

# JAVA II

Spring web framework configuration

# Initial project set up

- Go to <https://start.spring.io/>
- Generate project as in screen below

Generate a Gradle Project ▾ with Java ▾ and Spring Boot 2.1.1 ▾

### Project Metadata

Artifact coordinates

Group

Artifact

Name

Description

Package Name

Packaging

Java Version

too many options? [Switch back to the simple version.](#)

### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

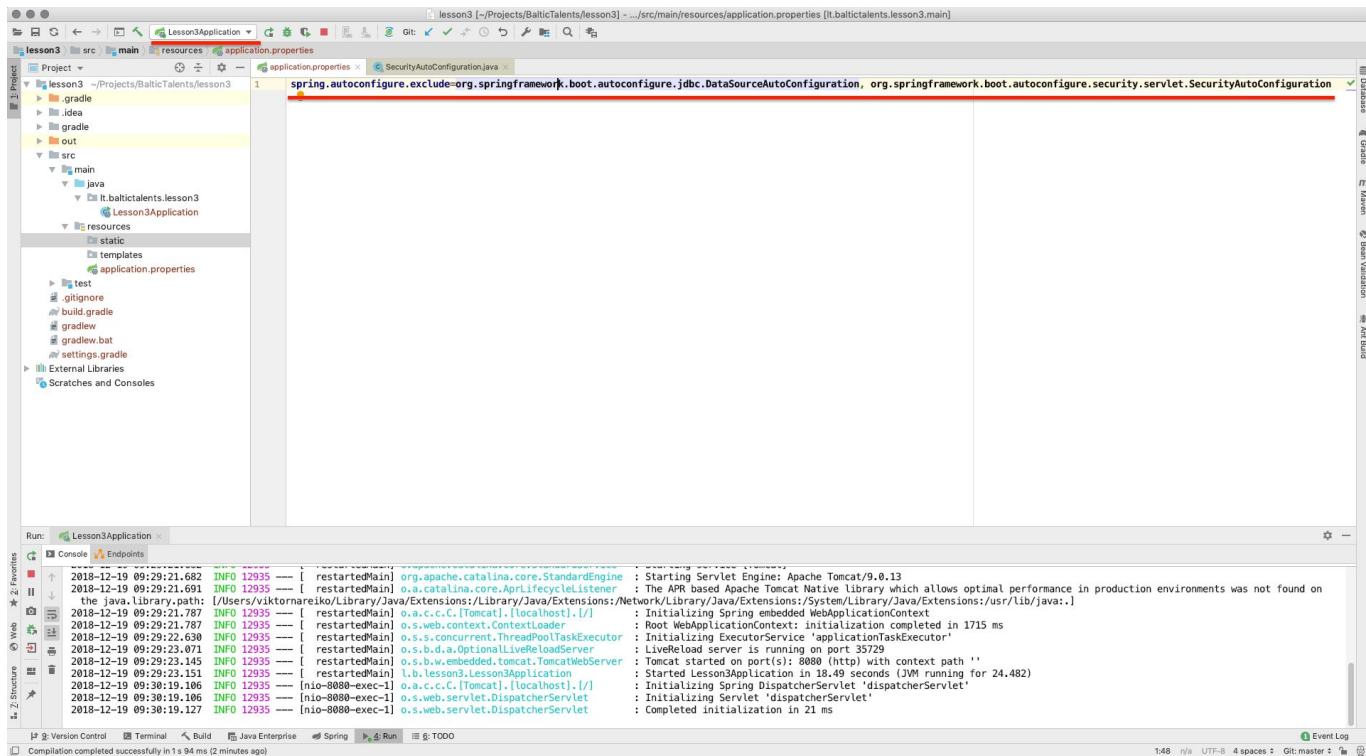
  

Selected Dependencies

Web × Rest Repositories × MySQL × JPA × Security ×  
DevTools × Validation × Session × Lombok ×

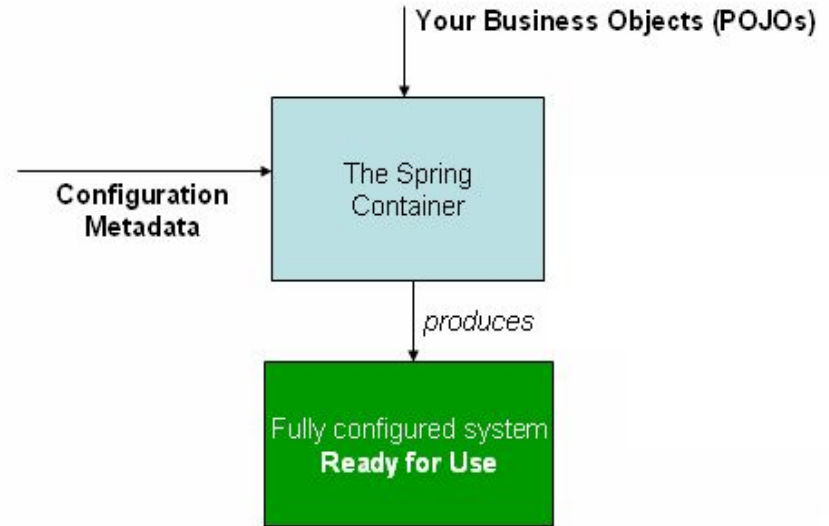
Generate Project ⌘ + ⌘

# Initial project set up



# The Spring IoC (DI) Container and Beans

IoC is also known as dependency injection (DI). It is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates them (in Spring such objects are called beans).



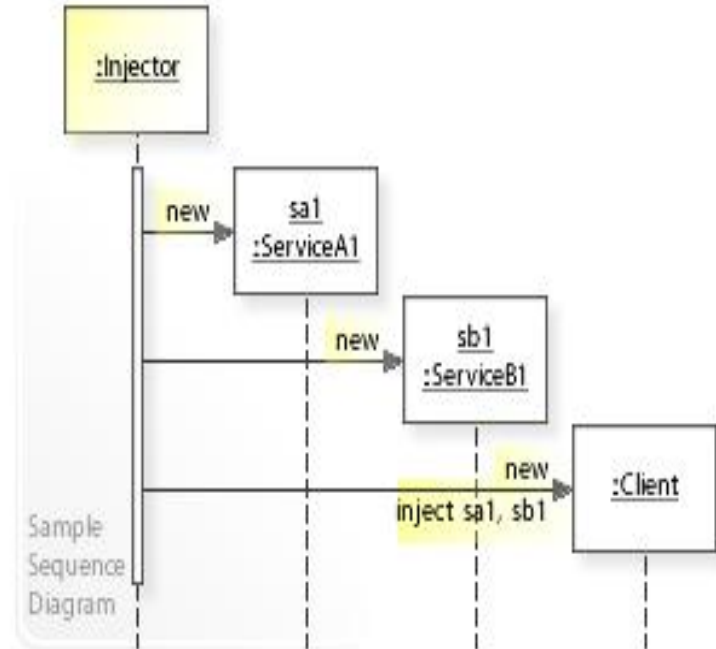
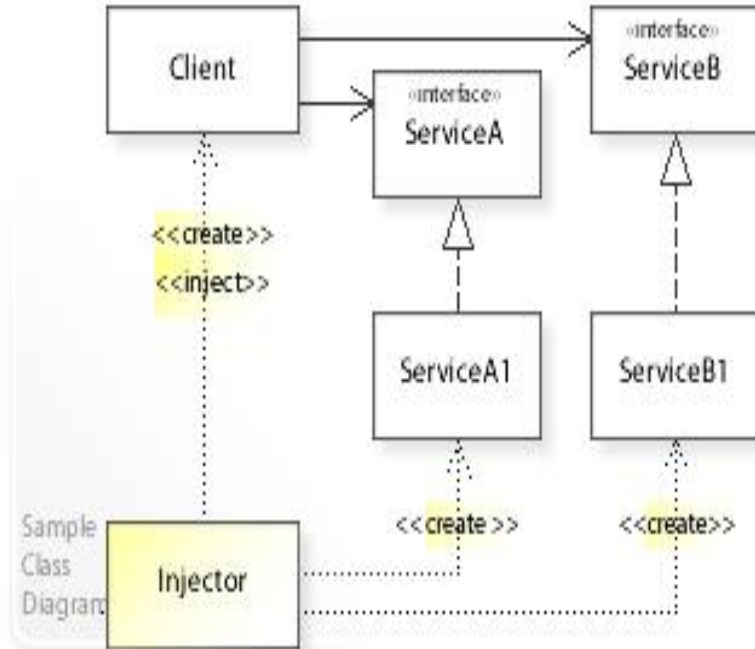
# Dependency Injection pattern

According to JSR330 the injection is done in the following order:

- constructor injection
- field injection
- method injection

Right now, we will speak only about constructor injection

# Dependency Injection



# Spring framework - configuration metadata

Configuration Metadata the Spring IoC container consumes a form of configuration metadata. This configuration metadata represents how you, as an application developer, tell the Spring container to instantiate, configure, and assemble the objects in your application. Configuration metadata is supplied:

- XML based configuration
- Java annotation
- Java configuration

# Spring framework - xml based metadata configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="..." class="..."> ❶ ❷
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions go here -->

</beans>
```

- ❶ The `id` attribute is a string that identifies the individual bean definition.
- ❷ The `class` attribute defines the type of the bean and uses the fully qualified classname.



# XML Bean configuration examples (Single)

```
<bean id="accountDao"
```

```
    class="lt.baltictalents.lesson3.beans.example.dao.AccountDao">
```

```
    <!-- additional collaborators and configuration for this bean go here -->
```

```
</bean>
```

```
<bean id="userDao" class="lt.baltictalents.lesson3.beans.example.dao.UserDao">
```

```
    <constructor-arg type="java.lang.String" name="name" value="Viktor"/>
```

```
    <constructor-arg type="java.lang.String" name="surname" value="Nareiko"/>
```

```
    <constructor-arg type="int" name="age" value="32"/>
```

```
</bean>
```

# XML Bean configuration examples (Circular)

```
<bean id="allDao" class="lt.baltictalents.lesson3.beans.example.dao.AllDao">
```

```
  <property name="accountDao">
```

```
    <ref bean="accountDao"/>
```

```
  </property>
```

```
  <property name="userDao" ref="userDao"/>
```

```
</bean>
```

```
<bean id="allDaoConstructor" class="lt.baltictalents.lesson3.beans.example.dao.AllDaoConstructor">
```

```
  <constructor-arg name="accountDao" ref="accountDao"/>
```

```
  <constructor-arg name="userDao" ref="userDao"/>
```

```
</bean>
```

# XML Beans configurations examples

<http://localhost:8080/account>

<http://localhost:8080/user>

<http://localhost:8080/all>

<http://localhost:8080/all-constructor>

# More about XML configuration

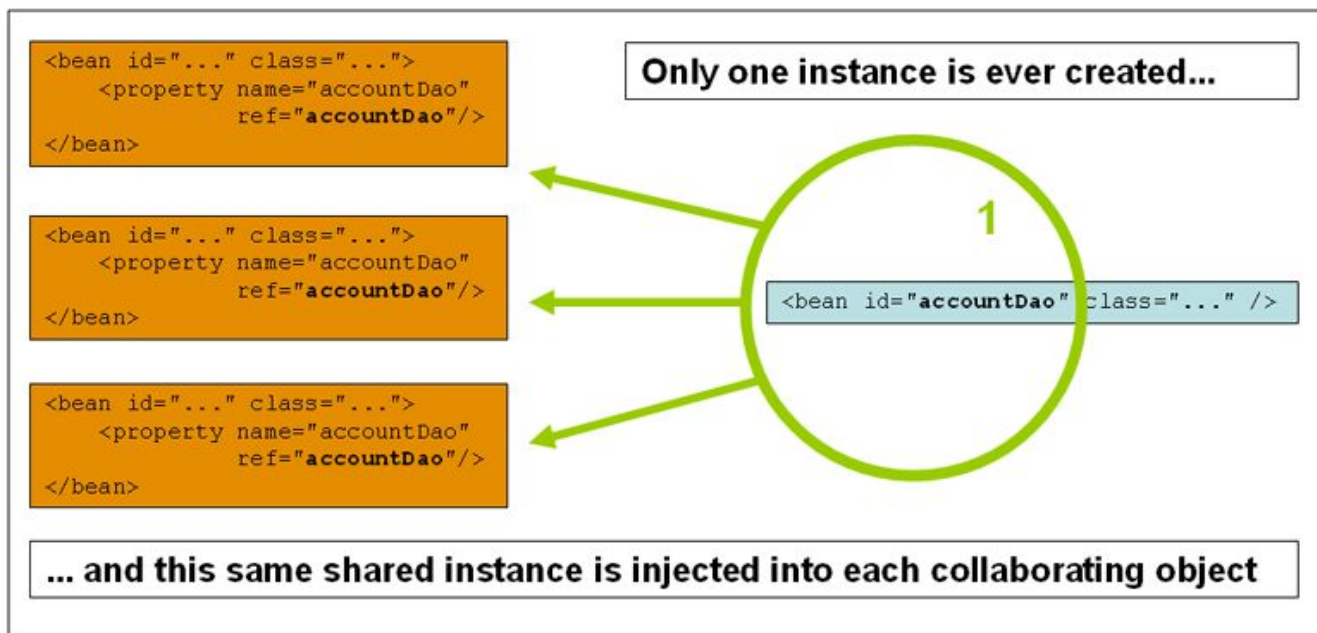
<https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#spring-core>

# Beans scopes

Scope	Description
<a href="#"><u>singleton</u></a>	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
<a href="#"><u>prototype</u></a>	Scopes a single bean definition to any number of object instances.
<a href="#"><u>request</u></a>	Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
<a href="#"><u>session</u></a>	Scopes a single bean definition to the lifecycle of an HTTP Session . Only valid in the context of a web-aware Spring ApplicationContext .
<a href="#"><u>application</u></a>	Scopes a single bean definition to the lifecycle of a ServletContext . Only valid in the context of a web-aware Spring ApplicationContext .
<a href="#"><u>websocket</u></a>	Scopes a single bean definition to the lifecycle of a WebSocket . Only valid in the context of a web-aware Spring ApplicationContext .

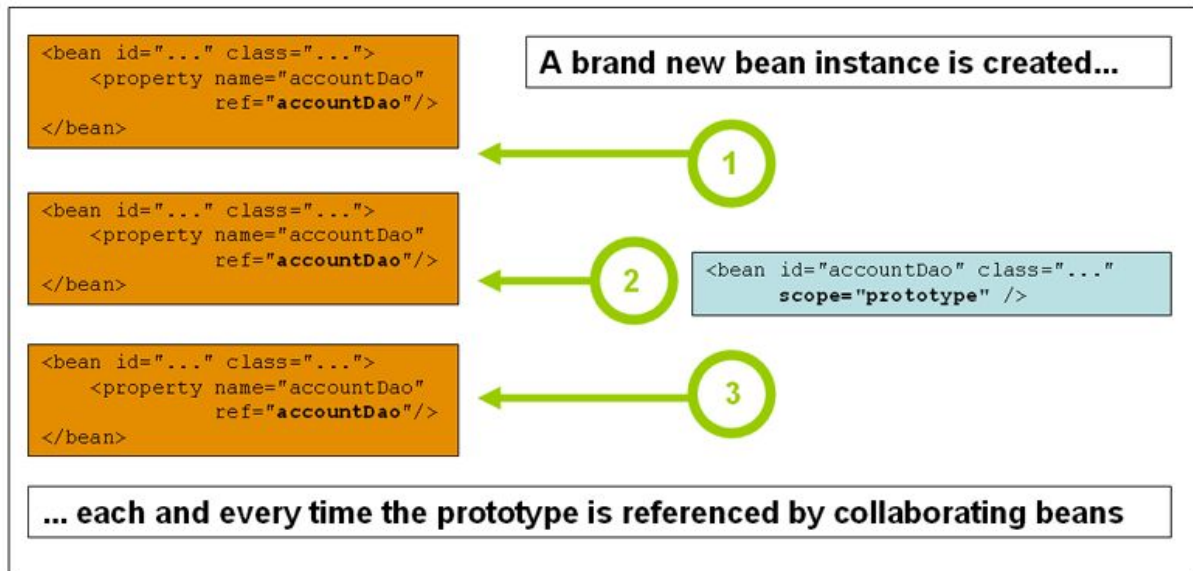
# The Singleton scope

Only one shared instance of a singleton bean is managed, and all requests for beans with an ID or IDs that match that bean definition result in that one specific bean instance being returned by the Spring container.



# The Prototype scope

The non-singleton prototype scope of bean deployment results in the creation of a new bean instance every time a request for that specific bean is made. That is, the bean is injected into another bean or you request it through a `getBean()` method call on the container. As a rule, you should use the prototype scope for all stateful beans and the singleton scope for stateless beans.



# Java based container configuration

@Bean

@Configuration

@PropertySource



# @Bean @Configuration @ComponentScan annotation

The `@Bean` annotation is used to indicate that a method instantiates, configures, and initializes a new object to be managed by the Spring IoC container. For those familiar with Spring's `<beans/>` XML configuration, the `@Bean` annotation plays the same role as the `<bean/>` element. You can use `@Bean`-annotated methods with any Spring `@Component`. However, they are most often used with `@Configuration` beans.

```
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

```
@Configuration
@ComponentScan(basePackages = "com.acme") ❶
public class AppConfig {

    ...
}
```

❶ This annotation enables component scanning.

# @Bean @Configuration @ComponentScan annotation

```
@Configuration
public class AppConfig {

    @Bean(initMethod = "init")
    public BeanOne beanOne() {
        return new BeanOne();
    }

    @Bean(destroyMethod = "cleanup")
    public BeanTwo beanTwo() {
        return new BeanTwo();
    }
}
```

```
@Configuration
public class MyConfiguration {

    @Bean
    @Scope("prototype")
    public Encryptor encryptor() {
        // ...
    }
}
```

# Java annotation based configuration

@Autowired

```
@Autowired  
BeansGetter beansGetter;
```

@Required

```
@Autowired  
AccountDao accountDaoAnnotated;
```

@Primary

```
@Autowired  
MessageService emailService;
```

@Qualifier

```
@Autowired  
MessageService smsService;
```

```
@Autowired  
@Qualifier("main")  
private MovieCatalog movieCatalog;
```

# Read more about Spring configuration and DI

<https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html>