Lets talk about the JUnit Life cycle hooks

So what Junit does is it creates a new instance before running each methods which we have been seeing also right ? We are creating the mathUtil instance every time.  
Now Junit gives you some hooks, like we can initialize the instance in one method and ask junit to execute this particular method to run before it runs every tests. Like that we write a particular method to clean up of tear down after each method has run. For that Junit gives us some hooks and those hooks can be invoked by using annotations

LifeCycle hooks of JUnit:

There are some methods annotations in the lifecycle hooks of JUnit

1) @BeforeAll

2) @BeforeEach

3) @AfterEach

4) @AfterAll

Let's talk about the @BeforeEach and the @AfterEach first.

So earlier I have talked about the instance creation right ?

So we were creating the instance of the particular test method

in each and every method, which is actually duplicate code. Correct?

So that's not a good practice right.

So what we could do is, instead of creating instances every time,

we can just make use of the @BeforeEach.

Let's see how.

class CircleTest {

Circle circle; // required for BeforeEach explaination as we have to take it as a member variable to make the scope visible to every method   
 @BeforeAll // required for BeforeAll explaination

(static) void beforeAllInit() { // mention about static later

System.out.println("This has to run before all! ");

}

@BeforeEach // required for BeforeEach explaination

void init() {

circle = new Circle(); // First Write it as Circle circle = new Circle(); , but there

will be an error as circle wouldn't be accessible to the

other methods

}

@AfterEach // required for AfterEach explaination

void afterEach() {

System.out.println("Cleaning Up... ");

}

@AfterAll // required for BeforeAll explaination

void name() { // mention about static later

System.out.println("This has to run after all! ");

}

@Test

void circleTest() {

double expected = 314.1592653589793;

double actual = circle.computeRadiusArea(10);

assertEquals(expected, actual);

}

@Test

void circleTest1() {

double expected = 31415.926535897932;

double actual = circle.computeRadiusArea(100);

assertEquals(expected, actual);

}

}

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

Now let's come to the beforeAll and AfterAll.

@BeforeAll and @AfterAll are the methods that execute even before the instance has been created

You might be wondering well if the new instance is created everytime, when does the

@BeforeAll/@AfterAll run, it run before the instance is created

So, this leads to a problem right. Okay, let's say we have a method here

(write the before all method)

The problem is this method has to execute even before the class is instantiated

Now, if you're creating a framework you wouln't be able to run a method of that class

without the instance being created right ?

As the particular methods is in that same class itself.

You seeing the problem here. So, that's why if you use @BeforeAll, you will be getting a nasty error.

In order to this problem there is a restriction for @BeforeAll and @AfterAll.

----> The methods that you annotate with @BeforeAll and @AfterAll has to be static, as you know

right that static methods doesn't have any dependency on the instance creation.

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

@TestInstance(TestInstance.LifeCycle.PER\_CLASS)

This will create only one instance for the class and then run on the same instance.

Using this would help us get rid of the static keyword as only one instance is created.

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

I am gonna tell you about some annotations which would make your life a bit easier:

@DisplayName("the name")

@Disabled

Conditional statements:

EnabledonJre(JRE.JAVA\_9)

EnabledOnOS(OS.Linux)

@EnabledIf  
@EnableIfSystemProperty

@EnabledIfEnvironmentVariable