

CURSO DE DESENVOLVIMENTO DE SOFTWARE WEB COM JAVA SERVER PAGES

Autor

Prof. Me. Jefferson Antonio Ribeiro Passerini

Fernandópolis

2021

LISTA DE FIGURAS

Figura 1 – Verificando a instalação do JDK / JRE	13
Figura 2 – Removendo versões do java no windows	14
Figura 3 – Página de download do Java Development Kit	15
Figura 4 – Instalação da JDK	15
Figura 5 – Acessando configurações avançadas do windows.....	16
Figura 6 – Tela Sobre (Windows)	16
Figura 7 – Tela Propriedades do Sistema e Variáveis de Ambiente.....	17
Figura 8 – Tela Propriedades do Sistema.....	18
Figura 9 – Tela Propriedades do Sistema.....	19
Figura 10 – Processo de Instalação do PostgreSQL.....	20
Figura 11 – Processo de Instalação do PostgreSQL - Continuação	21
Figura 12 – Processo de Instalação do PostgreSQL - Continuação	22
Figura 13 - Página Apache Netbeans.....	23
Figura 14 – Versões para download – Apache Netbeans.....	24
Figura 15 – Download do Apache Netbeans	24
Figura 16 – Tela Inicial do instalador do Netbeans	25
Figura 17 – Passo a Passo da instalação do Netbeans	26
Figura 18 – Configuração do Apache Netbeans	27
Figura 19 – Configuração do Java.....	28
Figura 20 – Configuração do Java.....	28
Figura 21 – Configuração do Servidor Web Payara no Netbeans	29
Figura 22 – Estrutura de uma aplicação monolítica.....	31
Figura 23 – Estrutura de uma aplicação web.	34
Figura 24 – Estrutura de Camadas da aplicação desenvolvida no curso.	36
Figura 25 – Criando Projeto Java JSP no Netbeans	37
Figura 26 – Estrutura do Projeto Web (Netbeans).....	38
Figura 27 – Inserindo bibliotecas no projeto	40
Figura 28 – Adicionando pacotes no projeto	41
Figura 29 – Adicionando pacote da camada controller.....	42
Figura 30 – Pacotes criados no projeto	42
Figura 31 – Criando a index.jsp.....	43
Figura 32 – Executando o projeto pela primeira vez	44

Figura 33 – Código para index.jsp.....	44
Figura 34 – Criando header.jsp	45
Figura 35 – Código header.jsp.....	45
Figura 36 – Criando pasta JS para JQuery	46
Figura 37 – Criando menu.jsp	47
Figura 38 – Código menu.jsp.....	47
Figura 39 – Criando footer.jsp	48
Figura 40 – Código footer.jsp.....	48
Figura 41 – Criando home.jsp	48
Figura 42 – Código home.jsp.....	49
Figura 43 – Estrutura Básica da Interface (View)	49
Figura 44 – Estrutura do Projeto do curso.....	50
Figura 45 – Tela do Sistema	50
Figura 46 – Criando o banco de dados – “bdaplcurs”.....	51
Figura 47 – Criando a tabela estado.	52
Figura 48 – Incluindo registro na tabela estado.....	52
Figura 49 – Criando arquivo banco.sql.....	53
Figura 50 – Criando classe de conexão da aplicação	54
Figura 51 – Programando a classe de conexão	55
Figura 52 – Criando a classe Filter.....	56
Figura 53 – Criando a classe Filter.....	57
Figura 54 – Classe FilterAutenticacao.....	57
Figura 55 – Implementando a Classe Filter (1).....	58
Figura 56 – Implementando a Classe Filter – método init (2)	59
Figura 57 – Implementando a Classe Filter – método doFilter (2).....	59
Figura 58 – Implementando a Classe Filter – método doFilter (2).....	60
Figura 59 – Criando a Classe Estado na camada Model.	62
Figura 60 – Criando a Construtor da Classe Estado na model.....	63
Figura 61 – Criando os métodos Getter e Setter	64
Figura 62 – Código Completo da Model de Estado (Parte 1)	64
Figura 63 – Código Completo da Model de Estado (Parte 2)	65
Figura 64 – Implementando a GenericDAO	66
Figura 65 – Criando a Classe EstadoDAO	67
Figura 66 – Classe EstadoDAO com os métodos abstratos implementados	68

Figura 67 – Implementando o método Construtor da EstadoDAO	68
Figura 68 – Método Listar()	69
Figura 69 – Implementando o método Listar	69
Figura 70 – Fazendo importações de recursos do java	69
Figura 71 – Criando Classe EstadoListar na camada controller.....	71
Figura 72 – Java APIs - HTTPServlet.....	71
Figura 73 – Classe EstadoListar.....	71
Figura 74 – Classe EstadoListar – doGet e doPost.....	72
Figura 75 – Classe EstadoListar – método processRequest.....	72
Figura 76 – Implementando a classe EstadoListar	73
Figura 77 – Criando a estrutura de pastas no Front-End da aplicação	74
Figura 78 – Código fonte da JSP – Estado.jsp (Listar) – Parte 1	75
Figura 79 – Código fonte da JSP – Estado.jsp (Listar) – Parte 1	76
Figura 80 – Executando o Estado.jsp.....	76
Figura 81 – EstadoDAO – Método Cadastrar.....	77
Figura 82 – EstadoDAO – Método Incluir	78
Figura 83 – Controller – Criando o Servlet - EstadoNovo.....	79
Figura 84 – Controller – Criando o Servlet - EstadoCadastrar	80
Figura 85 – View – Criando estadoCadastrar.jsp	81
Figura 86 – View – Codificando estadoCadastrar.jsp	82
Figura 87 – View – Resultado da inclusão.....	83
Figura 88 – DAO – Programando o método carregar()	84
Figura 89 – DAO – Programando o método alterar().....	84
Figura 90 – Controller – Programando o servlet EstadoCarregar()	85
Figura 91 – DAO – Programando o método excluir()	86
Figura 92 – Controller – Programando o servlet EstadoExcluir().....	87
Figura 93 – Criando a Tabela de Cidade	89
Figura 94 – Criando a Tabela de Cidade	89
Figura 95 – Criando a Tabela de Cidade	90
Figura 96 – Criando os atributos na classe Cidade	91
Figura 97 – Criando os construtores da classe Cidade	91
Figura 98 – Métodos Gets e Sets	92
Figura 99 – Criando a classe CidadeDAO.....	93
Figura 100 – Implementando os métodos abstratos da GenericDAO	93

Figura 101 – Métodos Abstratos implementados.....	94
Figura 102 – Método Construtor e atributo para conexão	94
Figura 103 – Método Cadastrar (CidadeDAO)	95
Figura 104 – Método Inserir (CidadeDAO)	95
Figura 105 – Método Alterar (CidadeDAO).....	96
Figura 106 – Método Excluir (CidadeDAO)	97
Figura 107 – Método Carregar (CidadeDAO).....	98
Figura 108 – Método Listar (CidadeDAO)	99
Figura 109 – Método Listar (CidadeDAO)	100
Figura 110 – Criando Servlet CidadeNovo	100
Figura 111 – Implementando o método processRequest em CidadeNovo	101
Figura 112 – Implementando o método processRequest em CidadeCarregar	102
Figura 113 – Implementando o método processRequest em CidadeListar	102
Figura 114 – Implementando o método processRequest em CidadeCadastrar	103
Figura 115 – Implementando o método processRequest em CidadeExcluir	104
Figura 116 – Criando a pasta de cidade na camada view	105
Figura 117 – Criando “cidade.jsp”.....	105
Figura 118 – Criando “cidadeCadastrar.jsp”	106
Figura 119 – Codificando a Página cidade.jsp (parte 1)	106
Figura 120 – Codificando a Página cidade.jsp (continuação).....	107
Figura 121 – Codificando a Página cidadeCadastrar.jsp	108
Figura 122 – Alterando o menu.jsp.....	108
Figura 123 – Telas de Cadastro de Cidades.....	109
Figura 124 – Diagrama de Classe	110
Figura 125 – Código SQL Tab. Despesa	111
Figura 126 – Criando a classe “Conversao”	112
Figura 127 – Criando os métodos de manipulação de datas	113
Figura 128 – Importações da Classe Conversao.....	114
Figura 129 – Implementando os métodos de manipulação de moeda.	114
Figura 130 – Camada Model – Criando a classe Despesa.	115
Figura 131 – Camada Model – Criando a classe Despesa.	115
Figura 132 – Camada Model – Gerando os métodos construtores.	116
Figura 133 – Camada Model – Alterando o método construtor.....	117
Figura 134 – Camada Model – Importando classe Conversão.....	117

Figura 135 – Camada Model – Classe Despesa (Construtores)	117
Figura 136 – Camada Model – Classe Despesa	118
Figura 137 – Camada DAO – Classe DespesaDAO	119
Figura 138 – Camada DAO – Classe DespesaDAO – Implementando GenericDAO	119
Figura 139 – Camada DAO – Classe DespesaDAO – Atributo “conexão”.	120
Figura 140 – Camada DAO – Classe DespesaDAO – Método Construtor	120
Figura 141 – Camada DAO – Classe DespesaDAO – Método Cadastrar.	121
Figura 142 – Camada DAO – Classe DespesaDAO – Método Inserir.	121
Figura 143 – Camada DAO – Classe DespesaDAO – Método Alterar.	122
Figura 144 – Camada DAO – Classe DespesaDAO – Método Excluir.	123
Figura 145 – Camada DAO – Classe DespesaDAO – Método Carregar.....	123
Figura 146 – Camada DAO – Classe DespesaDAO – Método listar.	124
Figura 147 – Adicionando biblioteca GSON.	125
Figura 148 – Exemplos de Estrutura JSON.....	126
Figura 149 – Camada DAO – Classe DespesaDAO – Método listarJSON().	127
Figura 150 – Método listarJSON() - Implementação.	127
Figura 151 – Criando pacote Controller para cadastro de despesas.....	128
Figura 152 – Criando Servlet – DespesaListar.	128
Figura 153 – Criando Servlet – DespesaListarJSON.	129
Figura 154 – View – Criando pasta para JSPs de despesa.....	129
Figura 155 – View – Criando Despesa.jsp.....	130
Figura 156 – Controller – Servlet DespesaListar	130
Figura 157 – View – menu.jsp	131
Figura 158 – View – despesa.jsp.....	131
Figura 159 – View – despesa.jsp – continuação	132
Figura 160 – View – Resultado.....	133
Figura 161 – Servlet DespesaNovo – criação.	134
Figura 162 – Servlet DespesaNovo – código.	134
Figura 163 – JSP cadastrar despesa – Criação.	135
Figura 164 – JSP Cadastrar Despesa – Codificação HTML.....	135
Figura 165 – JSP Cadastrar Despesa – Codificação HTML - Continuação.....	136
Figura 166 – JSP Cadastrar Despesa - Tela	137
Figura 167 – JSP Cadastrar Despesa – Aplicando CSS.	137

Figura 168 – JSP Cadastrar – Upload Arquivo.....	138
Figura 169 – JSP Cadastrar – Upload Arquivo – Função uploadFile().....	138
Figura 170 – JSP Cadastrar – Configuração campos de moeda.....	139
Figura 171 – JSP Cadastrar – Botão Salvar Documento.....	140
Figura 172 – JSP Cadastrar – Função validarCampos().....	140
Figura 173 – JSP Cadastrar – função gravarDados().....	141
Figura 174 – Controller – Servlet: DespesaCadastrar.....	142
Figura 175 – View Cadastrar Despesa – Resultado Final.....	142
Figura 176 – Controller – Servlet DespesaCarregar.....	143
Figura 177 – Controller – Servlet DespesaExcluir.....	144
Figura 178 – Diagrama de Classe	145
Figura 179 – Código SQL – Criando a tabela pessoa.....	145
Figura 180 – Código SQL – Criando a tabela pessoa.....	146
Figura 181 – Camada Model – Criando a classe Pessoa.....	147
Figura 182 – Camada Model – Criando a classe Pessoa.....	147
Figura 183 – Camada Model – Gerando os métodos construtores	148
Figura 184 – Camada Model – Gerando os métodos Get e Set.....	148
Figura 185 – Camada Model – Criando a classe Administrador.....	149
Figura 186 – Camada Model – Criando os atributos da classe Administrador	149
Figura 187 – Camada Model – Estabelecendo a relação de herança	149
Figura 188 – Camada Model – Criando construtor com parâmetros	150
Figura 189 – Camada Model – Criando método para gerar objetos vazios.....	150
Figura 190 – Camada Model – Administrador (código final).....	151
Figura 191 – Camada DAO – Classe PessoaDAO.....	152
Figura 192 – Camada DAO – Classe “PessoaDAO” – Atributo “conexão”	152
Figura 193 – Camada DAO – Classe “PessoaDAO” – Construtor.....	153
Figura 194 – Camada DAO – Classe “PessoaDAO” – método cadastrar()	153
Figura 195 – Camada DAO – Classe “PessoaDAO” – inserir()	154
Figura 196 – Camada DAO – Classe “PessoaDAO” – alterar()	154
Figura 197 – Camada DAO – Classe “PessoaDAO” – carregar().....	155
Figura 198 – Camada DAO – Classe “PessoaDAO” – carregarCpf().....	155
Figura 199 – Camada DAO – Classe AdministradorDAO.....	156
Figura 200 – Camada DAO–Classe AdministradorDAO–Implementando GenericDAO	156

Figura 201 – Camada DAO – Classe AdministradorDAO – GenericDAO	157
Figura 202 – Camada DAO – Classe AdministradorDAO– cadastrar()	157
Figura 203 – Camada DAO – Classe AdministradorDAO– verificarCpf()	158
Figura 204 – Camada DAO – Classe AdministradorDAO– inserir().....	158
Figura 205 – Camada DAO – Classe AdministradorDAO– alterar()	159
Figura 206 – Camada DAO – Classe AdministradorDAO– excluir()	159
Figura 207 – Camada DAO – Classe AdministradorDAO– carregar()	160
Figura 208 – Camada DAO – Classe AdministradorDAO– listar().....	161
Figura 209 – Criando o pacote para controller de Pessoaa	162
Figura 210 – Criando o Servlet PessoaBuscarCpfCnpj.....	162
Figura 211 – Programando o Servlet PessoaBuscarCpfCnpj	163
Figura 212 – Criando o pacote para controller de Administrador.....	164
Figura 213 – Criando o Servlet AdministradorListar	164
Figura 214 – Programando o Servlet AdministradorListar	165
Figura 215 – Controller – Criando o Servlet AdministradorCadastrar	165
Figura 216 – Controller – Codificando o Servlet AdministradorCadastrar	166
Figura 217 – Controller – Criando o Servlet AdministradorCarregar	167
Figura 218 – Controller – Codificando o Servlet AdministradorCarregar	167
Figura 219 – Controller – Criando o Servlet AdministradorExcluir.....	168
Figura 220 – Controller – Codificando o Servlet AdministradorExcluir	168
Figura 221 – Criando método listar(int estado) em CidadeDAO.....	169
Figura 222 – Criando Servlet – CidadeBuscarPorEstado.....	170
Figura 223 – Programando Servlet “CidadeBuscarPorEstado”	170
Figura 224 – Criando pasta para interface de Administrador.....	171
Figura 225 – Importando “app.js” no arquivo “header.jsp”	171
Figura 226 – Criando View – administrador.jsp	172
Figura 227 – View – Implementando o administrador.jsp	172
Figura 228 – Criando a view Usuario no banco de dados	180
Figura 229 – Criando a classe model de Usuario	181
Figura 230 – Criando a classe model de UsuarioDAO	182
Figura 231 – Criando a Servlet UsuarioBuscarPorLogin	185
Figura 232 – Criando a Servlet UsuarioLogar	186
Figura 233 – Criando a Servlet UsuarioDeslogar	187
Figura 234 – Interface Sistema (JSPs).....	187

Figura 235 – Revisando o Index.jsp	188
Figura 236 – Revisando o menu.jsp	188
Figura 237 – Revisando o home.jsp	189
Figura 238 – Revisando o login.jsp	189
Figura 239 – Revisando o menuCliente.jsp	191
Figura 240 – Revisando o menuFornecedor.jsp	191
Figura 241 – Revisando o menuAdministrador.jsp	191
Figura 242 – Revisando o menuLogado.jsp	192
Figura 243 – Revisando o homeLogado.jsp	192
Figura 244 – Revisando a chamada de menu em nossas páginas	193
Figura 245 – Revisando a chamada de menu	193
Figura 246 – Método verificaUsuario – Classe Usuario (Model).....	195
Figura 247 – Alterações método doFilter na classe de filtro do projeto.	196

LISTA DE QUADROS

Quadro 1 - Variáveis de Ambiente (JAVA)	17
Quadro 2 – Camadas da Aplicação desenvolvida durante o curso.	35

SUMÁRIO

Introdução	13
Capítulo 1 – Instalação do Java e configuração de ambiente	13
Capítulo 2 – Instalação e configuração do postgresql.....	19
Capítulo 3 – Instalação e configuração do NETBEANS	23
Capítulo 4 – Criação do projeto em java web.....	31
4.1 Arquitetura MVC.....	32
4.2 Arquitetura do projeto desenvolvido no curso	34
4.3 Criando o projeto do curso no netbeans	36
4.4 estruturando a interface do projeto	44
4.5 CRIANDO O BANCO DE DADOS	51
4.6 CRIANDO A conexão ao banco de dados.....	53
Capítulo 5 – Implementando um Crud Simples (Estado)	61
5.1 Criando o Recurso de Listar Estado	61
5.2 Criando o Recurso de Incluir Estado.....	77
5.3 Criando o Recurso de Alterar Estado.....	83
5.4 Criando o Recurso de Excluir Estado	86
Capítulo 7 – Desenvolvimento: crud 1-N.....	89
7.1 Criando a camada dao (CidadeDAO)	93
7.2 Criando a camada Controller	99
7.2.1 Controller Cidade: Servlet CidadeNovo	100
7.2.2 Controller Cidade: Servlet CidadeCarregar	101
7.2.3 Controller Cidade: Servlet CidadeListar	102
7.2.4 Controller Cidade: Servlet CidadeCadastrar.....	103
7.2.5 Controller Cidade: Servlet CidadeExcluir	104
7.3 Criando a camada View	104

7.3.1 Camada View: cidade.jsp	106
7.3.2 Camada View: cidadeCadastrar.jsp.....	107
7.3.3 Camada View: menu.jsp.....	108
7.3.4 Telas Desenvolvidas.....	109
Capítulo 8 – Desenvolvimento de crud com ajax	110
8.1 Funções Auxiliares de Manipulação de Dados.....	112
8.2 Desenvolvendo a Camada Model	114
8.3 Desenvolvendo a Camada DAO	119
8.3.1 Trabalhando com JSON	124
8.3.2 Criando o método ListarJSON()	126
8.4 Desenvolvendo a Camada Controller e View	127
8.4.1 Implementando o Listar	130
8.4.2 Implementando o Cadastrar	133
8.4.3 Implementando o Alterar.....	143
8.4.4 Implementando o Excluir	143
Capítulo 9 – Desenvolvimento de crud com Herança	145
9.1 Desenvolvendo a Camada Model	146
9.2 Desenvolvendo a Camada DAO	151
9.2.1 Implementando DAO de Administrador	156
9.2.2 Implementando DAO de Cliente e Fornecedor.....	161
9.3 Desenvolvendo a Camada View/Controller.....	161
9.3.1 Implementando a controller de Pessoa	162
Figura 209 – Criando o pacote para controller de Pessoa	162
9.3.2 Implementando a controller de Administrador	163
9.3.3 Implementando Servlet de atualização de lista de cidades	168
9.3.4 Implementando a Interface de Cadastro de Administrador.....	170
Capítulo 10 – Implementandando a segurança no sistema	180

10.1 Login de Usuário	180
10.2 controle de Usuário (Nível de Acesso)	195

INTRODUÇÃO

Vamos começar a trabalhar com a linguagem Java utilizando Java Server Faces para programação para plataforma Web. Para isso vamos começar instalando as ferramentas que utilizaremos em nosso curso.

Para o andamento de nossas aulas iremos utilizar a IDE de desenvolvimento Apache Netbeans que atualmente se encontra na versão 12.0 e para a persistência de dados iremos utilizar o SGBD PostgreSQL.

CAPÍTULO 1 – INSTALAÇÃO DO JAVA E CONFIGURAÇÃO DE AMBIENTE

Para iniciarmos no mundo devemos verificar inicialmente qual a JRE que é o software onde se encontra nossa Java Virtual Machine está instalada em nossa máquina. Além disso também iremos necessitar do JDK (Java Development Kit) do java para que possamos desenvolver nossos softwares em java.

Para verificar qual versão está instalada em sua máquina abra um prompt de comando no seu Windows e digite os comandos “java –version” e depois “javac –version” como você pode observar na Figura 1.

Figura 1 – Verificando a instalação do JDK / JRE



```
Prompt de Comando
Microsoft Windows [versão 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos os direitos reservados.

C:\Users\jeffe>java --version
java 14.0.1 2020-04-14
Java(TM) SE Runtime Environment (build 14.0.1+7)
Java HotSpot(TM) 64-Bit Server VM (build 14.0.1+7, mixed mode, sharing)

C:\Users\jeffe>javac --version
javac 14.0.1

C:\Users\jeffe>
```

Fonte: O autor.

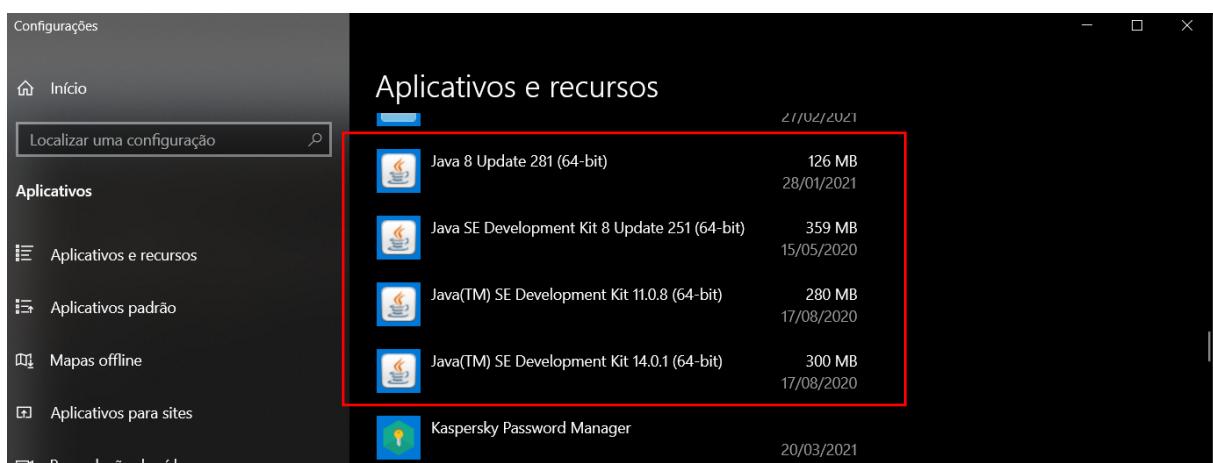
Como verifica-se na imagem em nosso computador temos instalado a versão do java 14.0.1 que representa nossa JRE (Java Virtual Machine) e do compilador java (JDK) javac na versão 14.0.1.

O Netbeans possui algumas incompatibilidades com versões mais novas da plataforma java como 11 ou 14 então recomenda-se durante o curso para evitar tais

problemas a utilização da versão do Java 8 que iremos fazer o download e instalar, essas versões da JDK são distribuídas pela Oracle detentora atual da plataforma.

Existem também versões da JDK feitas pela comunidade as chamadas “OpenJDK”, nosso Netbeans que utilizaremos durante as aulas também demonstra algumas incompatibilidades com elas. Deste modo verifique como demonstrado na Figura 1 qual a versão instalada em sua máquina e se necessário vá em “Adicionar e Remover Programas” no seu Windows e faça a desinstalação das versões existentes, como demonstrado na Figura 2.

Figura 2 – Removendo versões do java no windows



Fonte: O autor.

Assim queremos ficar apenas com a versão 8 do java em nossa máquina. Para alunos mais avançados é possível manter várias versões da plataforma instalada na máquina e utilizar o Netbeans com a versão 8, mas se vocês estão iniciando agora vamos pelo caminho mais simples, faça desinstalação.

Para realizarmos o download da JDK vamos até o site da Oracle (<https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html>), neste site (Figura 3) estarão disponíveis versões da JDK para vários sistemas operacionais diferentes escolha a versão para Windows de acordo com o seu sistema operacional de 64bits ou X86 32bits.

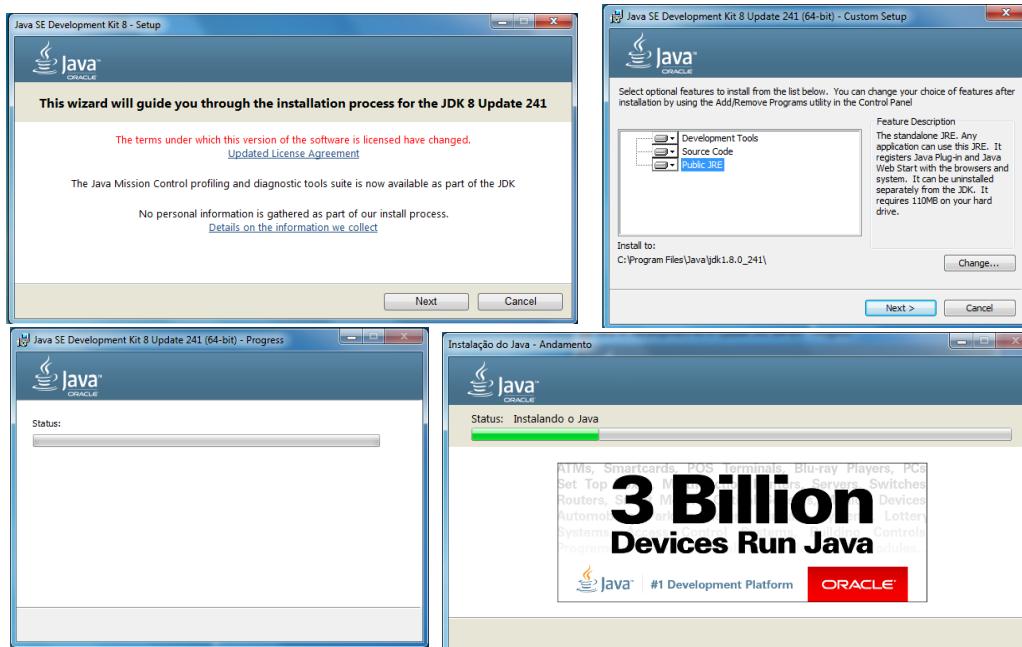
Figura 3 – Página de download do Java Development Kit

ORACLE		Q	Produtos	Recursos	Suporte	Eventos	Desenvolvedor	Exibir Contas	Entrar em contato com o departamento de vendas
Linux x64 RPM Package	108.06 MB								jdk-8u281-linux-x64.rpm
Linux x64 Compressed Archive	137.06 MB								jdk-8u281-linux-x64.tar.gz
macOS x64	205.26 MB								jdk-8u281-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	125.96 MB								jdk-8u281-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	88.77 MB								jdk-8u281-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	134.68 MB								jdk-8u281-solaris-x64.tar.Z
Solaris x64	92.66 MB								jdk-8u281-solaris-x64.tar.gz
Windows x86	154.69 MB								jdk-8u281-windows-i586.exe
Windows x64	166.97 MB								jdk-8u281-windows-x64.exe

Fonte: Oracle

Efetuado o download vamos fazer a instalação do java, assim execute seu instalador e siga os passos de acordo com as instruções que lhe forem fornecidas pelo software instalador, não tem segredo. Ao final recomendo que você reinicie seu Windows.

Figura 4 – Instalação da JDK

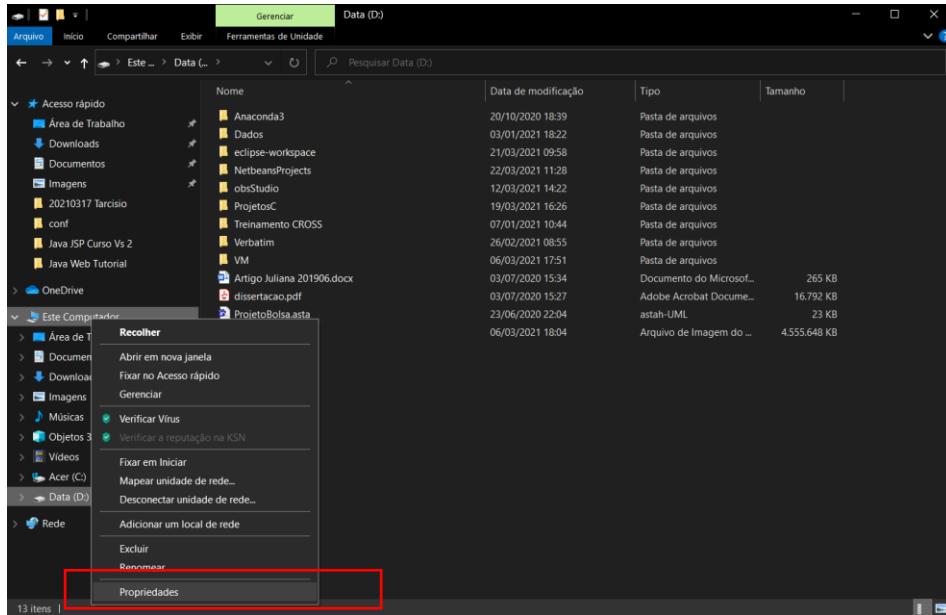


Fonte: O autor.

Agora que nossa instalação concluiu recomendo que novamente você reinicie sua máquina antes de continuarmos, com esses passos concluídos vamos configurar

nosso ambiente java em nosso sistema operacional Windows configurando as variáveis de ambiente necessárias para o bom funcionamento de nossos softwares.

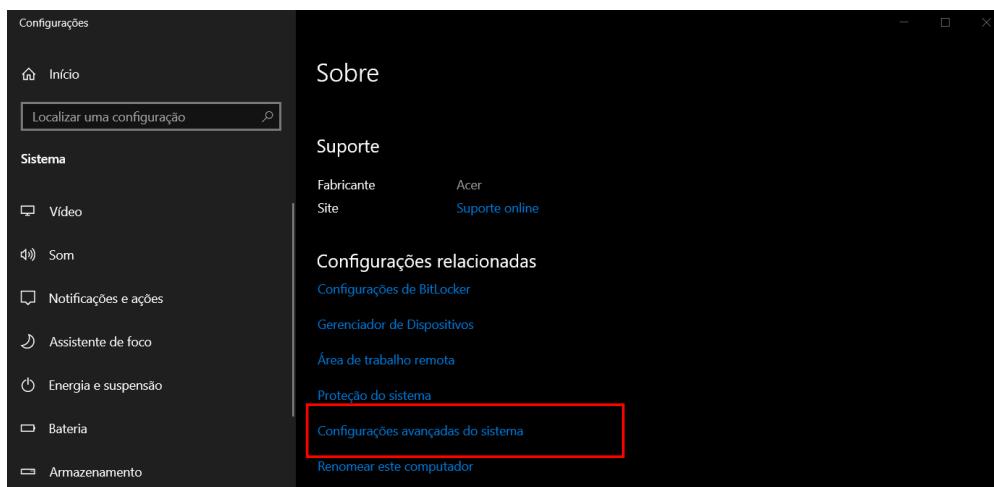
Figura 5 – Acessando configurações avançadas do windows



Fonte: O autor.

Podemos encontrar abrindo seu gerenciador de arquivos como está demonstrado na Figura 5 e clicar com o botão direito do seu mouse em “Este Computador” ou “Meu Computador” (podemos encontrar com nomenclaturas diferentes dependendo da versão do Windows) e escolher propriedades onde abrirá uma tela de “Sobre” descrevendo as configurações de seu computador e do Windows instalado.

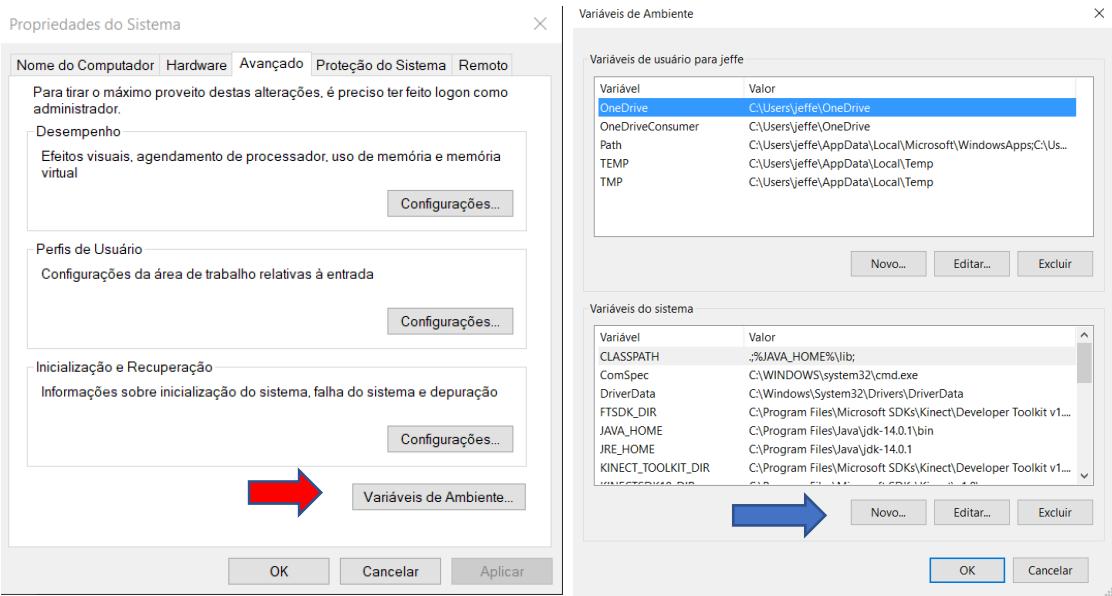
Figura 6 – Tela Sobre (Windows)



Fonte: O autor.

Com essa tela aberta role o scroll para baixo e encontre o link “Configurações avançadas do sistema” e clique nesta opção (Figura 6) abrindo a tela de propriedades do Sistema onde devemos clicar no botão “Variáveis de Ambiente” como demonstrado com a seta vermelha (Figura 7).

Figura 7 – Tela Propriedades do Sistema e Variáveis de Ambiente



Fonte: O autor.

Com a tela de variáveis de ambiente aberta vamos configurar nosso ambiente java, para isso vamos clicar na opção “Novo” em na caixa variáveis de ambiente conforme a seta azul na Figura 7 e crie as variáveis que estiverem faltando conforme a necessidade em sua máquina. As variáveis devem apontar para os diretórios de instalação do Java que você acabou de realizar, você irá encontrá-lo no drive C em arquivos de programas.

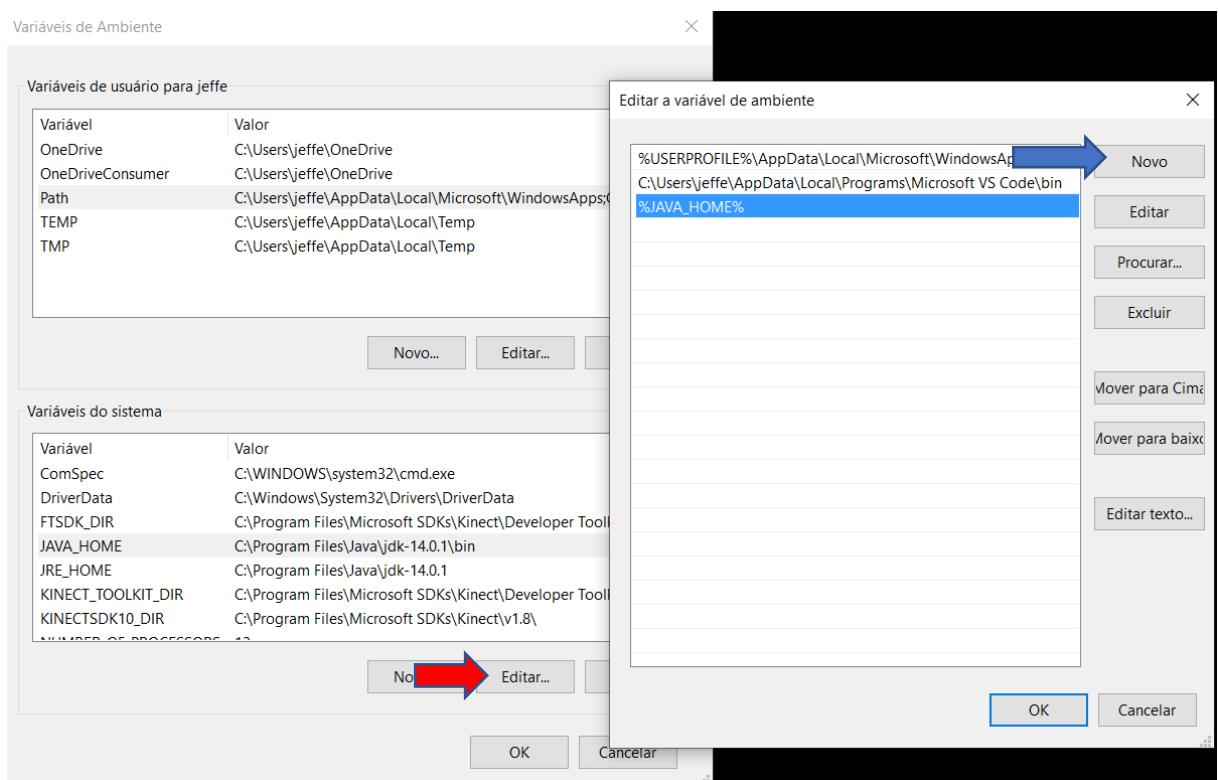
Quadro 1 - Variáveis de Ambiente (JAVA)

Variável	Valor
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_281
JRE_HOME	C:\Program Files\Java\jdk1.8.0_281\jre
CLASSPATH	.;%JAVA_HOME%\lib;

Fonte: O autor.

Agora que você criou as variáveis de ambiente necessárias vamos adicionar ao final da variável PATH o valor ;%JAVA_HOME%\bin; exatamente como está escrito inclusive os pontos e vírgula como segue o exemplo (Figura 8).

Figura 8 – Tela Propriedades do Sistema



Fonte: O autor.

Selecione a variável PATH e clique em editar (seta vermelha) e na tela que se abrir vocês poderão verificar o conteúdo pré-existente nesta variável, clique no botão novo (seta azul) e coloque o conteúdo conforme descrito anteriormente. Feito isso confirme todas as alterações e feche as janelas. Agora é necessário que você reinicie sua máquina novamente para que as configurações dessas variáveis se apliquem ao seu sistema operacional.

Após reiniciar vocês podem verificar sua instalação do Java e das variáveis de ambiente configuradas fazendo o teste como foi descrito anteriormente na Figura 1, rodando os comandos no prompt de comando “java –version” e “javac –version”.

CAPÍTULO 2 – INSTALAÇÃO E CONFIGURAÇÃO DO POSTGRESQL

Agora vamos instalar e configurar nosso sistema gerenciador de banco de dados o PostgreSQL que utilizaremos para realizar a persistência dos dados em nossos projetos java durante o curso.

O PostgreSQL é um SGBD sob a licença de software livre podemos encontrar seu instalador no site : <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> onde podemos observar diversas versões disponíveis para os mais diferentes sistemas operacionais, vamos selecionar a versão 12.6 para Windows se sua máquina for de 64bits, senão você deverá escolher uma versão mais antiga com compatibilidade ao seu SO x86-32bits como a versão 10.16 ou 9.6 (Figura 9).

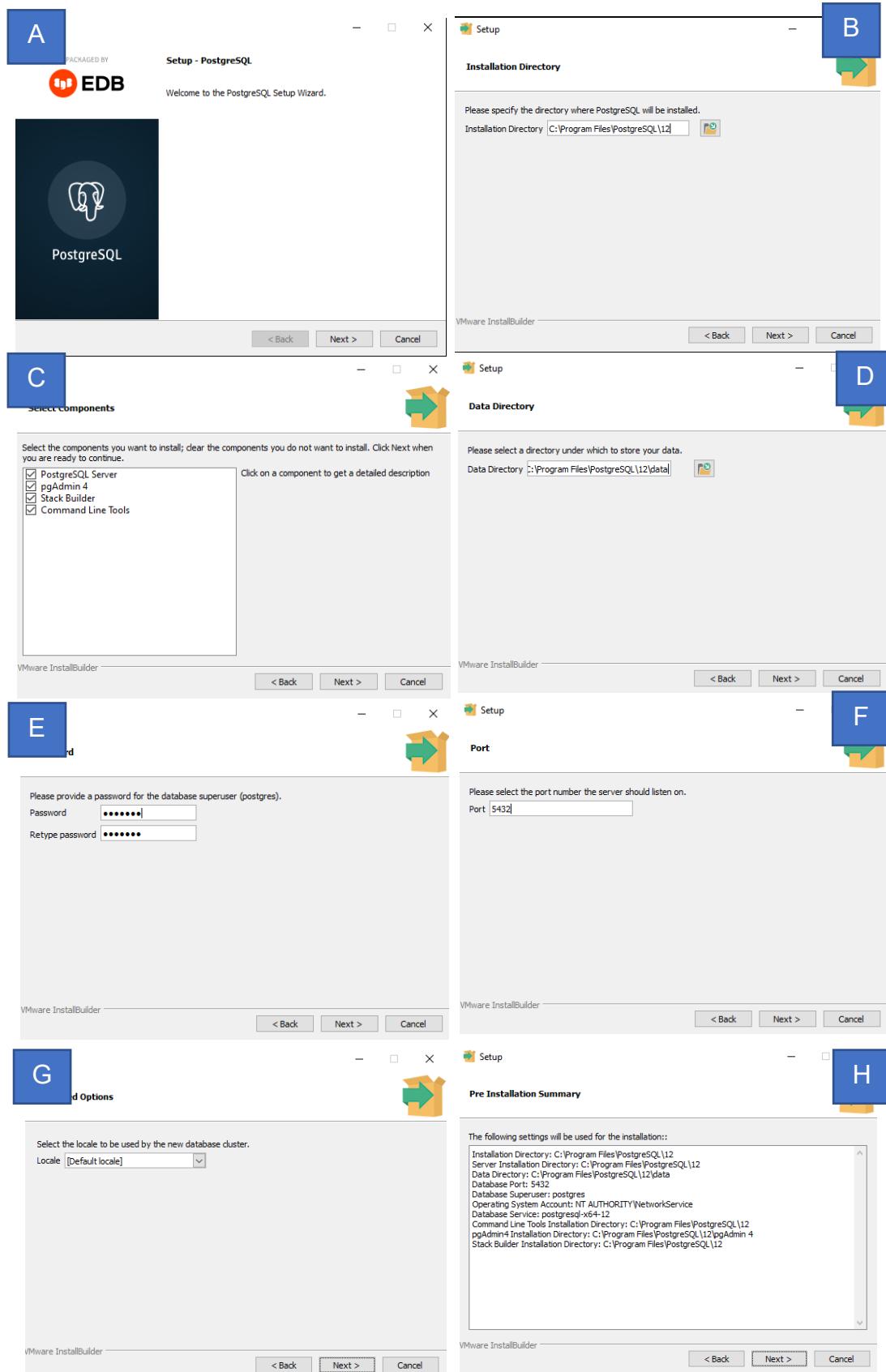
Figura 9 – Tela Propriedades do Sistema

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
13.2	N/A	N/A	Download	Download	N/A
12.6	N/A	N/A	Download	Download	N/A
11.11	N/A	N/A	Download	Download	N/A
10.16	Download				
9.6.21	Download				
9.5.25 (Not Supported)	Download				
9.4.26 (Not Supported)	Download				
9.3.25 (Not Supported)	Download				

Fonte: PostgreSQL.org.

Realizado o download podemos iniciar a instalação do PostgreSQL, clique no instalador que vamos acompanhar os passos de instalação através da Figura 10. O processo consiste basicamente em confirmar as opções mostradas pelo software instalador. Na Figura10a é informado o diretório de instalação, clique em next; na Figura 10c os componentes que serão instalados, cliquem em next; na Figura 10d o diretório onde os dados ficarão armazenados, clique em next.

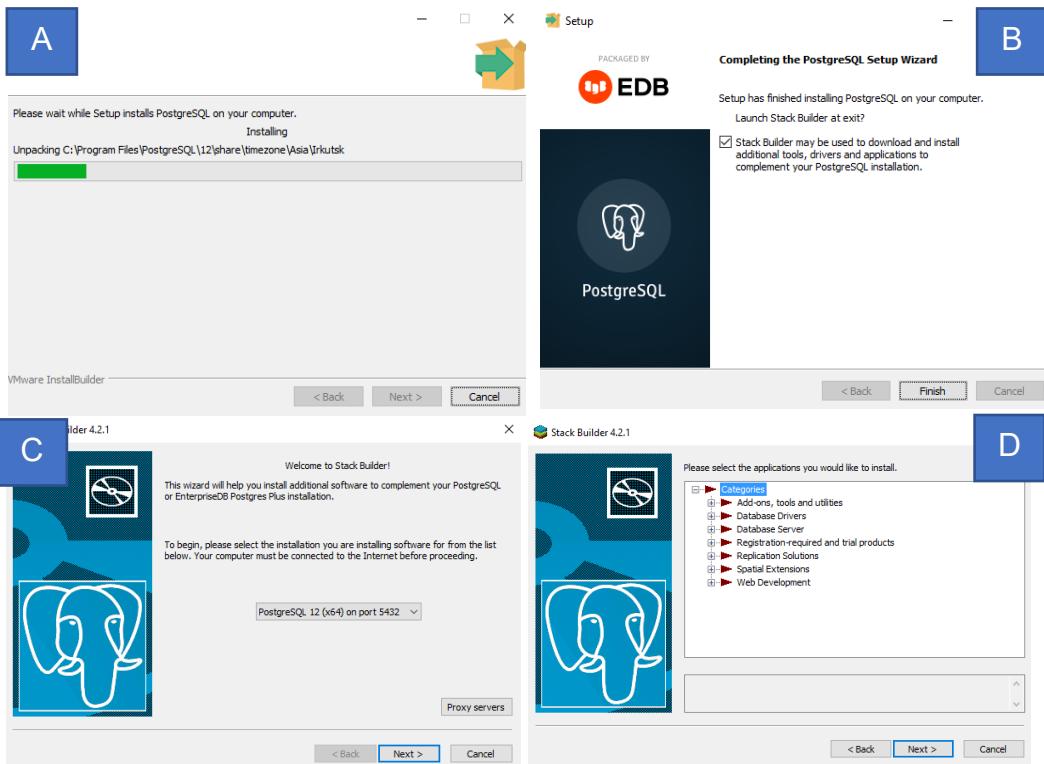
Figura 10 – Processo de Instalação do PostgreSQL



Fonte: O autor.

A Figura 10e demanda sua atenção você precisa informar uma senha para o administrador do SGBD e depois clicar em next; a Figura 10f demonstra a porta padrão por onde o servidor responderá as requisições e nas telas Figura 10g e Figura 10h fazem um resumo do processo de instalação, você deve clicar em next em ambos os casos.

Figura 11 – Processo de Instalação do PostgreSQL - Continuação

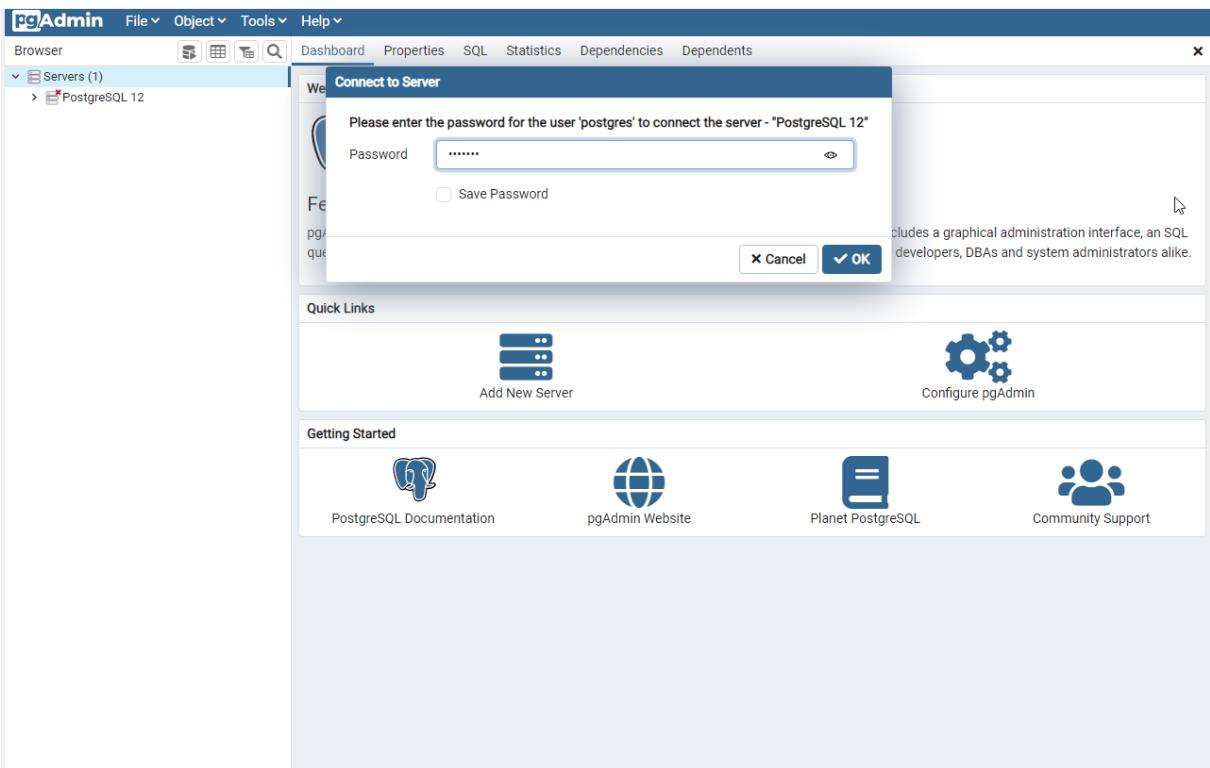


Fonte: O autor.

Na Figura 11a observamos a instalação ocorrendo; na Figura 11b o instalador avisa que abrirá o Stack Builder que é o software gerenciador da instalação do SGBD e nos permite adicionar novos recursos ao nosso software instalado, o Stack Builder aparece na Figura 11c na combo no meio da tela selecione o servidor instalado (será o único de sua máquina provavelmente) e na Figura 11d ele mostrará uma listagem dos recursos instalados e não instalados, neste momento é só clicar em cancel e fechar o aplicativo. Pronto seu servidor de banco de dados PostgreSQL está instalado.

Para verificarmos procure em seu computador a aplicação pgAdmin4 que é por onde realizaremos as operações dentro do SGBD. A tela do pgAdmin4 pode ser observada na Figura 12, durante a abertura se for solicitado uma senha, é só informar a senha configurada durante o processo de instalação.

Figura 12 – Processo de Instalação do PostgreSQL - Continuação



Fonte: O autor.

CAPÍTULO 3 – INSTALAÇÃO E CONFIGURAÇÃO DO NETBEANS

Para iniciar no mundo Java vamos instalar nossa ferramenta de desenvolvimento o Apache Netbeans. O Apache Netbeans é um software desenvolvido pela comunidade open source de onde recebe muitas contribuições que são gerenciadas pela Fundação Apache. A Apache recebeu originalmente o código fonte do Netbeans em sua versão 8 da Oracle quando a mesma decidiu descontinuar o produto em 2016.

Para fazer o download de nosso Netbeans vamos acessar a página (<https://netbeans.apache.org/>) onde iremos visualizar a página (Figura 13).

Figura 13 - Página Apache Netbeans



Fonte: Apache Foundation

Na página principal vamos escolher a opção download onde poderemos visualizar as versões disponíveis para utilizarmos, conforme pode-se visualizar na Figura 14.

Figura 14 – Versões para download – Apache Netbeans

The screenshot shows the Apache NetBeans releases page. At the top, it features a large blue banner for 'Apache NetBeans 12.3' with a 'Find out more' button. Below the banner, the heading 'Apache NetBeans Releases' is displayed. Underneath, it lists 'Apache NetBeans 12 feature update 3 (NB 12.3)' as the latest version, released on March 3, 2021. It also lists 'Apache NetBeans 12 LTS (NB 12.0)' as the latest LTS version, released on June 4, 2020. Both entries have 'Features' and 'Download' buttons. A link to 'Older releases' is also present.

Fonte: Apache Foundation

Como pode-se verificar existem diversas versões disponíveis, mas sempre prefira as versões do tipo LTS, pois são versão já testadas e livres de erros indesejados. Atualmente devemos então fazer o download da versão Apache Netbeans 12 (LTS), clicando no respectivo botão download (Figura 14).

Então irá abrir a página de download onde devemos escolher o instalador de acordo com o nosso sistema operacional (Figura 15).

Figura 15 – Download do Apache Netbeans

Downloading Apache NetBeans 12.3

Apache NetBeans 12.3 was released March 3, 2021. See [Apache NetBeans 12.3 Features](#) for a full list of features.

Apache NetBeans 12.3 is available for download from your closest Apache mirror.

- Binaries: [netbeans-12.3-bin.zip \(SHA-512, PGP ASC\)](#)
- Installers:
 - [Apache-NetBeans-12.3-bin-windows-x64.exe \(SHA-512, PGP ASC\)](#)
 - [Apache-NetBeans-12.3-bin-linux-x64.sh \(SHA-512, PGP ASC\)](#)
 - [Apache-NetBeans-12.3-bin-macosx.dmg \(SHA-512, PGP ASC\)](#)
- Source: [netbeans-12.3-source.zip \(SHA-512, PGP ASC\)](#)
- Javadoc for this release is available at <https://bits.netbeans.org/12.3/javadoc>

[Deployment Platforms](#)

[Building from Source](#)

[Community Approval](#)

[Earlier Releases](#)

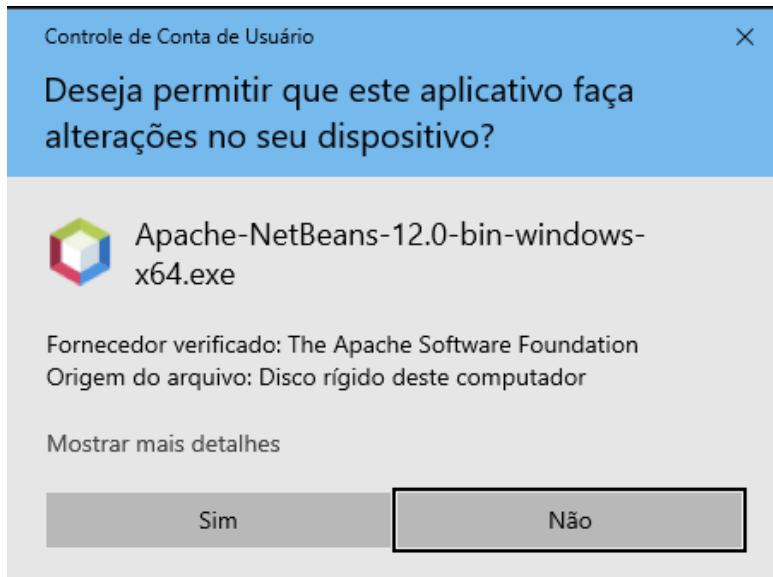
Officially, it is important that you [verify the integrity](#) of the downloaded files using the PGP signatures (.asc file) or a hash (.sha512 files). The PGP keys used to sign this release are available [here](#).

Apache NetBeans can also be installed as a self-contained [snap package](#) on Linux.

Fonte: Apache Foundation

Então irá abrir a página de download onde devemos escolher o instalador de acordo com o nosso sistema operacional (Figura 15) e fazer e baixar o instalador, então podemos iniciar a instalação do Apache Netbeans.

Figura 16 – Tela Inicial do instalador do Netbeans



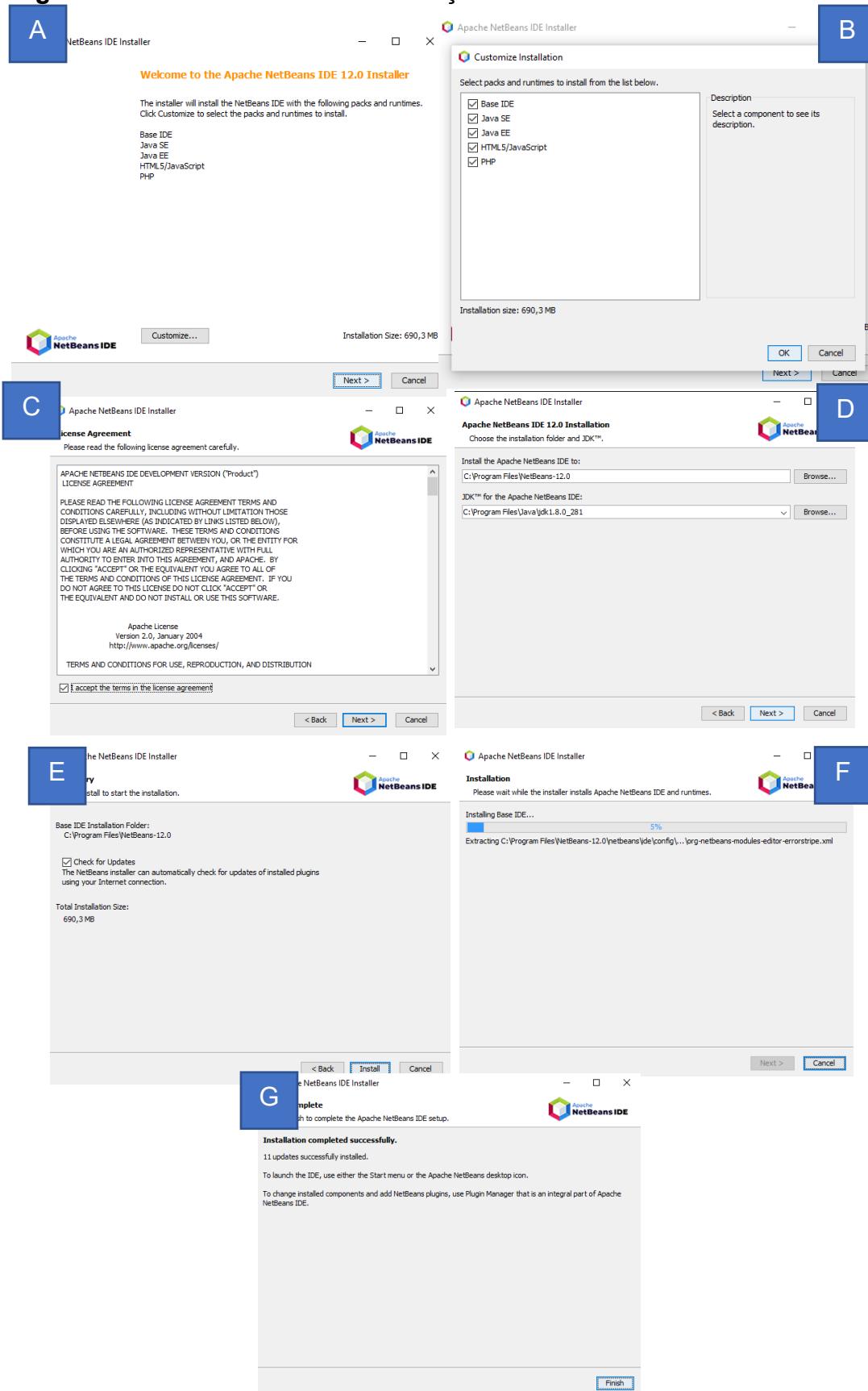
Fonte: O autor.

Na Figura 17 pode-se acompanhar os passos do software durante o processo de instalação do Apache Netbeans 12. Na Figura 17a observa-se a tela de boas-vindas do instalador e devemos clicar no botão “customize” que abrirá a tela demonstrada na Figura 17b onde temos que conferir se todas as opções estão selecionadas.

Na Figura 17c devemos concordar com os termos do software e clicar em next. Na Figura 17d será exibido os diretórios de instalação do Netbeans e o diretório onde está instalado o JDK do java, clique em next.

Na Figura 17e temos um resumo final antes da instalação, clique em next e na Figura 17f temos o processo de instalação. Quando finalizado o instalador exibirá a tela demonstrada na Figura 17g encerrando o processo.

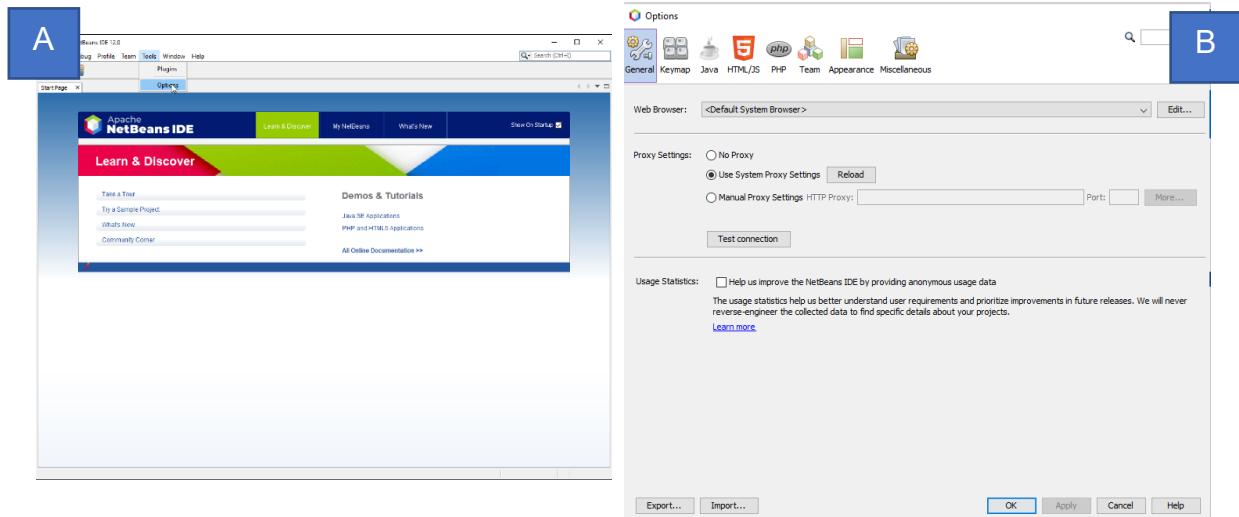
Figura 17 – Passo a Passo da instalação do Netbeans



Fonte: O autor.

Após finalizado o processo de instalação devemos abrir o Netbeans para realizarmos as configurações necessárias para o seu funcionamento correto. Com o Netbeans aberto escolha o menu nas opções “Tools → Options”, conforme demonstrado na Figura 18a.

Figura 18 – Configuração do Apache Netbeans



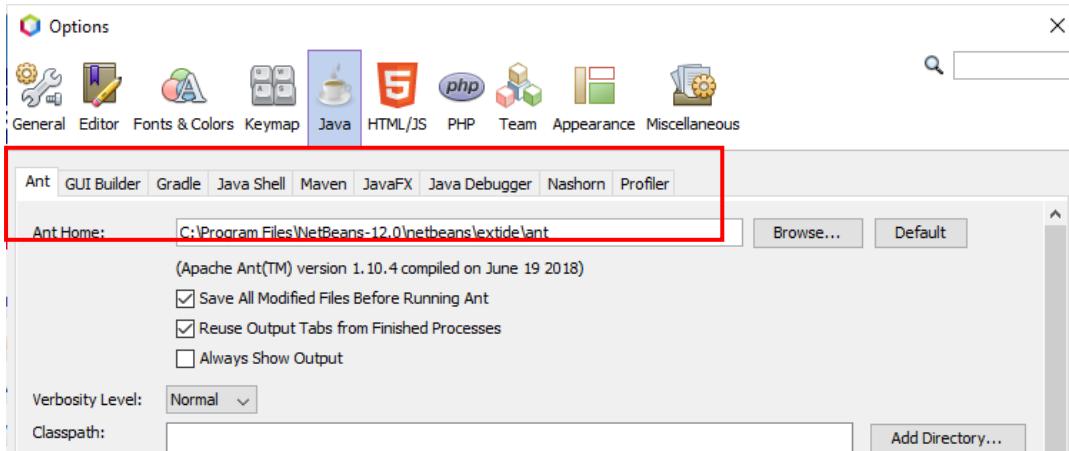
Fonte: O autor.

Na Figura 18b podemos observar os botões “General”, “Editor”, “Fonts & Colors”, “Keymap”, “Java”, “Html/Js”, “PHP”, “Team”, “Appearance” e “Miscellaneous”, clique em cada um dos botões para o Netbeans ativar os recursos e inserir os plugins necessários para o funcionamento, a janela pode sumir e reaparecer algumas vezes não se assuste.

Conforme pode ser visualizado na Figura 19 iremos realizar a ativação de recursos para o Java. Então com o botão “Java” selecionado percorra (clique) em cada uma das abas da tela para que o Netbeans ative e instale os plugins necessários ao funcionamento do Java.

Atenção: Durante esse processo o Apache Netbeans pode solicitar a instalação de algum software/plugin adicional, você deve concordar e instalar!

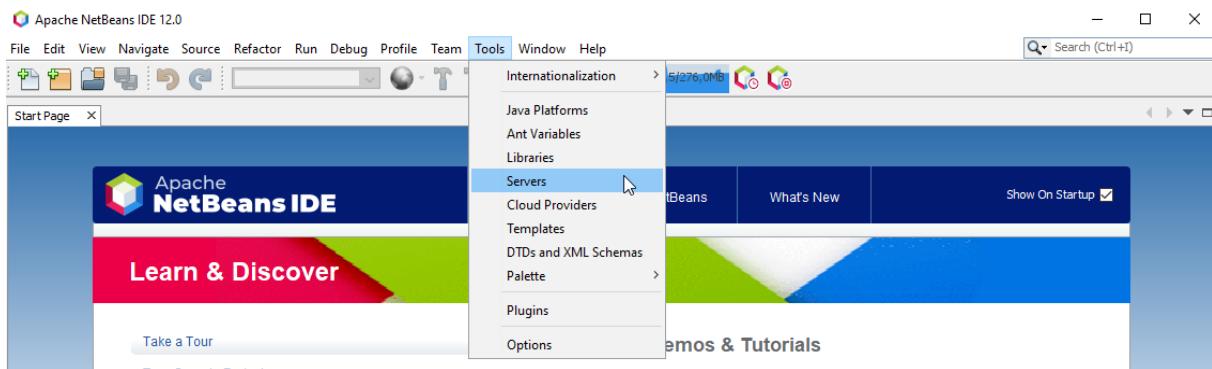
Figura 19 – Configuração do Java



Fonte: O autor.

Realizado a operação de ativação dos recursos do Netbeans devemos instalar nosso servidor de Web Container que suportará nossas aplicações Java Web. Para isso escolha a opção no menu “Tools → Servers” conforme demonstrado na Figura 20a.

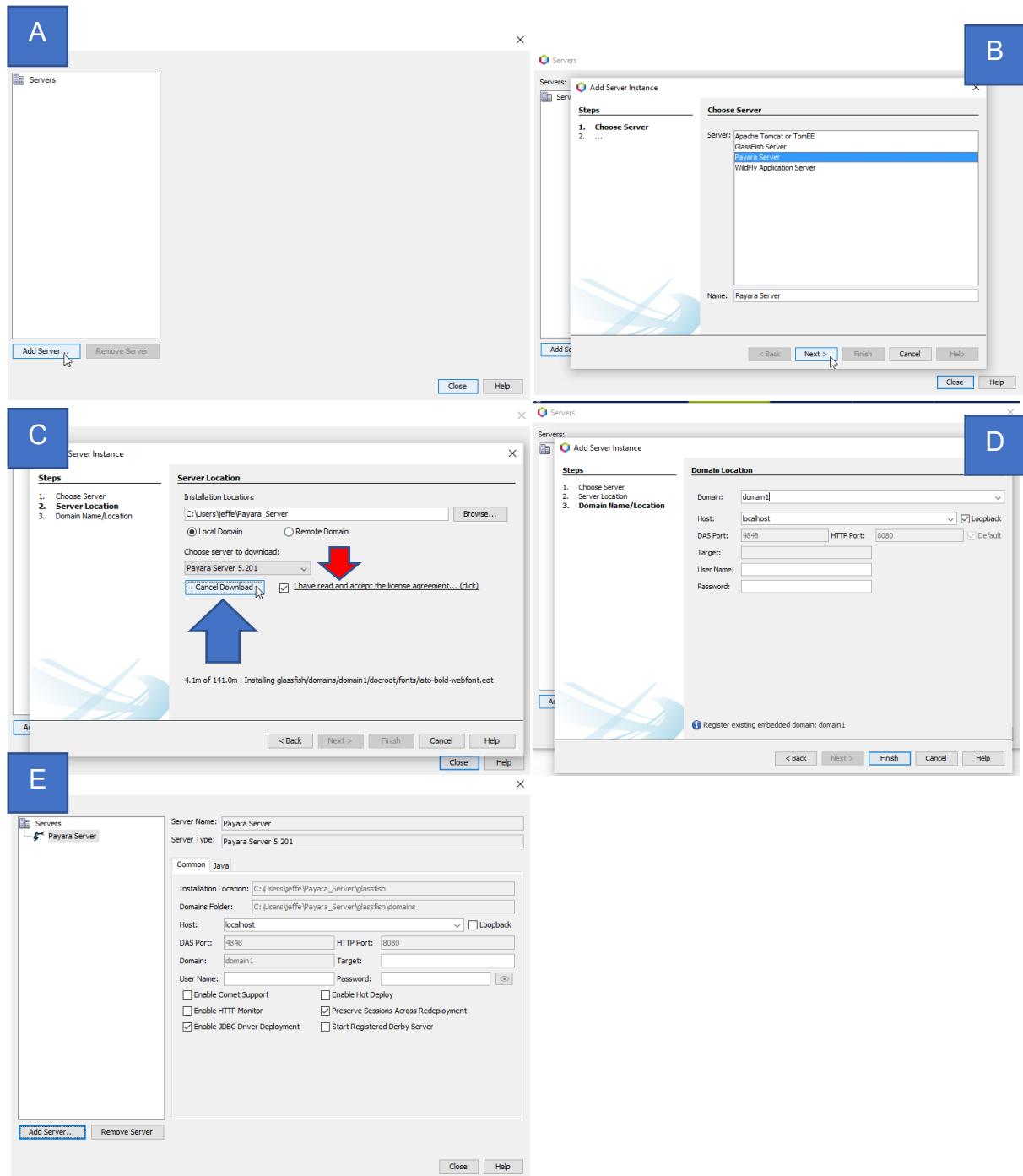
Figura 20 – Configuração do Java



Fonte: O autor.

Realizado a operação de ativação dos recursos do Netbeans devemos instalar nosso servidor de aplicação Payara no Netbeans, para isso siga os passos de acordo como o demonstrado na Figura 21.

Figura 21 – Configuração do Servidor Web Payara no Netbeans



Fonte: O autor.

Na Figura 21a pode-se visualizar a tela de instalação de servidores no Apache Netbeans 12, clique em “Add Server” o que lhe encaminhará a tela descrita na Figura 21b onde deveremos escolher a opção Payara Server e clicar em “next”. Após clicar em next o Netbeans irá realizar a ativação do Java EE o que nos permitirá fazer desenvolvimento de software para web.

Na Figura 21c deve-se checar a caixa para permitir o download do software do servidor web Payara e clicar no botão de download, o mesmo iniciará o download do software como demonstrado nesta figura. Terminado o download na Figura 21d pode-se observar as configurações gerais do servidor, você deve apenas clicar em next e na tela descrita na Figura 21e temos a tela final da instalação do sevidor web e podemos clicar em close.

CAPÍTULO 4 – CRIAÇÃO DO PROJETO EM JAVA WEB

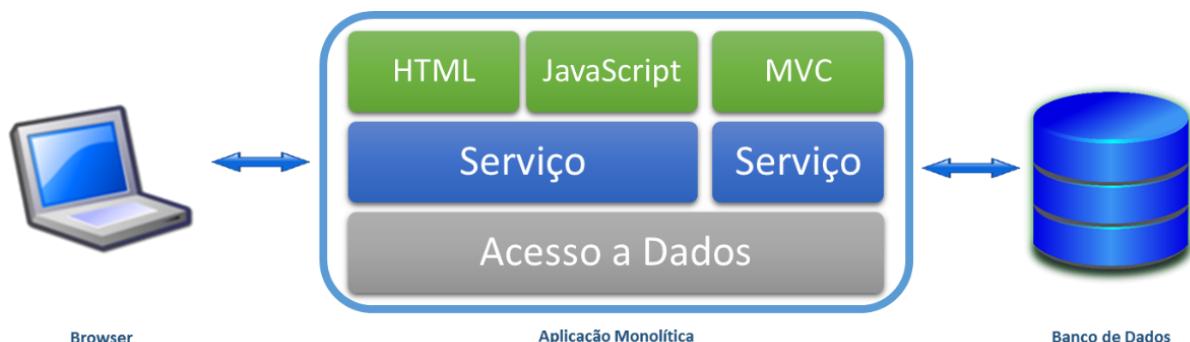
Uma aplicação Web possui uma complexidade maior que uma aplicação desktop pois envolve conexões remotas de rede através da internet, servidores e conhecimento de protocolos.

Podemos construir softwares utilizando arquiteturas diferentes, neste projeto iremos construir um software com arquitetura monolítica ao invés de utilizarmos micro serviços que são um tema mais avançado e requer maior conhecimento do desenvolvedor.

As principais linguagens de programação de aplicações oferecem abstrações para quebrar a complexidade dos sistemas em módulos. Entretanto são projetadas para a criação de um único executável monolítico, no qual toda a modularização utilizada é executada em uma mesma máquina e em um mesmo processo no servidor.

Deste modo, os módulos compartilham recursos de processamento, memória, banco de dados e arquivos. Uma arquitetura monolítica típica de um sistema complexo pode ser representada pela Figura 22, na qual todas as funções de negócio estão implementadas em um único processo.

Figura 22 – Estrutura de uma aplicação monolítica



Fonte: Opus Software (opus-software.com.br)

O desenvolvimento de software monolíticos são mais simples do que em micro serviços, mas nos impõe desafios como o aumento da complexidade e tamanho ao longo do tempo tornando a manutenção mais cara e lenta, pois os desenvolvedores tem que navegar em um infinidade de códigos.

Também estamos expostos a problemas com alta dependência de componentes de código, onde muitas funções são implementadas e entrelaçadas de forma que a inclusão ou manutenção de componentes do sistema pode causar inconsistências ou comportamentos inesperados.

Os desenvolvedores também enfrentam a falta de flexibilidade ficando amarrados a tecnologia originalmente escolhida no projeto. A escalabilidade também é prejudicada pois o sistema deve sempre ser aplicado por completo em qualquer nova instalação ou instância de servidores.

Existem também dificuldades para colocar alterações em produção pois qualquer mudança exige que o desenvolvedor reinicialize todo o sistema, obrigando as equipes de desenvolvimento, testes e manutenção acompanham essas implantações.

A tecnologia monolítica de desenvolvimento de software ainda é muito utilizada apesar das dificuldades expostas e é uma excelente maneira de iniciar o aprendizado na área de desenvolvimento de software. Lembrando sempre que você deve buscar novos conhecimentos a partir dessa base e o desenvolvimento de aplicações com micro serviços deve ser sempre um dos seus objetivos.

4.1 ARQUITETURA MVC

A arquitetura MVC (model-view-controller) foi criada nos anos 80 na Xerox Parc, por Trygve Reenskaug, sua implementação original foi descrita no artigo “Applications Programming in Smalltalk-80: How to use Model-View-Controller”.

A camada Model (Camada de modelo ou da lógica da aplicação) é a ponte entre as camadas Visão (View) e Controle (Controller), consiste na parte lógica da aplicação, que gerencia o comportamento dos dados através de regras de negócios, lógica e funções. Esta fica apenas esperando a chamada das funções, que permite o acesso para os dados serem coletados, gravados e, exibidos.

É o coração da execução, responsável por tudo que a aplicação vai fazer a partir dos comandos da camada de controle em um ou mais elementos de dados, respondendo a perguntas sobre a sua condição e a instruções para mudá-las. O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema que está se tentando resolver. Modela

os dados e o comportamento por trás do processo de negócios. Se preocupa apenas com o armazenamento, manipulação e geração de dados. É um encapsulamento de dados e de comportamento independente da apresentação.

A camada de apresentação ou visualização (View) é responsável por qualquer saída de representação dos dados, como uma tabela ou um diagrama. É onde os dados solicitados do Modelo (Model) são exibidos. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. A Visão também provoca interações com o usuário, que interage com o Controle (Controller). O exemplo básico disso é um botão gerado por uma Visão, no qual um usuário clica e aciona uma ação no Controle.

Não se dedica em saber como o conhecimento foi retirado ou de onde ela foi obtida, apenas mostra a referência. Segundo Gamma et al (2006), “A abordagem MVC separa a View e Model por meio de um protocolo inserção/notificação (subscribe/notify). Uma View deve garantir que sua expressão reflita o estado do Model. Sempre que os dados do Model mudam, o Model altera as Views que dependem dele. Em resposta, cada View tem a oportunidade de modificar-se”. Adiciona os elementos de exibição ao usuário: HTML, ASP, XML, JSP. É a camada de interface com o usuário. É utilizada para receber a entrada de dados e apresentar visualmente o resultado.

A camada de controle ou controlador (Controller) é o componente final da tríade, faz a mediação da entrada e saída, comandando a visão e o modelo para serem alterados de forma apropriada conforme o usuário solicitou através do mouse e teclado. O foco do Controle é a ação do usuário, onde são manipulados os dados que o usuário insere ou atualiza, chamando em seguida o Modelo.

O Controle (Controller) envia essas ações para o Modelo (Model) e para a janela de visualização (View) onde serão realizadas as operações necessárias.

O padrão de arquitetura de software MVC (Model-View-Controller) é muito utilizado nos projetos de software. Analisamos os seus elementos e como eles interagem entre si. Além disso, vimos as principais vantagens de adotar um padrão como o MVC que se caracteriza pela facilidade na obtenção de múltiplas visões dos mesmos dados, desacoplagem da interface da lógica da aplicação, entre outras vantagens. No entanto, vimos que definir a arquitetura de um software é ainda mais do que escolher o seu padrão arquitetural, precisamos definir a tecnologia (J2EE ou .Net), linguagens a integrar, forma de persistência, interfaces com o usuário e muito

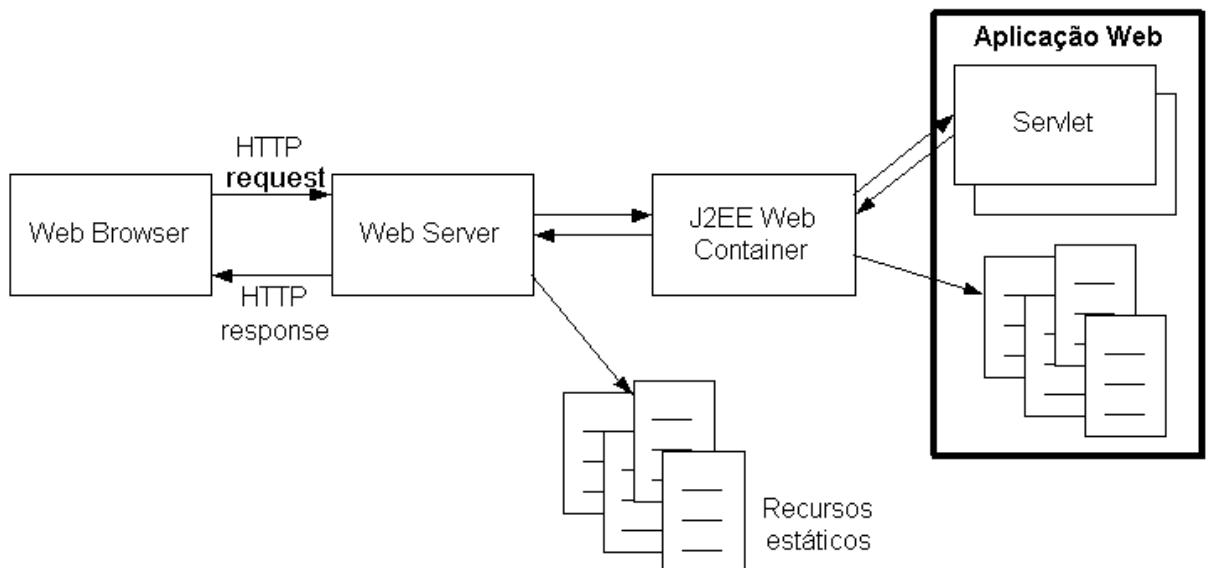
mais. Dessa forma, devemos sempre reunir o máximo de pessoas experientes possíveis, conhecimentos, análise do projeto, todos os seus requisitos funcionais e não funcionais, a fim de definirmos a melhor arquitetura possível para o software que está sendo desenvolvido.

4.2 ARQUITETURA DO PROJETO DESENVOLVIDO NO CURSO

Uma aplicação web é composta por um lado cliente (client side) onde um usuário acessa recursos de um servidor através de um browser geralmente utilizando para isso o protocolo HTTP. Existe também o lado servidor da aplicação (server side) onde encontramos a máquina servidora com os recursos disponíveis para atender a requisição do cliente.

O server side geralmente é composto por um servidor web (web server), no nosso caso específico de um J2EE Web Container (Tomcat, Glassfish, Payara por ex.) que hósperá nossas aplicações java e da aplicação web hospedada no servidor. Na Figura 23 pode-se observar essa estrutura.

Figura 23 – Estrutura de uma aplicação web.



Fonte: Devmedia

Nossa aplicação também seguirá o padrão MVC (Model-View-Controller) mas iremos aumentar a modularização de nosso projeto, separando por exemplo a camada Model em Model e DAO (Data Access Object) que será responsável pelas transações com o banco de dados.

O Quadro 2 descreve as camadas de nossa aplicação, devendo-se enfatizar que cada uma das camadas será representada na aplicação por um pacote java.

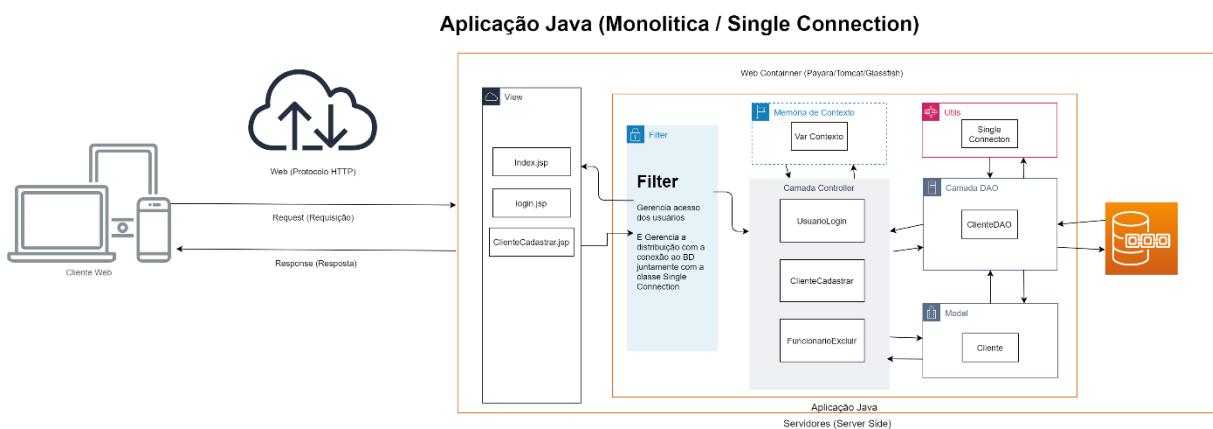
Quadro 2 – Camadas da Aplicação desenvolvida durante o curso.

Camada	Pacote/Pasta	Descrição
View	Web Pages ou Páginas Web	Gera a interface da aplicação web através de arquivos JSP que utilizam linguagens de marcação HTML, CSS e javascript e permite a utilização conjunta de código java que será interpretado antes de enviado ao cliente.
Filter	br.com.curso.filter	Camada Filter funciona como um funil onde todas as requisições passam por ela antes de chegar a alguma controller na aplicação. Assim é possível gerenciar a conexão com o banco de dados e garantir a segurança do sistema.
Controller	br.com.curso.controller	Camada responsável por atender as requisições do lado cliente e encaminhar as operações para a model e dao.
Model	br.com.curso.model	Define o modelo de dados e as regras de negócio de cada uma das classes do sistema.
DAO	br.com.curso.dao	Classe responsável por gerenciar as operações com o banco de dados.
Utils	br.com.curso.utils	Camada que armazena nossa classe geradora de conexões com o banco de dados e outras classes acessórias ao sistema.

Fonte: O autor.

As camadas de nossa aplicação podem ser representadas como exposto na Figura 24.

Figura 24 – Estrutura de Camadas da aplicação desenvolvida no curso.



Fonte: O autor.

Como podemos observar na estrutura proposta para a aplicação a ser desenvolvida optamos por utilizar um padrão Singleton para a classe geradora de conexões com o banco de dados. Deste modo essa classe administrará as conexões com o banco de dados de modo a que não existe mais de uma conexão aberta ao mesmo tempo. O sistema irá se conectar ao SGBD quando iniciar no servidor de aplicações web.

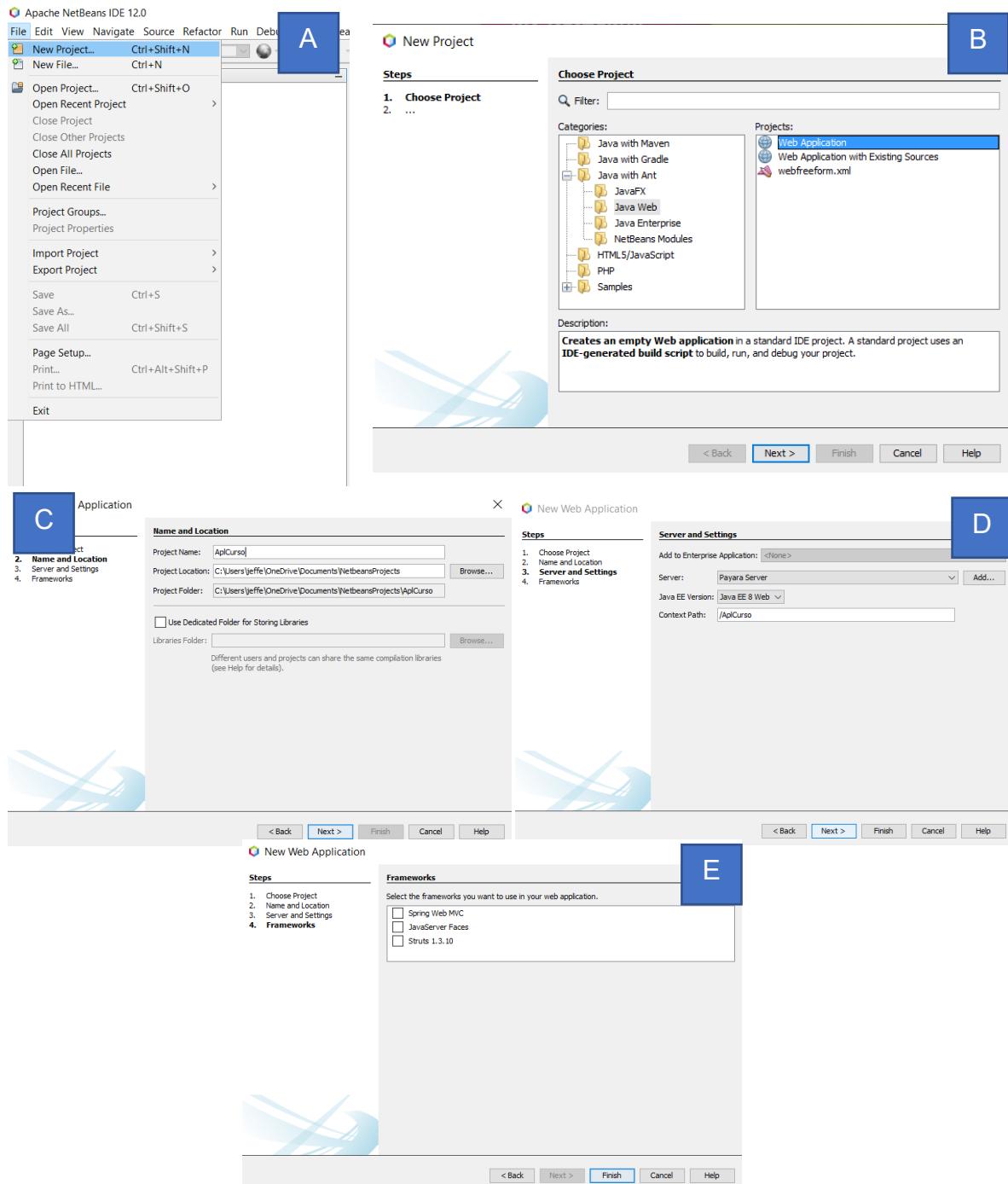
O padrão singleton visa garantir que uma determinada classe tenha apenas uma única instância na aplicação, deste modo, o padrão singleton desabilita todos os outros meios de criar objetos de uma classe exceto pelo método especial de criação. Este método tanto cria um objeto ou retorna um objeto já existente se ele já tenha sido criado.

4.3 CRIANDO O PROJETO DO CURSO NO NETBEANS

Iniciaremos nossas tarefas práticas de programação com a criação do projeto java web no Apache Netbeans e suas estruturas como a criação dos pacotes, importação de bibliotecas necessárias ao funcionamento do sistema e em um segundo momento a criação do banco de dados que fará a persistência de informações em nosso software.

Com o netbeans aberto vamos clicar no menu File → New Project para termos acesso a tela de criação de projetos conforme visualiza-se na Figura 25a.

Figura 25 – Criando Projeto Java JSP no Netbeans



Fonte: O autor.

Na Figura 25b podemos verificar o tipo de projeto que iremos criar, assim na caixa “Categories” escolhemos a opção “Java with Ant” e “Java Web” e na caixa “Projects” escolhemos o tipo de projeto “Web Application” e em seguida clicamos em next.

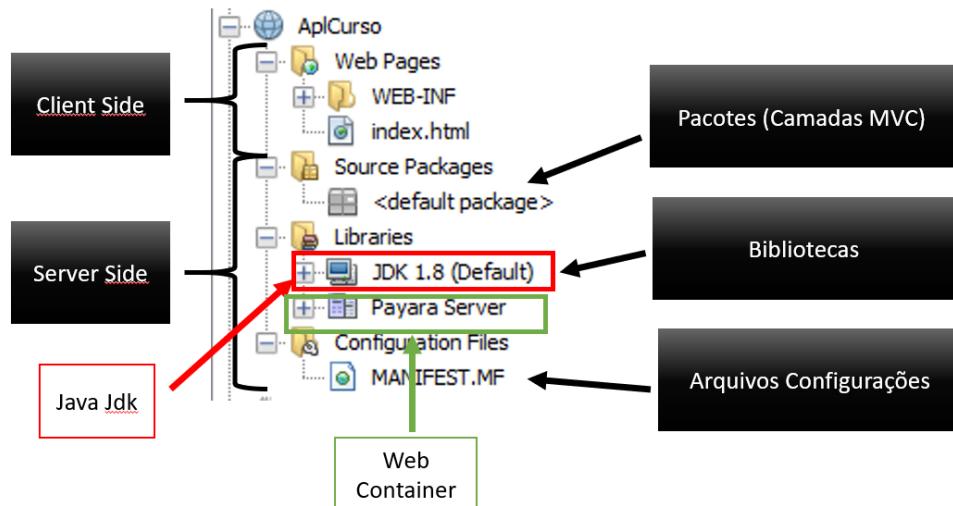
Na Figura 25c informamos o nome do projeto, então preencha a caixa com “ApICurso”, essa tela também nos informa os diretórios onde o projeto será criado, fique atento pois é ai que você conseguirá fazer uma cópia de seus projetos.

Na Figura 25d visualizamos o Web Container que utilizaremos, aparece então selecionado na caixa server o servidor Payara e se observa a versão do Java utilizada no projeto – Java EE 8 Web. Nesta figura também temos o “Context Path” de nossa aplicação, que representa a raiz da estrutura de nosso projeto web.

Na Figura 25e temos os frameworks disponíveis para adicionarmos ao projeto, mas no momento vamos trabalhar com JSP puro, então é só clicar no botão finish e aguardar a criação do projeto.

Na Figura 26 pode-se observar a estrutura gerada no projeto web pelo Netbeans.

Figura 26 – Estrutura do Projeto Web (Netbeans)



Fonte: O autor.

O projeto possui uma estrutura composta de 3 partes principais Web Pages (Páginas Web), Source Packages (Pacotes de Código Fonte) e Libraries (Bibliotecas), conforme pode ser observado na Figura 25.

Em Web Pages encontram-se os arquivos JSP e será responsável para a interface de nossa aplicação web, sendo construída com HTML, CSS, Javascript e comandos Java através de scriptlets ou da biblioteca JSTL.

Em Source Packages temos o back end de nossa aplicação web nesta parte do projeto serão criadas as camadas Controller, DAO e Model de nosso software.

A camada controller é responsável pelo controle da comunicação entre o Front End da aplicação e o Back End da aplicação.

Ela implementa o protocolo HTTP e gerencia as requisições (request) e as respostas (response) dentro da aplicação desenvolvida.

A camada DAO é responsável por gerenciar a comunicação com o banco de dados e realiza a conversão entre os paradigmas Orientado a Objetos (Java) e relacional (Banco de Dados).

A camada Model implementa as classes que representarão nosso modelo de dados do sistema, como por exemplo classes cliente, estado, fornecedor etc.

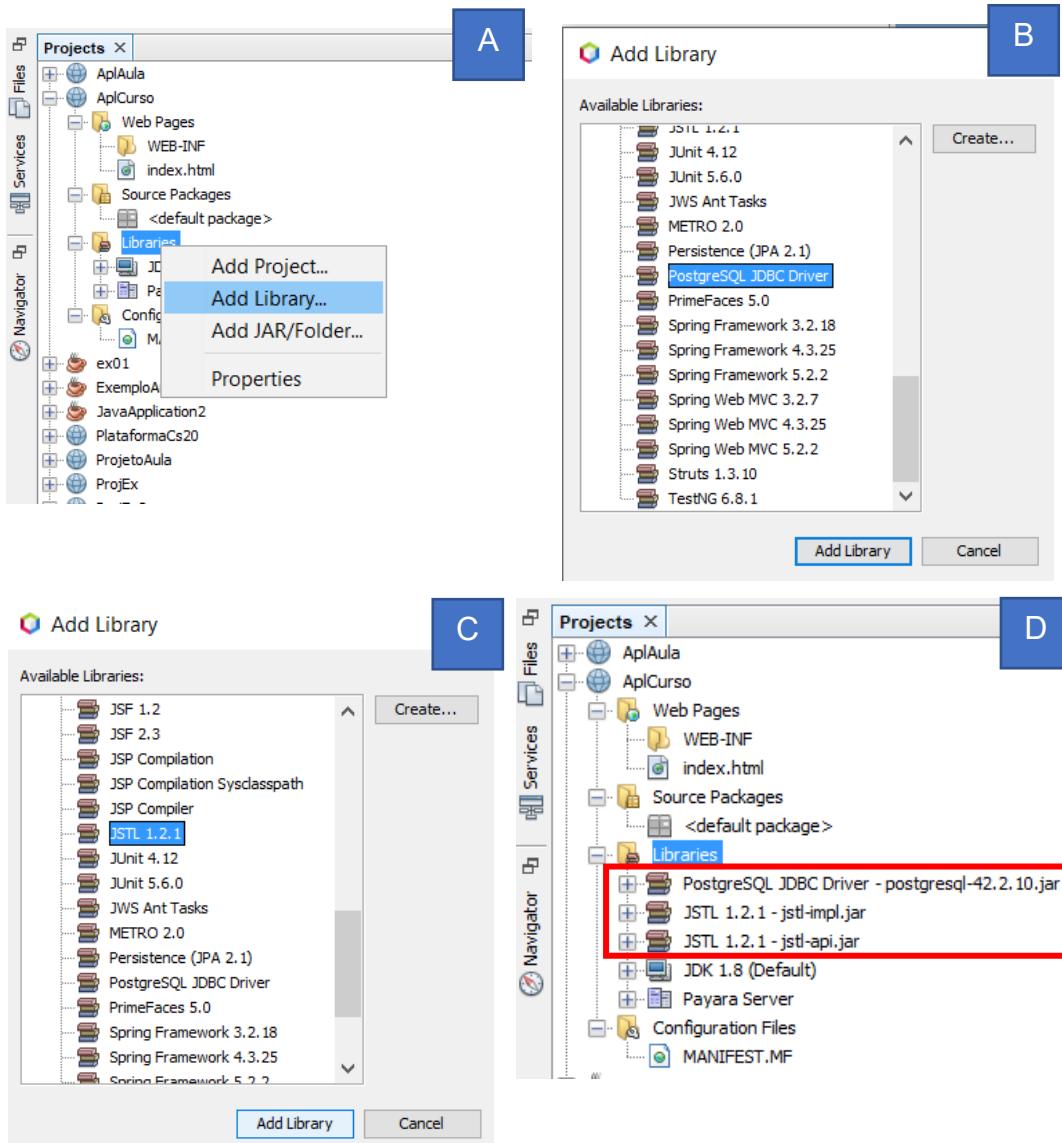
Uma vez criado o projeto vamos prepará-lo importando as bibliotecas necessárias. Para isso demos clicar com o botão direito sobre a opção Libraries (bibliotecas) e escolher a opção adicionar biblioteca (Add Libraries), como demonstrado na Figura 27a. Abrindo a tela de importação de bibliotecas para o projeto.

Agora vamos importar as bibliotecas “PostgreSQL JDBC Driver” que é o driver de acesso ao sistema gerenciador de banco de dados e a biblioteca “JSTL 1.2.1”, como pode verificar-se nas Figuras 27b e 27c.

A biblioteca “JSTL 1.2.1”, que significa JavaServer Pages Standard Tag Library, é um componente da plataforma de desenvolvimento web Java EE. Ela estende a especificação JSP adicionando uma biblioteca de tags das tags JSP para tarefas comuns, tais como processamento de dados XML, execução condicional, loops e internacionalização.

Na Figura 27d podemos verificar as bibliotecas adicionadas ao projeto na pasta “Libraries”.

Figura 27 – Inserindo bibliotecas no projeto



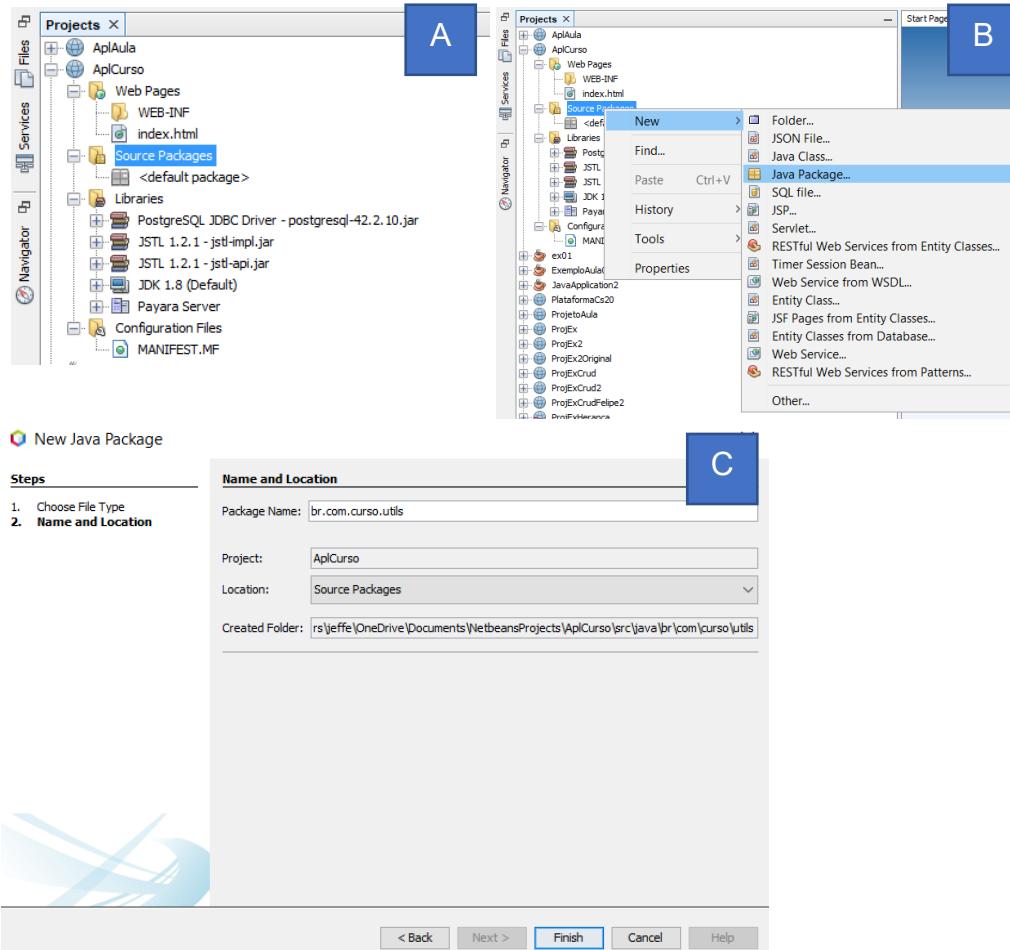
Fonte: O autor.

Próximo passo é criar os pacotes java dentro do Source Packages, neste momento nosso projeto não tem nenhum pacote criado como observamos na Figura 28a. Para inserir pacotes no projeto basta clicar com o botão direito do mouse no grupo Source Packages (Pacotes de código fonte) e escolher New (Novo) → Java Package (Pacote Java) como demonstrado na Figura 28b onde abrirá a tela de criação de pacotes Figura 28c. Deverão ser criados os pacotes conforme o Quadro 2:

- br.com.curso.utils;
- br.com.curso.model;
- br.com.curso.dao;

- br.com.curso.filter;
- os pacotes representam as camadas do lado servidor de nosso projeto.

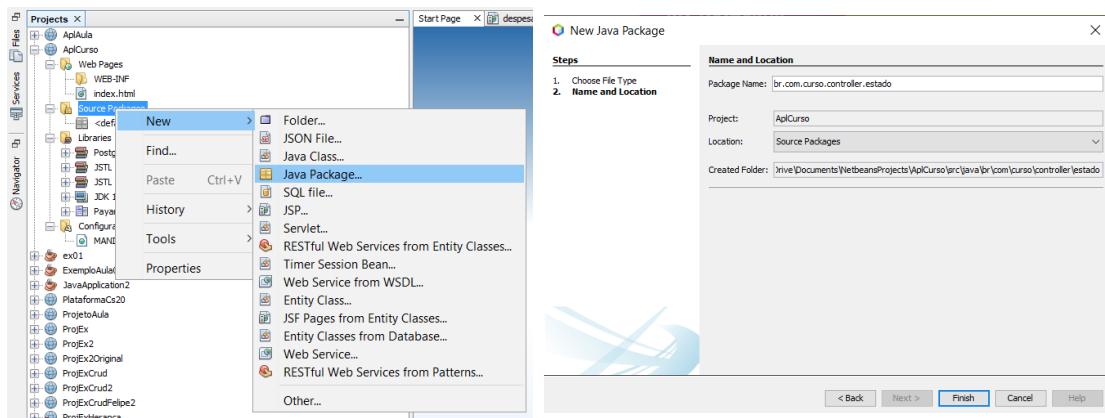
Figura 28 – Adicionando pacotes no projeto



Fonte: O autor.

Em relação a camada controller iremos criar um pacote para cada model existente no sistema, por exemplo o primeiro cadastro que iremos fazer no curso será o cadastro de Estado, então, vamos criar um pacote controller para essa classe Figura 29. Quando inserirmos uma nova classe model em nosso sistema incluiremos outro pacote controller.

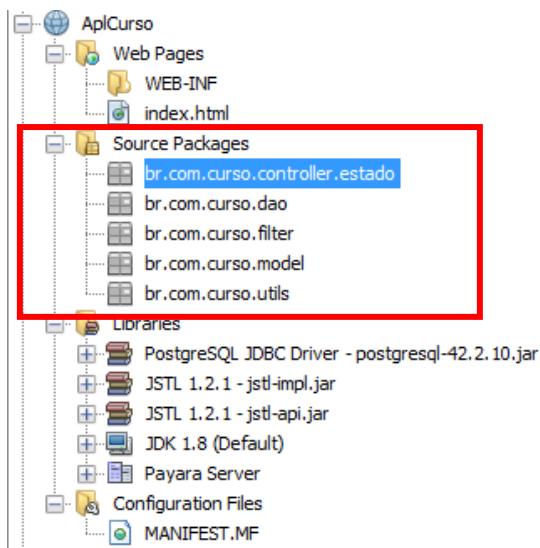
Figura 29 – Adicionando pacote da camada controller



Fonte: O autor.

O resultado da criação dos pacotes podemos observar na Figura 30.

Figura 30 – Pacotes criados no projeto



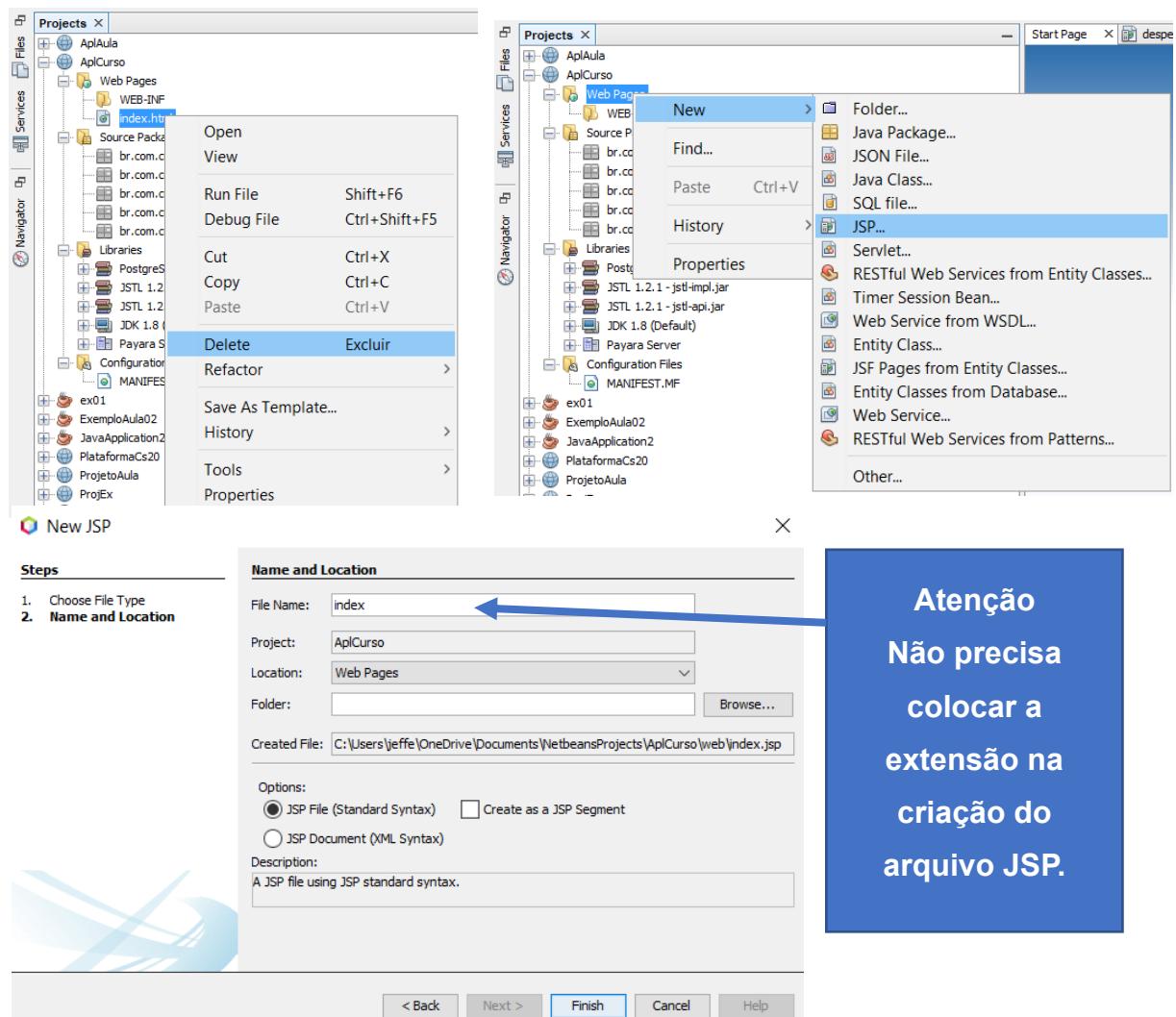
Fonte: O autor.

Na camada view precisamos trocar o arquivo index.html gerado pelo netbeans por um arquivo do tipo jsp, então devemos excluir referido arquivo, para isso clique com o botão direito sobre o arquivo e escolha a opção “delete” (Figura 31a).

Excluído o arquivo index.html vamos criar um novo arquivo index agora do tipo jsp, clique com o botão direito do seu mouse na pasta “Web Pages” escolha a opção “New” e depois “JSP” (Figura 31b) abrindo a caixa de criação de arquivos JSPs,

preencha com o nome do arquivo a ser criado “index.jsp” e confirme a criação (Figura 31c).

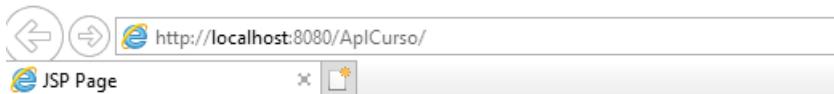
Figura 31 – Criando a index.jsp



Fonte: O autor.

Você já pode executar o seu projeto web, pressione o botão executar na IDE Netbeans e teremos o resultado como demonstrado na Figura 31.

Figura 32 – Executando o projeto pela primeira vez



Hello World!

Fonte: O autor.

4.4 ESTRUTURANDO A INTERFACE DO PROJETO

Agora precisamos organizar a modularização de nossa interface de modo a reaproveitarmos cabeçalho (header), menu e rodapé (footer) iguais e em estrutura única em todo o sistema.

Assim vamos começar criando os arquivos JSP necessários para a nossa interface, iniciando por programarmos o index.jsp que está criado em nosso projeto. Apague o código gerado pelo netbeans e substitua pelo código demonstrado na figura 33.

Figura 33 – Código para index.jsp

```
1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <jsp:include page="home.jsp"/>
3 |
```

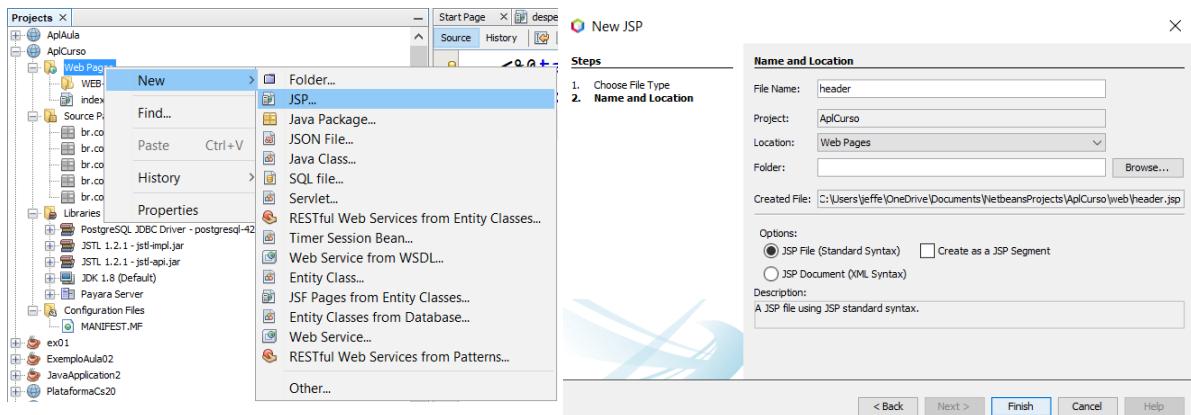
Fonte: O autor.

Na linha 2 de código observa-se a tag importando a biblioteca JSTL Core e na linha seguinte temos a tag include inserindo a home.jsp.

Você pode se perguntar – por que não faço o código da home.jsp direto na index se já estou importando mesmo?, bom quando evoluirmos em nosso projeto a index precisará tomar a decisão de encaminhar o usuário para a home ou para a tela de login de acordo com situação de o usuário já estar logado ou não.

Na sequência antes de desenvolvermos a home.jsp, vamos preparar alguns arquivos jsp acessórios que irão delinear o formato de modularização que adotaremos no sistema, assim vamos criar o arquivo header.jsp clicando com o botão direito do mouse sobre a pasta Web Pages escolhendo a opção novo e jsp conforme temos na Figura 34.

Figura 34 – Criando header.jsp



Fonte: O autor.

E vamos apagar o código gerado pelo netbeans e substituir como está na Figura 35.

Figura 35 – Código header.jsp

```

1 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6     <title>JSP Page</title>
7     <!-- JQuery -->
8     <script src="${pageContext.request.contextPath}/js/jquery-3.3.1.min.js"></script>
9     <script src="${pageContext.request.contextPath}/js/jquery.mask.min.js"></script>
10    <script src="${pageContext.request.contextPath}/js/jquery.maskMoney.min.js"></script>
11    <!-- Bootstrap -->
12    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
13    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/umd/popper.min.js"></script>
14    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
15
16    <!-- Datatable -->
17    <link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/1.10.22/css/jquery.dataTables.min.css"/>
18    <script src="https://cdn.datatables.net/1.10.22/js/jquery.dataTables.min.js" type="text/javascript"></script>
19
20    <!-- Mensagem alerta -->
21    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@10.3.1/dist/sweetalert2.all.min.js" type="text/javascript">
22      </script>
23    </head>
24    <body>

```

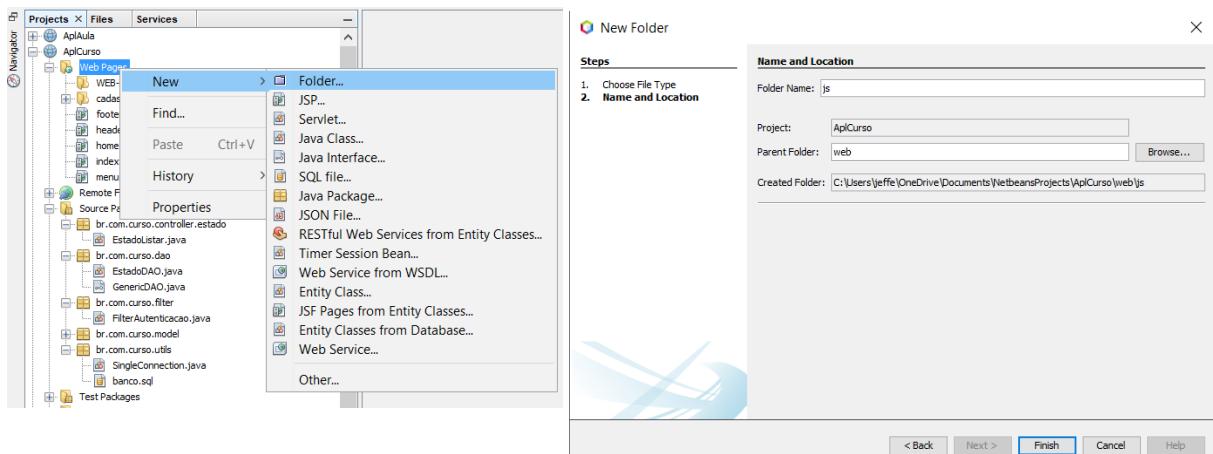
Fonte: O autor.

Na Figura 35 temos nas linhas 8,9 e 10 as importações de bibliotecas referentes ao Jquery e o mask e maskmoney que servirão para colocarmos máscaras

em nossos campos html input. Nas linhas 12, 13 e 14 temos importações referentes ao bootstrap, nas linhas 17 e 18 importações referentes ao datatables e na linha 21 biblioteca para envio de mensagens de aviso na aplicação.

Durante o andamento de nosso curso trabalharemos todos esses componentes. Especificamente para os componentes Jquery, Mask e Maskmoney temos que importá-los em nosso projeto, para isso vamos criar uma pasta clicando com o botão direito sobre “Web Pages” e escolhendo a opção “New->Folder”, abrindo a tela de criação da pasta vamos denominá-la como “js”, onde serão armazenados os arquivos correspondentes ao componentes importados na header. Esses arquivos serão fornecidos junto com este tutorial.

Figura 36 – Criando pasta JS para JQuery

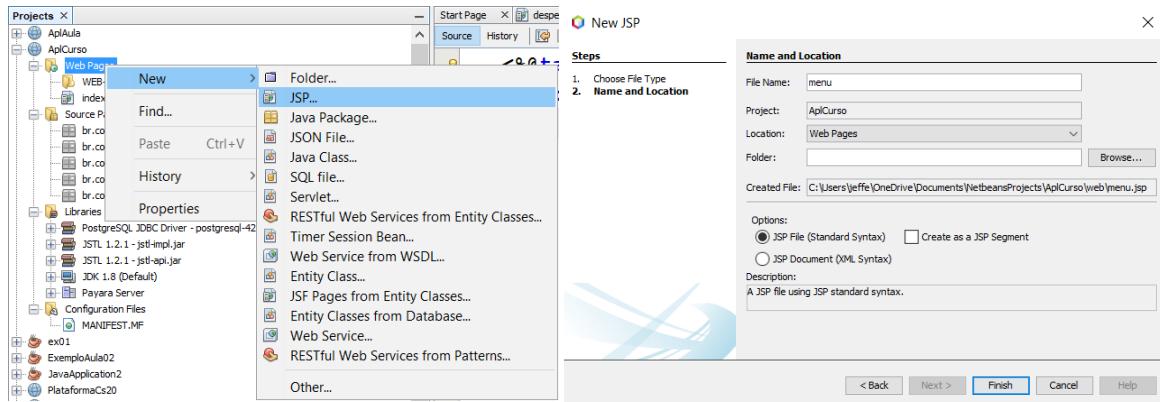


Fonte: O autor

Para colocar os arquivos JS fornecidos na pasta JS de nosso projeto basta arrastá-los e soltá-los na pasta indicada.

Agora vamos criar o arquivo para armazenarmos nosso menu, para isso repita o processo de clicar com botão direito do seu mouse em Web Pages e escolher a opção New → JSP como está demonstrado na Figura 36, criando nosso “menu.jsp”.

Figura 37 – Criando menu.jsp



Fonte: O autor.

E vamos apagar o código gerado pelo netbeans e substituir como está na Figura 38.

Figura 38 – Código menu.jsp

```

1 <h1>Módulo Cadastros</h1>
2 <hr>
3     <center>
4         <h2>Menu Principal</h2>
5         <a href="${pageContext.request.contextPath}/EstadoListar">Estado</a>
6         <a href="${pageContext.request.contextPath}/CidadeListar">Cidade</a>
7     </center>
8 <hr>

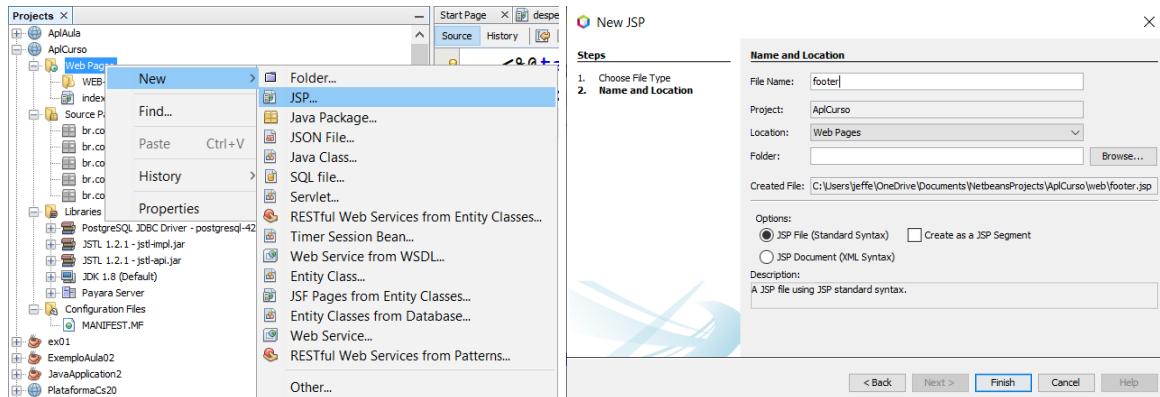
```

Fonte: O autor.

Podemos verificar na Figura 38 o código HTML para nosso menu e nas linhas 5 e 6 chamadas para outros recursos de nosso sistema que logo desenvolveremos. A Expression Language (EL) → "\${pageContext.request.contextPath}" tem como função posicionar a chamada ao recurso a partir da raiz do projeto.

Agora vamos criar o nosso footer.jsp conforme demonstrado anteriormente e representado na Figura 39.

Figura 39 – Criando footer.jsp



Fonte: O autor.

Como já fizemos com os outros JSPs vamos apagar o código gerado pelo netbeans e substituir pelo nosso código. Na Figura 40 podemos verificar o código HTML que fecha as tags <body> e <html> abertos em nosso header.jsp.

Figura 40 – Código footer.jsp

```

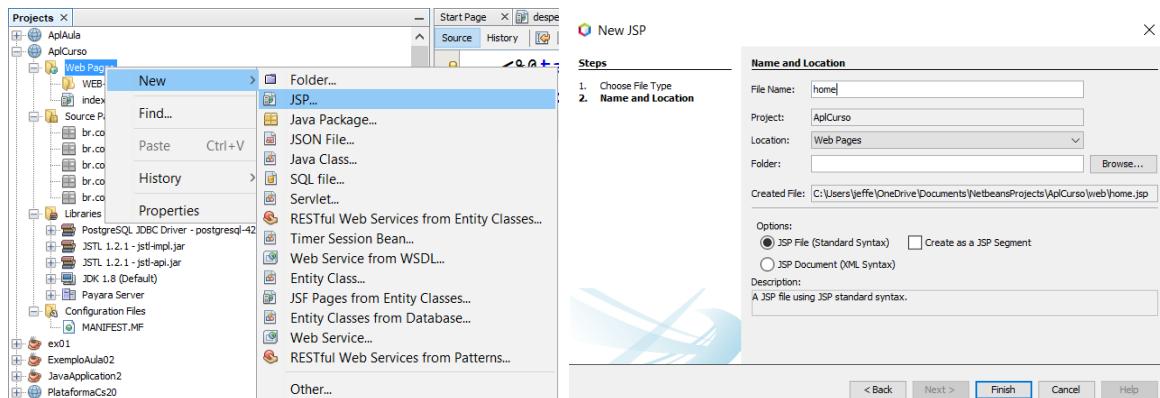
1 <hr>
2 <p>Desenvolvendo Aplicações com Java Web</p>
3 </body>
4 </html>

```

Fonte: O autor.

Agora vamos criar nossa home.jsp que será a página principal de nossa aplicação, para isso basta seguir os passos anterior representados na Figura 41.

Figura 41 – Criando home.jsp



Fonte: O autor.

Na Figura 42 podemos verificar que a home.jsp é uma junção das JSPs criadas anteriormente (linhas 4,5 e 9) isso ocorrerá em todas as páginas criadas em nosso sistema de modo que o header o menu e o rodapé da aplicação sejam únicos facilitando a manutenção de nosso sistema devido a sua modularização.

Podemos observar também nas linhas 1 e 2 as tags TagLib com as importações da biblioteca JSTL.

Figura 42 – Código home.jsp

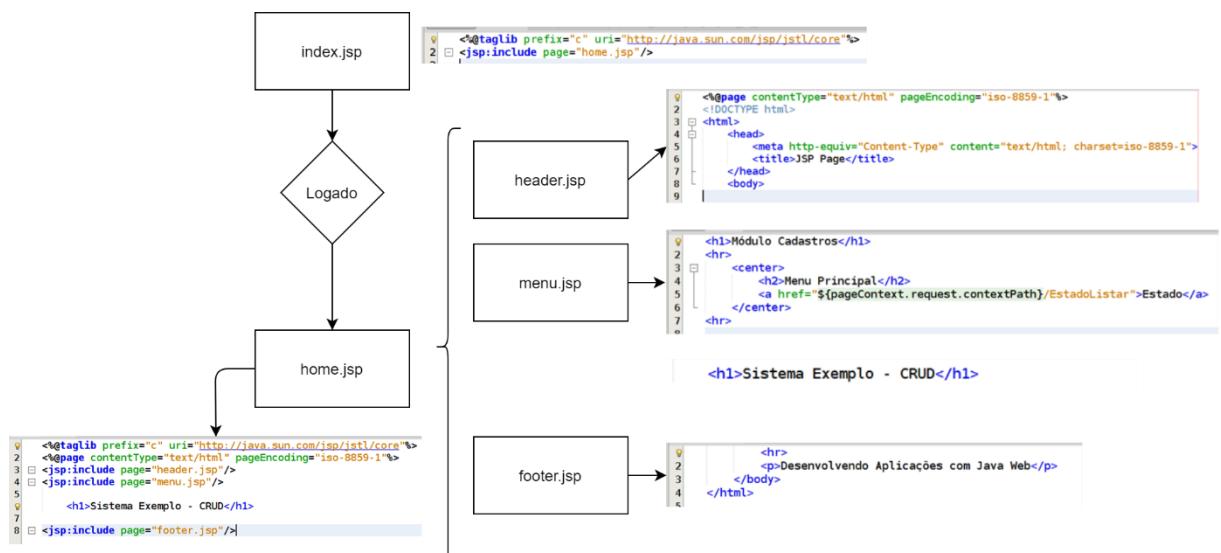
```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4   <jsp:include page="header.jsp"/>
5   <jsp:include page="menu.jsp"/>
6
7     <h1>Sistema Exemplo - CRUD</h1>
8
9   <jsp:include page="footer.jsp"/>
```

Fonte: O autor.

Na Figura 43 temos uma visão geral do que fizemos com nossa interface até agora e demonstra a composição da página home utilizando cada uma das JSPs desenvolvidas e que será nosso padrão para todo o sistema.

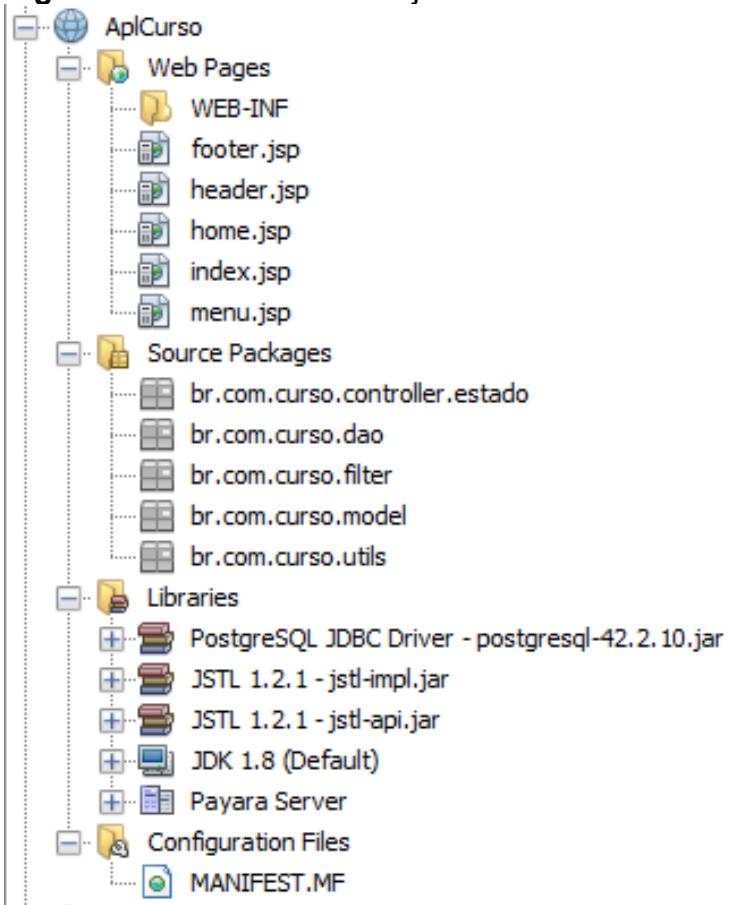
Figura 43 – Estrutura Básica da Interface (View)



Fonte: O autor.

E nosso projeto está no momento com essa estrutura, como pode-se verificar na Figura 44, confira o seu projeto.

Figura 44 – Estrutura do Projeto do curso.



Fonte: O autor

E se novamente quiser executar o projeto ele estará assim (Figura 45).

Figura 45 – Tela do Sistema

A tela do sistema mostra o seguinte layout:

- Cabeçalho com o link "Estado Cidade".
- Título "Módulo Cadastros".
- Menu Principal com opções "Estado" e "Cidade".
- Section "Sistema Exemplo - CRUD".
- Informação "Desenvolvendo Aplicações com Java Web".

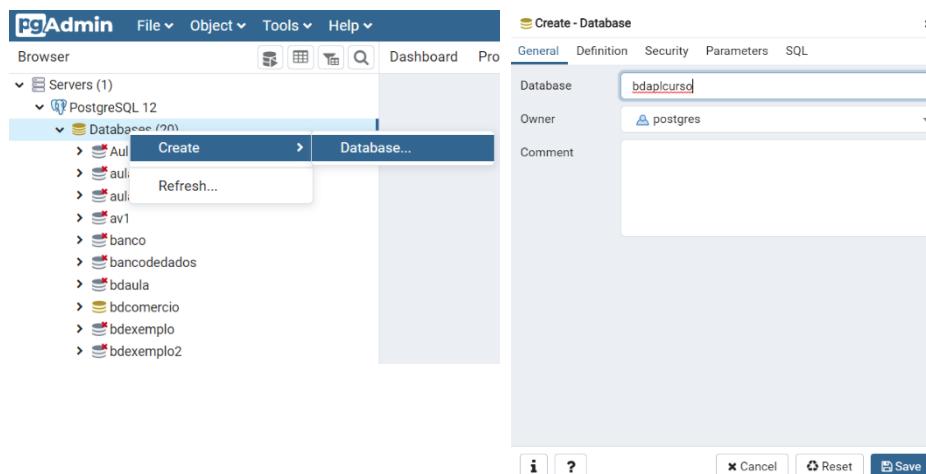
Fonte: O autor

4.5 CRIANDO O BANCO DE DADOS

Nesta etapa vamos criar nosso banco de dados para armazenar as informações manipuladas pela nossa aplicação web. Para isso você deve abrir o pgAdmin que é o software gerenciador do banco de dados PostgreSQL.

Com o servidor logado (usuario: postgres e a senha definida na instalação do software em seu computador) devemos criar um banco de dados, para isso clicar com o botão direito sobre Databases, e escolher a opção “create → database” como demonstra-se na Figura 46.

Figura 46 – Criando o banco de dados – “bdaplcurs0”



Fonte: O autor

Informar o nome do banco a ser criado “**bdaplcurs0**” e determinar o proprietário do banco (owner) como **postgres** e clicar em **Save** para confirmar a criação do banco de dados.

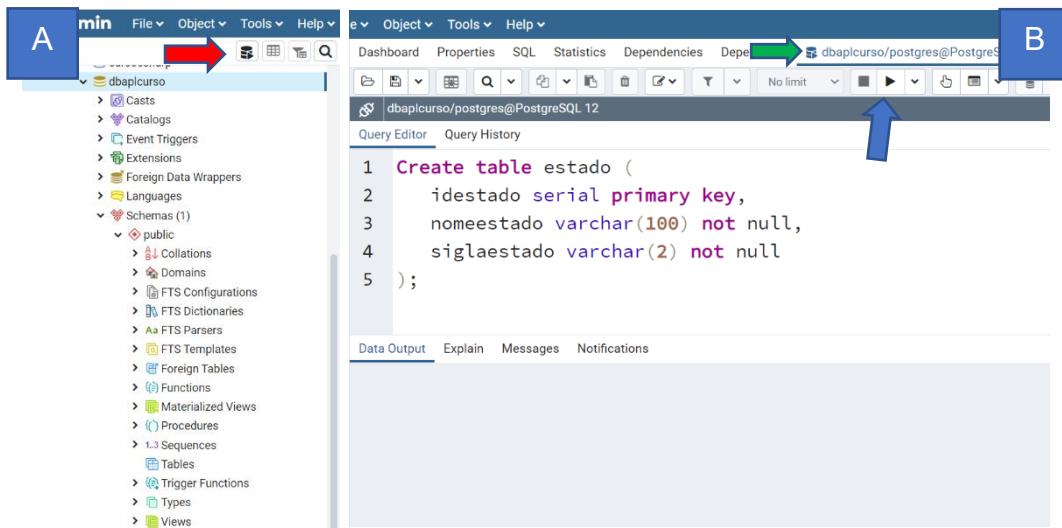
Na Figura 47(a) pode-se observar o banco de dados criado, se for necessário não se esqueça de dar um refresh sobre o servidor.

Também pode-se observar indicado com uma seta vermelha o botão que devemos clicar para criarmos a primeira tabela de nosso banco de dados. Clicando sobre o botão indicado será aberto uma janela para digitação de nossos comandos SQL.

Verifique como indicado na seta verde se você está realmente conectado ao banco de dados correto antes de digitar e executar o comando indicado.

Uma vez que tenha certeza digite o comando SQL e clique no botão indicado com a seta azul para executá-lo e criar a tabela **estado**.

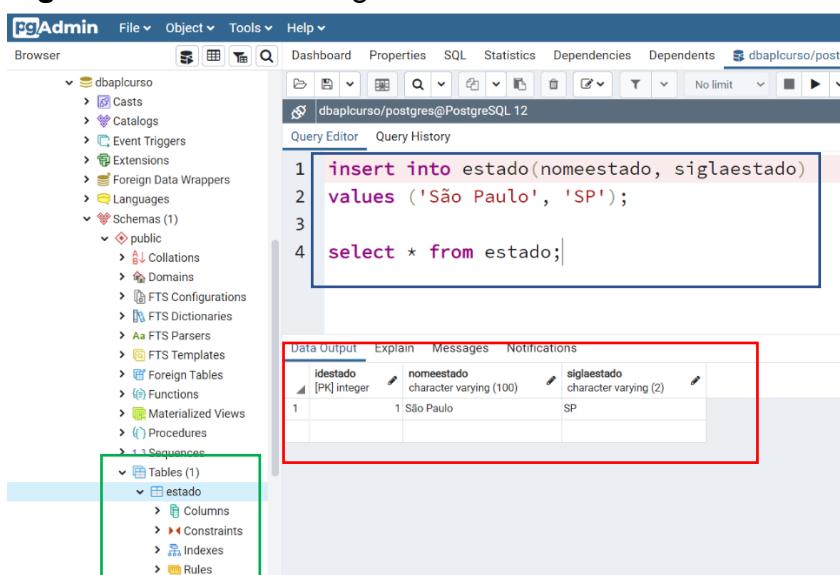
Figura 47 – Criando a tabela estado.



Fonte: O autor

Na Figura 48 podemos visualizar a tabela Estado criada em nosso banco de dados e aproveite para inserir um registro na tabela lembrando que nossa primary key é serial assim não devemos informar idestado para esse registro pois o postgresql se encarregará da tarefa de gerar as chaves primarias da tabela. Digite o comando como indicado na tabela e execute-o e depois dê um Select para confirmar.

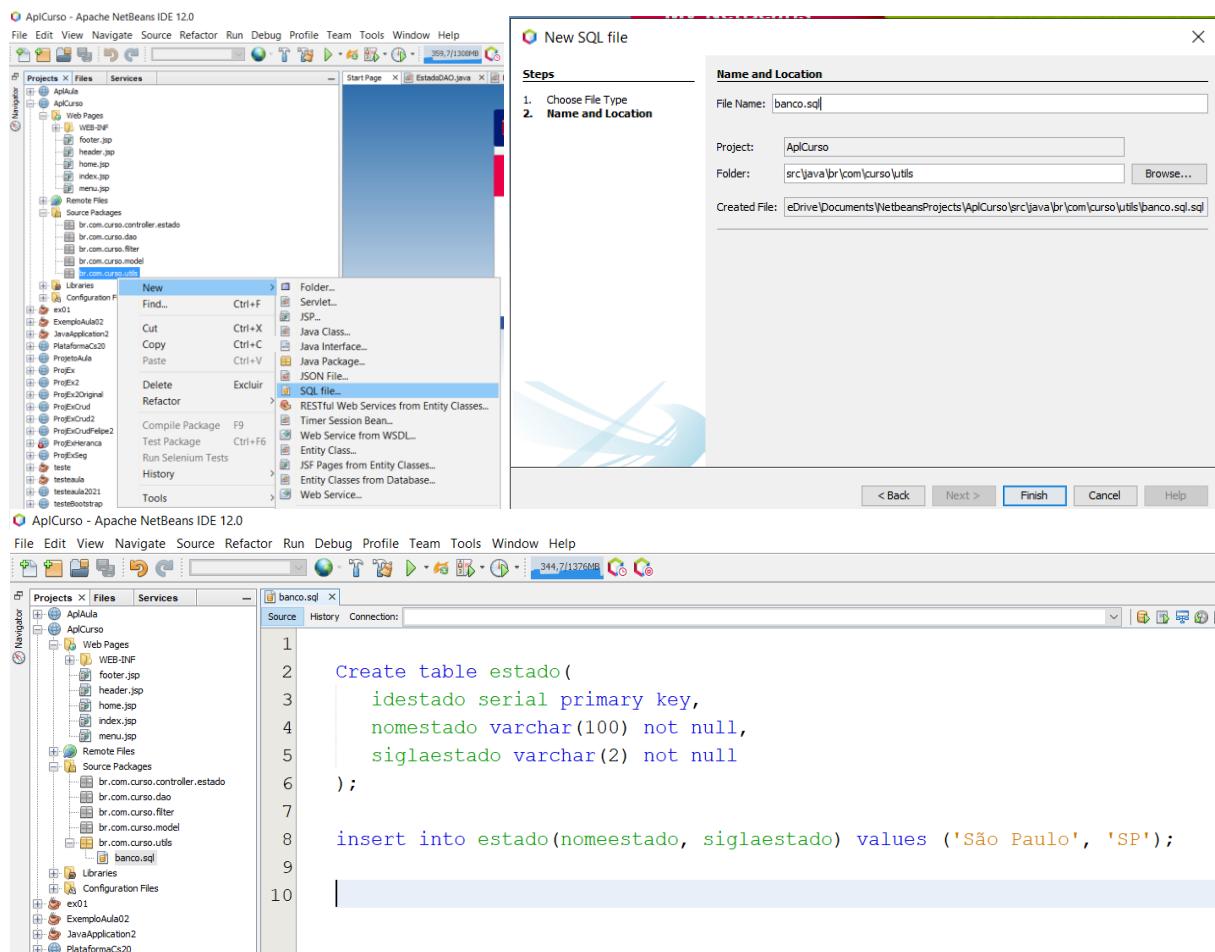
Figura 48 – Incluindo registro na tabela estado.



Fonte: O autor

Após criar a tabela não feche a janela, copie o código SQL executado e vamos guarda-lo dentro de nosso projeto java, neste modo vá ao Netbeans em Source Packages (Pacotes de Código fonte) no pacote br.com.curso.utils e crie um arquivo SQL (Figura 49).

Figura 49 – Criando arquivo banco.sql



Fonte: O autor

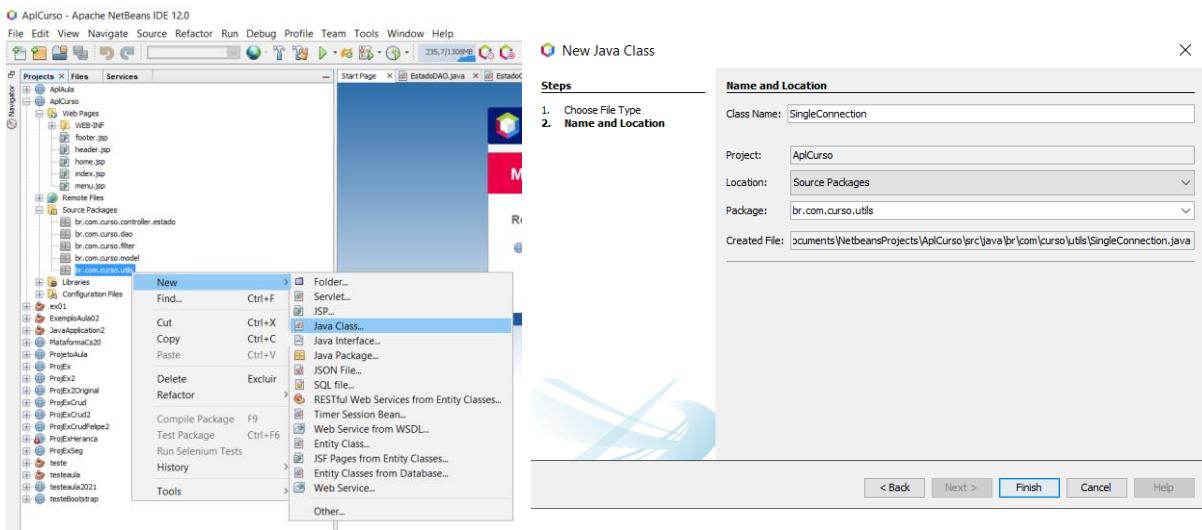
4.6 CRIANDO A CONEXÃO AO BANCO DE DADOS

Nesta etapa vamos preparar nosso projeto para conectar ao banco de dados, utilizaremos para isso uma classe Singleton que realizará a conexão ao banco de dados quando o sistema iniciar no servidor web e depois dessa conexão inicial apenas fornecerá a instância desse objeto a cada requisição recebida, encerrando a conexão quando o sistema for finalizado.

Para controlar esse processo também utilizaremos uma classe que chamaremos de Filter, responsável por cumprir um papel de funil centralizando todas as requisições de usuário antes de chegarem as nossas servlets na camada controller.

Assim vamos inicialmente criar a classe responsável pela conexão com o banco de dados, para isso clique com o botão direito sobre a camda Utils de nosso sistema e escolha a opção New → Java Class como indicado na Figura 50.

Figura 50 – Criando classe de conexão da aplicação



Fonte: O autor

Uma vez criada a classe vamos fazer sua codificação de acordo com a Figura 50. Pode-se verificar na linha de programação 9 que nossa string de conexão possui agora um parâmetro de auto reconexão ao SGBD o que antes não utilizávamos no padrão connection factory.

Também a partir de agora vamos trabalhar com transações assim ao executar algum comando no SGBD devemos fazer o COMMIT confirmindo a transação ou um ROLLBACK se quisermos cancelar a mesma. Para isso na linha 31 de código na Figura 50 pode-se visualizar essa configuração para que o SGBD não faça isso automaticamente.

Feita essas explicações você pode fazer a programação de sua classe de conexão no seu projeto (Figura 51).

Figura 51 – Programando a classe de conexão

```

1 package br.com.curso.utils;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5
6 public class SingleConnection {
7
8     private static Connection conexao = null;
9     private static String servidor = "jdbc:postgresql://localhost:5432/bdapcurso?autoReconnect=true";
10    private static String usuario = "postgres";
11    private static String senha = "postdba";
12
13    static {
14        try {
15            conectar();
16        } catch (Exception ex) {
17            System.out.println("Erro ao conectar ao banco de dados");
18            ex.printStackTrace();
19        }
20    }
21
22    public SingleConnection() throws Exception{
23        conectar();
24    }
25
26    public static void conectar() throws Exception {
27        try {
28            if (conexao == null){
29                Class.forName("org.postgresql.Driver");
30                conexao = DriverManager.getConnection(servidor, usuario, senha);
31                conexao.setAutoCommit(false);
32            }
33        } catch (Exception ex) {
34            throw new Exception(ex.getMessage());
35        }
36    }
37
38    public static Connection getConnection(){
39        return conexao;
40    }
41}

```

Fonte: O autor

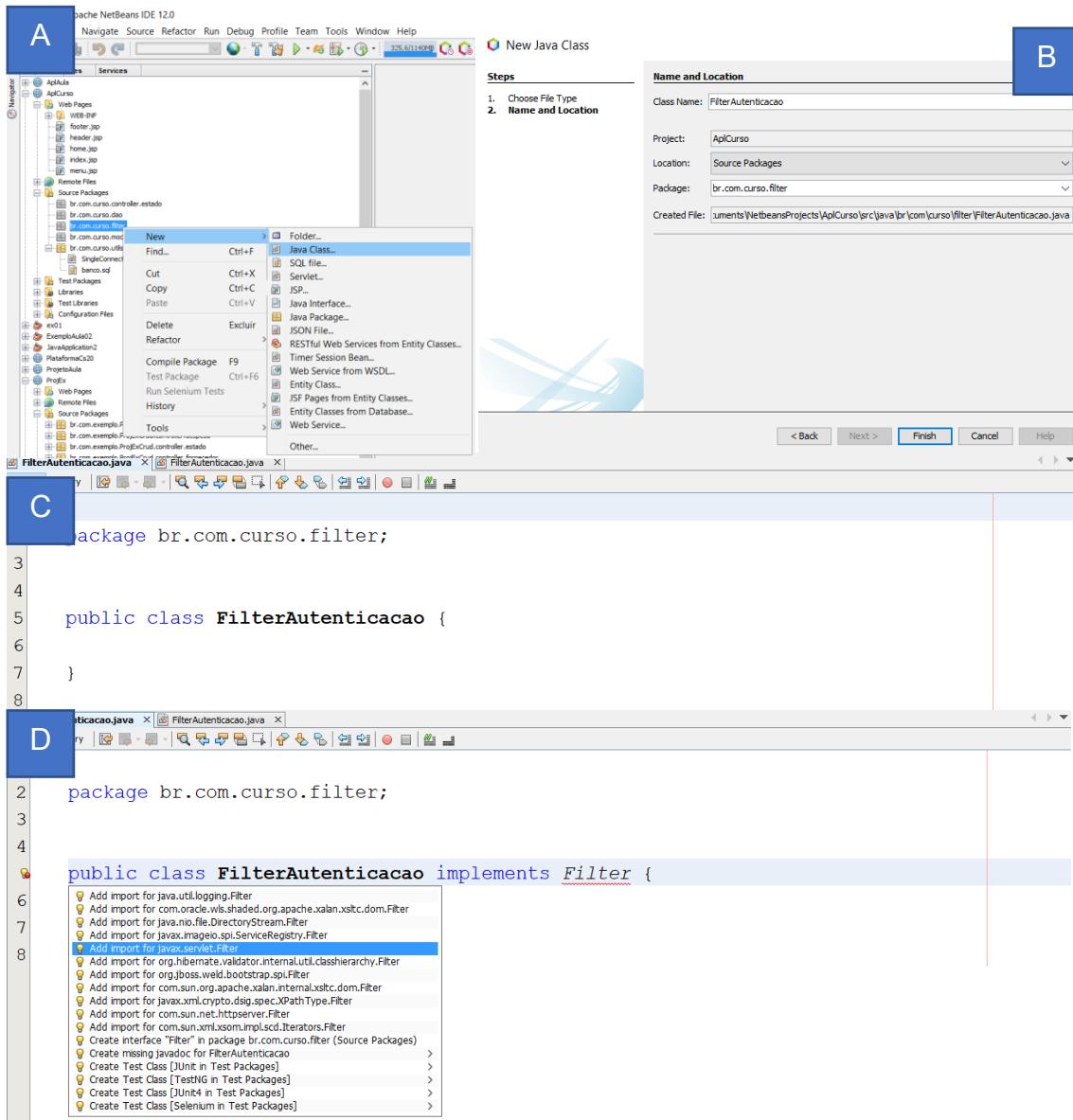
Agora vamos implementar o Filter de nossa aplicação, para isso clique com o botão direito no pacote **br.com.curso.filter** e escolha **New -> Java Class** (Figura 52a) e de o nome de **FilterAutenticacao** (Figura 52b) para a classe.

Na Figura 52c pode-se ver a classe criada.

Essa classe deve implementar a interface Filter do pacote javax.servlet que deve ser importado assim que declaramos a implementação da interface pela classe criada. Você pode observar isso na Figura 52(d) onde temos o comando *implements Filter* a frente da declaração da classe, assim como a escolha de importação do filtro no pacote referenciado anteriormente.

Então faça como demostrado na Figura 52 em seu projeto.

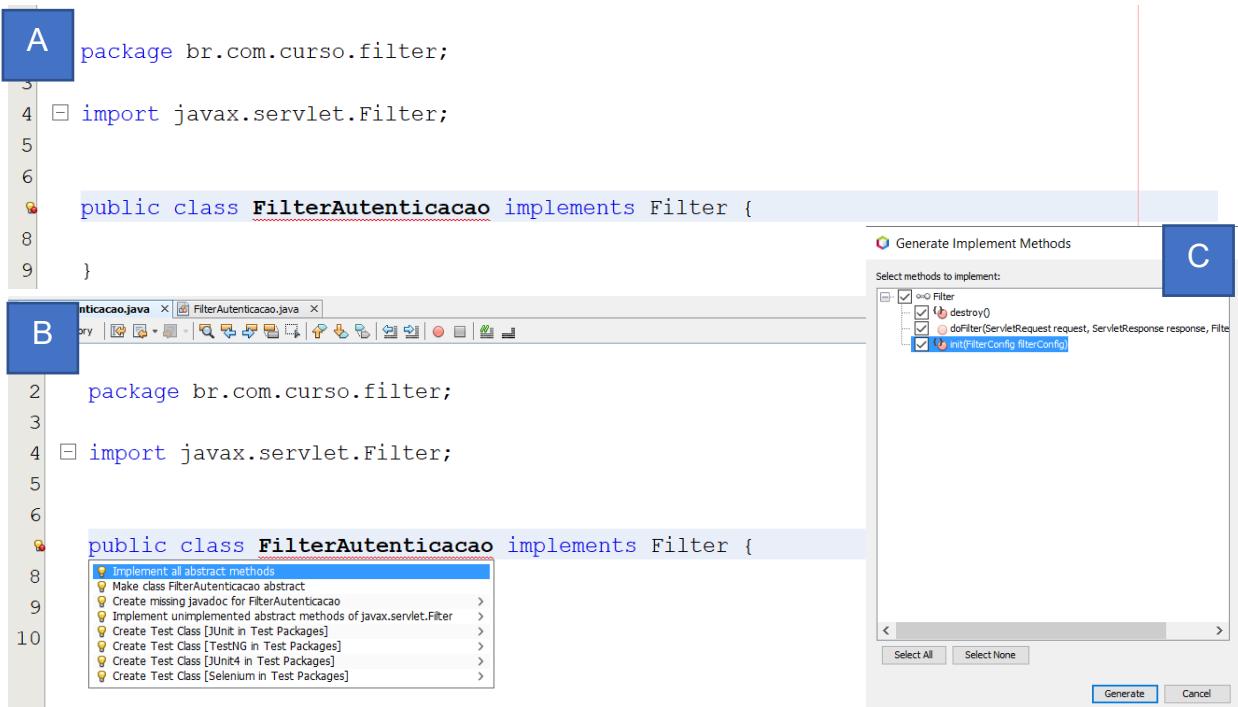
Figura 52 – Criando a classe Filter



Fonte: O autor

Neste momento sua classe está acusando erro como demonstrado na Figura 53a pois como estamos implementando a interface **Filter** de **javax.servelet** precisamos implementar seus métodos. Para isso clique na dica do Netbeans e escolha a opção “*implements all abstracts methods*” como pode ser visualizado na Figura 53b e na janela que abrir escolha todos as opções como demonstrado na Figura 53c

Figura 53 – Criando a classe Filter



Fonte: O autor

Na Figura 54 pode-se observar o código gerado.

Figura 54 – Classe FilterAutenticacao

```

package br.com.curso.filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class FilterAutenticacao implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        Filter.super.init(filterConfig); //To change body of generated methods, choose Tools | Temp
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generate
    }

    @Override
    public void destroy() {
        Filter.super.destroy(); //To change body of generated methods, choose Tools | Templates.
    }
}

```

Fonte: O autor

Os métodos da classe filter são o init, doFilter e destroy e possuem funções que explicadas abaixo.

- **Init** → as instruções implementadas aqui são executadas quando a aplicação é iniciada no servidor, por exemplo, abrir a conexão com o banco de dados.
- **doFilter** → instruções implementadas aqui são executadas a cada requisição que o servidor recebe e após são encaminhadas para a camada controller (servlets)
- **destroy** → as instruções aqui implementadas são executadas quando a aplicação é encerrada no servidor.

Agora precisamos implementar nossa classe FilterAutenticacao, inicialmente vamos determinar a **annotation @WebFilter** (destacado na Figura 54 com um quadro vermelho) e faça a importação do pacote **javax.servlet.annotation.WebFilter** e criar o atributo que irá armazenar nossa conexão (destacado em um quadro verde na Figura 55) e importe o pacote **java.sql.Connection**.

Figura 55 – Implementando a Classe Filter (1)

```

3  import java.io.IOException;
4  import java.sql.Connection;
5  import javax.servlet.Filter;
6  import javax.servlet.FilterChain;
7  import javax.servlet.FilterConfig;
8  import javax.servlet.ServletException;
9  import javax.servlet.ServletRequest;
10 import javax.servlet.ServletResponse;
11 import javax.servlet.annotation.WebFilter;
12
13 @WebFilter(urlPatterns={"/"})
14 public class FilterAutenticacao implements Filter {
15
16     private static Connection conexao;
17

```

Fonte: O autor.

A annotation **@WebFilter** serve para definir que nossa classe de filtro irá interceptar todas as requisições no método **doFilter** que se destinarem a qualquer recurso de nossa aplicação a partir da raiz do projeto.

Agora vamos implementar o método **init** de nossa classe **filter**, exclua a linha indicada com um quadro vermelho conforme pode ser visualizado na Figura 56a e insira o código requisitando uma conexão ao banco através da classe

SingleConnection que desenvolvemos anteriormente como destacado na Figura 56b e importe a classe *br.com.curso.utils.SingleConnection* conforme demonstrado na Figura 56c.

Figura 56 – Implementando a Classe Filter – método init (2)

The screenshot shows three stages (A, B, C) of implementing the `init` method in a Java Filter class:

- A:** The initial state where the generated code by Netbeans is shown. The line `Filter.super.init(filterConfig);` is highlighted with a red box.
- B:** The state after manually replacing the generated code with the correct implementation: `conexao = SingleConnection.getConnection();`
- C:** The state after adding the import statement for `SingleConnection`. A tooltip from Netbeans suggests creating the class, which is highlighted with a green box.

```

18     @Override
19     public void init(FilterConfig filterConfig) throws ServletException {
20         Filter.super.init(filterConfig); //To change body of generated methods, choose Tools->Templates->File->Edit
21     }
22
23
24     @Override
25     public void init(FilterConfig filterConfig) throws ServletException {
26         conexao = SingleConnection.getConnection();
27     }
28
29
30     @Override
31     public void init(FilterConfig filterConfig) throws ServletException {
32         conexao = SingleConnection.getConnection();
33     }
  
```

Fonte: O autor.

Agora devemos implementar o método `doFilter` (Figura 57), para isso apague o código gerado pelo Netbeans (linha contornada em vermelho) e insira em seu lugar o código contornado em verde. Neste bloco de código em verde observamos a linha 28 onde temos o comando `chain.doFilter`, esse comando faz o envio da requisição interceptada para a camada controller.

Figura 57 – Implementando a Classe Filter – método doFilter (2)

The screenshot shows the implementation of the `doFilter` method in a Java Filter class:

- The first part of the code (lines 24-28) is the generated code by Netbeans, highlighted with a red box.
- The second part (lines 24-33) is the manual implementation where the generated code is replaced by a try-catch block that prints errors to the console.

```

24     @Override
25     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
26         throws IOException, ServletException {
27             throw new UnsupportedOperationException("Not supported yet.");
28         }
29
30
31     @Override
32     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
33         throws IOException, ServletException {
34         try{
35             chain.doFilter(request, response);
36         } catch(Exception e){
37             System.out.println("Erro: "+e.getMessage());
38             e.printStackTrace();
39         }
40     }
  
```

Fonte: O autor.

Finalmente vamos implementar o método `destroy` que terá a função de fechar a conexão como banco de dados quando o sistema for encerrado, faça como

demonstrado na Figura 58 apagando a linha em vermelho e inserindo em seu lugar o código identificado em verde.

Figura 58 – Implementando a Classe Filter – método doFilter (2)

```
35     @Override  
36     public void destroy() {  
37         Filter.super.destroy(); //To change body of generated methods, choose Tools | Templates.  
38     }  
39  
40     @Override  
41     public void destroy() {  
42         try {  
43             conexao.close();  
44         } catch (SQLException ex) {  
45             System.out.println("Erro :" + ex.getMessage());  
46             ex.printStackTrace();  
47         }  
48     }  
49 }
```

Fonte: O autor.

CAPÍTULO 5 – IMPLEMENTANDO UM CRUD SIMPLES (ESTADO)

Neste capítulo vamos implementar o crud de estado, onde termos classes nas camadas model, dao e controller além de desenvolvermos interface com o usuário (view). Iremos dividir o processo de desenvolvimento deste cadastro em cada uma de suas operações Listar, Incluir, Alterar e Excluir, começando agora com o recurso de Listar.

Essa primeira parte (listar) demandará um pouco mais de trabalho pois criaremos todas as classes necessárias ficando apenas algumas classes específicas na controller para as outras operações.

Então vamos iniciar nosso desenvolvimento.

5.1 CRIANDO O RECURSO DE LISTAR ESTADO

Na camada Model estão as classes java que representam os modelos de dados em nosso sistema. No projeto deste tutorial estamos desenvolvendo um cadastro de estados.

Para isto definimos em nosso banco de dados que um estado irá possuir um idEstado, o nomeEstado e a siglaEstado, neste modo iremos criar nossa classe Estado.java no pacote ***br.com.curso.model*** em nosso projeto.

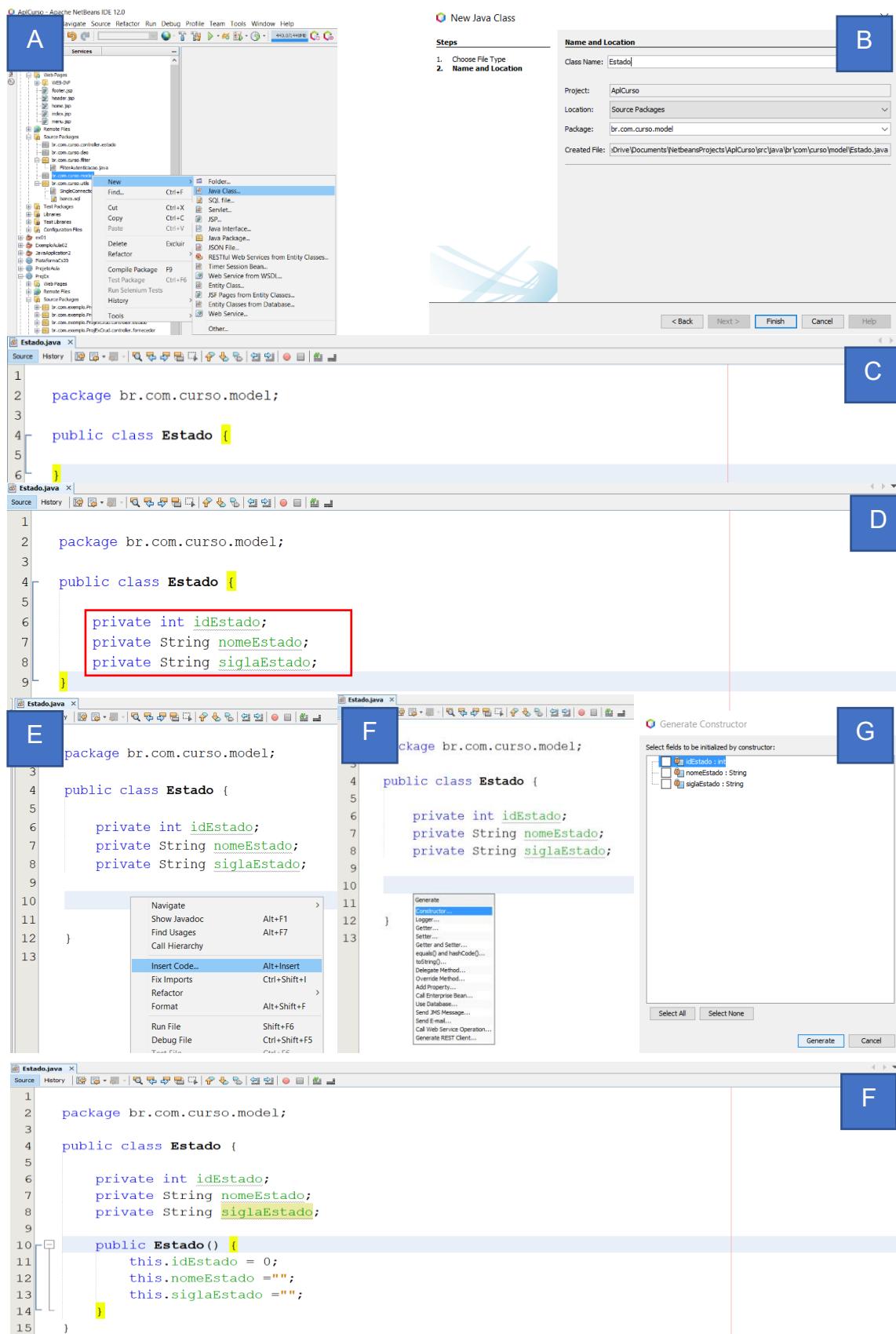
A seguir vamos clicar com o botão direito sobre o pacote model e escolher a opção New(Novo) → Java Class (Classe Java) (Figura 59a) que abrirá a janela para criação da classe onde daremos o nome de “Estado” (Figura 59b) e a seguir na Figura 59c você pode observar o código gerado pelo Netbeans.

Você deve criar os atributos idEstado, nomeEstado e siglaEstado do tipo private como demonstrado na Figura 59d.

Crie então um construtor sem parâmetros para nossa classe Estado, para isso clique com o botão direito em qualquer espaço vazio dentro da classe e escolha as opções “Insert Code → Constructor” como demonstrado nas Figuras 59e e 59f.

Na Figura 58g não marque nenhum atributo e clique no botão “Generate” e no código gerado pelo Netbeans complemente a programação com os códigos descartados com um quadro verde como demonstrado na Figura 59h.

Figura 59 – Criando a Classe Estado na camada Model.

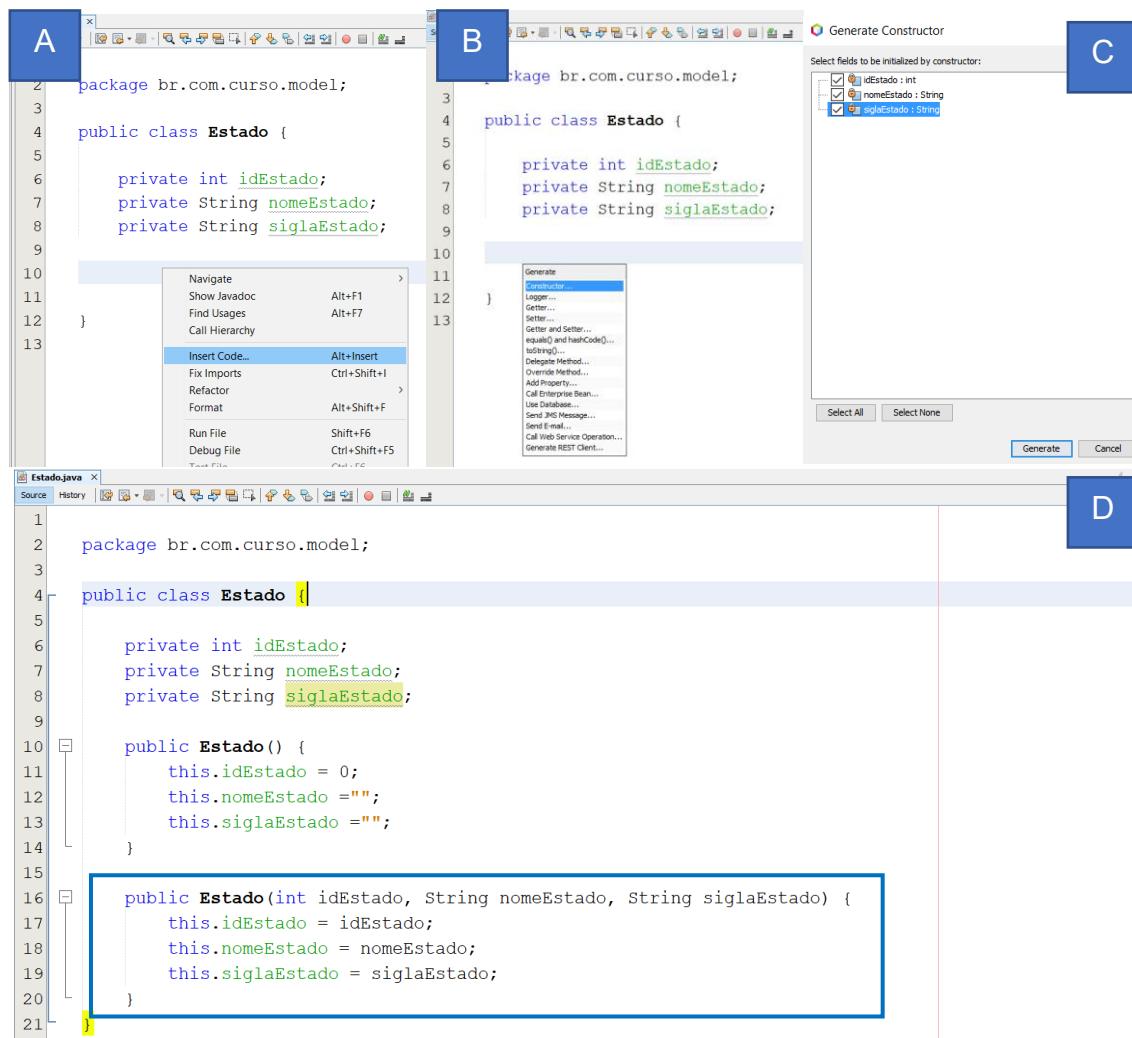


Fonte: O autor.

Agora vamos implementar um construtor que gera um objeto com informações, então esse construtor deverá receber os parâmetros de acordo com os atributos da classe. Deste modo vamos repetir os passos como anteriormente, clique com o botão direito em qualquer espaço vazio dentro da classe e escolha as opções “Insert Code → Constructor” como demonstrado nas Figuras 60a e 60b.

Na Figura 60c marque todos os atributos e clique no botão “Generate” e na Figura 60d pode-se visualizar o código gerado destacado em um quadro verde e todo o código feito até o momento para sua conferência.

Figura 60 – Criando a Construtor da Classe Estado na model.

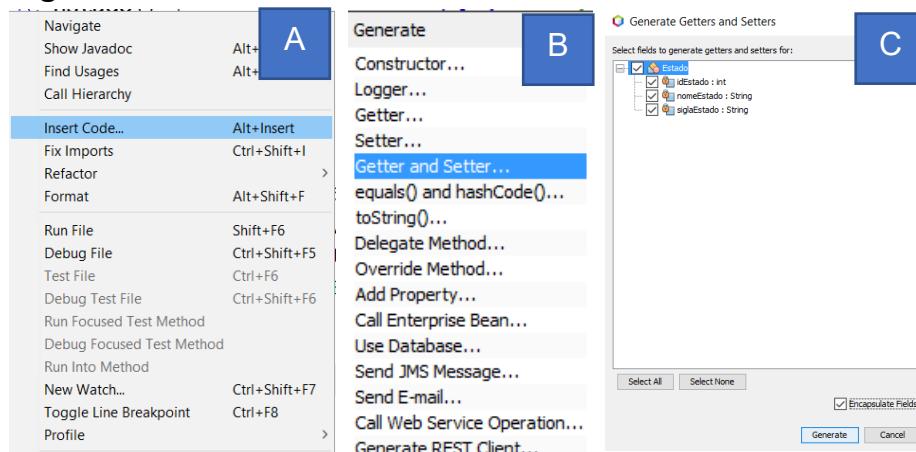


Fonte: O autor.

Agora em nossa model de Estado resta gerar os métodos Getter e Setter para isso clique com o botão direito novamente e escolha a opção “Insert Code → Getter

and Setter” como nas Figuras 61a e 61b. Na Figura 61c selecione todos os atributos e mande gerar os métodos Getter e Setter.

Figura 61 – Criando os métodos Getter e Setter



Fonte: O autor.

O código de nossa model ficou como demonstrado na Figura 62 a primeira parte do código.

Figura 62 – Código Completo da Model de Estado (Parte 1)

```

1 package br.com.curso.model;
2
3 public class Estado {
4
5     private int idEstado;
6     private String nomeEstado;
7     private String siglaEstado;
8
9     public Estado() {
10         this.idEstado = 0;
11         this.nomeEstado = "";
12         this.siglaEstado = "";
13     }
14
15     public Estado(int idEstado, String nomeEstado, String siglaEstado) {
16         this.idEstado = idEstado;
17         this.nomeEstado = nomeEstado;
18         this.siglaEstado = siglaEstado;
19     }
20
21     public int getIdEstado() {
22         return idEstado;
23     }
24 }
```

Fonte: O autor.

O código continua na Figura 63.

Figura 63 – Código Completo da Model de Estado (Parte 2)

```

25  public void setIdEstado(int idEstado) {
26      this.idEstado = idEstado;
27  }
28
29  public String getNomeEstado() {
30      return nomeEstado;
31  }
32
33  public void setNomeEstado(String nomeEstado) {
34      this.nomeEstado = nomeEstado;
35  }
36
37  public String getSiglaEstado() {
38      return siglaEstado;
39  }
40
41  public void setSiglaEstado(String siglaEstado) {
42      this.siglaEstado = siglaEstado;
43  }
44 }
```

Fonte: O autor.

Agora vamos criar nossa camada DAO. A camada DAO é responsável por gerenciar a comunicação entre nossa aplicação e o banco de dados de forma a realizar a conversão entre os paradigmas Orientado a Objetos e Relacional.

As classes Java que estão na camada DAO irão possuir vários métodos como cadastrar, incluir, alterar, excluir, carregar e listar e cada classe que criarmos na model deveremos ter uma equivalente na DAO.

Deste modo quando criamos a classe Estado na model deveremos criar uma classe equivalente na DAO e assim ocorrerá com todas as classes que criarmos em nosso sistema.

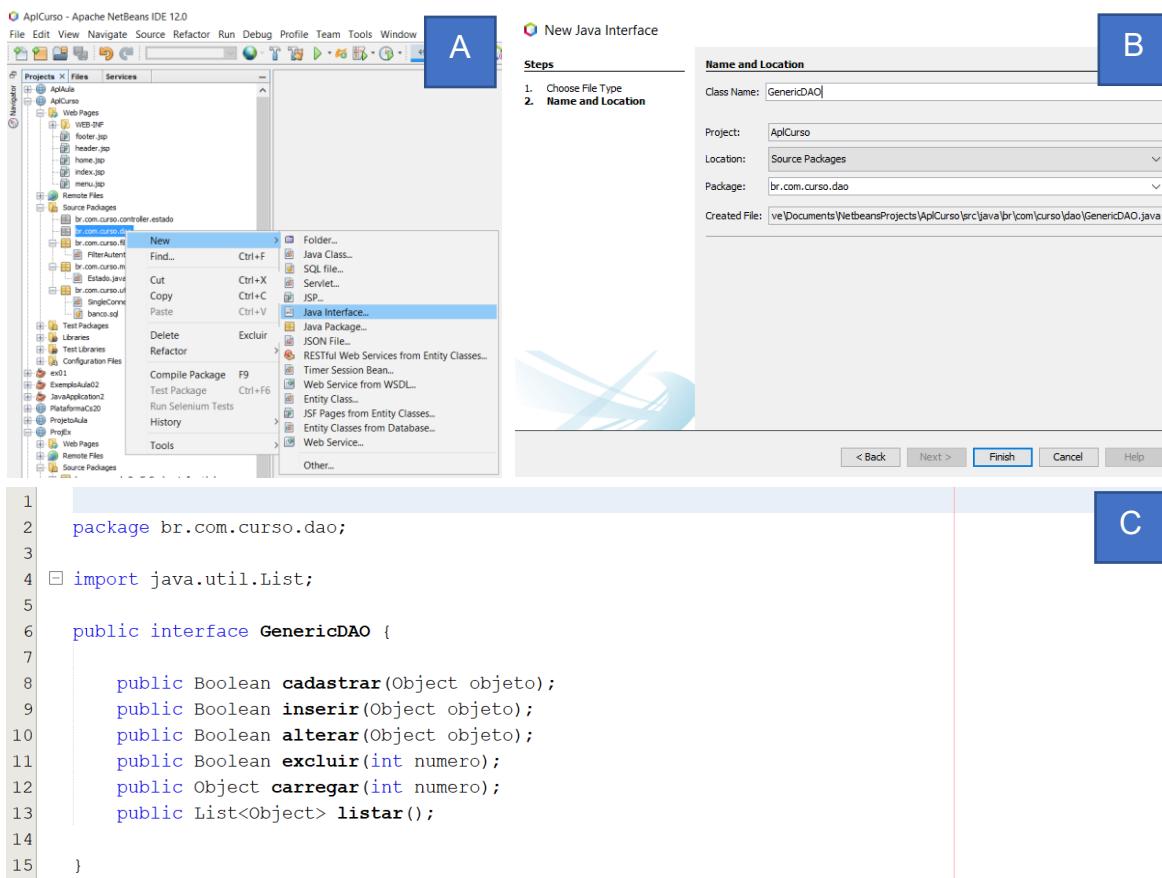
Desta forma é necessário padronizar o acesso a essas classes da camada DAO evitando nomenclatura diferentes como inserir ou gravar, excluir ou apagar, para isto iremos criar uma Interface de forma a padronizar todas as classes da camada DAO que criarmos.

Para criarmos uma Interface Java em nosso projeto iremos clicar com o botão direito sobre o pacote **br.com.curso.dao** e escolher a opção **New(Novo) → Java Interface(Interface Java)**, como pode-se observar na Figura 64a.

Se não encontrar a opção no menu auxiliar escolha a opção outros o que abrirá a janela abaixo onde escolheremos a categoria Java e o tipo de arquivo Interface.

Na janela demonstrada na Figura 64b informe o nome da interface como **GenericDAO** e clique em “Finish”.

Figura 64 – Implementando a GenericDAO



Fonte: O autor.

Não se esqueça de fazer a importação da **java.util.List** conforme observa-se na linha 4, digite a declaração das assinaturas dos métodos das linhas 8 a 13 na Figura 64c.

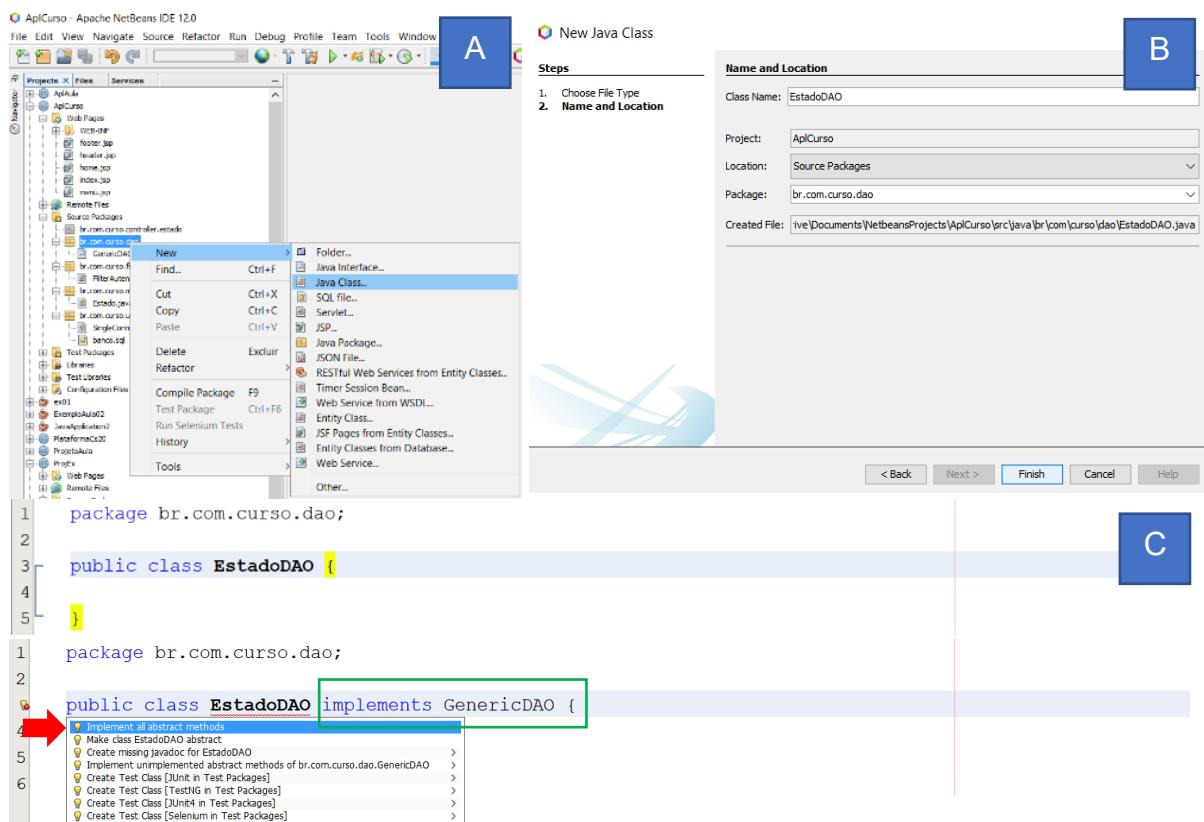
Uma Interface Java não permite a implementação dos métodos apenas a declaração de suas assinaturas que serão implementadas em uma Classe Java. Essas assinaturas de métodos são chamadas de métodos abstratos.

Agora vamos implementar a classe EstadoDAO que implementará os recursos necessários para a persistência dos dados no PostgreSQL.

Para criar a classe devemos clicar com o botão direito sobre o pacote **br.com.curso.dao** e escolher a opção **New(Novo)→ Java Class (Classe Java)** como na Figura 65a e abrindo a tela de criação da classe informe o nome como **EstadoDAO** confirmando sua criação como demonstrado na Figura 65b.

Na Figura 65c você deve na frente da declaração da classe colocar o código “**implements GenericDAO**” como demonstrado em destaque em verde, neste momento o Netbeans irá dar uma dica para você implementar os métodos todos os abstratos, faça isso como demonstrado na Figura 65c com o destaque da seta vermelha.

Figura 65 – Criando a Classe EstadoDAO

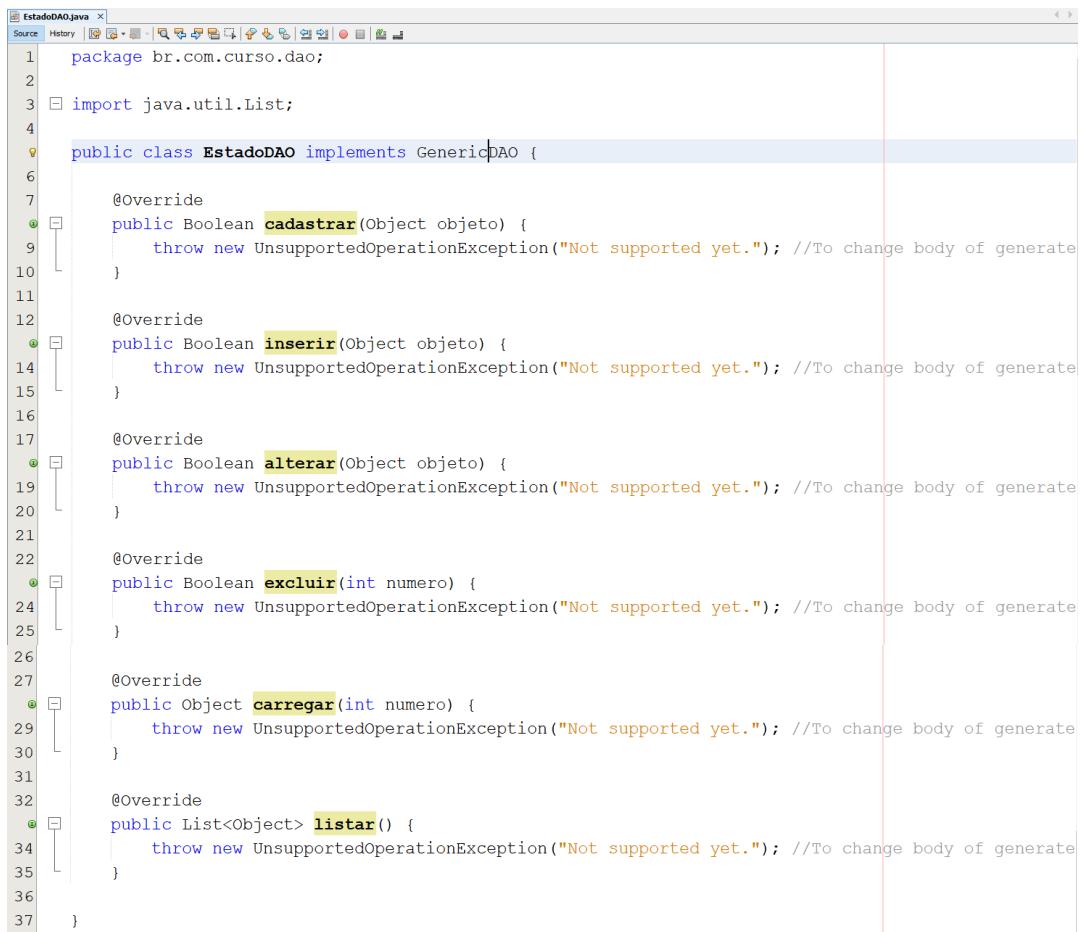


Fonte: O autor.

Podemos observar na Figura 66 o código da classe EstadoDAO com os métodos abstrados definidos na interface GenericDAO implementados como gerados automaticamente pelo Netbeans.

Cada método implementado está com uma declaração de exceção que acusa um erro caso seja acionado dessa forma (Por. Ex. linha 9), assim quando formos implementar de fato cada um desses métodos deveremos sempre apagar essas linhas de exceção.

Figura 66 – Classe EstadoDAO com os métodos abstratos implementados



```

1 package br.com.curso.dao;
2
3 import java.util.List;
4
5 public class EstadoDAO implements GenericDAO {
6
7     @Override
8     public Boolean cadastrar(Object objeto) {
9         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
10    }
11
12     @Override
13     public Boolean inserir(Object objeto) {
14         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
15    }
16
17     @Override
18     public Boolean alterar(Object objeto) {
19         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
20    }
21
22     @Override
23     public Boolean excluir(int numero) {
24         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
25    }
26
27     @Override
28     public Object carregar(int numero) {
29         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
30    }
31
32     @Override
33     public List<Object> listar() {
34         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
35    }
36
37 }

```

Fonte: O autor.

Vamos implementar um método construtor para nossa classe que não foi gerado automaticamente e o atributo que armazenará nossa conexão ao banco de dados e faça as importações necessárias (Figura 67).

Figura 67 – Implementando o método Construtor da EstadoDAO



```

1 package br.com.curso.dao;
2
3 import br.com.curso.utils.SingleConnection; ←
4 import java.sql.Connection; ←
5 import java.util.List;
6
7 public class EstadoDAO implements GenericDAO {
8
9     private Connection conexao;
10
11     public EstadoDAO() throws Exception{
12         conexao = SingleConnection.getConnection();
13     }
14
15     @Override
16     public Boolean cadastrar(Object objeto) {
17         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, select a method node in the tool bar or right-click the
18    }

```

Fonte: O autor.

Agora vamos implementar em nossa camada DAO o método Listar(), localize-o na sua DAO de Estado e apague a linha de comando indicada na Figura 68.

Figura 68 – Método Listar()

```
40     @Override
41     public List<Object> listar() {
42         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated...
43     }
```

Fonte: O autor.

E implemente o método como demonstrado na Figura 69.

Figura 69 – Implementando o método Listar

```
45     @Override
46     public List<Object> listar() {
47         List<Object> resultado = new ArrayList<>();
48         PreparedStatement stmt = null;
49         ResultSet rs = null;
50         String sql = "Select * from estado order by idEstado";
51         try {
52             stmt = conexao.prepareStatement(sql);
53             rs=stmt.executeQuery();
54             while (rs.next()) {
55                 Estado oEstado = new Estado();
56                 oEstado.setIdEstado(rs.getInt("idEstado"));
57                 oEstado.setNomeEstado(rs.getString("nomeestado"));
58                 oEstado.setSiglaEstado(rs.getString("siglaestado"));
59                 resultado.add(oEstado);
60             }
61         } catch (SQLException ex) {
62             System.out.println("Problemas ao listar Estado! Erro: "
63                         +ex.getMessage());
64         }
65         return resultado;
66     }
```

Fonte: O autor.

A digitação deste código no método listar() demanda a importação dos recursos das linhas 3, 6, 7, 8 e 9, que se não foram solicitadas pelo Netbeans durante a digitação faço-o agora (Figura 70).

Figura 70 – Fazendo importações de recursos do java.

```
3  import br.com.curso.model.Estado;
4  import br.com.curso.utils.SingleConnection;
5  import java.sql.Connection;
6  import java.sql.PreparedStatement;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import java.util.ArrayList;
10 import java.util.List;
```

Fonte: O autor.

O objetivo do método listar() é gerar uma lista de objetos do Tipo Estado (classe) com todos os dados selecionados a partir da tabela estado do banco de dados bdaplcurso.

Para isto na linha 47 da listagem é instanciada uma lista deste tipo chamada resultado, na linha seguinte, 48, declaramos um objeto do tipo Statement que permitira preparar o comando select para ser executado no banco de dados.

Na linha 49 declara-se o ResultSet chamado de “rs” que armazenará o retorno dos dados do banco de dados.

Na linha 50 monta-se a String com o comando SQL a ser executado, neste caso um select de todos os dados da tabela estado.

Na linha 52 fornece a conexão para o Statement assim como o comando a ser executado, como não existe nenhum parâmetro para ser substituído na linha 64 executa-se o comando select através do método executeQuery(), armazenando o resultado em “rs” (ResultSet).

Na linha 54 tem-se um while para percorrer todas as linhas do ResultSet e cada iteração do laço é gerado um objeto de Estado para cada linha da tabela no banco de dados (linhas de 55 a 58) e adicionado a lista de resultado (linha 59).

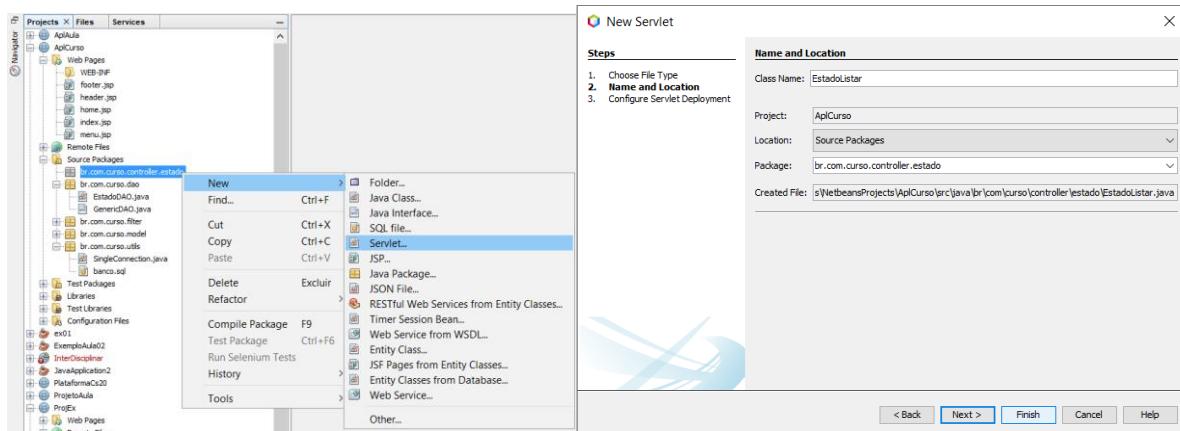
Se houver erros são capturados e tratados no catch e o resultado (lista de estados) será retornado na linha 66 para a camada controller que será implementada a seguir.

A camada controller é responsável pela comunicação entre o front end e o back end de nossa aplicação, ela implementa os métodos de comunicação do protocolo Http que são o Post e o Get.

Para isso devemos criar uma classe Java especial nesta camada chamada **Servlet**.

Para criar vamos clicar com o botão direito do mouse no pacote br.com.curso.controller.estado e escolher a opção New(Novo) → Servlet, abrindo a janela para a criação de nosso servlet EstadoListar, como observa-se na Figura 71, informe o nome da classe e clique no botão “Finish”.

Figura 71 – Criando Classe EstadoListar na camada controller.



Fonte: O autor.

O Netbeans deve ter gerado para você os códigos necessários para nossa servlet, bom para começarmos observemos que a classe em questão importa diversas APIs e principalmente aquelas que implementam o protocolo HTTP, como destacado na Figura 72.

Figura 72 – Java APIs - HttpServlet

```

8  import java.io.IOException;
9  import java.io.PrintWriter;
10 import javax.servlet.ServletException;
11 import javax.servlet.annotation.WebServlet;
12 import javax.servlet.http.HttpServlet;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
```

Fonte: O autor.

Na Figura 73 pode-se visualizar a linha de código 21 onde está sendo declarada nossa classe EstadoListar e que ela tem uma relação de herança com a classe HttpServlet definida pela clausula extends.

Figura 73 – Classe EstadoListar

```

20 @WebServlet(name = "EstadoListar", urlPatterns = {"/EstadoListar"})
21 public class EstadoListar extends HttpServlet {
```

Fonte: O autor.

Na linha anterior, 20, temos a annotation que configura a publicação de nosso Servlet para o servidor Web de forma que ele seja encontrado pela camada front-end de nossa aplicação.

Devemos observar que os Servlets implementam os métodos Get e Post do protocolo HTTP conforme o código abaixo. Tanto o doGet quanto o doPost redirecionam o processamento para o método processRequest como observa-se na Figura 74.

Figura 74 – Classe EstadoListar – doGet e doPost

```

58     @Override
59     protected void doGet(HttpServletRequest request, HttpServletResponse response)
60         throws ServletException, IOException {
61         processRequest(request, response);
62     }
63
64     /** Handles the HTTP <code>POST</code> method ...8 lines */
65     @Override
66     protected void doPost(HttpServletRequest request, HttpServletResponse response)
67         throws ServletException, IOException {
68         processRequest(request, response);
69     }
70
71
72
73
74
75
76

```

Fonte: O autor.

O método processRequest foi definido pelo netbeans como se encontra na Figura 75, apague o código selecionado na figura para que possamos implementá-lo.

Figura 75 – Classe EstadoListar – método processRequest

```

32     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33         throws ServletException, IOException {
34         response.setContentType("text/html;charset=UTF-8");
35         try (PrintWriter out = response.getWriter()) {
36             /* TODO output your page here. You may use following sample code. */
37             out.println("<!DOCTYPE html>");
38             out.println("<html>");
39             out.println("<head>");
40             out.println("<title>Servlet EstadoListar</title>");
41             out.println("</head>");
42             out.println("<body>");
43             out.println("<h1>Servlet EstadoListar at " + request.getContextPath() + "</h1>");
44             out.println("</body>");
45             out.println("</html>");
46         }
47

```

Fonte: O autor.

Agora implemente o código como demonstrado na Figura 76 e não se esqueça de fazer as importações da GenericDAO e da EstadoDAO em seu código como solicitado pelo Netbeans.

Figura 76 – Implementando a classe EstadoListar

```

34     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35         throws ServletException, IOException {
36     response.setContentType("text/html;charset=iso-8859-1");
37     try{
38         GenericDAO dao = new EstadoDAO();
39         request.setAttribute("estados", dao.listar());
40         request.getRequestDispatcher("/cadastros/estado/estado.jsp")
41             .forward(request, response);
42
43     } catch (Exception ex){
44         //ex.printStackTrace();
45         System.out.println("Problemas no Servlet ao Listar"
46             + " Estados! Erro: " + ex.getMessage());
47         ex.printStackTrace();
48     }
49 }
```

Fonte: O autor.

Na linha 36 configura-se o conjunto de caracteres “iso-8859-1” e na seguinte abre-se um try catch.

Na linha 38 instancia-se um objeto DAO (EstadoDAO) para obtermos acesso aos recursos necessários dessa camada. E na linha 39 prepara-se um atributo na memória do web container denominado “estados”, que receberá o resultado do método listar de EstadoDAO() em dao.listar().

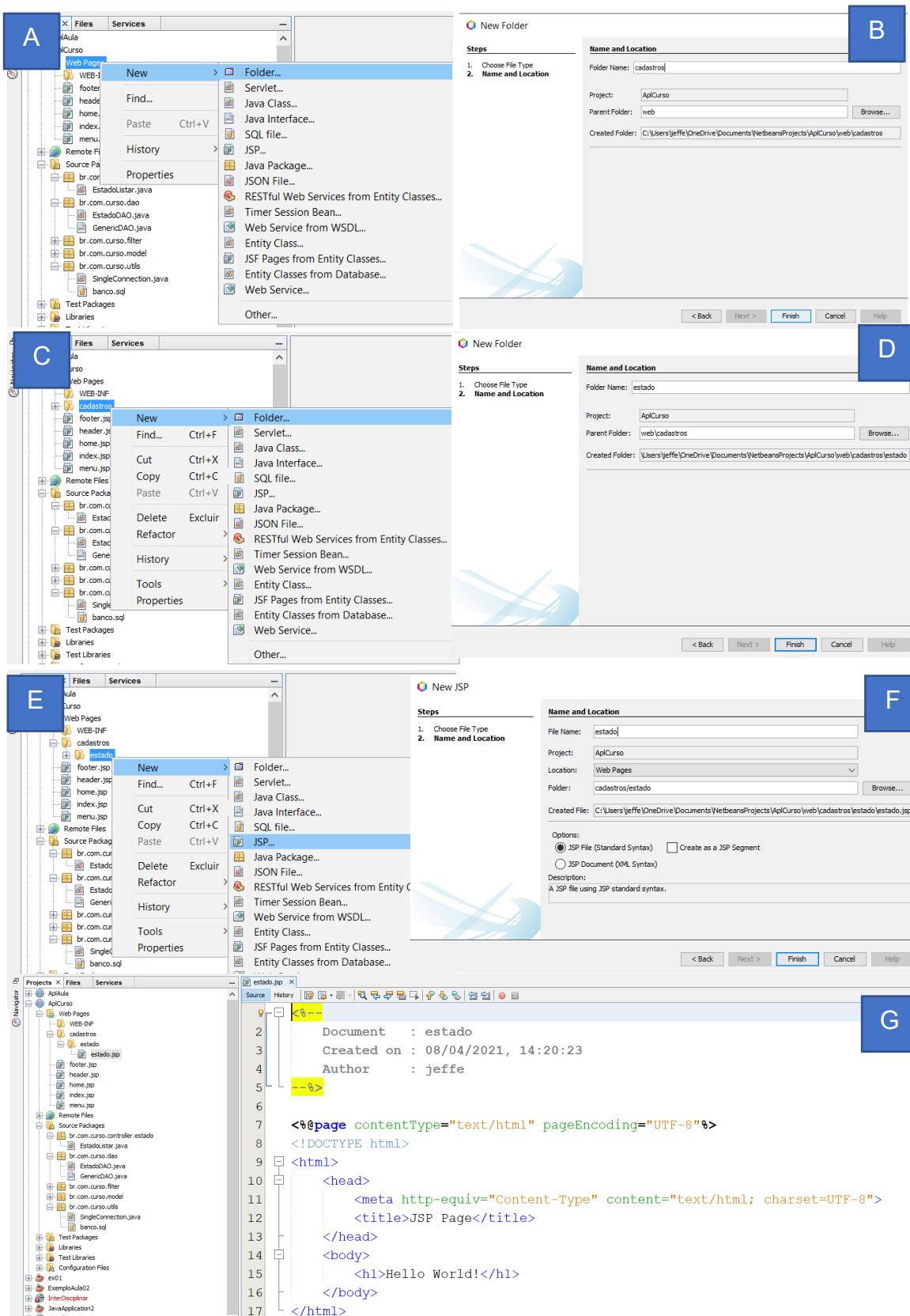
Assim “estados” conterá uma lista de estados proveniente do banco de dados e será enviada a camada view (Página Web) no arquivo estado.jsp que criaremos a seguir.

A camada view é o front end de nosso sistema, gera a interface com o usuário final de nossa aplicação. Para finalizarmos a implementação do listar de nosso Crud de Estados devemos criar alguns recursos em nosso projeto.

Primeiramente vamos criar uma nova pasta dentro da pasta “Web Pages” já existente o projeto, clicando com o botão direito do mouse sobre Web Page escolher a opção New->Folder e na tela que será aberta informe o nome da pasta como “cadastros” e confirme sua criação (Figura 77a e 77b). Agora devemos criar uma pasta denominada “estado” dentro da pasta cadastros, assim repita o processo clicando com o botão direito sobre a pasta “cadastros” e escolhendo “New->Folder”, informe o nome da pasta como “estado” e confirme sua criação (Figura 77c e 77d).

Agora clique com o botão direito na pasta “estado” e escolha as opções “New->JSP” e vamos criar o JSP para nosso listar de estados, dando-lhe o nome de “estado” a extensão JSP não precisa ser colocada. Faça como podemos observar na Figura 77e e 77f.

Figura 77 – Criando a estrutura de pastas no Front-End da aplicação



Fonte: O autor.

Na Figura 77g temos a estrutura de pastas criada em nosso projeto e o código fonte gerado pelo Netbeans e que iremos alterá-lo como demonstrado na Figura 78.

Figura 78 – Código fonte da JSP – Estado.jsp (Listar) – Parte 1

```

1  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2  <%@taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt"%>
3  <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4  <jsp:include page="/header.jsp"/>
5  <jsp:include page="/menu.jsp"/>
6
7      <h2>Estados</h2>
8      <table id="datatable" class="display">
9          <thead>
10             <tr>
11                 <th align="left">ID</th>
12                 <th align="left">Nome</th>
13                 <th align="left">Sigla</th>
14                 <th align="right"></th>
15                 <th align="right"></th>
16             </tr>
17         </thead>
18         <tbody>
19             <c:forEach var="estado" items="${estados}">
20                 <tr>
21                     <td align="left">${estado.idEstado}</td>
22                     <td align="left">${estado.nomeEstado}</td>
23                     <td align="left">${estado.siglaEstado}</td>
24                     <td align="center">
25                         <a href=
26                             "${pageContext.request.contextPath}/EstadoExcluir?idEstado=${estado.idEstado}">
27                             Excluir</a></td>
28                         <td align="center">
29                             <a href=
30                             "${pageContext.request.contextPath}/EstadoCarregar?idEstado=${estado.idEstado}">
31                             Alterar</a></td>
32                     </tr>
33             </c:forEach>
34         </tbody>
35     </table>
36
37     <div align="center">
38         <a href="${pageContext.request.contextPath}/EstadoNovo">Novo</a>
39         <a href="index.jsp">Voltar à Página Inicial</a>
40     </div>
41
42

```

Fonte: O autor.

Na sequência do código da Figura 78 temos que criar um script que irá configurar nossa tabela com o componente Datatables. Como pode ser visualizado na Figura 79.

Figura 79 – Código fonte da JSP – Estado.jsp (Listar) – Parte 1

```

43 <script>
44 $(document).ready(function() {
45     console.log('entrei ready');
46     //Carregamos a datatable
47     //$("#datatable").DataTable({}); 
48     $('#datatable').DataTable({
49         "oLanguage": {
50             "sProcessing": "Processando...",
51             "sLengthMenu": "Mostrar _MENU_ registros",
52             "sZeroRecords": "Nenhum registro encontrado.",
53             "sInfo": "Mostrando de _START_ até _END_ de _TOTAL_ registros",
54             "sInfoEmpty": "Mostrando de 0 até 0 de 0 registros",
55             "sInfoFiltered": "",
56             "sInfoPostFix": "",
57             "sSearch": "Buscar:",
58             "sUrl": "",
59             "oPaginate": {
60                 "sFirst": "Primeiro",
61                 "sPrevious": "Anterior",
62                 "sNext": "Seguinte",
63                 "sLast": "Último"
64             }
65         }
66     });
67 });
68 <%@ include file="/footer.jsp" %>
69
70

```

Fonte: O autor.

Nosso estado.jsp finaliza com a importação do footer.jsp na linha 70. Agora você pode executar seu projeto e que estará conforme a figura 80.

Figura 80 – Executando o Estado.jsp.

Módulo Cadastros

Menu Principal
Estado Cidade

Estados

ID	Nome	Sigla	Excluir	Alterar
1	São Paulo	SP	Excluir	Alterar

Mostrando de 1 até 1 de 1 registros

Anterior [1](#) Seguinte

[Novo](#) [Voltar à Página Inicial](#)

Desenvolvendo Aplicações com Java Web

Fonte: O autor.

5.2 CRIANDO O RECURSO DE INCLUIR ESTADO

O desenvolvimento do processo de inclusão no sistema inicia na camada DAO de nosso projeto implementando os métodos cadastrar e inserir.

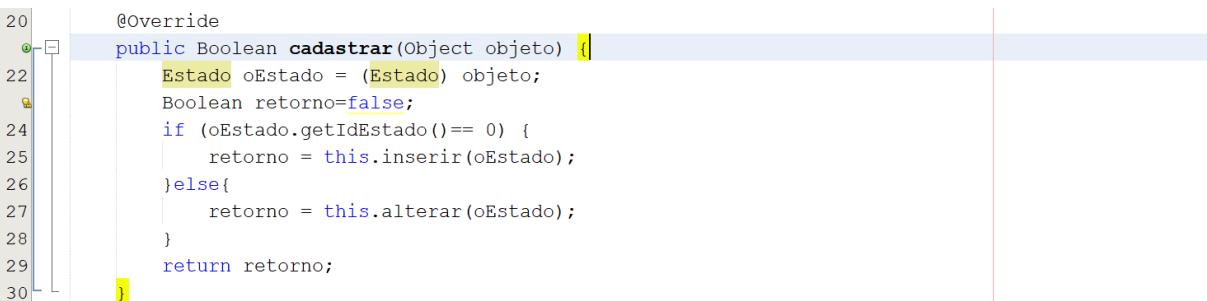
Ambos os métodos podem parecer redundantes, mas possuem funções diferentes o inserir será apenas utilizado internamente na camada DAO assim como o método alterar também será.

As informações provenientes da controller chegarão a camada DAO através do método cadastrar sendo esse responsável por decidir se encaminha o processo para realizar uma inclusão (método inserir) ou uma alteração (método alterar).

Essa decisão é tomada verificando se o idEstado possui valor igual a 0 (inclusão) o que significa um novo registro ou se é maior que zero (alteração) o registro já existe no banco de dados.

Deste modo vamos editar nossa classe EstadoDAO no método cadastrar deixando-o como demonstrado na Figura 81.

Figura 81 – EstadoDAO – Método Cadastrar



```

20
21     @Override
22     public Boolean cadastrar(Object objeto) {
23         Estado oEstado = (Estado) objeto;
24         Boolean retorno=false;
25         if (oEstado.getIdEstado()== 0) {
26             retorno = this.inserir(oEstado);
27         }else{
28             retorno = this.alterar(oEstado);
29         }
30     }

```

Fonte: O autor.

Na linha 22 o parâmetro objeto que o método recebeu que é do tipo Object sobre uma conversão de tipos através de cast para o tipo Estado, a partir da linha 24 verifica-se se o idestado que veio da controller com o objeto é igual a 0 e encaminha para o método incluir senão o idestado for maior que 0 encaminha para o método alterar.

Agora para finalizar a parte de inclusão em nossa camada DAO vamos implementar o método incluir() conforme demonstrado na Figura 82.

Figura 82 – EstadoDAO – Método Incluir

```

34     @Override
35     public Boolean inserir(Object objeto) {
36         Estado oEstado = (Estado) objeto;
37         PreparedStatement stmt = null;
38         String sql = "insert into estado (nomeestado,siglaestado) values (?,?)";
39         try {
40             stmt =conexao.prepareStatement(sql);
41             stmt.setString(1, oEstado.getNomeEstado());
42             stmt.setString(2, oEstado.getSiglaEstado());
43             stmt.execute();
44             conexao.commit();
45             return true;
46         } catch (Exception ex) {
47             try {
48                 System.out.println("Problemas ao cadastrar a Estado! Erro: "+ex.getMessage());
49                 ex.printStackTrace();
50                 conexao.rollback();
51             } catch (SQLException e) {
52                 System.out.println("Erro:"+e.getMessage());
53                 e.printStackTrace();
54             }
55         }
56     }
57

```

Fonte: O autor.

O método inserir também recebe o objeto e realiza a conversão de tipos para Estado, declara um objeto do tipo Statement na linha 35 e na linha 36 monta o comando SQL para ser executado.

Reparem que o comando SQL possui simbolos de “?”, essas são mascaras que permitirão inserir os dados que vieram da controller e finalizar o comando para ser executado. Isso ocorre nas linhas 39 a 40 com os comandos stmt.setString(...).

Na linha 38 é instanciado o objeto da Statement “stmt” passando para ele a conexão com o banco e a string com o comando SQL a ser executado.

Uma vez alterado os parametros com os dados do objeto, aciona-se o método stmt.execute(), que envia o comando para execução no banco de dados retornando True para operação executada com sucesso ou caso contrário False.

Se tudo executar corretamente na linha 44 é executado um commit no banco de dados confirmando o comando enviado, senão será enviado um rollback cancelando as operações no banco.

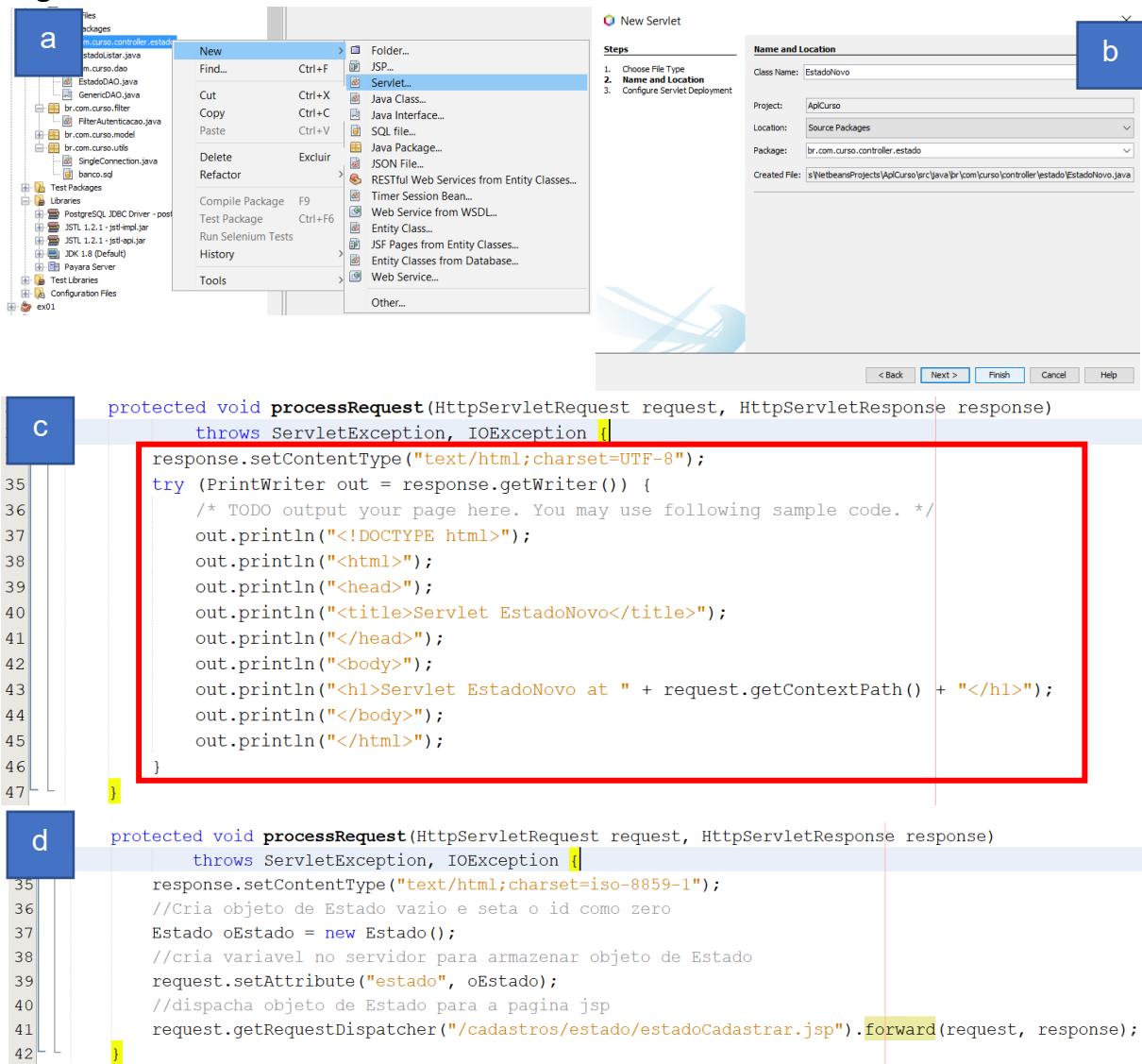
Com a camada DAO preparada para o processo de inclusão de estados iremos começar a trabalhar a camada controller, nesta camada deveremos criar dois servlets.

O primeiro servlet a ser criado é o EstadoNovo cuja a função é enviar para o formulário na interface web um objeto vazio de estado de modo a garantir que nosso campo idEstado esteja com o valor 0.

Para criar o servlet EstadoNovo vamos clicar com o botão direito no pacote br.com.curso.controller.estado e escolher a opção “New->Servlet” e na janela informe o nome do novo servlet “EstadoNovo” e clique em finish (Figura 83a e 83b) e depois apague o código gerado conforme indicado na Figura 83c com um quadro em vermelho.

Em seguida digite o código indicado na Figura 83d.

Figura 83 – Controller – Criando o Servlet - EstadoNovo



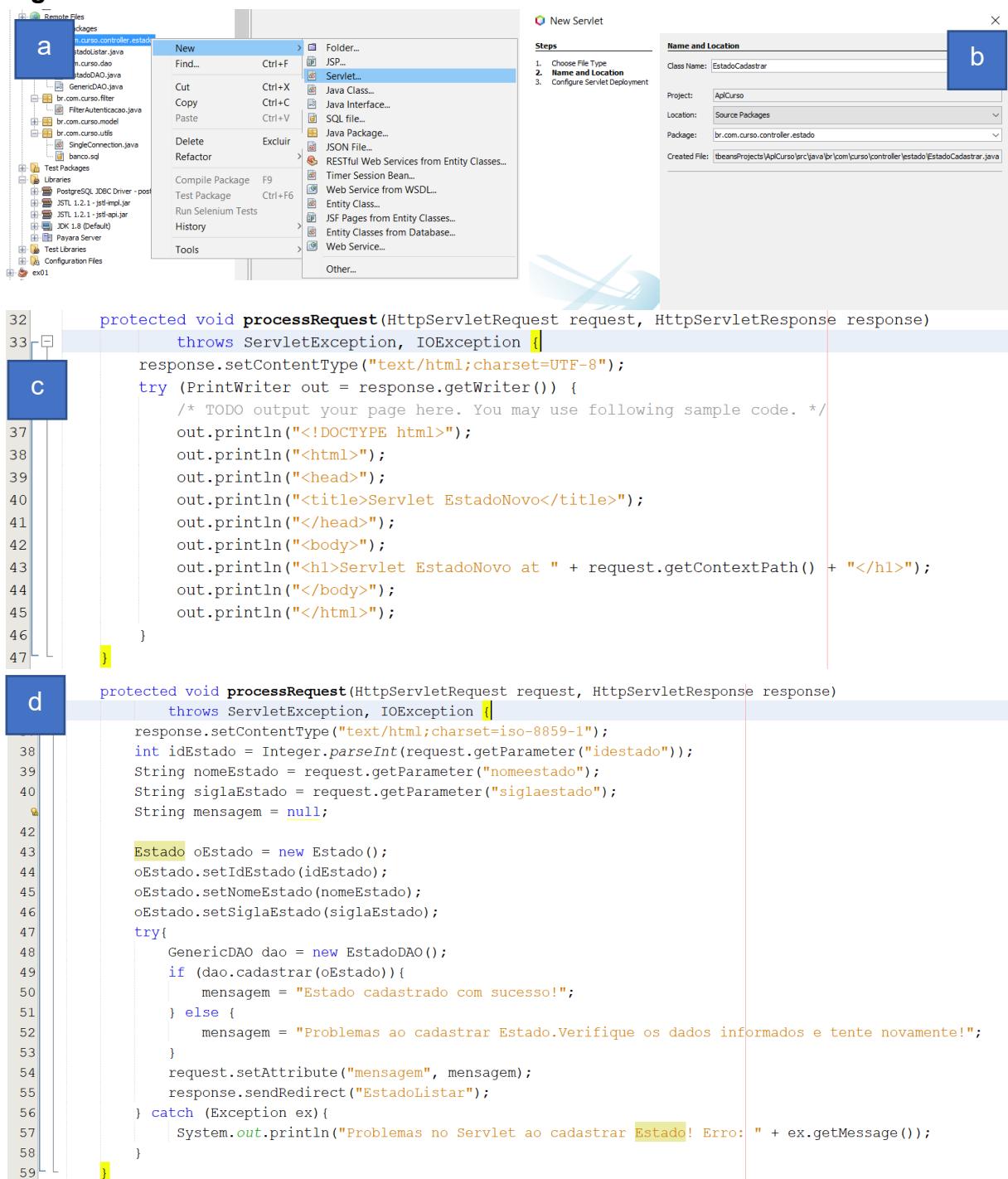
Fonte: O autor.

Neste código na linha 38 é setado o conjunto de caracteres iso-8859-1, na linha 37 é instanciado um objeto vazio de Estado, esse objeto (oEstado) é preparado como um atributo no servidor para que possa ser enviado para o lado Cliente da

aplicação, esse atributo é chamado de “estado”. Na linha 41 é enviado para a página estadoCadastrar.jsp que vai estar na pasta cadastros/estado.

Agora repita os passos e crie o servlet EstadoCadastrar deixando seu código como demonstrado na Figura 84d.

Figura 84 – Controller – Criando o Servlet - EstadoCadastrar



Fonte: O autor.

Faça as importações necessárias, Classe Estado da camada Model, GenericDAO e EstadoDAO da camada dao.

O código recebe os valores digitados no formulário html pelo usuário nas linhas 39,40 e 41 através do comando getParameter onde o parâmetro deste método é exatamente igual a html tag “name” dos inputs do formulário no JSP (então tenha sempre atenção a esse detalhe).

Na linha 44 a 47 é instanciado um objeto Estado chamado oEstado e atribuído a ele as informações capturadas do formulário.

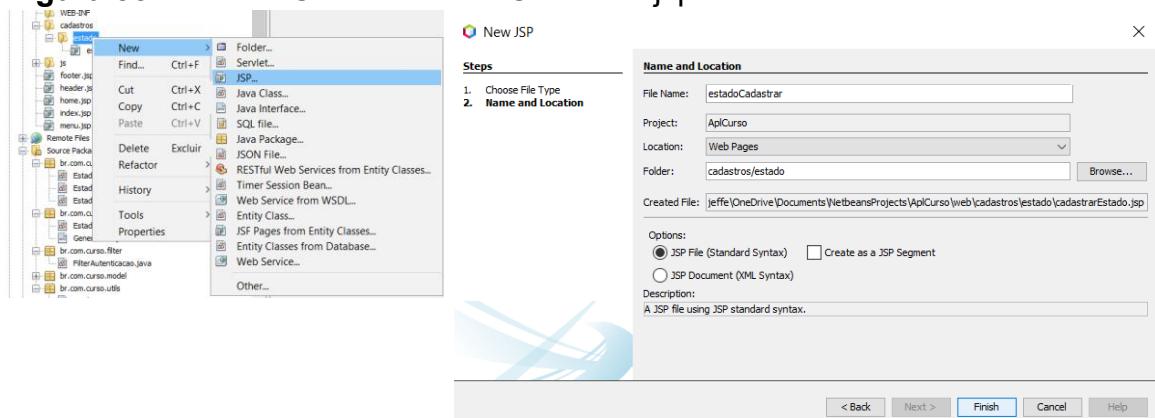
Na linha 51 instancia-se um objeto da camada DAO (EstadoDAO) que chama-se “dao”, esse objeto permite acessar os recursos dessa classe como o método “cadastrar” que precisamos e que fornecemos como parâmetro o objeto oEstado com as informações a serem gravadas.

Se no método cadastrar da dao ocorrer tudo ok no processo é gerada uma mensagem de ok senão uma mensagem de erro.

E na linha 60 a controller redireciona o fluxo de execução do sistema após a operação de cadastrar para o método EstadoListar, gerando novamente para o usuário da aplicação uma lista atualizada com o novo estado cadastrado.

Agora iremos criar em nossa interface web o arquivo estadoCadastrar.jsp que ficará dentro da pasta criada anteriormente /cadastros/estado. Para isso vamos clicar com o botão direito sobre a pasta “estado” em nossa Web Pages e escolhermos as opções “New -> Jsp”, na janela informe o nome do nosso jsp → “estadoCadastrar” e confirme sua criação (Figura 85).

Figura 85 – View – Criando estadoCadastrar.jsp



Fonte: O autor.

Na Figura 86 pode-se verificar o código para nosso JSP (estadoCadastrar.jsp), codifique seu programa de acordo com o demonstrado na Figura.

Figura 86 – View – Codificando estadoCadastrar.jsp

```

1  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2  <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
3  <jsp:include page="/header.jsp"/>
4  <jsp:include page="/menu.jsp"/>
5
6  <form name="cadastarestado" action="EstadoCadastrar"
7      method="POST">
8
9      <table align="center" border="0">
10         <thead>
11             <tr>
12                 <th colspan="2" align="center">
13                     Cadastro de Estado</th>
14             </tr>
15             <tr>
16                 <th colspan="2" align="center">${mensagem}</th>
17             </tr>
18         </thead>
19         <tbody>
20             <tr><td>ID:</td>
21             <td><input type="text" name="idestado" id="idestado" value="${estado.idEstado}" readonly="readonly" /></td></tr>
22             <tr><td>Nome:</td>
23             <td><input type="text" name="nomeestado" id="nomeestado" value="${estado.nomeEstado}"
24                 size="50" maxlength="50" /></td></tr>
25             <tr><td>Sigla:</td>
26             <td><input type="text" name="siglaestado" id="siglaestado" value="${estado.siglaEstado}" /></td></tr>
27             <tr><td colspan="2" align="center">
28                 <input type="submit" name="cadastrar" id="cadastrar" value="Cadastrar" />
29                 <input type="reset" name="limpar" id="limpar" value="Limpar" />
30             </td>
31         </tr>
32         <tr>
33             <td align="center" colspan="2"><h5><a href="index.jsp">Voltar à Página Inicial</a></h5></td>
34         </tr>
35     </tbody>
36 </table>
37 </form>
38 <%@ include file="/footer.jsp" %>
```

Fonte: O autor.

O código acima é um formulário comum de html, com a diferença na expression. language \${estado.idEstado}, \${estado.nomeEstado} e \${estado.siglaEstado}, onde o nome “estado” em vermelho tem que ser exatamente igual ao nome do objeto que geramos lá no Servlet EstadoNovo (lembra-se??) e os atributos em azul devem ser exatamente o nome dos atributos que estão definidos na classe Estado da model.

Quando o usuário clicar em cadastrar nesse formulário terá que gerar um submit que encaminhará as informações para o Servlet EstadoCadastrar que também já desenvolvemos e isto está definido na Html tag “Action” na linha 6 do código acima.

Na Figura 87 temos representado a tela de cadastramento e o resultado na lista de estados após confirmar a inclusão.

Figura 87 – View – Resultado da inclusão
Módulo Cadastros

The figure consists of two screenshots of a Java Web application. The left screenshot shows a 'Cadastro de Estado' (State Registration) form with fields for ID (0), Nome (MINAS GERAIS), and Sigla (MG). Buttons for 'Cadastrar' (Register) and 'Limpar' (Clear) are present, along with a link 'Voltar à Página Inicial' (Back to Home Page). The right screenshot shows a 'Estados' (States) list table with three entries: São Paulo (SP), RIO DE JANEIRO (RJ), and MINAS GERAIS (MG). Each entry has 'Excluir' (Delete) and 'Alterar' (Edit) buttons. Navigation links 'Anterior' (Previous), 'Seguinte' (Next), and 'Novo' (New) are at the bottom.

ID	Nome	Sigla	Excluir	Alterar
1	São Paulo	SP	Excluir	Alterar
2	RIO DE JANEIRO	RJ	Excluir	Alterar
3	MINAS GERAIS	MG	Excluir	Alterar

Fonte: O autor.

5.3 CRIANDO O RECURSO DE ALTERAR ESTADO

Para o desenvolvimento do processo de alteração em nosso sistema web Java devemos criar dois processos o 1º processo é o carregar onde carregamos as informações na tela para que o usuário possa alterá-la. O 2º processo refere-se à alteração dos dados (update).

Para isso vamos desenvolver na camada DAO (EstadoDAO) os dois métodos que faltam serem codificados que são o método “alterar” e o método “carregar”.

Vá ao pacote DAO e abra o programa EstadoDAO.java, e atualize o método carregar() conforme o código que está na Figura 88.

Este método realizará uma consulta das informações já cadastradas referente ao id (registro) que se deseja alterar. Realizada a consulta via select o método cria um objeto para armazenar as informações e retorna o objeto de Estado para a camada controller que o enviará de volta para a tela do usuário para que este possa realizar a alteração.

Figura 88 – DAO – Programando o método carregar()

```

69     @Override
70     public Object carregar(int numero) {
71         int idEstado = numero;
72         PreparedStatement stmt = null;
73         ResultSet rs= null;
74         Estado oEstado = null;
75         String sql="select * from estado where idEstado=?";
76
77         try {
78             stmt = conexao.prepareStatement(sql);
79             stmt.setInt(1, idEstado);
80             rs=stmt.executeQuery();
81             while (rs.next()) {
82                 oEstado = new Estado();
83                 oEstado.setIdEstado(rs.getInt("idEstado"));
84                 oEstado.setNomeEstado(rs.getString("nomeestado"));
85                 oEstado.setSiglaEstado(rs.getString("siglaestado"));
86             }
87             return oEstado;
88         } catch (Exception ex) {
89             System.out.println("Problemas ao carregar Estado! Erro:"+ex.getMessage());
90             return false;
91         }
92     }

```

Fonte: O autor.

Ainda na camada DAO vamos para a segunda etapa de programação que será o desenvolvimento do código referente ao processo de alteração que será realizado no método alterar(), conforme demonstra-se na Figura 89.

Figura 89 – DAO – Programando o método alterar()

```

59     @Override
60     public Boolean alterar(Object objeto) {
61         Estado oEstado = (Estado) objeto;
62         PreparedStatement stmt = null;
63         String sql = "update estado set nomeestado=?,siglaestado=? where idestado=?";
64         try {
65             stmt = conexao.prepareStatement(sql);
66             stmt.setString(1, oEstado.getNomeEstado());
67             stmt.setString(2, oEstado.getSiglaEstado());
68             stmt.setInt(3, oEstado.getIdEstado());
69             stmt.execute();
70             conexao.commit();
71             return true;
72         } catch (Exception ex) {
73             try {
74                 System.out.println("Problemas ao alterar a Estado! Erro: "+ex.getMessage());
75                 ex.printStackTrace();
76                 conexao.rollback();
77             } catch (SQLException e) {
78                 System.out.println("Erro:"+e.getMessage());
79                 e.printStackTrace();
80             }
81         }
82     }

```

Fonte: O autor.

O método alterar receberá o objeto de Estado com as informações alteradas para serem gravadas no banco de dados.

Seguindo a lógica de funcionamento dos tutoriais anteriores nas linhas 61 recebe o objeto de estado carregado com os dados que vierem da tela onde o usuário manipulou as informações.

Na linha 62 declara o stmt a ser utilizado. Na linha 63 monta o comando SQL que é finalizado com seus parâmetros nas linhas 65 a 70, quando é executado e se ocorrer tudo bem no processo de update no banco de dados então realiza-se o commit da transação, senão, se houver algum erro será realizado um rollback da transação de alteração no banco de dados. Se tudo ocorrer ok o método envia como retorno um true senão false.

O próximo passo é criar o **Servlet** que irá atender as requisições de alteração, também serão dois servlets um **EstadoCarregar** que será executado no momento que o usuário clicar na opção alterar na interface da aplicação e que acionará o carregar na camada DAO e enviará os dados a serem alterados para a interface de forma que o usuário possa visualizá-lo. Para criar no novo servlet basta repetir os passos como fizemos anteriormente, clique com o botão direito no pacote da camada controller e escolha no menu – “**New > Servlet**” e caixa de criação coloque o nome do servlet como **EstadoCarregar**. Uma vez criado seu servlet altere seu código no método processRequest conforme a Figura 90.

Figura 90 – Controller – Programando o servlet EstadoCarregar()

```

34
35 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
36     throws ServletException, IOException {
37     response.setContentType("text/html; charset=iso-8859-1");
38     int idEstado = Integer.parseInt(request.getParameter("idEstado"));
39     try {
40         GenericDAO dao = new EstadoDAO();
41         request.setAttribute("estado", dao.carregar(idEstado));
42         request.getRequestDispatcher("cadastros/estado/estadoCadastrar.jsp").forward(request, response);
43     } catch (Exception e) {
44         System.out.println("Problema na Servelet carregar despesa!Erro: " + e.getMessage());
45         e.printStackTrace();
46     }

```

Fonte: O autor.

Neste código na linha 36 é setado o conjunto de caracteres iso-8859-1, na linha 37 pega o parametro idEstado que é enviado junto do link (url) no estado.jsp (localhost:8080/apiCurso/EstadoCarregar?idEstado=1).

Na linha 39 é instânciado um objeto do tipo DAO de Estado, onde na linha 40 é enviado o Id para o método carregar. O retorno deste método será um objeto com as informações do estado preterido.

Na linha 41 é redirecionado o fluxo de execução de nossa aplicação para a página JSP em “/cadastros/estado/estadoCadastrar.jsp” que já havíamos criado no processo de inclusão de nosso sistema. A tela então exibirá os dados carregados para que o usuário possa alterá-los.

Quando terminada a alteração dos dados o usuário clicando no botão “Cadastrar” irá encaminhar o processo de nosso sistema para o 2º servlet que será utilizado e que realizará o processo de alteração efetivamente.

O servlet em questão é o **EstadoCadastrar**, este servlet também já possuímos desenvolvido desde a criação da inclusão em nosso sistema.

5.4 CRIANDO O RECURSO DE EXCLUIR ESTADO

O desenvolvimento do processo de exclusão no sistema inicia na camada DAO de nosso projeto implementando o método excluir. Vá ao pacote DAO e abra o programa EstadoDAO.java, e no método excluir altere a codificação conforme o demonstrado na Figura 91.

Figura 91 – DAO – Programando o método excluir()

```

85     @Override
86     public Boolean excluir(int numero) {
87         int idEstado = numero;
88         PreparedStatement stmt= null;
89
90         String sql = "delete from estado where idestado=?";
91         try {
92             stmt = conexao.prepareStatement(sql);
93             stmt.setInt(1, idEstado);
94             stmt.execute();
95             conexao.commit();
96             return true;
97         } catch (Exception ex) {
98             System.out.println("Problemas ao excluir a Estado! Erro: "
99                         +ex.getMessage());
100            try {
101                conexao.rollback();
102            } catch (SQLException e) {
103                System.out.println("Erro rollback "+e.getMessage());
104                e.printStackTrace();
105            }
106        }
107    }
108 }
```

Fonte: O autor.

O método excluir irá receber da controller o id do estado a ser excluído. Segundo a lógica de funcionamento dos processos anteriores nas linhas 87 e 88 recebe o valor do id a ser excluído e declara o stmt a ser utilizado. Na linha 90 monta o comando SQL que é finalizado com seu parâmetro nas linhas 92 a 95, quando é executado e a operação é confirmada com o commit.

Se tudo ocorrer ok, o método envia como retorno um true senão false e executa um rollback nas instruções sql no banco desfazendo a operação.

O próximo passo é criar o Servlet que irá atender as requisições de exclusão, que será denominado **EstadoExcluir** que será executado no momento que o usuário clicar na opção excluir na interface da aplicação e que acionará o método excluir na camada DAO. Para criar um novo servlet basta repetir os passos como fizemos anteriormente, clique com o botão direito no pacote da camada controller de estado e escolha no menu – “**New > Servlet**” e caixa de criação coloque o nome do servlet como **EstadoExcluir**. Uma vez criado seu servlet altere seu código no método processRequest conforme a Figura 92.

Figura 92 – Controller – Programando o servlet EstadoExcluir()

```

34     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35         throws ServletException, IOException {
36             response.setContentType("text/html;charset=iso-8859-1");
37             int idEstado = Integer.parseInt(request.getParameter("idEstado"));
38             String mensagem = null;
39             try {
40                 GenericDAO dao = new EstadoDAO();
41                 if (dao.excluir(idEstado)) {
42                     mensagem = "Estado excluido com Sucesso!";
43                 } else {
44                     mensagem = "Problemas ao excluir Estado";
45                 }
46                 request.setAttribute("mensagem", mensagem);
47                 response.sendRedirect("EstadoListar");
48             } catch (Exception ex) {
49                 System.out.println("Problemas no Servelet ao excluir Estado! Erro: "+ ex.getMessage());
50                 ex.printStackTrace();
51             }
52         }

```

Fonte: O autor.

Neste código na linha 36 é setado o conjunto de caracteres iso-8859-1, na linha 37 pega o parâmetro idEstado que é enviado junto do link (url) no estado.jsp.

Na linha 40 é instanciado um objeto do tipo DAO de Estado, onde na linha 41 é enviado o Id para o método excluir da camada DAO. O retorno deste método será True ou False.

Na linha 47 é redirecionado o fluxo de execução de nossa aplicação para o servlet EstadoListar para que o sistema mostre para o usuário uma lista atualizada após o processo de exclusão.

CAPÍTULO 7 – DESENVOLVIMENTO: CRUD 1-N

Agora vamos iniciar o desenvolvimento de nosso primeiro Crud 1-N utilizando-se de 2 tabelas no banco de dados, a tabela de cidade e a tabela de estado o qual trabalhamos nos capítulos anteriores.

Verifique no seu banco de dados se possui a tabela estado (se não tiver crie novamente) também faça a criação da tabela cidade, como pode-se visualizar na Figura 93.

Figura 93 – Criando a Tabela de Cidade

```
Create table estado(
    idestado serial primary key,
    nomeestado varchar(50) not null,
    siglaestado varchar(02) not null
);

insert into estado(nomeestado,siglaestado) values ('São Paulo','SP');

create table cidade(
    idcidade serial primary key,
    nomecidade varchar(100) not null,
    situacao varchar(1) not null,
    idestado int not null,
    constraint fk_estado foreign key (idestoado) references estado(idestoado)
);

insert into cidade(nomecidade,situacao,idestoado) values ('Fernandopolis','A',1);
insert into cidade(nomecidade,situacao,idestoado) values ('Jales','A',1);
```

Verifique se já está criada

A tabela cidade você deve criar

Fonte: O autor.

Em relação as nossas classes ficarão como demonstrado na Figura 94.

Figura 94 – Criando a Tabela de Cidade



Fonte: O autor.

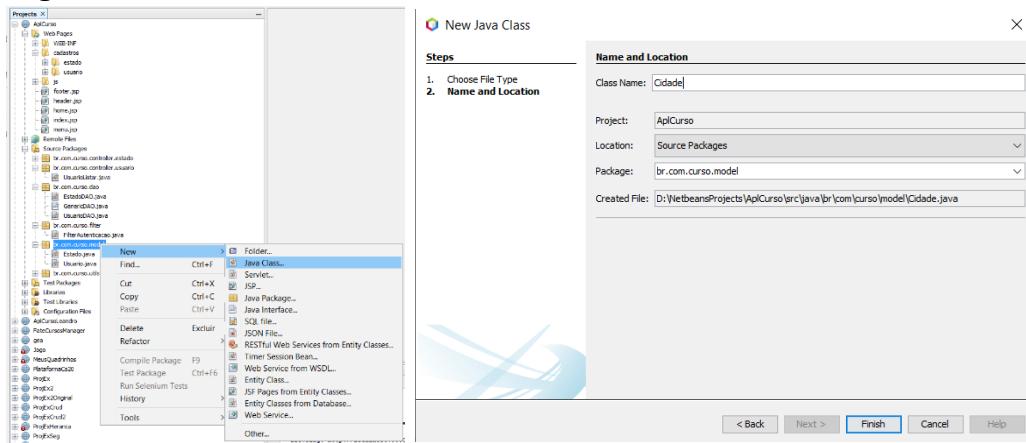
Como podemos observar temos uma relação de multiplicidade de 0..* entre a classe Estado e a classe Cidade, assim como temos uma relação de cardinalidade 0..* entre as tabelas estado e cidade em nosso banco de dados.

No banco de dados (modelo relacional) esse processo é representado pela chave estrangeira (foreign key) que faz a ligação entre as referidas tabelas, no caso de nossas classes no modelo orientado a objeto não existem chaves estrangeiras.

A representação de relações entre classes no modelo orientado a objeto se dá através de associações como estudamos na matéria de engenharia de software mais especificamente quando abordamos diagramas de classe.

No modelo demonstrado na Figura 95 observa-se uma associação simples entre as duas classes e a multiplicidade demonstra a direção em que a relação acontece. Entendido esses conceitos iniciais vamos implementar a Model em nossa aplicação, **primeiro crie a classe Cidade no pacote br.com.curso.model** como demonstrado na Figura 95.

Figura 95 – Criando a Tabela de Cidade



Fonte: O autor.

Uma vez criada nossa classe Cidade vamos programar os atributos dessa classe que são idCidade do tipo int, nomeCidade do tipo String, situação do tipo String. Mas temos que representar a relação entre as classes Estado e Cidade em nossa classe Cidade, para isso vamos criar um atributo denominado estado que será do tipo da classe Estado, assim você pode concluir que as classes que criamos também podem ser utilizadas como um tipo de dado.

Quando criamos esse atributo estado como tipo de dado Estado (Classe), estamos fazendo um processo que denominamos na orientação a objeto como agregação, ou seja, estamos incorporando dentro dos objetos de cidade um objeto do tipo estado. Isto em um primeiro momento pode parecer confuso para você, mas fique tranquilo pois esse conceito é muito simples e vamos trabalhá-lo durante o desenvolvimento deste CRUD.

Assim crie os atributos conforme demonstrado na Figura 96 e observe especialmente a linha 7 do código onde estamos definindo a agregação estado em cidade.

Figura 96 – Criando os atributos na classe Cidade

```

1 package br.com.curso.model;
2
3 public class Cidade {
4
5     private int idCidade;
6     private String nomeCidade;
7     private Estado estado; ←
8     private String situacao;
9
10 }
11

```

Fonte: O autor.

A seguir crie os construtores com parâmetros e sem parâmetros utilizando os atalhos que você já conhece do Netbeans, e complemente o código que for necessário.

Figura 97 – Criando os construtores da classe Cidade

```

9
10    public Cidade(int idCidade, String nomeCidade, Estado estado, String situacao) {
11        this.idCidade = idCidade;
12        this.nomeCidade = nomeCidade;
13        this.estado = estado;
14        this.situacao = situacao;
15    }
16
17    public Cidade() {
18        this.idCidade = 0;
19        this.nomeCidade = "";
20        this.situacao = "A";
21        this.estado = new Estado();
22    }
23

```

Fonte: O autor.

Observe que na linha de código 21 na Figura 97 temos que passar para o atributo um objeto da classe Estado pois esse é o nosso tipo de dado. Agora vamos criar os métodos Gets and Sets de nossa classe Cidade, utilize novamente os atalhos do Netbeans para gerá-los e confira o código com a Figura 98.

Figura 98 – Métodos Gets e Sets

```

24  public int getIdCidade() {
25      return idCidade;
26  }
27
28  public void setIdCidade(int idCidade) {
29      this.idCidade = idCidade;
30  }
31
32  public String getNomeCidade() {
33      return nomeCidade;
34  }
35
36  public void setNomeCidade(String nomeCidade) {
37      this.nomeCidade = nomeCidade;
38  }
39
40  public Estado getEstado() {
41      return estado;
42  }
43
44  public void setEstado(Estado estado) {
45      this.estado = estado;
46  }
47
48  public String getSituacao() {
49      return situacao;
50  }
51
52  public void setSituacao(String situacao) {
53      this.situacao = situacao;
54  }
55

```

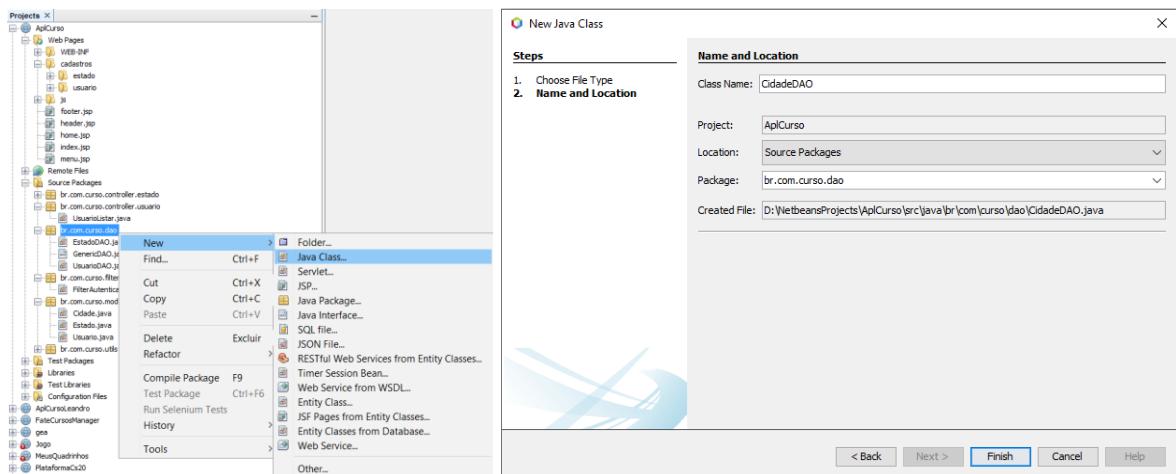
Fonte: O autor.

Diferentemente do capítulo anterior que fomos criando recurso por recurso (listar, cadastrar, excluir e alterar), agora vamos implementar uma camada do sistema por vez. Já implementamos nossa camada Model para a classe Cidade, nosso próximo passo será implementar a camada DAO e depois a Controller até chegarmos nos JSPs.

7.1 CRIANDO A CAMADA DAO (CIDADEDAO)

Agora vamos criar nossa classe CidadeDAO em nosso pacote br.com.curso.dao, para isso clique com o botão direito sobre o pacote e escolha New->Java Class e crie a classe com o nome de CidadeDAO (Figura 99).

Figura 99 – Criando a classe CidadeDAO



Fonte: O autor.

Não se esqueça de colocar o implements para a nossa interface GenericDAO na declaração de nossa nova classe e implementar os métodos abstratos (Figura 100).

Figura 100 – Implementando os métodos abstratos da GenericDAO

```

1 package br.com.curso.dao;
2
3
4 public class CidadeDAO implements GenericDAO {
5     Implement all abstract methods
6     Make class CidadeDAO abstract
7     Implement unimplemented abstract methods of br.com.curso.dao.GenericDAO
8     Create missing javadoc for CidadeDAO
9     Create Test Class [JUnit in Test Packages]
10    Create Test Class [TestNG in Test Packages]
11    Create Test Class [JUnit4 in Test Packages]
12    Create Test Class [Selenium in Test Packages]
13

```

Fonte: O autor.

Na Figura 101 pode-se observar os métodos abstratos implementados em nossa classe CidadeDAO ainda com a exception padrão de “not implemented”, a partir de agora vamos codificar cada um dos métodos abstratos.

Figura 101 – Métodos Abstratos implementados

```

7   public class CidadeDAO implements GenericDAO {
8
9     @Override
10    public Boolean cadastrar(Object objeto) {
11      throw new UnsupportedOperationException("Not supported yet."); //To change this template, choose "File | Settings | File Templates".
12    }
13
14    @Override
15    public Boolean inserir(Object objeto) {
16      throw new UnsupportedOperationException("Not supported yet."); //To change this template, choose "File | Settings | File Templates".
17    }
18
19    @Override
20    public Boolean alterar(Object objeto) {
21      throw new UnsupportedOperationException("Not supported yet."); //To change this template, choose "File | Settings | File Templates".
22    }
23
24    @Override
25    public Boolean excluir(int numero) {
26      throw new UnsupportedOperationException("Not supported yet."); //To change this template, choose "File | Settings | File Templates".
27    }
28
29    @Override
30    public Object carregar(int numero) {
31      throw new UnsupportedOperationException("Not supported yet."); //To change this template, choose "File | Settings | File Templates".
32    }
33
34    @Override
35    public List<Object> listar() {
36      throw new UnsupportedOperationException("Not supported yet."); //To change this template, choose "File | Settings | File Templates".
37    }
38
39  }

```

Fonte: O autor.

Primeiramente temos que declarar em nossa classe o atributo que irá receber o objeto de conexão e criarmos o método construtor de nossa classe CidadeDAO, então codifique como demonstrado na Figura 102, não se esqueça das importações das classes nas linhas 4 e 5.

Figura 102 – Método Construtor e atributo para conexão

```

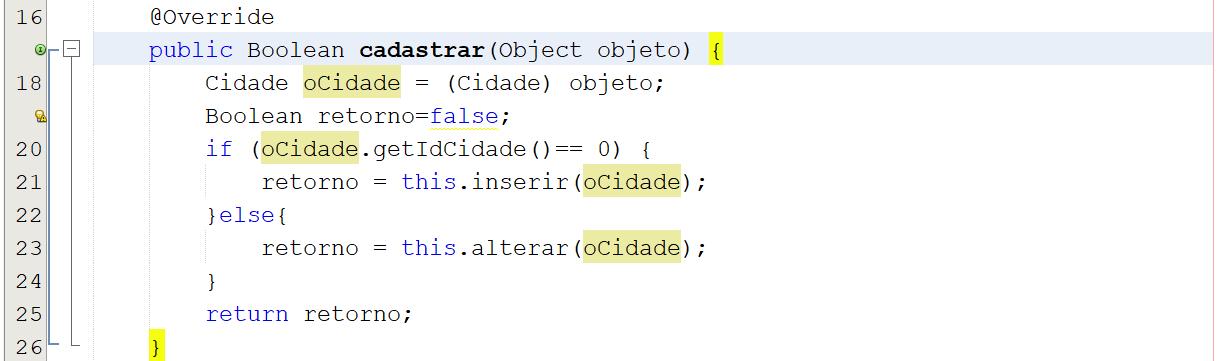
1 package br.com.curso.dao;
2
3 import br.com.curso.utils.SingleConnection;
4 import java.sql.Connection; ←
5 import java.util.List; ←
6
7 public class CidadeDAO implements GenericDAO {
8
9     private Connection conexao; ←
10
11    public CidadeDAO() throws Exception{ ←
12        conexao = SingleConnection.getConnection(); ←
13    }
14

```

Fonte: O autor.

Lembre-se sempre de a cada novo código que criarmos salvar seu projeto e dar um build para encontrarmos possíveis erros de programação (syntax). Agora vamos implementar o método cadastrar, para isso faça a codificação conforme demonstrado na Figura 103, não se esqueça de fazer a importação da classe Cidade da model.

Figura 103 – Método Cadastrar (CidadeDAO)



```

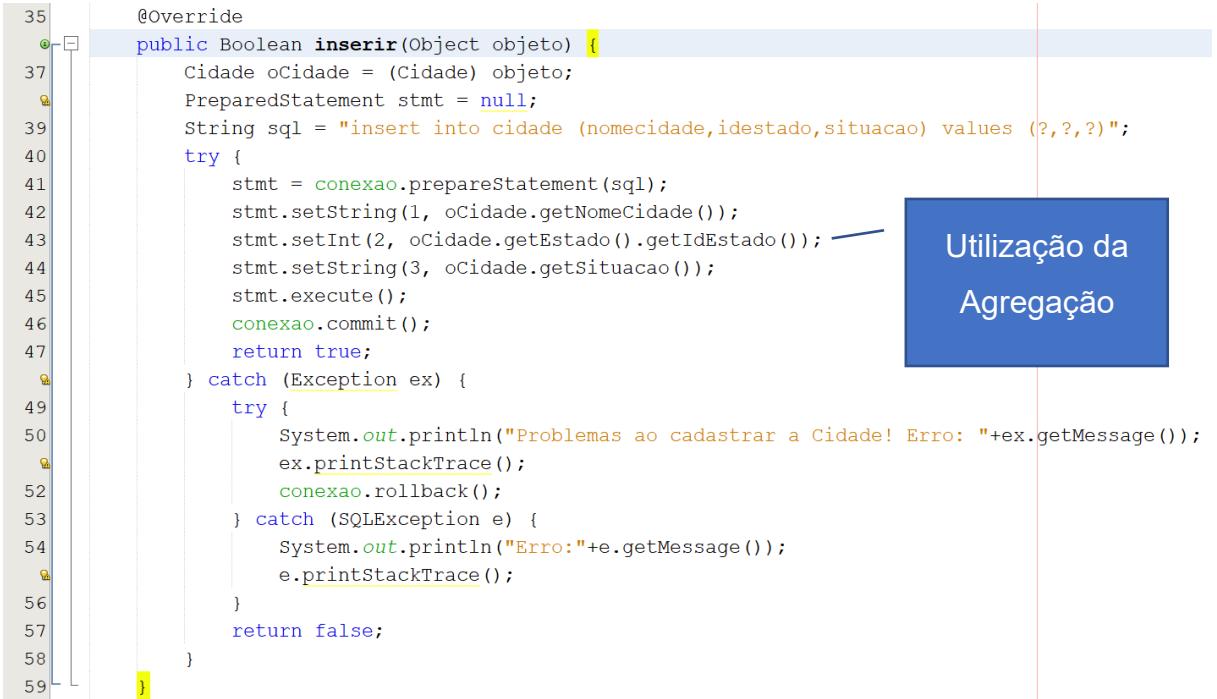
16     @Override
17     public Boolean cadastrar(Object objeto) {
18         Cidade oCidade = (Cidade) objeto;
19         Boolean retorno=false;
20         if (oCidade.getIdCidade()== 0) {
21             retorno = this.inserir(oCidade);
22         }else{
23             retorno = this.alterar(oCidade);
24         }
25         return retorno;
26     }

```

Fonte: O autor.

Vamos na sequência implementar o método inserir (Figura 104).

Figura 104 – Método Inserir (CidadeDAO)



```

35     @Override
36     public Boolean inserir(Object objeto) {
37         Cidade oCidade = (Cidade) objeto;
38         PreparedStatement stmt = null;
39         String sql = "insert into cidade (nomecidade,idestado,situacao) values (?,?,?)";
40         try {
41             stmt = conexao.prepareStatement(sql);
42             stmt.setString(1, oCidade.getNomeCidade());
43             stmt.setInt(2, oCidade.getEstado().getIdEstado());
44             stmt.setString(3, oCidade.getSituacao());
45             stmt.execute();
46             conexao.commit();
47             return true;
48         } catch (Exception ex) {
49             try {
50                 System.out.println("Problemas ao cadastrar a Cidade! Erro: "+ex.getMessage());
51                 ex.printStackTrace();
52                 conexao.rollback();
53             } catch (SQLException e) {
54                 System.out.println("Erro:"+e.getMessage());
55                 e.printStackTrace();
56             }
57         }
58     }

```

Utilização da
Agregação

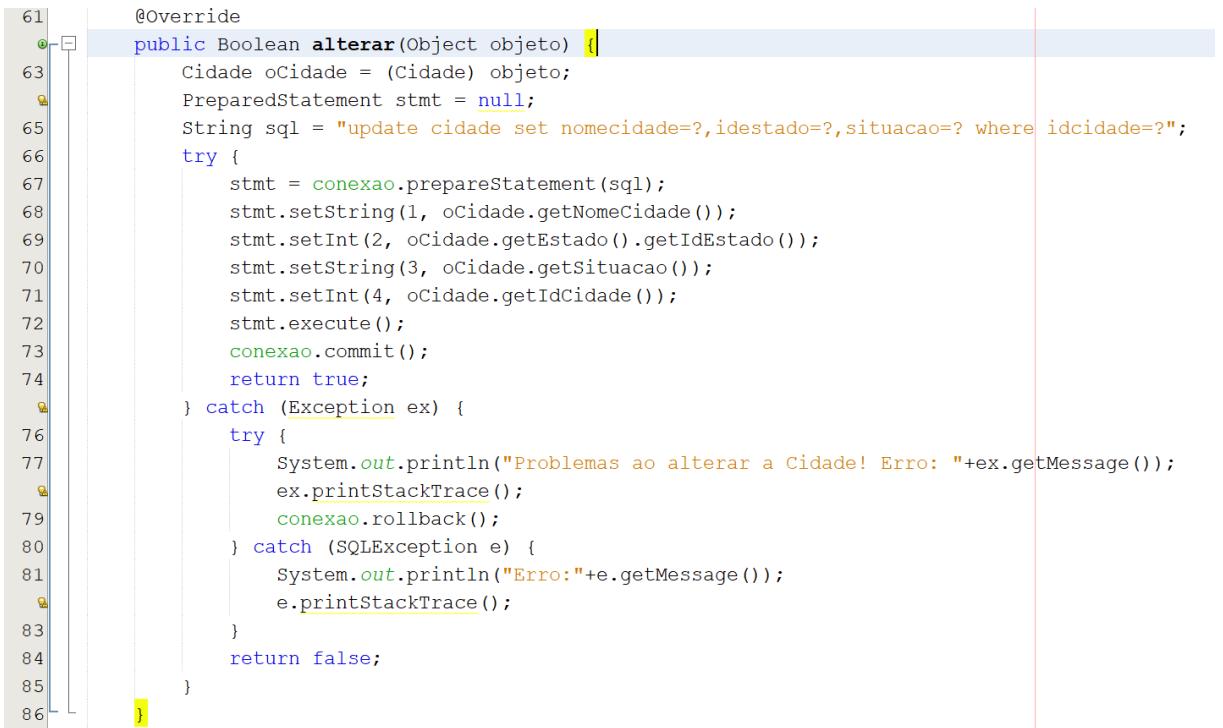
Fonte: O autor.

Observe especificamente a linha de código 43 da Figura 104, nela temos a utilização da agregação que declaramos lá na camada model. Assim temos o atributo idEstado que estamos buscando através do método getIdEstado() que pertence a classe Estado.

Esse atributo está sendo invocado através do atributo estado da classe Cidade que é do tipo de dado Estado. Então temos um objeto completo de Estado incorporado dentro da classe de Cidade, ou seja, agregado na classe Cidade.

Utilizando os mesmos conceitos estabelecidos na inserção vamos agora implementar o método de alteração, faça como demonstrado na Figura 105.

Figura 105 – Método Alterar (CidadeDAO)



```

61  @Override
62  public Boolean alterar(Object objeto) {
63      Cidade oCidade = (Cidade) objeto;
64      PreparedStatement stmt = null;
65      String sql = "update cidade set nomecidade=?,idestado=?,situacao=? where idcidade=?";
66      try {
67          stmt = conexao.prepareStatement(sql);
68          stmt.setString(1, oCidade.getNomeCidade());
69          stmt.setInt(2, oCidade.getEstado().getIdEstado());
70          stmt.setString(3, oCidade.getSituacao());
71          stmt.setInt(4, oCidade.getIdCidade());
72          stmt.execute();
73          conexao.commit();
74          return true;
75      } catch (Exception ex) {
76          try {
77              System.out.println("Problemas ao alterar a Cidade! Erro: "+ex.getMessage());
78              ex.printStackTrace();
79              conexao.rollback();
80          } catch (SQLException e) {
81              System.out.println("Erro:"+e.getMessage());
82              e.printStackTrace();
83          }
84      }
85  }
86 }
```

Fonte: O autor.

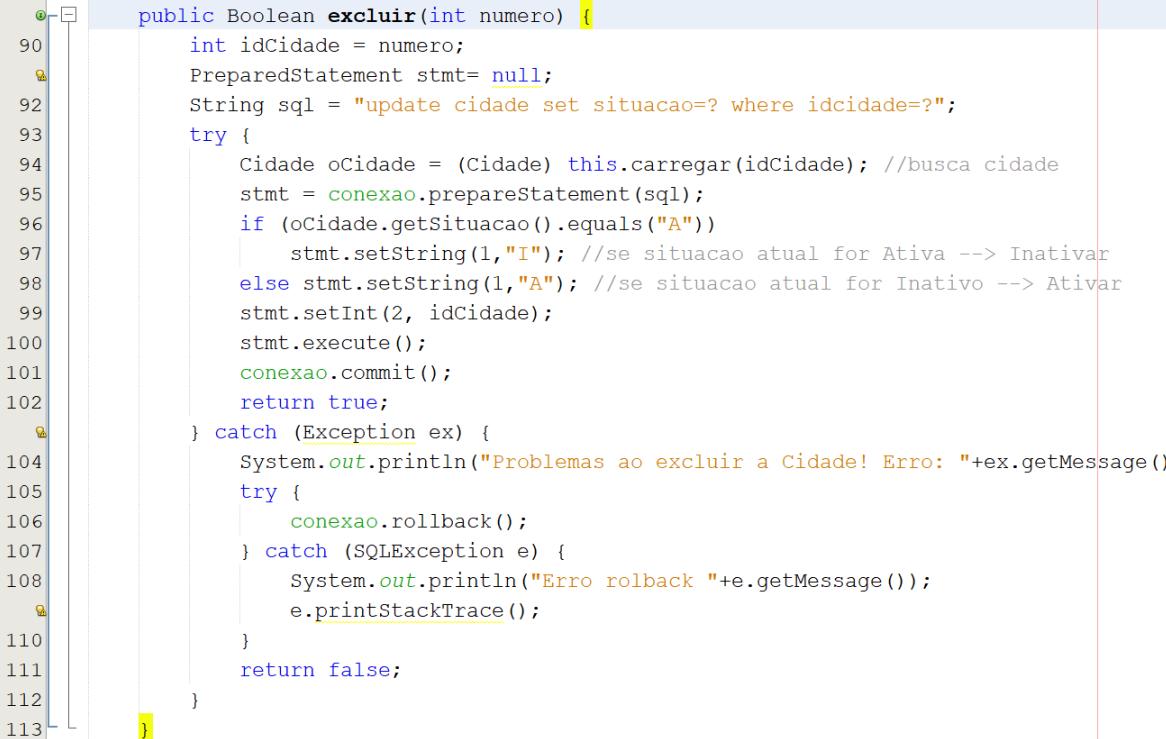
No método de alteração não temos novidades, apenas a utilização do recurso de agregação da classe Estado em Cidade como já visto anteriormente no método inserir().

Agora vamos desenvolver o método excluir() em nosso crud de cidade. Neste Crud vamos optar por não realizar a exclusão efetiva do cadastro no banco de dados, então você deve observar que o comando SQL utilizado não será o delete, mas sim o

update pois atualizaremos o atributo situação com a letra “I” de Inativo ou a “A” ativo, determinando seu status dentro do sistema.

Vamos programar o método excluir() conforme demonstrado na Figura 106.

Figura 106 – Método Excluir (CidadeDAO)



```

public Boolean excluir(int numero) {
    int idCidade = numero;
    PreparedStatement stmt= null;
    String sql = "update cidade set situacao=? where idcidade=?";
    try {
        Cidade oCidade = (Cidade) this.carregar(idCidade); //busca cidade
        stmt =conexao.prepareStatement(sql);
        if (oCidade.getSituacao().equals("A"))
            stmt.setString(1,"I"); //se situacao atual for Ativa --> Inativar
        else stmt.setString(1,"A"); //se situacao atual for Inativo --> Ativar
        stmt.setInt(2, idCidade);
        stmt.execute();
        conexao.commit();
        return true;
    } catch (Exception ex) {
        System.out.println("Problemas ao excluir a Cidade! Erro: "+ex.getMessage());
        try {
            conexao.rollback();
        } catch (SQLException e) {
            System.out.println("Erro rollback "+e.getMessage());
            e.printStackTrace();
        }
        return false;
    }
}

```

Fonte: O autor.

Na Figura 107 temos a implementação do método carregar(), deve-se observar a linha de código 132 e 133 onde é realizado o carregamento dos dados de estado para um objeto do mesmo tipo para depois ser atributo ao objeto de cidade.

Deste modo quando carregamos uma cidade automaticamente temos também agregado a ela um objeto de estado com seus dados carregados, este recurso facilita nosso trabalho na camada view no transporte e exibição de dados.

Quando implementarmos o método Listar na Figura 109 neste cadastro de cidade também será feito o carregamento de todos os dados de estado em cidade, mas você deve tomar cuidado ao efetuar esse trabalho em um cadastro com várias agregações e principalmente se perceber que ocorrerá uma agregação dentro da outra pois pode ocasionar lentidão na execução do sistema.

Assim é métodos do tipo listar devemos analisar a necessidade do carregamento completo dos dados dos objetos agregados, geralmente deve-se dar

preferência neste caso para o carregamento do objeto agregado apenas com o id carregado.

Mas vamos implementar o método carregar conforme a Figura 107.

Figura 107 – Método Carregar (CidadeDAO)

```

115  @Override
116  public Object carregar(int numero) {
117      int idCidade = numero;
118      PreparedStatement stmt = null;
119      ResultSet rs= null;
120      Cidade oCidade = null;
121      String sql="select * from cidade where idcidade=?";
122      try {
123          stmt = conexao.prepareStatement(sql);
124          stmt.setInt(1, idCidade);
125          rs=stmt.executeQuery();
126          while (rs.next()) {
127              oCidade = new Cidade();
128              oCidade.setIdCidade(rs.getInt("idcidade"));
129              oCidade.setNomeCidade(rs.getString("nomecidade"));
130              oCidade.setSituacao(rs.getString("situacao"));
131
132              EstadoDAO oEstadoDAO = new EstadoDAO();
133              oCidade.setEstado((Estado) oEstadoDAO.carregar(rs.getInt("idestado")));
134          }
135          return oCidade;
136      } catch (Exception ex) {
137          System.out.println("Problemas ao carregar Cidade! Erro:"+ex.getMessage());
138          return false;
139      }
140  }

```

Fonte: O autor.

Como referido anteriormente no método listar do cadastro de cidade vamos efetuar o carregamento de dados no objeto de estado pois precisaremos de suas informações na camada view e também por ser apenas uma agregação e estar em apenas em 1 nível o que não ocasionará lentidão.

Você deve ficar atento a essas situações durante seu desenvolvimento para evitar problemas de lentidão durante o carregamento ou listar dados em seu sistema por excesso de agregações encadeadas.

Então vamos implementar o método listar de nossa classe CidadeDAO conforme está demonstrado na Figura 108.

Figura 108 – Método Listar (CidadeDAO)

```

142     @Override
143     public List<Object> listar() {
144         List<Object> resultado = new ArrayList<>();
145         PreparedStatement stmt = null;
146         ResultSet rs = null;
147         String sql = "Select * from cidade order by nomecidade";
148         try {
149             stmt =conexao.prepareStatement(sql);
150             rs=stmt.executeQuery();
151             while (rs.next()) {
152                 Cidade oCidade = new Cidade();
153                 oCidade.setIdCidade(rs.getInt("idcidade"));
154                 oCidade.setNomeCidade(rs.getString("nomecidade"));
155                 oCidade.setSituacao(rs.getString("situacao"));
156
157                 EstadoDAO oEstadoDAO = null;
158                 try {
159                     oEstadoDAO = new EstadoDAO();
160                 } catch (Exception ex) {
161                     System.out.println("Erro buscar estado "+ex.getMessage());
162                     ex.printStackTrace();
163                 }
164                 oCidade.setEstado((Estado) oEstadoDAO.carregar(rs.getInt("idestado")));
165                 resultado.add(oCidade);
166             }
167         } catch (SQLException ex) {
168             System.out.println("Problemas ao listar Cidade! Erro: "
169                         +ex.getMessage());
170         }
171         return resultado;
172     }

```

Fonte: O autor.

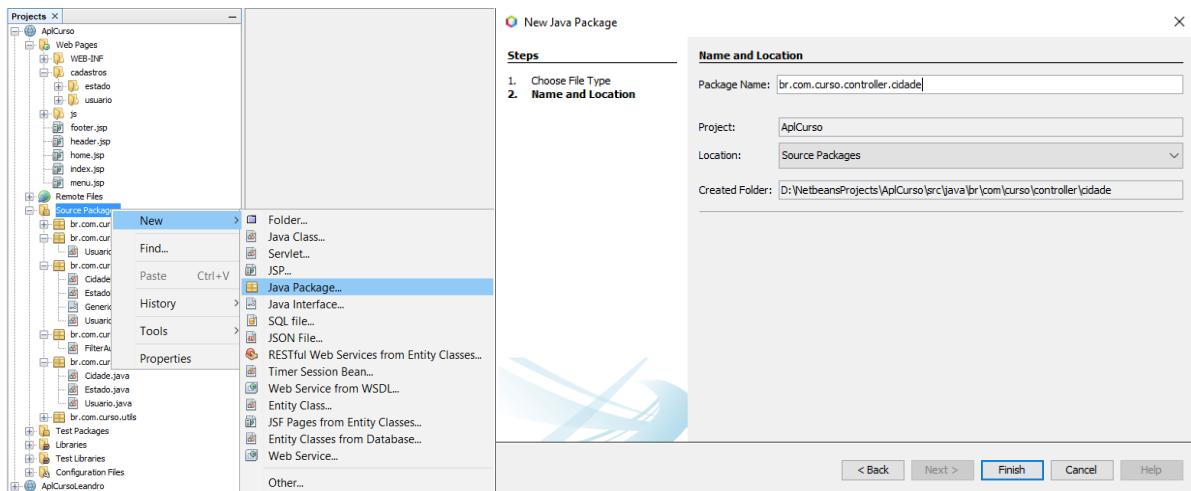
Assim finalizamos o desenvolvimento da camada DAO, a seguir iremos codificar a camada controller.

7.2 CRIANDO A CAMADA CONTROLLER

Nossa camada controller em nosso projeto está sendo desenvolvida em diferentes pacotes, um para cada tipo de dado tratado em nosso sistema assim para o Estado temos o pacote br.com.curso.controller.estado, então agora iremos criar um novo pacote para nosso crud de Cidade.

Deste modo, faça como demonstrado na Figura 109, clique com o botão direito em source packages e escolha a opção Java Package e crie um novo pacote com o nome br.com.curso.controller.cidade.

Figura 109 – Método Listar (CidadeDAO)



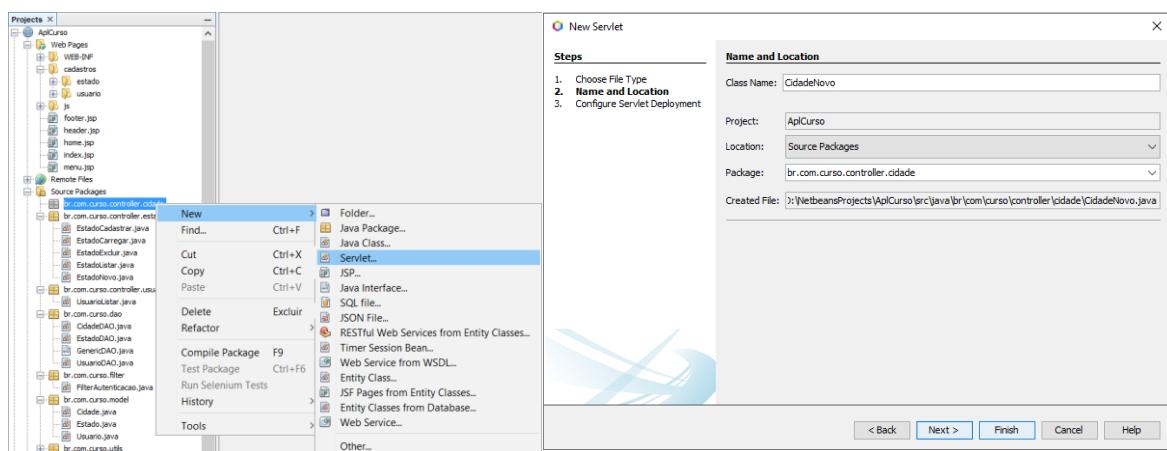
Fonte: O autor.

Uma vez criado nosso pacote de controller para cidade podemos iniciar o desenvolvimento dos servlets CidadeNovo, CidadeCarregar, CidadeListar, CidadeExcluir e CidadeListar, como veremos a seguir.

7.2.1 Controller Cidade: Servlet CidadeNovo

Vamos inicialmente criar nosso servlet CidadeNovo em nosso pacote `br.com.curso.controller.cidade`, para isso faça como demonstrado na Figura 110, clique com o botão direito sobre o pacote de controller de cidade e escolha a opção New → Servlet e crie o novo servlet como o nome de CidadeNovo.

Figura 110 – Criando Servlet CidadeNovo



Fonte: O autor.

Uma vez criado o seu servlet CidadeNovo vá até o método processRequest() apague o código gerado pelo Netbeans e implemente este método conforme a Figura 111.

Figura 111 – Implementando o método processRequest em CidadeNovo

```

36     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37         throws ServletException, IOException, Exception {
38             response.setContentType("text/html;charset=iso-8859-1");
39             //Cria objeto de Cidade vazio e seta o id como zero
40             Cidade oCidade = new Cidade();
41             //cria variavel no servidor para armazenar objeto de Cidade
42             request.setAttribute("cidade", oCidade);
43             //busca uma lista de estados para alimentar a caixa de seleção na view (jsp)
44             EstadoDAO oEstadoDAO = new EstadoDAO();
45             request.setAttribute("estados", oEstadoDAO.listar());
46             //dispacha os objetos de cidade e a lista de estados para a pagina jsp
47             request.getRequestDispatcher("/cadastros/cidade/cidadeCadastrar.jsp").forward(request, response);
48         }
    
```

Fonte: O autor.

Como podemos observar na linha de código 44 e 45 estamos carregando uma lista de estados para que o usuário possa escolher em uma select (tag html) na camada view, qual o estado que nossa cidade que será cadastrada pertence.

7.2.2 Controller Cidade: Servlet CidadeCarregar

Vamos repetir os passos que fizemos anteriormente e criar nosso servlet CidadeCarregar em nosso pacote br.com.curso.controller.cidade, para isso clique com o botão direito sobre o pacote de controller de cidade e escolha a opção New → Servlet e crie o novo servlet como o nome de CidadeCarregar.

Uma vez criado o seu servlet CidadeCarregar vá até o método processRequest() apague o código gerado pelo Netbeans e implemente este método conforme a Figura 112.

Agora vamos realizar a busca dos dados de cidade através da camada DAO (linhas 41 e 43) e criarmos a variável de contexto de servidor para enviar os dados a ao front-end.

Como podemos observar na linha de código 45 e 46, também estamos carregando uma lista de estados para que o usuário possa escolher em uma select (tag html) na camada view, qual o estado que nossa cidade que será cadastrada pertence.

Figura 112 – Implementando o método processRequest em CidadeCarregar

```

35     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
36         throws ServletException, IOException {
37     response.setContentType("text/html;charset=iso-8859-1");
38     try {
39         int idCidade = Integer.parseInt(request.getParameter("idCidade"));
40         //Cria objeto de CidadeDAO - camada dao
41         CidadeDAO oCidadeDAO = new CidadeDAO();
42         //cria variável no servidor para armazenar objeto de Cidade com os dados carregados
43         request.setAttribute("cidade", oCidadeDAO.carregar(idCidade));
44         //busca uma lista de estados para alimentar a caixa de seleção na view (jsp)
45         EstadoDAO oEstadoDAO = new EstadoDAO();
46         request.setAttribute("estados", oEstadoDAO.listar());
47         //dispacha os objetos de cidade e a lista de estados para a pagina jsp
48         request.getRequestDispatcher("/cadastros/cidade/cidadeCadastrar.jsp").forward(request, response);
49     } catch (Exception ex){
50         System.out.println("Erro carregar cidade " +ex.getMessage());
51         ex.printStackTrace();
52     }
53 }
```

Fonte: O autor.

7.2.3 Controller Cidade: Servlet CidadeListar

Vamos repetir os passos que fizemos anteriormente e criar nosso servlet CidadeListar em nosso pacote br.com.curso.controller.cidade, para isso clique com o botão direito sobre o pacote de controller de cidade e escolha a opção New → Servlet e crie o novo servlet como o nome de CidadeListar.

Uma vez criado o seu servlet CidadeListar vá até o método processRequest() apague o código gerado pelo Netbeans e implemente este método conforme a Figura 113.

Figura 113 – Implementando o método processRequest em CidadeListar

```

33     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
34         throws ServletException, IOException {
35     response.setContentType("text/html;charset=iso-8859-1");
36     try{
37         GenericDAO dao = new CidadeDAO();
38         request.setAttribute("cidades", dao.listar());
39         request.getRequestDispatcher("/cadastros/cidade/cidade.jsp").forward(request, response);
40     } catch (Exception ex){
41         //ex.printStackTrace();
42         System.out.println("Problemas no Servlet ao Listar"
43             + " Cidades! Erro: " + ex.getMessage());
44         ex.printStackTrace();
45     }
46 }
```

Fonte: O autor.

Nesta implementação estamos gerando uma lista de cidades sendo armazenadas na variável de contexto do servidor denominada “cidades” (utiliza-se o

nome no plural por ser uma lista de cidades) que será utilizada para gerar a visualização para o usuário no JSP na camada View.

7.2.4 Controller Cidade: Servlet CidadeCadastrar

Vamos repetir os passos que fizemos anteriormente e criar nosso servlet CidadeCadastrar em nosso pacote br.com.curso.controller.cidade, para isso clique com o botão direito sobre o pacote de controller de cidade e escolha a opção New → Servlet e crie o novo servlet como o nome de CidadeCadastrar.

Uma vez criado o seu servlet CidadeCadastrar vá até o método processRequest() apague o código gerado pelo Netbeans e implemente este método conforme a Figura 114.

Figura 114 – Implementando o método processRequest em CidadeCadastrar

```

36     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37         throws ServletException, IOException {
38         response.setContentType("text/html;charset=iso-8859-1");
39         int idCidade = Integer.parseInt(request.getParameter("idcidade"));
40         int idEstado = Integer.parseInt(request.getParameter("idestado"));
41         String nomeCidade = request.getParameter("nomecidade");
42         String situacao = request.getParameter("situacao");
43         String mensagem = null;
44
45         try{
46
47             Cidade oCidade = new Cidade();
48             oCidade.setIdCidade(idCidade);
49             oCidade.setNomeCidade(nomeCidade);
50             oCidade.setSituacao(situacao);
51             oCidade.setEstado(new Estado(idEstado, "", ""));
52
53             GenericDAO dao = new CidadeDAO();
54             if (dao.cadastrar(oCidade)){
55                 mensagem = "Cidade cadastrado com sucesso!";
56             } else {
57                 mensagem = "Problemas ao cadastrar Cidade.Verifique os dados informados e tente novamente!";
58             }
59             request.setAttribute("mensagem", mensagem);
60             response.sendRedirect("CidadeListar");
61         } catch (Exception ex){
62             System.out.println("Problemas no Servlet ao cadastrar Cidade! Erro: " + ex.getMessage());
63             ex.printStackTrace();
64         }
65     }

```

Fonte: O autor.

O servlet CidadeCadastrar recebe os dados enviados a partir do front-end armazenando-os em variáveis e instancia-se um objeto de cidade com as informações, devemos enfatizar a linha 51 onde é gerado um objeto de estado e este é agregado dentro do atributo estado no objeto de cidade.

7.2.5 Controller Cidade: Servlet CidadeExcluir

Vamos repetir os passos que fizemos anteriormente e criar nosso servlet CidadeExcluir em nosso pacote br.com.curso.controller.cidade, para isso clique com o botão direito sobre o pacote de controller de cidade e escolha a opção New → Servlet e crie o novo servlet como o nome de CidadeExcluir.

Uma vez criado o seu servlet CidadeExcluir vá até o método processRequest() apague o código gerado pelo Netbeans e implemente este método conforme a Figura 115.

Figura 115 – Implementando o método processRequest em CidadeExcluir



```

34 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35     throws ServletException, IOException {
36     response.setContentType("text/html; charset=iso-8859-1");
37     int idCidade = Integer.parseInt(request.getParameter("idCidade"));
38     String mensagem = null;
39     try {
40         GenericDAO dao = new CidadeDAO();
41         if (dao.excluir(idCidade)) {
42             mensagem = "Cidade excluido com Sucesso!";
43         } else {
44             mensagem = "Problemas ao excluir Cidade";
45         }
46         request.setAttribute("mensagem", mensagem);
47         response.sendRedirect("CidadeListar");
48     } catch (Exception ex) {
49         System.out.println("Problemas no Servelet ao excluir Cidade! Erro: "+ ex.getMessage());
50         ex.printStackTrace();
51     }
52 }
```

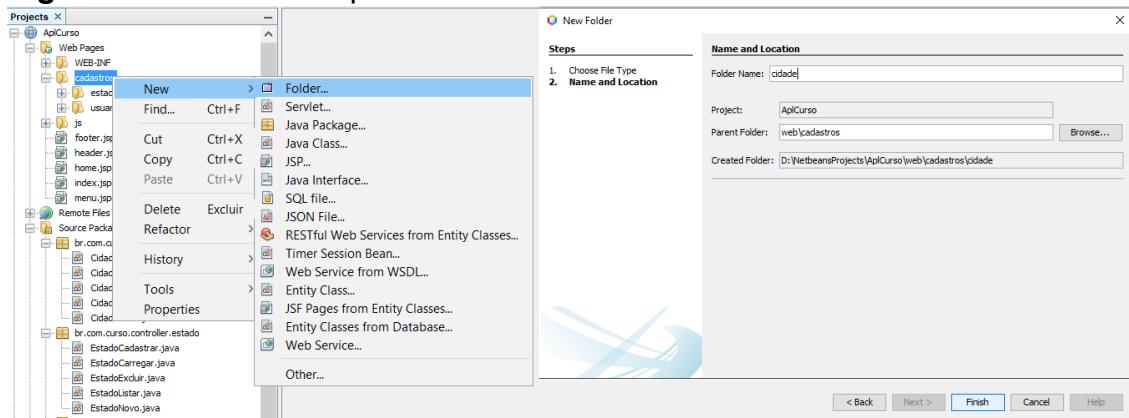
Fonte: O autor.

7.3 CRIANDO A CAMADA VIEW

Com o back-end de nossa aplicação de crud de cidades implementado, vamos começar a trabalhar nosso front-end. Iremos seguir os mesmos passos que efetuamos no cadastro de estado no Capítulo 5 desta obra.

Assim criaremos uma pasta para armazenar os jsp's referentes ao cadastro de cidade, para isso deve-se clicar com o botão direito na pasta “cadastros” que está no “Web Pages” e escolher as opções New → Folder. Na janela de criação da pasta informe o nome desejado “cidade” e conforme sua criação. O processo encontra-se representado na Figura 116.

Figura 116 – Criando a pasta de cidade na camada view

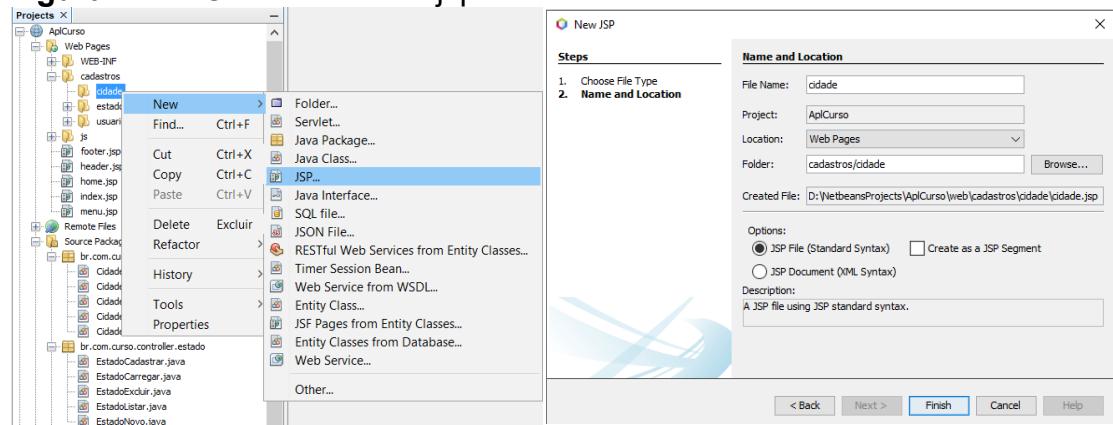


Fonte: O autor.

Uma vez criada a pasta cidade vamos criar nossos jsp's que serão dois, o primeiro cidade.jsp será responsável por exibir a lista de cidades cadastradas, além da função de exclusão e de permitir o acesso do usuário as funções de inclusão e alteração. Depois iremos criar o cidadeCadastrar.jsp que conterá o formulário para edição dos dados de cidade, permitindo sua inclusão ou alteração.

Para criarmos o cidade.jsp clique com o botão direito na pasta cidade que você acabou de criar e escolha a opção New→Jsp, e na janela de criação informe no nome do jsp “cidade”, este processo pode ser visualizado na Figura 117.

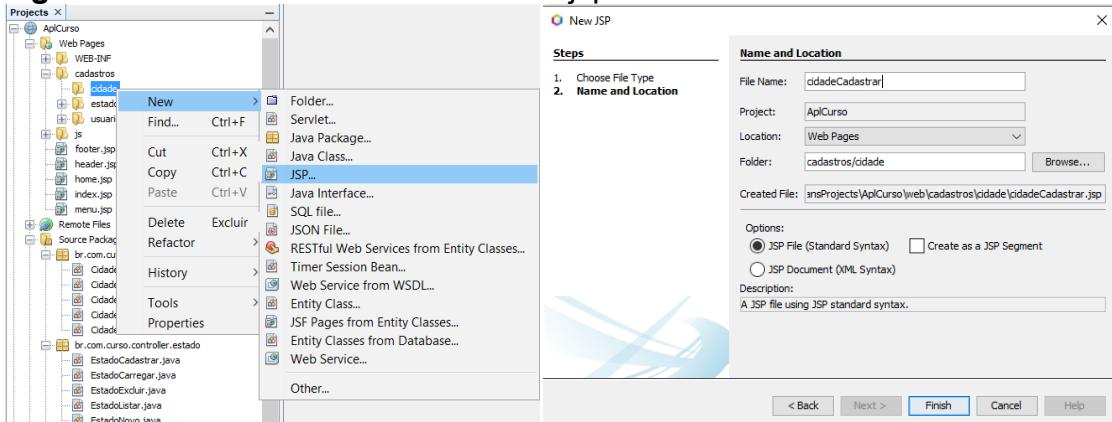
Figura 117 – Criando “cidade.jsp”



Fonte: O autor.

Agora vamos criar o cidadeCadastrar.jsp clique com o botão direito na pasta cidade que você acabou de criar e escolha a opção New→Jsp, e na janela de criação informe no nome do jsp “cidadeCadastrar”, este processo pode ser visualizado na Figura 118.

Figura 118 – Criando “cidadeCadastrar.jsp”



Fonte: O autor.

Uma vez criado nossos jsp's vamos iniciar o desenvolvimento de cada uma das páginas jsp.

7.3.1 Camada View: cidade.jsp

Vamos iniciar o desenvolvimento da interface da aplicação de cadastro de cidades com a tela que irá listar as cidades cadastradas para o usuário. Apague o código gerado pelo Netbeans em nosso arquivo cidade.jsp e codifique conforme pode-se observar na Figura 119 e Figura 120.

Figura 119 – Codificando a Página cidade.jsp (parte 1)

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt"%>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4 <jsp:include page="/header.jsp"/>
5 <jsp:include page="/menu.jsp"/>
6
7 <h2>Estados</h2>
8 <div class="col-8 panel-body">
9 <table id="datatable" class="table table-striped table-bordered basic-datatable">
10 <thead>
11 <tr>
12 <th align="left">ID</th>
13 <th align="left">Cidade</th>
14 <th align="left">Estado</th>
15 <th align="right"></th>
16 <th align="right"></th>
17 </tr>
18 </thead>
19 <tbody>
20 <c:forEach var="cidade" items="${cidades}">
21 <tr>
22 <td align="left">${cidade.idCidade}</td>
23 <td align="left">${cidade.nomeCidade}</td>
24 <td align="left">${cidade.estado.siglaEstado}</td>
25 <td align="center">
26 <a href="${pageContext.request.contextPath}/CidadeExcluir?idCidade=${cidade.idCidade}">
```

Fonte: O autor.

Figura 120 – Codificando a Página cidade.jsp (continuação)

```

27     <button class="btn btn-group-lg"
28           <c:out value="${cidade.situacao == 'A' ? 'btn-danger': 'btn-success'}"/>
29           <c:out value="${cidade.situacao == 'A' ? 'Inativar': 'Ativar'}"/>
30     </button>
31   </td>
32   <td align="center">
33     <a href="${pageContext.request.contextPath}/CidadeCarregar?idCidade=${cidade.idCidade}">
34       <button class="btn btn-group-lg btn-success"/>Alterar</button>
35     </a>
36   </td>
37 </tr>
38 </c:forEach>
39 </tbody>
40 </table>
41 </div>
42 <div align="center">
43   <a href="${pageContext.request.contextPath}/CidadeNovo">Novo</a>
44   <a href="index.jsp">Voltar à Página Inicial</a>
45 </div>
46
47
48 <script>
49   $(document).ready(function() {
50     console.log('entrei ready');
51     //Carregamos a datatable
52     $('#datatable').DataTable();
53     $('#datatable').DataTable({
54       "Language": {
55         "sProcessing": "Processando...",
56         "sLengthMenu": "Mostrar _MENU_ registros",
57         "sZeroRecords": "Nenhum registro encontrado.",
58         "sInfo": "Mostrando de _START_ até _END_ de _TOTAL_ registros",
59         "sInfoEmpty": "Mostrando de 0 até 0 de 0 registros",
60         "sInfoFiltered": "",
61         "sInfoPostFix": "",
62         "sSearch": "Buscar:",
63         "sUrl": "",
64         "oPaginate": {
65           "sFirst": "Primeiro",
66           "sPrevious": "Anterior",
67           "sNext": "Seguinte",
68           "sLast": "Último"
69         }
70       }
71     });
72   });
73 </script>
74
75 <%@ include file="/footer.jsp" %>

```

Fonte: O autor.

7.3.2 Camada View: cidadeCadastrar.jsp

Agora vamos realizar o desenvolvimento da interface da aplicação de cadastro de cidades na tela que irá efetuar a manutenção das cidades cadastradas para o usuário. Apague o código gerado pelo Netbeans em nosso arquivo cidadeCadastrar.jsp e codifique conforme pode-se observar na Figura 121.

Observe as linhas de código 26 a 34 que representam a montagem da Html tag Select para a seleção de estados.

Figura 121 – Codificando a Página cidadeCadastrar.jsp

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
3 <jsp:include page="/header.jsp"/>
4 <jsp:include page="/menu.jsp"/>
5
6 <form name="cadastrarcidade" action="CidadeCadastrar" method="POST">
7   <table align="center" border="0">
8     <thead>
9       <tr>
10      <th colspan="2" align="center">
11        Cadastro de Cidade</th>
12      </tr>
13      <tr>
14        <th colspan="2" align="center">${mensagem}</th>
15      </tr>
16    </thead>
17    <tbody>
18      <tr><td>ID: </td>
19      <td><input type="text" name="idcidade" id="idcidade" value="${cidade.idCidade}" readonly="readonly" /></td></tr>
20      <tr><td>Nome: </td>
21      <td><input type="text" name="nomecidade" id="nomecidade" value="${cidade.nomeCidade}" size="50" maxlength="50" /></td></tr>
22
23      <tr>
24        <td>Estado: </td>
25        <td>
26          <select name="idestado" id="ideстado">
27            <option value="">Selecione</option>
28            <c:forEach var="estado" items="${estados}">
29              <option value="${estado.idEstado}" ${cidade.estado.idEstado == estado.idEstado ? "selected" : ""}>
30                ${estado.nomeEstado}
31              </option>
32            </c:forEach>
33          </select>
34        </td>
35      </tr>
36      <tr><td>
37        <input type="hidden" name="situacao" id="situacao" value="${cidade.situacao}" readonly="readonly" />
38      </td></tr>
39      <tr><td colspan="2" align="center">
40        <input type="submit" name="cadastrar" id="cadastrar" value="Cadastrar" />
41        <input type="reset" name="limpar" id="limpar" value="Limpar" />
42      </td>
43      </tr>
44      <tr>
45        <td align="center" colspan="2"><h5><a href="index.jsp">Voltar à Página Inicial</a></h5></td>
46      </tr>
47    </tbody>
48  </table>
49 </form>
50 <%@ include file="/footer.jsp" %>
51
52

```

Fonte: O autor.

7.3.3 Camada View: menu.jsp

Não se esqueça de alterar o menu.jsp para incluir a opção de acesso ao cadastro de cidades.

Figura 122 – Alterando o menu.jsp

```

1 <h1>Módulo Cadastros</h1>
2 <hr>
3   <center>
4     <h2>Menu Principal</h2>
5     <a href="${pageContext.request.contextPath}/EstadoListar">Estado</a>
6     <a href="${pageContext.request.contextPath}/CidadeListar">Cidade</a>
7     <a href="${pageContext.request.contextPath}/UsuarioListar">Usuario</a>
8   </center>
9 <hr>

```

Fonte: O autor.

7.3.4 Telas Desenvolvidas

Ao final do processo de programação temos as telas de cidade desenvolvidas conforme pode-se observar na Figura 123.

Figura 123 – Telas de Cadastro de Cidades
Módulo Cadastros

The figure consists of two screenshots of a Java Web application. The top screenshot shows a table of cities with columns for ID, City, State, Inativar (Inactivate), and Alterar (Alter). The bottom screenshot shows a form for registering a new city with fields for ID, Name, and State, along with buttons for Cadastrar (Register) and Limpar (Clear).

Menu Principal
Estado Cidade Usuario

Estados

ID	Cidade	Estado	Inativar	Alterar
1	Fernandopolis	SP	Inativar	Alterar
2	Jales	SP	Inativar	Alterar
6	Votuporanga	SP	Inativar	Alterar
7	Santa Fé do Sul	SP	Inativar	Alterar

Mostrando de 1 até 4 de 4 registros

Anterior 1 Seguinte
Novo Voltar à Página Inicial

Desenvolvendo Aplicações com Java Web

Módulo Cadastros

Menu Principal
Estado Cidade Usuario

Cadastro de Cidade

ID:	<input type="text" value="1"/>
Nome:	<input type="text" value="Fernandopolis"/>
Estado:	<input type="text" value="São Paulo"/>

[Voltar à Página Inicial](#)

Desenvolvendo Aplicações com Java Web

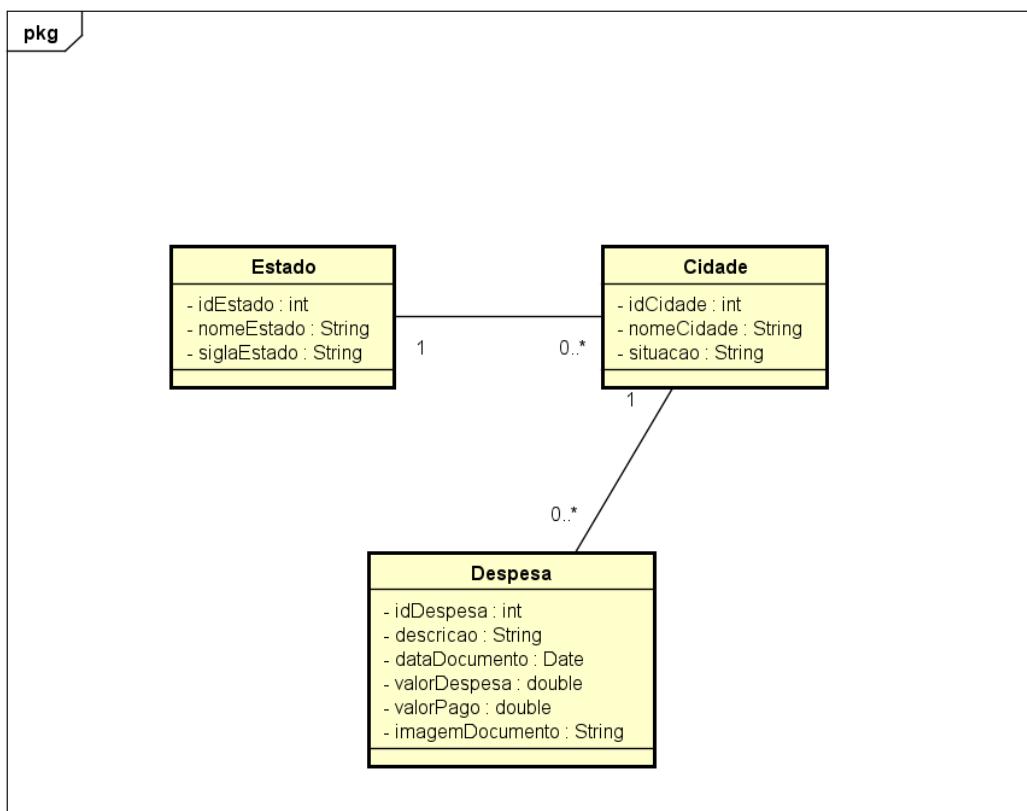
Fonte: O autor;

CAPÍTULO 8 – DESENVOLVIMENTO DE CRUD COM AJAX

Neste capítulo de nosso curso vamos avançar bastante em como fazemos nossos CRUDs, agora iremos manipular campos do tipo data e de imagem, iremos também realizar novas melhorias em nossa interface com o usuário adicionando recursos de “modal” aos nossos formulários, utilização de Ajax e logicamente aprender a manipular as imagens na cada web de nossa aplicação.

Agora vamos desenvolver a classe Despesa que possui uma relação de multiplicidade 1..* com a classe Cidade.

Figura 124 – Diagrama de Classe



Fonte: O autor

Observando o diagrama de classe vamos criar nossa tabela no banco de dados. O código SQL de nossa tabela Despesa ficará conforme está descrito na Figura 125. Não se esqueça que devemos estabelecer na tabela de despesas a relação de cardinalidade 1-N com a tabela de cidades.

Figura 125 – Código SQL Tab. Despesa

```

create table despesa(
    iddespesa serial primary key,
    descricao varchar(100) not null,
    datadocumento date not null,
    valordespesa numeric(15,2) not null,
    valorpago numeric(15,2),
    imagemdocumento text
);

insert into despesa(descricao,datadocumento,valordespesa,valorpago)
values ('descricao','2021-08-23',20.5,10.5);

```

Fonte: O autor

Neste código sql podemos observar o campo datadocumento do tipo “Date” onde serão armazenados valores do tipo de data e é também importante observar o campo imagemdocumento do tipo “Text” que será utilizado para armazenar a imagem do documento a pagar.

As imagens serão armazenadas dentro do banco de dados em formato Base64. Esse formato é um método para codificação de dados para transferência na internet, sendo utilizado frequentemente para transmitir dados binários por meios de transmissão que lidam apenas com texto, como é nosso caso que utilizaremos requisições HTTP via AJAX.

É possível converter qualquer informação neste formato “base64” como imagens, arquivos de planilhas, pdfs etc.

É importante também conhecermos a definição de AJAX que é o acrônimo de JavaScript assíncrono + XML, essa não é uma tecnologia nova, mas um termo empregado em 2005 por Jesse Garrett para descrever uma nova forma de utilizar em conjunto com algumas tecnologias, incluindo HTML, XHTML, CSS, javascript, DOM, XML, XSLT e o mais importante: objeto XMLHttpRequest.

Quanto essas tecnologias são combinadas no modelo Ajax as aplicações web que a utilizam são capazes de fazer rapidamente atualizações incrementais para a interface do usuário sem recarregar a página inteira do navegador. Isso torna a aplicação mais rápida e sensível as ações do usuário.

Embora a letra X em AJAX corresponda a XML, atualmente o JSON é mais utilizado que o XML devido suas vantagens, como ser mais leve e ser parte do

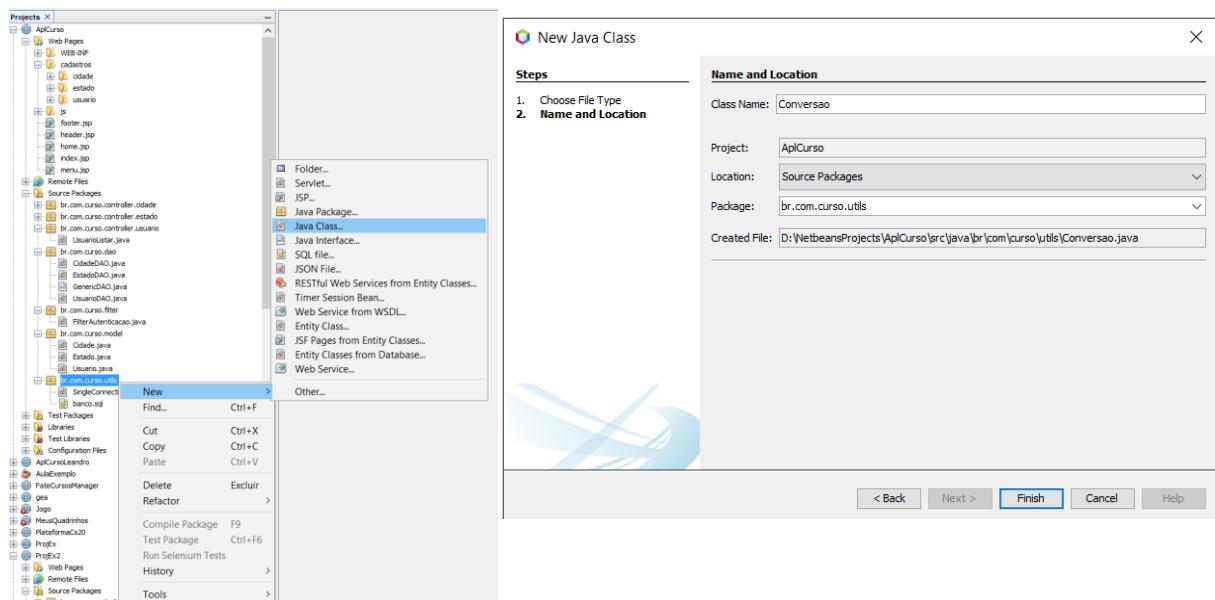
javascript. Ambos (Json e XML) são utilizados para obter informações do pacote no modelo AJAX. Neste capítulo vocês aprenderá como utilizar as requisições assíncronas AJAX em seus cadastros JAVA(JSP).

8.1 FUNÇÕES AUXILIARES DE MANIPULAÇÃO DE DADOS

Neste capítulo iremos desenvolver algumas funções na camada Utils de nosso projeto que nos auxiliarão na manipulação de dados em formato de data e também em formato de moeda.

Para isso vamos criar uma nova classe java denominada “Conversao”, assim clique com o botão direito na sua camada utils “br.com.curso.utils” e escolha a opção “Novo → Classe Java” ou em inglês “New → Java Class” e crie a classe na próxima janela como já estamos acostumados fazer em nosso curso.

Figura 126 – Criando a classe “Conversao”



Fonte: O autor

Agora vamos começar implementar os métodos em nossa classe “Conversao” que nos auxiliarão em nossos trabalhos com datas e moedas. Vamos começar com os métodos de manipulação de datas. Vocês irão perceber que os métodos criados serão do tipo “Static”.

Os métodos do tipo “Static” em Java nos permitem executá-los diretamente da classe sem a necessidade de gerar uma instância dela.

O método “converterData” recebe uma data em formato String e retorna uma data em formato Date, enquanto o método “data2String” recebe a data em formato Date e retorna em String.

O método “dataAtual” como seu próprio nome faz referência nos retorna a data atual em seu sistema.

Digite os métodos no seu projeto como demonstrado na Figura 128.

Figura 127 – Criando os métodos de manipulação de datas

```

8  import java.text.ParseException;
9  import java.text.SimpleDateFormat;
10 import java.util.Date;
11
12 public class Conversao {
13     public static Date converterData(String data) throws ParseException{
14         SimpleDateFormat fmt = new SimpleDateFormat("yyyy-MM-dd");
15         if(data == null || data.trim().equals("")){
16             return null;
17         } else{
18             Date date = fmt.parse(data);
19             return date;
20         }
21     }
22
23     public static String data2String(Date data){
24         SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
25         String dataFormatada = fmt.format(data);
26         return dataFormatada;
27     }
28
29     public static Date dataAtual() {
30         SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
31         Date novaData = new Date(System.currentTimeMillis());
32         return novaData;
33     }
34 }
```

Fonte: O autor

Agora verifique as importações em seu projeto e complemente com as faltantes para continuarmos com a implementação dos métodos de moeda.

Figura 128 – Importações da Classe Conversao

```

6  package br.com.curso.utils;
7
8  import java.text.NumberFormat;
9  import java.text.ParseException;
10 import java.text.SimpleDateFormat;
11 import java.util.Date;
12 import java.util.Locale;
```

Fonte: O autor

Na Figura 129 temos os métodos “valorDinheiro(String valor)” que recebe em sua assinatura apenas uma String faz a conversão uma String em um valor double e o método “valorDinheiro(double valor, String pais)” que recebe um valor double e uma String faz a conversão de um valor double para uma String colocando o padrão monetário brasileiro ou americano.

Figura 129 – Implementando os métodos de manipulação de moeda.

```

37  public static double valorDinheiro(String valor){
38      String conversao = valor.substring(2, valor.length());
39      conversao = conversao.replaceAll("[./-]", "");
40      conversao = conversao.replace(",", ".").trim();
41      return Double.parseDouble(conversao);
42  }
43
44  public static String valorDinheiro(double valor, String pais){
45      NumberFormat formatter = null;
46      if (pais.equals("BR")){
47          formatter = NumberFormat.getCurrencyInstance();
48      } else if (pais.equals("US")){
49          formatter = NumberFormat.getInstance(new Locale("en", "US"));
50      }
51      String moneyString = formatter.format(valor);
52      return moneyString;
53  }
54 }
```

Fonte: O autor

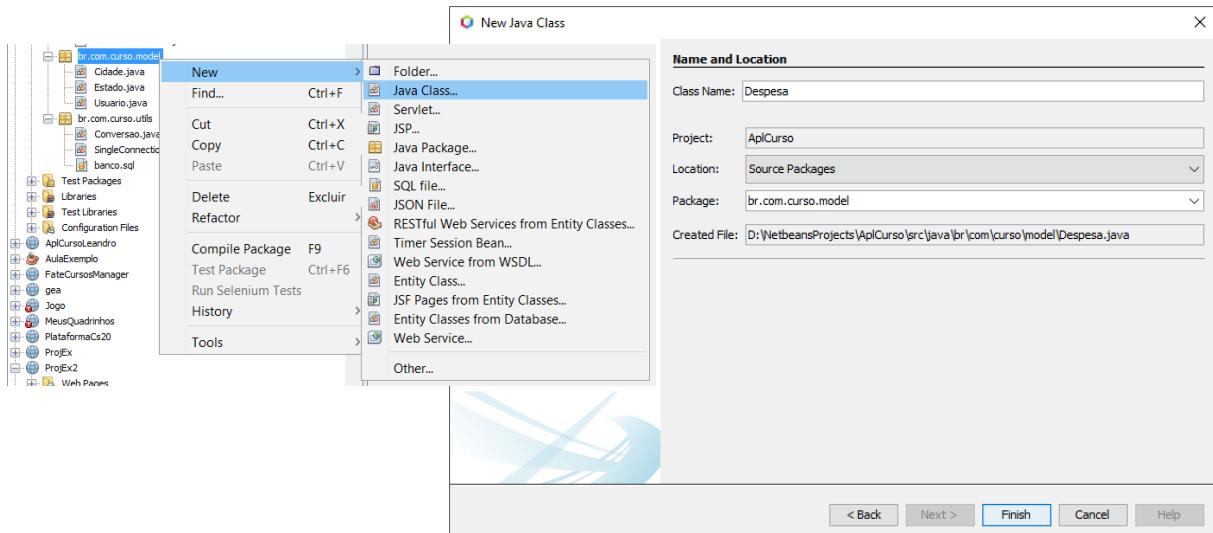
8.2 DESENVOLVENDO A CAMADA MODEL

Agora vamos desenvolver nossa camada model para isso vamos criar a classe java “Despesa” conforme o modelo de dados definido no início deste capítulo.

Para isso clique com o botão direito em seu pacote da camada model em seu projeto, “br.com.curso.model”.

Escolha a opção “New → Java Class” e na janela coloque o nome da classe “Despesa” e confirme sua criação, como podemos observar na Figura 130.

Figura 130 – Camada Model – Criando a classe Despesa.



Fonte: O autor

Agora vamos criar nossos atributos na classe Despesa.

Figura 131 – Camada Model – Criando a classe Despesa.

```

6  package br.com.curso.model;
7
8  import java.util.Date;
9
10 public class Despesa {
11
12     private int idDespesa;
13     private String descricao;
14     private Date dataDocumento;
15     private double valorDespesa;
16     private double valorPago;
17     private String imagemDocumento;
18
19 }
```

Fonte: O autor

Quando vamos trabalhar com os dados do tipo data devemos importar a classe “java.util.Date” que nos permite manipular essas informações, tome cuidado pois

existem outras classes que também trabalham com datas e possuem nomes parecidos como a “java.sql.Date”.

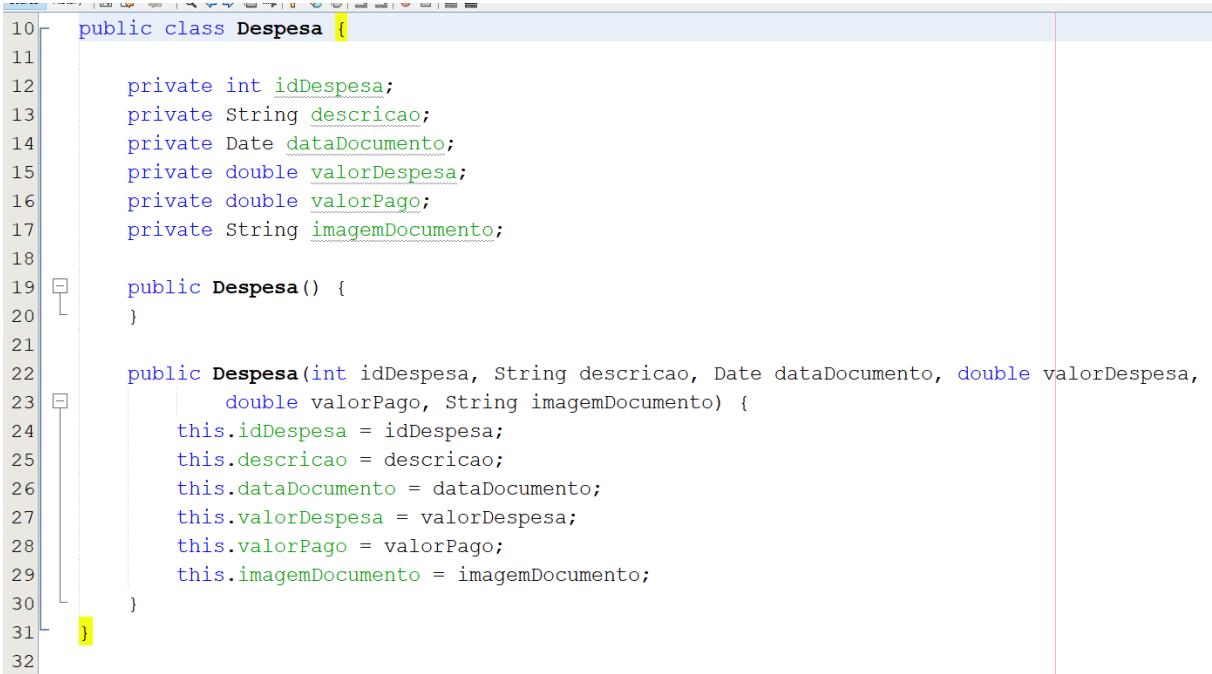
Perceba também que nosso atributo “imagemDocumento” que possui a função de armazenar uma imagem do documento que está sendo lançado em nosso sistema é do tipo String.

Isso ocorre pois quando nosso projeto transmitir uma imagem selecionada pelo usuário no *front-end* da aplicação, ela será convertida para um formato de dados denominado **base64** que é a representação da imagem em formato String.

O Formato base64 por ser uma String é facilmente transferida através do protocolo HTTP de nosso projeto e através deste formato podemos transferir qualquer tipo de dados, imagens (JPG, PNG, BMP, etc), arquivos do office (DOCX, XLSX, PPTX, etc), arquivos do tipo PDF entre outros.

Dando continuidade a nossa camada model vamos criar os métodos construtores, para isso gere os métodos através dos atalhos de sua IDE. Gere tanto o método construtor vazio, assim como o método com parâmetros para preenchimento do objeto gerado (Figura 132).

Figura 132 – Camada Model – Gerando os métodos construtores.



```

10  public class Despesa {
11
12      private int idDespesa;
13      private String descricao;
14      private Date dataDocumento;
15      private double valorDespesa;
16      private double valorPago;
17      private String imagemDocumento;
18
19      public Despesa() {
20      }
21
22      public Despesa(int idDespesa, String descricao, Date dataDocumento, double valorDespesa,
23                     double valorPago, String imagemDocumento) {
24          this.idDespesa = idDespesa;
25          this.descricao = descricao;
26          this.dataDocumento = dataDocumento;
27          this.valorDespesa = valorDespesa;
28          this.valorPago = valorPago;
29          this.imagemDocumento = imagemDocumento;
30      }
31
32  }

```

Fonte: O autor

Após gerado os métodos vamos alterar o método construtor vazio (linhas 19 e 20 da Figura 132) para quando criar o objeto de despesa os atributos possuam algumas informações básicas necessárias ao funcionamento correto do projeto.

Deste modo você deve fazer como demonstrado na Figura 133.

Figura 133 – Camada Model – Alterando o método construtor.

```

19 public Despesa() {
20     idDespesa = 0;
21     descricao = "";
22     valorDespesa = 0;
23     valorPago = 0;
24     dataDocumento = Conversao.dataAtual();
25 }

```

Fonte: O autor

Na linha 24 estamos utilizando o método dataAtual() da classe Conversão que criamos anteriormente para isso faça a importação da mesma.

Figura 134 – Camada Model – Importando classe Conversão

```

19 public Despesa() {
20     idDespesa = 0;
21     descricao = "";
22     valorDespesa = 0;
23     valorPago = 0;
24     dataDocumento = Conversao.dataAtual();
25 }

```

Add import for br.com.curso.utils.Conversao
 Create Class "Conversao" in package br.com.curso.model (Source Packages)
 Create class "Conversao" in br.com.curso.model.Despesa
 Create field "Conversao" in br.com.curso.model.Despesa

Fonte: O autor

Analisando ainda a linha 24 de nosso projeto repare que para utilizarmos o método Conversao.dataAtual() não precisamos instanciar um objeto da classe Conversão isso se deve aos nossos métodos nesta classe serem do tipo “Static”. Nossa camada model de Despesa está assim até o momento.

Figura 135 – Camada Model – Classe Despesa (Construtores)

```

11 public class Despesa {
12     private int idDespesa;
13     private String descricao;
14     private Date dataDocumento;
15     private double valorDespesa;
16     private double valorPago;
17     private String imagemDocumento;
18
19     public Despesa() {
20         idDespesa = 0;
21         descricao = "";
22         valorDespesa = 0;
23         valorPago = 0;
24         dataDocumento = Conversao.dataAtual();
25     }
26
27     public Despesa(int idDespesa, String descricao, Date dataDocumento, double valorDespesa,
28                     double valorPago, String imagemDocumento) {
29         this.idDespesa = idDespesa;
30         this.descricao = descricao;
31         this.dataDocumento = dataDocumento;
32         this.valorDespesa = valorDespesa;
33         this.valorPago = valorPago;
34         this.imagemDocumento = imagemDocumento;
35     }
36 }

```

Fonte: O autor

Agora através de sua IDE (Netbeans) faça a geração dos métodos “*Getter and Setter*” necessários para o correto funcionamento de nossa classe Despesa. Você pode visualizar na Figura 136 a classe Despesa completa.

Figura 136 – Camada Model – Classe Despesa

```

6  package br.com.curso.model;
7
8  import br.com.curso.utils.Conversao;
9  import java.util.Date;
10
11 public class Despesa {
12     private int idDespesa;
13     private String descricao;
14     private Date dataDocumento;
15     private double valorDespesa;
16     private double valorPago;
17     private String imagemDocumento;
18
19     public Despesa() {
20         idDespesa = 0;
21         descricao = "";
22         valorDespesa = 0;
23         valorPago = 0;
24         dataDocumento = Conversao.dataAtual();
25     }
26
27     public Despesa(int idDespesa, String descricao, Date dataDocumento, double valorDespesa,
28         double valorPago, String imagemDocumento) {
29         this.idDespesa = idDespesa;
30         this.descricao = descricao;
31         this.dataDocumento = dataDocumento;
32         this.valorDespesa = valorDespesa;
33         this.valorPago = valorPago;
34         this.imagemDocumento = imagemDocumento;
35     }
36
37     public int getIdDespesa() {
38         return idDespesa;
39     }
40
41     public void setIdDespesa(int idDespesa) {
42         this.idDespesa = idDespesa;
43     }
44
45     public String getDescricao() {
46         return descricao;
47     }
48
49     public void setDescricao(String descricao) {
50         this.descricao = descricao;
51     }
52
53     public Date getDataDocumento() {
54         return dataDocumento;
55     }
56
57     public void setDataDocumento(Date dataDocumento) {
58         this.dataDocumento = dataDocumento;
59     }
60
61     public double getValorDespesa() {
62         return valorDespesa;
63     }
64
65     public void setValorDespesa(double valorDespesa) {
66         this.valorDespesa = valorDespesa;
67     }
68
69     public double getValorPago() {
70         return valorPago;
71     }
72
73     public void setValorPago(double valorPago) {
74         this.valorPago = valorPago;
75     }
76
77     public String getImagenDocumento() {
78         return imagemDocumento;
79     }
80
81     public void setImagenDocumento(String imagemDocumento) {
82         this.imagemDocumento = imagemDocumento;
83     }
84

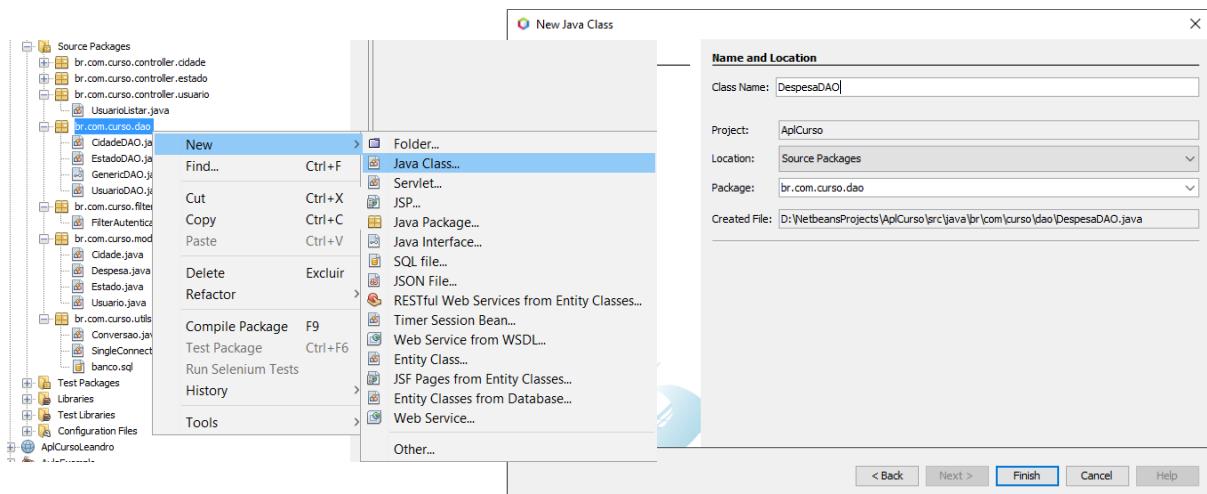
```

Fonte: O autor

8.3 DESENVOLVENDO A CAMADA DAO

Agora vamos criar nossa classe DespesaDAO em nosso pacote de camada DAO no nosso projeto. Para isso clique com o botão direito sobre “br.com.curso.dao” e escolha “New → Java Class” e na janela de criação informe no nome da classe “DespesaDAO” e confirme.

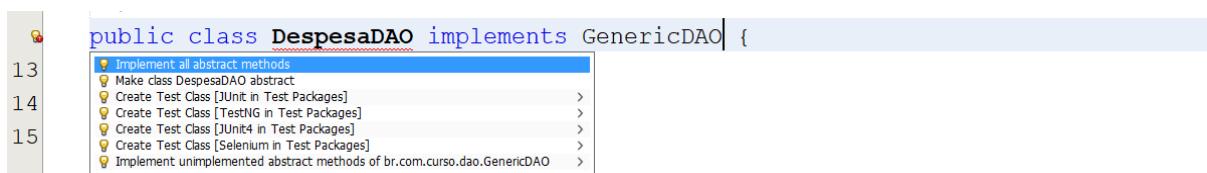
Figura 137 – Camada DAO – Classe DespesaDAO



Fonte: O autor

Nossa classe DespesaDAO como já é padrão em nosso projeto irá implementar os métodos declarados na Interface GenericDAO, assim altere a declaração da classe como demonstrado na Figura 138.

Figura 138 – Camada DAO – Classe DespesaDAO – Implementando GenericDAO



Fonte: O autor

No momento que você informar para a classe DespesaDAO implementar a GenericDAO o Netbeans irá acusar erro que pode ser corrigido escolhendo na Dica da IDE a opção “Implements All abstract Methods”, como demonstrado na Figura 138.

Agora vamos criar o atributo que irá armazenar nossa conexão, como já realizado em outras classes de camada DAO já desenvolvidas nesse projeto. Implemente em seu projeto conforme a linha 17 demonstrada na Figura 139.

Figura 139 – Camada DAO – Classe DespesaDAO – Atributo “conexão”.

```

15 public class DespesaDAO implements GenericDAO {
16
17     private Connection conexao;
18
19     @Override
20     public Boolean cadastrar(Object objeto) {
21         throw new UnsupportedOperationException("Not supported yet."); //To change
22     }

```

Fonte: O autor

Vamos também criar nosso método construtor para nossa classe DespesaDAO, implemente o método conforme descrito nas linhas 20 a 22 na Figura 140. Você deve se atentar também para a importação de nossa classe SingleConnection que será solicitada e ficará conforme demonstrado na linha 8.

Figura 140 – Camada DAO – Classe DespesaDAO – Método Construtor.

```

8  import br.com.curso.utils.SingleConnection;
9  import java.sql.Connection;
10 import java.util.List;
11
12 /**
13 * @author jeffe
14 */
15 public class DespesaDAO implements GenericDAO {
16
17     private Connection conexao;
18
19     public DespesaDAO() throws Exception{
20         conexao = SingleConnection.getConnection();
21     }
22
23     @Override

```

Fonte: O autor

Após estarmos com o método construtor implementado vamos iniciar a implementação dos métodos sobrescritos da interface GenericDAO (@override).

Vamos iniciar pelo método cadastrar, para isso você deve substituir as linhas do método em seu projeto conforme visualiza-se na Figura 141 nas linhas de código 26 a 35.

Figura 141 – Camada DAO – Classe DespesaDAO – Método Cadastrar.

```

21  public DespesaDAO() throws Exception{
22      conexao = SingleConnection.getConnection();
23  }
24
25  @Override
26  public Boolean cadastrar(Object objeto) {
27      Despesa oDespesa = (Despesa) objeto;
28      boolean retorno = false;
29      if(oDespesa.getIdDespesa() == 0){
30          retorno = inserir(oDespesa);
31      }else{
32          retorno = alterar(oDespesa);
33      }
34      return retorno;
35  }

```

Fonte: O autor

Agora vamos implementar o método inserir, conforme a Figura 142.

Figura 142 – Camada DAO – Classe DespesaDAO – Método Inserir.

```

41  @Override
42  public Boolean inserir(Object objeto) {
43      Despesa oDespesa = (Despesa) objeto;
44      PreparedStatement stmt = null;
45      String sql = "Insert Into Despesa (descricao, valorDespesa, valorPago,"
46                  + "datadocumento, imagemdocumento) values (?, ?, ?, ?, ?)";
47      try {
48          stmt = conexao.prepareStatement(sql);
49          stmt.setString(1, oDespesa.getDescricao());
50          stmt.setDouble(2, oDespesa.getValorDespesa());
51          stmt.setDouble(3, oDespesa.getValorPago());
52          stmt.setDate(4, new java.sql.Date(oDespesa.getDataDocumento().getTime()));
53          stmt.setString(5, oDespesa.getImagenDocumento());
54          stmt.execute();
55          conexao.commit();
56          return true;
57      } catch (Exception e){
58          try {
59              System.out.println("Problemas ao cadastrar Despesa!Erro: " + e.getMessage());
60              e.printStackTrace();
61              conexao.rollback();
62          } catch (SQLException ex) {
63              System.out.println("Problemas ao executar rollback" + ex.getMessage());
64              ex.printStackTrace();
65          }
66      }
67  }
68

```

Fonte: O autor

Na linha de código 52 da Figura 142 temos a implementação da gravação de uma data no banco de dados, repare que utilizamos neste momento a classe "java.sql.Date" pois precisamos da conversão da data para um formato em que nosso SGBD consiga entender. Quando precisar gravar uma data está a forma de fazer.

Na linha 53 você pode observar como vamos gravar a imagem do documento de despesa no banco de dados. Isso ocorre como uma gravação de uma String qualquer.

Realizando os mesmos passos você pode observar a implementação do método alterar na Figura 143.

Figura 143 – Camada DAO – Classe DespesaDAO – Método Alterar.

```

70     @Override
71     public Boolean alterar(Object objeto) {
72         Despesa oDespesa = (Despesa) objeto;
73         PreparedStatement stmt = null;
74         String sql = "update despesa set descricao=?, valorDespesa=?, valorPago=?," +
75             " datadocumento=?, imagemdocumento=? where iddespesa=?";
76         try {
77             stmt = conexao.prepareStatement(sql);
78             stmt.setString(1, oDespesa.getDescricao());
79             stmt.setDouble(2, oDespesa.getValorDespesa());
80             stmt.setDouble(3, oDespesa.getValorPago());
81             stmt.setDate(4, new java.sql.Date(oDespesa.getDataDocumento().getTime()));
82             stmt.setString(5, oDespesa.getImagenDocumento());
83             stmt.setInt(6, oDespesa.getIdDespesa());
84             stmt.execute();
85             conexao.commit();
86             return true;
87         } catch (Exception e) {
88             try {
89                 System.out.println("Problemas ao alterar Despesa!Erro: " + e.getMessage());
90                 e.printStackTrace();
91                 conexao.rollback();
92             } catch (SQLException ex) {
93                 System.out.println("Problemas ao executar rollback" + ex.getMessage());
94                 ex.printStackTrace();
95             }
96         }
97     }
98 }
```

Fonte: O autor

Vamos implementar agora o método excluir() de acordo com o demonstrado na Figura 144.

Figura 144 – Camada DAO – Classe DespesaDAO – Método Excluir.

```

100  @Override
101  public Boolean excluir(int numero) {
102      int idDespesa = numero;
103      PreparedStatement stmt = null;
104      String sql = "delete from despesa where iddespesa=?";
105      try {
106          stmt =conexao.prepareStatement(sql);
107          stmt.setInt(1, idDespesa);
108          stmt.execute();
109          conexao.commit();
110          return true;
111      } catch (Exception e){
112          try {
113              System.out.println("Problemas ao excluir Despesa!Erro: " + e.getMessage());
114              e.printStackTrace();
115              conexao.rollback();
116          } catch (SQLException ex) {
117              System.out.println("Problemas ao executar rollback" + ex.getMessage());
118              ex.printStackTrace();
119          }
120      }
121  }
122 }
```

Fonte: O autor

Iremos implementar a seguir o método carregar, implemente conforme demonstrado na Figura 145.

Figura 145 – Camada DAO – Classe DespesaDAO – Método Carregar.

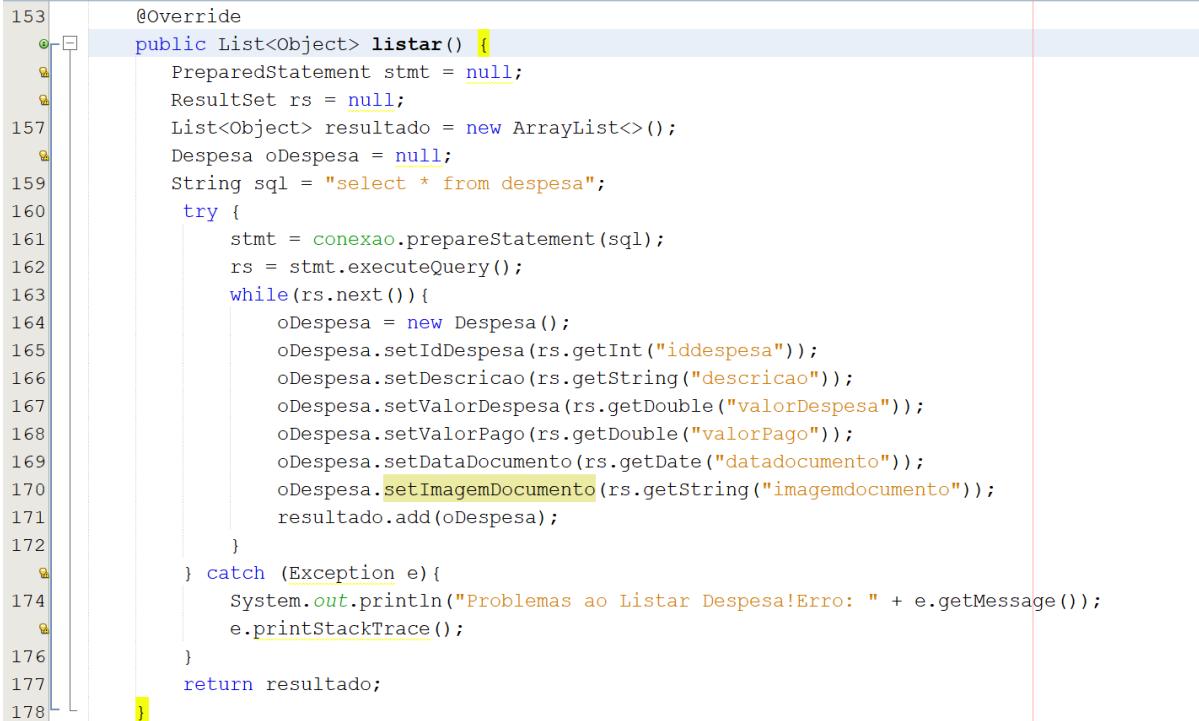
```

125  @Override
126  public Object carregar(int numero) {
127      int idDespesa = numero;
128      PreparedStatement stmt = null;
129      ResultSet rs = null;
130      Despesa oDespesa = null;
131      String sql = "Select * From despesa Where idDespesa = ?";
132      try {
133          stmt =conexao.prepareStatement(sql);
134          stmt.setInt(1, idDespesa);
135          rs = stmt.executeQuery();
136          while(rs.next()){
137              oDespesa = new Despesa();
138              oDespesa.setIdDespesa(rs.getInt("idDespesa"));
139              oDespesa.setDescricao(rs.getString("descricao"));
140              oDespesa.setValorDespesa(rs.getDouble("valorDespesa"));
141              oDespesa.setValorPago(rs.getDouble("valorPago"));
142              oDespesa.setDataDocumento(rs.getDate("datadocumento"));
143              oDespesa.setImageDocumento(rs.getString("imagedocumento"));
144          }
145      } catch (Exception e) {
146          System.out.println("Problemas ao carregar despesa!Erro: " + e.getMessage());
147          e.printStackTrace();
148      }
149      return oDespesa;
150  }
```

Fonte: O autor

E vamos implementar o último dos métodos sobrescritos de nossa interface GenericDAO o método listar(). Implemente em seu projeto conforme demonstrado na Figura 146.

Figura 146 – Camada DAO – Classe DespesaDAO – Método listar.



```

153     @Override
154     public List<Object> listar() {
155         PreparedStatement stmt = null;
156         ResultSet rs = null;
157         List<Object> resultado = new ArrayList<>();
158         Despesa oDespesa = null;
159         String sql = "select * from despesa";
160         try {
161             stmt = conexao.prepareStatement(sql);
162             rs = stmt.executeQuery();
163             while(rs.next()){
164                 oDespesa = new Despesa();
165                 oDespesa.setIdDespesa(rs.getInt("iddespesa"));
166                 oDespesa.setDescricao(rs.getString("descricao"));
167                 oDespesa.setValorDespesa(rs.getDouble("valorDespesa"));
168                 oDespesa.setValorPago(rs.getDouble("valorPago"));
169                 oDespesa.setDataDocumento(rs.getDate("datadocumento"));
170                 oDespesa.setImageDocumento(rs.getString("imagemdocumento"));
171                 resultado.add(oDespesa);
172             }
173         } catch (Exception e){
174             System.out.println("Problemas ao Listar Despesa!Erro: " + e.getMessage());
175             e.printStackTrace();
176         }
177         return resultado;
178     }

```

Fonte: O autor

Agora vamos criar um método listar que trabalho com JSON. Para implementar esse método vamos ter que importar em nosso projeto a biblioteca que nos permite trabalhar com JSON em nosso projeto.

8.3.1 Trabalhando com JSON

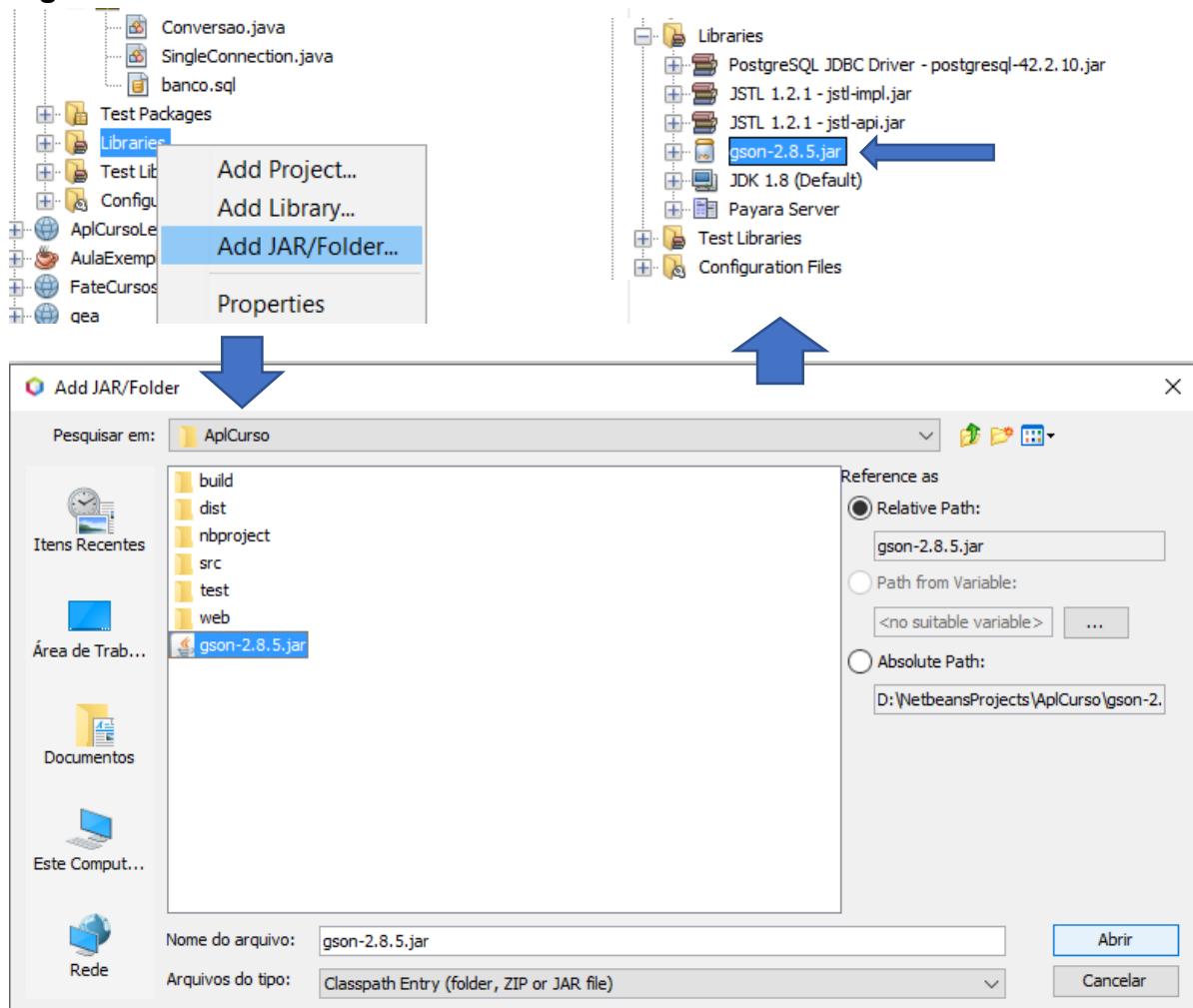
Para facilitar nosso trabalho com JSON vamos adicionar a biblioteca **gson-2.8.5.jar** que foi fornecida junto com esse tutorial na pasta de seu projeto (Netbeans).

A biblioteca gson-2.8.5.jar foi criada pelo Google e nos permite converter objecto de nosso projeto automaticamente em JSON, assim como listas (ArrayLists) facilitando o trabalho de programação.

Após colocar a biblioteca gson-2.8.5.jar na pasta de seu projeto, vamos importar a biblioteca para nosso projeto. Para isso clique com o botão direito na pasta

Libraries de seu projeto e escolha a opção “Add JAR/Folder” e na janela vá até a pasta de seu projeto e escolha o arquivo gson-2.8.5.jar e clique em abrir para confirmar sua importação.

Figura 147 – Adicionando biblioteca GSON.



Fonte: O autor

JSON significa *JavaScript Object Notation*, é uma representação de dados no formato de um objeto javascript. Hoje é muito utilizado quando precisamos consumir dados via API por exemplo. Mas você pode utilizá-lo em seu projeto e o seu conteúdo será sempre texto apenas.

Uma string JSON pode se construído de duas formas, conforme demonstrado na Figura 148.

A primeira forma representa uma coleção de dados no formato Chave-Valor e geralmente é a representação dos dados de um objeto de seu sistema.

O segundo formato representa um Array ou seja é uma coleção de objetos como temos em ArrayList em java. Assim esse formato é a conversão de um ArrayList de objetos de nosso sistema.

Figura 148 – Exemplos de Estrutura JSON

1. Uma coleção de dados representados no formato Chave-Valor

```
{
  "cliente": {
    "nome": "Joao da Silva",
    "telefone": "9999999"
  }
}
```

2. Array

```
{
  "clientes": [
    {
      "cliente": {
        "nome": "Joao da Silva",
        "telefone": "9999999"
      }
    },
    {
      "cliente": {
        "nome": "Trololo Oliveira",
        "telefone": "787898899"
      }
    }
  ]
}
```

Fonte: O autor

8.3.2 Criando o método ListarJSON()

Agora vamos criar um novo método que irá trabalhar com JSON, para isso crie o método ListarJSON() após o término do método listar em seu classe DespesaDAO, conforme a Figura 149.

Neste método iremos criar um JSON manualmente para aprendermos como criar essa estrutura em nosso código Java, oportunamente neste curso iremos utilizar a biblioteca GSON para convertermos nossos objetos para JSON diretamente.

Figura 149 – Camada DAO – Classe DespesaDAO – Método listarJSON().

```

151     @Override
152     public List<Object> listar() { ...25 lines }
177
178     public String listarJSON() {
179         String strJson="";
180
181         return strJson;
182     }
183 }
```

Fonte: O autor

Agora termine a implementação do método conforme a Figura 150.

Figura 150 – Método listarJSON() - Implementação.

```

180     public String listarJSON() {
181         String strJson="";
182         PreparedStatement stmt = null;
183         ResultSet rs = null;
184         List<Object> resultado = new ArrayList<>();
185         Despesa oDespesa = null;
186         String sql = "select * from despesa";
187         try {
188             stmt =conexao.prepareStatement(sql);
189             rs = stmt.executeQuery();
190             strJson = "[";
191             int i = 0;
192             while(rs.next()){
193                 if (i>0) strJson+=",";
194                 strJson += "\\"idDespesa\\":\\""+rs.getInt("iddespesa")+"\\",
195                     + "\\"descricao\\":\\""+rs.getString("descricao")+"\\",
196                     + "\\"dataDocumento\\":\\""+data2String(rs.getDate("datadocumento"))+"\\",
197                     + "\\"valorDespesa\\":\\""+valorDinheiro(rs.getDouble("valorDespesa"), "BR")+"\\",
198                     + "\\"valorPago\\":\\""+valorDinheiro(rs.getDouble("valorPago"), "BR")+"\\";
199                 i++;
200             }
201             strJson += "]";
202         } catch (Exception e){
203             System.out.println("Problemas ao Listar Despesa!Erro: " + e.getMessage());
204             e.printStackTrace();
205         }
206         System.out.println(strJson);
207         return strJson;
208     }
```

Fonte: O autor

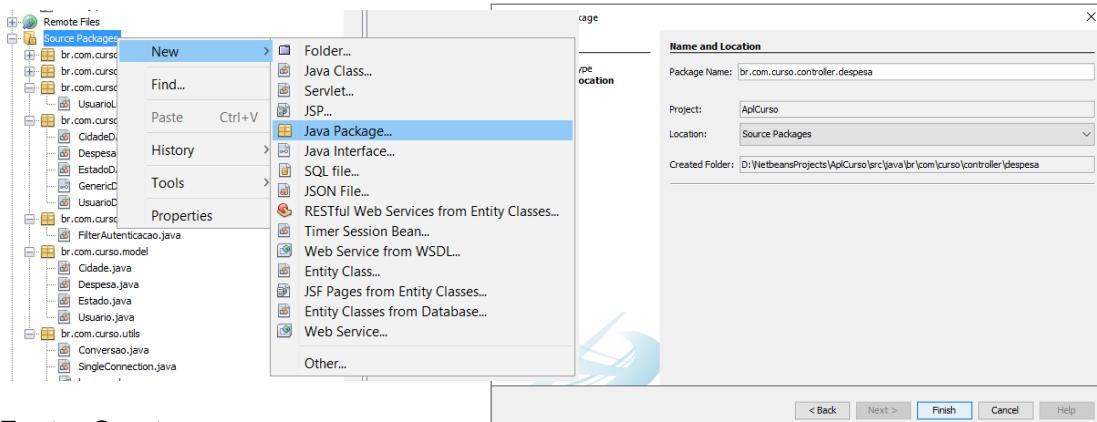
Quando avançarmos no desenvolvimento de nosso projeto poderemos visualizar o resultado deste método.

8.4 DESENVOLVENDO A CAMADA CONTROLLER E VIEW

Agora vamos iniciar o desenvolvimento de nossa camada controller e juntamente vamos desenvolvendo nossa camada view.

Para iniciarmos vamos criar o pacote java que irá armazenar nossas classes do tipo Servlet referentes ao cadastro de despesas em nosso projeto, conforme pode-se visualizar na Figura 151.

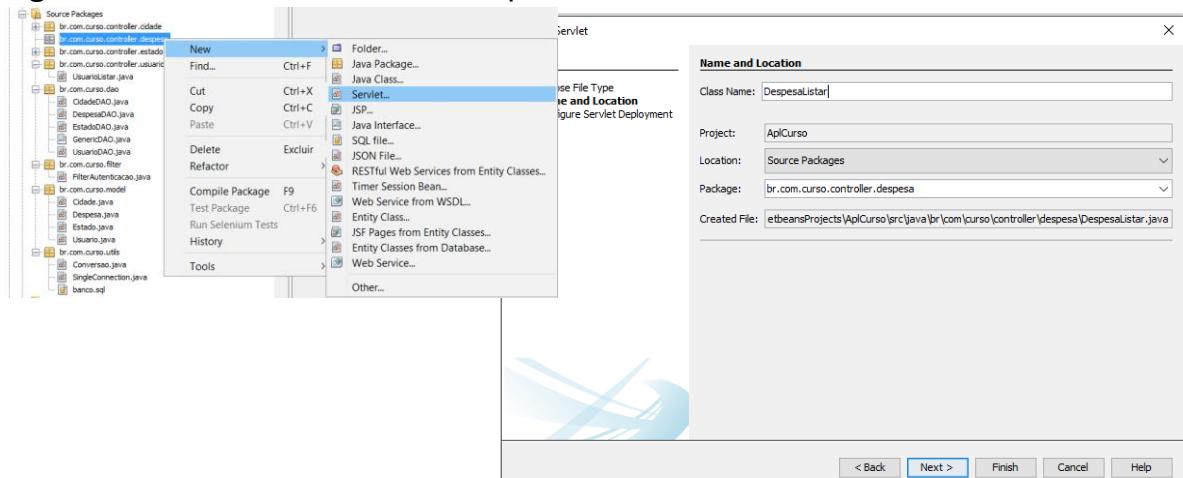
Figura 151 – Criando pacote Controller para cadastro de despesas.



Fonte: O autor

Agora vamos criar o Servlet no pacote controller do cadastro de despesas, para criar o servlet clique com o botão direito no pacote “br.com.curso.controller.despesa” e escolha a opção “New → Servlet”, isto deverá ser feito para criar o Servlet – DespesaListar.

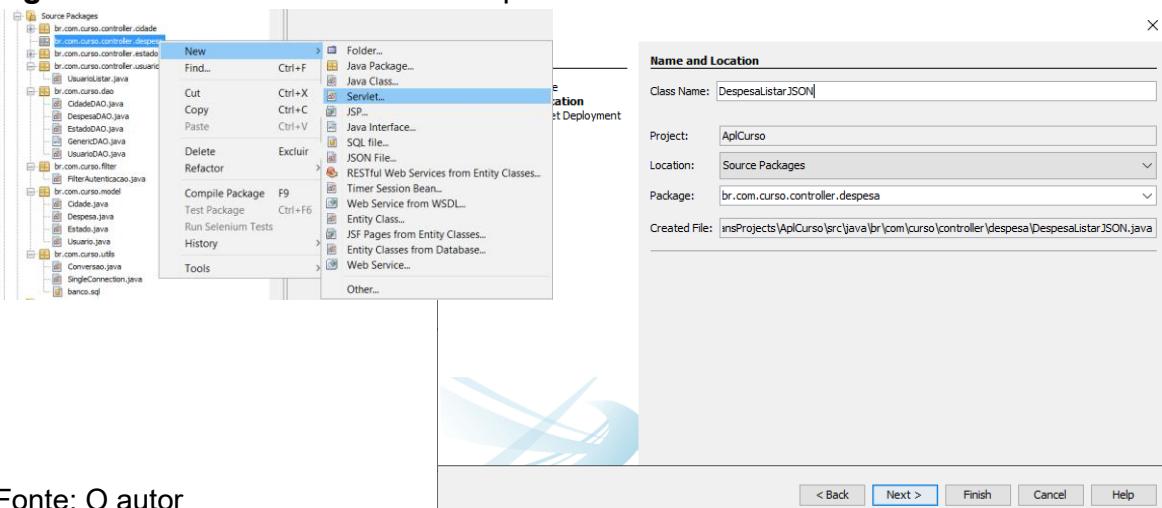
Figura 152 – Criando Servlet – DespesaListar.



Fonte: O autor

A seguir você deve criar o Servlet – DespesaListarJSON que executará a função de listar só que agora utilizando JSON.

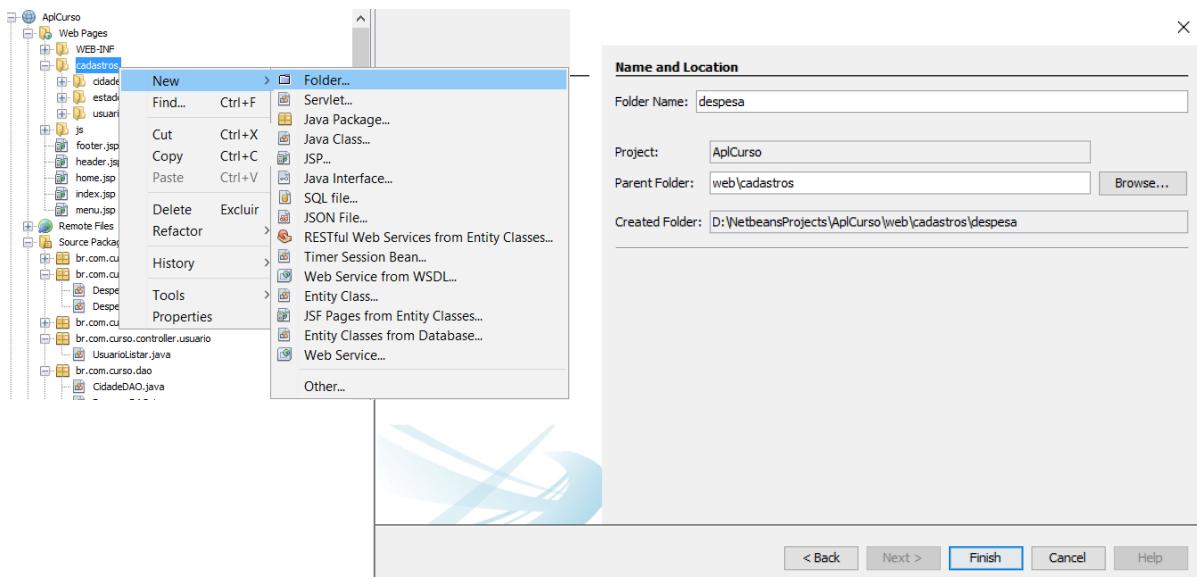
Figura 153 – Criando Servlet – DespesaListarJSON.



Fonte: O autor

Antes de iniciarmos o desenvolvimento vamos preparar também nossa view com a estrutura necessária, para isso em Web Pages, na pasta “cadastros” clique com o botão direito e escolha a opção “New → Folder” e crie a pasta “despesa”, conforme a Figura 154.

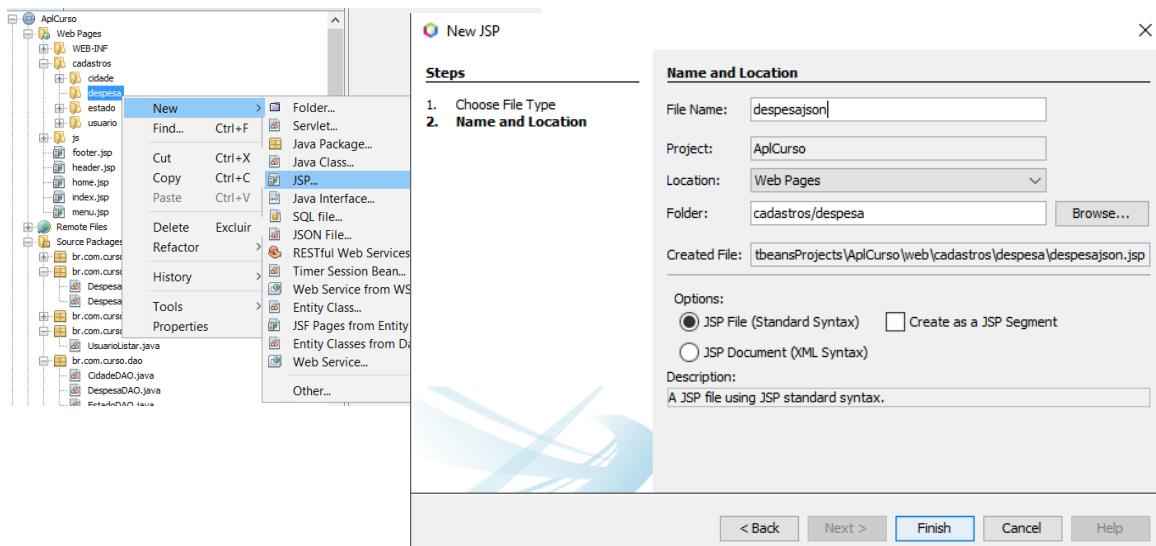
Figura 154 – View – Criando pasta para JSPs de despesa.



Fonte: O autor

Agora dentro da pasta despesa vamos criar o JSP para o listar. Na Figura 155 é criado o despesa.jsp.

Figura 155 – View – Criando Despesa.jsp



Fonte: O autor

8.4.1 Implementando o Listar

Vamos implementar o Servlet DespesaListar , para isso abra o seu servlet na camada controller e altere o método processRequest de acordo com o código apresentado na Figura 156.

Figura 156 – Controller – Servlet DespesaListar

```

34     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35         throws ServletException, IOException {
36             response.setContentType("text/html; charset=iso-8859-1");
37             try{
38                 GenericDAO dao = new DespesaDAO();
39                 request.setAttribute("despesas", dao.listar());
40                 request.getRequestDispatcher("/cadastros/despesa/despesa.jsp").forward(request, response);
41             } catch (Exception ex){
42                 //ex.printStackTrace();
43                 System.out.println("Problemas no Servlet ao Listar Estados! Erro: " + ex.getMessage());
44                 ex.printStackTrace();
45             }
46         }
    
```

Fonte: O autor

Insira no “menu.jsp” as opções de acesso como descritas abaixo, para podermos chamar o listar de despesas.

Figura 157 – View – menu.jsp

```

1 <h1>Módulo Cadastros</h1>
2 <hr>
3 <center>
4   <h2>Menu Principal</h2>
5   <a href="${pageContext.request.contextPath}/EstadoListar">Estado</a>
6   <a href="${pageContext.request.contextPath}/CidadeListar">Cidade</a>
7   <a href="${pageContext.request.contextPath}/UsuarioListar">Usuario</a>
8   <a href="${pageContext.request.contextPath}/DespesaListar">Despesa</a>
9 </center>
10 <hr>

```

Fonte: O autor

Agora vamos modificar o nosso despesa.jsp na camada View de nosso projeto. Desenvolva o “Despesa.jsp” de seu projeto conforme demonstrado na Figura 158.

Figura 158 – View – despesa.jsp

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4 <jsp:include page="/header.jsp"/>
5 <jsp:include page="/menu.jsp"/>
6
7 <div class="container-fluid">
8   <!-- Page Heading -->
9   <h1 class="h3 mb-2 text-gray-800">Despesas</h1>
10  <p class="mb-4">Planilha de Registros</p>
11
12  <a class="btn btn-success mb-4" href="${pageContext.request.contextPath}/DespesaNovo">
13    <i class="fas fa-sticky-note"></i>
14    <strong>Novo</strong>
15  </a>
16
17  <div class="card shadow">
18    <div class="card-body">
19      <table id="datatable" class="display">
20        <thead>
21          <tr>
22            <th align="right">ID</th>
23            <th align="left">Descrição</th>
24            <th align="center">Data</th>
25            <th align="right">ValorDespesa</th>
26            <th align="right">ValorPago</th>
27            <th Align="center">Excluir</th>
28            <th Align="center">Alterar</th>
29        </tr>
30      </thead>
31      <tbody>
32        <c:forEach var="despesa" items="${despesas}">
33          <tr>
34            <td align="right">${despesa.idDespesa}</td>
35            <td align="left">${despesa.descricao}</td>
36            <td align="center"><fmt:formatDate pattern = "dd/MM/yyyy" value = "${despesa.dataDocumento}" /></td>
37            <td align="right"><fmt:formatNumber value = "${despesa.valorDespesa}" type = "currency"/></td>
38            <td align="right"><fmt:formatNumber value = "${despesa.valorPago}" type = "currency"/></td>
39            <td align="center">
40              <a href="#" id="deletar" title="Excluir" onclick="deletar(${despesa.idDespesa})">
41                <button>Excluir</button>
42              </a>
43            </td>
44            <td align="center">
45              <a href="${pageContext.request.contextPath}/DespesaCarregar?idDespesa=${despesa.idDespesa}">
46                <button>Alterar</button>
47              </a>
48            </td>
49          </tr>
50        </c:forEach>
51      </tbody>
52    </table>
53  </div>
54</div>
55</div>

```

Fonte: O autor

Figura 159 – View – despesa.jsp – continuação

```

57 <script>
58     $(document).ready(function() {
59         console.log('entrei ready');
60         //Carregamos a datatable
61         //$("#datatable").DataTable();
62         $('#datatable').DataTable({
63             "oLanguage": {
64                 "sProcessing": "Processando...",
65                 "sLengthMenu": "Mostrar _MENU_ registros",
66                 "sZeroRecords": "Nenhum registro encontrado.",
67                 "sInfo": "Mostrando de _START_ até _END_ de _TOTAL_ registros",
68                 "sInfoEmpty": "Mostrando de 0 até 0 de 0 registros",
69                 "sInfoFiltered": "",
70                 "sInfoPostFix": "",
71                 "sSearch": "Buscar:",
72                 "sUrl": "",
73                 "oPaginate": {
74                     "sFirst": "Primeiro",
75                     "sPrevious": "Anterior",
76                     "sNext": "Seguinte",
77                     "sLast": "Último"
78                 }
79             });
80         });
81     });
82
83     function deletar(codigo){
84         var id = codigo;
85         console.log(codigo);
86         Swal.fire({
87             title: 'Você tem certeza?',
88             text: 'Você não poderá recuperar depois!',
89             icon: 'warning',
90             showCancelButton: true,
91             confirmButtonColor: '#3085d6',
92             cancelButtonColor: '#d33',
93             confirmButtonText: 'Sim, apague a despesa!',
94             cancelButtonText: 'Cancelar'
95         }).then((result) => {
96             if (result.isConfirmed) {
97                 $.ajax({
98                     type: 'post',
99                     url: '${pageContext.request.contextPath}/DespesaExcluir',
100                    data: {
101                        idDespesa: id
102                    },
103                    success:
104                     function(data){
105                         if(data == 1){
106                             Swal.fire({
107                             position: 'top-end',
108                             icon: 'success',
109                             title: 'Sucesso',
110                             text: 'Despesa excluída com sucesso!',
111                             showConfirmButton: false,
112                             timer: 2000
113                         })
114                         } else {
115                             Swal.fire({
116                             position: 'top-end',
117                             icon: 'error',
118                             title: 'Erro',
119                             text: 'Não foi possível excluir a despesa!',
120                             showConfirmButton: false,
121                             timer: 2000
122                         })
123                         }
124                         window.location.href = "${pageContext.request.contextPath}/DespesaListar";
125                     },
126                     error:
127                     function(data){
128                         window.location.href = "${pageContext.request.contextPath}/DespesaListar";
129                     }
130                 });
131             );
132         });
133     }
134 </script>
135 <%@include file="/footer.jsp"%>
```

Fonte: O autor

Figura 160 – View – Resultado

Módulo Cadastros

Menu Principal
Estado Cidade Usuario Despesa

Despesas
Planilha de Registros

Novo

ID	Descrição	Data	ValorDespesa	ValorPago	Excluir	Alterar
1	descricao	23/08/2021	R\$ 20,50	R\$ 10,50	Excluir	Alterar
2	inclusao	03/09/2021	R\$ 21,50	R\$ 31,50	Excluir	Alterar
3	despesa sem imagem	10/12/2021	R\$ 0,15	R\$ 0,15	Excluir	Alterar
4	teste	01/01/2021	R\$ 15,00	R\$ 15,00	Excluir	Alterar
5	tste documento	25/12/2021	R\$ 151,01	R\$ 152,15	Excluir	Alterar

Mostrando de 1 até 5 de 5 registros

Anterior 1 Seguinte

Desenvolvendo Aplicações com Java Web

Fonte: O autor

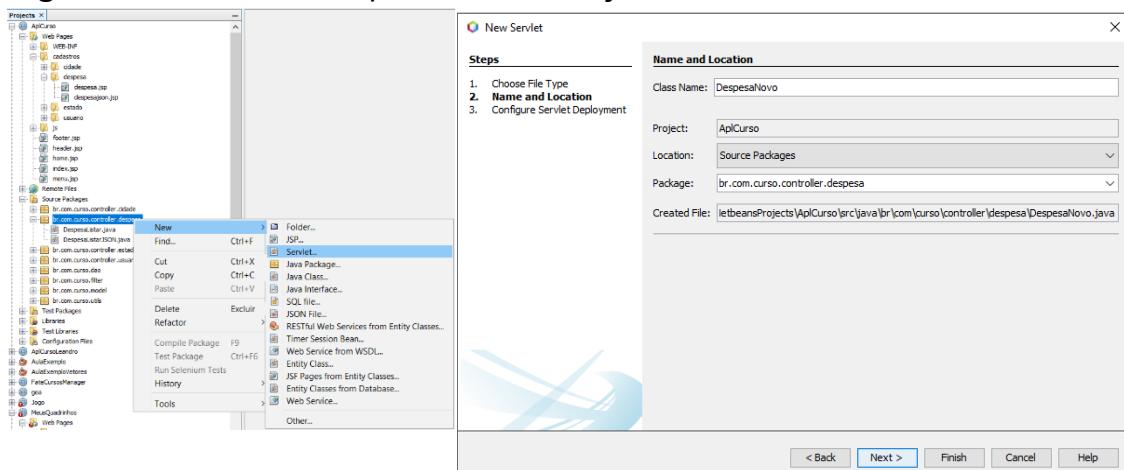
8.4.2 Implementando o Cadastrar

Agora vamos implementar o cadastramento de despesas, vamos começar implementando os servlets necessários para o processo de processamento das requisições do front-end.

Vamos criar o servlet DespesaNovo que irá inicializar nossa tela de cadastro de despesas, para isso vamos clicar com o botão direito no pacote br.com.curso.controller.despesa e escolha a opção New→Servlet e na janela preencha o nome do nosso novo Servlet e clique em finalizar. Você pode observar esse processo de criação na Figura 162.

Esse servlet irá instanciar um objeto vazio do tipo despesa e enviar para o nosso formulário de cadastramento de despesas que ficará na passa de despesa na nossa interface.

Figura 161 – Servlet DespesaNovo – criação.



Fonte: O autor

Depois de criado o Servlet DespesaNovo vamos codificá-lo, para isso vamos apagar o código que foi gerado pelo Netbeans no método processRequest e codificá-lo de acordo com a Figura 162.

Figura 162 – Servlet DespesaNovo – código.

```

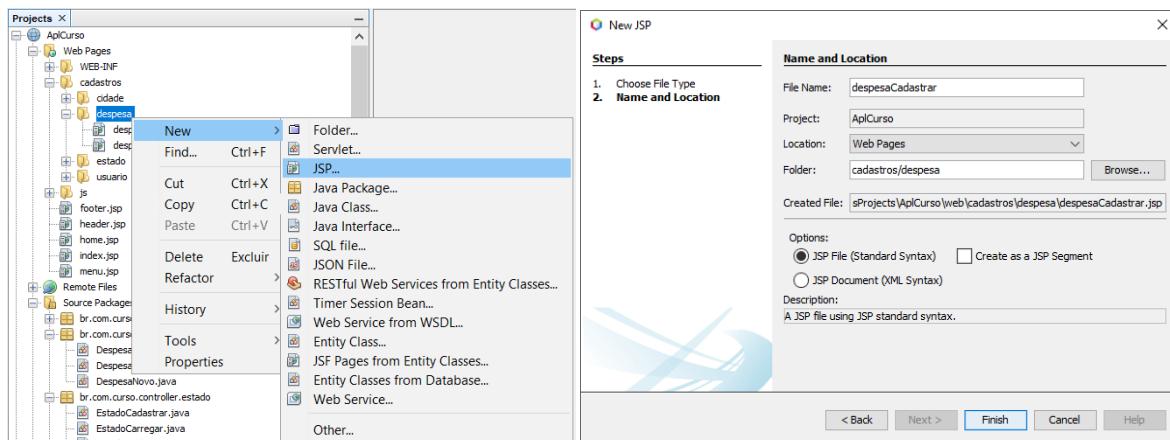
35 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
36     throws ServletException, IOException {
37
38     response.setContentType("text/html;charset=iso-8859-1");
39     try {
40         Despesa oDespesa = new Despesa();
41         request.setAttribute("despesa", oDespesa);
42         request.getRequestDispatcher("cadastros/despesa/despesaCadastrar.jsp").forward(request, response);
43     } catch (Exception e) {
44         System.out.println("Problema na Servlet carregar despesa!Erro: " + e.getMessage());
45         e.printStackTrace();
46     }
47 }
```

Fonte: O autor

Agora vamos criar nosso JSP para nossa interface funcionar, para isso vamos clicar com o botão direito na pasta “Web Pages/cadastros/despesa” escolher a opção New→JSP. Na janela informe o nome de nosso JSP como “despesaCadastrar” sem colocar a extensão e confirme sua criação.

Este processo você pode observar na Figura 163.

Figura 163 – JSP cadastrar despesa – Criação.



Fonte: O autor

Criado o arquivo “cadastrarDespesa.JSP” vamos implementar nossa interface, você para verificar a codificação na Figura 165. Nesta tela nós vamos realizar algumas atividades diferentes como configuração de campo de data e monetário, assim como fazer o upload de imagens.

Vamos construir esse formulário em etapas pois vamos observar algumas implementações diferentes, para isso faça a codificação da parte HTML como especificado na Figura 164.

Figura 164 – JSP Cadastrar Despesa – Codificação HTML

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4 <jsp:include page="/header.jsp"/>
5 <jsp:include page="/menu.jsp"/>
6
7 <div class="container-fluid">
8   <!-- Page Heading -->
9   <h1 class="h3 mb-2 text-gray-800">Despesas</h1>
10  <p class="mb-4">Formulário de Cadastro</p>
11
12  <a class="btn btn-secondary mb-4" href="${pageContext.request.contextPath}/DespesaListar">
13    <i class="fas fa-undo-alt"></i>
14    <strong>Voltar</strong>
15  </a>
16  <div class="row">
17    <!-- Imagem do Documento -->
18    <div class="col">
19      <div class="card shadow mb-4">
20        <div class="card-body">
21          <div class="form-group">
22            <center>
23              
26              <br><br>
27              <input type="file" id="gallery-photo-add"
28                  class="inputfile" onchange="uploadFile();"/>
29              <label for="gallery-photo-add" class="btn btn-success">
30                <i class="fas fa-file-upload"></i>
31                Selecionar Capa...
32              </label>
33            </center>
34          </div>
35        </div>
36      </div>
37    </div>

```

Fonte: O autor

A codificação HTML prossegue na Figura 165.

Figura 165 – JSP Cadastrar Despesa – Codificação HTML - Continuação

```

38 <!-- Campos de cadastramento -->
39 <div class="col-lg-9">
40   <div class="card shadow mb-4">
41     <div class="card-body">
42       <div class="form-group">
43         <label>Id</label>
44         <input class="form-control" type="text" name="idDespesa" id="iddespesa"
45           value="${despesa.idDespesa}" readonly="readonly"/>
46       </div>
47       <div class="form-group">
48         <label>Descrição da Despesa</label>
49         <input class="form-control" type="text" name="descricao" id="descricao"
50           value="${despesa.descricao}" size="100" maxlength="100"/>
51       </div>
52       <div class="form-group">
53         <div class="form-line row">
54           <div class="col-sm">
55             <label>Data da Despesa</label>
56             <input class="form-control" type="date" name="datadocumento" id="datadocumento"
57               value="${despesa.dataDocumento}"/>
58           </div>
59           <div class="col-sm">
60             <label>Valor da Despesa</label>
61             <input class="form-control" type="text" style="text-align:right;" 
62               name="valordespesa" id="valordespesa"
63               value=<fmt:formatNumber value='${despesa.valorDespesa}' type='currency' />
64           </div>
65           <div class="col-sm">
66             <label>Valor Pago</label>
67             <input class="form-control" type="text" style="text-align:right;" 
68               name="valorpago" id="valorpago"
69               value=<fmt:formatNumber value='${despesa.valorPago}' type='currency' />
70           </div>
71         </div>
72       </div>
73       <!-- Botão de Confirmação -->
74       <div class="form-group">
75         <button class="btn btn-success" type="submit" id="submit" onclick="validarCampos() ">
76           Salvar Documento</button>
77       </div>
78     </div>
79   </div>
80 </div>
81 </div>
82 </div>
83 <style type="text/css">
84 </style>
85 <script>
86 </script>
87 <jsp:include page="/footer.jsp"/>

```

Fonte: O autor

Vamos verificar como ficou nossa página, observe a Figura 167. Pode-se visualizar duas regiões em nossa tela de cadastros, a primeira a esquerda irá receber a imagem do documento e a segunda a direita contém os dados do documento a ser cadastrado no sistema.

Observe também que por exemplo na região da imagem do documento as coisas estão desorganizadas e componentes aparecendo onde não deviam. Isso ocorre porque ainda não implementamos em nosso JSP os códigos de javascript e também desta vez iremos configurar alguns CSS's específicos para nossa região de imagem.

Figura 166 – JSP Cadastrar Despesa - Tela

Módulo Cadastros

Menu Principal

Estado Cidade Usuário Despesa (Listar sem JSON) Despesa (Listar com JSON)

Despesas

Formulário de Cadastro

Voltar

Id: 0

Descrição da Despesa:

Data da Despesa: dd/mm/aaaa

Valor da Despesa: R\$ 0,00

Valor Pago: R\$ 0,00

Salvar Documento

Desenvolvendo Aplicações com Java Web

Fonte: O autor

Para corrigir a nossa região de imagem implemente na página o código CSS descrito na Figura 167, esse código deverá ser colocado na região que já foi deixada durante a codificação do HTML para o código CSS entre as tags <style>. Você também pode observar o resultado dessa modificação.

Figura 167 – JSP Cadastrar Despesa – Aplicando CSS.

```

84 <style type="text/css">
85   .inputfile {
86     /* visibility: hidden etc. wont work */
87     width: 0.1px;
88     height: 0.1px;
89     opacity: 0;
90     overflow: hidden;
91     position: absolute;
92     z-index: -1;
93   }
94   .inputfile:focus + label {
95     /* keyboard navigation */
96     outline: 1px dotted #000;
97     outline: -webkit-focus-ring-color auto 5px;
98   }
99   .inputfile + label * {
100     pointer-events: none;
101   }
102   .borda {
103     position: relative;
104     margin: 0 20px 30px 0;
105     padding: 10px;
106     border: 1px solid #e1e1e1;
107     border-radius: 3px;
108     background: #fff;
109     -webkit-box-shadow: 0px 0px 3px rgba(0,0,0,0.06);
110     -moz-box-shadow: 0px 0px 3px rgba(0,0,0,0.06);
111     box-shadow: 0px 0px 3px rgba(0,0,0,0.06);
112   }
113 </style>

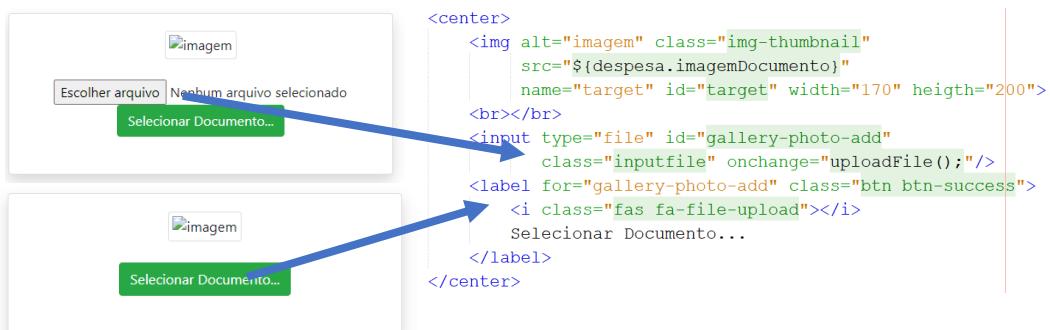
```

Fonte: O autor

Agora vamos dar a atenção aos nossos códigos em javascript, novamente a seção que irá receber essa codificação já foi deixada preparada para receber tais códigos durante a criação do HTML de nossa página.

Vamos começar fazendo nosso botão de selecionar documento funcionar, só para recordar na edição do CSS anteriormente realizada ocultamos o botão “Escolher Arquivo” que na verdade era uma tag HTML Input do tipo “file” e deixamos aparecendo apenas a label de cor verde que fica com uma estética melhor na interface (Figura 168).

Figura 168 – JSP Cadastrar – Upload Arquivo.



Fonte: O autor

Agora para definirmos o comportamento desses componentes vamos criar uma função em javascript dentro da tag `<script></script>` denominada “`uploadFile()`”, repare na Figura 169 que essa função ficará associada ao evento “`onchange`” de nossa Tag Input do tipo “File”.

Assim vamos criar a codificação conforme a Figura 169.

Figura 169 – JSP Cadastrar – Upload Arquivo – Função `uploadFile()`.

```

111 <script>
112
113     function uploadFile() {
114         //pega o componente html image
115         var target = document.getElementById("target");
116         //limpa o image
117         target.src = "";
118         //abre a janela para seleção do arquivo.
119         var file = document.querySelector("input[type='file']").files[0];
120         //verifica se o arquivo existe
121         if (file) {
122             //faz a leitura do arquivo da imagem
123             var reader = new FileReader();
124             reader.readAsDataURL(file);
125             reader.onloadend = function () {
126                 //atribui a imagem do arquivo ao componente html image
127                 target.src = reader.result;
128             };
129         } else {
130             target.src = "";
131         }
132     }
133
134 </script>
135 <jsp:include page="/footer.jsp"/>

```

Fonte: O autor

A partir de agora conseguimos quando clicarmos no botão selecionarmos uma imagem e carregá-la para a tela de nossa aplicação.

Agora vamos configurar o comportamento dos campos de valores monetários em nossa tela pois quando digitamos algum valor nesses o comportamento não é adequado, por exemplo não colocando o símbolo de moeda após a digitação e também não facilitando na questão da digitação do valor decimal.

Para isso vamos utilizar o JQuery MaskMoney que tratará o comportamento desses campos, mas devemos carregar essas configurações antes da página HTML terminar de carregar para o usuário.

Isso é possível através da tag “\$(document).ready()”, então vamos programar nossas configurações dentro da área de script de nossa página nesta tag, conforme a Figura 170.

Figura 170 – JSP Cadastrar – Configuração campos de moeda.

```

111<script>
112 $(document).ready(function () {
113     console.log("entrei na ready do documento");
114     $("#valordespesa").maskMoney({
115         prefix: 'R$',
116         suffix: '',
117         allowZero: false,
118         allowNegative: true,
119         allowEmpty: false,
120         doubleClickSelection: true,
121         selectAllOnFocus: true,
122         thousands: ',',
123         decimal: ",",
124         precision: 2,
125         affixesStay: true,
126         bringCareAtEndOnFocus: true
127     });
128
129     $("#valorpago").maskMoney({
130         prefix: 'R$',
131         suffix: '',
132         allowZero: false,
133         allowNegative: true,
134         allowEmpty: false,
135         doubleClickSelection: true,
136         selectAllOnFocus: true,
137         thousands: ',',
138         decimal: ",",
139         precision: 2,
140         affixesStay: true,
141         bringCareAtEndOnFocus: true
142     });
143 }
144 function uploadFile() {

```

Fonte: O autor

Agora faça o teste de digitação dos valores monetários nos campos o observe como ficou o seu comportamento.

Então fazer o tratamento dos dados para a sua gravação no banco, antes de enviarmos os dados para a camada controller de nossa aplicação temos que realizar

as consistências no Front-End. Esse processo vai ser realizado pela função javascript validarCampos() que iremos implementar.

Nossa página JSP não possui um formulário (<Form>) como um formulário comum pois estamos trabalhando de uma forma que iremos enviar os dados através de uma requisição Ajax. Assim em nosso botão “Salvar Documento” através do evento “onClick” (Figura 171) acionaremos a função validarCampos() que se não encontrar nenhum erro irá chamar a função javascript gravarDados().

Figura 171 – JSP Cadastrar – Botão Salvar Documento.



Fonte: O autor

Desenvolva a função validarCampos() conforme a Figura 172.

Figura 172 – JSP Cadastrar – Função validarCampos().

```

145 function validarCampos() {
146     console.log("entrei na validação de campos");
147     if (document.getElementById("descricao").value == '') {
148         Swal.fire({
149             position: 'center',
150             icon: 'error',
151             title: 'Verifique a descrição da despesa!',
152             showConfirmButton: false,
153             timer: 1000
154         });
155         $("#descricao").focus();
156     } else if (document.getElementById("datadocumento").value == '') {
157         Swal.fire({
158             position: 'center',
159             icon: 'error',
160             title: 'Verifique a Data da despesa!',
161             showConfirmButton: false,
162             timer: 1000
163         });
164         $("#datadocumento").focus();
165     } else if (document.getElementById("valordespesa").value == '') {
166         Swal.fire({
167             position: 'center',
168             icon: 'error',
169             title: 'Verifique o valor da despesa!',
170             showConfirmButton: false,
171             timer: 1000
172         });
173         $("#valordespesa").focus();
174     } else {
175         gravarDados();
176     }
177 }
178
179 function uploadFile() {

```

Fonte: O autor

Se a função validar campos não encontrar nenhum erro irá chamar a função javascript gravarDados() que será responsável por gerar a requisição Ajax para a gravação dos dados. Vamos codificá-la conforme demonstrado na Figura 173.

Figura 173 – JSP Cadastrar – função gravarDados().

```

179 	function gravarDados() {
180      console.log("Gravando dados ....");
181      var target = document.getElementById("target").src;
182      $.ajax({
183        type: 'post',
184        url: 'DespesaCadastrar',
185        data: {
186          iddespesa: $('#iddespesa').val(),
187          descricao: $('#descricao').val(),
188          datadocumento: $('#datadocumento').val(),
189          valordespesa: $('#valordespesa').val(),
190          valorpago: $('#valorpago').val(),
191          imagemdocumento: target
192        },
193        success:
194          function (data) {
195            console.log("resposta servlet->");
196            console.log(data);
197            if (data == 1) {
198              Swal.fire({
199                position: 'center',
200                icon: 'success',
201                title: 'Sucesso',
202                text: 'Despesa gravada com sucesso!',
203                showConfirmButton: false,
204                timer: 1000
205              })
206            } else {
207              Swal.fire({
208                position: 'center',
209                icon: 'error',
210                title: 'Erro',
211                text: 'Não foi possível gravar a despesa!',
212                showConfirmButton: false,
213                timer: 1000
214              })
215              window.location.href = "${pageContext.request.contextPath}/DespesaListar";
216            }
217          },
218        error:
219          function (data) {
220            window.location.href = "${pageContext.request.contextPath}/DespesaListar";
221          }
222      });
223    };
224  }

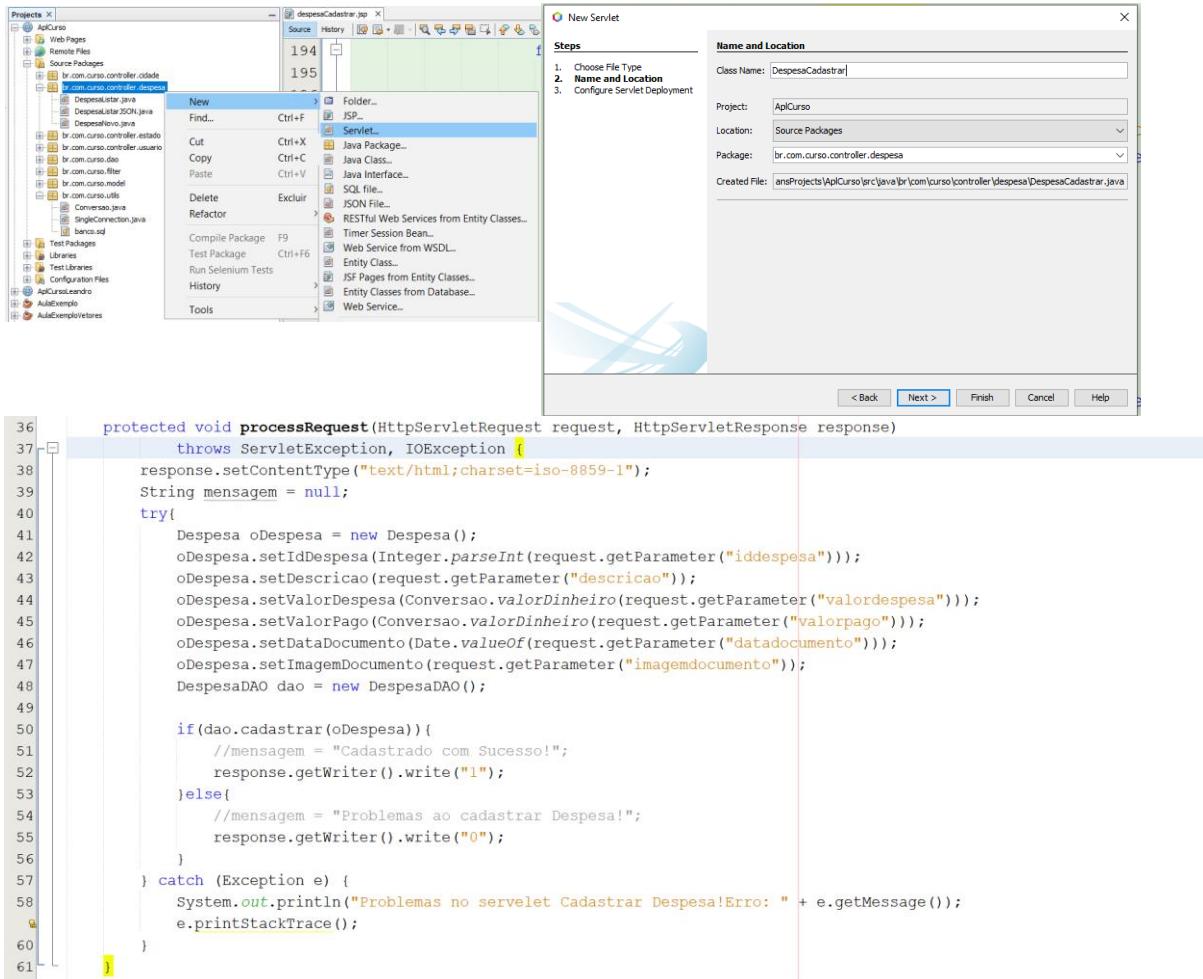
```

Fonte: O autor

Depois de implementado nossa função Javascript de gravar dados vamos implementar em nosso back-end a controller (Servlet) que irá receber os dados. Para isso clique novamente com o botão direito na controller de despesa e escolha a opção New->Servlet na janela informe o nome de nosso novo servlet “DespesaCadastrar” e confirme sua criação (Figura 174).

Apague o código em processRequest gerado pelo Netbeans e implemente conforme demonstrado abaixo.

Figura 174 – Controller – Servlet: DespesaCadastrar.



Fonte: O autor

A partir desta implementação você testar seu projeto, a página ficará como abaixo.

Figura 175 – View Cadastrar Despesa – Resultado Final

The screenshot shows a web application interface titled 'Módulo Cadastros'. At the top, there is a 'Menu Principal' with links for 'Estado', 'Cidade', 'Usuário', and 'Despesa'. Below the menu, the page title is 'Despesas' and the subtitle is 'Formulário de Cadastro'. There is a 'Voltar' button. On the left, there is a preview of a document with a green 'Selecionar Documento...' button. The main form contains the following fields:

- Id:** A text input field containing '0'.
- Descrição da Despesa:** A text input field containing 'Depósito teste'.
- Data da Despesa:** A date input field showing '03/09/2021'.
- Valor da Despesa:** A text input field showing 'R\$150,00'.
- Valor Pago:** A text input field showing 'R\$135,25'.
- Salvar Documento:** A green button at the bottom of the form.

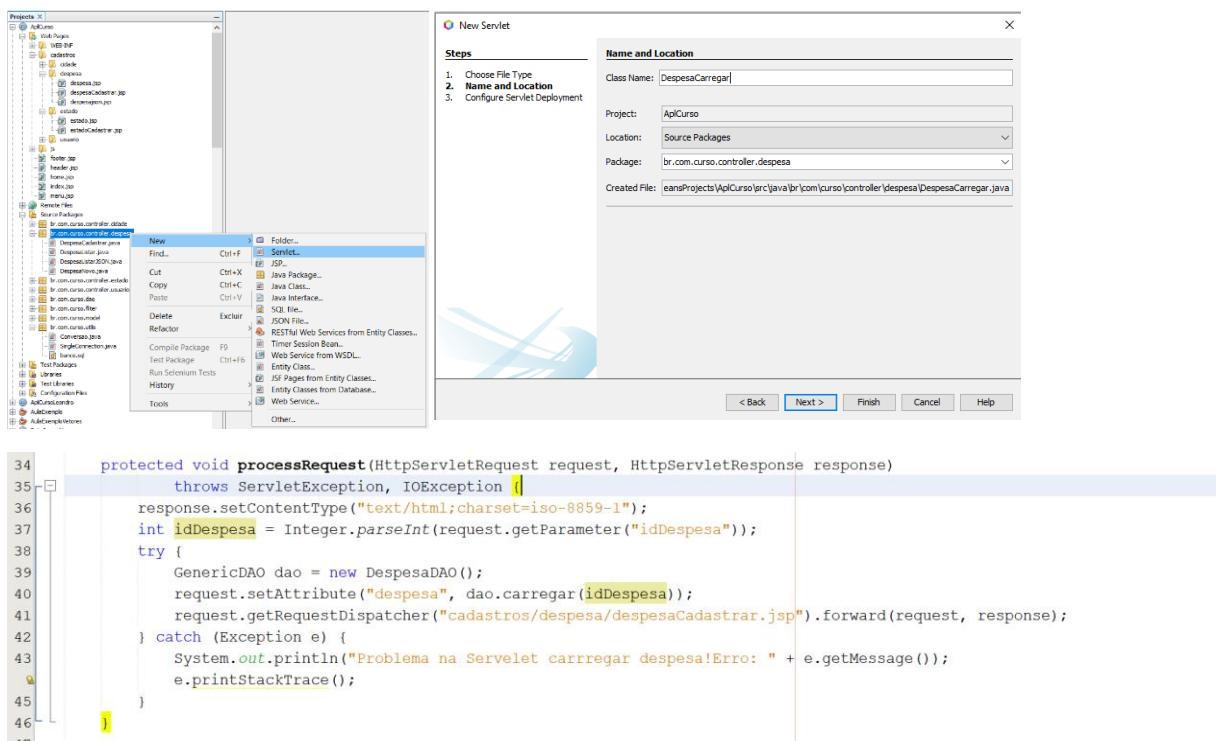
Fonte: O autor

8.4.3 Implementando o Alterar

Para implementar o processo de alteração em nosso CRUD devemos implementar o Servlet DepesaCarregar na controller. Para isso clique com o botão direito no pacote br.com.curso.controller.despesa e escolha a opção New→Servlet e na janela informe o nome do servlet “DepesaCarregar” e confirme sua criação (Figura 176).

Após criado o novo servlet altere seu processRequest deixando como demonstrado na Figura 176.

Figura 176 – Controller – Servlet DespesaCarregar.



Fonte: O autor

Agora temos a alteração implementada e funcional em nosso projeto.

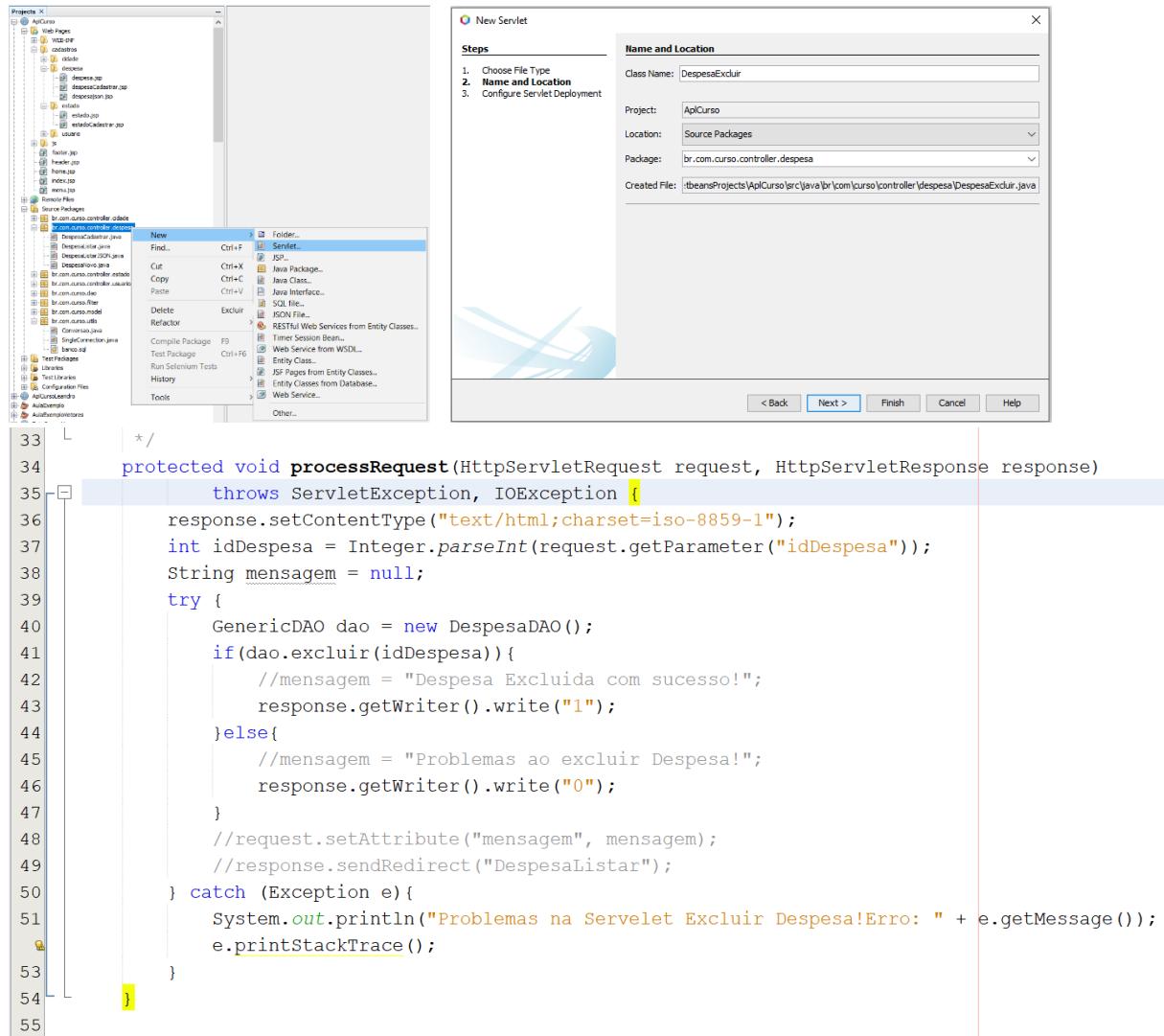
8.4.4 Implementando o Excluir

Para implementar o processo de exclusão em nosso CRUD devemos implementar o Servlet DepesaExcluir na controller. Para isso clique com o botão direito no pacote br.com.curso.controller.despesa e escolha a opção New→Servlet e na

janela informe o nome do servlet “DespesaExcluir” e confirme sua criação (Figura 177).

Após criado o novo servlet altere seu processRequest deixando como demonstrado na Figura 177.

Figura 177 – Controller – Servlet DespesaExcluir



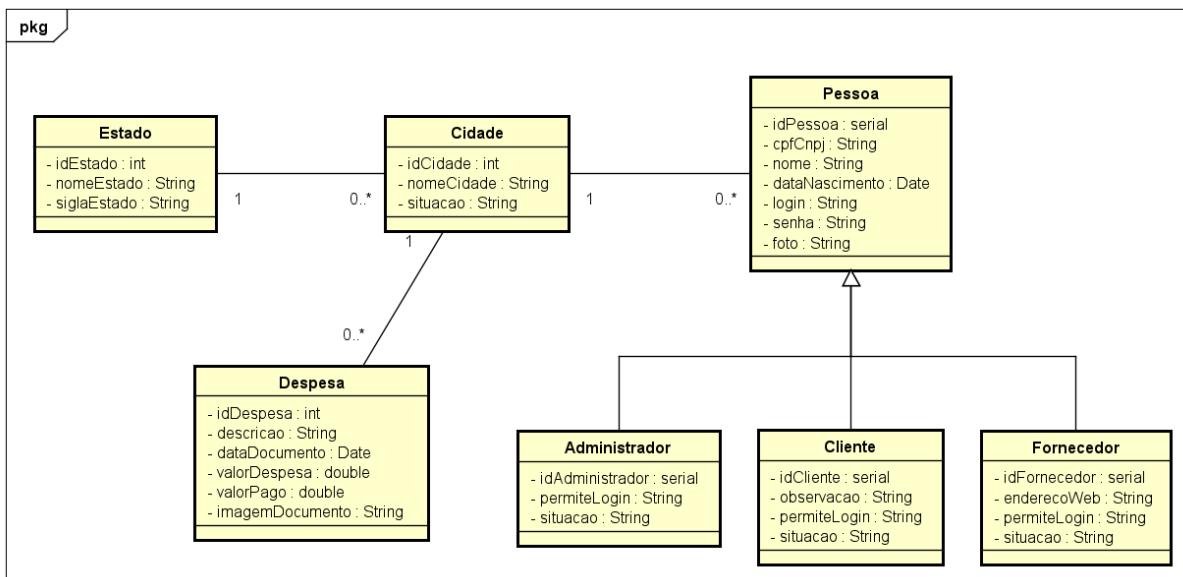
Fonte: O autor

Este servlet será solicitado pela função javascript deletar() que temos em nosso depesa.jsp que criará uma requisição Ajax para a exclusão.

CAPÍTULO 9 – DESENVOLVIMENTO DE CRUD COM HERANÇA

Neste capítulo de nosso curso vamos começar manipular classes com relacionamento generalização/especialização. Agora vamos desenvolver as classes Pessoa, Administrador, Cliente, Fornecedor.

Figura 178 – Diagrama de Classe



Fonte: O autor

Cria sua tabela Pessoa no seu banco de dados.

Figura 179 – Código SQL – Criando a tabela pessoa.

```

3  create table pessoa (
4      idpessoa serial primary key,
5      nome varchar(100) not null,
6      cpfcnpj varchar(14) not null unique,
7      datanascimento date,
8      idcidade int,
9      login varchar(20),
10     senha varchar(20),
11     foto text,
12     constraint fk_cidade foreign key (idcidade) references cidade
13 );
14
15 insert into pessoa (nome, cpfcnpj, datanascimento, idcidade, login, senha, foto)
16     values ('adm','42745947001', '01-01-2020', 1, 'adm','123',null);
  
```

Fonte: O autor

Agora vamos criar as tabelas filhas.

Figura 180 – Código SQL – Criando a tabela pessoa.

```

1  create table administrador (
2      idadministrador serial primary key,
3      idpessoa int unique,
4      situacao varchar(1),
5      permitelogin varchar(1),
6      constraint fk_administrador_pessoa foreign key (idpessoa) references pessoa
7  );
8
9  insert into administrador (idpessoa,situacao,permitelogin)
10    values (1,'A','S');
11
12 create table cliente (
13     idcliente serial primary key,
14     idpessoa int unique,
15     observacao varchar(100),
16     situacao varchar(1),
17     permitelogin varchar(1),
18     constraint fk_cliente_pessoa foreign key (idpessoa) references pessoa
19 );
20
21 create table fornecedor (
22     idfornecedor serial primary key,
23     idpessoa int unique,
24     enderecowaeb varchar(100),
25     situacao varchar(1),
26     permitelogin varchar(1),
27     constraint fk_fornecedor_pessoa foreign key (idpessoa) references pessoa
28 );

```

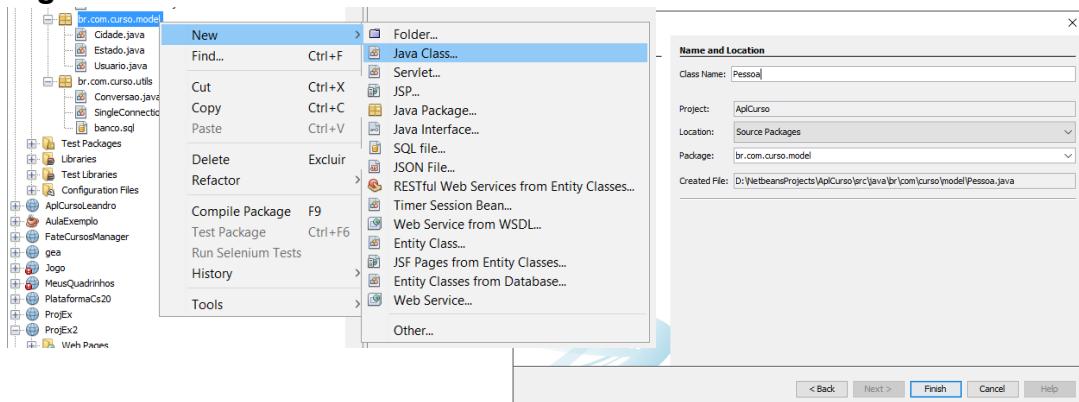
Fonte: O autor

9.1 DESENVOLVENDO A CAMADA MODEL

Agora vamos desenvolver nossa camada model para isso vamos criar a classe java “Pessoa” conforme o modelo de dados definido no início deste capítulo. Para isso clique com o botão direito em seu pacote da camada model em seu projeto, “br.com.curso.model”.

Escolha a opção “New → Java Class” e na janela coloque o nome da classe “Despesa” e confirme sua criação, como podemos observar na **Figura 181 – Camada Model – Criando a classe Pessoa.**

Figura 181 – Camada Model – Criando a classe Pessoa.



Fonte: O autor

Agora vamos criar nossos atributos na classe Pessoa.

Figura 182 – Camada Model – Criando a classe Pessoa.

```

1 package br.com.curso.model;
2
3 import java.util.Date;
4
5 public class Pessoa {
6
7     private int idPessoa;
8     private String cpfCnpj;
9     private String nome;
10    private Date dataNascimento;
11    private Cidade cidade;
12    private String login;
13    private String senha;
14    private String foto;
15
16 }

```

Fonte: O autor

Quando vamos trabalhar com o dados do tipo data devemos importar a classe “java.util.Date” que nos permite manipular essas informações, tome cuidado pois existem outras classes que também trabalham com datas e possuem nomes parecidos como a “java.sql.Date”.

Dando continuidade à nossa camada model vamos criar os métodos construtores, para isso gere os métodos através dos atalhos de sua IDE. Gere apenas o construtor com parâmetros para preenchimento do objeto gerado pois a classe Pessoa será classe pai de nossas classes Administrador, Cliente e Fornecedor (**Figura 183 – Camada Model – Gerando os métodos construtores**).

Figura 183 – Camada Model – Gerando os métodos construtores

```

5   public class Pessoa {
6
7     private int idPessoa;
8     private String cpfCnpj;
9     private String nome;
10    private Date dataNascimento;
11    private Cidade cidade;
12    private String login;
13    private String senha;
14    private String foto;
15
16    public Pessoa(int idPessoa, String cpfCnpj, String nome, Date dataNascimento, Cidade cidade,
17      String login, String senha, String foto) {
18      this.idPessoa = idPessoa;
19      this.cpfCnpj = cpfCnpj;
20      this.nome = nome;
21      this.dataNascimento = dataNascimento;
22      this.cidade = cidade;
23      this.login = login;
24      this.senha = senha;
25      this.foto = foto;
26    }
27  }

```

Fonte: O autor

Agora faça a geração dos métodos Get e Set através dos atalhos de sua IDE.

Figura 184 – Camada Model – Gerando os métodos Get e Set

```

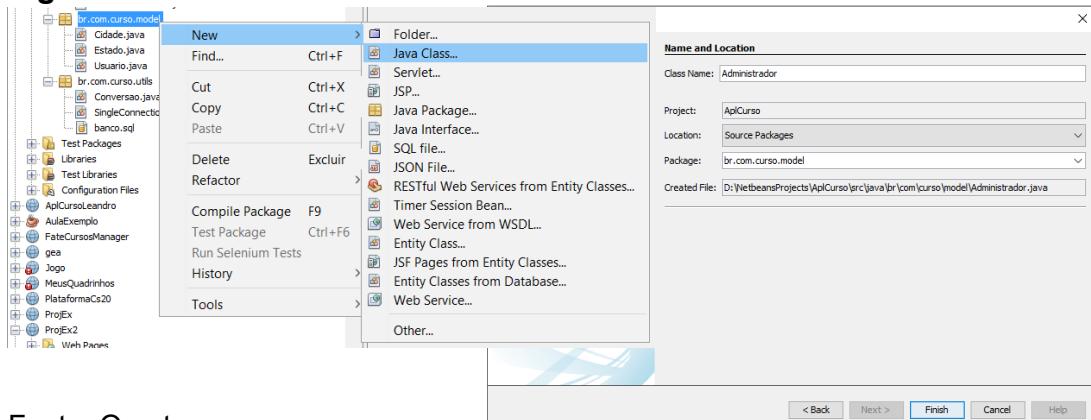
28  public int getIdPessoa() {
29    return idPessoa;
30  }
31
32  public void setIdPessoa(int idPessoa) {
33    this.idPessoa = idPessoa;
34  }
35
36  public String getCpfCnpj() {
37    return cpfCnpj;
38  }
39
40  public void setCpfCnpj(String cpfCnpj) {
41    this.cpfCnpj = cpfCnpj;
42  }
43
44  public String getNome() {
45    return nome;
46  }
47
48  public void setNome(String nome) {
49    this.nome = nome;
50  }
51
56  public void setDataNascimento(Date dataNascimento) {
57    this.dataNascimento = dataNascimento;
58  }
59
60  public Cidade getCidade() {
61    return cidade;
62  }
63
64  public void setCidade(Cidade cidade) {
65    this.cidade = cidade;
66  }
67
68  public String getLogin() {
69    return login;
70  }
71
72  public void setLogin(String login) {
73    this.login = login;
74  }
75
76  public String getSenha() {
77    return senha;
78  }
79
80  public void setSenha(String senha) {
81    this.senha = senha;
82  }
83
84  public String getFoto() {
85    return foto;
86  }
87
88  public void setFoto(String foto) {
89    this.foto = foto;
90  }
91

```

Fonte: O autor

Agora vamos desenvolver na nossa camada model a classe java “Administrador” conforme o modelo de dados definido no início deste capítulo. Para isso clique com o botão direito em seu pacote da camada model em seu projeto, “br.com.curso.model”.

Escolha a opção “New → Java Class” e na janela coloque o nome da classe “Despesa” e confirme sua criação, como podemos observar **Figura 185 – Camada Model – Criando a classe Administrador**.

Figura 185 – Camada Model – Criando a classe Administrador.

Fonte: O autor

A classe Administrador é nossa primeira classe filha em nossa estrutura de generalização/especialização, vamos então digitar os atributos de nossa classe de acordo como modelo que foi demonstrado no início deste capítulo.

Figura 186 – Camada Model – Criando os atributos da classe Administrador

```

1 package br.com.curso.model;
2
3 public class Administrador {
4
5     private int idAdministrador;
6     private String permiteLogin;
7     private String situacao;
8
9 }
```

Fonte: O autor

Agora devemos estabelecer o vínculo de herança entre as classes Administrador e Pessoa, para isso adicione o código “extends Pessoa” como demonstrado na Figura 187.

Figura 187 – Camada Model – Estabelecendo a relação de herança

```

1 package br.com.curso.model;
2
3 import java.util.Date;
4
5 public class Administrador extends Pessoa {
6
7     private int idAdministrador;
8     private String permiteLogin;
9     private String situacao;
10
11 }
```

Fonte: O autor

Agora vamos criar nossos construtores, primeiramente vamos criar o construtor principal que é o construtor com parâmetros pois esse é responsável por criar a classe Administrador e juntamente cria também a classe Pessoa. Gere se código como demonstrado na **Figura 188**, utilizando os atalhos de geração de código do sua IDE.

Figura 188 – Camada Model – Criando construtor com parâmetros

```

1 package br.com.curso.model;
2
3 import java.util.Date;
4
5 public class Administrador extends Pessoa {
6
7     private int idAdministrador;
8     private String permiteLogin;
9     private String situacao;
10
11    public Administrador(int idAdministrador, String permiteLogin, String situacao, int idPessoa,
12                         String cpfCnpj, String nome, Date dataNascimento, Cidade cidade, String login,
13                         String senha, String foto) {
14        super(idPessoa, cpfCnpj, nome, dataNascimento, cidade, login, senha, foto);
15        this.idAdministrador = idAdministrador;
16        this.permiteLogin = permiteLogin;
17        this.situacao = situacao;
18    }
19
20}

```

Fonte: O autor

Durante a programação das camadas superiores de nossa aplicação web vamos precisar de utilizar um objeto de Administrador vazio o que deveria ser gerado por um construtor simples (sem parâmetros), mas em uma relação de herança não é possível criar este tipo de construtor.

Para resolvemos esse problema vamos criar um método que será responsável por criar os objetos vazios de Administrador, conforme a **Figura 189**.

Figura 189 – Camada Model – Criando método para gerar objetos vazios

```

22    public static Administrador administradorVazio() throws ParseException {
23        Cidade oCidade = new Cidade();
24        Date dataNascimento = Conversao.dataAtual();
25        Administrador oAdministrador = new Administrador(0, "S", "A", 0, "", "", dataNascimento, oCidade, "", "", null);
26        return oAdministrador;
27    }

```

Fonte: O autor

Ao criarmos esse método deveremos importar a classe “Conversao” do nosso pacote Utils, observe também que o método criado irá utilizar o construtor criado anteriormente para gerar um objeto vazio, facilitando nosso trabalho.

Também é interessante observar que o método foi criado como “static” o que nos permite utilizar este método sem instanciar um objeto desta classe, como veremos mais a frente durante nosso desenvolvimento.

Agora gere os métodos get’s e set’s utilizando as ferramentas de sua IDE, o código ficará como abaixo.

Figura 190 – Camada Model – Administrador (código final)

```

1 package br.com.curso.model;
2
3 import br.com.curso.utils.Conversao;
4 import java.text.ParseException;
5 import java.util.Date;
6
7 public class Administrador extends Pessoa {
8
9     private int idAdministrador;
10    private String permiteLogin;
11    private String situacao;
12
13    public Administrador(int idAdministrador, String permiteLogin, String situacao, int idPessoa,
14        String cpfCnpj, String nome, Date dataNascimento, Cidade cidade, String login,
15        String senha, String foto) {
16        super(idPessoa, cpfCnpj, nome, dataNascimento, cidade, login, senha, foto);
17        this.idAdministrador = idAdministrador;
18        this.permiteLogin = permiteLogin;
19        this.situacao = situacao;
20    }
21
22    public static Administrador administradorVazio() throws ParseException{
23        Cidade oCidade = new Cidade();
24        Data dataNascimento = Conversao.dataAtual();
25        Administrador oAdministrador = new Administrador(0,"S","A","","",dataNascimento,oCidade,"","");
26        return oAdministrador;
27    }
28
29    public int getIdAdministrador() {
30        return idAdministrador;
31    }
32
33    public void setIdAdministrador(int idAdministrador) {
34        this.idAdministrador = idAdministrador;
35    }
36
37    public String getPermiteLogin() {
38        return permiteLogin;
39    }
40
41    public void setPermiteLogin(String permiteLogin) {
42        this.permiteLogin = permiteLogin;
43    }
44
45    public String getSituacao() {
46        return situacao;
47    }
48
49    public void setSituacao(String situacao) {
50        this.situacao = situacao;
51    }
52 }
```

Fonte: O autor

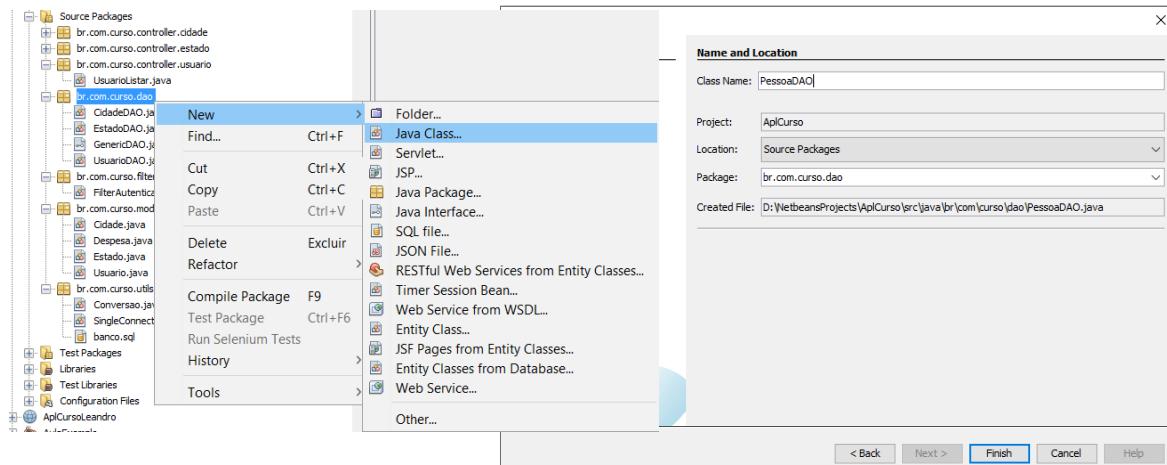
Agora repita os passos realizados anteriormente e crie as model’s de Cliente e Fornecedor seguindo os atributos conforme o nosso diagrama de classe.

9.2 DESENVOLVENDO A CAMADA DAO

Agora vamos criar nossa classe PessoaDAO em nosso pacote de camada DAO no nosso projeto. Para isso clique com o botão direito sobre “br.com.curso.dao”

e escolha “New → Java Class” e na janela de criação informe no nome da classe “PessoaDAO” e confirme.

Figura 191 – Camada DAO – Classe PessoaDAO



Fonte: O autor

Diferentemente das outras classes DAO que criamos anteriormente essa classe não implementará a interface “GenericDAO”, pois é uma classe DAO de uma model especial que é a classe pai de nossa generalização/especialização (Herança).

Agora vamos criar o atributo que irá armazenar nossa conexão, como já realizado em outras classes de camada DAO já desenvolvidas nesse projeto. Implemente em seu projeto conforme a linha 7 demonstrada na **Figura 192**.

Figura 192 – Camada DAO – Classe “PessoaDAO” – Atributo “conexão”

```

1 package br.com.curso.dao;
2
3 import java.sql.Connection;
4
5 public class PessoaDAO {
6     private Connection conexao;
7 }

```

Fonte: O autor

Agora crie o método construtor de nossa classe “PessoaDAO” conforme pode-se visualizar na Figura 193.

Figura 193 – Camada DAO – Classe “PessoaDAO” – Construtor

```

1 package br.com.curso.dao;
2
3 import br.com.curso.utils.SingleConnection;
4 import java.sql.Connection;
5
6 public class PessoaDAO {
7
8     private Connection conexao;
9
10    public PessoaDAO() throws Exception{
11        conexao = SingleConnection.getConnection();
12    }
13
14 }

```

Fonte: O autor

Agora implemente o método cadastrar().

Figura 194 – Camada DAO – Classe “PessoaDAO” – método cadastrar()

```

22 public int cadastrar(Object objeto) throws ParseException {
23     Pessoa oPessoa = (Pessoa) objeto;
24     int retorno = 0;
25     if (oPessoa.getIdPessoa()==0)
26     {
27         Pessoa objPessoa = this.carregarCpf(oPessoa.getCpfCnpj());
28         if (objPessoa.getIdPessoa()==0)
29             retorno = this.inserir(oPessoa);
30         else
31             retorno = objPessoa.getIdPessoa();
32     }
33     else {
34         retorno = this.alterar(oPessoa);
35     }
36     return retorno;
37 }

```

Fonte: O autor

Agora implemente o método inserir().

Figura 195 – Camada DAO – Classe “PessoaDAO” – inserir()

```

39  public int inserir(Object objeto) {
40      Pessoa oPessoa = (Pessoa) objeto;
41      PreparedStatement stmt = null;
42      ResultSet rs=null;
43      Integer idPessoa=null;
44      String sql = "insert into pessoa (cpfCnpj, nome, dataNascimento, idcidade, login,"
45          + "senha, foto) values (?, ?, ?, ?, ?, ?, ?) returning idPessoa";
46      try{
47          stmt = conexao.prepareStatement(sql);
48          stmt.setString(1, oPessoa.getCpfCnpj());
49          stmt.setString(2, oPessoa.getNome());
50          stmt.setDate(3, new java.sql.Date(oPessoa.getDataNascimento().getTime()));
51          stmt.setInt(4, oPessoa.getIdCidade());
52          stmt.setString(5, oPessoa.getLogin());
53          stmt.setString(6, oPessoa.getSenha());
54          stmt.setString(7, oPessoa.getFoto());
55          rs=stmt.executeQuery();
56          conexao.commit();
57
58          while (rs.next()){
59              idPessoa = rs.getInt("idPessoa");
60          }
61      } catch (SQLException e){
62          try {
63              System.out.println("Problemas ao cadastrar Pessoa!Erro: " + e.getMessage());
64              e.printStackTrace();
65              conexao.rollback();
66          } catch (SQLException ex) {
67              System.out.println("Problemas ao executar rollback" + ex.getMessage());
68              ex.printStackTrace();
69          }
70      }
71      return idPessoa;
72  }

```

Fonte: O autor

Agora implemente o método alterar().

Figura 196 – Camada DAO – Classe “PessoaDAO” – alterar()

```

74  public int alterar(Object objeto) {
75      Pessoa oPessoa = (Pessoa) objeto;
76      PreparedStatement stmt = null;
77      Integer idPessoa=oPessoa.getIdPessoa();
78      String sql = "update pessoa set nome=?, dataNascimento=?, idcidade=?,"
79          + "login=?, senha=?, foto=? "
80          + "where idpessoa=?;";
81      try{
82          stmt = conexao.prepareStatement(sql);
83          stmt.setString(1, oPessoa.getNome());
84          stmt.setDate(2, new java.sql.Date(oPessoa.getDataNascimento().getTime()));
85          stmt.setInt(3, oPessoa.getIdCidade());
86          stmt.setString(4, oPessoa.getLogin());
87          stmt.setString(5, oPessoa.getSenha());
88          stmt.setString(6, oPessoa.getFoto());
89          stmt.setInt(7, oPessoa.getIdPessoa());
90          stmt.execute();
91          conexao.commit();
92      } catch (SQLException e){
93          try {
94              System.out.println("Problemas ao alterar Pessoa!Erro: " + e.getMessage());
95              e.printStackTrace();
96              conexao.rollback();
97          } catch (SQLException ex) {
98              System.out.println("Problemas ao executar rollback" + ex.getMessage());
99              ex.printStackTrace();
100         }
101     }
102     return idPessoa;
103 }

```

Fonte: O autor

Agora implemente o método carregar().

Figura 197 – Camada DAO – Classe “PessoaDAO” – carregar()

```

105-□ public Pessoa carregar(int id) {
106    int idPessoa = id;
107    PreparedStatement stmt = null;
108    ResultSet rs = null;
109    Pessoa oPessoa = null;
110    String sql = "Select * from pessoa where idpessoa=?";
111
112    try{
113        stmt=conexao.prepareStatement(sql);
114        stmt.setInt(1, idPessoa);
115        rs=stmt.executeQuery();
116
117        while(rs.next()){
118
119            Cidade oCidade = null;
120            try{
121                CidadeDAO oCidadeDAO = new CidadeDAO();
122                int idCidade = rs.getInt("idcidade");
123                oCidade = (Cidade) oCidadeDAO.carregar(idCidade);
124            }catch(Exception ex){
125                System.out.println("Problemas ao carregar usuario! Erro:"+ex.getMessage());
126            }
127
128            oPessoa = new Pessoa(rs.getInt("idpessoa"),
129                                rs.getString("cpfcnpj"),
130                                rs.getString("nome"),
131                                rs.getDate("datanascimento"),
132                                oCidade,
133                                rs.getString("login"),
134                                rs.getString("senha"),
135                                rs.getString("foto"));
136        }
137        return oPessoa;
138    }catch(SQLException ex){
139        System.out.println("Problemas ao carregar pessoa! Erro "+ex.getMessage());
140        return null;
141    }
142}

```

Fonte: O autor

Agora implemente o método carregarCpf().

Figura 198 – Camada DAO – Classe “PessoaDAO” – carregarCpf()

```

144-□ public Pessoa carregarCpf(String cpf) throws ParseException {
145    PreparedStatement stmt = null;
146    ResultSet rs = null;
147    Pessoa oPessoa = null;
148    String sql = "Select * from pessoa where cpf=?;";
149
150    try{
151        stmt=conexao.prepareStatement(sql);
152        stmt.setString(1, cpf);
153        rs=stmt.executeQuery();
154        while (rs.next()){
155            oPessoa = this.carregar(rs.getInt("idpessoa"));
156        }
157        if (oPessoa == null)
158        {
159            Date novaData = Conversao.dataAtual();
160            Cidade oCidade = new Cidade();
161            oPessoa = new Pessoa(0,"","","novaData",oCidade,"","");
162        }
163    }catch(SQLException ex){
164        System.out.println("Problemas ao carregar pessoa! Erro:"+ex.getMessage());
165    }
166    return oPessoa;
167}
168

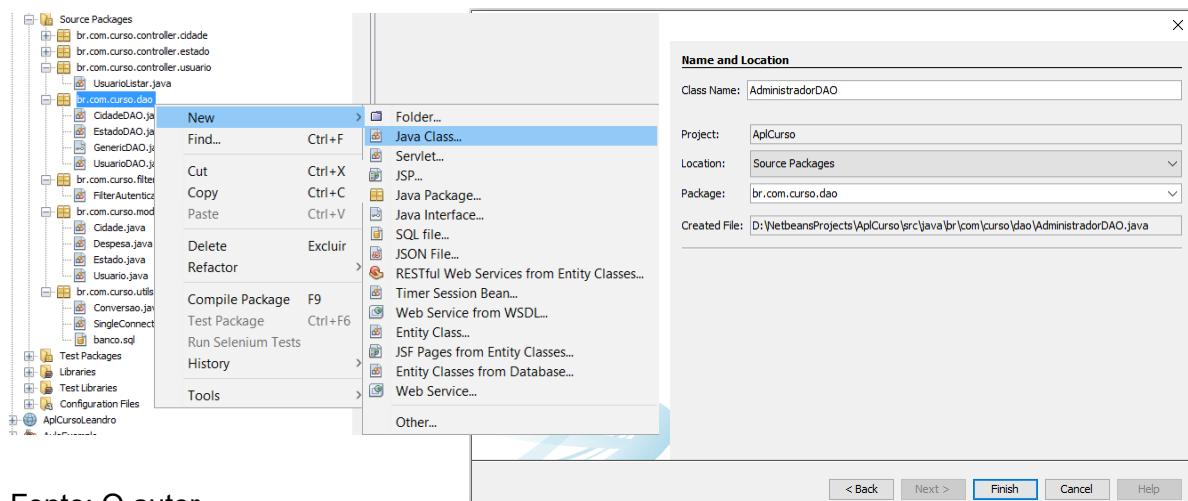
```

Fonte: O autor

9.2.1 Implementando DAO de Administrador

Agora vamos criar nossa classe AdministradorDAO em nosso pacote de camada DAO no nosso projeto. Para isso clique com o botão direito sobre “br.com.curso.dao” e escolha “New → Java Class” e na janela de criação informe no nome da classe “AdministradorDAO” e confirme.

Figura 199 – Camada DAO – Classe AdministradorDAO



Fonte: O autor

Nossa classe AdministradorDAO como já é padrão em nosso projeto irá implementar os métodos declarados na Interface “GenericDAO”, assim altere a declaração da classe como demonstrado na Figura 138.

Figura 200 – Camada DAO–Classe AdministradorDAO–Implementando GenericDAO

```

1 package br.com.curso.dao;
2
3 public class AdministradorDAO implements GenericDAO {
4     Implement all abstract methods
5     Make class AdministradorDAO abstract
6 }
```

Fonte: O autor

No momento que você informar para a classe DespesaDAO implementar a GenericDAO o Netbeans irá acusar erro que pode ser corrigido escolhendo na Dica da IDE a opção “Implements All abstract Methods”, como demonstrado na **Figura 200**.

Agora implemente no início de sua classe o atributo que irá armazenar a conexão com o banco de dados e o método construtor de nossa classe, conforme abaixo, nas linhas de 9 a 13.

Figura 201 – Camada DAO – Classe AdministradorDAO – GenericDAO

```

1 package br.com.curso.dao;
2
3 import br.com.curso.utils.SingleConnection;
4 import java.sql.Connection;
5 import java.util.List;
6
7 public class AdministradorDAO implements GenericDAO {
8
9     private Connection conexao;
10
11    public AdministradorDAO() throws Exception{
12        conexao = SingleConnection.getConnection();
13    }
14
15    @Override
16    public Boolean cadastrar(Object objeto) {
17        throw new UnsupportedOperationException("Not supported yet."); //To change
18    }

```

Fonte: O autor

Agora implemente o método cadastrar de nossa classe AdministradorDAO.

Figura 202 – Camada DAO – Classe AdministradorDAO– cadastrar()

```

19
20    @Override
21    public Boolean cadastrar(Object objeto) {
22        Boolean retorno = false;
23        try {
24            Administrador oAdministrador = (Administrador) objeto;
25            if (oAdministrador.getIdAdministrador()==0) { //inserção
26                //verifica se já existe pessoa com este CPF cadastrada.
27                int idAdministrador = this.verificarCpf(oAdministrador.getPfCnpj());
28                if (idAdministrador==0) {
29                    //se não encontrou insere
30                    retorno = this.inserir(oAdministrador);
31                } else{
32                    //se encontrou administrador com o cpf altera
33                    oAdministrador.setIdAdministrador(idAdministrador);
34                    retorno = this.alterar(oAdministrador);
35                }
36            } else {
37                retorno = this.alterar(oAdministrador);
38            }
39        } catch (Exception ex){
40            System.out.println("Problemas ao incluir administrador! Erro "+ex.getMessage());
41        }
42        return retorno;

```

Fonte: O autor

Na linha 26 ficará acusando erro do método “verificarCpf()” para retirar o erro vamos implementar esse método que não consta da nossa genericDAO. Vá após o seu último método gerado, tomando cuidado para não digitar fora da classe e crie o método conforme o código abaixo.

Figura 203 – Camada DAO – Classe AdministradorDAO– verificarCpf()

```

69  public int verificarCpf(String cpf){
70      PreparedStatement stmt = null;
71      ResultSet rs= null;
72      int idAdministrador = 0;
73      String sql = "Select c.* from administrador c, pessoa p "
74          + "where c.idpessoa = p.idPessoa and p.cpf=?;";
75      try{
76          stmt=conexao.prepareStatement(sql);
77          stmt.setString(1, cpf);
78          rs=stmt.executeQuery();
79          while(rs.next()){
80              idAdministrador = rs.getInt("idadministrador");
81          }
82          return idAdministrador;
83      }catch(SQLException ex){
84          System.out.println("Problemas ai carregar pessoa! Erro: "+ex.getMessage());
85          return idAdministrador;
86      }
87  }

```

Fonte: O autor

Agora implemente o método “inserir()” de nossa classe AdministradorDAO.

Figura 204 – Camada DAO – Classe AdministradorDAO– inserir()

```

44  @Override
45  public Boolean inserir(Object objeto) {
46      Administrador oAdministrador = (Administrador) objeto;
47      PreparedStatement stmt = null;
48      String sql = "insert into administrador (idPessoa, situacao, permiteLogin)"
49          + " values (?, ?, ?)";
50      try{
51          PessoaDAO oPessoaDAO = new PessoaDAO();
52          //manda informações para o cadastrar de pessoa.
53          int idPessoa = oPessoaDAO.cadastrar(oAdministrador);
54          stmt = conexao.prepareStatement(sql);
55          stmt.setInt(1, idPessoa);
56          stmt.setString(2, "A");
57          stmt.setString(3, oAdministrador.getPermiteLogin());
58          stmt.execute();
59          conexao.commit();
60          return true;
61      }catch(Exception e){
62          try {
63              System.out.println("Problemas ao cadastrar Administrador!Erro: " + e.getMessage());
64              e.printStackTrace();
65              conexao.rollback();
66          } catch (SQLException ex) {
67              System.out.println("Problemas ao executar rollback" + ex.getMessage());
68              ex.printStackTrace();
69          }
70          return false;
71      }
72  }

```

Fonte: O autor

Agora implemente o método “alterar()” de nossa classe AdministradorDAO.

Figura 205 – Camada DAO – Classe AdministradorDAO– alterar()

```

74  @Override
75  public Boolean alterar(Object objeto) {
76      Administrador oAdministrador = (Administrador) objeto;
77      PreparedStatement stmt = null;
78      String sql = "update administrador set permiteLogin=? where idAdministrador=?";
79      try{
80          PessoaDAO oPessoaDAO = new PessoaDAO();
81          oPessoaDAO.cadastrar(oAdministrador); //envia para classe PessoaDAO
82          stmt = conexao.prepareStatement(sql);
83          stmt.setString(1, oAdministrador.getPermiteLogin());
84          stmt.setInt(2, oAdministrador.getIdAdministrador());
85          stmt.execute();
86          conexao.commit();
87          return true;
88      }catch(Exception e){
89          try {
90              System.out.println("Problemas ao alterar Administrador!Erro: " + e.getMessage());
91              e.printStackTrace();
92              conexao.rollback();
93              conexao.rollback();
94          } catch (SQLException ex) {
95              System.out.println("Problemas ao executar rollback" + ex.getMessage());
96              ex.printStackTrace();
97          }
98          return false;
99      }
100 }

```

Fonte: O autor

Agora implemente o método “excluir()” de nossa classe AdministradorDAO.

Figura 206 – Camada DAO – Classe AdministradorDAO– excluir()

```

101 @Override
102 public Boolean excluir(int numero) {
103     PreparedStatement stmt = null;
104     try{
105         //carrega dados de administrador
106         AdministradorDAO oAdministradorDAO = new AdministradorDAO();
107         Administrador oAdministrador = (Administrador) oAdministradorDAO.carregar(numero);
108         String situacao="A";//verifica e troca a situação do administrador
109         if(oAdministrador.getSituacao().equals(situacao))
110             situacao = "I";
111         else situacao = "A";
112
113         String sql = "update administrador set situacao=? where idAdministrador=?";
114         stmt = conexao.prepareStatement(sql);
115         stmt.setString(1, situacao);
116         stmt.setInt(2, oAdministrador.getIdAdministrador());
117         stmt.execute();
118         conexao.commit();
119         return true;
120
121     }catch (Exception e){
122         try {
123             System.out.println("Problemas ao excluir Administrador!Erro: " + e.getMessage());
124             e.printStackTrace();
125             conexao.rollback();
126         } catch (SQLException ex) {
127             System.out.println("Problemas ao executar rollback" + ex.getMessage());
128             ex.printStackTrace();
129         }
130         return false;
131     }

```

Fonte: O autor

Agora implemente o método “carregar()” de nossa classe AdministradorDAO.

Figura 207 – Camada DAO – Classe AdministradorDAO– carregar()

```

134     @Override
135     public Object carregar(int numero) {
136         int idAdministrador = numero;
137         PreparedStatement stmt = null;
138         ResultSet rs = null;
139         Administrador oAdministrador = null;
140         String sql = "Select * from administrador c, pessoa p "
141             + "where c.idpessoa = p.idpessoa and c.idadministrador=?";
142         try{
143             stmt=conexao.prepareStatement(sql);
144             stmt.setInt(1, idAdministrador);
145             rs=stmt.executeQuery();
146             while(rs.next()){
147                 //Busca a cidade
148                 Cidade oCidade = null;
149                 try{
150                     CidadeDAO oCidadeDAO = new CidadeDAO();
151                     oCidade = (Cidade) oCidadeDAO.carregar(rs.getInt("idcidade"));
152                 }catch(Exception ex){
153                     System.out.println("Problemas ao carregar cidade!Erro:"+ex.getMessage());
154                 }
155                 oAdministrador = new Administrador(rs.getInt("idadministrador"),
156                     rs.getString("permitelogin"),
157                     rs.getString("situacao"),
158                     rs.getInt("idpessoa"),
159                     rs.getString("cpfcnpj"),
160                     rs.getString("nome"),
161                     rs.getDate("datanascimento"),
162                     oCidade,
163                     rs.getString("login"),
164                     rs.getString("senha"),
165                     rs.getString("foto"));
166             }
167         }catch(SQLException e){
168             System.out.println("Problemas ao carregar Administrador!Erro: " + e.getMessage());
169             e.printStackTrace();
170         }
171         return oAdministrador;
172     }

```

Fonte: O autor

Agora implemente o método “listar()” de nossa classe AdministradorDAO.

Figura 208 – Camada DAO – Classe AdministradorDAO– listar()

```

175     @Override
176     public List<Object> listar() {
177         List<Object> resultado = new ArrayList<>();
178         PreparedStatement stmt = null;
179         ResultSet rs = null;
180         String sql= "Select p.*, c.idadministrador, c.situacao, c.permitelogin "
181             + "from administrador c, pessoa p "
182             + "where c.idpessoa = p.idpessoa order by idPessoa";
183         try{
184             stmt = conexao.prepareStatement(sql);
185             rs = stmt.executeQuery();
186             while (rs.next()){
187                 Cidade oCidade = null;//busca cidade
188                 try{
189                     CidadeDAO oCidadeDAO = new CidadeDAO();
190                     oCidade = (Cidade) oCidadeDAO.carregar(rs.getInt("idcidade"));
191                 }catch(Exception ex){
192                     System.out.println("Problemas ao carregar usuario!Erro:"+ex.getMessage());
193                 }
194
195                 Administrador oAdministrador = new Administrador(rs.getInt("idadministrador"),
196                                         rs.getString("permiteLogin"),
197                                         rs.getString("situacao"),
198                                         rs.getInt("idpessoa"),
199                                         rs.getString("cpfcnpj"),
200                                         rs.getString("nome"),
201                                         rs.getDate("datanascimento"),
202                                         oCidade,
203                                         rs.getString("login"),
204                                         rs.getString("senha"),
205                                         rs.getString("foto"));
206                 resultado.add(oAdministrador);
207             }
208         }catch(SQLException ex){
209             System.out.println("Problemas ao listar administrador! Erro "+ex.getMessage());
210         }
211         return resultado;
212     }

```

Fonte: O autor

9.2.2 Implementando DAO de Cliente e Fornecedor

Agora, seguindo o exemplo anterior você deve implementar as classes DAO de Cliente e Fornecedor atentando-se para os atributos de cada uma das classes de acordo como o diagrama de classe que foi proposto.

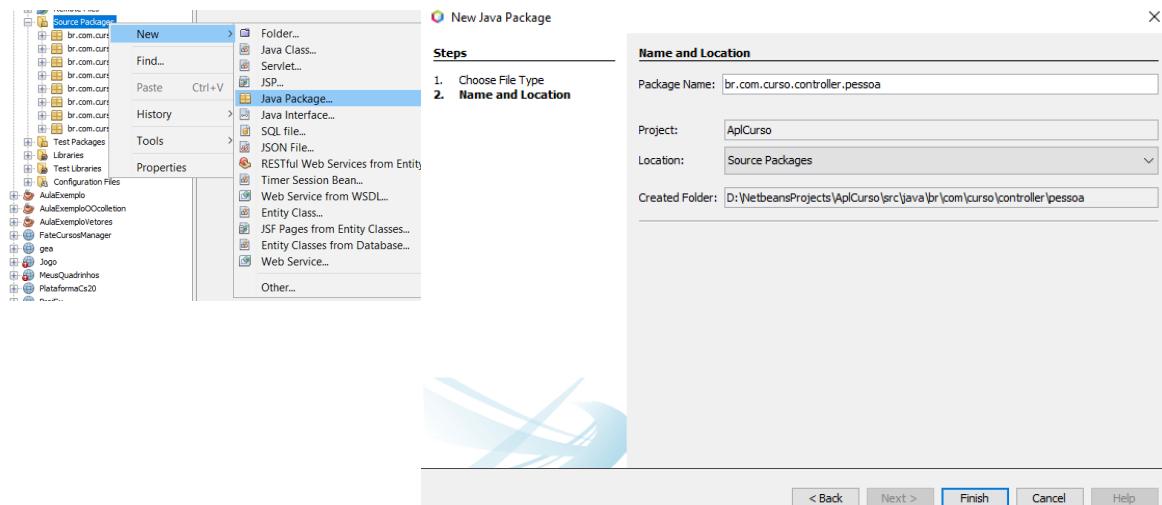
9.3 DESENVOLVENDO A CAMADA VIEW/CONTROLLER

Agora vamos desenvolver nossa camada controller e nossa camada view de administrador e você deve ir aplicando os conceitos para os cadastros de cliente e fornecedor.

9.3.1 Implementando a controller de Pessoa

Primeiramente vamos criar nosso pacote para armazenar nossos servlets para nosso Administrador, então crie o pacote java "br.com.curso.controller.administrador" como demonstrado na Figura 209.

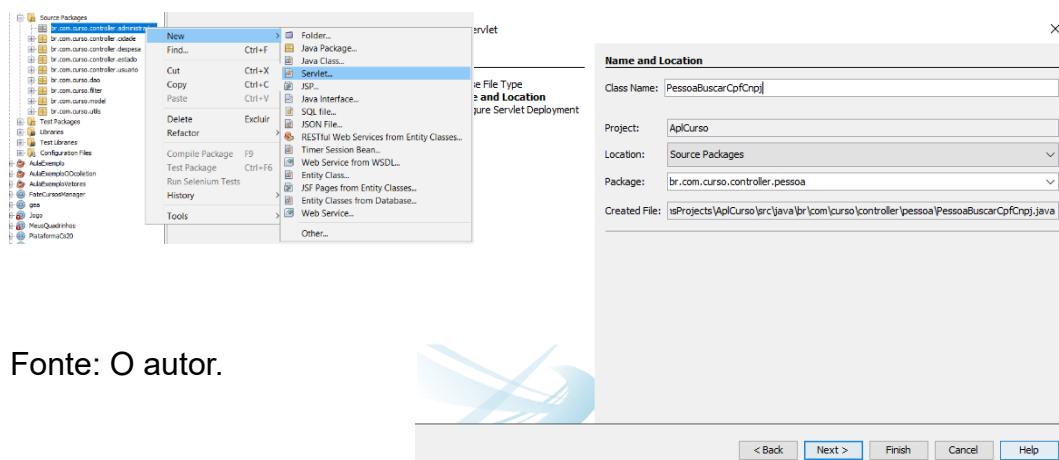
Figura 209 – Criando o pacote para controller de Pessoa



Fonte: O autor

Agora vamos criar o servlet PessoaBuscarCpfCnpj para podermos realizarmos uma busca por CPF/CNPJ na através da classe pessoa, para que seja possível o carregamento dos dados pré cadastrados ao incluir um novo administrador. Clique com o botão direito no pacote controller de Pessoa e escolha New→Servlet e faça como demonstrado na

Figura 210 – Criando o Servlet PessoaBuscarCpfCnpj



Fonte: O autor.

Agora vamos programar nosso servlet PessoaBuscarCpfCnpj, apague o conteúdo gerado automaticamente pelo Netbeans no método “processRequest” e substitua conforme demonstrado na **Figura 211**.

Figura 211 – Programando o Servlet PessoaBuscarCpfCnpj

```

38     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
39             throws ServletException, IOException {
40             response.setContentType("text/html;charset=iso-8859-1");
41         try{
42             String cpfCnpj = request.getParameter("cpfcnpjpessoa");
43             String tipoPessoa = request.getParameter("tipopessoa");
44             int id = 0;
45             String jsonRetorno="";
46             if (tipoPessoa.equals("administrador")){
47                 AdministradorDAO oAdmDAO = new AdministradorDAO();
48                 //busca Adm por cpf.
49                 Administrador oAdm = (Administrador) oAdmDAO.carregar(oAdmDAO.verificarCpf(cpfCnpj));
50                 //gera retorno
51                 Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd").create();
52                 jsonRetorno = gson.toJson(oAdm);
53             } else if (tipoPessoa.equals("cliente")){
54                 //implementar
55
56             } else if (tipoPessoa.equals("fornecedor")){
57                 //implementar
58
59             } else {
60                 //não tem ADM/Cliente ou Forn. -- então verifica Pessoa por CPF
61                 PessoaDAO oPessoaDAO = new PessoaDAO();
62                 Pessoa oPessoa = oPessoaDAO.carregarCpf(cpfCnpj);
63
64                 Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd").create();
65                 jsonRetorno = gson.toJson(oPessoa);
66             }
67
68             response.setCharacterEncoding("iso-8859-1");
69             response.getWriter().write(jsonRetorno);
70
71         } catch (Exception ex) {
72             System.out.println("Problemas ao carregar pessoa por CPF/CNPJ"
73                         + " Erro: " + ex.getMessage());
74         }
75     }

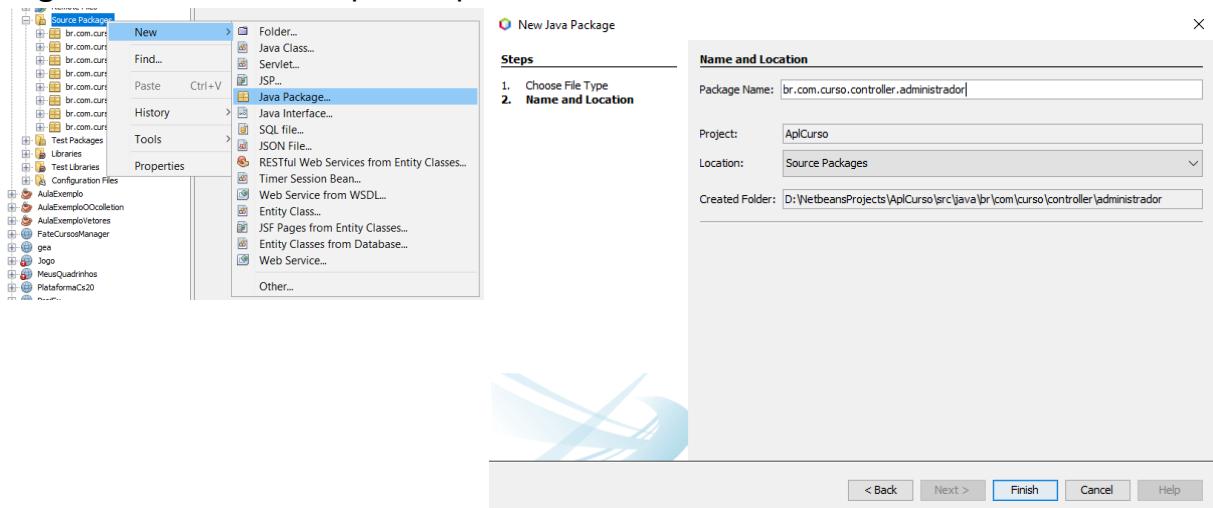
```

Fonte: O autor.

9.3.2 Implementando a controller de Administrador

Primeiramente vamos criar nosso pacote para armazenar nossos servlets para nosso Administrador, então crie o pacote java “br.com.curso.controller.administrador” como demonstrado na **Figura 212**.

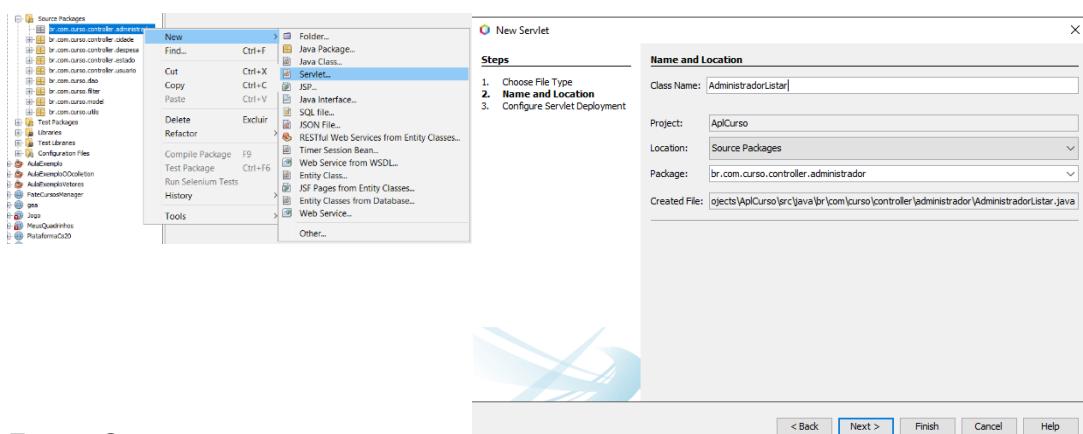
Figura 212 – Criando o pacote para controller de Administrador



Fonte: O autor

Agora vamos criar o servlet AdministradorListar para podermos alimentarmos de informações nossa lista de administradores na tela de nosso sistema. Clique com o botão direito no pacote controller de administrador e escolha New→Servlet e faça como demonstrado na **Figura 213**.

Figura 213 – Criando o Servlet AdministradorListar



Fonte: O autor

Agora vamos programar nosso servlet AdministradorListar, apague o conteúdo gerado automaticamente pelo Netbeans no método “processRequest” e substitua conforme demonstrado na Figura 214.

Figura 214 – Programando o Servlet AdministradorListar

```

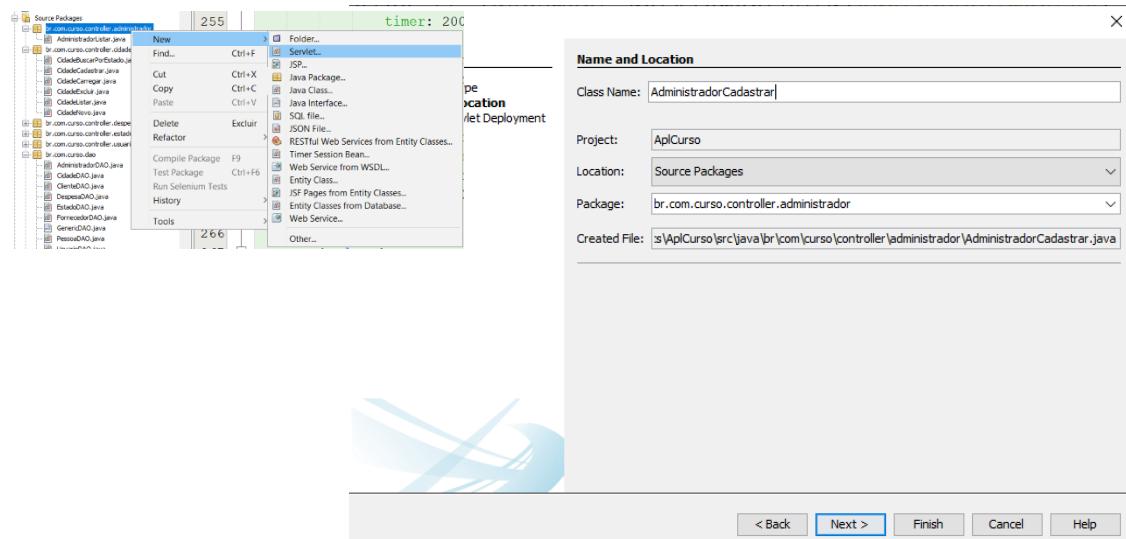
35     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
36         throws ServletException, IOException {
37             response.setContentType("text/html;charset=iso-8859-1");
38             try{
39                 GenericDAO dao = new AdministradorDAO();
40                 request.setAttribute("administradores",dao.listar());
41                 GenericDAO oEstadoDAO = new EstadoDAO();
42                 request.setAttribute("estados", oEstadoDAO.listar());
43                 request.getRequestDispatcher("/cadastros/administrador/administrador.jsp")
44                     .forward(request, response);
45             } catch (Exception ex){
46                 System.out.println("Problemas no Servlet ao Listar"
47                         + "Administrador! Erro: " + ex.getMessage());
48             }
49         }

```

Fonte: O autor

Agora vamos criar o servlet AdministradorCadastrar que receberá a requisição ajax de nosso view de administrador. Clique com o botão direito no pacote controller de administrador e escolha New→Servlet e faça como demonstrado na**Figura 224 – Criando pasta para interface de Administrador. Figura 218.**

Figura 215 – Controller – Criando o Servlet AdministradorCadastrar



Fonte: O autor

Agora vamos programar nosso servlet AdministradorCadastrar, apague o conteúdo gerado automaticamente pelo Netbeans no método “processRequest” e substitua conforme demonstrado na **Figura 216**.

Figura 216 – Controller – Codificando o Servlet AdministradorCadastrar

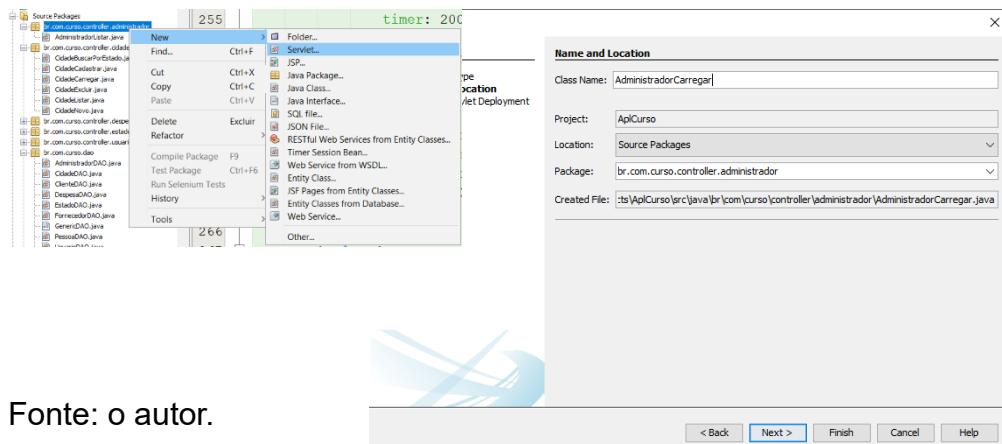
```

36     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37         throws ServletException, IOException {
38             response.setContentType("text/html;charset=iso-8859-1");
39             String mensagem = null;
40             try{
41                 //busca parametros do formulario (ajax) - view
42                 int idPessoa = Integer.parseInt(request.getParameter("idpessoa"));
43                 int idAdministrador = Integer.parseInt(request.getParameter("idadministrador"));
44                 String cpfCnpjPessoa = request.getParameter("cpfcnpjpessoa");
45                 String nomePessoa = request.getParameter("nomepessoa");
46                 Date dataNascimento = Date.valueOf(request.getParameter("datanascimento"));
47                 int idCidade = Integer.parseInt(request.getParameter("idcidade"));
48                 String login = request.getParameter("login");
49                 String senha = request.getParameter("senha");
50                 String permitelogin = request.getParameter("permitelogin");
51                 String situacao = request.getParameter("situacao");
52                 String fotoPessoa = request.getParameter("fotopessoa");
53
54                 //cria objeto de cidade.
55                 Cidade oCidade = new Cidade();
56                 oCidade.setIdCidade(idCidade);
57
58                 //gera objeto de administrador
59                 Administrador oAdministrador = new Administrador(idAdministrador, permitelogin, situacao,
60                     idPessoa, cpfCnpjPessoa, nomePessoa, dataNascimento, oCidade, login, senha,
61                     fotoPessoa);
62                 //instancia camada dao de administrador
63                 AdministradorDAO dao = new AdministradorDAO();
64
65                 if(dao.cadastrar(oAdministrador)){
66                     //mensagem = "Cadastrado com Sucesso!";
67                     response.getWriter().write("1");
68                 }else{
69                     //mensagem = "Problemas ao cadastrar Despesa!";
70                     response.getWriter().write("0");
71                 }
72             } catch (Exception e) {
73                 System.out.println("Problemas no servelet Cadastrar Administrador!Erro: " + e.getMessage());
74                 e.printStackTrace();
75             }
76         }
    
```

Fonte: O autor

Agora vamos criar o servlet AdministradorCarregar que receberá a requisição ajax de nosso view de administrador. Clique com o botão direito no pacote controller de administrador e escolha New→Servlet e faça como demonstrado na **Figura 217**.

Figura 217 – Controller – Criando o Servlet AdministradorCarregar



Fonte: o autor.

Agora vamos programar nosso servlet AdministradorCarregar, apague o conteúdo gerado automaticamente pelo Netbeans no método “processRequest” e substitua conforme demonstrado na **Figura 218**.

Figura 218 – Controller – Codificando o Servlet AdministradorCarregar

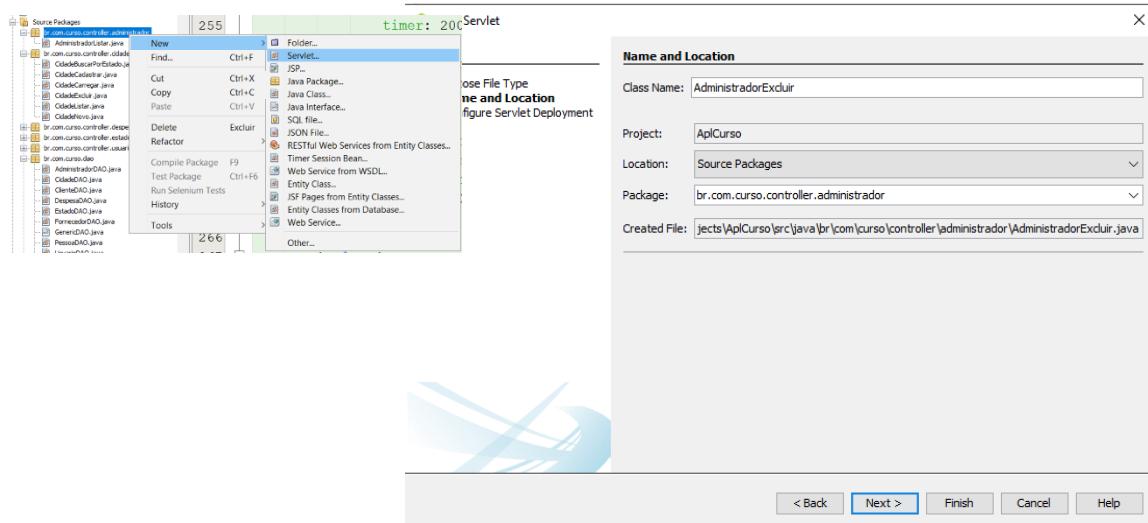
```

36 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37     throws ServletException, IOException {
38     response.setContentType("text/html;charset=iso-8859-1");
39     try{
40         int idAdministrador = Integer.parseInt(request.getParameter("idAdministrador"));
41         AdministradorDAO dao = new AdministradorDAO();
42         Administrador oAdm = (Administrador) dao.carregar(idAdministrador);
43
44         Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd").create();
45         String json = gson.toJson(oAdm);
46         response.getWriter().write(json);
47
48     }catch(Exception ex){
49         System.out.println("Erro ao buscar administrador- "+ex.getMessage());
50         ex.printStackTrace();
51     }
52 }
```

Fonte: O autor

Agora vamos criar o servlet AdministradorExcluir que receberá a requisição ajax de nosso view de administrador com o id do administrador a ser inativado no sistema. Clique com o botão direito no pacote controller de administrador e escolha New→Servlet e faça como demonstrado na **Figura 219**.

Figura 219 – Controller – Criando o Servlet AdministradorExcluir



Fonte: O autor

Agora vamos programar nosso servlet AdministradorExcluir, apague o conteúdo gerado automaticamente pelo Netbeans no método “processRequest” e substitua conforme demonstrado na **Figura 220**.

Figura 220 – Controller – Codificando o Servlet AdministradorExcluir

```

33 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
34     throws ServletException, IOException {
35     response.setContentType("text/html;charset=iso-8859-1");
36     int idDespesa = Integer.parseInt(request.getParameter("idAdministrador"));
37     String mensagem = null;
38     try {
39         AdministradorDAO dao = new AdministradorDAO();
40         if(dao.excluir(idDespesa)){
41             response.getWriter().write("1");
42         }else{
43             response.getWriter().write("0");
44         }
45     } catch (Exception e){
46         System.out.println("Problemas na Servelet Excluir Administrador!Erro: " + e.getMessage());
47         e.printStackTrace();
48     }
49 }
```

Fonte: O autor

9.3.3 Implementando Servlet de atualização de lista de cidades

Quando nossa interface estiver funcionando teremos que escolher a cidade e o estado do administrador, assim é desejável que o usuário que estiver operando o sistema escolha o estado e o sistema atualize a lista de cidades automaticamente, para isso vamos criar um método **listar por estado** em nossa **CidadeDAO**. Deste modo crie um método após o método listar já existente, conforme demonstrado na **Figura 221**.

Figura 221 – Criando método listar(int estado) em CidadeDAO.

```

142     @Override
143     public List<Object> listar() { ...30 lines }
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173     public List<Cidade> listar(int idEstado) {
174         List<Cidade> resultado = new ArrayList<>();
175         PreparedStatement stmt = null;
176         ResultSet rs = null;
177         String sql = "Select * from cidade where idestado = ? order by nomecidade";
178         try{
179             stmt = conexao.prepareStatement(sql);
180             stmt.setInt(1, idEstado);
181             rs = stmt.executeQuery();
182             while(rs.next()){
183                 Cidade oCidade = new Cidade();
184                 oCidade.setIdCidade(rs.getInt("idcidade"));
185                 oCidade.setNomeCidade(rs.getString("nomecidade"));
186
187                 try{
188                     EstadoDAO oEstadoDAO = new EstadoDAO();
189                     oCidade.setEstado((Estado) oEstadoDAO.carregar(rs.getInt("ideestado")));
190                 }catch(Exception ex){
191                     System.out.println("Erro ao carregar estado"+ex.getMessage());
192                     ex.printStackTrace();
193                 }
194                 resultado.add(oCidade);
195             }
196         }
197         }catch(SQLException ex){
198             System.out.println("Problemas ao listar Cidade! Erro: "+ex.getMessage());
199         }
200         return resultado;
201     }

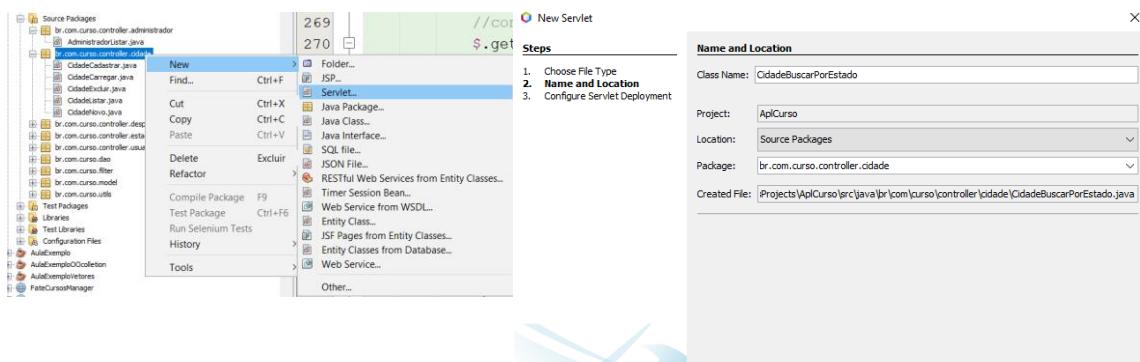
```

Fonte: O autor

Esse método deverá ser acionado por um servlet que receberá a requisição ajax quando o usuário do sistema alterar o estado e assim solicitar uma lista de cidades atualizada para aquele estado específico.

Vamos criar um servlet no pacote de “br.com.curso.controller.cidade” denominado “CidadeBuscarPorEstado” este servlet é responsável por atualizar a lista de cidades no momento em que o usuário escolher na tela um estado. Para isso faça como demonstrado na **Figura 222**.

Figura 222 – Criando Servlet – CidadeBuscarPorEstado



Fonte: O autor

Agora faça o desenvolvimento do método “processRequest” do servlet “CidadeBuscarPorEstado” de acordo com a **Figura 223**.

Figura 223 – Programando Servlet “CidadeBuscarPorEstado”

```

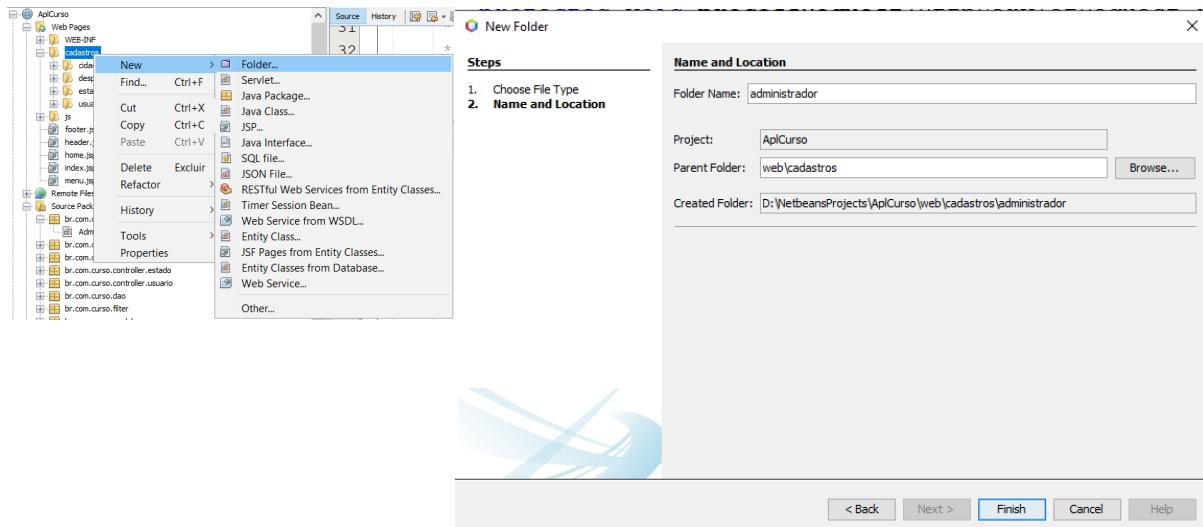
36     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37         throws ServletException, IOException {
38     response.setContentType("text/html;charset=iso-8859-1");
39     try {
40         int idEstado = Integer.parseInt(request.getParameter("idestado"));
41
42         CidadeDAO oCidadeDAO = new CidadeDAO();
43         List<Cidade> lstCidades = oCidadeDAO.listar(idEstado);
44
45         Gson gson = new Gson();
46         String jsonCidades = gson.toJson(lstCidades);
47
48         response.setCharacterEncoding("iso-8859-1");
49         response.getWriter().write(jsonCidades);
50
51     } catch (Exception ex) {
52         System.out.println("Problemas ao validar cpf/cnpj"
53                         + " Erro: " + ex.getMessage());
54     }
55 }
```

Fonte: O autor

9.3.4 Implementando a Interface de Cadastro de Administrador

Com os servlets implementados vamos construir nossa JSP na camada View. Primeiro crie uma pasta para armazenar os arquivos referente a interface de administrador, conforme demonstrado na **Figura 224**.

Figura 224 – Criando pasta para interface de Administrador.



Fonte: O autor

Antes de criarmos efetivamente nossa interface “JSP” vamos importar a biblioteca “app.js” fornecida junto com esse tutorial, essa biblioteca de funções javascript tem como objetivo controlar o campo “cpfcnpjpessoa” de nosso formulário de cadastro de forma a colocar e retirar máscara de acordo com o seu tipo CPF ou CNPJ e ainda fazer a validação do número digitado.

Então arraste o arquivo “app.js” para a pasta “js” de seu projeto, e inclua a importação do arquivo no “header.jsp” conforme demonstrado na Figura 225.

Figura 225 – Importando “app.js” no arquivo “header.jsp”

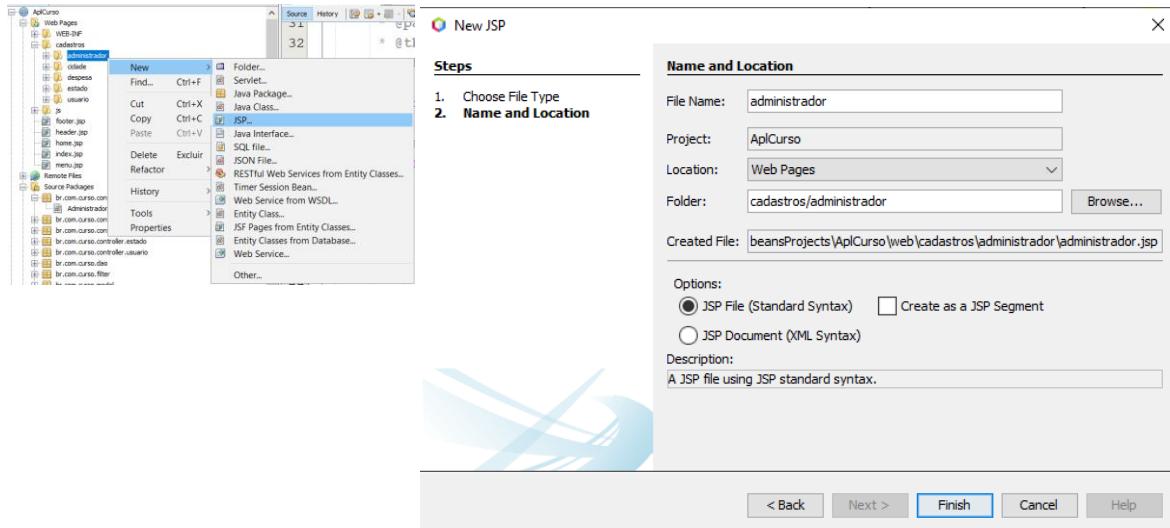
```

3 | <html>
4 |   <head>
5 |     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6 |     <title>JSP Page</title>
7 |     <!-- JQuery -->
8 |     <script src="${pageContext.request.contextPath}/js/jquery-3.3.1.min.js"></script>
9 |     <script src="${pageContext.request.contextPath}/js/jquery.mask.min.js"></script>
10 |    <script src="${pageContext.request.contextPath}/js/jquery.maskMoney.min.js"></script>
11 |
12 |    <!-- Importação da minha biblioteca de javascript -->
13 |    <script src="${pageContext.request.contextPath}/js/app.js" type="text/javascript"></script>
14 |
15 |
16 |    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
17 |    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/umd/popper.min.js"></script>
18 |    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
19 |
20 |    <!-- Datatable -->
21 |    <link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/1.10.22/css/jquery.dataTables.min.css"/>
22 |    <script src="https://cdn.datatables.net/1.10.22/js/jquery.dataTables.min.js" type="text/javascript"></script>
23 |
24 |    <!-- Mensagem alerta -->
25 |    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@10.3.1/dist/sweetalert2.all.min.js" type="text/javascript">
26 |    </script>
27 |
28 |  </head>
|  <body>
```

Fonte: O autor

Agora podemos criar o arquivo JSP para o Administrador. Proceda como demonstrado na Figura 226 e crie o arquivo administrador.jsp.

Figura 226 – Criando View – administrador.jsp



Fonte: O autor

Então podemos desenvolver nossa interface para administrador no arquivo “administrador.jsp”, neste exemplo de cadastro iremos trabalhar com formulários do tipo modal e além de utilizarmos todas as tecnologias já abordadas anteriormente em nosso projeto. Assim implemente o seu JSP de acordo com o demonstrado na **Figura 227**.

Figura 227 – View – Implementando o administrador.jsp

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4 <%@include page="/header.jsp"/>
5 <%@include page="/menu.jsp"/>
6
7 <div class="container-fluid">
8   <!-- Page Heading -->
9   <h1 class="h3 mb-2 text-gray-800">Administrador</h1>
10  <p class="mb-4">Planilha de Registro</p>
11  <a href="#modaladicionar" class="btn btn-success mb-4 adicionar" data-toggle="modal"
12    data-id="" onclick="setDadosModal(${0})">
13    <i class="fas fa-plus fa-fw"></i> Adicionar </a>
14 <div class="card shadow">
15   <div class="card-body">
16     <table id="datatable" class="display">
17       <thead>
18         <tr>
19           <th align="center">ID</th>
20           <th align="center">Nome</th>
21           <th align="center">CPF/CNPJ</th>
22           <th align="center">Cidade - UF</th>
23           <th align="center">Alterar</th>
24           <th align="center">Excluir</th>
25         </tr>
26       </thead>

```

Continuação

```

27 | <tbody>
28 |   <c:forEach var="administrador" items="${administradores}">
29 |     <tr>
30 |       <td align="center">${administrador.idAdministrador}</td>
31 |       <td align="center">${administrador.nome}</td>
32 |       <td align="center">${administrador.cpfCnpj}</td>
33 |       <td align="center">${administrador.cidade.nomeCidade} - ${administrador.cidade.estado.siglaEstado}</td>
34 |       <td align="center">
35 |         <a href="#modaladicionar" class="btn btn-success adicionar" data-toggle="modal"
36 |           data-id="" onclick="setDadosModal(${administrador.idAdministrador})">
37 |           <i class="fas fa-plus fa-fw"></i> Alterar </a>
38 |       </td>
39 |       <td align="center">
40 |         <a href="#" onclick="deletar(${administrador.idAdministrador})">
41 |           <button class="btn"
42 |             <c:out value="${administrador.situacao == 'A' ? 'btn-danger': 'btn-success'}"/>
43 |             <i class="fas fa-fw
44 |               <c:out value="${administrador.situacao == 'A' ? 'fa-times' : 'fa-plus'}"/></i>>
45 |           <Strong>
46 |             <c:out value="${administrador.situacao == 'A' ? 'Inativar': 'Ativar'}"/>
47 |           </Strong>
48 |         </button></a>
49 |       </td>
50 |     </tr>
51 |   </c:forEach>
52 | </tbody>

53 |       </table>
54 |     </div>
55 |   </div>
56 |
57 | <div class="modal fade" id="modaladicionar" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">
58 |   <div class="modal-dialog modal-xl">
59 |     <div class="modal-content">
60 |       <div class="modal-header">
61 |         <h5 class="modal-title">Adicionar</h5>
62 |         <button type="button" class="close" data-dismiss="modal" aria-label="Close">
63 |           <span aria-hidden="true">&times;</span>
64 |         </button>
65 |       </div>
66 |       <div class="modal-body">
67 |         <div class="form-group">
68 |           <img alt="imagem" class=""
69 |             name="foto" id="foto" width="85" height="100">
70 |           <input type="file" id="gallery-photo-add" class="inputfile" onchange="uploadFile();"/>
71 |           <label for="gallery-photo-add" class="btn btn-success">
72 |             <i class="fas fa-file-upload"></i>
73 |             Selecionar Foto...
74 |           </label>
75 |         </div>

76 |       <div class="form-group">
77 |         <input class="form-control" type="text" name="idpessoa" id="idpessoa" value=""
78 |           readonly="readonly"/>
79 |         <input class="form-control" type="text" name="idadministrador" id="idadministrador"
80 |           value="" readonly="readonly"/>
81 |         <input class="form-control" type="text" name="situacao" id="situacao"
82 |           value="" readonly="readonly"/>
83 |
84 |       </div>
85 |
86 |       <div class="form-group">
87 |         <label>CPF/Cnpj</label>
88 |         <input class="form-control" type="text" name="cpfcnpjpessoa" id="cpfcnpjpessoa"
89 |           value="" />
90 |       </div>
91 |
92 |       <div class="form-group">
93 |         <label>Nome/Razão Social</label>
94 |         <input class="form-control" type="text" name="nomepessoa" id="nomepessoa"/>
95 |       </div>
96 |

```

Continuação

```

97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116

<div class="form-group">
    <div class="form-line row">
        <div class="col-sm">
            <label class="m-t-0 header-title">Data Nascimento</label>
            <input class="form-control" type="date" name="datanascimento" id="datanascimento"
                   value="" />
        </div>
        <div class="col-sm">
            <label>Estado</label>
            <select class="form-control" name="idestado" id="idestado"
                   onchange="BuscarCidadesPorEstado()" required>
                <option value="nulo">Selecione</option>
                <c:forEach var="estado" items="${estados}">
                    <option value="${estado.idEstado}" ${administrador.cidade.estado.idEstado == estado.idEstado ? "selected" : ""}>
                        ${estado.nomeEstado}
                    </option>
                </c:forEach>
            </select>
        </div>
    </div>

    <div class="col-sm">
        <label>Cidade</label>
        <select class="form-control" name="idcidade" id="idcidade" required>
            <option value="nulo">Selecione</option>
            <c:forEach var="cidade" items="${cidades}">
                <option value="${cidade.idCidade}" ${administrador.cidade.idCidade == cidade.idCidade ? "selected" : ""}>
                    ${cidade.nomeCidade}
                </option>
            </c:forEach>
        </select>
    </div>
</div>

<div class="form-group">
    <div class="form-line row">
        <div class="col-sm">
            <label>Login</label>
            <input class="form-control" type="text" name="login" id="login" value="" size="20"
                   maxlength="20" required/>
        </div>
    </div>
</div>

139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
    <div class="col-sm">
        <label>Senha</label>
        <input class="form-control" type="password" name="senha" id="senha" value="" size="20"
               maxlength="20" required/>
    </div>
    <div class="col-sm">
        <label>Permite Login</label>
        <select class="form-control" name="permiteLogin" id="permiteLogin">
            <option value="N" ${administrador.permiteLogin == 'N' ? "selected" : ""}>Não</option>
            <option value="S" ${administrador.permiteLogin == 'S' ? "selected" : ""}>Sim</option>
        </select>
    </div>
</div>

<div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancelar</button>
    <a href="#" onclick="validarCampos()">
        <button type="button" class="btn btn-success">Salvar</button>
    </a>
</div>
</div>
</div>

```

Continuação

```

165 <style type="text/css">
166     .inputfile {
167         /* visibility: hidden etc. wont work */
168         width: 0.1px;
169         height: 0.1px;
170         opacity: 0;
171         overflow: hidden;
172         position: absolute;
173         z-index: -1;
174     }
175     .inputfile:focus + label {
176         /* keyboard navigation */
177         outline: 1px dotted #000;
178         outline: -webkit-focus-ring-color auto 5px;
179     }
180     .inputfile + label * {
181         pointer-events: none;
182     }
183     .borda{
184         position: relative;
185         margin: 0 20px 30px 0;
186         padding: 10px;
187         border: 1px solid #elele1;
188         border-radius: 3px;
189         background: #fff;
190         -webkit-box-shadow: 0px 0px 3px rgba(0,0,0,0.06);
191         -moz-box-shadow: 0px 0px 3px rgba(0,0,0,0.06);
192         box-shadow: 0px 0px 3px rgba(0,0,0,0.06);
193     }
194 </style>
195

```

CSS para configuração do botão de inserir a foto da pessoa.

```

196 <script>
197     $(document).ready(function() {
198         console.log('entrei ready');
199         //Carregamos a datatable
200         //$("#datatable").DataTable({});;
201         $('#datatable').DataTable({
202             "oLanguage": {
203                 "sProcessing": "Processando...",
204                 "sLengthMenu": "Mostrar _MENU_ registros",
205                 "sZeroRecords": "Nenhum registro encontrado.",
206                 "sInfo": "Mostrando de _START_ até _END_ de _TOTAL_ registros",
207                 "sInfoEmpty": "Mostrando de 0 até 0 de 0 registros",
208                 "sInfoFiltered": "",
209                 "sInfoPostfix": "",
210                 "sSearch": "Buscar:",
211                 "sUrl": "",
212                 "oPaginate": {
213                     "sFirst": "Primeiro",
214                     "sPrevious": "Anterior",
215                     "sNext": "Seguinte",
216                     "sLast": "Último"
217                 }
218             }
219         });
220     });
221

```

Configura a tabela de dados com o DataTables.

```

222     var cidade = ''; //variavel para controle do carregamento de cidades.
223     function limparDadosModal(){
224         $('#idpessoa').val("0");
225         $('#idadministrador').val("0");
226         $('#situacao').val("");
227         $('#cpfcnpjpessoa').val("");
228         $('#nomepessoa').val("");
229         $('#datanascimento').val("");
230         $('#idestado').val("0");
231         //cidade = 0;
232         //BuscarCidadesPorEstado(); //atualiza lista de cidades
233         $('#login').val("");
234         $('#senha').val("");
235         $('#permitelogin').val("S");
236         foto.src = "";
237     }
238

```

Limpa campos na tela.

Continuação

```
239 function setDadosModal(valor) {
240     limparDadosModal();
241     var foto = document.getElementById("foto");
242     document.getElementById('idpessoa').value = valor;
243     document.getElementById('idadministrador').value = valor;
244     var idAdm = valor;
245     if (idAdm != "0"){
246         //existe administrador para buscar (alteração)
247         $.getJSON('AdministradorCarregar', {idAdministrador: idAdm}, function(respostaServlet){
248             console.log(respostaServlet);
249             var id = respostaServlet.idAdministrador;
250             if(id != "0"){
251                 $('#idpessoa').val(respostaServlet.idPessoa);
252                 $('#idadministrador').val(respostaServlet.idAdministrador);
253                 $('#situacao').val(respostaServlet.situacao);
254                 $('#cpfcnpjpessoa').val(respostaServlet.cpfCnpj);
255                 $('#nomepessoa').val(respostaServlet.nome);
256                 $('#datanascimento').val(respostaServlet.dataNascimento);
257                 $('#idendade').val(respostaServlet.cidade.estado.idEstado);
258                 cidade = respostaServlet.cidade.idCidade;
259                 BuscarCidadesPorEstado(); //atualiza lista de cidades
260                 $('#login').val(respostaServlet.login);
261                 $('#senha').val(respostaServlet.senha);
262                 $('#permiteLogin').val(respostaServlet.permiteLogin);
263                 foto.src = respostaServlet.foto;
```

Carrega dados na tela
na alteração através
do servlet
AdministradorCarregar

```
264     }
265   });
266 }
267 }
268
269 function carregarPessoa(v) {
270   //console.log("Entrou");
271   var idM = v;
272   var tipoPessoa = 'administrador';
273   //console.log("Usuario = " + idM);
274   $(document).ready(function () {
275     $.getJSON('PessoaBuscarCpfCnpj', {cpfcnpipessoa: idM, tipopessoa: tipoPessoa}, function (respostaAdm) {
```

Verifica se já existe
Adm com CPF/CNPJ
cadastrado!!
Senão busca por
pessoa

```
276 console.log(respostaAdm);
277 //var id = respostaAdm.idAdministrador;
278 if (respostaAdm != null)
279 {
280     $('#idadministrador').val(respostaAdm.idAdministrador);
281     $('#permiteLogin').val(respostaAdm.permiteLogin);
282     $('#situacao').val(respostaAdm.situacao);
283     $('#idpessoa').val(respostaAdm.idPessoa);
284     $('#nomepessoa').val(respostaAdm.nome);
285     $('#login').val(respostaAdm.login);
286     var datanasc = respostaAdm.dataNascimento;
287     console.log(datanasc);
288
289     $('#datanascimento').val(datanasc);
290     $('#idestado').val(respostaAdm.cidade.estado.idEstado);
291     cidade = respostaAdm.cidade.idCidade;
292     BuscarCidadesPorEstado();
293     var foto = document.getElementById("foto");
294     foto.src = respostaAdm.foto;
295 } else {
296     //se não encontrou administrador busca por pessoa some
297     tipoPessoa = 'pessoa';
298     $.getJSON('PessoaBuscarCpfCnpj', {cpfcnpjpessoa: idM,
299         console.log(respostaPessoa);
300         var id = respostaPessoa.idPessoa;
301         if (id != "0")
302         {
303             $('#idpessoa').val(respostaPessoa.idPessoa);
304             $('#nomepessoa').val(respostaPessoa.nome);
305             $('#login').val(respostaPessoa.login);
306             var datanasc = respostaPessoa.dataNascimento;
307             console.log(datanasc);
308             $('#datanascimento').val(datanasc);
309             $('#idestado').val(respostaPessoa.cidade.estado.idEstado);
310             cidade = respostaPessoa.cidade.idCidade;
311             BuscarCidadesPorEstado();
312             var foto = document.getElementById("foto");
313             foto.src = respostaPessoa.foto;
```

```
313     }
314     });
315   });
316   });
317   });
318 }
```

Continuação

```

319
320    function deletar(codigo){
321        var id = codigo;
322        console.log(codigo);
323        Swal.fire({
324            title: 'Você tem certeza?',
325            text: "Você deseja realmente inativar/ativar o administrador?",
326            icon: 'warning',
327            showCancelButton: true,
328            confirmButtonColor: '#3085d6',
329            cancelButtonColor: '#d33',
330            confirmButtonText: 'Sim',
331            cancelButtonText: 'Cancelar'
332        }).then((result) => {
333            if (result.isConfirmed) {
334                $.ajax({
335                    type: 'post',
336                    url: '${pageContext.request.contextPath}/AdministradorExcluir',
337                    data: {
338                        idAdministrador: id
339                    },
340                    success:

```

**Ativa e Inativa
administrador através
do servlet
AdministradorExcluir**

```

341            function(data){
342                if(data == 1){
343                    Swal.fire({
344                        position: 'top-end',
345                        icon: 'success',
346                        title: 'Sucesso',
347                        text: 'Administrador inativado com sucesso!',
348                        showConfirmButton: true,
349                        timer: 10000
350                    }).then(function(){
351                        window.location.href = "${pageContext.request.contextPath}/AdministradorListar";
352                    })
353                } else {
354                    Swal.fire({
355                        position: 'top-end',
356                        icon: 'error',
357                        title: 'Erro',
358                        text: 'Não foi possível inativar administrador!',
359                        showConfirmButton: true,
360                        timer: 10000
361                    }).then(function(){
362                        window.location.href = "${pageContext.request.contextPath}/AdministradorListar";
363                    })
364                }
365            },
366            error:
367            function(data){
368                window.location.href = "${pageContext.request.contextPath}/DespesaListar";
369            }
370        });
371    );
372});
373}
374

```

```

375    function validarCampos() {
376        console.log("entrei na validação de campos");
377        if (document.getElementById("nomepessoa").value == '') {
378            Swal.fire({
379                position: 'center',
380                icon: 'error',
381                title: 'Verifique o Nome do Administrador!',
382                showConfirmButton: true,
383                timer: 2000
384            });
385            $("#nome").focus();
386        } else if (document.getElementById("datanascimento").value == '') {
387            Swal.fire({
388                position: 'center',
389                icon: 'error',
390                title: 'Verifique a Data de nascimento!',
391                showConfirmButton: true,
392                timer: 2000
393            });
394            $("#datanascimento").focus();

```

**Faz a validação dos
dados no campos na
tela**

Continuação

```

396     } else if (document.getElementById("idcidade").value == 'nulo') {
397         Swal.fire({
398             position: 'center',
399             icon: 'error',
400             title: 'Verifique a cidade!',
401             showConfirmButton: true,
402             timer: 2000
403         });
404         $("#idcidade").focus();
405     } else {
406         gravarDados();
407     }
408 }

409 function gravarDados() {
410     console.log("Gravando dados ....");
411     var target = document.getElementById("foto").src;
412
413     $.ajax({
414         type: 'post',
415         url: 'AdministradorCadastrar',
416         data: {
417             idadministrador: $('#idadministrador').val(),
418             idpessoa: $('#idpessoa').val(),
419             cpfcnpjpessoa: $('#cpfcnpjpessoa').unmask().val(),
420             nomepessoa: $('#nomepessoa').val(),
421             datanascimento: $('#datanascimento').val(),
422             idcidade: $('#idcidade').val(),
423             login: $('#login').val(),
424             senha: $('#senha').val(),
425             permitelogin: $('#permiteLogin').val(),
426             situacao: $('#situacao').val(),
427             fotopessoa: target
428         },
429
430         success:
431             function (data) {
432                 console.log("resposta servlet->");
433                 console.log(data);
434                 if (data == 1) {
435                     Swal.fire({
436                         position: 'center',
437                         icon: 'success',
438                         title: 'Sucesso',
439                         text: 'Administrador gravado com sucesso!',
440                         showConfirmButton: true,
441                         timer: 10000
442                     }).then(function(){
443                         window.location.href = "${pageContext.request.contextPath}/AdministradorListar";
444                     })
445                 } else {
446                     Swal.fire({
447                         position: 'center',
448                         icon: 'error',
449                         title: 'Erro',
450                         text: 'Não foi possível gravar o administrador!',
451                         showConfirmButton: true,
452                         timer: 10000
453                     }).then(function(){
454                         window.location.href = "${pageContext.request.contextPath}/AdministradorListar";
455                     })
456                 }
457             },
458             error:
459                 function (data) {
460                     window.location.href = "${pageContext.request.contextPath}/AdministradorListar";
461                 }
462         });
463     }

```

**Grava dados da tela
através do servlet
AdministradorCadastrar**

Continuação

```

464     function BuscarCidadesPorEstado() {
465         $('#idcidade').empty();
466         idEst = $('#idestado').val();
467         console.log("entrou buscar estado");
468         if (idEst != 'null')
469         {
470             console.log("estado = " + idEst);
471             url = "CidadeBuscarPorEstado?idestado=" + idEst;
472             //console.log(url);
473             $.getJSON(url, function (result) {
474                 //alert(result);
475                 $.each(result, function (index, value) {
476                     $('#idcidade').append('<option id="cidade_' + value.idCidade + '" value=' + value.idCidade + '>' +
477                         value.nomeCidade + '</option>');
478                     if (cidade !== '') {
479                         $('#cidade_' + cidade).prop({selected: true});
480                     } else {
481                         $('#cidade_').prop({selected: true});
482                     }
483                 });
484             }).fail(function (obj, textStatus, error) {
485                 alert('Erro do servidor: ' + textStatus + ', ' + error);
486             });
487         }
488     }
489
490     function uploadFile() {
491         var target = document.getElementById("foto");
492         target.src = "";
493         var file = document.querySelector("input[type='file']").files[0];
494
495         if (file) {
496             var reader = new FileReader();
497             reader.readAsDataURL(file);
498             reader.onloadend = function () {
499                 target.src = reader.result;
500             };
501         } else {
502             target.src = "";
503         }
504     }
505 </script>
506 <%@include file="/footer.jsp"%>
```

Atualiza lista de
cidades

Faz o upload da foto

Fonte: O autor

**Não se esqueçam de implementar os
cadastros de Cliente e Fornecedor**

CAPÍTULO 10 – IMPLEMENTANDANDO A SEGURANÇA NO SISTEMA

Neste capítulo implementaremos a segurança de nosso sistema. A segurança será composta por um controle de acesso (login) e por um controle de usuário.

O login permitirá identificar o usuário que acessa o sistema, assim como o seu nível de acesso ou tipo de usuário que irá definir ao que ele terá acesso ao sistema.

10.1 LOGIN DE USUÁRIO

Vamos iniciar nosso controle de login pela camada model de nossa aplicação para isso vamos criar uma classe Usuário. A classe Usuário irá representar em nosso sistema não uma tabela no banco de dados, mas sim um view.

Sim! Vamos utilizar o conceito de view para nosso controle de acesso. Os usuários serão definidos a partir de nosso cadastro de herança Pessoa, Administrador, Fornecedor e Cliente, onde cada uma das classes filhas irá identificar o tipo de usuário que estará logando no sistema.

Vamos criar nossa view conforme na **Figura 228**.

Figura 228 – Criando a view Usuario no banco de dados

```

1  create or replace view usuario as
2      select p.idpessoa, p.nome, p.cpfcnpj, p.login, p.senha,
3             c.idcliente as id,'Cliente' as tipo
4      from pessoa p, cliente c
5     where c.idpessoa = p.idpessoa and c.situacao = 'A' and
6           c.permitelogin = 'S'
7      union
8      select p.idpessoa, p.nome, p.cpfcnpj, p.login, p.senha,
9             f.idfornecedor as id,'Fornecedor' as tipo
10     from pessoa p, fornecedor f
11    where f.idpessoa = p.idpessoa and f.situacao = 'A' and
12        f.permitelogin = 'S'
13      union
14      select p.idpessoa, p.nome, p.cpfcnpj, p.login, p.senha,
15             a.idadministrador as id,'Administrador' as tipo
16     from pessoa p, administrador a
17    where a.idpessoa = p.idpessoa and a.situacao = 'A' and
18        a.permitelogin = 'S'
```

Fonte: O autor

Por exemplo se um usuário qualquer tiver cadastro nas 3 classes (tabelas no banco de dados) ele terá na view de usuário 3 registros identificando as três possíveis formas de acesso ao sistema. Deste modo o usuário poderá se logar como administrador, cliente ou fornecedor tendo acesso a recursos diferentes do sistema.

Não se esqueça de guardar o código da view no arquivo banco.sql que está em nossa camada utils.

Agora vamos criar a classe Usuario na camada model de nosso sistema, para isso clique como o botão direito sobre o **pacote br.com.curso.model** e escolha a opção **Novo -> Classe Java**, e **crie uma classe denominada Usuario** (sem acento, não esqueça!). Digite os atributos conforme na **Figura 229** e **crie os construtores e gere os gets e sets**.

Figura 229 – Criando a classe model de Usuario

```

1 package br.com.curso.model;
2
3 public class Usuario {
4
5     private int idPessoa;
6     private String nome;
7     private String cpfcnpj;
8     private String login;
9     private String senha;
10    private String tipo;
11    private int id;
12
13    public Usuario() {
14        this.idPessoa = 0;
15        this.id = 0;
16        this.tipo = "";
17    }
18
19    public Usuario(int idPessoa, String nome, String cpfcnpj, String login, String tipo, int id) {
20        this.idPessoa = idPessoa;
21        this.nome = nome;
22        this.cpfcnpj = cpfcnpj;
23        this.login = login;
24        this.tipo = tipo;
25        this.id = id;
26    }
27
28    public int getIdPessoa() {
29        return idPessoa;
30    }
31
32    public void setIdPessoa(int idPessoa) {
33        this.idPessoa = idPessoa;
34    }
35
36    public String getNome() {
37        return nome;
38    }
39
40    public void setNome(String nome) {
41        this.nome = nome;
42    }
43
44    public String getCpfcnpj() {
45        return cpfcnpj;
46    }
47
48    public void setCpfcnpj(String cpfcnpj) {
49        this.cpfcnpj = cpfcnpj;
50    }
51

```

Continuação

```

52  public String getLogin() {
53      return login;
54  }
55
56  public void setLogin(String login) {
57      this.login = login;
58  }
59
60  public String getSenha() {
61      return senha;
62  }
63
64  public void setSenha(String senha) {
65      this.senha = senha;
66  }
67
68  public String getTipo() {
69      return tipo;
70  }
71
72  public void setTipo(String tipo) {
73      this.tipo = tipo;
74  }
75
76  public int getId() {
77      return id;
78  }
79
80  public void setId(int id) {
81      this.id = id;
82  }
83
84
85

```

Fonte: O autor

Agora vamos criar a classe UsuarioDAO na camada DAO de nosso sistema, para isso clique como o botão direito sobre o pacote **br.com.curso.dao** e escolha a opção **Novo > Classe Java**, e crie uma classe denominada **UsuarioDAO** (sem acento, não esqueça!). Por ser uma cada DAO de uma View em nosso banco de dados não existirá métodos como cadastrar, incluir, alterar ou excluir nessa classe, apenas métodos de consulta, assim **nossa classe UsuarioDAO não implementará nossa interface GenericDAO**. Implemente os códigos conforme na **Figura 230**.

Figura 230 – Criando a classe model de UsuarioDAO

```

1  package br.com.curso.dao;
2
3
4  import br.com.curso.model.Usuario;
5  import br.com.curso.utils.SingleConnection;
6  import java.sql.Connection;
7  import java.sql.PreparedStatement;
8  import java.sql.ResultSet;
9  import java.sql.SQLException;
10 import java.util.ArrayList;
11 import java.util.List;
12

```

Continuação

```

13  public class UsuarioDAO {
14
15      private Connection conexao;
16
17      public UsuarioDAO() throws Exception{
18          conexao = SingleConnection.getConnection();
19      }
20
21      public Usuario logar(String login, String senha, String tipo)
22      {
23          PreparedStatement stmt = null;
24          ResultSet rs = null;
25          Usuario oUsuario = null;
26          String sql = "select * from usuario u where u.login=? and u.senha=? and u.tipo=?";
27          try{
28              stmt=conexao.prepareStatement(sql);
29              stmt.setString(1, login);
30              stmt.setString(2, senha);
31              stmt.setString(3, tipo);
32              rs=stmt.executeQuery();
33
34              while (rs.next()){
35                  oUsuario = new Usuario(rs.getInt("idpessoa"),
36                                         rs.getString("nome"),
37                                         rs.getString("cpfcnpj"),
38                                         rs.getString("login"),
39                                         rs.getString("senha"),
39                                         rs.getString("tipo"),
40                                         rs.getInt("id"));
41              }
42          }
43          return oUsuario;
44      }
45      catch(SQLException ex){
46          System.out.println("Problemas ao carregar usuario!"
47                             + "Erro:"+ex.getMessage());
48          ex.printStackTrace();
49          return null;
50      }
51  }
52
53  public List<Usuario> listar(String loginUsuario){
54      List<Usuario> lstUsuario = new ArrayList<>();
55      PreparedStatement stmt = null;
56      ResultSet rs = null;
57      String sql = "select * from usuario u where u.login = ?;";
58      try {
59          stmt = conexao.prepareStatement(sql);
60          stmt.setString(1, loginUsuario);
61          rs = stmt.executeQuery();
62          while (rs.next()){
63              Usuario oUsuario = new Usuario(rs.getInt("idpessoa"),
64                                         rs.getString("nome"),

```

Continuação

```

63     Usuario oUsuario = new Usuario(rs.getInt("idpessoa"),
64                                         rs.getString("nome"),
65                                         rs.getString("cpfcnpj"),
66                                         rs.getString("login"),
67                                         rs.getString("senha"),
68                                         rs.getString("tipo"),
69                                         rs.getInt("id"));
70         lstUsuario.add(oUsuario);
71     }
72 }catch (SQLException ex){
73     System.out.println("Problemas ao listar Usuario! Erro:"
74             + ex.getMessage());
75 }
76 return lstUsuario;
77 }
78 }
```

Fonte: O autor

Agora vamos criar o pacote para implementarmos nossa **camada Controller**, então clique com o **botão direito sobre “Pacotes de código fonte” e crie um novo “Pacote Java” denominado “br.com.curso.controller.usuario”**. Neste pacote vamos implementar 3 Servlets.

O primeiro Servlet a ser implementado terá a função de receber a requisição através de AJAX quando o usuário digitar seu login na tela. Então efetuará uma busca na view de usuário por todos os perfis disponíveis para o usuário através de método implementado na camada DAO de usuário.

A lista de retorno gerada será mostrada na tela de login para que o usuário possa escolher com qual perfil deseja entrar no sistema e assim será possível configurar os menus de acesso para este usuário no perfil adequado para sua utilização.

Vamos criar a classe do tipo Servlet **UsuarioBuscarPorLogin** na camada Controller de Usuario de nosso sistema, para isso clique como o botão direito sobre o **pacote br.com.curso.controller.usuario** e escolha a opção **Novo -> Servlet**, e **crie uma classe denominada UsuarioBuscarPorLogin** (sem acento, não esqueça!). Implemente os códigos no método **processRequest** da classe **UsuarioBuscarPorLogin** conforme na **Figura 231**.

Figura 231 – Criando a Servlet UsuarioBuscarPorLogin

```

36     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37         throws ServletException, IOException {
38     response.setContentType("text/html;charset=iso-8859-1");
39     String loginUsuario = request.getParameter("loginusuario");
40     try{
41         UsuarioDAO oUsuarioDAO = new UsuarioDAO();
42         List<Usuario> lstUsuario = oUsuarioDAO.listar(loginUsuario);
43
44         Gson gson = new Gson();
45         String jsonUsuario = gson.toJson(lstUsuario);
46
47         response.setCharacterEncoding("iso-8859-1");
48         response.getWriter().write(jsonUsuario);
49
50     } catch (Exception ex) {
51         System.out.println("Problemas ao listar Usuário! Erro: "
52                         + ex.getMessage());
53         ex.printStackTrace();
54     }
55 }
```

Fonte: O autor

O segundo Servlet a ser implementado terá a função de receber a requisição através de AJAX quando o usuário clicar no botão de login na tela. Então através da camada DAO realizará a busca pelo usuário com sua senha e perfil para poder determinar se o usuário está apto a acessar o sistema.

Se estiver apto a acessar o sistema vai gerar **uma sessão de usuário no contexto do servidor** com dados que podem ser verificados a qualquer momento da execução do sistema.

Se não vai retornar com uma mensagem de erro a interface do sistema.

Vamos criar a classe do tipo Servlet **UsuarioLogar** na camada Controller de Usuario de nosso sistema, para isso clique como o botão direito sobre o pacote **br.com.curso.controller.usuario** e escolha a opção **Novo -> Servlet**, e **crie uma classe denominada UsuarioLogar** (sem acento, não esqueça!). Implemente os códigos no **método processRequest da classe UsuarioLogar** conforme na **Figura 232****Figura 231**.

Figura 232 – Criando a Servlet UsuarioLogar

```

35     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
36         throws ServletException, IOException {
37     response.setContentType("text/html;charset=iso-8859-1");
38
39     try{
40         String loginUsuario = request.getParameter("login");
41         String senhaUsuario = request.getParameter("senha");
42         String tipoUsuario = request.getParameter("tipo");
43         String usuarioLogado = "";
44
45         UsuarioDAO oUsuarioDAO = new UsuarioDAO();
46         Usuario oUsuario = oUsuarioDAO.logar(loginUsuario, senhaUsuario, tipoUsuario);
47
48         if (oUsuario != null) {
49             //cria a sessão no contexto do sistema
50             HttpSession sessao = request.getSession();
51             sessao.setAttribute("idusuario", oUsuario.getId());
52             sessao.setAttribute("nomeusuario", oUsuario.getNome());
53             sessao.setAttribute("tipousuario", oUsuario.getTipo());
54             usuarioLogado = "ok";
55         } else {
56             //usuario recusado
57             System.out.println("Usuário ou senha invalida, verificar dados");
58             usuarioLogado = "";
59         }
60
61         response.setCharacterEncoding("UTF-8");
62         response.getWriter().write(usuarioLogado);
63
64     } catch (Exception ex) {
65         System.out.println("Problemas ao logar Usuário! Erro: "+ ex.getMessage());
66         ex.printStackTrace();
67     }
}

```

Fonte: O autor

Observe as linhas de código 50 a 53 nestas linhas observamos a criação da sessão de usuário no contexto do servidor com 3 atributos que podem ser lidos a qualquer momento “idusuario, nomeusuario, tipousuario”.

Através da verificação dessas variáveis vamos gerenciar o acesso do usuário ao nosso sistema, mas a frente neste tutorial iremos implementar essa verificação em nossa classe de Filtro.

Finalmente vamos implementar nosso terceiro Servlet que terá a função de fazer o logoff de nosso sistema ou seja eliminar nossa sessão do contexto do servidor e redirecionar o usuário novamente para página de login.

Vamos criar a classe do tipo Servlet **UsuarioDeslogar** na camada Controller de Usuario de nosso sistema, para isso clique como o botão direito sobre o pacote **br.com.curso.controller.usuario** e escolha a opção **Novo -> Servlet**, e **crie uma classe denominada UsuarioDeslogar** (sem acento, não esqueça!). Implemente os

códigos no **método processRequest da classe UsuarioDeslogar** conforme na **Figura 233**.

Figura 233 – Criando a Servlet UsuarioDeslogar

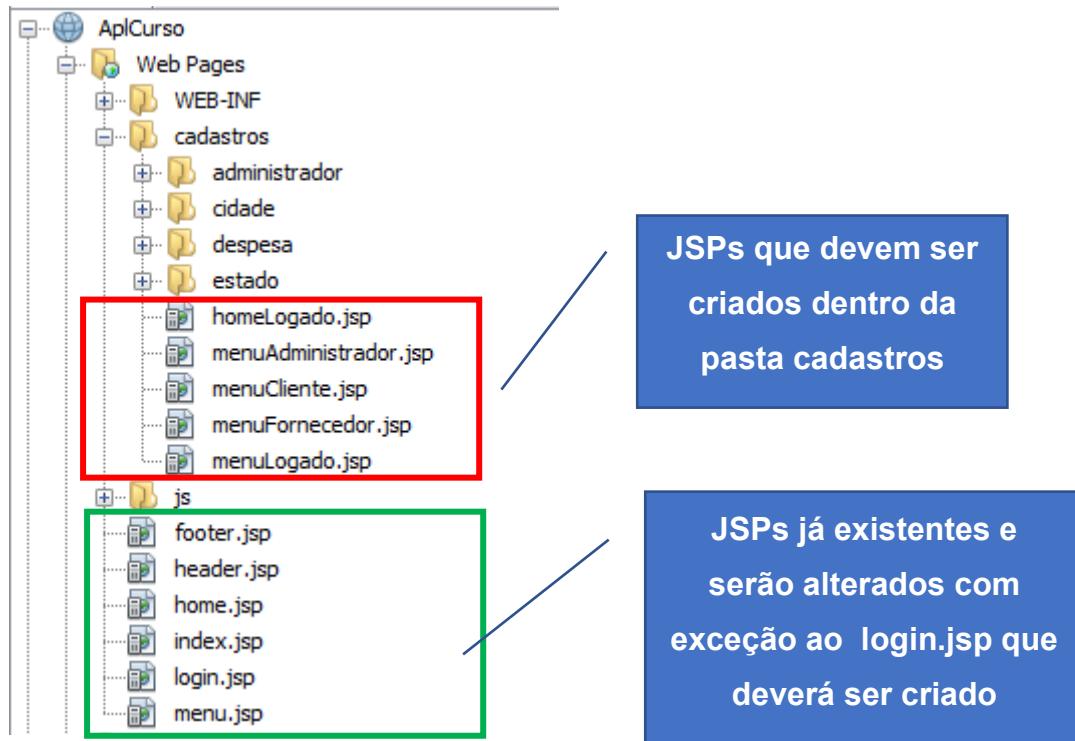
```

33     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
34         throws ServletException, IOException {
35             response.setContentType("text/html; charset=UTF-8");
36             try{
37                 HttpSession sessao = request.getSession(false);
38                 sessao.invalidate();
39                 response.sendRedirect("index.jsp");
40             } catch (Exception ex) {
41                 System.out.println("Problemas ao logar Usuário! Erro: " + ex.getMessage());
42                 ex.printStackTrace();
43             }
44         }
    
```

Fonte: O autor

Agora precisamos reorganizar nossa interface criando JSPs, na **Figura 234** apresenta-se a estrutura de como deve ficar em nossa interface.

Figura 234 – Interface Sistema (JSPs)



Fonte: O autor

Muito bom! Então agora clique com o **botão direito na pasta cadastros e escolha a opção Novo -> JSP** e crie cada um dos novos JSPs conforme demonstrado na **Figura 234** no quadro vermelho.

Crie também o login.jsp só que agora **clicando em Páginas Web (Web Pages)** com o **botão direito e escolha a opção Novo -> JSP** de modo que nosso novo JSP fique na raiz de nosso projeto.

Depois de criado os nossos novos JSPs vamos começar a codificação alterando os JSPs já existentes, então vamos passar por cada um deles.

O index.jsp você pode visualizar na **Figura 235** confira com o seu projeto mas este não sofrerá qualquer alteração.

Figura 235 – Revisando o Index.jsp

```
1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <jsp:include page="home.jsp"/>
```

Fonte: O autor

Os arquivos footer.jsp e header.jsp também não sofrerão alterações então não exibiremos os mesmos agora.

Vamos revisar o menu.jsp, deixe o código deste arquivo como demonstrado na **Figura 236**. Agora nosso menu.jsp ficará apenas com a opção de login.

Figura 236 – Revisando o menu.jsp

```
1 <h1>Módulo Cadastros</h1>
2 <hr>
3 <center>
4   <h2>Faça Login no Sistema</h2>
5   <a href="${pageContext.request.contextPath}/login.jsp">Login</a>
6 </center>
7 <hr>
8
```

Fonte: O autor

O home.jsp é próximo a ser revisado, então revise e codifique o seu arquivo de acordo com o demonstrado na **Figura 237**.

Figura 237 – Revisando o home.jsp

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4 <jsp:include page="header.jsp"/>
5 <jsp:include page="menu.jsp"/>
6
7     <h1>Sistema Exemplo - CRUD - Deslogado</h1>
8
9 <jsp:include page="footer.jsp"/>
```

Fonte: O autor

Agora vamos codificar os novos JSPs criados, então começaremos pela nossa tela de login.jsp. Faça a codificação conforme o demonstrado na **Figura 238**.

Figura 238 – Revisando o login.jsp

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <jsp:include page="header.jsp"/>
3
4     <h1>Sistema Exemplo - CRUD - Deslogado</h1>
5         <h2>Login Sistema</h2>
6             <div>
7                 <label for="login">Login</label>
8                 <input type="text" id="login" name="login" required="" placeholder="Coloque seu login"
9                     onblur="BuscarUsuariosPorNome()">
10            </div>
11            <div>
12                <label for="senha">Senha</label>
13                <input type="password" required="" name="senha" id="senha" placeholder="Coloque sua senha">
14            </div>
15            <div>
16                <label for="tipo">Tipo Usuário</label>
17                <select name="tipo" id="tipo" tabindex="3">
18                    <option value="">Selecione</option>
19                </select>
20            </div>
21            <div><button id="submit">Logar</button></div>
22            <br>
23            <div id="erro"></div>
24
25             <script>
26                 $(document).ready(function () {
27                     console.log("entrei função");
28                     $("#submit").on("click", function () {
29                         console.log("entrei click submit");
30                         if ($("#login").val() === "") {
31                             $("#login").focus();
32                             $("#erro").html("<div>Por favor, preencher o campo usuário.</div>").show();
33                             tempo();
34                             return false;
35                         }
36                         if ($("#senha").val() === "") {
37                             $("#senha").focus();
38                             $("#erro").html("<div>Por favor, preencher o campo senha.</div>").show();
39                             tempo();
40                             return false;
41                         }
42                         $("#submit").prop("disabled", true);
43                         $("#submit").html('<i class="fa fa-spinner" aria-hidden="true"></i> Aguarde...');
```

Continuação

```

45   $.ajax({
46     type: 'post',
47     url: 'UsuarioLogar',
48     data: {
49       acao: "login",
50       login: $('#login').val(),
51       senha: $('#senha').val(),
52       tipo: $('#tipo').val()
53     },
54     success:
55     function (data) {
56       if (data == 'ok') {
57         window.location.href = "${pageContext.request.contextPath}/cadastros/homeLogado.jsp";
58       } else {
59         $('#submit').removeAttr('disabled');
60         $('#submit').html('Entrar');
61         $('#wrapper_error').html("<div class='alert alert-danger'>Usuário ou senha incorreto.</div>").show();
62         tempo();
63       }
64     },
65     error:
66     function (data) {
67       RefreshTable();
68     }
69   });
70 });

71   function tempo() {
72     setTimeout(function () {
73       $('#wrapper_error').hide();
74     }, 3000); // 3 segundos
75   });
76 });

77 function BuscarUsuariosPorNome()
78 {
79   usuario = '';
80   console.log("entrei na function");
81   $('#tipo').empty(); //..limpa select de tipo de usuario.
82   loginUsuario = $('#login').val();
83   console.log(loginUsuario);
84   if (loginUsuario != 'null')
85   {
86     console.log("vai rodar o ajax");
87     //console.log(idEst);
88     url = "UsuarioBuscarPorLogin?loginusuario="+loginUsuario;
89     //console.log(url);

90   $.getJSON(url, function (result) {
91     //alert(result);
92     $.each(result, function (index, value) {
93       $('#tipo').append(
94         '<option id="usuario_' + value.idUsuario
95         + 'value="' + value.tipo + '"'
96         + value.tipo + '</option>'
97       );
98       if(usuario != ''){
99         $('#usuario_'+usuario).prop({selected: true});
100      }else{
101        $('#usuario_').prop({selected: true});
102      }
103    });
104    console.log("montou o select");
105  }
106  .fail(function (obj, textStatus, error) {
107    alert('Erro do servidor: ' + textStatus + ', ' + error);
108  });
109 });
110 }

111 
```

Fonte: O autor

Agora vamos codificar os outros JSPs que estão na raiz da pasta Cadastros. Observe a **Figura 239** e **faça a codificação do menuCliente.jsp** conforme demonstrado.

Figura 239 – Revisando o menuCliente.jsp

```

1 <hr>
2   <center>
3     <h2>Menu Principal</h2>
4     <a href="${pageContext.request.contextPath}/EstadoListar">Estado</a>
5     <a href="${pageContext.request.contextPath}/CidadeListar">Cidade</a>
6   </center>
7   <hr>

```

Fonte: O autor

Seguindo o que foi demonstrado na **Figura 240** **faça a codificação do menuFornecedor.jsp**.

Figura 240 – Revisando o menuFornecedor.jsp

```

1 <hr>
2   <center>
3     <h2>Menu Principal</h2>
4     <a href="${pageContext.request.contextPath}/EstadoListar">Estado</a>
5   </center>
6   <hr>
7

```

Fonte: O autor

Seguindo o que foi demonstrado na **Figura 241** **faça a codificação do menuAdministrador.jsp**.

Figura 241 – Revisando o menuAdministrador.jsp

```

1 <hr>
2   <center>
3     <a href="${pageContext.request.contextPath}/EstadoListar">Estado</a>
4     <a href="${pageContext.request.contextPath}/CidadeListar">Cidade</a>
5     <a href="${pageContext.request.contextPath}/DespesaListar">Despesa</a>
6     <a href="${pageContext.request.contextPath}/AdministradorListar">Administrador</a>
7   </center>
8   <hr>

```

Fonte: O autor

Agora iremos codificar o **menuLogado.jsp** este jsp é responsável por fazer o chaveamento do tipo de menu que será carregado para a tela de acordo com o tipo do usuário que estiver logado no sistema no momento.

Assim, siga o proposto na **Figura 242** e codifique o seu menuLogado.jsp.

Figura 242 – Revisando o menuLogado.jsp

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <h1>Módulo Cadastros</h1>
3 <h2>Menu Principal - Logado: ${sessionScope.nomeusuario} - ${sessionScope.tipousuario} -
4   <a href="${pageContext.request.contextPath}/UsuarioDeslogar">Sair do Sistema</a></h2>
5 <c:if test="${sessionScope.tipousuario == 'Administrador'}">
6   <jsp:include page="menuAdministrador.jsp"/>
7 </c:if>
8 <c:if test="${sessionScope.tipousuario == 'Cliente'}">
9   <jsp:include page="menuCliente.jsp"/>
10 </c:if>
11 <c:if test="${sessionScope.tipousuario == 'Fornecedor'}">
12   <jsp:include page="menuFornecedor.jsp"/>
13 </c:if>
```

Fonte: O autor

Para finalizar nossos novos JSPs vamos codificar o homeLogado.jsp que irá ser apresentada quando o usuário estiver logado ao sistema. Faça a codificação conforme demonstrado na **Figura 243**.

Figura 243 – Revisando o homeLogado.jsp

```

1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
3 <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4 <jsp:include page="/header.jsp"/>
5 <jsp:include page="menuLogado.jsp"/>
6
7 <h3>Sistema Exemplo - CRUD - Logado</h3>
8
9 <jsp:include page="/footer.jsp"/>
10
```

Fonte: O autor

Você também deve alterar a linha como demonstrado abaixo em todos os seus outros JSPs em todas as subpastas (administrador, estado, cidade, fornecedor, cliente etc).

Observe a **Figura 244** e faça a troca do código que está grifado em vermelho pelo o que está grifado em verde.

Figura 244 – Revisando a chamada de menu em nossas páginas

```

1   <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2   <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
3   <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4   <jsp:include page="/header.jsp"/> Apagar
5   <jsp:include page="/menu.jsp"/>
6
7   <jsp:include page="/cadastros/menuLogado.jsp"/> Inserir
8

```

O diagrama ilustra a modificação de um código JSP. A estrutura original é a seguinte:

```

1   <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2   <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
3   <%@page contentType="text/html" pageEncoding="iso-8859-1"%>
4   <jsp:include page="/header.jsp"/>
5   <jsp:include page="/menu.jsp"/>
6
7   <jsp:include page="/cadastros/menuLogado.jsp"/>
8

```

A seta aponta para o bloco de código entre a linha 4 e a linha 5, que contém o comando `<jsp:include page="/menu.jsp"/>`. Este bloco é circundado por uma caixa vermelha e rotulado com "Apagar". Abaixo desse bloco, uma seta aponta para o comando `<jsp:include page="/cadastros/menuLogado.jsp"/>`, que está circundado por uma caixa verde e rotulado com "Inserir".

Fonte: O autor

Depois de todas essas alterações realizadas estamos prontos para efetivamente travarmos o acesso do nosso sistema. Para isso vamos alterar nossa classe de filtro, então abra o filtro de seu sistema e altere a codificação como demonstrado na **Figura 245**.

Figura 245 – Revisando a chamada de menu

```

17  @WebFilter(urlPatterns={"/"})
18  public class FilterAutenticacao implements Filter {
19
20      private static Connection conexao;
21
22      @Override
23      public void init(FilterConfig filterConfig) throws ServletException {
24          conexao = SingleConnection.getConnection();
25      }
26
27      @Override
28      public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
29          throws IOException, ServletException {
30          try{
31              HttpServletRequest req = (HttpServletRequest) request;
32              //Pega a sessão vigente no contexto do servidor
33              HttpSession sessao = req.getSession(false);
34              String urlParaAutenticar = req.getServletPath(); //Url que está sendo acessada
35
36              if ((sessao != null && sessao.getAttributeNames().hasMoreElements()) ||
37                  (urlParaAutenticar.equalsIgnoreCase("/index.jsp") ||
38                  urlParaAutenticar.equalsIgnoreCase("/home.jsp") ||
39                  urlParaAutenticar.equalsIgnoreCase("/login.jsp") ||
40                  urlParaAutenticar.equalsIgnoreCase("/UsuarioBuscarPorLogin") ||
41                  urlParaAutenticar.equalsIgnoreCase("/UsuarioLogar"))){

```

Continuação

```

42     //passou pela validação de segurança encaminha para a execução
43     chain.doFilter(request, response);
44 } else {
45     //se a sessão for nula volta para tela sem logar
46     request.getRequestDispatcher("/index.jsp").forward(request, response);
47     return; //Para a execução e redireciona para o index.jsp
48 }
49
50 } catch (Exception e) {
51     System.out.println("Erro: "+e.getMessage());
52     e.printStackTrace();
53 }
54
55 @Override
56 public void destroy() {
57     try {
58         conexao.close();
59     } catch (SQLException ex) {
60         System.out.println("Erro : " +ex.getMessage());
61         ex.printStackTrace();
62     }
63 }
64 }
```

Fonte: O autor

Vamos entender o que foi alterado em nosso método doFilter. Na linha 31 capturamos a requisição que foi enviada a partir da interface, então na linha 33 recuperamos a sessão atual, como o comando está com parâmetro FALSE ele não criará uma sessão se não encontrar, retornando NULL.

Na linha 34 capturamos a partir do request qual o recurso que deseja ser acessado ele será útil nas verificações de acesso.

Na linha 36 a 41 temos a condição do IF que permitira que a requisição continue para o backend do sistema. Nesta condição estão sendo validados:

- sessão != null → se tiver sessão criada permite acesso.
- sessao.getAttributeNames().hasMoreElements() → verifica se existe atributos criados
- Ou se for acesso a algum dos recursos que não exigem autenticação.

Passando pelas verificações permite o acesso a controller, senão reencaminha ao index.jsp novamente.

10.2 CONTROLE DE USUÁRIO (NÍVEL DE ACESSO)

O controle de usuário tem como responsabilidade gerenciar o acesso do usuário aos módulos do sistema de acordo com o seu perfil (tipo). Para implementarmos esse controle no sistema vamos criar um método na classe de Usuario na camada model.

Como pode-se verificar na **Figura 246** vamos criar o método chamado verificaUsuario que receberá o recurso que está sendo acessado e a sessão atual do sistema. Esse método deverá ser acessado a partir da classe de filtro de nosso projeto.

Figura 246 – Método verificaUsuario – Classe Usuario (Model)

```

82
83     public void setId(int id) {
84         this.id = id;
85     }
86
87     public static boolean verificaUsuario(String recurso, HttpSession sessao){
88         boolean status=false;
89
90         try{
91             //se for um acesso liberado permitir passagem
92             if ( recurso.equalsIgnoreCase("/index.jsp") ||
93                 recurso.equalsIgnoreCase("/home.jsp") ||
94                 recurso.equalsIgnoreCase("/login.jsp") ||
95                 recurso.equalsIgnoreCase("/UsuarioBuscarPorLogin") ||
96                 recurso.equalsIgnoreCase("/UsuarioLogar") ||
97                 recurso.equalsIgnoreCase("/js/jquery-3.3.1.min.js") ||
98                 recurso.equalsIgnoreCase("/js/jquery.mask.min.js") ||
99                 recurso.equalsIgnoreCase("/js/jquery.maskMoney.min.js") ||
100                recurso.equalsIgnoreCase("/js/app.js")) {
101                 status = true;
102             }
103         }
104
105         if(sessao != null && sessao.getAttributeNames().hasMoreElements()){
106             //pega dados do usuário
107             int idUsuario = Integer.parseInt(sessao.getAttribute("idusuario").toString());
108             String tipoUsuario = sessao.getAttribute("tipousuario").toString();
109
110             //verifica permissões
111             //se for administrador libera todos os recursos
112             if (tipoUsuario.equalsIgnoreCase("administrador")){
113                 status=true;
114             } else {
115                 if (tipoUsuario.equalsIgnoreCase("Cliente")){
116                     if (tipoUsuario.equalsIgnoreCase("/CidadeCarregar") ||
117                         recurso.equalsIgnoreCase("/CidadeCarregar") ||
118                         recurso.equalsIgnoreCase("/CidadeAlterar") ||
119                         recurso.equalsIgnoreCase("/CidadeListar") ||
120                         recurso.equalsIgnoreCase("/CidadeNovo") ||
121                         recurso.equalsIgnoreCase("/EstadoCadastrar") ||
122                         recurso.equalsIgnoreCase("/EstadoCarregar") ||
123                         recurso.equalsIgnoreCase("/EstadoAlterar") ||
124                         recurso.equalsIgnoreCase("/EstadoListar") ||
125                         recurso.equalsIgnoreCase("/EstadoNovo") ||
126                         recurso.equalsIgnoreCase("/cadastros/homeLogado.jsp")){
127                             status=true;//permite acesso ao usuario tipo cliente
128                         }
129                 }
130             }
131         }
132     }
133 }
```

Continuação

```

130
131         if (tipoUsuario.equalsIgnoreCase("Fornecedor"))
132     {
133         if (recurso.equalsIgnoreCase("/EstadoCadastrar") ||
134             recurso.equalsIgnoreCase("/EstadoCarregar") ||
135             recurso.equalsIgnoreCase("/EstadoAlterar") ||
136             recurso.equalsIgnoreCase("/EstadoListar") ||
137             recurso.equalsIgnoreCase("/EstadoNovo") ||
138             recurso.equalsIgnoreCase("/cadastros/homeLogado.jsp")){
139                 //permite acesso ao usuario tipo cliente
140                 status=true;
141             }
142         }
143     }
144 } catch (Exception ex){
145     System.out.println("Erro: " + ex.getMessage());
146     ex.printStackTrace();
147 }
148 return status;
149
150

```

Fonte: O autor

Agora devemos alterar o filtro novamente para contemplar esse novo controle implementado. Pode-se verificar na **Figura 247** as alterações realizadas no método doFilter da classe Filter do projeto.

Figura 247 – Alterações método doFilter na classe de filtro do projeto.

```

28
@Override
29 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
30 throws IOException, ServletException {
31     try{
32         HttpServletRequest req = (HttpServletRequest) request;
33         //Pega a sessão vigente no contexto do servidor
34         HttpSession sessao = req.getSession(false);
35         String urlParaAutenticar = req.getServletPath(); //Url que está sendo acessada
36
37         if ((sessao != null && sessao.getAttributeNames().hasMoreElements()) ||
38             (urlParaAutenticar.equalsIgnoreCase("/index.jsp") ||
39             urlParaAutenticar.equalsIgnoreCase("/home.jsp") ||
40             urlParaAutenticar.equalsIgnoreCase("/login.jsp") ||
41             urlParaAutenticar.equalsIgnoreCase("/UsuarioBuscarPorLogin") ||
42             urlParaAutenticar.equalsIgnoreCase("/UsuarioLogar") ||
43             urlParaAutenticar.equalsIgnoreCase("/js/jquery-3.3.1.min.js") ||
44             urlParaAutenticar.equalsIgnoreCase("/js/jquery.mask.min.js") ||
45             urlParaAutenticar.equalsIgnoreCase("/js/jquery.maskMoney.min.js") ||
46             urlParaAutenticar.equalsIgnoreCase("/js/app.js")) {
47
48             //valida controle de usuário
49             if (Usuario.verificaUsuario(urlParaAutenticar, sessao)){
50                 //passou pela validação de segurança encaminha para a execução
51                 chain.doFilter(request, response);
52             } else{
53                 //se a sessão for nula volta para tela sem logar
54                 request.getRequestDispatcher("/cadastros/homeLogado.jsp").forward(request, response);
55                 return; //Para a execução e redireciona para o index.jsp
56             }
57
58         } else {
59             //se a sessão for nula volta para tela sem logar
60             request.getRequestDispatcher("/index.jsp").forward(request, response);
61             return; //Para a execução e redireciona para o index.jsp
62         }
63     } catch(Exception e){
64         System.out.println("Erro: "+e.getMessage());
65         e.printStackTrace();
66     }
67 }

```

Fonte: O autor