**JUnit**

**JUnit** - JUnit is a popular open-source framework for unit testing java applications. Junit is a tool used for testing java code to make sure it works correctly. It's especially useful for testing small parts of our code.

**JUnit 5 Annotations** –

1. **@Test-**

* The @Test annotation is one of the most important and frequently used annotations in Junit 5. It is used to mark a method as a test method, which means that the method should be executed as a test. The purpose of the @Test annotation tells Junit which methods should be run as tests
* We place the @Test annotation above a method to indicate that this method contains test code
* When we annotate a method with @Test Junit knows to run this method when executing tests
* Inside the method we can use assertions to check if the expected outcomes are met.
* We can also use the DisplayName annotation with @test to give out test method name to understanding the purpose of the test
* We can disable a test method by using @Disbaled annotation

1. **@BeforeEach and @AfterEach -**

* @BeforeEach and @AfterEach annotation are used to run methods before and after each test method in Junit5
* @BeforeEach Annotation purpose is run a method before each test method to set up any necessary conditions
* @AfterEach Annotation purpose is run a method after each test method to reset any condition

1. **@BeforeAll and @AfterAll –**

* @BeforeAll and @AfterAll annotations are used to run methods once before and after all test methods in a test class. These annotations are useful for performing setup and reset task
* @BeforeAll Annotation run a method once before all test methods in the class to perform setup task that only need to be done once
* @AfterAll Annotations run a method once after all test methods in the class to perform reset tasks that only need to be done once

**4.Conditional Tests -**

* Conditional tests in Junit5 allows us to decide whether a test should run based on specific conditions like the operating system, java version, and system properties
* Operating system condition – 1.@EnabledOnOs – this annotation runs a test only if the operating system matches the specified one

2.@DisabledOnOs – this annotation runs a test from running on the specified operating system

* Java version Condition – 1. @EnabledOnJre – this annotation runs a test only if the JRE version matches the specified one

2. @DisabledOnJre – this annotation prevents a test from running on the specified JRE version

* System property Conditions – 1.@EnabledIfSystemProperty - this annotation runs a test if a specific system property matches a defined value

2. @DisabledIfSystemProperty – t his annotation runs to prevent a test from running if a particular environment variable matches a defined value.

**5.Test Execution Order –**

* JUnit 5 provides @TestMethodOrder annotation to specify the order in which test methods should be executed with a test class we can use Methodorderer.orderAnnoation.class

**6.Nested Tests -**

* Nested tests help us to organize our tests in a structured way. This makes it easier to read and understand. the main logic behind nested tests is to provide a way to structured tests in a nested manner
* Tests that are related can be grouped together within a parent class
* Each nested test class can have its own methods such as @BeforeEach and @AfterEach to setup and reset
* We use the @nested annotation to define nested test classes within the test class

**7. Repeated Tests-**

* Repeated tests allow us to run the same tests multiple times. this is useful for testing code to ensure the stability of our tests
* Running a test multiple times can help us to ensure that the code behaves correctly across multiple executions
* we use @RepeatedTest annotation to indicate that a test should be repeated
* We also specify the number of times the test should be repeated
* We use assertTrue to verify that the generated number is within the specified range
* We can also use Repetition Info Parameter this parameter provides information about the current repetitions and the total number of repetitions
* Display name with @RepeatedTest Annotations allows us to customize the display name for each repetitions

**8.Assertions -**

* Assertions in JUnit are statements used to test if a specific condition in our code is true. When we write a test we use assertions to check if the code produces the expected result if the condition is not met and the test fails.
* Assertions check if the results of our code are correct

**Basic assertions -**

1. **assertEquals -**

* assertEquals is the most commonly used assertions in JUnit. it checks if two values are equal. if they are not the test fails and additionally displays a custom error message.
* the assertEquals method is used to compare two values expected and actual it verifies that the actual value produced by the code matches the expected value
* Basic syntax - assertEquals(expected,actual);

- assertEquals(expected,actual,message);

1. **assertNotEqual -**

* assertNotEqual is an assertion used to verify that two values are not equal. if the values are equal the test fails indicating that the code did not behave as expected
* this method is used to ensure that the actual value produced is different from the expected value
* Basic syntax - assertNotEquals(Unexpected,actual);
* assertNotEquals(unexpected,actual,message);

1. **assertArrayEquals -**

* assertArrayEquals is an assertion used to verify that two arrays are equal. it checks if the array has the same length and element in the same order. if they are not the test fails
* this method is used to ensure that the actual array produced matches the expected array
* Basic syntax - assertArrayEquals(expectedArray, actualArray);
* assertArrayequals(expectedArray, actualArray, message);

1. **assertNull -**

* assertNull is an assertion used to verify that an object is null if the object is not null the test fails
* assertNull method is used to ensure that the actual object produced is null
* Basic syntax - assertNull(actual);
* assertNull(actual,message);

1. **assertNotNull-**

* assertNotNull is used to verify that an object is not null. if the object is null the test fails
* assertNotNull is used to ensure that the actual object produced is not null
* we also adding custom message to our assertions to make it clear why a test fails
* Basic syntax - assertNotNull(actual);
* assertNotNull(actual,message);

1. **assertTrue-**

* assertTrue assertion method is used to verify boolean conditions if the condition are not met the test fails
* assertTrue method is used to ensure that condition is true
* Basic Syntax - assertTrue(condition);
* assertTrue(condition,message);

1. **assertFalse-**

* assertFalse method is used to ensure that a condition is false. if the condition is not false the test fails and displaying the message
* Basic syntax - assertFalse(condition);
* assertFalse(condition,message);

**9. Assumptions in JUnit -**

* Assumptions in JUnit are used to skip tests based on certain conditions.
* They are particularly useful for tests that should only run if certain conditions are met .
* If the condition is not met the test is skipped instead of failing.
* Assumptions.assumeTrue(false) is used to skip a test if the assumption is false,which is useful for conditional test execution
* if(false) is used for conditional code execution within a test, skip some code but the test continues
* assertTrue(false) fails the test if the condition is false.

**10.Timeouts in JUnit -**

* Timeout JUnit5 ensures that a test finishes within a certain time.
* If the test takes longer than expected, it fails
* Two ways to setting up Timeout in Junit5

1. @Timeout Annotations
2. assertTimeout Methods

* @Timeout Annotations - we can put the @Timeout annotation on a test method to specify how long it should run. if the test takes longer then it fails
* assertTimeout - Waits for the test to complete if takes too long then fails

**11.Expected Exceptions -**

* We can test that a specific exception is thrown using assertThrows.
* The assertThrows method checks that the specified exception is thrown by the code within an expression.
* if the exception is not thrown the test fails
* Using assertThrows in JUnit5 helps ensure that our code handles exceptions as expected.

**12. Parameterization in JUnit5 -**

* Parameterized test allows us to run same test logic multiple times with different inputs

1. ParameterizedTest - Each execution of the method used different data provided by one of the source annotation
2. @ValueSource - This source annotation provides a simple array of values
3. @EnumSource - This source annotation provided values from an enumeration
4. @MethodSource - This Source annotation provides values from a static method
5. @CsvSource - This Source annotation provides values in a list each row in the CSV represent a set of parameters

**Using JUnit4**

**13.Execute classes and packages with JUnit runner -**

* JUnit runner is a component of the JUnit that is responsible for running JUnit tests
* Use different runners for different testing tests
* we Use @RUnWith to specify a runner
* we can control how and in what order our test are run
* we can run multiple test classes together

**14. Tagging and Filtering -**

* **Tagging -** Tagging is a way to label our test method with tags using “@Tag” annotation. These tags can describe the type of tests and help us to categorize our tests. we can choose to run specific group of test based on their tags
* **Filtering -** Filtering allows us to run only the tests that have specific tags. This is useful when we want to focus on a running test that belongs to a particular category. use @tag to label tests