Transcript

Some parts of session are dynamic and hands-on in nature,
so transcript availability is not feasible for those sections of session

Course Overview

SLIDE - 1

Hello everyone. My name is Kuldeep Singh. Welcome to the course of, JUnit 5 Fundamentals. I have been working with Java for more than 4 years. I love the language and sharing all I know about it.

SLIDE - 2

An important part of developing high-quality software is testing and as a developer, you are in charge of unit testing your code, and in Java,

SLIDE - 3

JUnit is the most popular framework for this task. This course is designed for developers who are new to unit testing and want to learn JUnit 5.

SLIDE - 4

Some of the major topics that we will cover include the new architecture of JUnit 5 and the problem it solves, how to use JUnit to create a static and dynamic test and how to integrate JUnit with Maven. By the end of this course, you'll understand how to use JUnit 5 to write unit tests.

SLIDE - 5

Before beginning the course, you should have a basic knowledge of Java 8, streams, lambda expressions, and default methods in interfaces. I hope you'll join me on this journey to learn JUnit 5.

At any point of time during the session if you are unable to hear or see presentation or having questions please let me know.

Introducing JUnit 5

SLIDE - 6

If I'd ask you what are the most important or popular Java frameworks, what would you answer? Probably, you would mention Spring and Hibernate. But would you mention JUnit too? If you don't think of JUnit as one of the most important or popular Java frameworks,

SLIDE - 7

let me tell you that JUnit is the standard tool for unit testing in Java, and it has played a big role in the adoption of agile methodologies where testing is an important part.

SLIDE - 8

In the words of Martin Fowler, "Never in the field of software development was so much owed by so many to so few lines of code." And that's because JUnit is simple, simple to use and simple to learn.

SLIDE - 9

Now what is JUnit,

SLIDE - 10

It’s a tool. And that's what I want you to remember during this course. JUnit is just a tool.

SLIDE - 11

Testing is all about getting feedback, feedback that tells you if your code is working as expected or not. Fixing a bug is usually something quick. But finding the bug, checking if it was really corrected, and that it will not appear again or that it didn't cause other bugs, can take a lot of time.

SLIDE - 12

For that reason, having tests that not only give us feedback, but also that are easy to understand, easy to read, and easy to Modify is very important. Knowing how to make dynamic tests or how to use some cool assertion methods doesn't mean you have to use them. Maybe the simplest solution would work just fine. And all things being equal, you should favor what is easier to understand and read.

Types of Tests

SLIDE - 13

There are many ways in which either developers or a QA department can test an application.

SLIDE – 14

For example, according to the Knowledge of the System, we can have black box testing. That is testing without any knowledge of the internal structure of a component or a system or white box testing, testing that is based on the internal structure of a component or a system.

SLIDE - 15

We can have other types of tests depending on what you are testing, the degree of isolation of the components you are testing, and even when in the development cycle you are testing.

SLIDE - 16

In this course, we are going to focus on unit testing.

SLIDE - 17

Unit tests are tests written by programmers. The classical definition of unit tests is that they test a piece of code by invoking it and checking the correctness of some assumptions. A unit test can test just a method,

SLIDE - 18

for example calculateCommission, or the whole class involving more than one method.

SLIDE - 19

A good unit test should focus on productivity. It should be automated, easy to run, ideally by executing just a command or by clicking a button. It should be repeatable. Everyone running the test should be able to repeat it without much problem. It

should be fast, not taking a long time to run, because fast feedback is important. But why are these points important? Well, fast, repeatable, automated tests can be run as often as possible without hurting productivity. By doing so, we'll receive feedback that allows us to detect and correct the problem as early as possible.

SLIDE - 20

If there's a dependency on a database or another component, for example a WebService, the test might not be as fast because almost always those dependencies are slower than just running a class. And with many tests, execution time adds up. Besides, if the test depends on an external component to be up, it might not be as automated or repeatable as we would want, and that's the difference between unit and integration tests.

SLIDE - 21

In integration tests, we have external components because the purpose is to test how they work together.

SLIDE - 22

If we want to unit test this piece of code, we have to replace the external component with either a fake dummy object or a mock to keep the component under test isolated and everything under our control. So the concept of unit test is simple. But why is having a good suite of tests important?

Why Unit Tests Are Important

SLIDE - 23

Let's talk about a few of the benefits of unit tests. They reduce debugging time. A good suite of tests and some good practices can reduce bugs and debug time. Unit tests also work as low-level documentation. They are written in code. Instead of reading a manual, you can read the code of the test to see how a library works and even copy/paste snippets of that code and modify them to fit your needs. And tests can help to improve the design of your application. Think about this. Tests are another client of your application. A good design makes your code accessible to those clients to your test. So tests can have a big influence on the design of your application to make it testable, especially when you write tests first in a practice called test-driven development. Maybe you're thinking all right, this sounds very good.

SLIDE - 24

But in the real world, it would be very hard and time-consuming to create a test suite that I can trust.

SLIDE - 25

Well, yes, it's hard, but not impossible. And at the end, you'll save time by reducing debugging time. And if done correctly, you can also get something that is worth more than all these things put together.

SLIDE - 26

Think about this. What if we could have tests that were as easy to run as the click of a button? What if the test of the whole application could run in a matter of minutes? What if those tests were so complete that almost no bugs could escape from them? What if those tests were always up to date with the code of the application? Well, something like that

SLIDE - 27

will give you courage, courage to change the code to make it better. Sometimes, we are afraid to make changes because if something breaks, it will be our fault, and we'll have to fix it. But over time, we'll have to make changes or add new features anyway. And eventually, the quality of the code will degrade.

SLIDE - 28

What do you think is better? An ugly and terrible designed system, but with a good suite of tests or a system perfectly designed, but with a horrible suite of tests or no tests at all. Over time, you could improve the ugly system because you won't be afraid to change it. But the perfectly designed system, it will degrade and become an ugly system, all because you didn't cover your back with a suite of tests of enough quality to give you the trust and courage to eliminate the fear of change.

Ok, so now, when all being said and heard, let's get started.

Writing JUnit5 Tests

SLIDE – 30 *Course Scenario*

To learn JUnit, this course will use the following scenario. Imagine we are working for the company Wired Brain Coffee. This company owns a chain of successful coffee shops all over the country. We want to grow and stay competitive, so we are implementing a loyalty program for our customers. In this program, every order generates points, and you can exchange points for rewards. In particular, the system can give three types of rewards. A conversion reward. In this case, the customers can convert their points into money. A discount reward. In this case, the system gives a discount for an amount of points. And a gift reward. In this case, the system gives a product of the customer's order as a gift for an amount of points. So all the examples and demos of this course will be related to testing the implementation of these types of rewards.

Discuss Test Set 1

Key Highlights from Test Set 1

- Tests should not depend on the order they are executed. But if you want to control the order in which test methods are executed, you can annotate your Test class with TestMethodOrder and specify a method order implementation.
- Do you see something in common? These methods have three parts or phases.
- In the first one, you set up the objects you need for the test. In the second one, you perform an action, and, in the third phase, to check if the result was the one you expected. So these methods have three phases.

SLIDE - 31

However, in general terms, every test has four phases that are executed in sequence. The first one is arrange where you set up the state that will allow you to perform the test. This state is also called the test fixture. In the act phase, you call the function or perform the action that you are testing and, in the third phase, to check whether you got the expected outcome or not. Finally, in the annihilation phase, you put the system back into the state in which you found it. This phase is not often explicit. For example, if we just create an object and set some of its properties, at the end of the test, Java's garbage collection will take care of that object. For that reason, this phase is often neglected. And you may hear many people talking about only the first three phases, but that final phase can be important when working with resources that must be closed, for example a network connection or file handle to close.

SLIDE - 33

Transient fresh fixture, is about setting up a brand new fixture every time you run a test. On the other hand, we have persistent fixtures that survive from one test to the next. So we have a persistent fresh fixture, a fixture that is used for all tests, but is initialized BeforeEach test runs and a persistent shared fixture, which allows some state to accumulate from test to test.

SLIDE – 34

Discuss Test Set 2