

Faculdades Integradas Santa Cruz

Tutorial de Java – MVC – WEB

Criando uma tela de Login com Banco de Dados

2009

Introdução

O objetivo deste tutorial é apresentar os conceitos de estruturação de um projeto Web MVC utilizando para isso uma aplicação exemplo baseada em uma simples tela de login.

Descrição do Sistema

O sistema consiste em uma tela simples (Web) onde o usuário deve informar seu login e sua senha para ter acesso. Caso o usuário informe um login ou uma senha incorreta, uma mensagem deve ser apresentada para ele, possibilitando que ele faça uma nova tentativa. Caso o usuário informe um login e senha válidos, o sistema deve redirecioná-lo para uma tela de boas vindas, onde será apresentada uma mensagem do tipo: "... Olá <Fulano>! Seja bem vindo ao nosso sistema!

Modelagem do Sistema

Abaixo, seguem alguns diagramas que representam as funcionalidades do sistema proposto.

Modelo de Caso de Uso

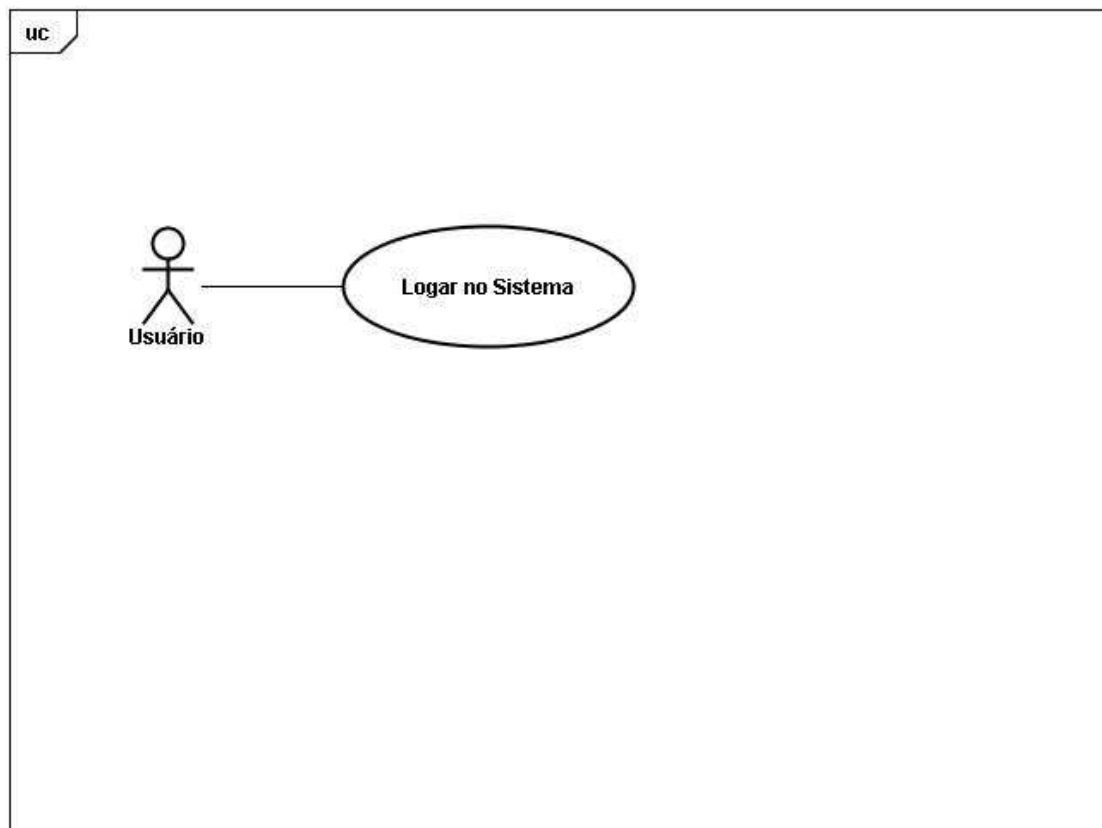
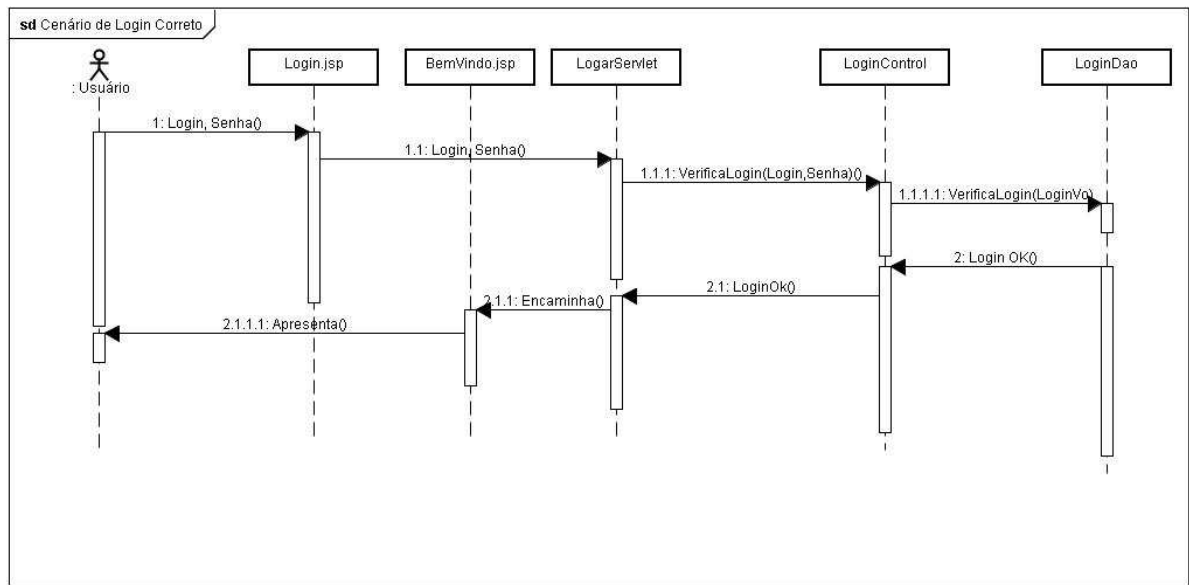


Diagrama de Seqüência

Este diagrama não segue todas as formalidades necessárias previstas na UML. O objetivo dele é de apresentar de maneira sucinta os dois cenários possíveis da aplicação proposta.

Cenário de Login OK



Cenário de Login Não OK

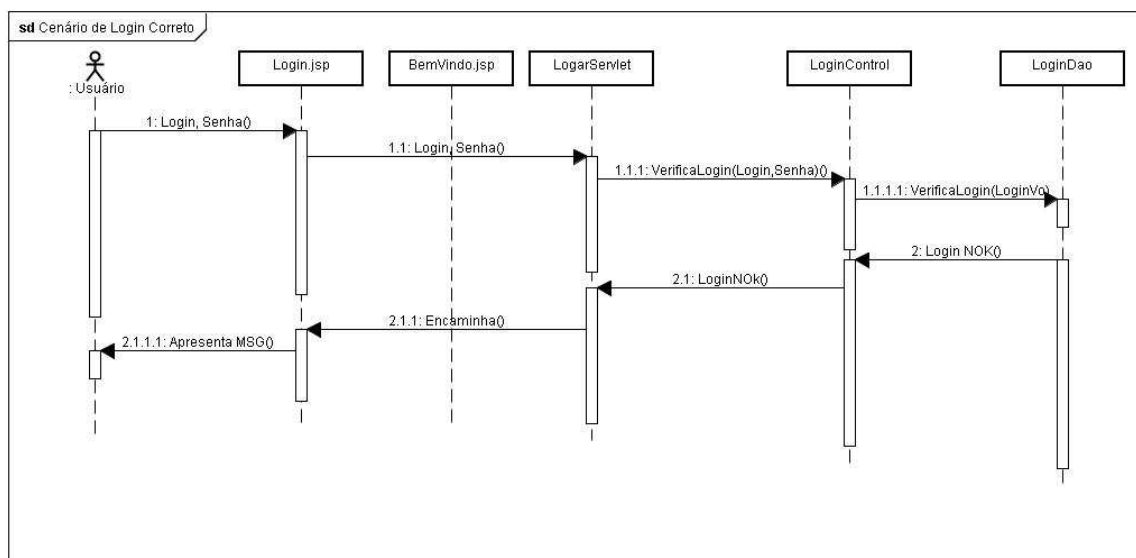
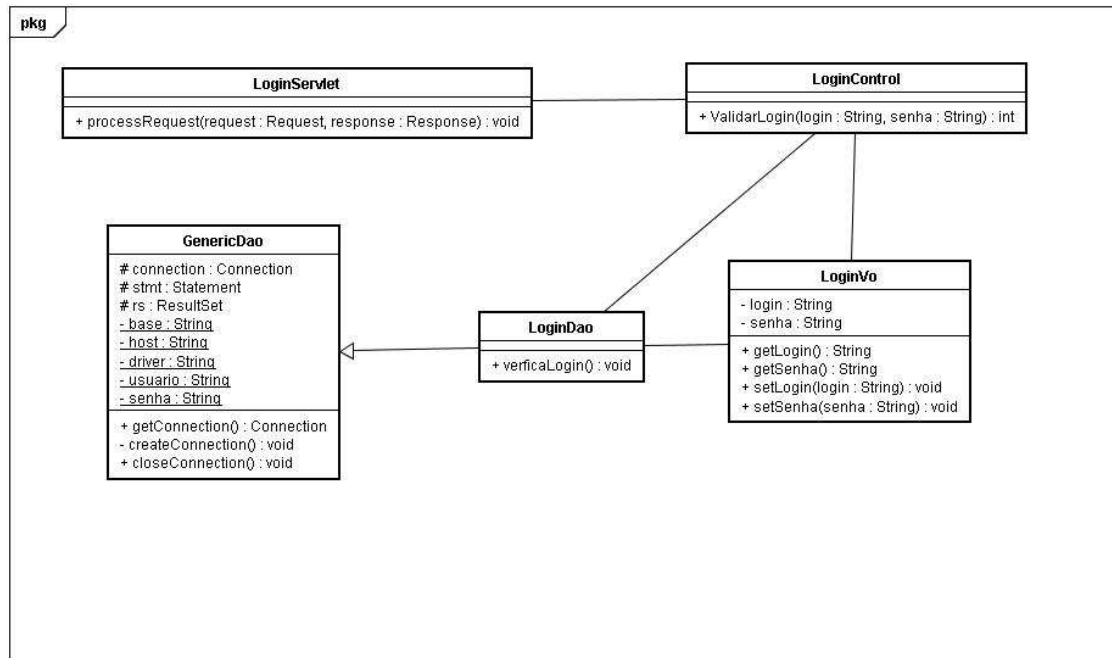


Diagrama de Classes

É importante lembrar que neste diagrama de classes, não foi representada a divisão de pacotes e também não foi demonstrada as *views* representadas por jsps.



Criando a Aplicação

Para iniciarmos a criação da aplicação faz-se necessário compreender alguns conceitos de desenvolvimento Web com Java.

Plataforma e Requisitos Técnicos

Para o desenvolvimento desta aplicação utilizaremos a IDE Eclipse :

www.eclipse.org

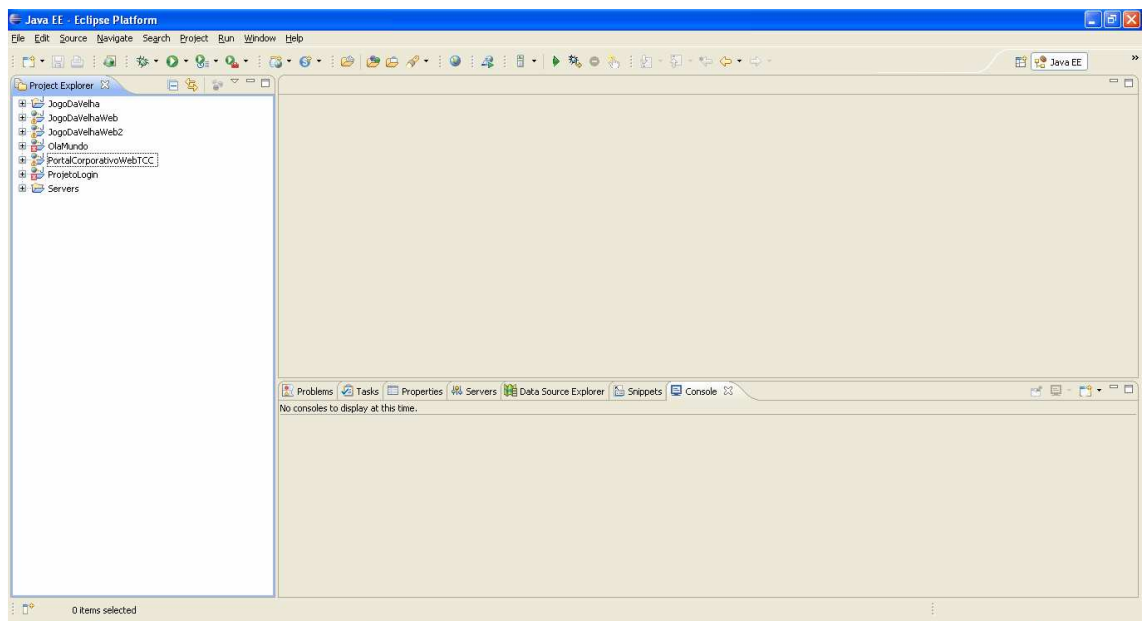
Também utilizaremos a JDK 1.5 ou 1.6 : www.sun.com

Para armazenamento de dados utilizaremos o Mysql : www.mysql.com

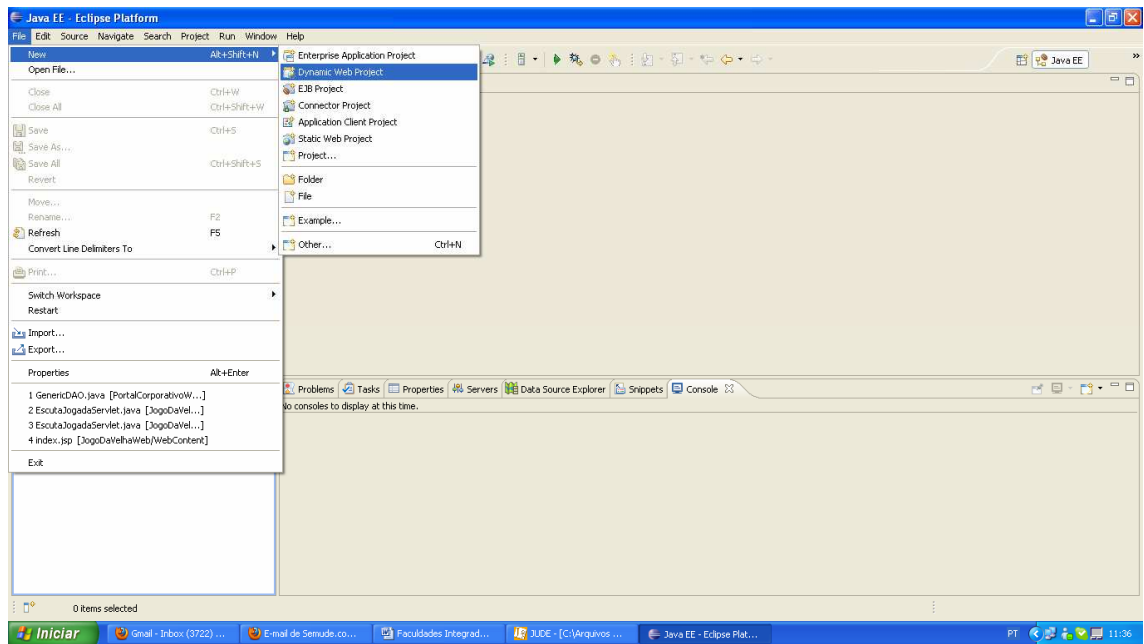
Criando a Estrutura Básica do Projeto

Abaixo descreveremos o passo a passo para criarmos a estrutura básica do projeto Web no Eclipse.

1. Abra o Eclipse



2. No menu File, vá em New -> Dynamic Web Project



3. Na tela seguinte, digite o nome do projeto : “LoginMvc”

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: LoginMvc

Contents
☒ Use default
Directory: C:\Projetos\SC\eclipse\LoginMvc Browse...

Target Runtime
Apache Tomcat v6.0 New...

Dynamic Web Module version
2.5

Configuration
Default Configuration for Apache Tomcat v6.0 Modify...
A good starting point for working with Apache Tomcat v6.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR Membership
☐ Add project to an EAR
EAR Project Name: EAR New...

? < Back Next > Finish Cancel

4. Caso o campo Target Runtime esteja em branco siga os procedimentos abaixo
5. Clique no botão New, ao lado do campo

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: LoginMvc

Contents
☒ Use default
Directory: C:\Projetos\SC\eclipse\LoginMvc Browse...

Target Runtime
Apache Tomcat v6.0 New...

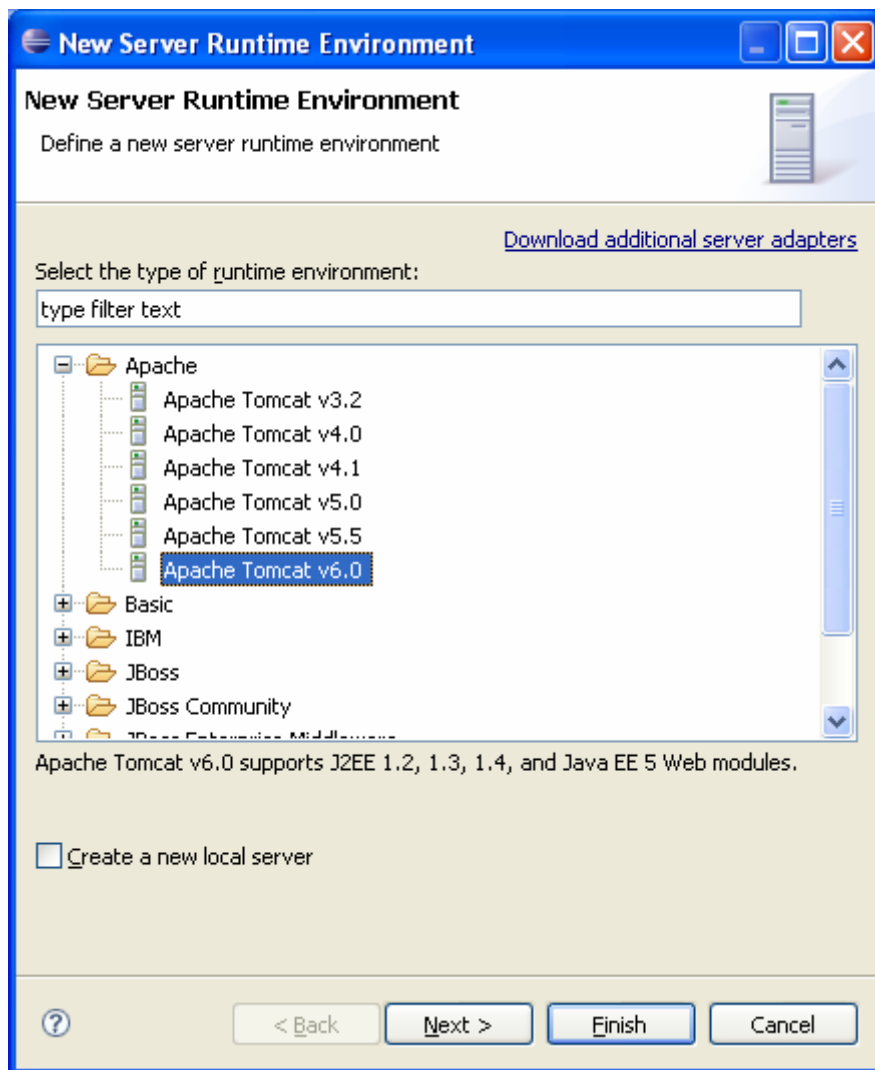
Dynamic Web Module version
2.5

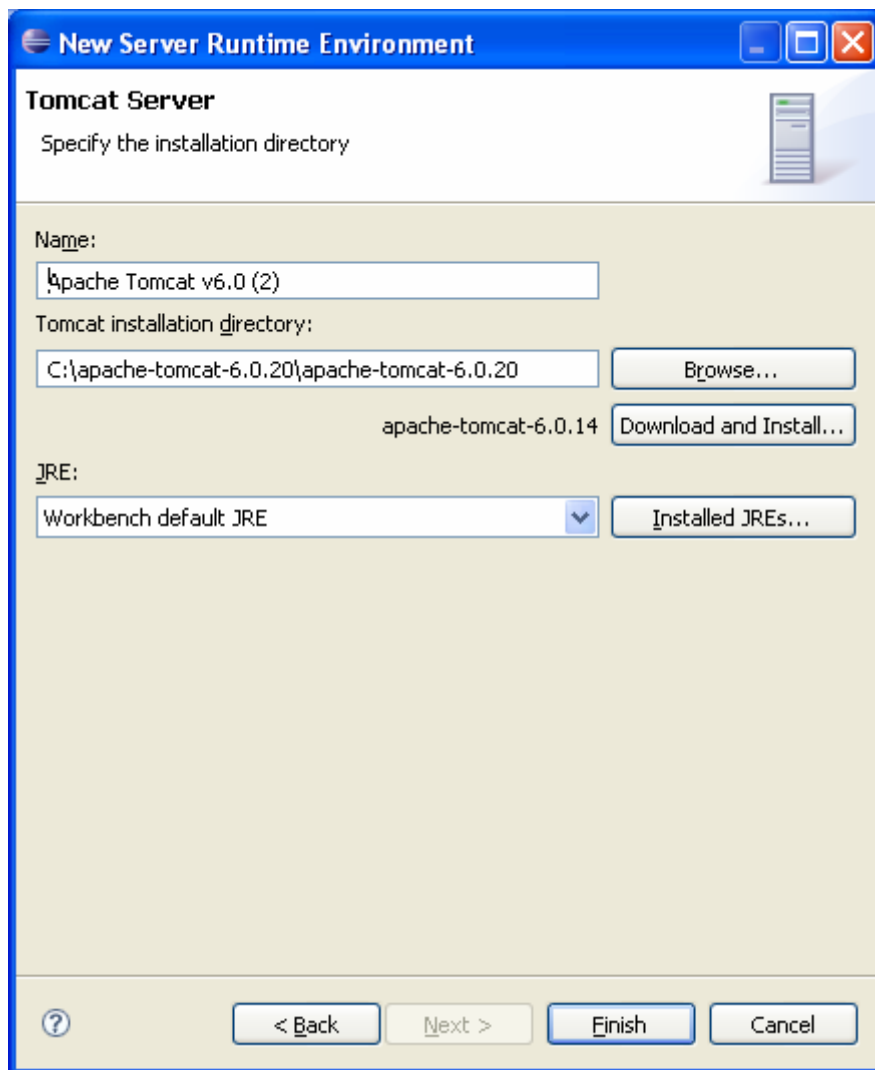
Configuration
Default Configuration for Apache Tomcat v6.0 Modify...
A good starting point for working with Apache Tomcat v6.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR Membership
☐ Add project to an EAR
EAR Project Name: EAR New...

? < Back Next > Finish Cancel

6. Direcione para o diretório Raiz do TomCat





7. Click em Finish para concluir a criação do projeto base

Criando a Separação em Camadas (MVC)

Como já é sabido de todos, uma aplicação bem estruturada deve ser construída utilizando a separação em camadas. Esta separação serve como organização e principalmente como reutilização de código.

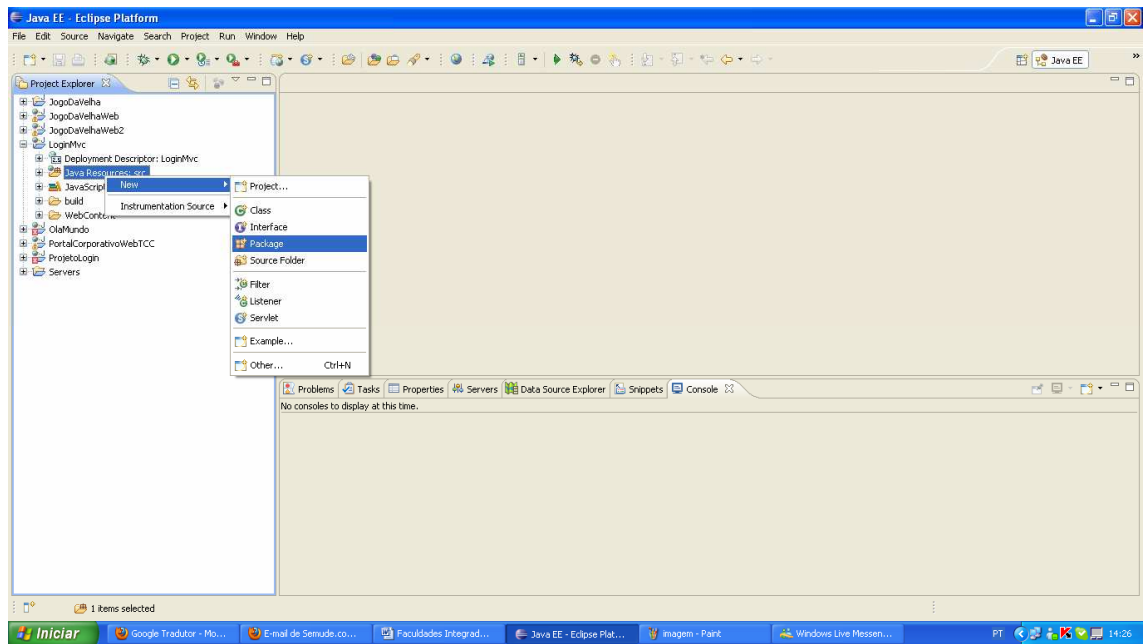
No Java, a divisão em camadas começa com a criação dos chamados *Pacakes*. *Pacakes* ou Pacotes nada mais são que pastas, criadas para dividir a aplicação fisicamente.

Em uma aplicação MVC tradicional, basicamente possuímos as seguintes camadas:

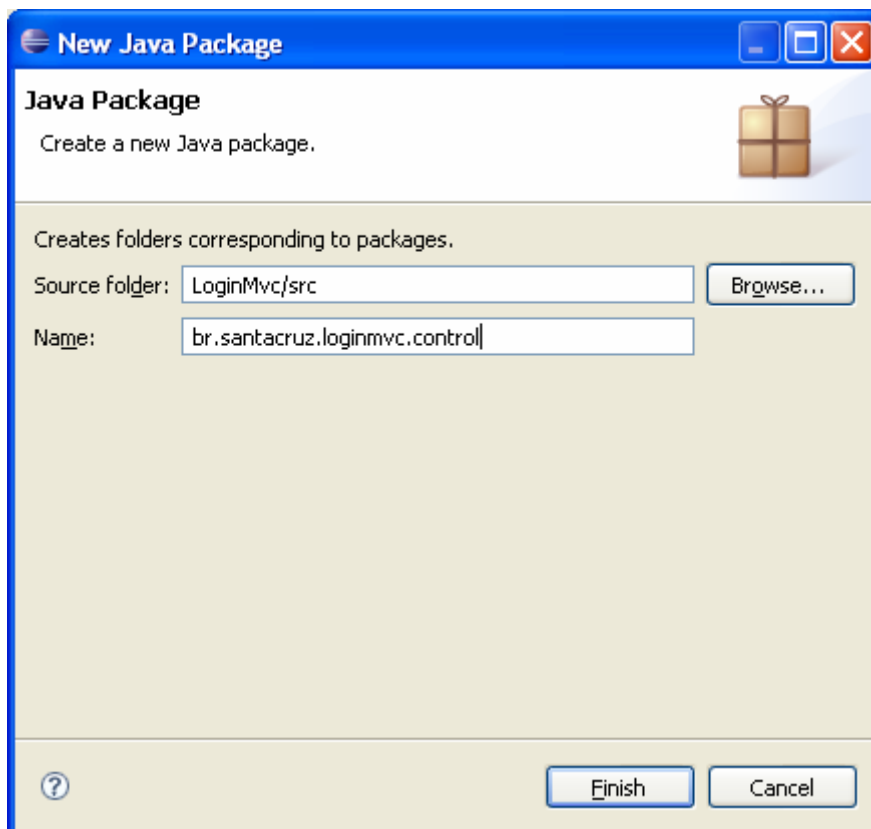
- Visualização: Camada responsável por toda a comunicação com o usuário. Em nossa aplicação Web, esta camada é representada por nossas páginas .jsps.
- Controle: Camada responsável por conter toda a lógica de nosso sistema. É nesta camada onde são feitas as validações e implementadas as lógicas de negócio. Normalmente em aplicações Java, esta camada é dividida em duas:
 - Servlets: Camada intermediária, responsável por obter os dados enviados pela camada de visualização e repassar para a camada de controle propriamente dita.
 - Controle: Esta é a camada de controle efetivamente.
- Modelo: Camada responsável por todas as tarefas de busca e armazenamento de dados em nossa aplicação. Em Java, esta camada também sofre novas divisões:
 - Dao: Camada responsável pela conexão com o banco de dados e todas as operações de persistência.
 - Vo: Camada responsável por conter Classes que representem o modelo de dados. Através delas é que o fluxo de informações trafegará entre as camadas da aplicação.

Criando pacotes em nossa aplicação

8. Na janela a esquerda, clique no + da árvore de sua aplicação para expandir, então click com o botão direito na pasta src, vá na opção New -> Package.



9. Digite o nome do pacote, usando a notação de DNS reverso. No nosso caso será `br.santacruz.loginmvc.control`. Repita os passos 8 e 9 para os pacotes de servlets, `dao`, `vo` como são apresentados nas figuras abaixo.



New Java Package

Java Package
Create a new Java package.

Creates folders corresponding to packages.

Source folder:

Name:

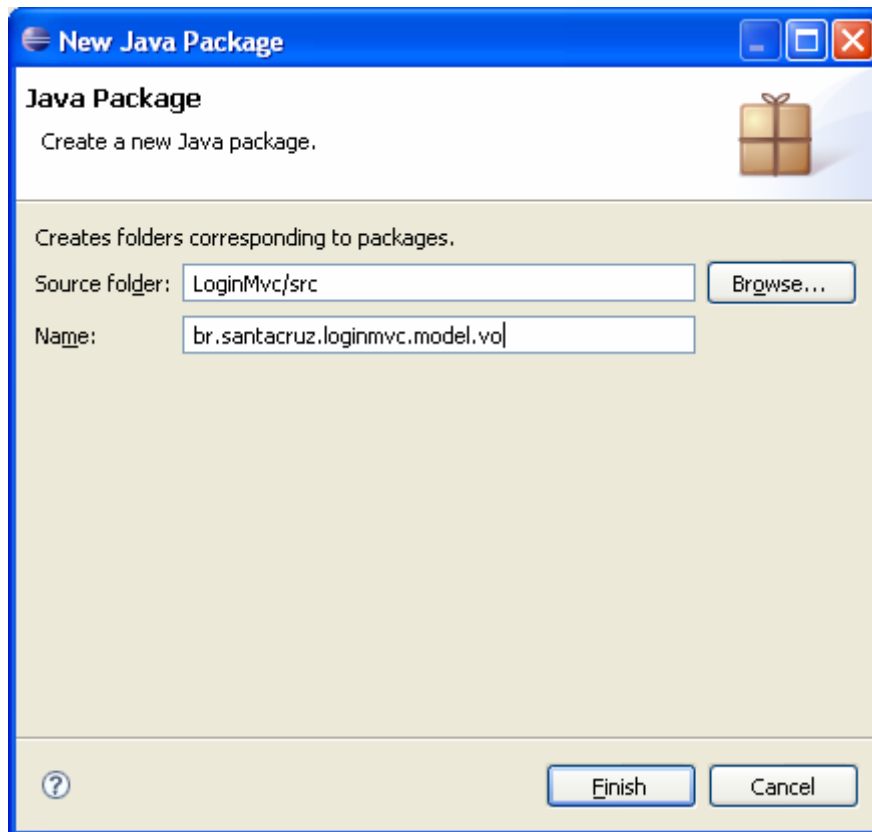
New Java Package

Java Package
Create a new Java package.

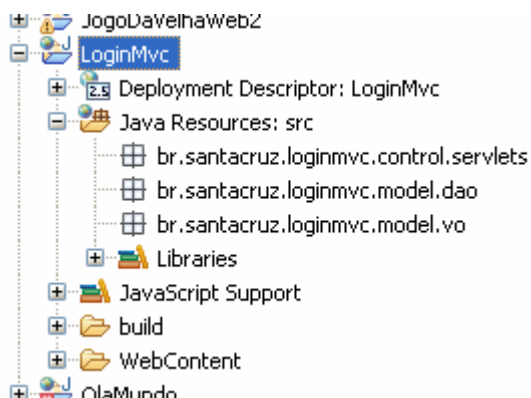
Creates folders corresponding to packages.

Source folder:

Name:



10. Após a criação dos pacotes, a estrutura de nossa aplicação deverá estar parecida com a figura apresentada abaixo.



Importando bibliotecas necessárias

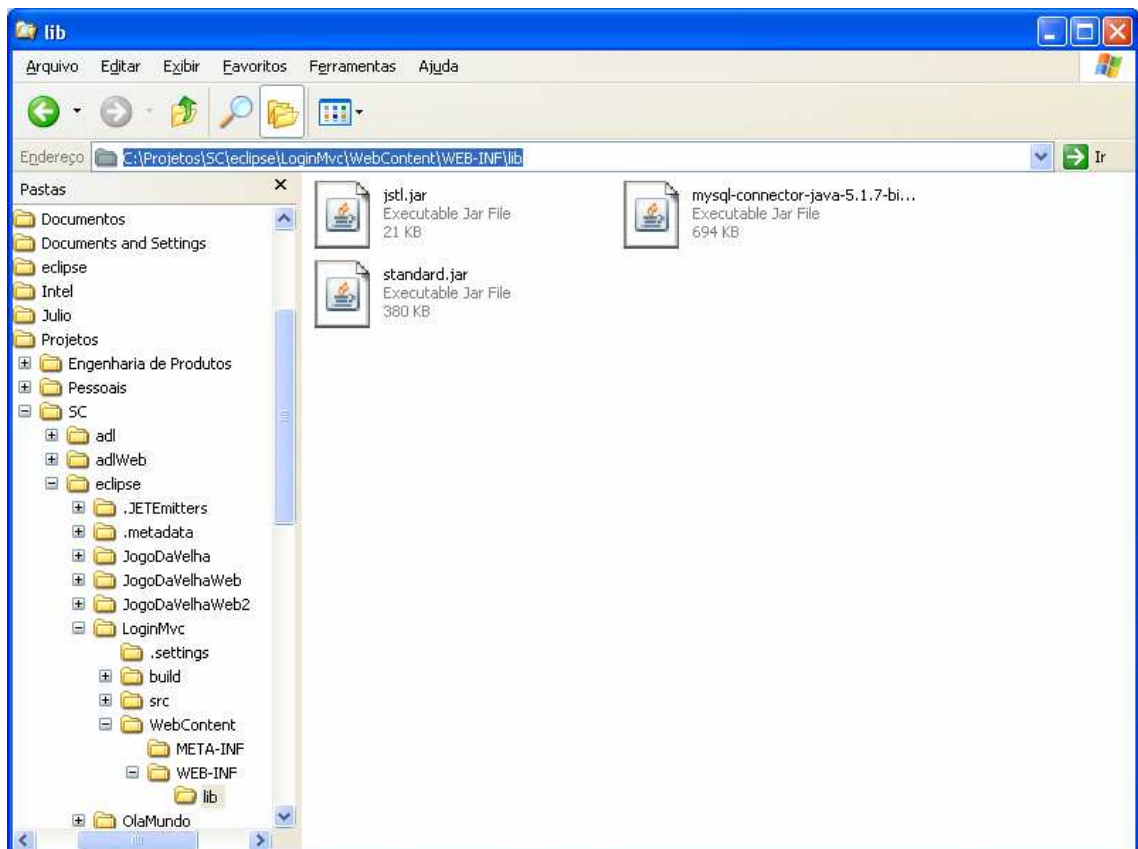
Para que nossa aplicação possa funcionar corretamente, precisamos adicionar duas bibliotecas externas. São elas:

- **MySQLJdbcDriver:** Classe responsável pela comunicação da aplicação com o banco de dados. Deixei-a disponível em minha página pessoal:
<http://juliomartins.pbworks.com/f/mysql-connector-java-5.1.7-bin.jar>
- **JSTL:** Conjunto de classes que representam uma taglib que possibilita a comunicação entre Servlets e JSP de maneira transparente.
<http://juliomartins.pbworks.com/f/jstl.jar>
<http://juliomartins.pbworks.com/f/standard.jar>

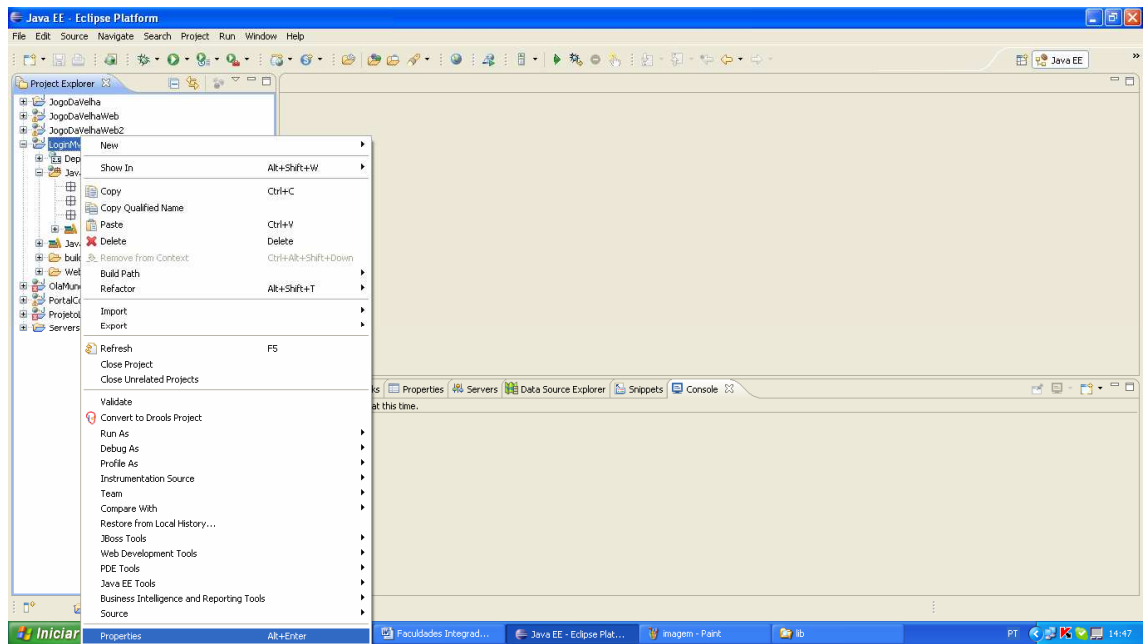
Procedimentos para a importação das bibliotecas

11. Click no link e salve os arquivos na pasta Lib do seu projeto. No meu caso o caminho ficou assim: C:\Projetos\SC\eclipse>LoginMvc\WebContent\WEB-INF\lib

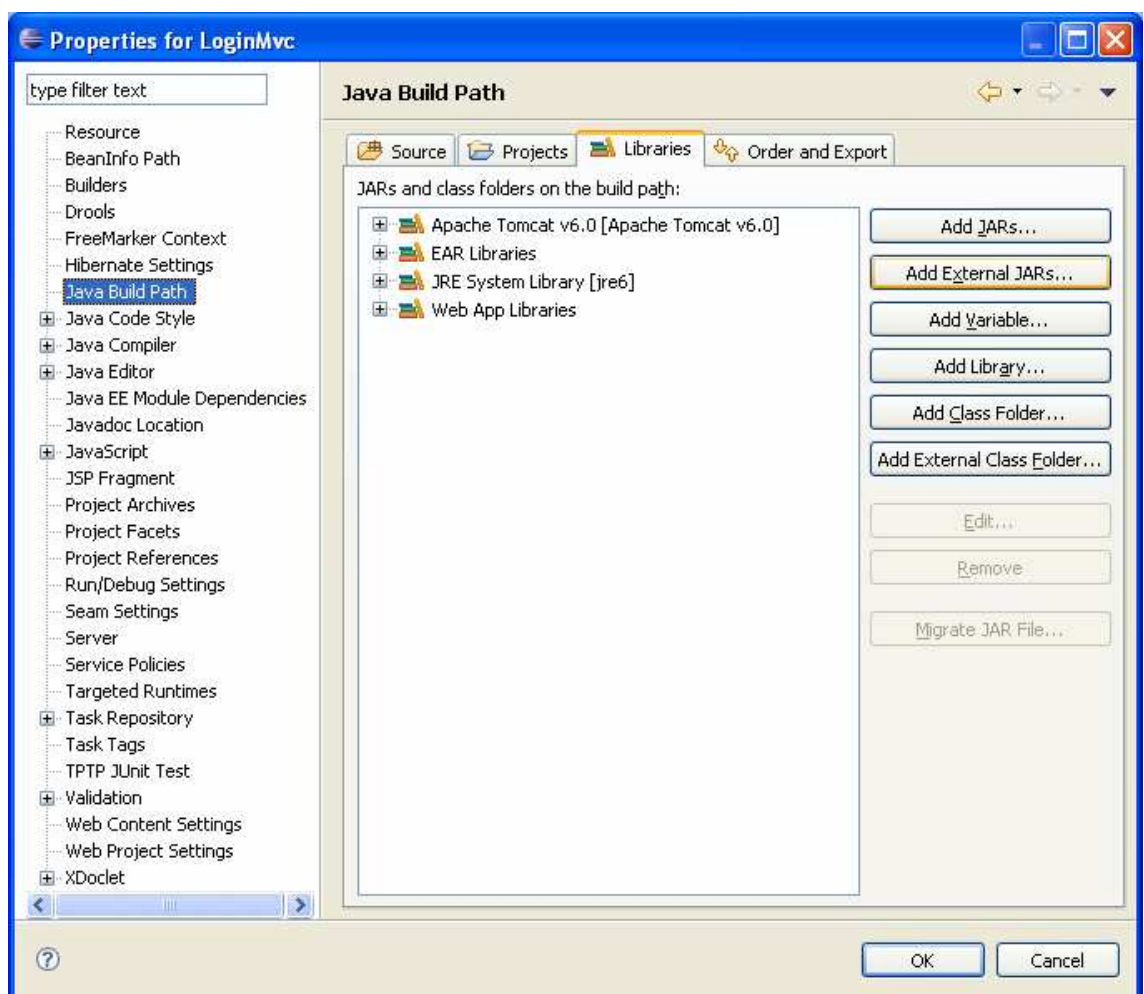
OBS: Preste atenção ao final do caminho. Os arquivos devem ser colocados dentro da pasta WebContent\Web-Inf\lib do seu projeto.



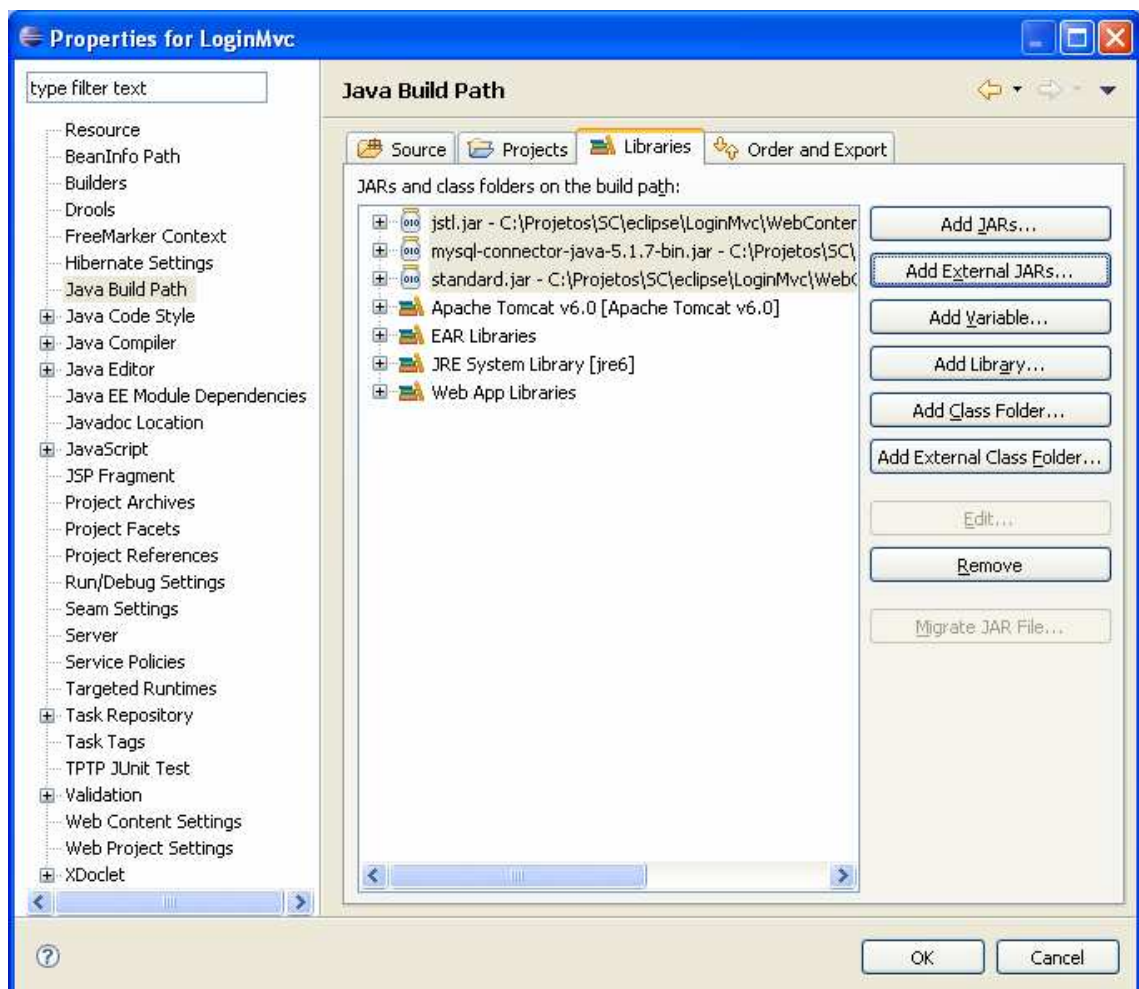
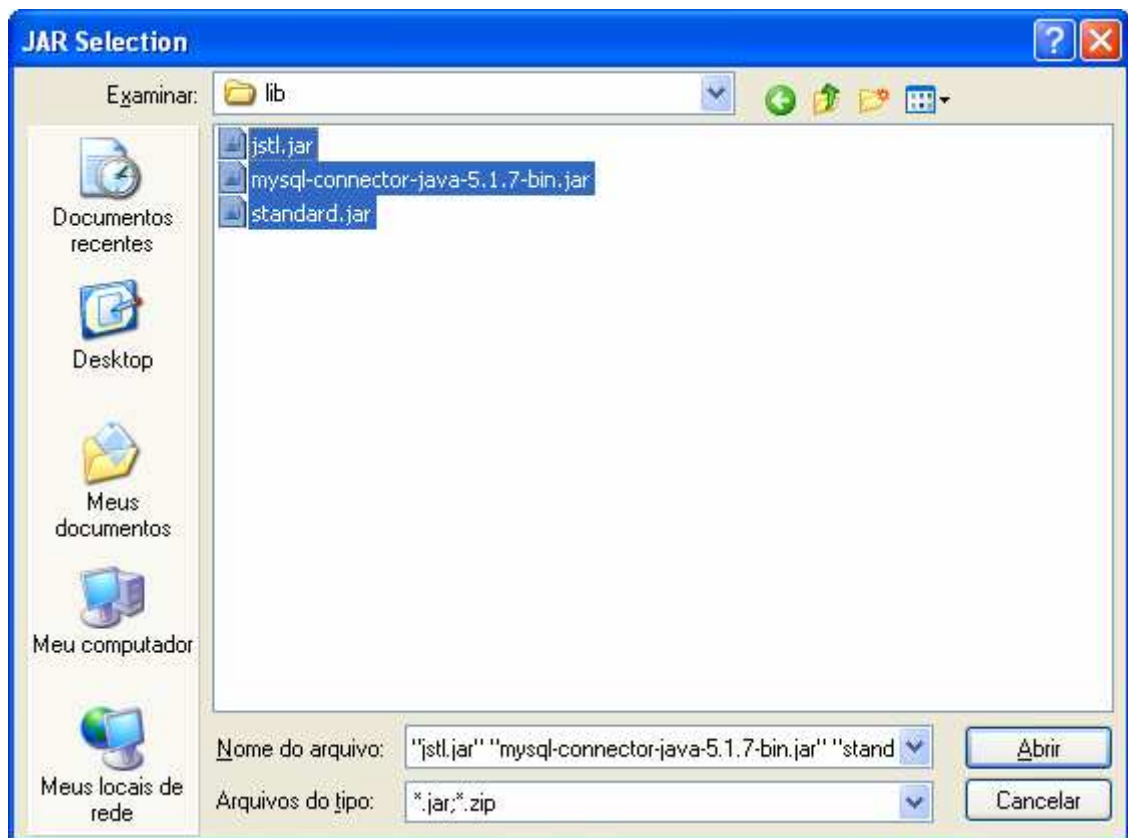
12. Após copiar os arquivos na pasta lib, volte para o Eclipse, clique com o botão direito sobre o nome do seu projeto e vá na opção Properties



13. Click na opção Java Build Path e em seguida na aba Libraries



14. Click no botão Add External Jar e escolha os três arquivos localizados na pasta lib.



15. Click em Ok para finalizar o processo de importação.

Criando o banco de dados

Nossa aplicação de login terá apenas uma tabela chamada Logins. Esta tabela conterá a seguinte estrutura:

Login : varchar(200) not null (PK)

Senha: varchar(100) not null

Esta tabela será criada em um banco chamado LoginBd. Abaixo segue o script para criação do banco e da tabela.

```
CREATE DATABASE IF NOT EXISTS loginbd;
USE loginbd;

--
-- Definition of table `logins`
--

DROP TABLE IF EXISTS `logins`;
CREATE TABLE `logins` (
  `login` varchar(200) NOT NULL,
  `senha` varchar(100) NOT NULL,
  PRIMARY KEY (`login`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

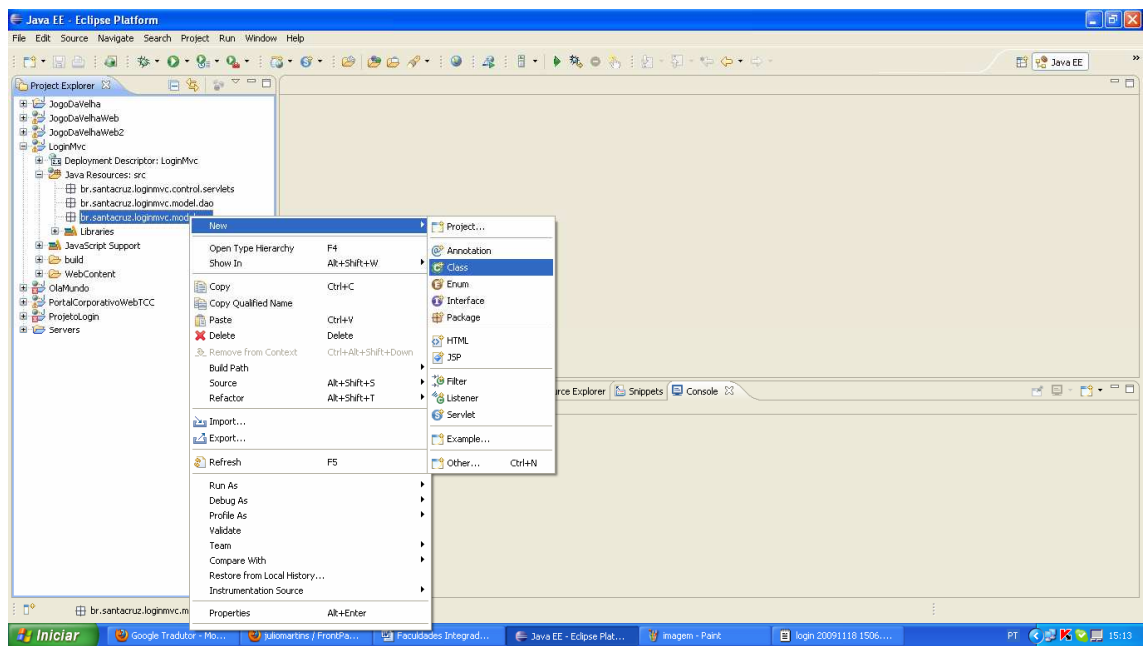
--
-- Dumping data for table `logins`
--

/*!40000 ALTER TABLE `logins` DISABLE KEYS */;
INSERT INTO `logins` (`login`,`senha`) VALUES
('camila@gmail.com','567'),
('elaine@gmail.com','890'),
('martins.julio@gmail.com','1234');
/*!40000 ALTER TABLE `logins` ENABLE KEYS */;
```

Criando o VO

Como já dito anteriormente, o VO normalmente é criado para representar o modelo de dados e facilitar o trafego de informações na aplicação. No nosso caso, criaremos um único VO chamado LoginVo.

16. Click com o botão direito sobre o pacote `br.santacruz.loginmvc.model.vo`. Escolha a opção New -> Class



17. Na tela seguinte digite o nome da classe: LoginVo

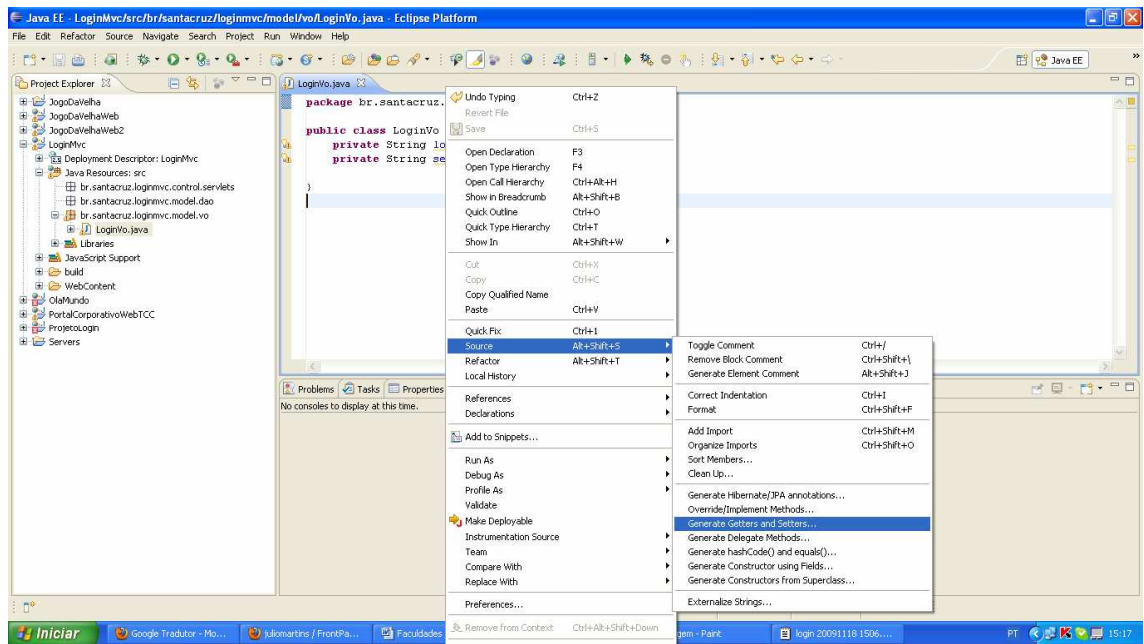


18. No corpo da classe criada, você deve inserir os mesmos atributos existentes em nossa tabela do banco de dados, conforme segue o código abaixo.

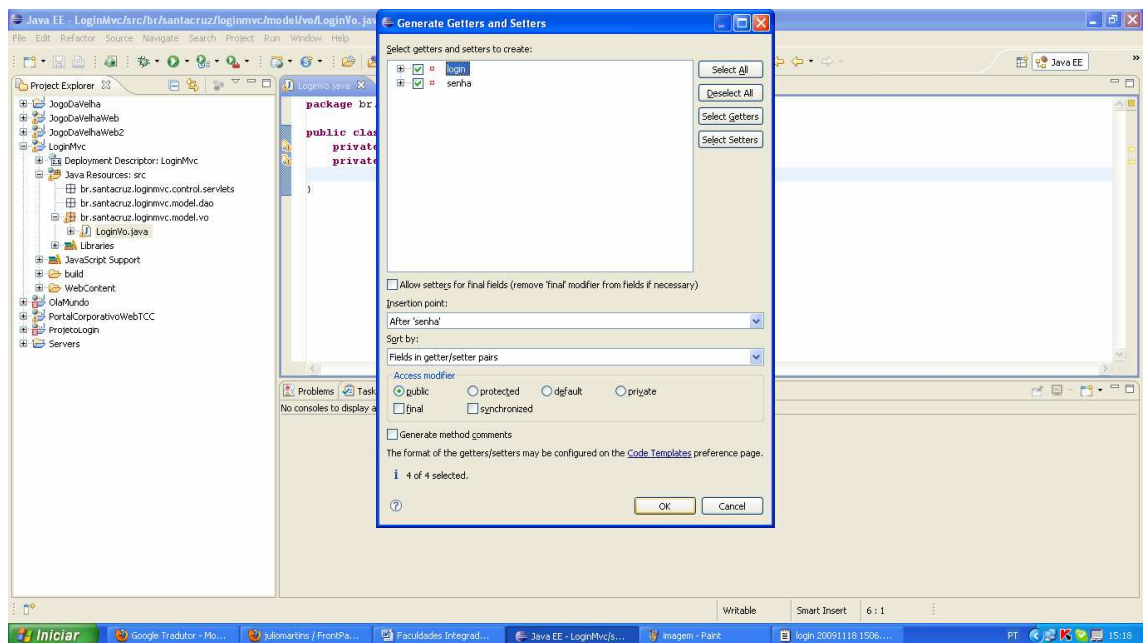
```
package br.santacruz.loginmvc.model.vo;

public class LoginVo {
    private String login;
    private String senha;
}
```

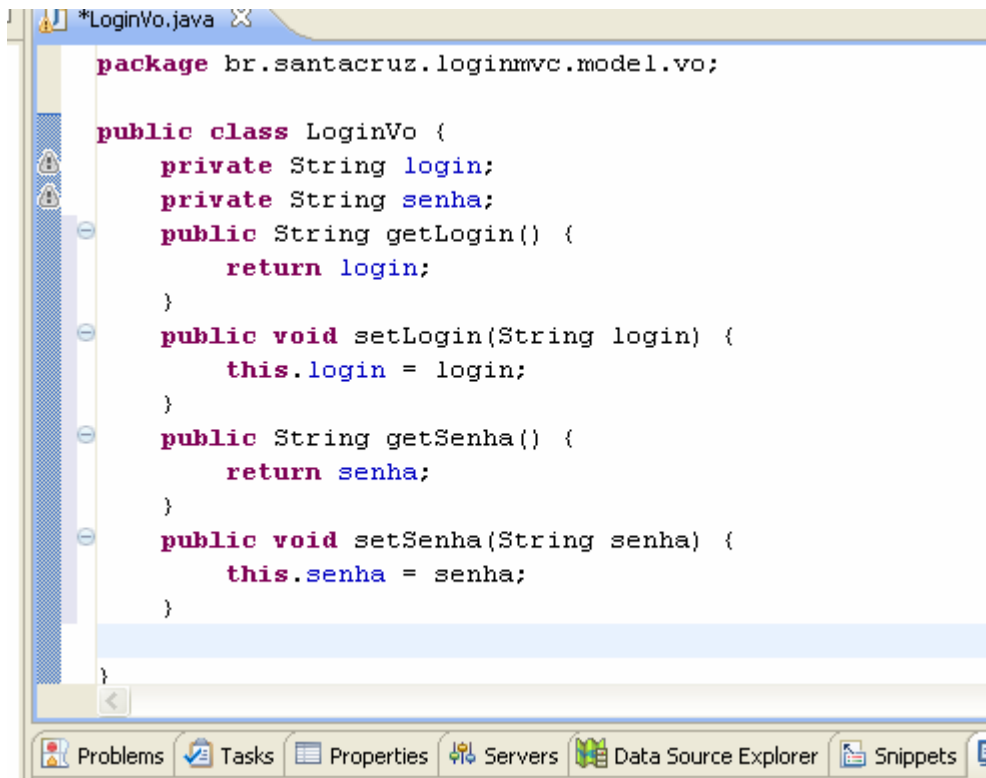
19. Após ter inserido os atributos, click com o botão direito sobre o código e selecione a opção Source -> Generate Getters and Setters



20. Marque os dois atributos e click em OK



21. Nosso VO acaba de ser criado e deve estar parecido com a tela a seguir;



Criando o DAO

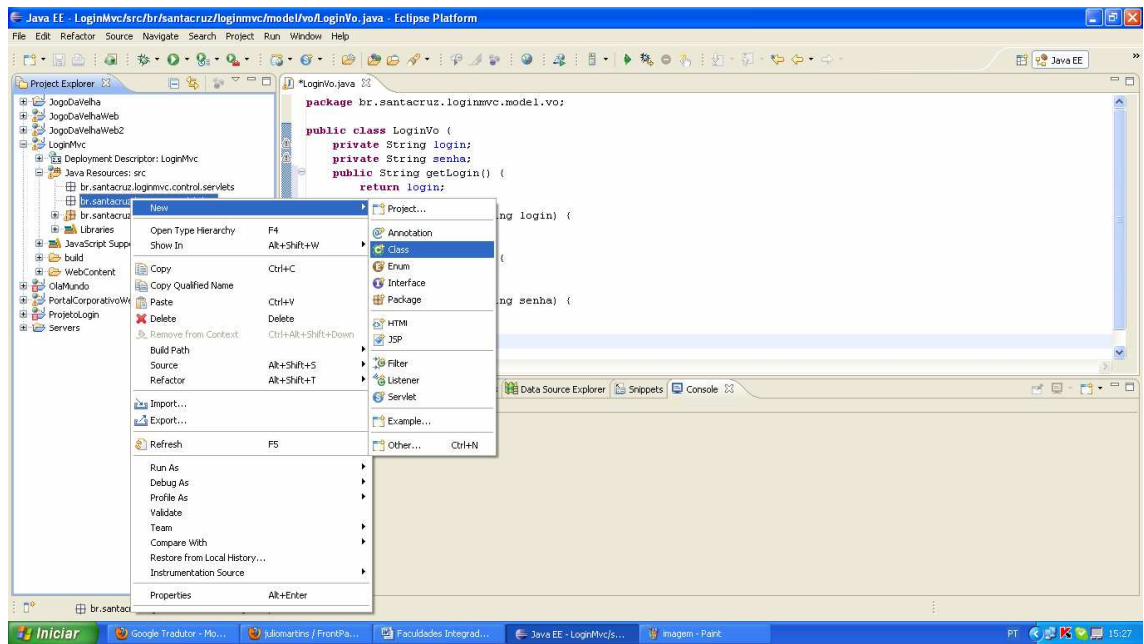
Nossa aplicação terá basicamente duas classes para representar o DAO:

- GenericDao: Classe responsável por conter todos os métodos genéricos de banco tais como getConnection, CloseConnection, além de conter os atributos de ResultSet, Statement e a própria Connection. Além destes atributos, é na GenericDao que encontram-se as informações de usuário, senha, base e host. Esta classe servirá de base para qualquer outra Dao que venha a ser criada.
- LoginDao: Classe criada especificamente para tratar das ações a serem realizadas no banco, necessárias para o funcionamento correto de nossa aplicação. É esta classe que irá conter o método de VerificaLogin que irá buscar no banco informações relevantes a um login específico.

Criando o GenericDao

Basicamente o GenericDao contém os elementos básicos para que a conexão com o banco seja estabelecida.

22. Click com o botão direito sobre o pacote br.santacruz.model.dao e selecione a opção New-> Class. Na tela seguinte digite o nome da Classe : **GenericDao** e confirme



23. Após, insira o código abaixo na classe recém criada.

```
package br.santacruz.loginmvc.model.dao;

/*
 * Classe genérica para gerenciar as conexões com o banco de dados mysql
 */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * @author Administrator
 */
public class GenericDao {
    protected Statement stmt = null;
    protected ResultSet rs = null;
    protected Connection connection = null;
    private String base = "loginbd";
    private String host = "localhost:3306";
    private String driver = "com.mysql.jdbc.Driver";
    private String user = "root";
    private String senha = "root";

    public Connection getConnection(){
        try{
            if( connection == null ){
                createConnection();
            }
            else
            if( connection.isClosed() ){
                createConnection();
            }
        }
        catch(SQLException sEx){
            sEx.printStackTrace();
        }
        return connection;
    }
}
```



```

private void createConnection(){
    try {
        Class.forName(driver).newInstance();
        connection = DriverManager.getConnection("jdbc:mysql:" + host + "/" + base, user,
senha);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

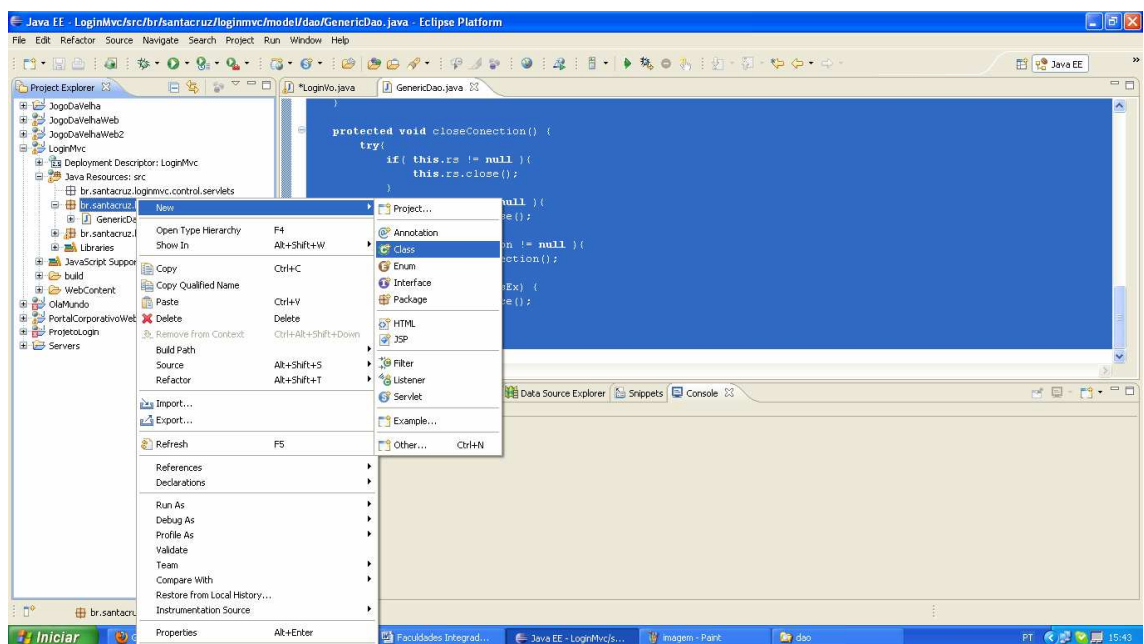
protected void closeConection() {
    try{
        if( this.rs != null ){
            this.rs.close();
        }
        if( this.stmt != null ){
            this.stmt.close();
        }
        if( this.connection != null ){
            this.closeConection();
        }
    } catch (SQLException sEx) {
        sEx.printStackTrace();
    }
}
}
}

```

Criando a Classe de Login Dao

A nossa classe LoginDao possui somente um método que é o verificaLogin. Este método recebe um LoginVo preenchido com login e senha e verifica no banco se existe tais dados. Caso exista aquele login e senha no banco, o método retorna 1. Caso contrário, retorna 0 (zero).

24. Click com o botão direito sobre o pacote br.santacruz.loginmvc.model.dao e escolha a opção New -> Class



25. Na próxima tela, digite o nome da classe: LoginDao

New Java Class

Create a new Java class.

Source folder: LoginMvc/src Browse...

Package: br.santacruz.loginmvc.model.dao Browse...

☐ Enclosing type: Browse...

Name: LoginDao

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

? Finish Cancel

26. Click no botão Finish
27. Produza o código que faça o select no banco e retorne. Lembre de que o LoginDao é uma especialização de GenericDao, observe a colocação do extends após a declaração da classe.

```

package br.santacruz.loginmvc.model.dao;
import br.santacruz.loginmvc.model.vo.LoginVo;

public class LoginDao extends GenericDao{

    public int verificaLogin(LoginVo lv) {

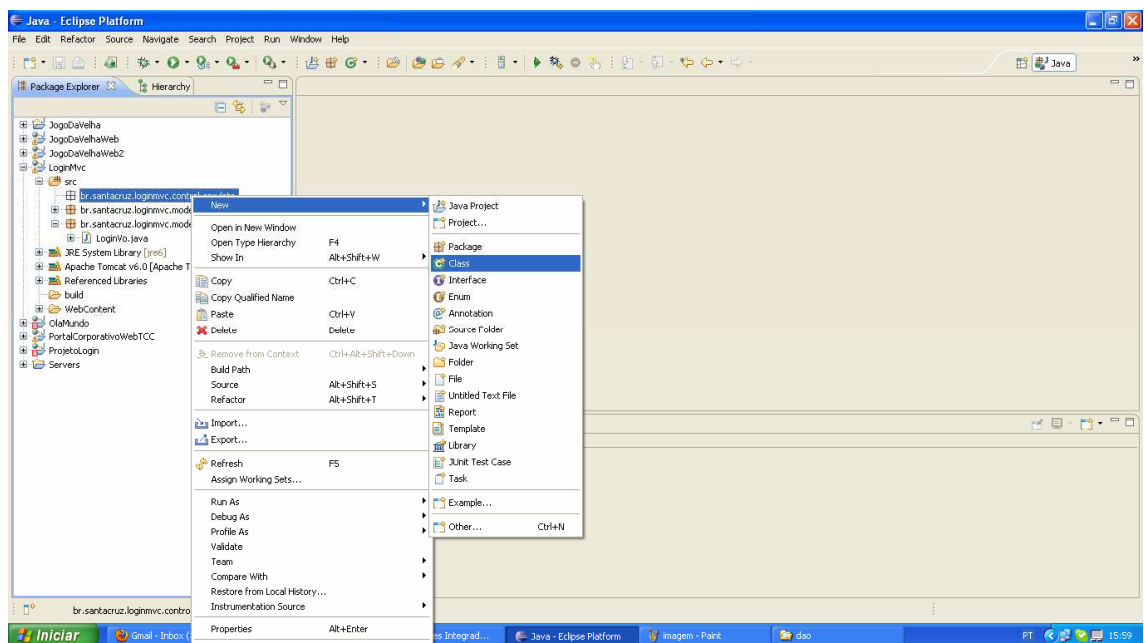
        try {
            this.stmt = this.getConnection().createStatement();
            this.rs = this.stmt.executeQuery("select login,senha from logins where
login='"+lv.getLogin()+"' and senha = '"+lv.getSenha()+"'");
            if (rs.next()) {
                return 1; //Achou o login!
            }
            else
                return 0; //Não achou o login! Login invalido!
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
}

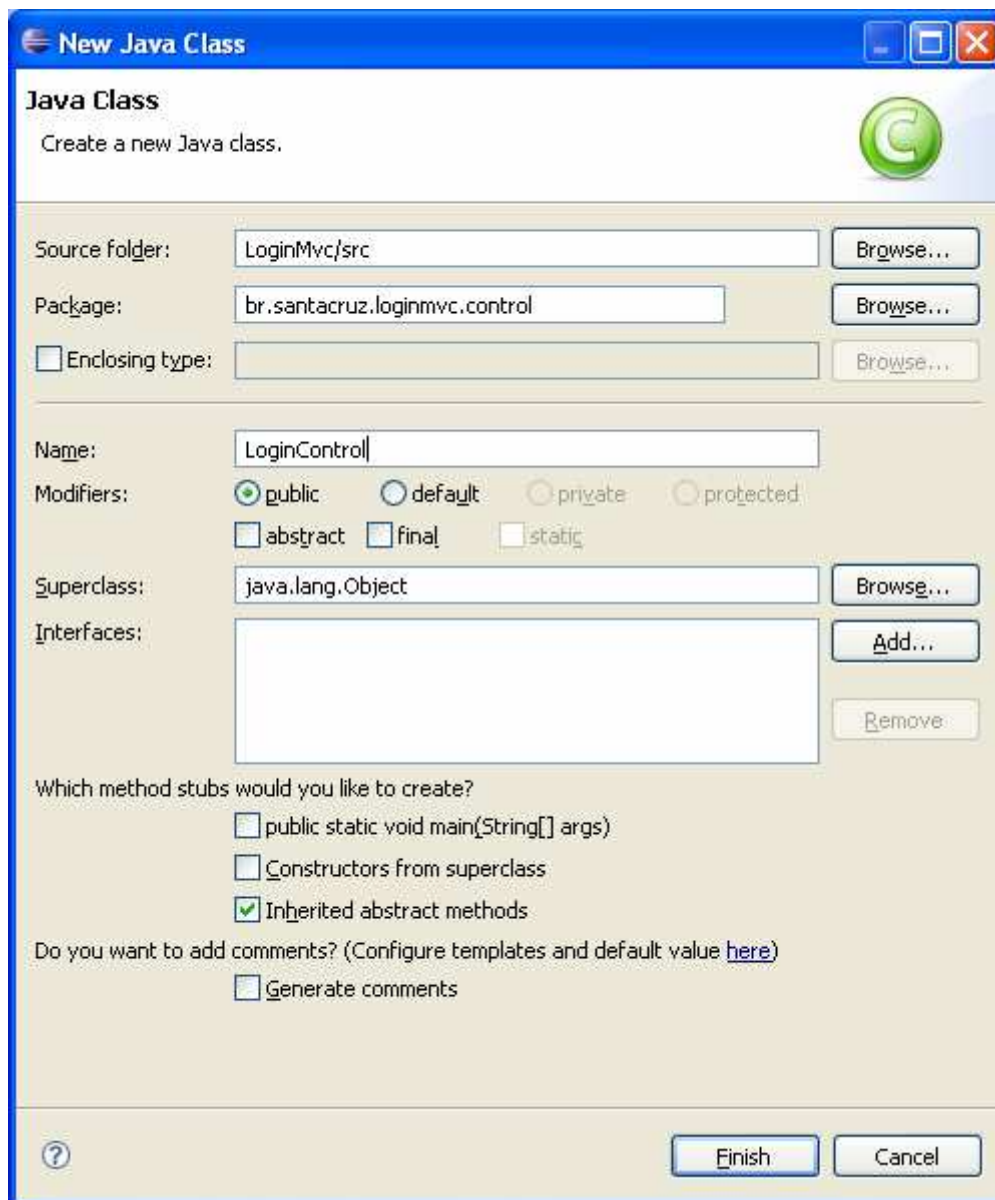
```

Criando o LoginControl

Nossa aplicação tem apenas uma classe de controle. Nesta classe de controle, o único método que temos é o método de validarLogin. Este método irá receber o Login e a Senha digitada e irá verificar com Dao a existência ou não.

28. Click com o botão direito sobre o pacote br.santacruz.loginmvc.control e selecione a opção New -> Class





29. Após criada a classe, devemos fazer um código simples como demonstrado abaixo.

```
package br.santacruz.loginmvc.control;

import br.santacruz.loginmvc.model.dao.LoginDao;
import br.santacruz.loginmvc.model.vo.LoginVo;

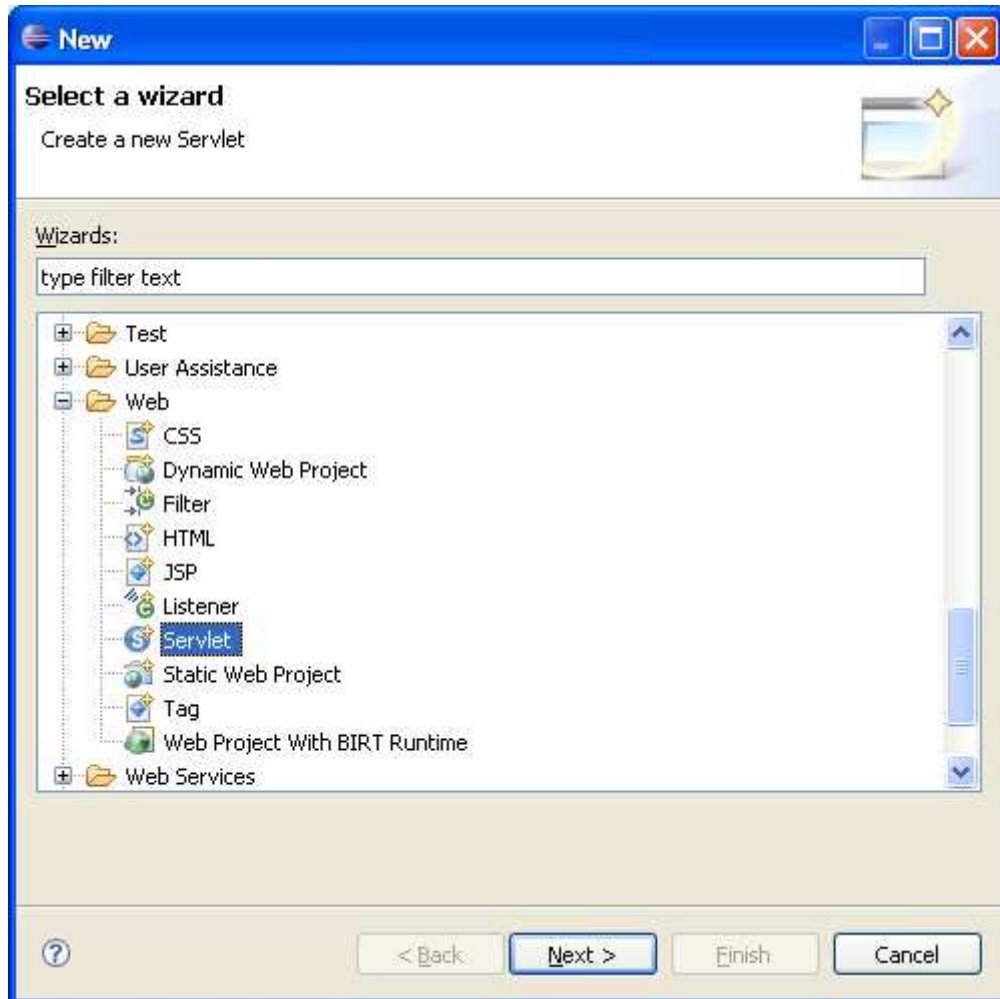
public class LoginControl {

    public int ValidarLogin(String login, String senha) {
        LoginVo lv = new LoginVo();
        lv.setLogin(login);
        lv.setSenha(senha);
        //Instanciando o DAO
        LoginDao ld = new LoginDao();
        int retorno = ld.verificaLogin(lv);
        return retorno;
    }
}
```

Criando o Servlet

O servlet é a classe responsável por pegar os atributos informados pelo usuário e repassar para o controle.

30. Click com o botão direito no pacote `br.santacruz.loginmvc.control.servlets`, escolha a opção `New -> Other`. Vá na pasta `Web` e selecione a opção `Servlet`



31. Digite o nome do servlet e confirme.

Create Servlet

Specify class file destination.

Web project: LoginMvc

Source folder: /LoginMvc/src [Browse...](#)

Java package: br.santacruz.loginmvc.control.servlets [Browse...](#)

Class name: LoginServlet

Superclass: javax.servlet.http.HttpServlet [Browse...](#)

☐ Use an existing Servlet class or JSP

Class name: LoginServlet [Browse...](#)

[?< Back](#) [Next >](#) [Finish](#) [Cancel](#)

32. Na classe do servlet, crie um método denominado `processRequest` e implemente o código abaixo.

```
package br.santacruz.loginmvc.control.servlets;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.santacruz.loginmvc.control.LoginControl;

public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public LoginServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void processRequest(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {

        RequestDispatcher view;
        String login = request.getParameter("txtLogin");
        String senha = request.getParameter("txtSenha");

        LoginControl lc = new LoginControl();
        int retorno = lc.ValidarLogin(login, senha);

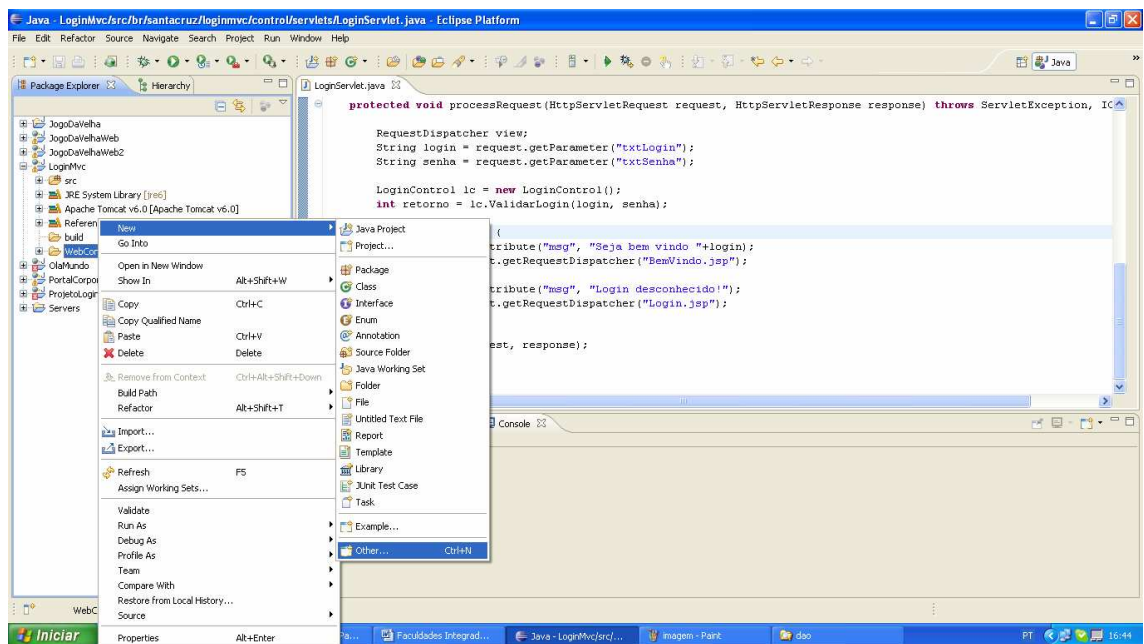
        if (retorno == 1) {
            request.setAttribute("msg", "Seja bem vindo "+login);
            view = request.getRequestDispatcher("BemVindo.jsp");
        } else {
            request.setAttribute("msg", "Login desconhecido!");
            view = request.getRequestDispatcher("Login.jsp");
        }

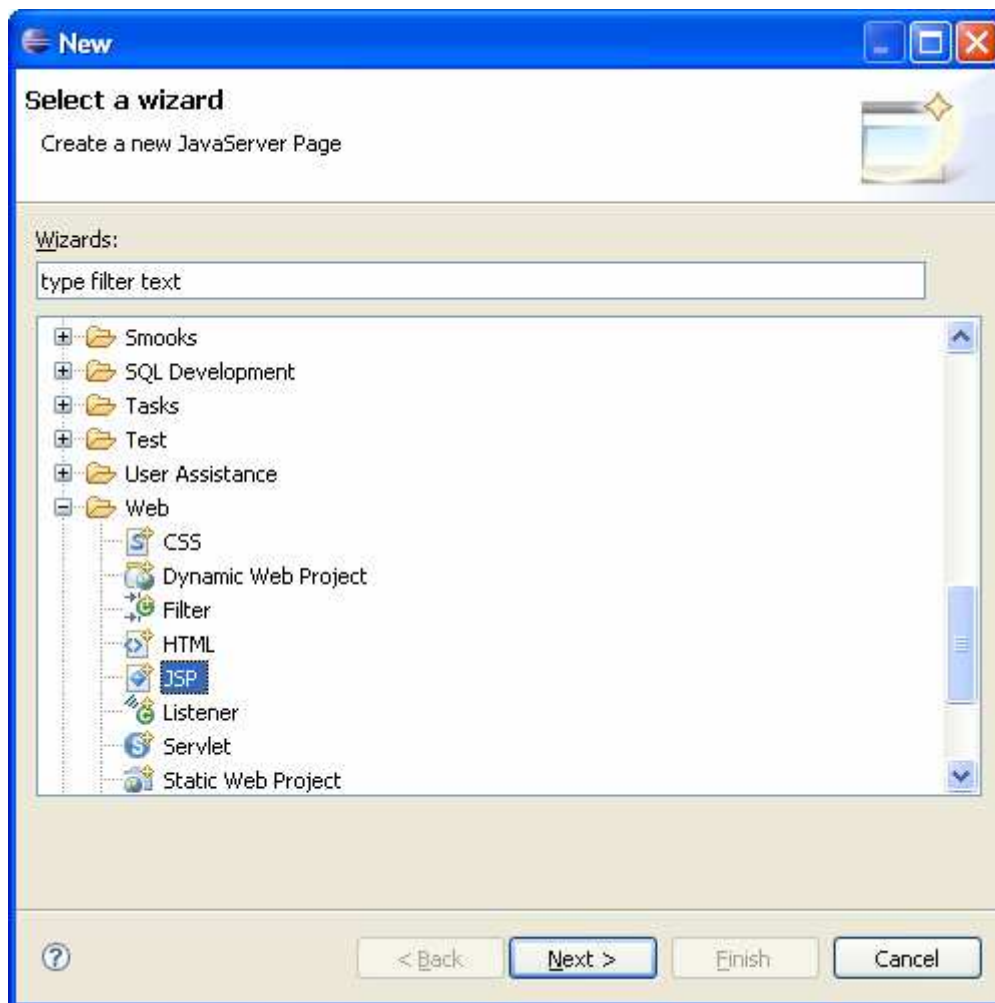
        view.forward(request, response);
    }
}
```

Criando as páginas JSP

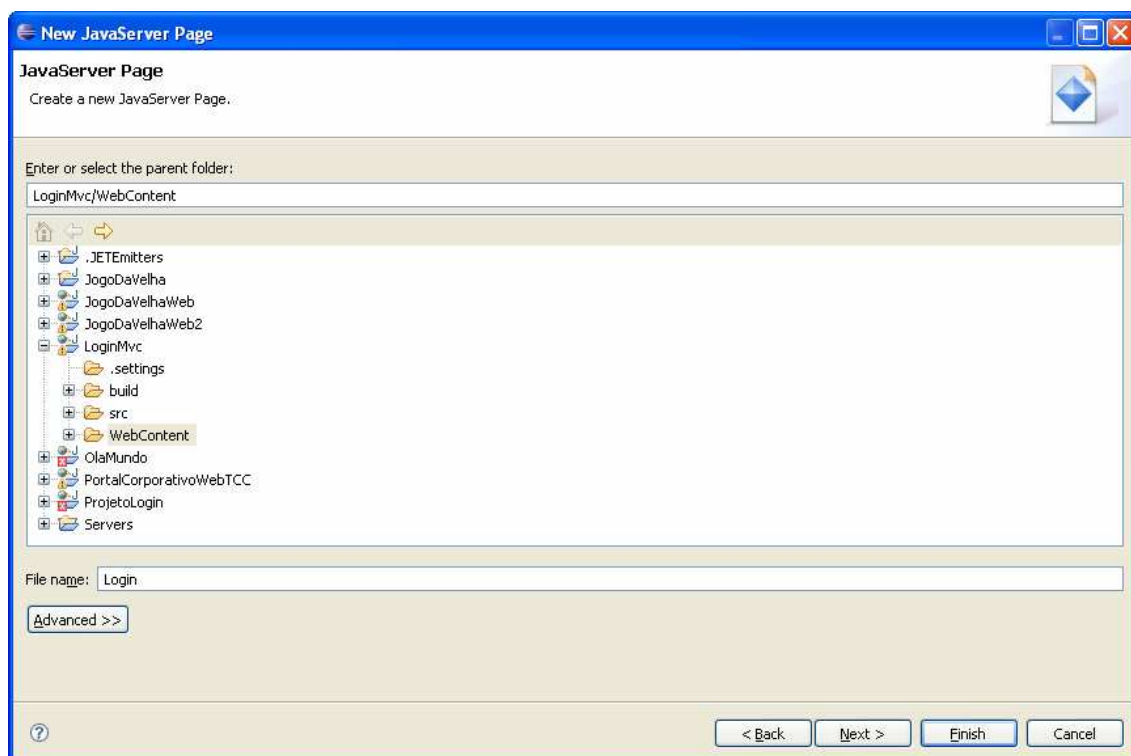
Precisamos criar duas páginas básicas para concluir o nosso sistema. Uma delas é a página de Login, onde o usuário irá inserir os dados para logar. A outra é a página de BemVindo.jsp onde o usuário será cumprimentado pelo seu logon bem sucedido.

33. Click na pasta WebContent do seu projeto, com o botão direito e selecione a opção New -> Other e selecione a opção JSP





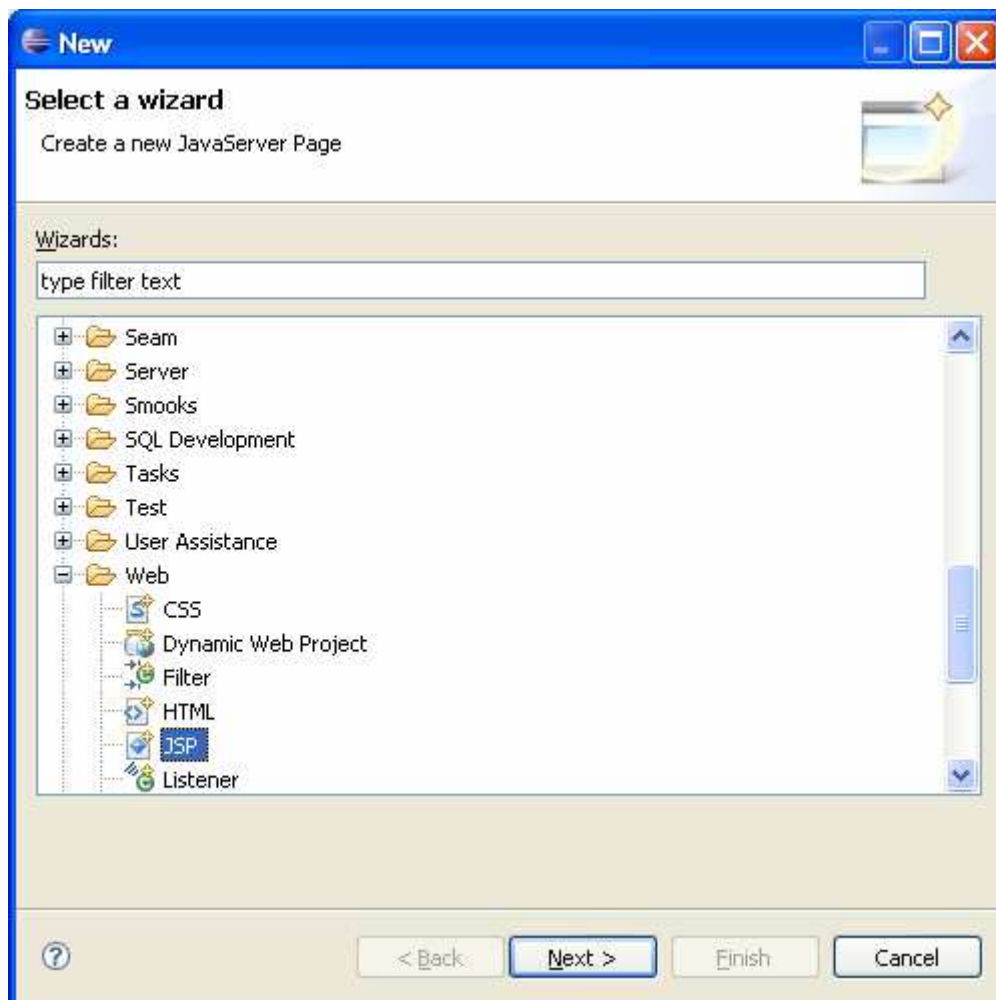
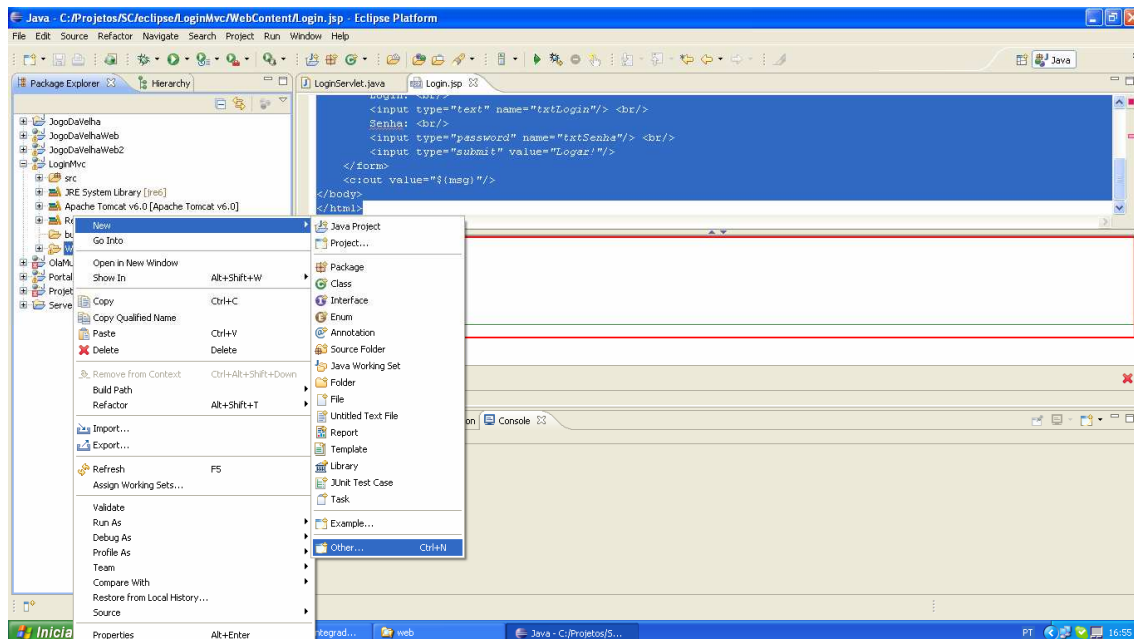
34. Na próxima tela, digite o nome da JSP: Login (sem a extensão) e click e finish.

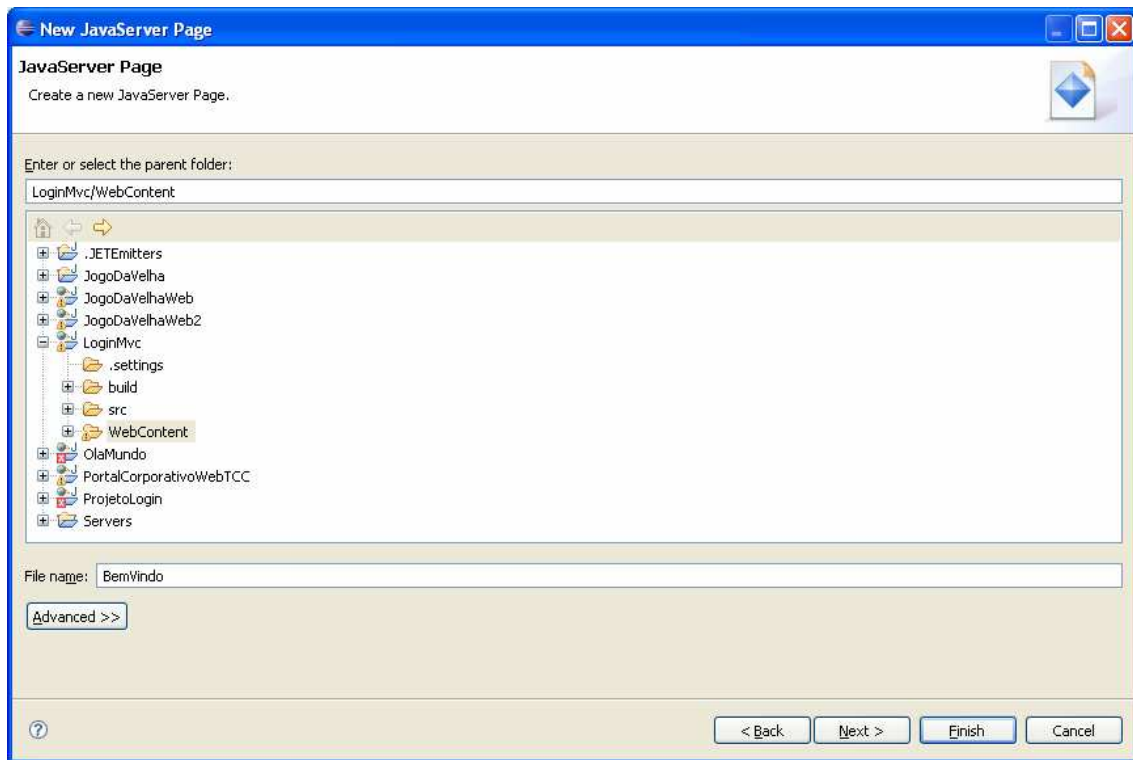


35. Após, construa o código HTML que contenha um formulário com dois inputs e um botão. Semelhante ao código abaixo:

```
<%@ page contentType="text/html; charset=ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login MVC</title>
</head>
<body>
    <form name="formLogin" action="LoginServlet" method="post">
        Login: <br/>
        <input type="text" name="txtLogin" /> <br/>
        Senha: <br/>
        <input type="password" name="txtSenha" /> <br/>
        <input type="submit" value="Logar!" />
    </form>
    <c:out value="${msg}" />
</body>
</html>
```

36. Após, precisamos criar a página BemVindo.jsp. Para tal, repita os últimos passos anteriores somente trocando o nome da JSP a ser criada.





37. Produza um código semelhante ao abaixo:

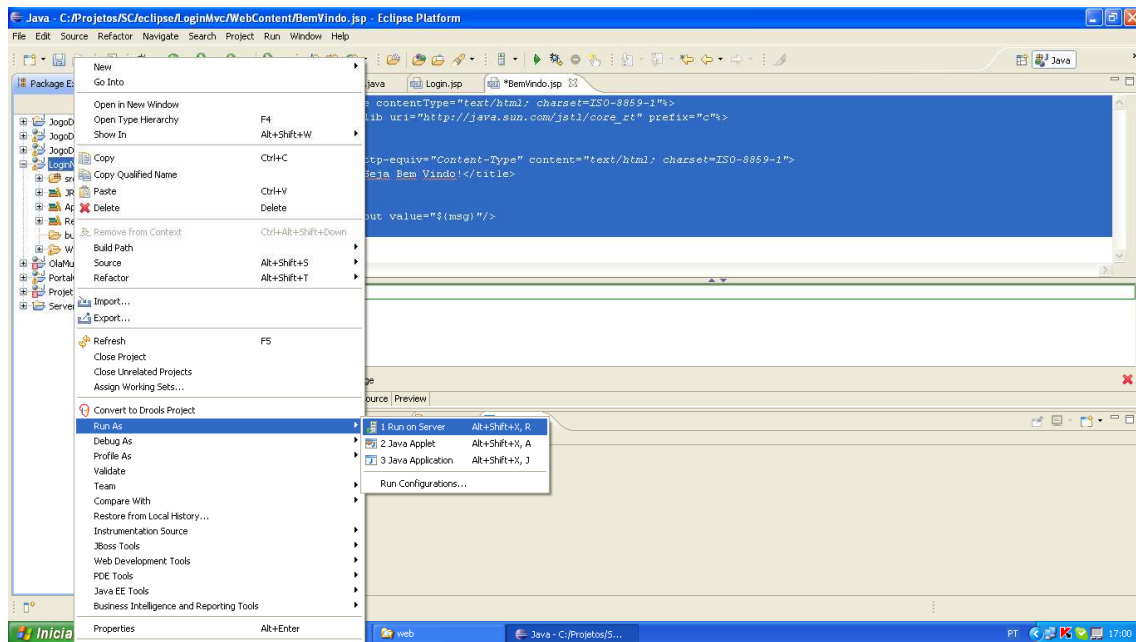
```
<%@ page contentType="text/html; charset=ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Seja Bem Vindo!</title>
</head>
<body>
    <c:out value="$ {msg}" />
</body>
</html>
```

Finalizando

Se você acompanhou até aqui, agora falta pouco! Basta clicar com o botão direito em cima do nome do projeto e escolher a opção Run as -> Run on Server

PRONTO!


Abaixo, segue a sequência de telas produzidas com os testes!



Run On Server

Run On Server

Select which server to use



How do you want to select the server?


☒ Choose an existing server


☐ Manually define a new server

Select the server that you want to use:

type filter text


localhost

 Tomcat v6.0 Server at localhost

 Stopped

Apache Tomcat v6.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 Web modules.

☐ Always use this server when running this project



< Back

Next >

Finish

Cancel

