SYNTAX TECHNOLOGIES

Cucumber

Class 3

Agenda

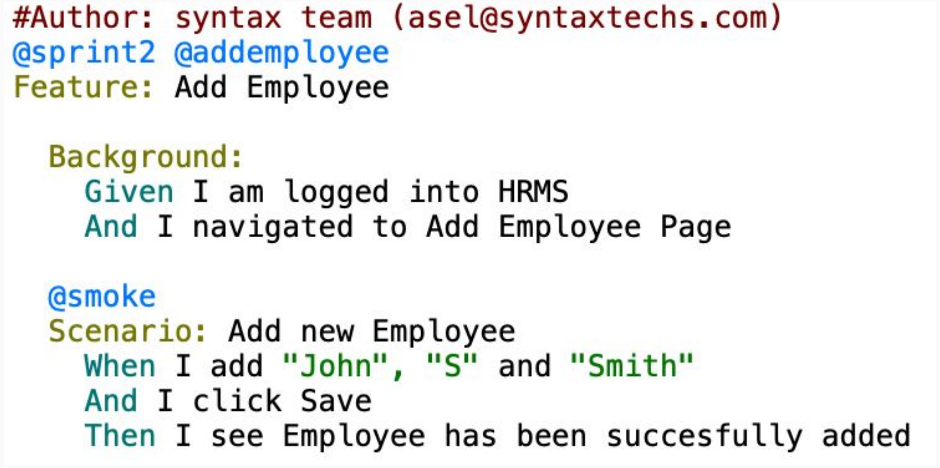
What is Cucumber data driven testing?

Data Driver testing with and Without Examples Keyword

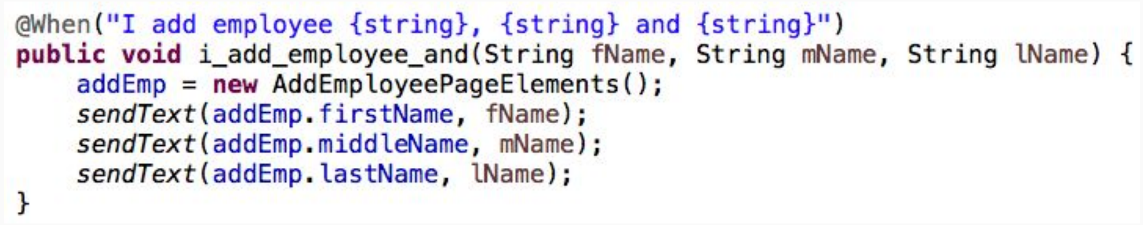
What is Data Table in Cucumber?

Parameterization in Cucumber?

In Cucumber we can easily parameterize the tests.
To parameterize the values, put them in double quotes



```
#Author: syntax team (asel@syntaxtechs.com)
@sprint2 @addemployee
Feature: Add Employee

  Background:
    Given I am logged into HRMS
    And I navigated to Add Employee Page

  @smoke
  Scenario: Add new Employee
    When I add "John", "S" and "Smith"
    And I click Save
    Then I see Employee has been succesfully added
```



```
@When("I add employee {string}, {string} and {string}")
public void i_add_employee_and(String fName, String mName, String lName) {
    addEmp = new AddEmployeePageElements();
    sendText(addEmp.firstName, fName);
    sendText(addEmp.middleName, mName);
    sendText(addEmp.lastName, lName);
}
```

What is Cucumber data driven testing?

Data Driven Testing, which allows to automatically run a test case multiple times with different input and validation values.

In simple terms, cucumber data driven testing means that we can pass data from cucumber feature files to our test cases.

There are different ways to use the data insertion within Cucumber and outside the Cucumber with external files.

- **Data Driven Testing in Cucumber using Scenario Outline with Examples keyword**
- **Data Driven Testing in Cucumber using Cucumber DataTable**

Data Driven Testing Using Examples Keyword

Cucumber supports Data Driven testing by the use of the Scenario Outline and Examples section.

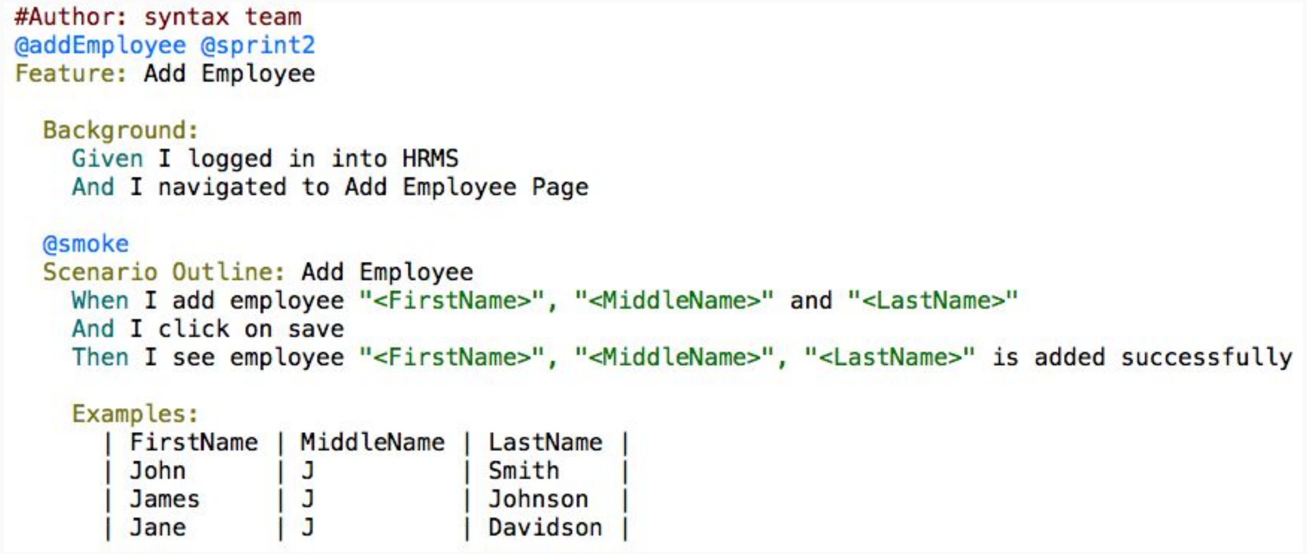
With these keywords cucumber allows us easy Data Driven testing to be completed where no changes need to be made to the Java file.

Example keyword can **only** be used with the **Scenario Outline** Keyword.

Scenario Outline – This is used to run the same scenario for 2 or more different set of test data.

Examples – All scenario outlines have to be followed with the Examples section. This contains the data that has to be passed on to the scenario.

Data Driven Testing Using Examples Keyword



```
#Author: syntax team
@addEmployee @sprint2
Feature: Add Employee

  Background:
    Given I logged in into HRMS
    And I navigated to Add Employee Page

  @smoke
  Scenario Outline: Add Employee
    When I add employee "<FirstName>", "<MiddleName>" and "<LastName>"
    And I click on save
    Then I see employee "<FirstName>", "<MiddleName>", "<LastName>" is added successfully

    Examples:
      | FirstName | MiddleName | LastName |
      | John      | J          | Smith    |
      | James     | J          | Johnson  |
      | Jane      | J          | Davidson |
```

Data Driven Testing Using using Data Tables

With Cucumber data tables, we can pass parameters from feature files in tabular format. And we can then use this data in step definition methods in the form of Lists and Maps.

To use DataTable we do not need to specify any keyword and parameters that will be passed under specific step are applicable only for the that step.

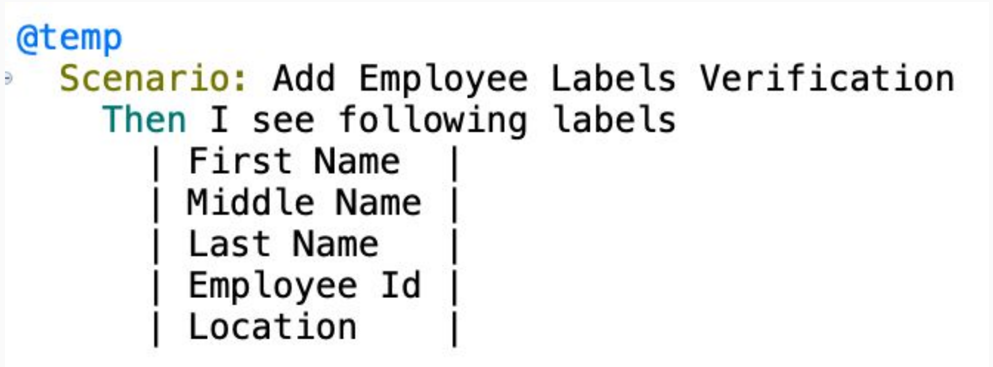
A separate code is needed to understand the test data and then it can be run single or multiple times but again just for the single step, not for the complete test

In Cucumber, we can add data tables in two different formats :

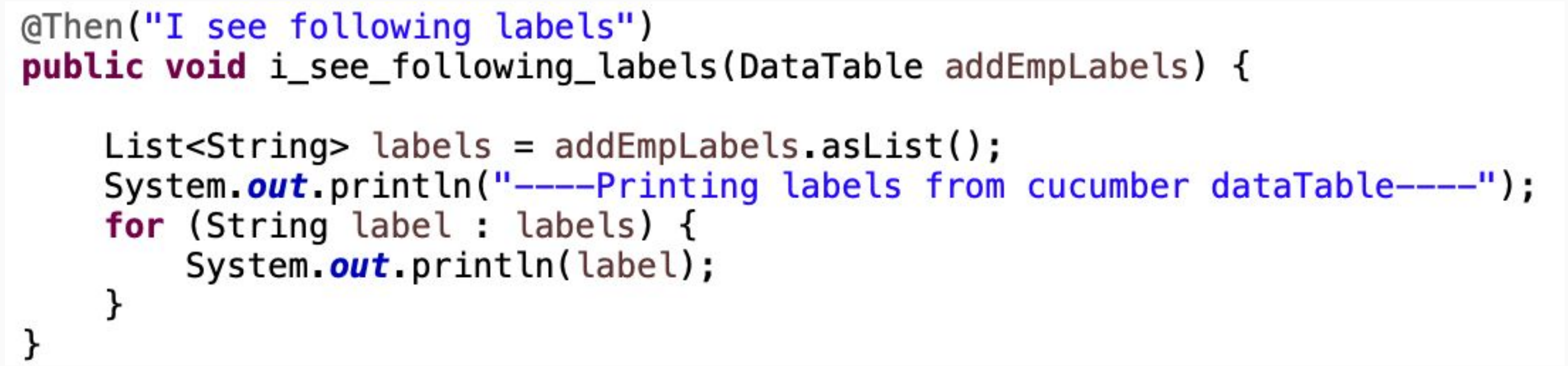
- Data table without a header
- Data table with a header

Data table without a header

The complete scenario is same as what we have done earlier. But the only difference is in this, we are not passing parameters in the step line and even we are not using Examples test data. We declared the data under the step only. So we are using Tables as arguments to Steps.



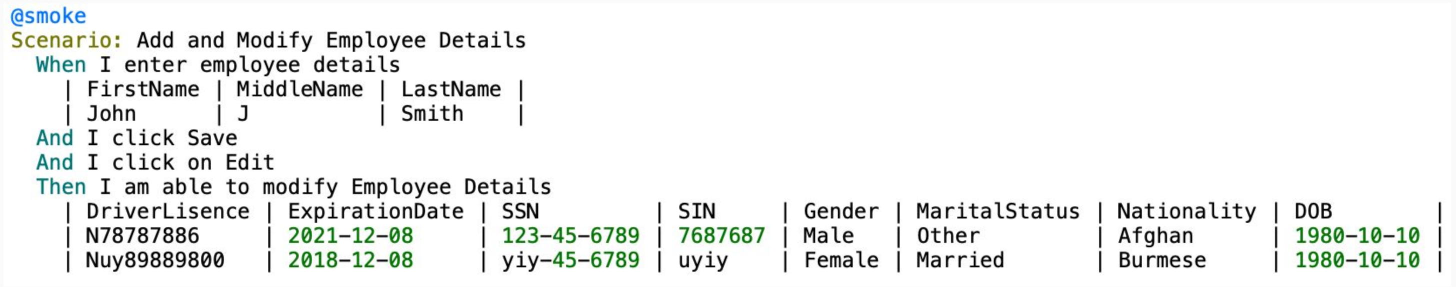
```gherkin
@temp
  Scenario: Add Employee Labels Verification
    Then I see following labels
      | First Name  |
      | Middle Name |
      | Last Name   |
      | Employee Id |
      | Location    |
```



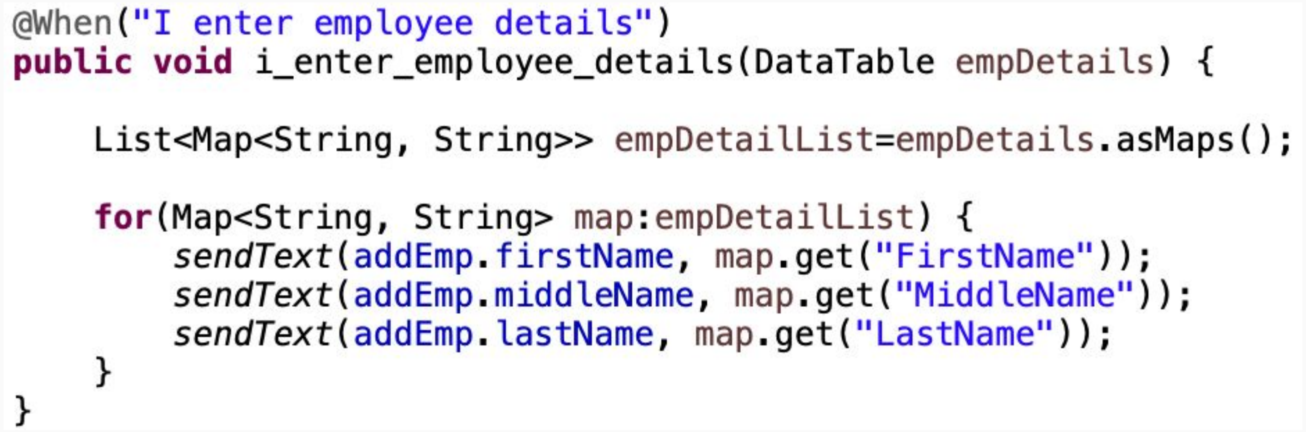
```java
@Then("I see following labels")
public void i_see_following_labels(DataTable addEmpLabels) {

    List<String> labels = addEmpLabels.asList();
    System.out.println("----Printing labels from cucumber dataTable----");
    for (String label : labels) {
        System.out.println(label);
    }
}
```

Data table with a header

We add the header to the Data table when we have more columns in the data table, so that we can improve the readability of our cucumber scenarios and scripts. Whenever we have multiple columns of test data present the best approach will be to use of Maps in Data Tables.



```
@smoke
Scenario: Add and Modify Employee Details
  When I enter employee details
    | FirstName | MiddleName | LastName |
    | John      | J          | Smith    |
  And I click Save
  And I click on Edit
  Then I am able to modify Employee Details
    | DriverLisence | ExpirationDate | SSN         | SIN     | Gender | MaritalStatus | Nationality | DOB        |
    | N78787886     | 2021-12-08     | 123-45-6789 | 7687687 | Male   | Other         | Afghan      | 1980-10-10 |
    | Nuy89889800   | 2018-12-08     | yiy-45-6789 | uyiy    | Female | Married       | Burmese     | 1980-10-10 |
```



```java
@When("I enter employee details")
public void i_enter_employee_details(DataTable empDetails) {

    List<Map<String, String>> empDetailList=empDetails.asMaps();

    for(Map<String, String> map:empDetailList) {
        sendText(addEmp.firstName, map.get("FirstName"));
        sendText(addEmp.middleName, map.get("MiddleName"));
        sendText(addEmp.lastName, map.get("LastName"));
    }
}
```

Maven Lifecycles

Maven has the following three standard lifecycles:

- clean Lifecycle - involves cleaning of the project. Deletes all artifacts and targets which are created already. It handles everything related to removing temporary files from the output directory, including generated source files, compiled classes, previous JAR files etc. It has 3 phases: pre-clean, clean, post-clean.

- default Lifecycle (build lifecycle) - handles the complete deployment of the project (has 23 phases)

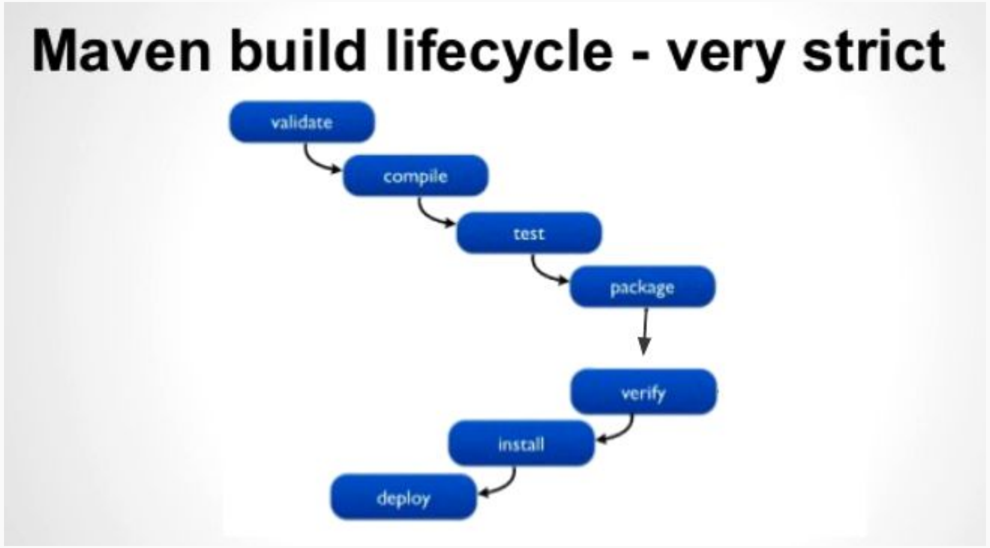
- site Lifecycle - handles generating the java documentation of the project. In fact, the site can generate a complete website with documentation for your project. It has 4 phases: pre-site, site, post-site, site-deploy.

Build Life Cycle

A Build Lifecycle is a sequence of tasks we used to build a software. For example, compile, test, package and publish or deploy are all tasks we need to do to build a software.

A Maven build lifecycle is a sequence of phases we need to go through in order to finishing building the software.

The following table lists some of the build lifecycle.



Build Life Cycle

Basic maven phases:

validate: validate the project is correct and all necessary information is available.

compile: used to compile the source code of the project.

test: test the compiled code and these tests do not require to be packaged or deployed.

package: package is used to convert your project into a jar or war etc.

verify: run any checks to verify the package is valid and meets quality criteria.

install: install the package into the local repository for use of another project.

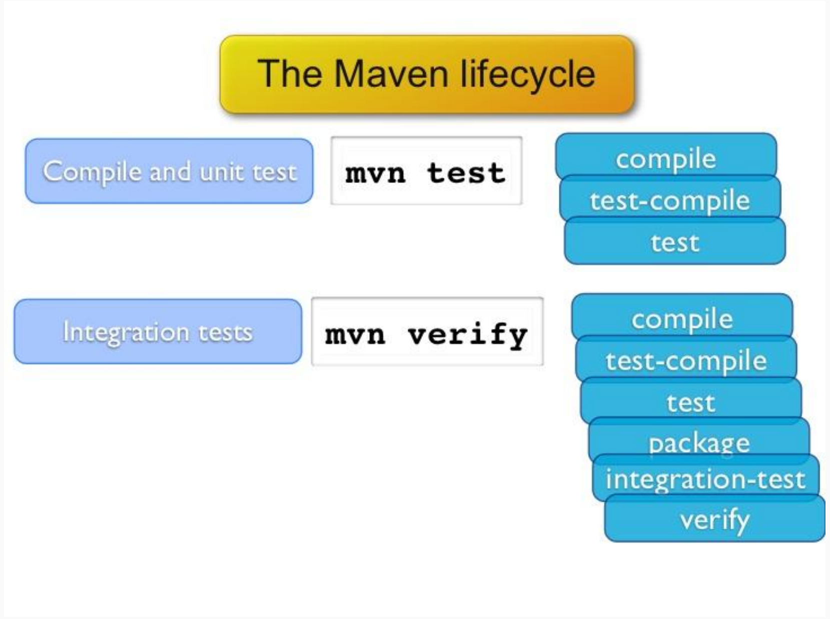
deploy: publish your packaged jar file into remote repo

Maven Commands

Basic maven commands:

mvn clean : it deletes all class files, the java docs, the jars, reports and so on)

mvn test: run all the unit or testNG test classes.


The Maven lifecycle
Compile and unit test
mvn test
compile
test-compile
test
Integration tests
mvn verify
compile
test-compile
test
package
integration-test
verify