**@Configuration**
É uma annotation que indica que determinada classe possui métodos que expõe novos beans.

**@Controller**
Associada com classes que possuem métodos que processam requests numa aplicação web.

**@Repository**
Associada com classes que isolam o acesso aos dados da sua aplicação. Comumente associada a DAO's.

**@Service**
Associada com classes que representam a ideia do Service do Domain Driven Design. Para ficar menos teórico pense em classes que representam algum fluxo de negócio da sua aplicação. Por exemplo, um fluxo de finalização de compra envolve atualizar manipular o carrinho, enviar email, processar pagamento etc. Este é o típico código que temos dificuldade de saber onde vamos colocar, em geral ele pode ficar num **Service** :).

**@Component**
A annotation básica que indica que uma classe vai ser gerenciada pelo container do Spring. Todas as annotations descritas acima são, na verdade, derivadas de @Component. A ideia é justamente passar mais semântica.

**@ComponentScan**
Em geral você a usa em classes de configuração(@Configuration) indicando quais pacotes ou classes devem ser scaneadas pelo Spring para que essa configuração funcione.

**@Bean**
Anotação utilizada em cima dos métodos de uma classe, geralmente marcada com **@Configuration,** indicando que o Spring deve invocar aquele método e gerenciar o objeto retornado por ele. Quando digo gerenciar é que agora este objeto pode ser injetado em qualquer ponto da sua aplicação.

**@Autowired**
Anotação utilizada para marcar o ponto de injeção na sua classe. Você pode colocar ela sobre atributos ou sobre o seu construtor com argumentos.

**@Scope**
Annotation utilizada para marcar o tempo de vida de um objeto gerenciado pelo container. Pode ser utilizada em classes anotadas com @Component, ou alguma de suas derivações. Além disso também pode usada em métodos anotados com @Bean. Quando você não utiliza nenhuma, o escopo default

do objeto é o de aplicação, o que significa que vai existir apenas uma instância dele durante a execução do programa. Você alterar isso, anotando o local e usando alguma das constantes que existem na classe *ConfigurableBeanFactory* ou *WebApplicationContext.*

**@RequestMapping**
Geralmente utilizada em cima dos métodos de uma classe anotada com **@Controller.** Serve para você colocar os endereços da sua aplicação que, quando acessados por algum cliente, deverão ser direcionados para o determinado método.

**@ResponseBody**
Utilizada em métodos anotados com **@RequestMapping** para indicar que o retorno do método deve ser automaticamente escrito na resposta para o cliente. Muito comum quando queremos retornar JSON ou XML em função de algum objeto da aplicação.

@RequestBody
Simplificando, a anotação @RequestBody mapeia o corpo HttpRequest para um objeto de transferência ou domínio, permitindo a desserialização automática do corpo HttpRequest de entrada em um objeto Java.

@PathVariable
Partes do URI de mapeamento podem ser vinculadas a variáveis por meio da anotação @PathVariable.

**@Primary**
Caso você tenha dois métodos anotados com **@Bean** e com ambos retornando o mesmo tipo de objeto, como o Spring vai saber qual dos dois injetar por default em algum ponto da sua aplicação? É para isso que serve a annotation **@Primary.** Indica qual é a opção padrão de injeção. Caso você não queira usar a padrão, pode recorrer a annotation **@Qualifier.**

**@Profile**
Indica em qual profile tal bean deve ser carregado. Muito comum quando você tem classes que só devem ser carregadas em ambiente de dev ou de produção. Essa eu utilizo bastante e acho uma ideia simples, mas super relevante dentro do Spring.

**@SpringBootApplication**
Para quem usa Spring Boot, essa é uma das primeiras que você. Ela engloba a **@Component, @ComponentScan** e mais uma chamada **@EnableAutoConfiguration**, que é utilizada pelo Spring Boot para tentar advinhar as configurações necessárias para rodar o seu projeto.
Bom, acho que é isso aí. Adoro escrever posts que foram solicitados diretamente pela galera que acompanha o meu trabalho, é uma motivação

bem grande. Querendo de outros temas, é só avisar por aqui ou pelo meu twitter.

### @EnableAsync

Essa aqui não é tão comum, mas muitas vezes você precisa ações no sistema em background(outra thread). Essa annotation deve ser colocada em alguma classe marcada com **@Configuration,** para que o Spring habilite o suporte a execução assíncrona.

### @Async

Uma vez que seu projeto habilitou o uso de execução de métodos assíncronos com a **@EnableAsync,** você pode marcar qualquer método de um bean gerenciado do projeto com essa annotation. Quando tal método for invocado, o Spring vai garantir que a execução dele será em outra thread.