Anotações Spring Annotations

Inversão de controle:

É um padrão de projeto, é algo abstrato onde a gente define todas as dependências de um determinado objeto sem a necessidade de criar (gerenciar), pois passamos esse papel de gerenciamento para o framework no caso o Spring para o seu Core, então é o Core do Spring que vai ter toda essa responsabilidade de gerenciar todas essas dependências conforme necessidade.

E como o Spring faz isso então? Ele utiliza de uma implementação, a forma concreta que é a **injeção de dependências**, então a**injeção de dependências** é a implementação concreta dainversão de controle, então assim ele consegue gerenciar todos esses Beans que são os objetos que vamos criando conforme vamos construindo as nossas aplicações ou vamos utilizando de bibliotecas externas por exemplo, vamos definindo esses Beans e assim o Spring ele vai cuidar das instâncias de todos esses objetos Java.

Tudo que envolve a base do Spring está contido no seu Core que fica dentro do projeto Spring Framework, que é aonde já traz então várias configurações prontas, muitas delas a gente já consegue usar de forma bem mais simples, a possibilidade de customizar e de criar todas as configurações necessárias para nossa aplicação.

Principais Anotações:

Stereotype:

As anotações de Stereotypes servem para nos mostrarmos as classes que o Spring deve gerenciar, e assim o Spring consegue verificar que essas classes vão ser os Beans que ele vai gerenciar.

@Component: Quando executarmos a aplicação o Spring vai detectar quais são os objetos que estão com essa anotação e esses objetos vão ser gerenciados por ele e também ele vai injetar essas dependências aonde for necessário e quando necessário. Ela é uma anotação genérica, então qualquer Bean que quisermos detalhar/especificar para o Spring podemos utilizar dessa anotação @Component, mas quando estamos desenvolvendo nossa aplicação fica muito mais sugestivo a gente já utilizar de Stereotypes específicos para cada responsabilidade (@Service, @Controller,

@Repository), função de determinada classe e assim também fica muito mais fácil para outros desenvolvedores quando forem trabalhar no nosso código ele quando ver a classe já entender qual a função dela.
@Service: Tem regras/logicas de negócios.
@Repository: Tem lógica de negócios do banco de dados, transações, lógica de banco, etc.
@Controller: Específica para end-points, para expormos os end-points da nossa aplicação web.

Core:

É a base *coração* do Spring Framework onde estão diversas das anotações que utilizamos para configuração deste framework e também para obtermos todo este suporte da inversão de controle com a injeção de dependências e definir todos esses Beans para que o Spring consiga então gerenciar todo o seu ciclo de vida entre outras coisas.

Beans:

@Autowired: Quando definimos que certas classes, objetos vão ser Beans para o Spring gerenciar ele já sabe quais vão ser as classes e quais São seus Stereotypes, já que ele sabe quais são essas definições de Beans temos que sinalizar de uma certa forma aonde que ele vai injetar essas instâncias quando necessárias, no controller por exemplo se usarmos algum @Service ou seja se usarmos uma classe @Service dentro do controller para obtermos por exemplo métodos de findById, findAll, temos que injetar o Service de alguma forma seja por constructor ou por set, porém tem um método melhor e menos verboso que é usar a anotação @Autowired.

Exemplo de injeção do Service no controller por meio de constructor para termos os métodos do Service nos nossos métodos GET, PUT, POST, DELETE:

- ```
  final ParkingSpotService parkingSpotService;

  public ParkingSpotController(ParkingSpotService parkingSpotService) {
      this.parkingSpotService = parkingSpotService;
  }
  ```

Exemplo de injeção do Service no controller por meio da anotação Core@Autowired para termos os métodos do Service nos nossos
  ```

métodos GET, PUT, POST, DELETE:

- ```
  @Autowired
  private ParkingSpotService parkingSpotService;
  ```

  Então agora sempre que necessário o Spring vai criar essa injeção, vai injetar o Bean SERVICE dentro do Bean Controller e assim a classe Controller vai conseguir utilizar de todos os métodos deste Service.

  @Qualifier: Quando temos por exemplo um Service que é implementado por mais de uma classe Service o Spring acaba gerando um erro por não saber qual Bean ele vai injetar no @Autowired por exemplo do controller acima, logo para resolver este erro usamos da anotação @Qualifier com o valor do Bean sem a primeira letra ser maiúscula para mostrar para o Spring qual que é Bean que ele vai levar em consideração quando ele for injetar essa dependência.

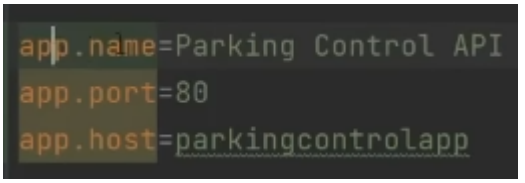
- ```
  @Autowired
  @Qualifier("parkingSpotServiceImpl")
  private ParkingSpotService parkingSpotService;
  ```

  Com isso estamos dizendo para o Spring que na hora que ele for criar este ponto de injeção (injetar as dependências) neste ponto de injeção que estamos criando ele vai considerar a implementação parkingSpotServiceImpl da nossa interface ParkingSpotService.

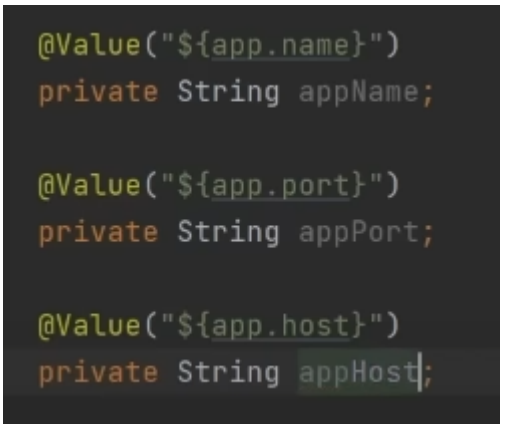
  @Value: Para explicar sobre o @Value podemos olhar para o application.properties, muitas vezes precisamos definir variáveis, propriedades que utilizamos dentro do código, mas ao invés de deixarmos isso fixo dentro de um controller por exemplo ou um Servicetemos a opção e boa prática de colocar (declarar) essas propriedades dentro de umpropertiese assim fica muito mais fácil depois se no caso precisarmos alterar ou qualquer outra modificação, pois não precisamos mexer dentro do código somente apenas nesses arquivos de propriedades, e ainda mais se tivermos utilizando arquitetura de micro-services e estivermos contemplando o pattern que é o global config onde temos um serviço especifico que faz o gerenciamento de todos esses arquivos de propriedades, então por exemplo podemos criar variáveis, propriedades, que vão estar com o global config para ele gerenciar e assim podemos na maioria dos casos

alterar o valor nos arquivos de propriedades sem precisar até de parar a aplicação.

Exemplo de propriedades e seus valores sendo definidos no properties:

- 

Eles sendo implementada em alguma classe:

- 

Ou seja esses atributos recebem o valor das propriedades que estão no properties por causa do value e podem fazer normalmente o que os atributos fazem só que estarão com o valor definido no properties.