Cucumber no repara en la palabra reservada para hacer el mapeo. Solo se fija en el texto con sus respectivas expresiones regulares. Por lo tanto:

Given Kathy is **100** meters from the coffee shop

Es lo mismo que

Given Kathy is **100** meters from the gas station

En tiempo de ejecución.

Entonces: **Given**, **Then**, **When**, **And** y **But** son intercambiables y su único motivo es hacer **más legible** los escenarios desde el punto de vista del negocio.

Feature: Order a coffe  
  
 In order to save time when I pick up my morning coffee  
 As a coffe lover  
 I want to be able to order a coffee in advance  
  
 Scenario: Buyer orders a coffee when they are close to de coffee shop  
 Given Kathy is **100** meters from the coffee shop  
 When Kathy orders a **large cappuccino** Then Barry should reiceive the order  
 And Barry should know that the coffee is Urgent

@Given("^Kathy is (\\d+) meters from the coffee shop$")

@When("^Kathy orders a (.\*)$")

@Then("^Barry should reiceive the order$")

@Then("^Barry should know that the coffee is Urgent$")

**Matcheos:**

Given Kathy is **100** meters from the coffee shop

Matchea directamente con:

@Given("Kathy is 100 meters from the coffee shop")

Pero también matchea con:

@Given("Kathy is 100 meters from the gas station(or any other place”)

Y con cualquier expresión que solo tenga una porción del texto del primer Given.

A veces es el comportamiento que queremos pero la mayoría de las veces no. La solución es usar anchors o anclajes como ^ Y $

^ indica que la expresión solo matcheará con el texto si el texto está en el comienzo.

$ indica que allí donde la expresión termina, el texto del step debe terminar también.

**Como usar expresiones regulares para no implementar parámetros harcodeados:**

.\* es la expresión más simple para llegar a ello. Matchea cualquier cadena de caracteres:

@When("^Kathy orders a (.\*)$")

Matchea con:

When Kathy orders a **large cappuccino**

**large cappuccino** está en azul por que Cucumber tomará cualquier cadena como parámetro debido a la expresión .\*

El punto **.** representa cualquier carácter incluyendo blancos como espacios o tabs. Y el asterisco representa la repetición de lo primero, es decir: **large cappuccino without sugar** también sería admitido.

El asterisco \* es un modificador de repetición. Le dice a Cucumber cuantas veces un carácter en particular puede aparecer incluyendo CERO veces. -> When Kathy orders a también es admitido

Lo anterior cubre el 75-80% de los casos en que necesitaremos expresiones regulares. Pero puede haber un puñado de situaciones que veremos a continuación:

Given Kathy is **100** meters from the coffee shop

Matchea con:

Given("^Kathy is (.\*) meters from the coffee shop$")

Si el texto es como sigue:

Given Kathy is **10 minutes late and 100** meters from the coffee shop

También matcheará y tomara el 10 en lugar del 100 por que java toma el primer entero de la cadena e ignora el resto (recordar que una de las funciones espera un int como parámetro).

En este caso, Cucumber tomará 10, pero lo correcto es que tome 100. En ese caso, le decimos a Cucumber que tome solo dígitos (//d). La barra / le dice a Cucumber que es un símbolo especial pero se introducen dos barras // puesto que una sola es tomada por Java como un carácter de escape. Entonces ->

**//d** matchea con un solo dígito, ej: 7

**//d//d//d** matchea con 3 dígitos, ej: 143

**//d\*** matchea con cualquier cantidad de dígitos incluyendo CERO dígitos, ej: 143, 7 o vacío (ningún digito) cualquier número.

**//d+** matchea con cualquier cantidad de dígitos, pero al menos tiene que tenert un dígito.

Minuto 9:10

En resumen:

**.\*** matchea cualquier cadena incluyendo una vacía.

**.+** matchea cualquier cadena no vacía.

**//d** matchea un dígito.

**//d\*** matchea cero o más dígitos.

**//d+** matchea uno o más dígitos.

**//w** matchea cualquier caracter de “palabra” (alfanumértico o underscore(¿?)->los de debajo de las teclas…creo).

**//s** matchea cualquier carácter blanco (espacio, tab, etc).

De esta manera, podemos combinar expresiones para lograr que Cucumber tome cadenas como por ejemplo: 123.25 -> en este caso, la expresión será: //d+//.//d+ Esto necesitará obviamente que la función Java no acepte un (**int** distanceInMeters) sino un (**float** distanceInMeters).

Ahora supongamos que no solo queremos que Cucumber acepte a Kathy como personaje sino también a James o cualquier nombre.

Entonces,

Given Kathy is **100** meters from the coffee shop

@Given("^Kathy is (\\d+) meters from the coffee shop$")

Ahora también puede ser

Given **James** is **100** meters from the coffee shop

En cuyo caso,

@Given("^(.\*) is (\\d+) meters from the coffee shop$")

Y ahora tenemos dos parámetros para recibir:

public void kathy\_is\_n\_meters\_from\_the\_coffee\_shop(String name, int distanceInMeters)

Solo nos queda una sola cosa por aprender que hace a una mayor flexibilidad:

Supongamos que tenemos:

Given Kathy is **1** meters from the coffee shop

Sintacticamente está mal por que debería decir 1 meter y no 1 meters. Lo que se hace es agregar un ? al final de meters. Esto hace que la “s” tenga carácter de opcional. Entonces queda - >

@Given("^(.\*) is (\\d+) meters? from the coffee shop$")

Por ultimo, ocasionalmente necesitaremos en un escenario un set fijo de variables (como si fuera un listbox). Por ejemplo: Queremos separar el tipo del tamaño del café.

When Kathy orders a **large cappuccino**

Entonces queremos solo las opciones: **small**, **regular** y **large**.

Notar como queda el step junto a la función (Se agrega parámetro String size):

When("^Kathy orders a (small|regular|large) (.\*)$")  
public void kathy\_orders\_a(String size, String order){

IMPORTANTE: Es importantísimo el espacio entre large) (.\*) de lo contrario, la variable order toma el valor “ capuchino” y se traba el test.

Y ahora si, por ultimo: Podemos estar probando el tema del pedido. Tenemos:

When Kathy orders a **large cappuccino**

Pero en un escenario diferente, por ejemplo, en el delivery, donde la orden ya ha sucedido, podríamos decir:

Given Kathy has ordered a **large cappuccino**

Podemos entonces implementar esto mediante lo que se llama un non-capturing group.

Un non-capturing group es una expresión regular que no será parseada como ningún parámetro, pero agrega mayor flexibilidad (y legibilidad) al texto del escenario.

Queremos aceptar para ese escenario tanto “orders” como “has ordered”. Se logra como sigue:

@Given("^Kathy (?:orders|has ordered) a (small|regular|large)(.\*)$")

Obviamente, prestar atención a:

(?:orders|has ordered)

**IMPORTANTE**: Recordemos que @Given o @When, no es tomado en cuenta por Cucumber por lo que es perfectamente intercambiable. Esto nos da la posibilidad de tener varios escenarios escritos de manera diferente pero que apunten a la misma implementación. Vendria a ser como una sobrecarga de métodos.