CURSO EBAC - Especialista Back-End Java

Módulo 16: Padrões de projetos – Parte 2

Aluno: Sávio Emerick

Injeção de Dependência:

Injeção de dependências é um padrão de projeto que tem como objetivo aumentar a coesão dos programas ao mesmo tempo que diminui seu acoplamento com suas dependências. Quando o código depende diretamente de objetos concretos, qualquer mudança nesses objetos pode ser custosa, pois o programa fica restrito a uma implementação específica.

Exemplo sem injeção de dependências:

```
public class OrderService {

    private OrderRepository orderRepository = new
OrderRepository();


    public void placeOrder(Order order) {

        orderRepository.save(order);

    }

}
```

Nesse exemplo, a classe **OrderService** cria uma instância de **OrderRepository** diretamente. Se quisermos mudar a implementação de **OrderRepository**, teríamos que modificar **OrderService**.

O padrão de injeção de dependências substitui o uso direto de objetos concretos pelo uso de abstrações através de interfaces. Isso significa que, em vez de uma classe instanciar diretamente suas dependências, ela recebe essas dependências de um cliente externo que injeta os objetos necessários. Este processo é uma forma de inversão de controle, onde a responsabilidade de criar e fornecer dependências é transferida para outro componente.

Exemplo com injeção de dependências:

```
public class OrderService {
    private OrderRepository orderRepository;

    public OrderService(OrderRepository orderRepository) {
        this.orderRepository = orderRepository;
    }
    public void placeOrder(Order order) {
        orderRepository.save(order);
    }
}
```

Agora, **OrderService** recebe uma instância de **OrderRepository** através do construtor, permitindo que diferentes implementações de **OrderRepository** sejam injetadas conforme necessário.

No Spring, o controle das dependências e sua injeção é gerenciado pelo próprio framework, ao invés de ser feito manualmente pelo cliente. O Spring faz isso através da conteinerização de beans e componentes, que são definidos e gerenciados através de anotações no código, como **@Component**, **@Autowired**, e outras. Isso facilita a configuração e gerenciamento das dependências, promovendo um código mais modular e flexível.

Exemplo com Spring:

```
@Component
public class OrderRepository {
    public void save(Order order) {
        // implementação para salvar o pedido
    }
}

@Service
public class OrderService {
    private final OrderRepository orderRepository;

    @Autowired
    public OrderService(OrderRepository orderRepository) {
        this.orderRepository = orderRepository;
    }

    public void placeOrder(Order order) {
        orderRepository.save(order);
    }
}
```

Neste exemplo, **OrderRepository** e **OrderService** são anotados com **@Component** e **@Service**, respectivamente, indicando ao Spring que eles são beans gerenciados. A anotação **@Autowired** no construtor de **OrderService** instrui o Spring a injetar uma instância de **OrderRepository** automaticamente, eliminando a necessidade de instanciar **OrderRepository** manualmente.