Junit:

Each task given to a programmer is called a unit.

Unit Testing : Test done by programmer on his own piece of code is called Unit Testing.

Peer Testing : The unit testing done on one programmer’s code/task by his colleague programmer

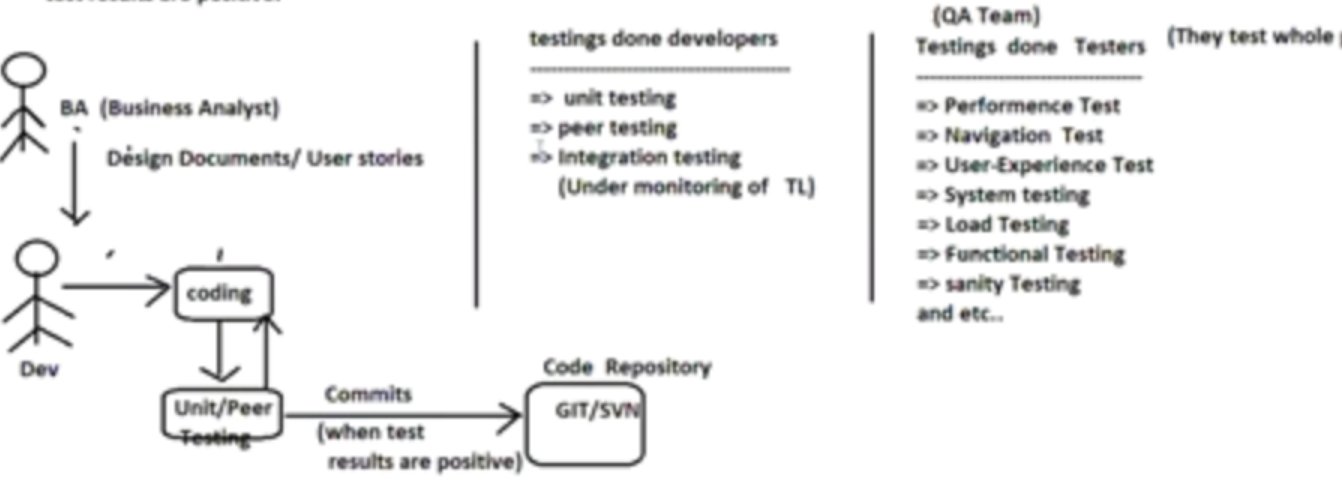
Note : testing = matching expected results with actual results.

If matched then test result is positive(Test succedded)

If not matched test result is negative (Test failed)

Development 🡪 Unit/Peer testing should be done by continuously by programmers until test results are positive.

In unit testing entire project will not be tested only programmer related code will be tested.



Integration testing talks about modules/ app integration.

Unit testing can be done in two ways

1. Manual unit testing
2. Automated unit testing

**Limitations of Manual unit testing:**

🡪No Productivity (takes time)

🡪Writing test reports manually is complex process

🡪Presenting test cases/plans to TL/superior is complex process

🡪Test Regression(Repeating the test) is complex

🡪Not Industry standard

To overcome all this limitations, we need to go for automated unit testing. For this purpose we have some automated tools to automate the unit testing. Junit is one of the tools.

To overcome these problems take the support of Unit testing automation tools like Junit, HttpUnit(for web applications), Mockito(extension of junit), TestNG and etc..

For HttpUnit, Mocito, TestNG----- Junit is the base tool



In test cases we can see,

a)success : expected results matches with actual results

b)failures: expected results not matches with actual results(In failure anticipated output came but it’s not matching with actual results) anticipated == expected

c)errors: Unanticipated/unexpected exceptions has come while testing the code.

What is the difference b/w failure and error?

Failure: actual code has given result…but not matching with expected result

Error: actual code has not given result…rather it has thrown exception.

--------------------------------------------------------------------------------------------------------------------------------------

Junit5:

**While working with Junit we can see three main components:**

a)Service class/main class(class to be tested) --- (1 or more)

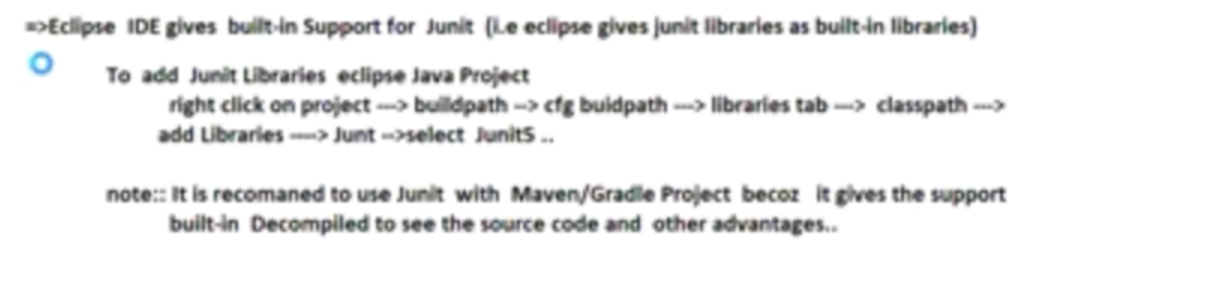
b)Test case class(Class that contains test methods) – (1 or more)

c)Test suite(allow to combine multiple test case class to generate the test report) --- (0 or 1) and (optional)  
To see report of all test case classes, then take the support of Test Suite

note: we can run each test case manually to generate test report

If we want to get test report of all the classes together then take the support of test suite class.

Eclipse give built-in support to work with Junit, but source code is not visible.



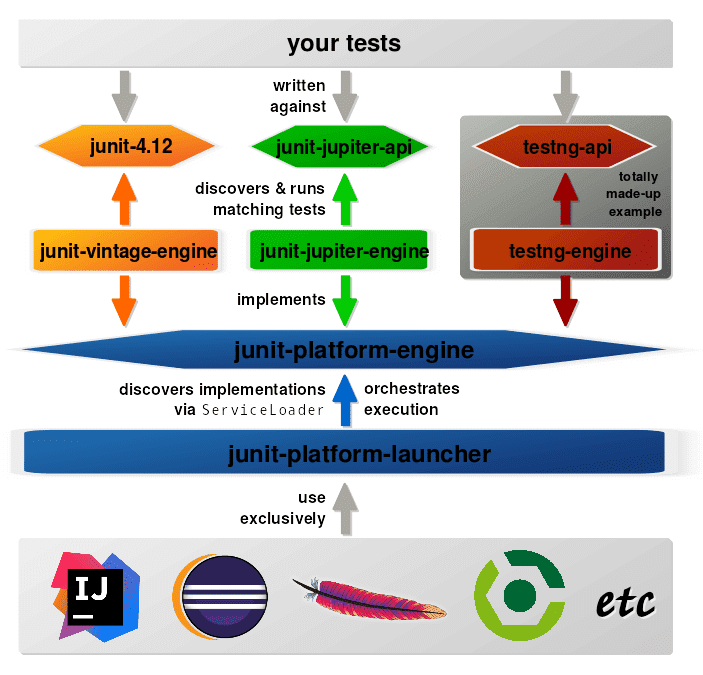
**Junit5 contains three runtime libraries:**

a)Junit Jupiter --- Junit5 libraries

b)Junit Vintage – Junit 3/4 version (for backward compatibility)

c)Junit Integrations -- to allow Junit integration with TestNG, Mockito

Junit5 architecture diagram:



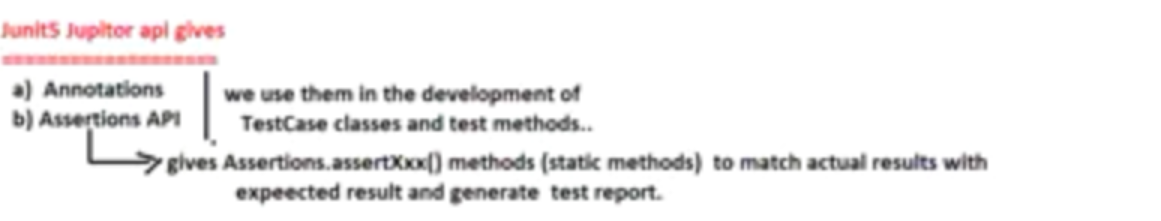
Added text(for my reference):

Junit5 Jupiter api:

a)Annotations

b)Assertions API

a and b -🡪 we use then in the development of TestCase classes and test methods



tes



Generally TestCase classname starts or ends with Test word and all test methods generally begins with “test” word

BankService(main class) --- TestBankService / BankServiceTest(Testcase class name)

🡪p float calcSimpleIntrest() ---- p v testcalcSimpleIntrest()

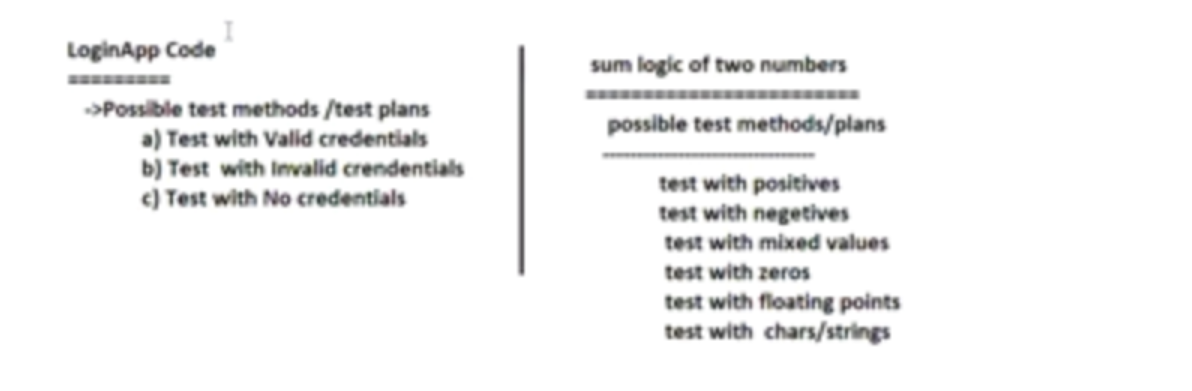
🡪p float getBalance() ----- p v testgetBalance()

Return type is void why?

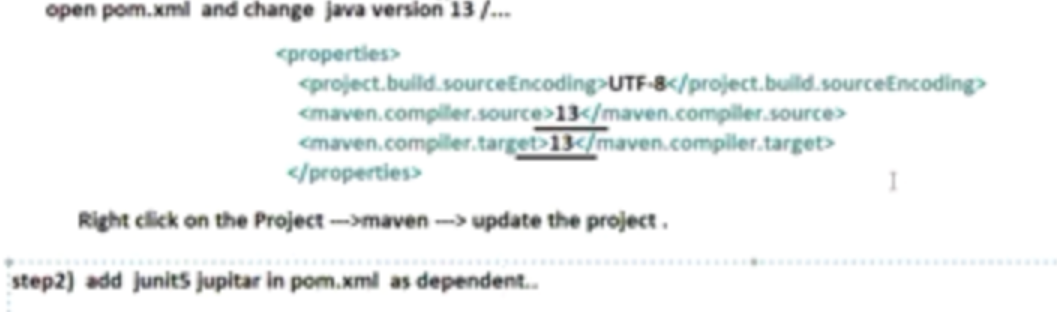
Inside methods there are assertXXX() methods to catch/generate the result and compare the result.

AssrtXXX() are static methods of assertion class.

In testcase classes, for each business method or service we need to write variety of test methods not the quantity test methods.











Step3)develop main class or service class in com.nit.service package of src/main/java

Step4)develop Testcase class with Test methods in src/test/java folder having package com.nit.test

@Test : To make the method of Testcase class as Test method

@assertEqual : To check whether expected value is equal to actual value or not and to generate test report

@assertThrows: to check expected exception has come or not.

Throwing exception means error.

@assertTimeout:to check business method execution is completed in the specified time or not

@BeforeEach: To place common logic that should execute before each test method execution

@AfterEach : To place common logic that should execute after each test method execution.

To write common logic only fo r1 time for all test methods. Then place in @BeforeAll

Clean up logic for all test methods. Then place in @AfterAll

@BeforeAll and @AfterAll -🡪 These methods must be static methods.

These all methods will be called by junit-platform-engine

These engine is responsible to load testcase classes to create objects and to call test methods and other life cycle methods.

Order of execution of annotations:



**BankLoanService.java:**

//BusinessLoanService.java(Main class/ Service class)

**package** com.nit.services;

**public** **class** BankLoanService {

**public** **float** calcSimpleInterestAmount(**float** pAmt,**float** rate,**float** time) {

// System.out.println("BankLoanService.calcSimpleInterestAmount");

**if**(pAmt <=0 || rate<=0 || time<=0)

**throw** **new** IllegalArgumentException("Invalid Inputs");

// try {

// Thread.sleep(10000);

// }

// catch(Exception e) {

// e.printStackTrace();

// }

**return** pAmt\*rate\*time/100.0f;

}

}

**TestBankLoadService.java:**

**package** com.nit.services;

**import** **static** org.junit.jupiter.api.Assertions.*assertDoesNotThrow*;

**import** **static** org.junit.jupiter.api.Assertions.*assertEquals*;

**import** **static** org.junit.jupiter.api.Assertions.*assertThrows*;

**import** **static** org.junit.jupiter.api.Assertions.*assertTimeout*;

**import** java.time.Duration;

**import** org.junit.jupiter.api.AfterAll;

**import** org.junit.jupiter.api.AfterEach;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**public** **class** TestBankLoadService {

**private** **static** BankLoanService *service*;

// @BeforeEach

// public void setUp() {

// System.out.println("TestBankLoadService.setUp()");

// service = new BankLoanService();

//

// }

@BeforeAll

**public** **static** **void** setUpOnce() {

System.***out***.println("TestBankLoadService.setUpOnce()");

*service* = **new** BankLoanService();

}

@Test

**public** **void** calcSimpleInterestAmountWithSmallNumbers() {

System.***out***.println("TestBankLoadService.calcSimpleInterestAmountWithSmallNumbers()");

**float** actual=*service*.calcSimpleInterestAmount(100000, 2, 12); // actual result

**float** expected=24000; // through manual calculation

Assertions.*assertEquals*(expected, actual);

}

@Test

**public** **void** calcSimpleInterestAmountWithBigNumbers() {

System.***out***.println("TestBankLoadService.calcSimpleInterestAmountWithBigNumbers()");

**float** actual=*service*.calcSimpleInterestAmount(10000000, 2, 12); // actual result

**float** expected=2400000; // through manual calculation

Assertions.*assertEquals*(expected, actual);

// Assertions.assertEquals(expected, actual,0.5,"may be result no tmatching..");

//0.5 is delta value --> the difference value that is allowed in the result

}

@Test

**public** **void** calcSimpleInterestAmountWithInvalidInputs() {

System.***out***.println("TestBankLoadService.calcSimpleInterestAmountWithInvalidInputs()");

*assertThrows*(IllegalArgumentException.**class**, ()->

*service*.calcSimpleInterestAmount(0, 0, 0),"may be exception raised is not matching.."

);

}

@Test

**public** **void** calcSimpleInterestAmountWithTimer() {

System.***out***.println("TestBankLoadService.calcSimpleInterestAmountWithTimer()");

*assertTimeout*(Duration.*ofMillis*(5000),()->

*service*.calcSimpleInterestAmount(1000000, 2, 12)

);

}

@Test

**public** **void** calcSimpleInterestAmountForNotException() {

System.***out***.println("TestBankLoadService.calcSimpleInterestAmountForNotException()");

*assertDoesNotThrow*(()->{

*service*.calcSimpleInterestAmount(100000, 2, 12);

}

);

}

// @AfterEach

// public void clear() {

// System.out.println("TestBankLoadService.clear()");

// service=null;

// }

@AfterAll

**public** **static** **void** clear() {

System.***out***.println("TestBankLoadService.clear()");

*service*=**null**;

}

}