

# Criando Web Services RESTful no Netbeans 8.2

APS 3

# Introdução

- Esta apresentação tem como objetivo realizar passo a passo um tutorial para a construção de um Web Service RESTful em Java que realiza um CRUD de produtos, inicialmente sem persistência.
- Operações do CRUD foram baseadas nos métodos HTTP seguindo:  
<http://www.restapitutorial.com/lessons/httpmethods.html>
- Para realização desse tutorial, foi utilizado o IDE Netbeans 8.2, o JDK 8, o servidor GlassFish 4.

# Etapas

1. Criar o projeto da aplicação Web
2. Criar a entidade Produto
3. Criar a camada service (Lógica de Negócio)
4. Criar/Gerar o Web Service Restful
5. Testar o Web Service

# Criar o projeto contendo a aplicação Web

Etapa 1

# Etapa 1

- Introdução
  - Essa etapa consiste em criar o projeto com a aplicação Web.
  - O nome do projeto é CrudProdutos
  - Como servidor Java Web, vamos trabalhar com o Glassfish Server 4.
  - Nenhum framework deverá ser selecionado.

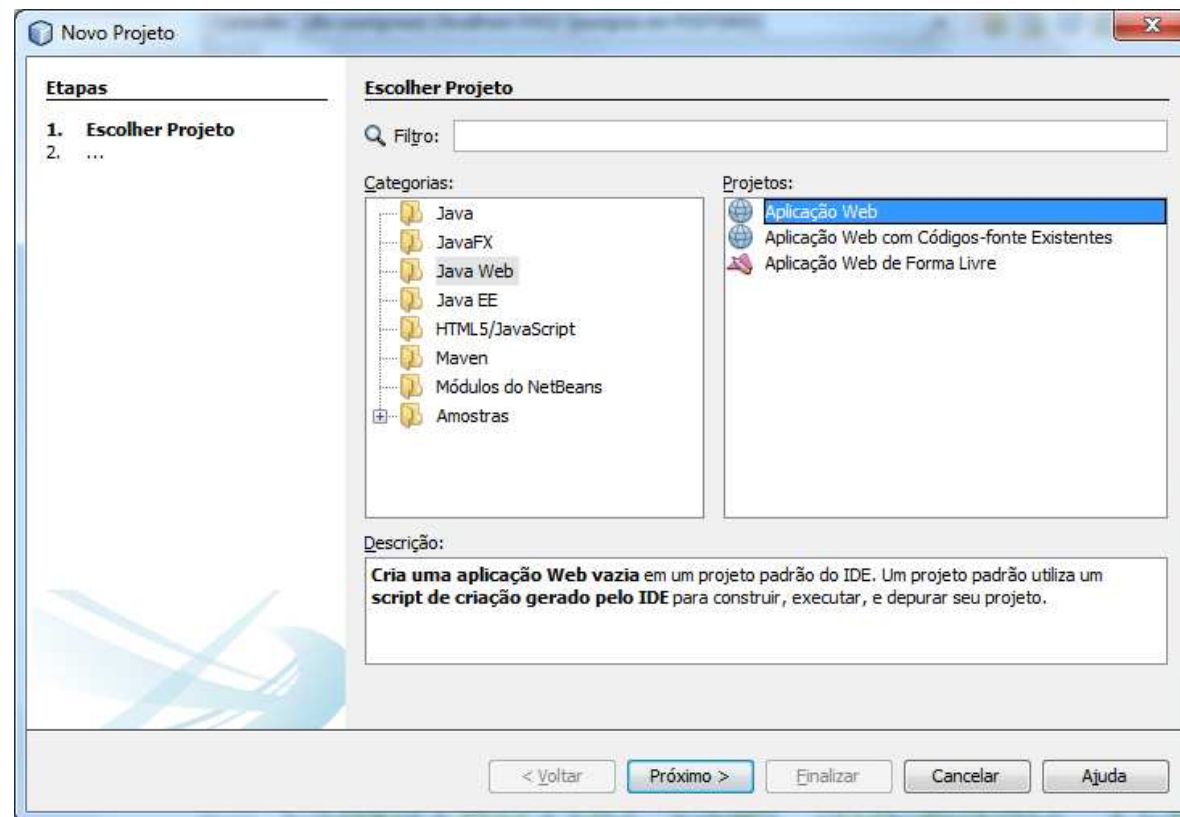
# Etapa 1

- No menu Arquivo do Netbeans, selecione a opção “Novo Projeto”.



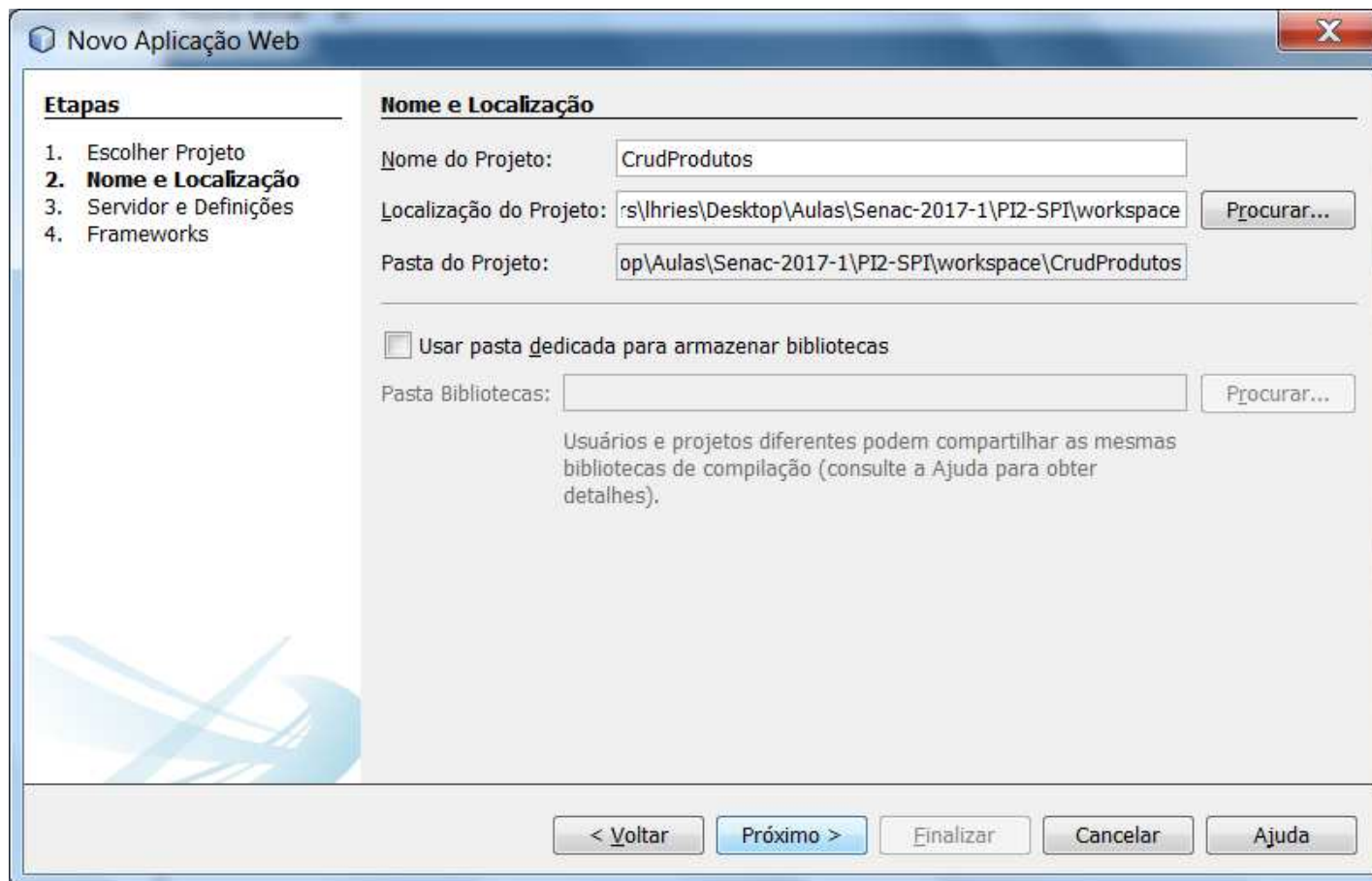
# Etapa 1

- Ao abrir a caixa de diálogo do “Novo Projeto”, selecione a categoria “Java Web” e em Projetos, selecione “Aplicação Web”. Clique no botão “Próximo”



# Etapa 1

- Digite o nome do projeto “CrudProdutos” e pressione o botão “Próximo >”.

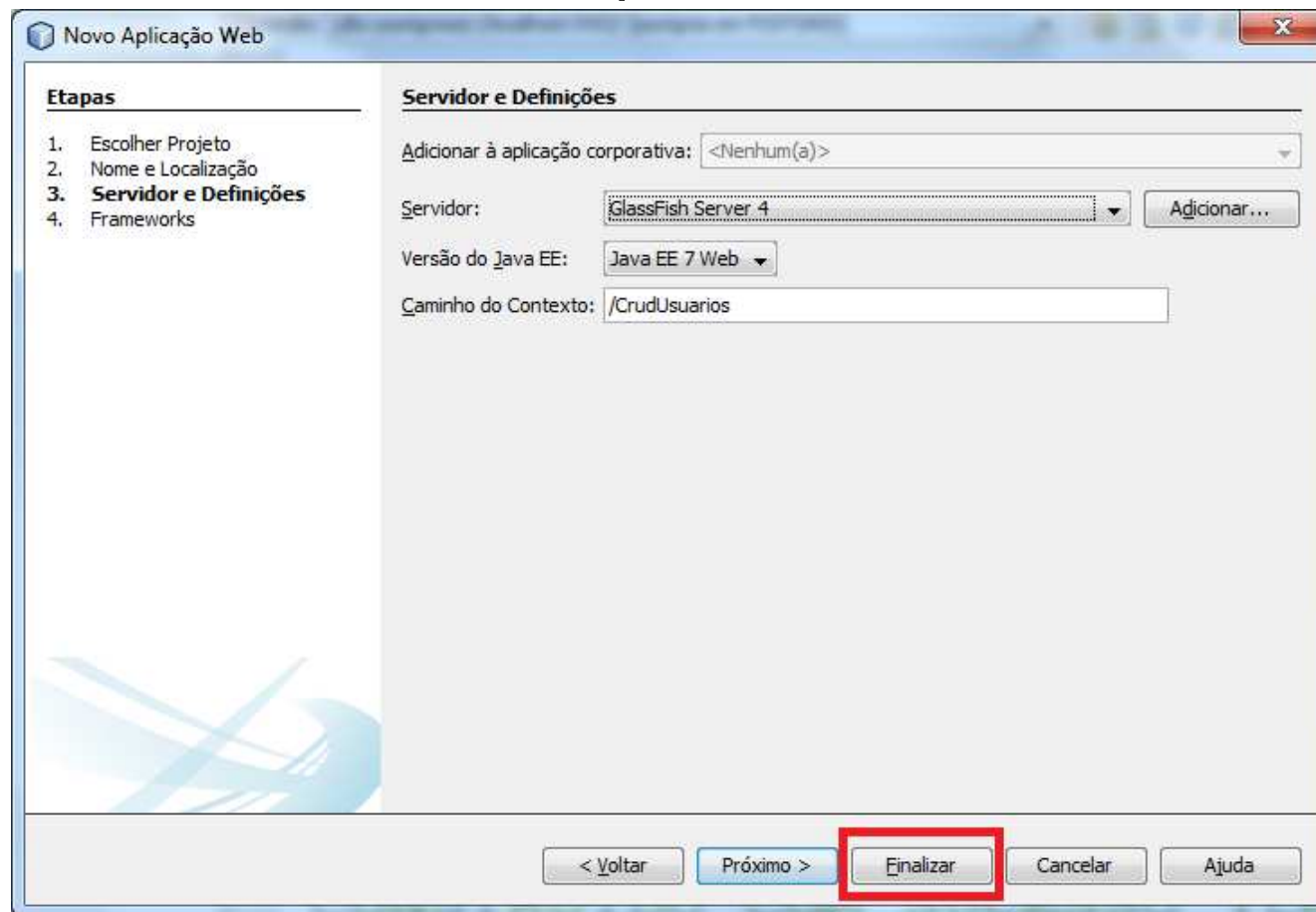


The screenshot shows the 'Novo Aplicação Web' (New Web Application) wizard in Visual Studio. The window has a title bar with the text 'Novo Aplicação Web' and a close button. On the left, there is a 'Etapas' (Steps) pane with a list of four steps: 1. Escolher Projeto, 2. Nome e Localização (which is the current step and is highlighted), 3. Servidor e Definições, and 4. Frameworks. The main area of the wizard is titled 'Nome e Localização'. It contains three text input fields: 'Nome do Projeto:' with the value 'CrudProdutos', 'Localização do Projeto:' with the value 'rs\lhries\Desktop\Aulas\Senac-2017-1\PI2-SPI\workspace', and 'Pasta do Projeto:' with the value 'op\Aulas\Senac-2017-1\PI2-SPI\workspace\CrudProdutos'. To the right of the 'Localização do Projeto' and 'Pasta do Projeto' fields are buttons labeled 'Procurar...'. Below these fields is a checkbox labeled 'Usar pasta dedicada para armazenar bibliotecas' which is currently unchecked. Below the checkbox is a text input field for 'Pasta Bibliotecas:' with a 'Procurar...' button to its right. At the bottom of the wizard, there is a row of five buttons: '< Voltar', 'Próximo >' (which is highlighted in blue), 'Finalizar', 'Cancelar', and 'Ajuda'. A small text note at the bottom of the main area reads: 'Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes)'.



# Etapa 1

- Em “Servidor e Definições”, mantenha o “Glassfish Server 4” como servidor e pressione o botão “Finalizar”



# Criar a entidade Produto

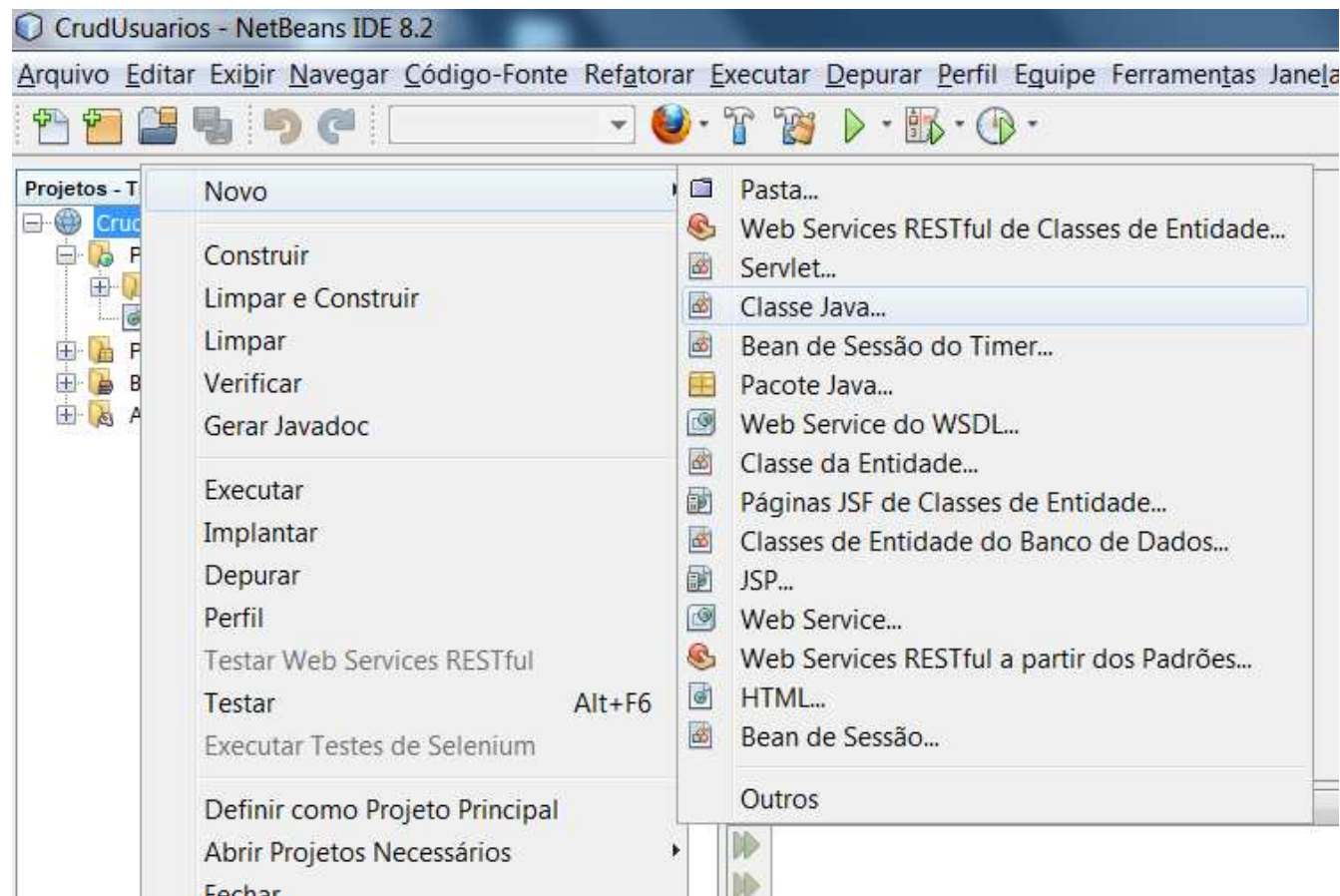
## Etapa 2

# Etapa 2

- Introdução
  - Na criação da entidade iremos criar uma classe Java para salvar os dados de produto.
  - Essa classe ficará no pacote model e terá como atributos: código, nome e preço.
  - Ela terá uma anotação “XMLRootElement” que serve para indicar que a entidade pode ser representado como um elemento raiz XML (ou JSON).

# Etapa 2

- Iniciamos a criação da Entidade selecionando o projeto e com botão direito “Novo > Classe Java...”.



# Etapa 2

- Indique o nome da classe (“Produto”) e o pacote em que será salvo (“model”). Caso não exista, o pacote será criado. Clique no botão “Finalizar”.

**New Classe Java**

**Etapas**

- Escolher Tipo de Arquivo
- Nome e Localização**

**Nome e Localização**

Nome da Classe:

Projeto:

Localização:

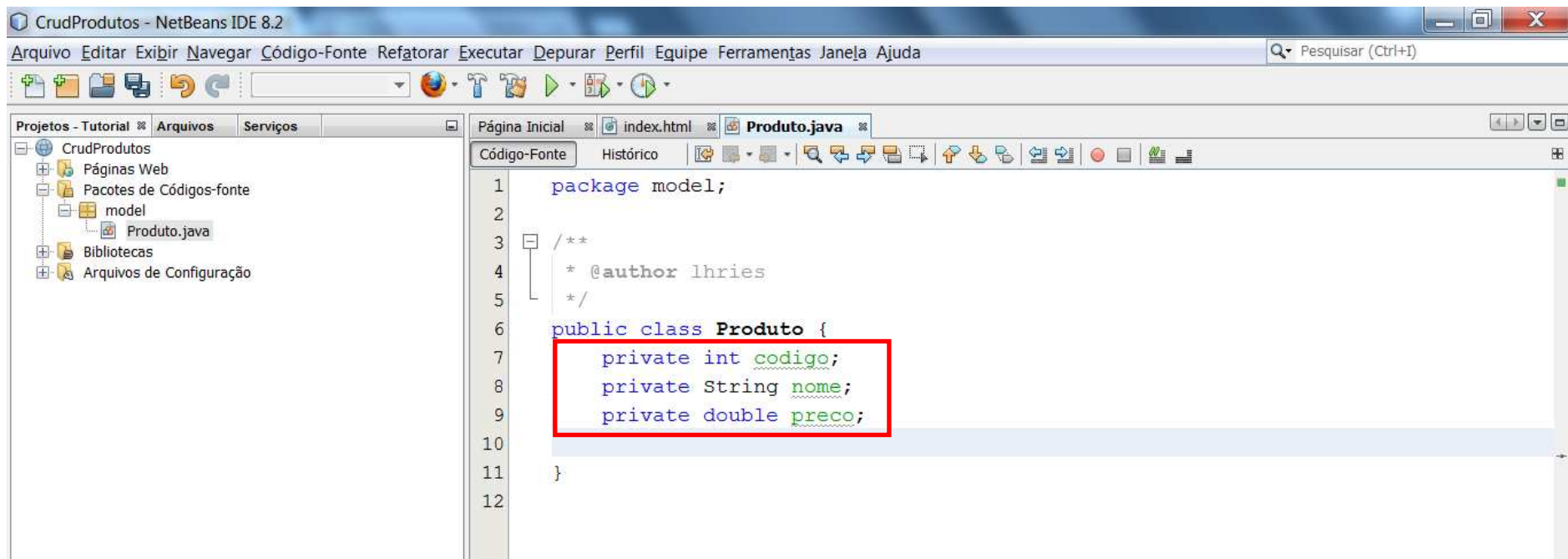
Pacote:

Arquivo Criado: s:\Senac-2017-1\PI2-SPI\workspace\CrudProdutos\src\java\model\Produto.java

< Voltar   Próximo >   **Finalizar**   Cancelar   Ajuda

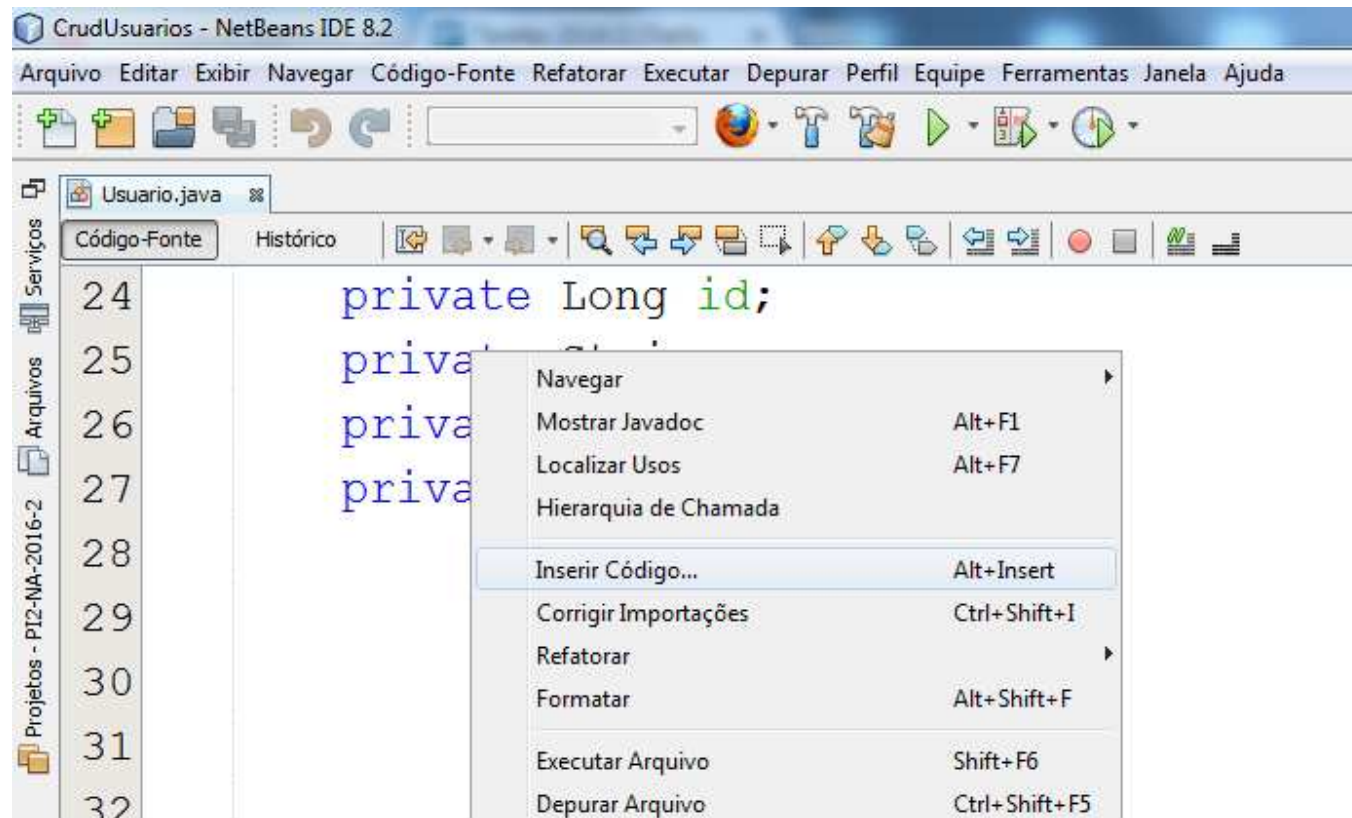
# Etapa 2

- Acrescente os atributos “codigo”, “nome” e “preco”.



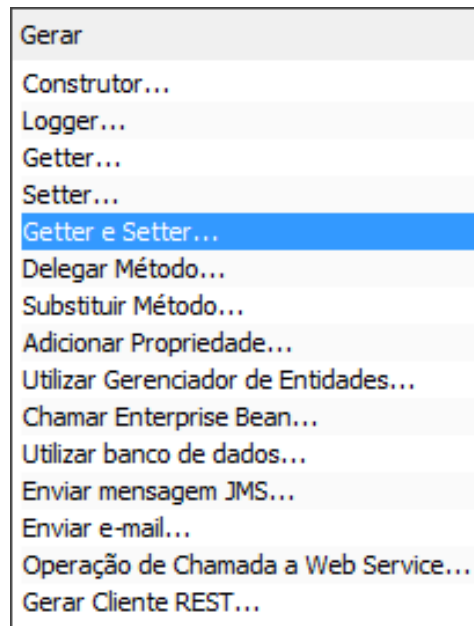
# Etapa 2

- Na linha abaixo dos atributos, clique com o botão direito do mouse e selecione “Inserir Código”.



# Etapa 2

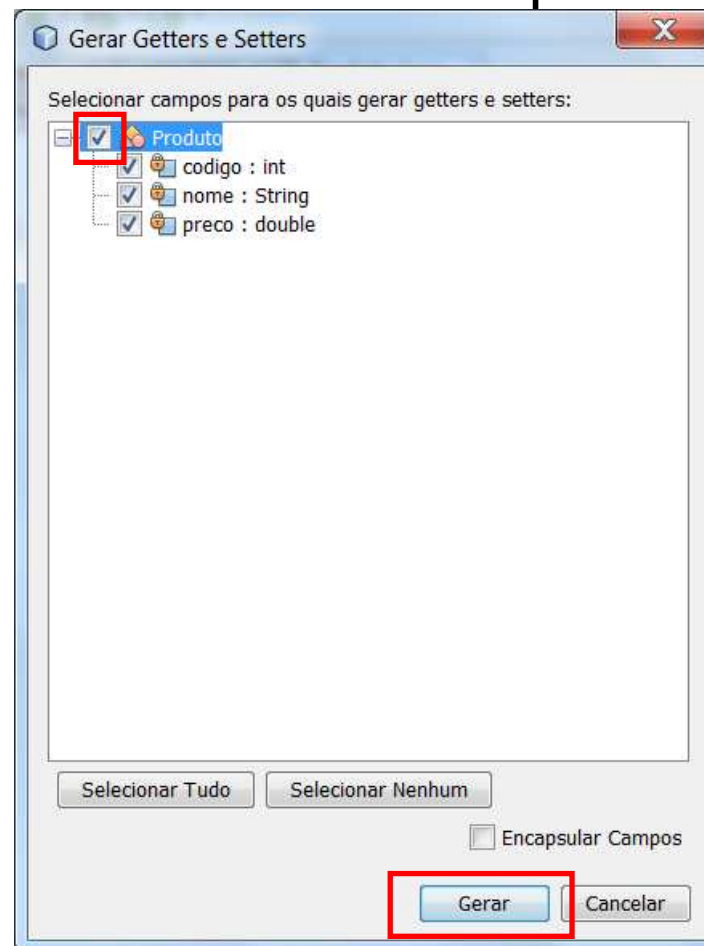
- Selecione “Getter e Setter”. A ideia é criarmos os métodos get e set para os atributos.





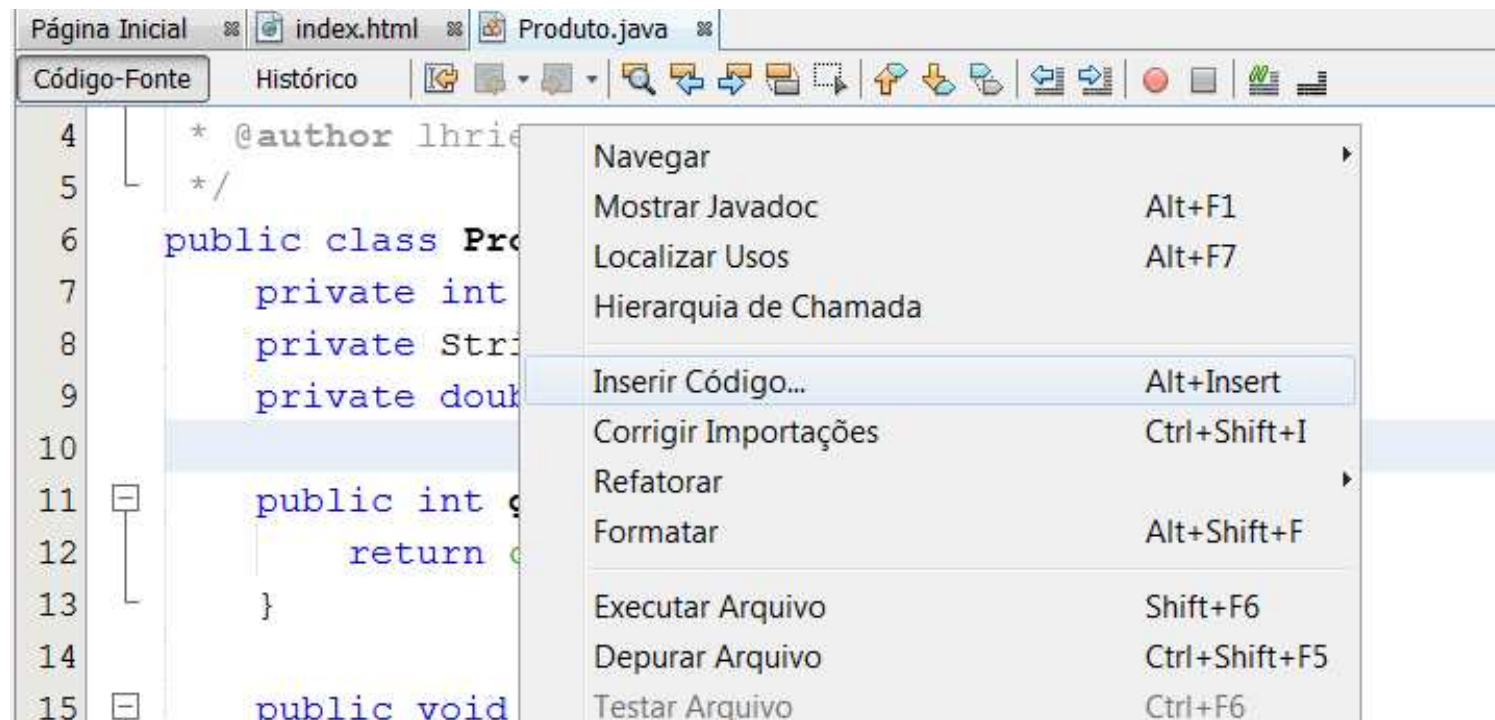
# Etapa 2

- Selecione a marcação indicada e ele marcará todos atributos automaticamente. Clique no botão “Gerar”



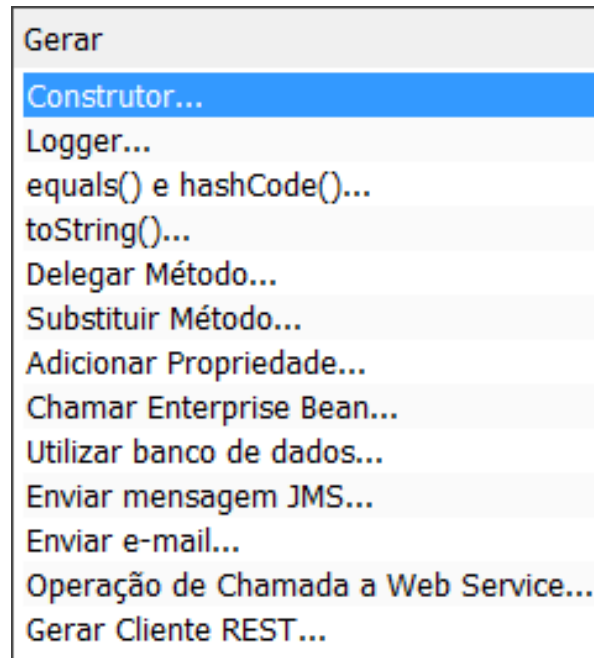
# Etapa 2

- Novamente, na linha abaixo dos atributos, clique com o botão direito do mouse e selecione “Inserir Código”.



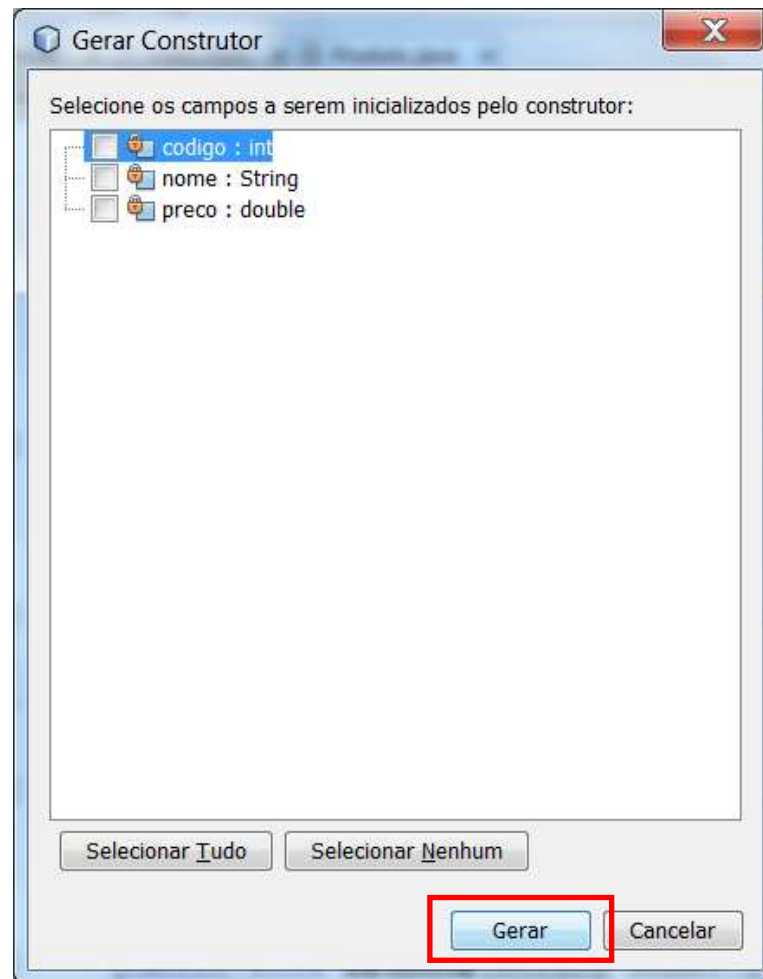
# Etapa 2

- Desta vez, selecione “Construtor...”. A ideia é criarmos um construtor vazio - POJO.



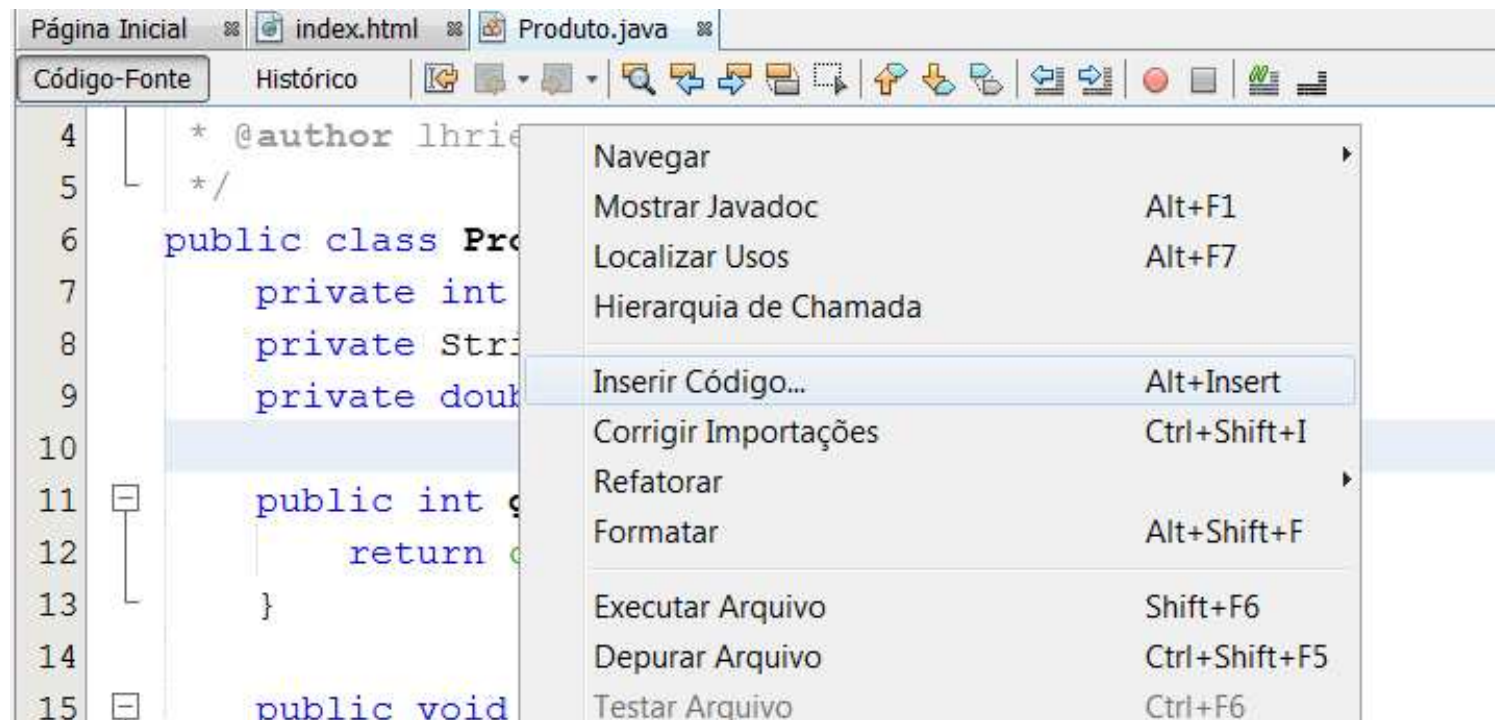
# Etapa 2

- Desta vez, não selecione nada. Apenas clique no botão “Gerar”



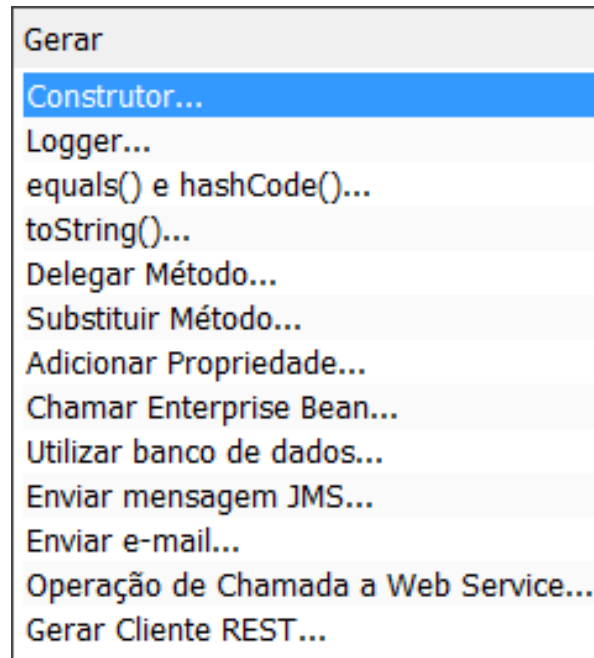
# Etapa 2

- Por fim, repita os passos novamente, clicando com o botão direito do mouse e selecione “Inserir Código”.



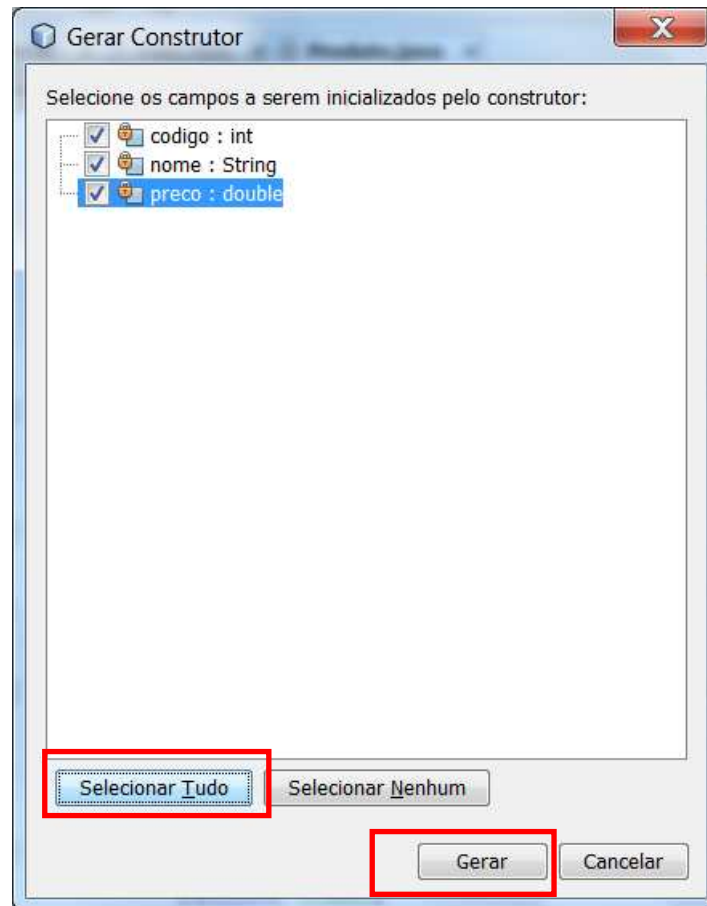
# Etapa 2

- Selecione “Construtor...” novamente. A ideia é criarmos um construtor com todos atributos para popularmos nossa lista de produtos.



# Etapa 2

- Desta vez, pressione o botão “Selecionar Tudo” para marcar todos atributos e pressione o botão “Gerar”



# Etapa 2

- Acrescente depois do nome da classe “implements Serializable”. Quando indicar um erro, pressione “CTRL+SHIFT+i” para fazer o import da biblioteca “java.io.Serializable”

```
import java.io.Serializable;
```

```
public class Produto implements Serializable {
```



# Etapa 2

- Em seguida, adicione uma anotação “XMLRootElement” na classe (na linha acima dela). Essa anotação serve para indicar que a entidade pode ser representado como um elemento raiz XML (ou JSON).

```
@XMLRoot Complemente com “CTRL+espaço”  
@XmlRootElement (javax.xml.bind.annotation.  
private int codigo;  
private String nome;  
private double preco;  
  
public Produto(int codigo, String nome, double preco) {  
    this.codigo = codigo;  
    this.nome = nome;  
    this.preco = preco;  
}  
  
public Produto() {  
}
```

# Etapa 2

- Complementando com o “CTRL+espaço”, ele importa a biblioteca automaticamente. Salve a classe.

```
1 package model;
2
3 import java.io.Serializable;
4 import javax.xml.bind.annotation.XmlRootElement;
5
6 @XmlRootElement
7 public class Produto implements Serializable {
8     private int codigo;
9     private String nome;
10    private double preco;
11
12    public Produto(int codigo, String nome, double preco) {
13        this.codigo = codigo;
14        this.nome = nome;
15        this.preco = preco;
16    }
```

# Criar a camada service: UsuarioService

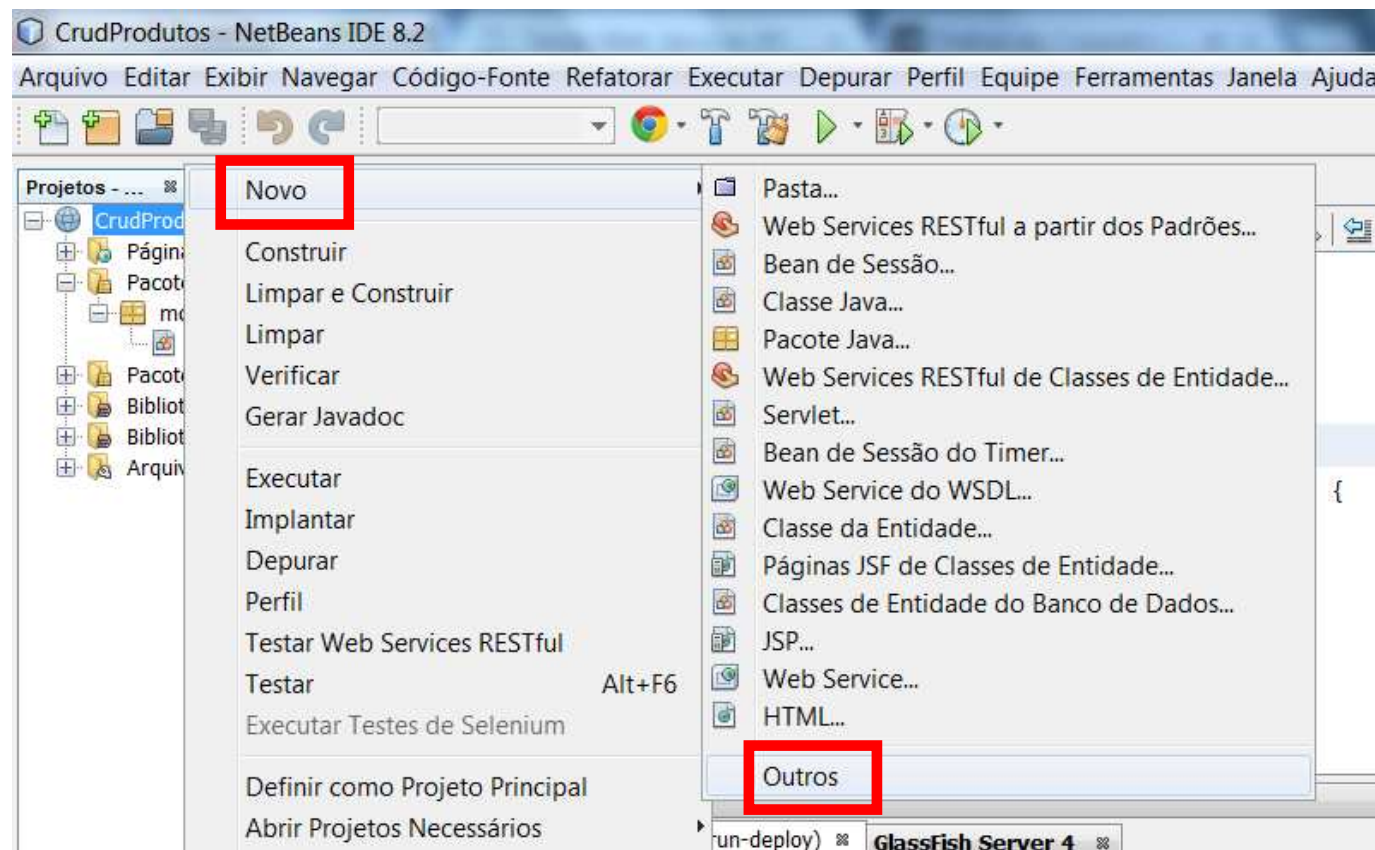
Etapa 3

# Etapa 3

- Introdução
  - Nessa etapa, será criada a camada de negócio (Service) chamada UsuarioService, que manipulará uma lista de usuários através de métodos que realizarão o CRUD de usuários.
  - Os métodos são: inserir, listar, atualizar, excluir e buscarPorCodigo.
  - Para mantermos a lista por um tempo de vida, precisamos trabalhar com um objeto do servidor. Para isso, criaremos o UsuarioService como um componente Java, relacionando-o como um bean de sessão (EJB).

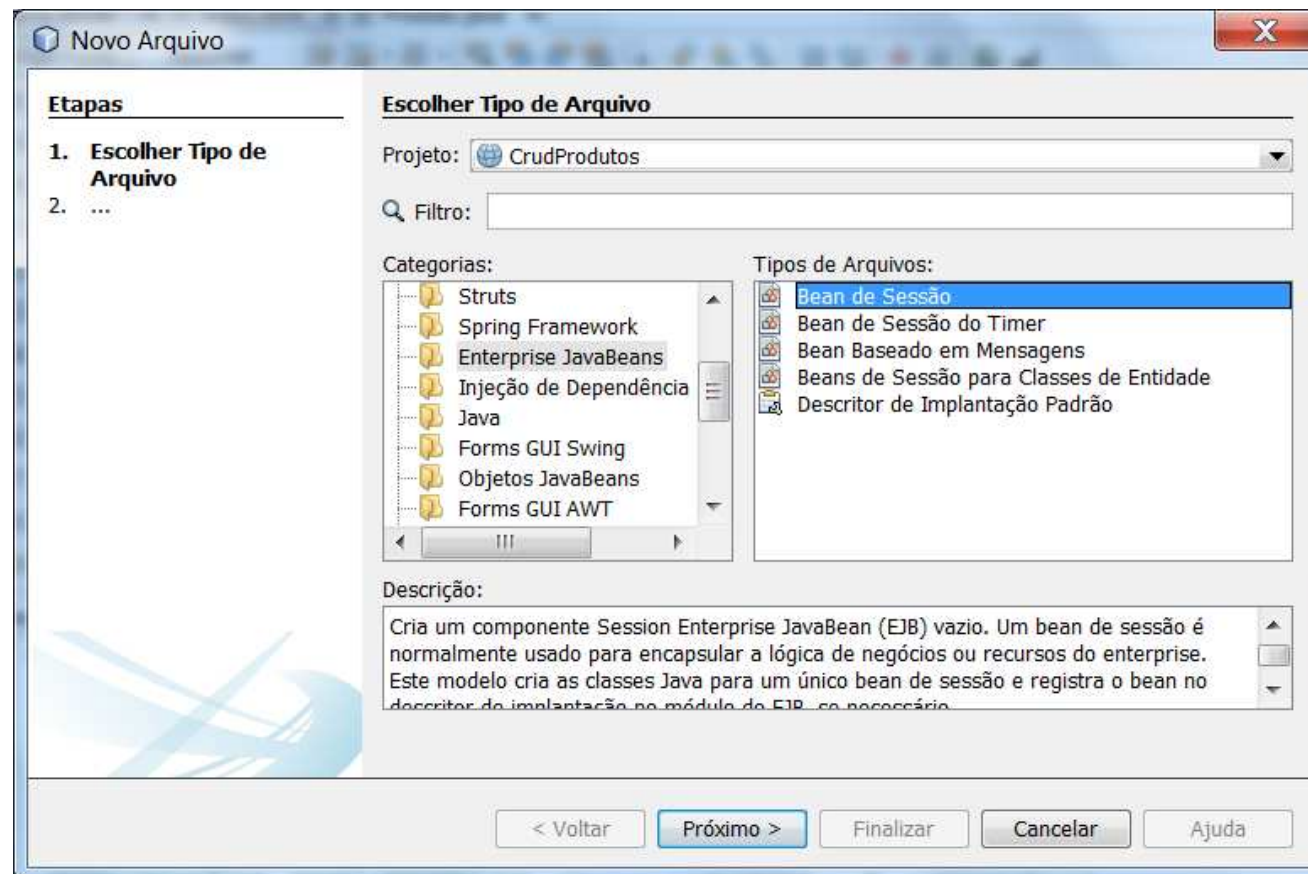
# Etapa 3

- Inicie essa etapa criando o bean de sessão, clicando com o botão direito no projeto, selecionando Novo e Outros.



# Etapa 3

- Selecione a categoria “Enterprise JavaBeans” e o tipo de arquivo “Bean de Sessão”. Clique no botão “Próximo”



# Etapa 3

- Escreva “ProdutoService” para ser o nome EJB (nome do classe) e relacione “service” como pacote. Mantenha o tipo de sessão para “Sem estado (Stateless)” e clique em Finalizar.

**New Bean de Sessão**

**Etapas**

1. Escolher Tipo de Arquivo
2. **Nome e Localização**

**Nome e Localização**

Nome EJB:

Projeto:

Localização:

Pacote:

**Tipo de sessão:**

☒ **Sem estado (Stateless)**

☐ Com estado (Stateful)

☐ Único

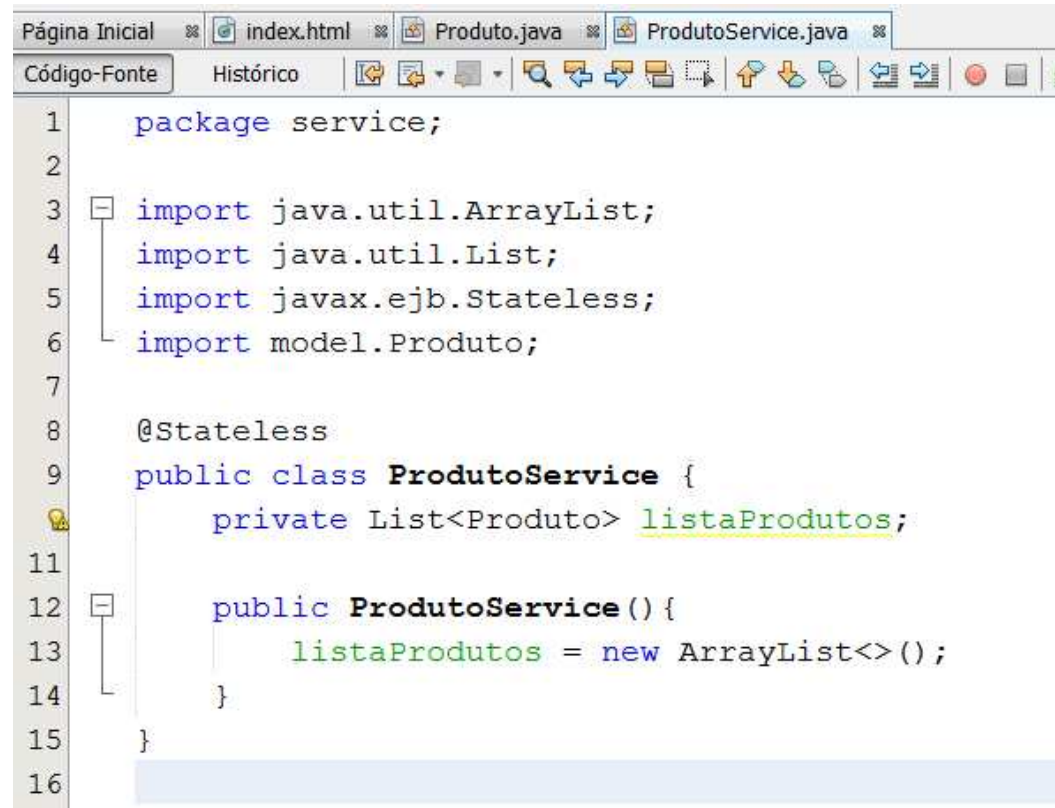
**Criar Interface:**

☐ Local

< Voltar   Próximo >   **Finalizar**   Cancelar   Ajuda

# Etapa 3

- A nova classe ProdutoService será criada junto com a anotação “@Stateless”. Essa anotação indica que essa classe será um componente gerenciado pelo servidor. Adicione a lista de produtos nessa classe (e faça os imports necessários) conforme o código:



The screenshot shows an IDE window with the file 'ProdutoService.java' open. The code is as follows:

```
1 package service;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import javax.ejb.Stateless;
6 import model.Produto;
7
8 @Stateless
9 public class ProdutoService {
10     private List<Produto> listaProdutos;
11
12     public ProdutoService() {
13         listaProdutos = new ArrayList<>();
14     }
15 }
16
```



# Etapa 3

- Adicione os métodos do CRUD para a lista – Parte 1:

```
16 public void inserir(Produto prod) {
17     listaProdutos.add(prod);
18 }
19
20 public List<Produto> listar(){
21     return listaProdutos;
22 }
23
24 public Produto buscarPorCodigo(int cod){
25     for(Produto p: listaProdutos){
26         if(p.getCodigo() == cod)
27             //Clonando objeto -> só para leitura
28             return new Produto(p.getCodigo(),
29                                 p.getNome(),p.getPreco());
30     }
31     return null;
32 }
```

A ideia é  
fazer a  
leitura  
como no  
BD.

# Etapa 3

- Adicione os métodos do CRUD para a lista – Parte 2:

```
34  private int getIndice(int codigo) {  
35      for(int i=0;i<listaProdutos.size();i++)  
36          if(listaProdutos.get(i).getCodigo() == codigo)  
37              return i;  
38      return -1;  
39  }  
40  
41  public void atualizar(Produto prod) {  
42      listaProdutos.set(  
43          this.getIndice(prod.getCodigo()),  
44          prod);  
45  }  
46  
47  public void excluir(Produto prod) {  
48      listaProdutos.remove(  
49          this.getIndice(prod.getCodigo()));  
50  }
```

# Criar/Gerar o Web Service Restful

Etapa 4

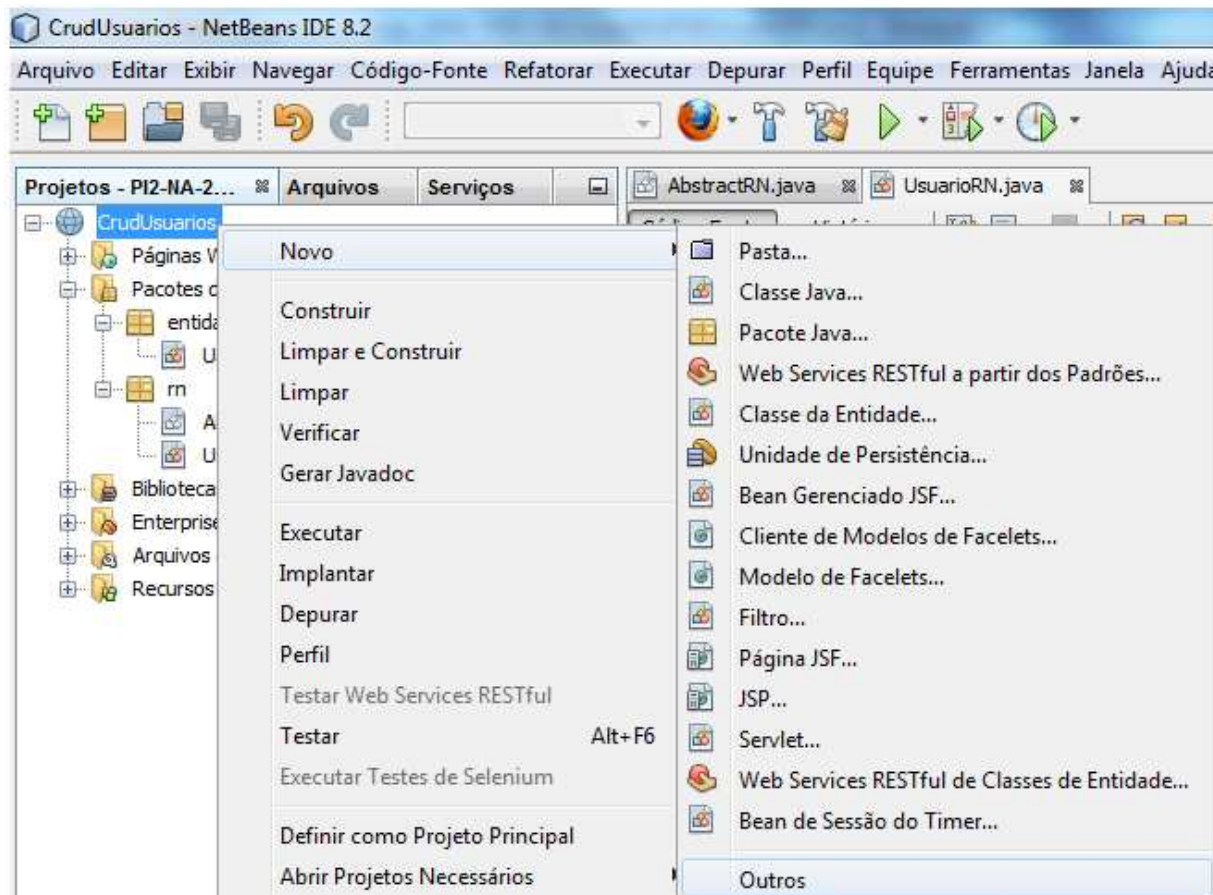
# Etapa 4

- Introdução

- Nessa etapa, o Web Service RESTful é criado e gerado a partir da API Jersey.
- O nome da classe é “ProdutoWS” e o nome do recurso é “produtos”.
- Dois arquivos serão criados:
  - ApplicationConfig: voltado para configurar todos os Web Services RESTful do projeto.
  - ProdutoWS: voltado para definir o Web Service que realiza o CRUD de Produtos.

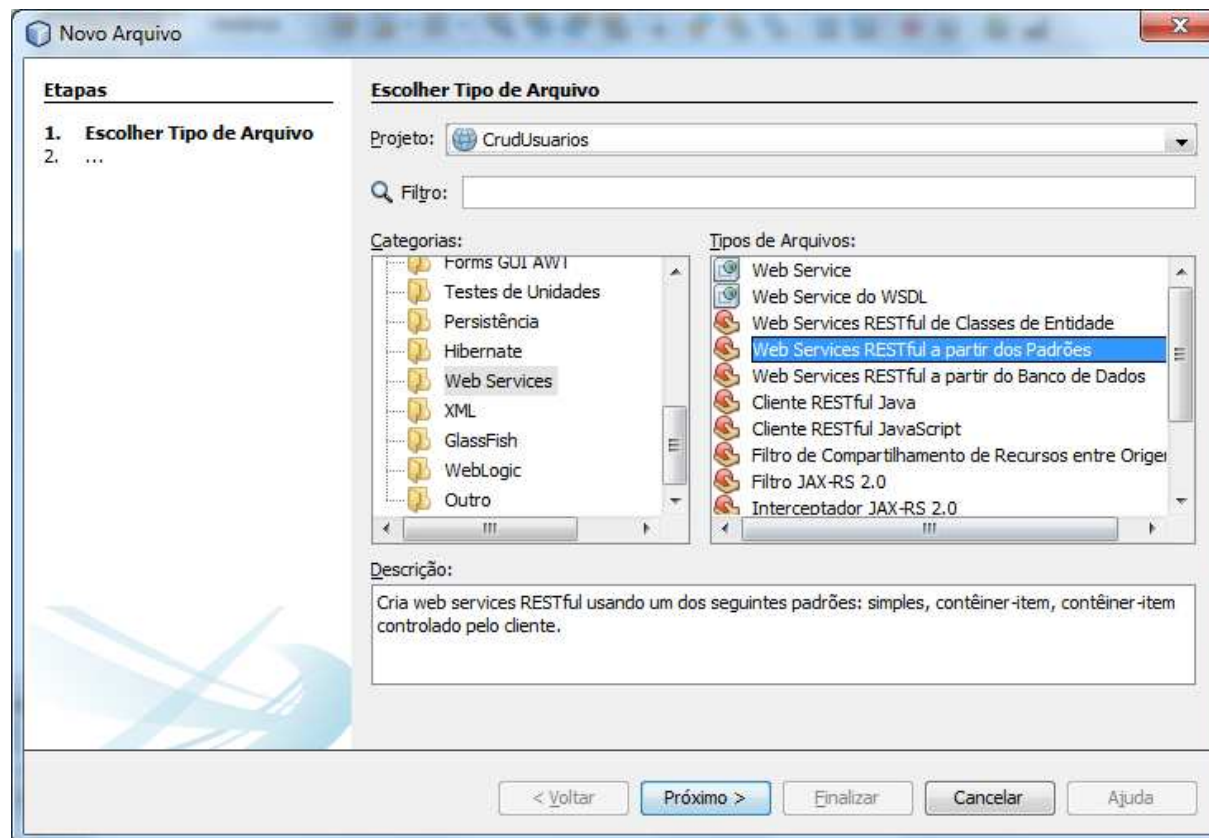
# Etapa 4

- Iniciamos a criação do Web Service selecionando o projeto e com botão direito “Novo > Outros”.



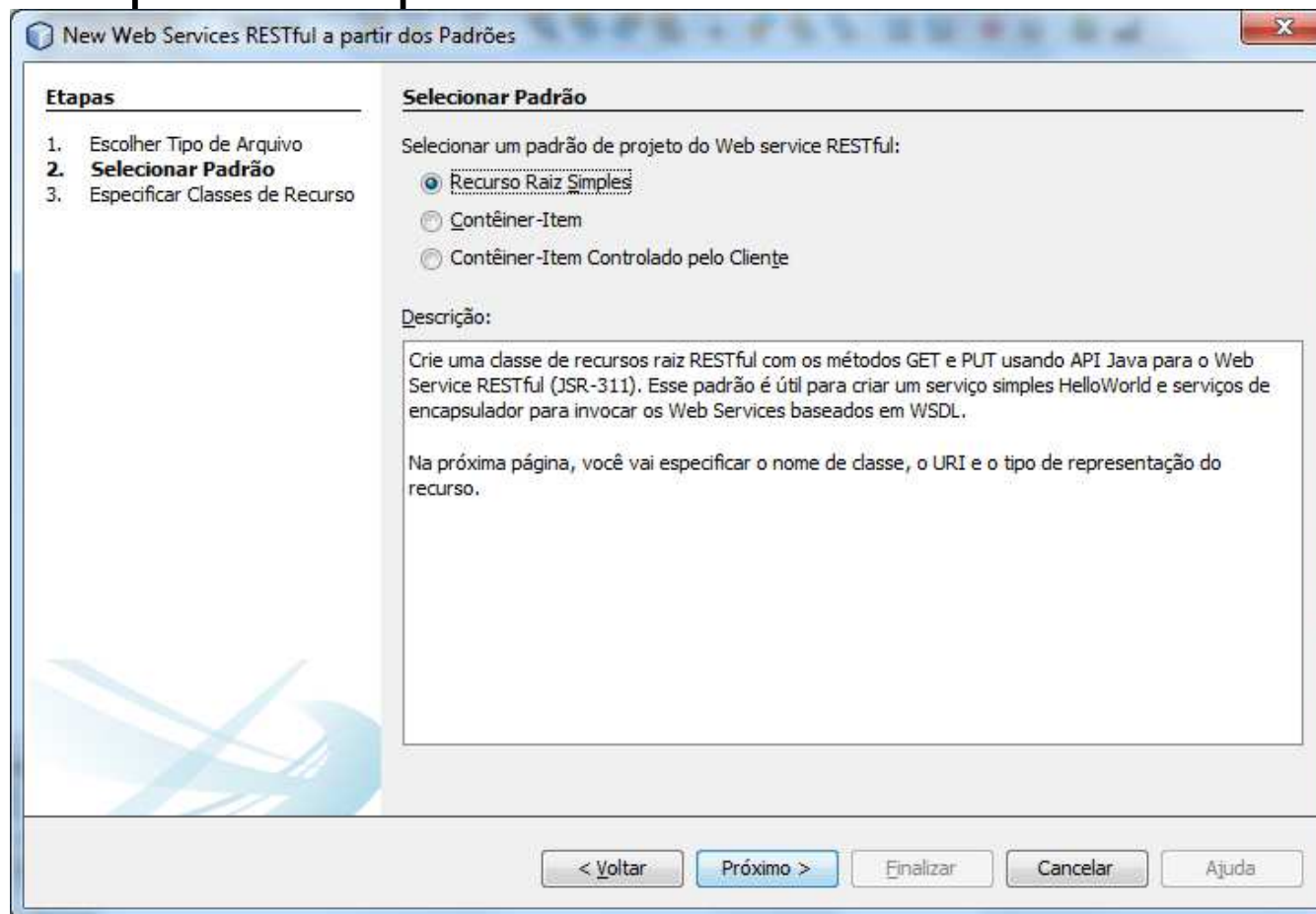
# Etapa 4

- Em “Categorias” selecione “Web Services” e em “Tipos de Arquivos” selecione “Web Services RESTful a partir dos Padrões”. Clique no botão “Próximo >”.



# Etapa 4

- Na opção “Selecionar o Padrão”, mantenha o recurso raiz simples e clique em “Próximo >”



# Etapa 4

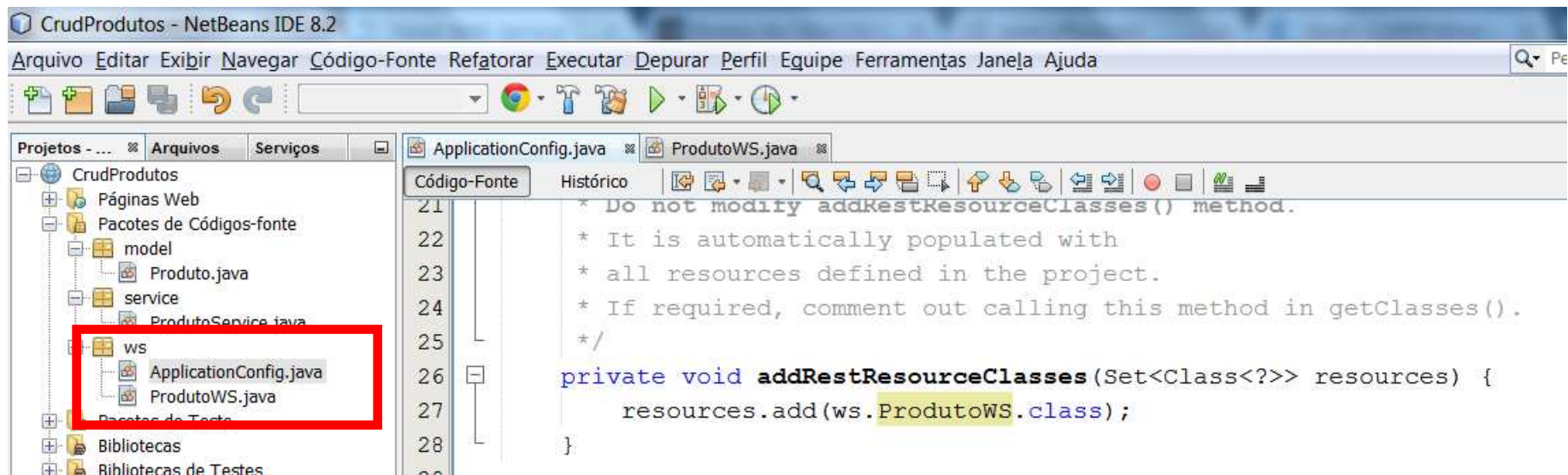
- Na especificação do recurso, nomeie o pacote para “ws”, o nome da classe para “ProdutoWS” e o caminho (“nome do recurso”) para “produtos”. Altere o Tipo MIME para “application/json” e Classe de Representação para “model.Produto”. Clique em “Finalizar”.

The screenshot shows a dialog box titled "New Web Services RESTful a partir dos Padrões" with a close button (X) in the top right corner. The dialog is divided into two main sections. On the left, under the heading "Etapas", there is a list of three steps: 1. Escolher Tipo de Arquivo, 2. Selecionar Padrão, and 3. Especificar Classes de Recurso, with the third step being the active one. On the right, under the heading "Especificar Classes de Recurso", there are several input fields and dropdown menus: "Projeto:" with the text "CrudProdutos"; "Localização:" with a dropdown menu showing "Pacotes de Códigos-fonte"; "Pacote do Recurso\:" with a dropdown menu showing "ws"; "Caminho\:" with the text "produtos"; "Nome da Classe:" with the text "ProdutoWS"; "Tipo MIME:" with a dropdown menu showing "application/json"; and "Classe de Representação:" with the text "model.Produto" and a "Selecionar..." button next to it. At the bottom of the dialog, there are five buttons: "< Voltar", "Próximo >", "Finalizar" (which is highlighted with a blue border), "Cancelar", and "Ajuda".



# Etapa 4

- Note que duas classes foram criadas dentro do pacote “ws”:
  - ApplicationConfig: classe voltada para configurar o Web Service. O método abaixo relaciona as classes que serão Web Services.
  - “ProdutoWS”: classe que definiremos o Web Service que realizará o CRUD de produtos. No próximo slide explicaremos mais o funcionamento desse Web Service.



# Etapa 4

- Inicie a alteração do código da classe “ProdutoWS”, relacionando o componente (EJB) no código, logo abaixo da classe – não esqueça os imports (Ctrl+Shift+i)

```
import javax.ejb.EJB;  
import model.Produto;  
import service.ProdutoService;
```

**Imports necessários**

```
@Path("produtos")  
public class ProdutoWS {
```

```
    @EJB  
    private ProdutoService produtoService;
```

**Objeto EJB**

```
    @Context  
    private UriInfo context;
```

# Etapa 4

- Explicação do código anterior:
  - A classe ProdutoWS vem com a anotação “*@Path*” para indicar o caminho para acessar o recurso: “produtos”.
  - Utilizamos o ProdutoService através da anotação EJB para tratar as regras de negócio (não relacionado) e a manipulação da lista.
  - Esse objeto é instanciado automaticamente pelo servidor e ele mantém o objeto para a manipulação do Web Service, ou seja, o servidor vai manter o estado da lista de produtos.
  - Assim, poderemos realizar as operações com o Web Service mantendo os dados atualizados na lista (através do ProdutoService).


# Etapa 4

- Remova os métodos abaixo do construtor e acrescente os seguintes métodos (não esqueça dos imports) – Parte 1:

```
45 public ProdutoWS() {  
46     }  
47  
48     @GET  
49     @Produces(MediaType.APPLICATION_JSON)  
50     public List<Produto> getProdutos() {  
51         return produtoService.listar();  
52     }  
53  
54     @GET  
55     @Path("/{codigo}")  
56     @Produces(MediaType.APPLICATION_JSON)  
57     public Produto getProduto(@PathParam("codigo") int cod) {  
58         return produtoService.buscarPorCodigo(cod);  
59     }
```

# Etapa 4

- Remova os métodos abaixo do construtor e acrescente os seguintes métodos (não esqueça dos imports) – Parte 2:

```
61      @POST
62      @Consumes(MediaType.APPLICATION_JSON)
63      public void adicionarProduto(Produto p,
64            @Context final HttpServletResponse response) {
65
66          produtoService.inserir(p);
67          //Alterar o codigo para 201 (Created)
68          response.setStatus(HttpServletResponse.SC_CREATED);
69          try {
70              response.flushBuffer();
71          } catch (IOException e) {
72              //Erro 500
73              throw new InternalServerErrorException();
74          }
75      }
```

# Etapa 4

- Remova os métodos abaixo do construtor e acrescente os seguintes métodos (não esqueça dos imports) – Parte 3:

```
77     @PUT
78     @Path("/{codigo}")
79     @Consumes(MediaType.APPLICATION_JSON)
80     public void alterarProduto(@PathParam("codigo") int cod,
81                               Produto produto) {
82
83         Produto p = produtoService.buscarPorCodigo(cod);
84         p.setNome(produto.getNome());
85         p.setPreco(produto.getPreco());
86         produtoService.atualizar(p);
87     }
88
89     @DELETE
90     @Path("/{codigo}")
91     @Produces(MediaType.APPLICATION_JSON)
92     public Produto removerProduto(@PathParam("codigo") int cod) {
93         Produto p = produtoService.buscarPorCodigo(cod);
94         produtoService.excluir(p);
95         return p;
96     }
```

# Etapa 4

- Explicação do código anterior:
  - Cada operação do CRUD é acessada a partir dos métodos HTTP a partir das anotações: *@POST*, *@GET*, *@PUT*, *@DELETE*
    - Métodos foi baseado em <http://www.restapitutorial.com/lessons/httpmethods.html>
  - As anotações *@Path* especificam um novo caminho para o recurso. Exemplo: “produtos/10” referencia o usuário com id=10.
  - Os ids são recebidos nos métodos como parâmetros a partir da anotação *@PathParam*(“id”).
  - As anotações *@Consumes* e *@Produces* especificam o *MediaType* (no caso, XML) que será trabalhado no conteúdo Request ou Response respectivamente.

# Etapa 4

- Continuando:
  - Os dados de atualização do usuário (adicionar e atualizar) possuem o parâmetro Produto e o conteúdo do HTTP Request deverá conter o conteúdo de Produto em JSON.
  - Os dados de retorno de usuário (getProdutos e getProduto) possuem o retorno Produto e o conteúdo do HTTP Response deverá retornar o conteúdo do produto (ou lista) em JSON.
  - Os dados são serializados/desserializados automaticamente convertendo JSON em



# Testar o Web Service

Etapa 5

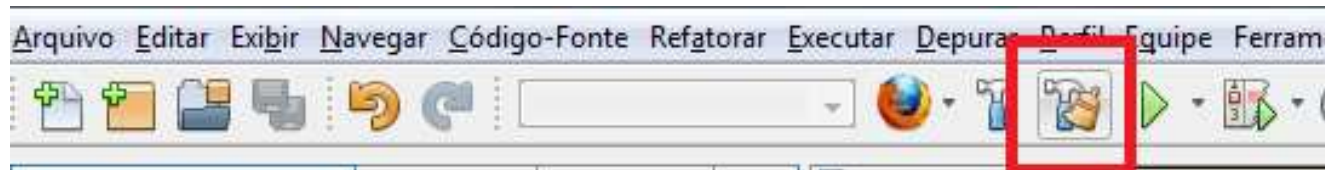
# Etapa 5

- Introdução

- Na etapa final, iremos compilar e implantar o projeto e testar o Web Service RESTful.
- Para testar o Web Service, utilizaremos a própria IDE que disponibiliza um cliente de teste do servidor (Glassfish).
- Cada um dos testes a serem realizados será acessando o recurso criado – através dos métodos HTTP de acordo com a operação do CRUD que desejarmos.

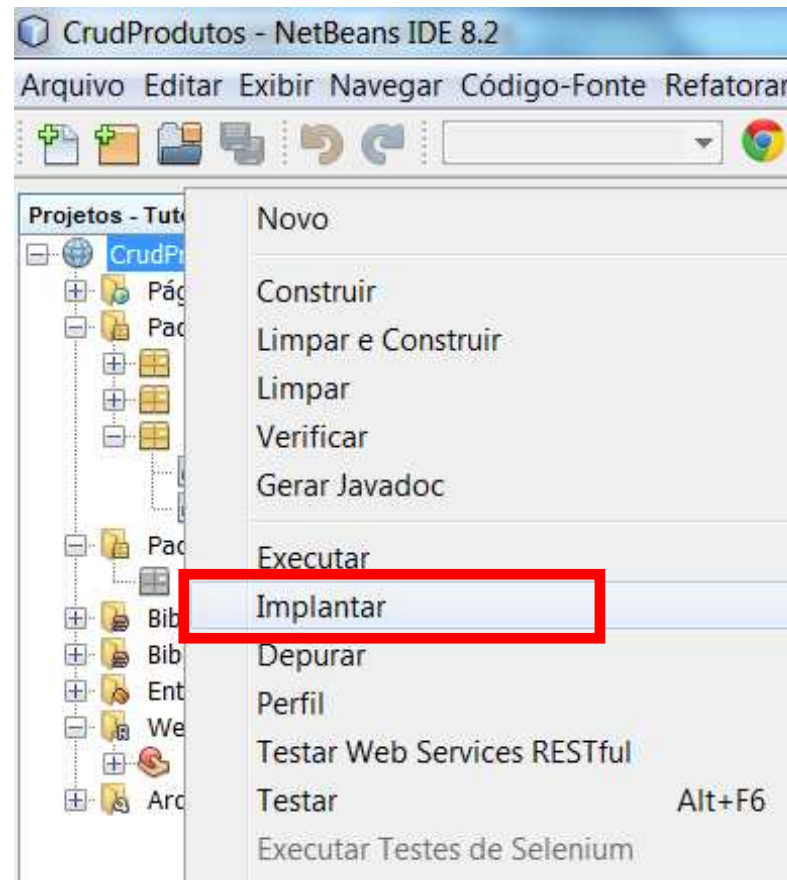
# Etapa 5

- Inicie a etapa 5 limpando e construindo o projeto. Selecione o projeto e clique no botão indicado abaixo.



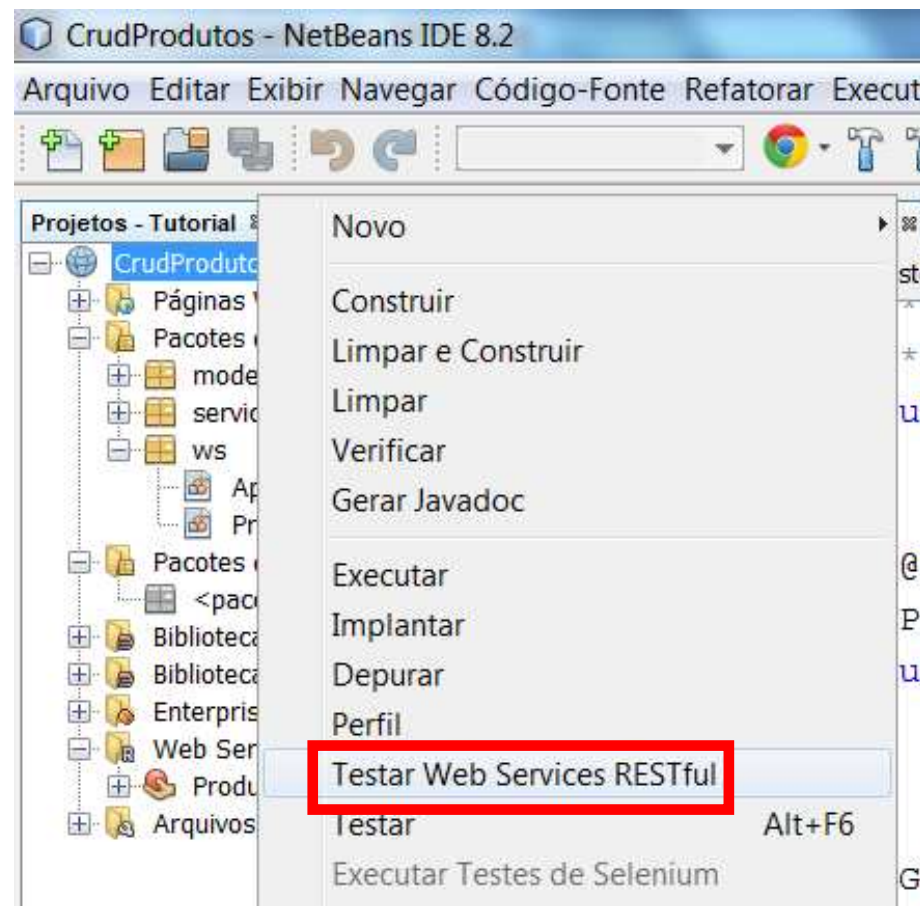
# Etapa 5

- Agora, selecione o projeto com o botão direito do mouse, e selecione a opção “Implantar”.



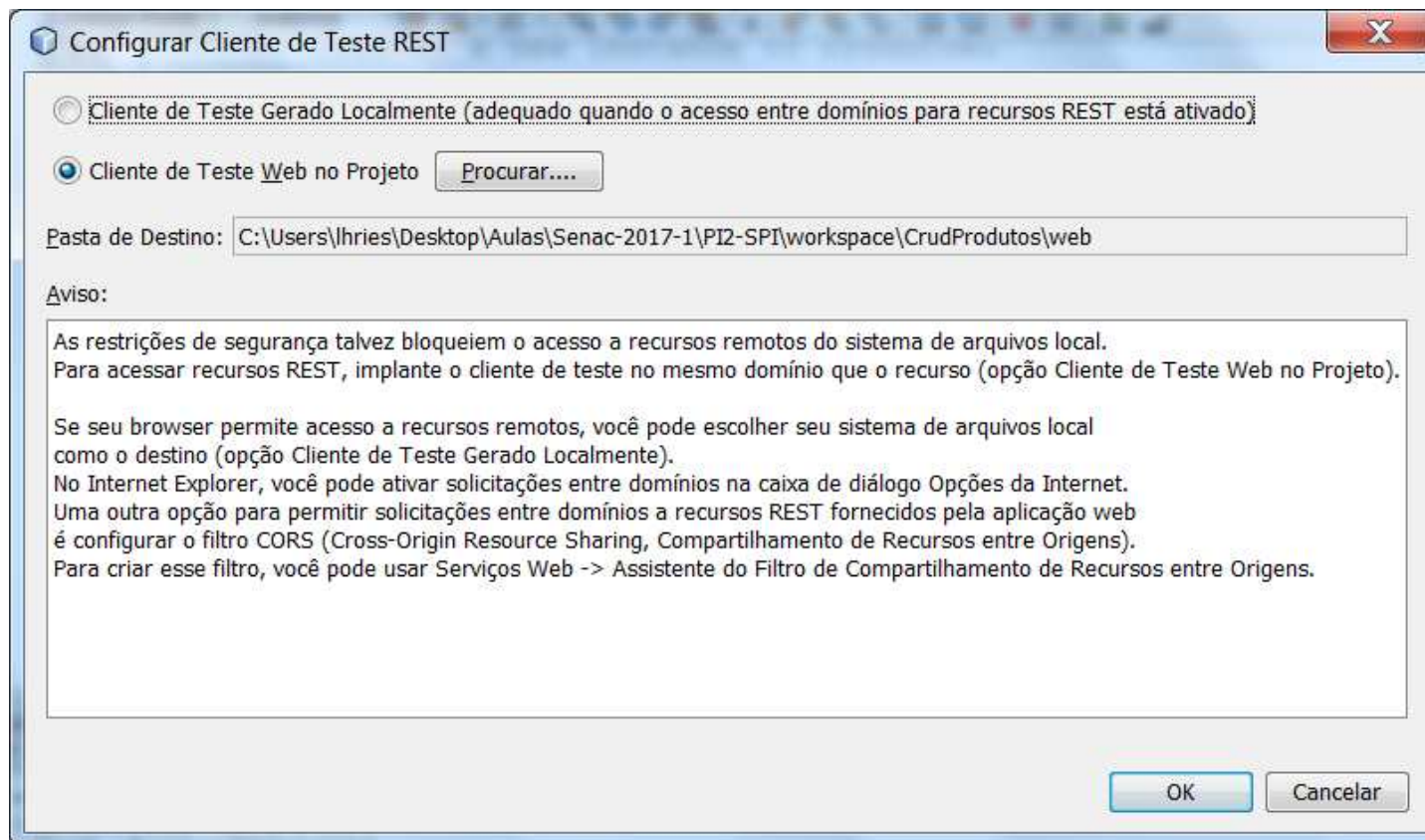
# Etapa 5

- Após terminar o processo de compilação e implantação, selecione a opção “Testar Web Services RESTful”.



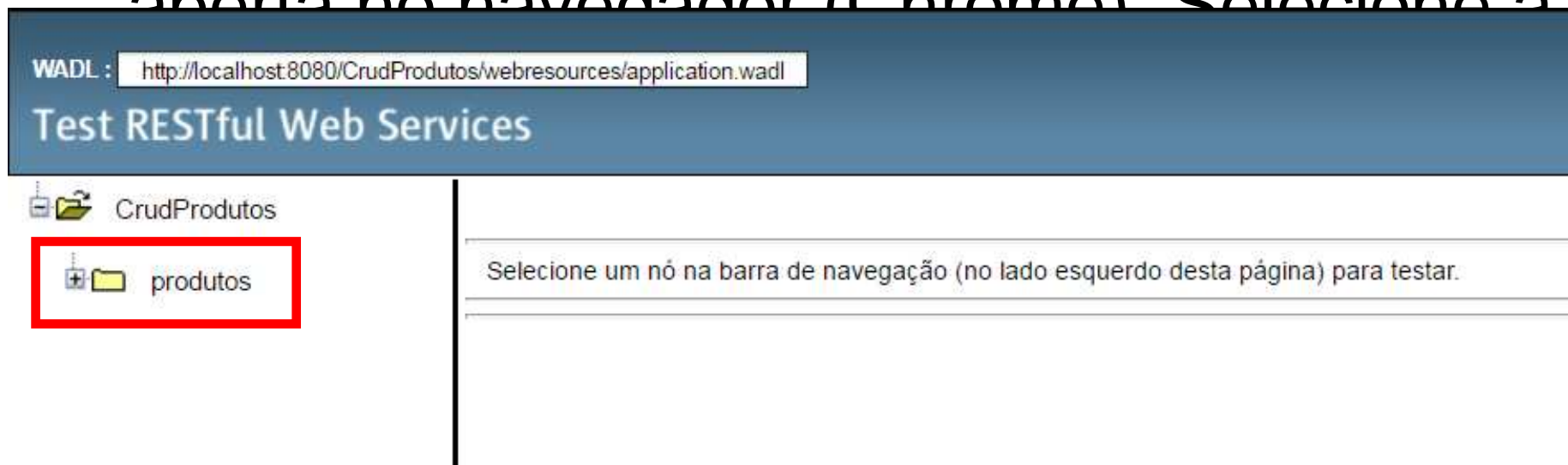
# Etapa 5

- Mantenha a opção de Cliente de Teste Web no projeto, clicando direto no botão “OK”.



# Etapa 5

- Após terminar o processo de implantação novamente e a página Web de teste será aberta no navegador (Chrome). Selecione o



# Etapa 5

- Após clicar na pasta “produtos”, podemos visualizar o recurso “produtos” e os métodos existentes para acessar diretamente esse recurso. Selecione o método “POST”.





# Etapa 5

- Após selecionar o método, descreva o conteúdo do produto (em JSON) e clique no botão “Testar”.

CrudProdutos > produtos

---

**Recurso:** produtos  
(<http://localhost:8080/CrudProdutos/webresources/produtos>)

---

Escolher método para testar: POST(application/json) ▼ Testar

Conteúdo

```
{  
  "codigo":1,  
  "nome":"produto1",  
  "preco":1  
}
```

---

↓ Custom Request Headers

# Etapa 5

- O resultado é o status 201 (recurso criado), conforme apresentado abaixo.

**Status: 201 (Created)**

**Resposta:**

View Tabular	View Bruta	Sub-Recurso	Cabeçalhos	Monitor Http
Nenhum conteúdo na resposta				

# Etapa 5

- Altere o método para “GET” e clique no botão “Testar”

---

CrudProdutos > produtos

---

**Recurso:** produtos

(<http://localhost:8080/CrudProdutos/webresources/produtos>)

---

**Escolher método para testar**

GET(application/json) ▼

Testar

---

↓ Custom Request Headers

# Etapa 5

- O resultado do status é 200 (OK) e retorna a lista de produtos (somente com o produto recém cadastrado), conforme a figura a seguir.

**Status:** 200 (OK)

**Resposta:**

View Tabular	View Bruta	<u>Sub-Recurso</u>	Cabeçalhos	Monitor Http
[{"codigo":1,"nome":"produto1","preco":1.0}]				

# Etapa 5

- Para finalizar, expanda a pasta “produtos” e selecione “{codigo}” [1]. Indique o valor 1 para o campo “codigo” [2] e clique no botão “Testar” [3].



# Etapa 5

- O resultado é status 200 e o conteúdo contendo o produto com código igual a 1, conforme a figura a seguir.

**Status:** 200 (OK)

**Resposta:**

View Tabular	View Bruta	Sub-Recurso	Cabeçalhos	Monitor Http
{\"codigo\":1,\"nome\":\"produto1\",\"preco\":1.0}				

# Etapa 5

- [OPCIONAL] Realize novos testes, criando novos produtos e teste outras operações: Atualizar e Remover).