***Summary***

* We’ve built on the foundational bean-wiring techniques explored in previous chapter with some powerful advanced wiring tricks.
* We started by using Spring profiles to address a common problem where Spring beans must vary across deployment environments. By resolving environment-specific beans at runtime by matching them against one or more active profiles, Spring makes it possible to deploy the same deployment unit across multiple environments without rebuilding.
* Profiles beans are one way to conditionally create beans at runtime, but Spring 4 offers a more generic way to declare beans that are created ( or not created ) depending on the outcome of a given condition. The *@Conditional* annotation, paired with an implementation of Spring’s *Condition* interface, offers developers a powerful and flexible mechanism for conditionally creating beans.
* We also looked at two techniques for resolving autowiring ambiguity: primary beans and qualifiers. Although designating a bean as a primary bean is simple, it’s also limited, so we discussed using qualifiers to narrow the list of autowire candidates to a single bean. In additions , you saw how to create custom qualifier annotations that describe a bean by its traits.
* Although most Spring beans are created as singleton, there are times when other creation strategies are more appropriate. Out of the box, Spring allows bean to be created as singletons, prototypes, request-scoped, or session-scoped. When declaring request or session-scoped beans, you also learned how to control the way scoped proxies are created, either as class-based proxies or interface-based proxies.
* Finally, we looked at the Spring Expression Language, which gives you a way to resolve values to be injected into bean properties at runtime.