

Tópicos Avançados de Programação

Genilson Medeiros

Professor do curso de Sistemas de Informação
Mestre em Ciência e Tecnologia em Saúde
Engenheiro de Software



Sobre a disciplina



- O conceito de Tópicos Avançados de Programação é muito amplo.
- O professor que ministra (área de pesquisa e experiência dele)
- As tendências tecnológicas do momento;
- O foco do curso (mais acadêmico ou mais aplicado)

A disciplina busca



- Explorar conceitos e técnicas avançadas de programação que vão além do básico e intermediário;
- Apresentar novas linguagens, paradigmas ou **frameworks**;
- Trabalhar boas práticas e padrões de projeto em contextos complexos;
- Integrar ferramentas modernas usadas no mercado e na pesquisa.

Metodologia



- Projetos práticos: os alunos desenvolvem um sistema ou protótipo usando as tecnologias estudadas.
- Seminários e apresentações: cada aluno ou grupo pesquisa um tema avançado e apresenta para a turma.
- Estudo de casos: análise de soluções reais de mercado.
- Estaremos, em sua grande parte, trabalhando com Spring / Spring Boot.

Avaliações



- Por unidade teremos um seminário (30%) e projeto prático (70%).
- Cada entrega será feita em grupo (máx. 4 alunos por grupo).
- O tempo da apresentação será definido após a formação dos grupos.
- Com o andamento da disciplina, (quase) **tudo pode ser revisto** e podemos conduzir diferente.

Conceitos básicos



A principal linguagem de programação que utilizaremos na disciplina é Java (amem ou odeiem :P)

Quanto a IDE, estaremos utilizando o IntelliJ IDEA versão community (mas, caso você prefira, pode usar outra, por sua conta).

Esteja com a IDE instalada até a próxima aula.

Java



JDK, JVM e JRE - O trio do Java



JDK – Java Development Kit

É o **kit de desenvolvimento** Java.

Contém:

- O compilador (javac) para transformar código .java em .class
- Ferramentas de desenvolvimento
- A **JVM**

Ou seja, **se você quer programar em Java, precisa instalar o JDK.**

JDK, JVM e JRE - O trio do Java



JVM – Java Virtual Machine

- É a **máquina virtual** que roda os programas Java.
- Quando você compila um programa em Java, ele vira um **bytecode** (`.class`), que **não é código nativo** da máquina.
- A **JVM interpreta ou compila** esse bytecode em tempo de execução, adaptando para o sistema operacional onde está rodando. Isso permite que o mesmo `.class` rode no Windows, Linux, Mac, etc.

JDK, JVM e JRE - O trio do Java



JRE – Java Runtime Environment

- Contém a JVM + bibliotecas necessárias para executar aplicações Java.
- Se você só quer rodar programas Java (e não programar), o JRE é suficiente.
- **OBS: A partir do Java 11, a Oracle integrou o JRE dentro do JDK.**

Motivação Inicial



Motivação Inicial



Na programação, a gente se depara com muitas preocupações, como configuração de ambiente, padronizações, integração, segurança... além de toda a lógica em si a ser desenvolvida.

Motivação Inicial



Tudo isso implica em custo, em tempo, em dor de cabeça, etc.

É comum que sejam utilizados frameworks para ajudar os devs com muitas dessas tarefas.

Motivação Inicial



Alguns exemplos desses frameworks:

- **React Native** (mobile c/ Javascript e React)
- **Ionic** (mobile, híbrido c/ JS, HTML, CSS e Angular)
- **TensorFlow** (para aprendizagem de máquina)
- **Bootstrap** (compon. front-end c/ HTML, CSS e JS)

Motivação Inicial



Nesta disciplina, estaremos utilizando o framework **Spring**, principalmente o que se refere ao **Spring Boot**



Spring Framework



O Spring Framework



Register for SpringOne at Explore 2025!

spring® by VMware Tanzu

Why Spring ▾ Learn ▾ Projects ▾ Academy ▾ Community ▾ Tanzu Spring

All projects >

- Spring Boot
- Spring Framework
- > **Spring Data**
- > **Spring Cloud**
- Spring Cloud Data Flow
- > **Spring Security**
- Spring Authorization Server
- Spring for GraphQL
- > **Spring Session**
- Spring Integration
- Spring HATEOAS
- Spring Modulith
- Spring REST Docs
- Spring AI
- Spring Batch
- Spring AMQP
- Spring CredHub
- Spring for Apache Kafka

Spring Boot 3.5.4

OVERVIEW LEARN SUPPORT SAMPLES

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

If you're looking for information about a specific version, or instructions about how to upgrade from an earlier release, check out [the project release notes section](#) on our wiki.

Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

O Spring Framework



Em novembro de 2002, Rod Johnson publicou pela editora Wrox, o livro: Expert One-on-One J2EE Design and Development. O objetivo de seu trabalho foi:

- 1) apontar o quanto complexo se tornava o desenvolvimento e manutenção da maioria das aplicações corporativas quando se seguia ortodoxamente o guia do J2EE; e
- 2) apresentar uma alternativa que fosse mais simples, menos custosa, com tempo de entrega de mercado mais realista, mais gerenciável e com melhor performance.

O Spring Framework



Acompanhavam o texto mais de 30.000 linhas de código em Java de um framework inicialmente batizado como Interface21 e futuramente renomeado como Spring Framework.

Muitos dos conceitos e mecanismos fundamentais do Spring já estavam nessas linhas, a saber: um container de IoC – Inversion of Control, ou Inversão de Controle – com um BeanFactory, um ApplicationContext e um DI, ou Dependency Injection – Injeção de Dependência – capaz e funcional

Os primórdios do Spring MVC com Controller, HandlerMapping e associados, o JdbcTemplate e o conceito de acesso a dados agnóstico à tecnologia

O Spring Framework



Spring é muito baseada em dois conceitos, chamados de **inversão de controle** e **injeção de dependência**.

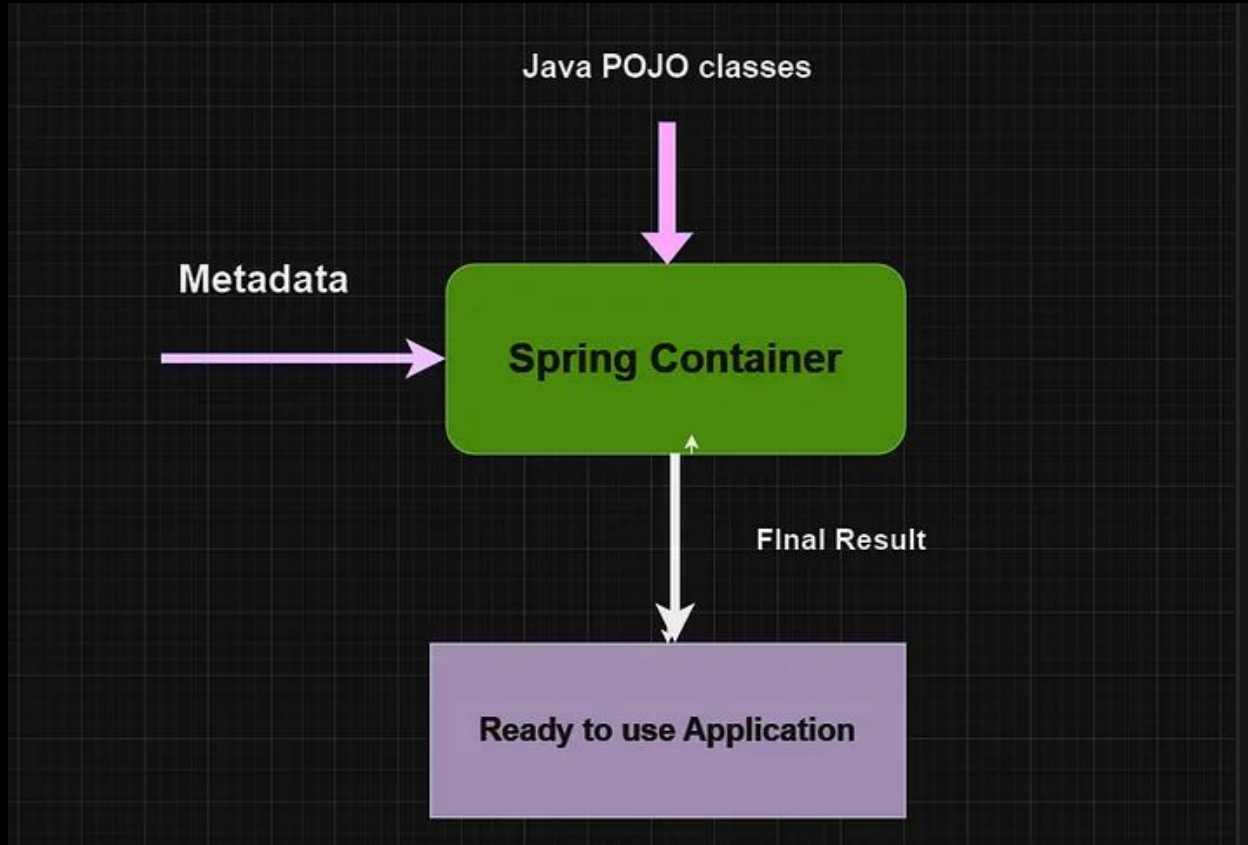
O Spring Framework



Inversão de controle (inversion of control) repassa o controle de objetos para um framework.

No modelo tradicional, nosso código controla o fluxo de controle (ex. chamada a função). Com inversão de controle, frameworks externos controlam o fluxo (ex. chamada ao nosso código).

O Spring Framework



O Spring Framework



Injeção de dependência (dependency injection) indica que um objeto recebe outros objetos ou funções, ao invés deles mesmos criarem esses objetos ou funções (ex. são injetadas neles).

É daí que vem a ideia de “dependências” (delas que o objeto recebe outros objetos e funcionalidades).

O Spring Framework



Injeção de Dependência (ID) é uma técnica específica de Inversão de Controle (IoC). IoC é um princípio de design amplo, enquanto ID é uma maneira de aplicar esse princípio, onde um objeto recebe suas dependências de fontes externas ao invés de criá-las internamente.

O Spring Framework



Inversão de Controle (IoC)	Injeção de Dependência (ID)
IoC inverte o controle sobre a criação e gerenciamento de objetos, tirando essa responsabilidade da classe e passando para um componente externo (container ou framework).	Uma técnica específica de IoC onde as dependências de uma classe são fornecidas por meio de construtores, setters ou interfaces.
Torna o código mais flexível, testável e com menor acoplamento, permitindo que diferentes implementações de dependências sejam usadas sem modificar a classe que as utiliza.	Facilita a substituição de implementações de dependências, melhora a testabilidade e reduz o acoplamento entre classes.
Exemplo: Um framework IoC gerencia o ciclo de vida de objetos e suas dependências, injetando-os em outras classes quando necessário.	Exemplo: Uma classe <code>Servico</code> que usa um objeto <code>Repositorio</code> pode receber o <code>Repositorio</code> através do construtor, em vez de criar uma instância diretamente dentro de <code>Servico</code> .

Quais as diferenças entre esses dois códigos?



```
public class Loja {  
    private Item item;  
  
    public Loja() {  
        item = new Comida();  
    }  
}
```

```
public class Loja {  
    private Item item;  
  
    public Loja(Item item) {  
        this.item = item;  
    }  
}
```

O Spring Framework



Características:

A classe **Loja** cria diretamente a instância de **Comida**.

Forte **acoplamento**: **Loja** depende concretamente de **Comida**.

Difícil de testar: não dá para trocar **Comida** por outro tipo de **Item** (ex.: **Roupa**) sem modificar o código da classe.

● Viola o princípio da inversão de dependência (**D** do SOLID), pois a classe de alto nível (**Loja**) depende de uma classe de baixo nível (**Comida**) diretamente.

```
public class Loja {  
    private Item item;  
  
    public Loja() {  
        item = new Comida();  
    }  
}
```

O Spring Framework



Características:

A classe não cria o objeto Item; ela apenas o recebe de fora.

Baixo acoplamento: Loja só conhece a interface/abstração Item.

```
public class Loja {  
    private Item item;  
  
    public Loja(Item item) {  
        this.item = item;  
    }  
}
```

Fácil de testar: é possível passar qualquer implementação de Item (ex.: new Roupa(), new Comida(), ou um mock de teste).

Segue o princípio de inversão de dependência (DIP), pois Loja depende de uma abstração (Item), não de uma implementação.

Facilita Injeção de Dependência — o framework (Spring, por exemplo) ou código cliente é responsável por fornecer a implementação.

● Bom para IoC/DI: o controle da criação do objeto está “invertido” — a classe não cria suas dependências, elas são injetadas.

Exemplo



1ª versão (não recomendada para IoC/DI)

```
@Component
public class Loja {
    private Item item;

    public Loja() {
        // Aqui a Loja decide a implementação
        this.item = new Comida();
    }

    public void vender() {
        item.usar();
    }
}
```

```
@Component
public class Comida implements Item {
    @Override
    public void usar() {
        System.out.println("Comendo...");
    }
}
```

Exemplo



1ª versão (não recomendada para IoC/DI)

```
@Component
public class Loja {
    private Item item;

    public Loja() {
        // Aqui a Loja decide a implementação
        this.item = new Comida();
    }

    public void vender() {
        item.usar();
    }
}
```

```
@Component
public class Comida implements Item {
    @Override
    public void usar() {
        System.out.println("Comendo...");
    }
}
```

Se você rodar isso com Spring, **mesmo que tenha várias implementações de Item**, o Spring não consegue trocar a implementação porque o `new Comida()` está fixo no código. Isso **quebra o propósito** de IoC — o controle continua dentro da própria classe.

Exemplo



2ª versão (correta com IoC/DI)

```
@Component
public class Loja {
    private final Item item;

    // Injeção via construtor (recomendada)
    @Autowired
    public Loja(Item item) {
        this.item = item;
    }

    public void vender() {
        item.usar();
    }
}
```

```
@Component
public class Comida implements Item {
    @Override
    public void usar() {
        System.out.println("Comendo...");
    }
}
```

```
@Component
public class Roupa implements Item {
    @Override
    public void usar() {
        System.out.println("Vestindo...");
    }
}
```

Exemplo



2ª versão (correta com IoC/DI)

Como o Spring faz a mágica

No segundo exemplo:

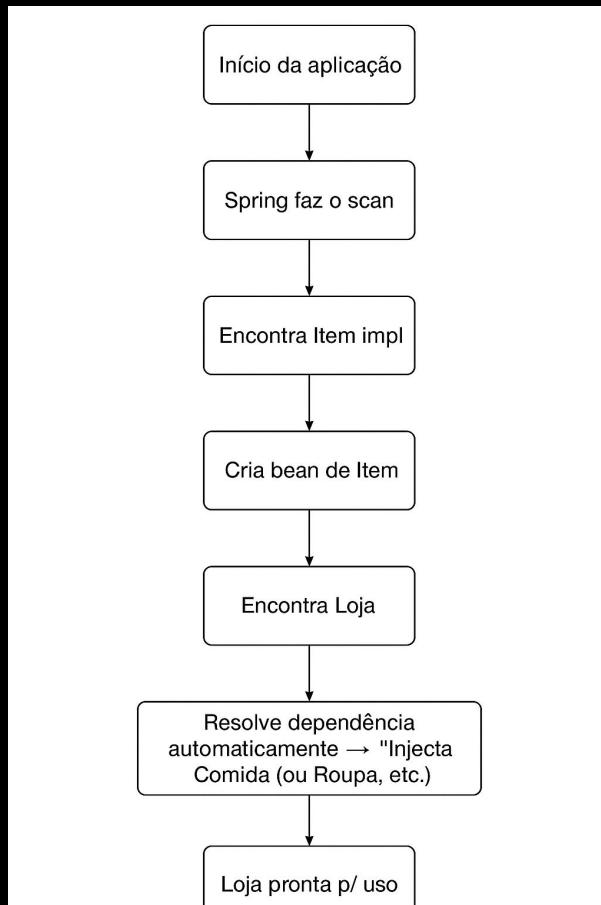
- O Spring vê que Loja precisa de um Item.
- Ele procura um *bean* no contexto que implemente Item.
- Ele cria e injeta essa instância automaticamente, sem você precisar dar new.

📌 Trocar implementação é simples:

```
@Configuration
public class AppConfig {

    @Bean
    public Item item() {
        return new Roupa(); // agora Loja vende roupa, sem mudar o código da Loja
    }
}
```


Fluxo



<-- Procura por `@Component`, `@Bean`, `@Configuration`...


<-- Ex.: Comida (implementa Item)



<-- Guarda no `ApplicationContext`

<-- Loja precisa de Item no construtor

Spring initializr



 **spring initializr**

Project
☐ Gradle - Groovy
☐ Gradle - Kotlin
☒ Maven

Language
☒ Java ☐ Kotlin
☐ Groovy

Spring Boot
☐ 4.0.0 (SNAPSHOT) ☐ 4.0.0 (M1)
☐ 3.5.5 (SNAPSHOT) ☒ 3.5.4
☐ 3.4.9 (SNAPSHOT) ☐ 3.4.8

Project Metadata

Group

Artifact

Name


Description

Package name

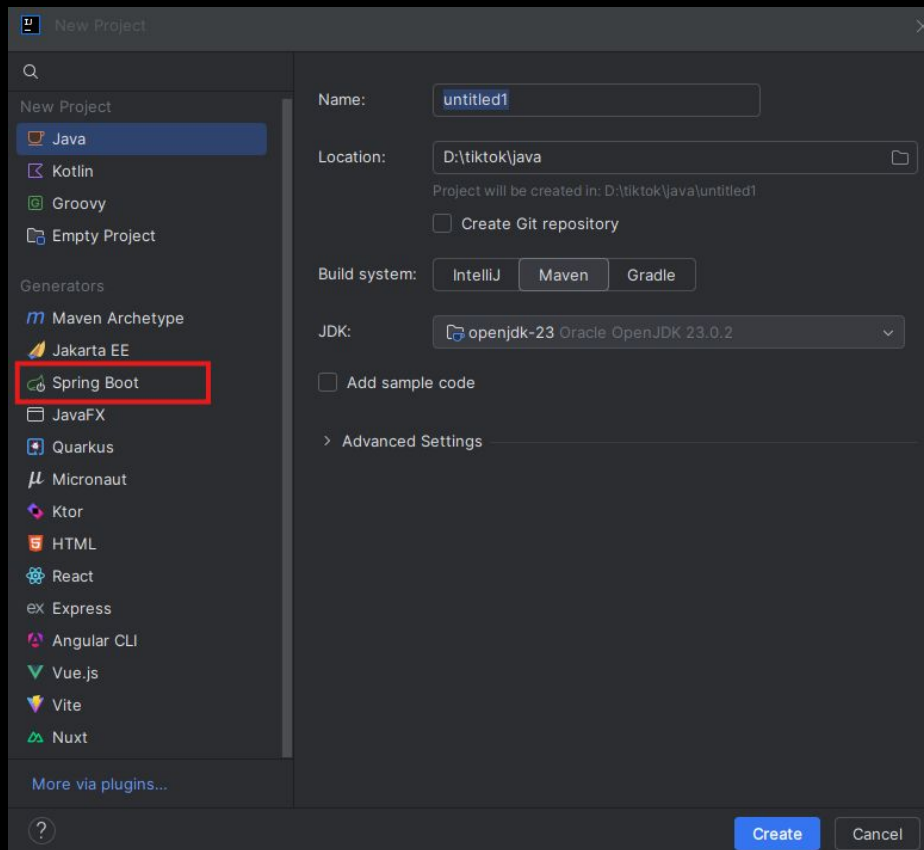
Packaging ☒ Jar ☐ War

Dependencies ADD ... CTRL + B

No dependency selected

 GENERATE CTRL + G EXPLORE CTRL + SPACE ...

IntelliJ IDEA Ultimate



Eclipse



Spring Tools

Spring Tools is the next generation of Spring tooling for your favorite coding environment. It provides world-class support for developing Spring-based enterprise applications, whether you prefer Eclipse, Visual Studio Code, or Theia IDE.

Spring Tools for Eclipse

Free. Open source.

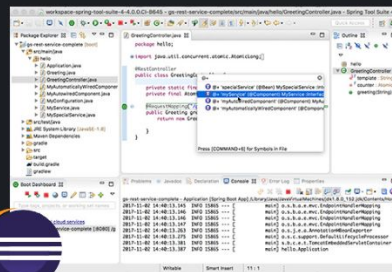
4.31.0 - LINUX X86_64

4.31.0 - LINUX ARM_64

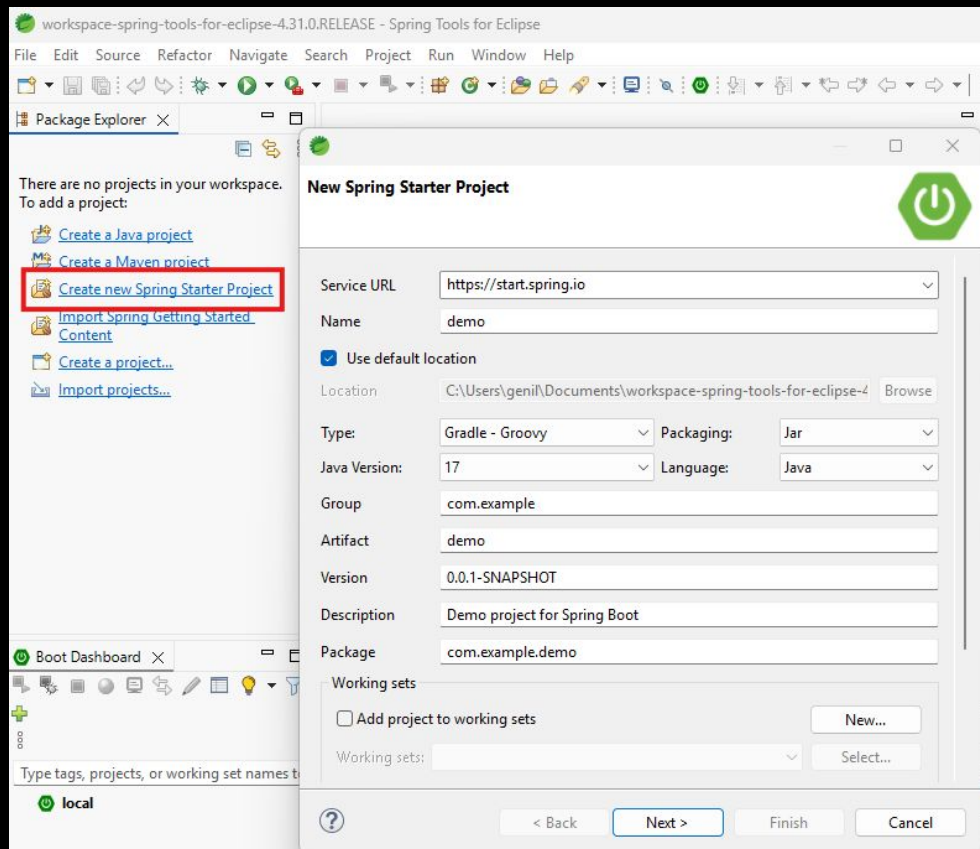
4.31.0 - MACOS X86_64

4.31.0 - MACOS ARM_64

4.31.0 - WINDOWS X86_64



Eclipse



Principais annotations do Spring Boot



Configuração e Inicialização

- **@SpringBootApplication** → Combina **@Configuration**, **@EnableAutoConfiguration** e **@ComponentScan**. Marca a classe principal da aplicação.
- **@Configuration** → Define uma classe de configuração de beans.
- **@Bean** → Registra um bean manualmente no contexto Spring.

Principais annotations do Spring Boot



Injeção de Dependência

- **@Component** → Marca uma classe para ser gerenciada pelo Spring.
- **@Service** → Versão especializada de @Component para classes de serviço.
- **@Repository** → Versão especializada de @Component para classes de persistência; trata exceções de banco.
- **@Controller** → Versão especializada de @Component para controlar requisições web.
@RestController → Combina @Controller + @ResponseBody (retorna JSON/XML).
- **@Autowired** → Injeta dependências automaticamente.
- **@Qualifier** → Especifica qual bean injetar quando há múltiplas implementações.
- **@Primary** → Marca um bean como a escolha padrão na injeção.

Principais annotations do Spring Boot



Web e REST

- **@RequestMapping** → Mapeia URLs para métodos ou classes.
- **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping** → Mapeiam métodos HTTP específicos.
- **@PathVariable** → Captura valores de variáveis na URL.
- **@RequestParam** → Captura parâmetros de query string.
- **@RequestBody** → Recebe o corpo da requisição e converte para objeto Java.
- **@ResponseBody** → Retorna o valor do método diretamente no corpo da resposta.

Principais annotations do Spring Boot



Banco de Dados / JPA

- **@Entity** → Marca uma classe como entidade JPA.
- **@Id** → Marca o campo como chave primária.
- **@GeneratedValue** → Define a estratégia de geração da chave primária.
- **@Table** → Define o nome da tabela no banco.
- **@Column** → Configura o mapeamento de colunas.
- **@OneToMany**, **@ManyToOne**, **@OneToOne**, **@ManyToMany** → Mapeiam relacionamentos.

Resumo



CONFIGURATION AND INITIALIZATION

@SpringBootApplication

@Configuration

@Bean

DEPENDENCY INJECTION

@Component

@Service

@Repository

@Controller

@RestController

@Autowired

@Qualifier

@Primary

WEB AND REST

@RequestMapping

@GetMapping

@PutMapping

@PutMapping

@DeleteMapping

@PathVariable

@RequestParam

@RequestBody

DATABASE / JPA

@Entity

@Id

@GeneratedValue

@Table

@Column

@Column

@OneToMany

@ManyToOne

@OneToOne

@ManyToMany

Principat notations do Spring Boot



Conceitos importantes

Exemplo: Injeção de dependência



O Spring disponibiliza muitos recursos prontos e facilmente configuráveis, chamados de **starters**.

Exemplo: Injeção de dependência



Com o Spring Boot, podemos automaticamente incluir uma ou algumas das dependências necessárias.

Exemplo: Injeção de dependência



Para gerenciamento das dependências, pode-se utilizar ferramentas como [Maven](#), da Apache.

Exemplo: Injeção de dependência



Maven se caracteriza, dentre outras coisas, pela utilização de um modelo de objeto de projeto (POM), que concentra informações do projeto em um XML (e.g. dependências, plugins, informações gerais).

Exemplo: Injeção de dependência



Nós estaremos desenvolvendo serviços Web com o estilo de arquitetura chamado REST (Representational State Transfer).

Isso permite sistemas diferentes se comunicarem de forma eficiente e simples, c/ uso de métodos HTTP

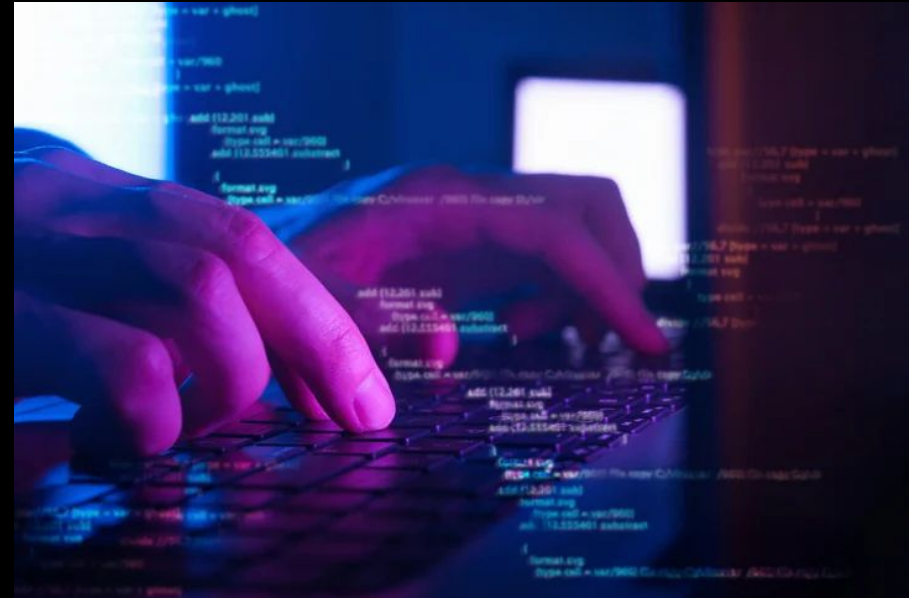
Exemplo: Injeção de dependência



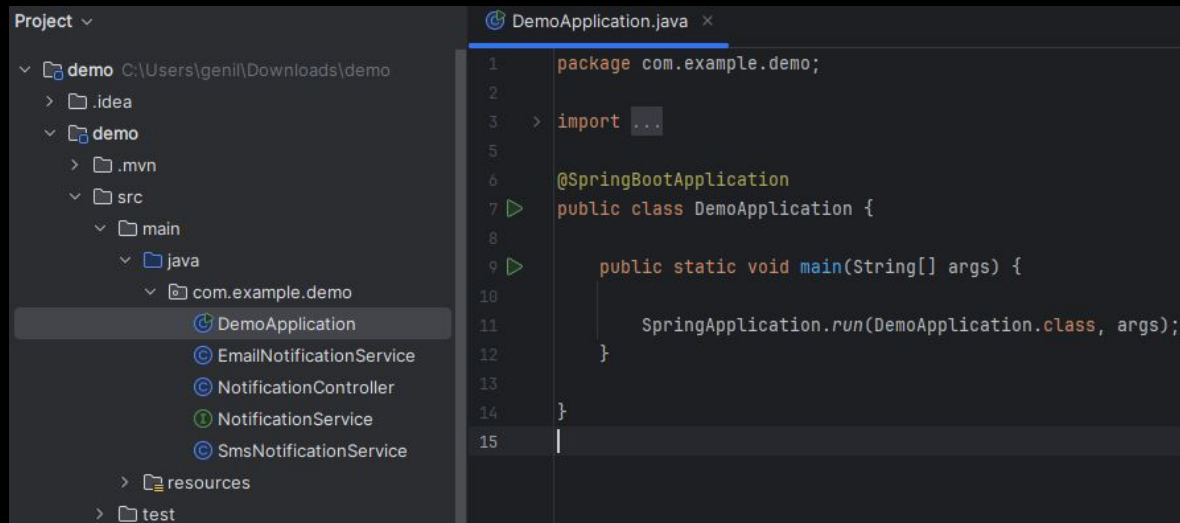
Os métodos HTTP lidam com recursos, de modo que:

- GET recupera representação do recurso;
- POST cria um recurso novo;
- PUT atualiza um recurso já existente;
- DELETE deleta um recurso.

Exemplo



Exemplo: Injeção de dependência



```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```

Exemplo: Injeção de dependência



```
NotificationService.java ×  
1 package com.example.demo;  
2  
3 public interface NotificationService { 2 usages 2 implementations  
4     void sendNotification(String message); 2 usages 2 implementations  
5 }
```

Exemplo: Injeção de dependência



```
© EmailNotificationService.java x
1  package com.example.demo;
2
3  import org.springframework.stereotype.Service;
4
5  @Service 2 usages
6  public class EmailNotificationService implements NotificationService {
7      @Override 2 usages
8      public void sendNotification(String message) {
9          System.out.println("Sending email notification: " + message);
10     }
11 }
```

Exemplo: Injeção de dependência



```
SmsNotificationService.java ×  
1 package com.example.demo;  
2  
3 import org.springframework.stereotype.Service;  
4  
5 @Service 2 usages  
6 public class SmsNotificationService implements NotificationService{  
7     @Override 2 usages  
8     public void sendNotification(String message) {  
9         System.out.println("Sending SMS notification: " + message);  
10    }  
11 }
```

Exemplo: Injeção de dependência



```
NotificationController.java ×
1 package com.example.demo;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestParam;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController no usages
9 public class NotificationController {
10     private final EmailNotificationService emailNotificationService; 2 usages
11     private final SmsNotificationService smsNotificationService; 2 usages
12
13     @Autowired no usages
14     public NotificationController(EmailNotificationService emailNotificationService,
15                                 SmsNotificationService smsNotificationService) {
16         this.emailNotificationService = emailNotificationService;
17         this.smsNotificationService = smsNotificationService;
18     }
19
20     @GetMapping("/sendEmail") no usages
21     public String sendEmail(@RequestParam String message) {
22         emailNotificationService.sendNotification(message);
23         return "Email notification sent!";
24     }
25
26     @GetMapping("/sendSms") no usages
27     public String sendSms(@RequestParam String message) {
28         smsNotificationService.sendNotification(message);
29         return "SMS notification sent!";
30     }
31 }
```

[localhost:8080/sendEmail?
message=ola turma de TAP](http://localhost:8080/sendEmail?message=ola turma de TAP)

[localhost:8080/sendSms
?message=Hello via SMS](http://localhost:8080/sendSms?message=Hello via SMS)