iTPSXMJ

Cucumber : is a tool which supports BDD

**Requirements** 🡪 SRS🡪Scenarios🡪TestCases🡪 FT->UAT

Testers

Developers

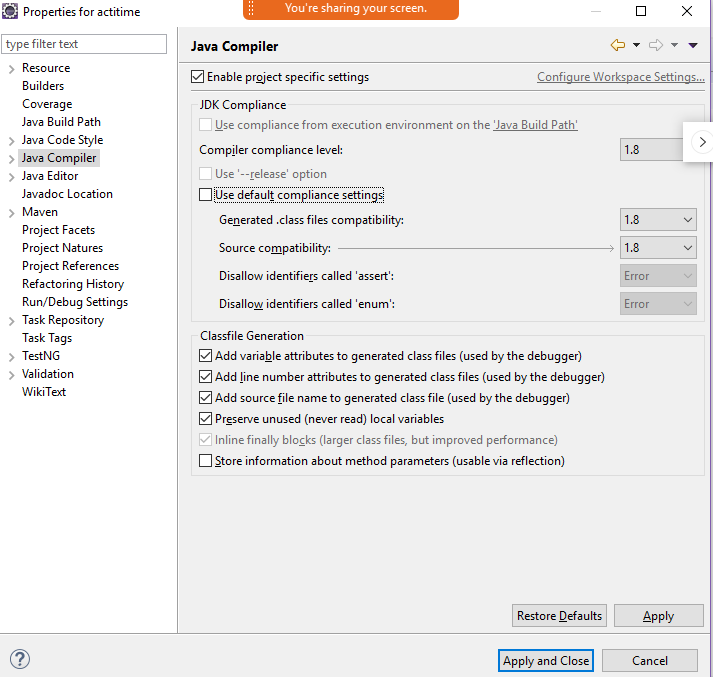
BA

Clients

BDD-> Testers, Developers, BA, Clients

Plain English Statements

Setup :

1. Java-Jdk
   1. Google - > download java jdk 1.8
   2. Install
   3. Update environment variables
      1. JAVA\_HOME C:\Program Files\Java\jdk1.8.0\_201
      2. PATH C:\Program Files\Java\jdk1.8.0\_201\bin
2. Eclipse **or** Intellij **or** netBeans
   1. Download from official website
   2. <https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2019-09/R/eclipse-java-2019-09-R-win32-x86_64.zip>
   3. Unzip the file -> double click on eclipse.exe
3. Cucumber plugin
4. Maven
   1. Download from official website
   2. <https://maven.apache.org/download.cgi>
   3. Unzip-- D:\maven\
   4. Set MAVEN\_HOME - D:\maven\apache-maven-3.6.1
   5. M2 - D:\maven\apache-maven-3.6.1\bin
   6. PATH D:\maven\apache-maven-3.6.1\bin
   7. PATH %M2%
5. Maven Dependencies for cucumber (<https://github.com/AravindaHB/BDD/blob/master/pom.xml>
   1. Cucumber-core
   2. Cucumber-html
   3. Cucumber-java
   4. Cucumber-junit
   5. Cucumber-jvm-deps
   6. Cucumber-reporting
   7. Gherkin
   8. Junit
   9. mockito--‐all
   10. cobertura
6. Update the compiler: \*\*\* \*\*\* \*\*\*
   1. Right click on project->
   2. Click on properties
   3. Select java compiler
   4. If it shows 1.5 change it to 1.8
   5. Add jre from jdk folder
   6. And also select java 1.8 in all dropdown
   7. 

TDD - > Test Driven Development (Unit Tests)

1. Test – Execute - > Fail - >Fix the code - > test will be passed
2. Code

BDD - > UAT (User Acceptance Test)

BA

Customer

Developers

QA

------------------------------------------ BDD--------------------------------------------------------------------------

Anyname.feature (DSL- Domain Specific Language)

Feature File

Gherkin KeyWords:

Feature:

Scenario:

Background

Given

When

Then

And

But

TestRunner

+

Junit Runner

Step Definition

Java, JavaScript, Ruby

Annotations

AUT

**Features** files will be present inside a folder under src

features = "complete location of feature file from src",

features = "src/test/java/features/Sum.feature",

**Step definitions** will be present inside a package(It can be in a single java file or multiple java files)

glue = {"excluding src till Class i.e., complete package name"}

glue = {"com/synechrone/bdd/actitime/bdd/stepdefinitions/"}

Cucumber Options in test Runner:

@cucumberOptions(

**Features** =”location of the feature file from src”,

**Glue** = {“package name – location of your step definition ”},

**Monochrome** = Boolean ( to remove the junk cahrs in console output )

**dryRun** = Boolean ( to perform dryrun of steps { steps are not executed but only existence is verified } )

**plugin** / format = to generate Cucumber reports and to save Junit and Json Reports

**strict** = true 🡪 fail the TC and not continue if there is any failure in the middle of TC

)

SELENIUM

Let us integrate Selenium With Cucumber :

1. Dependencies ( Add all the selenium dependencies to pom.xml)

Data driven testing in Cucumber :

1. Hardcoded data in step definition
2. Passing values from feature file - > we have to use Regular Expression in step definition annotation (\"([^\"]\*)\" )
3. Scenario Outline:
   1. Execute same test for multiple set of test data
   2. While using scenario outline we must have to use Examples keyword
   3. We can specify the data which is “|” separated
4. DataTable:
   1. List<List<E>>

A B C

0 1 2

* 1. List<Map<K,V>>

Tags: are the annotations (UDA) which can be used to the features/ scenarios

If you want to achieve grouping of tests then you can go with TAGS

Automated Tests in Regression Suite :

* Sanity Automated Tests - 100 (10 Reg, 1 UAT)
* Regression Tests - 500 (10 Sanity, 5 UAT,10 Stage )
* UAT Tests -50
* Stage Tests -100
* Modules
  + Module1
  + Module2
  + Module3

NOTE - if you maintain multiple copy of Scenario files then in long run maintenance cost is high

**Hooks**

**@Before**

* **Create a driver**
* **Launching the application**
* **Db connections**
* **Enabling cookies or setting proxy to your application**

**@After**

* **Logout**
* **Closing db connection**
* **Resetting your browser proxy**
* **Close the driver**

***Tagged hooks in Cucumber:***

**When you want to execute some Precondition to a specific scenario**

**Syntax :**

**@Before(“@annotation of scenario”)**

***Order of execution of hooks***

**@Before( order = int) 🡺 execution on incremental basis 0 will be executed first and then 1,2......**

**@After( order = int)🡺 execution is on decrement basis 10 will be executed first and then 9,8,....**

**String programName = “C#”;**

**String str = “hello, Welcome to “ + programName + “ Programming”**

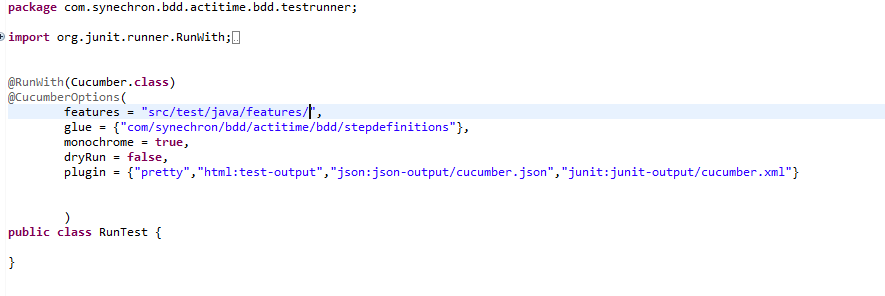
**Sop(str);**

**GoogleSearch.feature**

mvn test -Dcucumber.options="src/test/java/features/GoogleSearch.feature:22" -Dcucumber.options="--tags @regression"

mvn test -Dcucumber.options="src/test/java/features/GoogleSearch.feature:22"

mvn test -Dcucumber.options="--tags @regression"



Injecting the scenario Object

@Before(Scenario name)

Access methods

Dependency Injection

Class A and Class B

Which will be done automatically by the framework, FW will automatically creates a Object which is called dependency and provided to class A for utilization

1. FW Creates a obje which is called Dependency
2. And provide it to calss which is called dependency injection

POM

Page Properties

+

Actions(business Logic)

Step Definition -1

Page properties is changed

Page properties

Page properties is changed

Step Definition -10

Page properties

Page properties is changed

Step Definition -15

Page properties

Page Object Model:

1. Basic Page Class
2. Using Page Factory Methodology

Pages Step Definition TestRunner

Each java program is for one Page in Application

1. Variables - WebElements
2. Functions - Actions WE
3. Constructor – Initialize the Elements of a page

Feature file