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 Certification Exam** needs an update on pages 3 and 4. Starting with January 2017 the exam is taken directly from home and the voucher is valid only three months. *(Submitted by Vittorio Marino)*

Chapter 2: Spring Bean LifeCycle and Configuration

In the section **Setter Injection** section, page 34 in the examples a bean named `simpleBean0` is declared, but not used as a dependency in the `complexBean`. The intended name, was obviously `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`.

Other small typos and mishaps.

Page	Original	Correction
43	Equivalent to `list.add(new SimpleBeanImpl).`	Equivalent to `list.add(new SimpleBeanImpl()).`
63	`for (String name : ctx.getAliases("sb02/sb03") ){ logger.info ("Alias for sb04 ->" + name); }`	`for (String name : ctx.getAliases("sb02/sb03") ){ logger.info ("Alias for` **sb02/sb03** `->" + name); }`
66	`INFO c.p.ApplicationContextTest - >> bye bye.`	Leftover log entry from code that was removed from the book.
72	The `InstantiatingBean` interface...	The **InitializingBean** interface...

Table B.1: Corrections Table (part 1)

In the section **Bean Naming section**, page 94, 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 `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{
...
}
```

! 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 refer 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/springframework/wiki/Spring-Annotation-Programming-Model#1-can-aliasfor-be-used-with-the-value-attributes-for-componentand- qualifier. [1]

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

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

Other small typos and mishaps.

Page	Original	Correction
103	The `@Autowired` annotation by default requires the dependency to be mandatory, but this behavior can be changed, by setting the required attribute to `true:`	The `@Autowired` annotation by default requires the dependency to be mandatory, but this behavior can be changed, by setting the required attribute to **false:**
105	Table 2-3. *Prefixes and corresponding paths*	Table 2-3. **Spring and JSR 250 annotation equivalents**
144	When classes are annotated with this anclasses become...	When classes are annotated with this **annotation, classes** become
145	`public DataSource _underlinedataSource() throws SQLException {`	`public DataSource` **dataSource()** `throws SQLException {`

Table B.2: Corrections Table (part 2)

Chapter 4: Aspect Oriented Programming with Spring

Page	Original	Correction
160	`import org.aspectj. lang.annotation.Component; import org.aspectj. lang.annotation.Aspect; import underlineorg.aspectj. lang.annotation.Before;`	**import org.springframework.stereotype.Component;** `import org.aspectj. lang.annotation.Aspect;` **import org.aspectj.lang.annotation.Before;**
165	`pointcut="execution(public com.ps.repos .'JdbcTemplateUserRepo+ .findById(..))"`	`pointcut="execution(public * com.ps.repos.* .JdbcTemplateUserRepo+ .findById(..))"`
166	`public * com.ps.repos .*.JdbcTemplateUserRepo+ .findById(..)) && +underlinewithin(com.ps.*)`	`public * com.ps.repos .*.JdbcTemplateUserRepo+ .findById(..)) && within(com.ps.*)`
171 172	`if (StringUtils.indexOfAny(pass, new String{"$", "#", "$", "%"}) != -1)`	`if (StringUtils.indexOfAny(pass, new String[]{"$", "#", "$", "%"}) != -1)`
182	`execution(* update(Logn,..))`	**execution(* update(Long,..))**

Table B.3: Corrections Table (part 3)

Chapter 5: Data Access

Page	Original	Correction
193	`public JdbcTemplate ~underlinejdbcTemplate()`	`public JdbcTemplate` **jdbcTemplate()**
205	`new Object{username}, rowMapper));`	`new Object[]{username}, rowMapper));`
211	Figure 5.8 Bad caption: Created by a `EntityManagerFactoryBean`	Figure 5.8 Correct caption: Created by a `FactoryBean<EntityManagerFactory>`
225	The `@Transactiona` annotation...	The `@Transaction` annotation...
227	`<tx:advice id= "transactionalAdvice"> <tx:atributes>`	`<tx:advice id= "transactionalAdvice">` **<tx:attributes>**
241	`List<Users> list = session .createQuery("from User u where username= ?")`	`List<Users> list = session()` `.createQuery("from User u where username= ?")` **.setParameter(0, username).list();**
249	`public class TestDataConfig {@Bean...`	`public class TestDataConfig { ...`
263	Every `RepoMongoRepositorysitory` interface...	Every **MongoRepository** interface...

Table B.4: Corrections Table (part 4)

In page 220, in the nested transaction example, there is a message in the log that needs more explaining `# Not actually nested.` That message is necessary because the current version of H2 when this book is being written does not support nested transactions.

In page 222 there is a code snippet used as example for the `rollbackFor` attribute that is not correct.
! The original (erroneous) snippet is this:

```
@Transactional(rollbackFor = MailSendingException.class)
@Override
public int updatePassword(Long userId, String newPass)
          throws MailSendingException {
    User u = userRepo.findById(userId);
    String email = u.getEmail();
    sendEmail(email);
    return userRepo.updatePassword(userId, newPass);
}
```

! The problem with this code snippet is that the exception is thrown before calling `userRepo.updatePassword(..)` to execute the actual operation that we are interested in. And in future editions of the book, will be replaced with this:

```
@Transactional(rollbackFor = MailSendingException.class)
@Override
public int updatePassword(Long userId, String newPass)
           throws MailSendingException {
    User u = userRepo.findById(userId);
    String email = u.getEmail();
    int result = userRepo.updatePassword(userId, newPass);
```

```
        sendEmail(email);
        return result;
    }
```

Same goes for the example in section **Spring Programatic Transaction Model**, page 233, where the code below:

```
try {
        User user = userRepo.findById(userId);
        String email = user.getEmail();
        sendEmail(email);
        return userRepo.updatePassword(userId, newPass);
} catch (MailSendingException e) {
        status.setRollbackOnly();
}
```

Should be replaced with:

```
try {
        int result = userRepo.updatePassword(userId, newPass);
        User user = userRepo.findById(userId);
        String email = user.getEmail();
        sendEmail(email);
        return result;
} catch (MailSendingException e) {
        status.setRollbackOnly();
}
```

In page 267 question 10, the quiz solution reads A,B,D,E,F. But one of the interfaces in the answer is never mentioned in the book: `TransactionDefinition`. *(Observation by Patrick Dobner)*Thus we are compensating for that here. The **Spring Programatic Transaction Model** section is missing the following paragraphs and code samples.

Another way of using transactions programmatically is to use the transaction manager directly. Spring's abstract transaction management unit is the `PlatformTransactionManager` interface. Regardless of the transaction manager framework provider used in an application a bean of type implementing `PlatformTransactionManager` can be injected and used directly. This interface defines three methods depicted in the code snippet below:

```
package org.springframework.transaction;

public interface PlatformTransactionManager {
   TransactionStatus getTransaction(TransactionDefinition definition)
         throws TransactionException;

   void commit(TransactionStatus status) throws TransactionException;

   void rollback(TransactionStatus status) throws TransactionException;
}
```

The `getTransaction(..)` returns a currently active transaction or creates and returns a new one, all depending on the transaction manager configuration. The parameter of type `TransactionDefinition` encapsulates transaction properties like isolation level, propagation behaviour, timeout, etc. It returns an object representing a transaction status of type implementing Spring's `TransactionStatus` interface. This object is used by the transaction manager to modify the transaction object.
The `commit(..)` and `rollback(...)` methods purpose is obvious because of their naming, they are the methods

used by the transaction manager to change transaction status, depending on the success or failure of a transaction. The code snippet below depicts the previous code sample, written by using the transaction manager bean directly. As you can see the `PlatformTransactionManager` is autowired by Spring and can be used to commit or rollback a transaction.

```
package com.ps.services.impl;

...
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;
import javax.sql.DataSource;

@Service("programaticUserService2")
public class ProgramaticUserService2 implements UserService {

    private JdbcTemplate jdbcTemplate;
    private PlatformTransactionManager transactionManager;

    @Autowired
    public ProgramaticUserService2(DataSource dataSource,
          PlatformTransactionManager txManager) {
        this.transactionManager = txManager;
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

   @Override
    public int updatePassword(Long userId, String newPass)
         throws MailSendingException {
        TransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);

        String sql = "update P_USER set PASSWORD=? where ID = ?";
        int result;
        try {
            result  = jdbcTemplate.update(sql, new Object[]{newPass, userId});

            sql = "select u.ID as ID, u.USERNAME as USERNAME," +
                       " u.USER_TYPE as USER_TYPE," +
                       " u.EMAIL as EMAIL, u.PASSWORD as PASSWORD" +
                       " from P_USER u where u.ID = ?";
            User user = jdbcTemplate.queryForObject(sql,
                       new Object[]{userId}, new UserRowMapper());
            transactionManager.commit(status);
            String email = user.getEmail();
            sendEmail(email);
            return result;
        } catch (MailSendingException e) {
            status.setRollbackOnly();
        } catch (Exception e) {
```

```
            // do rollback for any exception except MailSendingException
            transactionManager.rollback(status);
            throw e;
        }
        return 0;
    }
   ...
}
```

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` 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*

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

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" />
```

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`

Most small typos and formatting corrections were submitted by *Marco Pelucchi*. Thank you kind sir for using your attention to detail to make this book better!