

Roteiro Aula 3 – Thymeleaf e Controller

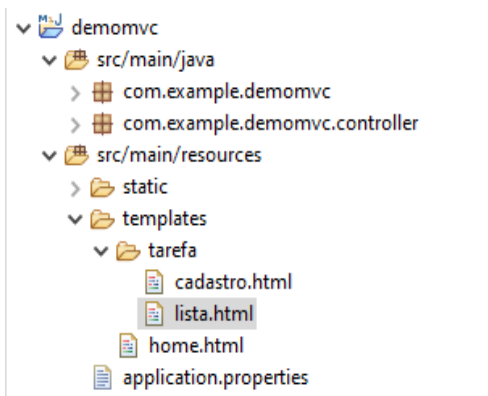
Tempo estimado para esta prática: 30min

IDE utilizada: Eclipse JavaEE

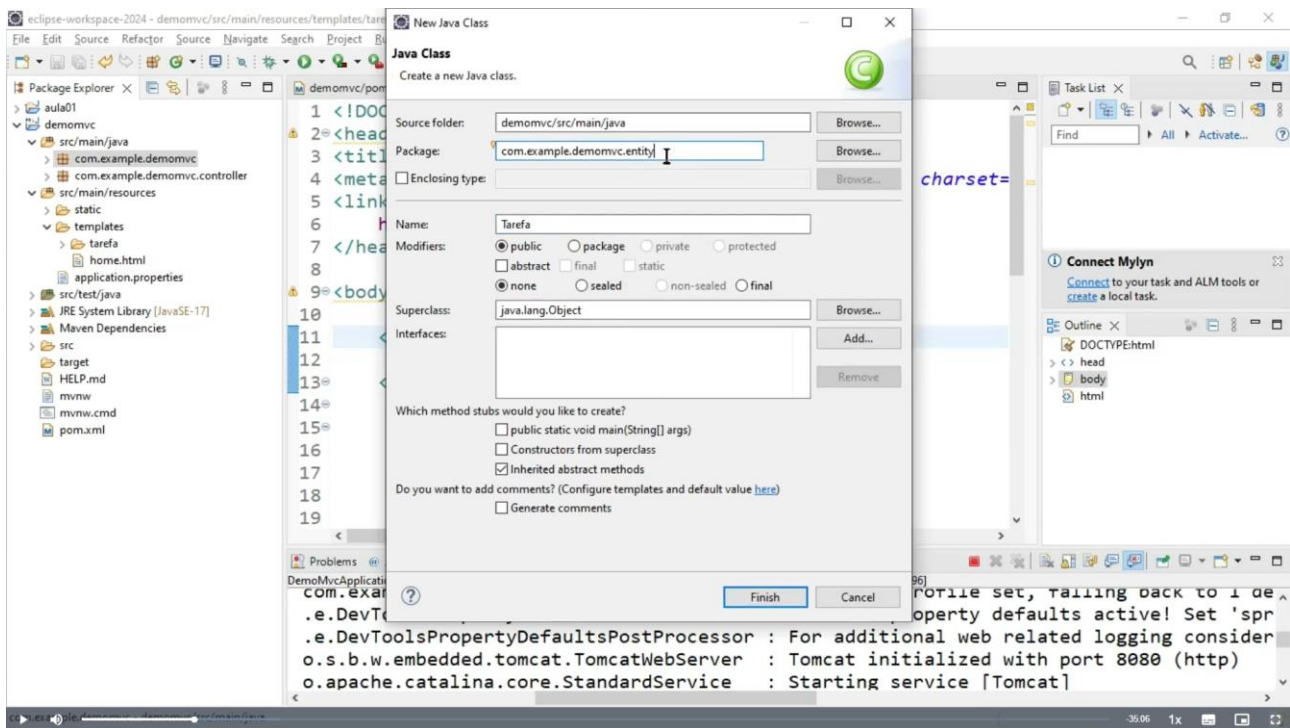
Nessa aula iremos criar os controllers e utilizar algumas expressões do thymeleaf para renderização das páginas html. Nas aulas posteriores continuaremos nesse mesmo projeto. Os arquivos necessário para essa aula estão disponíveis na rota de aprendizagem.

1. Criar pasta tarefa dentro da pasta templates

Dentro da pasta tarefa criar as páginas cadastro e lista — pegar pronta disponibilizada no arquivo_inicio_aula3



2. Criar a classe Tarefa.java na pasta Entity



Colocaremos os atributos id, nome, dataEntrega e responsável.

```
package com.example.demomvc.entity;
import java.time.LocalDate;
import java.util.Objects;
import org.springframework.format.annotation.DateTimeFormat;
public class Tarefa {
    private Long id;
    private String nome;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
    private LocalDate dataEntrega;
    private String responsavel;

    //getters e setters
}
```

As anotações serão abordadas posteriormente, mas já vamos deixar colocado a anotação `DateTimeFormat`.

A anotação `@DateTimeFormat(iso = DateTimeFormat.ISO.DATE)` é usada em aplicações Java, especialmente em ambientes Spring, para especificar o formato de uma data ao realizar a conversão de uma string para um objeto `LocalDate` ou `Date`, e vice-versa.

Gerar getter e setter, construtor, `toString`, `hashCode` e `equals` pela IDE (conforme visto na aula 1).

Nossa classe Tarefa.java tem a seguinte aparência

```
package com.example.demomvc.entity;

import java.time.LocalDate;
import java.util.Objects;

import org.springframework.format.annotation.DateTimeFormat;

public class Tarefa {

    private Long id;
    private String nome;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
    private LocalDate dataEntrega;
    private String responsavel;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public LocalDate getDataEntrega() {
        return dataEntrega;
    }
    public void setDataEntrega(LocalDate dataEntrega) {
        this.dataEntrega = dataEntrega;
    }
    public String getResponsavel() {
        return responsavel;
    }
    public void setResponsavel(String responsavel) {
        this.responsavel = responsavel;
    }
    @Override
    public int hashCode() {
        return Objects.hash(dataEntrega, id, nome, responsavel);
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
```

```

        Tarefa other = (Tarefa) obj;
        return Objects.equals(dataEntrega, other.dataEntrega) &&
Objects.equals(id, other.id)
                && Objects.equals(nome, other.nome) &&
Objects.equals(responsavel, other.responsavel);
    }
    @Override
    public String toString() {
        return "Tarefa [id=" + id + ", nome=" + nome + ",
dataEntrega=" + dataEntrega + ", responsavel=" + responsavel
                + "]";
    }
}

```

3. Modificando página de Cadastro de tarefa

Na tag <form> (página de **cadastro**) vamos adicionar o metodo post e a action para salvar:

```
<form method="POST" action="/tarefas/salvar">
```

nos inputs, verificar se o atributo name está igual ao nome dos atributos que colocamos na tarefa.java

```
<input type="text" class="form-control" id="nome" name =
"nome" >
```

Verificar o atributo name para todos os campos.

A classe de cadastro ficará assim como mostrado abaixo. As marcações amarelas referem-se as modificações que deve ser realizadas na página tarefa.html.

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Cadastro de Tarefas</title>
<!-- <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH"
crossorigin="anonymous">-->
<!-- Bootstrap core CSS -->
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
<link rel="stylesheet" type="text/css" media="all" href="../../css/style.css" />
</head>
<body>

<form method="POST" action="/tarefas/salvar">

```

```

<div class="mb-3">
  <label for="exampleInputTarefa" class="form-label">Nome da Tarefa</label>
  <input type="text" class="form-control" id="nome" name = "nome" >
</div>

<div class="mb-3">
  <label for="exampleInputDataEntrega" class="form-label">Data entrega: </label>
  <input type="date" class="form-control" id="datacriacao" aria-
describedby="dataHelp" name="dataEntrega" >

</div>
<div class="mb-3">
  <label for="exampleInputResponsavel" class="form-label">Responsável: </label>
  <input type="text" class="form-control" id="responsavel" aria-
describedby="responsavelHelp" name="responsavel" >

</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
<div >&copy; 2024 modelo estático</div>
</body>
</html>

```

4. criar a classe TarefaController.java

Na pasta controller vamos criar uma classe chamada TarefaController.java.

criar get para cadastro o post para salvar

vamos colocar a anotação @Controller

vamos colocar a anotação @RequestMapping para mapear uma URL para a classe TarefaController que lidará com requisições HTTP associadas a essa URL.

@GetMapping("/cadastro") indica que o método anotado deve ser executado quando uma requisição HTTP GET for enviada para a URL /cadastro.

@Controller

@RequestMapping("/tarefas")

public class TarefaController {

@GetMapping("/cadastro")

public String cadastro(Tarefa tarefa) {

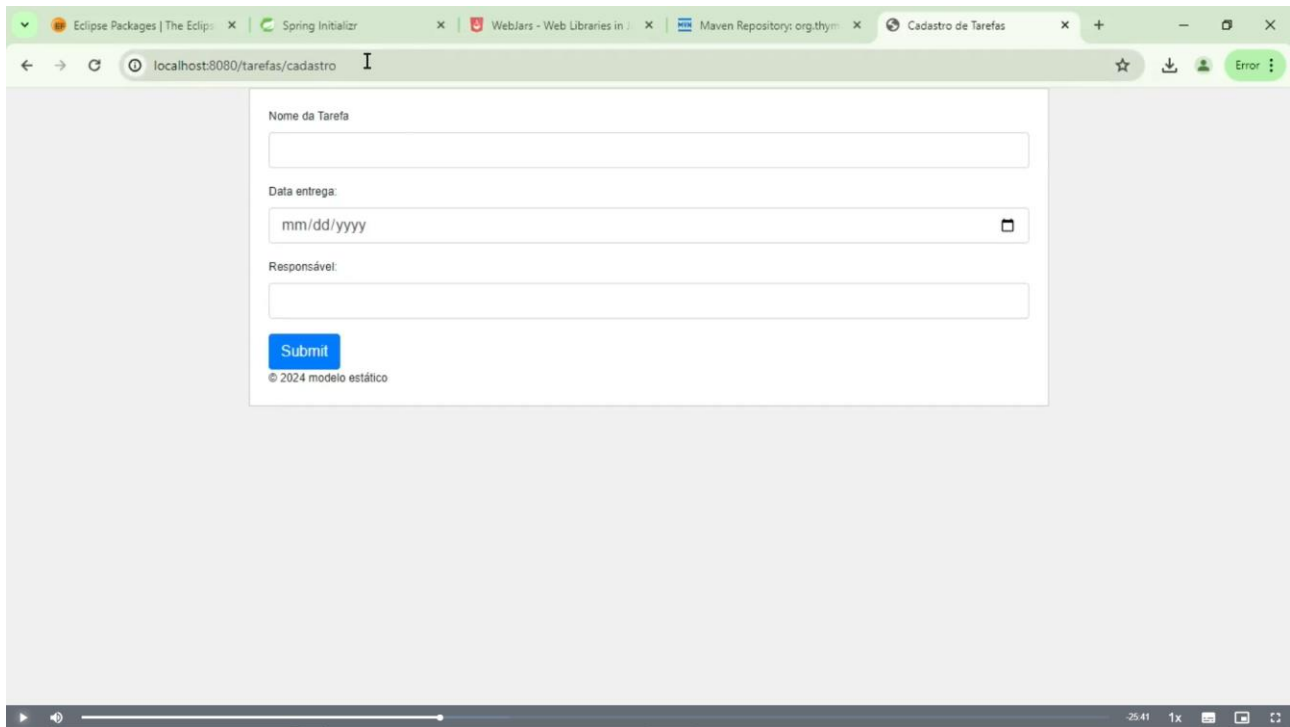
return "/tarefa/cadastro";

}

}

Testar o mapeamento:

<http://localhost:8080/tarefas/cadastro>



O botão ainda não funciona, vamos retornar e implementar o método para salvar.

Adicionar na nossa classe `TarefaController` o método salvar. Usaremos a anotação `@PostMapping("/salvar")`.

`@PostMapping("/salvar")` indica que o método anotado deve ser executado quando uma requisição HTTP POST for enviada para a URL `/salvar`.

Usaremos um `arraylist` para guardar os objetos.

```
@Controller
@RequestMapping("/tarefas")
public class TarefaController {
    List<Tarefa> tarefas = new ArrayList<>();
    @GetMapping("/cadastro")
    public String cadastro(Tarefa tarefa) {
        return "/tarefa/cadastro";
    }
}
```

```
}
```

```
@PostMapping("/salvar")
public String salvar(Tarefa tarefa) {
    Long id = tarefas.size() + 1L; //incrementando o id
    Tarefa t = new Tarefa(); //criando a tarefa t
    t.setId(id);
    t.setNome(tarefa.getNome());
    t.setDataEntrega(tarefa.getDataEntrega());
    t.setResponsavel(tarefa.getResponsavel());
    tarefas.add(t); //adicionando a tarefa à lista tarefas
    return "redirect:/tarefas/lista"; //redireciona para URL
}
```

Setando os atributos

Testar a aplicação

Testar o mapeamento:

<http://localhost:8080/tarefas/cadastro>

fazer um cadastro e clicar no botão submit

Nome da Tarefa
Documento

Data entrega:
12/12/2024

Responsável:
Luciane

Submit

© 2024 modelo estático

Vai dar erro, vamos criar o controller para lista

Criar get para lista

cria agora a listagem

ModelAndView mv = new ModelAndView("tarefa/lista");

Cria uma nova instância de ModelAndView. O parâmetro "tarefa/lista" é o nome da vista (view) que será renderizada, como uma página HTML (por exemplo, tarefa/lista.html ou tarefa/lista.jsp).

```

@GetMapping("/lista")
public ModelAndView lista() {
    ModelAndView mv = new ModelAndView("tarefa/lista"); // Cria uma
    nova instância de ModelAndView. O parâmetro "tarefa/lista" é o nome da
    vista (view) que será renderizada, como uma página HTML (no caso, tarefa/lista.html).

    mv.addObject("tarefas", tarefas); // Adiciona um objeto ao
    modelo

    return mv;
}

```

5. Fazer modificações na lista.html

colocar os thymeleafs

fazer modificações no lista.html e colocar os thymeleafs

modificar dentro do <tbody>

agora com thymeleaf irá ficar assim:

```

<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Gerenciador de Tarefas</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
<link rel="stylesheet" type="text/css" media="all" href="../../css/style.css" />
</head>
<body>
    <h1>Lista de Tarefas</h1>
    <table>
        <thead>
            <tr>
                <th>Identificação da Tarefa</th>
                <th>Nome da Tarefa</th>
                <th>Data entrega</th>
                <th>Responsável</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="tarefa : ${tarefas}">
                <td th:text="${tarefa.id}">1</td>
                <td th:text="${tarefa.nome}">Criação de conteúdo</td>
                <td th:text="${tarefa.dataEntrega}">15/08/2024</td>
                <td th:text="${tarefa.Responsavel}">Luciane</td>
                <td colspan="2">
                    <a class="btn btn-info btn-sm" th:href="@{/tarefas/editar/{id}
                    (id=${tarefa.id})}" role="button">
                        <span class="oi oi-brush" title="Editar" aria-hidden="true"></span>
                    </a>
                    <a class="btn btn-info btn-sm"
                    th:href="@{/tarefas/excluir/{id} (id=${tarefa.id}) }" role="button">
                        <span class="oi oi-brush" title="Excluir" aria-hidden="true"></span>

```



```

                </a></td>
            </tr>
        </tbody>
    </table>
    <p>
        <a href="..">Return to home</a>
    </p>

    <div >&copy; 2024 modelo estático</div>
</body>
</html>

```

fazer a implementação do editar e excluir em tarefa controller ---copiar código disponibilizado no txt

//código do editar e excluir

//editar e excluir

```
@PostMapping("/editar")
```

```
public String editar(Tarefa tarefa) {
```

```
    Tarefa t = new Tarefa();
```

```
    for (int i = 0; i < tarefas.size(); i++) {
```

```
        if (tarefas.get(i).getId().equals(tarefa.getId())) {
```

```
            t = tarefas.get(i);
```

```
        }
```

```
    }
```

```
    tarefas.set(tarefas.indexOf(t), tarefa);
```

```

        return "redirect:/tarefas/lista";
    }

    @GetMapping("/excluir/{id}")
    public String excluir(@PathVariable("id") Long id ) {

        Tarefa tarefa;

        for (int i = 0; i < tarefas.size(); i++) {
            if (tarefas.get(i).getId().equals(id)) {
                tarefa = tarefas.get(i);
                tarefas.remove(tarefa);
            }
        }

        return "redirect:/tarefas/lista";
    }

```

```

    @GetMapping("/editar/{id}")
    public ModelAndView preEditar(@PathVariable("id") Long id) {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("tarefa/cadastro");

        Tarefa tarefa = null;

        for (int i = 0; i < tarefas.size(); i++) {
            if (tarefas.get(i).getId().equals(id)) {
                tarefa = tarefas.get(i);
            }
        }
    }

```

```
        }  
    }  
  
    mv.addObject("tarefa", tarefa);  
    return mv;  
}
```

colocar os thymeleafs no form do cadastro

Vai ficar assim:

tudo que está com th, foi colocado agora

Vamos ter utilizado o th:field:

no pdf está th:field

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Cadastro de Tarefas</title>
<!-- <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH"
crossorigin="anonymous">-->
<!-- Bootstrap core CSS -->
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
<link rel="stylesheet" type="text/css" media="all" href="../../css/style.css" />
</head>
<body>

<form th:action="${tarefa.id == null} ? @{/tarefas/salvar} :
@{/tarefas/editar}" th:object="${tarefa}" method="POST">
<input type="hidden" th:field="*{id}">

<div class="mb-3">
<label for="exampleInputTarefa" class="form-label">Nome da Tarefa</label>
<input type="text" class="form-control" id="nome" name = "nome"
th:field="*{nome}">
</div>

<div class="mb-3">
<label for="exampleInputDataEntrega" class="form-label">Data entrega: </label>
<input type="date" class="form-control" id="datacriacao" aria-
describedby="dataHelp" name="dataEntrega" th:field="*{dataEntrega}">
</div>

<div class="mb-3">
<label for="exampleInputResponsavel" class="form-label">Responsável: </label>
<input type="text" class="form-control" id="responsavel" aria-
describedby="responsavelHelp" name="responsavel" th:field="*{dataEntrega}">
</div>

<button type="submit" class="btn btn-primary">Submit</button>
</form>
```

```
<div th:insert=~{footer::copy}">&copy; 2024 modelo estático</div>  
</body>  
</html>
```

o fim dessa aula 3 está na pasta: projeto_com_pag_html_com_controller_e_thymeleaf

ou aula back-end > aula 3