Introduction to unit and integration testing in Spring Boot

Simeon Monov, Hristo Karaperev

Introduction

- Writing tests under Spring Boot:
  - Unit tests, that run in isolation
  - Integration tests bootstrapping Spring context before execution

Needed dependencies

- **spring-boot-starter-test** is the main dependency that contains all the required elements that are needed to run the tests
- In **Gradle**:

```
testImplementation 'org.springframework.boot:spring-boot-starter-test'
testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
```

- In **Maven**:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <version>3.2.0</version>
</dependency>
```

Unit tests with Mockito

- Tasty mocking framework for unit tests in Java
- Enable Mockito:

```
@ExtendWith(MockitoExtension.class)
public class TodoServiceTest {
```

- Main annotations, part of the Mockito framework: **@Mock, @InjectMocks, @Captor**
- **Mockito.when** and **Mockito.verify** methods are used to configure and verify behavior

@Mock annotation

- **@Mock** is used to create and inject mocked instances

```
@ExtendWith(MockitoExtension.class)
public class TodoServiceTest {
    @Mock
    TodoRepository todoRepository;

    @Mock
    TeamMemberRepository teamMemberRepository;
```

@InjectMocks annotation

- **@InjectMocks** is used to inject mock fields into the tested object automatically

```
@ExtendWith(MockitoExtension.class)
public class TodoServiceTest {
    @Mock
    TodoRepository todoRepository;

    @Mock
    TeamMemberRepository teamMemberRepository;

    @InjectMocks
    TodoService todoService;
```

@Captor Annotation

- **ArgumentCaptor** is used to capture arguments for mocked methods
- Captor's capture() method is used with verify() methods
- We can get the captured arguments with getValue() or getAllValues() methods

```java
@ExtendWith(MockitoExtension.class)
public class TodoServiceTest {
    @Mock
    TodoRepository todoRepository;

    @Mock
    TeamMemberRepository teamMemberRepository;

    @InjectMocks
    TodoService todoService;

    @Captor
    ArgumentCaptor<Todo> todoCaptor;
```

```java
verify(todoRepository).save(todoCaptor.capture());
Todo actualSaveTodo = todoCaptor.getValue();
```

Mockito.when methods

- Mockito.when methods are used to configure behavior

```
TeamMember teamMember = new TeamMember();
teamMember.setId(1L);
teamMember.setName("Pesho");
when(teamMemberRepository.findById(Long.valueOf(1))).thenReturn(Optional.of(teamMember));
```

```
when(teamMemberRepository.existsById(Mockito.anyLong())).thenReturn(true);
```

Mockito.verify methods

- Mockito.verify methods are used to check that certain behavior happened

```
verify(todoRepository, times(1)).save(Mockito.any(Todo.class));
```

```
verify(todoRepository).save(todoCaptor.capture());
```

```
verify(todoService, times(1)).createTodo(request);
```

AssertJ - rich assertions in Java tests.

- Assertions are written with: assertThat(referenceOrValue)
- Examples:

```java
@Test
void test() {
    String carName = "Audi";
    assertThat(carName).isEqualTo("Audi");

    List<String> cars = new ArrayList<String>();
    cars.add("Audi");
    cars.add("Marcedes");
    cars.add("Toyota");
    assertThat(cars).hasSize(3);

    assertThat(cars).isNotEmpty().contains("Toyota", "Audi");

    assertThat(cars).doesNotContain("BMW");

    Car car1 = new Car("Toyota", 2020);
    Car car2 = new Car("Toyota", 2020);

    // objects reference / equality comparison
    assertThat(car1).isNotEqualTo(car2);

    // objects recursive field by field comparison
    assertThat(car1).usingRecursiveComparison().isEqualTo(car2);
}
```

Integration testing with @SpringBootTest

- Testing the integration of different layers of the application
- The **@SpringBootTest** annotation is used to bootstrap the entire container
- We can use **@Autowire** to get the beans
- **@TestPropertySource** can be used to specify special test configuration
- **MockMvc** provides support for testing Spring MVC applications and **@AutoConfigureMockMvc** is used to enable and configure auto-configuration of MockMvc

@SpringBootTest example

```
@SpringBootTest
@AutoConfigureMockMvc
@TestPropertySource(locations = "classpath:application-integrationtest.properties")
class TodoApplicationTests {

    @Autowired
    TeamMemberService teamMemberService;

    @Autowired
    MockMvc mockMvc;

    @Autowired
    TeamMemberRepository teamRepo;

    @Test
    void teamMembersApiTest() throws Exception {
        createTeamMember("Bob");
        createTeamMember("Mary Smith");

        mockMvc.perform(get("/team-members").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
                .andExpectAll(jsonPath("$[0].name", is("Bob")), jsonPath("$[1].name", containsString("Mary")));
    }
```