Depends on groups and Dependson methods :-

Dependson groups :-

→ This is the one of the arguments at @Test annotation

→ By using this argument we can Succeed dependencies of the two or multiple groups.

→ It means if we create two groups like Smoke & Regression then we have to execute the Regression group after successful execution of Smoke group.

→ To achieve above synario or concept we have to use dependsongroups argument at regressionTestmetho annotation.

Syn :-

```
@Test (groups = "Smoke")
Public void method1 ( ) {
    ==========
}
@ Test ( groups = "Regression", dependsongroups="Smok
  Public void method2 ( ) {
      ==========
  }
```

Ex :-

Dependsonmethods :-

→ This is the one of the arguments at @Test annotation to use dependencies of the two or more methods.

→ It means if we declare two methods like method1 and method2, the method2 depends on method1 then @Test annotation of method2 we have to use dependsonmethods argument and Provide the methodname as a value of that argument.

Syn :-
```
@Test
P.V. method1 {

}
@Test(dependsonmethods = "method1")
P.V. method2() {

}
```

Ex :-

Groups argument in @Test annotation :-

→ This groups argument, we can use for to decide the current @Test annotation method for which group.

→ It means if we declare @Test annotation test method as one group like Smoke then that test method we can believe for only Smoke group.

Syn :-
```
@Test (groups = "Smoke")
P.v. method1 {
}
@Test (groups = "Regression")
P.v. method2 {
}
```

Ex :-

Priority argument in @Test annotation :-

→ This is argument is most useful argument to execute our test methods in state transaction flow.

→ It means if we would like to execute our test method in some Particular order then we can use Priority argument at @Test annotation.

→ The Priority starts from 'zero'.

→ By using this Priority argument only we can decide which functionality of @Test annotation method need to execute first and need to execute next.

Syn :- @Test (priority = 1)

Public void method1() {

}

@Test (priority = 0)

Public void method2() {

}

Output :- method 2
method 1

Ex :-

Enable argument in @Test annotation :-

→ This enable argument at @Test annotation method, if we would like to ignore or not to treat that method itself as a not implemented then we can use enable argument as false and declare that one in @Test annotation.

Syn :-

→ If we are not given the enable argument itself then it will treat enable = true by default in @Test annotation.

Syn :-
```
@Test (enable = false)
    P.V. method1( ){
    }
@Test (enable = true)
    P.V.method2( ){
    }
@Test
P.V. method 3( ){
}
```

// ouput :- method 2
method 3

Ex :-

invocationCount argument in @Test annotation :-

→ By using invocationCount argument we can execute a @Test method multiple times without using loops.

→ To execute multiple times to one @Test method we have to declare a invocationCount argument value in @Test annotation.

Syn :-

```
@Test (invocationCount = 10)
p. v. method1 ( ) {
}
@Test ( invocationCount = 5)
P.v. method2 ( ) {
}
```

Ex:-

invocationtimeouts argument in @Test annotation :-

→ This argument we can use in @Test annotation method with combination of invocationCount and invocationtimeouts.

→ Syntax :- @Test (invocationCount = 10, invocationTimeOut = millisec.)

```
Public void method1() {

}
```

```
@Test(invocationCount = 5, invocationTimeOut = millisec)
P.V. method2() {

}
```

Ex :-

Parameters :-

→ @ Parameters annotation in TESTNG :-

⇒ By using this @Parameters annotation we can read the Parameters which we declare @ Suit file.xml and we can pass those Parametus as arguments of @Test annotation test method.

Syn :-

```
<Suite name = "Smoke"
    < Parameter name = "username" Value = "Admin" />
    < Parameter name = "password" Value = "admin123" />
    <Test name = "test1"
    < classes
        < class name = " Package.classname" />
        < class name = " Package.classname" />
    </classes>
</test>
</suite>
```

Syn:- @Parameters({"username", "Password"})

```
@Test
P.V. method1(String un, String Pwrs){
    S.O.Pln(un);
    S.O.Pln(Pwrs);
}
// O/P:- Admin
        admin123
```

Note:1 :- once we have use @Parameter annotation for @Test methods then we have to execute those @Test methods from @Parameter declared Suit file.xml.

Note_2 :- If we declare multiple Parameters and calling those multiple Parameters in Single @Test annotation method then we should follow below Syntax.

```
@Parameters({"username", "Password"})
```

If we have to call Single parameter for one Single @test annotation method then we have to follow below Syntax.

```
@Parameter("username")
@Test
P.V. method1(String un){
    S.O.Pln(un);
}
```

Note:3:- While creating Parameters in suit file we should always give values in String formate.

Note:4:- In suit file, the parameters we can create either suit level or test level or class level.

Ex:-
```
<Suite name = "Suite" >
    <Parameter name = "username" Value = "admin" />
<test thread-count = "5" name = "Test" >
    <Parameter name = "password" Value = "admin123" />
    <classes>
        <class name = "com.Testngframework.org.Data Provider" />
    <classes >
    </test >
    </Suite>
```

Ex:-2
```
@Parameter({"username", "password"})
@Test
  Run | Debug
Public void loginofOrangeHRM (String un, String Pws)
Thread.Sleep(5000);
driver.findElement(By name("username")).sendKeys(un);
driver.findElement(By name("password")).sendKeys(Pws);
driver.findElement(By xpath("//*[@type='submit']")).click;
Reporter.log("login success");
}
```

Data Provider in testNG :-

→ Data provider is a extraordinary feature in TestNG which can help to provide the data in @Test method.

→ Here in TestNG we Create a seperate method which can Create data or Provide the data.

→ That method we should declare as a dataprovider method by using @Data provider annotation

→ In inside the @Data provider Annotation we should declare a name by using name argument Provider then that method name will become as a Data provider name.

Syn :-

```
@DataProvider (name = "id dataProvider")
Public object [ ][ ] datamethod ( ) {
object [ ][ ] obj = new object [2][2]
object [0][0] = 100;
object [0][1] = 200;
object [1][0] = 300;
obi [1][1] = 400;
return obj;
}
@Test (dataprovider = "id dataprovider")
P.V. getmethod ( ) {
}
```

	0	1
0	100	200
1	300	400

Note :- The above dataprovider method is creating in a class and calling in same class then we can provide the dataprovider name in @test annotation by using dataprovider argument only.

→ If we are creating dataprovider in different class and calling in different class of @Test annotation method then we must provide dataprovidername and dataprovider implemented class name by using dataProvider and data provider class arguments in @Test annotation.

```
@dataProvider (name = "logindata")
Public object[][] orangeHRMloginData () {
    object[][] data = new object[2][2];
    data[0][0] = "Admin";
    data[0][1] = "admin123";
    data[1][0] = "admin";
    data[1][1] = "admin123";
@Test (dataProvider = "login data")
Public void login of orangeHRm (String un, String psw)
    driver.get (URL);
    Thread.Sleep (7000);
driver.findElement (By.name ("username")).sendkeys (un);
driver.findElement (By.name ("Password")).SendKeys (Psw);
driver.findElement (By.xpath ("//*[@type="submit"]")).cli
                                                    ();
    Reporter.log ("login success");
}
```

Ex :- @Test (dataprovider = "logindata", dataProviderClass
= dataProviderexample.class)

Public void loginOrangeHRM(String un, String pws)
driver.findElement(By.name("username")).sendKeys(Un);
driver.findElement(By.name("Password")).sendKeys(Pws);

Parallel execution TestNG :-

The TestNG framework is Providing Parallel execution of either Suitelevel or Tes level classes level, group level and method level. But this Parallel execution we need to succeed by using parallel keyword in Testing.xml file

→ while giving parallel keyword we must declare thread_count as well and while giving Parallel execution we must tell which level of Parallel execution should happen

→ To give Parallel level for parallel execution below are the names we need to give.
methods, classes, groups, tests.

Note :- The Parallel execution we can conduct in b/w methods, classes, groups, tests but that declaration of the Parallel keyword in testNG.xml file we must give either suite tag level (or) test tag level.

Note 2 :- To succeed Parallel execution either methods or classes or groups or tests we must execute from suite file.xml.

Syn :-

```
<suite name = "suite" >
<test thread-count = "5" name = "test" Parallel="classes">
<classes >
<class name = "com.TestNGframework.org.Parallelexample1"/>
<class name = "com.TestNGframework.org.Parallelexample2"/>
</classes >
</test >
</suite >
```

Listeners in TestNG :-

→ Listener is a one of the test feature in TestNG which can help to track every @Test annotation test method.

→ It means if we would like to response on each @Test methods according to the method status.

→ This listeners we can implement by using ITestListener interface.

→ ITestlistener interface will Provide below methods.

- onTestStart
- onstart
- onfinish
- onTestsuccess
- OnTestfailure
- onTestskipped
- onTestfailed But within Success Percentage.

→ By using above methods we can track the each & every @Test methods like how many testmethods are Passed and how many test methods are skipped and how many test methods an failed.

onStart() :-

→ By using onstart() we can drop a manage and write some functionality when test is started the execution.

→ It means when our class or .java file execution Started then on start ITest listener interface

method will execute.

Syn :- Public void onStart( ){

Syso("Test execution Started");

}

onfinish :-

→ This method will execute when our .class file on .java file executed is completed..

→ It means whenever our test got completed onfinish() method start the execution.

Syn :- Public void onfinish() {

Syso("Test execution is done");

}

onTestStart :- This is one of the methods in ITestListener interface, which can execute once our @Test method got started.

→ It means, if we would like to know our @Test method got started or not then we can use this method to track it.

Syn :- Public void onTestStart() {

S.O.ptln("@Test method got started the execution");

}

onTestSuccess :-

→ This is one of the method in ITestlistner interface which can execute once our @Test method got successfully executed.

Syn :- Public void onTestSuccess ( ) {

Syso ("@Test method Success");

}

→ It would like to know the our @Test method got successfully executed then we can use this method to track it.

OnTest-failure :-

→ This is one of the method in ITest listener interface, which can execute once our @Test method got failed in execution.

→ It means we would like to know our @Test method got failed in execution, then we can use this method to tract it.

Syn :- Public void onTest-failure ( ) {

Syso ("@Test method failure");

}

OnTest skipped :-

→ This is one of the methods in ITestlisteners interface, which can execute once our @Test method got skipped in execution.

→ It means, if we would like to know the our @Test method got skipped in execution, then we can use these method to track it.

Syn :- Public void onTestSkipped() {
    Syso("@Test method skipped");
}

onTestFailed But within Success Percentage() :-

→ This is one of the method in ITestlistener intaface, which can help is to give passed percentage in execution.

→ It means if we would like to know the passed and failed percentage the we can use this method for track it.

Syn :- Public void onTestFailureBut
    Syso("@Test method failed Percentage");
}

→ Above the methods in ITestlistener which can help us to track the each and every @Test methods based on the status of those methods.

Creating listener class By using ITestlistener :-

→ ITestlistener is a interface, which contain unimplemented methods.

→ To implement those unimplemented methods, child class of ITestlistener interface is responsible.

→ It means which class is going to use the ITestlistener interface that class is responsible to

implement the ITestlistener interface method.

→ To give the Parent & child relation in b/w class & interface we must use implements keyword.

Ex :- Public Class TestNGlistenerExample implements ITestlistener {

```
@Override
Public void onTestStart (ITestResult result) {
Reporter.log (result.getname()+"onTestStart");
Syso ("onTestStart");
}
@Override
Public void onTestSuccess (ITestResult result){
Syso ("onTestSuccess");
}
@Override
Public void onTestfailure (ITestResult result){
Syso ("onTest-failure");
}
@Override
public void OnTestSkipped (ITestResult result ){
Syso("onTestSkipped");
}
@Override
Public void onTestfailedBut within SuccessPercentage (ITestResult result) {
Syso ("onTestResult");
}
```

```
@ Override
Public void context (ITestContext Context){
    Syso("Onstart");
  }
@Override
Public void cofinish (ITestcontolt context ){
      Syso("onfinish");
    }
  }
```

Attaching listener class to our @Test methods or Test Class.

→ To attach the listner class to our @Test method or Test class, we have two ways.

way1:- By using @listeners annotation we can attach our listener class to our test class.

Syn:- @listeners (com.TestNG framework.org.TestNG listeners exp.class)

```
Public class Example {
}
```

Ex:- @listeners (com.TestNG.framework.org.TestNG listeners exp.class)

```
Public class invocationcountexample {
@Test (invocationCount = 10000, invocatim Timeout = 1)
Public void testmethod1() {
  Syso ("Hello");
}
@Test
```

```
Public void testmethod2() {
    Assert.fail();
    Syso("Hello");
}
@Test(dependsonmethods = "test method 2")
Public void testmethod3() {
    Syso("Hello");
}
}
```

Note :- In above class we declare a @listeners annotation at top of the class to attach our listener class with our test class.

way2 :- If we have multiple test cases and we have to provide our listener class to all test classes then we can create a suite file and inside the suite file we can add our listener class,

```
<Suite name = "Suite" >
<listeners>
<listener class_name = "com.TestNG.framework.org.TestNG
                        listenerExample"/>
</listeners>
< test thread_count = "5" name = "Test" >
< class name = "com.TestNG.framework.org.Priorityexample"/>
</classes>
</test>
</suite>
```

IRetryAnalyzer :

→ Whenever we would like to add the recovery for times of execution then we can use the IRetry Analizer.

→ It means if any @Test method got failed due to unwanted interruption then we have to execute that @Test method for second time. To acheive this situation we have IRetryanalyzer in Test NG.

→ I RetryAnalyzer interface allows you to rerun a failed test method a set of times before declaring it as failed test method.

Implementation of IRetryanalyzer interface :-

```
Public class RetryAnalzer class implements IRetryAnalyzer
{
    int retryCount = 0;
    int maxretrycount = 3;
  @ Override
    Public boolean retry (ITest Result Result ){
        if (retry count < max retry count ){
          retrycount ++;
          return true;
        }
        return false;
    }
}
```

→ The above class containing RetryMethod, which can help rerun the failed that method on fixed times.

→ To attach this retry method or retry method provided class to our @Test method we must use RetryAnalyzer argument of the @Test method

→ By using RetryAnalyzer argument we should provide the RetryMethod provider class.

Ex:-

```
Public class Test class 1 {
    @Test (retryAnalyzer = RetryAnalyzer.class)
    Public void method1() {
        S.O.Pln ("Hello");
        Assert.fail();
    }
    @Test
    Public void method2() {
        S.O.Pln ("Good morning");
    }
}
```

Conclusion for TestNG:-

→ By using TestNG framework we can successfully cover all possible testing concepts like @Test methods, Precondition, Post Conditions, groups, depending groups, dependon methods, parallel execution, cross browsing testing, Suite execution,

Tracking of every @Test methods & recovering the @Test methods, etc...

Here TestNG means executing the testing in next level to bring quality Project or Product.