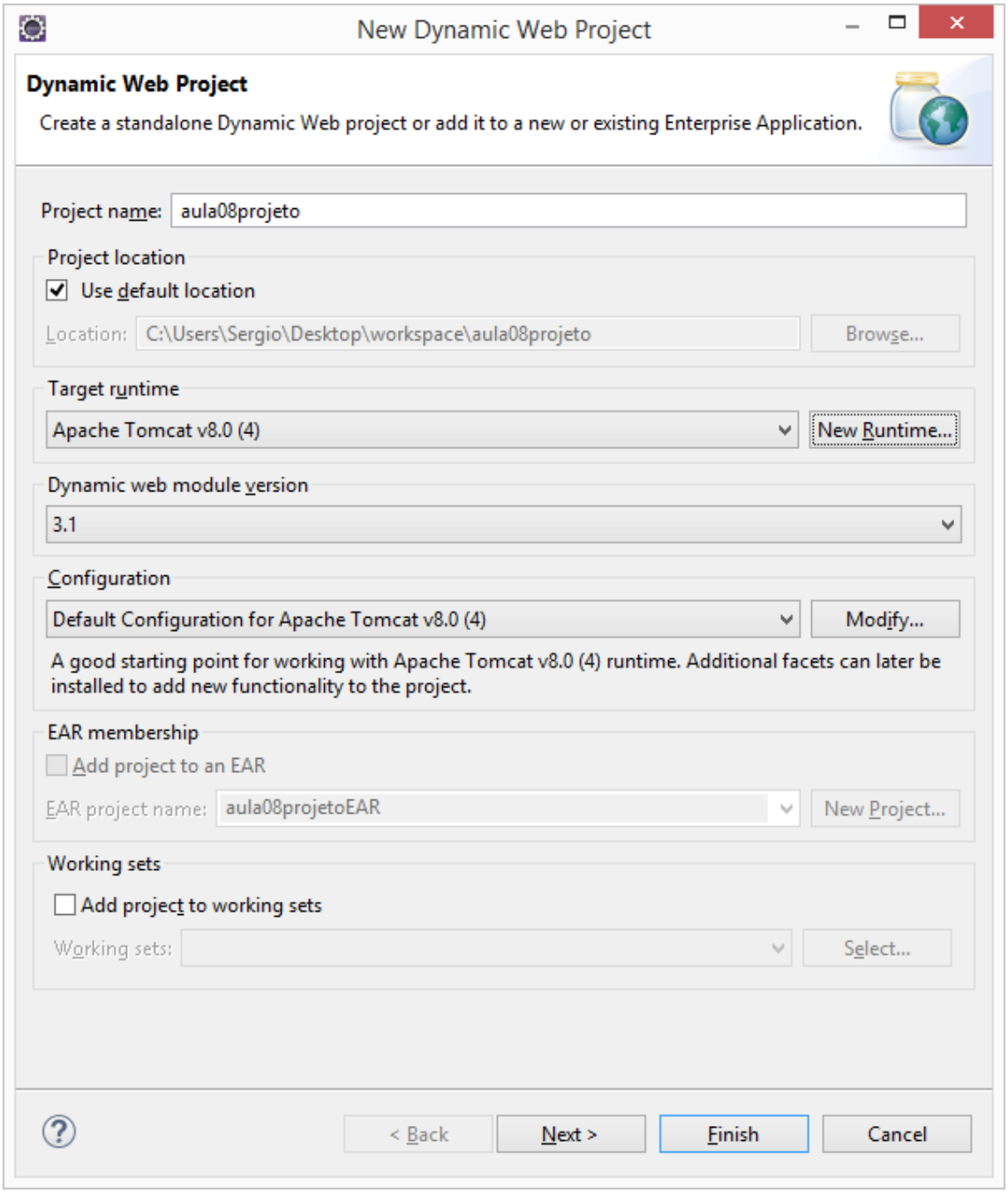
Criando o projeto web

- **File > New > Dynamic Web Project**



Criando a entidade Cliente:
entities/Cliente.java

```
package br.com.brq.entities;

import br.com.brq.entities.types.EstadoCivil;
import br.com.brq.entities.types.Sexo;

public class Cliente { //JavaBean

	private Integer idCliente;
	private String nome;
	private String email;
	private Sexo sexo;
	private EstadoCivil estadoCivil;

	public Cliente() {
		// TODO Auto-generated constructor stub
	}

	public Cliente(Integer idCliente, String nome, String email,
			Sexo sexo, EstadoCivil estadoCivil) {
		this.idCliente = idCliente;
		this.nome = nome;
		this.email = email;
		this.sexo = sexo;
		this.estadoCivil = estadoCivil;
	}

	public Integer getIdCliente() {
		return idCliente;
	}

	public void setIdCliente(Integer idCliente) {
		this.idCliente = idCliente;
	}

	public String getNome() {
		return nome;
	}

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

```
	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public Sexo getSexo() {
		return sexo;
	}

	public void setSexo(Sexo sexo) {
		this.sexo = sexo;
	}

	public EstadoCivil getEstadoCivil() {
		return estadoCivil;
	}

	public void setEstadoCivil(EstadoCivil estadoCivil) {
		this.estadoCivil = estadoCivil;
	}

	@Override
	public String toString() {
		return "Cliente [idCliente=" + idCliente + ", nome=" + nome + ", email="
				+ email + ", sexo=" + sexo
				+ ", estadoCivil=" + estadoCivil + "]";
	}
}
```

Criando a base de dados:
\script.sql

```
drop database if exists aula08;
create database aula08;
use aula08;

create table cliente(
	idcliente		integer			auto_increment,
	nome			varchar(50)		not null,
	email			varchar(50)		not null unique,
	sexo			enum('Masculino', 'Feminino') not null,
	estadocivil enum('Solteiro', 'Casado', 'Divorciado', 'Viuvo') not null,
```

```
        primary key(idcliente));

desc cliente;
```

Classe DAO (Data Access Object)

Conexão com o banco de dados

```
package br.com.brq.persistence;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class DAO {

        private static final String DRIVER = "com.mysql.jdbc.Driver";
        private static final String URL = "jdbc:mysql://localhost:3306/aula08";
        private static final String USER = "root";
        private static final String PASSWORD = "brqbrq";

        protected Connection con;
        protected PreparedStatement stmt;
        protected CallableStatement call;
        protected ResultSet rs;

        protected void openConnection() throws Exception{
                Class.forName(DRIVER);
                con = DriverManager.getConnection(URL, USER, PASSWORD);
        }

        protected void closeConnection() throws Exception{
                if(con != null){
                        con.close();
                }
        }
}
```

Criando a Classe de persistência para Cliente:

\DAOCliente.java

```
package br.com.brq.persistence;

import java.util.ArrayList;
import java.util.List;

import br.com.brq.entities.Cliente;
import br.com.brq.entities.types.EstadoCivil;
import br.com.brq.entities.types.Sexo;

public class DAOCliente extends DAO{

	//método para inserir um cliente na tabela..
	public void insert(Cliente c) throws Exception{

		String query = "insert into cliente(nome, email, sexo, estadocivil) "
					+ "values(?, ?, ?, ?)";

		openConnection();

		stmt = con.prepareStatement(query);
		stmt.setString(1, c.getNome());
		stmt.setString(2, c.getEmail());
		stmt.setString(3, c.getSexo().name());
		stmt.setString(4, c.getEstadoCivil().name());
		stmt.execute();
		stmt.close();

		closeConnection();
	}

	//metodo para atualizar um cliente na tabela..
	public void update(Cliente c) throws Exception{

		String query = "update cliente set nome=?, email=?, sexo=?, estadocivil=? "
					+ "where idcliente = ?";

		openConnection();

		stmt = con.prepareStatement(query);
		stmt.setString(1, c.getNome());
		stmt.setString(2, c.getEmail());
		stmt.setString(3, c.getSexo().name());
		stmt.setString(4, c.getEstadoCivil().name());
		stmt.setInt(5, c.getIdCliente());
		stmt.execute();
		stmt.close();
```

```
        closeConnection();
}

//método para excluir 1 cliente na tabela..
public void delete(Integer idCliente) throws Exception{

        String query = "delete from cliente where idcliente = ?";

        openConnection();

        stmt = con.prepareStatement(query);
        stmt.setInt(1, idCliente);
        stmt.execute();
        stmt.close();

        closeConnection();
}

//método para obter 1 cliente pelo id..
public Cliente findById(Integer idCliente) throws Exception{

        String query = "select * from cliente where idcliente = ?";

        openConnection();

        stmt = con.prepareStatement(query);
        stmt.setInt(1, idCliente);
        rs = stmt.executeQuery();

        Cliente c = null; //sem espaço de memória..

        if(rs.next()){ //se algum registro foi encontrado..
                c = new Cliente(); //espaço de memória..

                c.setIdCliente(rs.getInt("idcliente"));
                c.setNome(rs.getString("nome"));
                c.setEmail(rs.getString("email"));
                c.setSexo(Sexo.valueOf(rs.getString("sexo")));
                c.setEstadoCivil(EstadoCivil.valueOf(rs.getString("estadocivil")));
        }

        stmt.close();
        closeConnection();

        return c; //retornar o cliente...
}
//metodo para listar todos os clientes da tabela..
```

```
public List<Cliente> findAll() throws Exception{

        String query = "select * from cliente";

        openConnection();

        stmt = con.prepareStatement(query);
        rs = stmt.executeQuery();

        List<Cliente> lista = new ArrayList<Cliente>(); //lista vazia..

        while(rs.next()){ //enquanto houver registros..
                Cliente c = new Cliente(); //criando objeto..
                c.setIdCliente(rs.getInt("idcliente"));
                c.setNome(rs.getString("nome"));
                c.setEmail(rs.getString("email"));
                c.setSexo(Sexo.valueOf(rs.getString("sexo")));
                c.setEstadoCivil(EstadoCivil.valueOf(rs.getString("estadocivil")));

                lista.add(c); //adicionar dentro da lista..
        }

        stmt.close();
        closeConnection();

        return lista; //retornando a lista..
}

//método para listar os clientes pelo nome..
public List<Cliente> findAllByNome(String nome) throws Exception{

        String query = "select * from cliente where nome like ?";

        openConnection();

        stmt = con.prepareStatement(query);
        stmt.setString(1, "%" + nome + "%"); //contendo..
        rs = stmt.executeQuery(); //executa e le os dados da consulta..

        List<Cliente> lista = new ArrayList<Cliente>(); //lista vazia..

        while(rs.next()){ //enquanto houver registros..
                Cliente c = new Cliente(); //criando objeto..
                c.setIdCliente(rs.getInt("idcliente"));
                c.setNome(rs.getString("nome"));
                c.setEmail(rs.getString("email"));
                c.setSexo(Sexo.valueOf(rs.getString("sexo")));
                c.setEstadoCivil(EstadoCivil.valueOf(rs.getString("estadocivil")));
```

```
                lista.add(c); //adicionar dentro da lista..
            }

            stmt.close();
            closeConnection();

            return lista; //retornando a lista..
        }
}
```

JSP – Java Server Pages

JSP é o acrônimo para Java Server Pages, uma linguagem criada pela SUN gratuita, JSP é uma linguagem de script com especificação aberta que tem como objetivo primário a geração de conteúdo dinâmico para páginas da Internet. Podemos ao invés de utilizar HTML para desenvolver páginas Web estáticas e sem funcionalidade, utilizar o JSP para criar dinamismo.

É possível escrever HTML com códigos JSP embutidos. Como o HTML é uma linguagem estática, o JSP será o responsável por criar dinamismo. Por ser gratuita e possuir especificação aberta possui diversos servidores que suportam a linguagem, entre eles temos: Tomcat, GlassFish, JBoos, entre outros. O JSP necessita de servidor para funcionar por ser uma linguagem Server-side script, o usuário não consegue ver a codificação JSP, pois esta é convertida diretamente pelo servidor, sendo apresentado ao usuário apenas codificação HTML.

Uma pagina JSP possui extensão .jsp e consiste em uma página com codificação HTML e com codificação Java, inserida entre as tag´s , denominada scriptlets e funcionando da seguinte forma: o servidor recebe uma requisição para uma página JSP, interpreta esta página gerando a codificação HTML e retorna ao cliente o resultado de sua solicitação. A página JSP que foi interpretada pelo servidor não precisa ser compilada como aconteceria com um servlet java por exemplo, esta tarefa é realizada em tempo real pelo servidor. É necessário apenas desenvolver as páginas JSP e disponibilizá-las no Servlet Container (Tomcat, por exemplo).

Por padrão, todo projeto Java Web deverá ter uma página JSP inicial, chamada de **index.jsp**

Criando a index.jsp

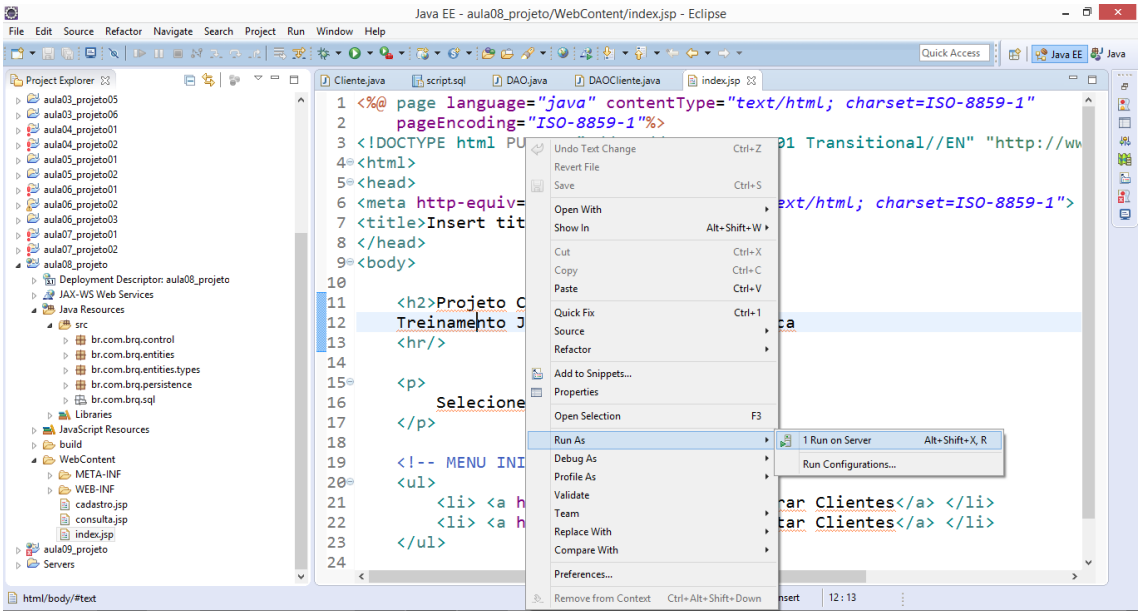
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
      <h2>Projeto Controle de Clientes</h2>
      Treinamento Java BRQ/SP COTI Informatica
      <hr/>

      <p>
            Selecione a operação desejada:
      </p>

      <!-- MENU INICIAL -->
      <ul>
            <li> <a href="cadastro.jsp">Cadastrar Clientes</a> </li>
            <li> <a href="consulta.jsp">Consultar Clientes</a> </li>
      </ul>
</body>
</html>
```

Testando:

Run As > Run on Server



Resultado:

Criando a página de cadastro de Clientes:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

	<h3>Formulário de Cadastro de Clientes</h3>
	Para incluir um novo cliente, informe os dados abaixo:
	<hr/>

	<p>
		<a href="index.jsp">Voltar</a> para a página inicial.
	</p>

	<!-- FORMULARIO -->
	<form name="formulario" method="post"
		action="ControleCliente?acao=cadastrar">

		<label>Nome do Cliente:</label> <br/>
		<input type="text" name="nome"/>
		<br/><br/>

		<label>Email:</label> <br/>
		<input type="text" name="email"/>
		<br/><br/>

		<label>Sexo:</label> <br/>
		<select name="sexo">
			<option value="">- Escolha uma Opção -</option>
```

```
                <option value="Masculino">Masculino</option>
                <option value="Feminino">Feminino</option>
            </select>
            <br/><br/>

            <label>Estado Civil:</label> <br/>
            <input type="radio" name="estadocivil" value="Solteiro"/>
                Solteiro <br/>
            <input type="radio" name="estadocivil" value="Casado"/>
                Casado <br/>
            <input type="radio" name="estadocivil" value="Viuvo"/>
                Viuvo <br/>
            <input type="radio" name="estadocivil" value="Divorciado"/>
                Divorciado <br/>
            <br/>

            <!-- Botão para enviar os dados do formulario para o servidor..
-->
            <input type="submit" value="Cadastrar Cliente"/>

            <!-- Botão para apagar o conteudo dos campos -->
            <input type="reset" value="Limpar"/>
        </form>

</body>
</html>
```

Executando:

Servlets

Servlets são classes Java, desenvolvidas de acordo com uma estrutura bem definida que quando instaladas e configuradas em um Servidor que implemente um Servlet Container, podem tratar requisições recebidas de clientes Web, como por exemplo os Browsers (Internet Explorer® e Mozilla Firefox®).

Ao receber uma requisição, um Servlet pode capturar os parâmetros desta requisição, efetuar qualquer processamento inerente a uma classe Java, e devolver uma página HTML.

A utilização de Servlets e de páginas JSP oferecem diversas vantagens em relação ao uso de outras tecnologias (como PHP, ASP e CGI). As principais vantagens são herdadas da própria linguagem Java, como:

- **Portabilidade**: a aplicação desenvolvida pode ser implantada em diversas plataformas, como por exemplo, Windows, Unix, GNU/Linux e Macintosh, sem que seja necessário modificar ou mesmo reconstruir a aplicação.
- **Facilidade de programação:** a programação é orientada a objetos, simplificando o desenvolvimento de sistemas complexos. Além disso, a linguagem oferece algumas facilidades, como por exemplo, o gerenciamento automático de memória (estruturas alocadas são automaticamente liberadas, sem que o desenvolvedor precise se preocupar em gerenciar esse processo).
- **Flexibilidade:** o Java já se encontra bastante difundido, contando com uma enorme comunidade de desenvolvedores, ampla documentação e diversas bibliotecas e códigos prontos, dos quais o desenvolvedor pode usufruir.

Além dessas vantagens, a arquitetura de Servlets e páginas JSP possibilitam alguns benefícios adicionais, como:

- **Escalabilidade:** na maior parte dos servidores de aplicações modernos, é possível distribuir a carga de processamento de aplicações desenvolvidas em diversos servidores, sendo que servidores podem ser adicionados ou removidos de maneira a acompanhar o aumento ou decréscimo dessa carga de processamento.
- **Eficiência:** os Servlets carregados por um servidor persistem em sua memória até que ele seja finalizado. Assim, ao contrário de outras tecnologias, não são iniciados novos processos para atender cada requisição recebida; por outro lado, uma mesma estrutura alocada em memória pode ser utilizada no atendimento das diversas requisições que chegam a esse mesmo Servlet.
- **Recompilação automática:** páginas JSP modificadas podem ser automaticamente recompiladas, de maneira que passem a incorporar imediatamente as alterações sem que seja necessário interromper o funcionamento da aplicação como um todo.

Criando o Servlet e implementando o cadastro do Cliente:

```
package br.com.brq.control;

import java.io.IOException;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.brq.entities.Cliente;
import br.com.brq.entities.types.EstadoCivil;
import br.com.brq.entities.types.Sexo;
import br.com.brq.persistence.DAOCliente;

@WebServlet("/ControleCliente") //nome que será chamado pelo formulario na JSP..
public class ControleCliente extends HttpServlet {
	private static final long serialVersionUID = 1L;

	//construtor.
	public ControleCliente() {
		super();
	}

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

		//resgatar a variavel 'acao' enviada pelos formularios..
		String acao = request.getParameter("acao");

		if(acao != null){ //se a variavel obteve valor..
			if(acao.equals("cadastrar")){

				try{

					Cliente c = new Cliente(); //entidade

					c.setNome(request.getParameter("nome"));
					c.setEmail(request.getParameter("email"));
					c.setSexo(Sexo.valueOf
						(request.getParameter("sexo")));
					c.setEstadoCivil(EstadoCivil.valueOf
						(request.getParameter("estadocivil")));

					DAOCliente d = new DAOCliente(); //persistencia
					d.insert(c); //gravando..

					//enviar um valor para a página..
					request.setAttribute("mensagem",
						"Cliente cadastrado com sucesso.");
				}
				catch(Exception e){

					//imprimir mensagem de erro..
```

```
                                request.setAttribute("mensagem", "Erro: "
                                        + e.getMessage());
                        }
                        finally{ //finalizador..
                                //redirecionar de volta para a página de cadastro..
                                request.getRequestDispatcher("cadastro.jsp")
                                        .forward(request, response);
                        }

                }
        }
    }

    //receber requições do tipo HTTP GET
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
        //PrintWriter out = response.getWriter(); //impressão..
        //out.print("recebendo chamada GET..");
        execute(request, response);
    }

    //receber requisições do tipo HTTP POST
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        //PrintWriter out = response.getWriter();
        //out.println("recebendo chamada POST..");
        execute(request, response);
    }
}
```

EL – Expression Language

A EL é uma linguagem de expressões utilizada na criação de páginas web dinâmicas na plataforma Java EE

Exibindo a mensagem na página:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
```

```
        <h3>Formulário de Cadastro de Clientes</h3>
        Para incluir um novo cliente, informe os dados abaixo:
        <hr/>

        <p>
                <a href="index.jsp">Voltar</a> para a página inicial.
        </p>

        <!-- FORMULARIO -->
        <form name="formulario" method="post"
                action="ControleCliente?acao=cadastrar">

                <label>Nome do Cliente:</label> <br/>
                <input type="text" name="nome"/>
                <br/><br/>

                <label>Email:</label> <br/>
                <input type="text" name="email"/>
                <br/><br/>

                <label>Sexo:</label> <br/>
                <select name="sexo">
                        <option value="">- Escolha uma Opção -</option>
                        <option value="Masculino">Masculino</option>
                        <option value="Feminino">Feminino</option>
                </select>
                <br/><br/>

                <label>Estado Civil:</label> <br/>
                <input type="radio" name="estadocivil" value="Solteiro"/>
                        Solteiro <br/>
                <input type="radio" name="estadocivil" value="Casado"/>
                        Casado <br/>
                <input type="radio" name="estadocivil" value="Viuvo"/>
                        Viuvo <br/>
                <input type="radio" name="estadocivil" value="Divorciado"/>
                        Divorciado <br/>
                <br/>

                <!-- Botão para enviar os dados do formulario para o servidor..
-->
                <input type="submit" value="Cadastrar Cliente"/>

                <!-- Botão para apagar o conteudo dos campos -->
                <input type="reset" value="Limpar"/>

                <!-- EL Expression Language -->
                <h4>${mensagem}</h4>

        </form>

</body>
</html>
```

Criando a ação no Servlet para gerar a consulta de Clientes:

```
package br.com.brq.control;

import java.io.IOException;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.brq.entities.Cliente;
import br.com.brq.entities.types.EstadoCivil;
import br.com.brq.entities.types.Sexo;
import br.com.brq.persistence.DAOCliente;

@WebServlet("/ControleCliente") //nome que será chamado pelo formulario na JSP..
public class ControleCliente extends HttpServlet {
	private static final long serialVersionUID = 1L;

	//construtor.
	public ControleCliente() {
		super();
	}

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

		//resgatar a variavel 'acao' enviada pelos formularios..
		String acao = request.getParameter("acao");

		if(acao != null){ //se a variavel obteve valor..
			if(acao.equals("cadastrar")){

				try{

					//resgatar os campos do formulario de cadastro de clientes..
					Cliente c = new Cliente(); //entidade

					c.setNome(request.getParameter("nome"));
					c.setEmail(request.getParameter("email"));
					c.setSexo(Sexo.valueOf(request.getParameter("sexo")));
					c.setEstadoCivil(EstadoCivil.valueOf
						(request.getParameter("estadocivil")));

					DAOCliente d = new DAOCliente(); //persistencia
					d.insert(c); //gravando..

					//enviar um valor para a página..
					request.setAttribute("mensagem",
						"Cliente cadastrado com sucesso.");
```

```
                }
                catch(Exception e){
                        //imprimir mensagem de erro..
                        request.setAttribute("mensagem", "Erro: "
                                                + e.getMessage());
                }
                finally{ //finalizador..
                        //redirecionar de volta para a página de cadastro..
                        request.getRequestDispatcher("cadastro.jsp")
                                .forward(request, response);
                }

        }
        else if(acao.equals("consultar")){

                try{

                        //resgatando o valor do campo nome do formulario..
                        String nome = request.getParameter("nome");
                                //campo nome..

                        //Classe de persistencia..
                        DAOCliente d = new DAOCliente();
                        List<Cliente> lista = d.findAllByNome(nome);

                        //enviando para a página..
                        request.setAttribute("dados", lista); //enviando a lista..
                        request.setAttribute("mensagem",
                                "Consulta realizada com sucesso.");

                }
                catch(Exception e){
                        request.setAttribute("mensagem", "Erro: "
                                 + e.getMessage());
                }
                finally{
                        //redirecionar de volta para a página..
                        request.getRequestDispatcher("consulta.jsp")
                                .forward(request, response);
                }
        }
    }
}

//receber requições do tipo HTTP GET
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        //PrintWriter out = response.getWriter(); //impressão..
        //out.print("recebendo chamada GET..");
        execute(request, response);
}
```

```
        //receber requisições do tipo HTTP POST
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
                //PrintWriter out = response.getWriter();
                //out.println("recebendo chamada POST..");
                execute(request, response);
        }

}
```

JSTL – JSP Standart Tag Libraries

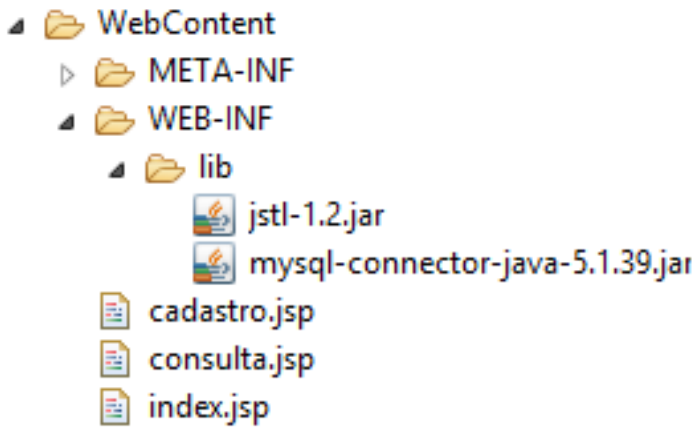
JSTL é o acrônimo de JavaServer Pages Standard Template Library, o qual podemos usar para recuperar dados de forma transparente usando como componente básico da JEE o qual é muito usado na programação pura, como costumo chamar quando programamos diretamente e tão somente no JSP (Java Server Pages).

Pode servir, como dito no parágrafo anterior, como mecanismo básico de recuperação de dados, de um banco de dados, de um arquivo de contexto e (ou) XML (XML (Extensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.).

Ainda falando sobre o uso da JSTL é bem interessante frisar que podemos através dela (JSTL), usar de maneira embutida, o código de lógica Java, sem necessariamente usar uma classe Java. Porém através do uso de Beans, ficando bem mais transparente e organizado. JSTL visa permitir que os programadores JSP usem tags em vez de código Java.

Incluindo as bibliotecas no projeto:

WebContent\WEB-INF\lib



Página de consulta de Clientes:

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

<!-- Registrar o JSTL (Biblioteca de TAGS do JSP) -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

	<h3>Consulta de Clientes</h3>
	Informe o nome do cliente desejado para realizar a busca.
	<hr/>

	<p>
		<a href="index.jsp">Voltar</a> para a página inicial.
	</p>

	<!-- FORMULARIO DE PESQUISA -->
	<form name="formulario" method="post"
		action="ControleCliente?acao=consultar">

		<label>Nome do Cliente:</label>
		<input type="text" name="nome"/>

		<!-- Botão para enviar os dados para o servidor -->
		<input type="submit" value="Pesquisar"/>

		<!-- Botão para limpar o conteudo dos campos -->
		<input type="reset" value="Limpar"/>

		<h4>${mensagem}</h4>

	</form>

	<h4>Resultado da busca:</h4>

	<table width="100%" border="1">
		<thead>
			<tr>
				<th>Id do Cliente</th>
				<th>Nome do Cliente</th>
				<th>Sexo</th>
				<th>Estado Civil</th>
				<th>Email</th>
			</tr>
		</thead>
```

```
            <tbody>

                  <!-- Varrer a lista (dados) usando JSTL -->
                  <c:forEach items="${dados}" var="cli">

                        <tr>
                              <td> ${cli.idCliente} </td>
                              <td> ${cli.nome} </td>
                              <td> ${cli.email} </td>
                              <td> ${cli.sexo} </td>
                              <td> ${cli.estadoCivil} </td>
                        </tr>

                  </c:forEach>

            </tbody>
      </table>

</body>
</html>
```