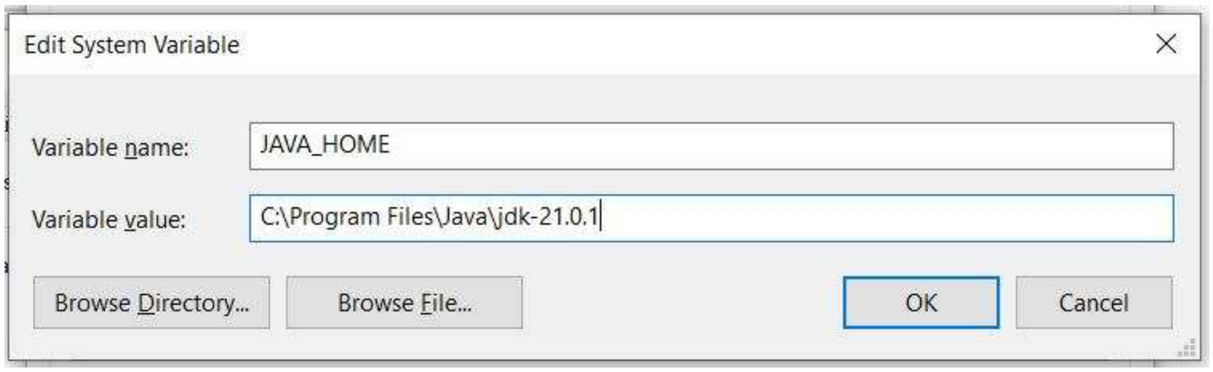
Cucumber - Java Testing

Prerequisite:

- Cucumber
- Eclipse
- Junit5

Step 1 − Install Java −

1. *Download jdk and jre from http://www.oracle.com/technetwork/java/javase/downloads/index.html*
2. *Accept license agreement. Install JDK and JRE.*
3. *Make sure that your environment variable as shown in the following picture.*



Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk-21.0.1

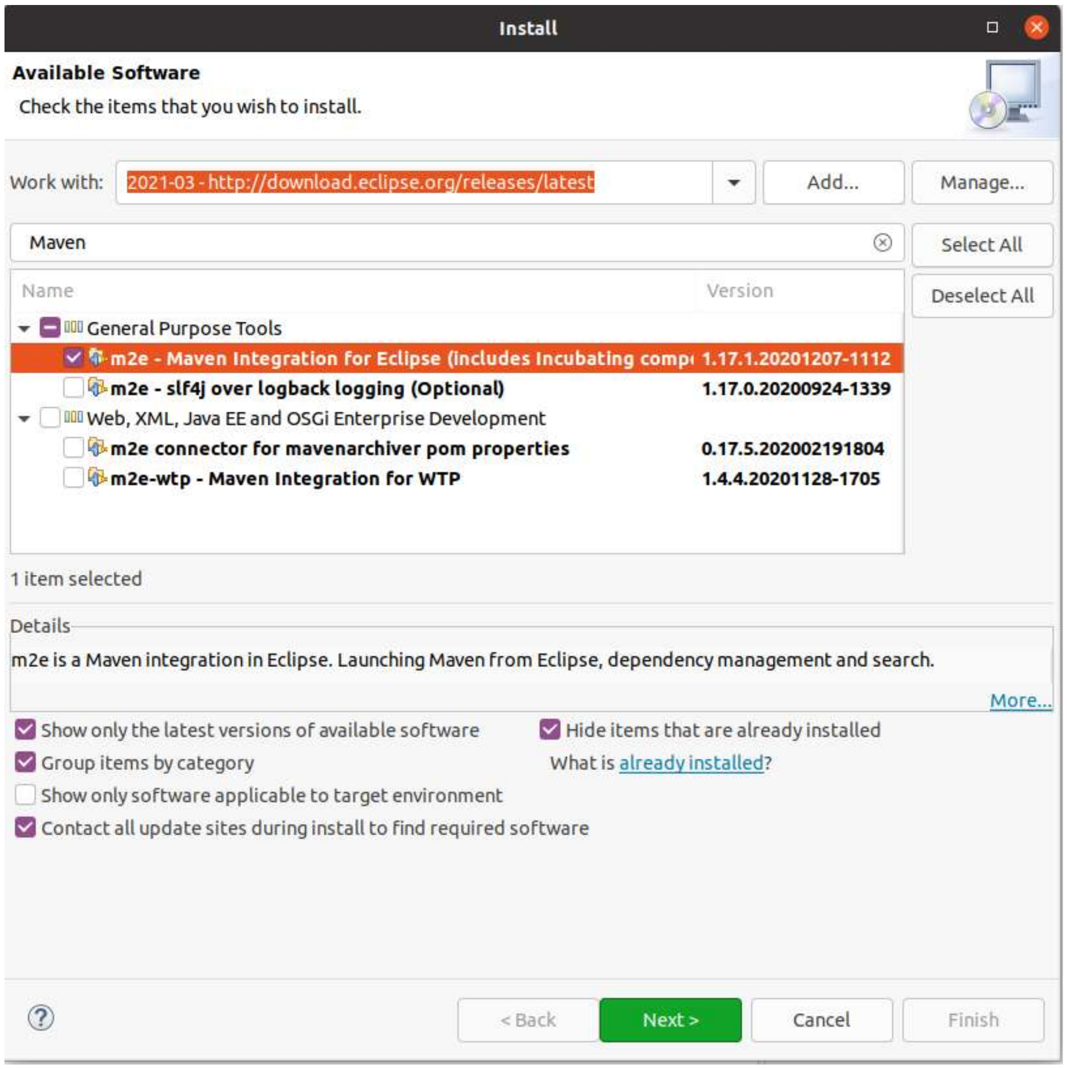
Step 2 − Install Eclipse IDE –

1. *Download Eclipse from https://eclipse.org/downloads/*
2. *Unzip and Eclipse installed.*

Step 3 – Install Maven –

Most Eclipse IDE downloads already include support for the Maven build system. To check, use **Help** > **About** and check if you can see the Maven logo (with the M2E) sign. If Maven support is not yet installed, the following description can be used to install it.

1. *Open Eclipse.*
2. *Got to Help → Eclipse Marketplace → Search maven → Maven Integration for Eclipse →INSTALL.*



Install

Available Software

Check the items that you wish to install.

Work with: 2021-03 - http://download.eclipse.org/releases/latest

Maven

Name	Version
☐ General Purpose Tools	
☑ m2e - Maven Integration for Eclipse (includes Incubating comp	1.17.1.20201207-1112
☐ m2e - slf4j over logback logging (Optional)	1.17.0.20200924-1339
☐ Web, XML, Java EE and OSGi Enterprise Development	
☐ m2e connector for mavenarchiver pom properties	0.17.5.202002191804
☐ m2e-wtp - Maven Integration for WTP	1.4.4.20201128-1705

1 item selected

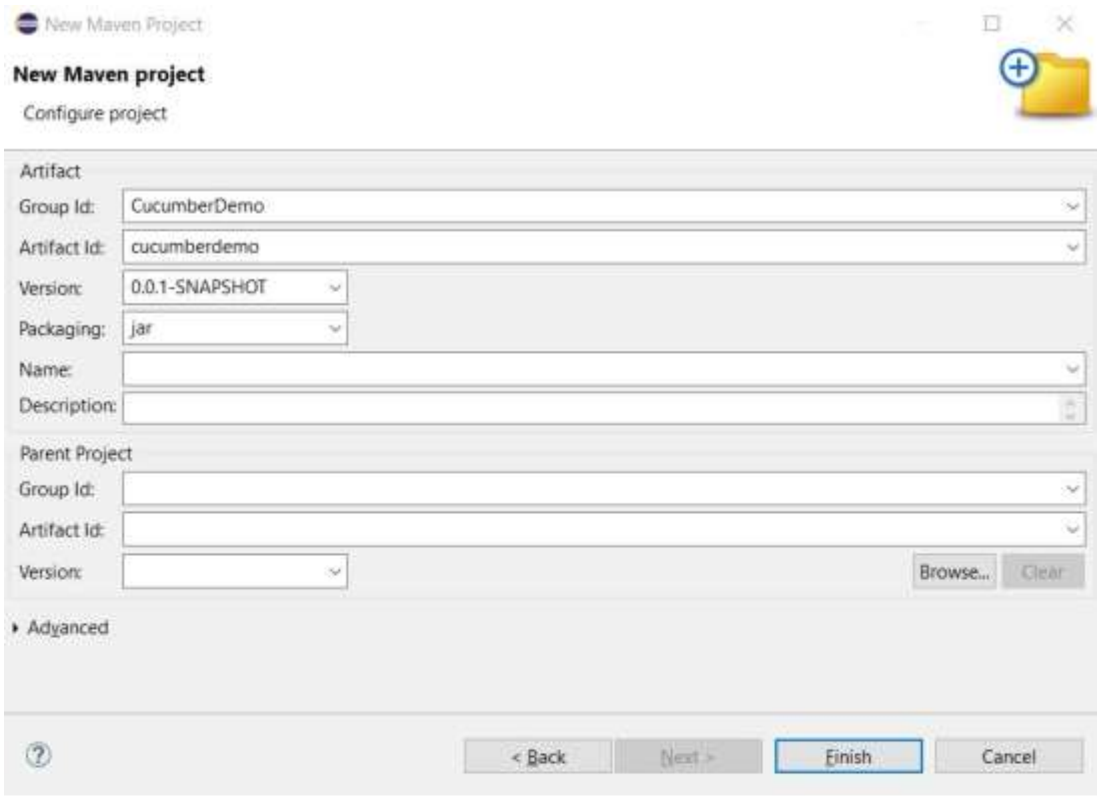
Details

m2e is a Maven integration in Eclipse. Launching Maven from Eclipse, dependency management and search.

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

Step 4 – Create Maven project.

1. *Go to File → New → Others → Maven → Maven Project.*
2. *For simplicity (no archetype), select "Create simple project" to skip the archetype selection.*
3. *Provide group Id (group Id will identify your project uniquely across all projects).*
4. *Provide artifact Id (artifact Id is the name of the jar without version. You can choose any name which is in lowercase).*
5. *Click on Finish.*



New Maven project

Configure project

Artifact

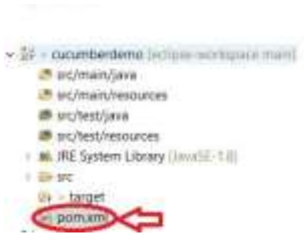
Group Id:	CucumberDemo
Artifact Id:	cucumberdemo
Version:	0.0.1-SNAPSHOT
Packaging:	jar
Name:	
Description:	

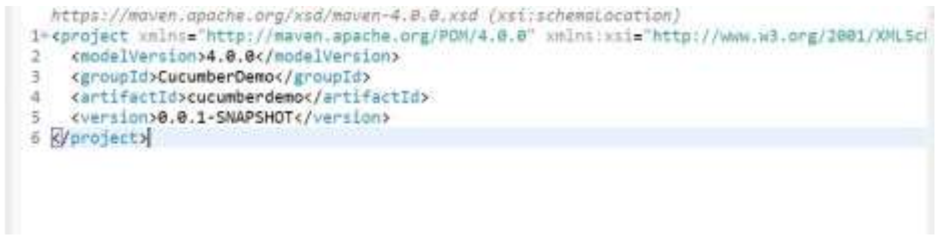
Parent Project

Group Id:	
Artifact Id:	
Version:	

Step 5 – Locate pom.xml –

1. *Go to the package explorer on the left hand side of Eclipse*
2. *Expand you project.*
3. *Locate pom.xml file.*
4. *Double click on pom.xml or Right-click and select the option, Open with "Text Editor".*





```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSc
  <modelVersion>4.0.0</modelVersion>
  <groupId>CucumberDemo</groupId>
  <artifactId>cucumberdemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Step 6 – Add dependency for JUnit5

JUnit5 provides a modern **foundation for developer-side testing** on the JVM.

1. *Open pom.xml is in edit mode, create dependencies tag (<dependencies> </dependencies>), inside the project tag. Inside the dependencies tag, create dependency tag (<dependency> </dependency>)*
2. *Provide the following information within the dependency tag will indicate Maven, which JUnit5 jar files are to be downloaded from the central repository to the local repository*

```
<dependencies>
	<dependency>
		<groupId>org.junit.jupiter</groupId>
		<artifactId>junit-jupiter-engine</artifactId>
		<version>5.9.1</version>
		<scope>test</scope>
	</dependency>
	<dependency>
		<groupId>org.junit.platform</groupId>
		<artifactId>junit-platform-suite</artifactId>
		<scope>test</scope>
		<version>1.10.2</version>
	</dependency>
</dependencies>
```

*****Note****: This tutorial uses Junit5 with the latest version for Maven project . If you want to use other versions, search them out on “https://mvnrepository.com/”*

Step 7 – Add dependency for Selenium and Webdriver

Selenium and Webdriver are necessary for **automating web applications for testing** purposes**.**

1. *Create one more dependency tag.*
2. *Provide the following information within the dependency tag.*

*****Note****: This tutorial uses the current version of Selenium for Maven project. If you want to check other versions, search out on: “https://mvnrepository.com/”*

```
<dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>4.17.0</version>
</dependency>
<dependency>
        <groupId>io.github.bonigarcia</groupId>
        <artifactId>webdrivermanager</artifactId>
        <version>5.6.3</version>
</dependency>
```

Step 8 – Add dependency for Cucumber-Java

This tutorial is for integrating **Cucumber** with **JUnit5** only, if you want to use JUnit4 please referring to the following link "https://cucumber.io/docs/installation/java/"

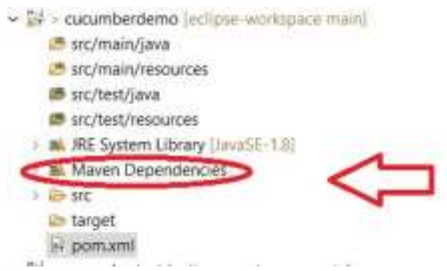
1. Create one more dependency tag.
2. Provide the following information within the dependency tag.

```
<dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>7.15.0</version>
</dependency>
<dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-junit-platform-engine</artifactId>
        <scope>test</scope>
        <version>7.15.0</version>
</dependency>
<dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>7.15.0</version>
        <scope>test</scope>
</dependency>
```

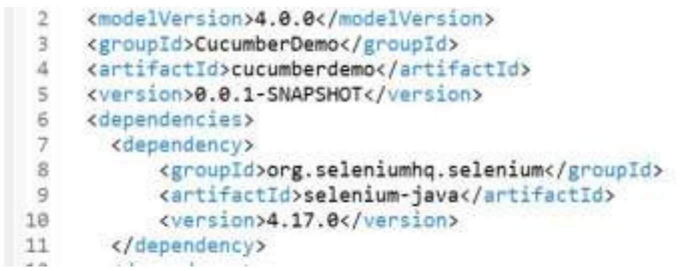
**Note*: This tutorial uses the current version of Cucumber for Maven project. If you want to check other versions, search out on: “https://mvnrepository.com/”*

Step 9 − Verify binaries

1. *Once pom.xml is edited successfully, save it.*
2. *Go to Project → Clean − It will take a few minutes.*
3. *You will be able to see a Maven repository.*



```
cucumberdemo [eclipse-workspace main]
  src/main/java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

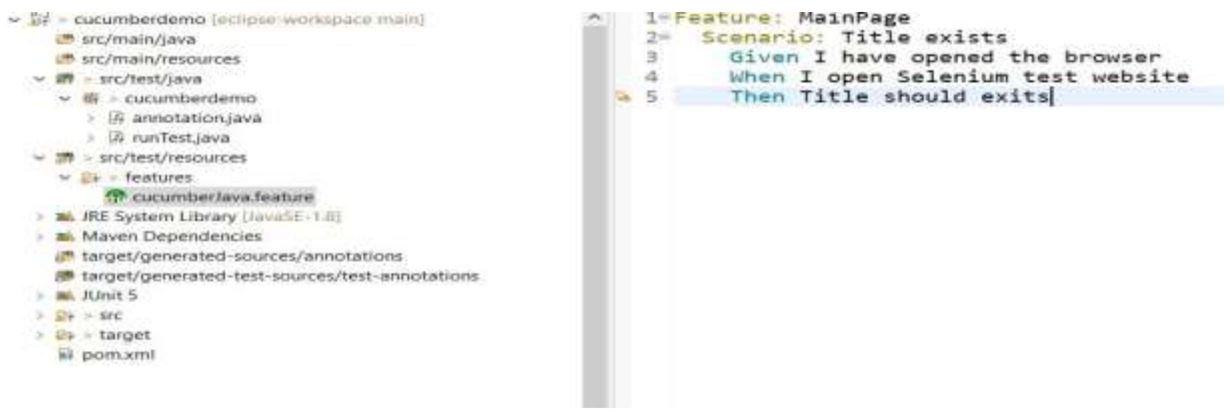


```
<modelVersion>4.0.0</modelVersion>
<groupId>CucumberDemo</groupId>
<artifactId>cucumberdemo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
  <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.17.0</version>
  </dependency>
```

Step 10 − Create feature file

1. *Create a package Under ‘**src/test/resources**’ named as ‘**features**’*
2. *Select and right-click on the package outline.*
3. *Click on ‘New’ file. Give the file a name such as **cucumberJava.feature**.*
4. *Write the following text within the file and save it.*

```
Feature: MainPage
 Scenario: Title exists
       Given I have opened the browser
       When I open Selenium test website
       Then Title should exits
```



```
cucumberdemo [eclipse-workspace main]
  src/main/java
  src/main/resources
  src/test/java
    cucumberdemo
      annotation.java
      runTest.java
  src/test/resources
    features
      cucumberJava.feature
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  JUnit 5
  src
  target
  pom.xml
```

```
Feature: MainPage
 Scenario: Title exists
   Given I have opened the browser
   When I open Selenium test website
   Then Title should exits
```

Step 11 − Create step definition file –

1. *Create a package under 'src/test/java' named as 'cucumberdemo'*
2. *Select and right-click on the package outline.*
3. *Click on 'New' then 'Class'.*
4. *Give the file name a name such as 'annotation'.*
5. *Pick **'Public'** modifier .*
6. *Write the following text within the file and save it.*

*Note: Step definition class must be in **Public** or Cucumber will not be able to find these step and result in "Undefined error".

```
package cucumberdemo;

import static org.junit.Assert.assertNotNull;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.github.bonigarcia.wdm.WebDriverManager;

public class annotation {

        WebDriver driver;
        @Given("I have opened the browser")
        public void openbrowser()
        {
                WebDriverManager.chromedriver().setup();
          driver=new ChromeDriver();
        }
        @When("I open Selenium test website")
        public void openwebsite()
        {
                driver.get("https://www.selenium.dev/selenium/web/web-form.html");
        }
        @Then ("Title should exits")
        public void test1pageTitle()
        {
                String at=driver.getTitle();
                assertNotNull(at);
                driver.close();
        }
}
```

Step 12 − Create a runner class file.

1. *Select and right-click on the package outline.*
2. *Click on 'New' file.*
3. *Give the file name as runTest.java.*
4. *Write the following text within the file and save it.*

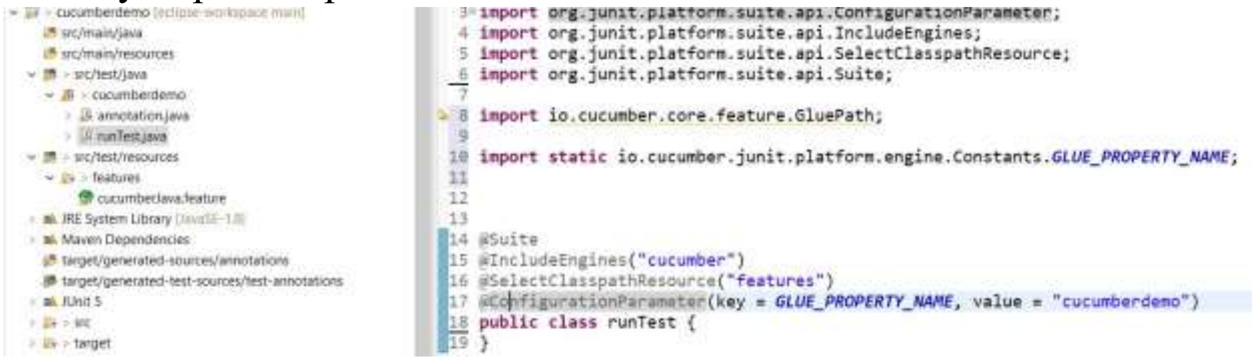
```
package cucumberdemo;

import org.junit.platform.suite.api.ConfigurationParameter;
import org.junit.platform.suite.api.IncludeEngines;
import org.junit.platform.suite.api.SelectClasspathResource;
import org.junit.platform.suite.api.Suite;
import static io.cucumber.junit.platform.engine.Constants.GLUE_PROPERTY_NAME;

@Suite
@IncludeEngines("cucumber")
@SelectClasspathResource("features")
@ConfigurationParameter(key = GLUE_PROPERTY_NAME, value = "cucumberdemo")
public class runTest {
}
```

*Note: This tutorial is for JUnit5 therefore runner file is vast difference from JUnit4. More detail as below:

- @Suite: annotation is used for JUnit5 only, use @RunWith if you want to implement them in JUnit4.
- @IncludeEngines("cucumber"): specifically request running on ‘cucumber’ engine. (JUnit5 only)
- @SelectClasspathResource("features"): instruct cucumber where to find ‘feature’ file (Name of the package)
- @ConfigurationParameter(key = ***GLUE_PROPERTY_NAME***, value = "cucumberdemo"): is used to config others parameters, the most important parameter is ‘GluePath’ (by the constant ‘***GLUE_PROPERTY_NAME’)*** which define the name of package where you put “step definition file”.



```
cucumberdemo [eclipse-workspace main]
  src/main/java
  src/main/resources
  src/test/java
    cucumberdemo
      annotation.java
      runTest.java
  src/test/resources
    features
      cucumberJava.feature
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  JUnit 5
  src
  target
```

```
import org.junit.platform.suite.api.ConfigurationParameter;
import org.junit.platform.suite.api.IncludeEngines;
import org.junit.platform.suite.api.SelectClasspathResource;
import org.junit.platform.suite.api.Suite;

import io.cucumber.core.feature.GluePath;

import static io.cucumber.junit.platform.engine.Constants.GLUE_PROPERTY_NAME;



@Suite
@IncludeEngines("cucumber")
@SelectClasspathResource("features")
@ConfigurationParameter(key = GLUE_PROPERTY_NAME, value = "cucumberdemo")
public class runTest {
}
```

Step 13 − Run the test

1. *Select runTest.java file from the package explorer.*
2. *Right-click and select the option, Run as.*
3. *Select JUnit test.*

This is the minimum we need to make the scenario pass, but we may need a more complex and more flexible structure. Let's update our scenario to use variables and evaluate more possibilities.

Step 14 − Using variables

Let's go back to our scenario and update the 'cucumberJava.feature' file. When Cucumber executes a Gherkin step in a scenario, it will look for a matching *step definition* to execute. For example, The "**I have 48 cakes in my belly**" part of the step will match the following step definition "**I have {int} cakes in my belly**". In our case, we need an URL and an expected result.

```
Feature: Title
I want to check the title of a website provided I have its url
  Scenario: What is the title
        Given I have opened the browser
        When I open this url "https://www.selenium.dev/selenium/web/web-form.html"
        Then I should receive its title as "Web form"
```

Since we change our scenario, don't forget to update
the 'annotation.java' file

```
package cucumberdemo;

import static org.junit.jupiter.api.Assertions.*;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.github.bonigarcia.wdm.WebDriverManager;

public class annotation {
            WebDriver driver;
            @Given("I have opened the browser")
            public void openbrowser()
            {
                        WebDriverManager.chromedriver().setup();
               driver=new ChromeDriver();
            }
            @When("I open this url {string}")
            public void openwebsite(String url)
            {
                        driver.get(url);
            }
            @Then ("I should receive its title as {string}")
            public void test1pageTitle(String title)
            {
                        String at=driver.getTitle();
                        assertEquals(title, at);
                        driver.close();
            }
}
```

Step 15 − Using And

"And" keyword is used to add conditions to your steps. Let's look at it by modifying our feature a little:

```
Feature: Title
I want to check the title of a website provided I have its url
  Scenario: What is the title
            Given I have opened the browser
            When I open this url "https://www.google.com/"
            Then I should receive its title as "Google"
  Scenario: Where is the Index page
            Given I have opened chrome
            When I go to Selenium "https://www.selenium.dev/selenium/web/web-form.html"
            And I click on the link "Return to index"
            Then I should be at index "https://www.selenium.dev/selenium/web/index.html"
            And The index title should be "Index of Available Pages"
```

Since we change our scenario, don’t forget to update the ‘annotation.java’ file

```
package cucumberdemo;

import static org.junit.jupiter.api.Assertions.*;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.github.bonigarcia.wdm.WebDriverManager;

public class annotation {
	WebDriver driver;
	@Given("I have opened the browser")
	public void openbrowser()
	{
		WebDriverManager.chromedriver().setup();
	   driver=new ChromeDriver();
	}
	@When("I open this url {string}")
	public void openwebsite(String url)
	{
		driver.get(url);
	}
	@Then ("I should receive its title as {string}")
	public void test1pageTitle(String title)
	{
		String at=driver.getTitle();
		assertEquals(title, at);
		driver.close();
	}
	@Given("I have opened chrome")
	public void openChrome()
	{
	   driver=new ChromeDriver();
	}
```

```
        @Given("I go to Selenium {string}")
        public void openTestLink(String seleniumTestSite)
        {
                driver.get(seleniumTestSite);
        }
        @Given("I click on the link {string}")
        public void gotoIndex(String indexLink)
        {
                driver.findElement(By.linkText(indexLink)).click();
        }
        @Then("I should be at index {string}")
        public void atIndexPage(String indexTitle)
        {
                assertEquals(indexTitle, driver.getCurrentUrl());
        }
        @Then("The index title should be {string}")
        public void indexTitle(String titleValue)
        {
                assertEquals(titleValue, driver.getTitle());
                driver.close();
        }
}
```

Step 16 − Using Scenario Outline and Data Tables

Data Tables are handy for passing a list of values to a step definition:

```
Feature: Title
I want to check the title of a website provided I have its url
 Scenario Outline: What is the title
        Given I have opened the browser
        When I open this url "<mlink>"
        Then I should receive its title as "<mtitle>"
 Examples:
  | mlink                                                   | mtitle       |
  | https://www.google.com/                                 | Google       |
  | https://www.selenium.dev/selenium/web/web-form.html     | Web form     |
 Scenario: Where is the Index page
        Given I have opened chrome
        When I go to Selenium "https://www.selenium.dev/selenium/web/web-form.html"
        And I click on the link "Return to index"
        Then I should be at index "https://www.selenium.dev/selenium/web/index.html"
        And The index title should be "Index of Available Pages"
```