Differences and similarities between TestNG and JUnit 5

TestNG	JUnit 5
@BeforeClass: Executes before all test methods in the class, but only once.	**@BeforeAll:** Executes before all test methods in the class, but only once.
@AfterClass: Executes after all test methods in the class, but only once.	**@AfterAll:** Executes after all test methods in the class, but only once.
@BeforeMethod: Executes before each test method.	**@BeforeEach:** Executes before each test method.
@AfterMethod: Executes after each test method.	**@AfterEach:** Executes after each test method.
@BeforeSuite: Executes once before each test suite	
@AfterSuite: Executes once after each test suite	
@BeforeTest: Executes before all test methods in the class.	
@AfterTest: Executes after all test methods in the class.	

TestNG	JUnit 5
Disable Test: In both JUnit and TestNG, annotations allow for the deactivation of specific test methods. In JUnit, the @Ignore annotation can be used to prevent the execution of a test method. The enabled attribute of the @Test annotation can be utilised in TestNG to disable the execution of a test method.	

`@Test(enabled = false)`
`public void test1(){`
`  // test code`
`}`
	`@Disabled`
	`@Test`
	`public void test1(){`
	`  // test code  }`

Timeout Test: In JUnit and TestNG, both test frameworks allow you to set a timeout for a test method using the @Test annotation with the timeout parameter. A test must be completed within the specified time. Otherwise a timeout error will be returned.	

`@Test(timeOut = 5000)  //millisecond`
`public void test1(){`
`  // test code`
`}`
	`@Test`
	`@Timeout(3)  //second`
	`public void test1(){`
	`  // test code  }`

TestNG | JUnit 5

Group Test: Group Test: Organising and categorising tests is standard practice in test automation frameworks. TestNG provides a thorough solution, whereas JUnit offers a more restricted approach.. In JUnit, test methods can be categorised using the @Nested annotation, while in TestNG, you can group your test methods by grouping tests with the "groups" attribute of the @Test annotation.

Report: TestNG provides built-in reporting features that help you create detailed test reports. These reports can be customised to suit your needs and are usually generated in HTML format. JUnit 5 needs third-party libraries or extensions to generate detailed test reports. It does not provide built-in reporting features like TestNG.

Exception: In both JUnit and TestNG, you can handle exceptions during testing. In TestNG, you can use the expectedExceptions attribute in the @Test annotation to specify the expected exception class. In JUnit 5, you can use the assertThrows() method, which takes the expected exception type and a lambda expression containing the code that should throw the exception. If the expected exception is thrown, assertThrows() returns the exception for further inspection.

```
@Test(expectedExceptions = ArithmeticException.class)
public void test1() {
        int result = 20/0;
}
```

```
@Test
public void test1() {
    assertThrows(ArithmeticException.class, () -> {
        int result = 20/0;
    });
}
```

TestNG

Parallel Test: TestNG offers superior support for parallel test execution compared to JUnit. It allows tests to run at various levels (method, test, class, instance), whereas JUnit primarily supports parallelism at the class level. TestNG also provides extensive flexibility for configuring parallel execution, such as specifying thread count and test execution order.

```
public class MyTestClass {
    WebDriver driver;
    @Test
    public void testMethod1( ) {
        driver = new ChromeDriver( );
        driver.get(”https://www.google.com”); }
    @Test
    public void testMethod2( ) {
        driver = new EdgeDriver( );
        driver.get(”https://www.google.com”); } }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="MyTestSuite" parallel="methods">
    <test name="MyTest">
        <classes><class name="MyTestClass" /></classes>
    </test>
</suite>
```

JUnit 5

Parallel Test: JUnit 4 does not have built-in support for parallel test execution. In contrast, JUnit 5 introduced limited support for parallelism with its own parallel test engine, although it is not as proficient as TestNG.

A "junit-platform.properties" file is created in the resources packages. The content of the junit-platform.properties file is as follows:

```
junit.jupiter.execution.parallel.enabled = true
junit.jupiter.execution.parallel.mode = default
junit.jupiter.execution.parallel.max.threads = 4
```

```
@Execution(ExecutionMode.CONCURRENT)
public class MyTestClass {
    WebDriver driver;
    @Test
    public void testMethod1( ) {
        driver = new ChromeDriver( );
        driver.get(”https://www.google.com”); }
    @Test
    public void testMethod2( ) {
        driver = new EdgeDriver( );
        driver.get(”https://www.google.com”); }
}
```

TestNG

JUnit 5

Parameterized Test: JUnit 5 and TestNG both support parameterized tests, which allow you to run the same test method multiple times with different sets of parameters. In order to create a parameterised test in JUnit 5, the '@ParameterisedTest' annotation is used, whereas in TestNG, you can use the '@Test' annotation with the 'dataProvider' attribute.

```
public class testNG {
   @Test(dataProvider = "dataProvider")
   public void testMethod(int a, int b) {
      System.out.println(a+b);
   }

  @DataProvider
   public Object[ ][ ] dataProvider( ) {
      return new Object[ ][ ]{
            { 1, 2 },
            { 4, 5 }
         };
    }
}
```

```
public class junit{
   @ParameterizedTest
   @MethodSource("dataProvider")
   public void testMethod(String name, int age) {
      System.out.println(name);
}

static Stream<Arguments> dataProvider( ) {
      return Stream.of(
            Arguments.of("Alice", 25),
            Arguments.of("Bob", 30)
         );
    }
}
```