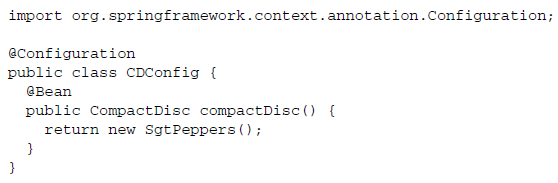
***Importing and mixing configurations***

* You’re free to mix component scanning and autowiring with JavaConfig and /or XML configuration.
* The first thing to know about mixing configuration styles is that when it comes to autowiring, it doesn’t matter where the bean to be wired comes from. Autowiring considers all beans in the Spring container, regardless of whether they were declared in JavaConfig or XML or picked up by component scanning.
* That leaves you with how to reference beans when doing explicit configuration, either with XML configuration or with Java configuration. Let’s start by seeing how to reference XML-configured beans from JavaConfig.

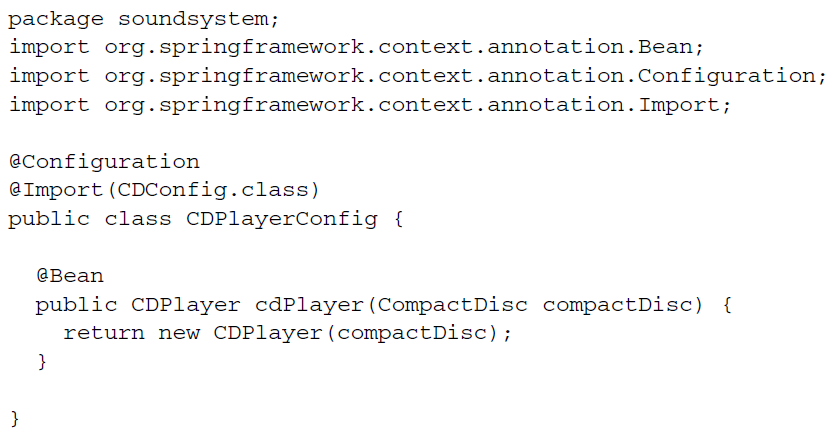
***Referencing XML configuration in JavaConfig***

* Let’s break out the *BlankDisc* bean from *CDPlayerConfig* into its own *CDConfig* Class, like this:

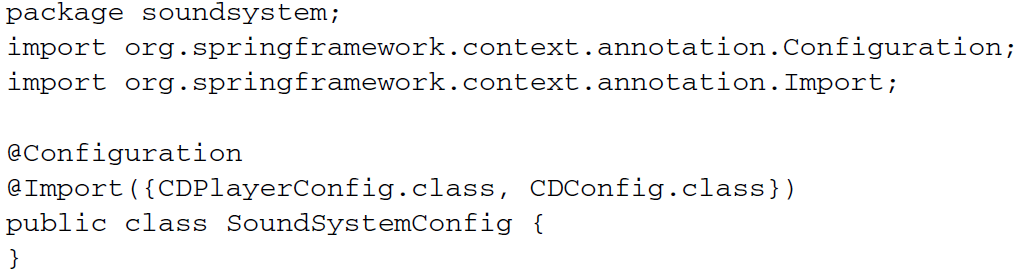
**eclipse_2017-06-30_11-01-03.png**

****

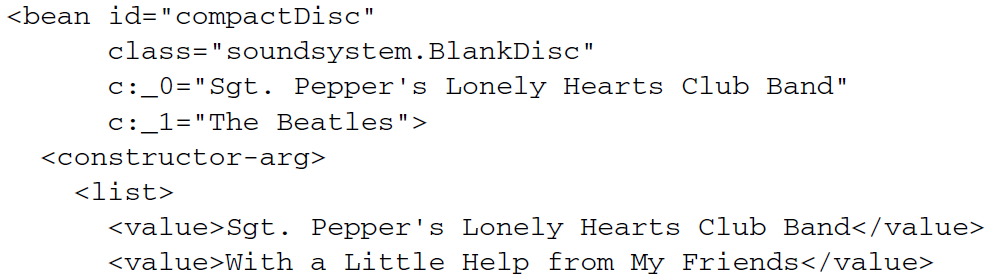
* Now that the *compactDisc()* method is gone from *CDPlayerConfig*, you need a way to bring the two configuration classes together. One way is to import *CDConfig* from *CDPlayerConfig* using the *@Import* annotation:

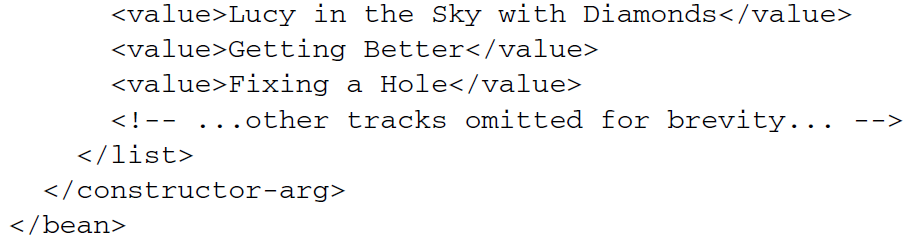
****

* Or, better yet, you can leave @*Import* out of *CDPlayerConfig* and instead create a higher-level *SoundSystemConfig* that uses @*Import* to bring configurations together:

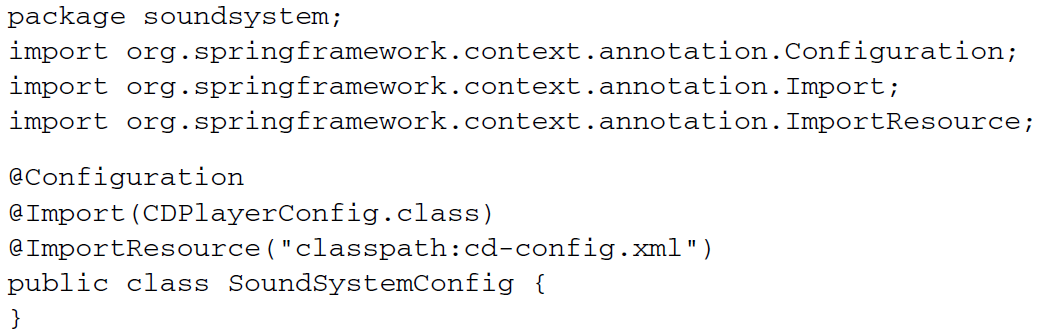
****

* Either way, you’ve separated the configuration of *CDPlayer* from the configuration of *BlankDisc.* Now let’s suppose that you want to configure the *BlankDisc* bean in XML like this:

****

****

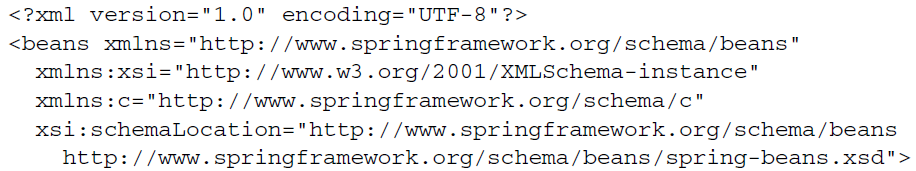
* With  *BlankDisc* being declared in XML, how can you have Spring load it in along with the rest of your Java-based configuration?
* The answer lies with the @*ImportResource* annotation. Assuming that the *BlankDisc* bean is declared in a file named cd-config.xml that can be found at the root of the classpath, you can change *SoundSystemConfig* to use @*ImportResource* like this:

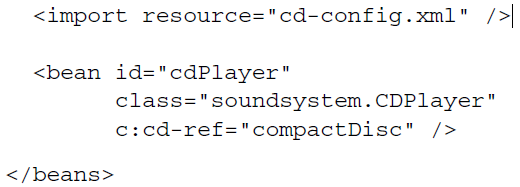
****

* Both beans – *CDPlayer* configured in *JavaConfig* *and BlankDisc* configured in XML – will be loaded into the Spring container. And because *CDPlayer’s* @*Bean* method accepts a *CompactDisc* as a parameter, the *BlankDisc* bean will be wired into it, even though it’s configured in XML.

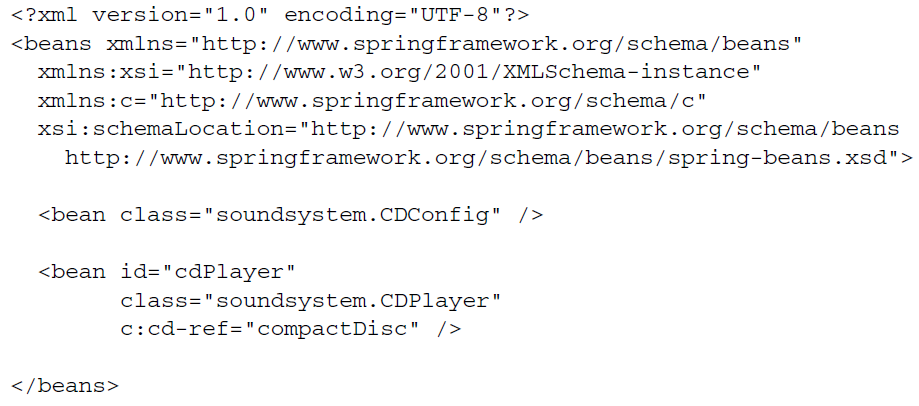
***Referencing JavaConfig in XML configuration***

* In XML, you can use the *<import>* element to split up the XML configuration.
* For example, suppose you were to split out the *BlankDisc* bean into its own configuration file called cd-config.xml, as you did when working with @*ImportResource.* You can reference that file from the XML configuration file using *<imort>:*

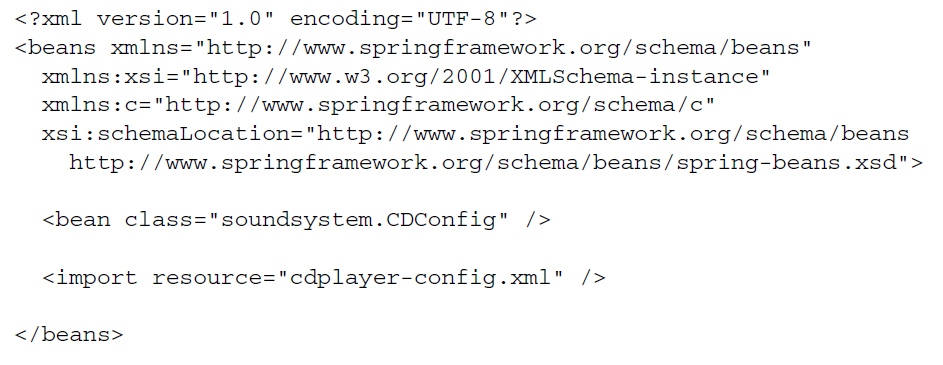
****

****

* Now, suppose that instead of configuring *BlankDisc* in XML, you want to configure it in XML while leaving the *CDPlayer* configuration in *JavaConfig.* How can your XML-based configuration reference a *JavaConfig* class?
* The *<import>* element only work to import other XML configuration files, and there isn’t an XML element whose job it is to import JavaConfig class into an XML configuration, you declare it as bean like this:

****

* And just like that, the two configurations – one expressed in XML and one expressed in Java – have been brought together. Similarly, you might consider a higher level configuration file that doesn’t declare any beans but that brings two or more configuration together. For example, you could leave the *CDConfig* bean out of the previous XML configuration and instead have a third configuration file that joins them:

****