Spring Beans and Dependency Injection

You are free to use any of the standard Spring Framework techniques to define your beans and their injected dependencies. We generally recommend using constructor injection to wire up dependencies and @ComponentScan to find beans.

If you structure your code as suggested above (locating your application class in a top package), you can add @ComponentScan without any arguments or use the @SpringBootApplication annotation which implicitly includes it. All of your application components (@Component, @Service, @Repository, @Controller, and others) are automatically registered as Spring Beans.

The following example shows a <code>@Service</code> Bean that uses constructor injection to obtain a required <code>RiskAssessor</code> bean:

```
Java Kotlin
@Service
public class MyAccountService implements AccountService {
```

Spring Boot / Reference / Developing with Spring Boot / Spring Beans and Dependency Injection

```
this.riskAssessor = riskAssessor;
```

Edit this Page

GitHub Project

Stack Overflow

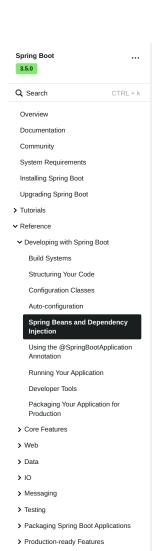
If a bean has more than one constructor, you will need to mark the one you want Spring to use with <code>@Autowired</code>:

```
Java Kotlin
public class MyAccountService implements AccountService {
    private final RiskAssessor riskAssessor;
    private final PrintStream out;
    public MyAccountService(RiskAssessor riskAssessor) {
        this.riskAssessor = riskAssessor:
        this.out = System.out;
    public MyAccountService(RiskAssessor riskAssessor, PrintStream out) {
        this.riskAssessor = riskAssessor;
        this.out = out;
    // ...
```

Notice how using constructor injection lets the riskAssessor field be marked as final, indicating that it cannot be subsequently changed.



Using the @SpringBootApplication Annotation >



> How-to Guides > Build Tool Plugins > Spring Boot CLI > Rest APIs > Java APIs > Kotlin APIs

spring