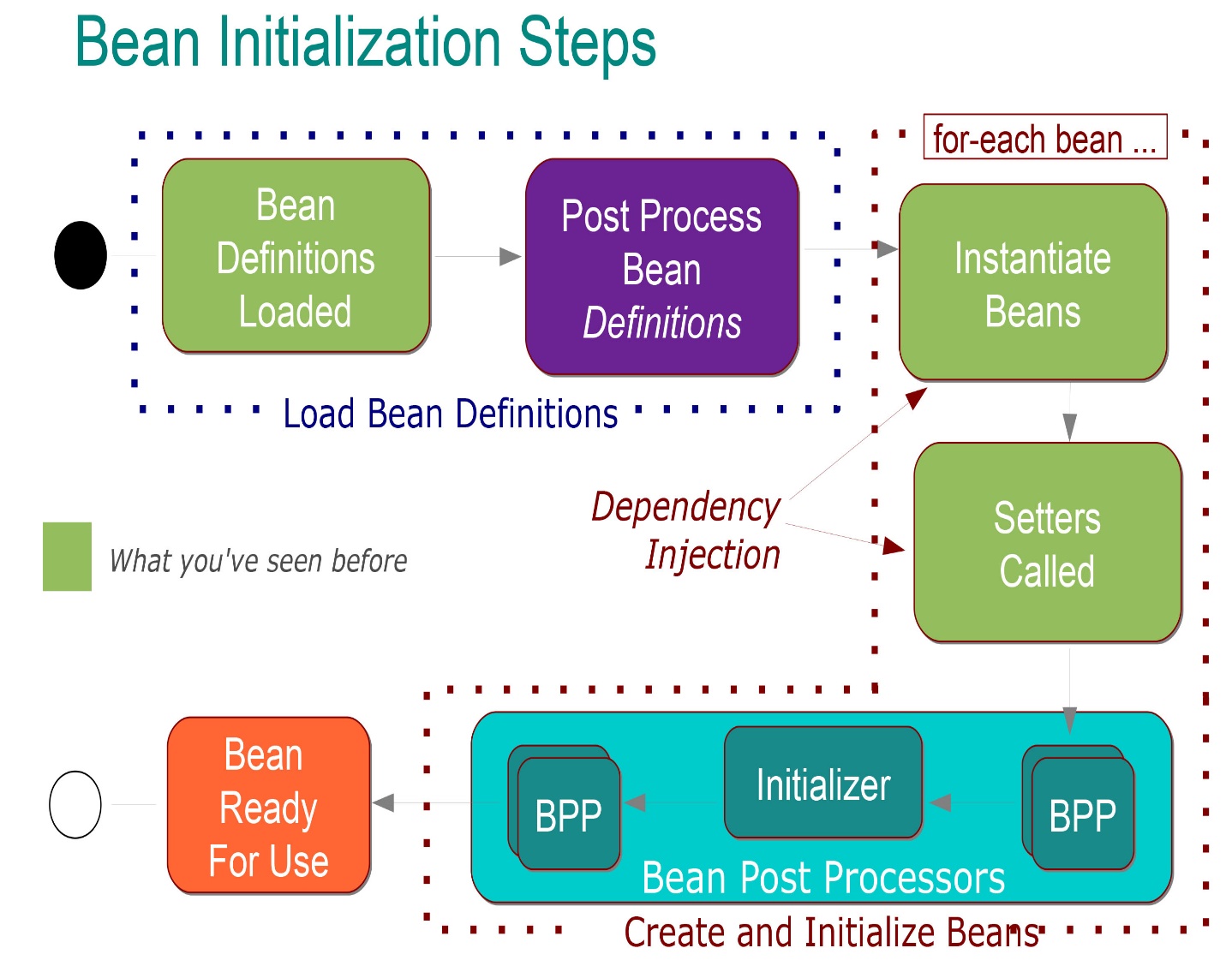
[*How exactly works the Spring bean post processor?*](http://stackoverflow.com/questions/29743320/how-exactly-works-the-spring-bean-post-processor)

I am studying for the Spring Core certification an I have some doubts about how Spring handle the **beans lifecycle** and in particular about the **bean post processor**.

So I have this schema:



It is pretty clear for me what it means:

Into the **Load Bean Definitions** phase happens that:

* The **@Configuration** classes are processed and/or **@Components** are scanned for and/or **XML files** are parsed.
* Bean definitions added to BeanFactory (each indexed under its id)
* Special **BeanFactoryPostProcessor** beans invoked, it can modify the definition of any bean (for example for the property-placeholder values replacements).

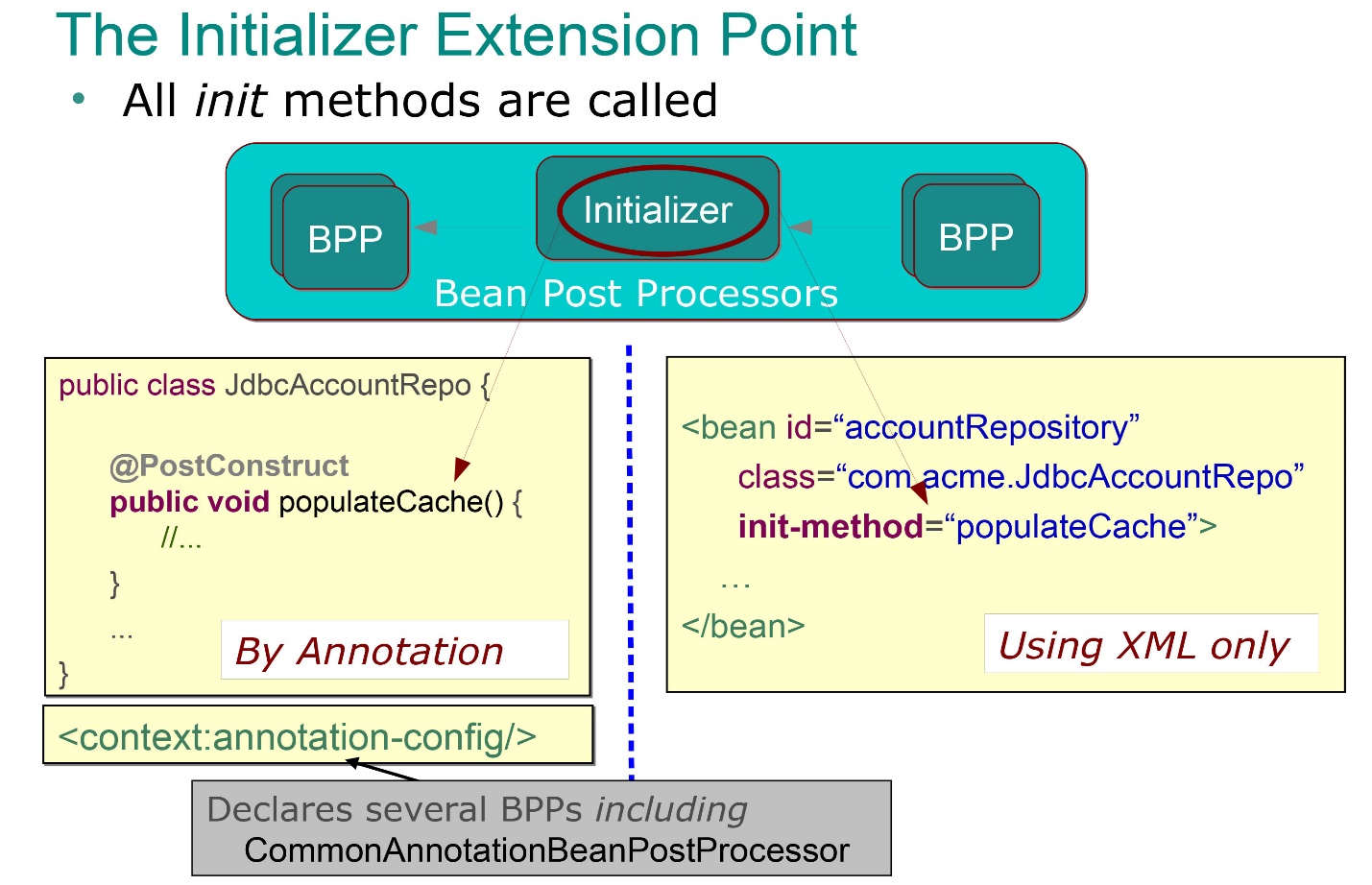
Then in the **beans creation phase** happens that:

* Each bean is eagerly instantiated by default (created in right order with its dependencies injected).
* After dependency injection each bean goes through a post-processing phase in which further configuration and initialization may occur.
* After post processing the bean is fully initialized and ready for use (tracked by its id until the context is destroyed)

Ok, this is pretty clear for me and I also know that **there are two types of bean post processors**that are:

* **Initializers:** Initialize the bean if instructed (i.e. @PostConstruct).
* **All the rest:** that allow for additional configuration and that **may run before or after the initialize step**

And I post this slide:



So it is very clear for me what does the **initializers** bean post processors (they are the methods annoted with **@PostContruct** annotation and that are automatically called immediately after the setter methods (so after the dependecy injection), and I know that I can use to perform some initialization batch (as populate a cache as in the previous example).

But what exactly represents the other bean post processor? What means that these are performed **before or after the initialization phase**?

So my bean are instantiated and it is performed the dependency injection so then the initialization phase is perfromed (by the execution of a **@PostContruct** annoted method). What means that an Bean Post Processor is performed before the initialization phase? It means that it happen before the **@PostContruct** annoted method execution? So int means that it could happen before the dependency injection (before that the setter methods are called)?

And what exactly means that it is performed **after the initialization step**. It means that it happens after that the exectuon of a **@PostContruct** annoted method, or what?

I can easily figure into my head why I need a **@PostContruct** annoted method but I can't figure some typical example of the other kind of bean post processor, can you show me some typical example of when are used?

[***Customizing beans using BeanPostProcessor***](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-factory-extension-bpp)***?***

<http://www.dataart.ru/blog/2014/04/malen-kie-sekrety-spring/>

Spring doc explains the BPPs under [Customizing beans using BeanPostProcessor](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-factory-extension-bpp). BPP beans are a special kind of beans that get created before any other beans and interact with newly created beans. With this construct, Spring gives you means to hook-up to and customize the lifecycle behavior simply by implementing a BeanPostProcessor yourself.

Having a custom BPP like

public class CustomBeanPostProcessor implements BeanPostProcessor {

public CustomBeanPostProcessor() {

System.out.println("0. Spring calls constructor");

}

@Override

public Object postProcessBeforeInitialization(Object bean, String beanName)

throws BeansException {

System.out.println(bean.getClass() + " " + beanName);

return bean;

}

@Override

public Object postProcessAfterInitialization(Object bean, String beanName)

throws BeansException {

System.out.println(bean.getClass() + " " + beanName);

return bean;

}

}

would be called and print out the class and bean name for every created bean.

To undersand how the method fit the bean's lifecycle, and when exactly the method's get called check the [docs](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/beans/factory/config/BeanPostProcessor.html#postProcessAfterInitialization-java.lang.Object-java.lang.String-)

**postProcessBeforeInitialization(Object bean, String beanName)** Apply this BeanPostProcessor to the given new bean instance before any bean initialization callbacks (like InitializingBean's afterPropertiesSet or a custom init-method).

**postProcessAfterInitialization(Object bean, String beanName)** Apply this BeanPostProcessor to the given new bean instance after any bean initialization callbacks (like InitializingBean's afterPropertiesSet or a custom init-method).

The important bit is also that

The bean will already be populated with property values.

For what concerns the relation with the @PostConstruct note that this annotation is a convenient way of declaring a postProcessAfterInitialization method, and Spring becomes aware of it when you either by registerCommonAnnotationBeanPostProcessor or specify the <context:annotation-config /> in bean configuration file. Whether the @PostConstruct method will execute before or after any other postProcessAfterInitialization depends on the order property

You can configure multiple BeanPostProcessor instances, and you can control the order in which these BeanPostProcessors execute by setting the order property.

# *Spring bean custom scope?*

<https://habrahabr.ru/post/225397/>