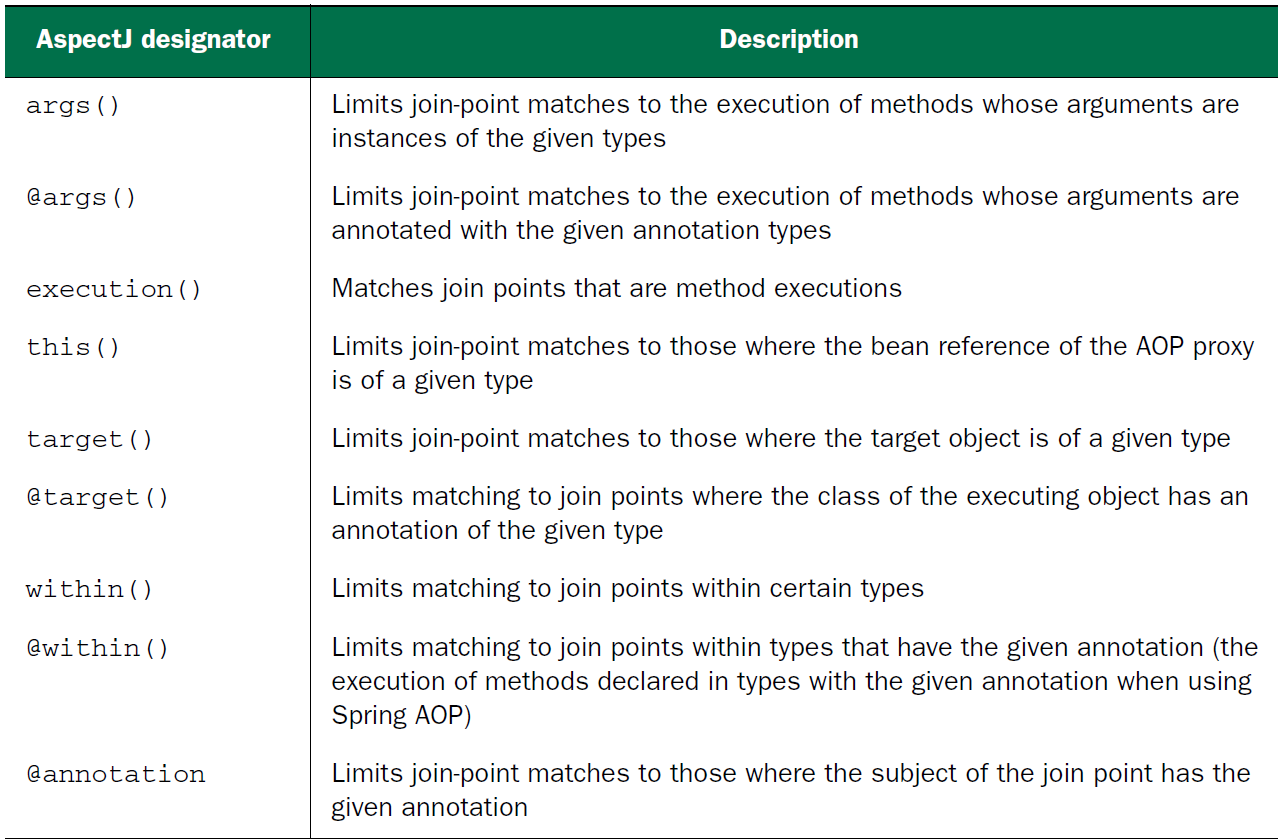
***Selecting join points with pointcuts***

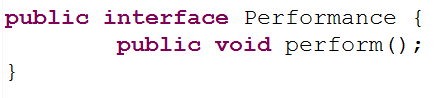
* As mentioned before, pointcuts are used to pinpoint where an aspect’s advice should be applied.
* Along with an aspect’s advice, pointcuts are among the most fundamental elements of an aspect. Therefore, it’s important to know how to write pointcuts.
* In Spring AOP, pointcuts are defined using AspectJ’s pointcut expression language.
* The most important thing to know about AspectJ pointcuts as they pertain to Spring AOP is that Spring only supports a subset of the pointcut designators available in AspectJ.
* Lists of AspectJ pointcut designators that are supported in Spring AOP:



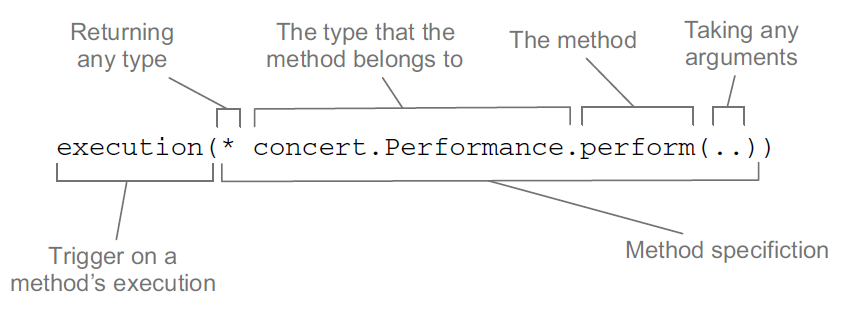
* Attempting to use any of AspectJ’s other designators will result in an IllegalArgumentException being thrown.
* As you browse through the supported designators, note that the execution designator is the only one that actually performs matches.
* This means execution is the primary designator you’ll use in every pointcut definition you write. You’ll use the other designators to constrain the pointcut’s reach.

***Writing pointcuts***

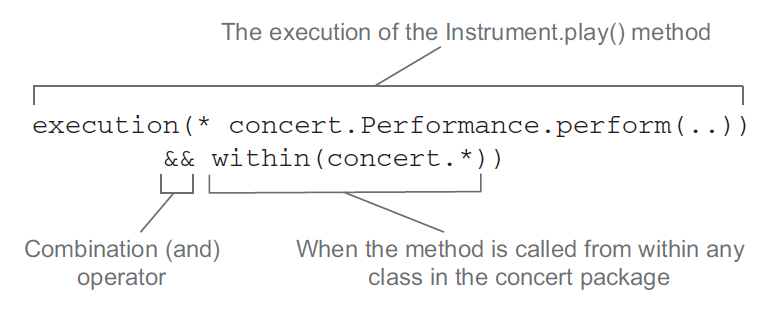
To demonstrate aspects in Spring, you need something to be the subject of the aspect’s pointcuts. For that purpose, let’s define a Performance interface:



* Performance represents any kind of live performance, such as a stage play, a movie, or a concert.
* Let’s say that you want to write an aspect that triggers off Performance’s perform() method.
* Below Code shows a pointcut expression that can be used to apply advice whenever the perform() method is executed



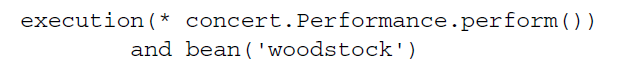
* You use the execution() designator to select Performance’s perform() method. The method specification starts with an asterisk, which indicates that you don’t care what type the method returns. Then you specify the fully qualified class name and the name of the method you want to select. For the method’s parameter list, you use the double dot (..), indicating that the pointcut should select any perform() method, no matter what the argument list is.
* Now let’s suppose that you want to confine the reach of that pointcut to only the concert package. In that case, you can limit the match by tacking on a within() designator, as shown:



* Note that you use the && operator to combine the execution() and within()designators in an “and” relationship (where both designators must match for the pointcut to match).
* Similarly, you could use the || operator to indicate an “or” relationship. And the ! operator can be used to negate the effect of a designator.
* Because ampersands have special meaning in XML, you’re free to use and in place of && when specifying pointcuts in a Spring XML-based configuration. Likewise, or and not can be used in place of || and !, respectively.

***Selecting beans in pointcuts***

* In addition to the designators listed in table, Spring adds a bean()designator that lets you identify beans by their ID in a pointcut expression.
* bean() takes a bean ID or name as an argument and limits the pointcut’s effect to that specific bean.
* For example, consider the following pointcut:



* Here you’re saying that you want to apply aspect advice to the execution of Performance’s perform() method, but limited to the bean whose ID is woodstock.
* Narrowing a pointcut to a specific bean may be valuable in some cases, but you can also use negation to apply an aspect to all beans that don’t have a specific ID:

aop.png

* In this case, the aspect’s advice will be woven into all beans whose ID isn’t woodstock.