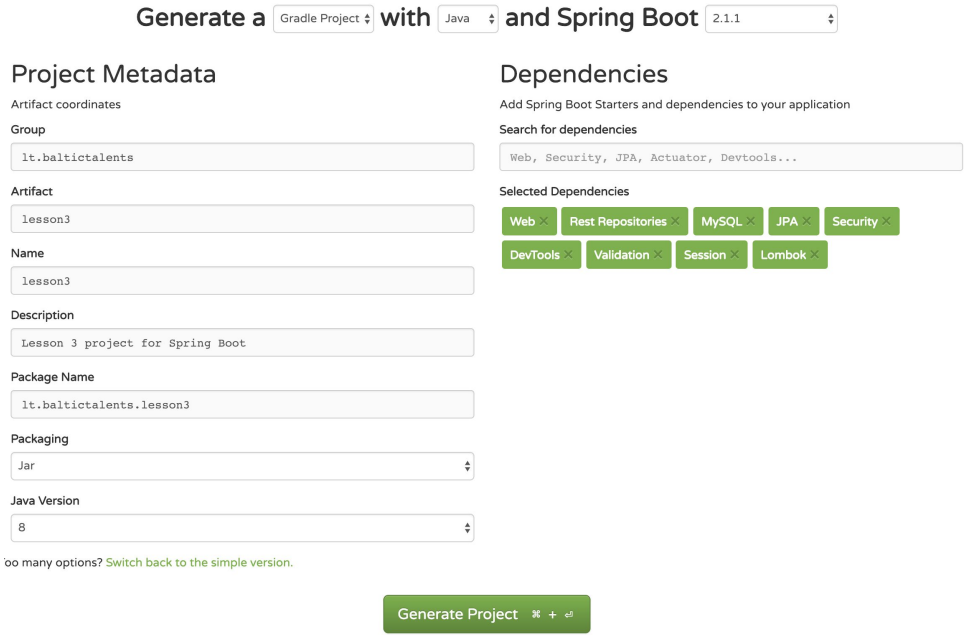
JAVA II

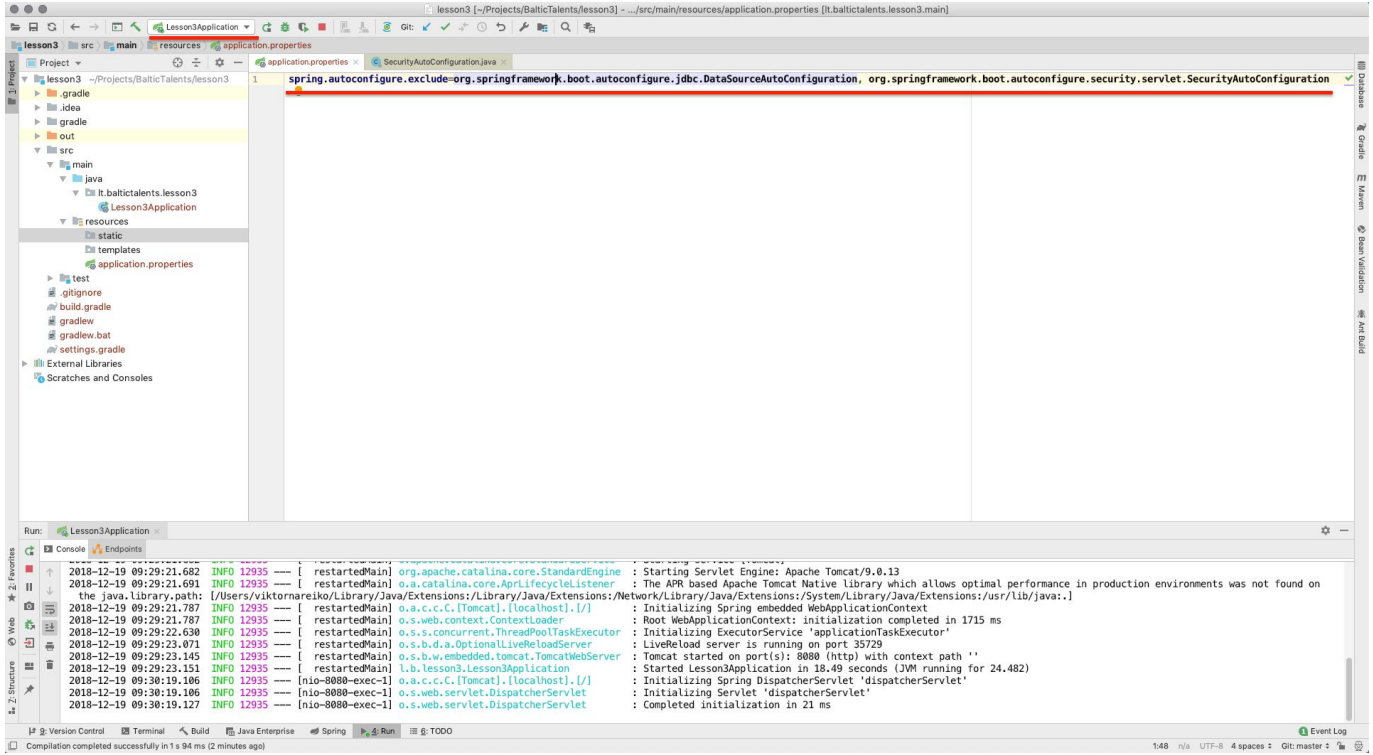
Spring web framework configuration

Initial project set up

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

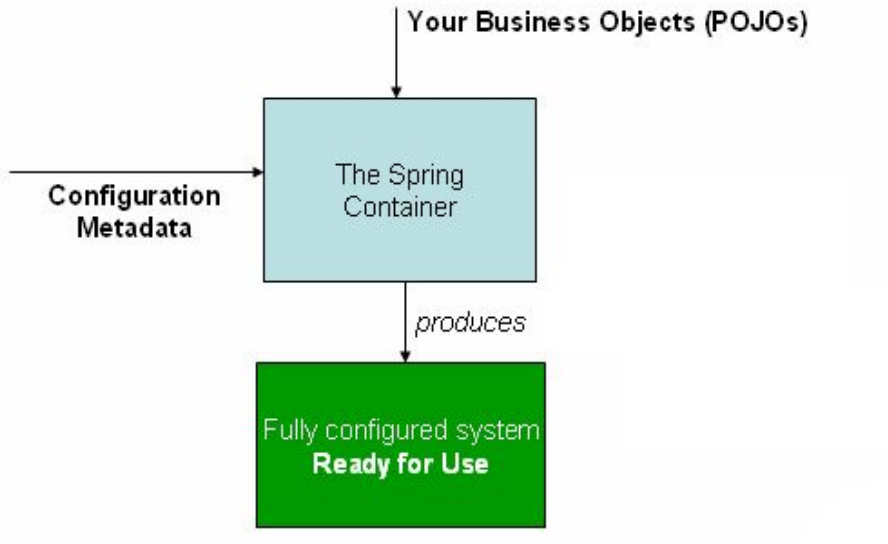


Initial project set up



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).



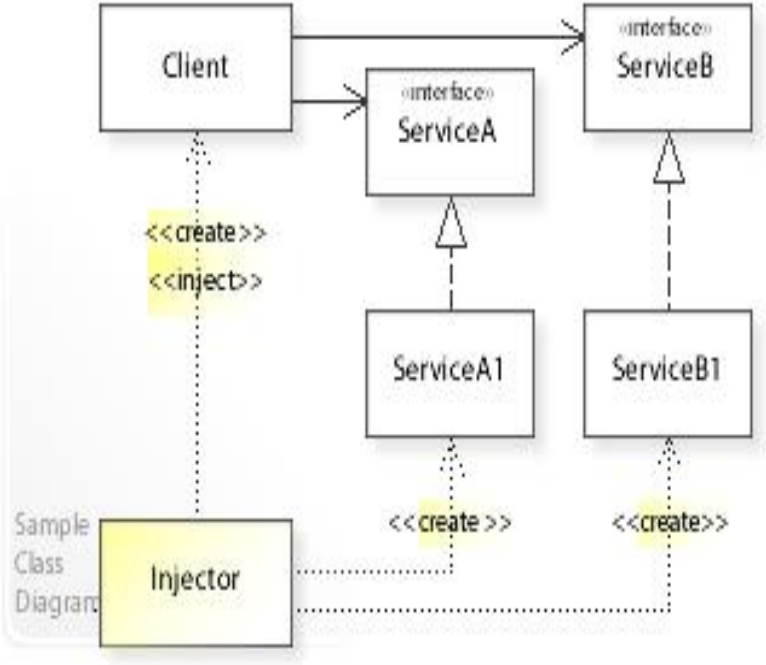
Dependency Injection pattern

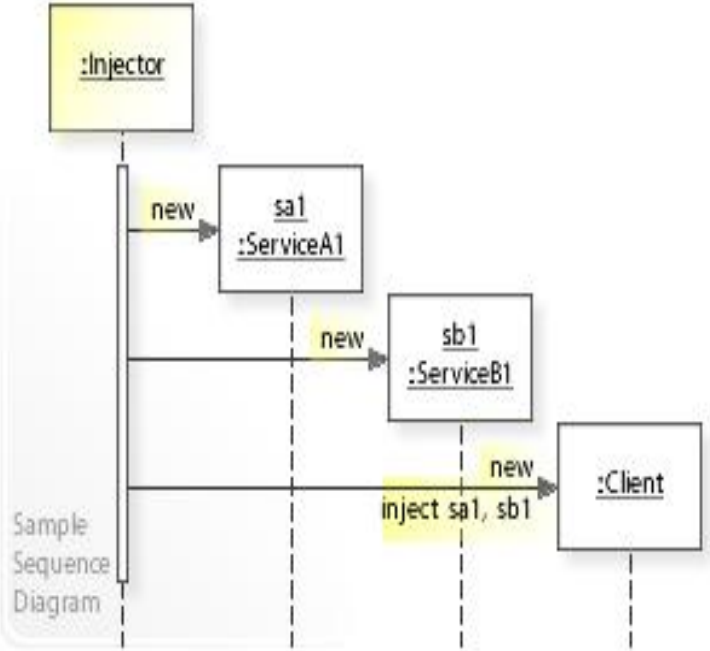
According to JSR330 the injection is done in the following order:

- 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

```xml
<?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="..."> (1) (2)
        <!-- 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>
```

(1) The `id` attribute is a string that identifies the individual bean definition.

(2) 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

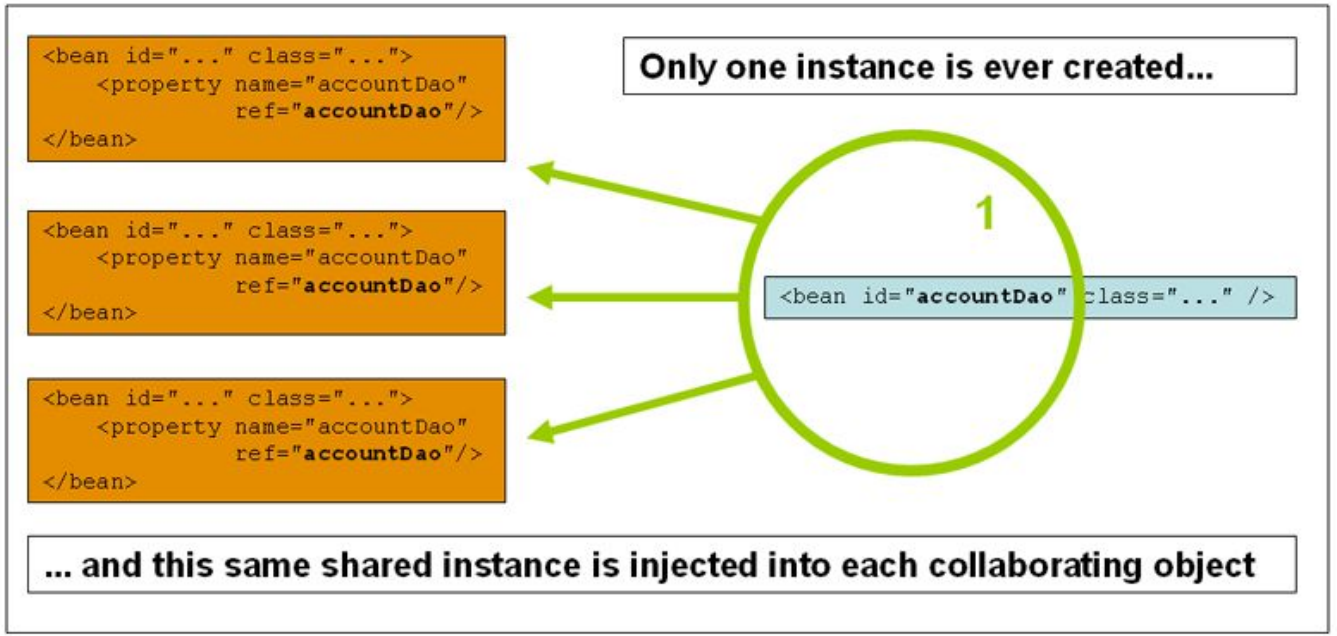
https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#spring-core

Beans scopes

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	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`.
session	Scopes a single bean definition to the lifecycle of an HTTP `Session`. Only valid in the context of a web-aware Spring `ApplicationContext`.
application	Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`.
websocket	Scopes a single bean definition to the lifecycle of a `WebSocket`. Only valid in the context of a web-aware Spring `ApplicationContext`.

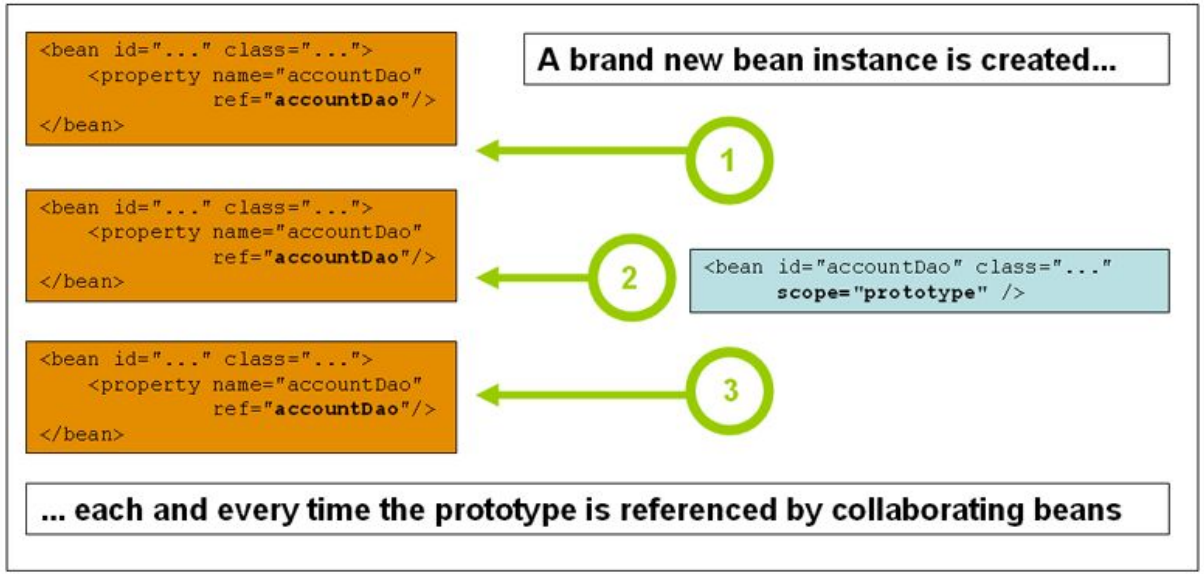
The Singleton scope

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") (1)
public class AppConfig  {
    ...
}
```

(1) This annotation enables component scanning.

@Bean @Configuration @ComponentScan annotation

```java
@Configuration
public class AppConfig {

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

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

```java
@Configuration
public class MyConfiguration {

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

Java annotation based configuration

@Autowired

@Required

@Primary

@Qualifier

```
@Autowired
BeansGetter beansGetter;
```

```
@Autowired
AccountDao accountDaoAnnotated;
```

```
@Autowired
MessageService emailService;
```

```
@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

Homework :)

Implement two services:

- MobilePaymentServiceImpl
- OnlineBankPaymentServiceImpl

They must implement PaymentService interface

By using method **String pay(String accontNumber, Integer amount)** you should get a String that will return something like that: 10 Eur were send to LT000000.

By using class **ApplicationContextController** modify getMobile and getOnlineBank methods so they could return relevant to payment strings by using /mobile-payment and /online-bank-payment endpoints.

You need to create two beans - **mobilePaymentService** and **onlineBankPaymentService**. Use them in relevant methods (**getMobile** and **getOnlineBank**)