

Plataforma Spring

Cenário

- **Aplicações corporativas: Servlets e EJBs**
 - **EJB 2.1**
 - **Estrutura "pesada" e pouco produtiva**

Spring

- **Criado em 2002 por Rod Johnson**
 - **Expert One-on-One J2EE Design and Development**
- **Container de IoC (Inversion of Control)**
- **Não intrusivo**
- **Interfaces e POJOs**

Spring

Benefícios

- Mais aplicação, menos infraestrutura
- Componentes leves
- Facilidades para testes automatizados
- Sem essa de reinventar a roda

Spring

Resumindo...

O foco do Spring é permitir a criação de aplicações utilizando somente POJOs, aplicando a eles serviços corporativos de forma não intrusiva.

Projetos

[DOCS](#)[GUIDES](#)[PROJECTS](#)[BLOG](#)[QUESTIONS](#)

Main Projects

From configuration to security, web apps to big data – whatever the infrastructure needs of your application may be, there is a Spring Project to help you build it. Start small and use just what you need – Spring is modular by design.



SPRING IO PLATFORM

Provides a cohesive, versioned platform for building modern applications. It is a modular, enterprise-grade distribution that delivers a curated set of dependencies.



SPRING BOOT

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.



SPRING FRAMEWORK

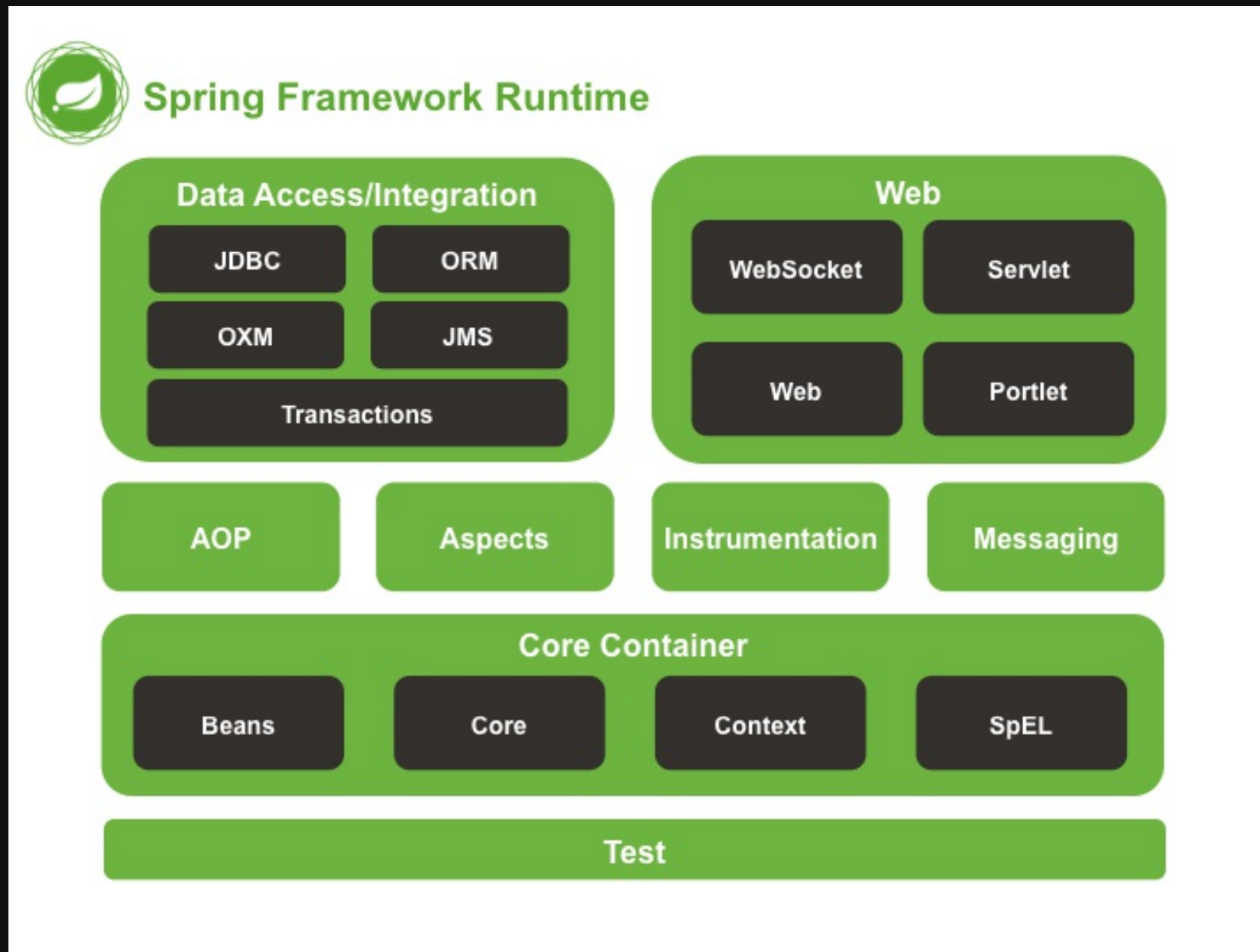
Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.

Spring Framework

- **Núcleo do Spring**
- **Suporte a**
 - **Injeção de Dependência**
 - **Aspectos**
 - **MVC**
 - **JDBC, JPA, JMS, etc**

Spring Framework

Módulos



Spring Framework

Core Container

- **Injeção de Dependência**
 - spring-core e spring-beans
- **Contexto e acesso aos beans**

Spring Framework

AOP

- **Inclusão de responsabilidades ortogonais à aplicação**
 - **Compatibilidade com AspectJ**
- **Instrumentação**

Spring Framework

Messaging

Fornece as principais estruturas para aplicações orientadas a mensagem.

Spring Framework

Data Access

- **Abstrações para JDBC**
- **Contexto transacional de forma programática ou declarativa**
- **Integração com os principais frameworks ORM**
- **Mapeamento Objeto / XML**

Spring Framework

Web

- **Funcionalidades básicas do mundo Web**
- **MVC**
- **Portlets**

Spring Framework

Test

- Testes de unidades e integração
- Suporte JUnit e TestNG
- Cache e Mocks

Container de IoC

Container

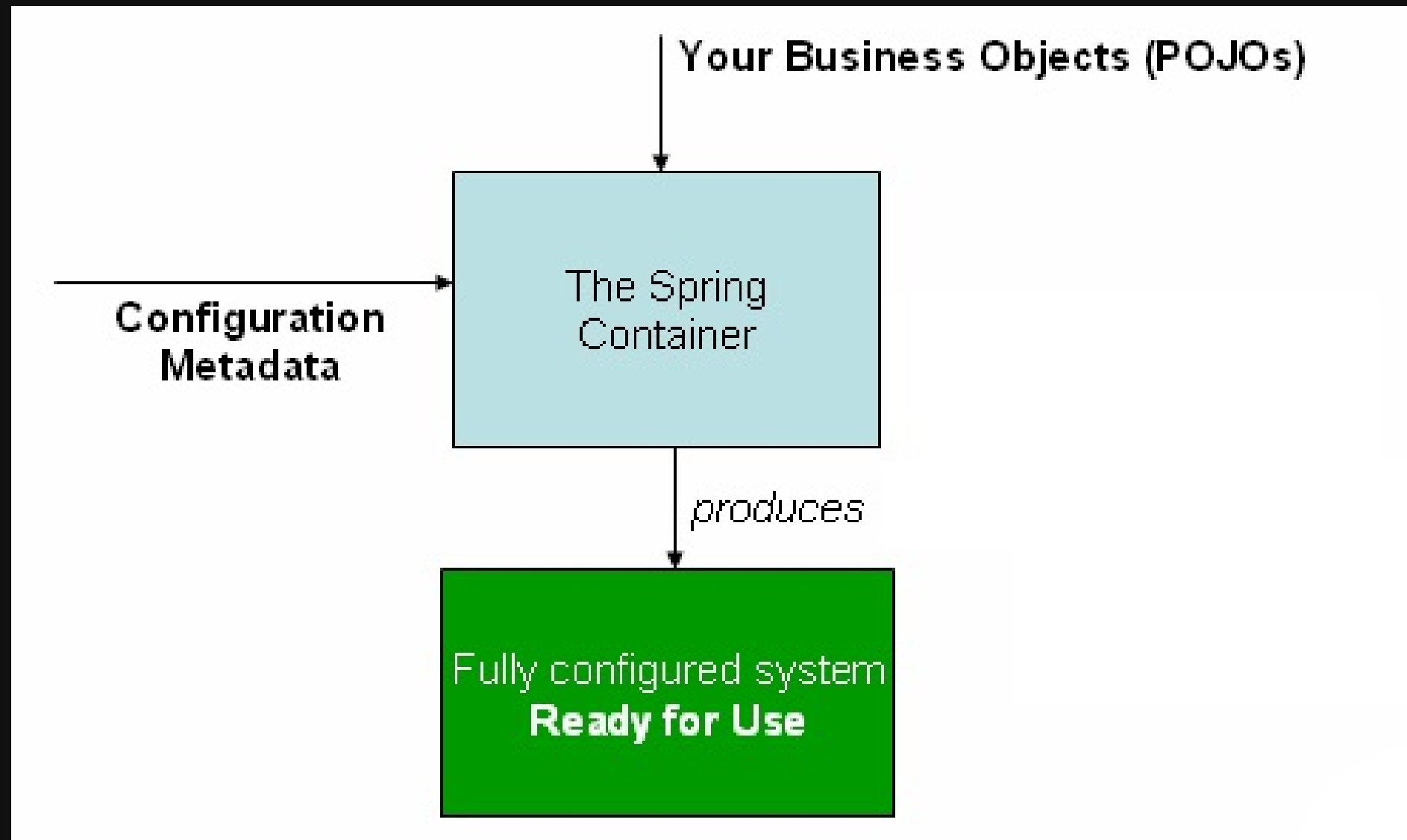
- **Gerencia todos os componentes da aplicação usando IoC**
- **Configurável por meio XML, anotações ou código Java**

Container

Principais Interfaces

- **BeanFactory**
 - Container básico para uso de injeção de dependência
- **ApplicationContext**
 - Acrescenta funcionalidades corporativas ao BeanFactory

Container



Container

Exemplo de Configuração

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="...">
  ...
  <bean id="braveKnight" class="test.dnd.BraveKnight">
    <constructor-arg ref="slayQuest"></constructor-arg>
  </bean>
  <bean id="slayQuest" class="test.dnd.SlayDragonQuest">
    <constructor-arg value="..."></constructor-arg>
  </bean>
</beans>
```

Beans

Beans

O que são?

- **Objetos que formam a essência da aplicação**
- **Instanciados, conectados e gerenciados pelo container**

Beans

Definição \ Configuração

- Como criar o bean
- Detalhes do seu ciclo de vida
- Dependências

Beans

Propriedade	Descrição
class	Classe usada para criação do bean
id \ name	Identifica unicamente o bean
scope	Escopo do bean
constructor-arg	Argumentos do construtor bean
property	Injeta dependências por meio de métodos 'set'
autowire	Modo de <i>autowiring</i>
init-method	Método executado logo após a atribuição das propriedades necessárias do bean
destroy-method	Método executado quando o container é destruído

Beans

Formas de Instanciação

- Construtor
- Fábrica
 - Método estático vs instância

Beans

Constructor

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans">
  ...
  <bean id="braveKnight" class="test.dnd.BraveKnight">
    <constructor-arg ref="slayQuest"></constructor-arg>
  </bean>
  <bean id="slayQuest" class="test.dnd.SlayDragonQuest">
    <constructor-arg value="..."></constructor-arg>
  </bean>
</beans>
```

Beans

Fábrica - método estático

```
<bean id="clientService" class="examples.ClientService"  
    factory-method="createInstance"/>
```

```
public class ClientService {  
    private static ClientService clientService =  
        new ClientService();  
    private ClientService() {}  
    ...  
    public static ClientService createInstance() {  
        return clientService;  
    }  
}
```

Beans

Fábrica - método de instância

```
<bean id="serviceLocator" class="examples.DefaultServiceLocator">
  <!-- inject any dependencies required by this locator bean -->
</bean>

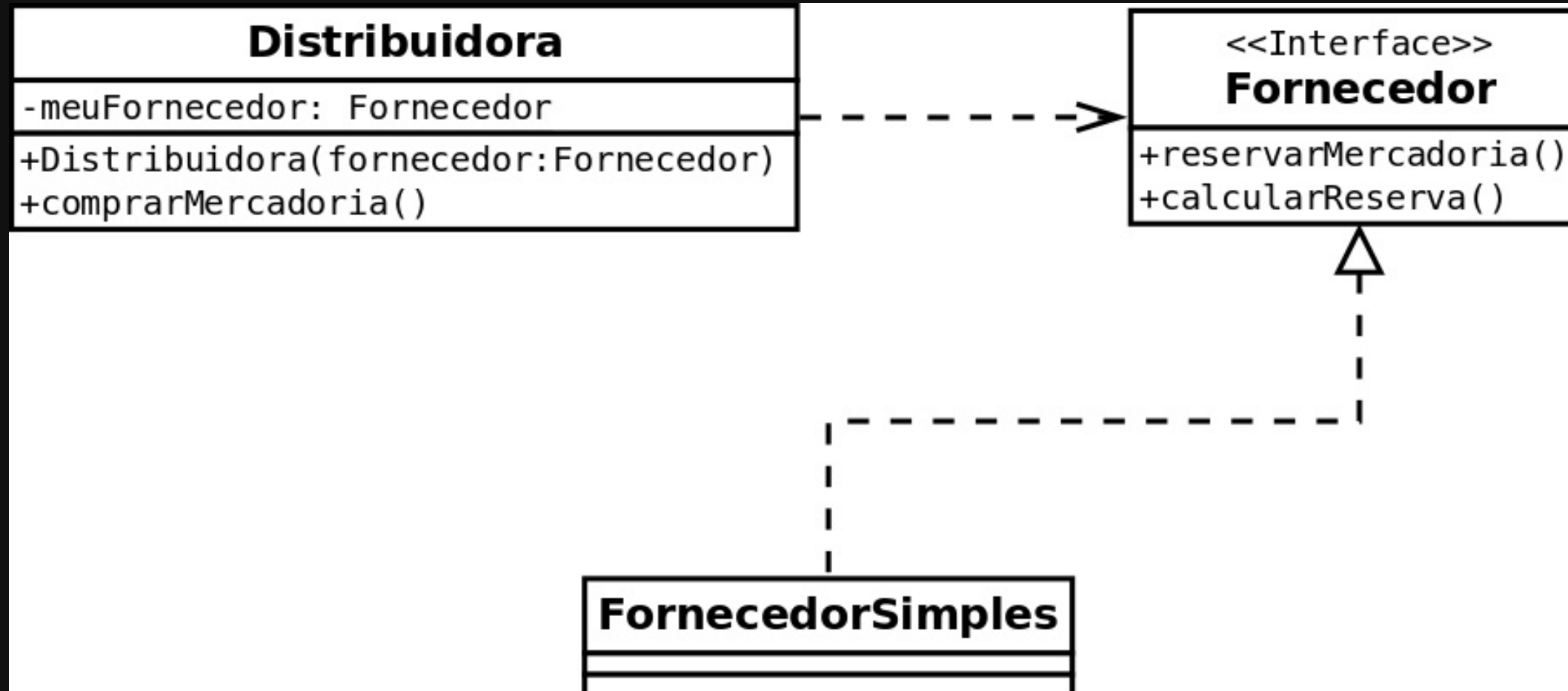
...
<bean id="client" factory-bean="serviceLocator"
  factory-method="createClientServiceInstance"/>
```

```
public class DefaultServiceLocator {
  ...
  private static ClientService clientService =
    new ClientServiceImpl();
  private DefaultServiceLocator() {}
  ...
  public ClientService createClientServiceInstance() {
    return clientService;
  }
}
```

Exercício 1

Uma distribuidora coordena ações de reserva e compra de mercadorias para vários clientes. Ela pode interagir com vários fornecedores e decidir qual o melhor para realizar a reserva. A classe que representa os distribuidores interage com a abstração de fornecedor por meio da interface `Fornecedor`. Enquanto a classe `Distribuidora` expõe uma operação de `comprarMercadoria`, ela dispara uma avaliação de preço e após isso faz a reserva efetivamente. Implementar a aplicação utilizando o Spring Framework.

Exercício 1



Exercício 1

Dicas

- **Crie um projeto legado (Spring + Maven) do Spring no STS**
- **Crie um XML para definir os beans e suas relações**
- **Utilize as propriedades constructor-arg e ref para passar as informações adequadas para os construtores**
- **Utilize a classe `ClassPathXmlApplicationContext` como Contexto passando o xml adequadamente**

Injeção de Dependência

Injeção de Dependência

Tipos

- Construtores
- Métodos set

Injeção de Dependência

Construtor

```
<beans>
  <bean id="foo" class="x.y.Foo">
    <constructor-arg ref="bar"/>
    <constructor-arg ref="baz"/>
  </bean>
  ...
  <bean id="bar" class="x.y.Bar"/>
  <bean id="baz" class="x.y.Baz"/>
</beans>
```

```
package x.y;
...
public class Foo {
  ...
  public Foo(Bar bar, Baz baz) {...}
}
```

Injeção de Dependência

Construtor

```
<bean id="exampleBean" class="examples.ExampleBean">  
  <constructor-arg type="int" value="7500000"/>  
  <constructor-arg type="java.lang.String" value="42"/>  
</bean>
```

```
<bean id="exampleBean" class="examples.ExampleBean">  
  <constructor-arg index="0" value="7500000"/>  
  <constructor-arg index="1" value="42"/>  
</bean>
```

```
public class ExampleBean {  
  ...  
  public ExampleBean(int years, String ultimateAnswer) {  
    this.years = years;  
    this.ultimateAnswer = ultimateAnswer;  
  }  
}
```

Injeção de Dependência

Método Set

```
<bean id="exampleBean" class="examples.ExampleBean">
  <property name="beanOne">
    <ref bean="anotherExampleBean"/>
  </property>
  <property name="beanTwo" ref="yetAnotherBean"/>
  <property name="integerProperty" value="1"/>
</bean>
...
<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

```
public class ExampleBean {
  private AnotherBean beanOne;
  private YetAnotherBean beanTwo;
  private int i;
  ...
  public void setBeanOne(AnotherBean beanOne) {...}
  ...
  public void setBeanTwo(YetAnotherBean beanTwo) {...}
  ...
  public void setIntegerProperty(int i) {...}
}
```

Injeção de Dependência

Injeção por Construtor vs Método

Injeção de Dependência

Coleções

- É possível injetar coleções utilizando os seguintes elementos:
 - `<list/>`
 - `<set/>`
 - `<map/>`
 - `<props/>`

Injeção de Dependência

Setando Coleções

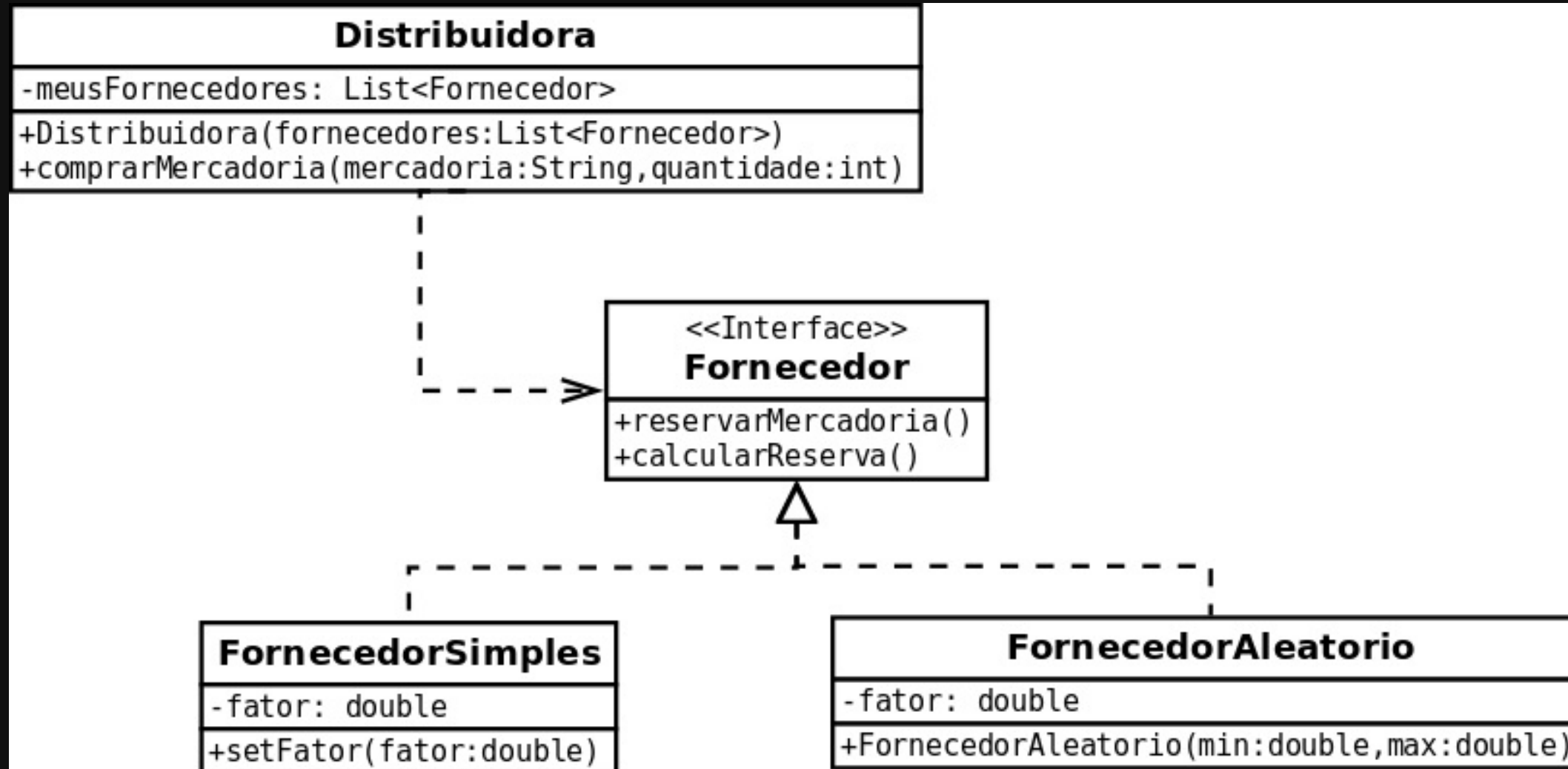
```
<bean id="moreComplexObject" class="example.ComplexObject">
  <property name="adminEmails">
    <props>
      <prop key="administrator">administrator@example.org</prop>
      <prop key="support">support@example.org</prop>
    </props>
  </property>
  <property name="someList">
    <list>
      <value>a list element followed by a reference</value>
      <ref bean="myDataSource" />
    </list>
  </property>
  <property name="someMap">
    <map>
      <entry key="an entry" value="just some string"/>
      <entry key="a ref" value-ref="myDataSource"/>
    </map>
  </property>
</bean>
```

Exercício 2

Refatore a aplicação da Distribuidora para compreender os seguintes requisitos:

- O cálculo da reserva de mercadoria é baseado em uma taxa:
 - $\text{valor} = \text{preço} \times \text{quantidade} \times \text{taxa}$
- A distribuidora contará com dois fornecedores:
 - Simples: A taxa é informada por um método setter
 - Aleatorio: Calcula a taxa de forma randômica baseado em um limite máximo e mínimo
- A distribuidora consultará o valor da reserva em todos os fornecedores e selecionará o quem tem menor custo

Exercício 2



Autowiring

Autowiring

O que é isso?!

Autowiring

Vantagens

- **Redução de declarações de propriedades e afins no XML de configuração**
- **Dinamicidade e adaptabilidade aos objetos enquanto eles evoluem**

Autowiring

Como utilizar?

Atributo **autowire** da tag `<bean>`.

Autowiring

Modo	Descrição
no	Padrão. Sem autowire.
byName	Faz a ligação por nome da propriedade
byType	Faz a ligação por tipo da propriedade
constructor	Exatamente como byType mas acontece no construtor

Autowiring

Limitações

- Dependências explícitas sobrescrevem a ligação automática
- Não resolve dependências para tipos simples: primitivos, Strings, Doubles, etc
- Por não ser tão exata como a ligação explícita, pode gerar resultados inesperados
- Caso haja múltiplas definições para um mesmo tipo de bean, o container não consegue resolver a dependência quando o componente aguarda um valor \ referência único

Exercício 3

Modificar o projeto da Distribuidora para que os fornecedores sejam injetados por meio de autowiring

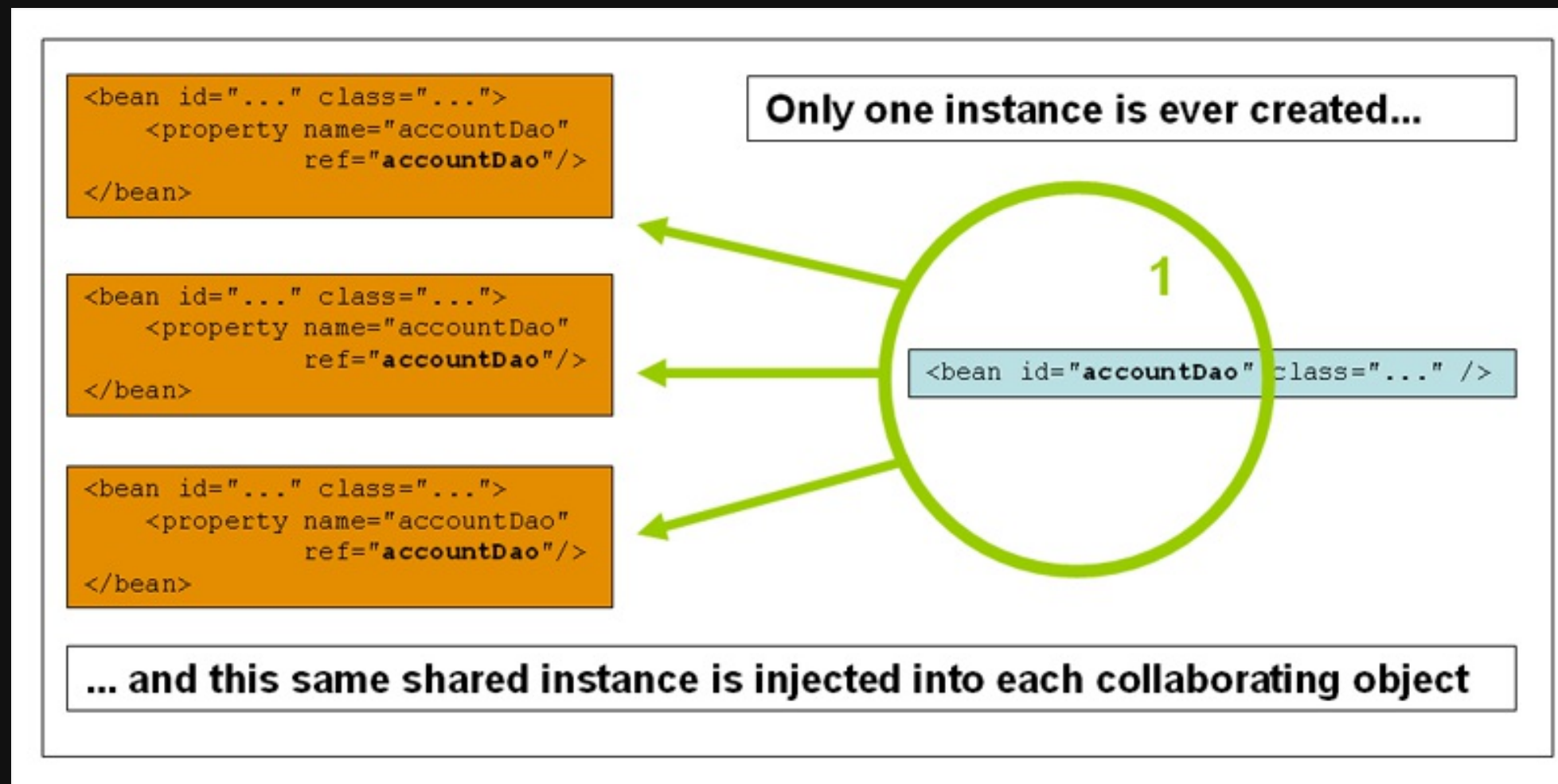
.

Escopo

Escopo	Descrição
singleton	Padrão. Uma única instância criada por definição
prototype	Uma definição pode gerar diversas instâncias
request	Uma definição por requisição HTTP
session	Uma definição por sessão HTTP
globalSession	Uma definição por sessão global HTTP. Válido no contexto de portlets
application	Definido por todo o ciclo de vida de um ServletContext
websocket	Definido por todo o ciclo de vida de um WebSocket

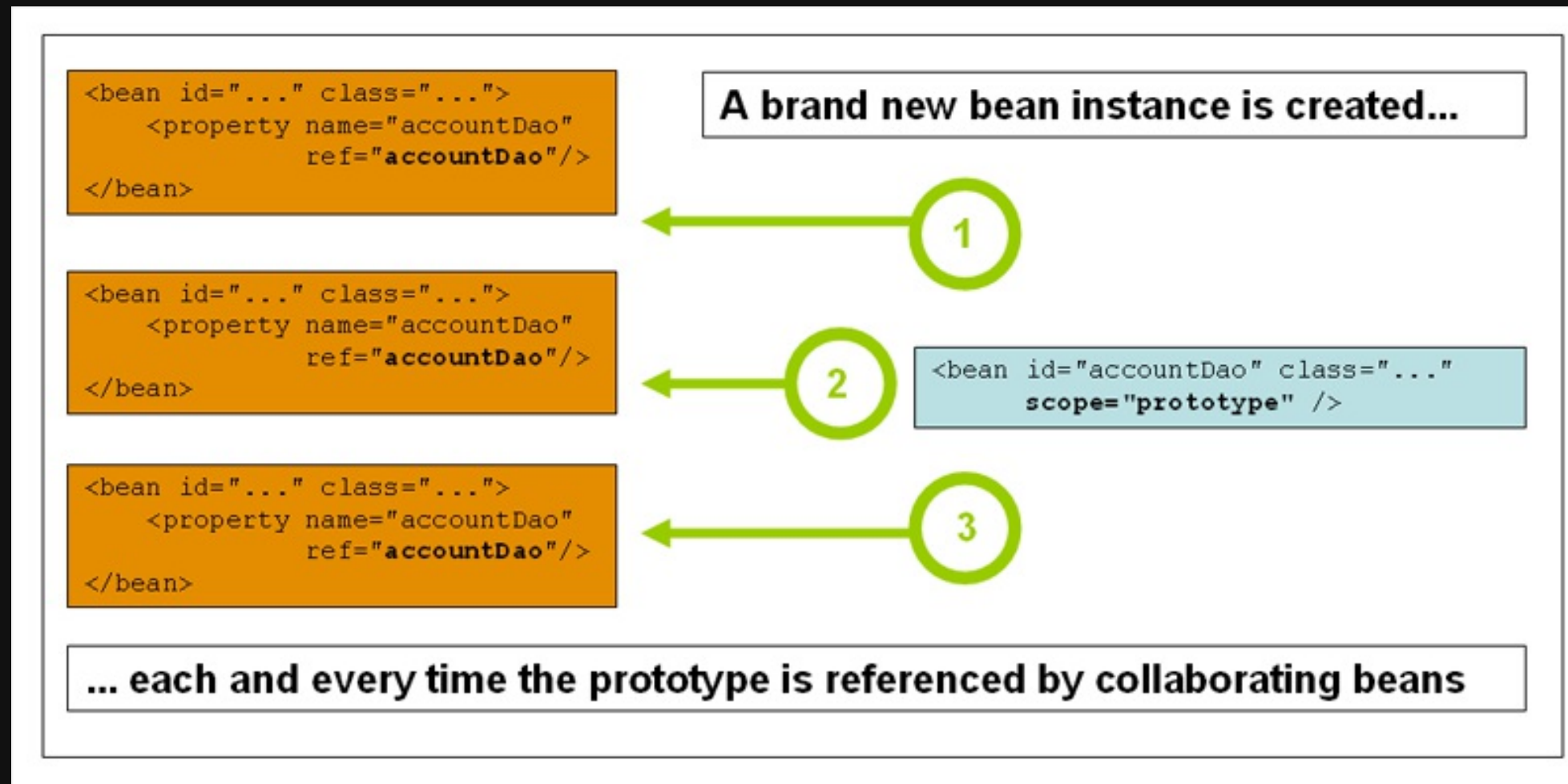
Escopo

Singleton



Escopo

Prototype



Singleton vs Prototype

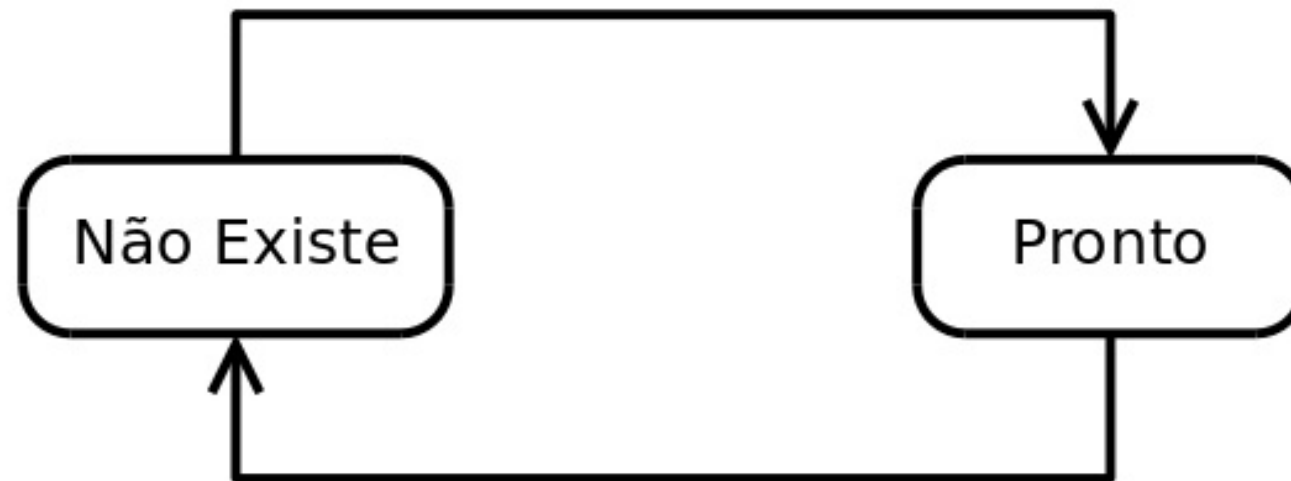
Exercício 4

Importem e executem o projeto escopo disponível no repositório ou pasta da rede.

Ciclo de Vida

Ciclo de Vida

1. @PostConstruct
2. InitializingBean#afterPropertiesSet()
3. <bean ... init-method="..." />



1. @PreDestroy
2. DisposableBean#destroy()
3. <bean ... destroy-method="..." />

Anotações

Anotações

Devem ser habilitadas usando o elemento `context:annotation-config`

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context ...">
  <context:annotation-config/>
</beans>
```


Anotações

@Autowired

Pode ser usada em construtores e métodos Set

```
public class Distribuidora {  
    ...  
    List<Fornecedor> meusFornecedores;  
    ...  
    @Autowired  
    public Distribuidora(List<Fornecedor> fornecedores) {  
        meusFornecedores = fornecedores;  
    }  
}
```

Anotações

@Required

Utilizada em métodos Set

```
public class SimpleMovieLister {  
    ...  
    private MovieFinder movieFinder;  
    ...  
    @Required  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
}
```

Anotações

@Component

Forma genérica para definir uma classe como Bean

```
@Component  
public class Distribuidora {  
    ...  
    @Autowired  
    Fornecedor meuFornecedor;  
    ...  
}
```

Anotações

@Component

- Outras formas de declarar um bean como gerenciado:
 - @Service
 - @Controller
 - @Repository

Anotações

@Scope

Informa o escopo do bean

```
@Scope("prototype")
@Component
public class Distribuidora {
    ...
    @Autowired
    Fornecedor meuFornecedor;
    ...
}
```

Exercício 5

Modifique o projeto da distribuidora para utilizar anotações durante o autowiring.