**Junit 5 Testing**

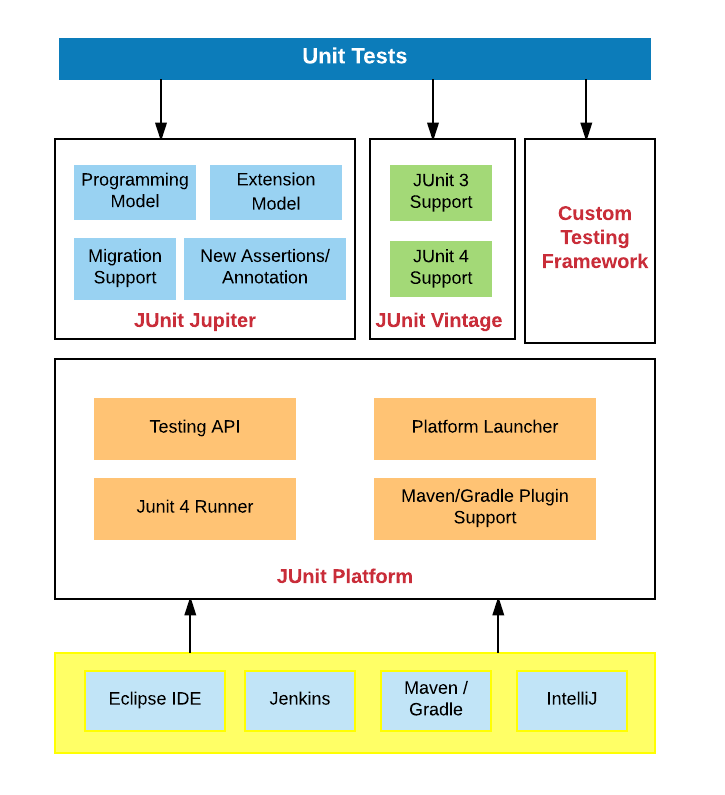
Testing in Software Development refers to something which is done post/after the development, and which is done by a separate dedicated tester team/QA Engineers/Tester in the industry.

* **Unit Testing:**
  + During the development of the application, there are so many classes and in those classes there will be so many methods implemented. One Unit refers to one method of those classes.
  + In method, we implement some behavior/operation to perform that method consider as an one unit of an application. In small applications, it could be one method considered as one unit and in the bigger application, there could be group of methods or classes considered as an one Unit of an application.
  + Dividing the application into the Unit based on the behavior or features trying to develop.
  + Generally, we refer Unit means one method, in which some behavior will be there, some functionality will be there and some task/operation to perform written. And testing this one method is considered as a Unit Testing.
  + In the Unit testing, first we write a Test, and based on the test we develop specific Unit then will run the test to validate is it passing or not. If it pass successfully then we have developed Unit perfectly and if it does not pass, we need to make modifications into the method to pass the test successfully.
  + While we write an Unit test, it has to work during that implementation time as well, also in future when new features will add, at that particular time also it has to work.
* **Pre-requisites for Testing:** Java(from beginner to intermediate)
* **What will you get:** Junit-5 Framework (Theory + Practical)
* **Outcome**: Junit5 or Junit Jupiter Testing Framework Expert for Unit Testing
* **Normal Testing vs Unit Testing:**
  + Testing in Software Development, it refer to something which is done post/after the development, and which is done by a separate dedicated tester team/QA Engineers/Tester in the industry.
  + Unit testing also tests the application and testing the functionality of the application which is done by Software Developer itself. The one who write or develop the code, he will only perform the Unit testing.
* **Flow of completing the development of an application:**
  + **Follow SDLC (Software Development Lifecycle):** 
    - **Designing:** Before we start the development or implementation, first we design the application or make blueprint of it.
    - **Development:** Once designing is done, then in Development stage, the actual code writing and implementation will done in Development stage.
    - **Testing:** Once implementation has been done, then will perform Testing of an application. In this stage, will validate the application whether it is working fine and producing expected outputs or not. This type of testing will done post the development.

**(This testing stage is different from the Unit Testing. In this stage, testing will done by Testing team/QA Team)**

**(While Unit test done during we write a code and it is done by Software Developer Team/Software Developer who are responsible to develop the application)**

* **Performing Unit test without having framework:**
  + If we want to perform a Unit test without having framework, we need to write individual test cases in the main method of the application and need to write code logic for all individual scenarios.
* **Steps need to follow in the Unit Testing:**
  + **Prepare:** 
    - First we need to set up a test environment that will includes importing Junit, Creating test classes and writing test methods.
    - Set up environment & test class.
    - Add annotations @Test, @BeforeEach, @AfterEach, @BeforeAll, @AfterAll, @Disabled
  + **Provide Testing Input:**
    - Pass inputs to the method you want to test.
    - Calling method with predefined input parameters.
    - Simulating user input, API parameters, or mock data.
  + **Run the Test:**
    - **To execute the test methods using JUnit runner.**
    - **Annotating test methods with @Test.**
    - Execute the test case using the test runner (either through your IDE or Maven/Gradle).
    - JUnit automatically runs the methods marked with @Test.
    - Using build tools (e.g., Jenkins, GitHub Actions) for CI.
    - Use @Test annotation, run via IDE or CLI
  + **Provide Expected Output:**
    - Define what correct output should be.
    - Define what the correct result should be for the given input.
    - Use hardcoded, clear values as expectations
  + **Perform Assertion / Verify the Result:**
    - Compare actual vs. expected results.
    - To compare actual output against expected and determine pass/failure.
    - Compare the actual output from the test to the expected output using assertion methods.
    - There are lot of APIs which will perform the assertion with Junit-5.
    - Use ***assertEquals***, ***assertTrue***, ***assertThrows***, include helpful messages
  + **Report Test Result:**
    - Output pass/fail for feedback.
    - JUnit will report whether the test passed or failed.
    - Integrate with CI/CD, generate readable reports, use test coverage tools
    - Generating reports with tools like **Surefire**, **JUnit HTML Reports**, or **Allure**.
  + In the place of manually testing unit, while we use Junit-5 testing framework, we just have to do Prepare step, have to provide Input to the test method, and just have to provide an Expected output of the method. After that Junit will take care of running the test cases, perform the assertion and verify that is that actual test matching with actual expected results and Junit will use own way to alert the developers whether test case is pass or fail.
* **Junit 5 Architecture: (Junit 5 is not Junit 4 + new features)**
  + **Junit Platform:**
    - It is core of component of Junit 5 architecture.
    - It is a foundation for running test engines (e.g., Jupiter or third-party engines).
    - Used by IDEs, build tools (Maven/Gradle), and CI tools.
    - It provides an environment to run tests which consist of test runners.
    - Developers do not directly works with platform even though it is core of Junit 5 architecture.
    - One of the key components of Junit Platform is “***Launcher API***” that allows test frameworks and IDEs to launch and execute tests.
    - Platform is responsible for run your test cases.
  + **JUnit Jupiter:**
    - JUnit Jupiter API provides the API for writing tests and test cycle control and the engine for running those tests.
    - JUnit Jupiter API contains all the methods to perform an Assertion / verify test results, all the annotations, which we will going to use in order to write a test and platform will be able to run the test which we have written.
  + **JUnit Vintage:**
    - If we already written Test cases in JUnit 4 and now want to migrate to JUnit 5, JUnit 5 does not provide backward compatibility as 5 is the whole new thing.
    - So, for backward compatibility, JUnit5 provides JUnit Vintage API which we can use to run existing test cases which we have written in old JUnit (3 or 4) and want to run in Junit 5.
    - JUnit 5 does not have direct backward compatibility, but we need to use JUnit Vintage API for it.
    - Run existing JUnit 3/4 test code in the JUnit 5 ecosystem.
    - Migrate to JUnit 5 incrementally, not all at once.
    - Use modern test tools (IDE, CI/CD, reporting) with older tests.
  + **Extension:**
    - Extension is the enhance the capability of JUnit 5 means we can have our own API and we make platform works for it without having to use Jupiter.
    - Usually we mainly use Jupiter libraries / API whenever we are working with JUnit 5 and all the annotations and methods for assertions which are associated in order to write a test all of them are part of Jupiter. If we want to use those libraries and method, we need to add the Jupiter dependencies into our project using Maven and we can use those libraries, annotations and methods within that Jupiter API.
    - But if we want to use third party API and don’t want to use Jupiter API then we need to use Extension.
    - When we want to run a Test, we can use any of the IDEs such as Eclipse, VS Code, IntelliJ etc.., all are integrated with JUnit 5 framework and we can run a test by just right-clicking on the test case.



* **Implementation of Unit Testing code for application:**
  + To start with writing a code for unit testing, we will need an IDE. We can use any IDEs such as Eclipse, VS Code, IntelliJ, etc. but make sure it can create a Maven project.
  + **Why Maven Project? Maven is a build tool:**
    - Because Maven provide a specific structure in which we can easily write test by using JUnit.
    - To write JUnit test, it expected us to follow specific structure and Maven will provide that structure automatically and also in the real-world industry we use a build tool Maven.
  + **Steps to create a Maven Project for JUnit test in Eclipse:**
    - **Step 1: File > New > Maven Project**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Or File > New > Other and In the Dialog Box Select Maven > Maven Project**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* + - **Step 2: Select checkbox “Create a simple project (skip archetype selection)” and “Use default Workspace location” > click on “Next”**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* + - **Step 3: Provide the details below and click on “Finish” button:**
      * **Group Id:** It is whatever we give convention for the package / folder name. We create a packages and inside that package we create multiple class.
        + **Ex.: com.JavaJUnitTest.****Learning**
        + While we pass package name with “.”, each “.” Consider as go inside package and create another package in it. For above example, we have written com.JavaJUnitTest.Learning, so it will create “com” package/folder inside that will create “JavaJUnitTest” package and inside that it will create “Learning” package
        + **com > JavaJUnitTest > Learning**
      * **Artifact Id:** It is a name of the project. Whatever the name we will provide, it will be the project build output.
        + **Ex.: employment-management**
        + For the Artifact ID: employment-management, it will create a JAR or WAR file with the name employment-management.jar file.
      * **Version:** It is a version of the project. We can keep its default value.
        + **Ex.: 0.0.1 SNAPSHOT**
      * **Packaging:** Select JAR, WAR or POM based upon our application type.
        + **jar:** Default. Creates a Java Archive file. Used for libraries and standalone Java apps.
        + **war:** Web Application Archive. Used for Java web applications (runs in servlet containers like Tomcat).
        + **pom:** Used for parent or aggregator projects that manage dependencies but don’t generate code/artifacts.
        + **ear:** Enterprise Archive. Used for Java EE applications (contains JARs + WARs).
        + **maven-plugin:** Used to build Maven plugins.
      * **Name:** It’s mainly used for display purposes (e.g., in logs, documentation, or Maven UIs like Jenkins, IntelliJ, or Maven Central). It does not affect the artifact's filename or Maven’s build process.
        + **We can leave it empty as well.**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Once you provide all the required details, it will create below empty Java-Maven project with below file structure**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + **File Structure of Java-Maven Project:**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **src/main/java:** On this location, we actually write code for our application. Means for our application, whatever classes are needed, whatever files are needed, will be located to this location.
    - **src/main/resources:** On this location, we store resources such as configuration files (application.properties file, .yaml, .xml files) , data files (.CSV, JSON, XML), database scripts, Images, fonts ect.
    - **src/test/java:** On this location, we will write the test cases for our application.
    - **src/test/resources:** On this location, It holds **resources needed only during testing**, not during regular application runtime. These resources are included in the **test classpath**, not in the final JAR/WAR.
      * **Below are the Typical contents:** 
        + Test configuration files (e.g., test-config.properties)
        + Mock data files (e.g., test-data.json, sample-input.csv)
        + XML or YAML files for unit test inputs
        + Test templates or scripts
    - **pom.xml:** In this file, we actually write our dependencies requirements that will use for our application. There are several dependencies code available on the internet, we need to use as per our requirement.
      * **When we want to add more dependencies, we need to write dependencies code between <dependencies></ dependencies> tag.**
      * **We can get most of the dependencies on** [**https://mvnrepository.com/**](https://mvnrepository.com/) **website for personal projects.**
      * **For industry projects, we need to pass industries’ private environment code.**
  + **Steps to add Test Case for Application: Add Manually**
    - **Step 1:** Add Java class file in src/main/java location. For that “Right-click-on src/main/java package > Select “New” > Select “Class””

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Step 2:** Provide the details below and click on “**Finish**”
      * **Package:** **Ex.:** com.JavaJUnitTest.Learning
      * **Name:** Provide class name. **Ex.:** Calc

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Step 3:** After we implement and add the application code to the class files in “src/main/java”, we need to write a test cases for each of the units/methods. That we have to add in the “src/test/java” location.

**To add the test case first** “Right-click-on src/test/java package > Select “New” > Select “JUnit Test Case”

**A screenshot of a computer

AI-generated content may be incorrect.**

**If “JUnit Test Case” option is not available in the list then** “Right-click-on src/test/java package > Select “New” > Select “Other…”

**A screenshot of a computer program

AI-generated content may be incorrect.**

**In the dialog box, search “JUnit Test Case” or Select "Java > JUnit > JUnit Test Case and click on “Next”**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Step 4:** In the next step, click on “**Brows**” button next to the “**Class under the test**” and Search for class name on which you want to write test case > select it > Press “**OK**” button. **Ex.** Calc.

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Step 5:** In the “**New JUnit Test Case**” window, provide below details > Click on “**Finish**” button:
      * **Package:** Provide package name same as “**Class under test**” and do not add class name.

**Ex:** If we have “**com.JavaJUnitTest.Learning.Calc**” then copy only “**com.JavaJUnitTest.Learning**”

* + - * **Name:** We can pass any name as per our convenience but for better convenience, write ClassName followed by Test.

**Ex.**: If we have class name “Calc” then our test case name will be “CalcTest”

**A screenshot of a computer

AI-generated content may be incorrect.**

**It will add test file on “src/test/java” location along with the package name we provided.**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Step 6:** To run the Unit test, right click on test file > Select “Run As” > Select “Junit Test”

**A screenshot of a computer program

AI-generated content may be incorrect.**

**It will display result in Junit tab: for test Fail**

**A screenshot of a computer

AI-generated content may be incorrect.**

**It will display result in Junit tab: for test Success**

**A screenshot of a computer

AI-generated content may be incorrect.**

**In the concel for above “Hello Test” SOUT**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + **Steps to add Test Case for Application: Add Automatically using IDE**
    - **Step 1:** After we implement and add the application code to the class files in “src/main/java”, right click on the class file we have created in “src/main/java (EX.: Calc.java)” > Select “New” > Select “JUnit Test Case”.

**A screenshot of a computer

AI-generated content may be incorrect.**

**If “JUnit Test Case” option is not available in the list then** right click on the class file we have created in “src/main/java (EX.: Calc.java)” > Select “New” > Select “Other…”

**A screenshot of a computer

AI-generated content may be incorrect.**

**In the dialog box, search “JUnit Test Case” or Select "Java > JUnit > JUnit Test Case and click on “Next”**

**A screenshot of a computer

AI-generated content may be incorrect.**

* + - **Step 2:** All the required details will be populated automatically in the next dialog box and we just have to click “**Finish**”

**A screenshot of a computer

AI-generated content may be incorrect.**

* **Methods of JUnit 5 testing:**
  + **assertEquals (**ExpectedResultValue/Object, ActualResultMethodGeneraedValue/Object**);**
    - In this method either we can pass expected and actual values directly as a parameter or we can create an object of Class(on which we want to write a test case) in the test file and call the method of the class and store its result in the variable(This will be actual value) and create another variable and store expected result in it. And pass both expected and actual variables to the method parameters.
    - assertEquals(e,a) method is mainly use to compare actual and expected results from the method.

**A screenshot of a computer

AI-generated content may be incorrect.**