16. Java_QA (Mocks, Fakes, Stubbs)

- Методика тестирования с использованием Mock, Stub объектов

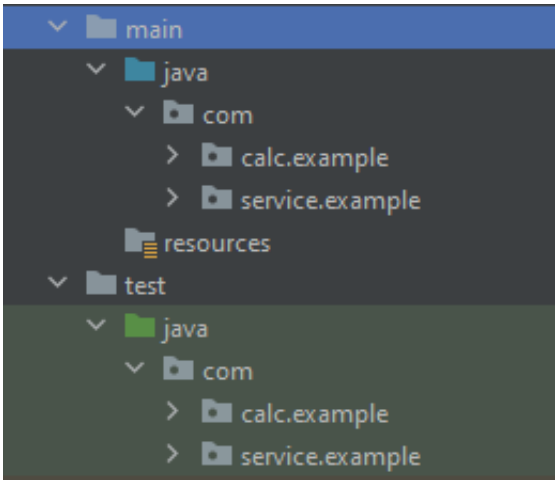
- Примеры заглушки интерейсов

- Методы when(), doReturn(), verify()

- Параметры вызова методов

- Обработка исключений

Структура проекта:



Сценарий:

calc.example – использование функционала mock объектов, на примере калькулятора

service.example – пример заглушки интерфейса по работе с БД

pom.xml: (добавление зависимостей)

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <dependency>
```

```
        <groupId>org.assertj</groupId>
        <artifactId>assertj-core</artifactId>
        <version>3.12.2</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>2.28.2</version>
        <scope>test</scope>
    </dependency>

</dependencies>
```

calc.example

Создание интерфейса iCalculator.java для калькулятора с простейшими арифметическими операциями и реализация этих методов в классе Calculator.java:

```
public interface Icalculator
{
   public double add      (double d1, double d2);
   public double subtract (double d1, double d2);
   public double multiply (double d1, double d2);
   public double divide   (double d1, double d2);

   public double double15();
}
```

```
public class Calculator
{
   Icalculator icalc;

   public Calculator(){
   }
   public Calculator(Icalculator icalc){
      this.icalc = icalc;
   }
   public double add(double d1, double d2) {
      return icalc.add(d1, d2);
   }
   public double subtract(double d1, double d2) {
      return icalc.subtract(d1, d2);
   }
   public double multiply(double d1, double d2) {
      return icalc.multiply(d1, d2);
   }
   public double divide(double d1, double d2) {
      return icalc.divide(d1, d2);
   }

   public double double15() {
      return 15.0;
   }
}
```

Блок тестирования:

@RunWith – определение лаунчера кейса

@Mock, @InjectMocks - определение и внедрение mock объектов

```
@RunWith(MockitoJUnitRunner.class)
public class Test_Mockito
{
   @Mock
   ICalculator mcalc;
// ICalculator mcalc = Mockito.mock(ICalculator.class);

   @InjectMocks
   Calculator calc = new Calculator(mcalc);
```

Определение поведения и проверка для операции сложения (when(), assertEquals(), verify(), doReturn(), thenReturn()):

```
@Test
public void testCalcAdd()
{
   when(calc.add(10.0, 20.0)).thenReturn(30.0);
   assertEquals(calc.add(10, 20), 30.0, 0);
   verify(mcalc).add(10.0, 20.0);
   doReturn(15.0).when(mcalc).add(10.0, 5.0);
   assertEquals(calc.add(10.0, 5.0), 15.0, 0);
   verify(mcalc).add(10.0, 5.0);
}
```

Операция вычитания, и проверка вызова методов (atLeast(), atLeastOnce(), atMost(), never()):

```
@Test
public void testCallMethod()
{
   when(mcalc.subtract(15.0, 25.0)).thenReturn(10.0);
   when(mcalc.subtract(35.0, 25.0)).thenReturn(-10.0);

   assertEquals (calc.subtract(15.0, 25.0), 10, 0);
   assertEquals (calc.subtract(15.0, 25.0), 10, 0);

   assertEquals (calc.subtract(35.0, 25.0),-10, 0);

   verify(mcalc, atLeastOnce()).subtract(35.0, 25.0);
   verify(mcalc, atLeast   (2)).subtract(15.0, 25.0);
```

Операция деления и пример обработки и определения поведения для исключений (doThrow(), thenThrow()):

```
@Test
public void testDivide()
{
   when(mcalc.divide(15.0, 3)).thenReturn(5.0);

   assertEquals(calc.divide(15.0, 3), 5.0, 0);
   verify(mcalc).divide(15.0, 3);

   RuntimeException exception = new RuntimeException ("Division by zero");
   doThrow(exception).when(mcalc).divide(15.0, 0);

   assertEquals(calc.divide(15.0, 0), 0.0, 0);
   verify(mcalc).divide(15.0, 0);
}
```

Работа с объектом Answer, метод обработки ответов: (getMock(), getArguments()):

```
private Answer<Double> answer = new Answer<Double>() {
    @Override
    public Double answer(InvocationOnMock invocation) throws Throwable {
        Object mock = invocation.getMock();
        System.out.println("mock object : " + mock.toString());

        Object[] args = invocation.getArguments();
        double d1 = (double) args[0];
        double d2 = (double) args[1];
        double d3 = d1 + d2;
        System.out.println("" + d1 + " + " + d2);
        return d3;
    }
};

@Test
public void testThenAnswer() {
    when(mcalc.add(11.0, 12.0)).thenAnswer(answer);
    assertEquals(calc.add(11.0, 12.0), 23.0, 0);
}
```

Использование объекта Spy:

```
@Test
public void testSpy()
{
   Calculator scalc = spy(new Calculator());
   when(scalc.double15()).thenReturn(23.0);

   double d15 = scalc.double15();
   assertEquals(23.0, d15, 0);
   verify(scalc).double15();

   assertEquals(23.0, scalc.double15(), 0);
   verify(scalc, atLeast(2)).double15();
}
```

Параметр Timeout, проверка вызова метода втечение определнного времени:

```
@Test
public void testTimeout() {
    when(mcalc.add(11.0, 12.0)).thenReturn(23.0);
    assertEquals(calc.add(11.0, 12.0), 23.0, 0);
    verify(mcalc, timeout(100)).add(11.0, 12.0);
}
```

Создание mocks при работе с любыми классами (на примере Iterator.class, Comparable.class):

```
@Test
public void testJavaClasses() {
    Iterator<String> mis = mock(Iterator.class);
    when(mis.next()).thenReturn("Привет").thenReturn("Mockito");
    String result = mis.next() + ", " + mis.next();
    assertEquals("Привет, Mockito", result);

    Comparable<String> mcs = mock(Comparable.class);
    when(mcs.compareTo("Mockito")).thenReturn(1);
    assertEquals(1, mcs.compareTo("Mockito"));

    Comparable<Integer> mci = mock(Comparable.class);
    when(mci.compareTo(anyInt())).thenReturn(1);
    assertEquals(1, mci.compareTo(5));
}
```

Class ServiceStub.java: (пример создания какого либо функционального интерфейса, и его упрощённой “заглушки”)

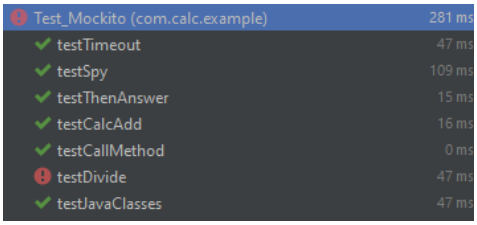
```
interface  Service{
    String doSomething();
}

public class ServiceStub implements Service{
    @Override
    public String doSomething() {
        return "myStubbedReturn";
    }
}

class ServiceRealUsing implements Service{

    @Override
    public String doSomething() {
        return "myRealUsingFunctional";
    }
}
```

Запуск сценариев:



service.example

Создание сценария по работе с клиентской БД и тестирование сервисной части

Модель User (данные пользователей):

```
public class User {

    private int id;
    private String username;
    private String role;

    public User(String username, String role) {
        this.username = username;
        this.role = role;
    }

    public User(String username) {
        this.username = username;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        User user = (User) o;
```

```
        if (id != user.id) return false;
        if (username != null ? !username.equals(user.username) :
user.username != null) return false;
        return role != null ? role.equals(user.role) : user.role == null;
    }

    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (username != null ? username.hashCode() : 0);
        result = 31 * result + (role != null ? role.hashCode() : 0);
        return result;
    }
}
```

Объект DAO (найти всех пользователей, и найти пользователя по имени):

```
public interface UserDao {

    User getUserByUsername(String username) throws Exception;

    List<User> findAllUsers();
}
```

```
public class UserDaoImpl implements UserDao {

    private List<User> users;

    public UserDaoImpl() {
        this.users = Arrays.asList(
                new User("olesya@gmail.com", "GUEST"),
                new User("kirill@gmail.com", "USER"),
                new User("remy@gmail.com", "ADMIN")
        );
    }

    @Override
    public User getUserByUsername(String username) throws Exception {
        return users.stream().filter(u -> u.getUsername().equals(username))
                .findAny().orElse(null);
    }

    @Override
    public List<User> findAllUsers() {
        return users;
    }
}
```

Сервисная часть (содержит метод проверки, что введённый пользователь есть в базе):

```
public class UserService {

    private UserDao dao;

    public UserService(UserDao dao) {
```

```
        this.dao = dao;
    }

    public boolean checkUserPresence(User user) throws Exception {
        User u = dao.getUserByUsername(user.getUsername());
        return u != null;
    }
}
```

Блок тестирования сервсисной части (UserService.class):

Определение mock-объектов:

```
public class UserServiceTest {

    @Mock
    private UserDao dao;

    private UserService userService;

    public UserServiceTest() {
        MockitoAnnotations.initMocks(this);
        this.userService = new UserService(dao);
    }
```

Позитивная проверка метода checkUserPresence() (определение поведения для dao, и последующая передача результата в userService):

```
@Test
public void checkUserPresence_Should_Return_True() throws Exception {
    given(dao.getUserByUsername("olga@gmail.com")).willReturn(
            new User("olga@gmail.com"));

    boolean userExists = userService.checkUserPresence(
            new User("olga@gmail.com"));

    assertThat(userExists).isTrue();

    verify(dao).getUserByUsername("olga@gmail.com");
}
```

Негативная проверка метода checkUserPresence():

```
@Test
public void checkUserPresence_Should_Return_False() throws Exception {

    given(dao.getUserByUsername("olga@gmail.com")).willReturn(
            null);

    boolean userExists = userService.checkUserPresence(
            new User("olga@gmail.com"));

    assertThat(userExists).isFalse();
}
```

Ожидание исключения:

```
@Test(expected = Exception.class)
public void checkUserPresence_Should_Throw_Exception() throws Exception {

    given(dao.getUserByUsername(anyString())).willThrow(Exception.class);
    userService.checkUserPresence(
            new User("olga@gmail.com"));
}
```

Объект Captor, проверка аргумента метода:

```
@Test
public void testCaptor() throws Exception {
    given(dao.getUserByUsername(anyString())).willReturn(
            new User("olga@gmail.com"));

    boolean b = userService.checkUserPresence(new User("olga@gmail.com"));

    ArgumentCaptor<String> captor = ArgumentCaptor.forClass(String.class);

    verify(dao).getUserByUsername(captor.capture());
    String argument = captor.getValue();

    assertThat(argument).isEqualTo("olga@gmail.com");
}
```

Запуск сценариев: