JUnit

Unit Testing with JUnit →

1. Step-1 → Why is test unit testing important?
2. Step-2 → Setting up your first JUnit
3. Step-3 → First successful JUnit. Green Bar & assertEqu
4. Step-4 → Refracting Your 1st JUnit test
5. Step-5 → 2nd JUnit example assertTrue & assertFalse.
6. Step-6 → @Before @After
7. Step-7 → @BeforeClass @AfterClass
8. Step-8 → Comparing Arrays in JUnit tests.
9. Step-9 → Testing Exceptions in JUnit tests
10. Step-10 → Testing performance in JUnit tests.
11. Step-11 → Parameterized tests,
12. Step-12 → Organize JUnits into suites

Haq, ek behtar zindagi ka.

5-3-22
Tuesday, December 05

* Mockito → It is an open source testing framework. It allows the creation of test double objects in automated unit test for purpose of TDD or BDD.

① Step-1 → Setting up a Maven Project.
- Setting up Eclipse project, JUNIT and Mockito Framework.

② Step-2 → Setting up SUT (System Under Test)
- Start creating an example to start understanding why we need mocks.
- We want to interact with a Todo Management app.
- we want to provide a filtering around spring ~~Het~~ related Todo's.

// TodoService (Dependency)
// TodoBussinessImpl (SUT)

③ Step-3 → Stubbing Example - with disadvantages of stubbing

Stub ⇒ It is a sample implementation of this particular TodoService

Disadvantage ⇒ ① Dynamic Condition
② Service Definition

④ Step-4 → Your first Mockito code
- Lets use Mockito to Mock TodoService.

Mocking ⇒ It is creating objects that simulate the behaviour of real objects. Unlike stubs, Mocks can be dynamically created from code - at runtime.
Mocks offer more functionality than stubbing.
You can verify method calls and a lot of other things.

Haq, ek behtar zindagi ka.

Wednesday, December 06

(5) Step-5 → Stubbing variations with Mockito - Argument Matchers & more

- A few Mockito examples Mocking list class.
- Multiple return values
- Introduction to Argument Matchers
- Stub Method
- Throwing exception

(6) Step-6 → BDD style - Given When Then (Behaviour Driven Development)

- BDD is a software development process that originally emerged from Test Driven Development (TDD).
- It is using examples at Multiple levels to create a shared understanding and surface uncertainty to deliver s/w that Matter.
- It uses examples to illustrate the behaviour of the system that are written in a readable and understandable language for everyone involved in the development
- BDD Mockito Syntax.

(7) Step-7 → Verify calls on mocks.

- Verify how many times a Method is called.
- We will add deleteTodo Method to the TodoService.

```
verify(todoService).deleteTodo("Learn to Dance");
verify(todoService, Mockito.never()).deleteTodo("Learn Spring MVC");
```

Haq, ek behtar zindagi ka.

2022/03/13 12:14

(8) Step-8 → Capturing arguments passed to a Mock

- How to capture an argument which is passed to a Mock?
  - // Declare Argument Captor
  - // Define Argument Captor on specific method call.
  - // Capture the argument

(9) Step-9 → Hamcrest Matchers

- It is a framework for writing matcher objects allowing 'match' rules to be defined declratively. There are a no. of situations where matchers are invaluable, such as UI validation or data filtering.
- Setup static import for -

```
- import static.org.hamcrest.CoreMatchers.hasItems;
```

(10) Step-10 → Mockito Annotations

- @Mock
- @InjectMocks
- @RunWith(MockitoJUnitRunner.class)
- @Captor

(11) Step-11 → Mockito JUnit Rule

- Using MockitoJUnit.rule instead of @RunWith(MockitoJUnitRunner.class)

```
@Rule
public MockitoRule mockitoRule = MockitoJUnit.rule();
```

(12) Step-12 → Real world Mockito example with Spring

- Not very different from previous example.

(13) Step-13 → Mockito Spy

- Understand what a spy does
- Creating a spy with mockito
- Overriding specific methods in a spy.
- Mockito provides option to create spy on real objects. When spy is called, then actual method of real object is called.
- A spy is like a partial mock, which will track the interactions with the object like a mock. Additionally, it allows us to call all the normal methods of the object.

(14) Step-14 → Theory: Why does Mockito not allow stubbing final and private methods?

- It requires hacking of classloaders that is never bullet proof and it changes the API.
- It is very easy to work around.
- It requires the team to spend time implementing and maintaining it.
- Mocking private methods is a hint that there is something wrong with Object Oriented understanding.

ng, ek behtar zindagi ka.

(15) Step-15 → Setting up powerMock and systemUnderTest and Mocking Static Method.

- Add dependency on PowerMock.
- Using PowerMock and Mockito to Mock a static Method.
- PowerMockitoMocking Static Method Test
- @RunWith(PowerMockRunner.class)
  @PrepareForTest({UtilityClass.class})

  PowerMockito.mockStatic(UtilityClass.class)
  when(UtilityClass.staticMethod(anyLong())).then Return(150);
  PowerMockito.verifyStatic();
  UtilityClass.staticMethod(1+2+3);

(16) Step-16 → Invoking Private Methods

- Using PowerMock and Mockito to invoke a private method.
- PowerMockitoTestingPrivateMethod Test
- long value = (Long) Whitebox.invokeMethod(systemUnderTest, "privateMethodUnderTest");

(17) Step-17 → Mocking a constructor

Sunday, December 10

- Using PowerMock and Mockito to Mock a constructor.
- PowerMockitoMockingConstructorTest.java -- PowerMockito.when New(ArrayList.class).withAnyArguments().thenReturn (mockList);

Haq ek behtar zin

Monday, December 11

(18) Step-18 → Writing Good Unit Tests

- Good Unit Tests -- Basics --- Readable --- Fails only when there are real logic failures.
- Using tests are FIRST : Fast, Isolated, Repeatable, Self-verifying and Timely.