da lista de estudantes presente na classe foi criada a função remover que recebe um dos estudantes como parâmetro e o remove da lista. Figura 16 mostra o código da página para chamar a função e Figura 17 mostra o resultado.

```
<o:form includeRequestParams="true">
    <h:panelGrid columns="2">
        <ui:repeat value="#{comunicacaoTeste10Bean.estudanteList}" var="estudante">
            <h:outputText value="#{estudante.nome}"/>
            <h:commandLink value="excluir" actionListener="#{comunicacaoTeste10Bean.remover(estudante)}/>
        </ui:repeat>
    </h:panelGrid>
</o:form>
```



## Aula 33

A aula 33 trabalhou com a adição de objetos na sessão de forma dinâmica.

### Application Map Util

Para realizar a adição de objetos na sessão foi criada uma classe chamada 'ApplicationMapUtil'. Figura 1 mostra o código.

```
public class ApplicationMapUtil {  
    public static Object getValueFromApplicationMap(String key){  
        return FacesContext.getCurrentInstance().getExternalContext().getApplicationMap().get(key);  
    }  
  
    public static void setValueInApplicationMap(String key, Object value){  
        FacesContext.getCurrentInstance().getExternalContext().getApplicationMap().put(key, value);  
    }  
}
```

### Exibir Nome

Para realizar o teste duas novas classes foram criadas, 'ComunicacaoTeste11Bean' e 'ComunicacaoTeste12Bean', além delas duas páginas xhtml também foram criadas, uma para exibir a lista de estudantes e outra para exibir algumas informações quando o objeto for selecionado. Figuras 2 e 3 mostram os códigos das classes e Figuras 4 e 5 mostram os códigos das páginas.

```
@Named  
@ViewScoped  
public class ComunicacaoTeste11Bean implements Serializable {  
    private List<Estudante> estudanteList = Estudante.estudanteList();  
  
    public String editar(Estudante estudante){  
  
        ApplicationMapUtil.setValueInApplicationMap("estudante", estudante);  
        return "comunicacao12?faces-redirect=true";  
    }  
}
```

```
@Named  
@RequestScoped  
public class ComunicacaoTeste12Bean implements Serializable {  
    Estudante estudante;  
  
    public void init(){  
        estudante = (Estudante) ApplicationMapUtil.getValueFromApplicationMap( key: "estudante");  
    }  
}
```

```

<h:body>
    <o:form includeRequestParams="true">
        <h:panelGrid columns="2">
            <ui:repeat value="#{comunicacaoTeste11Bean.estudanteList}" var="estudante">
                <h:outputText value="#{estudante.nome}"/>
                <h:commandLink value="editar" action="#{comunicacaoTeste11Bean.editar(estudante)}"/><br/>
            </ui:repeat>
        </h:panelGrid>
    </o:form>

```

```

<f:metadata>
    <f:viewAction action="#{comunicacaoTeste12Bean.init()}" />
</f:metadata>
<h:body>
    <h:outputText value="#{comunicacaoTeste12Bean.estudante.nome}"/><br/>
    <h:outputText value="#{comunicacaoTeste12Bean.estudante.sobrenome}"/>
</h:body>

```

## Resultados

A primeira página exibi uma lista de estudantes com a função para editar ao lado, ao clicar no link editar um redirecionamento é feito e o nome e sobrenome do objeto selecionado é exibido. Figura 6 mostra a primeira página e Figura 7 mostra a segunda página.

---

[Jaime](#)[editar](#)  
[Miguel](#)[editar](#)  
[Diego](#)[editar](#)

Jaime  
 Eduardo



## Aula 34

A aula 34 aborda a teoria do ciclo de vida do JSF.

### Nova Requisição

Ciclo de vida do JSF quando ocorre uma nova requisição.

- **Restore View:** Verifica se a requisição é nova ou já foi feita anteriormente.
- **Render Response:** Caso seja uma nova requisição as etapas intermediárias são ignoradas e a etapa render response é chamada que constroi a árvore de componentes e manda diretamente a resposta para o cliente.

### Requisição já Existente

Ciclo de vida do JSF quando a requisição já existe.

- **Restore View:** Verifica e confirma que requisição já existe
- **Apply Request Value:** Recebe todos os valores presente na árvore de componentes como request parameter.
- **Process Validation:** Todos os componentes que tiverem alguma forma de validação é executada. Caso validação falhe uma mensagem de erro é passada para o render response.
- **Update Model:** Realiza o set dos componentes aos beans.
- **Invoke Application:** Fase em que o formulário é submetido. Responsável por chamar os métodos diretamente dos beans. Caso o retorno do método seja void a próxima etapa é a render response, porém se for uma nova página isso significa que uma nova requisição foi feita o que reinicia o processo voltando ao restore view.

### Immediate = true

O immediate=true pode ser utilizado em um inputText ou commandButton e o uso dele altera o ciclo de JSF.

- **UIInput:** Caso o immediate=true seja utilizado nesse caso sua validação ocorre no apply request ao invés do process validation
- **UICommand:** Nesse caso todas as etapas após o apply request seriam ignoradas seguindo direto para o render response.

## Aula 35

Aula 35 fornece um exemplo para o uso de immediate = true.

### Immediate

Na página 'comunicacao12' é criado um botão para voltar, esse botão exige que a entrada do input text seja um número, logo irá falhar na validação, porém utilizando o immediate = true é possível passar pela etapa de validação e voltar para a página 'comunicacao11'. Figura 1 e 2 mostra os códigos sem e com o immediate=true.

```
<h:form>
    <h:inputText value="#{comunicacaoTeste12Bean.estudante2.nome}" converter="javax.faces.Number"/>
    <h:commandButton value="Voltar" action="comunicacao11?faces-redirect=true"/>
</h:form>
```

```
<h:form>
    <h:inputText value="#{comunicacaoTeste12Bean.estudante2.nome}" converter="javax.faces.Number"/>
    <h:commandButton value="Voltar" action="comunicacao11?faces-redirect=true" immediate="true"/>
</h:form>
```

Figura 3 mostra a mensagem de erro caso o immediate=true não seja utilizado.



j\_idt9:j\_idt10: 'Jaime' não é um número.

Jaime	Voltar
-------	--------

## Aula 36

A aula 36 trabalha com o uso de diferentes línguas para uma mesma página.

### Resource

Foram criadas duas variáveis em dois idiomas diferentes, português e inglês, para aplicar esses dois arquivos foi feito uma modificação no arquivo faces-config. Figura 4 mostra a configuração feita.

```
<application>
    <resource-bundle>
        <base-name>messages</base-name>
        <var>m</var>
    </resource-bundle>
</application>
```

### Menu de Seleção

Para criar um menu de seleção de língua foi adicionada uma lista do tipo locale a classe 'loginBean', Figura 5 mostra o código, e um selectOneMenu na página xhtml. Figura 6 mostra o código da página e Figura 7, 8 e 9 mostram os resultados.

```
private List<Locale> localeList = asList(new Locale( language: "en"), new Locale( language: "pt"));
private String language;
```

```
<h:selectOneMenu value="#{loginBean.language}">
    <f:selectItems value="#{loginBean.localeList}" var="locale"
                   itemValue="#{locale.language}" itemLabel="#{locale.displayLanguage}"/>
</h:selectOneMenu>
<h:commandButton value="Logar" action="#{loginBean.logar()}" />
```

The screenshot shows a login form with the following elements:

- Nome:** An input field containing the letter 'J'.
- Senha:** An input field containing a single dot ('.') as a placeholder.
- Language Selection:** A dropdown menu set to 'inglês'.
- Logar:** A red rectangular button labeled 'Logar'.

pt

# Usuário Logado: Jaime

[Logout](#)

en

# User Logged: Jaime

[Logout](#)

## Adição de Alertas

Utilizando o mesmo arquivo que determina a língua do nome do usuário é possível exibir um alerta que indica quantas vezes a página foi acessada. Figura 10 mostra o arquivo em português.

```
userLogged = Usuário Logado: {0} {1}
alert = Você tem {0} {0, choice,0#alertas|1#alerta|2#alertas}
```

A cada vez que a página é acessada uma variável do tipo int criada na classe 'loginBean' é incrementada e seu valor é exibido na página. Figura 11 mostra o resultado.

Usuário Logado: Jaime Eduardo

Você tem 2 alertas [Logout](#)

## Aula 37

A aula 37 trabalha com o uso de validação customizada.

### Classe para Validar e-mail

O primeiro exemplo desenvolvido na aula foi o de criar uma classe específica com uma forma de validação customizada, no exemplo da aula se trata de uma validação para e-mail. Figura 1 mostra o código da classe, Figura 2 mostra o código da página e Figura 3 mostra o código da classe de validação.

```
@Named  
@ViewScoped  
public class ValidatorTesteBean implements Serializable {  
    private Estudante estudante = new Estudante();  
  
    public void save(){  
        System.out.println("Salvando...");  
        System.out.println(estudante.getEmail());  
    }  
  
    public Estudante getEstudante() {  
        return estudante;  
    }  
  
    public void setEstudante(Estudante estudante) {  
        this.estudante = estudante;  
    }  
}
```

```
<h:body>  
    <p:messages autoUpdate="true"/>  
    <h:outputText value="E-mail"/><br/>  
    <h:form>  
        <h:inputText value="#{validatorTesteBean.estudante.email}"  
                    validator="#{validators.validateDuplicatedEmail}" size="50"/>  
        <h:commandButton value="Save"  
                        actionListener="#{validatorTesteBean.save()}" />  
    </h:form>  
</h:body>
```

```

@FacesValidator
public class DuplicatedEmailValidator implements Validator {
    private List<String> emailsDB = asList("jaimearanha96@gmail.com", " contato@devdojo.com.br");

    @Override
    public void validate(FacesContext facesContext, UIComponent uiComponent, Object o)
        throws ValidatorException {
        String email = (String) o;
        if(emailsDB.contains(email)){
            FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                summary: "E-mail já existe!", detail: "");
            throw new ValidatorException(message);
        }
    }
}

```

## Classe de Validação Geral

O outro método é o de criar uma classe geral de validação e as validações específicas são métodos dessa classe. A Figura 4 mostra o código da classe.

```

@Named
@RequestScoped
public class Validators implements Serializable {
    private List<String> emailsDB = asList("jaimearanha96@gmail.com", " contato@devdojo.com.br");

    @Inject
    private LoginBean loginBean;

    public void validateDuplicatedEmail(FacesContext facesContext, UIComponent uiComponent, Object o)
        throws ValidatorException {
        System.out.println(loginBean.getLanguage());
        String email = (String) o;
        if(emailsDB.contains(email)){
            FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                summary: "E-mail já cadastrado!", detail: "");
            throw new ValidatorException(message);
        }
    }
}

```

## Resultados

Em ambos os exemplos os resultados são os mesmos, caso o e-mail inserido seja um dos da lista de e-mail, ele será barrado, caso seja um e-mail novo ele será exibido no terminal. Figura 5 e 6 mostram os resultados.

E-mail

jaimeduardo@hotmail.com

Save

```
13:40:06,490 INFO [stdout] (default task-1) Salvando...
13:40:06,490 INFO [stdout] (default task-1) jaimeduardo@hotmail.com
```



**E-mail já existe!**

E-mail

jaimearanha96@gmail.com

Save

## Aula 38

Na aula 38 se trabalha com o uso de converters.

### Select One Menu

O exemplo consiste de exibir um barra de seleção em que os valores sejam objetos. Figura 7 mostra o código de classe e Figura 8 mostra o código da página.

```
public class ConverterTesteBean implements Serializable {
    private List<Estudante> estudanteList = Estudante.estudanteList();
    private Estudante estudante;

    public void save(){
        System.out.println("Salvando");
        System.out.println(estudante);
    }
}
```

```
<h:body>
    <p:messages autoUpdate="true"/>
    <h:outputText value="Selecione um Estudante!"/><br/>
    <h:form>
        <h:selectOneMenu value="#{converterTesteBean.estudante}"
            converter="estudanteConverter">
            <f:selectItems value="#{converterTesteBean.estudanteList}"
                var="estudante" itemLabel="#{estudante.nome}"
                itemValue="#{estudante}"/>
        </h:selectOneMenu>
        <h:commandButton value="Save"
            actionListener="#{converterTesteBean.save()}" />
    </h:form>
</h:body>
```

Inicialmente o programa foi compilado sem o uso do conversor e ao tentar selecionar um dos estudantes uma mensagem de erro aparece informando da necessidade de um conversor. Figura 9 mostra o resultado.



### Conversor

Para realizar a conversão foi criada uma classe de conversão customizada. Figura 10 mostra o código do conversor e Figura 11 mostra o resultado no terminal após a conversão ser bem sucedida.

```

public class EstudanteConverter implements Converter {
    private List<Estudante> estudanteList = Estudante.estudanteList();

    @Override
    public Object getAsObject(FacesContext facesContext, UIComponent uiComponent, String value) {
        if(value == null || !value.matches( regex: "\\\d+")) {
            return null;
        }
        // Estudante estudante = new Estudante();
        // estudante.setId(Integer.parseInt(value));
        // int index = estudanteList.indexOf(estudante);
        // return estudanteList.get(index);
        return this.getAttributesFrom(uiComponent).get(value);
    }

    @Override
    public String getAsString(FacesContext facesContext, UIComponent uiComponent, Object value) {
        if(value != null && !value.equals("")){
            Estudante estudante = (Estudante) value;
            if(estudante.getId() != null){
                this.addAttribute(uiComponent,estudante);
                return estudante.getId().toString();
            }
        }
    }
}

```

Selecione um Estudante!

```

14:48:16,782 INFO [stdout] (default task-1) Salvando
14:48:16,782 INFO [stdout] (default task-1) br.com.maratonajsf.model.Estudante@21
14:48:19,849 INFO [stdout] (default task-1) Salvando
14:48:19,853 INFO [stdout] (default task-1) br.com.maratonajsf.model.Estudante@22

```

## Aula 39

Na aula 39 utiliza o omnifaces para realizar validação por igualdade.

### Validação por Igualdade

O exemplo da aula consiste em digitar uma senha e realizar a validação apenas se a confirmação da senha for igual, para realizar essa validação foi utilizado o omnifaces. Figura 1 mostra o código da classe e Figura 2 mostra o código da página.

```
<h:panelGrid columns="1">
    <h:outputText value="Senha"/>
    <h:inputText id="password" value="#{validationTesteBean.password}"
        required="true" requiredMessage="Senha Obrigatória"/>

    <h:outputText value="Confirme a Senha"/>
    <h:inputText id="confirm"
        required="true" requiredMessage="Confirme a Senha"/>
    <o:validateEqual components="password confirm" message="Senhas não são Iguais"/>
    <h:commandButton value="Salvar" actionListener="#{validationTesteBean.save()}" />
</h:panelGrid>
```

```
@Named
@ViewScoped
public class ValidationTesteBean implements Serializable {
    private String password;

    public void save(){
        System.out.println(password);
    }
}
```

Na página caso as senhas não sejam iguais uma mensagem de erro é exibida, caso as senhas sejam compatíveis a senha será validada e exibida no terminal. Figura 3 mostra a mensagem de erro e Figura 4 mostra o resultado no terminal.



## Senhas não são Iguais

Senha

Confirme a Senha

[Salvar](#)

Senha

Confirme a Senha

```
13:50:34,613 INFO [stdout] (default task-1) 1222
```

## Aula 40

Na aula 40 se dá inicio ao uso de data tables.

### Data Tables

Para o uso de data tables foi criada uma classe chamada 'DataTablesTesteBean' com diferentes tipos do coleções e uma página xhtml chamada 'datatable'. Figura 5 mostra o código da classe e Figura 6 mostra o código descrevendo a data table.

```
@Named  
@ViewScoped  
public class DataTableTesteBean implements Serializable {  
    private List<Estudante> estudanteList = Estudante.estudanteList();  
    private Set<Estudante> estudanteSet = new HashSet<>(Estudante.estudanteList());  
    private List<Estudante> estudanteLinkedList = new LinkedList<>(Estudante.estudanteList());
```

```
<f:facet name="header">Nome</f:facet>  
    <h:outputText value="#{estudante.nome}" />  
</h:column>  
<h:column>  
    <f:facet name="header">Sobrenome</f:facet>  
    <h:outputText value="#{estudante.sobrenome}" />  
</h:column>  
<h:column>  
    <f:facet name="header">Turno</f:facet>  
    <h:outputText value="#{estudante.turno}" />  
</h:column>  
<h:column>  
    <f:facet name="header">Email</f:facet>  
    <h:outputText value="#{estudante.email}" />  
</h:column>
```

O primeiro teste foi feito com uma lista de estudantes exibindo o nome, sobrenome, turno e e-mail. Figura 7 mostra o resultado na porta 8080.

Nome	Sobrenome	Turno	Email
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

Em seguida foi criada uma linked list e um set. Figura 8 mostra o código da página e Figura 9 mostra o resultado.

```
<p:messages autoUpdate="true"/>
<h:form>
    <h: dataTable value="#{dataTableTesteBean.estudanteList}" var="estudante" ...>
    <h: dataTable value="#{dataTableTesteBean.estudanteSet}" var="estudante" ...>
    <h: dataTable value="#{dataTableTesteBean.estudanteLinkedList}" var="estudante" ...>
</h:form>
```

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

Para verificar o funcionamento adequado do set foi adicionado um novo estudante com os mesmos dados de um estudante previamente criado e podemos ver na imagem que a tabela do set não o reproduz, já que o set ignora repetições.

## Utilizando Map

Para criar uma tabela utilizando map é preciso obter seus valores através do método values. Figura 10 mostra o código do map na classe e Figura 11 mostra o resultado.

```
private Map<String, Estudante> mapEstudanteList = new HashMap<>();

public void init(){
    mapEstudanteList.put("Estudante 1", new Estudante( id: 4, nome: "Otto", sobrenome: "Victor", n
    mapEstudanteList.put("Estudante 2", new Estudante( id: 5, nome: "Antonio", sobrenome: "Ricardo
    mapEstudanteList.put("Estudante 3", new Estudante( id: 6, nome: "Matheus", sobrenome: "Alves", )
}
```

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Jaime	Eduardo	MATUTINO	
Miguel	Xavier	MATUTINO	
Miguel	Xavier	MATUTINO	
Diego	Alexandre	MATUTINO	

<b>Nome</b>	<b>Sobrenome</b>	<b>Turno</b>	<b>Email</b>
Matheus	Alves	MATUTINO	
Antonio	Ricardo	MATUTINO	
Otto	Victor	MATUTINO	

Para poder exibir as chaves do map é precios utilizar o método setKey. Figura 12 mostra o código da página chamando o setKey e Figura 13 mostra o resultado.

```
<h:dataTable value="#{dataTableTesteBean.mapEstudanteList.keySet()}> var="string">
    <h:column>
        <f:facet name="header">Chave</f:facet>
        <h:outputText value="#{string}" />
    </h:column>
</h:dataTable>
```

## Chave

Estudante 3

Estudante 2

Estudante 1

## Aula 41

Na aula 41 é feita a ordenação em um data table.

### Ordenação

O exemplo trabalhado em aula consiste de adicionar dois métodos na classe 'DataTableTesteBean' para ordenar a tabela pelo nome ou sobrenome. Figura 1 mostra o código da classe, Figura 2 mostra o código da página e Figura 3 mostra o resultado.

```
public void orderByNome(String ordem){  
    if(ordem.equals("asc"))  
        estudanteList.sort(Comparator.comparing(Estudante::getNome));  
    else  
        estudanteList.sort(Comparator.comparing(Estudante::getNome).reversed());  
}  
  
public void orderBySobrenome(String ordem){  
    if(ordem.equals("asc"))  
        estudanteList.sort(Comparator.comparing(Estudante::getSobrenome));  
    else  
        estudanteList.sort(Comparator.comparing(Estudante::getSobrenome).reversed());  
}
```

```
<f:facet name="header">  
    <h:commandLink actionListener="#{dataTableTesteBean.orderByNome('asc')}">  
        Nome asc  
    </h:commandLink>  
    <h:commandLink actionListener="#{dataTableTesteBean.orderByNome('desc')}">  
        Nome desc  
    </h:commandLink>  
</f:facet>
```

<u>Nome</u> asc	<u>Nome</u> desc	<u>Sobrenome</u> asc	<u>Sobrenome</u> desc	<u>Turno</u>	<u>Email</u>
Diego		Alexandre		MATUTINO	
Jaime		Eduardo		MATUTINO	
Miguel		Xavier		MATUTINO	
Miguel		Xavier		MATUTINO	

## Aula 42

Na aula 42 foi trabalhado como remover e editar os dados de uma data table.

### Remover

Para remover foi adicionado um método na classe e um command link na página. Figura 4 mostra o código da classe, Figura 5 mostra o código da página e Figura 6 mostra o resultado.

```
public void remove(Estudante estudante){  
    estudanteList.remove(estudante);  
}
```

```
facet name="header"></f:facet>  
commandLink actionListener="#{dataTableTesteBean.remove(estudante)}" rendered="#{not estudante.editing}" value="Remover" />  
<h:commandLink>
```

<u>Nome</u> <a href="#">asc</a>	<u>Nome</u> <a href="#">desc</a>	<u>Sobrenome</u> <a href="#">asc</a>	<u>Sobrenome</u> <a href="#">desc</a>	<u>Turno</u>	<u>Email</u>
Jaime	Eduardo			MATUTINO	<a href="#">Remover</a>
Miguel	Xavier			MATUTINO	<a href="#">Remover</a>
Miguel	Xavier			MATUTINO	<a href="#">Remover</a>

### Editar

Para editar o nome, sobrenome e e-mail foi adicionado um método na classe para editar e um para salvar, também foi adicionado um command link na página, além disso foi adicionada uma variável booleana na classe estudante que determina o que deve ser renderizado na tela. Figura 7 e 8 mostram os códigos da classe, Figura 9 mostra o código da página e Figura 10 mostra o resultado.

```
public void edit(Estudante estudante){  
    estudante.setEditing(true);  
}
```

```

public void save(){
    estudianteList.forEach(estudante -> {
        if (estudante.isEditing())
            System.out.println(estudante);
        estudante.setEditing(false);
    });
}

```

```

</f:facet>
<h:outputText value="#{estudante.sobrenome}" rendered="#{not estudante.editing}"/>
<h:inputText value="#{estudante.sobrenome}" rendered="#{estudante.editing}"/>

```

<a href="#">Nome asc</a>	<a href="#">Nome desc</a>	<a href="#">Sobrenome asc</a>	<a href="#">Sobrenome desc</a>	Turno	Email
Padilla		Aranha		MATUTINO	aranha.padilla@hotmail.com
Miguel		Xavier		MATUTINO	<a href="#">Remover</a> <a href="#">Editar</a>
Miguel		Xavier		MATUTINO	<a href="#">Remover</a> <a href="#">Editar</a>
Diego		Alexandre		MATUTINO	<a href="#">Remover</a> <a href="#">Editar</a>
<a href="#">Salvar</a>					

## Barra de Seleção

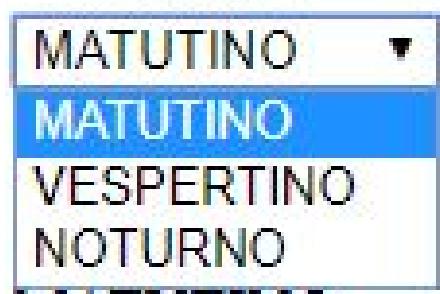
Para poder editar o turno foi preciso utilizar o primefaces para poder exibir os valores possíveis da classe turno em uma barra de seleção. Figura 11 mostra o código da página e Figura 12 mostra o resultado.

```

<h:column>
    <f:facet name="header">Turno</f:facet>
    <h:outputText value="#{estudante.turno}" rendered="#{not estudante.editing}"/>
    <h:selectOneMenu value="#{estudante.turno}" rendered="#{estudante.editing}">
        <f:selectItems value="#{Turno.ALL_ENUM_VALUES}"
                        var="turno" itemValue="#{turno}" itemLabel="#{turno}" />
    </h:selectOneMenu>
</h:column>

```

## Turno



## Aula 43

Na aula 43 se da inicio ao uso de requisições ajax.

### Requisição Ajax

O exemplo cria um nova classe chamada 'AjaxTesteBean' e uma nova página chamada 'ajax' que possui duas caixas de texto e uma vez que são preenchidas são exibidas em letras maiuscúlas. Figura 1 mostra o código da classe, Figura 2 mostra o código da página e Figura 3 mostra o resultado.

```
@Named  
@ViewScoped  
  
public class AjaxTesteBean implements Serializable {  
    private String nome;  
    private String sobrenome;  
  
    public void toUpperCase(){  
        this.nome = this.nome.toUpperCase();  
        this.sobrenome = this.sobrenome.toUpperCase();  
    }  
}
```

```
<h:form>  
    <h:panelGrid columns="1">  
        <h:inputText value="#{ajaxTesteBean.nome}" id="nome"/>  
        <h:inputText value="#{ajaxTesteBean.sobrenome}" id="sobrenome"/>  
        <h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">  
            <f:ajax execute="nome sobrenome" render="nomeOut sobrenomeOut"/>  
        </h:commandButton>  
        <h:outputText value="#{ajaxTesteBean.nome}" id="nomeOut"/>  
        <h:outputText value="#{ajaxTesteBean.sobrenome}" id="sobrenomeOut"/>  
    </h:panelGrid>  
</h:form>
```

JAIME
EDUARDO
<input type="button" value="Enviar"/>

JAIME  
EDUARDO

## Aula 44

Na aula 44 é utilizado o @form, @none, @this e @all.

### Form

O @form permite a execução e renderização de tudo dentro do formulário. Figura 4 mostra o código. Com o uso do @form é possível obter os mesmos resultados da aula 43.

```
<h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">
    <f:ajax execute="@form" render="@form"/>
```

### None

O @none impede a execução ou renderização dentro do formulário, no exemplo feito em aula é possível ver que os valores inseridos são exibidos no terminal, mas não são renderizado. Figura 5 mostra o código e Figura 6 mostra o resultado.

```
<h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">
    <f:ajax execute="@form" render="@none"/>
```



```
14:10:11,920 INFO [stdout] (default task-1) JAIME
14:10:11,921 INFO [stdout] (default task-1) EDUARDO
```

### All

O uso de @all permite a execução e renderização de tudo na página. Figura 7 mostra o resultado.

```
<h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">
    <f:ajax execute="@all" render="@all"/>
```

### This

O @this renderiza ou executa apenas o componente em que ele foi chamado, no exemplo da aula é possível renderizar o nome no output, mas o nome não é alterado na caixa de texto. Figura 8 mostra o código e Figura 9 mostra o resultado.

```
<h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">
    <f:ajax execute="@all" render="@this nomeOut"/>
```

The screenshot shows a user interface with two input fields and a command button. The first input field contains the text "jaime". The second input field contains the text "eduardo". Below these fields is a command button with the value "Enviar". The output section displays the values "JAIME" and "EDUARDO" respectively.

## Dados de Outro Formulário

Para exibir os dados de outro formulário é preciso utilizar o id desse formulário. Figura 10 mostra o código e Figura 11 mostra o resultado.

```

<h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">
    <f:ajax execute="@all" render="@form form2:nomeOut2 form2:sobrenomeOut2"/>
</h:commandButton>
<h:outputText value="#{ajaxTesteBean.nome}" id="nomeOut"/>
<h:outputText value="#{ajaxTesteBean.sobrenome}" id="sobrenomeOut"/>
</h:panelGrid>
</h:form>
<h:form id="form2">
    <h:outputText value="#{ajaxTesteBean.nome}" id="nomeOut2"/>
    <h:outputText value="#{ajaxTesteBean.sobrenome}" id="sobrenomeOut2"/>
</h:form>

```

The screenshot shows a user interface with two input fields and a command button. The first input field contains the text "JAIME" and the second contains the text "EDUARDO". Below these fields is a command button with the value "Enviar". The output section displays the values "JAIME", "EDUARDO", and "JAIMEEDUARDO" respectively.

## Dados sem Formulário

Para exibir dado que não estão inseridos em nenhum formulário é preciso apenas chamar a id da variável. Figura 12 mostra o código e Figura 13 mostra o resultado.

```
<h:outputText value="#{ajaxTesteBean.sobrenome}" id="outside"/>
<h:form id="form">
    <h:panelGrid columns="1">
        <h:inputText value="#{ajaxTesteBean.nome}" id="nome"/>
        <h:inputText value="#{ajaxTesteBean.sobrenome}" id="sobrenome"/>
        <h:commandButton value="Enviar" actionListener="#{ajaxTesteBean.toUpperCase()}">
            <f:ajax execute="@all" render="@form form2-nomeOut2 form2-sobrenomeOut2 outside"/>
        </h:commandButton>
    </h:panelGrid>
</h:form>
```

EDUARDO

JAIME

EDUARDO

Enviar

JAIME

EDUARDO

JAIMEEDUARDO

## Aula 45

Na aula 45 é utilizado o listener do ajax como um substituto para o actionListener.

### Listener

É possível substituir o actionListener pelo listener e obter o mesmo resultado. O exemplo da aula passada foi utilizado na comparação. Figura 1 mostra o código da página e Figura 2 mostra o resultado.

```
<f:ajax execute="@all" listener="#{ajaxTesteBean.toUpperCase}" />
```

ARANHA

PADILLA

ARANHA

Enviar

PADILLA

ARANHA

PADILLAARANHA

## Aula 46

Na aula 46 é trabalhado o uso de diferentes eventos em uma requisição ajax.

### Keyup

O keyup realiza a requisição assim que a tecla é pressionada. O exemplo da aula consiste de digitar em uma caixa de texto e as letras serem automaticamente convertidas para maiúscula. Figura 3 mostra o código da página e Figura 4 mostra o resultado.

```
<h:inputText value="#{ajaxTesteBean.nome}">
    <f:ajax event="keyup" listener="#{ajaxTesteBean.toUpperCaseNome}"
        render="@this nomeOut"/>
</h:inputText>
```

JAI<sup>M</sup>E EDUARDO

JAI<sup>M</sup>E EDUARDO|

### Blur

O blur realiza a requisição assim que o componente em foco é mudado, utilizando o mesmo exemplo do keyup, as letras só serão convertidas em maiúscula quando a caixa de texto perder o foco. Figura 5 mostra o código e Figura 6 mostra o resultado.

```
<f:ajax event="blur" listener="#{ajaxTesteBean.toUpperCaseNome}"
        render="@this nomeOut"/>
```

PADILLA ARANHA

PADILLA ARANHA

### Change

Para utilizar o evento change foi feito um novo exemplo. O exemplo consiste de duas caixas de seleção onde a primeira caixa determina os valores contidos na segunda. Figura 7 mostra o código da classe, Figura 8 mostra o código da página e Figura 9 mostra o resultado.

```

private Map<String, List<String>> animePersonagensMap;
private List<String> personagens;
private String animeSelecionado;
private String personagemSelecionado;
{
    animePersonagensMap = new TreeMap<>();
    animePersonagensMap.put("Hellsing", asList("Alucard", "Seras", "Alexander"));
    animePersonagensMap.put("Attack on Titan", asList("Eren", "Mikasa", "Armin"));
    animePersonagensMap.put("Berserk", asList("Guts", "Casca", "Griffith"));
}

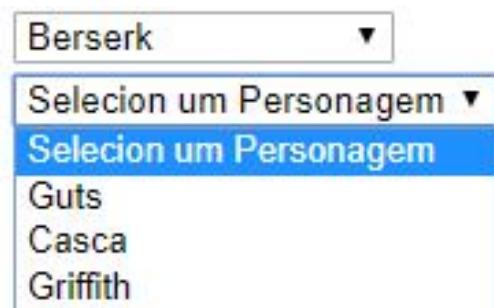
public void oneSelectAnime(){
    if(animeSelecionado == null){
        personagens = null;
        return;
    }
    personagens = animePersonagensMap.get(animeSelecionado);
}

```

```

<h:selectOneMenu value="#{ajaxTesteBean.animeSelecionado}">
    <f:selectItem value="#{null}" itemLabel="Selecion um Anime"/>
    <f:selectItems value="#{ajaxTesteBean.animePersonagensMap.keySet()}"
                    var="anime" itemLabel="#{anime}" itemValue="#{anime}"/>
    <f:ajax event="change" listener="#{ajaxTesteBean.oneSelectAnime}" render="persona"/>
</h:selectOneMenu>
<h:selectOneMenu id="persona" value="#{ajaxTesteBean.personagemSelecionado}">
    <f:selectItem value="#{null}" itemLabel="Selecion um Personagem"/>
    <f:selectItems value="#{ajaxTesteBean.personagens}"
                    var="personagem" itemLabel="#{personagem}" itemValue="#{personagem}"/>
</h:selectOneMenu>

```



O change realiza a mudança dos valores da segunda caixa assim que um dos valores da primeira é selecionado.

## Aula 47

Na aula 47 é demonstrado uma outra forma de utilizar a requisição ajax.

### Agrupamento de Componentes

Nas aulas anteriores a requisição ajax era chamada dentro de cada componente como um botão ou caixa de texto, na aula 47 foi feito o inverso com os componentes sendo chamados dentro da requisição. Figura 1 mostra o código da página e Figura 2 mostra o resultado.

```
<h:panelGrid columns="1">
    <f:ajax event="blur">
        <h:inputText value="#{ajaxTesteBean.email}" id="email"
                    validator="#{validators.validateDuplicatedEmail}" required="true"/>
        <h:inputText value="#{ajaxTesteBean.nome}" id="nome" required="true"/>
        <h:commandButton value="Enviar"/>
    </f:ajax>
</h:panelGrid>
```



## Aula 48

Na aula 48 foi trabalhado o uso do postback em requisições ajax.

### Postback

O uso do postback permite executar a função init antes da página ser renderizada. Figura 3 mostra o código da página, Figura 4 mostra o código da função init e Figura 5 mostra o resultado.

```
<p:messages autoUpdate="true" id="messages"/>
<h:outputText value="é uma requisição ajax? #{facesContext.postback}" id="postback"/>
<f:event listener="#{ajaxTesteBean.init}" type="preRenderView"/>
<h:form id="form">
    <h:panelGrid columns="1">
        <f:ajax event="blur" render="postback"/>
```

```
public void init(){
    if(!FacesContext.getCurrentInstance().isPostback()){
        System.out.println("entrou");
        animePersonagensMap = new TreeMap<>();
        animePersonagensMap.put("Hellsing", asList("Alucard", "Seras", "Alexander"));
        animePersonagensMap.put("Attack on Titan", asList("Eren", "Mikasa", "Armin"));
        animePersonagensMap.put("Berserk", asList("Guts", "Casca", "Griffith"));
    }
}
```



### View Action

Utilizando o event do exemplo anterior toda vez que uma requisição ajax for feita a função init seria chamada, para evitar esse problema foi utilizado um if na função init. O view action não necessita do if, pois ele não responde a requisições ajax. Figura 6 mostra o código.

```
<f:metadata>
    <f:viewAction action="#{ajaxTesteBean.init()}" />
</f:metadata>
```

## Aula 49

Na aula 49 é utilizado o delay para requisições ajax.

### Delay

O delay pode ser adicionado ao componente ajax para determinar o tempo em milisecondo em que as requisições serão ignoradas, no exemplo da foi adicionado um delay de 1000 ms. Figura 1 mostra o código e Figura 2 mostra resultado.

```
<h:inputText value="#{ajaxTesteBean.nome}" id="nome" required="true">
    <f:ajax event="keyup" listener="#{ajaxTesteBean.toUpperCaseNome}"
           render="nomeOut" delay="1000"/>
</h:inputText>
```

é uma requisição ajax? false

HAJHSGDNCIFURJKLSJ

## Aula 50

Na aula 50 se dá inicio ao uso de html 5 no curso.

### Utilizando Atributos do HTML5 no JSF

Para os exemplos da aula foi criada uma classe chamada 'Html5TesteBean' e uma página chamada 'html5'. Para poder adicionar uma mensagem na caixa de texto enquanto ela estiver vazia utilizando o JSF foi utilizado o passthrough. Figura 3 mostra o código da página e Figura 4 mostra o resultado.

```
<h:form id="form">
    <h:panelGrid columns="1">
        <h:inputText value="#{html5TesteBean.email}"
            pt:type="email"
            pt:placeholder="Digite o seu E-mail"/>
        <input type="email" placeholder="Digite o seu E-mail"/>
    </h:panelGrid>
</h:form>
```

The screenshot shows three separate input fields. Each field has a light gray border and contains the text "Digite o seu E-mail" in a dark gray font. The first two fields are enclosed in a horizontal line, while the third one is below them, indicating they are part of a panel grid.

### Outras Formas de Passar Atributos

Além do primeiro exemplo existem outras formas de passar os atributos, uma delas é o passThroughAttribute. Figura 5 mostra o código e Figura 6 mostra o resultado.

```
<h:inputText>
    <f:passThroughAttribute name="type" value="email"/>
    <f:passThroughAttribute name="placeholder" value="Digite o seu E-mail"/>
</h:inputText>
```

The screenshot shows three separate input fields. Each field has a light gray border and contains the text "Digite o seu E-mail" in a dark gray font. The first two fields are enclosed in a horizontal line, while the third one is below them, indicating they are part of a panel grid.

Também existe o passThroughAttributes que recebe um map com os atributos a serem utilizados pelo JSF. Figura 7 mostra o código da class com o map, Figura 8 mostra o código da página e Figura 9 mostra o resultado.

```
@Named  
@ViewScoped  
public class Html5TesteBean implements Serializable {  
    private String email;  
    private Map<String, String> attributes = new HashMap<>();  
  
    public void init(){  
        attributes.put("type", "email");  
        attributes.put("placeholder", "Digite o seu E-mail");  
    }  
}
```

```
<h:inputText value="#{html5TesteBean.email}">  
    <f:passThroughAttributes value="#{html5TesteBean.attributes}" />  
</h:inputText>
```

Digite o seu E-mail

## Aula 51

Na aula 51 foi trabalhado o uso de html 5 diretamente na página xhtml.

### Utilizando o HTML 5

Para utilizar o HTML 5 junto com JSF é preciso determinar o que é JSF dentro do html 5, o primeiro exemplo faz um input de um e-mail. Figura 1 mostra o código, Figura 2 mostra o resultado e Figura 3 mostra o resultado quando um e-mail válido é digitado.

```
<form JSF:id="form">
    Email: <input JSF:id="email" type="email" JSF:value="#{html5TesteBean.email}" /><br/>
    <button JSF:actionListener="#{html5TesteBean.salvar}">
        Salvar
    </button>
</form>
```

Email:

```
14:06:32,147 INFO  [stdout] (default task-1) jaimearanha96@gmail.com
```

Email:

 Inclua um "@" no endereço de e-mail. "jjhtdrsrs" está com um "@" faltando.

### Validação

É possível utilizar a classe de validação criada em uma aula anterior. Figura 4 mostra o código e Figura 5 mostra o resultado.

```
JSF:validator="#{validators.validateDuplicatedEmail}" /><br/>
```

 E-mail já cadastrado!

Email:

## Mais Componentes

Além do e-mail também foram adicionados um input do tipo URL e do tipo number. Figura 6 mostra o código e Figura 7 mostra o resultado na porta 8080 e no terminal.

```
<p:messages autoUpdate="true" id="messages"/>
Email: <input JSF:id="email" type="email" required="required"
JSF:value="#{html5TesteBean.email}"
JSF:validator="#{validators.validateDuplicatedEmail}"/>
<br/>
URL: <input JSF:id="url" type="url" required="required"
JSF:value="#{html5TesteBean.url}"/>
<br/>
Número: <input JSF:id="number" type="number" required="required"
JSF:value="#{html5TesteBean.numero}"/>
<br/>
```

Email:

URL:

Número:

```
14:26:20,241 INFO [stdout] (default task-1) jaimeduardo050396@hotmail.com
14:26:20,241 INFO [stdout] (default task-1) https://www.xboxpower.com.br/
14:26:20,241 INFO [stdout] (default task-1) 5
```

## Aula 52

Na aula 52 é trabalhado com diferentes formas de validação de beans.

### Constraints

O pacote javax.validation.constraints possui diferentes formas de validação. Os exemplos trabalhados em aula foram:

- Tamanho máximo para o nome.
- Padrão de e-mail a ser inserido
- Valor máximo e mínimo para um número racional.
- Número máximo da casas decimais e fracionárias.

A Figura 8 mostra o código da classe com as restrições sendo estabelecidas a cada variável, Figura 9 mostra o código da página, Figura 10 mostra o resultado no caso dos valores inseridos não serem adequados e Figura 11 mostra o resultado quando os valores são aceitos e exibidos no terminal.

```
@Size(max = 10, message = "O nome precisa ter entre 3 e 10 caracteres")
@NotNull(message = "Nome Obrigatório")
private String nome;
@Pattern(regexp = "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.\w{2,3}$",
         message = "Digite um email válido")
private String email;
@DecimalMax(value = "30.55", message = "Precisa ser menor que 30.55")
@DecimalMin(value = "2.35", message = "Precisa ser maior que 2.35")
private double decimal;
@Digits(integer = 3, fraction = 2,
        message = "No máximo duas casas decimais, número entre 0 e 999")
private double numero;
```

```
<p:messages autoUpdate="true" id="messages"/>
<h:form>
    <h:panelGrid columns="1">
        <h:inputText value="#{beanValidationTesteBean.nome}" />
        <h:inputText value="#{beanValidationTesteBean.email}" />
        <h:inputText value="#{beanValidationTesteBean.decimal}" />
        <h:inputText value="#{beanValidationTesteBean.numero}" />
```



O nome precisa ter entre 3 e 10 caracteres

Digite um email valido

Precisa ser maior que 2.35

No máximo duas casas decimais, número entre 0 e 999

```
16:04:13,135 INFO [stdout] (default task-1) jaime
16:04:13,135 INFO [stdout] (default task-1) jaimearanha96@gmail.com
16:04:13,135 INFO [stdout] (default task-1) 100.22
16:04:13,135 INFO [stdout] (default task-1) 30.54
```

## Aula 53

Na aulas 53 se dá inicio ao assunto de fazer upload de arquivos

### Upload

Para o exemplo da aula foi criada uma nova classe chamada 'UploadTesteBean' e uma nova página chamada 'upload'. O exemplo uso o inputFile para carregar o arquivo e após o carregamento ser bem sucedido seus dados são exibidos no terminal. Figura 1 mostra o código da classe, Figura 2 mostra o código da página, Figura 3 mostra a mensagem de erro caso uma imagem não tenha sido selecioanda e a Figura 4 mostra o resultado exibido no terminal.

```
public class UploadTesteBean implements Serializable {  
    private Part file;  
  
    public void upload(){  
        try(InputStream is = file.getInputStream()){  
            String fileName = file.getSubmittedFileName();  
            System.out.println(fileName);  
            System.out.println(file.getName());  
            System.out.println(file.getSize());  
            System.out.println(file.getContentType());  
            System.out.println(file.getHeaderNames());  
        }catch(IOException e){  
            e.printStackTrace();  
        }  
    }  
}
```

```
<p:messages autoUpdate="true" id="messages"/>  
<h:form id="form" enctype="multipart/form-data">  
    <h:panelGrid columns="1">  
        <h:inputFile value="#{uploadTesteBean.file}"  
                    required="true"  
                    requiredMessage="Arquivo obrigatório"/>  
        <h:commandButton value="Enviar" actionListener="#{uploadTesteBean.upload}"/>  
    </h:panelGrid>
```



## Arquivo obrigatório

Escolher arquivo

Nenhum arquivo selecionado

Enviar

```
13:46:08,850 INFO [stdout] (default task-1) fileParaEnvio.png
13:46:08,851 INFO [stdout] (default task-1) form-j_idt5
13:46:08,851 INFO [stdout] (default task-1) 381462
13:46:08,852 INFO [stdout] (default task-1) image/png
13:46:08,852 INFO [stdout] (default task-1) [Content-Disposition, Content-Type]
```

### Salvar

Para salvar o arquivo em um outra pasta com o mesmo nome foi adicionado a classe o código da Figura 5. A Figura 6 mostra a mensagem de arquivo enviado com sucesso após o arquivo ter sido salvo na pasta de destino.

```
Files.copy(is,
           new File( parent: "C:\\\\Users\\\\jaime\\\\IdeaProjects\\\\maratona-jsf",
                      fileName).toPath());
FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO,
                                         "Arquivo enviado com Sucesso",
                                         detail: "");
FacesContext.getCurrentInstance().addMessage( s: null, message);
}catch(IOException e){
    e.printStackTrace();
}
```



## Arquivo enviado com Sucesso

Escolher arquivo

Nenhum arquivo selecionado

Enviar

## Aula 54

Na aula 54 é dada continuidade ao assunto da aula 53, focando na validação para os arquivos enviados.

### Validação

Foi adicionada a classe 'Validators' um novo método de validação que verifica se o arquivo carregado é uma imagem png ou jpeg, o nome do arquivo têm no máximo 10 caracteres e seu tamanho não excede 1 Mb. Figura 7 mostra o código, Figuras 8, 9 e 10 mostram as 3 exceções sendo chamadas.

```
lic void validateFile(FacesContext facesContext, UIComponent uiComponent, Object o)
    throws ValidatorException {
    Part file = (Part) o;
    try{
        validateFileNameLength(file);
        validateContentType(file);
        validateFileSize(file);
    }catch (RuntimeException e){
        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
            e.getMessage(), detail: "");
        throw new ValidatorException(message);
    }
}
```

 **O nome + extensão não pode ser maior que 10 caracteres**

Nenhum arquivo selecionado

 **Apenas imagens PNG e JPEG**

Nenhum arquivo selecionado



**O arquivo não pode exceder 1Mb**

[Escolher arquivo](#)

Nenhum arquivo selecionado

[Enviar](#)

## Aula 55

Na aula 55 é trabalhado o gerenciamento de estados do view.

### Passando o View State para o cliente

O view state por padrão permanece no servidor, mas é possível passá-la para o computador do cliente. Figura 1 mostra o código e Figura 2 mostra o resultado com a view state do cliente.

```
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
</context-param>
```

```
<input type="hidden" name="javax.faces.ViewState" id="j_id1-javax.faces.ViewState-0"
value="3SYdXfCEjjMHR6fik6Sl+K62reQrR/
VUy6SkJddHYu7dcNIJjHaik9BsYsHYVwGdzmFNlw7cqSeyWxMIA3gAQyFoUWR14R/4Z0pQ9H/
Oyu8ZEZj4A15kaA5dJXUIkg5AkfR6CfTFjiQghQhi7AdFFUjkqpTjk2qRubyZ4VF0xDtpT2Nd8KA518weTU1eDj
ofHGbM3VW0YO3BSR7nqJ2FoKvrFGyTGhbj/
xpoQim53LvPIy+XkbPLStIghZSoNEPCGQXrs6fREAImeECQ3+jkJP4/
sIexveFFmUE8Q0drTIEafw4qXiG2Mexq0FtkG6bojhWfs1h415PDyVG/
4RPMLP0aB8fgwcf4PKzLWHnh877UAhvaJu6p1BusBs1yYq+wc550FOkE7L3FnyY1xL9HCvjxt7XL2DTq834hFkK
70gMyjdAk1nuvGPgcqyfdI0eZuMStSB0zLHQ1NUT4Q01doKKn51jE8m21DGfohzhLUHga7YAVRTC01G1sGyjb7W
r5xk+Eoe4Fn2Wf/YnjawshCBbz0Y1RPyDbX93CrJaticSTqUQPk3FwAFovb9P4wzyN/E4ubTU/
SSsmllwiS1MTgQ==" autocomplete="off">
<input type="hidden" name="javax.faces.ClientWindow" id="j_id1-
javax.faces.ClientWindow-0" value="SWTZ09hyLTXXoogGA91bwpwF571ejg_mpjmG7Ny3-0"
autocomplete="off">
```

A decisão entre manter o view state no cliente ou no servidor depende das necessidades do projeto, no caso da view state do cliente que é mostrada na Figura 2, se trata de uma view state parcial, para poder visualizar a view state completa foi utilizado o código da Figura 3 e a Figura 4 mostra a view state completa.

```
<context-param>
    <param-name>javax.faces.PARTIAL_STATE_SAVING</param-name>
    <param-value>false</param-value>
</context-param>
```

```

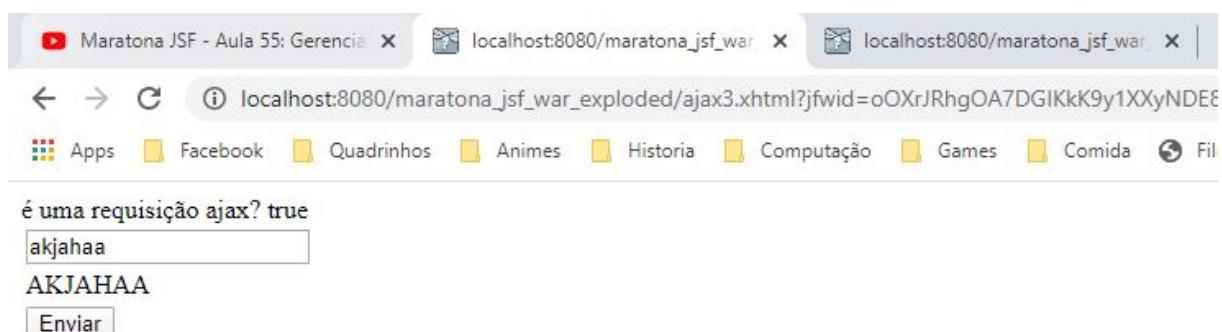
<input type="hidden" name="javax.faces.ViewState" id="j_id1-javax.faces.ViewState-0"
value="08w2jM0vlgfieIrzscAhCwScNZ++/
5cpk8pD368nU0cBKhd5kDR7fYMzshz5RsQskFILqjo06PtDjfeLFAkd/HBrFvyRbcg/
QvrtlOUhh4KAyCRgWY95NIHqh41+M7d21jNuwDkma/k6iW3ewNrIWP+uL7Nlq1IrbHXlpOV3yqEx1TerCcs/
8TkkwAJd3gJ8q3yjJ46UXzMmqrpqr6s93/SRJ2pA/
yXurutuVi3YsUp4nu++Wrxupw3VsA2RIcZw19YdS5iBGnJ3MCwLaK5tCvTs2xdE0108Tx4WEBKZDLqysZtejd/
ewjaKdUNmEUxjoKyWEZNNc0SiMuJ14nEk1200yGH4kTYtPRD1NTrGQUsoi34nD5fLoD8CwHFwGsPzd8dyM ZwNBp
p4bsFHhdN4AwxhzPIJGr6yv52k55V9VJ8igqFFwpgM/dgBhW7XWDUXGyFMu/VZ7V5eF/3i11gvVJ8cA1/
oNoNQAGA+cHc20K6aya5QLDAxtLFRN7mxD3nQw9kD1wNVHfteXT2de6nriFdadpfzTP+QgN0pygVedpzFshF51+
IEXT0m4a+mr8/LCUWBRKI5J3SDKKCMKvXUswAd8WcJ/
Mtu2jZcFyXti3OzpxE4uMkXOYKoSmTwH+tsTA4Oppwk1rSJ9C4ih+P7TuGOQpPeqUxrmCREJa2yJS1knxzFKmg:
kXky1cctKt0HJT0UsqGVE72Jstym8z0JGjHtDSi0jREHbWSmesi9uFX7uLRSFnWlcko/
H4ngDAuQ81ra8BnAwblhLvqTKJ/vzaCFQ6i1uqvWuzjxuWj7rI/
6Z7AKwztJxSFJ1C6Ti7cLbeeHizQG1VviC73ASqOKk7H5cVUnshAzCWWUFTG293d7hE9AonATyTwzlvEEis11+
7GHQPTCTigFw56vNEK0zwNqXk20zX/
BRybVXeq07T2LvuYhj6FGWDMc9DSk72ahbyff3ToR11iiy7TfHCFgeCTGwckWk+X1/
rf53NdotDnfjTHBoUSi8N95uGVuILRRuZW6GTuikmvSj4cuHhEbaF8phRXVom1KY0Qq+1kE5dxXPB523Vs5JRQt
QjSwHyh1WL0kv+Jzycz4s/
0rK37H53tqc9ezTYBtQ9tx810dtBohbhNk01hwS1bSSTrQCQNWk03JNKeVYcBgOwrU3phP4YY+IGqk7zNr7djjt
7M5DQ7i4483v4lueoeTOJF4d0oJs4Pz3GInwtkZL6DmYNccw7iej7yEqogU88Lnw/BAuR8+gOulu/
tbI9HJ6k+FXBBJHaouoUMmiXmk71auqUuBSTGiyAL4knN9hobu7xfBnrRMdUKxRyS4QwAbts+F/
58FQ6J+cVH+1jyI89rmvP3LAZcCQL4PL15Ld2SjPCT1TQiuqxJAsIxif5vLhvObF6LqezjTtJwFosxeSYJ2nMqf
-----
```

## Criando uma chave própria

Quando o servidor é reiniciado se tentarmos utilizar a página que foi chamada anteriormente ao reinicio do servidor obteremos um erro e aquela página não poderá ser utilizada, para resolver esse problema precisamos de uma chave de criptografia própria Figura 5 mostra o código que estabelece a chave e Figura 6 mostra a página funcionando apó o reinicio do servidor.

```

<env-entry>
    <env-entry-name>jsf/ClientSideSecretKey</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>8fSaFozcHpwNvK0S84MyoJ0QEJJX81AC65YdX1GseI-</env-entry-value>
</env-entry>
```



## Aula 56

A aula 56 mostra como gerenciar a view física e lógica das requisições GET e POST.

### Limite de requisições POST e GET

As requisições GET e POST possuem um limite de 15 chamadas por padrão, utilizando o código da Figura 7 podemos limitar esses valores para 2 requisições GET e 4 requisições POST.

```
<context-param>
    <param-name>com.sun.faces.numberOfLogicalViews</param-name>
    <param-value>2</param-value>
</context-param>
<context-param>
    <param-name>com.sun.faces.numberOfViewsInSession</param-name>
    <param-value>4</param-value>
</context-param>
```

Após realizar a configuração ao chamar 3 abas da página, a primeira aba não poderá mais ser recuperada, o mesmo acontece com requisições POST ao realizar 4 chamadas e voltar para a página da primeira chamada ela não poderá ser recuperada. Figura 8 mostra o erro que aparece.

**Context Path:** /maratona\_jsf\_war\_exploded

**Servlet Path:** /ajax3.xhtml

**Path Info:** null

**Query String:** jfwid=vSTenWTLVQUyCNgVuAotx9vD395K4aEWape5XBEP-0

**Stack Trace:**

```
javax.servlet.ServletException: viewId:/ajax3.xhtml - A exibição de /ajax3.xhtml não pode ser restaurada.
```

## Aula 57

A aula 57 trabalha em como recuperar uma sessão que foi finalizada.

### Invalidar Sessão

Para invalidar uma sessão foi criado o método da Figura 1 na classe 'ExpiredViewTesteBean' e a função foi chamada na página. O código da página está na Figura 2 e o resultado assim como a mensagem de erro após a sessão ser invalidada estão nas Figuras 3 e 4 respectivamente.

```
public String invalidateSession(){
    FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
    return "/login?faces-redirect=true";
}
```

```
<h:form id="form">
    <h:panelGrid columns="1">
        <h:commandButton value="Invalidar Sessão"
                          action="#{viewExpiredTesteBean.invalidateSession}"/>
    </h:panelGrid>
</h:form>
```

Invalidar Sessão

 Error processing request

Context Path: /maratona\_jsf\_war\_exploded

Servlet Path: /viewexpired.xhtml

Path Info: null

Query String: jfwid=f-hqNHOD\_UVzR82Pn6OzeMYQsJJ\_tjQuo-YNe5KV-0

Stack Trace:

```
javax.servlet.ServletException: viewId:/viewexpired.xhtml - A exibição de /viewexpired.xhtml não pode ser restaurada.
```

### Tratando o Erro

Para evitar que a mensagem de erro apareça foi realizada uma configuração no arquivo 'web.xml' que após detectar a sessão invalidada nos envia para uma nova página. Figura 5 mostra no código de configuração e Figura 6 mostra o resultado.

```
<error-page>
    <exception-type>javax.faces.application.ViewExpiredException</exception-type>
    <location>/expirederror.xhtml</location>
</error-page>
```

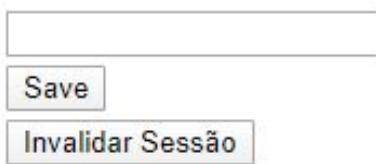
---

Sua Sessão Expirou  
[Voltar Para o Inicio](#)

## Recuperando a Sessão

Existe uma outra forma de configuração que permite que assim que sessão seja invalidada, ao tentar acessar a página uma nova sessão é criada. Figura 7 mostra o código de configuração e Figura 8 mostra uma nova sessão funcionando.

```
<context-param>
    <param-name>com.sun.faces.enableRestoreView11Compatibility</param-name>
    <param-value>true</param-value>
</context-param>
```



A screenshot of a web page titled "Recuperando a Sessão". It contains a text input field, a "Save" button, and an "Iniciar Sessão" button.

Save

Iniciar Sessão

## Aula 58

Na aula 58 é feito o carregamento de imagens e arquivos css e js.

### CSS

Para carregar um arquivo CSS primeiro se cria uma pasta chamada resources na pasta web e chama o arquivo no head. Figura 9 mostra o código chamando o arquivo css e Figura 10 mostra o resultado.

```
<h:head>
    <h:outputStylesheet name="css/style.css"/>
</h:head>
<h:body>
    <h:form id="form">
        <h:panelGrid columns="1">
            <h:inputText value="#{viewExpiredTesteBean.nome}" />
            <h:commandButton value="Save"
                action="#{viewExpiredTesteBean.save}"
                styleClass="btn"/>
    
```



### Imagen e arquivo JavaScript

Para adicionar um arquivo js assim como com o arquivo css é preciso chamar no head, já para a imagem usa-se o graphicImage. Figura 11 mostra o código do arquivo js, Figura 12 mostra o código da imagem e Figuras 13 e 14 mostram o resultado.

```
<h:outputScript name="js/main.js"/>
```

```
<h:graphicImage name="default/images/halo.jpg"/>
```

	maratona_jsf_war_ex...	200	document	Other	1.7 kB
	main.js	200	script	(index).	(memo...)
	9e2618ead20dc0128...	200	png	(index).	(memo...)

Save



## Aula 59

A aula 59 adiciona o comportamento de Java Script aos componentes.

### JS Componente

Para o exemplo da aula foi criado uma classe que retorna uma mensagem de confirmação. Figura 1 mostra o código da classe. Para utilizar essa função como um componente foi criado um arquivo de configuração. Figura 2 mostra o código de configuração.

```
@FacesBehavior(value = "confirm")
public class ConfirmDeleteBehavior extends ClientBehaviorBase {
    @Override
    public String getScript(ClientBehaviorContext behaviorContext) {
        return "return confirm('vc tem certeza?');";
    }
}
```

```
<description>Devdojo Components</description>
<namespace>http://devdojo.com.br/components</namespace>
<short-name>dc</short-name>
<tag>
    <tag-name>confirmDelete</tag-name>
    <behavior>
        <behavior-id>confirm</behavior-id>
    </behavior>
</tag>
```

Para exibir o resultado foi utilizada a página 'datatable' criada anteriormente e adicionado a mensagem de confirmação ao botão remover. Figura 3 mostra o código da página e Figura 4 mostra o resultado.

```
<h:commandLink actionListener="#{dataTableTesteBean.remove(estudante)}"
    <h:outputText value="Remover"/>
    <dc:confirmDelete/>
```

Nome asc	Nome desc	Sobrenome asc	Sobrenome desc	Turno	vc tem certeza?
Jaime	Eduardo			MATUTINO	
Miguel	Xavier			MATUTINO	
Miguel	Xavier			MATUTINO	
Diego	Alexandre			MATUTINO	<a href="#">Remover</a> <a href="#">Editar</a>
<a href="#">Salvar</a>					

## Aula 60

A aula 60 cria uma taglib customizada.

### Taglib Function

Para o exemplo da aula foi criada uma tabela com nomes escritos em letras minúsculas. Figura 5 mostra o código da classe, Figura 6 mostra o código da página e Figura 7 mostra o resultado.

```
@Named  
@ViewScoped  
public class TaglibFunctionTesteBean implements Serializable {  
    private List<String> names = asList("bruce wayne", "clark kent", "barry allen");
```

```
<h:form id="form">  
    <h:dataTable value="#{taglibFunctionTesteBean.names}" var="name">  
        <h:column>  
            <f:facet name="header">Name</f:facet>  
            <h:outputText value="#{name}"/>  
        </h:column>  
    </h:dataTable>  
</h:form>
```

### Name

bruce wayne

clark kent

barry allen

Para fazer os nomes aparecerem com as primeiras letras de cada palavra em maiúsculo foi utilizada a função WordUtils.capitalizeFully, inicialmente o resultado foi exibido no terminal. Figura 8 mostra o código e Figura 9 mostra o resultado.

```
public void init(){  
    names.forEach(name -> System.out.println(WordUtils.capitalizeFully(name)));  
}
```

```
15:15:20,110 INFO [stdout] (default task-1) Bruce Wayne  
15:15:20,110 INFO [stdout] (default task-1) Clark Kent  
15:15:20,110 INFO [stdout] (default task-1) Barry Allen
```

Para criar o componente customizado foi preciso criar um arquivo de configuração taglib.xml, Figura 10 mostra o código e Figura 11 mostra o resultado.

```
<description>Devdojo Functions</description>
<namespace>http://devdojo.com.br/functions</namespace>
<short-name>df</short-name>
<function>
    <function-name>capitalize</function-name>
    <function-class>br.com.maratonajsf.taglibfunction.TagLibFunctions</function-class>
    <function-signature>java.lang.String capitalize(java.lang.String) </function-signature>
</function>
```

Name
Bruce Wayne
Clark Kent
Barry Allen

## Aula 61

Na aula 61 foi trabalhado a construção de templates para as páginas web.

### Header, Content e Footer

Para o exemplo da aula foram criados três arquivos, um header, um content e um footer, assim como um arquivo que une esses três. Figuras 1, 2 e 3 mostram o header, content e footer, Figura 4 mostra o arquivo de composição e Figura 5 mostra o resultado.

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h1>Template Header</h1>
</ui:composition>
```

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
                  template="WEB-INF/template/layout.xhtml">
    <ui:define name="content">
        <h1>Conteúdo do Template</h1>
    </ui:define>
</ui:composition>
```

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h1>Template Footer</h1>
</ui:composition>
```

```
<div id="top">
    <ui:insert name="top">
        <ui:include src="header.xhtml"/>
    </ui:insert>
</div>
<div id="content">
    <ui:insert name="content">
        </ui:insert>
    </ui:insert>
</div>
<div id="footer">
    <ui:insert name="footer">
        <ui:include src="footer.xhtml"/>
    </ui:insert>
</div>
```

## Template Header

## Conteudo do Template

## Template Footer

### Adicionando um Footer e um Header

Após a criação dos arquivos foram adicionados ao footer e header códigos retirados da internet. Figura 6 mostra o código do footer, Figura 7 mostra o código de header e Figura 8 e 9 mostram os resultados.

```

<div class="container text-center">
    <hr />
    <div class="row">
        <div class="col-lg-12">
            <div class="col-md-3">
                <ul class="nav nav-pills nav-stacked">
                    <li><a href="#">About us</a></li>
                    <li><a href="#">Blog</a></li>
                </ul>
            </div>
            <div class="col-md-3">
                <ul class="nav nav-pills nav-stacked">
                    <li><a href="#">Product for Mac</a></li>
                    <li><a href="#">Product for Windows</a></li>
                </ul>
            </div>
        </div>
    </div>

```



```

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav">
            <a class="nav-item nav-link active" href="#">Home <span class="sr-only">(current)</span></a>
            <a class="nav-item nav-link" href="#">Features</a>
            <a class="nav-item nav-link" href="#">Pricing</a>
            <a class="nav-item nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
        </div>
    </div>
</nav>

```

## Template Header

### Conteudo do Template

---

About us	Product for Mac	Web analytics	Product Help
Blog	Product for Windows	Presentations	Developer API

---

© 2013 Company Name.

Terms of Service

Privacy



# Conteúdo do Template