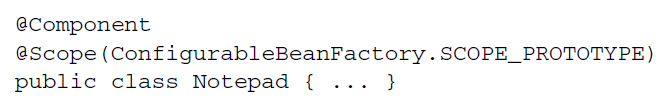
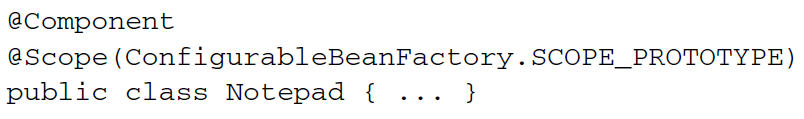
***Scoping beans***

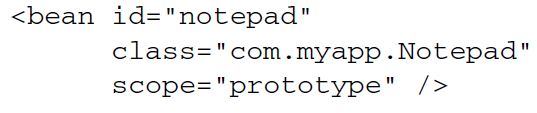
* By default, all beans created in the Spring Container are created as **singletons**. That is to say, no matter how many times a given bean is injected into other beans, it’s always the same instance that is injected each time.
* Most of the time, singleton beans are ideal. The cost of instantiating and garbage-collecting instances of objects that are only used for small tasks can’t be justified when an object is stateless and can be reused over and over again in an application.
* But sometimes you may find yourself working with a mutable class that does maintain some state and therefore isn’t safe for reuse. In that case, declaring the class as a singleton bean probably isn’t a good idea because that object can be tainted and create unexpected problems reused later.
* Spring defines several scopes under which a bean can be created, including the following:
* **Singleton**–One instance of the bean is created for the entire application.
* ***Prototype***–One instance of the bean is created every time the bean is injected into or retrieved from the Spring application context.
* ***Session***– In a web application, one instance of the bean is created for each session.
* ***Request***– In a Web application, one instance of the bean is created for each request.
* **Singleton scope is the default scope**, but as we’ve discussed, it isn’t ideal for mutable types. To select an alternative type, you can use the *@Scope* annotation, either in conjunction with the *@Component* annotation or with the *@Bean* annotation.
* For example, if you’re relying on component-scanning to discover and declare a bean, then you can annotate the bean class with *@Scope* to make it a prototype bean:



* Here, you specify prototype scope by using the *SCOPE\_PROTOTYPE* constant from the *ConfigurableBeanFactory* class. You could also use *@Scope(“prototype”),* but using the *SCOPE\_PROTOTYPE* is safer and less prone to mistakes.
* Alternatively, if you’re configuring the *Notepad* bean as a prototypein Java configuration, you can use *@Scope* along with *@Bean* to specify the desired scoping:



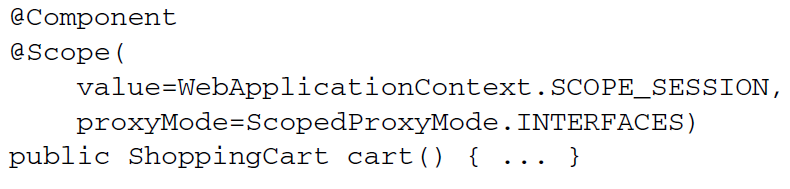
* And, in the event that you’re configuring the bean in XML, you can set the scope using the *scope* attribute of the *<bean>* element:



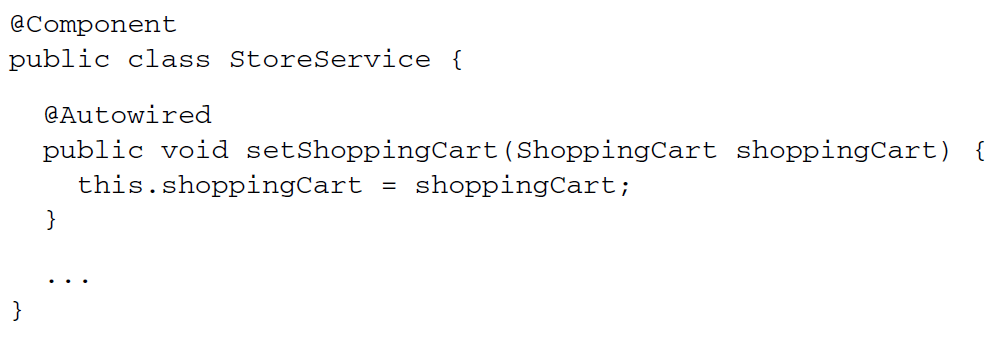
* Regardless of how you specify prototype scope, an instance of the bean will be created each and every time it’s injected into or retrieved from the Spring application context. Consequently, everyone gets their own instance of *Notepad.*

***Working with request and session scope***

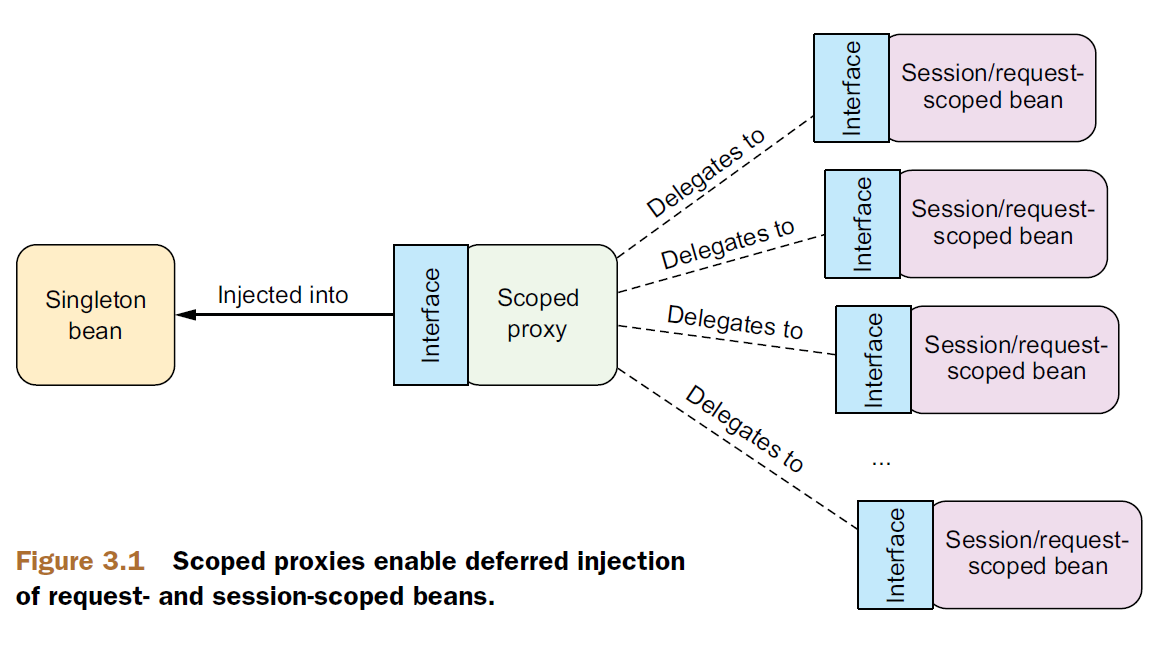
* In a web application, it may be useful to insatiate a bean that’s shared within the scope of a given request or session. For instance, in a typical e-commerce application, you may have a bean that represents the user’s shopping cart. If the shopping cart bean is a singleton, then all users will be adding products to the same cart. On the other hand, if the shopping cart is prototype-scoped, then products added to the cart in one area of the application may not be available in another part of the application where a different prototype-scoped shopping cart was injected.
* In the case of a shopping cart bean, session makes the most sense, because it’s most directly attached to a given user. To apply session scope, you can use the *@Scope* annotation in a way similar to how you specified prototype scope:



* Here you’re setting the *value* attribute to the *SCOPE\_SESSION* constant from *WebApplicationContext* (which has a value of session). This tells Spring to create an instance of the *ShoppingCart* bean, but only one will be created for a given session and it will essentially be a singleton as far as that session is concerned.
* Notice that *@Scope* also has a *proxyMode* attribute set to *ScopedProxyMode.INTERFACES.*
* This attribute addresses a problem encountered when injecting a session- or request-scoped bean into a singleton-scoped bean. But before explaining *proxyMode,* let’s look at a scenario that presents the problem that proxyMode addresses.
* Suppose you want to inject the *ShoppingCart* bean into the following setter method on a singleton *StoreService* bean:

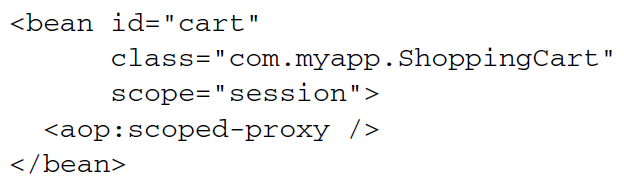


* Because *StoreService* is a singleton bean, it will be created as the Spring application context is loaded. As it’s created, Spring will attempt to inject *ShoppingCart* into the *setShopingCart()* method. But the *ShoppingCart* bean, being session scoped, doesn’t exist yet. There won’t be an instance of *ShoppingCart* until a user comes along and a session is created.
* Moreover, there will be many instances of *ShoppingCart:*  one per user. You don’t want Spring to inject just any single instance of *ShoppingCart* into *StoreService.* You want *StoreService* to work with the *ShoppingCart* instance for whichever session happens to be in play when *StoreService* needs to work with the shopping cart.
* Instead of injecting the actual *ShoppingCart* bean into *StoreService,* Spring should inject a proxy to the *ShoppingCart* bean. This proxy will expose the same methods as *ShoppingCart* so that for all *StoreService* knows, it is the shopping cart. But when *StoreService* calls methods on *ShoppingCart, the proxy will lazily resolve it and* delegate the call to actual session-scoped *ShoppingCart* bean.
* Now let’s take this understanding of scoped proxies and discuss the *proxyMode* attribute. As Configured, *proxyMode* is set to ScopedProxyMode.INTERFACES, indicating that the proxy should implement the *ShoppingCart* interface and delegate to the implementation bean.
* This is fine (and the most ideal proxy mode) as long as *ShoppingCart* is an interface and not a class. But if *ShoppingCart* is a concrete class, there’s no way Spring can create an interface-based proxy. Instead, it must use *CGLib* to generate a class-based proxy. So, if the bean type is a Concrete class, you must set *proxyMode* to *ScopedProxyMode.*TARGET\_CLASSto indicate that the proxy should be generated as an extension of the target class.
* Although I’ve focuded on session scope, know that request-scoped beans pose the same wiring challenges as session-scoped beans. Therefore, request-scoped beans should also injected as scoped proxies.

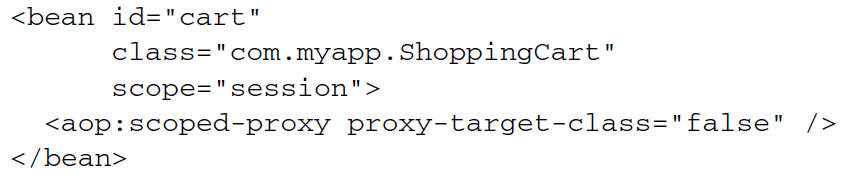


***Declaring scoped proxies in XML***

* If you are declaring your session-scoped or request-scoped beans in XML, can’t use the *@Scope* annotation or its *proxyMode* attribute. The scope attribute of the *<bean>* element lets you set the bean scope, but how can you specify the proxy mode?
* To set the proxy mode, you must use a new element from Spring’s *aop* namespace:



* *<aop:scoped-proxy>* is the Spring XML configuration’s counterpart to the *@Scoped* annotation’s *proxyMode* attribute. It tells Spring to create a scoped proxy for the bean.
* By default, it uses CGLib to create a target proxy. But you can ask it to generate an interface-based by setting the *proxy-target-class* attribute to false:



* In Order to use the *<aop:scoped-proxy>* element, you must declare Spring’s *aop* namespace in you XML configuration:

