Entendendo Interfaces, Injeção de Dependência e Spring Web para o Desenvolvimento Eficiente de Software

Interfaces, Injeção de Dependência e Spring Web: O que significam e por que são importantes?

Se você não é um desenvolvedor de software, termos ou expressões como “interfaces”, “injeção de dependência” e “Spring Web” podem soar como uma língua estrangeira. Mas não se preocupe, pois serão explicadas para você em termos simples.

Interfaces

As interfaces são como contratos entre dois objetos em um produto de software. Elas definem o que um objeto pode esperar do outro. Pense nisto como um acordo entre duas partes, onde cada parte sabe exatamente o que esperar da outra. As interfaces permitem que os desenvolvedores projetem sistemas com componentes de baixo acoplamento, o que reduz a interdependência entre as classes e facilita a modificação ou substituição de um componente sem afetar os demais. As interfaces também promovem a modularidade e a reusabilidade dos componentes de uma aplicação, o que agiliza muito a conclusão e entrega de produtos.

Injeção de Dependência

A injeção de dependência é um padrão de projeto que permite a inversão de controle, onde os objetos não são responsáveis pela criação de suas dependências, pois as recebem de uma fonte externa. Esta fonte externa pode ser um recipiente ou estrutura de injeção de dependência, que gerencia a criação dos objetos necessários a uma aplicação e os disponibiliza no momento que serão utilizados. Assim como as interfaces, a injeção de dependência torna o código mais modular, testável e manejável, separando a criação de objetos de seu uso, reduzindo o acoplamento entre classes e garantindo um melhor gerenciamento de memória.

Spring Web

Agora, vamos falar sobre o Spring Web. Trata-se de uma estrutura, popularmente

denominada *framework*, que os desenvolvedores usam para construir aplicações web em Java. Esse *framework* fornece um conjunto de recursos que ajudam os desenvolvedores a lidar com solicitações e respostas, gerenciar sessões e suportar diferentes tecnologias de visualização gráfica. O Spring Web também oferece suporte a interfaces e injeção de dependência e o faz por meio de um contêiner de controle, o ApplicationContext, que é o responsável pela criação e gerenciamento dos objetos que compõem a aplicação. Os desenvolvedores podem definir interfaces para serviços, repositórios, entre outros componentes, e o ApplicationContext se encarrega de injetar a implementação apropriada da interface em cada objeto consumidor de serviço ou funcionalidade. Isto, da mesma forma, resulta em códigos modulares, de fácil sustentabilidade e realização de testes mais rápidos e eficientes.

Então, por que interfaces, injeção de dependência e Spring Web são importantes?

Estes conceitos são essenciais não apenas para a construção de software de fácil atualização, manutenção e escalabilidade, mas também para reduzir custos e agilizar o processo desenvolvimento de um produto, seja ele um sistema, aplicativo móvel ou outro qualquer. As interfaces permitem que empresas de desenvolvimento projetem sistemas com componentes de baixo acoplamento, o que reduz significativamente a quantidade de tempo e recursos necessários para se fazer mudanças no código. A injeção de dependência permite que os objetos recebam suas dependências de uma fonte externa, o que facilita a escrita de códigos modulares e testáveis, reduzindo a necessidade de testes extensos e demorados. E o framework Spring Web é quem orquestra a criação de interfaces e opera a injeção de dependências, facilitando a construção de aplicações capazes de lidar com grandes quantidades de tráfego e usuários, reduzindo, inclusive, os custos de infra-estrutura. Além disso, essas tecnologias integradas minimizam problemas como a rotatividade de mão de obra de desenvolvedores, por exemplo, pois outros desenvolvedores podem facilmente entender o código e assumir o trabalho iniciado pelos desenvolvedores anteriores.

Em suma, interfaces, injeção de dependência e Spring Web surgiram como componentes-chave para o desenvolvimento de produtos de software modernos e competitivos, pois fornecem soluções eficientes, de fácil sustentabilidade e escaláveis, capazes de se adaptarem às crescentes necessidades e exigências de qualidade por parte dos usuários.

Sessão Prática

O jogo Movies Battle é uma aplicação web simples que exibe dois filmes ao competidor, que deve acertar qual deles possui a melhor avaliação no IMDB.

Nesta sessão prática, veremos trechos de códigos utilizados pelo jogo, que demonstram a integração de interfaces e injeção de dependência, gerenciadas por recursos do *framework* Spring Web.

Vamos criar interfaces para definir o contrato entre dois componentes da aplicação, usar a injeção de dependência para gerenciar as dependências e o framework Spring Web para lidar com as requisições HTTP.

MovieRepository

O código abaixo cria um repositório para a entidade Movie, que estende o JpaRepository para herdar os métodos básicos do CRUD.

```
@Repository
public interface MovieRepository extends JpaRepository<Movie, Long> {
}
```

MovieController

O código abaixo cria um controlador (controller) para a entidade Movie, com dois *end points*: um para obter um filme por sua identificação e outro para obter todos os filmes. O *controller* depende de um serviço, o MovieService, para recuperar os filmes, que é injetado usando a injeção do construtor.

```
@RestController
@RequestMapping("/movies")
public class MovieController {
    private final MovieService movieService;
```

```
    @Autowired
    public MovieController(MovieService movieService) {
        this.movieService = movieService;
    }

    @GetMapping("/{id}")
    public ResponseEntity<DetailedMovie> getMovie(@PathVariable Long id) {
        DetailedMovie movie = movieService.getMovie(id);
        if (movie == null) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(movie);
    }

    @GetMapping
    public List<SimpleMovie> getAllMovies() {
        return movieService.getAllMovies();
    }
}
```

MovieInterface

As interfaces abaixo definem os contratos para os métodos SimpleMovie e DetailedMovie, que serão implementados pela classe Movie.

```
public interface SimpleMovie {
    String getName();
    String getImageUrl();
}

public interface DetailedMovie extends SimpleMovie {
    Long getId();
    String getTitle();
    Double getImdbScore();
}
```

Movie

O código abaixo define a entidade Movie, que implementa ambas as interfaces SimpleMovie e DetailedMovie. A entidade é anotada com @Entity para indicar que é uma entidade JPA.

```
@Entity
public class Movie implements SimpleMovie, DetailedMovie {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String imageUrl;
    private Double imdbScore;

    // constructors, getters and setters (prefer Lombok)
    @Override
    public String getName() {
        return title;
    }

    @Override
    public Long getId() {
        return id;
    }

    @Override
    public String getTitle() {
        return title;
    }

    @Override
    public String getImageUrl() {
```

```
        return imageUrl;
    }

    @Override
    public Double getImdbScore() {
        return imdbScore;
    }
}
```

MovieService

O código abaixo define o serviço MovieService, que depende do MovieRepository para a recuperação de filmes. O serviço oferece dois métodos de recuperação de filmes: um para obter um filme específico por ID e o outro para obter todos os filmes. O serviço também fornece um método de recuperação de dois filmes aleatórios, que serão exibidos ao jogados se atenderem as regras de negócio definidas para o jogo. O serviço é anotado com @Service para indicar que é um serviço Spring e a dependência do MovieRepository é injetada.

```
@Service
public class MovieService {
    private final MovieRepository movieRepository;

    @Autowired
    public MovieService(MovieRepository movieRepository) {
        this.movieRepository = movieRepository;
    }

    public DetailedMovie getMovie(Long id) {
        Optional<Movie> optionalMovie = movieRepository.findById(id);
        return optionalMovie.map(movie -> (DetailedMovie) movie).orElse(null);
    }

    public List<SimpleMovie> getAllMovies() {
        List<Movie> movies = movieRepository.findAll();
```

```
        return movies.stream()
                .map(movie -> (SimpleMovie) movie)
                .collect(Collectors.toList());
    }

    public List<Movie> getTwoRandomMovies() {
        List<Movie> movies = movieRepository.findAll();
        Collections.shuffle(movies);
        return movies.stream()
                .limit(2)
                .collect(Collectors.toList());
    }
}
```

Embora a funcionalidade do jogo pareça simples à primeira vista, sua implementação é complexa, pois envolve muitos aspectos, como modelagem de dados, projeto de banco de dados, tratamento de erros, segurança, otimização de desempenho e testes. Os trechos de códigos aqui apresentados destinam-se tão somente a exemplificar como interfaces e injeção de dependência são tratadas pelo framework Spring WEB, sem jamais representar uma solução completa e operável.

Por fim, é importante esclarecer que, para garantir um produto de software bem sucedido no mercado, de fácil manutenção e escalável, sua construção requer atenção para inúmeras outras questões. Daí a necessidade de planejamento sobre todos os aspectos do ciclo de vida do desenvolvimento de um produto de software.