**What is Cucumber?**

* Cucumber is the BDD (Behavior Driven Development) framework thru which we can design and run automation test cases
* Cucumber does not automate your test cases
* We can implement different features using Cucumber BDD like Parameterization, Execution control, Hooks, Reports, Automation Utilities, Data Driven tests, etc.
* Cucumber BDD framework supports different Automation test cases such as Web , Mobile, API, Unit Testing which are written in Java/Ruby language.
* Here, Test Cases/Requirements are defined with BDD methodology (using Gherkin syntax)
* No much coding is required to implement Framework functionalities

**What is Gherkin?**

* It is a language to describe software’s behavior
* It’s Business Readable, Domain specific language
* In Testing point of view, it’s language to define the test cases
* Statements written in plain English language can be interpreted differently by different people and it may lead to confusion and mistakes. To overcome this problem, Gherkin is introduced to describe behavior of software so that no one will be confused if everybody uses same language or template

**Keywords used in Cucumber:**

* **Scenario**
  + In cucumber, Test case is called as ‘Scenario’
  + And this test case or scenario is written in Gherkin language
  + Scenario contains Steps (which are nothing but test case steps)
  + Scenario uses following keywords (case sensitive):
    - **Given**
      * Pre-conditions are mentioned under Given
    - **When**
      * Here, describes user actions
    - **Then**
      * To observe expected output
      * The Observation should be related to business value
    - **And**
      * If there are multiple pre-conditions, actions or expected outputs, we use ‘And’ keyword to add them
      * It is used when there are additional pre-conditions, actions or expected outputs representing Positive statement
    - **But**
      * If there are multiple pre-conditions, actions or expected outputs, we use ‘But’ keyword to add them
      * It is used when there are additional pre-conditions, actions or expected outputs representing Negative statement
  + E.g. **Scenario**: Make minimum due payment

**Given** User is on Pay Credit Card Bill page

**When** User enters all required details and selects minimum amount option

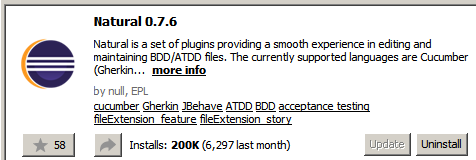
**And** user clicks on Pay button

**Then** Credit Card payment confirmation page displayed

**But** Error page is not displayed

* **Feature**
  + Feature represents Business Requirement
  + Feature File acts as a Test Suite which contains of all Scenarios
  + It’s high level business requirement; under this, we can design multiple Scenarios to test that a business requirement
  + In cucumber, feature file contains scenarios
  + We can create feature file with “.feature” extension (similar to java class in Java)
  + The text that immediately follows ‘Feature’ keyword in the same line, that text is the Title of the Feature File
  + The naming conventions for Feature file should be lower case with .feature extension
  + Feature file should contain either scenario or scenario outline
* **Feature File**
  + All the scenarios (in Given/when/then format) we write in a Feature File
  + “.feature” is the extension of feature file
  + Scenarios (test cases) we automate using Step Definition and Test Runner
* **Scenario Outline**
* **Step Definition**
  + While creating automation code for the scenario in a feature file, we map each line in the Step Definition
  + \*\*StepDefinition file matched with TagName and Descriptions in Feature file\*\*
  + \*\*There should be one mapping implementation for each Gherkin line\*\*

**Installing Cucumber:**

* We will add/install Cucumber plugin into **eClipse** editor
* For that, we have to first install Java, Eclipse editor for Java, set corresponding environment variables
* And then **install Cucumber plugin** into eClipse
* Goto Eclipse > Help > Eclipse Marketplace > search for ‘Natural’ plugin; which is used for maintaining/editing BDD files and supports Cucumber(Gherkin language)
* Install ‘Natural’ plugin
* 

**Creating Cucumber Project:**

* We can create Cucumber Project in eclipse once Cucumber plugin (‘Natural’) is installed
* Create Maven Projects in eClipse with **‘maven-archetype-quickstart’** project template; since this maven project template supports Cucumber BDD with Gherkin.

**Dependencies for Cucumber:**

* Add following dependencies in POM.xml after creating Project (you can find them in Maven Repository site)
  + Cucumber-Java
  + Cucumber-Junit/Cucumber-testng
* Install ‘Natural’ plugin; we already did it in above steps (installing Cucumber)

**Components used in Cucumber Framework:**

* **Feature File**
  + We already saw it
* **Step Definition**
  + We already saw it
* **Test Runner**
  + We use Test Runner to run the automation code
  + We have to provide Feature file and corresponding Step Definition info to Test Runner

**Feature File Creation:**

* Create a feature file under ‘src/test/java’ folder
* Create a new Package (here I created ‘com.FeatureFiles’)
* Under the package, create a simple/plain file with **extension .feature** (here I created ‘LoginToApp.feature’)
* And start writing scenarios in there as per given syntax:

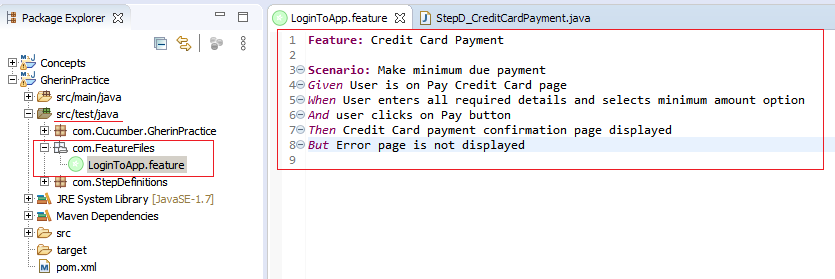
Feature: <feature name or business case>

Scenario: <scenario name>

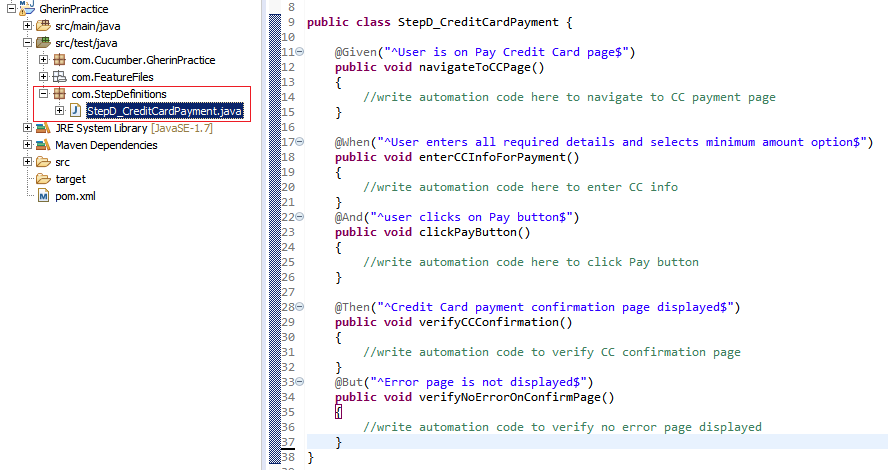
Given <pre-conditions>

When <actions to be performed>

Then <expected output>

* Here, note that “:” is given after ‘Feature’ and ‘Scenario’ only
* 

**Creation of Step Definition File:**

* Create another Package under ‘src/test/java’ folder
* Create a new Package (here I created ‘com.StepDefinitions’)
* Create a .java file and implement Step Definitions there for Feature file
* 
* As an example, Feature line “*Given* User is on Pay Credit Card page” is implemented like

@Given("^User is on Pay Credit Card page$")

**public** **void** navigateToCCPage()

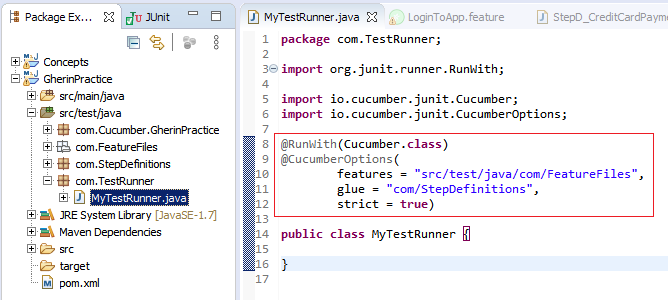
{

//write automation code here to navigate to CC payment page

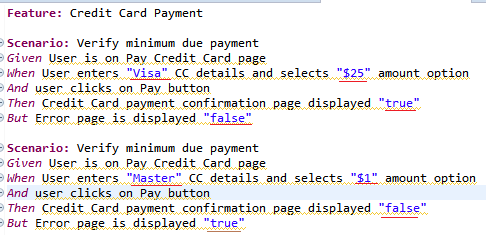
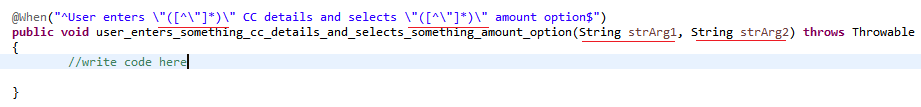
}

* Feature line is written between “^” and “$” symbols
* This line content should be same as it is in feature file
* \*\*StepDefinition file matched with TagName and Descriptions in Feature file\*\*
* \*\*There should be one mapping implementation for each Gherkin line\*\*
* You can have multiple similar tags i.e. multiple @Given, or @Then; however, it get’s executed bases on feature file whatever founds in there, will execute that piece of code in step definition file
* There is another way to create Step Definition File using ‘**Tidy Gherkin’** chrome plugin
* If you have install it as an app in Chrome, the goto <https://apps> in chrome
* Open “Tidy Gherkin” and copy the Feature file content there and select Java tab and you will get Step Definition
* Only the thing that you need to change/modify Package name, class name and method bodies according to your actual package, class names and method bodies
* There is one more way to generate the Step Definition code:
* Let’s consider you just created a Step Definition java file with nothing into the class block.
* And you have created Feature file and Test Runner file
* Then when you run the Test Runner file, your eclipse output console shows you the expected or missing step Definition code below the message “You can implement missing steps with the snippets below:”
* You can copy the code from there and paste it into Step Definition class block.
* Of course, you need to change method bodies according to your need

**Creating Test Runner File:**

* Create another Package
* Create a new Java File
* As a best practice, This test runner’s package and StepDefinition file’s package be placed in same folder; here we have placed both under ‘src/test/java’ folder
* 
* Above highlighted code is must into this Test Runner file to make it a test runner
* Note that, this piece of code is added above Class block.
* Into @CucumberOptions annotation, there are two important elements:
  + “**feature**”
    - Provide the path of Package where .feature file is located
    - Here, you can give that specific .feature file path too
    - If package path provided, meaning all the feature files will be executed under this package
  + “**glue**”:
    - Provide the Package name where StepDefinition file is located
* Now if you simple run this file, ‘StepD\_CreditCardPayment.java’ will be executed.

**Simple Regular Expression:**

* When we have to provide different data/dynamic data for same implementation, we use Regular expression. Let’s see below example
* 
* Here are two scenarios, almost all steps are same except some input values (and expected outputs). Note that, expected steps are slightly modified to avoid redundant step definition code; now we can use same code with these dynamic values
* For that we don’t have to create another Step Definition steps for another scenario
* We can use same step definition for both scenarios
* Here, Cucumber understand that if the text is inside double quotes “”, meaning these are dynamic values
* When we generate step definition in ‘Tidy’ chrome plugin, it automatically generate regular expression at the place of “xxxx” values; see in below screenshot
* That regular expression meaning, code can accept any value
* 

@When("^User enters \"([^\"]\*)\" CC details and selects \"([^\"]\*)\" amount option$")

**public** **void** user\_enters\_something\_cc\_details\_and\_selects\_something\_amount\_option(String strArg1, String strArg2) **throws** Throwable

{

//write code here

}

* Here, first dynamic string i.e. “visa” will go inside String variable ‘strArg1’ and second dynamic value will go inside ‘strArg2’
* You can use these variables into the corresponding method block

**Cucumber-Junit Framework (API Testing):**

To start with this framework development, we should have knowledge of followings:

* Rest Assured API Testing
  + Rest Assured Testing Basics
  + Serialization and De-Serialization using POJO classes
  + Request and Response Spec Builder
* Cucumber Framework
  + Understanding of Cucumber Framework
* Maven
  + Maven – build Management tool basics

Now we will create Cucumber-Junit framework for API Testing.

Create Project and add dependencies:

* Create a Maven Project in Eclipse IDE (with ‘maven-archetype-quickstart’ template)
* Add followings dependencies in Pom.xml
  + **Cucumber-java**
  + **Cucumber-junit**
  + **Junit**
  + **Rest-assured (from io.rest-assured)**
  + **Jackson-databind (along with below two dependencies)**
    - **Jackson-core**
    - **Jackson-annotations**
  + **Gson**
* Note: Jackson-\* jars should have of same version
* Also add below dependencies ( JARs) from Rest-Assured.io
  + **Rest-assured…**
  + **Json-path…**
  + **Xml-path…**
  + **Json-schema-validator…**
  + Spring-moc-mvc…
  + Scala-support…
  + Kotlin-extensions…
  + Spring-web-test-client…
* Make sure following plugins are installed in Eclipse IDE (Help > Eclipse Marketplace)
  + **Natural**
* **Add static Packages** manually in your class(.java) file
  + **import static io.restassured.RestAssured.\*;**
  + **import static org.junit.Assert.\*;**
* NOTE: If added dependencies in pom.xml not getting downloaded or added to your project, then check eclipse menu Project>Build Automatically. This option should be selected.
* If you goto eClipse Project > right click > Properties >Java Build Path > Libraries; and if you see Junit library outside of ‘Maven Dependencies’ folder, then **Delete it**; since we already have added it into Maven Dependencies (pom.xml). Having duplicate Junit entries or multiple junit versions, may create some problem while running the program.
* Also, Goto eclipse menu >Windows >Preferences > Java > Editor > Typing > “Escape Text when pasting into string literal”.. select this option.

This setting will be useful when you paste a JSON file body into string double quote( “”).

Create ‘Feature’ files:

* Add a separate Package; here, we named it as ‘features’
* Add a file into this package and give the name with extension ‘.feature’; here, we named it as ‘placeValidations.feature’