

Guia De Desenvolvimento
Spring MVC
Thymeleaf
Spring Boot
Git
2023
Parte 2

Introdução

Neste documento teremos o passo-a-passo para a construção de uma aplicação web utilizando as ferramentas Spring MVC, Thymeleaf, Spring Boot e Git.

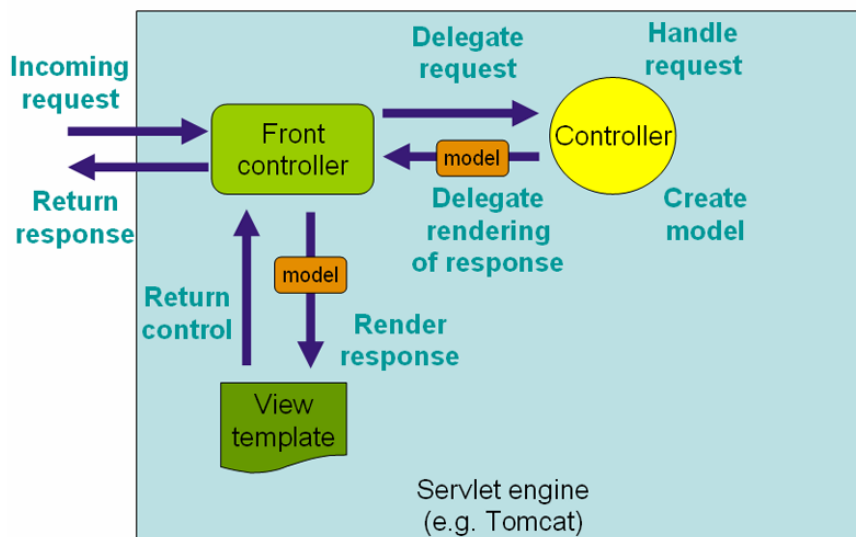
Este guia se destina a dar suporte as aulas de LIP1 do segundo módulo do curso de informática noturno do ITB Belval, portanto, algumas explicações mais detalhadas serão omitidas pois serão apresentadas pelo professor durante a aula.

Esta é a segunda parte do guia e para acompanhá-la é necessário obter o projeto desenvolvido na parte 1 em <https://github.com/ProfNpc/CrudExampleParte2/tree/main/inicial>.

Parte 2

Spring MVC, Controllers, Models e Views

Para manipular as requisições provenientes do *browser* (navegadores como o Chrome, Edge ou Firefox), o framework **Spring MVC** define a figura dos *Controllers*, responsáveis por controlar o fluxo da aplicação ao definir quais páginas(Views) devem ser devolvidas para o navegador para cada requisição que chega ao server além de controlar quais objetos do modelo(Model) devem ser acessados e devolvidos para o navegador preenchendo as View's com seus dados.



Na prática, um Controller é uma classe cujos métodos são mapeados, isto é, associados, a certas url's de forma que, quando uma requisição com uma url, por exemplo, "abc", for recebida pelo servidor web, o método associado a url "abc" será executado para processar essa requisição, produzindo, ao final, a resposta esperada, seja ela uma página html, seja a descrição de um objeto qualquer utilizando a notação JSON (JavaScript Object Notation) ou qualquer outro tipo de resposta.

Exemplo de mapeamento 1

Considerando que nosso servidor está rodando na mesma máquina do navegador, e que nossa aplicação esteja escutando a porta 8080, para mapear uma requisição HTTP do tipo GET com a url "http://localhost:8080/produto/novo" para ser processada pelo método novo() de uma classe Controller, podemos "anotar" o método como abaixo:

```
...
@GetMapping("/produto/novo")
public String novo() {
    ...
    return "novo-produto";
}
...
```



Atenção: Podemos substituir a anotação

```
@GetMapping("/produto/novo")
```

Por sua versão mais longa

```
@RequestMapping(value = "/produto/novo", method = RequestMethod.GET)
```

Observe que o retorno do método é uma String contendo o texto "novo-produto". Quando o Thymeleaf está configurado no projeto, a resposta para a requisição GET "http://localhost:8080/produto/novo" será o conteúdo de uma página "novo-produto.html".

Exemplo de mapeamento 2 – Mapeando campos de um form nos atributos de uma classe

As vezes precisaremos mapear métodos que precisarão receber valores provenientes de requisições oriundas de formulários html.

Considere que a página html “novo-produto.html” contém o código abaixo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Novo Produto</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Novo Produto</h1>

    <form action="/produto/novo" method="post">

      <input type="hidden" name="id" value="0"> <br>

      <span for="nome">Nome:</span>
      <input type="text" name="nome" > <br>

      <span for="descricao">Descrição:</span>
      <input type="text" name="descricao" > <br>

      <span for="preco">Preço:</span>
      <input type="text" name="preco" value="0.00"><br>

      <input type="submit" value="Enviar">

    </form>

  </body>
</html>
```

Quando for feita uma requisição do tipo GET para “http://localhost:8080/produto/novo”, será apresentado o formulário abaixo:

Onde está o campo id?

O campo id não aparece no navegador por que ele é do tipo “hidden”, “escondido”, isso é comum quando precisamos manter informações na página que queremos que o usuário não tenha acesso.

Observe que todos os campos deste formulário possuem o atributo “name” que os identifica.

```
<input type="text" name="nome" > <br>
```

O valor do atributo “name” é utilizado para recuperar o conteúdo desses campos quando a requisição chegar ao servidor web.

Além disso, a tag “form” possui, em seu atributo “action”, a url para a qual será feita a requisição, no caso, “/produto/novo”, e a definição de qual método HTTP o formulário utilizará na requisição, no caso, “post”.

Observação: Apesar de haver outros métodos definidos no protocolo http, o atributo “method” da tag “form” só admite os valores “post” e “get”.

```
<form action="/produto/novo" method="post">
```

São essas as informações utilizadas para mapearmos um método para processar a requisição proveniente deste formulário.

Vejamos como poderia ficar o método que processaria a requisição gerada por este form:

```
@PostMapping("/produto/novo")
public String novo(Produto produto) {
    ...
    return "novo-produto-criado";
}
```

Observe que agora utilizamos a anotação “@PostMapping” pois o método HTTP utilizado é o “POST” como indicado no atributo “method” da tag “form”.

```
@PostMapping("/produto/novo")
```

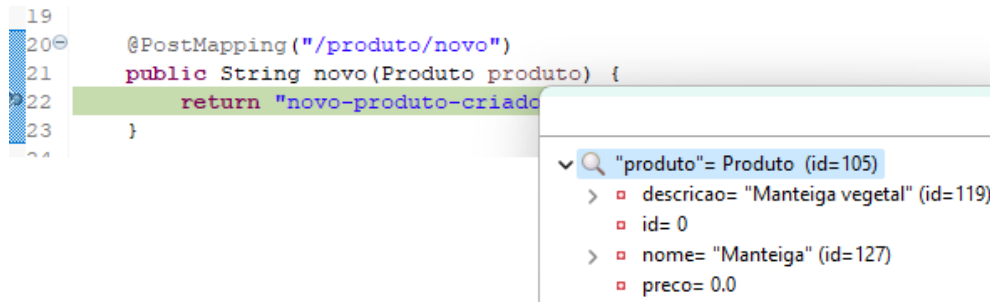
Além disso, o método novo() recebe como parâmetro um objeto do tipo **Produto**. Para que o objeto nesse parâmetro seja preenchido com os valores dos campos do form é necessário que a classe Produto possua atributos com os mesmos nomes dos campos do form, sem esquecer dos métodos getters e setters de cada um deles.

Produto precisa ser um **JavaBean**.

```
package br.com.belval.crud.model;

public class Produto {
    private int id;
    private String nome;
    private String descricao;
    private double preco;
    public Produto() {
    }
    public Produto(int id, String nome, String descricao, double preco) {
        this.id = id;
        this.nome = nome;
        this.descricao = descricao;
        this.preco = preco;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getDescricao() {
        return descricao;
    }
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }
    public double getPreco() {
        return preco;
    }
    public void setPreco(double preco) {
        this.preco = preco;
    }
    ...
}
```

Quando a requisição chegar ao método, o parâmetro produto possuirá os dados que foram preenchidos pelo usuário.



Atenção: Podemos substituir a anotação

`@PostMapping("/produto/novo")`

Por sua versão mais longa

`@RequestMapping(value = "/produto/novo", method = RequestMethod.POST)`

Exemplo de mapeamento 3 – Extraindo parâmetros diretamente da url

Em alguns casos, precisamos extrair valores diretamente da url. Nesses casos precisamos indicar no mapeamento da url quais partes dela são “variáveis”. Para isso utilizaremos a anotação “`@PathVariable`”.

```
@GetMapping("/produto/{id}")
public ModelAndView detalhe(@PathVariable int id) {
    ModelAndView modelAndView = new ModelAndView("detalhe-produto");
    modelAndView.addObject("id", id);
    return modelAndView;
}
```

Neste caso, caso seja feita uma requisição com a url “`http://localhost:8080/produto/3`”, o valor “3” será carregado no parâmetro “id” do método `detalhe()`.

Thymeleaf - Mecanismos de mesclagem das View’s com os objetos Model

Uma vez que o Controller tenha feito seu trabalho, ele precisará enviar uma resposta para o browser (navegador). Em uma aplicação web, frequentemente, essa resposta deverá ser uma página html onde certos trechos deverão ser substituídos por valores recuperados pelo Controller do banco de dados ou de outras fontes e que são recuperados através da instanciação de objetos do Model. Esses objetos do Model (modelo) representam entidades do negócio que a aplicação web procura tratar. Por exemplo, em uma aplicação de livraria virtual vamos encontrar classes que representam entidades presentes no negócio de venda de livros pela internet. Em um sistema como esse, provavelmente, encontraremos classes do Model como “Livro”, “Cliente”, “Autor”, “Editora”, “Avaliação” e etc.

O componente que utilizaremos para efetuar a mesclagem dos templates (páginas html) com os dados será o **Thymeleaf**. O Thymeleaf funciona como um motor de templates em que páginas semiacabadas podem ser mescladas com dados provenientes de parâmetros passados para esse motor.

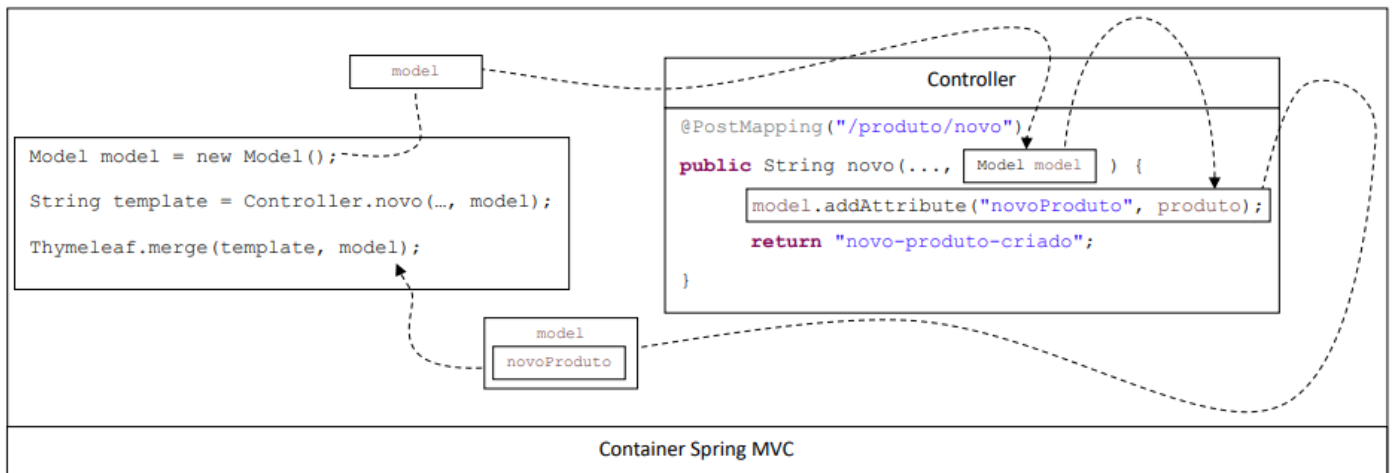
Para mesclarmos as view’s com objetos do Model utilizando o Thymeleaf, existem 2 formas principais, a primeira envolve a utilização da classe **Model** e a segunda envolve a classe **ModelAndView**.

Mesclagem utilizando a classe Model

Para mesclarmos uma View (template html) com objetos do Model devemos declarar um parâmetro adicional ao método do Controller mapeado para tratar certa requisição, como no exemplo a seguir:

```
@PostMapping("/produto/novo")
public String novo(Produto produto, Model model) {
    //Processa requisição, insere no banco de dados e etc...
    model.addAttribute("novoProduto", produto);
    return "novo-produto-criado";
}
```

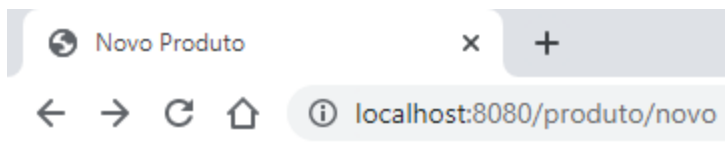
O objeto presente no parâmetro model é instanciado pelo próprio container do Spring MVC e este mantém a referência para o objeto depois do método novo() ser executado. Com esta referência, o Spring MVC consegue recuperar os atributos adicionados ao objeto Model durante a execução do método novo() do Controller.



Finalmente, para que a mesclagem funcione, precisamos que o template contenha as indicações de onde os dados passados para o mecanismo de mesclagem do Thymeleaf deverão ser inseridos. Isso é feito utilizando a notação típica do Thymeleaf `th:text${...}`. Veja abaixo o exemplo da página/template `novo-produto-criado.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Novo Produto Criado</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Novo produto criado!</h1>
    <table border="1">
      <tr>
        <th>ID:</th>
        <td>
          <span th:text="${novoProduto.id}"></span>
        </td>
      </tr>
      <tr>
        <th>Nome:</th>
        <td>
          <span th:text="${novoProduto.nome}"></span>
        </td>
      </tr>
      <tr>
        <th>Descrição:</th>
        <td>
          <span th:text="${novoProduto.descricao}"></span>
        </td>
      </tr>
      <tr>
        <th>Preço:</th>
        <td>
          <span th:text="${novoProduto.preco}"></span>
        </td>
      </tr>
    </table>
  </body>
</html>
```

Quando essa página for retornada para o navegador, depois de efetuarmos o “submit” do form, veremos algo semelhante a página a seguir.

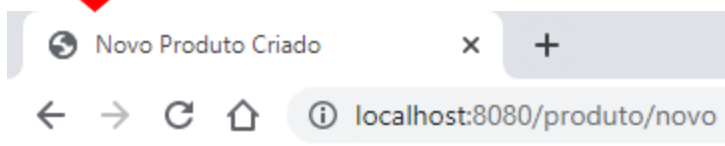


Novo Produto

Nome:

Descrição:

Preço:



Novo produto criado!

ID:	0
Nome:	Bala
Descrição:	Bala soft
Preço:	5.0

Mesclagem utilizando a classe ModelAndView

Para fazermos a mesclagem utilizando a classe **ModelAndView** basta instanciar um objeto do tipo **ModelAndView** dentro do método do Controller passando o template que deverá ser usado como parâmetro do construtor.

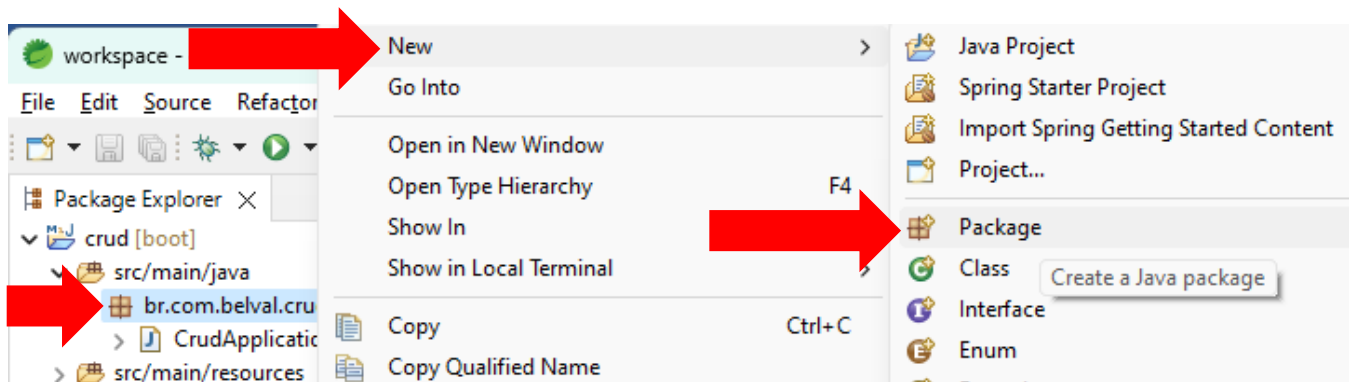
Em contraste com a utilização da classe **Model** quando usamos a classe **ModelAndView** devemos utilizar o método **addObject()** para adicionar os objetos cujos dados serão utilizados para preencher a página. Com **Model** utilizamos o método **addAttribute()**.

Outra alteração que deve ser feita é que obrigatoriamente devemos alterar o tipo de retorno do método de String para **ModelAndView** e retornar a instancia de **ModelAndView** que foi criada dentro do método.

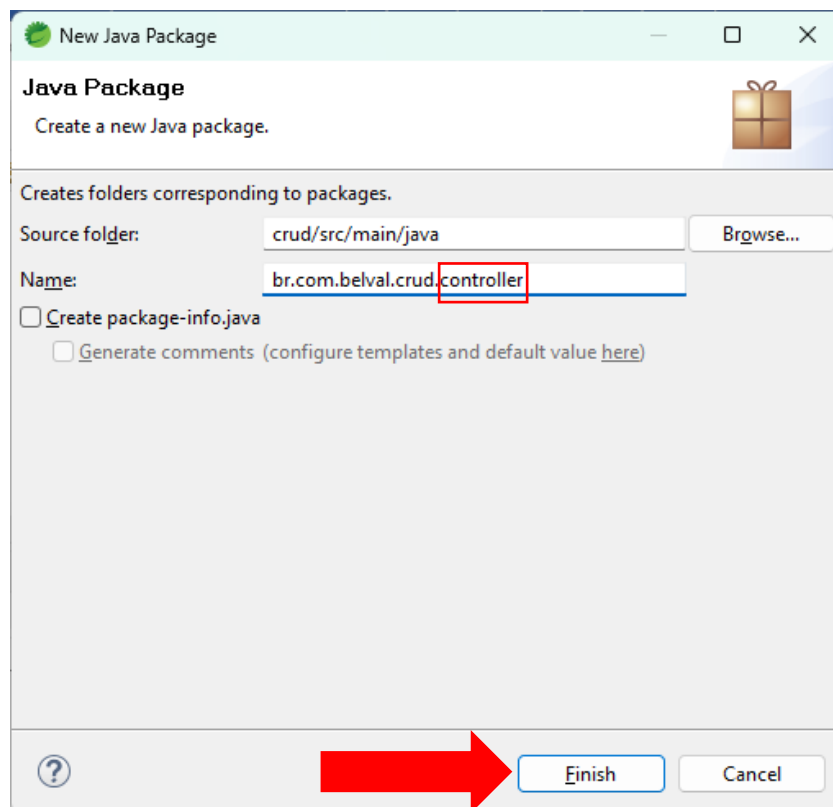
```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    //Processa requisição, insere no banco de dados e etc...
    ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
    modelAndView.addObject("novoProduto", produto);
    return modelAndView;
}
```


Criando uma tela de cadastro – adicionando ProdutoController

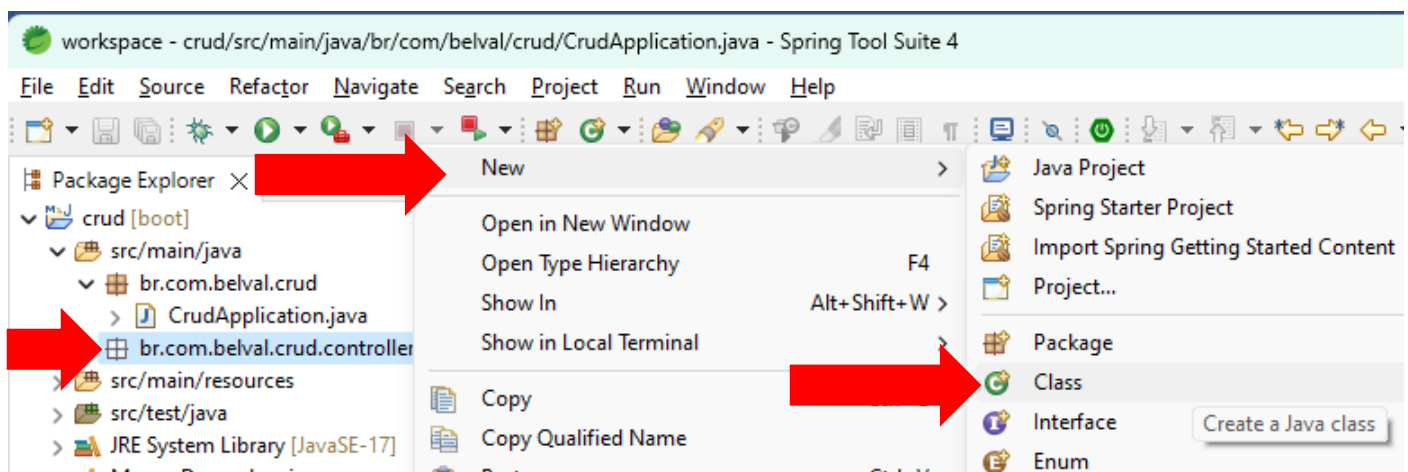
1 – Clique com o botão direito do mouse no pacote raiz da aplicação > New > Package



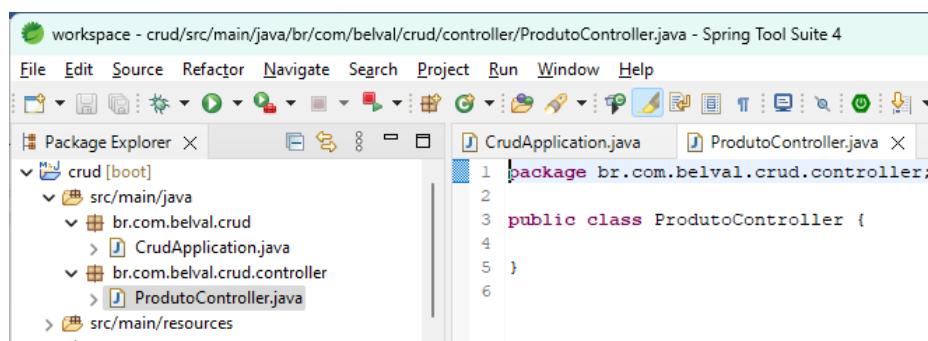
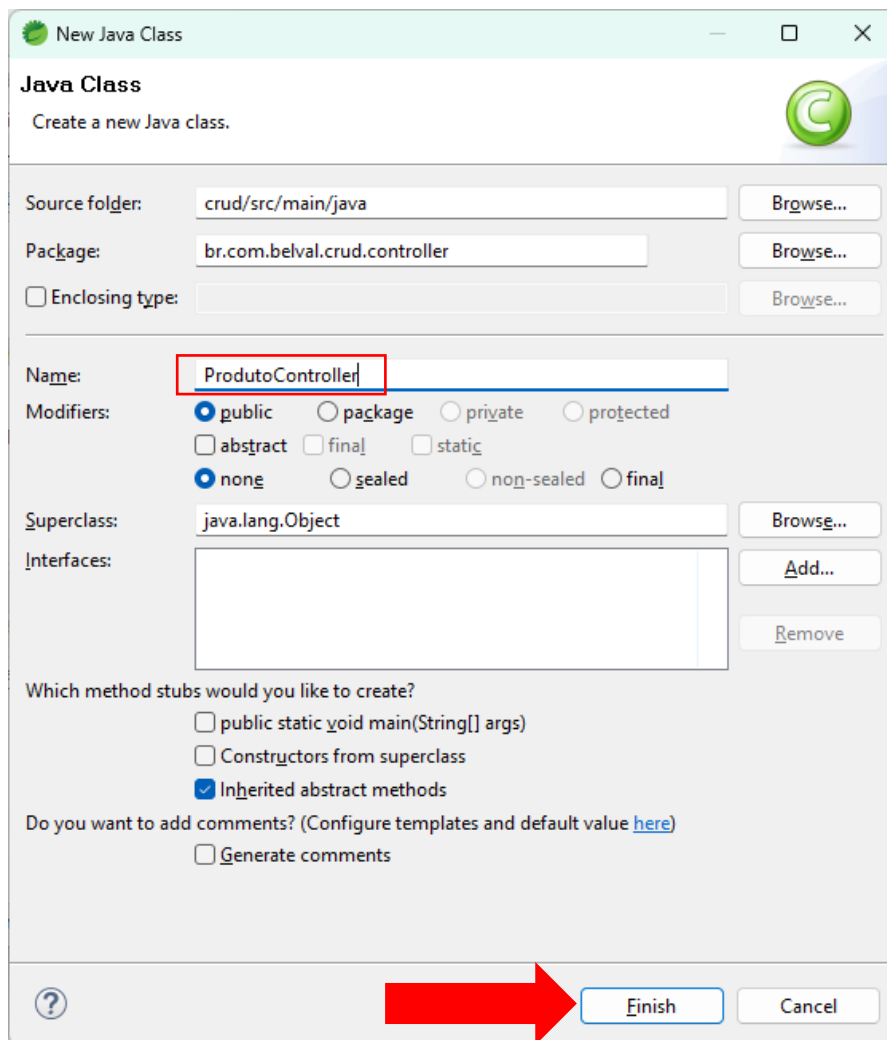
2 – Adicione “controller” ao final do nome do pacote e clique em “Finish”.



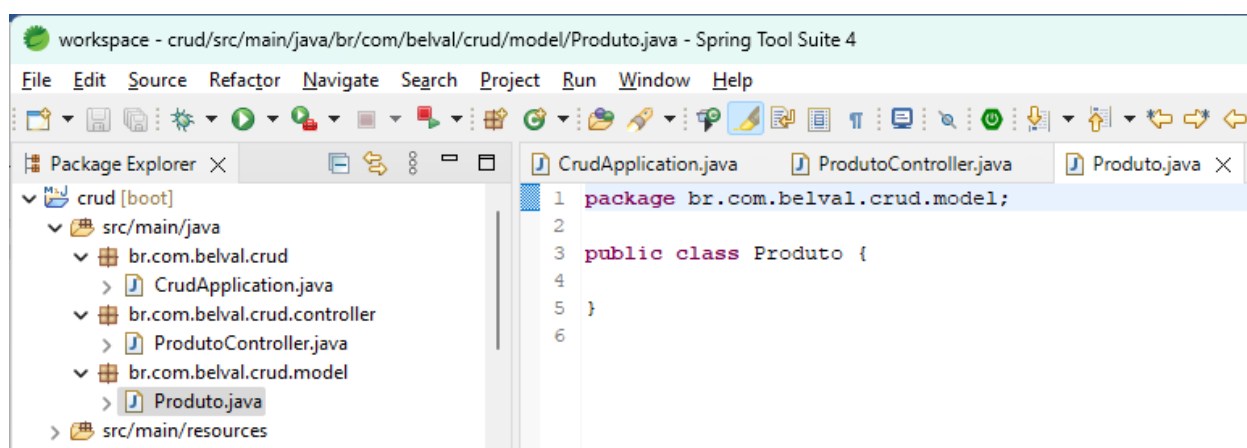
3 – Em seguida, clique com o botão direito do mouse no novo package criado > New > Class



4 – Preencha o nome da classe com “ProdutoController” e clique em “Finish”.



5 – Crie um pacote “model” e crie uma classe “Produto”



5 – Adicione o código a seguir na classe Produto e ProdutoController

```
package br.com.belval.crud.model;

public class Produto {
    private int id;
    private String nome;
    private String descricao;
    private double preco;

    public Produto() {
    }

    public Produto(int id, String nome, String descricao, double preco) {
        this.id = id;
        this.nome = nome;
        this.descricao = descricao;
        this.preco = preco;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public double getPreco() {
        return preco;
    }

    public void setPreco(double preco) {
        this.preco = preco;
    }

    @Override
    public String toString() {
        return "Produto [id=" + id + ", nome=" + nome + ", descricao=" +
descricao + ", preco=" + preco + "]\n";
    }
}
```

```

package br.com.belval.crud.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.ModelAndView;

import br.com.belval.crud.model.Produto;

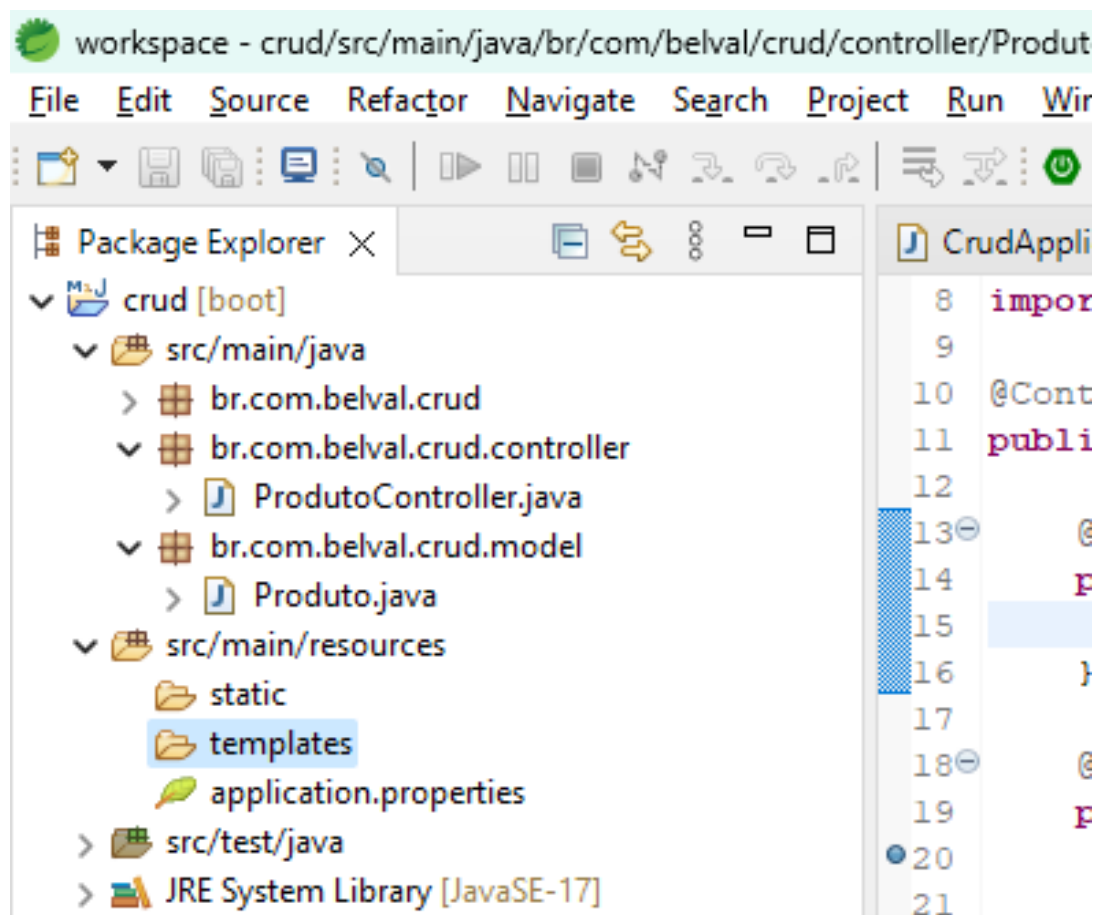
@Controller
public class ProdutoController {

    @GetMapping("/produto/novo")
    public String novo() {
        return "novo-produto";
    }

    @PostMapping("/produto/novo")
    public ModelAndView novo(Produto produto) {
        ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
        modelAndView.addObject("novoProduto", produto);
        return modelAndView;
    }
}

```

6 – Encontre o diretório “templates” em “src/main/resources”. É aqui que devemos colocar nossos templates/páginas para serem utilizadas pelo Thymeleaf.

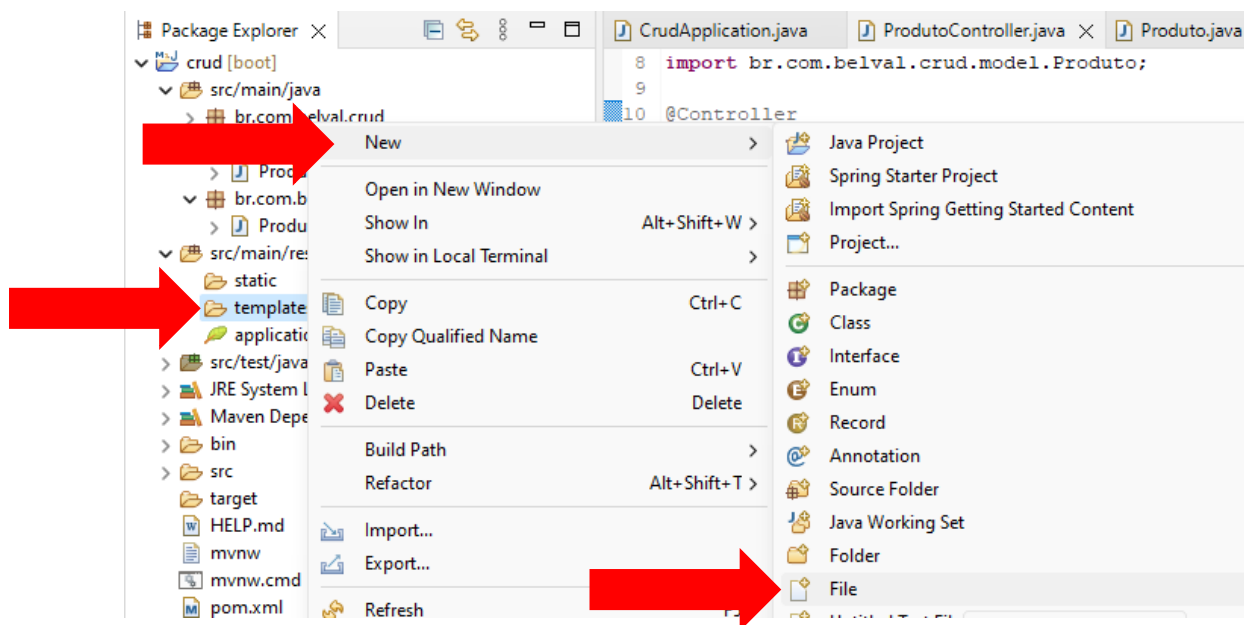


Observação: Nesse nosso CRUD colocaremos nossas páginas na raiz do diretório “template”, entretanto, em uma aplicação maior, com várias páginas, com outros CRUD’s, deixar todas as páginas na raiz pode ser um problema de organização, por isso, em um cenário como esse, é aconselhável criar subpastas dentro de “templates” para agrupar as páginas relacionadas como, por exemplo, as páginas que constituem o CRUD de uma certa entidade. Nesse caso,

a indicação da página no Controller deverá observar a estrutura de diretórios presente em “templates”. Se criarmos uma subpasta “produto” dentro de templates e colocarmos as páginas do CRUD de produto dentro dela, para que o Controller responda com as páginas de produto, devemos fazer como abaixo:

```
@GetMapping("/produto/novo")
public String novo() {
    return "produto/novo-produto";
}
```

7 – Clique com o botão direito do mouse sobre “templates” > New > File



8 – Cole o conteúdo abaixo na janela que foi aberta

```
<!DOCTYPE html>
<html>
  <head>
    <title>Novo Produto</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Novo Produto</h1>

    <form action="/produto/novo" method="post">

      <input type="hidden" name="id" value="0"> <br>

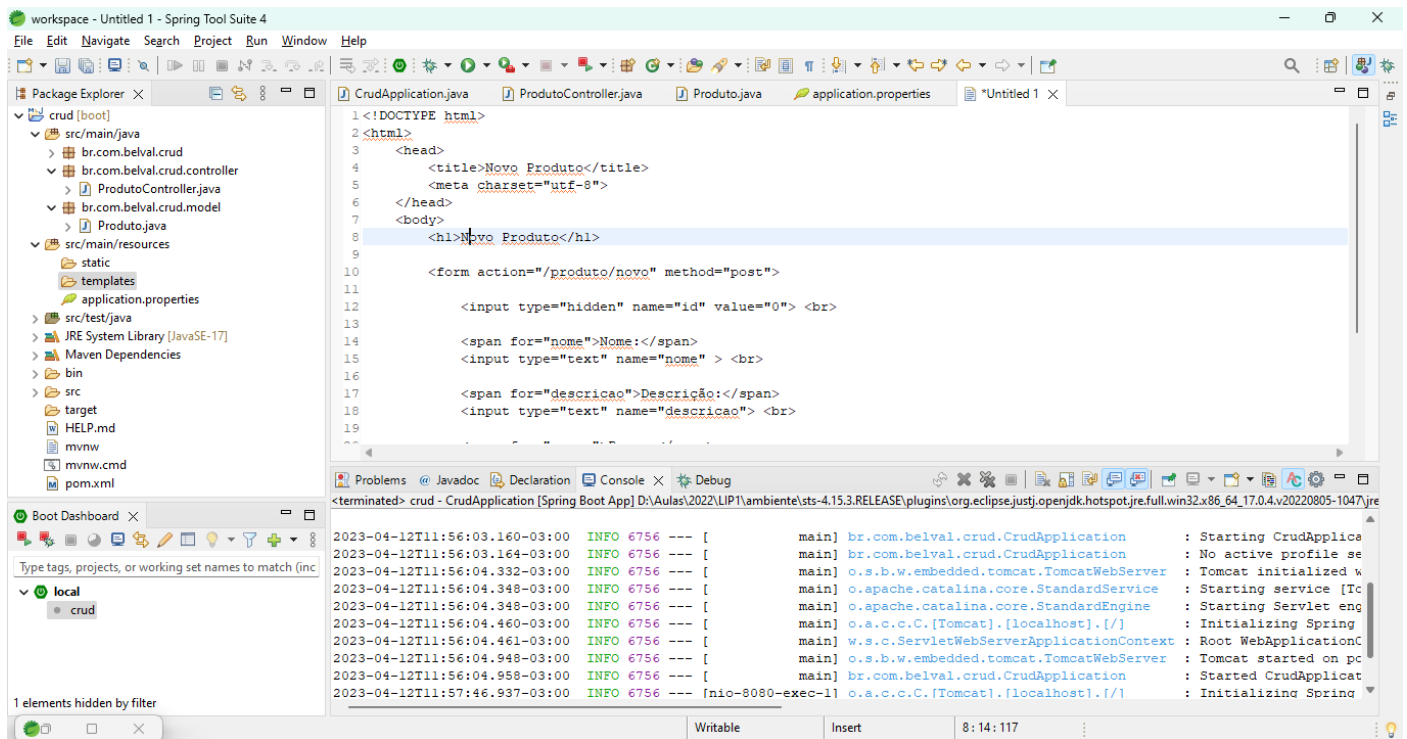
      <span for="nome">Nome:</span>
      <input type="text" name="nome" > <br>

      <span for="descricao">Descrição:</span>
      <input type="text" name="descricao" > <br>

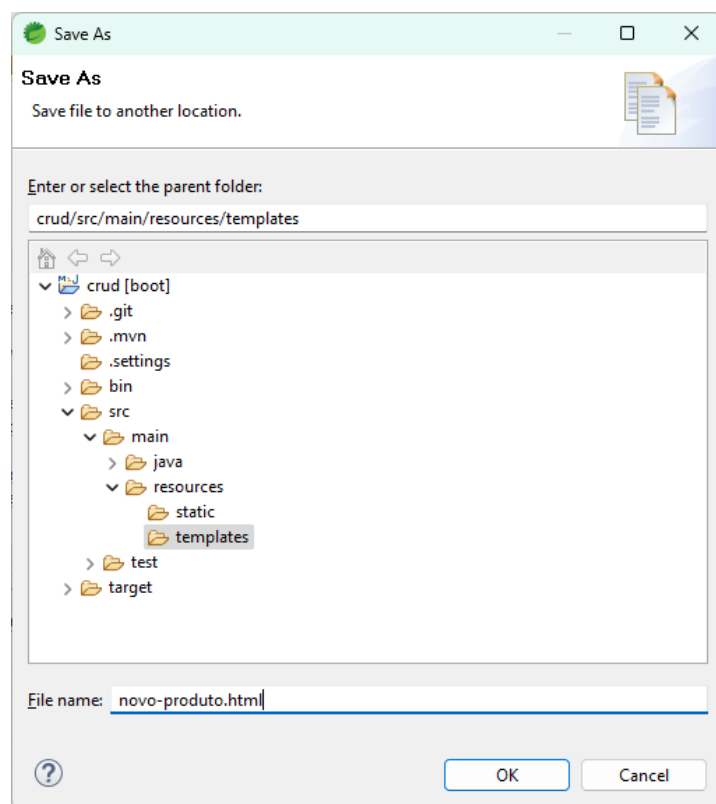
      <span for="preco">Preço:</span>
      <input type="text" name="preco" value="0.00"><br>

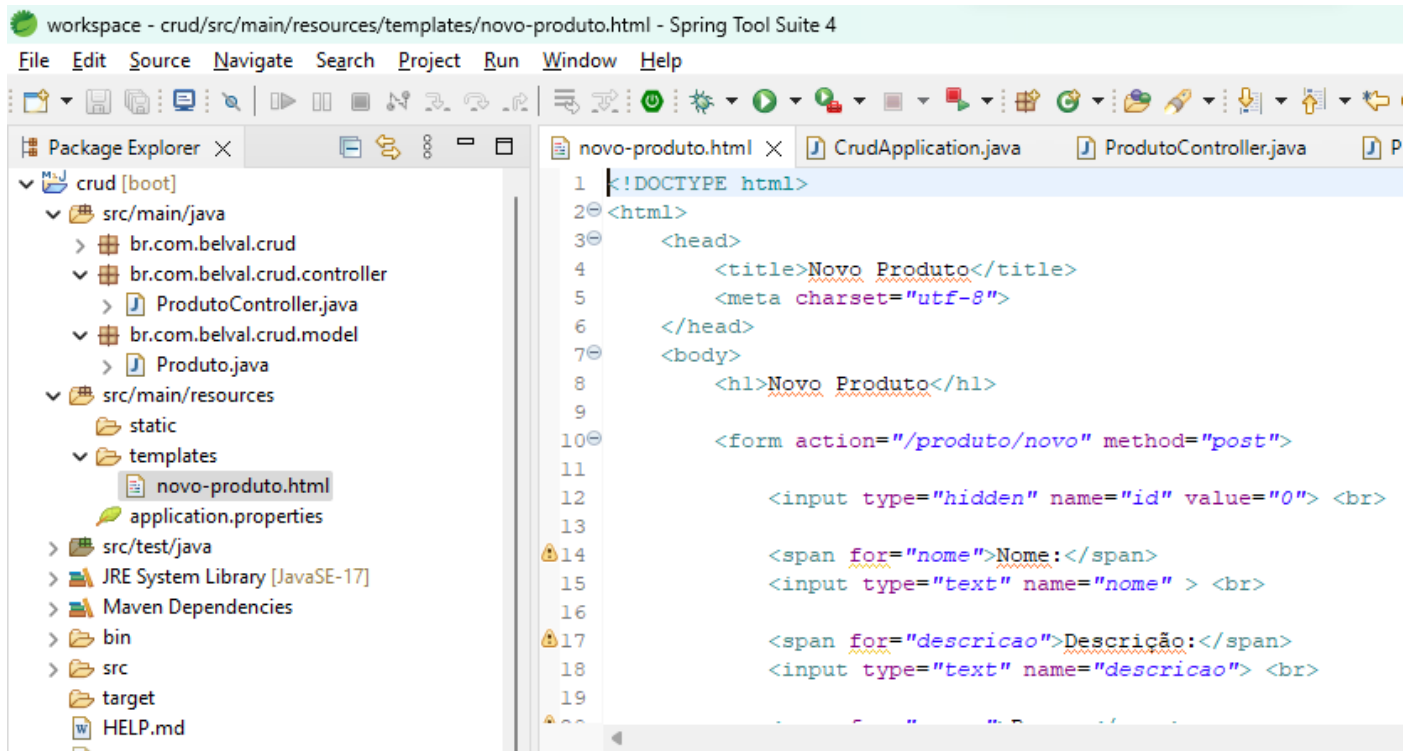
      <input type="submit" value="Enviar">
    </form>

  </body>
</html>
```



9 – Salve o arquivo com o atalho “Ctrl + S”, selecione o diretório “templates” e preencha o nome do arquivo com “novo-produto.html”.





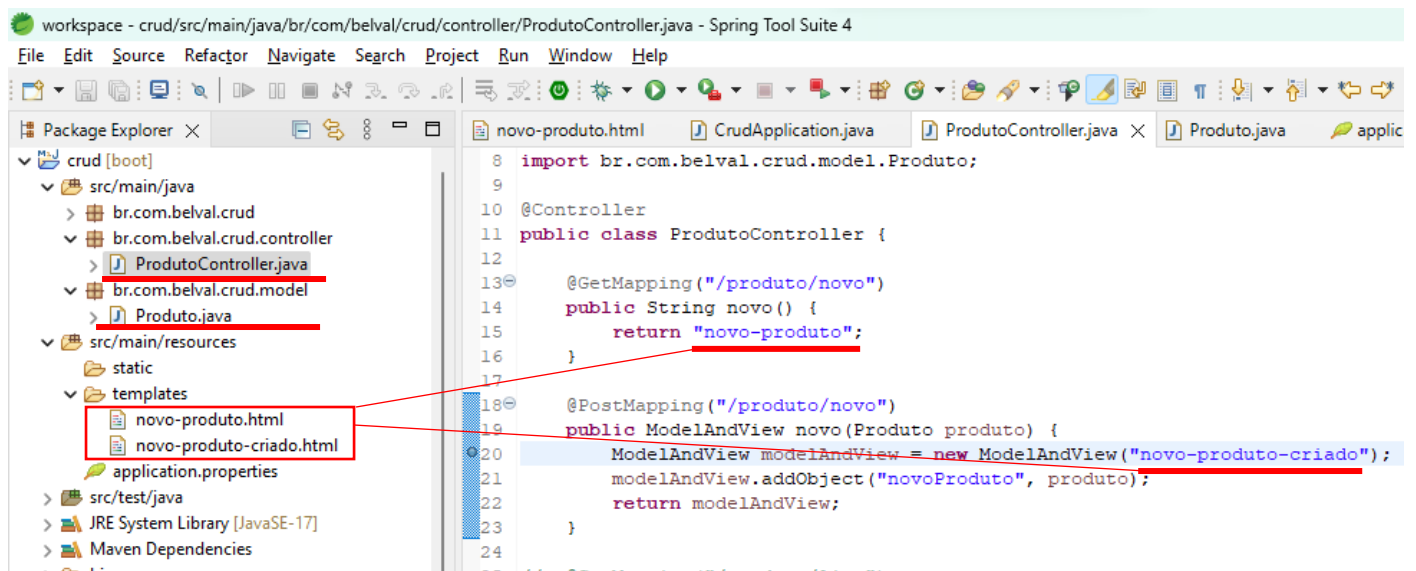
10 – Repita o processo e crie a página “novo-produto-criado.html” com o conteúdo abaixo(você pode criar o arquivo em outro editor como o notepad e depois copiar o arquivo selecionando a pasta “templates”):

```

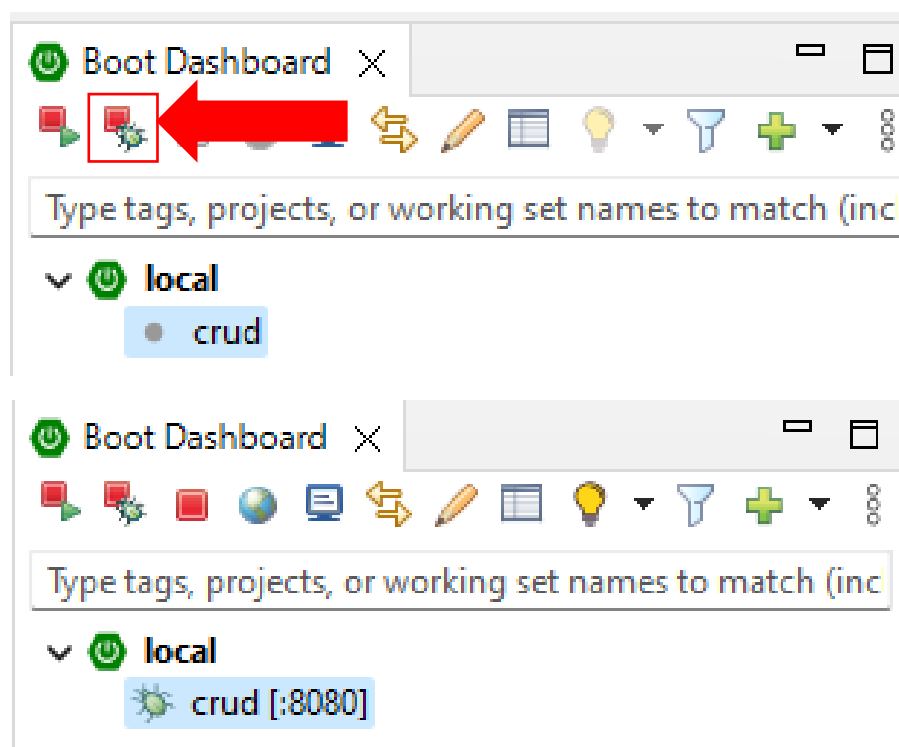
<!DOCTYPE html>
<html>
    <head>
        <title>Novo Produto Criado</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Novo produto criado!</h1>
        <table border="1">
            <tr>
                <th>ID:</th>
                <td>
                    <span th:text="${novoProduto.id}"></span>
                </td>
            </tr>
            <tr>
                <th>Nome:</th>
                <td>
                    <span th:text="${novoProduto.nome}"></span>
                </td>
            </tr>
            <tr>
                <th>Descrição:</th>
                <td>
                    <span th:text="${novoProduto.descricao}"></span>
                </td>
            </tr>
            <tr>
                <th>Preço:</th>
                <td>
                    <span th:text="${novoProduto.preco}"></span>
                </td>
            </tr>
        </table>
    </body>
</html>

```

Confira como deve estar o conteúdo dos diretórios e as páginas que são retornadas pelos métodos do Controller:



11 – Execute a aplicação em modo debug

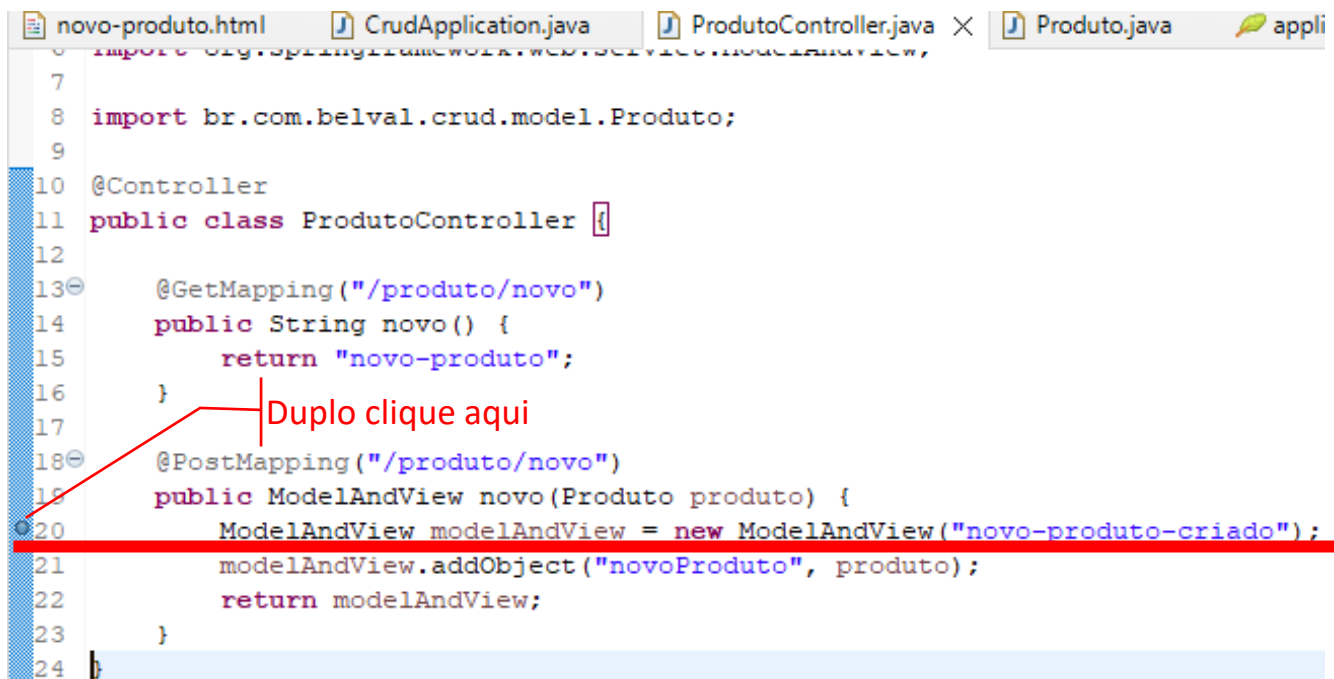


12 – Acesse a url "http://localhost:8080/produto/novo"

The screenshot shows a web browser window with the title "Novo Produto". The address bar shows the URL "localhost:8080/produto/novo". The page content displays a form with the following fields:

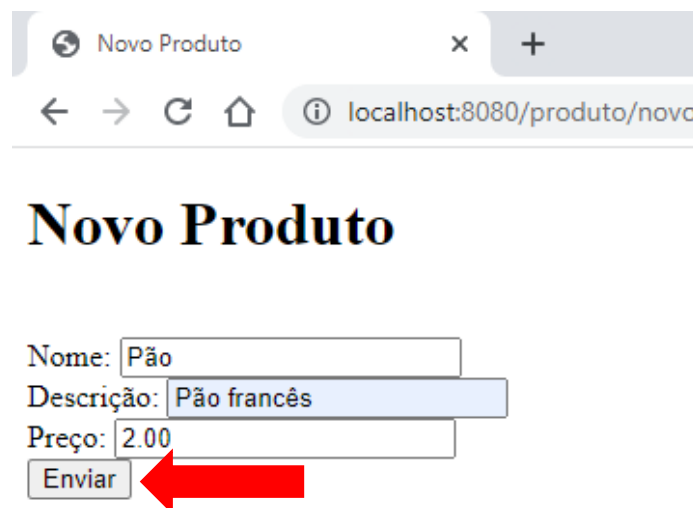
- Nome:
- Descrição:
- Preço:
- Enviar

13 – Dê um duplo clique na linha indicada abaixo, na classe ProdutoController



```
7
8 import br.com.belval.crud.model.Produto;
9
10 @Controller
11 public class ProdutoController {
12
13     @GetMapping("/produto/novo")
14     public String novo() {
15         return "novo-produto";
16     }
17
18     @PostMapping("/produto/novo")
19     public ModelAndView novo(Produto produto) {
20         ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
21         modelAndView.addObject("novoProduto", produto);
22         return modelAndView;
23     }
24 }
```

14 – Preencha os campos do formulário e clique no botão “Enviar”



Novo Produto

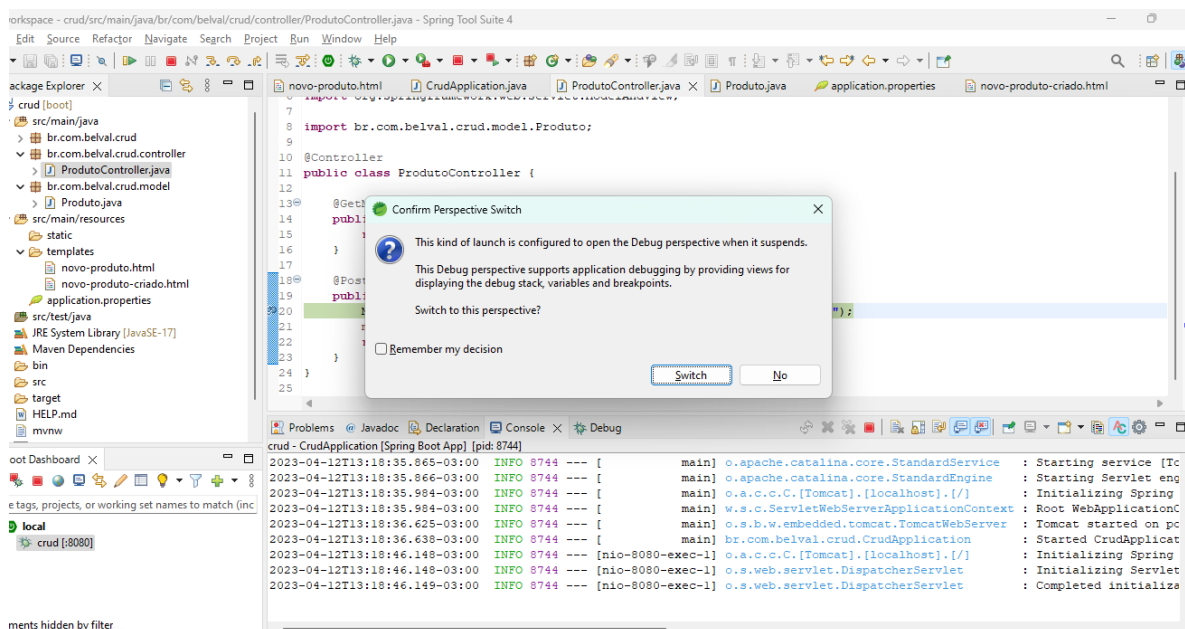
Nome: Pão

Descrição: Pão francês

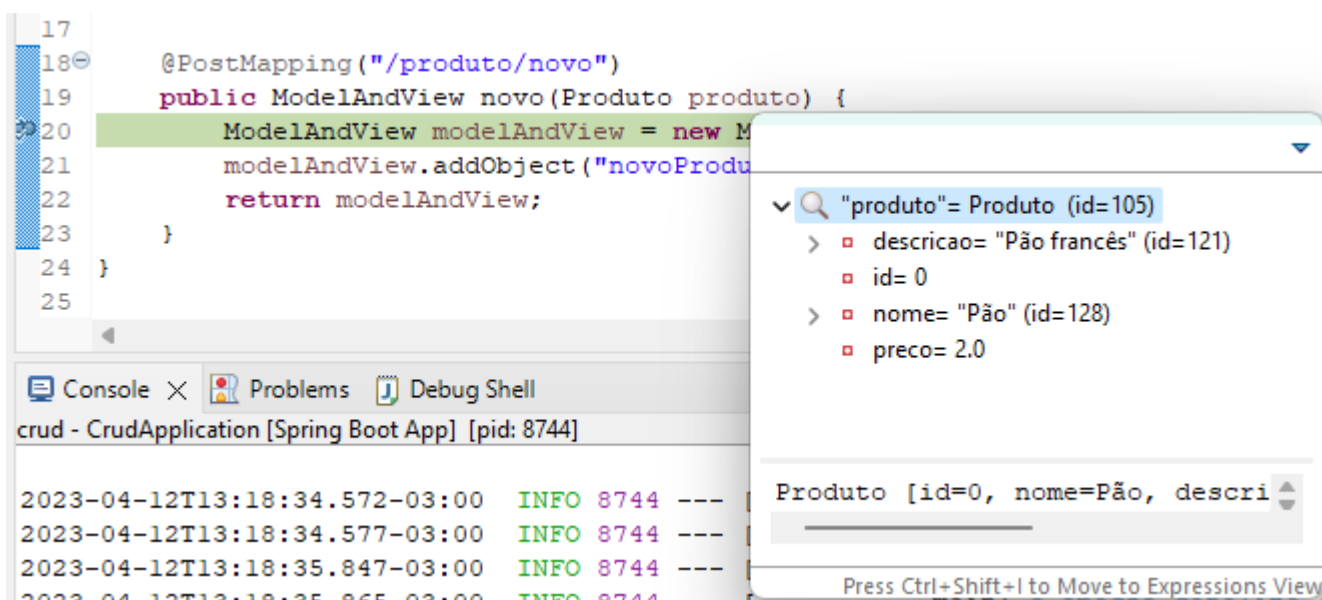
Preço: 2.00

Enviar

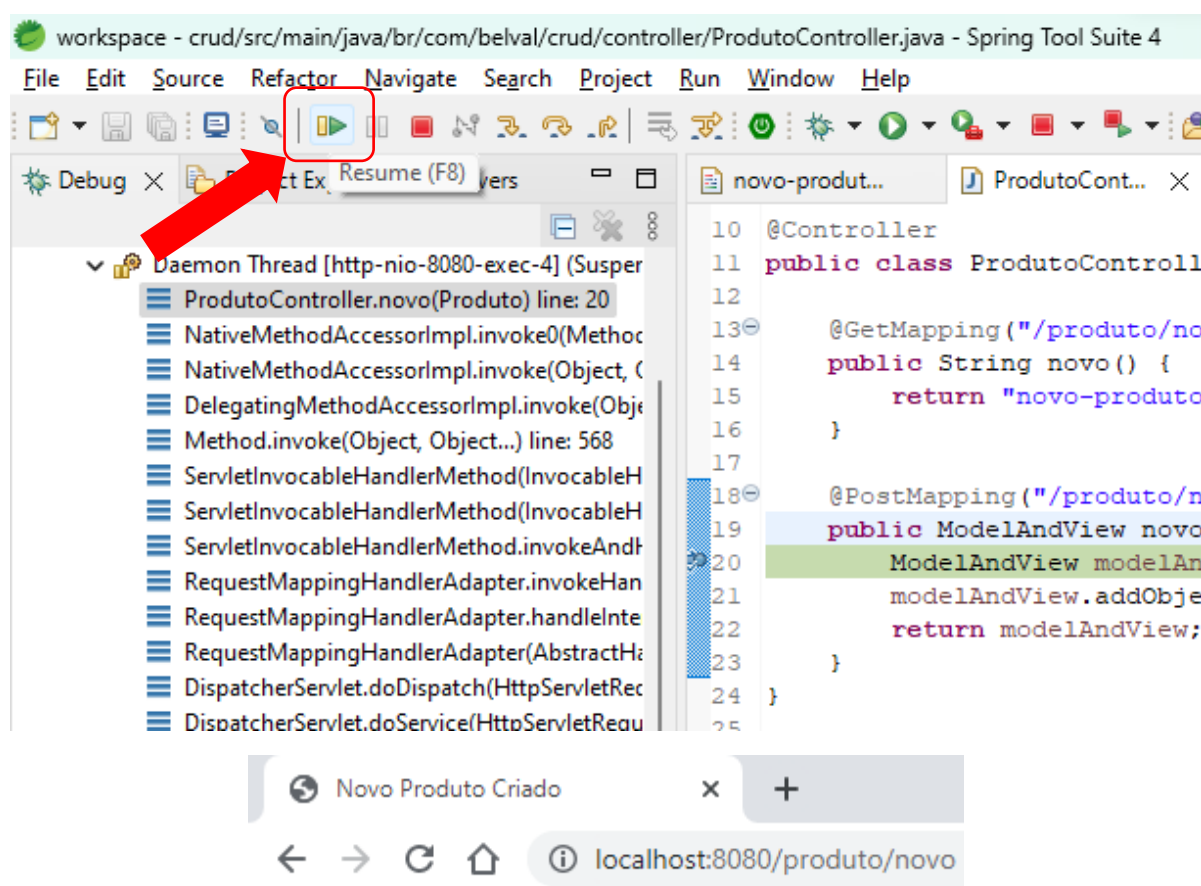
15 – Volte para a IDE e clique em “Switch”(trocar)



16 – Selecione o parâmetro “produto”, use o atalho “Ctrl + SHIFT + i” e poderá ver que o valores preenchidos no formulário foram carregados nos atributos do objeto recebido como parâmetro.



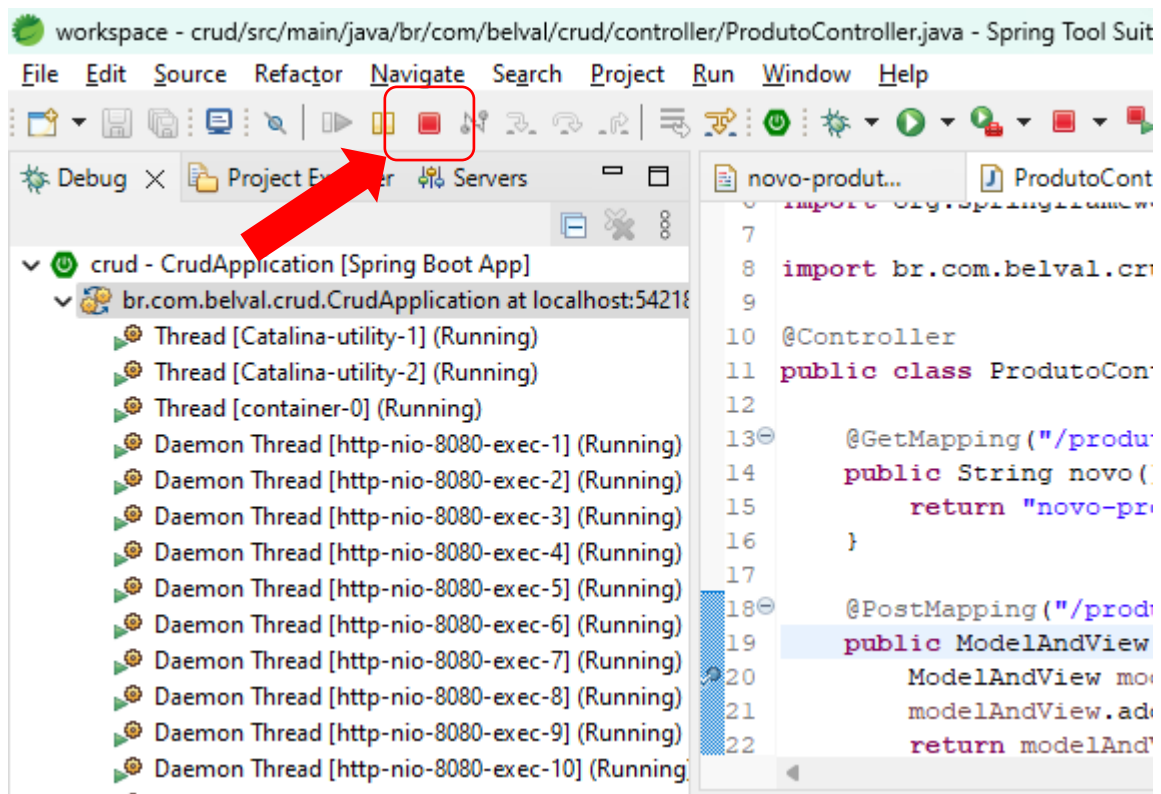
17 – Clique no botão “Resume” para fazer com que a execução da aplicação dispare novamente.



Novo produto criado!

ID:	0
Nome:	Pão
Descrição:	Pão francês
Preço:	2.0

18 – Pare a execução clicando no botão “stop”.



19 – Abra o “Git Bash” e execute os comandos a seguir

20 – Mude para o diretório do projeto, algo como “.../workspace/crud”. O comando “cd” vem de “Change Directory”.

```
cd /<sua unidade de rede>/<seu caminho>/workspace/crud
```

```
MINGW64/D:/Aulas/2023/Mo x + v
takol@DESKTOP-8THFTQM MINGW64 ~
$ cd /D/Aulas/2023/Modulo/Material/workspace/crud
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ |
```

21 – Veja o status do código fonte

```
git status
```

```
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  src/main/java/br/com/belval/crud/controller/
  src/main/java/br/com/belval/crud/model/
  src/main/resources/templates/

nothing added to commit but untracked files present (use "git add" to track)
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ |
```

22 – Prepare o código que será preservado no repositório e verifique o status do código fonte novamente

```
git add .
git status
```

```
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ git add .

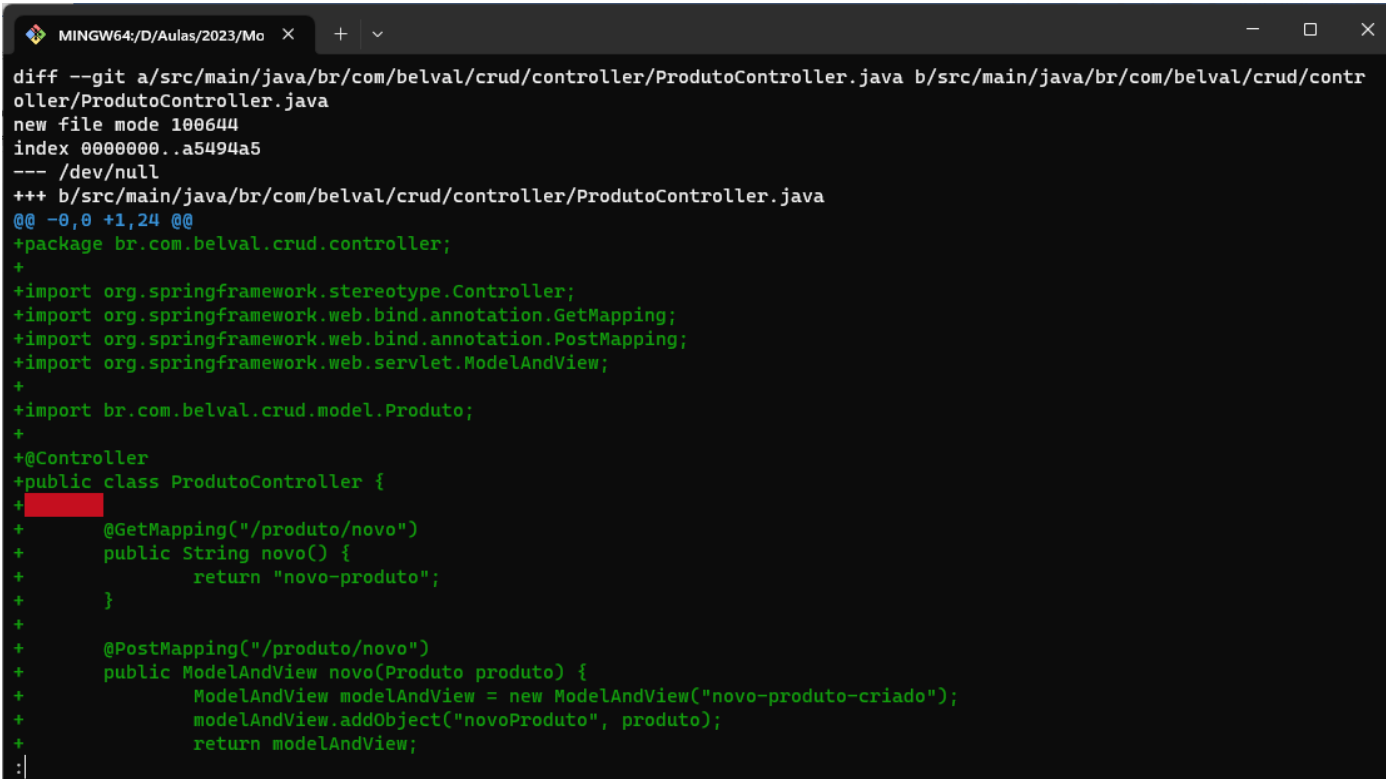
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   src/main/java/br/com/belval/crud/controller/ProdutoController.java
    new file:   src/main/java/br/com/belval/crud/model/Produto.java
    new file:   src/main/resources/templates/novo-produto-criado.html
    new file:   src/main/resources/templates/novo-produto.html

takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$
```

22 – Verifique quais alterações foram feitas no projeto com o comando abaixo:

```
git diff --cached .
```



```
diff --git a/src/main/java/br/com/belval/crud/controller/ProdutoController.java b/src/main/java/br/com/belval/crud/controller/ProdutoController.java
new file mode 100644
index 0000000..a5494a5
--- /dev/null
+++ b/src/main/java/br/com/belval/crud/controller/ProdutoController.java
@@ -0,0 +1,24 @@
+package br.com.belval.crud.controller;
+
+import org.springframework.stereotype.Controller;
+import org.springframework.web.bind.annotation.GetMapping;
+import org.springframework.web.bind.annotation.PostMapping;
+import org.springframework.web.servlet.ModelAndView;
+
+import br.com.belval.crud.model.Produto;
+
+@Controller
+public class ProdutoController {
+
+    @GetMapping("/produto/novo")
+    public String novo() {
+        return "novo-produto";
+    }
+
+    @PostMapping("/produto/novo")
+    public ModelAndView novo(Produto produto) {
+        ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
+        modelAndView.addObject("novoProduto", produto);
+        return modelAndView;
+    }
+}
```

No modo “comparação” é possível descer para ver todas as alterações utilizando as setas para cima e para baixo do teclado. Para sair do modo “comparação” do comando “git diff” e voltar para o prompt normal basta pressionar a tecla “q”.

23 – Adicione as alterações ao gerenciador de versões de forma permanente com o comando abaixo:

```
git commit -m "Adicionado form produto"
```

```
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ git commit -m "Adicionado form produto"
[main 8d1cd0a] Adicionado form produto
4 files changed, 132 insertions(+)
create mode 100644 src/main/java/br/com/belval/crud/controller/ProdutoController.java
create mode 100644 src/main/java/br/com/belval/crud/model/Produto.java
create mode 100644 src/main/resources/templates/novo-produto-criado.html
create mode 100644 src/main/resources/templates/novo-produto.html

takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ |
```

24 – Veja o histórico de versões do projeto com o comando **git log** ou sua versão simplificada **git log --oneline**

```
git log
```

```
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ git log
commit 8d1cd0a8571510acca80246bc5bc65cd4c892602 (HEAD -> main)
Author: Alessandro <professornpc@gmail.com>
Date: Wed Apr 12 13:58:04 2023 -0300

    Adicionado form produto

commit d7daee76a8e42698962837e6d341422ee1d49421 (origin/main)
Author: Alessandro <professornpc@gmail.com>
Date: Sun Apr 9 22:36:56 2023 -0300

    Add método olaMundo()

commit 5ed491a3ae059f49599727d989f929caad26f84c
Author: Alessandro <professornpc@gmail.com>
Date: Sun Apr 9 19:54:05 2023 -0300

    Versão inicial do projeto crud

commit cf20477fd86b19f67060519142259521cc297305
Author: Alessandro <professornpc@gmail.com>
Date: Sun Apr 9 19:47:11 2023 -0300

    Add .gitignore

takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ |
```

```
git log --oneline
```

```
takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$ git log --oneline
8d1cd0a (HEAD -> main) Adicionado form produto
d7daee7 (origin/main) Add método olaMundo()
5ed491a Versão inicial do projeto crud
cf20477 Add .gitignore

takol@DESKTOP-8THFTQM MINGW64 /D/Aulas/2023/Modulo/Material/workspace/crud (main)
$
```