**-------------------------------------------------------------------------------------------------------**

banco de dados-v2.1.pdf

**-------------------------------------------------------------------------------------------------------**

....



Adiciona chaves no if, (convenção google)

Usar classe exemplo do eclipse

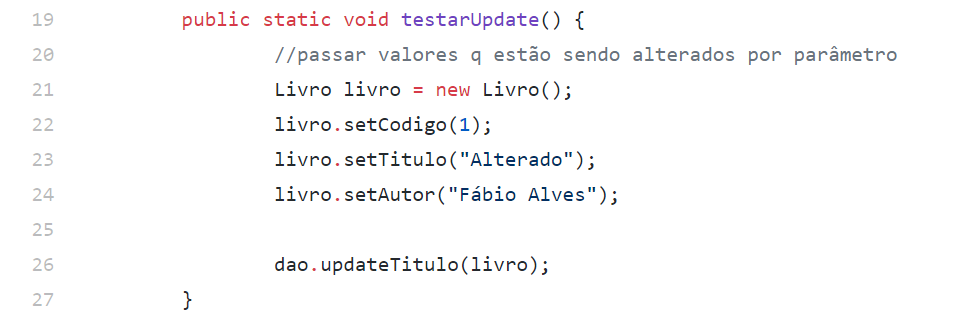
**- Slide 22:**

Executar SQL localizado em: <https://github.com/FabioDevGomes/aulas-3way/blob/master/sql/sqls%20usado%20em%20sala%20aula%20web01%20threeway.sql>

Implementar obter, após slide 23

**----------------------------------- Fim aula A2 T6 (1 de 8) parou no obter----------------------------------------**

**--------------------------------- fim a02 t7 web (2 de 10) falta testar o inserir com o método main 2 alunos--------------------------------------**



**//Criar repositório para armazenar códigos e adiantar a turma**

Para testar a chamada do método obter()

Criar a classe TestaObterClienteDao

**- Slide 23:**

Implementar inserir

**- Slide 24:**

Implementar alterar

**- Slide 28:**

Não implementar, explicar o q é CRUD (implementado no slide 23)

**- Slide 29:**

(Implementado no slide 22)

**- Slide 30:**

(Implementado no slide 24)

**- Slide 31:**

Implementar remover

Ajustar código

- setInt()

- getCodCliente

- Try com recursos

**--------------------------------- fim a03 t7 web (de 8) --------------------------------------**

Revisar crud

**-------------------------------------------------------------**

slides fábio.odp

**-------------------------------------------------------------**

What is HTML?

HTML (Hypertext Markup Language) is not a programming language; it is a markup language used to tell your browser

**- slide 3**

acessar **HTML Demo**: <strong>

alterar font-size: 1rem;

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/strong

**- slide 5**

Acessar novamente a página da tag strong e substituir para <b>

**- slide 6**

Implementar

- slide 9

Criar arquivo .css com o conteúdo mostrado em tela

- slide 10

implementar tag <style> css internal

- slide 11

alterar para css inline

**----------------------------------- Fim aula A3 T6 (3 de 8) ----------------------------------------**

- slide 12

Crie um estilo para o seletor **third** e **second**

- crier uma página HTML e adicione uma imagem

**<img src="dinosaur.png">**

- slide 13

implementar JS inline

alterar para atribuir um texto no lugar da data

- slide 14

implementar JS slide

criar function()

implementar com o texto em português

depois alterar tamanho da fonte

depois ocultar parágrafo

depois, inclua mais campos (<input>) e implemente a validação

- slide 15

testar mudança de estilo via JS (em outra função js)

outra função >> outro botão >> outra tag <p>

no google **import js html**

(JavaScript accepts both double and single quotes)

- slide 16

testar mudança ocultação de elemento via JS (em outra função js)

(JavaScript accepts both double and single quotes)

- slide 17

implementar validação via JS

- slide 18

implementar validação via atributo HTML

- testar api third-party-apis

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side\_web\_APIs/Third\_party\_APIs

<https://github.com/mdn/learning-area/blob/master/javascript/apis/third-party-apis/youtube/youtube.css>

Implementar exemplo **bootstrap.html**

**baixar bootstrap**

testar:

**<div class="col-sm-6">**

**<div class="container">**

A **tag <div> serve para alterar o estilo em partes específicas da página e posicionar objetos**.

Inclua mais dois campos: **Idade e Profissão** e mostre o valor ao enviar o formulário

**--------------------------------- fim a04 t7 web (de 8) git instalado--------------------------------------**

**Passar GitHub**

----------------------------------------

03\_Servlet\_e\_JSP.pdf

----------------------------------------

- slide 03:

Criar Servlet usando menu

- slide 04:

Explicar @WebServlet

- slide 07:

**falar sobre tomcat:**

**mostrar página de download**

**implementar**

- Implementar projeto **servlet**

**http://localhost:8080/servlet/HelloServlet**

*Import Projects from File System or Archive*

Criar segundo servlet em um novo projeto **servlet-jsp**

- servlet-jsp-v1 com hello.jsp

Configurar welcome page **hello.jsp**

- servlet-jsp-v1 com bootstrap.jsp

Ajustar **hello.jsp** para usar bootstrap

<link rel=*"stylesheet"* type=*"text/css"* href=*"css/bootstrap.css"*>

Mostrar como debugar a aplicação

- slide 09:

ajustar implementação anterior para receber parâmetro

criar tela para enviar parâmetro

Exercício: mostrar nome e profissão na tela a partir de um formulário jsp com estilos bootstrap

Saída: Meu nome é **XXXXX** e minha profissão é **XXXXX**

**----------------------------------- Fim aula A4 T6 (3 de 8) ----------------------------------------**

- slide 19:

c:forEach

- slide 20: Revisar

Exemplos a implemtentar de JSTL https://www.journaldev.com/2090/jstl-tutorial-jstl-tags-example

Criar **dynamic web project eclipse**

**Criar servlet que contenha uma lista de objetos Livro**



**Definir welcome-file**



**Criar o Servlet**

**Setar listar na requisição** -

request.setAttribute("listaLivrosNoAtributo", listaLivros);

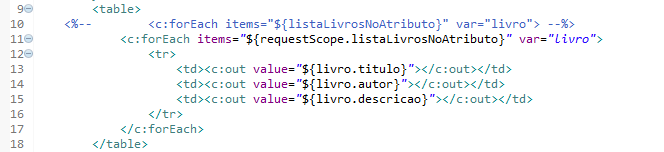
**Redirecionar**:



**Converter projeto para Maven Project para adicionar as dependências**



**Percorrer a lista na JSP usando JSTL:**



(The <c:out> tag can automatically escape XML tags so they aren't evaluated as actual tags.)

1 – Criar Dynamic Web Project (listagem-pessoas)

Configurar servidor (runtime > selecionar tomcat)

2 – Criar JSP para chamar o Servlet

3 – Configurar welcome file list (web.xml)

4 – Testa página inicial

Subir projeto no servidor e acessar o contexto (localhost:8080/ listagem-pessoas)

5 – Criar servlet (ServletPessoas)

Adicionar objetos pessoas em uma lista (ArrayList)

Adicionar lista no atributo da requisição:

- request.setAttribute("listaLivrosNoAtributo", listaLivros);

Adicionar encaminhamento para a página de listagem:

- RequestDispatcher rd = getServletContext().getRequestDispatcher("/listagemJstl.jsp");

rd.forward(request, response);

6 – Adicionar dependências no maven (jstl e standard)

7 – Criar JSP para listagem de pessoas

c:forEach

**----------------------------------- Fim aula A5 T6 (de 8) ----------------------------------------**

**Colorir as linhas:**

<c:forEach items="${requestScope.empList}" var="emp" varStatus="id"> **<tr bgcolor="#${id.count % 2 == 0 ? 'aaee88' : 'ffffff' }" >**

<td><c:out value="${emp.id}"></c:out></td>

<td><c:out value="${emp.name}"></c:out></td>

<td><c:out value="${emp.role}"></c:out></td> </tr>

</c:forEach>

**Configurar página de erro e passar template usado como exemplo 404.jsp**



<c:set var="Income" scope="session" value="${4000\*4}"/>

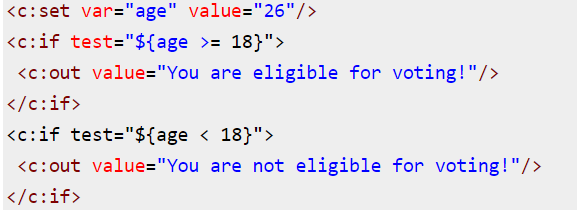
<c:out value="${Income}"/>

<c:remove var="income"/>

<p>After Remove Value is: <c:out value="${income}"/></p>

---------------

**Testar c if**



**Setar mais atributos:**

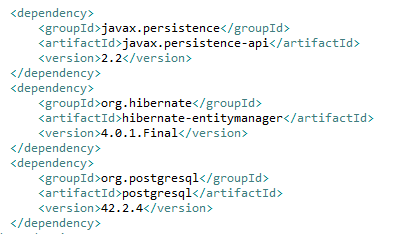
request.setAttribute("url", "https://www.journaldev.com");

------------- revisar conteúdo abaixo -----------------

------Hello world JSF

<http://www.thejavageek.com/2013/06/17/creating-jsf2-hello-world-application-in-eclipse/>

adicionar as bibliotecas do Hibernate e driver



criar uma Entity para representar a tabela Pessoa

**CREATE** **TABLE** Pessoa (

id INT **NOT** **NULL** AUTO\_INCREMENT,

nome VARCHAR(100) **NOT** **NULL**,

dataNasc DATE **NOT** **NULL**,

email VARCHAR(150),

**primary** **key**(id)

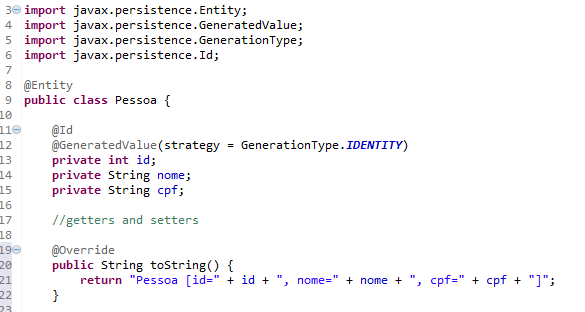
);

**Não precisa criar a tabela, o hibernate fará isso com a propriedade**

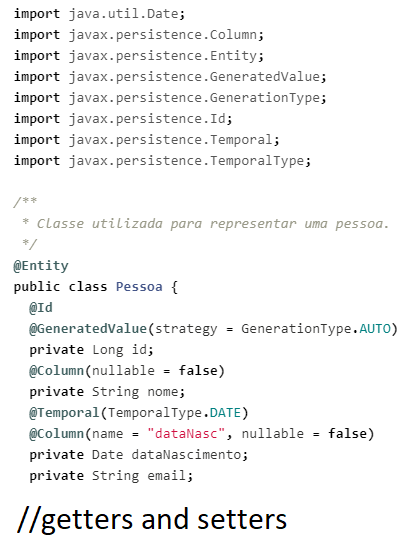
<property name=*"hibernate.hbm2ddl.auto"* value=*"update"* />

Criar a entity para representar a tabela Pessoa

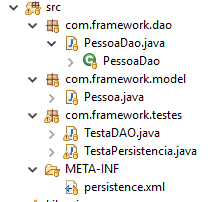
**Usaremos esse modelo**



**Exemplos de anotações que esaremos**



criar o arquivo persistence.xml



Criar uma classe **TestaPersistenceXML** para testar as configurações do persistence.xml

Criar e configurar novo projeto (**jpa-projeto-b**) com a entidade **Cliente** contendo os atributos **id**, **nome** e **profissão**

**----------------------------------- Fim aula A6 T6 (de 8) ----------------------------------------**

 Criar uma classe **PessoaDAO** que possui os métodos para manipular (salvar, atualizar, apagar e consultar por id) um objeto Pessoa

O método **salvar** recebe o objeto Pessoa que será salvo, neste exemplo usaremos este método salvar uma nova pessoa ou atualizar os dados de uma nova pessoa.

*No próximo projeto será assim:*

*Mas como sabemos quando temos que salvar e quando tem que atualizar, basta olhar o atributo id da classe Pessoa, se o****id****for null significa que é um novo objeto que ainda não foi salvo no banco de dados, então utilizaremos o método****persist****da EntityManager para salvá-lo, caso o id tenha algum valor então significa que o objeto já foi salvo anteriormente portanto ele deve ser atualizado então utilizaremos o método****merge****da EntityManager para atualiza-lo.*

Note que como vamos salvar (persist()) ou atualizar os dados, precisamos criar uma transação, com o método **getTransaction()** do EntityManager obtemos um objeto EntityTransaction com ele podemos iniciar a transação através do método **begin()**, finalizar a transação com sucesso através do método **commit()** ou desfazer as alterações em caso de erro com o método **rolback()**. Este mesmo conceito de transação será utilizado no método excluir.

O método **excluir /** removeById não precisa receber todos os dados da Pessoa, recebendo apenas o seu ID através do parametro **Long id**, podemos utilizar o método **find** do EntityManager para consultar os dados da Pessoa, depois com o objeto Pessoa consultado podemos usar o método **remove** do EntityManager para apagar os dados da Pessoa. \*\*\* **mas aqui usamos um objeto Pessoa**

O método **getById** recebe um objeto int chamado **id**, com o ID da tabela Pessoa, utilizando o método **find** do EntityManager passamos a classe da entidade **Pessoa.class** e seu **id** para que possamos consultar os dados da Pessoa.

No findAll()

utilizaremos o método createQuery que recebe um JPQL (Java Persistence Query Language) que é uma alternativa ao SQL

Testem os outros métodos da classe PessoaDAO

remove(Pessoa pessoa)

removeById(**final** **int** id)

getById(**final** **int** id)

Criar uma classe PessoaDAOTeste para testarmos os métodos da classe PessoaDAO

Criar uma classe ClienteDAO com base na entidade Cliente implementada no projeto B

Cliente contendo os atributos **id**, **nome** e **profissão**

No projeto **jsp-servlet-jpa**

Criar DAO com JPA para a entidade Pessoa (levemente diferente da entidade do projeto anterior)

- Aproveitar persistence.xml

- Criar Entidade Pessoa (mostrar implementação no eclipse)

- Criar DAO (também, levemente diferente do projeto anterior)

- Criar PessoaDAOTeste

- Criar PessoaController

- Criar listagemPessoas.jsp

- Criar pessoa.jsp