**Maven Project:**

**Most Asked Interview Question about CI/CD Pipeline: Let's Break It Down!**  
  
Continuous Integration (CI) and Continuous Deployment (CD) pipelines are the backbone of modern software development. Here's an explanation of the process:  
  
1. Continuous Integration (CI):  
  - Developers work on their code locally, making changes and improvements.  
  - As soon as they're ready to integrate their changes into the main codebase, they push their code to a version control system (like Git).  
  - Upon pushing the changes, the CI server (such as Jenkins, Travis CI, Circle CI, etc.) detects the new code and triggers an automated build process.  
  - During the build process, the CI server pulls the latest code from the repository, compiles the code, runs automated tests, and performs other necessary checks.  
  - If any issues are detected during this process (such as failing tests or compilation errors), the CI server notifies the developers immediately, allowing them to address the problems promptly.  
  
2. Continuous Deployment (CD):  
  - Once the CI process is successfully completed and the code passes all tests, it's ready for deployment.  
  - In a CD pipeline, the CI server can automatically deploy the code to various environments (like development, staging, or production) based on predefined rules and configurations.  
  - Deployment can involve various steps such as packaging the application, provisioning infrastructure (if necessary), configuring services, and deploying the application.  
  - Automated deployment ensures consistency and reliability, reducing the risk of errors that may occur during manual deployment processes.  
  - Continuous Deployment pipelines often include additional steps like smoke testing or integration testing in the deployed environment to ensure that the application behaves as expected in its production environment.

**Cucumber Framework:**

1.What are the layers in cucumber automation

Feature, step definition, runner

AbstractTestNGcucumber.

2. What is the cucumber options attribute to integrate step definitions?

Glue.

3.What is the significance of monochrome?

To remove junk characters.

4. What is the attribute to generate CUCUMBER REPORT?

Publisher=true

5.What are the gherkin keywords used to pass multiple data to the scenario?

Scenario outline followed by Examples

6. What is the gherkin keyword to setup common precondition for the scenarios in a feature file.

Background

7. What are hooks in cucumber?

@Before and @after- these annotations are used to have common pre and post conditions across all scenario

8.What is the exception for the step if it is not defined in step definition?

Undefined step exception. (If step is not defined in step definition).

**Gherkin keywords:**

* Feature
* Given
* When
* Then
* And
* But
* Step Definition
* Runner
* Scenario
* Scenario Outline
* Background- In order to expose the details or the pre-condition that has to be executed before the first statement of the scenario.

Only single pre-condition can set in one feature file. To add multiple pre-condition, you have to split the feature file.

* Examples

**Cucumber options:**

* Features
* Glue
* Dry run-Boolean. (It will check only for the mapping without execution)
* Strict- Boolean. (It will execute and check for the mappings)
* Tags
* Monochrome- To print human readable message in console.
* Publisher-Processing the test results and generating reports.
* Format (Plugin).

**Cucumber Tag(Used to categorize or group the testcases)**

Tag starts with ‘@’.

In runner class, under cucumber options, tags= {‘tagname’}.

1. Tag can be used in feature file level
2. Scenario or scenario outline level.
3. Tags={“~@login”} or not @tagname. (Ignore the tests using tags)
4. Tags=”@functional or @regression”//run all the functional and regression testcases(either one is available the test cases will be executed).
5. Tags=”@functional and @regression”.
6. Tag=@leads in above feature file.
7. Club different tags and run.

**Cucumber Hooks:**

* Used to set things up before/after doing something
* We have two annotations
* @Before, @After.
* We can set multiple precondition and post condition in hooks.

Hooks is to setting things up before and after execution.

Tagged Hooks:

1. We know Before runs before every scenario and after runs after every scenario.
2. Problem is there will be different pre-requisites before for different scenarios. What to do?
3. Solution: Customized (tagged) Hooks for scenarios.

**How to set the Priority of Cucumber Hooks?**

* Priority in Cucumber is almost the same as a priority in [**TestNG**](https://testng.org/doc/documentation-main.html).
* Cucumber executes Hooks in a certain order but there is a way to change the order of the execution according to the need for the test.
* **@Before (order = int):** This runs in increment order, means value 0 would run first and 1 would be after 0
* **@After (order = int):** This runs in decrements order, which means the opposite of **@Before.** Value 1 would run first and 0 would be after 1.

**Without Hooks:**

1. We can achieve common pre-condition and post condition using TestNG annotations.
2. On top of baseclass extends AbstractTestNGcucumber
3. CucumberRunner extends Baseclass.

Note: cucumber will be used by agile team, cucumber is used in UAT automation where TestNG used in SIT testing automation.

TestNG:

For coding(pre-requisite):

Dependency required

<dependency>

<modelVersion>4.0.0</modelVersion>

<groupId>testng.com</groupId>

<artifactId>testngsample</artifactId>

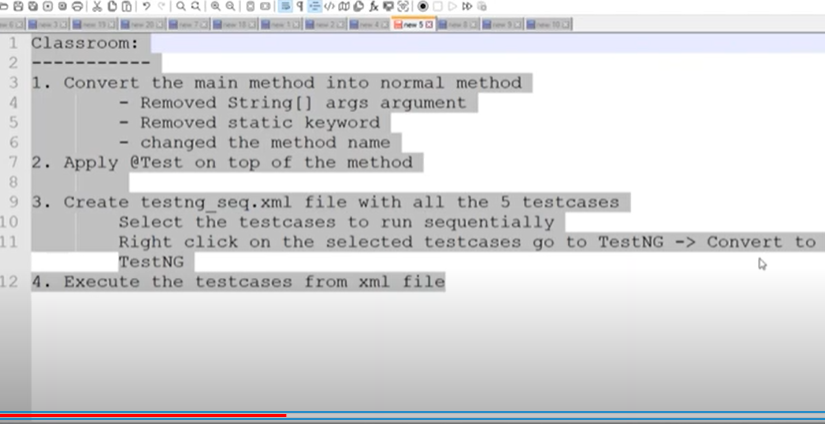
<version>0.0.1-SNAPSHOT</version>

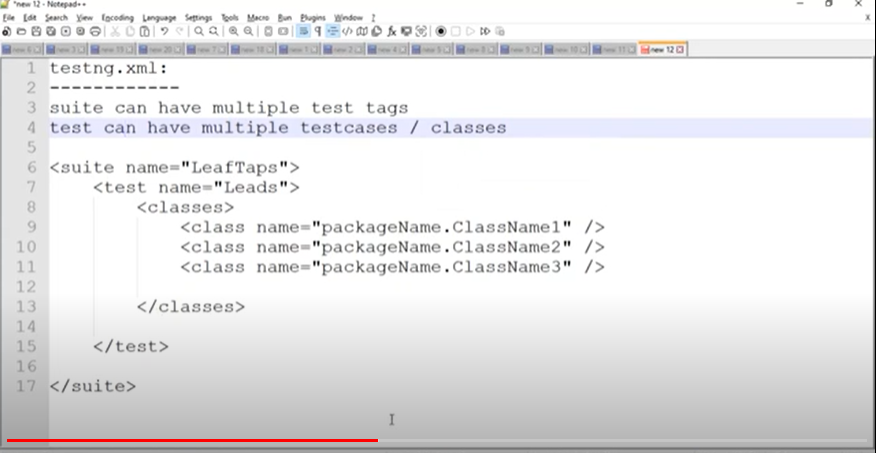
</dependency>

For execution:

<https://testng.org/testng-eclipse-update-site/7.4.0>.

How to convert normal testcase into TestNG testcase:





2.Apply @Test on top of the method

TestNG annotations Execution order:

@BeforeSuite

@BeforeTest

@Beforeclass

@DataProvider

@BeforeMethod

@Test

In TestNG, you can rerun only the failed test cases by utilizing the **rerun-failed.xml** file. This approach involves two steps: initially running your test suite to generate a **testng-failed.xml** file, and then rerunning the failed tests using the generated file.

TestNGAssertion:

Hard Assert:

Assertequals()

Asserttrue()

Assertfalse()

Asset.assetequals(actual value, expected value).

Assert.assertTrue(false)//It expects the Boolean value. True->pass, false->fail.

Ctrl+shift+/->To comment

Ctrl+shift+\->to uncomment.