

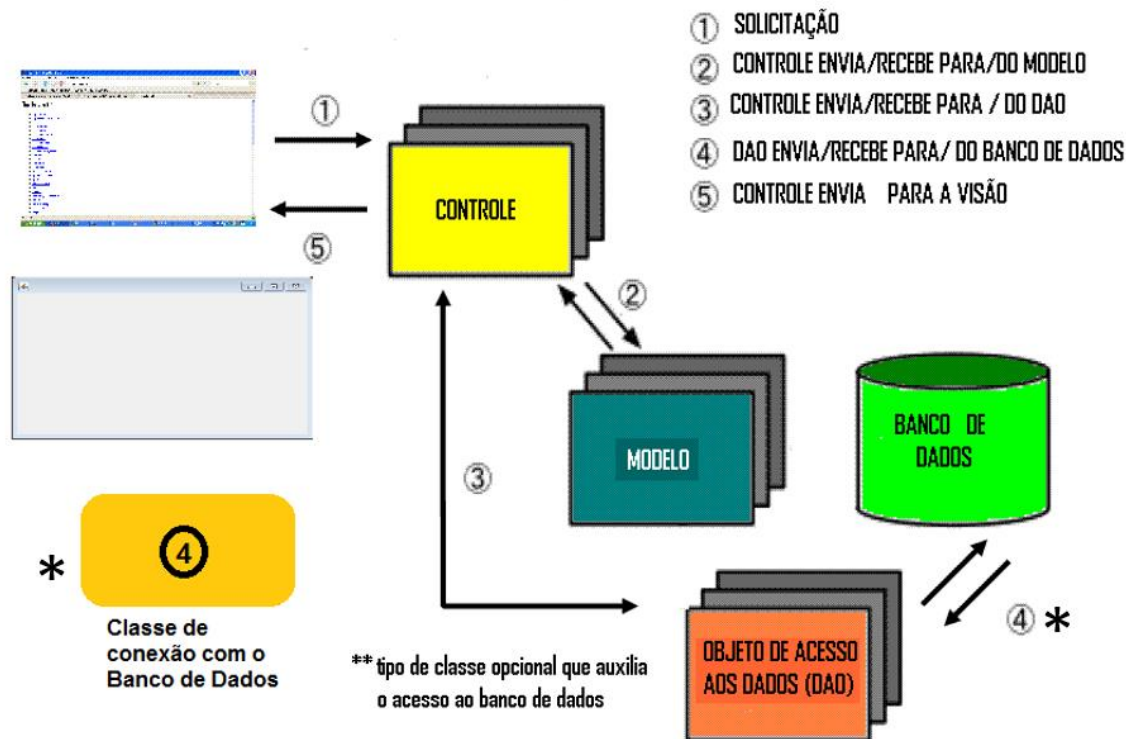


# MVC, JDBC e Banco de Dados

Professor Vicente Paulo de Camargo

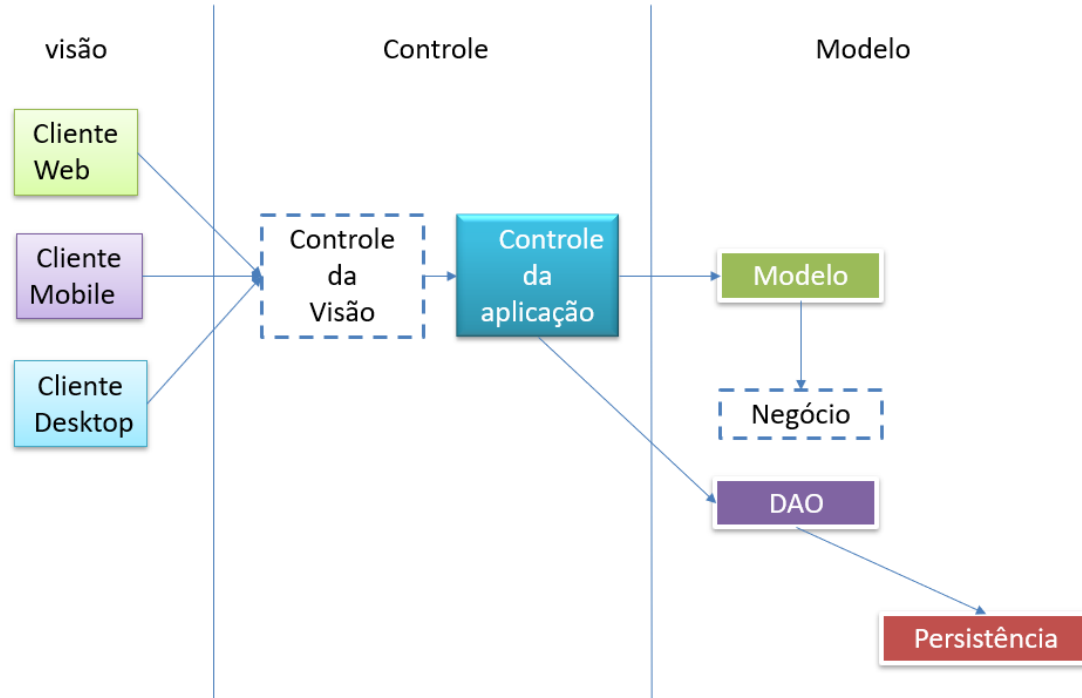
# MVC, JDBC e Banco de Dados

## Padrão MVC



# MVC, JDBC e Banco de Dados

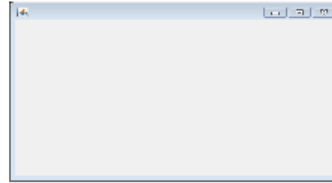
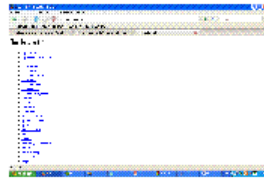
## Padrão MVC





# MVC, JDBC e Banco de Dados

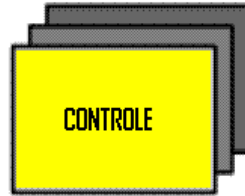
## Visão



- Responsável pela interação entre o usuário e o sistema
- Pode ser através de linha de comando, uma janela gráfica ou uma página web
- Se comunica com o sistema enviando informações para a camada de controle
- Geralmente valida os dados informados pelo usuário

# MVC, JDBC e Banco de Dados

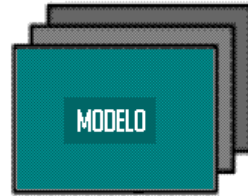
## Controle



- Responsável por receber informações (objetos e dados) da visão e controlar as ações com as demais camadas
- Permite efetuar validações de negócio
- Pode efetuar ações de eventos da camada de visão
- Pode existir um controle para a visão e um controle de negócio

# MVC, JDBC e Banco de Dados

## Modelo



- Responsável por armazenar provisoriamente os dados que serão persistidos
- Possui, entre outras, as classes beans da aplicação

# MVC, JDBC e Banco de Dados

## Persistência DAO e Conexão



- Recebe os objetos da camada de modelo do sistema
- Responsável por executar os scripts SQL
- Se conecta com a classe de conexão com o banco de dados
- Deve, preferencialmente, retornar objetos, coleções de objetos e retornos de métodos bem ou mal sucedidos
- A conexão permite o acesso flexível ao banco de dados



# MVC, JDBC e Banco de Dados

## DICAS

- Após a criação do projeto na IDE escolhida
- Incluir adequadamente a biblioteca do driver do banco de dados no projeto
- Criar os pacotes:  
**Exemplo: modelo, controle, dao, persistencia, visao, util e outros**
- Desenvolver as classes do modelo
- Desenvolver a classe de conexão
- Desenvolver as classes de acesso a dados (DAO)
- Desenvolver as classes de controle (s)
- Desenvolver as classes da camada de visão

### NOMES DOS PACOTES:

**urlContrária+nomeprojeto+pacote**

:

br.edu.pucgoias.teste.dao

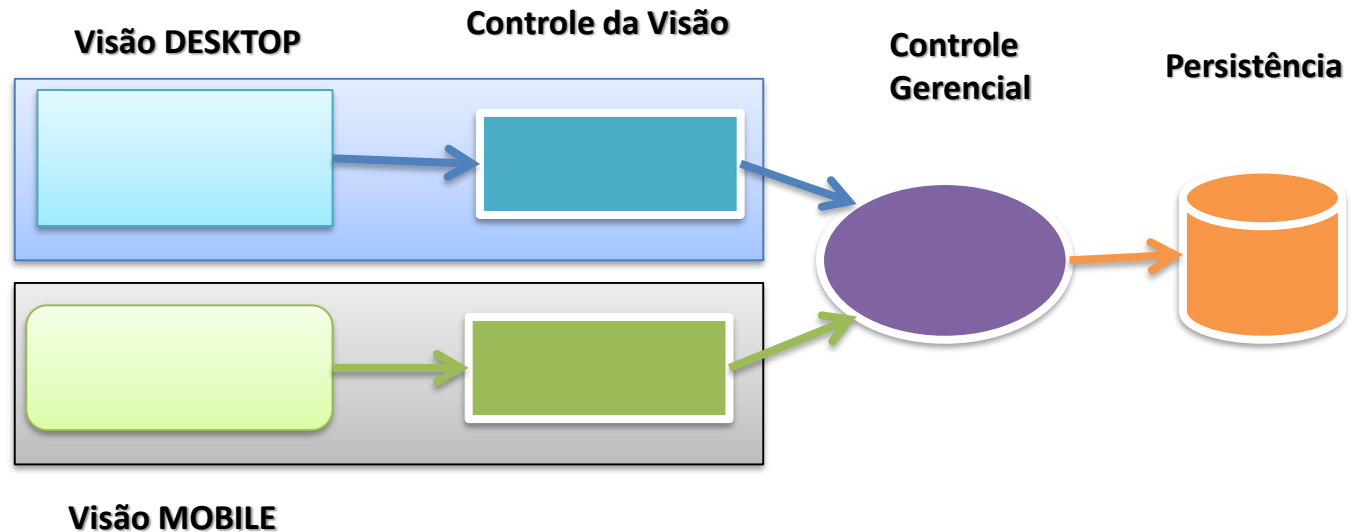
br.edu.pucgoias.teste.modelo



# MVC, JDBC e Banco de Dados

## Dica

- Para melhor flexibilidade da aplicação é interessante criar dois controles para uma mesma visão:
- **Controle da visão (CV)** : Responsável pela comunicação direta com a visão
- **Controle gerencial: (CG)**: Responsável pela comunicação entre o CV e com as demais camadas da aplicação , principalmente com a camada de persistência





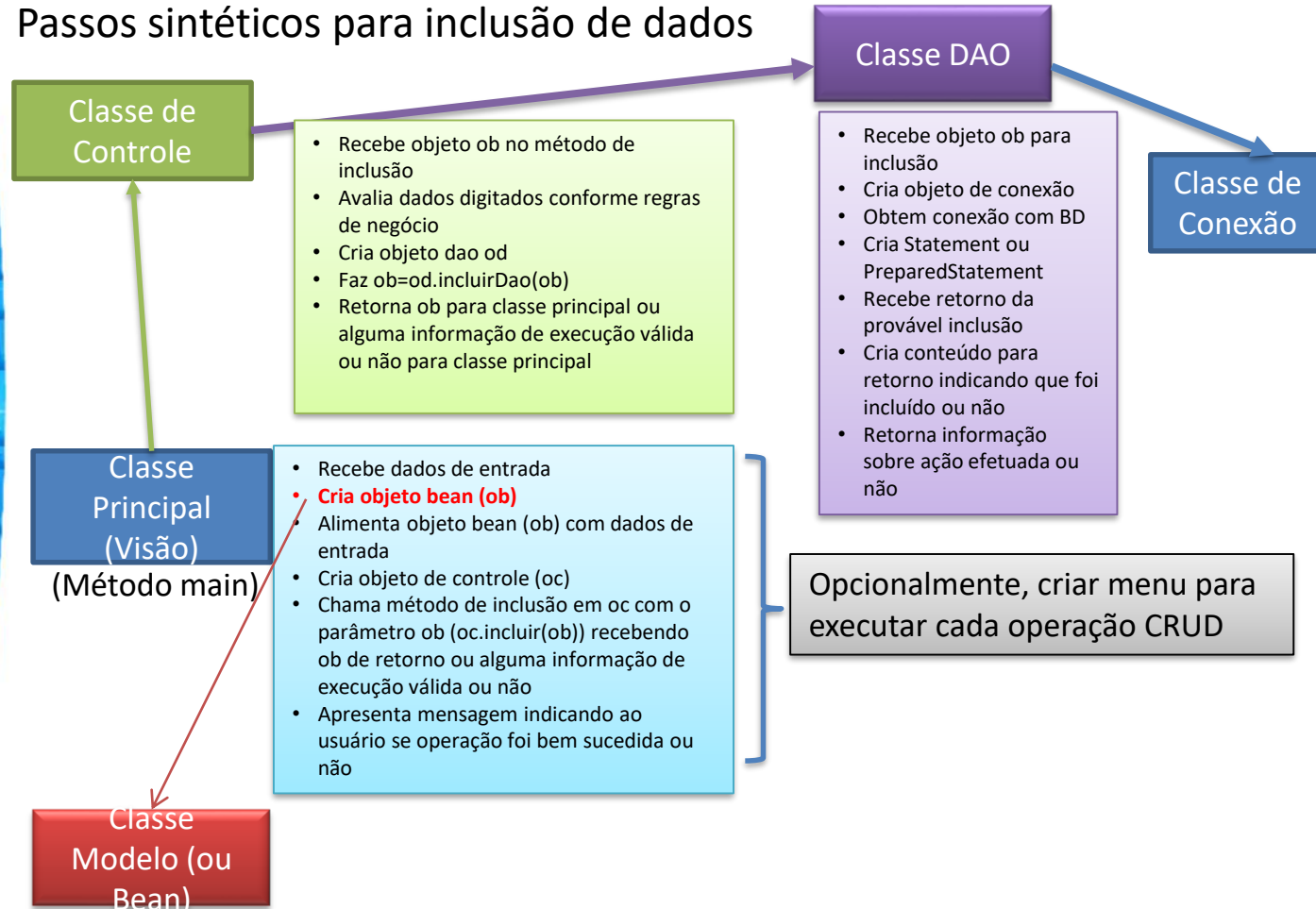
## MVC, JDBC e Banco de Dados

### Dica Importante

**NUNCA UTILIZAR QUALQUER TIPO DE INSTRUÇÃO QUE ENVOLVA A APRESENTAÇÃO DE DADOS PARA O USUÁRIO NAS CAMADAS QUE NÃO CORRESPONDAM À VISÃO (modelo, dao, etc), EXCETO NAS CLASSES DE VISÃO OU EM UMA CLASSE DE CONTROLE QUE ESTEJA DIRETAMENTE LIGADA A UMA VISÃO ESPECÍFICAMENTE.**

# MVC, JDBC e Banco de Dados

## Passos sintéticos para inclusão de dados







# MVC, JDBC e Banco de Dados

- Destaca-se nesses slides informações sobre a parte de persistência de dados
- Em Java, para se conectar com um banco de dados utiliza-se a biblioteca JDBC (Java Database Connectivity)
- Em Java, cada banco de dados possui um arquivo de conexão compatível com a biblioteca JDBC
- Esse arquivo, com extensão .jar, deve ser inserido no projeto para que a conexão com o respectivo banco de dados possa ocorrer satisfatoriamente



# MVC, JDBC e Banco de Dados

abra o Eclipse e crie o projeto dynamic web **SistemaEstoque**

- Para esse projeto será utilizado a plataforma MySQL de banco de dado
- Após criar o projeto, crie o pacote **br.edu.pucgoias.sistemaestoque.dao**



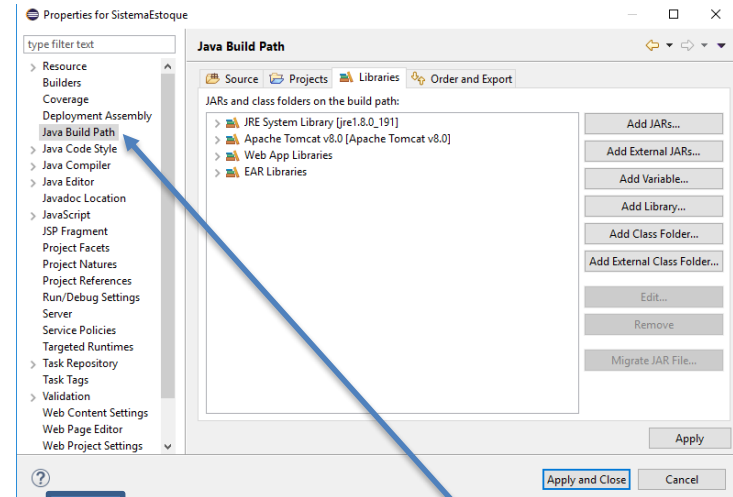
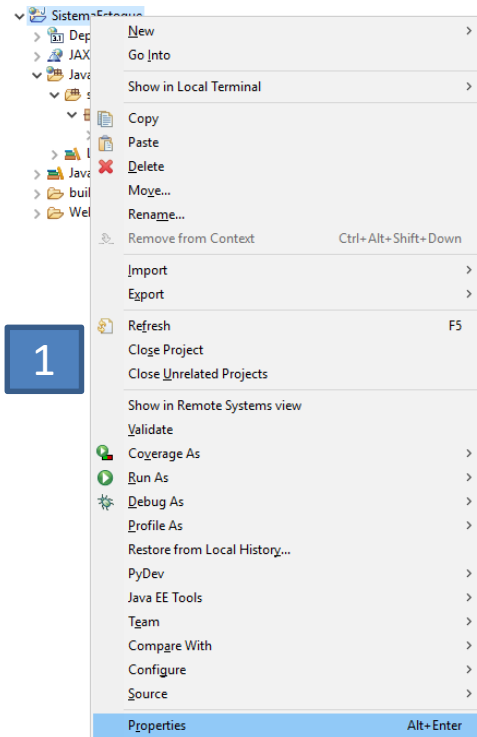
# MVC, JDBC e Banco de Dados

- O driver de conexão dos principais bancos de dados são encontrados na Internet
- Procure pelo driver de conexão do MySQL como **MySQL JDBC connector**
- Efetue o download do arquivo, geralmente compactado
- Descompacte esse arquivo
- Utilize apenas o arquivo com extensão .jar, cujo nome é formado como mysql-conector-java-x.x.xx-bin.jar, onde x.x.xx é a versão



# MVC, JDBC e Banco de Dados

- Com o Eclipse aberto, dê um clique direito sobre o nome do projeto SistemaEstoque, como indicado a seguir

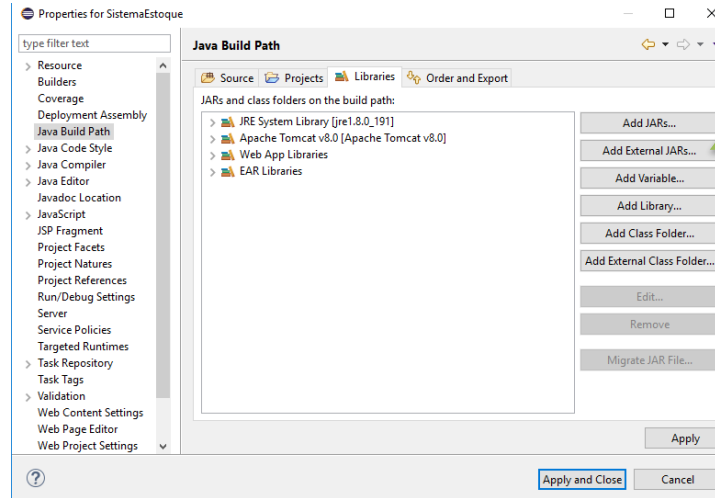


Selecione  
Build Path

# MVC, JDBC e Banco de Dados

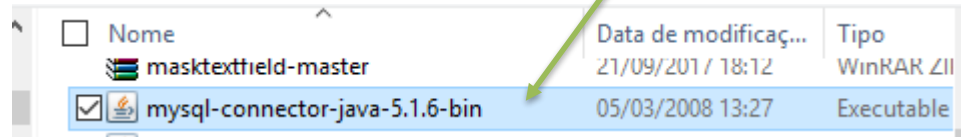
1

Selecione Add  
External  
JARs...

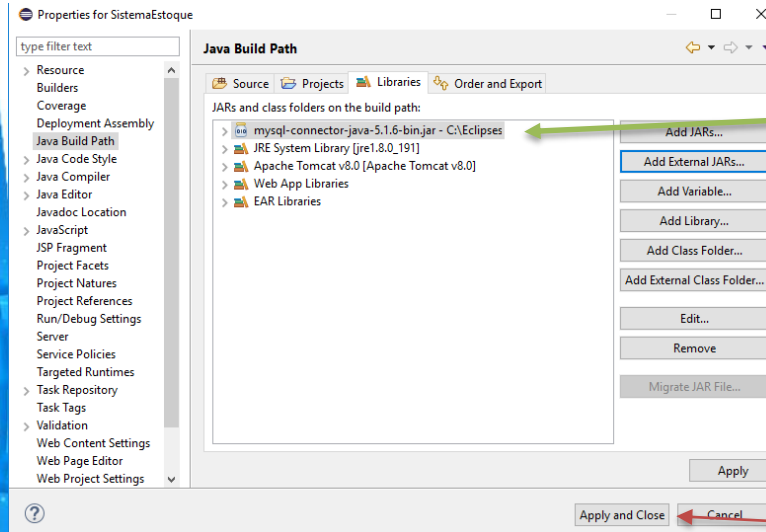


2

Selecione o  
driver  
desejado

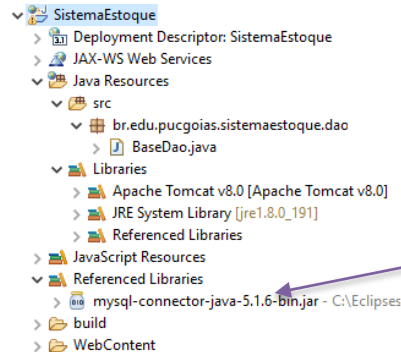


# MVC, JDBC e Banco de Dados



O driver será inserido na aba **Libraries** do projeto

Em seguida, confirme em **Apply and Close**



O driver do MySQL inserido na biblioteca do projeto



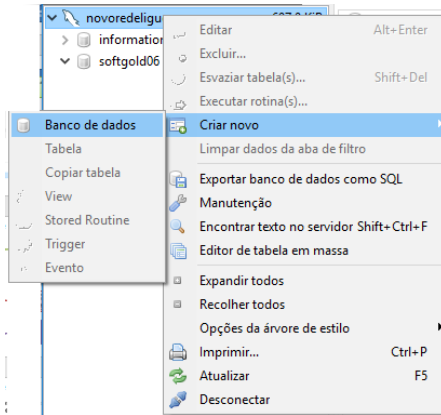


# MVC, JDBC e Banco de Dados

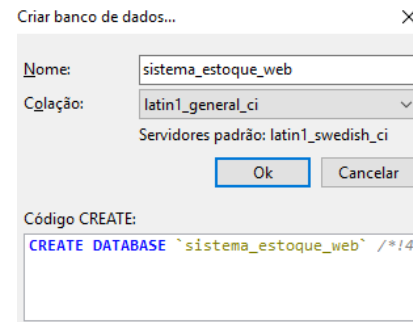
- Para administrar o banco de dados e manipular seus dados utilize os programas HeidiSQL ou Dbeaver
- Efetue o download desses dois programas
- Para servidor de banco de dados MySQL utilize um dos programas, WampServer ou Xampp
- Assim, instale um desses dois últimos programas para que sejam o servidor do seu banco de dados
- O servidor do banco de dados sempre deve estar ativo para que se possa efetuar a devida conexão com o banco de dados
- Esses programas são open source

# MVC, JDBC e Banco de Dados

- Ative o seu servidor MySQL
- Abra o HeidiSQL e crie o banco de dados **sistema\_estoque\_web**



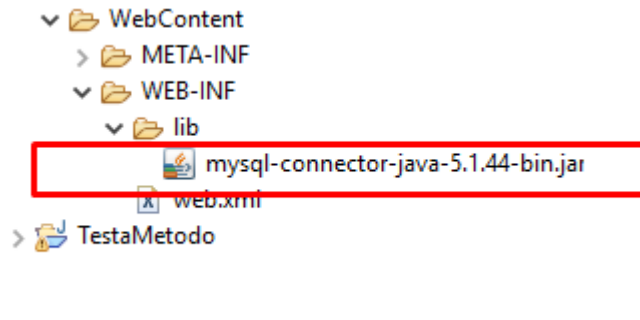
Botão direito sobre o nome da conexão



Na nova janela informe o nome do banco de dados e selecione a opção latin1\_general\_ci e confirme

# MVC, JDBC e Banco de Dados

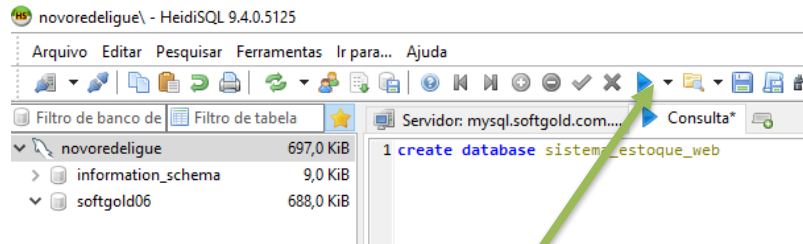
- Copie o arquivo .jar para a pasta indicada





# MVC, JDBC e Banco de Dados

- Ou:
- Abra o HeidiSQL, acesse a aba consulta
- Digite **create database sistema\_estoque\_web**



Selecione esse ícone para executar ou pressione F9



# MVC, JDBC e Banco de Dados

- Crie a classe **BaseDao.java** e digite o próximo trecho de código para esse classe

# MVC, JDBC e Banco de Dados

```
1 package br.edu.pucgoias.sistemaestoque.dao;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class BaseDao {
8
9     public BaseDao() {
10
11         try {
12             Class.forName("com.mysql.jdbc.Driver");
13
14         }
15         catch(ClassNotFoundException e) {
16
17         }
18     }
19
20     public Connection getConnection() throws SQLException {
21
22         String url="jdbc:mysql://localhost/sistema_estoque_web";
23
24         Connection conn = DriverManager.getConnection(url, "vicente", "vicente");
25
26         return conn;
27     }
28
29
30
31     public static void main(String[] args) throws SQLException
32     {
33         BaseDao bd = new BaseDao();
34         Connection conn = bd.getConnection();
35
36         if (conn != null)
37             System.out.println("Conectou !!");
38         else
39             System.out.println("Não conectou !!");
40     }
41 }
```

Linha 12: Estabelece a conexão com o driver do banco de dados

Linha 24: Faz a conexão com o banco de dados indicado na url e conforme usuário e senha definidos

Linha 34: Tenta efetuar conexão com o banco de dados.

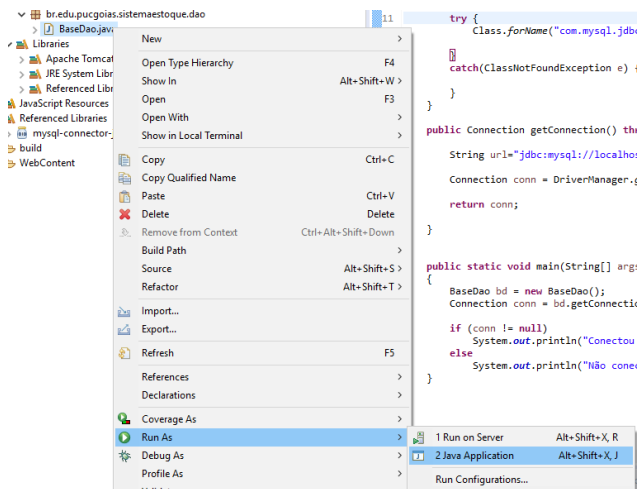
Linha 37: Se conectar, apresenta mensagem afirmativa

O método **main** está sendo usado aqui apenas para que se possa testar a conexão com o banco de dados, usando System.out.print



# MVC, JDBC e Banco de Dados

- Com o Eclipse aberto
- Acesse o projeto **SistemaEstoque**
- Dê um clique com o botão direito sobre a classe BaseDao.java
- Observe os passos a seguir:

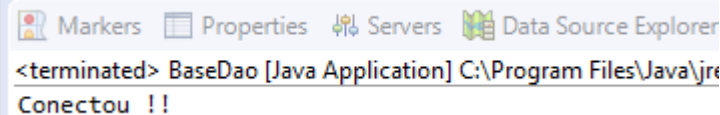


1

2

3

A conexão está ok

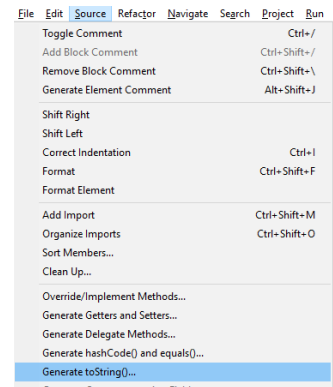


# MVC, JDBC e Banco de Dados

Crie a classe bean `Estoque.java` ( com getters/setters), com os atributos: `id` (int), `descricao`(String), `quantidade`(double) e `precounit`(double), no pacote **modelo** (**br.edu.pucgoias.sistemaestoque.modelo**)

```
1 package br.edu.pucgoias.sistemaestoque.modelo;
2
3 public class Estoque {
4
5     private int id;
6     private String descricao;
7     private double quantidade;
8     private double precounit;
9
10    public int getId() {
11        return id;
12    }
13    public void setId(int id) {
14        this.id = id;
15    }
16    public String getDescricao() {
17        return descricao;
18    }
19    public void setDescricao(String descricao) {
20        this.descricao = descricao;
21    }
22    public double getQuantidade() {
23        return quantidade;
24    }
25    public void setQuantidade(double quantidade) {
26        this.quantidade = quantidade;
27    }
28    public double getPrecounit() {
29        return precounit;
30    }
31    public void setPrecounit(double precounit) {
32        this.precounit = precounit;
33    }
34
35    @Override
36    public String toString() {
37        return "Estoque [id=" + id + ", descricao=" + descricao + ", quantidade=" + quantidade + ", precounit="
38            + precounit + "];"
39    }
40 }
```

Para criar o método **toString**, posicione o cursor no final da classe **Estoque.java** e acesse a opção **source>Generate toString...** no Eclipse



# MVC, JDBC e Banco de Dados

Crie a tabela **estoque** no banco de dados **sistema\_estoque\_web** com a estrutura:

Campo	Tipo	observação
id	int	Primary key, auto_incremente
descricao	varchar(50)	
precounit	double	
quantidade	double	

```
1 CREATE TABLE `estoque` (  
2   `id` INT(11) NOT NULL AUTO_INCREMENT,  
3   `descricao` VARCHAR(50) NULL DEFAULT NULL,  
4   `precounit` DOUBLE NULL DEFAULT NULL,  
5   `quantidade` DOUBLE NULL DEFAULT NULL,  
6   PRIMARY KEY (`id`)  
7 )
```



# MVC, JDBC e Banco de Dados

Crie a classe de persistência `EstoqueDao.java` no pacote **dao** (**br.edu.pucgoias.sistemaestoque.dao**)

```
1 package br.edu.pucgoias.sistemaestoque.dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Statement;
8 import java.util.ArrayList;
9 import java.util.List;
10
11
12
13 import br.edu.pucgoias.sistemaestoque.modelo.Estoque;
14
15 public class EstoqueDao extends BaseDao {
16
17     public Estoque getEstoqueViaId(int id) {
18
19         Estoque estoque = new Estoque();
20         PreparedStatement pstm = null;
21         Connection conn=null;
22         try
23         {
24             conn = this.getConnection();
25             pstm = (PreparedStatement) conn.prepareStatement("select * from estoque where id=?");
26             pstm.setInt(1,id);
27             ResultSet rs = pstm.executeQuery();
28             if (rs.next())
29             {
30                 estoque = criaEstoque(rs);
31             }
32         }
33         catch(SQLException e)
34         {
35             //
36         }
37         return estoque;
38     }
39
40 }
41
```

DAO (Data Access Object, Objeto de Acesso a Dados) é um padrão de nomenclatura para classes que acessam bases de dados. Esse padrão faz parte do núcleo de padrões JEE da Sun/Oracle.

# MVC, JDBC e Banco de Dados

## EstoqueDao.java (Cont.)

```
42 public Estoque criaEstoque(ResultSet rs) throws SQLException {
43     Estoque estoque = new Estoque();
44     estoque.setDescricao(rs.getString("descricao"));
45     estoque.setId(rs.getInt("id"));
46     estoque.setPrecounit(rs.getDouble("precounit"));
47     estoque.setQuantidade(rs.getDouble("quantidade"));
48     return estoque;
49 }
50
51 public List<Estoque> getEstoqueViaNome(String nome) {
52
53     List<Estoque> lista = new ArrayList<>();
54     Estoque estoque = new Estoque();
55     PreparedStatement pstmt = null;
56     Connection conn=null;
57     try
58     {
59         conn = this.getConnection();
60         String sql="select * from estoque where lower(descricao) like ? order by nome";
61         pstmt = (PreparedStatement) conn.prepareStatement(sql);
62         pstmt.setString(1,"%"+nome.toLowerCase()+"%");
63         ResultSet rs = pstmt.executeQuery();
64         while (rs.next())
65         {
66             estoque = criaEstoque(rs);
67             lista.add(estoque);
68         }
69     }
70     catch(SQLException e)
71     {
72         //
73     }
74     return lista;
75 }
76
77
78 }
```

# MVC, JDBC e Banco de Dados

## EstoqueDao.java (Cont.)

```
79 public boolean salvarEstoque(Estoque estoque) {
80     boolean retorno=false;
81     String sql="";
82     PreparedStatement pstm = null;
83     Connection conn=null;
84     try
85     {
86         conn = this.getConnection();
87         if (estoque.getId() == 0)
88         {
89             sql = "insert into estoque (descricao, precounit, quantidade) values ";
90             sql+=" (?, ?, ?)";
91             pstm = (PreparedStatement) conn.prepareStatement(sql,Statement.RETURN_GENERATED_KEYS);
92         }
93         else
94         {
95             sql = "update estoque set descricao=?, precounit=?, quantidade=?";
96             sql+=" where id=?";
97             pstm = (PreparedStatement) conn.prepareStatement(sql);
98         }
99         pstm.setString(1, estoque.getDescricao());
100         pstm.setDouble(2, estoque.getPrecounit());
101         pstm.setDouble(3,estoque.getQuantidade());
102         if (estoque.getId() !=0)
103         {
104             //update
105             pstm.setInt(4, estoque.getId());
106         }
107         int idAux=pstm.executeUpdate();
108         if (idAux==0)
109             retorno=false;
110         if (estoque.getId()==0)
111         {
112             int idInserir = getGeneratedId(pstm);
113             estoque.setId(idInserir);
114         }
115         retorno = true;
116     }
117     catch(SQLException e)
118     {
119         retorno = false;
120     }
121     return retorno;
122 }
```



# MVC, JDBC e Banco de Dados

## EstoqueDao.java (cont.)

```
123 public static int getGeneratedId(PreparedStatement stm) throws SQLException
124 {
125     ResultSet rs = stm.getGeneratedKeys();
126     if (rs.next())
127     {
128         int id = rs.getInt(1);
129         return id;
130     }
131     return 0;
132 }
133 public boolean excluir(int id) {
134     PreparedStatement pstmt = null;
135     Connection conn=null;
136     try
137     {
138         conn = this.getConnection();
139         pstmt = (PreparedStatement) conn.prepareStatement("delete from estoque where id=?");
140         pstmt.setInt(1,id);
141         int conta = pstmt.executeUpdate();
142         boolean retorno = conta > 0;
143         return retorno;
144     }
145     catch(SQLException e)
146     {
147         return false;
148     }
149 }
150
151
152 }
```

# MVC, JDBC e Banco de Dados

EstoqueDao.java (cont.)

```
153 public List<Estoque> getTodos() {
154
155     List<Estoque> lista = new ArrayList<>();
156     Estoque estoque = new Estoque();
157     PreparedStatement pstmt=null;
158     ResultSet rs;
159     Connection conn=null;
160     try
161     {
162         conn = this.getConnection();
163         String sql="select * from estoque order by descricao";
164         pstmt = conn.prepareStatement(sql);
165
166         rs = pstmt.executeQuery();
167         while (rs.next())
168         {
169             estoque = criaEstoque(rs);
170             lista.add(estoque);
171         }
172     }
173     catch(SQLException e)
174     {
175         //
176     }
177     return lista;
178 }
179
180 }
181 }
```



# MVC, JDBC e Banco de Dados

Crie o pacote **br.edu.pucgoias.sistemaestoque.controle**

Crie a classe `EstoqueControle.java`

Esta classe efetuará a intermediação entre a camada de visão e a de persistência



# MVC, JDBC e Banco de Dados

## EstoqueControle.java

```
1 package br.edu.pucgoias.sistemaestoque.controle;
2
3 import java.util.List;
4
5
6
7 public class EstoqueControle {
8
9
10     private EstoqueDao ed = new EstoqueDao();
11
12     public List<Estoque> getEstoque(){
13
14         List<Estoque> estoques = ed.getTodos();
15         return estoques;
16     }
17
18     public Estoque getEstoquePorId(int id) {
19         return ed.getEstoqueViaId(id);
20     }
21
22     public boolean excluir(int id) {
23         return ed.excluir(id);
24     }
25
26     public boolean salvar(Estoque estoque) {
27
28         return ed.salvarEstoque(estoque);
29     }
30
31     public List<Estoque> buscaEstoquePorNome(String nome){
32         return ed.getEstoquesViaNome(nome);
33     }
34 }
--
```



# MVC, JDBC e Banco de Dados

Crie o pacote **br.edu.pucgoias.sistemaestoque.servlets**

Esse pacote armazenará as classes Servlet para receber requisições e retornar resultados para o cliente solicitante.

Assim, nesse pacote, crie o Servlet **ServletAtualizar.java**

Esse Servlet permitirá efetuar a inserção, edição e deleção de itens do estoque, conforme as informações recebidas

# MVC, JDBC e Banco de Dados

## Código do Servlet **ServletAtualizar.java**:

```
1 package br.edu.pucgoias.sistemaestoque.servlets;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.annotation.WebServlet;
7 import javax.servlet.http.HttpServlet;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 import br.edu.pucgoias.sistemaestoque.controle.EstoqueControl;
12 import br.edu.pucgoias.sistemaestoque.modelo.Estoque;
13 @WebServlet("/ServletAtualizar")
14 public class ServletAtualizar extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16
17     public ServletAtualizar() {
18         super();
19     }
20 }
```

```
21 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
22     String descricao= request.getParameter("descricao");
23     double precunit= 0;
24     double quantidade=0;
25     int id=0;
26     String strPu = request.getParameter("precunit");
27     String strQt = request.getParameter("quantidade");
28     String strId = request.getParameter("id");
29
30     precunit = Double.parseDouble(strPu);
31     quantidade =Double.parseDouble(strQt);
32     id =Integer.parseInt(strId);
33
34     String retorno="ERRO";
35     boolean acao=false;
36     if ((descricao==null || descricao.length()==0) && id !=0)
37         retorno="Descrição inválida";
38     else
39     {
40         Estoque estoque = new Estoque();
41         if ((descricao==null || descricao.length()==0) && id!=0) // exclusao
42         {
43             EstoqueControl ec = new EstoqueControl();
44             acao = ec.excluir(id);
45         }
46         else
47         {
48             estoque.setDescricao(descricao);
49             estoque.setPrecunit(precunit);
50             estoque.setQuantidade(quantidade);
51             estoque.setId(id);
52             EstoqueControl ec = new EstoqueControl();
53             acao = ec.salvar(estoque);
54         }
55         if (acao)
56             retorno="OK";
57
58         response.setContentType("text/html;charset=utf-8");
59         response.getWriter().print("resposta="+retorno);
60     }
61 }
62
63 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
64     // TODO Auto-generated method stub
65     doGet(request, response);
66 }
67 }
```



# MVC, JDBC e Banco de Dados

Crie o Servlet **ServletAll.java**, que permitirá mostrar todos os itens do estoque, cujo código é o seguinte:

```
1 package br.edu.pucgoias.sistemaestoque.servlets;
2 import java.io.IOException;
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8 import javax.servlet.http.HttpServlet;
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 import br.edu.pucgoias.sistemaestoque.controle.EstoqueControle;
13 import br.edu.pucgoias.sistemaestoque.modelo.Estoque;
14
15 @WebServlet("/ServletAll")
16 public class ServletAll extends HttpServlet {
17     private static final long serialVersionUID = 1L;
18
19     public ServletAll() {
20         super();
21     }
22
23     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
24
25         List<Estoque> lista = new ArrayList<>();
26         Estoque estoque;
27         EstoqueControle ec = new EstoqueControle();
28         lista = ec.getEstoque();
29         response.setContentType("text/html;charset=utf-8");
30         String html=<div align="center">;
31         html+<table border="2">;
32         html+<tr><td><strong>CÓDIGO</strong></td><td><strong>DESCRIÇÃO</strong></td><td><strong>QUANTIDADE</strong></td></tr>;
33         html+<tr><td><strong>P.UNIT</strong></td></tr>;
34         for(int i=0; i<lista.size();i++)
35         {
36             estoque = new Estoque();
37             estoque.setDescricao(lista.get(i).getDescricao());
38             estoque.setPrecounit(lista.get(i).getPrecounit());
39             estoque.setQuantidade(lista.get(i).getQuantidade());
40             estoque.setId(lista.get(i).getId());
41             html+<tr><td>estoque.getId()+"</td><td>estoque.getDescricao()+"</td>;
42             html+<td>estoque.getQuantidade()+"</td><td>estoque.getPrecounit()+"</td></tr>;
43         }
44         html+</table></div>;
45         response.getWriter().print(html);
46     }
47
48
49     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
50         doGet(request, response);
51     }
52 }
```



# MVC, JDBC e Banco de Dados

Todo código deve passar por testes para que se possa garantir sua execução em produção

Os códigos desenvolvidos no servidor necessitam ser também validados com testes

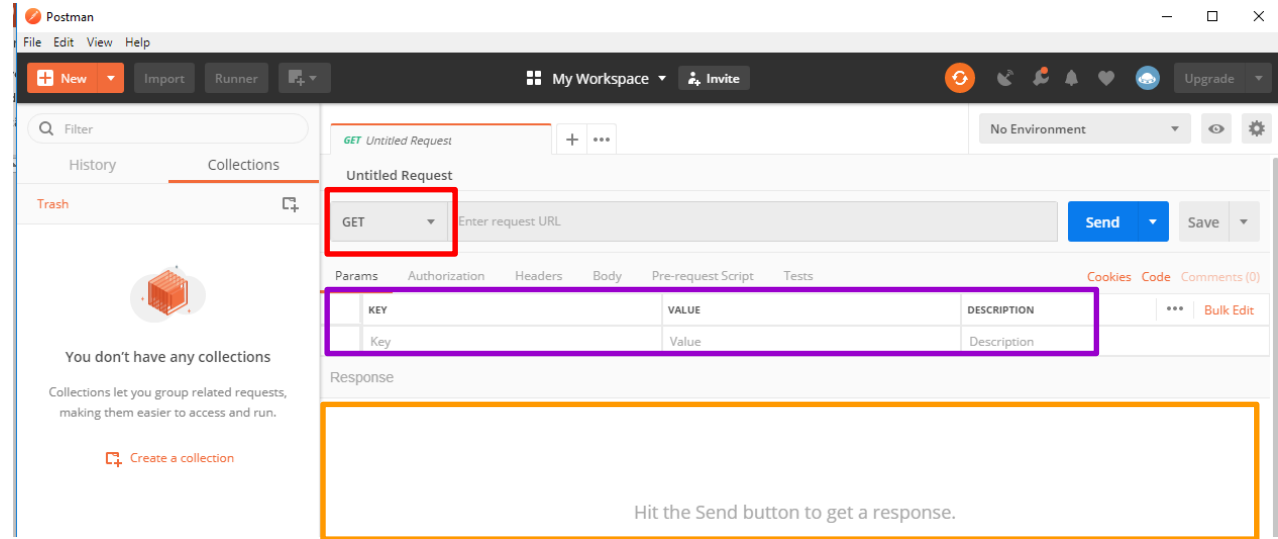
Um desenvolvedor back-end, por exemplo, não desenvolve a interface gráfica, pois não é de sua responsabilidade

No entanto, para realizar testes, mesmo que sejam simples, esse tipo de profissional deve utilizar de programas auxiliares que permitem acessar as respectivos endereços da aplicação (URL) e informar os parâmetros necessários para que atendam na realização dos diversos testes

Um desses programas é o **Postman**, que fornece recursos para acessar cada URL disponível para teste na aplicação, apresentando os resultados para análise pelo desenvolvedor

# MVC, JDBC e Banco de Dados

## Postman



Seleção de comandos 

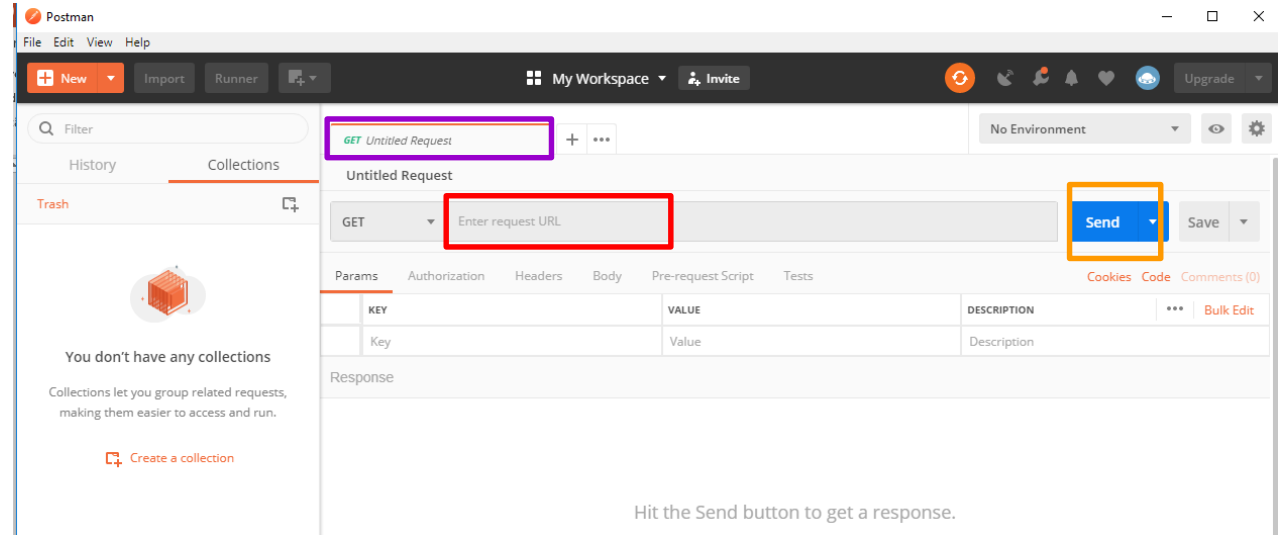
Parâmetros 

Resposta 



# MVC, JDBC e Banco de Dados

## Postman



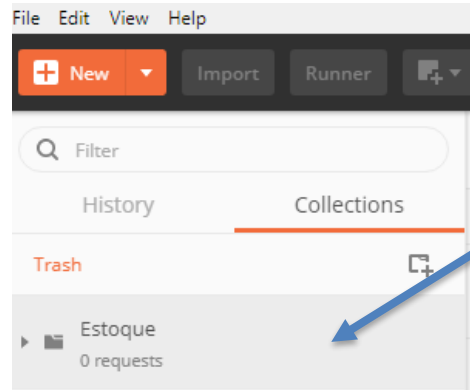
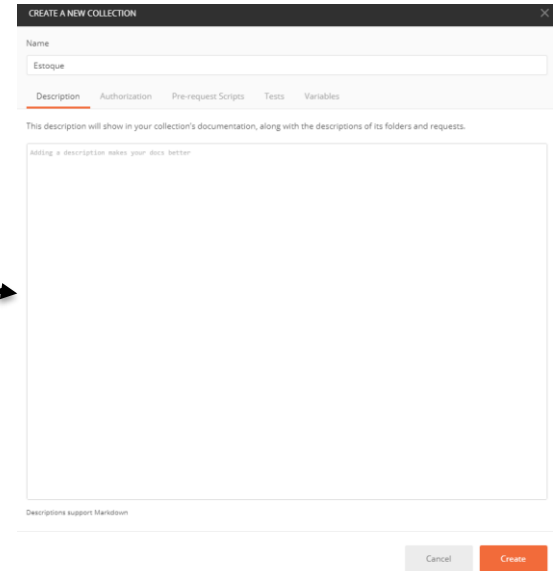
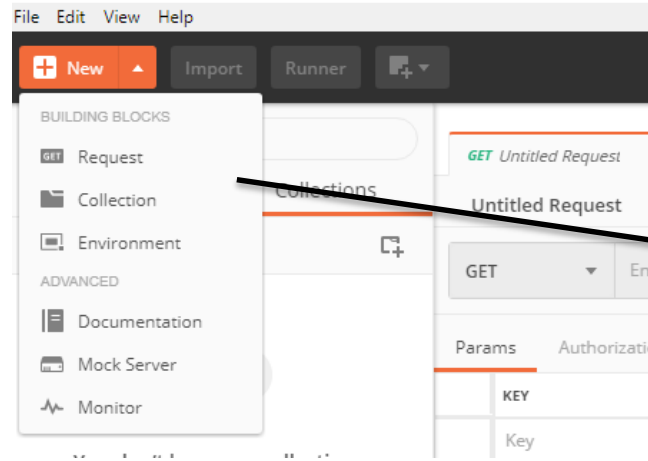
Inserir url (endpoints)  

Abas de urls  

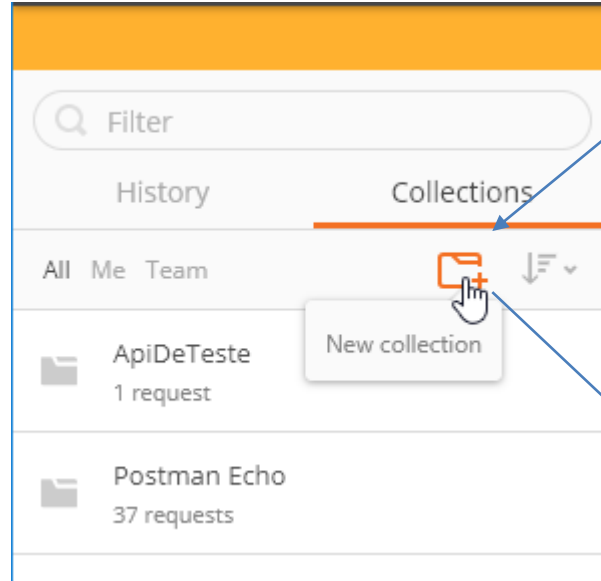
Envio

# MVC, JDBC e Banco de Dados

## Postman



# MVC, JDBC e Banco de Dados

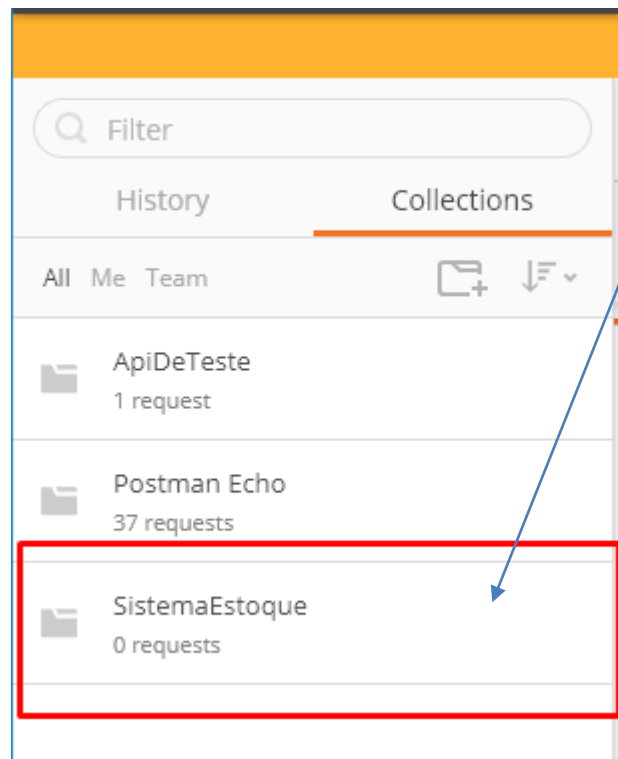


Aba o Postman  
Crie uma collection para  
acessar seu grupo de URLs

The screenshot shows the 'CREATE A NEW COLLECTION' dialog box. It has a title bar with a close button. The 'Name' field is filled with 'SistemaEstoque'. The 'Description' field is empty. At the bottom, there are 'Cancel' and 'Create' buttons, with the 'Create' button highlighted by a red box.

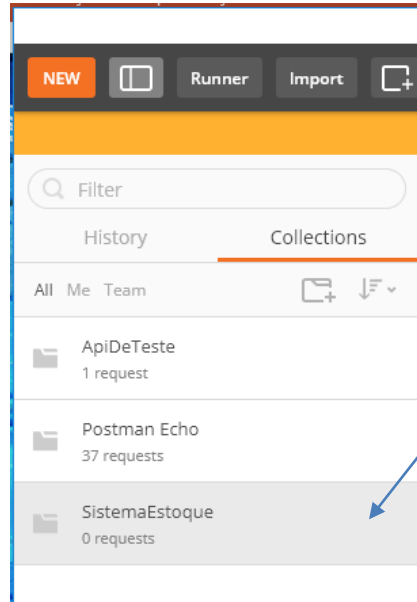


# MVC, JDBC e Banco de Dados



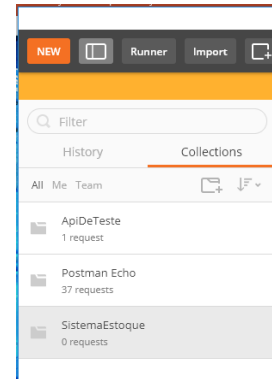
O grupo (collection) do SistemaEstoque foi criado.

# MVC, JDBC e Banco de Dados

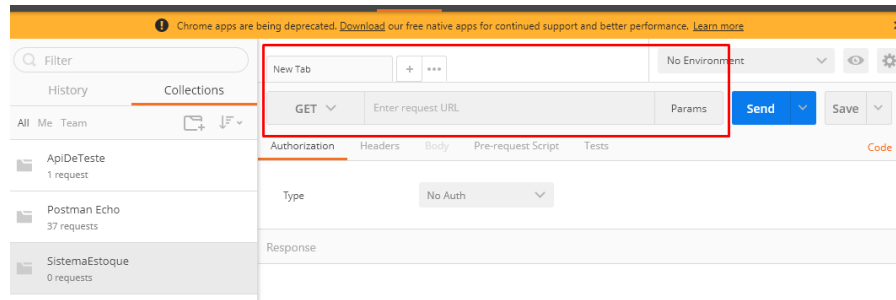


Selecione o grupo

Em seguida, clique nessa opção para abrir uma nova janela para o grupo desejado



# MVC, JDBC e Banco de Dados

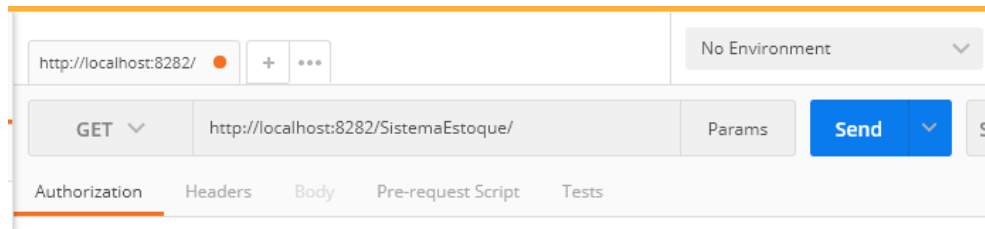


Se o Postman foi utilizado antes, a área em destaque se apresenta com muitas abas com URLs de outras aplicações. Quando se cria uma nova janela, esta se apresenta sem abas, permitindo a criação de novas abas, não misturando com as anteriores.

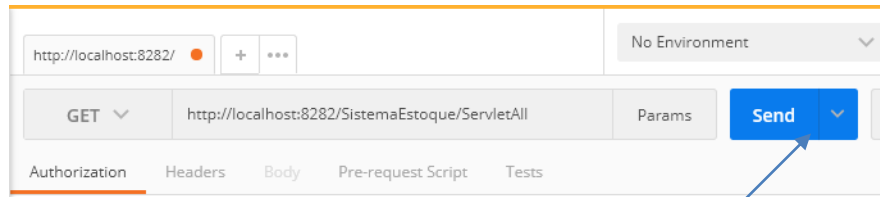


# MVC, JDBC e Banco de Dados

Execute a aplicação SistemaEstoque e deixe-a ativada. Copie o endereço da aplicação para o Postman, conforme está ilustrado:



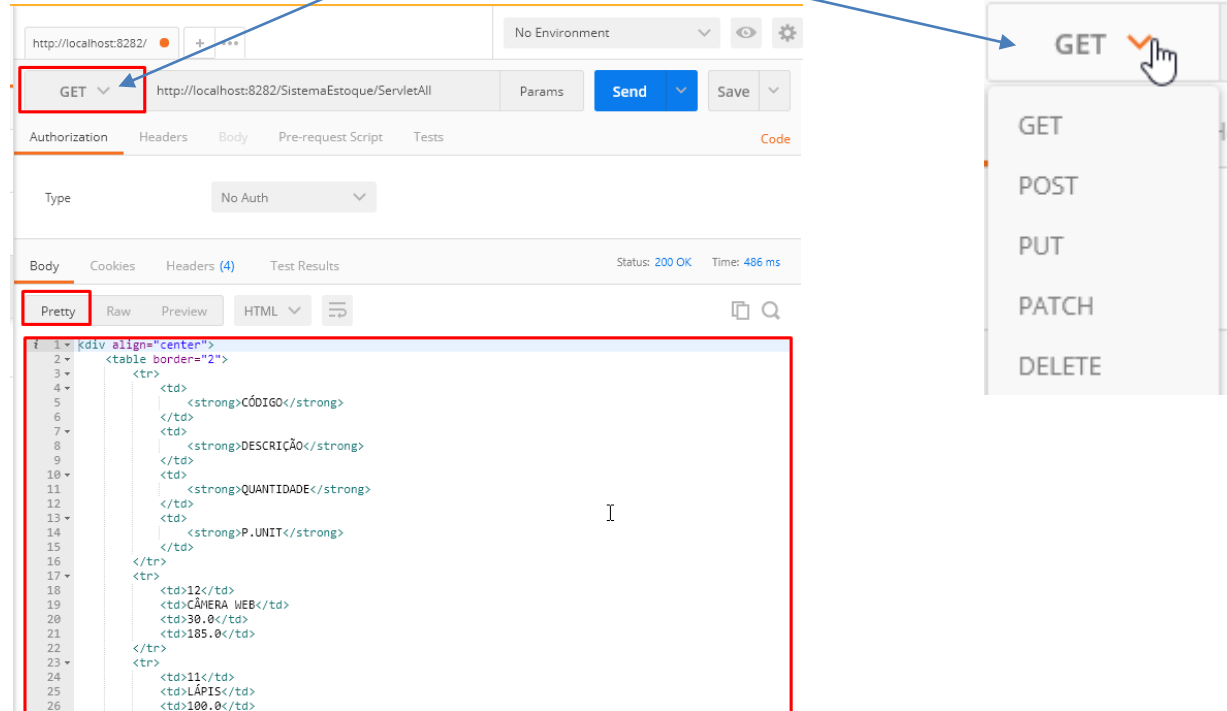
Complete a URL, como indicado a seguir:



Clique aqui para executar

# MVC, JDBC e Banco de Dados

Note que ao lado do GET há uma seta para baixo, indicando que se pode seleccionar outros tipos de métodos HTTP.



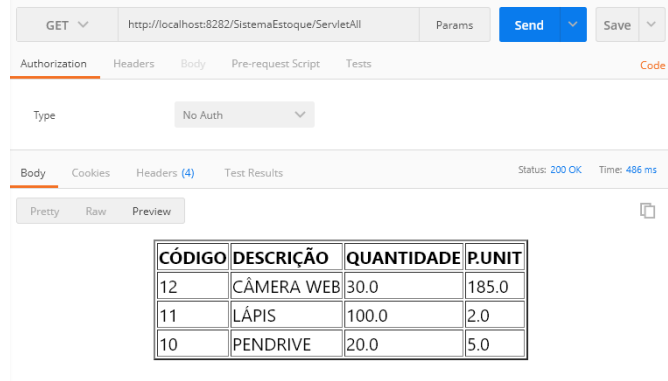
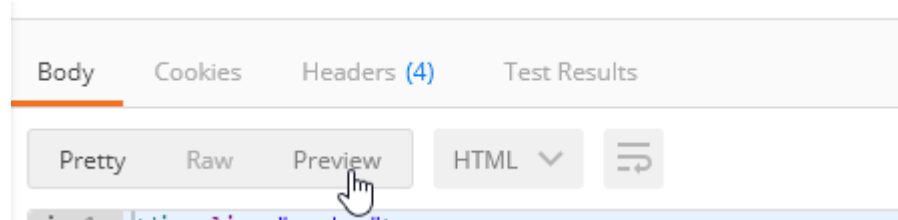
The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8282/`
- Method: `GET` (selected from a dropdown menu)
- Environment: `No Environment`
- Path: `http://localhost:8282/SistemaEstoque/ServletAll`
- Params: `Params`
- Buttons: `Send`, `Save`
- Authorization: `No Auth`
- Status: `200 OK`, Time: `486 ms`
- Response Body (Pretty):

```
1 <div align="center">
2   <table border="2">
3     <tr>
4       <td>
5         <strong>CÓDIGO</strong>
6       </td>
7       <td>
8         <strong>DESCRIÇÃO</strong>
9       </td>
10      <td>
11        <strong>QUANTIDADE</strong>
12      </td>
13      <td>
14        <strong>P.UNIT</strong>
15      </td>
16    </tr>
17    <tr>
18      <td>12</td>
19      <td>CÂMERA WEB</td>
20      <td>30.0</td>
21      <td>185.0</td>
22    </tr>
23    <tr>
24      <td>11</td>
25      <td>LÁPIS</td>
26      <td>100.0</td>
```

# MVC, JDBC e Banco de Dados

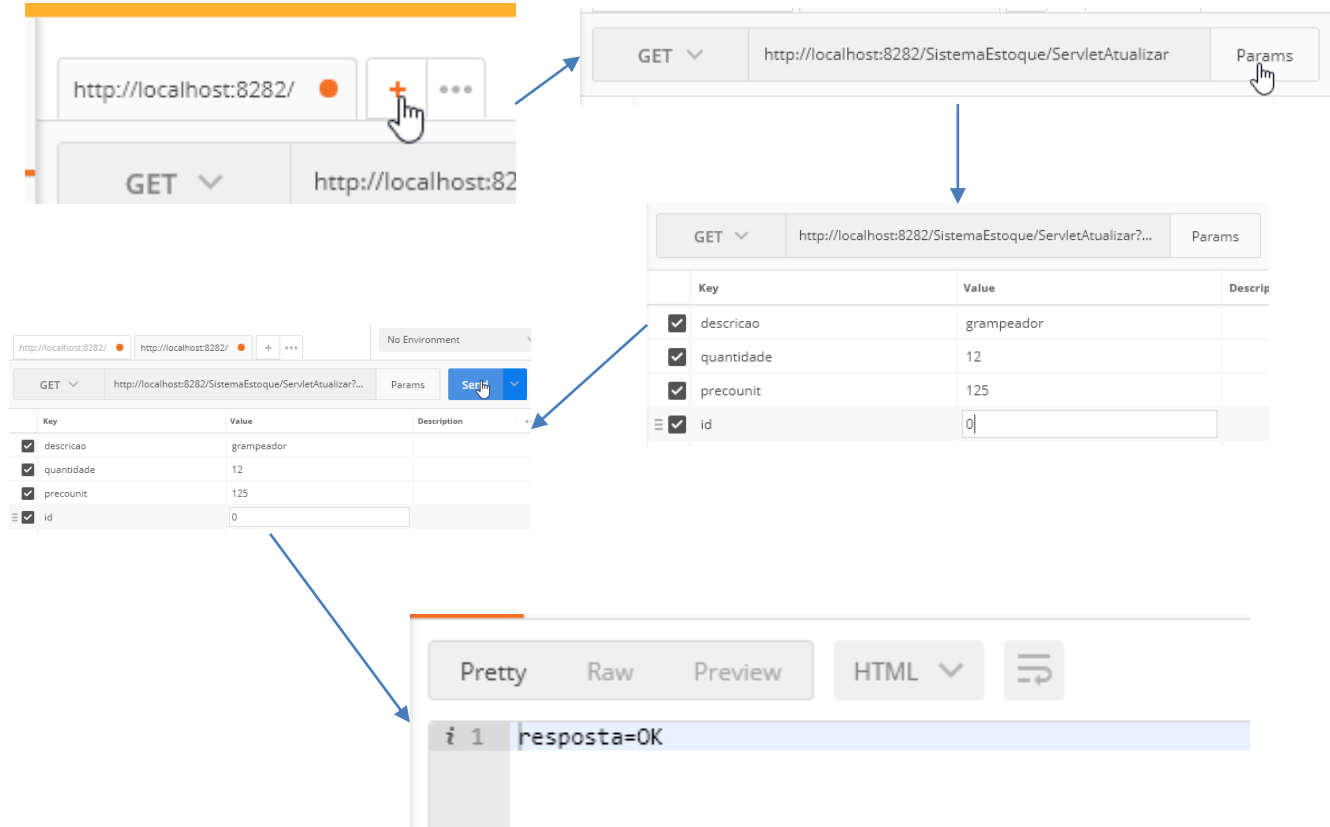
Mude a opção Pretty para Preview e observe o que ocorre:





# MVC, JDBC e Banco de Dados

Abra outra aba e informe a URL indicada:



The image shows a sequence of steps in a web browser to execute an HTTP request. Blue arrows indicate the flow from one step to the next.

**Step 1:** The browser address bar shows `http://localhost:8282/`. A hand icon clicks the '+' button to open a new tab.

**Step 2:** A new tab is opened with the address `http://localhost:8282/SistemaEstoque/ServletAtualizar...`. A hand icon clicks the 'Params' button.

**Step 3:** The 'Params' panel is displayed, showing a table of request parameters:

Key	Value	Description
<input checked="" type="checkbox"/> descricao	grampeador	
<input checked="" type="checkbox"/> quantidade	12	
<input checked="" type="checkbox"/> precounit	125	
<input checked="" type="checkbox"/> id	0	

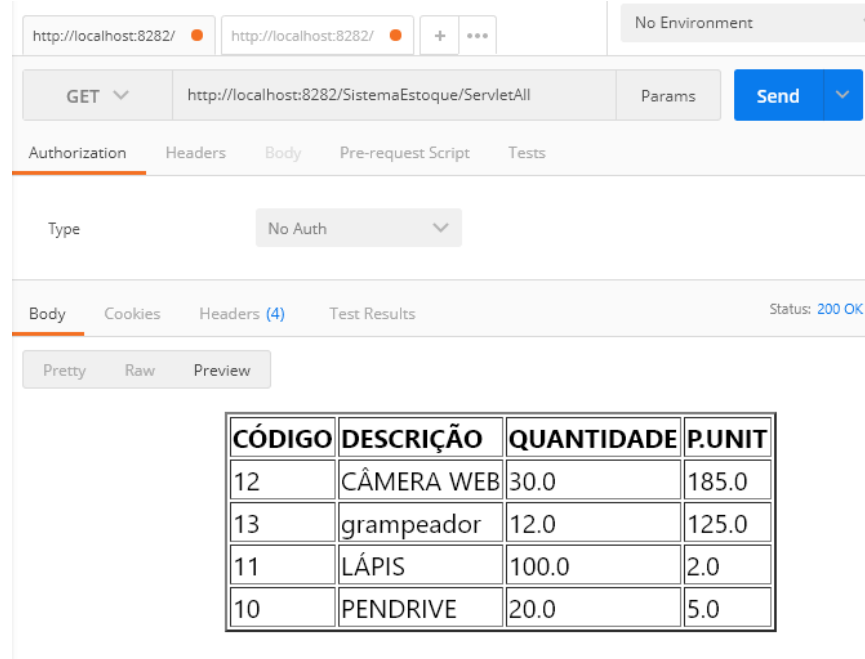
**Step 4:** A hand icon clicks the 'Send' button in the 'Params' panel.

**Step 5:** The browser shows the response in the 'Pretty' view:

```
i 1 resposta=OK
```

# MVC, JDBC e Banco de Dados

Mude a opção Pretty para Preview e observe o que ocorre:



The screenshot shows a web client interface with the following details:

- URL: `http://localhost:8282/`
- Method: `GET`
- Path: `http://localhost:8282/SistemaEstoque/ServletAll`
- Authorization: `No Auth`
- Body: `200 OK`
- Headers: `(4)`
- Test Results: `200 OK`
- Preview tab is selected.

CÓDIGO	DESCRIÇÃO	QUANTIDADE	P.UNIT
12	CÂMERA WEB	30.0	185.0
13	grampeador	12.0	125.0
11	LÁPIS	100.0	2.0
10	PENDRIVE	20.0	5.0



MVC, JDBC e Banco de Dados

FIM