2
ENCAPSULAMENTO E ABSTRAÇÃO

ROTEIRO

2 Abstração e Encapsulamento

PARTE 1
2.1 Visibilidade e Modificadores de Acesso na POO
2.2 Abstração
2.3 Encapsulamento

PARTE 2
2.4 Criando projeto em Spring Web

PARTE 1

Modificadores, Abstração e Encapsulamento

2.1

Visibilidade e Modificadores de Acesso na POO

Em UML e Java

> *A visibilidade define* ***o nível de acesso*** *que um atributo ou método de uma classe pode ter.*

2.1 Visibilidade em Classe, Atributos e Métodos

Visibilidade na UML

Na UML (Unified Modeling Language), a visibilidade da classe, atributos e métodos define o nível de acesso às informações dentro de uma classe:

- **+ (Público - public)** → Classe, atributo ou método acessível por qualquer classe.
- **- (Privado - private)** → Classe, atributo ou método acessível apenas dentro da própria classe.
- **# (Protegido - protected)** → Classe, atributo ou método acessível dentro da classe e de suas subclasses.
- **~ (Pacote - default/package)** → Classe, atributo ou método acessível apenas dentro do mesmo pacote.

2.1 Visibilidade em Classe, Atributos e Métodos

Exemplo UML:

+ nome: String
- cpf: String
idade: int
~ endereco: String
+ getNome(): String
- validarCPF(): boolean

2.1 Visibilidade em Classe, Atributos e Métodos

Visibilidade em Java

os **modificadores de acesso** definem a visibilidade dos atributos e métodos das classes. São quatro níveis principais:

Modificador	Visibilidade	Acesso permitido
public	Público	Qualquer aula pode ser acessada.
private	Privado	Somente a própria classe pode acessar.
protected	Protegido	A classe e suas subclasses podem acessar.
default	Pacote	Acesso apenas permitido dentro do mesmo pacote.

2.1 Visibilidade em Classe, Atributos e Métodos

Exemplo de código em Java

```
public class ContaBancaria {
    private double saldo; // Apenas a própria classe pode acessar
    public String titular; // Pode ser acessado de qualquer lugar
    protected int agencia; // Pode ser acessado por subclasses

    // Construtor
    public ContaBancaria(String titular, int agencia, double saldo) {
        this.titular = titular;
        this.agencia = agencia;
        this.saldo = saldo;
    }
```

2.1 Visibilidade em Classe, Atributos e Métodos

Exemplo de código em Java

```
// Método público para depositar dinheiro
public void depositar(double valor) {
    if (valor > 0) {
        saldo += valor;
    }
}
```

2.1 Visibilidade em Classe, Atributos e Métodos

Exemplo de código em Java

```
// Método público para sacar dinheiro
public boolean sacar(double valor) {
    if (valor > 0 && valor <= saldo) {
        saldo -= valor;
        return true;
    }
    return false;
}
```

2.1 Visibilidade em Classe, Atributos e Métodos

Exemplo de código em Java

```
    // Método privado para uso interno da classe
    private void atualizarSaldo() {
        saldo *= 1.02; // Simulando atualização de juros
    }
}
```

2.1 Visibilidade em Classe, Atributos e Métodos

Explicação do código

👉 private double saldo: Protege o saldo contra acesso direto.
👉 public String titular: Pode ser acessado por qualquer classe.
👉 protected int agencia: Pode ser acessado por subclasses.
👉 private void atualizarSaldo(): Apenas métodos internos podem ser manipulados.

2.2

Abstração

foca em **esconder detalhes desnecessários** e mostrar apenas funcionalidades essenciais.

> *A **abstração** é um princípio fundamental da Programação Orientada a Objetos que envolve a identificação e a modelagem das características e comportamentos essenciais de um objeto, ignorando os detalhes irrelevantes ou secundários para o contexto em questão..*

2.2 Abstração

Abstração

- É utilizado para a definição de entidades do mundo real. Sendo onde são criadas as classes. Essas entidades são consideradas tudo o que é real, tendo como consideração suas características e ações.

2.2 Abstração

Exemplo em Java

```
public class Conta {
    int numero;
    double saldo;
    double limite;

    void depositar(double valor){
        this.saldo += valor;
    }
```

2.2 Abstração

Exemplo em Java

```
    void sacar(double valor){
        this.saldo -= valor;
    }

    void imprimeExtrato(){
        System.out.println("Saldo: "+this.saldo);
    }
}
```

2.2 Abstração

Exemplo de Abstração em Java:

```
// Classe abstrata que define um modelo para contas bancárias
abstract class Conta {
    protected double saldo;

    public abstract void sacar(double valor);  // Método abstrato

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        }
    }
```

2.2 Abstração

Exemplo de Abstração em Java:

```
    public double getSaldo() {
        return saldo;
    }
}
```

2.2 Abstração

Exemplo de Abstração em Java:

```
// ContaCorrente implementa o comportamento da Conta
class ContaCorrente extends Conta {
    @Override
    public void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
        }
    }
}
```

2.2 Abstração

Utilizando uma classe Abstrata em Java:

```
// Desenvolver a classe main
public class Main2 {
    public static void main(String[] args) {
        // desenvolver
    }
}
```

2.2 Abstração

Benefícios

- **Simplificação:** Reduz a complexidade do desenvolvimento de software, permitindo que os desenvolvedores se concentrem nos aspectos relevantes.
- **Reusabilidade:** Promove a reusabilidade do código, uma vez que as abstrações podem ser usadas em diferentes partes de um aplicativo ou em diferentes projetos.
- **Manutenibilidade:** Facilita a manutenção do código, pois as modificações em uma abstração (como uma classe) são herdadas por todas as suas subclasses.
- **Extensibilidade:** Permite que novas funcionalidades sejam adicionadas com facilidade, estendendo as classes existentes.

2.3

Encapsulamento

esconder os detalhes internos de um objeto e expor **apenas o necessário**.

O Encapsulamento corresponde à uma proteção adicional aos dados de um objeto contra modificações impróprias. Neste sentido, deve-se garantir que modificadores de acesso sejam aplicados adequadamente nas declarações de classes, permitindo visibilidade externa, apenas através de determinados métodos.

2.3 Encapsulamento

- Em um processo de encapsulamento os atributos das classes são do tipo private . Para acessar esses tipos de modificadores, é necessário criar métodos setters e getters.
- A compreensão dos setters de métodos serve para **salvar** (incluir ou alterar) as informações de uma propriedade de um objeto. E os métodos getters para **retornar** o valor dessa propriedade.

2.3 Encapsulamento

Exemplo de getters e setters

Método getters	Método setters

public String getNome() {
  return nome;
}
 	
public double getSalario(){
  return salario;
}
	public void setNome (String nome) {
	  this.nome = nome;
	}
	
	public void setSalario (float salario) {
	  this. salario = salario;
	}

2.3 Encapsulamento

this em Java

- A palavra-chave this é uma referência do próprio objeto em questão. Ela é uma forma de se referir ao objeto em questão sem chama-lo pelo seu nome.

2.3 Encapsulamento

Usos da palavra-chave this em Java

- Referenciar a variável de instância da classe atual
- Invocar o método da classe atual
- Invocar o construtor da classe atual
- Acessar um atributo da instância do objeto que foi criada
- Chamar um construtor dentro de outro construtor na mesma classe

2.3 Encapsulamento

Exemplo de Encapsulamento em Java:

```
public class Carro {
    private String modelo;
    private int ano;
    private double velocidade;

    // Construtor
    public Carro(String modelo, int ano) {
        this.modelo = modelo;
        this.ano = ano;
        this.velocidade = 0;
    }
```

2.3 Encapsulamento

Exemplo de Encapsulamento em Java:

```
// Getter para obter o modelo do carro
public String getModelo() {
    return modelo;
}

// Setter para alterar o modelo do carro
public void setModelo(String modelo) {
    this.modelo = modelo;
}
```

2.3 Encapsulamento

Exemplo de Encapsulamento em Java:

```
// Getter para obter o ano do carro
public int getAno() {
    return ano;
}

// Getter para obter a velocidade
public double getVelocidade() {
    return velocidade;
}
```

2.3 Encapsulamento

Exemplo de Encapsulamento em Java:

```
    // Método para acelerar o carro
    public void acelerar(double incremento) {
        if (incremento > 0) {
            velocidade += incremento;
        }
    }
}
```

2.3 Encapsulamento

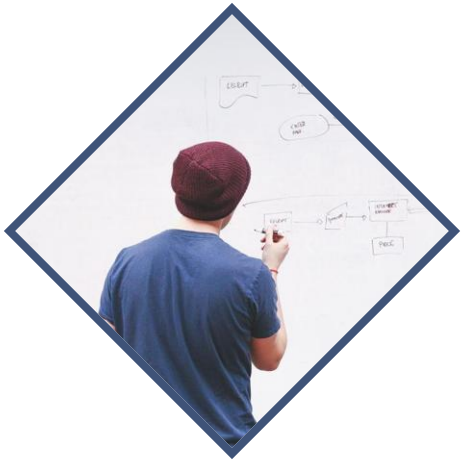
Explicação do código

- **private String modelo:** Protege o modelo do carro.
- **public String getModelo():** Permite acessar o modelo de forma segura.
- **public void setModelo(String modelo):** Permite alterar o modelo com validação.
- **public void acelerar(double incremento):** Método público para modificar velocidade sem exportar diretamente a variável.

2.3 Encapsulamento

Benefícios

- **Segurança:** Protege os atributos da classe contra acessos indevidos.
- **Facilidade de manutenção:** Permite mudanças internas sem afetar quem usa a classe.
- **Modularidade:** Melhora a organização e reutilização do código.



Desafio

Desenvolver em Java

Sistema de Gerenciamento de Funcionários

Desafio

Contextualização

1. Uma empresa de tecnologia chamada TechSoft deseja desenvolver um Sistema de Gerenciamento de Funcionários para organizar informações como nome, cargo e salário de seus colaboradores. Para garantir a segurança dos dados e um melhor controle sobre as informações, a empresa quer que o sistema siga os princípios de Encapsulamento e Abstração.

   Como desenvolvedor(a), você precisa projetar e implementar uma solução utilizando os conceitos de modificadores de acesso (public, private, protected), garantindo que os dados sensíveis dos funcionários sejam protegidos e acessados corretamente.

Desafio

Objetivo do Desafio

1. **Encapsular os atributos da classe** Funcionario para proteger informações sensíveis.
2. Criar métodos públicos (getters e setters) para manipular os dados de forma segura.
3. Utilizar **abstração** para definir um modelo genérico de funcionário, permitindo reutilização e expansão do código.
4. Criar uma classe principal para testar a implementação.

Desafio

Regras e Requisitos do Sistema

1. A empresa possui diferentes tipos de funcionários (exemplo: Desenvolvedor, Gerente).
2. Cada funcionário tem os seguintes atributos:
3. nome (público, pode ser acessado por qualquer classe).
4. cargo (público, pode ser acessado por qualquer classe).
5. salario (privado, apenas a classe pode modificar e acessar).
6. O salário não pode ser acessado diretamente fora da classe Funcionario. Ele deve ser manipulado usando métodos (getSalario() e setSalario()).
7. Criar uma classe Funcionario, Desenvolvedor e Gerente que herdam de Funcionario.
8. Criar um método calcularBonus() para cada cargo.

PARTE 2

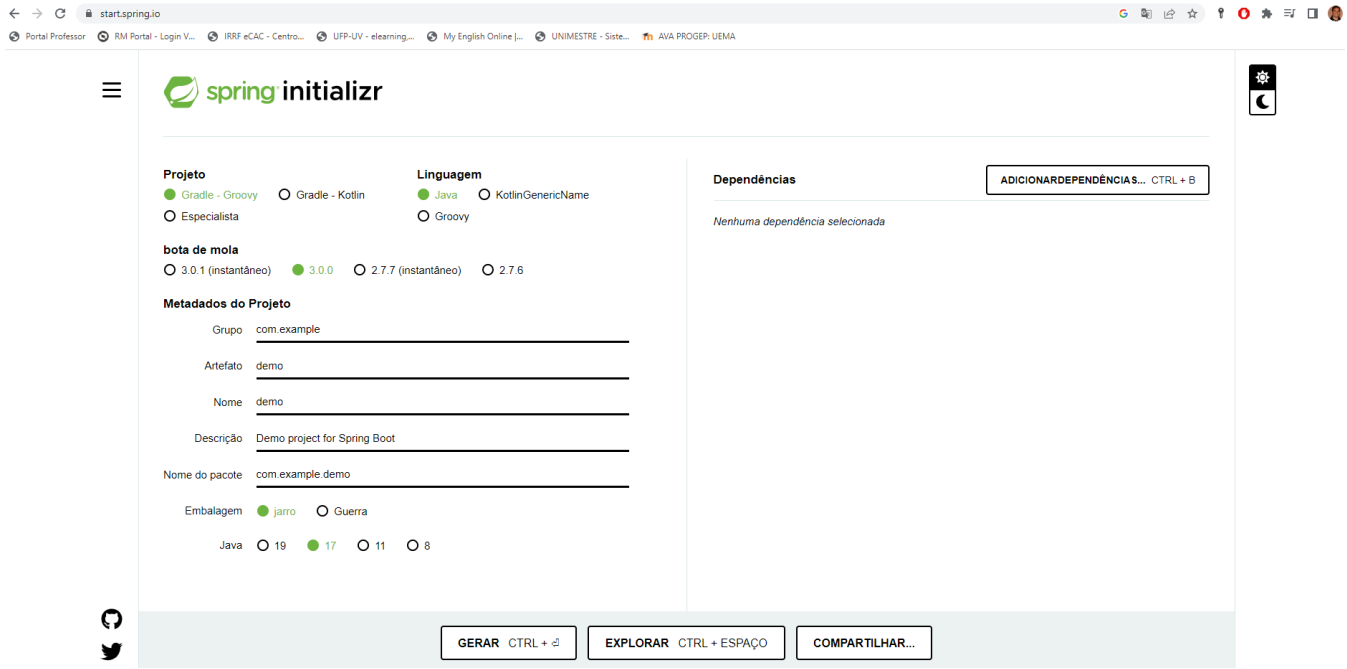
Criando um projeto Spring Boot Web

2.4

Criando projeto em Spring Web

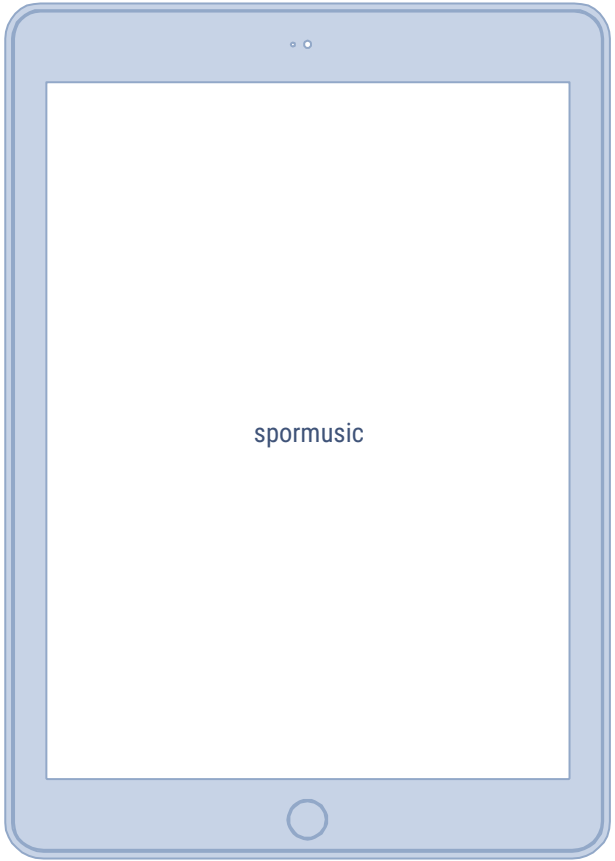
spotmusic

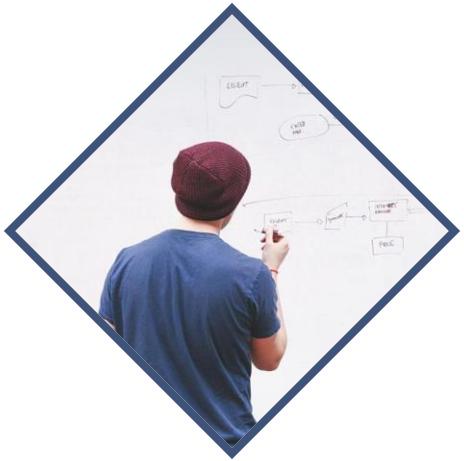
2.4 Criando projeto em Spring Web



pom.xml

mostrar o projeto



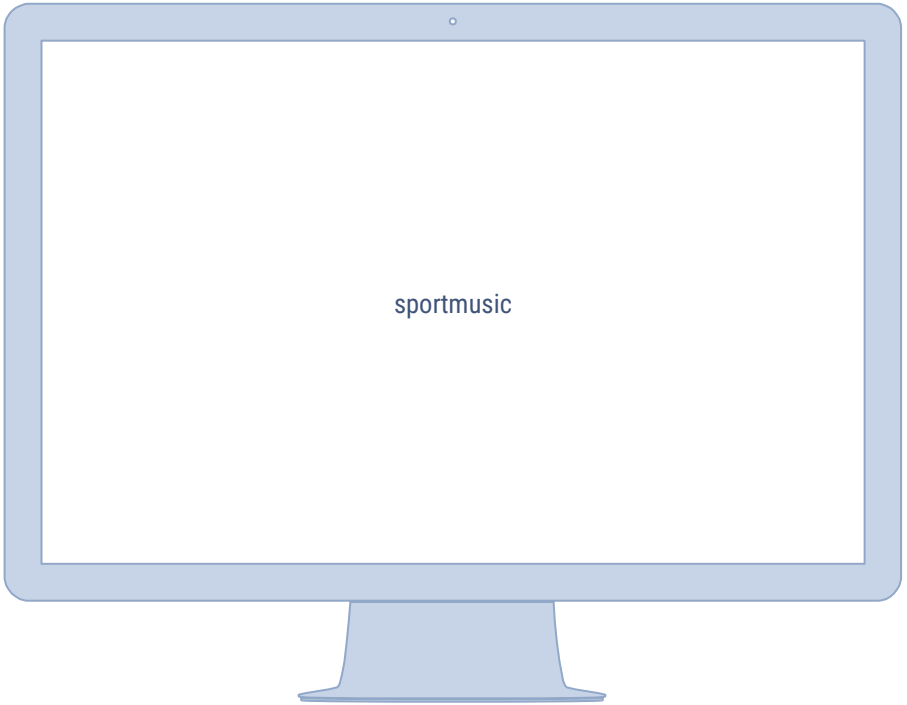


pom.xml

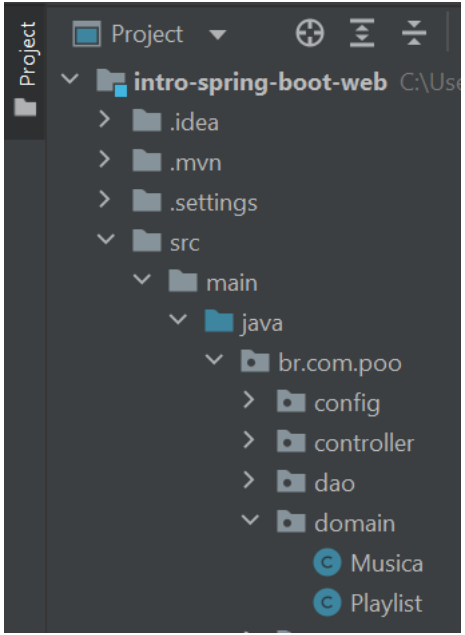
mostrar
projeto

Programando as classes de domínio PROJECT

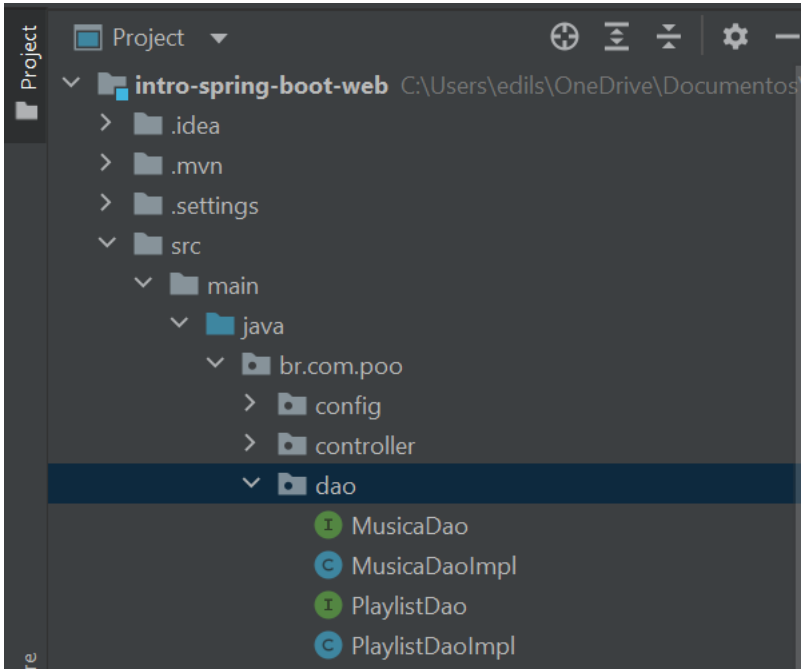
Conheça a estrutura inicial do projeto.



2.4 Criando projeto em Spring Web



2.4 Criando projeto em Spring Web



2.4 Criando projeto em Spring Web

PlaylistDao.java

```java
package br.com.poo.dao;

import br.com.poo.domain.Playlist;

import java.util.List;

public interface PlaylistDao {

    void salvar(Playlist playlist);
    List<Playlist> recuperar();
    Playlist recuperarPorID(long id);
    void atualizar(Playlist playlist);
    void excluir(long id);

}
```

2.4 Criando projeto em Spring Web

PlaylistDaoImpl.java

```java
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;

@Repository
public class PlaylistDaoImpl implements PlaylistDao {

    @PersistenceContext
    private EntityManager em;

    @Override
    public void salvar(Playlist playlist) {
        em.persist(playlist);
    }

    @Override
    public List<Playlist> recuperar() {
        return em.createQuery("select p from Playlist p", Playlist.class).getResultList();
    }
```

2.4 Criando projeto em Spring Web

MusicaDao.java

```java
package br.com.poo.dao;

import br.com.poo.domain.Musica;

import java.util.List;

public interface MusicaDao {

    void salvar(Musica musica);
    List<Musica> recuperarPorPlaylist(long playlistId);
    Musica recuperarPorPlaylistIdEMusicaId(long playlistId, long musicaId);
    void atualizar(Musica musica);
    void excluir(long musicaId);

}
```

2.4 Criando projeto em Spring Web

MusicaDaoImpl.java

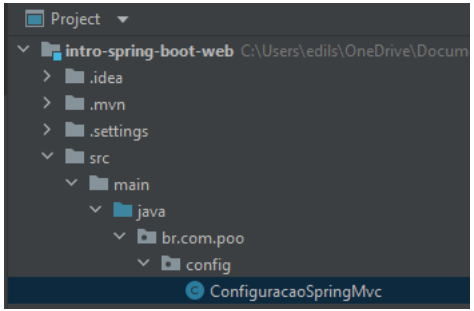
```
6   import javax.persistence.EntityManager;
7   import javax.persistence.PersistenceContext;
8   import java.util.List;
9
10  @Repository
11  public class MusicaDaoImpl implements MusicaDao {
12
13      @PersistenceContext
14      private EntityManager em;
15
16      @Override
17      public void salvar(Musica musica) { em.persist(musica); }
20
21      @Override
22      public List<Musica> recuperarPorPlaylist(long playlistId) {
23          return em.createQuery( s: "select m from Musica m where m.playlist.id = :playlistId", Musica.class)
24                  .setParameter( s: "playlistId", playlistId)
25                  .getResultList();
26      }
```

2.4 Criando projeto em Spring Web

home.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, user-scalable=no"/>

    <link href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css" rel="stylesheet"/>
    <link href="/css/style.css" rel="stylesheet"/>
</head>

<body>
<div class="container">
    <div class="jumbotron">
        <h1>Spotmusic</h1>
        <p class="lead">
            Crie playlists e avalie músicas
        </p>
    </div>
```

2.4 Criando projeto em Spring Web

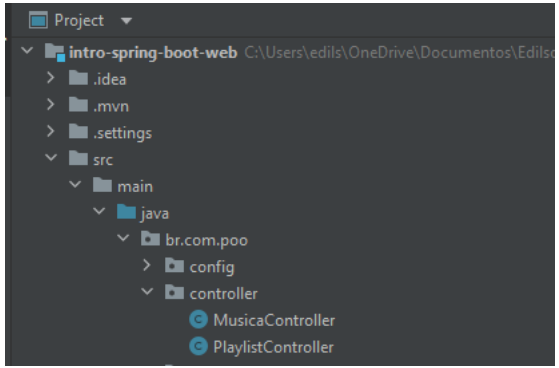


2.4 Criando projeto em Spring Web

SpotmusicApplication.java

```java
package br.com.poo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpotmusicApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpotmusicApplication.class, args);
    }

}
```

2.4 Criando projeto em Spring Web



2.4 Criando projeto em Spring Web

PlaylistController.java

```java
package br.com.poo.controller;

import br.com.poo.domain.Playlist;
import br.com.poo.service.PlaylistService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.validation.Valid;

@Controller
@RequestMapping("playlists")
public class PlaylistController {

    @Autowired
    private PlaylistService playlistService;

```

2.4 Criando projeto em Spring Web

```
28      @GetMapping("/cadastro")
29      public String preSalvar(@ModelAttribute("playlist") Playlist playlist) {
30          return "/playlist/add";
31      }
32
33      @PostMapping("/salvar")
34  @   public String salvar(@Valid @ModelAttribute("playlist") Playlist playlist, BindingResult result, RedirectAttributes attr) {
35          if (result.hasErrors()) {
36              return "/playlist/add";
37          }
38
39          playlistService.salvar(playlist);
40          attr.addFlashAttribute( attributeName: "mensagem",  attributeValue: "Playlist criada com sucesso.");
41          return "redirect:/playlists/listar";
42      }
```

2.4 Criando projeto em Spring Web

add.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Spotmusic</title>

    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, user-scalable=no"/>

    <link href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css" rel="stylesheet"/>
    <link href="/css/style.css" rel="stylesheet"/>
</head>
<body>

<div class="container">
    <div class="jumbotron">
        <h1>Spotmusic</h1>
    </div>

    <h4>
        Nova playlist
    </h4>
```

2.4 Criando projeto em Spring Web

```
44      @GetMapping("/{id}/atualizar")
45  @   public ModelAndView preAtualizar(@PathVariable("id") long id, ModelMap model) {
46          Playlist playlist = playlistService.recuperarPorId(id);
47          model.addAttribute( attributeName: "playlist", playlist);
48          return new ModelAndView( viewName: "/playlist/add", model);
49      }
50
51      @PutMapping("/salvar")
52  @   public String atualizar(@Valid @ModelAttribute("playlist") Playlist playlist, BindingResult result, RedirectAttributes attr) {
53          if (result.hasErrors()) {
54              return "/playlist/add";
55          }
56
57          playlistService.atualizar(playlist);
58          attr.addFlashAttribute( attributeName: "mensagem",  attributeValue: "Playlist atualizada com sucesso.");
59          return "redirect:/playlists/listar";
60      }
61
62      @GetMapping("/{id}/remover")
63  @   public String remover(@PathVariable("id") long id, RedirectAttributes attr) {
64          playlistService.excluir(id);
65          attr.addFlashAttribute( attributeName: "mensagem",  attributeValue: "Playlist excluída com sucesso.");
66          return "redirect:/playlists/listar";
67      }
```

2.4 Criando projeto em Spring Web

```
<table class="table">
    <thead>
    <tr>
        <th>Nome</th>
        <th>Descrição</th>
        <th>Ação</th>
    </tr>
    </thead>
    <tr th:each="playlist : ${playlists}">
        <td >
            <a th:text="${playlist.nome}" th:href="@{/playlists/{id}/musicas/listar(id=${playlist.id})}">nome</a>
        </td>
        <td th:text="${playlist.descricao}">descricao</td>
        <td>
            <a class="btn btn-sm btn-info" th:href="@{/playlists/{id}/atualizar(id=${playlist.id})}">Editar</a>
            <a class="btn btn-sm btn-danger" th:href="@{/playlists/{id}/remover(id=${playlist.id})}">Excluir</a>
        </td>
    </tr>
</table>
```

2.4 Criando projeto em Spring Web

```java
@Controller
@RequestMapping("playlists/{playlistId}/musicas")
public class MusicaController {

    @Autowired
    private MusicaService musicaService;

    @GetMapping("/listar")
    public ModelAndView listar(@PathVariable("playlistId") long playlistId, ModelMap model) {
        model.addAttribute( attributeName: "musicas", musicaService.recuperarPorPlaylist(playlistId));
        model.addAttribute( attributeName: "playlistId", playlistId);
        return new ModelAndView( viewName: "/musica/list", model);
    }

    @GetMapping("/cadastro")
    public String preSalvar(@ModelAttribute("musica") Musica musica, @PathVariable("playlistId") long playlistId) {
        return "/musica/add";
    }
```

2.4 Criando projeto em Spring Web

```
62      @GetMapping("/{id}/remover")
63  @   public String remover(@PathVariable("id") long id, RedirectAttributes attr) {
64          playlistService.excluir(id);
65          attr.addFlashAttribute( attributeName: "mensagem",  attributeValue: "Playlist excluída com sucesso.");
66          return "redirect:/playlists/listar";
67      }
```

2.4 Criando projeto em Spring Web

```
<tr th:each="playlist : ${playlists}">
    <td >
        <a th:text="${playlist.nome}" th:href="@{/playlists/{id}/musicas/listar(id=${playlist.id})}">nome</a>
    </td>
    <td th:text="${playlist.descricao}">descricao</td>
    <td>
        <a class="btn btn-sm btn-info" th:href="@{/playlists/{id}/atualizar(id=${playlist.id})}">Editar</a>
        <a class="btn btn-sm btn-danger" th:href="@{/playlists/{id}/remover(id=${playlist.id})}">Excluir</a>
    </td>
</tr>
```

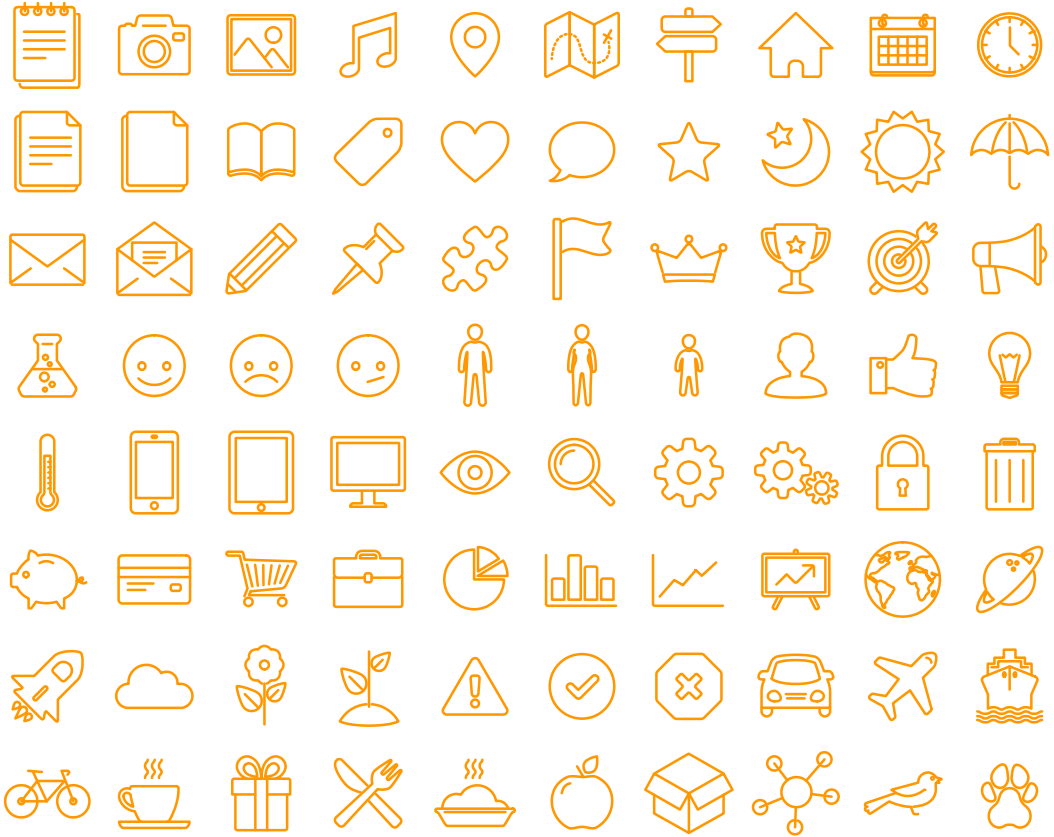
BIBLIOGRAFIA

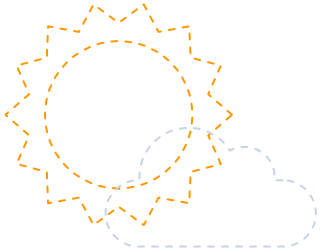
- ARNOLD, K.; GOSLING, J.; HOLMES, D. **A Linguagem de Programação Java**. 4. ed. Porto Alegre: Bookman, 2007.
- DEITEL, H. M. **Java: como programar**. 6. ed. São Paulo: Pearson Education do Brasil, 2005.
- SANTOS, R. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Campus, 2003.
- SIERRA, K.; BATES, B. **Use a cabeça: Java. 2**. ed. Rio de Janeiro: Alta Books, 2005.

OBRIGADO!

Perguntas?

edilson3lima@gmail.com
(98) 9 8806-8505





Vamos lá...

até a próxima aula...