**JUNIT 5 / junit Jupiter test**

**INTRODUCTION**

Testing (QA) vs unit testing (SDE)

1 method is called as 1 unit of an application. Sometimes in a bigger app group of methods / classes also been called as 1 unit.

Eg : 5 classes as a single unit.

But generally 1 method is 1 unit.

**NEED OF JUNIT5**

We can do unit test without junit as well

Checking by using if else sout for each & every cond. (manually)

Perform assertion (verify the result)

Report the results (Alert dev if test is failed / not)

JUNIT : Give i/p & give exp o/p. JUNIT will take care. Assertion has multiple api, JUNIT will take care of it.

Some features

Refactor -> inline to make it in a single line

Take a class , right click and add test case to perform junit

**Test Annotation : @Test**

JUNIT 5 : Can be public , protected , default

JUNIT 4 : Can be public alone

Default behavior of @Test is it will detect only failure.

Eg : @Test

Void test(){

}

**o/p : gives green (passed) since no failure here**

**Write test , then code -> test driven development**

**ASSERTIONS**

**AssertEquals** (expected, actual) // (both should match)

**AssertEquals**(expected, actual, "Wrong calculation") // The string will display if and only if there is a failure.

if you’ re using maven by default it will give java 1.5

but for JUNIT Supports >= java 8. So go to pom.xml -> properties -> change java version after creating maven project

Eclipse ide provides a dedicated junit runner to run tests which will show green for pass

we need to add surefire-plugin if we are in need to run the file from cloud. It will show as pass2, fail 0, errors 0, skipped 0 in the console.

**Supplier (Functional interface):**

shape s new shape();

@Test

void computeAreaTest(){

assertEquals(74.5, s.computeArea (5), ()->"Wrong calculation"); // lambda exp is used since its a fn interface.

**assertEqualArray() method:**

No of elements should match in both arrays Ele of both arrays are equal order of elements in both array are equal

int []expected = {1,2,3,4}; int []actual = {5,2,3,4};

**EXCEPTION HANDLING IN JUNITS :**

**Using try, catch:**

sort an array, if we give null for array value. it should throw exception; pass if not; make it explicitly fail by adding fail();

**using assertThrows:**

int unsorted[] = {3,1};

assertThrows (NullPointerException.class, ()-> array.sortingArray (unsorted) ); // fail

assertThrows (NullPointerException.class, ()-> array.sortingArray(unsorted) ); // pass

int unsorted[] = null;

**Test the performance of a unit:**

assertTimeout(Duration.ofMillis(10), ()-> array.sortingArray(unsorted)); // it should execute within 10 secs: ; else fa pass; fail

for junit 4:

@Test(timeout = 100)

assertEquals(Duration.ofMillis(10), ()-> array.sortingArray(unsorted));

**IMPORTANT ANNOTATIONS:**

**@BeforeEach** -> it will be called before running each & every test (obj creation can be done here to avoid code duplication) @ for

junit 4: @before

@**AfterEach** -> it will be called after running each & every test (similar to finally in exception handling)

for junit 4 : @after

@**BeforeAll** -> should be used for a static (no need of obj) method (it will be called before anything else; called before instance becuase' its static)

@**AfterAll** -> @ should be used for a static (no need of obj) method (it will be called after everything else got over)

@**TestInstance** -> using before each obj would be created everytime we run a test. we need to use single instance for all the tests

so, @TestInstance(TestInstance. LifeCycle.PER\_CLASS)