Chapter A

The purpose of this Appendix is to help you set up the development environment that you will use to write, compile, and execute the Spring applications specific to this book and to provide detailed responses for the questions in the quiz attached to each chapter.

Study Guide Projects

The appendix for this book is quite small because it was written in such a way that you are guided through the normal succession of steps that you would have to through every time you start development of an application.

- Choose your tools
- Install your tools
- Verify installation
- Design the application
- Develop & test

At the end of Chapter 1, the tools were presented to you and you were instructed how to install them and how to verify the correct installation. The code samples for the book were written on an Apple Mac computer. One of the strong points of Java is that is multi-platform, so the code can be run in any operating system. The tools recommended are Java based and versions for any operating system are available on their official sites. The installation instructions are almost identical for any operating system, the only difference is in how a environment variable is set. Information about doing this on different operation systems is widely available on the internet and considering this book is for developers with a little experience, this should not be a problem for you.

The code for this book is contained into a single gradle multi-module project named `pet-sitter`. The project follows the evolution of a Spring application. It was build incrementally and module names were prefixed with a number; if traversed in the ascending order of their prefixes, you will notice that every module contains the code of the previous one and something extra.The modules are referred in a chapter in the ascending order of their indexes. The only exceptions from the rule are the Spring Boot projects, because Spring Boot comes with a predefined set of dependencies, thus these projects cannot inherit too much configuration from the `pet-sitter` project. It simulates the evolution of the configuration, from XML to Java Configuration, and Spring Boot where the case is suitable. Exchanging and adding libraries until the final form of a complete web application is reached, when security and evolved components as web flows are in placed.

Using this study guide you will not only learn how to build Spring applications, you will learn how to design a workflow for you and your team and how to design a multi-layered application from scratch.

The code was split in two, because some topics like alternative configurations and alternative view technologies needed to be presented separately without overcrowding the `personal-records` project. So really small modules covering these were created and wrapped up in a different project called `book-code`.

The project structure is depicted in Figure A.1

Figure A.1: personal-records and book-code project structures in Intellij IDEA side by side

Gradle Configuration Explained

The `pet-sitter` is the parent project that defines a set of libraries available for the children modules to use. The parent project has the Gradle configuration in a file typically named `build.gradle`. All the modules have the Gradle configuration file name named as the module, `[module_name].gradle`, for example *16-ps-jmx-sample.gradle*, so the user can quickly find the configuration file. Also, there's a closure element in `pet-sitter/settings.gradle` that verifies at build time if all modules have their configuration file present.

```
rootProject.children.each { project ->
    project.buildFileName = "${project.name}.gradle"
    assert project.projectDir.isDirectory()
    assert project.buildFile.exists()
    assert project.buildFile.isFile()
}
```

If a Gradle build file is not named the same as the module, when executing any Gradle task and error is thrown. The error is similar to the one depicted in the following code snippet, where the `16-ps-jmx-sample.gradle` was renamed to `16-ps-jmx-sample2.gradle`.

```
$ gradle clean
Starting a Gradle Daemon (subsequent builds will be faster)
 This project is an asset management POC application

FAILURE: Build failed with an exception.

* Where:
Settings file '/Users/iuliana.grajdeanu/apress/workspace/pet-sitter/settings.gradle'
                                                                    line: 48

* What went wrong:
A problem occurred evaluating settings 'pet-sitter'.
> assert project.buildFile.exists()
         |       |         |
         |       |         false
         |       /Users/iuliana.grajdeanu/apress/workspace/pet-sitter/
                                    16-ps-jmx-sample/16-ps-jmx-sample.gradle
         :16-ps-jmx-sample

* Try:
Run with --stacktrace option to get the stack trace.
   Run with --info or --debug option to get more log output.

BUILD FAILED
```

This was a development choice, the configuration file of a module is also more visible in an editor this way. Plus, if you want to modify the configuration file for a module you can easily find the file in IntelliJ IDEA using a unique name. Imagine the pain if you use the `Command+Shift+N` to search for a specific `build.gradle` files and you have 20+ matches.

Another approach for a multi-modular project would have been to have only one `build.gradle` file for the whole project and use Gradle specific closures in order to customize configuration for each module. But in the spirit of good development practices, it was decided to keep configurations for the modules as decoupled as possible and in the same location with the module contents.

Building and troubleshooting

After you download the sources you need to import the project in the IntelliJ IDEA editor. To do this, the following steps must be followed:

[Step 1] Select from the Intellij IDEA menu `File -> New -> Project from Existing Sources .` (Menu options are depicted in Figure A.2)

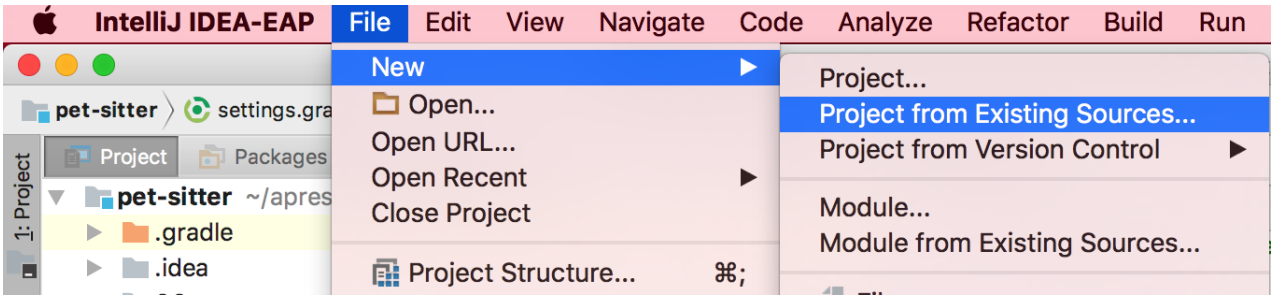

Figure A.2: Project import menu options in Intellij IDEA

After selecting the proper option a popup window will appear requesting the location of the project.(Figure A.3)

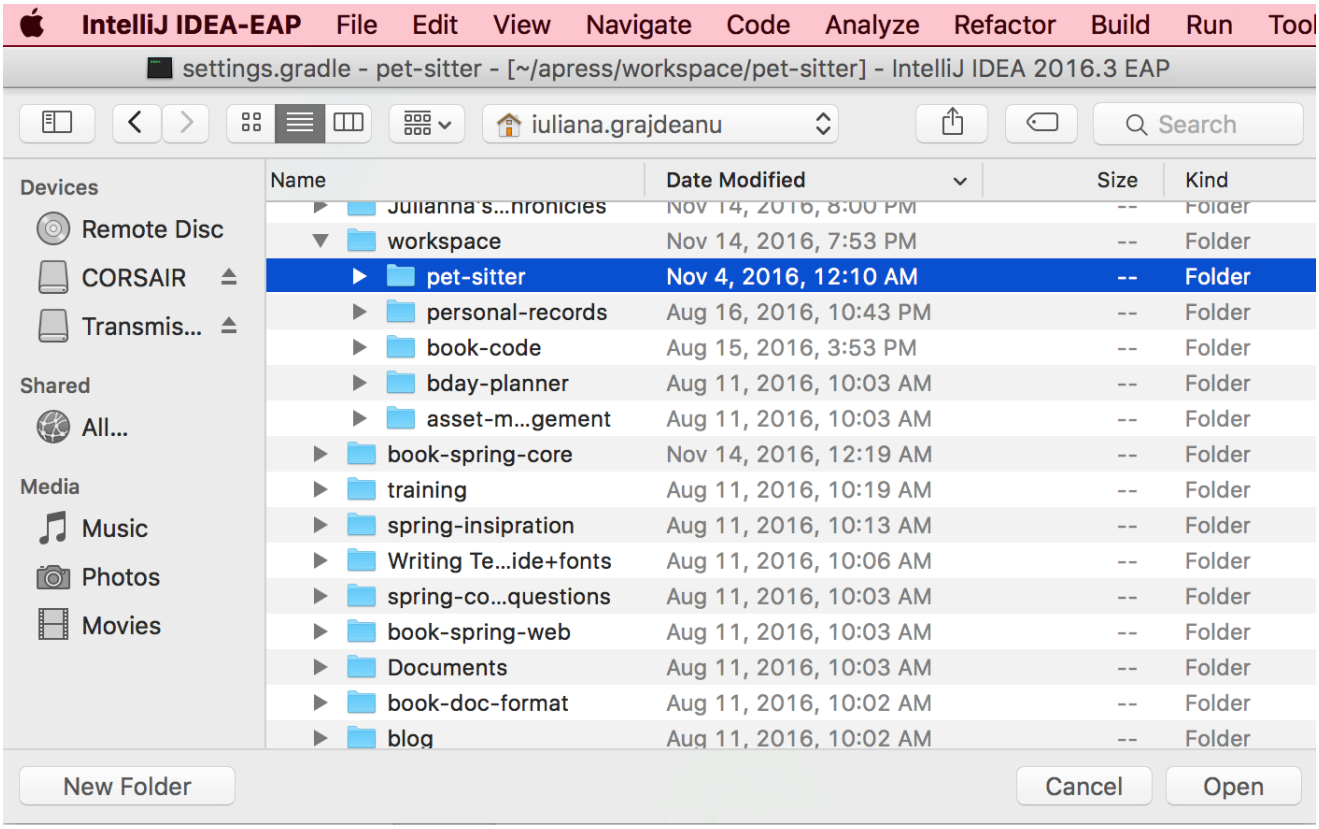

Figure A.3: Select project root directory popup in Intellij IDEA

[Step 2] Select the `pet-sitter` directory.The following popup will ask for the project type. Intellij IDEA can create its own type of project from the selected sources and build it with its internal Java builder, but this option is not useful here as `pet-sitter` is a Gradle project.

[Step 3] Check the `Import project from external model` radio button and select `Gradle` from the menu as depicted in Figure A.4

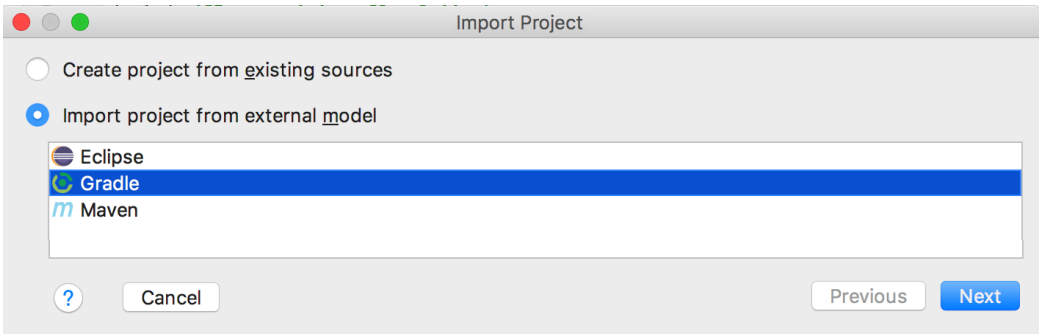

Figure A.4: Selecting the project type Intellij IDEA

[Step 4] The last popup will appear and ask for the location of the `build.gradle` file and the Gradle executable. The options will be already populated for you. If you have Gradle installed in the system you might want to use it.(Figure A.5)

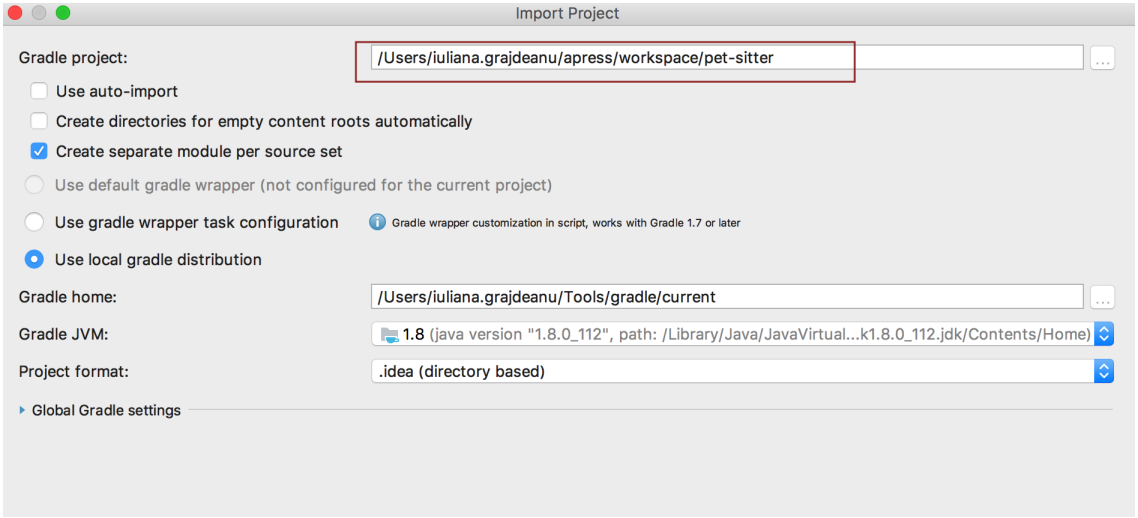

Figure A.5: Last popup for project import in Intellij IDEA

Before getting to work you should build the project. This can be done from Intellij IDEA by clicking the `Refresh` button, marked with (1) in in Figure A.6. Clicking this button will cause IntelliJ IDEA to do the following: scan configuration of the project, resolve dependencies, this includes downloading missing libraries and do an internal light build of the project, just enough to remove compile time errors cause by missing dependencies.

The Gradle build task executes a full build of the project. Can be used in the command line

```
.../workspace/pet-sitter $ gradle build
```

or from Intellij IDEA as depicted in Figure A.6, where the task is marked with (2).

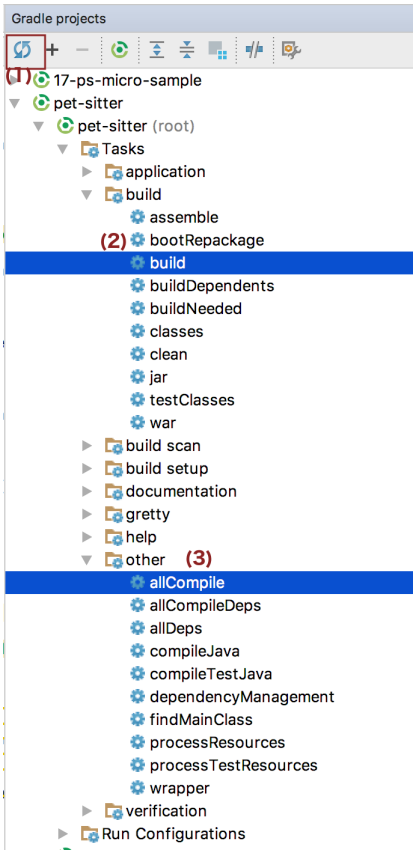

Figure A.6: Last popup for project import in Intellij IDEA

It will execute the following set of tasks on every module:

```
:00-ps-core:compileJava
:00-ps-core:processResources
:00-ps-core:classes
:00-ps-core:jar
:00-ps-core:assemble
:00-ps-core:check
:00-ps-core:build
```

In the previous example, the tasks were depicted only for module `00-ps-core`. The Gradle build task will execute all the tasks it depends on. As it can be seen, it does not run the `clean` task, so you need to make sure to run this task manually when building a project multiple times, to make sure the most recent versions of the classes will be used.

As the project contains incomplete sources in modules postfixed with `-practice` that you will have to complete, executing this task will fail. You could just execute tasks clean and compileJava, but there's a better way. A custom task was created in the project called `allCompile`. This task executes the clean and compileJava tasks for all modules. It is marked with (3) Figure A.6. in It is defined in file `build.gradle` and inherited in child modules, so it can be executed for a module separately.

Another option is to execute the Gradle `build` task, but to skip the tests with the following argument

```
.../workspace/pet-sitter $ gradle build -x test
```

Deploy on Apache Tomcat

There are a few web applications under the `pet-sitter`. Every web application in this project is run with the Jetty embedded web server to keep things simple. But there are certain advantages in using an external container like Apache Tomcat server. Starting the server in debug mode and using breakpoints to debug an application is much easier to do is only one advantage. An external container can run multiple application at a time without the need to stop the server.Embedded servers are as useful for testing and educational purposes as Spring Boot is, but in practice application servers are preferred.

Here is what you have to do if you are interested in doing this. First download the latest version of Apache Tomcat from their official site[1], you can get 8 or 9, they will both work with the sources of this book, and unpack it somewhere on your system. Then configure an IntelliJ IDEA launcher to start the server and deploy the chosen application. This is quite easy to do, but there is a number of steps to be executed and they are listed below:

[Step 1] From de runnable configuration menu choose `Edit configurations...`(1). A popup window will appear listing a set of Launchers. Click on the **+** and select the `Tomcat Server` option. The menu will expand, select `Local(2)` because you are using a server installed on your computer. Figure A.7 depicts these menu options.

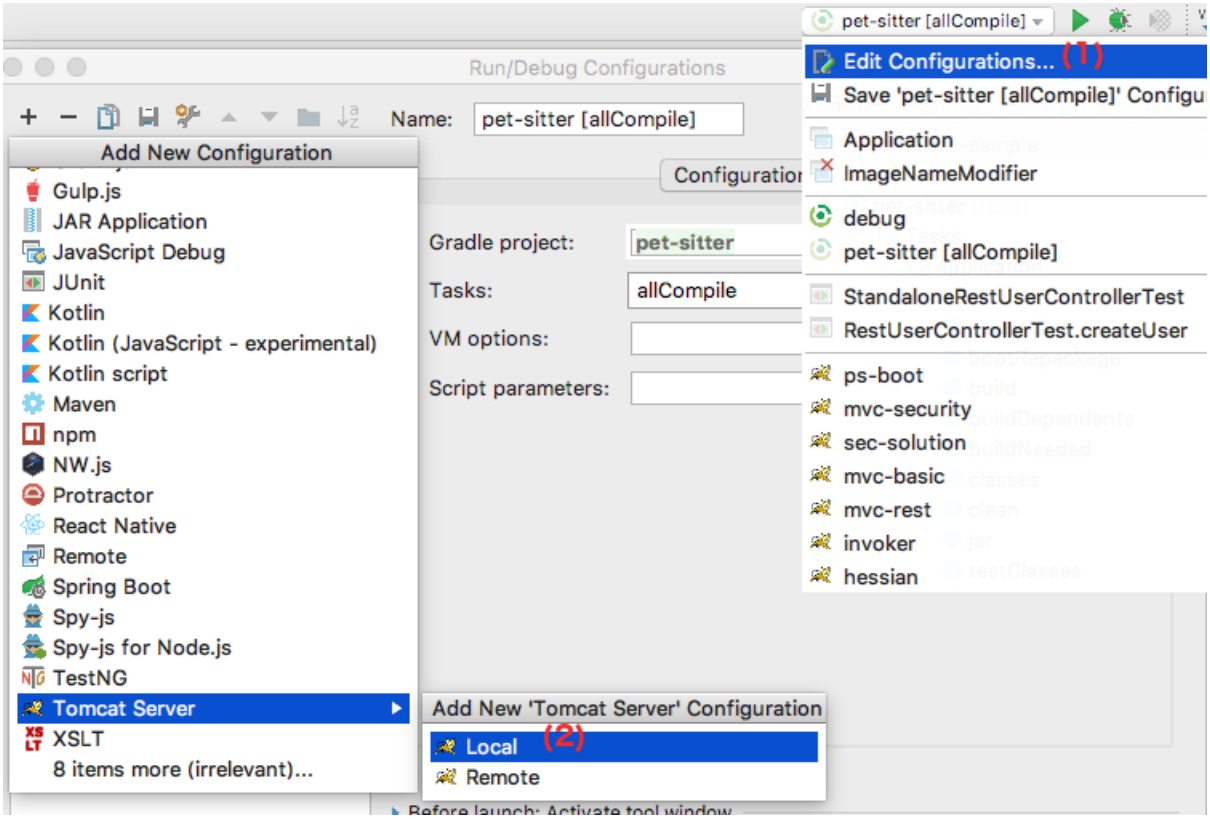

Figure A.7: Menu options to create a Tomcat launcher in Intellij IDEA

[1] Apache Tomcat official site `http://tomcat.apache.org/`

[Step 2] A popup window like the one in Figure A.8 will appear and will request some information.

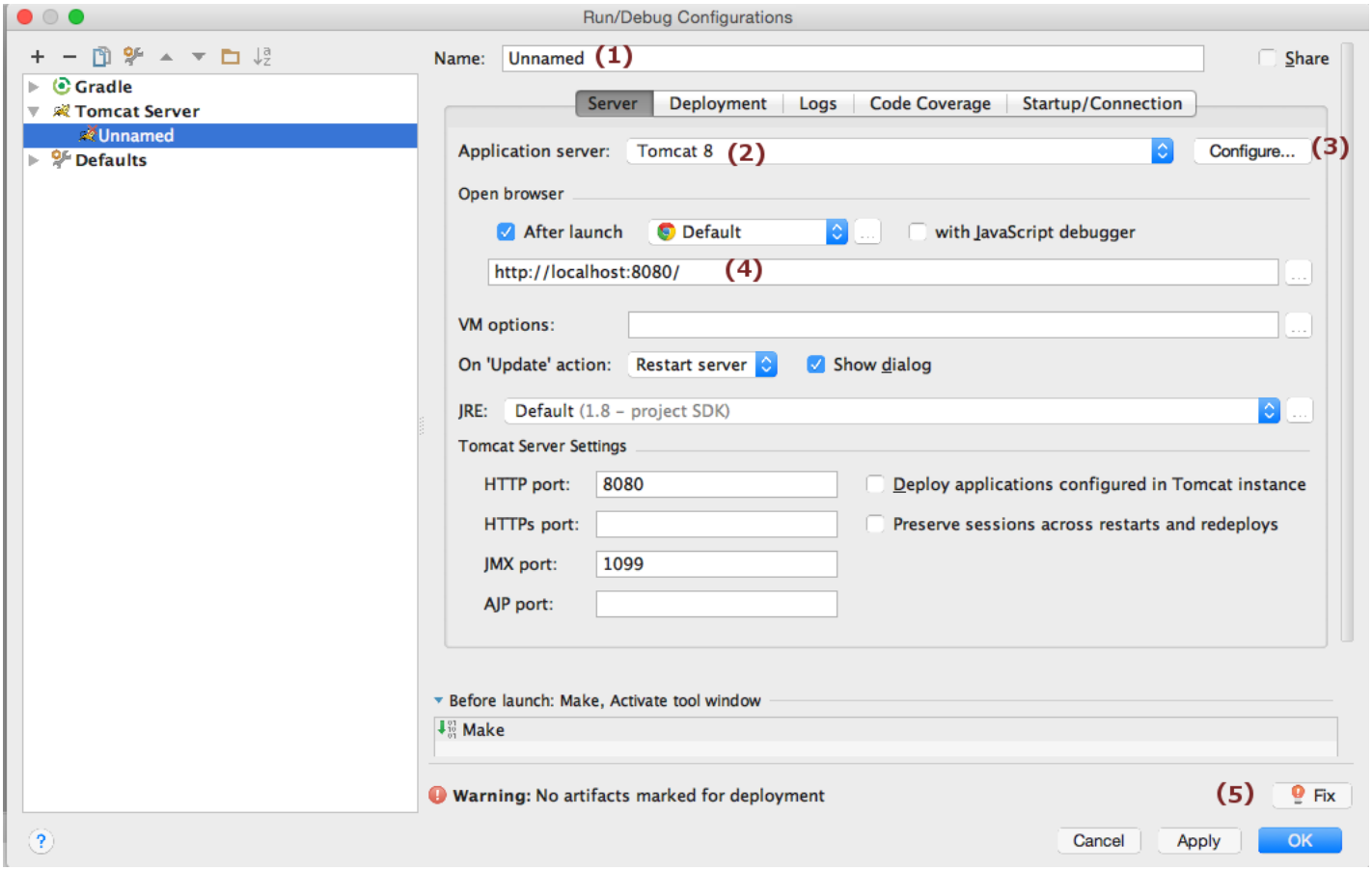

Figure A.8: Popup to create a Tomcat launcher in Intellij IDEA

In the previous figure, some items are numbered and their meaning is explained in following list:

1. The launcher name, yo
2. The Tomcat instance name.
3. The button that will open the popup window to insert the Tomcat instance location. The popup is depicted in Figure A.9.

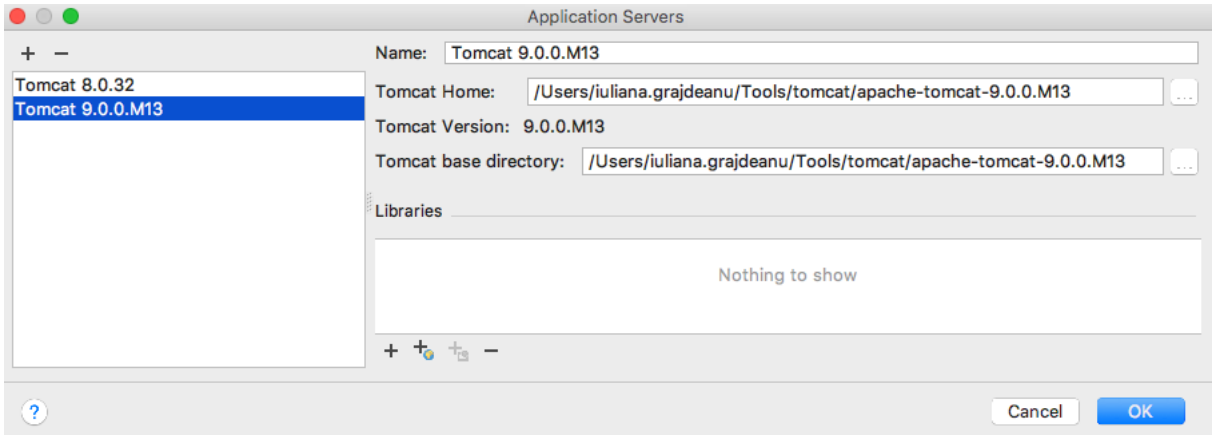

Figure A.9: Configure Tomcat instance in Intellij IDEA

4. The URL where the Tomcat server can be accessed
5. The `Fix` button is used to choose an artifact to deploy on the Tomcat instance. If there is no web archive set to be deployed to Tomcat, this button will be displayed with a red lightbulb icon on it.

[Step 3] Click the `Fix` button and select an artifact. IntelliJ IDEA will detect all artifacts available (Figure A.10) and present them to you in a list. If you intend to open the server in debug mode and use breakpoints in the code, select an artifact with the name postfixed with `(exploded)`, this way Intellij IDEA manages the contents of the exploded war and can link the actions in the browser with the breakpoints in the code.

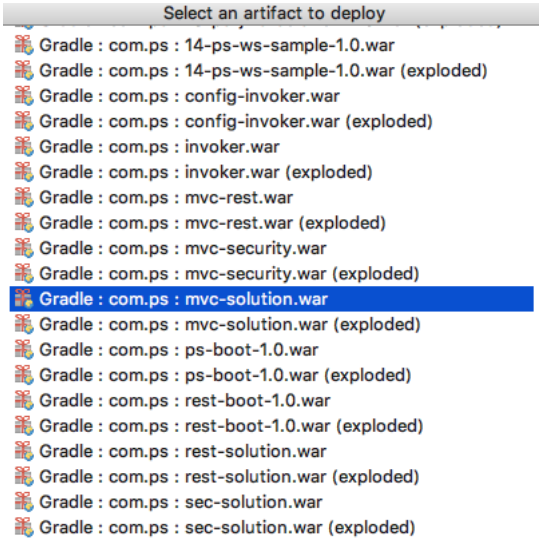

Figure A.10: Deployable artifact list in Intellij IDEA

[Step 4] Complete the configuration by clicking the `Ok` button. You can specify a different application context by inserting a new value in the `Application Context` field. Choosing a different application context will tell Tomcat to deploy the application under the given name and the application will be accessible via URL: `http://localhost:8080/[app_context_name]/`. In Figure A.10, you can see the the `Application Context` field. The application configured like in the image will be accessible via URL `http://localhost:8080/mvc-solution/`.

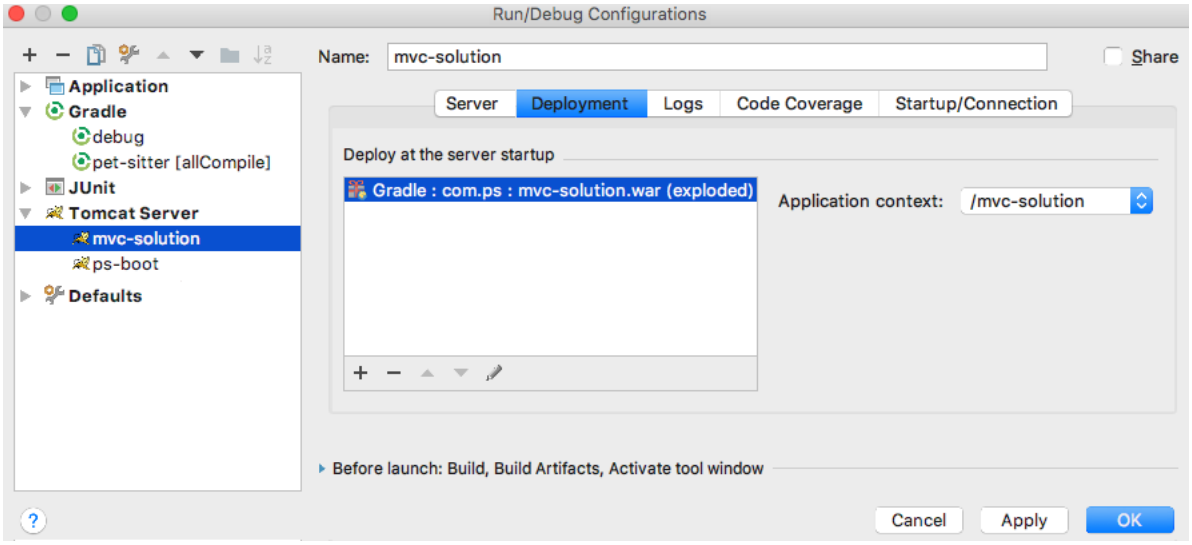

Figure A.11: Insert new application context in Intellij IDEA

Other application servers can be used in a similar way as long as IntelliJ IDEA provides support for them. Launcher configurations can be duplicated, multiple Tomcat instances can be started at the same time as long as they function on different ports, all for faster testing and comparisons between implementations. IntelliJ IDEA is really flexible and practical and that's why it was recommended for practicing the exercises in this study guide. The Gradle projects can also be imported in Eclipse and other JAVA editors that support Gradle.

Quiz answers

The following sections contains answers to the Quiz questions for all chapters. Answers to questions that are simple enough, will not be detailed here. Extra details will be provided only for questions that could be considered tricky.

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:** A, B, C (As stated in Chapter 2)
5. **Answer:** B (Only `@Component` is contained in the `org.springframework.stereotype` package)
6. **Answer:** A
7. **Answer:** C
8. **Answer:** A,B,C
9. **Answer:** B (`CommonAnnotationBeanPostProcessor` modifies beans)
10. **Answer:** C
11. **Answer:** B
12. **Answer:** A

Quiz Solution for Chapter 3

1. **Answer:** C
2. **Answer:** A, D
3. **Answer:** B
4. **Answer:** B
5. **Answer:** A (To write unit tests for a Java application, even a Spring application JUnit is mandatory.)

Quiz Solution for Chapter 4

1. **Answer:** B, C, D
2. **Answer:** A, B, C, D, E (`@Aspect` is used on the class containing advice methods, `@Pointcut` is used to define where the advice applies and `@Before`, `@After` and `@AfterReturning` are used to define when the advice applies. Thus all of them are needed to declare an advice.)
3. **Answer:** C (The `@Before` annotated advice executes before the target, so if the target method throws an exception, its execution won't be affected. The `@After` advice is executed no matter how the execution of the target methods ends: normally or by throwing an exception, but it cannot prevent exceptions propagation. The `@AfterThrowing` advice will be executed only if the target method execution end by throwing an exception.)

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

4. **Answer:** A
5. **Answer:** D
6. **Answer:** B
7. **Answer:** A
8. **Answer:** A, B

Quiz Solution for Chapter 5

1. **Answer:** A,B,C (D is only supported by Spring Data custom implementations)
2. **Answer:** A
3. **Answer:** A,C
4. **Answer:** A
5. **Answer:** C,D
6. **Answer:** A,B,C,E
7. **Answer:** A,B
8. **Answer:** A
9. **Answer:** A
10. **Answer:** A, B, D, E, F
11. **Answer:** B (Spring default behavior for declarative transaction management follows EJB convention, roll back is automatic only on unchecked exceptions[3])
12. **Answer:** C (when declarative transactions are used)
13. **Answer:** B (Because transactional behavior is implemented with AOP, there will be a single transaction if the methods are declared within the same bean, but if they are declared in different beans, like the example presented in the chapter with a transactional service calling a transactional repository - which is bad practice, by the way because of an useless transaction being created and atomicity of operations not being ensured - there will be two transactions. So, assume recommended and good practices when answering questions.)
14. **Answer:** A, B, C
15. **Answer:** A,C,D
16. **Answer:** A,C
17. **Answer:** A,B,C
18. **Answer:** A
19. **Answer:** B

[3]Official documentation reference: `http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/transaction.html#transaction-declarative`

Quiz Solution for Chapter 6

1. **Answer:** B,E
2. **Answer:** A
3. **Answer:** B
4. **Answer:** B
5. **Answer:** C
6. **Answer:** C
7. **Answer:** B
8. **Answer:** B
9. **Answer:** C
10. **Answer:** A
11. **Answer:** D, E (These are equivalent options to configure security at web and method level, so they can both be selected)
12. **Answer:** B,C,D

Quiz Solution for Chapter 7

1. **Answer:** C
2. **Answer:** A, B, D
3. **Answer:** C
4. **Answer:** A
5. **Answer:** A, B
6. **Answer:** B
7. **Answer:** A, B (C does not exist)
8. **Answer:** A, C, D
9. **Answer:** D

Quiz Solution for Chapter 8

1. **Answer:** A (B is not correct because other communication protocols can be used and C describes a monolith architecture)
2. **Answer:** A, B
3. **Answer:** C
4. **Answer:** A, B
5. **Answer:** C
6. **Answer:** A

Exam sample

Question 1: Which of the following affirmations is false?

A. inversion of control is a common pattern in the Java community that helps wire lightweight containers or assemble components from different projects into a cohesive application.

B. inversion of control is a programming technique, expressed in terms of object-oriented programming, in which object coupling is bound at run time by an assembler object and is typically not known at compile time using static analysis.

C. inversion of control is another name for dependency injection.

Question 2: Which of the following can be used to declare a bean with id "someBean"? (choose all that apply)

A. `<bean name="someBean" class="SomeBean"/>`

B.
```
@Repository("someBean")
public class SomeBean{
...
}
```

C.
```
@Service("someBean")
public class SomeBean{
...
}
```

Question 3: Are the following two bean declaration 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

Question 4: Which of the following are Spring Annotations? (choose all that apply)

A. @Resource

B. @Autowired

C. @PostConstruct

D. @PreDestroy

E. @Inject

F. @Component

Question 5: Which of the following is true about bean naming? (choose all that apply)

A. a bean can be declared without a name or id

B. it is possible for a bean to have multiple names

C. a bean can have multiple ids

D. two beans can have the same name

Question 6: What are the types of dependency injection supported by Spring IoC Container? (choose all that apply)

A. setter injection

B. constructor injection

C. interface-based injection

D. field-based injection

Question 7: Which of the following are valid bean scopes supported by Spring? (choose all that apply)

A. prototype

B. session

C. multiple

D. singleton

E. volatile

Question 8: What is true about the `@AliasFor` annotation? (choose one)

A. it can be used to declare an alias for a bean

B. is used to declare aliases for annotation attributes

C. is not part of the Spring framework

Question 9: What is true about classes annotated with `@Configuration`? (choose all that apply)

A. these class contains methods annotated with `@Bean` that are processed by the Spring IoC to generate bean definitions

B. these classes are bootstrapped using component scanning

C. these classes must be final

Question 10: A bean is created by Spring IoC in a few steps:

1. dependencies are injected
2. bean is instantiated
3. init-method is called

What is the correct order of the steps?

A. 2, 3, 1

B. 2, 1, 3

C. 1, 2, 3

Question 11: Given the following declaration of a bean:

```
@Configuration
public class Appconfiguration{

    @Bean
    public DataSource getDataSource() throws SQLException {
    ...
    }
}
```

What is the name of the bean that the Spring IoC container will create?

A. dataSource

B. getDataSource

C. DataSource

D. the name of the bean cannot be inferred.

Question 12: Your application contains join points, pointcuts and advice. What are you using?

A. data access

B. MVC

C. aspect oriented programming

Question 13: Which of the following pointcut expressions match methods in the class `com.ps.UseRepo`?

A. execution (* com.ps.UserRepo.update*(..))

B. execution (* com.*.UserRepo.*())

C. execution (* com.UserRepo.*(..))

Question 14: What type is the parameter of the `@Around` advice that is used to access the target method?

A. `JoinPoint`

B. `ProceedingJoinPoint`

C. `TargetJoinPoint`

D. `CallableJoinPoint`

Question 15: In the code snippet below, an advice and a repository class are depicted:

```
 @Before ( "execution * find*()")
 ... // advice body

public class UserRepo{
        public List<User> findAll(){
                if(restricted) {
                        return findRestricted();
                } else {
                        return allList;
                }
        }

        public List<User> findRestricted(){
                return restrictedList;
        }

  // allList and restrictedList are of type List<User>
}
```

If `userRepo` is used by a service class which methods will be advised?

A. `findAll`

B. `findRestricted`

C. `both`

D. neither

Question 16: What class is needed as argument for `@RunWith` to provide functionality of the Spring test framework to standard JUnit Tests?

A. `SpringUnitRunner`

B. `SpringJUnit4ClassRunner`

C. `SpringTestRunner`

Question 17: What annotation is needed on a class containing Spring integration tests to determine how to load and configure a Spring application context?

A. `TestContext`

B. `TestContextLoader`

C. `ContextConfiguration`

Question 18: What is the default rollback policy in transaction management?

A. Rollback for any Exception

B. Rollback for RuntimeException

C. Rollback for checked exceptions

D. Always commit

Question 19: `PlatformTransactionManager` is an interface that can be easily mocked or stubbed as necessary. Is this affirmation true?

A. yes

B. no

Question 20: An interface must be implemented and an object of this type is provided as argument to `JdbcTemplate` methods so rows in a table can be mapped to entity objects. Choose the interface from the following list.

A. `TableMapper<T>`

B. `RowMapper<T>`

C. `EntityMapper<t>`

Question 21: What kind of queries can `JdbcTemplate` execute? (choose all that apply)

A. JPQL queries

B. JDBC queries with '?' parameters

C. JDBC queries with named parameters

Question 22: Spring does not support programatic transactions. Is this true?

A. no

B. yes

Question 23: Which of the following describes the `REPEATABLE_READ` isolation level?

A. dirty reads are possible when a transaction has this isolation level set

B. repeatable reads are possible when a transaction has this isolation level set

C. phantom reads reads are possible when a transaction has this isolation level set

D. dirty, repeatable, phantom reads are not possible when a transaction has this isolation level set

Question 24: What can be said about the Spring Data Access Exceptions? (choose all that apply)

A. they hide the technology used to communicate with the database

B. they are checked and the root class of the hierarchy is `DataAccessException`

C. they extend `RuntimeException`

Question 25: Take a look at the code snippet below. IT depicts how programmatic transactions are implemented in Spring.

```
@Service("programaticUserService")
public class ProgramaticUserService
     implements UserService {
    private UserRepo userRepo;
    private TransactionTemplate txTemplate;

    @Override
    public User findById(Long id) {
        return txTemplate.execute(status -> {
            User user = null;
            try {
                user = userRepo.findById(id);
```

```
            } catch (Exception e) {
               status.setRollbackOnly();
            }
            return user;
        });
    }

}
```

What is the type of the `status` object?

A. `ExceptionStatus`

B. `JdbcStatus`

C. `TransactionStatus`

Question 26: Which is true about the `@SqlConfig` annotation

A. is not a test annotation, so it can be used outside of test context

B. defines metadata that is used to determine how to parse and execute SQL scripts configured by using `@Sql` annotation.

C. when declared at class level, the configuration defined by it is applied to all SQL scripts within the class hierarchy

Question 27: Why should a transaction be declared as `readOnly` ?

A. because this is mandatory with Hibernate

B. when a large set of data is read

C. because it may improve performance with Hibernate

Question 28: Analyze the following code snippet:

```
public interface UserRepo extends JpaRepository<User, Long> {

   @Query("select u from User u where u.username like %?1%")
    List<User> findAllByUserName(String username);
}
```

What can be said about the `findAllByUserName()` method?

A. it is not a valid Spring Data JPA repository method

B. the argument for `@Query` is a JPQL query

C. the class is not a repository, because it is not annotated with `@Repository`

D. this code will not compile

Question 29: Which of the following Spring annotations taken together are the equivalent of the `@SpringBootApplication` ?

A. `@Component`

B. `@Configuration`

C. `@ComponentScan`

D. `@EnableAutoConfiguration`

E. `@Controller`

Question 30: What is a principal?

A. an object storing the credentials for an user

B. the term that signified an user, device or system that could perform an action within the application

C. a term that signifies a secured resource

Question 31: What annotation is used to map HTTP Requests to handler methods?

A. `@HandlerMapping`

B. `@RequestMapping`

C. `@Mapping`

Question 32: Given the following controller declaration.

```
@Controller
@RequestMapping("/users")
public class UserController {

@RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public String show(@PathVariable Long id, Model model) {
        ...
        return "user/show";
        }

}
```

Is the definition of the `show` method correct?

A. yes

B. no

Question 33: Given the following controller declaration:

```
@RestController
@RequestMapping("/users")
public class UserController {

@RequestMapping(value = "/get", method = RequestMethod.GET, params="id")
        public User show(@RequestParam("id") Long id, Model model) {
        ...
        return user;
        }

}
```

Is the definition of the `show` method correct? Choose the most appropriate answer.

A. yes

B. no

C. yes, but it breaks the REST convention of using URI

Question 34: What does MVC stand for?

A. Model View Controller

B. Mapping View Controller

C. Module View Control

Question 35: Which class in the following list is the default view resolver in Spring:

A. `JspResourceViewResolver`

B. `ResourceViewResolver`

C. `InternalResourceViewResolver`

Question 36: To configure a Spring secure web application without using a `web.xml` configuration file you need to do the following: (choose all options that apply)

A. create a class that implements `WebSecurityConfigurer` or extends `WebSecurityConfigurerAdapter` and provide implementation for the appropriate methods

B. create a `security.xml` spring configuration file and import it with `@ImportResource`

C. annotate a configuration class with `@EnableWebSecurity`

Question 37: What is authorization?

A. is the process of verifying the validity of the principal's credentials

B. the process of making a decision if an authenticated user is allowed to perform a certain action within the application

C. the process of generating credentials for a user

Question 38: What is true about Spring Security?

A. authorization and authentication are provided by an external library

B. authorization and authentication cannot be decoupled

C. in a secured web environment the secured requests are handled by a chain of Spring-managed beans

Question 39: What is not true about Spring Boot?

A. Spring Boot simplifies configuration, but complicates deployment

B. Each release of Spring Boot provides a curated list of dependencies it supports

C. Spring Boot is a set of pre-configured set of frameworks/technologies designed to reduce boilerplate configuration

Question 40: Which of the following RestTemplates can be used to make a GET REST call to an url?

A. restTemplate.getForObject(...)

B. optionsForAllow(...)

C. getForEntity(...)

D. exchange(..., HttpMethod.GET,...)

Question 41: Which of the following HTTP message converters are supported by Spring MVC? (choose all that apply)

A. StringHttpMessageConverter

B. MappingJackson2HttpMessageConverter, but Jackson2 must be in the classpath

C. YamlMessageConverter

Question 42: What can be said about the @RestController annotation?

A. It is used to declare a controller providing REST Services.

B. Is annotated with @Controller and @ResponseBody

C. In controllers annotates with this annotation @RequestMapping methods assume @ResponseStatus semantics by default.

Question 43: What is true about `RestTemplate` ?

A. is a Spring specialized class used to consume REST services programatically

B. it is asynchronous

C. needs a `RestConsumer` instance as dependency

Question 44: Which of the following statements about declarative transactions is true?

A. `@Transaction` can be placed only at method level

B. `@Transactional` should be used on interfaces

C. `@Transactional` is taken into consideration only when configured so via `@EnableTransactionManagement` or `<tx:annotation-driven ../>`

Question 45: Spring Expression Language is part of the Core Container in Spring Framework. Is this true ?

A. yes

B. no

Question 46: Which of the following are advantages of the microservices architecture?

A. microservices based applications are highly scalable

B. improvement of fault isolation

C. transaction management is not needed

D. deploying microservices based application is painless

Question 47: Which of the following annotations is used to declare an instance of an Eureka server?

A. `@EnableNetflixEureka`

B. `@EurekaAutoConfiguration`

C. `@EnableEurekaServer`

Question 48: The default scope of a Spring bean is ...

A. prototype

B. session

C. multiple

D. singleton

E. volatile

Question 49: Which of the following annotations is used for service registration and discovery with a discovery server?

A. `@EnableDiscoveryClient`

B. `@EnableEurekaClient`

C. `@EnableSeviceRegistration`

Question 50: How can the lifecycle of a bean be customized? (choose all that apply)

A. implement `InitializingBean`

B. implement `disposableBean`

C. declare an initialization method by annotating it with `PostConstruct`

D. declare an initialization method by annotating it with `PreConstruct`

Answers

1. A,B
2. A,B,C
3. A
4. B,F
5. A,B
6. A,B,D
7. A,B,D
8. B
9. A,B
10. B
11. B
12. C
13. A,B
14. B
15. C
16. B
17. C
18. B
19. A
20. B
21. B
22. A
23. C
24. A, C
25. C
26. B,C
27. C
28. B
29. B,C,D
30. A
31. B

32. A
33. C
34. A
35. C
36. A,C
37. B
38. C
39. A
40. A,C,D
41. A,B
42. A,B,C
43. A
44. C
45. A
46. A,B
47. C
48. D
49. A
50. A,B,C

Footnote

1. Apache Tomcat official site: `http://tomcat.apache.org/`
2. Spring 4 Framework Reference: `http://docs.spring.io/spring/docs/4.2.3.RELEASE/spring-framework-htmlsingle`
3.