Spring Boot with Unit Test

2020/02/25

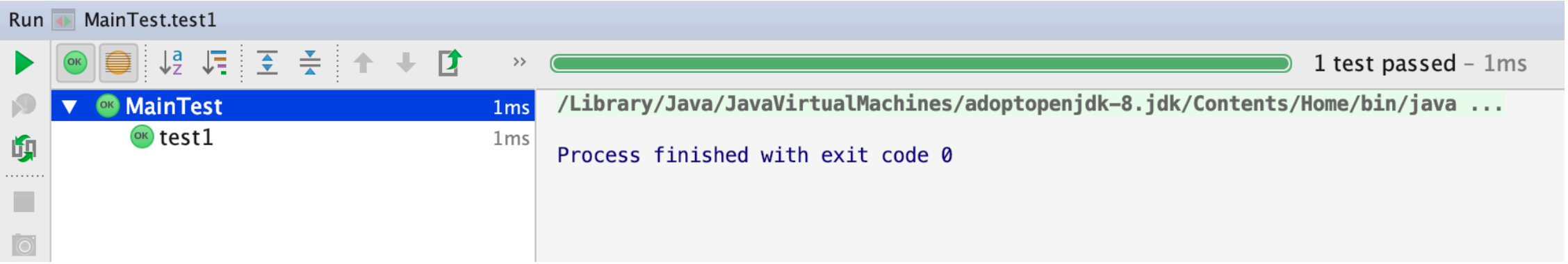
kucw

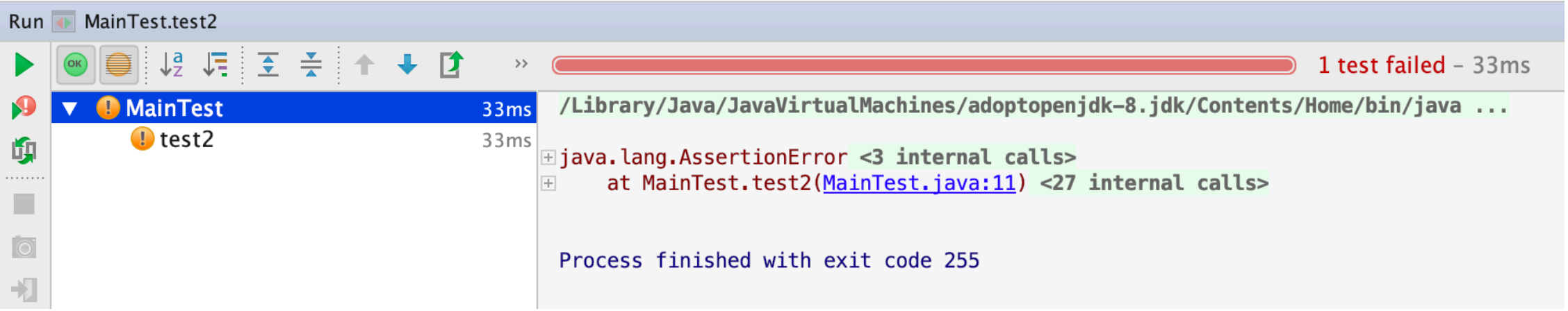
Outline

- What is JUnit?
- How to use JUnit?
  - JUnit annotation
  - Assert series method
- SpringBoot + JUnit
- SpringBoot + H2 database
- Spring test annotation
- Mockito
- MockMvc

What is JUnit?

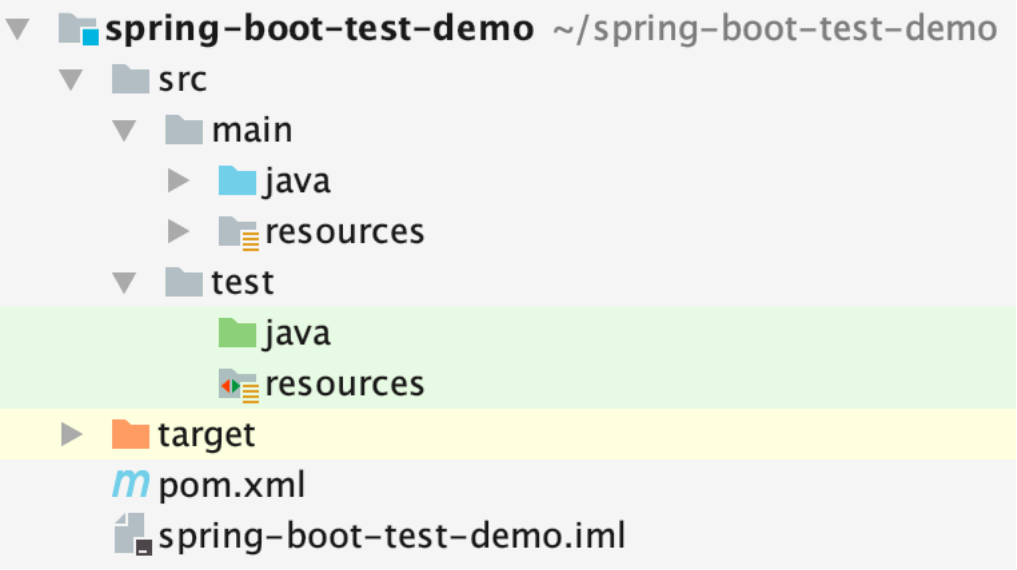
- JUnit is an open source Unit Testing Framework for Java.





How to use JUnit?

- Add a java class file under test directory and add @Test annotation on a method.




```
1   import org.junit.*;
2
3   public class MainTest {
4       @Test
5       public void test1() {
6           Assert.assertTrue( condition: true);
7       }
8
9       @Test
10      public void test2() {
11          Assert.assertTrue( condition: false);
12      }
13  }
```

JUnit annotation

- @BeforeClass
- @AfterClass
- @Before
- @After

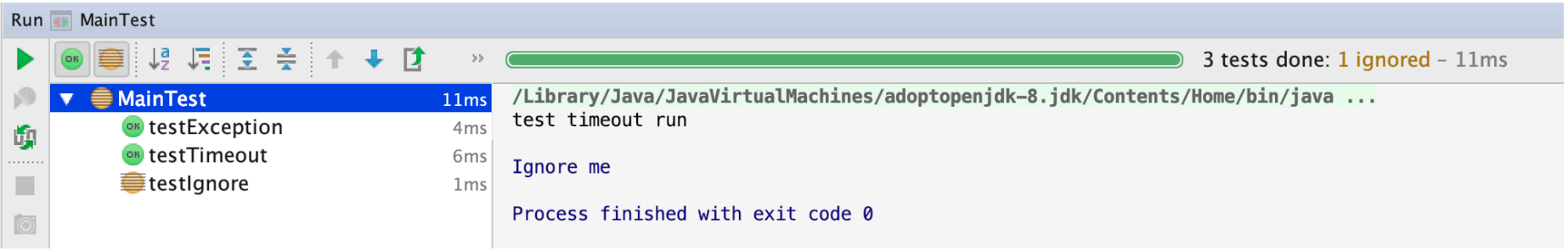
```java
public class MainTest {
    @BeforeClass
    public static void beforeClass() {
        System.out.println("This is BeforeClass");
    }

    @AfterClass
    public static void afterClass() {
        System.out.println("This is AfterClass");
    }

    @Before
    public void before() {
        System.out.println("This is Before");
    }

    @After
    public void after() {
        System.out.println("This is After");
    }

    @Test
    public void test1() {
        System.out.println("test1 run");
    }

    @Test
    public void test2() {
        System.out.println("test2 run");
    }
}
```

```
This is BeforeClass
This is Before
test1 run
This is After
This is Before
test2 run
This is After
This is AfterClass
```

JUnit annotation

- @Test with parameter
- @Ignore

```java
public class MainTest {
    @Test(expected = ArithmeticException.class)
    public void testException() {
        throw new ArithmeticException("test exception run");
    }

    @Test(timeout = 100)
    public void testTimeout() {
        System.out.println("test timeout run");
    }

    @Test
    @Ignore("Ignore me")
    public void testIgnore() {
        System.out.println("test ignore run");
    }
}
```



Assert series method

Assert series method	Result
Assert.assertTrue(A)	Claim A is true. If A isn’t true, then throw AssertError.
Assert.assertFalse(A)	Claim A is false.
Assert.assertEquals(A, B)	Claim A.equals(B) is true.
Assert.assertSame(A, B)	Claim A == B is true.
Assert.assertNotNull(A)	Claim A is not Null.

SpringBoot + JUnit

Include spring-boot-starter-test

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

Add @RunWith and @SpringBootTest annotation at each Test class

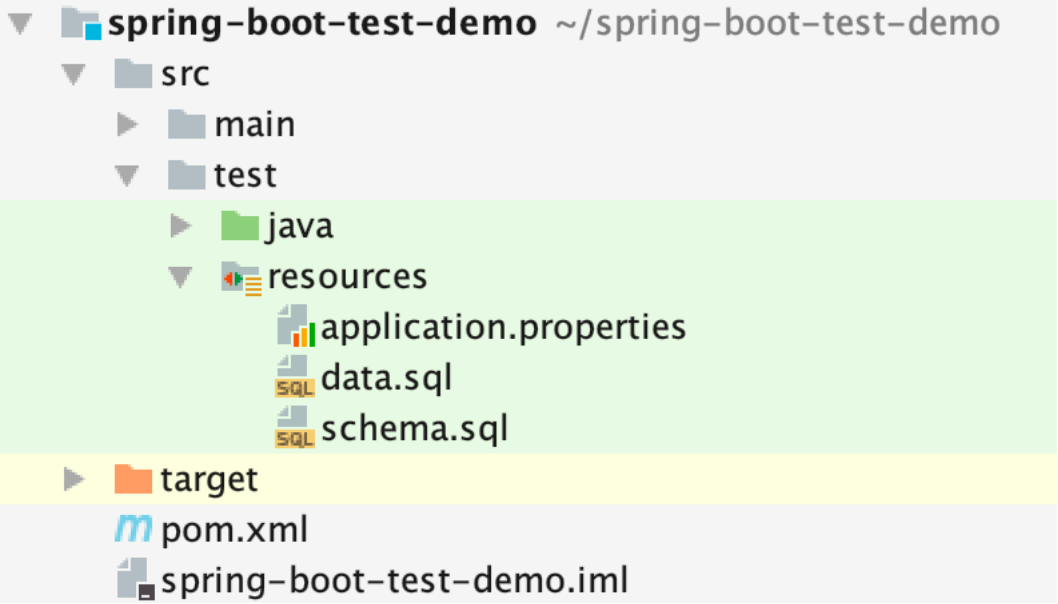
```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {

    @Autowired
    UserService userService;

    @Test
    public void testService() {
        // do some testing
    }
}
```

SpringBoot + H2 database

Include h2database

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
    <scope>test</scope>
</dependency>
```

Add application.properties, data.sql, schema.sql under test resources directory



SpringBoot + H2 database

- application.properties : setup h2 database for unit test environment
- schema.sql : initialize table schema in h2 database
- data.sql : initialize data in h2 database

application.properties

```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.jdbcUrl=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=sa
```

schema.sql

```sql
DROP TABLE IF EXISTS user;

CREATE TABLE user (
  id          INT          NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name        NVARCHAR(30) NOT NULL,
  update_time DATETIME     NOT NULL
);
```

data.sql

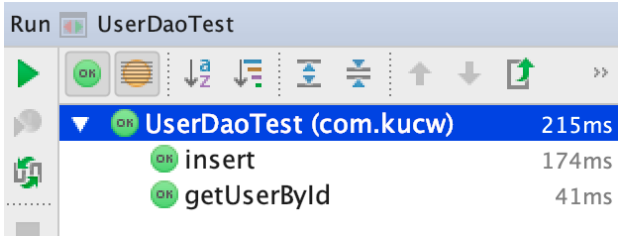
```sql
INSERT INTO user (name, update_time) VALUES ('John', current_date);
```

Spring test annotation

- @Transactional : rollback database change
- @DirtiesContext : restart Spring context

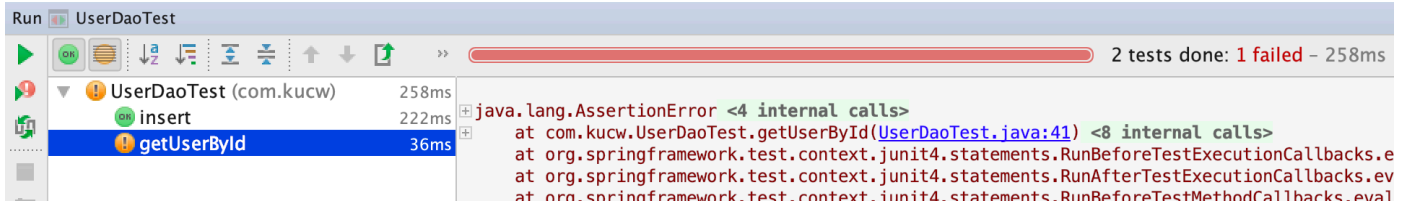
Without @Transactional

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserDaoTest {

    @Autowired
    private UserDao userDao;

    @Test
    public void insert() throws Exception {
        User user = new User();
        user.setName("test Boss");
        user.setUpdateTime(new Date());
        userDao.insertUser(user);
    }

    @Test
    public void getUserById() throws Exception {
        User user = userDao.getUserById( id: 1);
        Assert.assertNotNull(user);
        Assert.assertEquals(user.getName(),  actual: "test Boss");
    }
}
```



With @Transactional

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserDaoTest {

    @Autowired
    private UserDao userDao;

    @Test
    @Transactional
    public void insert() throws Exception {
        User user = new User();
        user.setName("test Boss");
        user.setUpdateTime(new Date());
        userDao.insertUser(user);
    }

    @Test
    public void getUserById() throws Exception {
        User user = userDao.getUserById( id: 1);
        Assert.assertNotNull(user);
        Assert.assertEquals(user.getName(),  actual: "test Boss");
    }
}
```

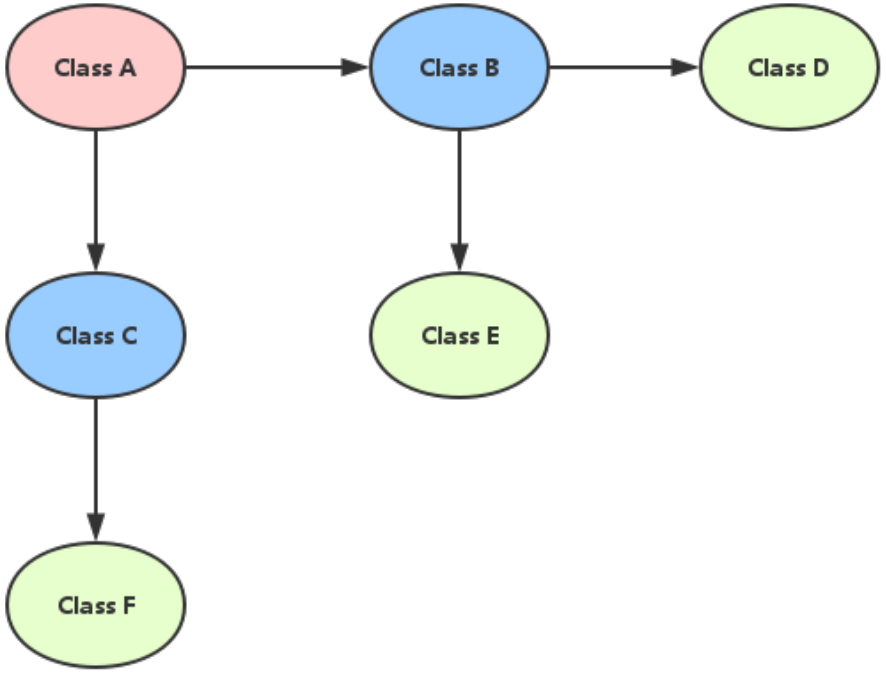


Mockito

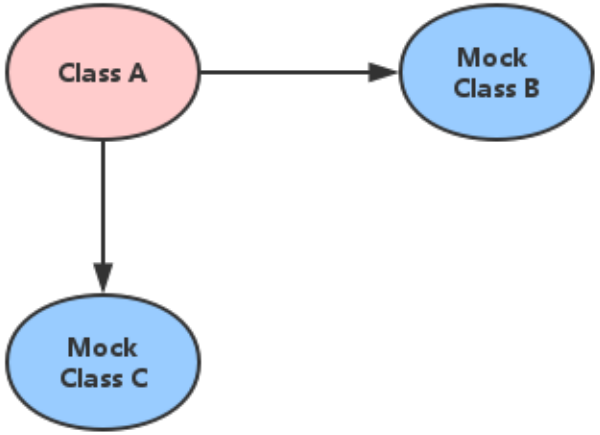
The framework's name and logo are a play on mojitos, a type of drink.

- Mockito is a mocking framework in Java.

Bean dependency



After using Mockito



@MockBean

Assume we have a UserService class

```java
@Component
public class UserService {

    @Autowired
    private UserDao userDao;

    public User getUserById(Integer id) {
        return userDao.getUserById(id);
    }
}
```

Use @MockBean to mock userDao bean

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {

    @Autowired
    private UserService userService;

    @MockBean
    private UserDao userDao;

    @Test
    public void getUserById() throws Exception {
        Mockito.when(userDao.getUserById(Mockito.anyInt()))
                .thenReturn(new User( id: 200,  name: "I'm mockito name", new Date()));

        User user = userService.getUserById( id: 1);

        Assert.assertNotNull(user);
        Assert.assertEquals(user.getId(), new Integer( value: 200));
        Assert.assertEquals(user.getName(),  actual: "I'm mockito name");
    }
}
```

@MockBean ≈ @Mock + @InjectMocks

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {

    @Autowired
    private UserService userService;

    @MockBean
    private UserDao userDao;

    @Test
    public void getUserById() throws Exception {
        Mockito.when(userDao.getUserById(Mockito.anyInt()))
                .thenReturn(new User( id: 200,  name: "I'm mock

        User user = userService.getUserById( id: 1);

        Assert.assertNotNull(user);
        Assert.assertEquals(user.getId(), new Integer( value: 
        Assert.assertEquals(user.getName(),  actual: "I'm mocki
    }
}
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {

    @InjectMocks
    private UserService userService;

    @Mock
    private UserDao userDao;

    @Test
    public void getUserById() throws Exception {
        Mockito.when(userDao.getUserById(Mockito.anyInt()))
                .thenReturn(new User( id: 200,  name: "I'm mock

        User user = userService.getUserById( id: 1);

        Assert.assertNotNull(user);
        Assert.assertEquals(user.getId(), new Integer( value: 
        Assert.assertEquals(user.getName(),  actual: "I'm mocki
    }
}
```

The difference between @MockBean and @Mock + @InjectMocks

- @MockBean
  - replace Spring bean
- @Mock + @InjectMocks
  - only replace target service's dependency bean

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {

    @Mock
    private UserDao userDao;

    @InjectMocks
    private UserService userService; // will replace it's userDao to mock userDao

    @Autowired
    private UserService2 userService2; // use Spring real userDao bean

    @Test
    public void test() {
        Mockito.when(userDao.getUserById(Mockito.anyInt()))
                .thenReturn(new User( id: 1,  name: "I'm mock", new Date()));

        User user1 = userService.getUserById( id: 1);
        Assert.assertEquals(user1.getName(),  actual: "I'm mock");

        User user2 = userService2.getUserById( id: 1);
        Assert.assertEquals(user2.getName(),  actual: "John");
    }
}
```

Mockito usage

- Mockito.when(object.methodName()).thenReturn(response)

```
Mockito.when(userService.getUserById( id: 3)).thenReturn(new User( id: 3,  name: "I'm no.3", new Date()));

User user = userService.getUserById( id: 3); // will return "I'm no.3" User
```

```
Mockito.when(userService.getUserById(Mockito.anyInt())).thenReturn(new User( id: 200,  name: "I'm mock", new Date()));

User user1 = userService.getUserById( id: 10); // will return "I'm mock" user
User user2 = userService.getUserById( id: 20); // will return "I'm mock" user as well
```

```
// since we didn't define mockBean userService, so user would be null
User user = userService.getUserById(100);
```

```
Mockito.when(userService.insertUser(Mockito.any(User.class))).thenReturn(100);

Integer i = userService.insertUser(new User()); // will return 100
```

Mockito usage

- Mockito.when(object.methodName()).thenThrow(exception)

```
Mockito.when(userService.getUserById( id: 9)).thenThrow(new RuntimeException("mock throw exception"));

User user = userService.getUserById( id: 9); // will throw a RuntimeException with message "mock throw exception"
```

- Mockito.doThrow(exception).when(object).methodName()

```
Mockito.doThrow(new RuntimeException("mock throw exception")).when(userService).print();
```

Mockito usage

- Mockito can log method called history

```
Mockito.when(userService.getUserById( id: 3)).thenReturn(new User( id: 3,  name: "I'm no.3", new Date()));

User user = userService.getUserById( id: 3);

// Check if certain method have been called 1 times with input 3
Mockito.verify(userService, Mockito.times( wantedNumberOfInvocations: 1)).getUserById(Mockito.eq( value: 3));
```

```
userService.getUserById( id: 3);
userService.getUserById( id: 5);
userService.insertUser(new User( id: 100,  name: "I'm 100", new Date()));

// Verify history method call order
InOrder inOrder = Mockito.inOrder(userService);
inOrder.verify(userService).getUserById( id: 3);
inOrder.verify(userService).getUserById( id: 5);
inOrder.verify(userService).insertUser(Mockito.any(User.class));
```

MockMvc

- MockMvc : Http request simulator
  - However, it doesn’t actually start serlvet controller.
  - So you can’t use MockMvc to test Jersey, just use it for SpringMVC.

```java
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testController() {
    }
}
```

MockMvc

```java
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetUser() throws Exception {
        RequestBuilder requestBuilder = MockMvcRequestBuilders
                .get( urlTemplate: "/user/get")          // Http get method
                .param("id", "1")                       // query parameter
                .accept(MediaType.APPLICATION_JSON);  // header

        mockMvc.perform(requestBuilder)                              // simulate http request
                .andExpect(MockMvcResultMatchers.status().isOk())   // check response http status code
                .andExpect(MockMvcResultMatchers.jsonPath( expression: "id").value( expectedValue: 1)); // validate response

    }
}
```

MockMvc

- andExpect()
  - quickly validate response
- andDo()
  - print or log something
- andReturn()
  - return MockMvc result, so that you can validate response detail on your own

You can randomly use all of them since MockMvc uses Builder design pattern of them.

MockMvc

```java
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testPostUser() throws Exception {
        RequestBuilder requestBuilder = MockMvcRequestBuilders
                .post( urlTemplate: "/user/insert/{name}",  ...uriVars: "John") // Http post request
                .contentType("application/x-www-form-urlencoded")
                .accept("application/json")
                .param("age", "18");

        MvcResult mvcResult = mockMvc.perform(requestBuilder)
                .andExpect(MockMvcResultMatchers.status().isOk()) // check http status
                .andExpect(MockMvcResultMatchers.content().contentType( contentType: "application/json")) // check header
                .andExpect(MockMvcResultMatchers.jsonPath( expression: "name").value( expectedValue: "John")) // check response body
                .andDo(MockMvcResultHandlers.print()) // print MockMvc result on console
                .andReturn();
    }
}
```

Thanks for listening!

- Demo code : https://github.com/kucw/demo/tree/master/spring-boot-test-demo
- Github : https://github.com/kucw
- Website : https://kucw.github.io/