### Exercício 10.2 – Injeção de Dependências por Setter

**Objetivo:** Demonstrar a injeção de dependências por setter no Spring Boot. Esta abordagem permite configurar dependências após a criação de objetos, sendo útil em situações específicas.

### Parte 1: Configuração do Projeto no Spring Initializr

1. Aceder A [Spring Initializr](https://start.spring.io/).
2. Configure o projeto com as seguintes opções:
   * **Project:** Maven
   * **Language:** Java
   * **Spring Boot:** Versão estável mais recente
   * **Group:** com.example
   * **Artifact:** demo
   * **Name:** demo
   * **Package Name:** com.example.demo
   * **Packaging:** Jar
   * **Java Version:** 21 (ou outra versão suportada pela sua máquina)
3. Adicione as seguintes dependências:
   * **Spring Web:** Para criar endpoints REST.
   * **Spring Boot DevTools (opcional):** Para facilitar o desenvolvimento com recarregamento automático.
   * **Spring Boot Actuator (opcional):** Para monitorizar a aplicação (não necessário para este exercício).
4. Clique no botão **Generate** para descarregar o projeto como um ficheiro ZIP.
5. Extraia o ficheiro ZIP e abra o projeto no VS Code.

### Parte 2: Estrutura Inicial do Projeto

Após abrir o projeto, a estrutura inicial será:

src/  
├── main/  
│ ├── java/  
│ │ └── com.example.demo/  
│ │ ├── DemoApplication.java  
└── resources/  
 └── application.properties

**Nota:** As pastas service e controller não são criadas automaticamente. Devem ser criadas manualmente.

### Parte 3: Criar a Estrutura de Pastas

1. **Na IDE:**
   * Dentro da pasta com.example.demo, crie duas subpastas:
     + service
     + controller
2. Após criar estas pastas, a estrutura será:

src/  
├── main/  
│ ├── java/  
│ │ └── com.example.demo/  
│ │ ├── DemoApplication.java  
│ │ ├── service/  
│ │ └── controller/  
└── resources/  
 └── application.properties

### Parte 4: Implementação do Serviço de Email

1. Na pasta service, crie o ficheiro ServicoEmail.java.
2. Insira o seguinte código:

package com.example.demo.service;  
  
import org.springframework.stereotype.Component;  
  
@Component  
public class ServicoEmail {  
  
 public void enviarEmail(String destinatario, String mensagem) {  
 System.out.println("Enviando e-mail para " + destinatario + ": " + mensagem);  
 }  
}

**Explicação:**

* A anotação @Component declara esta classe como um bean gerido pelo Spring.
* O método enviarEmail simula o envio de e-mails.

### Parte 5: Implementação do Serviço de Notificação

1. Na pasta service, crie o ficheiro ServicoNotificacao.java.
2. Insira o seguinte código:

package com.example.demo.service;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
@Component  
public class ServicoNotificacao {  
  
 private ServicoEmail servicoEmail;  
  
 @Autowired  
 public void setServicoEmail(ServicoEmail servicoEmail) {  
 this.servicoEmail = servicoEmail;  
 }  
  
 public void notificar(String destinatario, String mensagem) {  
 servicoEmail.enviarEmail(destinatario, mensagem);  
 }  
}

**Explicação:**

* A anotação @Autowired realiza a injeção da dependência através do método setter.
* O Spring chama o método setServicoEmail automaticamente para injetar o bean ServicoEmail.

### Parte 6: Criar o Controlador REST

1. Na pasta controller, crie o ficheiro NotificacaoController.java.
2. Insira o seguinte código:

package com.example.demo.controller;  
  
import com.example.demo.service.ServicoNotificacao;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class NotificacaoController {  
  
 private ServicoNotificacao servicoNotificacao;  
  
 @Autowired  
 public void setServicoNotificacao(ServicoNotificacao servicoNotificacao) {  
 this.servicoNotificacao = servicoNotificacao;  
 }  
  
 @GetMapping("/notificar")  
 public String notificar(@RequestParam String destinatario, @RequestParam String mensagem) {  
 servicoNotificacao.notificar(destinatario, mensagem);  
 return "Notificação enviada para " + destinatario;  
 }  
}

**Explicação:**

* @RestController define esta classe como um controlador REST.
* O endpoint /notificar expõe a funcionalidade de enviar notificações.

### Parte 7: Configuração do Projeto

1. No ficheiro application.properties (em src/main/resources), adicione:

logging.level.org.springframework=INFO

1. Certifique-se de que o ficheiro DemoApplication.java contém:

package com.example.demo;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class DemoApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(DemoApplication.class, args);  
 }  
}

### Parte 8: Testar o Projeto

**1. Iniciar o Spring Boot:**

* + Execute o projeto na sua IDE ou com o comando:
  + ./mvnw spring-boot:run
  + ou (Mac OS/Linux):
  + mvn spring-boot:run,

**2. Aceder o Endpoint:**

No navegador ou Postman, aceda a:

<http://localhost:8080/notificar?destinatario=joao@email.com&mensagem=Bem-vindo>!

**3. Verificar a Saída no Terminal:**

O terminal deverá exibir:

Enviando e-mail para joao@email.com: Bem-vindo!

O brower deverá exibir:

Notificação enviada para joao@email.com

### Reflexão sobre o Exercício

**Vantagens da Injeção por Setter:**

* Útil para dependências opcionais.
* Métodos setter tornam o código mais legível.

**Limitações:**

* Campos devem ser mutáveis, o que compromete a imutabilidade.
* Dependências obrigatórias não são explicitamente garantidas.

**Práticas Recomendadas:**

* Prefira a injeção por construtor para dependências obrigatórias.