# Chapter B



# Errata

---

Hi guys, I am really thankful to all of you that have bought the book and have contributed to finding mistakes in it and in the code. Because of you, I have taken another careful look at the book questions and answers and decided this errata was needed.

## B.1 Book corrections

### Chapter 1: Introduction

The section **About the Spring Web Certification Exam** needs an update. Starting with January 2017 the exam is taken directly from home and the voucher is valid only three months. *(Reported by Vittorio Marino)*

### Chapter 2: Spring Bean LifeCycle and Configuration

In the section **Setter Injection section** in the examples a bean named `simpleBean0` is declared, but not used as a dependency in the `complexBean`. The intended name, was obviouslt `simpleBean`. *(Reported by Gautham Kurup)*

! The original (erroneous) code sample is this:

```
<beans ...>
    <bean id="simpleBean0" class="com.ps.beans.SimpleBeanImpl"/>

    <bean id="complexBean" class="com.ps.beans.set.ComplexBeanImpl">
        <property name="simpleBean" ref="simpleBean"/>
    </bean>
</beans>
```

! And in future editions of the book, will be replaced with this:

```
<beans ...>
    <bean id="simpleBean" class="com.ps.beans.SimpleBeanImpl"/>

    <bean id="complexBean" class="com.ps.beans.set.ComplexBeanImpl">
        <property name="simpleBean" ref="simpleBean"/>
    </bean>
</beans>
```

The same bean is later referred in the paragraph explaining the behaviour.

! The original (erroneous) paragraph is this:

The code and configuration snippets above provide all the necessary information so that the Spring container can create a bean of type `SimpleBeanImpl` and a bean named `simpleBean0` and then inject it into a bean of type `ComplexBeanImpl` named `complexBean`.

! And in future editions of the book, will be replaced with this:

The code and configuration snippets above provide all the necessary information so that the Spring container can create a bean of type `SimpleBeanImpl` and a bean named `simpleBean` and then inject it into a bean of type `ComplexBeanImpl` named `complexBean`.

In the section **Bean Naming section** there is a huge error in the samples for `@AliasFor`. It was implied there that the `value` attribute of the `@Repository` can be aliased. At the time when the book was written, Spring 4.2 was under development and more information about the limits of the `@AliasFor` were published later on. Because autowiring by type was used, the tests still passed thus hiding the erroneous implementation. So there is a code sample and some text there which are erroneous, and should be skipped altogether.

! The original (erroneous) section is this:

And even more can be done. Aliases for meta-annotation attributes can be declared. Let's say that we do not like the `value` attribute name of the `@Repository` annotation and we want in our application to make it more obvious that the `@Repository` annotation can be used to set the id of a bean, by adding an `id` attribute that will be an alias for the `value` attribute. The only conditions for this to work are that the annotation declaring the alias has to be annotated with the meta-annotation, and the `annotation` attribute must must reference the meta-annotation.

```
package com.ps.sample;
import org.springframework.core.annotation.AliasFor;

@Repository
public @interface MyRepoCfg {

    @AliasFor(annotation = Repository.class, attribute = "value")
    String id() default "";
    }
```

Thus, the repository beans can be declared as depicted in the following code snippet:

```
...
import com.ps.MyRepoCfg;
...

   //JdbcRequestRepo. java bean definition
@MyRepoCfg(id = "requestRepo")
public class JdbcRequestRepo  extends JdbcAbstractRepo<Request>
        implements RequestRepo{
...
}
```

! And in future editions of the book, will be replaced with this:

And even more can be done. Aliases for meta-annotation attributes can be declared. Let's say that we do not like the `lTout` attribute name of the `@DsCfg` annotation and we just want to reffer to it differently. The only conditions for this to work are that the annotation declaring the alias has to be annotated with the meta-annotation, and the `annotation` attribute must must reference the meta-annotation.

```
package com.ps;

import org.springframework.core.annotation.AliasFor;

@DsCfg
public @interface MyDsCfg {

    @AliasFor(annotation = DsCfg.class, attribute = "lTout")
    int timeout() default 200;
}
```

Thus, we can now apply the new annotation on the repository beans, that can be declared as depicted in the following code snippet:

```
...
import com.ps.MyDsCfg;
...

   //JdbcRequestRepo. java bean definition
@MyDsCfg(timeout = 2400)
@Repository
public class JdbcRequestRepo  extends JdbcAbstractRepo<Request>
        implements RequestRepo{
...
}
```

---

! `@AliasFor` cannot be used on any stereotype annotations. The reason is that the special handling of these `value` attributes was in place years before `@AliasFor` was invented. Consequently, due to backward compatibility issues it is simply not possible to use `@AliasFor` with such value attributes. When writing code to do just so (aliasing `value` attributes in stereotype annotations), no compile errors will be shown to you, and the code might even run, but any argument provided for the alias will be ignored. the same goes for the `@Qualifier` annotation as well.

---

This information about the `@AliasFor` annotation can be found here: https://github.com/spring-projects/spring-framework/wiki/Spring-Annotation-Programming-Model#1-can-aliasfor-be-used-with-the-value-attributes-for-component-and-qualifier [1]

The sources have also been updated on the raw repository: `https://github.com/iuliana/pet-sitter`

## Chapter 4: Aspect Oriented Programming with Spring

In the quiz section, **Question 5, answer D** there is a small typo.
Original: *execution(* update(Logn,..))*
Correct: **execution(* update(Long,..))**

## Appendix A

In the Appendix A, the quiz solution section for Chapter 2, at editing, the numbering of the answers was misplaced, so the number of the questions and the answers, do not fit. There are also small errors in the answers to questions 4 and 8. The full list of answers with proper numbering and corrections is depicted below:

**Quiz Solution for Chapter 2**

1. **Answer:** A, B, C
2. **Answer:** B
3. **Answer:** A, B,D (C, Interface-based injection is not supported in Spring. D, field-based injection is supported by annotating fields with `@Autowired`, `@Value` or related annotations; JSR-250 `@Resource`, JSR-330 `@Inject`.[2])
4. **Answer:**
   Original: *Answer: A, B, C (As stated in Chapter 2)*
   Correct: **A** (`ClassPathXmlApplicationContext` is a simple convenience `ApplicationContext`

[1] The URL does not fit the page, but for electronic use the clickable URL is this: FAQ @AliasFor

[2] Spring Framework Reference: `http://docs.spring.io/spring/docs/4.2.3.RELEASE/spring-framework-reference/htmlsingle/#beans-annotation-config`

implementation that can be used to load the definitions from the given XML files and automatically refreshing the context.)

5. **Answer:**
   Original: *B (Only* `@Component` *is contained in the* `org.springframework.stereotype` package)
   Correct: **A (Only** `@Component` **is contained in the** `org.springframework.stereotype` package)
   (The comment pointed to the correct answer, the letter was incorrect. Many thanks to Tobias Hochgürtel for noticing this.)
   **Clarification**: Many readers have submitted questions about this answer asking why `@Configuration` is not considered a stereotype annotation. According to the Spring Reference Documentation we could consider that `@Configuration` is a marker for any class that fulfills the role or stereotype of a configuration class. But it is not explicitly mentioned as such in the documentation `https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/htmlsingle/#beans-stereotype-annotations`. Any new annotation using `@Component` is not necessary automatically a new stereotype. Sure, `@ComponentScan` is able to detect it according a technical perspective because it has `@Component`, but semantically would has no sense, because a bean of type configuration will rarely be used for anything else than bean declaration. Thus, `@Configuration` is in the edge in some way. It can be considered a *kind* of stereotype, but only for infrastructure.

6. **Answer:** A

7. **Answer:** C

8. **Answer:**
   Original: *Answer: A,B,C*
   Correct: **A, B**.
   C is not a good answer, because of the `p:petitBean` definition. The `quizBean` has a dependency on the `petitBean` bean, thus a reference to it is needed. The equivalent bean definition, that would have made the original answer correct is:

   ```
   <bean id="quizBean" class="QuizBean"
                   init-method="initMethod" p:petitBean-ref="petitBean" />
   ```

   The B answer is correct because the declaration of the bean, provides a setter to inject the `petitBean` dependency. That setter can be called on the bean after it is being created by the container. As the setters are used when properties are not actually mandatory, `@Autowired` is not actually needed here to make the answer valid. Using setters means that the dependency can be set later in the bean lifecycle, not only at bean creation time.

9. **Answer:** B (`CommonAnnotationBeanPostProcessor` modifies beans)

10. **Answer:** C

11. **Answer:** B

12. **Answer:** A

**Quiz Solution for Chapter 4**

In the Appendix A, the quiz solution section for Chapter 4, for unknown reasons, the answer for Question 8 contains a mistake. *(Reported by Patrick Dobner)*
Original: *Answer: A, B*
Correct: **B** (an expression to identify methods to which the advice applies to), **C** (a predicate used to identify join points)

**Sample Exam**

Question 1 has a *(choose one)* after the question content, which is confusing as the answer is made of more than one option. *(Reported by Tibor Kalman)*
Original: Which of the following affirmations is false? (choose one)
Correct: Which of the following affirmations is false? Question 3 has an incorrect numbering of the answers.
Original:
Are the following two bean declaration equivalent?

A. (1)
```
<bean id="someBean" p:someValue-ref="someOtherBean" />

(2)
@Component("someBean")
public class SomeBean {
        private SomeBean someValue;

        @Autowired
        @Qualifier("someOtherBean")
        public setSomeValue(SomeBean arg){
                this.someValue = arg;
        }
}
```
yes

B. no

C. the second is not a valid bean definition

Correct: Are the following two bean declarations equivalent?

```
(1)
<bean id="someBean" p:someValue-ref="someOtherBean" />

(2)
@Component("someBean")
public class SomeBean {
        private SomeBean someValue;

        @Autowired
        @Qualifier("someOtherBean")
        public setSomeValue(SomeBean arg){
                this.someValue = arg;
        }
}
```

A. yes

B. no

C. the second is not a valid bean definition

**Sample exam, Answers**

In the Answers section, the answer for question 30 is wrong.
Original: *30. A*
Correct: **30. B**

## B.2 Code updates and observations

In the 1.4.1 version of the Gretty plugin the `port` property was renamed to `httpPort`. Thus, old Gradle configurations like the one depicted below will cause a build failure.

```
gretty {
    port = 8080
    contextPath = '/mvc-layout'
}
```

The configuration must be corrected and the `port` property must be replaced with `httpPort`

```
gretty {
    httpPort = 8080
    contextPath = '/mvc-layout'
}
```

In case you want more information, or keep track of the issue I created related to this, take a look on the Gretty plugin GitHub page: `https://github.com/akhikhl/gretty`