MENU

Руководство Spring для начинающих

1- Введение

Данная статья основана на:

- **Spring Framework 4.x**
- **Eclipse 4.6 NEON (ok for Eclipse 4.5 MARS)**

В этой статье я использую **Maven** чтобы объявить библиотеки **Spring** которые буду использовать,вместо того чтобы скачать **Spring** и объявлять библиотку обычным способом.

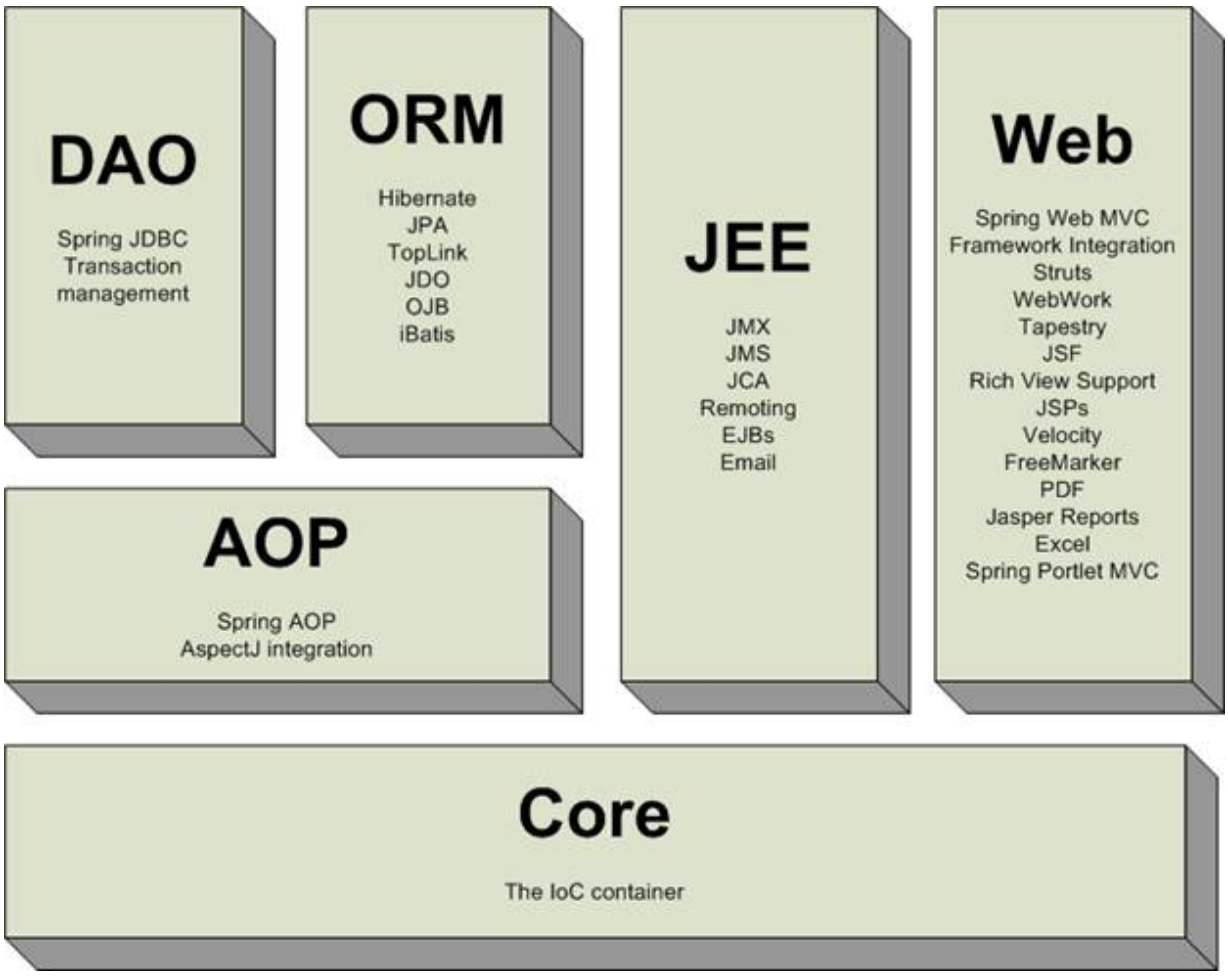
Maven это инструмент который помогает вам управлять библиотеками автоматически и эффективно, он стал распространенным, что каждый программист **Java** должен знать. Если вы не знаете про **Maven** вы можете уделить 10 минут и изучить про способ использования здесь:

- Руководство Maven для начинающих

В случае если вы хотите скачать **Spring** и объявить библиотеку традиционно, вы можете посмотреть аннотацию в конце статьи.

2- Spring Framework

Изображение ниже иллюстрирует структуру **Spring Framework**.



1. **IoC Container**: Это самая важная и самая основная часть, платформа **Spring**. Он играет роль конфигурации и управления жизненного цикла (Lifecycle) объектов java. В сегодняшней статье мы изучим эту часть.
2. **DAO, ORM, AOP, WEB**: Эти модули являются готовыми tool или framework интегрированные в **Spring**.

2.1- Inversion of Control & Dependency Injection

Чтобы понять эту ситуация рассмотрим некоторые классы ниже:

```
// Interface HelloWorld
public interface HelloWorld {
    public void sayHello();
}

// Class implements HelloWorld
public class SpringHelloWorld implements HelloWorld {
   public void sayHello()  {
         System.out.println("Spring say Hello!");
   }
}

// Other class implements HelloWorld
public class StrutsHelloWorld implements HelloWorld {
   public void sayHello()  {
         System.out.println("Struts say Hello!");
   }
}


// And Service class
public class HelloWorldService {

    // Field type HelloWorld
    private HelloWorld helloWorld;

    // Constructor HelloWorldService
    // It initializes the values for the field 'helloWorld'
    public HelloWorldService()  {
```

```
30          this.helloWorld = new StrutsHelloWorld();
31      }
32
33  }
```

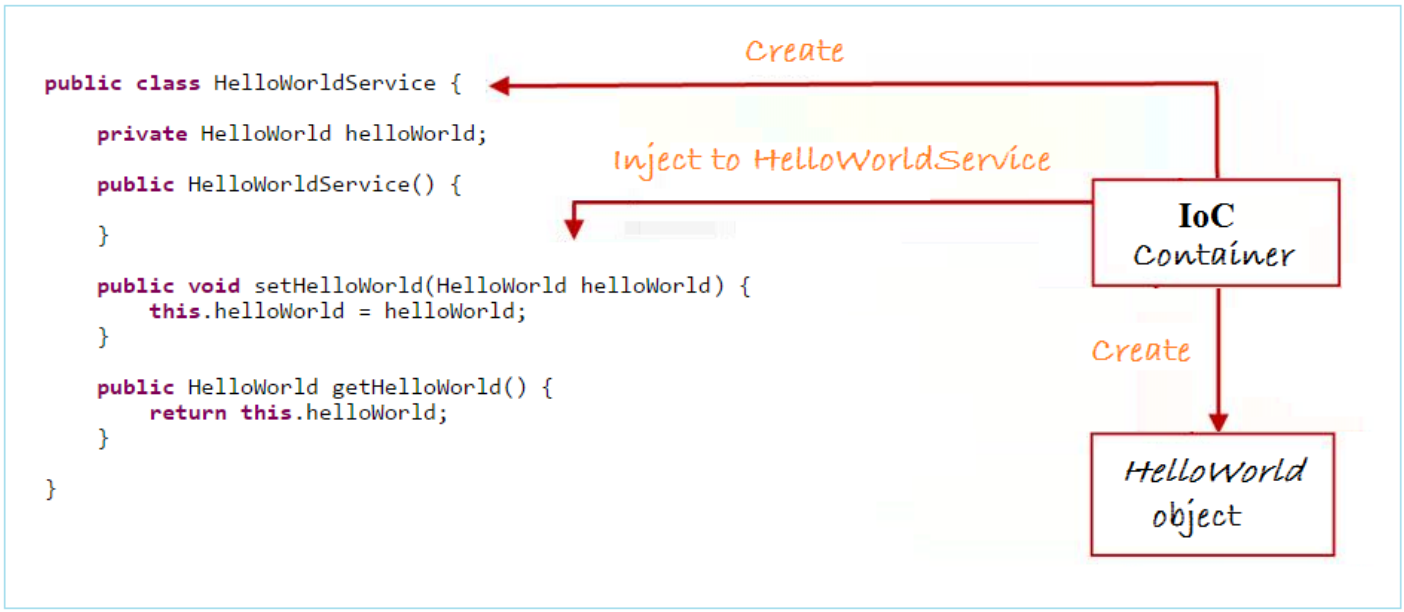
Можно увидеть что класс **HelloWorldService** управляет создание объекта ***HelloWorld***.

- В случае выше когда объект **HelloWorldService** создан от его конструктора (constructor), объект ***HelloWorld*** так же создается, и он создается от **StrutsHelloWorld**.

Вопрос состоится в том, что вы хотите создать объект **HelloWorldService** одновременно создается объект **HelloWorld**, но он должен быть **SpringHelloWorld**.
Поэтому здесь **HelloWorldService** управляет ***"object creation"*** в ***HelloWorld***. Почему мы не передаем создание ***HelloWorld*** третьей стороне для обработки вместо того, чтобы создавать его в ***HelloWorldService***. У нас есть понятие *"inversion of control"* то есть *"инверсия контроля"* (IoC).

И **IoC Container** будет играть роль управляющего создающего и **HelloWorldService** и ***HelloWorld***.

> *IoC = Inversion of Control*

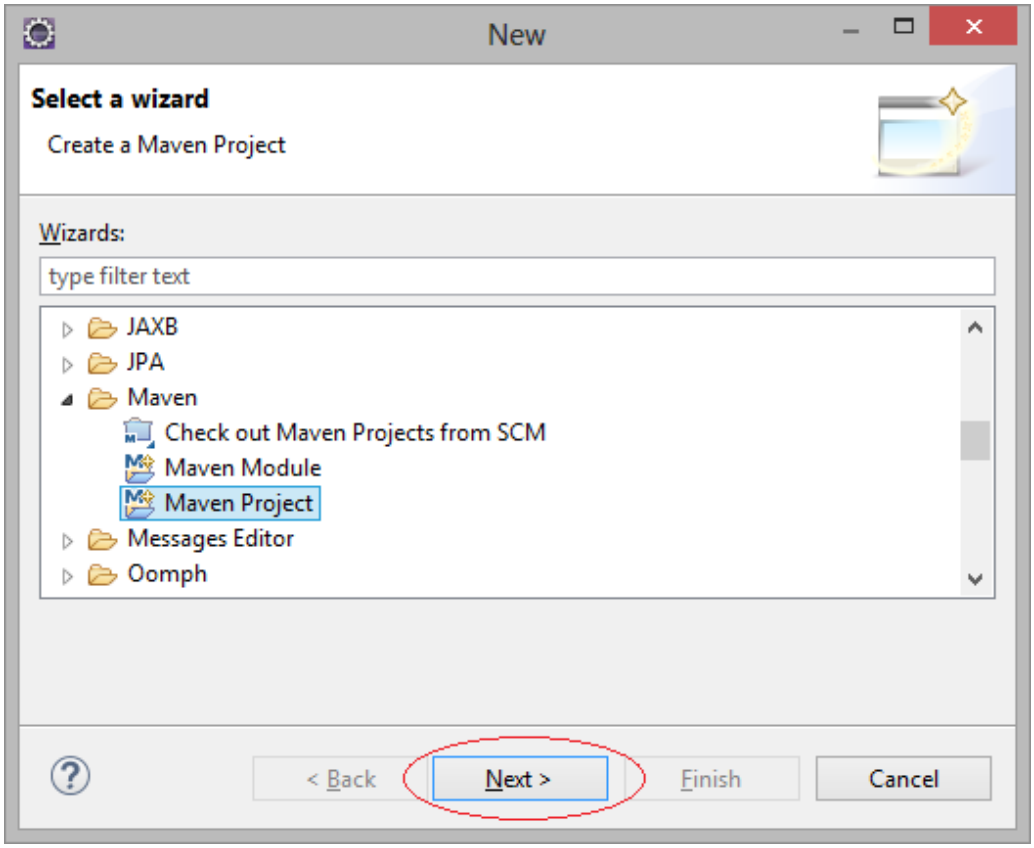


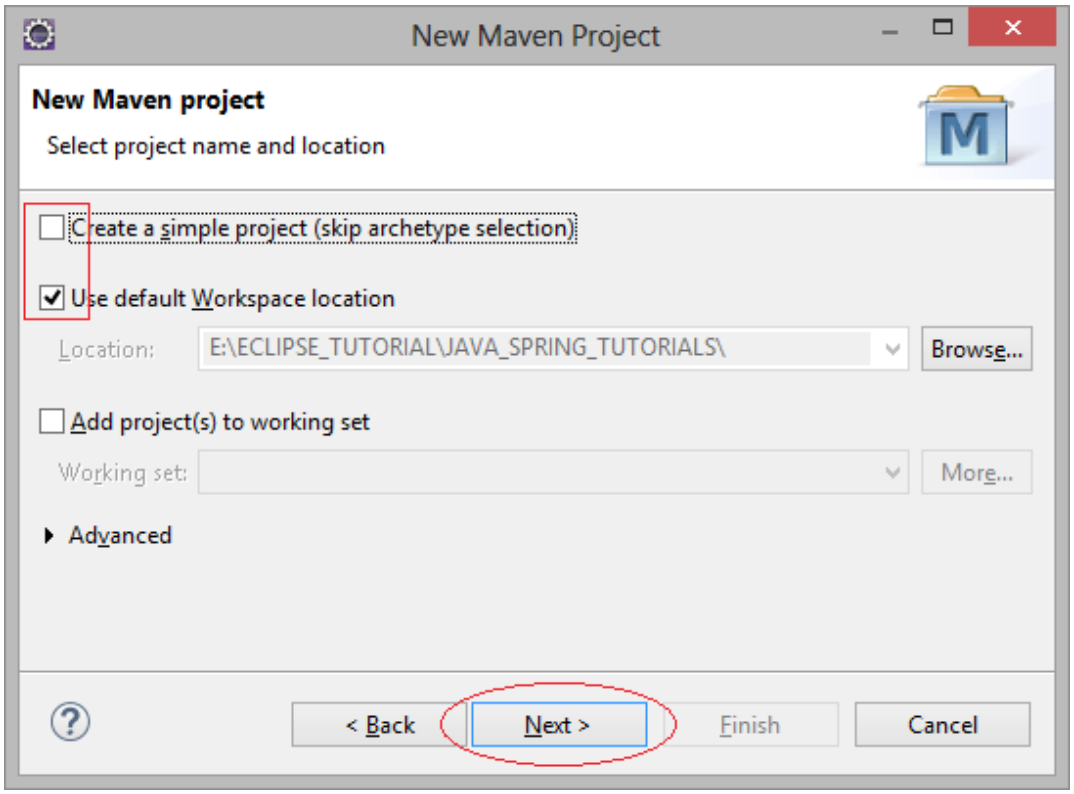
IoC Container создает объект **HelloWorldService** и объект ***HelloWorld*** потом передает ***HelloWorld*** в **HelloWorldService** через setter. Работа, которую выполняет **IoC Container** это *"внедрение зависимости"* (**Dependency Injection**) в **HelloWorldService** . Зависимость здесь означает зависимость между объектами: **HelloWorldService** и *HelloWorld*.

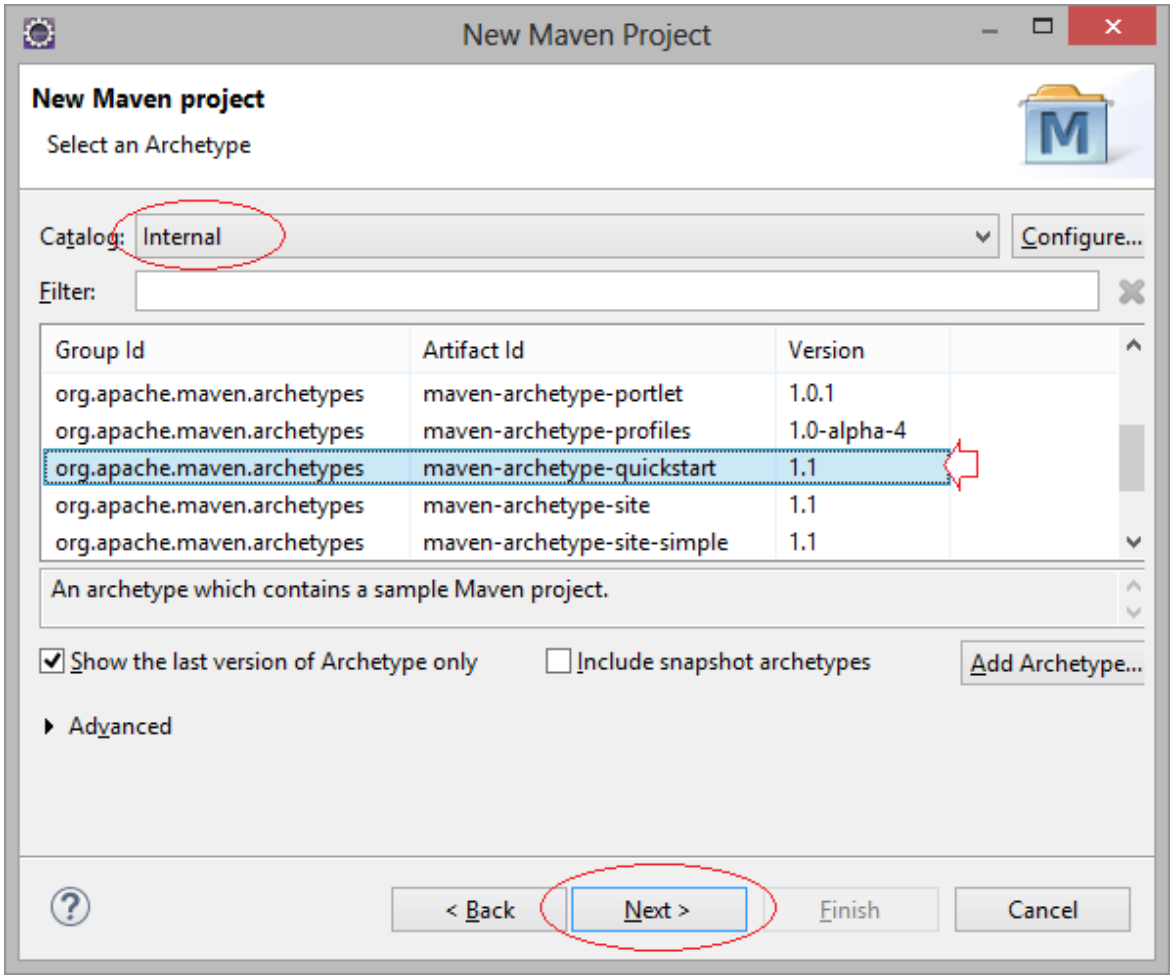
Здесь мы можем ясно определить что такое **IoC**& **DI**. Давайте вместо выполним пример **HelloWorld** чтобы лучше понять.

3- Создать Maven project

- **File/New/Other...**

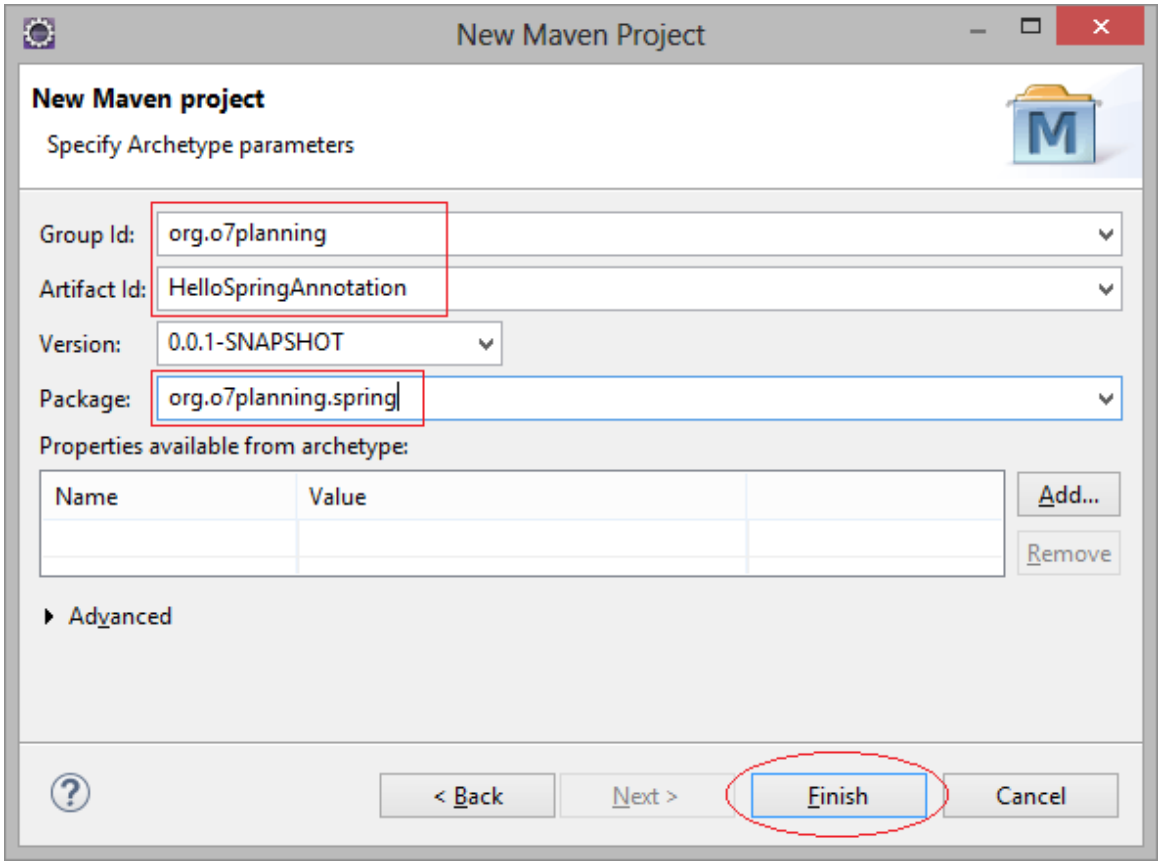

New
Select a wizard
Create a Maven Project
Wizards:
type filter text
JAXB
JPA
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
Messages Editor
Oomph
< Back
Next >
Finish
Cancel


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
E:\ECLIPSE_TUTORIAL\JAVA_SPRING_TUTORIALS\
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel

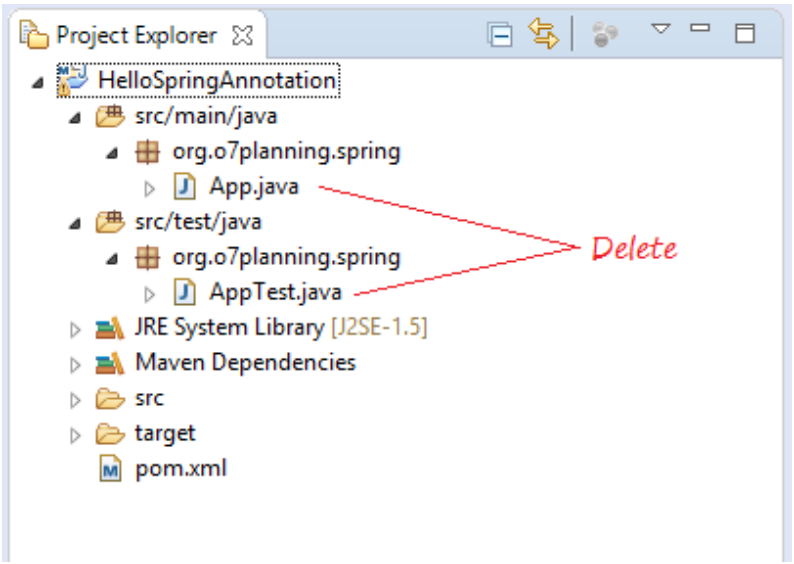


Ввести:

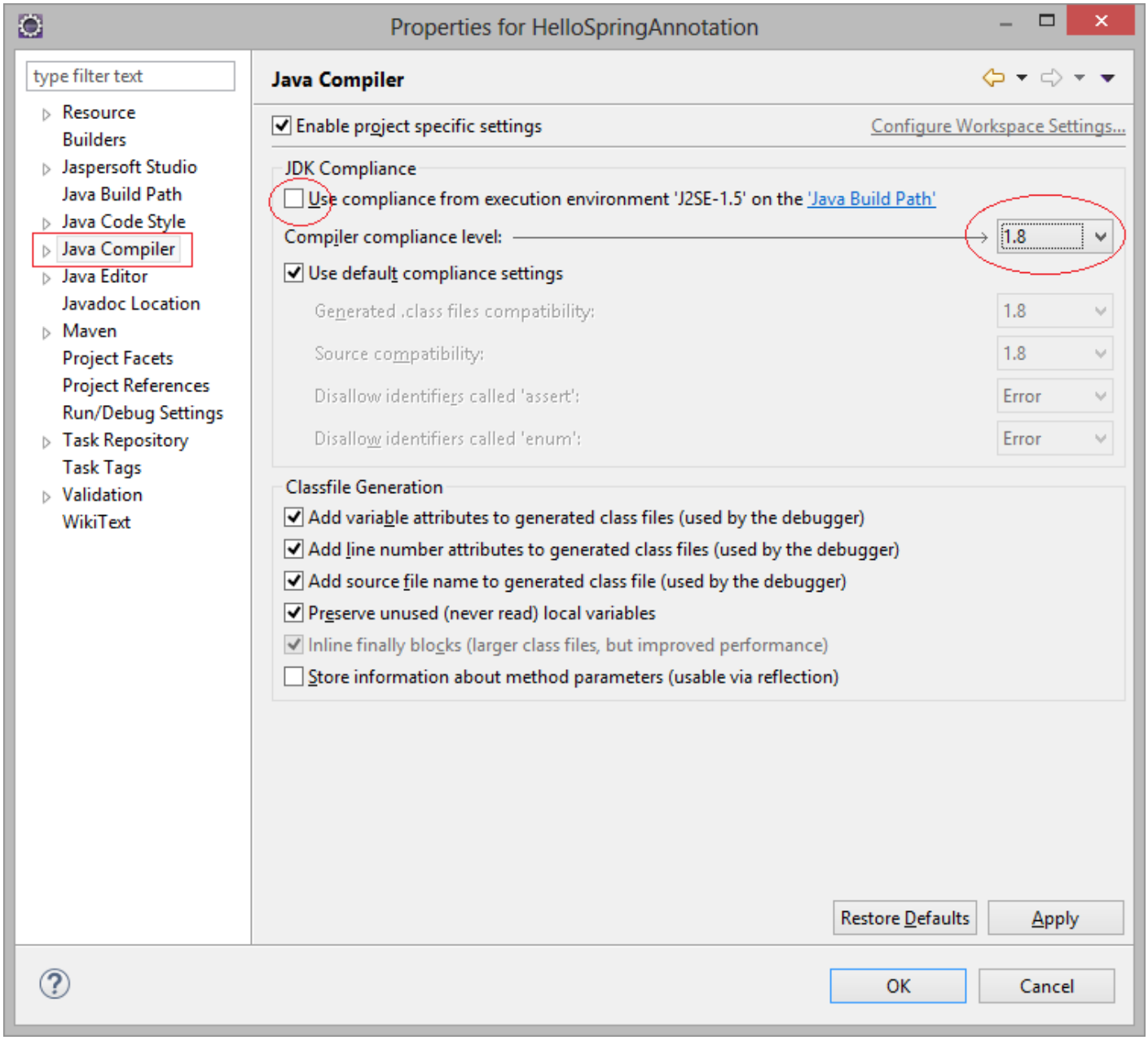
- **Group Id:** org.o7planning
- **Artifact Id:** HelloSpringAnnotation
- **package:** org.o7planning.spring

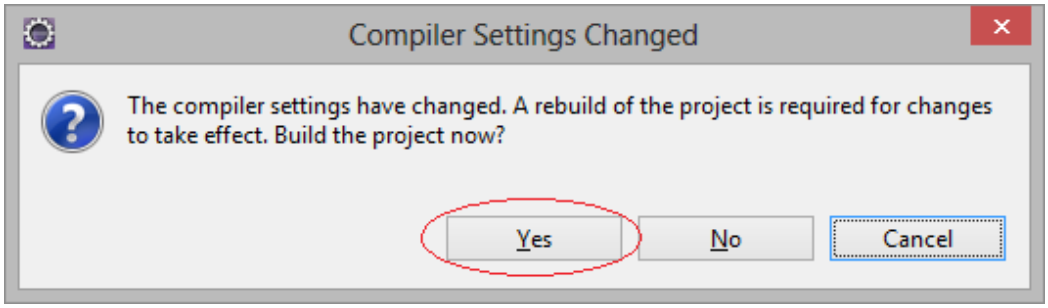


Ваш Project создан:



Удостоверьтесь что ваш Project **build** (построен) на **Java 7** или новее. Нажмите на правую кнопку мыши на project выберите **Properties**.





4- Объявить основные библиотеки Spring

Это пример **HelloWorld Spring**, поэтому мы будет использовать основную (Core) библиотеку **Spring**. Открыть файл ***pom.xml*** объявить библиотеку использования:

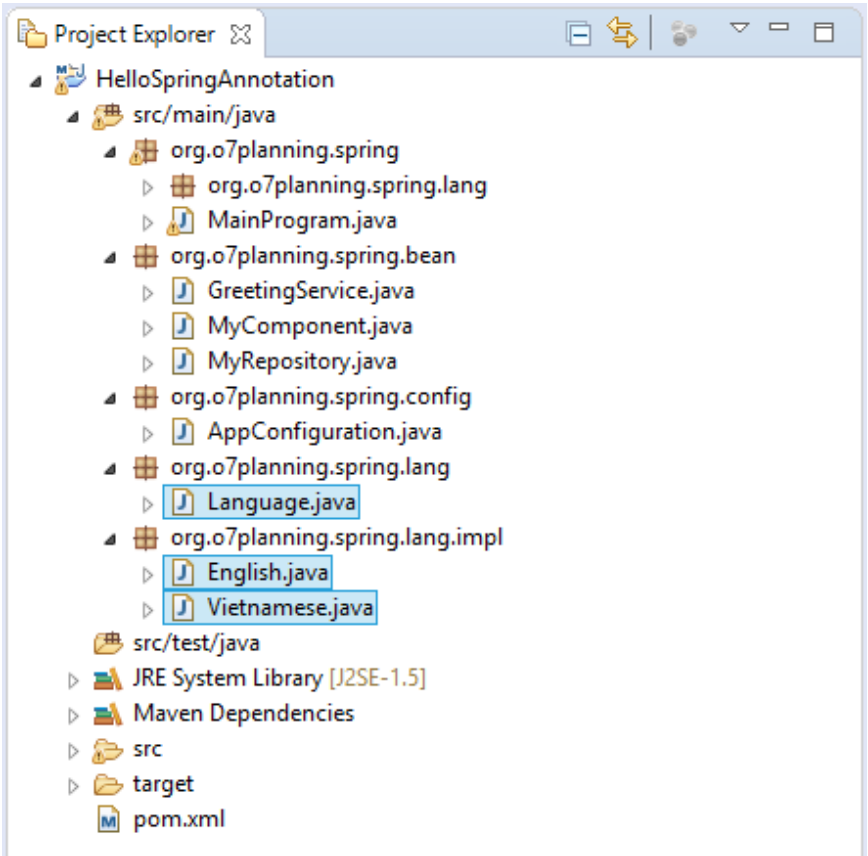
pom.xml

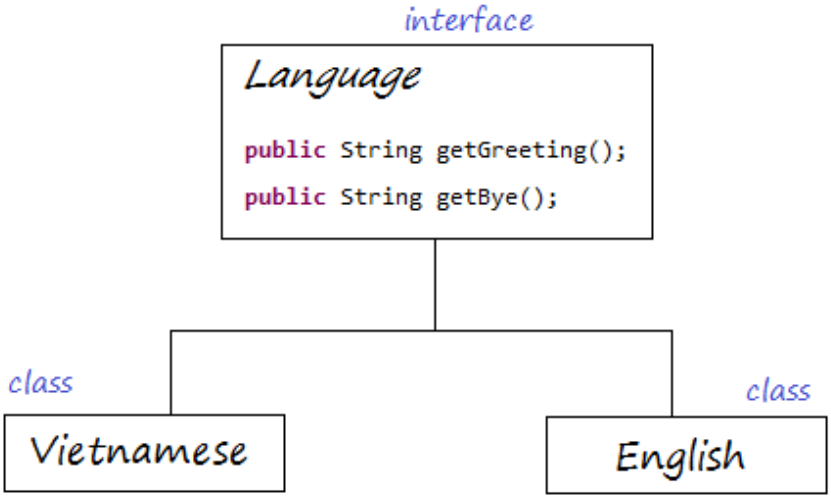
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.o7planning</groupId>
  <artifactId>HelloSpringAnnotation</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>HelloSpringAnnotation</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

    <dependencies>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

```

```
28        <!-- Spring Core -->
29        <!-- http://mvnrepository.com/artifact/org.springframework/spring-core -->
30        <dependency>
31           <groupId>org.springframework</groupId>
32           <artifactId>spring-core</artifactId>
33           <version>4.1.4.RELEASE</version>
34        </dependency>
35
36        <!-- Spring Context -->
37        <!-- http://mvnrepository.com/artifact/org.springframework/spring-context -->
38        <dependency>
39           <groupId>org.springframework</groupId>
40           <artifactId>spring-context</artifactId>
41           <version>4.1.4.RELEASE</version>
42        </dependency>
43
44     </dependencies>
45
46  </project>
```

5- Code Project

Ниже является изображение структуры project:





Language.java

```java
package org.o7planning.spring.lang;

// A Language
public interface Language {

    // Get a greeting
    public String getGreeting();

    // Get a bye
    public String getBye();

}
```

English.java

```java
package org.o7planning.spring.lang.impl;

import org.o7planning.spring.lang.Language;

// English
public class English  implements Language {

 @Override
 public String getGreeting() {
     return "Hello";
 }

 @Override
 public String getBye() {
     return "Bye bye";
 }


}
```

Vietnamese.java

```java
package org.o7planning.spring.lang.impl;

import org.o7planning.spring.lang.Language;

// Vietnamese
public class Vietnamese implements Language {

    @Override
    public String getGreeting() {
        return "Xin Chao";
    }

    @Override
    public String getBye() {
        return "Tam Biet";
    }

}
```

@Service это annotation, который используется для аннотации на одном классе чтобы сказать **Spring,** что этот класс является **Spring BEAN**.

@Autowired аннотирован на поле (field) чтобы сказать **Spring** что нужно вколоть (inject) значение в это поле. Примечание: вколоть здесь означает наподобии прикреплении значения этому полю.

GreetingService.java

```
1  package org.o7planning.spring.bean;
2
3  import org.o7planning.spring.lang.Language;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.stereotype.Service;
6
7  @Service
8  public class GreetingService {
9
10   @Autowired
11   private Language language;
12
13   public GreetingService() {
14
15   }
16
17   public void sayGreeting() {
18
19      String greeting = language.getGreeting();
20
21      System.out.println("Greeting: " + greeting);
22   }
23
24 }
```

@Repository это annotation, который используется для аннотации на классе чтобы сказать **Spring** что этот класс является **Spring BEAN**.

MyRepository.java

```
1  package org.o7planning.spring.bean;
2
3  import java.util.Date;
4
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public class MyRepository {
9
10    public String getAppName() {
11       return "Hello Spring App";
12    }
13
14    public Date getSystemDateTime() {
15       return new Date();
16    }
17
18
19 }
```

@Component это annotation, который используется для аннотации на классе чтобы сказать **Spring** что этот класс является **Spring BEAN**.

@Autowired аннотирован на поле (field) чтобы сказать **Spring** что нужно вколоть (inject) значение в это поле. Примечание: вколоть здесь означает наподобии прикреплении значения этому полю.

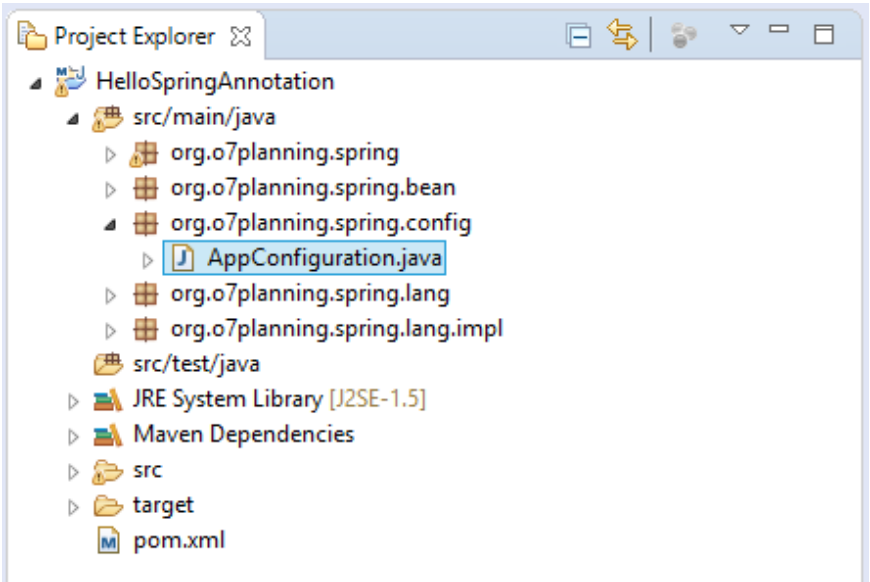
MyComponent

```
1  package org.o7planning.spring.bean;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Component;
5
6  @Component
7  public class MyComponent {
8
9     @Autowired
10    private MyRepository repository;
```

```
11
12      public void showAppInfo() {
13          System.out.println("Now is: "+ repository.getSystemDateTime());
14          System.out.println("App Name: "+ repository.getAppName());
15      }
16
17  }
```

*Нет разницы в способе использования **@Service**, **@Component** и **@Repository**, вы можете использовать для аннотации на вашем классе, который соответствует значению и контексту приложения.*

6- Spring @Configuration & IoC



@Configuration это annotation, который аннотируется на классе, этот класс определяет **Spring BEAN**.

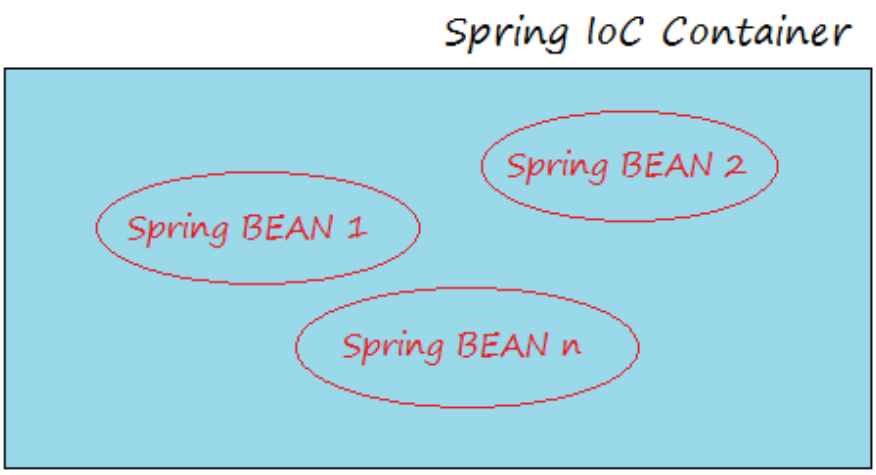
@ComponentScan - Говорит Spring про пакеты для поиска других **Spring BEAN**, **Spring** сканирует (scan) эти пакеты для поиска.

AppConfiguration.java

```
1  package org.o7planning.spring.config;
2
3  import org.o7planning.spring.lang.Language;
4  import org.o7planning.spring.lang.impl.Vietnamese;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan({"org.o7planning.spring.bean"})
public class AppConfiguration {

    @Bean(name ="language")
    public Language getLanguage() {

        return new Vietnamese();
    }

}
```

Созданные **Spring BEAN** будут управляемы в **Spring IoC Container** (Контейнер Spring IoC).

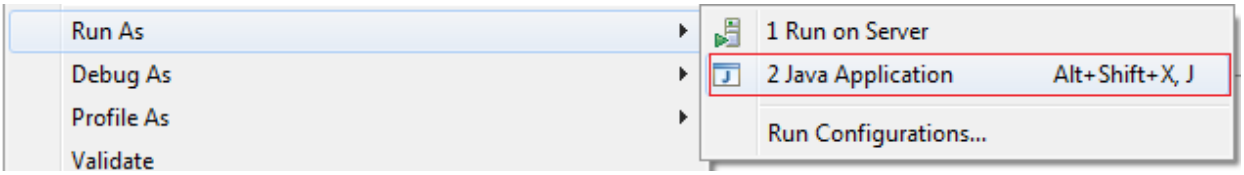


7- Spring ApplicationContext

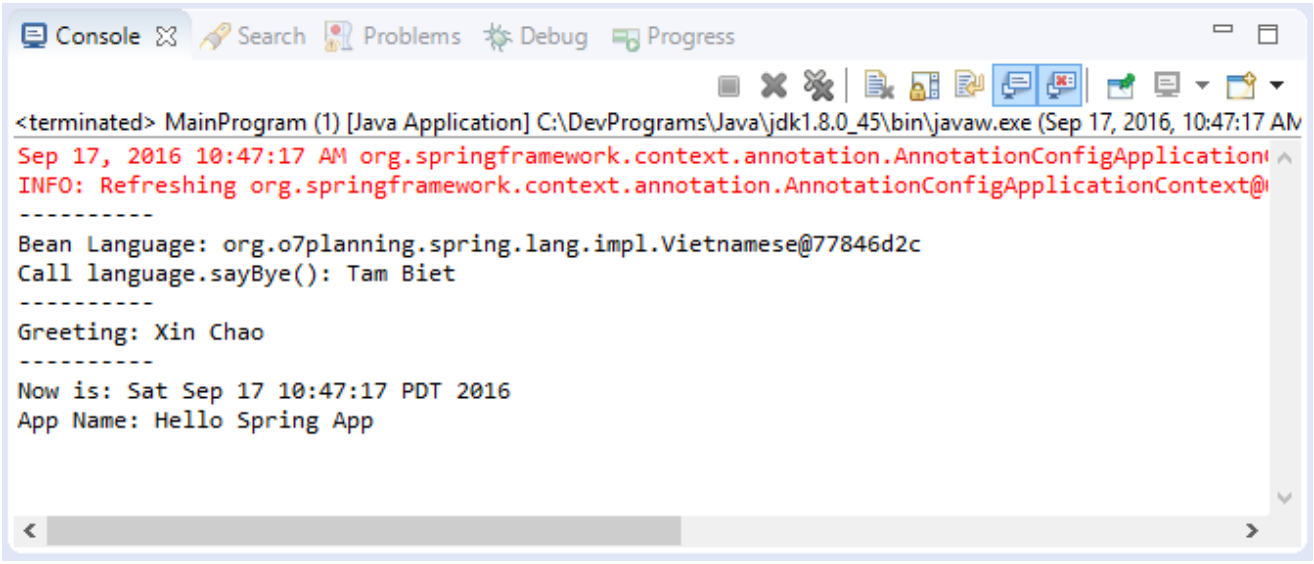
MainProgram.java

```
package org.o7planning.spring;

import org.o7planning.spring.bean.GreetingService;
import org.o7planning.spring.bean.MyComponent;
import org.o7planning.spring.config.AppConfiguration;
import org.o7planning.spring.lang.Language;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainProgram {

   public static void main(String[] args) {

      // Creating a Context Application object by reading
      // the configuration of the 'AppConfiguration' class.
      ApplicationContext context = new AnnotationConfigApplicationContext(AppConfiguration.class);

      System.out.println("----------");
      Language language = (Language) context.getBean("language");

      System.out.println("Bean Language: "+ language);
      System.out.println("Call language.sayBye(): "+ language.getBye());

      System.out.println("----------");

      GreetingService service = (GreetingService) context.getBean("greetingService");


      service.sayGreeting();

      System.out.println("----------");

      MyComponent myComponent = (MyComponent) context.getBean("myComponent");

      myComponent.showAppInfo();

   }

}
```

Запустите класс **MainProgram**



Результаты:

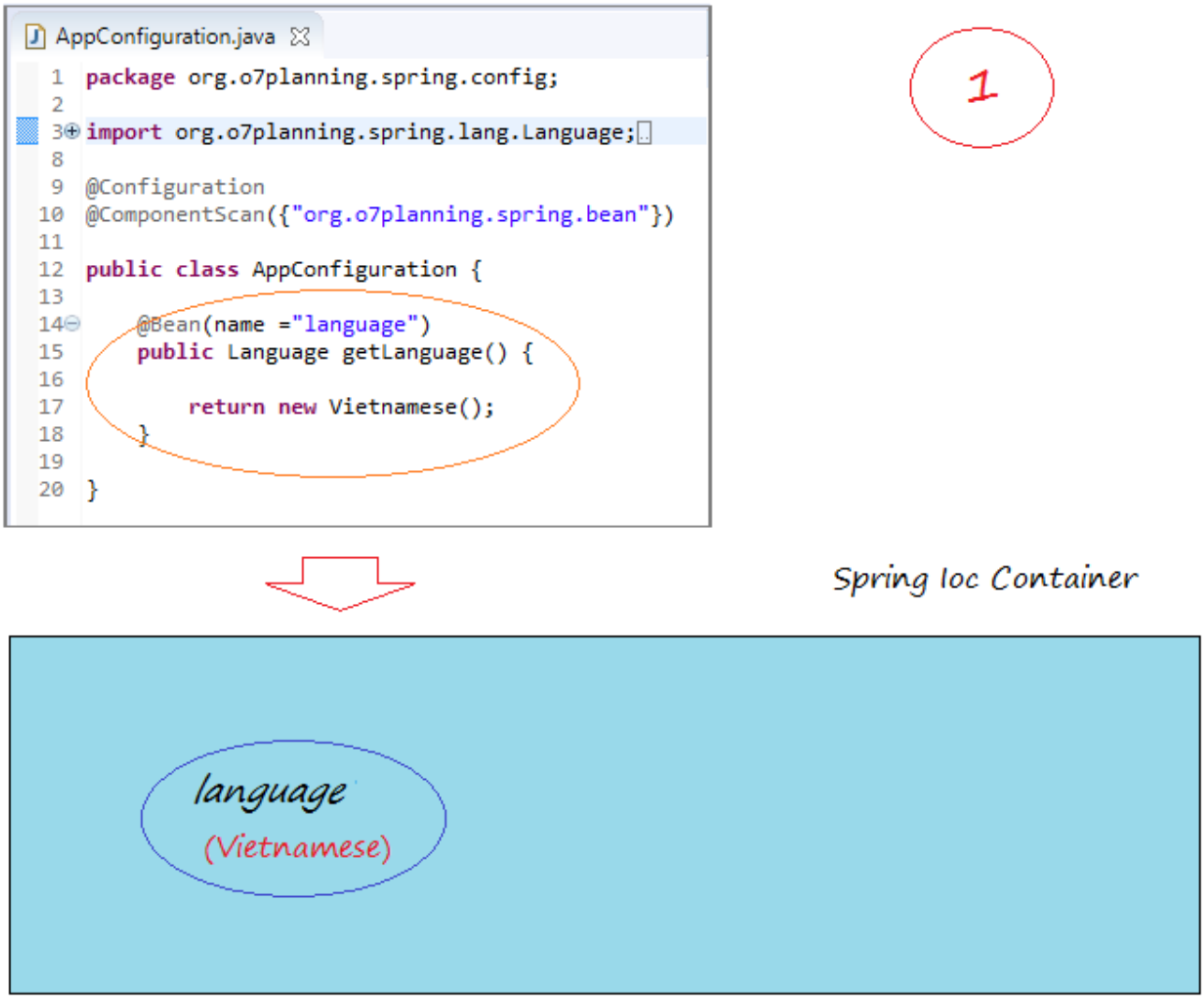


8- Правило работы Spring

Вы создаете объект **ApplicationContext** читая конфигурацию в классе **AppConfiguration**, как в коде ниже.

```
1 ApplicationContext context = new AnnotationConfigApplicationContext(AppConfiguration.class);
```

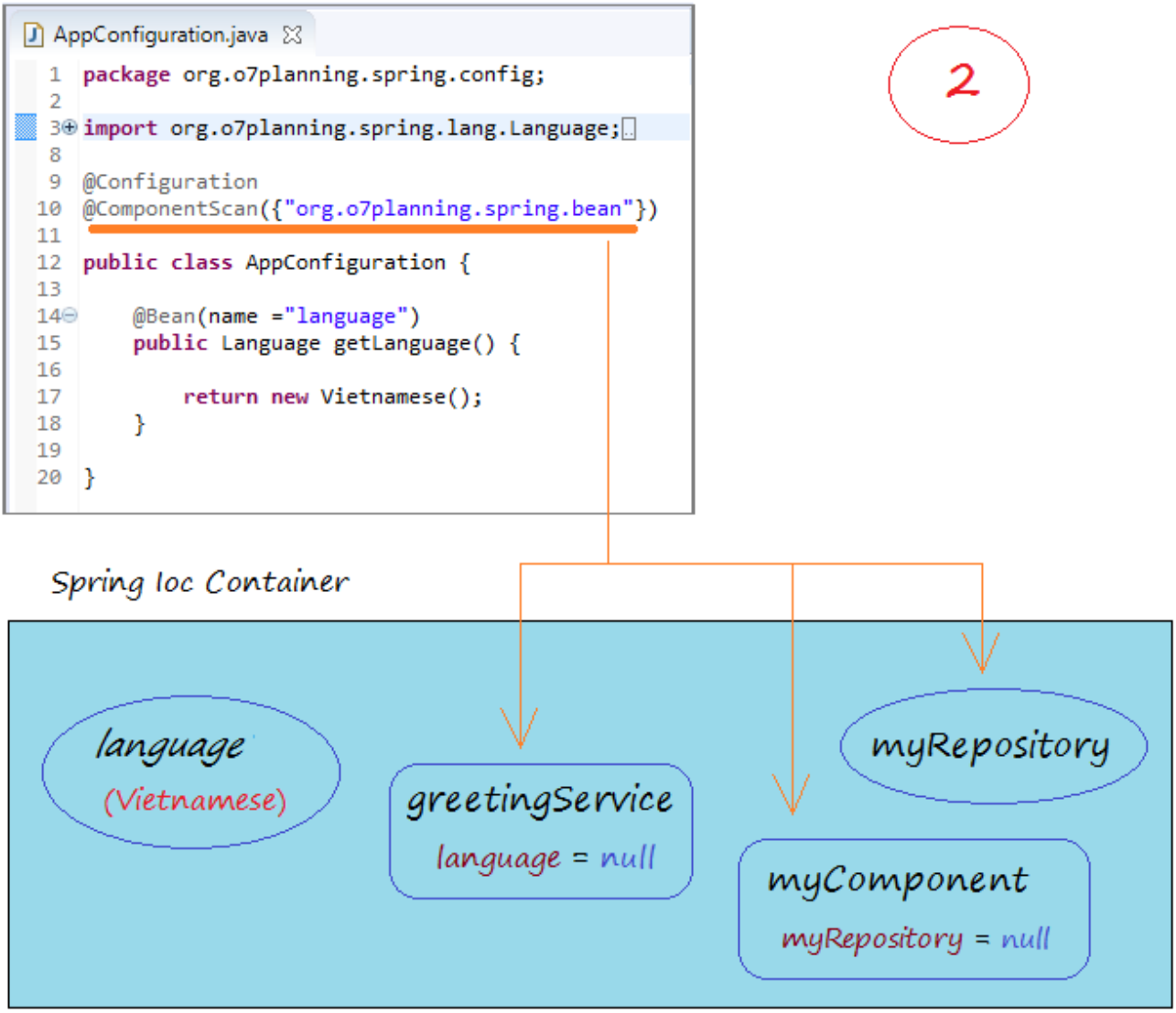
Spring создает **Spring BEAN,** в соответствии с определениями в классе **AppConfiguration**, (Примечание: Класс **AppConfiguration** должен быть аннотирован с помощью **@Configuration**).



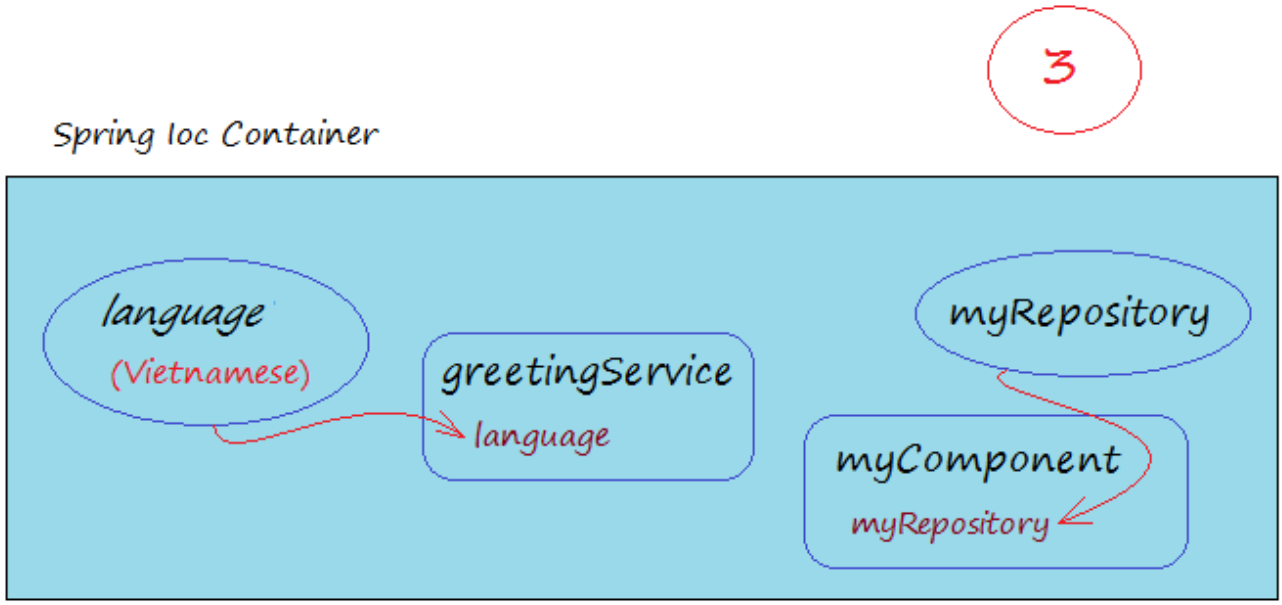
Далее, **Spring** ищет в пакете **"org.o7planning.spring.bean"** чтобы создать другие **Spring BEAN**, (Создать объекты из класса аннотированные с помощью **@Service**, **@Component** или **@Repository**).

AppConfiguration.java

```java
package org.o7planning.spring.config;

import org.o7planning.spring.lang.Language;

@Configuration
@ComponentScan({"org.o7planning.spring.bean"})

public class AppConfiguration {

    @Bean(name ="language")
    public Language getLanguage() {

        return new Vietnamese();
    }

}
```



Теперь **Spring BEAN** созданы и содержатся в **Spring IoC**. Поля **Spring BEAN** аннотированые с **@Autowired** значения будут введены, как в изображении ниже:



> *Вернемся к вопросу* ***"Что такое IoC?"***.

> *По традиционному способу объект создан из класса, его поля (field) будут иметь значения прикрепленные изнутри этого класса. **Spring** сделал обратное традиционному способу, объекты созданы и некоторые его поля имеют значения, которые вкололи снаружи, так называемым **IoC**.*
>
> ***IoC** это аббревиатура **"Inversion of Control"** - Значит **"Инверсия управления"**.*
>
> ***IoC Container** это контейнер содержащий все **Spring BEAN** используемые приложением.*

9- Программирование веб приложения используя Spring Boot

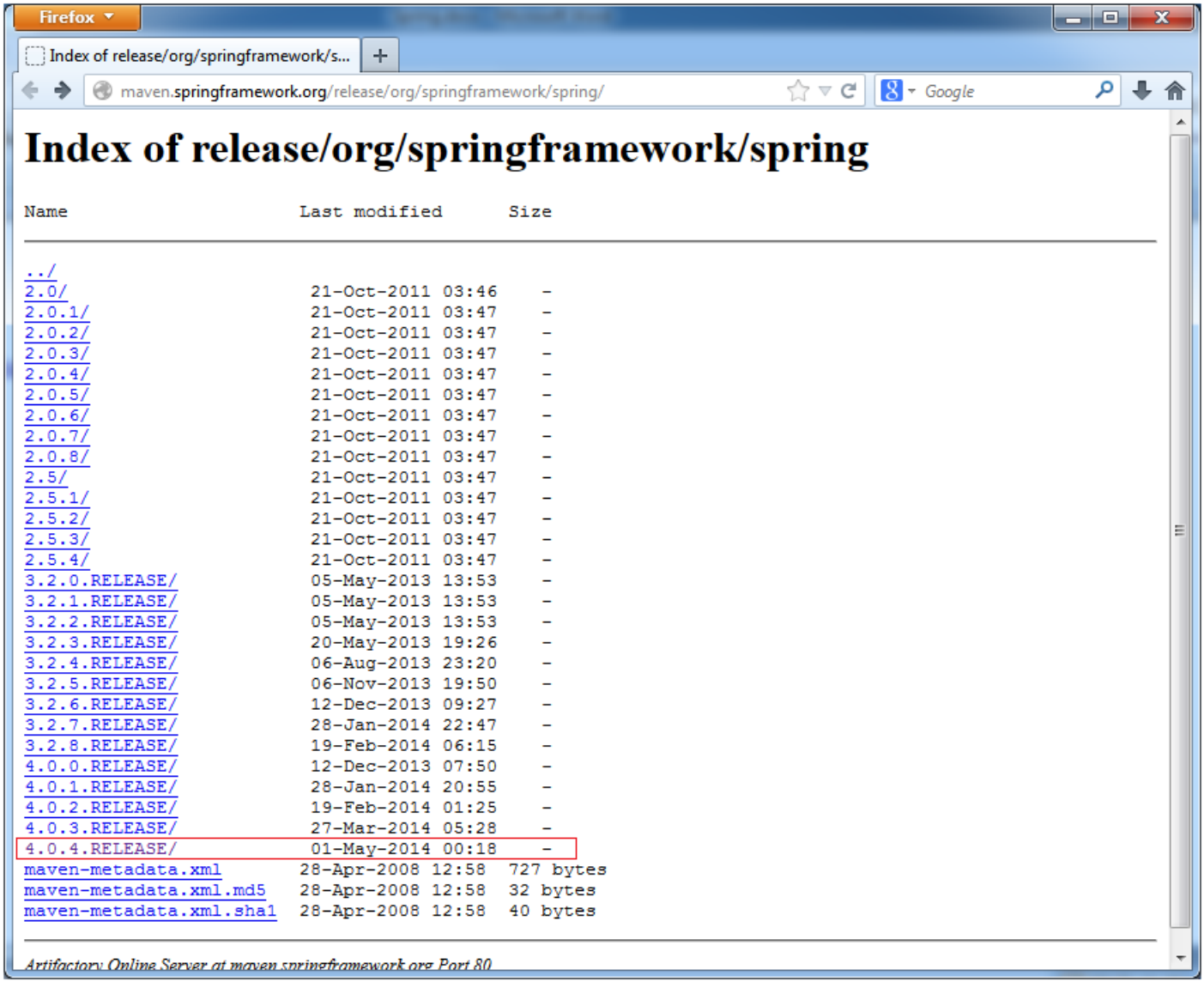
Далее вы можете узнать про программирование веб приложения с **Spring Boot**:

- Руководство Spring Boot для начинающих

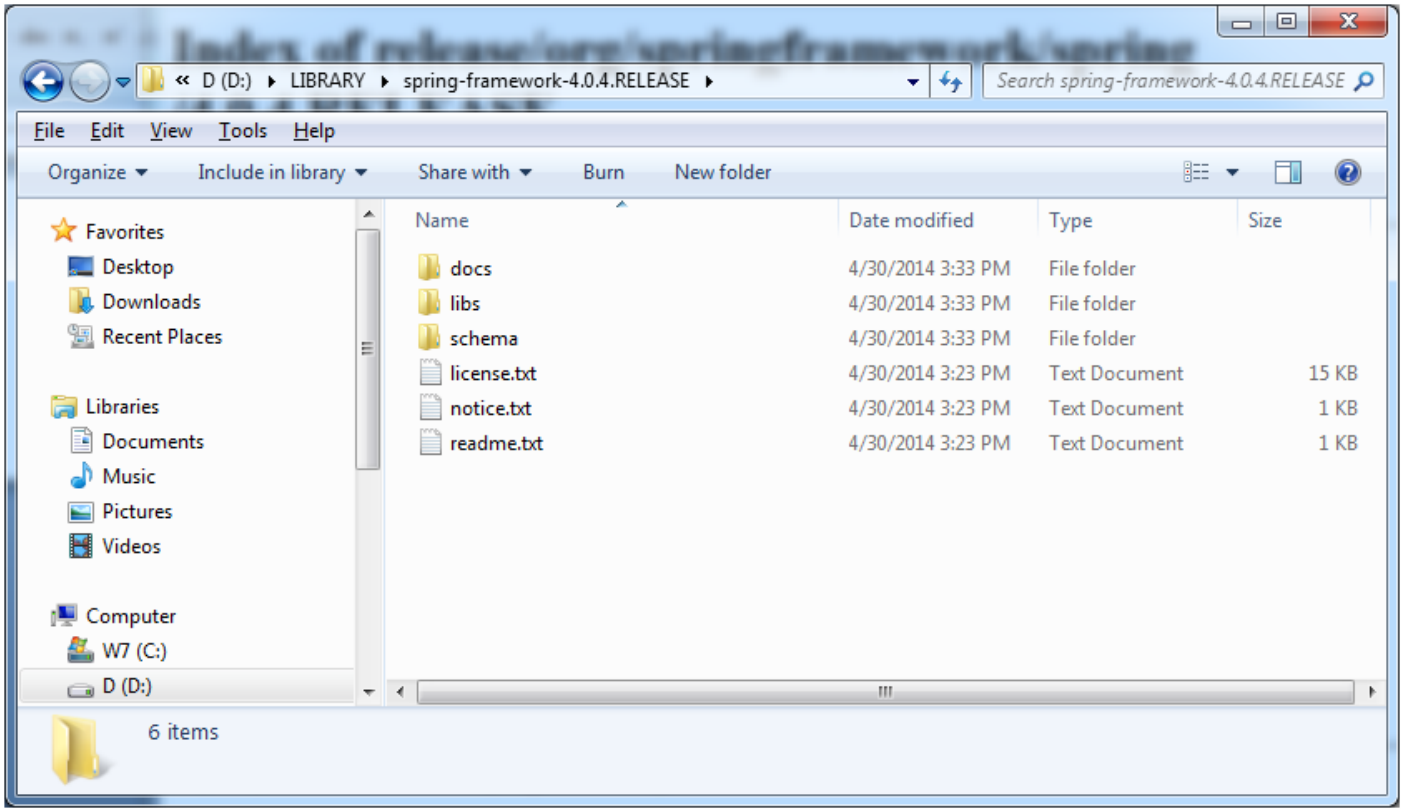
10- Аппендикс: Download библиотека Spring

Вы можете скачать **Spring** по ссылке:

- http://maven.springframework.org/release/org/springframework/spring/



Извлеки скачанный файл **zip** в папку жесткого диска:



View more Tutorials:

Руководства Spring MVC

Руководства Spring Boot

Руководства Spring MVC

- Руководство Spring для начинающих
- Руководство Spring Boot для начинающих

- Руководство Spring MVC для начинающих
- Установка Spring Tool Suite в Eclipse
- Настройка статических ресурсов в Spring MVC
- Руководство Spring MVC Interceptor
- Создание многоязычного веб-приложения с использованием Spring MVC
- Руководство Spring MVC File Upload/Download
- Руководство Spring JDBC
- Простой веб-приложение Java с использованием Spring MVC, Spring Security и Spring JDBC
- Социальный вход в Spring MVC с Spring Social Security
- Руководство Spring MVC и FreeMarker
- Руководство Spring MVC и Apache Tiles
- Использование нескольких DataSources в Spring MVC
- Руководство Spring MVC Form и Hibernate
- Запуск фоновых задач в Spring
- Создание Java Корзина веб-приложения с использованием Spring MVC и Hibernate
- Простой пример CRUD с Spring MVC RESTful Web Service
- Развертывание Spring MVC на сервере Oracle WebLogic Server

Руководства Spring Boot

- Установка Spring Tool Suite в Eclipse
- Руководство Spring для начинающих
- Руководство Spring Boot для начинающих
- Общие свойства Spring Boot
- Руководство Spring Boot и Thymeleaf
- Руководство Spring Boot и FreeMarker
- Руководство Spring Boot и Groovy
- Руководство Spring Boot и Mustache
- Руководство Spring Boot и JSP
- Руководство Spring Boot, Apache Tiles, JSP
- Мониторинг приложения с помощью Spring Boot Actuator
- Создание веб-приложения с несколькими языками с помощью Spring Boot
- Использование нескольких ViewResolver в Spring Boot
- Использование Twitter Bootstrap в Spring Boot
- Руководство Spring Boot, Spring JDBC и Spring Transaction
- Руководство Spring JDBC
- Руководство Spring Boot, JPA и Spring Transaction
- Руководство Spring Boot и Spring Data JPA
- Руководство Spring Boot, Hibernate и Spring Transaction
- Интеграция Spring Boot, JPA и H2 Database
- Руководство Spring Boot и MongoDB
- Использование нескольких DataSource с Spring Boot и JPA

- Использование нескольких DataSource с Spring Boot и RoutingDataSource
- Создайте приложение для входа с Spring Boot, Spring Security, Spring JDBC
- Создайте приложение для входа с Spring Boot, Spring Security, JPA
- Создайте приложение регистрации пользователей с помощью Spring Boot, Spring Form Validation
- Войти в систему используя социальную сеть с OAuth2 в Spring Boot
- Запуск фоновых задач в Spring
- Пример CRUD Restful Web Service с Spring Boot
- Пример Spring Boot Restful Client с RestTemplate
- Пример CRUD с Spring Boot, REST и AngularJS
- Защита Spring Boot RESTful Service используя Basic Authentication
- Защита Spring Boot RESTful Service используя Auth0 JWT
- Пример Upload file с Spring Boot
- Пример Download file с Spring Boot
- Пример Upload file с Spring Boot и jQuery Ajax
- Пример Upload file с Spring Boot и AngularJS
- Создание веб-приложения для корзины покупок с помощью Spring Boot, Hibernate
- Пример Thymeleaf Form Select option
- Руководство Spring Email
- Создайте простое приложение Chat с Spring Boot и Websocket
- Развертывание приложения Spring Boot на Tomcat Server
- Развертывание приложения Spring Boot на Oracle WebLogic Server

Самые новые руководства

- Понимание ECMAScript Iterable và Iterator (ES6)
- Обработка ошибок в ECMAScript (ES6)
- Наследование и полиморфизм в ECMAScript (ES6)
- Классы и объекты в ECMAScript (ES6)
- Руководство ECMAScript Date
- Руководство ECMAScript Module (ES6)
- Циклы в ECMAScript (ES6)
- Руководство ECMAScript Function (ES6)
- Руководство ECMAScript Symbol (ES6)
- Понимание Duck Typing в ECMAScript (ES6)

- **Spring MVC**
- **How To Install Ubuntu**
- **Java Programming Courses**