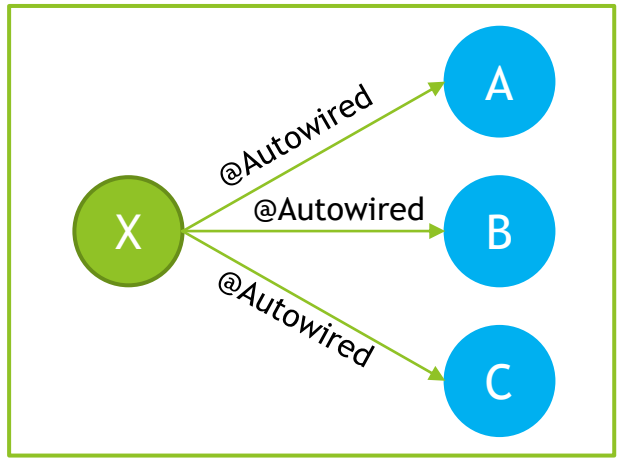
Spring Professional Exam Tutorial v5.0

Question 02

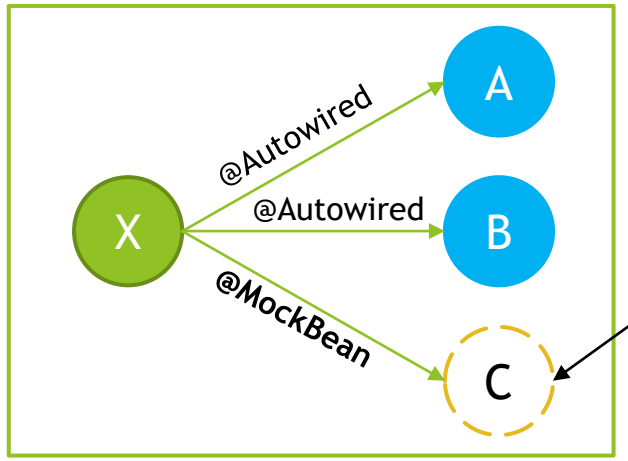
Question 02 - What type of tests typically use Spring?

Integration Tests are type of tests that typically use Spring. Reason for it is that on Integrate Test level we want to test multiple **components** that are **combined together**, and we want to check if those components provide requested functionalities when cooperating together under environment that should be close to production one, however with assumptions **that some of components might still be mocked**. When performing Integration Test we want to **initiate subset of system** and execute test against it. **IoC/DI Container is used** for this kind of testing, with some simplification upon deployment or container execution. **Dependencies are resolved and injected by Spring.**

Production – IoC/DI Container



Integration Testing – @RunWith(SpringRunner.class)



Question 02 - What type of tests typically use Spring?

Spring provides great support for integration testing, with main goals of the support being:

- Management of Spring IoC container and IoC container caching between tests
  - Spring will create and manage IoC container for tests
  - Context can be reused between tests
  - Main purpose of context being reused is to improve tests execution time
  - Execution times of integration tests might be long mainly because of Embedded Database, Hibernate and other components that are created once context is created
- Dependency Injection in tests
  - Allows for easy Spring Test definition with usage of `@RunWith(SpringRunner.class)`
  - `@ContextConfiguration` can be used in tests to configure context
  - Support for `@Autowired`, `@Inject` ...
  - Allows customization with `@TestExecutionListener`
- Transaction management appropriate to integration testing
  - Resolves issue with test affecting each other on data level by implementing proper transaction management
  - By default, all transactions are roll-back transactions
  - Gives ability to commit transactions if required

Question 02 - What type of tests typically use Spring?

Additionally Spring provides following tools to simplify Integration Testing:

- JDBC Testing Support
  - `JdbcTestUtils` provides JDBC related utility functions
  - `countRowsInTable`, `countRowsInTableWhere`, `deleteFromTables`, `deleteFromTableWhere`, ...
- Spring MVC Testing Support
  - Allow for easy setup with:
    - `@RunWith(SpringRunner.class)`
    - `@WebAppConfiguration`
    - `@ContextConfiguration`
  - Automatically creates `MockMvc`
- HtmlUnit Integration
  - Simplifies end-to-end testing for HTML views
- Client-Side REST Tests
  - Allows you to test Client code that interacts with mocked REST Service
  - Use `RestTemplate` with `MockRestServiceServer` to make assertions on mock

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @ContextConfiguration
  - Allows you to specify how to load and configure an ApplicationContext for integration tests
  - You can specify `@Configuration` classes that will be used during `ApplicationContext` loading
  - Optionally, you can specify XML configuration files locations, if you are using it instead of annotated `@Configuration` classes

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = ApplicationConfiguration.class)
public class ApplicationServiceIntegrationTest {

    ...

}
```

- @BootstrapWith
  - Allows for low-level control on how Context for Tests is created
  - To implement custom bootstrapped, it is best to extend `AbstractTestContextBootstrapper`
  - Used at class-level

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @DirtiesContext
  - Marks test as one that modifies state of context, and it means that context should be recreated prior next test execution because otherwise modified context state might affect test execution

```
@Test
@DirtiesContext
public void shouldBookAnyRoomForNewGuest() {
    ...
}
```

  - When used at class-level you can specify following modes:
    - BEFORE_CLASS
    - BEFORE_EACH_TEST_METHOD
    - AFTER_EACH_TEST_METHOD
    - AFTER_CLASS

```
@DirtiesContext(classMode = AFTER_EACH_TEST_METHOD)
```

  - When used at method-level you can specify following modes:
    - BEFORE_METHOD
    - AFTER_METHOD

```
@DirtiesContext(methodMode = AFTER_METHOD)
```

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @ActiveProfiles
  - class-level annotation that is used to declare which bean definition profiles should be active when loading an ApplicationContext

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = ApplicationConfiguration.class)
@ActiveProfiles({"test", "example-data"})
public class ApplicationServiceIntegrationTest {

    ...

}
```

- @TestPropertySource
  - class-level annotation that you can use to configure the locations of properties files and inlined properties

```
@TestPropertySource("/application-test.properties")
```

```
@TestPropertySource(properties = { "user = test-user", "group = test-group" })
```

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @WebAppConfiguration
  - class-level annotation that triggers creation of `MockServletContext`, which serves as the `ServletContext` for the test's `WebApplicationContext`
  - Indicates that `ApplicationContext` loaded for an integration test should be a `WebApplicationContext`

```
@RunWith(SpringRunner.class)
@ContextConfiguration
@WebAppConfiguration
public class ApplicationServiceIntegrationTest {

    ...

}
```

- @ContextHierarchy
  - Used when hierarchy of application contexts has to be used for integration test

```
@RunWith(SpringRunner.class)
@WebAppConfiguration
@ContextHierarchy({
        @ContextConfiguration(classes = AppConfig.class),
        @ContextConfiguration(classes = WebConfig.class)
})
public class WebIntegrationTests {
    ...
}
```

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @TestExecutionListeners
  - Allows registration of `TestExecutionListener` which allows for customization of test execution
  - Example of `TestExecutionListener` that is registered by default is `DirtiesContextTestExecutionListener`

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = ApplicationConfiguration.class)
@TestExecutionListeners({
        CustomTestExecutionListener1.class,
        CustomTestExecutionListener2.class
})
public class ApplicationServiceIntegrationTest {

    ...

}
```

- @Commit
  - class or method level annotation
  - indicates that after test execution, transaction should be committed

```
@Test
@Commit
public void shouldBookAnyRoomForNewGuest() {
    ...
}
```

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @Rollback
  - class or method level annotation that indicates that transaction should be rolled back after test execution
  - Even if `@Rollback` is not explicitly defined, all transactions under tests will be rolled backed by default

```
@Test
@Rollback
public void shouldBookAnyRoomForNewGuest() {
    ...
}
```

- @BeforeTransaction
  - Indicates method that should be executed before transaction is started
- @AfterTransaction
  - Indicates method that should be executed after transaction is started

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @Sql
  - Indicates SQL scripts that should be executed against database during integration test

```
@Test
@Sql({
        "/test-schema.sql",
        "/test-data.sql"
})
public void shouldBookAnyRoomForNewGuest() {
    ...
}
```

- @SqlConfig
  - Defines metadata used for SQL script parsing

```
@Test
@Sql(
        scripts = "/test-user-data.sql",
        config = @SqlConfig(commentPrefix = "`", separator = "@@")
)
public void shouldBookAnyRoomForNewGuest() {
    ...
}
```

Question 02 - What type of tests typically use Spring?

Spring annotations for Integration Testing

- @SqlGroup
  - Allows you to use multiple @Sql annotations

```
@Test
@SqlGroup({
        @Sql(scripts = "/test-schema.sql", config = @SqlConfig(separator = "@@")),
        @Sql("/test-data.sql")
})
public void shouldBookAnyRoomForNewGuest() {
    ...
}
```