Spring Certified #3

A question lead guide to prepare Spring certification

Testing

When using @SpringBootTest with webEnvironment=RANDOM_PORT, how can you inject the port number into a field at runtime in a Spring Boot application? (select 2)

- ➔ **Use @Value("${local.server.port}") annotation on a field in the test class.**
- ➔ **Use @LocalServerPort annotation on a field in the test class.**
- ➔ **Use @Autowired annotation on a field in the test class.**
- ➔ **Use @ConfigurationProperties annotation on a field in the test class.**

answer

Correct answers:

- Use `@Value("${local.server.port}")` annotation on a field in the test class. This will inject the port number value into the annotated field.
- Use `@LocalServerPort` annotation on a field in the test class. This will inject the actual local port number used for the test server.

Incorrect answers:

1. Use `@Autowired` annotation on a field in the test class. This will not inject the port number value into the annotated field.
2. Use `@ConfigurationProperties` annotation on a field in the test class. This is not related to injecting the port number value into the annotated field.

Configuration

What is one way to modify the behavior of a Spring Boot application for a specific profile? (select 2)

- ➔ **Create a properties file named application-{profileName}.properties or application-{profileName}.yml, and specify the desired configuration properties for the given profile.**
- ➔ **Annotate a configuration class with @Profile("profileName") and define the beans or settings specific to the profile.**
- ➔ **Add a new endpoint to the application's REST API.**
- ➔ **Modify the main method of the application's entry point class.**

answer

Two correct answers:

1. **Create a properties file named application-{profileName}.properties or application-{profileName}.yml, and specify the desired configuration properties for the given profile.**
2. **Annotate a configuration class with @Profile("profileName") and define the beans or settings specific to the profile.**

Spring Profiles let you load config/beans only in certain environments.
Mark beans/config with `@Profile("prod")` to include them only when that profile is active.
Activate profiles via `spring.profiles.active` (properties/env var/CLI: `--spring.profiles.active=dev,hsqldb`).
Spring Boot adds profile-specific files: `application-{profile}.properties|yml`.
It always loads `application.properties`, then overrides with the active profile file(s).
Example: `application-dev` uses H2; `application-production` uses MySQL.
This cleanly separates per-env credentials, URLs, and tuning.
Since Boot 2.4, you cannot activate a profile from inside another profile file; set active profiles externally.
Therefore, “Create `application-{profile}.properties|yml` with the needed settings” is correct.
And “Annotate config with `@Profile("profileName")` to wire profile-specific beans” is also correct.

Testing

Which components are automatically configured when using the @WebMvcTest annotation in a Spring Boot application? (select 2)

- ➔ **Jackson ObjectMapper**
- ➔ **@Component**
- ➔ **Datasource configurations**
- ➔ **Controller**

answer

When working with JSON payloads, Spring Boot auto-configures a Jackson **`ObjectMapper`** bean. This bean is responsible for converting Java objects to and from JSON.

public @interface **WebMvcTest**

Annotation that can be used for a Spring MVC test that focuses **only** on Spring MVC components.

Using this annotation will disable full auto-configuration and instead apply only configuration relevant to MVC tests (i.e. `@Controller`, `@ControllerAdvice`, `@JsonComponent`, `Converter`/`GenericConverter`, `Filter`, `WebMvcConfigurer` and `HandlerMethodArgumentResolver` beans but not `@Component`, `@Service` or `@Repository` beans).

By default, tests annotated with `@WebMvcTest` will also auto-configure Spring Security and `MockMvc` (include support for HtmlUnit WebClient and Selenium WebDriver). For more fine-grained control of MockMVC the `@AutoConfigureMockMvc` annotation can be used.

Typically `@WebMvcTest` is used in combination with `@MockBean` or `@Import` to create any collaborators required by your `@Controller` beans.

If you are looking to load your full application configuration and use MockMVC, you should consider `@SpringBootTest` combined with `@AutoConfigureMockMvc` rather than this annotation.

When using JUnit 4, this annotation should be used in combination with `@RunWith(SpringRunner.class)`.
https://docs.spring.io/spring-boot/docs/2.5.2/api/org/springframework/boot/test/autoconfigure/web/servlet/WebMvcTest.html

https://bit.ly/2v7222