15: Batch Execution/Test Suite:

--->It is an xml file which contains all the test classes that need to be executed.

-->Procedure to create Test Suite

-->select TestNG Package-->Right click-->TestNG-->Convert to TestNG-->Click on finish

Result: TestNG.Xml file will be created

-->Procedure to execute test Suite

-->Select TestNG.xml-->Right click--->Run as-->Test Suite

--->XML Suite file Syntax

```
<suite name="">
<test name="">
 <packages>
   <package name=""/>
</packages>
</test>
<test name="">
 <classes>
  <class name="packagename.classname">
   <methods>
   </methods>
</class>
</classes>
 </test>
</suite>
```

---

Script:

```
package testNG;

import org.testng.annotations.Test;

public class Script_7
{
	@Test
	public void test1()
	{
		System.out.println("Test 1");
	}
	@Test
	public void test2()
	{
		System.out.println("Test 2");
	}
	@Test
	public void test3()
	{
		System.out.println("Test 3");
	}
	@Test
	public void test4()
	{
		System.out.println("Test 4");
	}
	@Test
	public void test5()
	{
```

```
                System.out.println("Test 5");
        }

}
```

-->Above code to exclude test2 method

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="testNG.Script_7">
                <methods>
                        <exclude name="test2"></exclude>
                </methods>
      </class>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

---

-->Below code to include test1 and test2 method

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="testNG.Script_7">
```

```
            <methods>
                <include name="test1"></include>
                <include name="test2"></include>
            </methods>
    </class>
  </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

---

***TestNG Annotation:

1: @Test: This annotation is used to perform action

2: @BeforeMethod: This annotation is used to execute a method before every test method execution

3: @AfterMethod: This annotation is used to execute a method after every test method execution

4: @BeforeClass: This annotation is used to execute a method before every test class execution

5: @AfterClass: This annotation is used to execute a method after every test class execution

6: @BeforeTest: This annotation is used to execute a method before suite level test tag execution

7: @AfterTest: This annotation is used to execute a method after suite level test tag execution

8: @BeforeSuite: This annotation is used to execute a method before suite execution.

9: @AfterSuite: This annotation is used to execute a method after suite execution.

Note:

BeforeSuite

BeforeTest

Beforeclass

BeforeMethod

Test

AfterMethod

AFterClass

AfterTest

AfterSuite

Script:

```
package testNG;

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class Annotation
{
	@BeforeMethod
	public void Login()
	{
		System.out.println("Login to app");
	}

	@Test
	public void AddUSer()
	{
		System.out.println("User Added");
	}

	@Test(priority=1)
	public void ModifyUser()
	{
```

```
		System.out.println("User Deatils Modified");
	}

	@Test(priority=2)
	public void DeleteUser()
	{
		System.out.println("User Deleted");
	}

	@AfterMethod
	public void Logout()
	{
		System.out.println("Logout from app");
	}

}
```

Output:

```
Login to app
User Added
Logout from app
Login to app
User Deatils Modified
Logout from app
Login to app
User Deleted
Logout from app
```

---

****Inheritance in TestNG***

-->Test Class is used to convert Test Case

-->Every test case will contain common navigation

-->All the common navigation will be automated under Base Class Or SuperTestNG which need to be inherited to sub class

-->Super class method need to be executed either before sub class method execution or after sub class method execution

-->So we use @BeforeClass And @AfterClass annotation for base class methods.

Script:

```
package testNG;


import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;


public class BaseClass

{

	@BeforeClass

	public void Preconditions()

	{

		System.out.println("Open Browser");

		System.out.println("Open Application");

	}


	@AfterClass

	public void Postconditions()

	{

		System.out.println("close Browser");

	}
```

```
}

package testNG;

import org.testng.annotations.Test;

public class BaseClass1 extends BaseClass
{

	@Test
	public void Signup()
	{
		System.out.println("Application SignUp");
	}

}
```

output:

Open Browser

Open Application

Application SignUp

close Browser

---

***TestNG Verification***

--> To do verification we use if else condition which will increase test script length

-->In TestNG, we use Assert/Hard Assert class static methods for verifications.

1: assertEquals(arg1,arg2): This method is used to verify expected and actual result

-->If both results are same, verification is passed and test method will be pass else fail.

Script:

```
@Test
public void test1()
{
        String str1="hello";
        String str2="hii";
        Assert.assertEquals(str1, str2);
```

---

2: assertNotEquals(): Tis method is used to verify expected and actual result

---->if both results are not same, verification is passed and test method will be pass else fail.

Script:

```
@Test
public void test1()
{
        String str1="hello";
        String str2="hii";
        Assert.assertNotEquals(str1, str2);
```

---

3: assertTrue(): This method is used to verify condition is true or false

--> If it is true then verification is pass

---

4: assertFalse(): This method is used to verify condition is true or false

---->If it is false then verification is pass.

---

5: assertNull(): This method is used to verify element is empty or not

--->It it is empty then veriffication is pass

---

6: assertNotNull(): This method is used to verify element is empty or not.

--->if it is not empty then verification is pass.

---

7: fail(): This metod is used to fail test method execution.

Note: In a TestNG class, if a test method execution is failed, other test methods execution will not be affected

and its deoendent method execution will be skip.

Script:

```
package testNG;


import org.testng.Assert;

import org.testng.annotations.Test;


public class A_fail

{

	@Test

	public void test1()

	{

		String str1="abc";

		String str2="xyz";

		Assert.assertEquals(str1, str2);

	}

	@Test(dependsOnMethods="test1")

	public void test2()

	{

		System.out.println("hii");

	}

	@Test

	public void test3()
```

```
	{
		System.out.println("hiiii");
	}
```

Output:

Test1: fail

Test2: skip

Test3: pass

***Limitation Of Assert/HardAssert

-->In a test method, multiple verfication are existing

-->If one of the verification is failed then other verification execution in that method will be skipped

-->To overcome above limitation we use SoftAssert non-static method for verification

-->assertAll() should be the last command in test method which will notify failed verifications.