### Spring Framework / Spring boot

**Conceitos importantes**

IoC – Inversion of Control, ou Inversão de Controle – com um BeanFactory, um ApplicationContext e um DI, ou Dependency Injection – Injeção de Dependência

**Inversão de Controle ou Inversion of Control** - conhecido pela Sigla **IoC** é um Pattern que prega para usarmos o controle das instancias de uma determinada classe ser tratada externamente e não dentro da classe em questão, ou seja, Inverter o controle de uma classe delegando para uma outra classe, interface, componente, serviço, etc.

**E o que isso quer dizer?** Basicamente você não pode fazer isso:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Pedido  {      public void CriarPedido(){          List<ItemPedido> oItemPedido = new List<ItemPedido>();      }  } |

O Código ficaria assim:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class Pedido  {      private ItemPedido \_oItemPedido;      public ItemPedido OitemPedido      {          get { return \_oItemPedido; }          set { \_oItemPedido = value; }      }      public void CriarPedido()      {           F (OitemPedido == null)              throw new Exception(“Não existem Itens para esse Pedido.”);      }  } |

**D = Dependecy Inversion / Inversão de Dependência**: Módulos de alto nível não devem depender de módulos de baixo nível, ambos devem depender de abstrações.

Já ouvi muitos programadores falarem apenas de IoC e D.I, mas nunca de Dependecy Inversion. Esse termo é um conceito apenas, para depender de abstrações. E quem aplica ele? **A Injeção de Dependência!**

Injeção de dependência ou Dependency Injection ou D.I - É um design Pattern que prega um tipo de controle externo "Um container, uma classe, configurações via arquivo, etc." inserir uma dependência em uma outra classe, darei um exemplo simples:

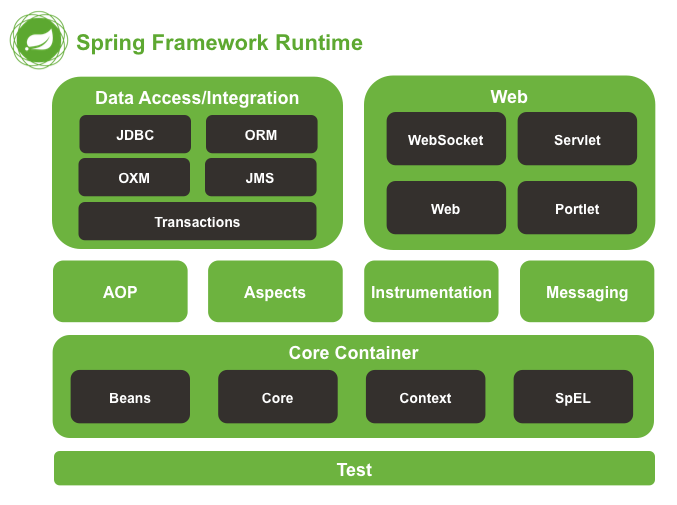
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public class Factory  {      public ItemPedido RetornarItemPedido()      {          //Realiza toda a programação necessária para criar a classe ItemPedido.          Return new ItemPedido();      }  }  public void CriarPedido()  {      Factory oFactory=new Factory();      ItemPedido oItemPedido = oFactory.RetornarItemPedido();        //... Código para criar o pedido.  } |

### Voltando ao eu importa

**Spring é uma plataforma Java que disponibiliza**, de forma ordenada e compreensiva, uma infraestrutura para o desenvolvimento de aplicações corporativas. O **Spring cuida da infraestrutura e você**, desenvolvedor, de sua aplicação/negócio.

Ao contrário, podemos desenvolver toda a aplicação utilizando apenas POJO – *Plain Old Java Object*, classes simples. Não há necessidade de se implementar ou estender nenhum artefato alienígena aos nossos requisitos, como EJB.

**O Spring simplesmente aplica serviços corporativos (JPA, JTA, JMX, JNDI, JMS, etc.)**



Seu poder de penetração como novo modelo de desenvolvimento corporativo fez com que a empresa VMware comprasse a SpringSource em 2009 por cerca de 420 milhões de dólares. Nos dias de hoje o Spring está sob o nome da Pivotal, junção criada pela VMware, EMC Corporation e GE para manter todos os produtos de software para aplicações, como o Spring.

Apesar de sempre ter estado sob a direção de alguma empresa, o Spring é e sempre será um projeto open source, sob licença Apache 2.0 e intrinsecamente comunitário.

Esse ecossistema, hoje denominado Spring IO, é um conjunto de projetos contendo o Spring Boot, Spring Framework, Spring XD, Spring Cloud, Spring Data, Spring Integration, Spring Batch, Spring Security, Spring HATEOAS, Spring Social, Spring AMQP, Spring Mobile, Spring for Android, Spring Web Flow, Spring Web Services, Spring LDAP, Spring Session, Spring Roo e alguns ainda por vir, como o Spring Statemachine.

### Spring Boot

O Spring Boot, de forma diferente do Roo (acima de tudo não há geração de código), busca solucionar a complexidade da inicialização e gerenciamento de dependências de um projeto com Spring, além de tratar de maneira coesa e eficiente a questão da configuração, fazendo uso extensivo de *Convention Over Configuration*.

Essencialmente, o Spring Boot pode ser considerado um plugin para a ferramenta de building, seja ela o Maven ou o Gradle. Seus principais objetivos são gerenciar dependências de maneira opinativa e automática, e simplificar a execução do projeto em tempo de desenvolvimento e depuração.

O Spring Boot também possui a funcionalidade de empacotamento da sua aplicação em um JAR executável contendo todas as dependências necessárias, inclusive o Servlet Container, seja ele o Tomcat, Jetty ou mesmo Undertow, apesar de ainda ser possível empacotar um WAR da forma tradicional.

O principal benefício do Boot, entretanto, é a configuração de recursos baseada no que se encontra no classpath. Se o POM de seu Maven inclui a dependência do JPA e o driver do PostgreSQL, ele irá criar uma unidade de persistência baseada no PostgreSQL.

Se adicionarmos alguma dependência web, iremos perceber que o Spring MVC assumirá configurações default e dependências com relação a diversos aspectos, como a tecnologia de apresentação (o default é o Thymeleaf), o mapeamento de recursos e marshalling de JSON (o default é o Jackson) e/ou XML (o default é o JAXB 2) para o tratamento de dados de requisição e resposta, que necessitamos em uma aplicação REST, por exemplo.

O **Spring Boot é tudo acerca de opinião, ou seja, a menos que especifiquemos explicitamente, ele irá trazer dependências predefinidas**, assim como instanciar e injetar beans predefinidos conforme o tipo de aplicação que estamos construindo (iremos ver como funciona esse mecanismo em nossa aplicação exemplo).

Podemos não concordar com algumas opiniões como, por exemplo, a versão de dada dependência ou o Servlet Container default, mas ao menos ele as tem e, caso insistamos com a nossa, ele não se opõe.

Por exemplo, se nosso POM declara o uso de persistência com o Spring Data, mas não declaramos nada mais, será assumido que queremos utilizar JPA e Hibernate para a camada de persistência. Se estamos implementando uma aplicação web e não declaramos nada em específico, o Boot irá assumir que usaremos o Thymeleaf como view resolver.

Em suma, o que o **Spring Boot faz principalmente, de maneira muito elegante, é nos livrar de qualquer preocupação desnecessária com dependências** e configurações que são as mesmas em 99% dos casos. E se o caso está no 1% restante, ele acata nossa opinião com a mesma elegância, como veremos em nosso projeto.

### Resumo Spring Boot

Ao passo que a configuração de infraestrutura e definição da arquitetura estão entre as etapas que mais consomem recursos durante o desenvolvimento de aplicações web com Java, emergiram iniciativas buscando mitigar questões dessa ordem. Nesse sentido, o Spring Boot provê o *bootstrap* e configuração automática, sempre que possível, de aplicações Spring *standalone* rodando em Apache Tomcat embarcado, além de suportar integração com uma gama diversa de ferramentas de mercado.

### Thymeleaf