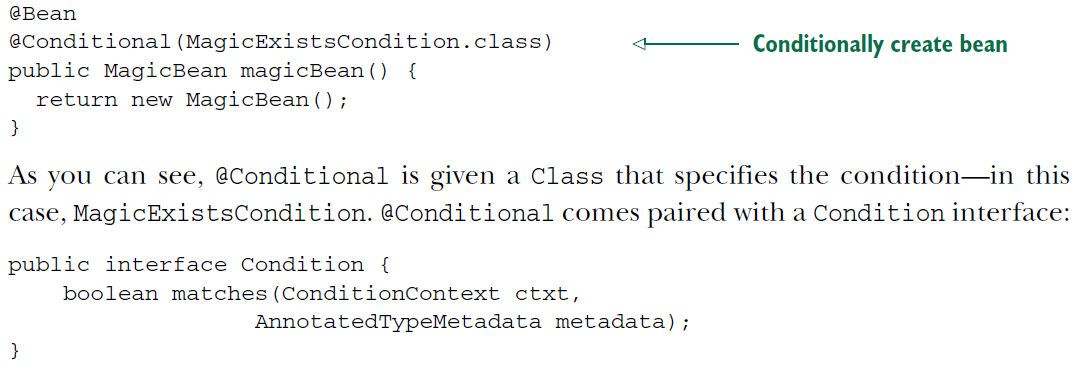
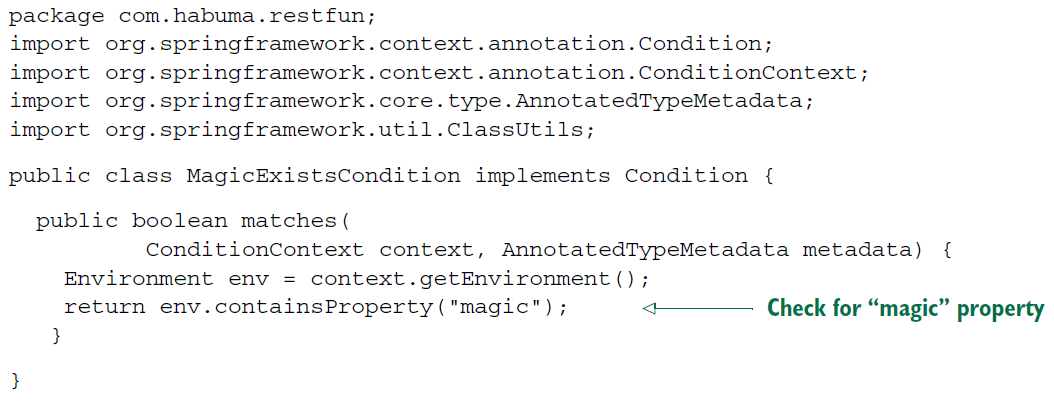
***Conditional beans***

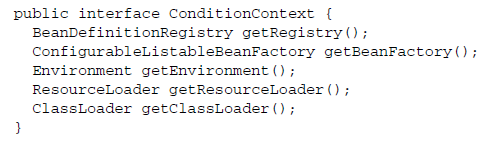
* Suppose you want one or more beans to be configured if and only if some library is available in the application’s classpath. Or let’s say you want a bean to be created only if a certain other bean is also declared. Maybe you want a bean to be created if and only if a specific environment variable is set.
* Until Spring 4, it was difficult to achieve this level of conditional configuration, but Spring 4 introduced a new *@Conditional* annotation that can be applied to *@Bean* methods. If the prescribed condition evaluates to true, then the bean is created. Otherwise the bean is ignored.
* For example, suppose you have a class named *MagicBean* that you only want Spring to instantiate if a *magic* environment property has been set. If the environment has no such property, then the *MagicBean* should be ignored.
* The following listing shows a configuration that conditionally configures the *MagicBean* Using @*Conditional* :

****

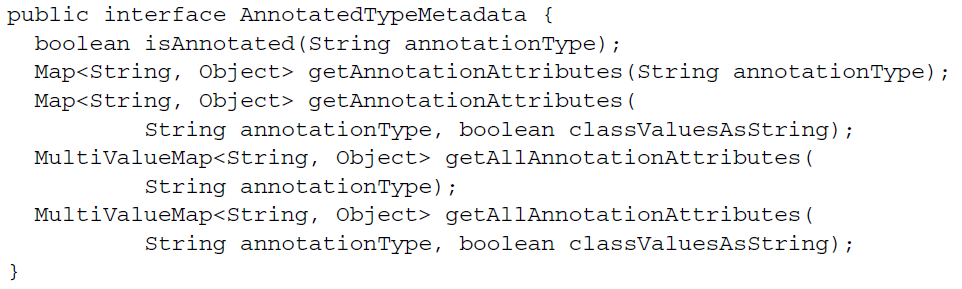
* The class given to *@Conditinal* can be any type that implements the *Condition* interface. As you can see, it’s a straightforward interface to implement, requiring only that you provide an implementation for the *matches()* method.
* If the *matches()*  method returns *true,* then the @*Conditional-*annotated beans are created.
* For this example, you need to create an implementation of *Condition* that hinges its decision on the presence of a magic property in the environment.
* The next listing shows *MagicExistisCondition,* an implementation of *Condition* that does the trick.

****

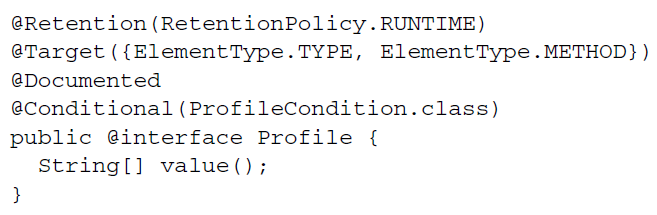
* The *matches()* method in this listing is simple but powerful. It must uses thee *Environment* obtained from the given *ConditionContext* object to check for the presence of an environment property named *magic.*
* For this example, the value of the property is irrelevant; it only needs to exist. This results in *true* being returned from *matches().*
* Consequently, the condition is met, and any beans whose @*Conditional* annotation refers to *MagicExistCondition* will be created.
* On the other hand, if the property doesn’t exist, the condition will fail, *false* will returned from *matches(),* and none of those beans will be created.
* *MagicExistCondition* only used the *Environment* from the *ConditionContext,* but ther’s much more that a *Condition* implementation can consider.
* The *matches()* method is given a *CondtionContext* and an *AnnotatedTypeMetadata* to use in making its decision.
* *ConditionContext* is an interface that looks something like this:

****

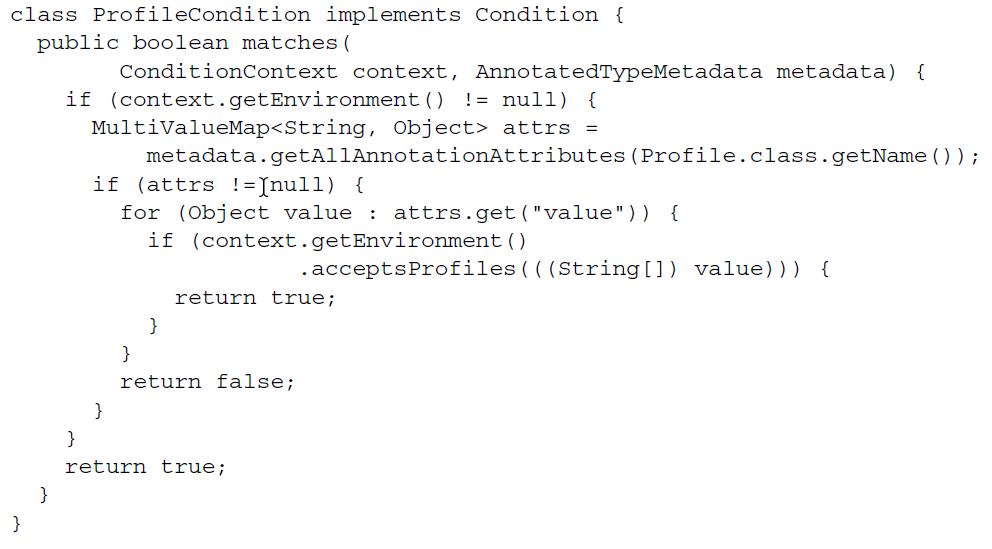
* From the *ConditionalContext,* you can do the following:
* Check for bean definitions via the *BeanDefinitionRegistry* returned from *getRegistry().*
* Check for the presence of beans, and even dig into bean properties via the *ConfigurableListableBeanFactory* returned from *getBeanFactory().*
* Check for the presence and values of environment variables via the *Environment* retrieved from *getEnvironment().*
* Read and inspect the contents of resources loaded via the *ResourceLoader* returned from *getResourceLoader().*
* Load and check for the presence of classes via the *ClassLoader* returned from *getClassLoader().*
* As for the AnnoyatedTypeMetadata, it offers you a chance to inspect annotations that may also be placed on the *@Bean* method. Like *ConditionContext, AnnotatedTypeMetadata* is an interface. It looks like this:

****

* Using the *isAnnotated()* method, you can check to see if the *@Bean* method is annotated with any particular annotation type. Using the other methods, you can check on the attributes of any annotation applied to the *@Bean* method.
* Interestingly, starting with Spring 4, the *@Profile* annotation has been refactored to be based on *@Conditional* and the *Condition* interface. As another example of how to work with *@Conditional* and *Condition,* let’s look at how *@Profile* is implemented in Spring 4.
* The *@Profile* annotation looks like this:

****

* Notice that *@Profile* is itself annotated with *@Conditional* and refers to *ProfileCondition* as the *Condition* implementation.
* As shown next, *ProfileCondition* implements *Condition* and considers several factors both *ConditionContext* and *AnnotatedTypeMetadata* in making its decision.

****

* As you can see, *ProfileConditon* fetches all the annotation attributes for the *@Profile* annotation from *AnnotatedTypeMetadata*. With that, it checks explicitly for the *value* attribute, which contains the name of the bean’s profile. It then consults with the *Environment* retrieved from the *ConditionContext* to see whether the profile is active (by calling the *acceptProfiles() method).*