**Spring** **Framework**

Spring se baseia nos conceitos de inversão de controle e injeção de dependências.

**Alto Acoplamento**

**Problemas Essenciais:**

**Fred** **Brooks** no seu texto clássico “No Silver Bullet” [2] aponta quatro dificuldades essenciais inerentes ao desenvolvimento de qualquer software:

**Conformidade**: Todo software deve ser compatível com o ambiente no qual será executado.

**Invisibilidade**: Se trata na dificuldade de tornar o software invisível em seus aspectos.

**Mutabilidade:** O software deve ser capaz de se mutar para atender a diferentes requisitos.

**Complexidade:** Se trata da dificuldade de quando o software se torna extremamente complexo, atingindo o limite do nosso intelecto.

A POO já soluciona parte desses problemas essenciais, porém traz alguns problemas relacionados a interação de classes e objetos, quando suas classes dependem muito umas das outras temos o **Alto Acoplamento** e consequentemente, alterações em uma normalmente acarretam em mudanças de comportamento em locais inesperados do sistema.

No Spring temos o termo dependência, que se refere ao relacionamento entre classes, dessa forma imaginamos uma classe integradora, que possui duas outras classes na sua dependência, para que a integradora exista, necessariamente as suas dependências devem existir.

Para suprir os problemas gerados pelas dependências, utilizamos o conceito de **principio da inversão de Dependência** utilizamos de classes abstratas, invertendo o processo, assim mudanças nas classes de dependência não influenciaram a classe abstrata.

**Inversão de Controle**

Todo framework é na verdade uma inversão de Controle, onde a interface utiliza-se de reflexão e classes abstratas, e quem define os tipos das classes é na verdade a classe que irá utilizar a interface.

O conceito de que o Spring é um framework de inversão de controle é um pleonasmo, um conceito melhor seria o de que se baseia na injeção de dependência que é uma especialização da inversão de controle.

Na **Injeção de Dependência** não é a classe que é responsável por definir quais serão suas dependências, sendo delegada a um **container de injeção de dependências.**

Simplificando ao máximo, o Spring é um competente container de injeção de dependências em cima do qual foram construídos diversos módulos com o objetivo de facilitar o desenvolvimento de aplicações corporativas.

**Conhecendo o Spring Framework**

**Programação orientada a aspectos (AOP):**

Trata-se dos comportamentos que aparecem por toda a aplicação, porém que não estão relacionados à logica de negócio que estamos implementando, geralmente utilizando apenas POO percebemos uma repetição generalizada por toda a aplicação, esses comportamentos chamamos de aspectos.

Enquanto a **injeção** **de** **dependências** modulariza nosso sistema usando o que a Orientação a Objetos tem a nos oferecer, a AOP diminui ainda mais o acoplamento isolando os aspectos que não são facilmente identificáveis e isolados pela orientação a objetos

Identificado e isolados os aspectos, tudo o que a AOP precisa fazer é interceptar em tempo de execução as chamadas aos métodos onde detectamos a necessidade de aplicação do comportamento identificado e, com base nisto, alterar o comportamento de nossas classes sem que estas sequer saibam disto.

**Resumindo Inversão de Controle, Injeção de Dependências e Programação Orientada a Aspectos**

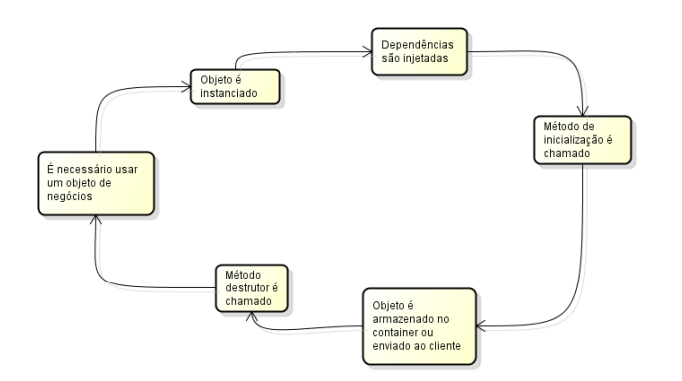
A grosso modo, você pode pensar o seguinte: a IoC controla o ciclo de vida de nossas aplicações, a DI define quais classes iremos instanciar e em quais lugares iremos injetá-las. Já a AOP vai além na modularização que a DI nos proporciona, adicionando novos comportamentos às nossas classes.

**Container:**

Este é a instância de uma classe aonde implementamos nossos requisitos funcionais. O EJB é um objeto de negócios e, no caso do Spring, nós o chamamos de bean.

Um bean é um objeto que possui seu ciclo de vida gerenciado pelo container de IoC/DI do Spring.

**Ciclo de vida de um Objeto:**



Voltando ao bean, este é mais que um mero JavaBean. Deve ser visto como um componente. O que diferencia um componente dos demais objetos, como um JavaBean, por exemplo, são as seguintes características:

**• Possui baixa granularidade,** ou seja, seus clientes não precisam saber quais as suas dependências internas;

• O que realmente importa para seus clientes é **a interface disponibilizada pelo objeto**, que define quais os serviços oferecidos pelo mesmo (é o seu contrato);

• São **facilmente substituíveis por outras implementações** que mantenham o mesmo contrato (plugabilidade).

**O que é um JavaBean?**

Foi um padrão adotado pela Sun para a escrita de componentes reutilizáveis. Todo JavaBean deve satisfazer três condições:

1) Deve possuir um construtor público que não receba parâmetros

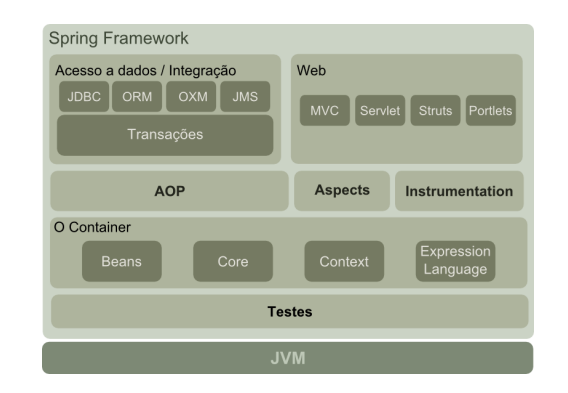
2) Todos os seus atributos visíveis devem ser declarados como privados (ou protegidos) e acessados apenas por métodos get e set.

3) Deve implementar a interface java.io.Serializable

Podemos dizer, portanto, que o container é o elemento arquitetural de nossa aplicação, responsável por gerenciar o ciclo de vida dos componentes do nosso sistema.

Um container pode ser intrusivo ou não. É intrusivo quando requer que seus objetos gerenciados dependam de si de alguma maneira.

**Partes do Spring**



**O container**

Se quiser trabalhar com o Spring, este provavelmente é o único módulo obrigatório. Os módulos **core** e **beans** são o núcleo do framework aonde são implementados o suporte à **inversão** **de** **controle** e **injeção** **de** **dependências**.

**Context:** encontra-se a implementação do **ApplicationContext**. No Spring há dois tipos de containers: **BeanFactory** e **ApplicationContext**.

**BeanFactory:** Existe dês da primeira versão do framework, oferece suporte básico a IoC e DI.

**ApplicationContext:** Se baseia no BeanFactory porém oferece recursos corporativos mais avançados, como por exemplo, recursos de internacionalização.

**Expression Language(SpEL):** A SpEl nos fornece uma linguagem muito parecida com a EL que usamos nos JSP, porém voltada a configuração do container. Ela torna nossos arquivos de configuração “vivos”, na medida em que a partir do seu uso podemos definir valores de configuração em tempo de execução e não em tempo de configuração.

**AOP**

O Spring vem acompanhado do seu próprio framework para trabalhar como AOP, pode vim de forma tradicional sendo a primeira implementação de AOP do Spring ou Aspects, que é o uso do AspectJ como motor do AOP.

**Instrumentation**

O módulo de instrumentação facilita a vida do pessoal de suporte oferecendo facilidades na implementação de JMX. Esta tecnologia nos permite acompanhar em tempo de execução tudo o que acontece com nossos sistemas, gerando diversas estatísticas interessantes.

**Acesso a dados e Integração**

O Spring oferece suporte as principais tecnologias do mercado, o suporte a estas tecnologias se dá através de templates, que reduzem significativamente a quantidade de código boilerplate.

Além do suporte a estas tecnologias, este módulo nos oferece também uma nova hierarquia de exceções.

**Aplicações na Web com o Spring**

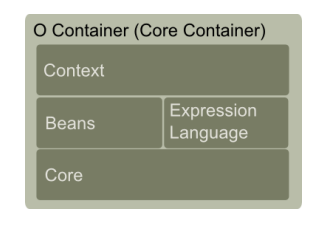
O Spring possui um framework para integração MVC chamado de Spring MVC.

Além deste framework MVC, há também suporte básico para que o container do Spring seja aplicado em uma aplicação Java web convencional, através de pequenas alterações no arquivo web.xml e suporte a alguns frameworks web de mercado como JSF, Struts 1.x e 2.x, WebWork e Tapestry.

**Componentes Extras**

Existe diversos componentes opcionais para o container, um exemplo São Spring Security, Spring Data, Spring Social e Spring Batch.

**Container**



**Core:**

Se encontram as classes básicas que permitirão a implementação no bean, é neste módulo que se encontra toda a infraestrutura necessária que permite a implementação do container.

**Beans:**

Encontramos a primeira versão do container o BeanFactory.

**Expression Language:**

Utiliza-se o Spring Expression Language (SpEL).

**Context:**

Se encontra a implementação mais avançada e comumente usada do container que é o ApplicationContext.

**Preparando o ambiente**

Há basicamente duas maneiras de configurar seu projeto: usando ou não um gerenciador de dependências baseado no Maven.

**Melhores IDE para Spring:**

O Netbeans, Eclipse e JetBrains IDEA oferecem suporte ao Spring através de plugins, porém a melhor implementação é no SpringSource Tool Suite.

O SpringSource Tool Suite (STS) é a distribuição customizada do Eclipse desenvolvida pela equipe da SpringSource e que pode ser baixada em http://www. springsource.org/downloads/sts. É o ambiente ideal para o desenvolvedor habituado a trabalhar com o Eclipse pois já vêm com todos os plugins necessários para que o desenvolvedor possa começar a trabalhar imediatamente sem que precise se preocupar com a instalação de complementos em sua IDE.

**Declarando Beans:**

Para que um bean exista são necessários três elementos: as configurações, o container e a classe que o implementa. No Spring as configurações podem ser: XML, anotações ou código Java.