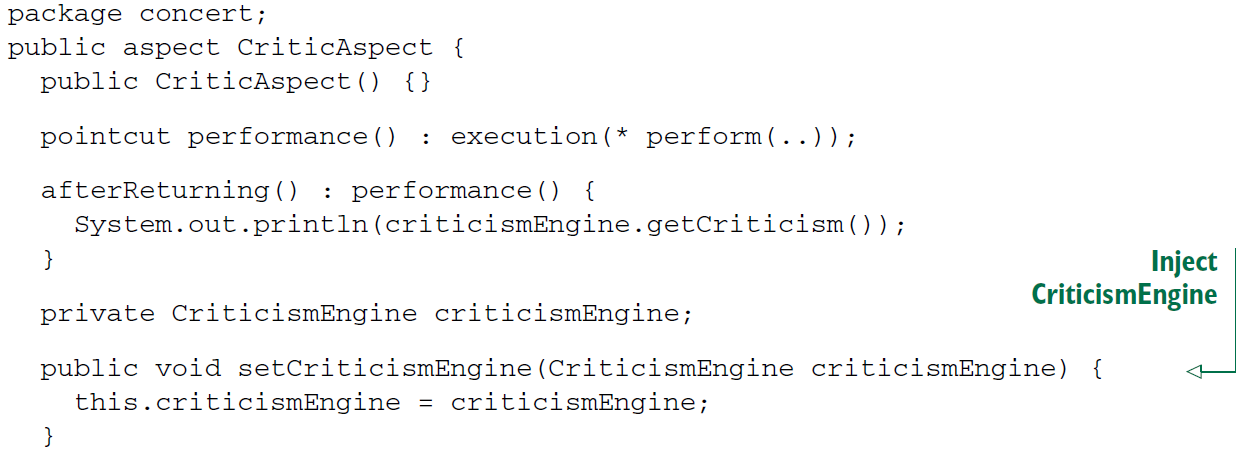
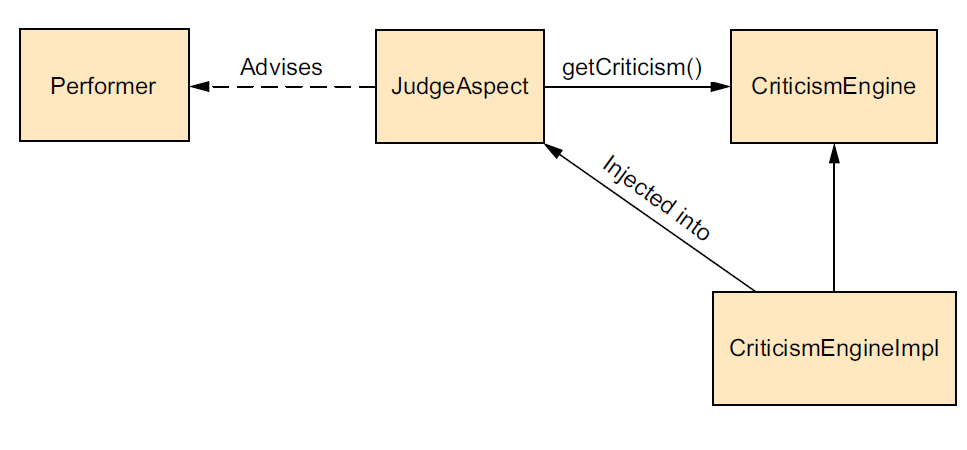
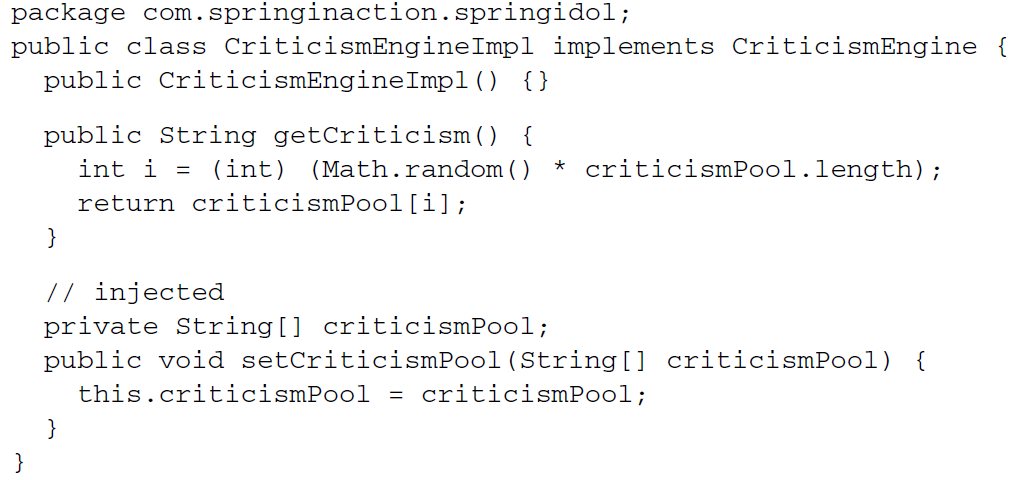
***Injecting AspectJ aspects***

* Although Spring AOP is sufficient for many applications of aspects, it’s a weak AOP solution when contrasted with AspectJ. AspectJ offers many types of pointcuts that aren’t possible with Spring AOP.
* Constructor pointcuts, for example, are convenient when you need to apply advice on the creation of an object. Unlike constructors in some other object-oriented languages, Java constructors are different from normal methods. This makes Spring’s proxy-based AOP woefully inadequate for advising the creation of an object.
* For the most part, AspectJ aspects are independent of Spring. Although they can be woven into any Java-based application, including Spring applications, there’s little involvement on Spring’s part in applying AspectJ aspects.
* But any well-designed and meaningful aspect will likely depend on other classes to assist in its work. If an aspect depends on one or more classes when executing its advice, you can instantiate those collaborating objects with the aspect itself. Or, better yet, you can use Spring’s dependency injection to inject beans into AspectJ aspects.
* To illustrate, let’s create a new aspect for performances. Specifically, let’s create an aspect that plays the role of a critic who watches a performance and provides a critical review afterward. CriticAspect is such an aspect:

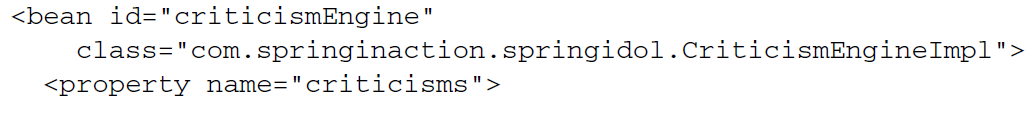


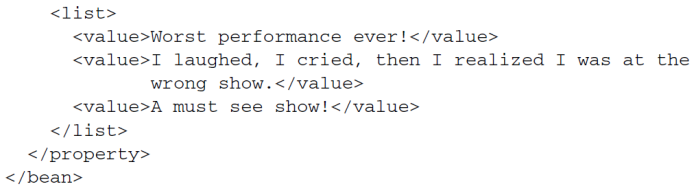


* The chief responsibility for CriticAspect is to comment on a performance after the performance has completed. The performance() pointcut matches the perform() method. When it’s married with the afterReturning() advice, you get an aspect that reacts to the completion of a performance.
* What makes this interesting is that the critic doesn’t make commentary on its own. Instead, CriticAspect collaborates with a CriticismEngine object, calling its getCriticism() method, to produce critical commentary after a performance. To avoid unnecessary coupling between CriticAspect and CriticismEngine, Critic-Aspect is given a reference to CriticismEngine through setter injection.
* CriticismEngine is an interface that declares a simple getCriticism() method. The next listing shows the implementation of CriticismEngine.

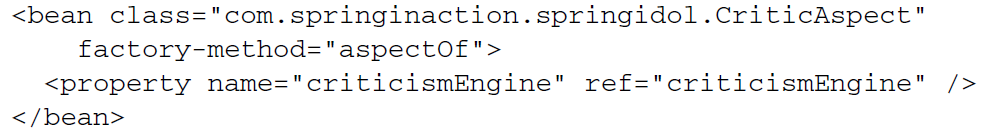


* CriticismEngineImpl implements the CriticismEngine interface by randomly choosing a critical comment from a pool of injected criticisms. This class can be declared as a Spring <bean> using the following XML:





* Now we have a CriticismEngine implementation to give to Critic-Aspect. All that’s left is to wire CriticismEngineImpl into CriticAspect.
* You should know that AspectJ aspects can be woven into your application without involving Spring at all. But if you want to use Spring’s dependency injection to inject collaborators into an AspectJ aspect, you’ll need to declare the aspect as a <bean> in Spring’s configuration. The following <bean> declaration injects the criticismEngine bean into CriticAspect:



* For the most part, this <bean> declaration isn’t much different from any other <bean> you may find in Spring. The big difference is the use of the factory-method attribute. Normally, Spring beans are instantiated by the Spring container, but AspectJ aspects are created by the AspectJ runtime. By the time Spring gets a chance to inject CriticismEngine into CriticAspect, CriticAspect has already been instantiated.
* Because Spring isn’t responsible for the creation of CriticAspect, it isn’t possible to declare CriticAspect as a bean in Spring. Instead, you need a way for Spring to get a handle to the CriticAspect instance that has already been created by AspectJ so that you can inject it with a CriticismEngine. Conveniently, all AspectJ aspects provide a static aspectOf() method that returns the singleton instance of the aspect.
* So to get an instance of the aspect, you must use factory-method to invoke the aspectOf() method instead of trying to call CriticAspect’s constructor.
* In short, Spring doesn’t use the <bean> declaration from earlier to create an instance of the CriticAspect—it has already been created by the AspectJ runtime. Instead, Spring retrieves a reference to the aspect through the aspectOf() factory method and then performs dependency injection on it as prescribed by the <bean> element.