Spring

Употребление термина внедрение зависимостей для обозначения инверсии управления всегда корректно. В контексте Spring эти термины можно употреблять равноценно, что совершенно не теряет их смысл.

По документации можно почитать (<https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#xsd-schemas-util-set>).

Документация на русском( <https://spring-projects.ru/about/>)

Конфигурация из XML

1. В Maven подгружаем зависимости:
   1. context
   2. core
   3. beans
   4. aop
2. Создаём конфигурационный файл:

<?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:util="http://www.springframework.org/schema/util" xsi:schemaLocation="  
 http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/util https://www.springframework.org/schema/util/spring-util.xsd"> <!-- bean definitions here -->  
  
 <bean id="provider" class="Classes.HelloWorldMessageProvider"/>  
  
 <bean id="render" class="Classes.StandardOutMessageRenderer">  
 <property name="messageProvider" ref="provider"/>  
 </bean>  
  
</beans>

1. Пишем код

public class Start {  
 public static void main(String[] args) {  
  
 ApplicationContext context = new ClassPathXmlApplicationContext("application.xml");  
 MessageRender rend = context.getBean("render", StandardOutMessageRenderer.class);  
 rend.render();  
 }  
}

Конфигурация Код + Annotation

1. Подгружаем в pom зависимости, как в первом случае конфигурации (XML)
2. Создаём класс для конфигурации

package ru.startpack;  
  
import Classes.HelloWorldMessageProvider;  
import Classes.StandardOutMessageRenderer;  
import Interface.MessageProvider;  
import Interface.MessageRender;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class ConfigurationProject {  
  
 @Bean  
 public MessageProvider provider() {  
 return new HelloWorldMessageProvider();  
 }  
  
 @Bean  
 public MessageRender render(){  
 MessageRender render = new StandardOutMessageRenderer();  
 render.setMessageProvider(provider());  
 return render;  
 }  
}

1. Пишем код

public class Start {  
 public static void main(String[] args) {  
  
 ApplicationContext context = new AnnotationConfigApplicationContext(ConfigurationProject.class);  
 MessageRender renderer = context.getBean("render", MessageRender.class);  
 renderer.render();  
 }  
}

Меняем один класс на другой, так как меняется логика приложения

ApplicationContext context = new ClassPathXmlApplicationContext("application.xml");

Меняем на:

ApplicationContext context = new AnnotationConfigApplicationContext(ConfigurationProject.class);