++
Pre-requisite: Core Java + Eclipse IDE knowledge + (Advance java basics)
++
JUnit
Unit Testing: The test done by programmer on own piece of code is called unit testing.
Peer Testing: The unit testing done on 1 programmer's code/ task by his colleague programmer is called Peer testing.

Note:
 Testing = matching expected results with actual results.
 if matched then test result is positive (Test succeeded).
 if not matched then test result is negative (Test failed).

Development --> Unit/ Peer Testing should be done continuously by programmer until test results are positive.

Testing done developers:
o Unit testing
o Peer testing
o Integration testing
 [Under monitoring of TL] (talks about modules/ applications integration]

Testing done Testers (QA Team): (They test whole project)
o Performance Test
o Navigation Test
o User-Experience Test
o System testing
o Load Testing
o Functional Testing
o Sanity Testing and etc.

Unit testing can be done in two ways:
a. Manual Unit Testing
b. Automated Unit Testing

Limitations of Manual Unit Testing:
▪ No Productivity (takes time).
▪ Writing test report manually is complex process (Excel sheet report).
▪ Presenting Test plans/ test cases to TL or superior is complex process.
▪ Test Regression (repeating the tests) is very complex.
▪ It is not industry standard.

=> To overcome these problems, take the support Unit testing automation tools like JUnit, HttpUnit (for web applications), Mockito,
 TestNG and etc.
=> For HttpUnit, Mockito and TestNG, JUnit is the base tool (Java based Unit Testing tools).

In Test results we can see:
a. success : expected results are matched actual results.
b. failures: expected results are not matched actual results.
c. errors : Unanticipated/ unexpected exception has come while testing the code.

Q. What is the difference between failure and error?
Ans. failure: actual code has given result but not matching expected result.
 error : actual code has not given result rather it has thrown exception.

While working with Junit we can see 3 main components:
1. Service class/ Main class (Class to be tested) (1 or more).
2. Test case class (The class that contains test methods) (1 or more).
3. Test Suite: Allows to combine multiple test case classes to generated the test report (0 or 1) (optional).

Note: We can run each Test case class manually to generate Test report. But, if want get test report of all the classes together
      then take the support Test suite class.

Eclipse IDE gives built-in Support for JUnit (i.e. eclipse gives JUnit libraries as built-in libraries)

To add Junit Libraries eclipse Java Project:
      right click on project --> BUILDPATH --> configuration BUILDPATH --> libraries tab --> classpath
                                                                        |->add Libraries JUnit --> select JUnit5.

Note: It is recommended to use Junit with Maven/Gradle Project because it gives the support built-in Decompiled to see the source code
      and other advantages.

JUnit5 contains 3 runtime libraries:
a. JUnit Jupiter: JUnit5 libraries
b. JUnit Vintage: JUnit 3/ 4 libraries (for backward compatibility)
c. JUnit Integrations: To allow JUnit integration with TestNg, Mockito and etc.

Junit5 Architecture diagram
      refer: Diagram2

Junit5 Jupiter API gives:
a. Annotations(using this we develop Test cases)
b. Assertions API
      |-> gives Assertions.assertXxx() methods (static methods) to match actual results with expected result and generate test report.

++++++++++++
Annotations
++++++++++++
@Test
@DisplayName
@BeforeEach
@AfterEach
@BeforeAll
@AfterAll
@Tag
@ParameterizedTest
@ValueSource
@NullSource
@EmptySource
@NullAndEmptySource
@TestMethodOrder
@Order and etc.

++++++++++++++++++++++++++++++++++
Static Methods of Assertions class
++++++++++++++++++++++++++++++++++

assertTrue
assertSame
assertNull
assertNotSame
assertNotEquals
assertNotNull
assertralse
assertEquals
assertArrayEquals
assertAll
assertThrows and etc.

Note:: Generally, The Testcase class name starts or ends with "Test" word and all test methods generally begins with "test" word.

e.g.
```
BankService (main class)
      public float calcSimplelntrest(-, -)
      public float getBalance(-)

TestBankService or BankServiceTest (Test case class name)
      public void testcalcSimplelnterest()
      public void testGetBalnace()
            :: we write multiple forms of these test methods to test main method/
service method in multiple angles.
```

Note: In Test Case classes, for each business method/ service method we need to write variety of test methods not quantity test method.

eg#1.
Login App Code:
Possible test methods/ test plans
a. Test with Valid credentials
b. Test with Invalid credentials
c. Test with No credentials

eg#2.
Sum logic of two numbers:
Possible test methods/ test plans
- test with positives
- test with negatives
- test with mixed values
- test with zeros
- test with floating points
- test with chars/ strings

++++++++++++++++++++++++++
First Example Application
++++++++++++++++++++++++++
Step 1:
Create Maven Project in eclipse IDE as standalone Project by taking maven-archetype-quickstart as the Project archetype and change java version to 17.
      [open pom.xml and change java version to 17, Right click on the Project -maven update the project].

File --> maven project --> next --> select maven-archetype-quickstart -> next -->
      group Id: ineuron

```
      artifact Id: JUnitTestProject1
      package: in.ineuron.service --> next --> finish.

Step 2: Add JUnit5 Jupiter jar in pom.xml as dependent by collecting from
mvnrepository.com in pom.xml under <dependencies> tag.
      o junit-jupiter-api.5.7.0.jar
      o junit-jupiter-params.5.7.0.jar [for some work with some advance topic]
      o junit-jupiter-engine.5.7.0.jar

Step 3: Develop main class or service class in in.ineuron.service package of
src/main/java folder.
      src/main/java: To place main source code
      src/test/java: To place unit testing code (test case/ test suit classes)

Step 4: Develop Testcase class with Test Methods in src/test/java folder having
package in.ineuron.test.

Step 5: Then right click on the test class – Run As – JUnit Test

pom.xml
<dependencies>
      <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api
-->
      <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter-api</artifactId>
          <version>5.10.0</version>
          <scope>test</scope>
      </dependency>

      <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-
params -->
      <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter-params</artifactId>
          <version>5.10.0</version>
          <scope>test</scope>
      </dependency>

      <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-
engine -->
      <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter-engine</artifactId>
          <version>5.10.0</version>
          <scope>test</scope>
      </dependency>

</dependencies>

++++++++++++++++++++
BankLoanService.java
++++++++++++++++++++
package in.ineuron.service;

public class BankLoanService {
      public float calculateSimpleInterest(float pamount,float rate,float time) {
            System.out.println("BankLoanService.calculateSimpleInterest()");
            if(pamount<=0 || rate<=0 || time <=0)
```

```
                throw new IllegalArgumentException("Invalid inputs");
            return (pamount*rate*time)/100.0f;
      }
}
```

@Test: To make the method of Testcase class as the Test method
assertEquals()/assertNotEquals(): To check whether expected result is equal or not with actual result and to generate test report.
assertThrows(): To check expected exception has come or not.

Case1:

```
@Test
public void calculateSimpleTestInterestWithSmallNumbers() {
      BankLoanService service = new BankLoanService();
      float actualInterest = service.calculateSimpleInterest(100000, 2, 12);
      float expected = 24000.0f;
      assertEquals(expected, actualInterest);
}
@Test
public void calculateSimpleTestInterestWithBigNumbers() {
      BankLoanService service = new BankLoanService();
      float actualInterest = service.calculateSimpleInterest(10000000, 2, 12);
      float expected = 2400000.12f;
      assertEquals(expected, actualInterest)
}
```

Output : 2 testcase pass

Case2:

```
@Test
public void calculateSimpleTestInterestWithBigNumbers() {
      BankLoanService service = new BankLoanService();
      float actualInterest = service.calculateSimpleInterest(10000000, 2, 12);
      float expected = 240000.0f;
      assertEquals(expected, actualInterest);
}
```

Output : 1 testcase failed(expected output not matching with actual output)

Case3:

```
@Test
public void calculateSimpleTestInterestWithInvalidInputs() {
      BankLoanService service = new BankLoanService();
      float actualInterest = service.calculateSimpleInterest(0,0,0);
      float expected = 2400000.00f;
      assertThrows(IllegalArgumentException.class,()-
>service.calculateSimpleInterest(0, 0, 0));
}
```

Output : 1 testcase passed(Excpetiongenerated is IllegalArgumentException and collected type is also IllegalArgumentException)

Case4:

```
@Test
public void calculateSimpleTestInterestWithInvalidInputs() {
      BankLoanService service = new BankLoanService();
      float actualInterest = service.calculateSimpleInterest(0,0,0);
      float expected = 2400000.00f;
      assertThrows(ArithmeticException.class,()->service.calculateSimpleInterest(0,
0, 0));
}
```

Output : 1 test case failed(expected IllegalArgumentException,found ArithmeticException)

Case5:(Customizing the exception message)

```
@Test
public void testcalcSimpleIntrestAmountWithInvalidInut() {
            BankLoanService service = new BankLoanService();
            assertThrows(ArithmeticException.class, ()-
>service.calcSimpleIntrestAmount(0, 0, 0), "may results not matching");
}
```

Output : 1 test case failed(may results not matching)

Case6:(change in the expected value[delta result])

```
@Test
public void calculateSimpleTestInterestWithBigNumbers() {
      BankLoanService service = new BankLoanService();
      float actualInterest = service.calculateSimpleInterest(10000000, 2, 12);
      float expected = 2400000.12f;
      assertEquals(expected, actualInterest,0.5,"Excepted is not matching with
equal");
}
```