**Curso: Programação Cliente Servidor**

**Aula 1 – Orientação a Objeto - OO**

1. **Revisão OO (Orientação a Objeto)**

É um paradigma de análise, projeto e programação de software baseada em composição e interação entre unidades de software chamadas Objetos. É uma forma de ver um problema, modelar uma solução e com isso desenvolver um sistema.

1. **Abstração**

É a capacidade de do ser humano concentrar-se apenas nos aspectos essenciais de um problema. Ignorando algumas outras que não são importantes.

No contexto da orientação a objetos, uma classe é uma abstração de conceitos do mundo real, sem que precisamos fazer uso de todas as características do conteúdo real, apenas usamos como modelo, as características que são fundamentais para o nosso sistema.

1. **Classe**

É uma representação genérica de um conjunto de indivíduo idênticos.

Teoricamente, uma classe é um modelo para a criação de **objetos**, então todos os objetos serão criados baseados a partir dessa classe, que possuirão os mesmos comportamentos(**classes**) e armazenarão as informações (**atributos**).

1. **Atributos**

Digamos que é quando se refere ao objeto, como características. Por exemplo, o **Aluno** Pedro cursa Análise e Desenvolvimento de Sistemas na Universidade Sr. Madruga e tem 20 anos de idade.

Nesse casso, a classe poderia se chamar **ALUNO**, que poderia ser formada pelos atributos **NOME DO ALUNO**, **CURSO**, **UNIVERSIDADE** e **IDADE**.

Poderíamos instanciar a classe aluno, fazendo uso de seus atributos, quantas vezes precisarmos para cadastrarmos novos aluno.

Exemplo:

|  |
| --- |
| **ALUNO** |
| **NOME**: Pedro **CURSO**: Análise e Desenvolvimento de Sistemas **IDADE**: 20 **Universidade**: Sr. Madruga |

Para os **atributos**, atribuímos **valores**, os atributos são abstratos, ou seja, a característica que abstraímos para formar um **objeto** **ALUNO**, já os valores podemos identifica-los (Pedro, Análise e Des. De Sistamas, 20, Sr. Madruga).

Obs.: O estado dos atributos, podem variar.

1. **Métodos**

São as **ações** que um Objeto pode fazer.

Por exemplo, a classe ALUNO pode fazer:

Estudar  
Ler  
Acordar  
Andar

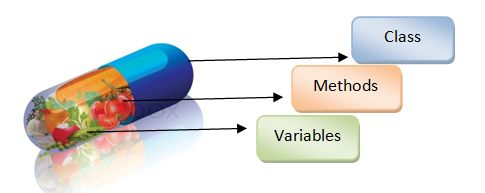
Podemos atribuir PARÂMETROS aos métodos, que são as informações que ele precisa para realizar uma ação e retornar valores, que é o produto da ação realizada.

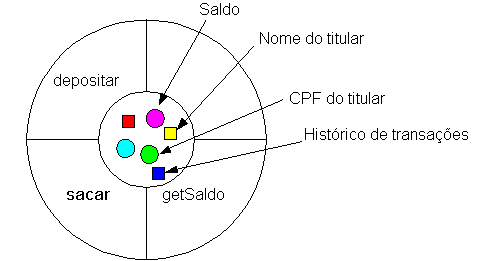
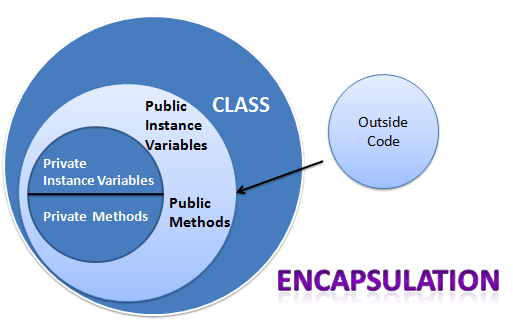
1. **Encapsulamento**

Os **dados ficam encapsulados** **no interior dos métodos** (ações) de um **objeto**, desta forma não saberemos o que há dentro dos métodos quando o estivermos manipulando. Impede acesso direto ao **Objeto.**

Exemplo:

Classe: TV  
Atributos: Cor, Polegadas, Modelo, Marca  
Métodos: Liga, Desliga, AumentaVolume, DiminuiVolume  
  
Cada método, possui a instrução que não poderá ser acessada pelo usuário enquanto estiver manipulando, mas podemos ver o que tem nos atributos como a cor da TV, a Polegada, o Modelo e a Marca, e podem ter seus valores alterados durante a manipulação.





1. **Sobrecarga**

Técnica que permite usar **vários métodos** que **tenham o mesmo identificador, variando somente a sua assinatura**.

1. **Herança**

Recurso em OO que permite que ***classes compartilhem atributos e métodos***, através de herança. (**SuperClasse** -> **SubClasse**

A classe **Felino**, poderia estender uma classe **Animal** e consequentemente a classe animal herdaria seus atributos e estados.

Exemplo 1:

Um Cross Fox, nada mais é que um Fox com mais recursos. Mas ambos possuem os mesmos atributos comuns entre si. (4 ou 2 portas, 5 pneus, Vidros, Teto, Motor, Etc). Podemos dizer que um Cross Fox(classe-filha) é Filho de um Fox(super-classe ou classe-pai).

Exemplo 2:

CLASSE-PAI: **FELINO**ATRIBUTOS**: TIPO(char), IDADE(inteiro), NOME(char), TAMANHO(char), ORIGEM (char)**MÉTODOS: **MIA**, **ANDA**, **CAÇA**

CLASSE-FILHA: **ANINAL** (INSTANCIA A CLASSE-PAI FELINO)  
HERDA ATRIBUTO E RECEBE VALORES: **TIPO=ONÇA, IDADE=8, NOME=MINGAU, TAMANHO=4M, ORIGEM=ÁFRICA**HERDA AÇÕES: **MIA, ANDA, CAÇA**

A conclusão seria que o **Animal** que **Mia**, **Anda** e **Caça**, é um **Felino**.

Conceito Visual:

ANIMAL

HERBIVORO

CARNÍVORO

ONIVORO

Coelho

Leão

Hiena

Homem

Class A

Class B

Class C

Podemos observar que no conjunto a cima, a **Class A** não herda nada, caracterizando-se como a Super-Classe ou Classe-Pai, a **Class B** herda todos os atributos e métodos da **Class A**, e a **Class C** herda todos os atributos e métodos das **Class A** e **Class B**, caracterizando-se com classes-filha,

No fluxograma ao lado, usando a mesma teoria, conseguimos ver que A *classe-pai* é **Animal** e as demais como Classe-Filha.

A classe mais baixa, herda as classes mais acima, por exemplo. As classes inferiores são todas Animais, pois herdam da classe-pai, e alguns herdam atributos e métodos da classe-filha acima que classifica alguns animais como características das classes herdadas do segundo nível.

1. **Sobrescrita**

Acontece quando precisamos **reescrever um método herdado pela classe-pai**. Nesse caso, o novo *método deve ter a mesma assinatura da classe pai*.

- Possuir o mesmo nome: **FELINO**  
- Retornar o mesmo tipo de dados: **Nome Atributo**(**char**), **Nome Atributo**(**inteiro**), **Nome Atributo**(**char**), **Nome Atributo**(**char**), **Nome Atributo**(**char**).  
- Recebe a mesma lista de parâmetros: **MÉTODOS(?)**.

1. **Polimorfismo**

Seria uma forma de **tratar as classes e sub-classes**, ***através de sua super-classe***.

Exemplo: Uma turma, onde o aluno é o 5º da segunda fileira. Para falar com esse aluno, a forma direta seria chamando-o pelo seu nome. Mas para proceder dessa forma, teríamos que saber o nome de cada aluno, ou seja, tratar nominalmente. Então a forma como se refere ao aluno, mesmo não sabendo seu nome é que faz a diferença (apontado para o aluno), mesmo aluno sendo um termo genérico.

Então, Aluno caberia para todos, mas o aluno em específico saberia que se trata dele pela forma como nos referimos a ele.

Mas a forma de fazer a chamada, serviria para todos os alunos. Daí o termo **Poli = Diversa + Morfismo = Forma.**

**Conceito**: Então, Polimorfismo permite que **referências de tipos** de *classes mais abstratas* *representem o comportamento das classes concretas que referenciam*.

É a capacidade de **Objetos instanciados de classes diferentes** usar um **mesmo método**, mas cada qual da sua maneira.

A superclasse define o comportamento esperado (interface pública) de todas as subclasses.

Programa na interface pública.

|  |
| --- |
| **Forma** |
| -cor: String |
| **+getArea():double** +toString():String |

Superclasse fornece as implementações reais

|  |  |  |
| --- | --- | --- |
| **Retangulo** |  | **Triangulo** |
| -altura:int -largura:int | -base:int -largura:int |
| **+getArea():double** +toString():String | **+getArea():double** +toString():String |

1. **Classes e Métodos Abstratos**

* **Classe Abstrata**: É toda classe que **não possui a capacidade de gerar objetos**, servindo apenas como **modelo para uma classe concreta**.
* **Métodos Abstratos**: Devem ser **implementados na classe concreta** que **estenda a classe na qual foi declarado na qual pode ter estendida**. Um método abstrato, somente podem ser definidos em classes abstratas.
* **Representa entidades e conceitos abstratos**
* **Não possui instancias**.
* **Template para classes derivadas**.
* **Normalmente possui métodos abstratos**.

É uma classe que não será objeto, ou seja, é uma classe que foi feita pra ser estendida.

Exemplo:

- Massa de uma Pizza  
- O que compramos é a pizza. A Massa serve de base para a pizza.

|  |
| --- |
| ***Animal*** |
| +habitoAlimentar: Setring = “”  +reproducao: String = “”  +pesoMedio: Double = 0.0 |
| +detelheAnimal(): void |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Peixe** |  | **Cavalo** |  | **Aguia** |
|  |  | - raca: String = “” - corPelagem: String = “” |  |  |
|  |  |  |  |  |

Peixe, Cavalo e Águia não são instancias de Animal, elas apenas herdam os atributos e métodos da classe animal que apenas serve como base e não pode ser instanciada como objeto.

1. **Classe Final**

É uma classe que não pode ser herdada.

1. **Método Final**

Seguindo a mesma lógica, o método final é um *método concreto que não podem ser sobrescritos*.

1. **Atributos Finais**

São atributos que se comportarão como constantes, ou seja, não poderão ter seus valores alterados.

1. **Interfaces**

São classes puramente abstratas, quando possuem atributos estes devem ser declarados como finais.

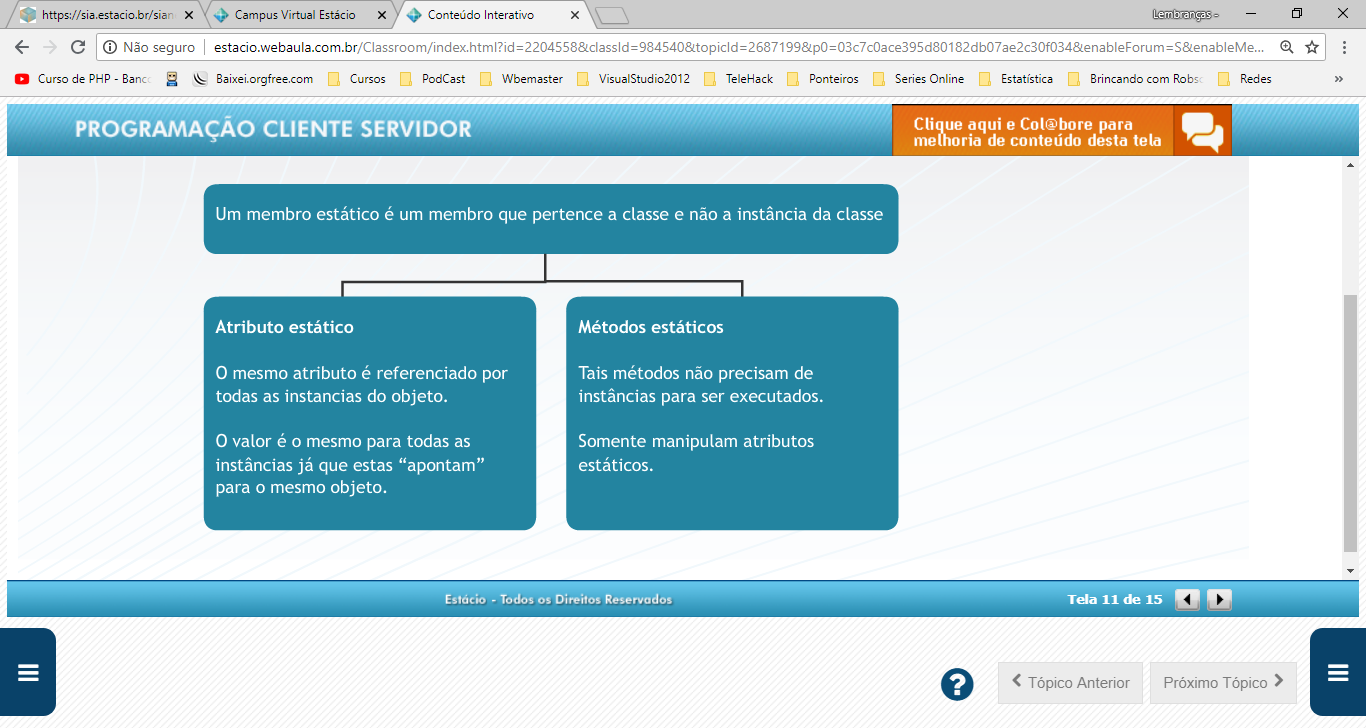
São utilizadas para ***definir padrões para especificação do comportamento de classes***.

1. Visibilidade de Membros de Classes (atributos)

* **Pública**: Podem ser *acessadas por qualquer objeto*.
* **Privada**: Podem ser *acessadas somente dentro da classe onde foi declarada*.
* **Protegida**: Podem ser *acessadas por qualquer objeto dentro do pacote no qual foi declarada* ou *fora desse pacote pelas classes que estendam a essa classe*.
* **Padrão**(**pacote**): Podem ser *acessadas somente por objetos que estejam no mesmo pacote em que a classe foi declarada*.

1. Membros de classe estáticos

Além dos *modificadores de visibilidade*, os *membros de uma classe* podem ainda ser *definidos como estáticos*.



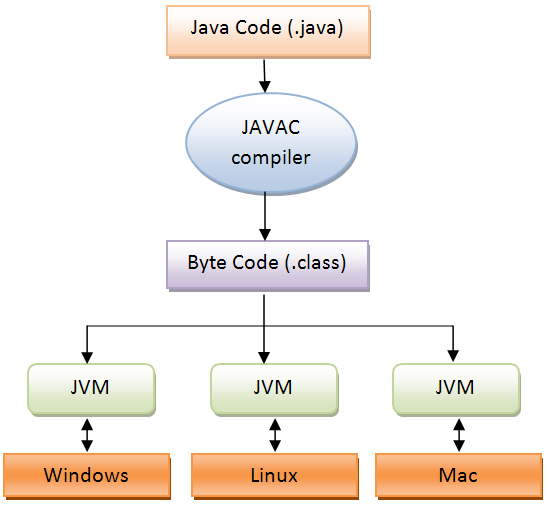
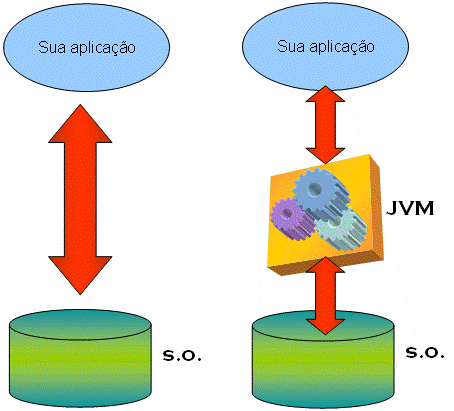
1. **A Linguagem JAVA**

“Write once, run anywhere” = Escreva uma vez e rode em qualquer lugar.

A ideia é escrever um código único e não precisar ter seus códigos alterados pelo simples fato de mudarmos de plataforma (sistema operacional).

1. **JVM = Java Virtual Machine**

É um programa que carrega e executa os aplicativos Java, convertendo os **bytecodes** em código **executável de máquina**.

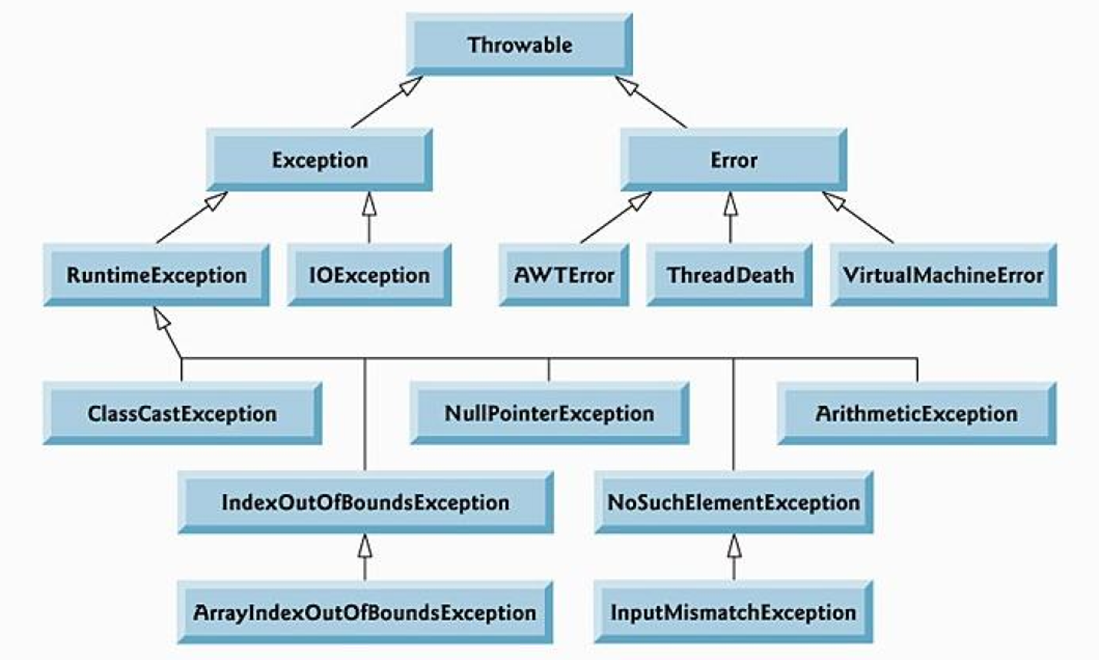


Qualquer sistema JAVA, roda em qualquer **S**istema **O**peracional com JVM específico.

1. **Exceptions em JAVA**

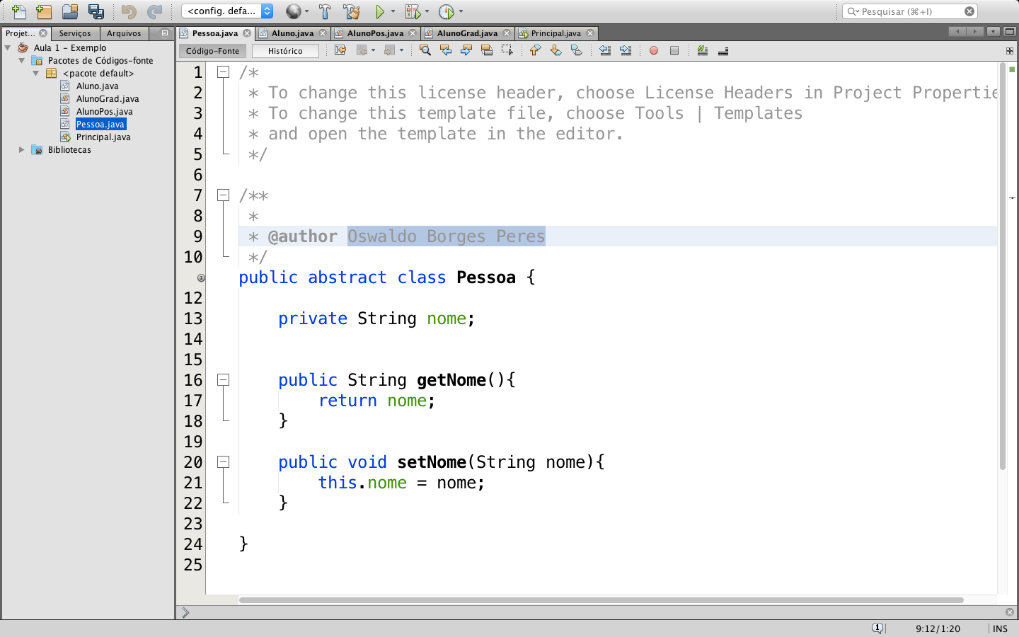
Existem dois tipos:

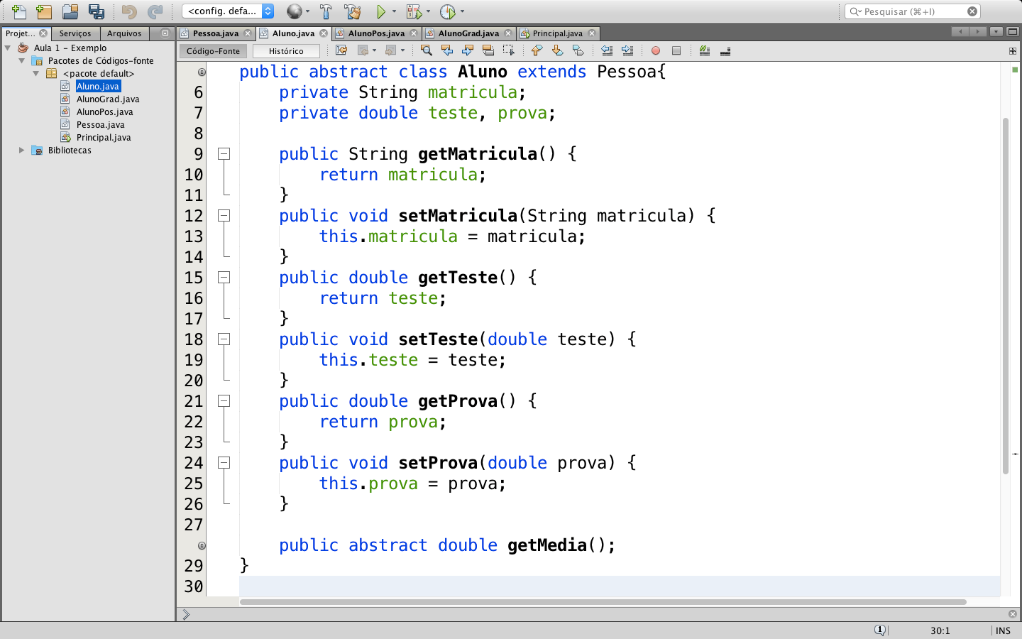
* Verificadas
* Não verificadas

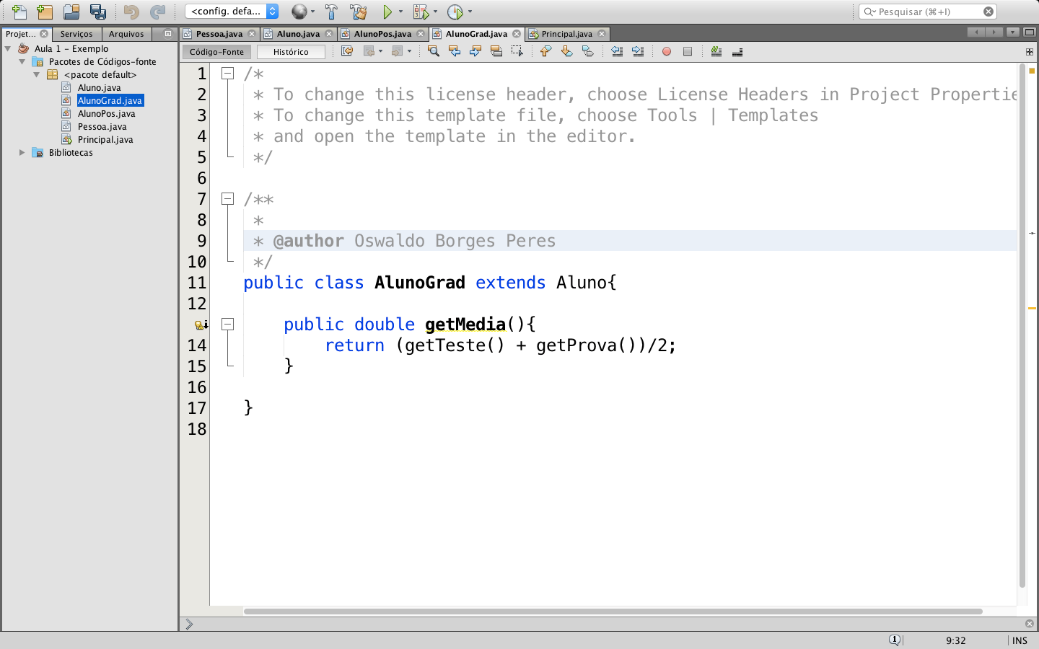


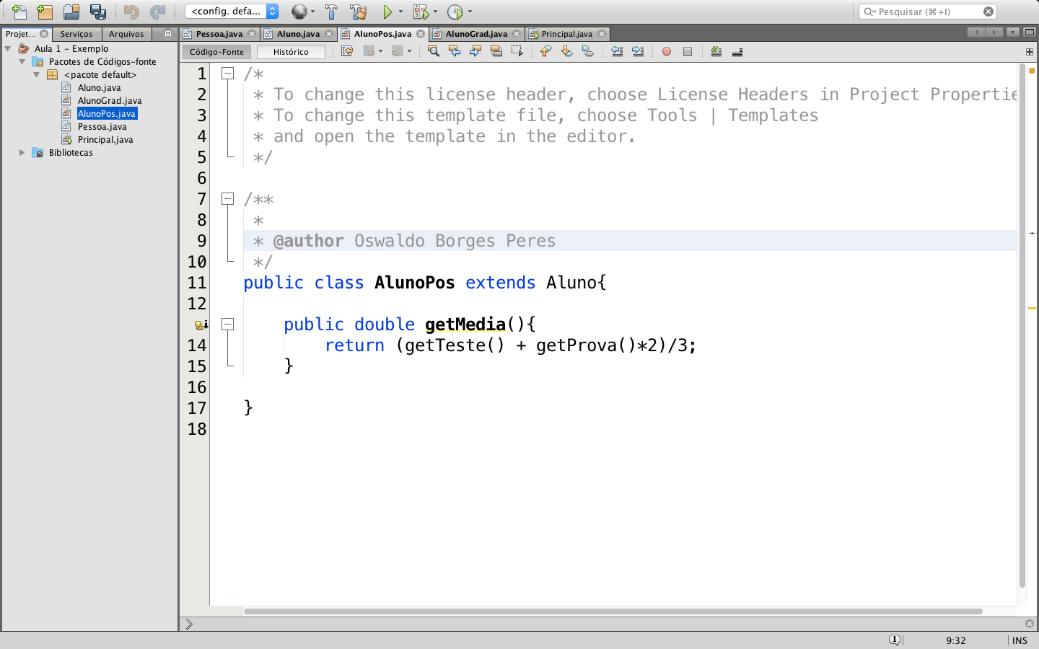
Obs.: Exemplos de Exception: Divisão com 0, posso controlar com IF, mas um erro de acesso a banco não permite fazer tratamento (Não verificadas) outros, podemos fazer via código.

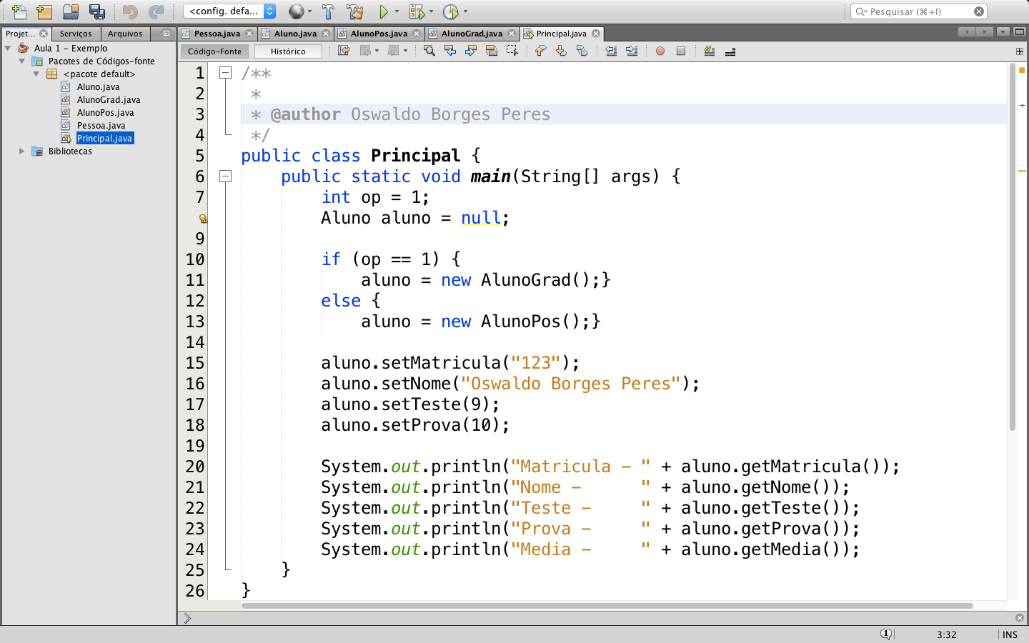
Exemplo:



1









**Bibliografia:**

Java: como programar - 8ª edição

Deitel, Paul J.; Deitel, Harvey M.

**Aula 2 – Para Internet**

Dividir um projeto em camadas é um método mais simples para quebrar a complexidade do desenvolvimento de um sistema. Geralmente as camadas mais altas usam os serviços da camada imediatamente inferior, mas a camada inferior ignorar a existência da camada mais alta. Cada camada normalmente, esconde suas camadas mais inferiores das camadas de cima.

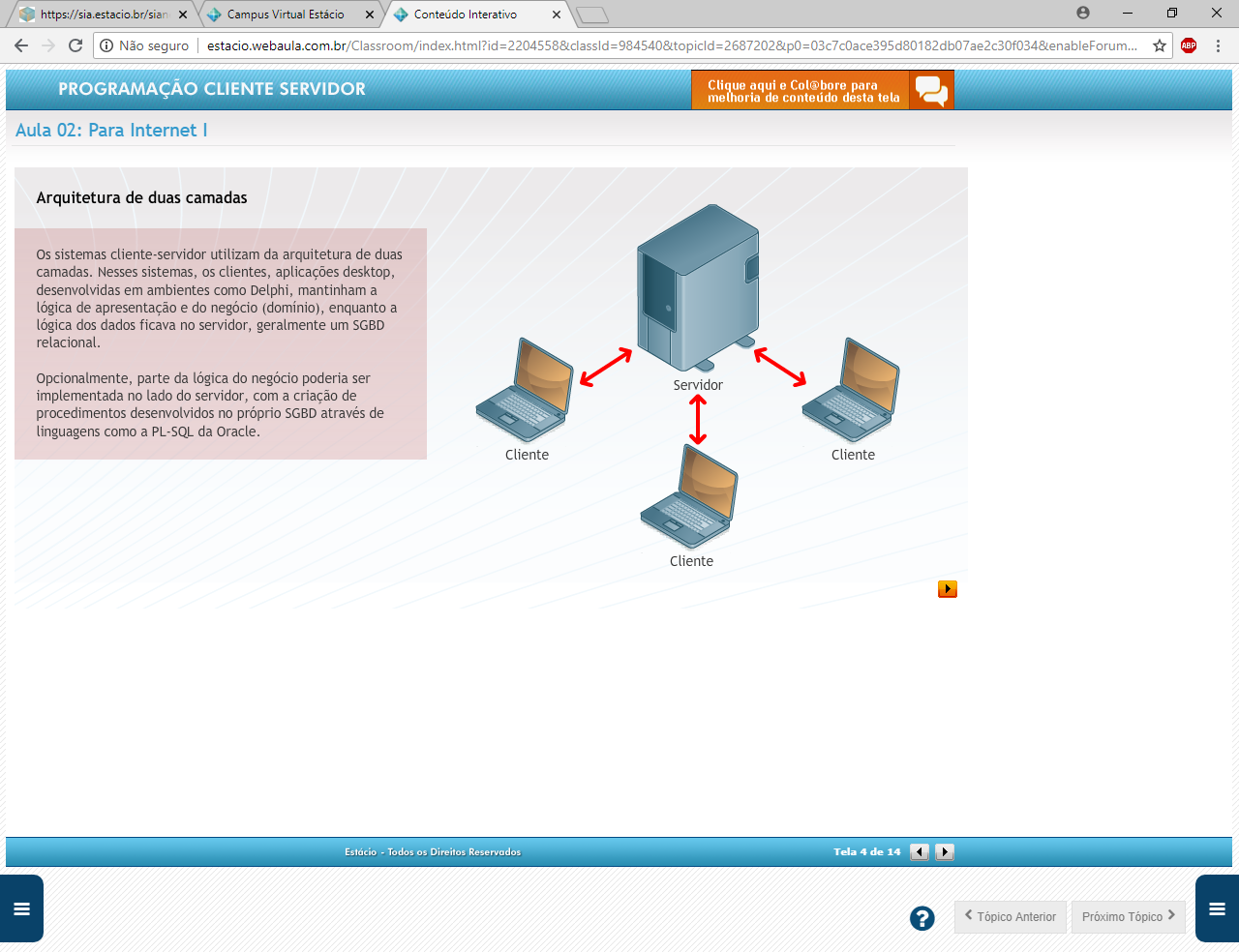
Vantagens em dividir sistemas em camadas:

* Podemos compreender uma camada coerentemente como um todo, sem precisar saber muito sobre as demais camadas.
* Uma camada pode ser substituída por uma camada por implementações alternativas dos mesmos serviços básicos.
* Menos dependências entre elas.
* Servem como padronização.
* Uma mesma camada pode ser usada por muitos serviços dos níveis mais altos.

1. **Arquitetura de Duas Camadas**

Um sistema Cliente-Servidor usa Arquitetura de Duas camadas.

Nesse tipo de sistema, a aplicação desktop mantinham-se nos computadores cliente, enquanto a lógica relacional ou de dados, geralmente eram mantidas em um servidos SGDB relacional.



Você tinha o sistema no Desktop e o Banco em um Servidor que mantinham a conectividade através de rede. Nesse caso a camada mais alta está no servidor. Isso era eficaz para aplicação de pequeno porte.

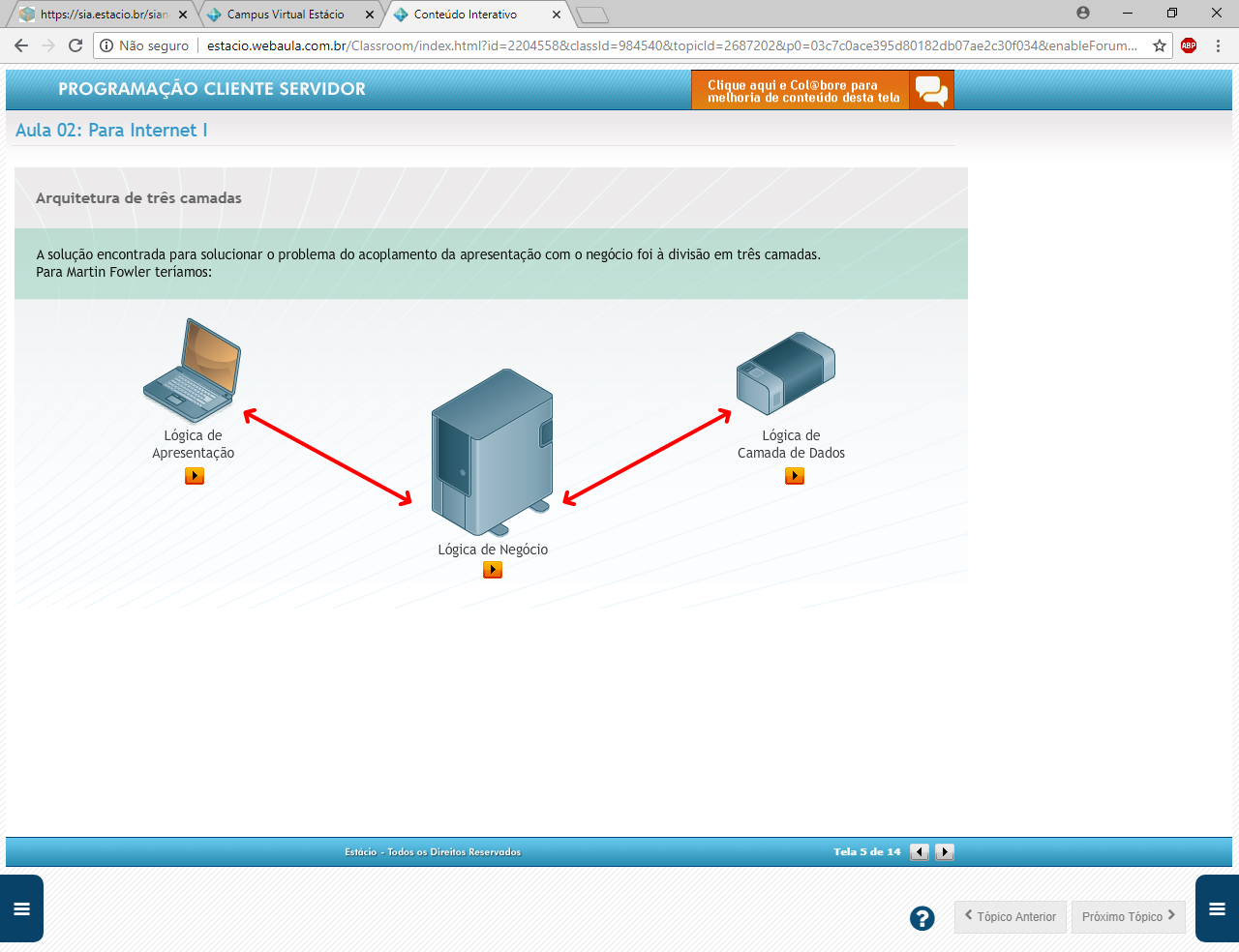
Em caso de uma empresa com maior usabilidade, essa arquitetura não era tão eficiente.

Imagine uma alteração no módulo de uma das aplicações em uma das máquinas clientes, essa mesma alteração teria que ser feita em cada máquina cliente. Em número maior de clientes, além do tempo, ocorria que algum dos clientes ficaria desatualizado.

1. **Arquitetura de Três Camadas**

Denominaremos **Programa na Máquina** **Cliente** como **Lógica de Apresentação** e **Servidor de Lógica de Négocio** (lógica relacionada com processo de negócios – venda, controle de inventário, etc).

A solução para resolver o problema de acoplamento entre Lógica de Apresentação e Lógica de Negócio foi a divisão de três camadas:



**Lógica de Apresentação**: É a camada onde tratamos a interação entre o usuário e o software.

**Lógica de Negócio**: Também conhecida como lógica de Domínio, é a inteligência da aplicação, ou seja, aquilo que o público alvo (stakeholders) espera que a aplicação faça. Isso inclui validação de dados de entrada armazenados, cálculos, e outras funções, dependendo inclusive dos comandos recebidos da camada de apresentação.

**Lógica de Camada de Dados**: Consiste no armazenamento dos dados persistentes da aplicação em um banco de dados.

1. **Lyer e Tier**

Normalmente encontraremos traduzido como Camadas.

A diferença é que:

Three Tier – Faz separação Física entre camadas.

Three Layer – Faz separação Lógica entre camadas.

1. **Aplicação Web com JAVA**

Uma aplicação Web é um Web Site onde os conteúdos são gerados dinamicamente antes de serem enviados para o Navegador.

1. **Java Enterprise Edition**

Plataforma Java usada para desenvolvimento de aplicações Web.

A plataforma Java Interprise Editon, possui diversas tecnologia que auxiliam os desenvolvedores, disponibilizando tecnologias como:

* Servlet.
* JavaServer Pages – JSP.
* A Standard Tag Library for JavaServer Pages.
* JavaServer Faces.

1. **O Web Container (Java Application Server)**

* É um ambiente que deve ser disponibilizado para os clientes que farão uso das aplicações Java Web.
* Podemos dizer que é um servidor capaz de tratar e manipular Servlets.
* Podemos dizer que é o ciclo de vida de toda aplicação Web (inicialização, chamada e destruição).

Podemos encontrar como principais servidores de aplicação Java disponíveis no mercado:

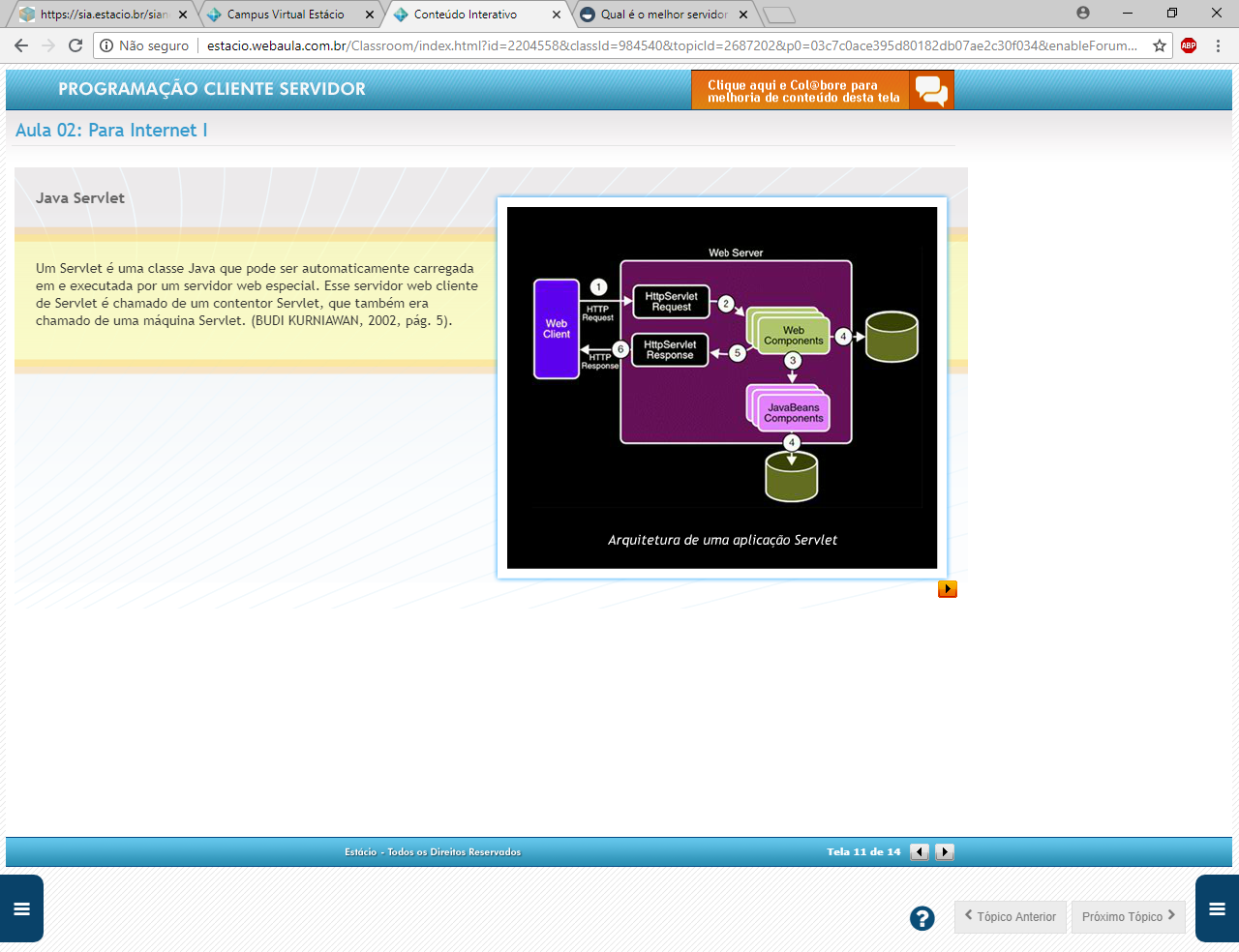
* JBoss Application Server.
* Apache Tomcat. (Web Container – melhor na opinião de alguns internautas)
* Jetty.
* WebLogic Server.
* GlassFish Application Server. (completo)

1. Arquitetura da Aplicação Web Java



1. Java Servlet

Um **Servlet** é uma **classe Java** que pode ser carregada e executada por um servido web especial. Esse **servidor web cliente** **de Servlet**  é chamado de um **contentor de Servlet** que também era chamado de **Servidor de Servlet**.



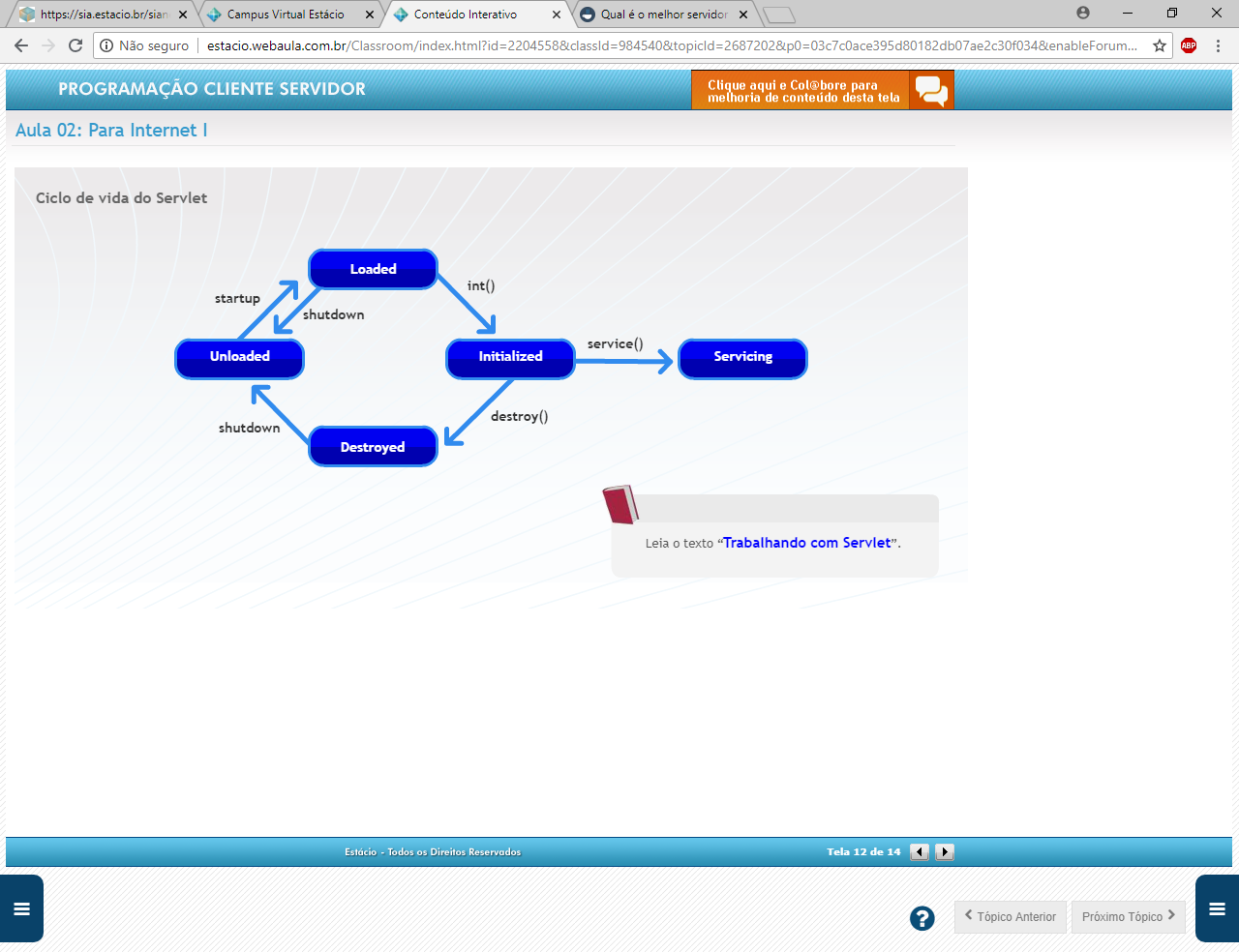
Quando digitamos um endereço no **Navegador**, o Browser envia essa solicitação para o **Servidor Web(JEE Web Container)**, que por sua vez passa o controle para o **Container** e o Container passa a interagir com a **Aplicação Web (Servlet)**

Por que usar Servlet?

Por serem basicamente classes Java, contam com as vantagens dessa linguagem; sendo uma tecnologia:

**Madura, Eficiente, Robusta, Segura, Independente de navegador e Multiplataforma**.

1. Ciclo de vida do Servlet



* [Introdução a Containers Web: A Primeira Aplicação Web com Servlets.](http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/servlets/intro.htm)
* [Tutorial Tomcat.](http://www.mhavila.com.br/topicos/java/tomcat.html)
* [Tutorial básico de Servlet.](http://javafree.uol.com.br/artigo/9127/Tutorial-basico-de-Servlet.html)
* [Introdução ao Tomcat e Servlets.](http://www.guj.com.br/articles/9)
* [Guia GlassFish de Início Rápido.](http://glassfish.java.net/downloads/quickstart/index_pt.html)
* [XML Tutorial](http://www.w3schools.com/xml/default.asp)
* [Segurança da informação, conceitos e mecanismos](http://www.oficinadanet.com.br/artigo/1307/seguranca_da_informacao_conceitos_e_mecanismos)
* [O que é CGI?](http://www.htmlstaff.org/ver.php?id=1659)

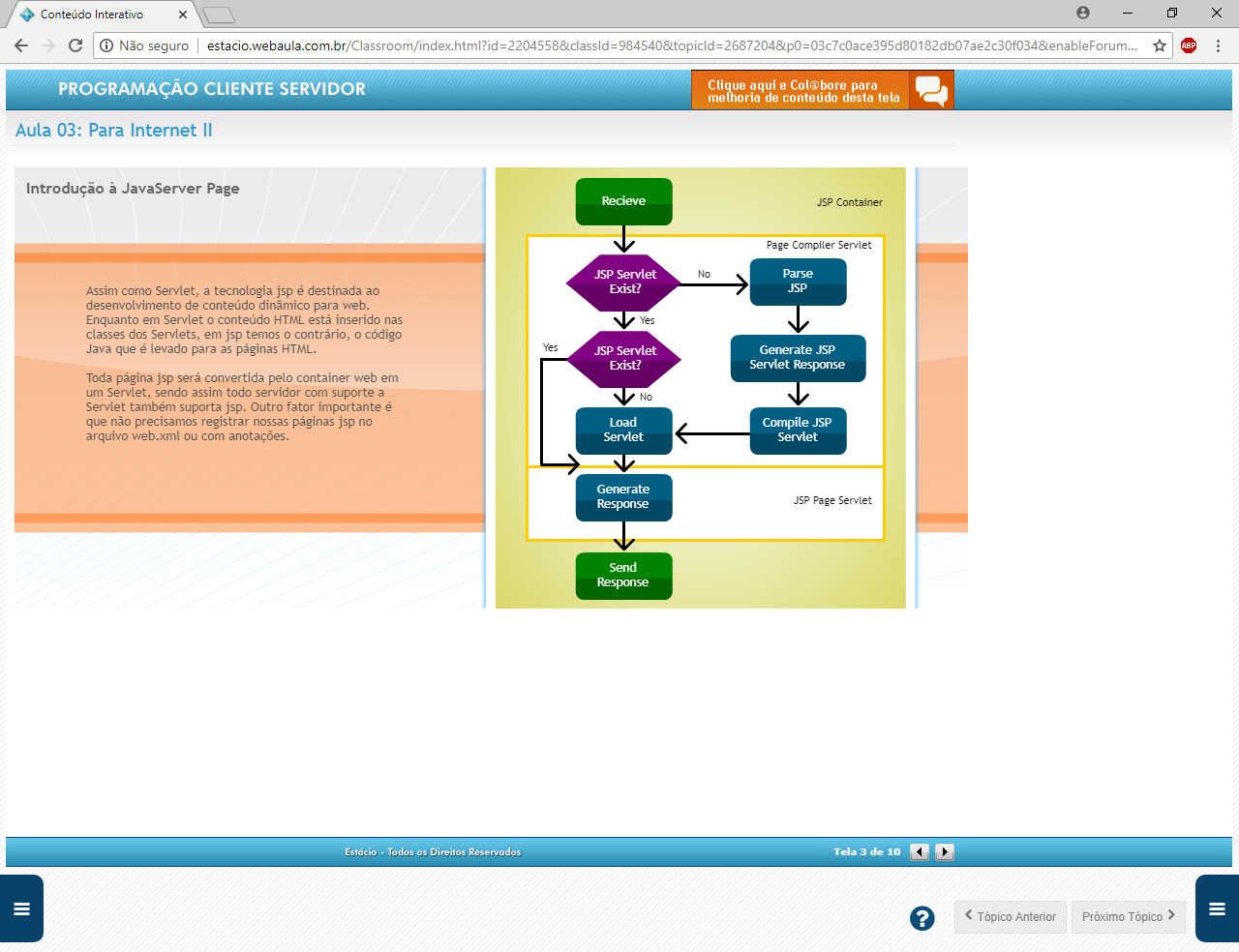
**Aula 3 – Para Internet II**

Introdução à Java Server Page

Assim como Servlet, também é uma tecnologia utilizada para desenvolvimento de conteúdo dinâmico em páginas Web.

Servlets = Tem o conteúdo HTML está inserido nas classes dos Servlets.  
JSP = Faz o contrário. Tem os códigos Java inseridos nas páginas HTML.

A página inteira será convertida pelo container Web em um Servlet, concluímos então que todo servidor com suporte Servlet é compatível com páginas JSP.

Outro fator importante é que não precisamos registrar as páginas JSP no arquivo web.xml ou com anotações.

1. **Criando Página JSP**

Assim como na aula de Servlet, utilizaremos o NetBeans como ambiente de desenvolvimento. Para as aulas de jsp, crie um novo projeto nos mesmos moldes do projeto criado na aula anterior, mudando somente o servidor para GlassFish.

Lembrado que você poderia utilizar outro ambiente de desenvolvimento como o Eclipse, ou outro servidor web.

Observe que, como no projeto anterior a IDE do NetBeans, já cria uma página jsp de boas vinda que pode ser utilizada para testar nosso ambiente. Execute o projeto para verificar se tudo está correto.

1.2. **Comentários**

<%-- Comentário --%>

Ao criar um comentário jsp, o conteúdo inserido entre suas tags não é enviado ao cliente, servindo apenas como uma forma do programador documentar o arquivo que está editando.

Note que no exemplo criado automaticamente pelo NetBeans as primeiras linhas são um comentário HTML.

<%--

    Document   : index

   Created on : XX/XX/XXXX, XX:XX:XX

   Author     : XXXXXX

--%>

1.3. **Expressões**

As expressões são usadas para inserir no documento expressões válidas criadas em Java.

Você pode escrever diretamente a expressão, ou pode utilizar uma variável para referenciar seu resultado.

<%= expressão %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<html>

<html>

   <head>

       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

       <title>Expressões</title>

   </head>

   <body>

       <h1>Utilizando expressões!</h1>

       Valor obtido apos o lançamento de um dado: <%= 1+(int)( 6\*Math.random() ) %>

   </body>

</html>

1.4. **Declaração**

As declarações permitem que sejam declarados **atributos** e/ou **métodos** que podem ser acessados em toda a página JSP. Tais membros equivalem às variáveis e métodos pertencente à instância de classe caso fossem declarados em um Servlet convencional.

<%! declarações %>

<%!  
 String texto = "Valor obtido apos o lançamento de um dado:";  
 int gerarValor(){  
 return 1+(int)( 6\*Math.random() ) ;  
 }  
%>

<html>  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
 <title>Expressões</title>  
 </head>  
<body>

       <h1>Utilizando expressões!</h1>  
         <%= texto%>  
         <%= gerarValor()%>  
    </body>  
</html>

1.5. **Scriptlets**

Corresponde a estrutura, onde o código java deverá ser embutido.  Em um Servlet, o código existente nos scriptlets estaria nos métodos doGet e doPost.

<%!  
 String nomes[] = {"João","Maria","Pedro","Ana","Paulo","José","Carla"};  
%>

<html>  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
 <title>Scriptlets</title>  
</head>  
<body>  
 <h1>Utilizando scriptlets!</h1>  
 <table>  
 <tr>  
 <td>Nomes dos alunos</td>  
 </tr>  
 <%  
          for(String nome:nomes){  
        %>  
 <tr>  
 <td><%=nome%></td>  
 </tr>  
 <%}%>  
 </table>

</body>

</html>

6. **Diretivas**

São **marcações utilizadas para indicar ao container como tratar a página em questão**. Uma diretiva é composta por um **nome descritor** e, pelo menos, **um atributo valorado**.

* 1. Diretiva include: Através dessa diretiva, podemos inserir texto existente em outro arquivo em uma página jsp.  
     O conteúdo inserido pode ser de qualquer tipo, HTML, Java, CSS... É comum utilizamos a diretiva include para suporte a utilização de templates.

<%@include file="url-relativa"%>  
 cabecalho.html  
 <h1 style="font-variant: small-caps; color: navy;"> Universidade Estácio de Sá</h1>  
index.jsp  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
 <title>Diretiva include</title>  
</head>  
<body>  
 [%@include file="cabecalho.html" %](mailto:%25@include%20file=%22cabecalho.html%22%20%25)  
</body>  
</html>

* 1. Diretivas Page

São informações de **configuração especificas do conteúdo Java existente na página**, isto é,  são as configurações indispensáveis para geração do Servlet correspondente.

 <%@page atributo1="valor1" atributo2="valor2" ... atributoN="valorN" %>  
É possível utilizar mais de uma diretiva de página, contudo cada uma com seus próprios atributos.  
  
<%@page atributo1="valor1"%>  
<%@page atributo2="valor2"%>  
<%@page atributoN="valorN"%>

Segue abaixo a listagem dos atributos possíveis para a diretiva page:  
autoFlush, buffer, contentType, errorPage, extends, import, info, isErrorPage, isThreadSafe, language, pageEnconding, isELIgnored, session

* 1. Diretivas de Taglib

Essa diretiva indica quais **bibliotecas de tags** customizadas iremos utilizar em nossa página.  Nesse caso, dois atributos devem ser informados: o prefixo que usaremos para acessar as tags e a url que indica a localização do descritor da biblioteca a ser usada.

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

1. **Criando um sistema simples de Login**

* **index.jsp**

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Login</title>

</head>

<body>

<form action="logar.jsp" method="post">

<fieldset>

<legend>Sistena de login</legend>

<table>

<tr>

<td>Login:</td>

<td><input type="text" name="login"/></td>

</tr>

<tr>

<td>Senha:</td>

<td><input type="password" name="senha"/></td>

</tr>

<tr>

<td><input type="submit" value="Enviar"/></td>

<td><input type="reset" value="Limpar"/></td>

</tr>

</table>

</fieldset>

</form>

</body>

</html>

* **Logar**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="java.util.\*" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>JSP Page</title>

</head>

<body>

<%

//insere três usuários no HashMap

HashMap<String, String> usuarios = new HashMap<String, String>();

usuarios.put("ana","zasxcd");

usuarios.put("joao","1qw3e2");

usuarios.put("maria","123ewq");

//pega os parâmetros enviadas pelo formulário

String login = request.getParameter("login");

String senha = request.getParameter("senha");

//valida usuário

if(login != null && senha != null && usuarios.get(login) != null && usuarios.get(login).equals(senha)){

session.setAttribute("login",login);

response.sendRedirect("principal.jsp");

}else{

%>

<h2 style="color: red">Usuário ou senha inválido</h2>

<a href="index.jsp">Clique aqui para voltar</a>

<%

}

%>

</body>

</html>

* **Principal**

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>JSP Page</title>

</head>

<body>

<%

String login = (String) session.getAttribute("login");

if(login == null){

response.sendRedirect("logar.jsp");

}else{

%>

<h3>Olá <%=login%> você está logado(a) no nosso sistema!</h3>

<%

}

%>

Para sair clique <a href="sair.jsp">Aqui</a>

<a></a>

</body>

</html>

* **Sair**

<%

try{

session.invalidate();

}catch(Exception e){

}

response.sendRedirect("index.jsp");

%>

1. **Aprimorando a Segurança**

Com o intuito de aumentar a segurança, as senhas podem estar criptografadas ou processadas através de um algoritmo de hash. Com isso, ao persistirmos nossas informações, os dados passados originalmente seriam convertidos em um código que, mesmo que um usuário mal intencionado tivesse acesso, não teria utilidade ao mesmo.

Utilizaremos uma instância do objeto MessageDigest, pertencente ao pacote java.security, para converter nossas senhas, utilizando o algoritmo de hash MD5.

**Ver arquivo (com código):**

[**Aula 3 - Passos do aprimoramento de segurança**](Materiais%20Didáticos/Aula%203%20-%20Passos%20do%20aprimoramento%20de%20segurança.pdf)

1. **Java Beans**

JavaBeans **são componentes de software escritos na linguagem Java**, sendo componentes reutilizáveis de software. Um **JavaBean** é uma classe normal (POJO) que deve seguir as seguintes  convenções:

* Implementar a interface java.io.Serializable.
* Possuir um construtor sem argumentos.
* Seus atributos devem ser privados.
* Deve ser fornecido um "get" e "set" para cada um desses atributos.
* Um JavaBean pode conter qualquer método, desde que siga as especificações anteriores.

**Ver arquivo (com código):**

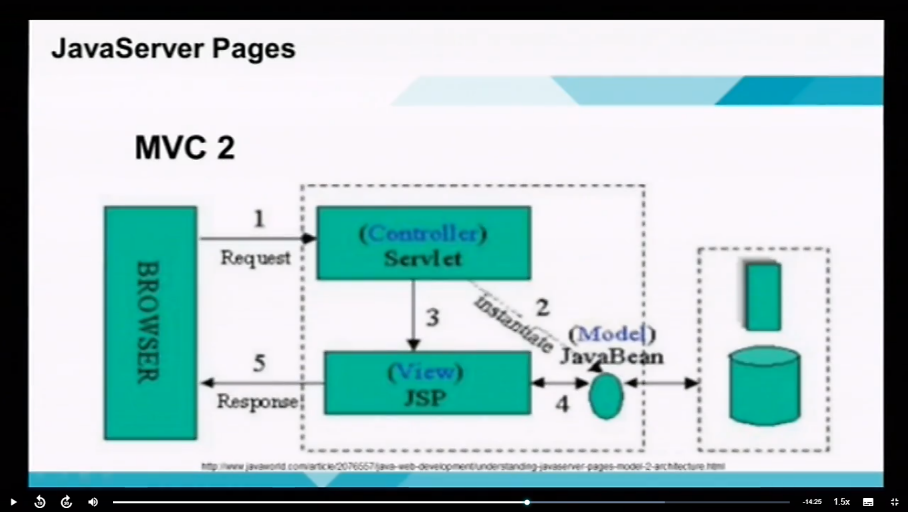
[**Aula 3 – Trabalhando com Javabeans**](Materiais%20Didáticos/Aula%203%20-%20Java%20Beans%20-%20Entenda.pdf)

**Resumo Final:**

- A estrutura do JSP é focado em View e não em controle, embora devo fazer.

- Padrão de Projeto MVC – Separa o Modelo, do View e do Controle.

- Usaremos o modelo de projeto MVC2 onde:



- JSP é um servilet após compilado, voltado para visualização, enquanto o Servlet é voltado para Controller.

Aula 03: Para Internet II

http://estacio.webaula.com.br/cursos/gon964/imagens/pixel_cinza.gif

Para saber mais sobre os tópicos estudados nesta aula, pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Se ainda tiver alguma dúvida, fale com seu professor on line, utilizando os recursos disponíveis no ambiente de aprendizagem.

* [Desenvolvendo sites dinâmicos usando JSP (Java Server Pages).](http://www.arquivodecodigos.net/arquivo/tutoriais/jsp/desenvolvendo_jsp_1_1.php)
* [Objetos implícitos em JSP.](http://neylorsousa.blogspot.com/2008/06/objetos-implicitos-em-jsp.html)
* [Conhecendo as diretivas JSP. Ciclo de vida de um JSP.](http://www.devmedia.com.br/post-5397-Conhecendo-as-diretivas-JSP.html)
* [JSTL - JSP Standard Tag Library.](http://yross.wordpress.com/2009/09/11/ciclo-de-vida-de-um-jsp/)

**Aula 4 – Para Internet II**