

Curso de formación

# Pruebas de Software Pruebas de aceptación

- Introducción
- Gherkin
- Cucumber
- Karate

- **Introducción**
- Gherkin
- Cucumber
- Karate

- Según el **ISTQB** (*International Software Testing Qualifications Board*), las pruebas de aceptación (*acceptance tests*) son los tests formales para determinar si un sistema satisface los criterios de aceptación y permite a los usuarios determinar si aceptan el sistema o no.
- Son las pruebas realizadas por los usuarios en las **versiones beta** del software (**beta testing**)

- Según la **Agile Alliance**, una prueba de aceptación es una descripción formal del comportamiento de un software usando un **ejemplo o escenario**
- Existen diferentes notaciones para especificar los tests, pero siempre son de **alto nivel (no técnicas y que pueden usar por negocio)**, pero que se pueden **ejecutar**

- Ejemplo de prueba de aceptación

```
1 Feature: Login Page
2
3   As a user I want to be able to login and out of the system.
4
5 Background:
6   Given I am at the login page
7
8 Scenario: Login as a user
9   When I login with user 'janedoe' and password 'password'
10  Then I should see the home page
11  And I can logout
```

- Beneficios
  - Fomenta la colaboración entre desarrolladores con usuarios, clientes o expertos del dominio porque los **requisitos** se deben expresar como un **contrato no ambiguo**
  - Un producto que pasa las pruebas de aceptación será considerado **adecuado** (aunque los usuarios pueden refinar las pruebas/ejemplo o sugerir nuevos si es necesario)
  - Se limitan los nuevos **defectos y regresiones**

- **Error comunes**
  - **Incluir demasiadas definiciones técnicas en los escenarios**
    - Los usuarios y expertos del dominio entienden peor las pruebas de aceptación que contienen detalles técnicos
    - Para evitarlo, lo ideal es que los propios usuarios y expertos sean los que escriban las pruebas de aceptación



- **ATDD: Desarrollo guiado por pruebas de aceptación**
  - ***Acceptance Test Driven Development***
  - Es una metodología de desarrollo que involucra a los desarrolladores, testers y clientes para escribir las pruebas de aceptación antes de implementar la funcionalidad
  - Estos tests de aceptación representan el punto de vista del usuario y actúan como una forma de requisitos que describen cómo debería funcionar el sistema.
  - Cuando se ejecutan de forma automática, verifican el comportamiento del sistema

- **BDD: Desarrollo guiado por comportamiento**
  - *Behaviour Driven Development*
  - También conocido como Especificación mediante ejemplos (Specification by example)
  - En muchos contextos se usa como sinónimo de ATDD, pero algunos autores indican algunas diferencias entre ellos

<https://lizkeogh.com/2011/06/27/atdd-vs-bdd-and-a-potted-history-of-some-related-stuff/>

<https://gabo esquivel.com/blog/2014/differences-between-tdd-atdd-and-bdd/>

- Introducción
- **Gherkin**
- Cucumber
- Karate

- **Gherkin** es un lenguaje específico del dominio, entendible por negocio que te permite describir el comportamiento del software sin detallar cómo se implementa
- Sirve dos propósitos: **documentar** y las **pruebas automáticas**
- Se puede usar en **cualquier idioma** (aquí la usaremos en inglés)
- Gherkin en **ficheros**:
  - Extensión **.feature**
  - Cada fichero contiene una única **funcionalidad**
  - Pueden tener **varias especificaciones**

- **Sintaxis**
  - Basado en la indentación de las líneas (Python, YAML)
  - Comentarios con #

**Feature:** Serve coffee

Coffee should not be served until paid for

Coffee should not be served until the button has been pressed

If there is no coffee left then money should be refunded

**Scenario:** Buy last coffee

**Given** there are 1 coffees left in the machine

**And** I have deposited 1\$

**When** I press the coffee button

**Then** I should be served a coffee

- **Pasos: Given / When / Then**
- Se usan con el mismo significado que hemos visto en los tests unitarios
  - **Given:** Definición del estado de partida del sistema
  - **When:** Acción realizada para ejercitar el SUT
  - **Then:** Resultado esperado

<https://cucumber.io/docs/gherkin/>

- **Pasos: And / But**

- Se usan para mejorar la legibilidad de las especificaciones

Scenario: Multiple Givens

Given one thing

Given another thing

Given yet another thing

When I open my eyes

Then I see something

Then I don't see something else



Scenario: Multiple Givens

Given one thing

And another thing

And yet another thing

When I open my eyes

Then I see something

But I don't see something else

- Esquemas de escenarios (scenario outlines)
- Cuando los escenarios son parecidos, se pueden usar plantillas y los datos en tablas

```
Scenario: Eat 5 out of 12
  Given there are 12 cucumbers
  When I eat 5 cucumbers
  Then I should have 7 cucumbers
```

```
Scenario: Eat 5 out of 20
  Given there are 20 cucumbers
  When I eat 5 cucumbers
  Then I should have 15 cucumbers
```



```
Scenario Outline: Eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers
```

Examples:

	start		eat		left	
	12		5		7	
	20		5		15	





- **Editores**

- Para facilitar la escritura de features con Gherkin existen múltiples editores
- El más sencillo de instalar es Tidy Gherkin, una extensión de Google Chrome



# Gherkin

**TIDY!** FILE + EDIT + INSERT + SUPPORT + ABOUT + **SETTINGS** Pg Try Pretty Gherki

rows columns **INSERT TABLE**  

```
12 Scenario Outline: Eating
13
14     Given that there are <start> cucumbers
15     When I eat <eat> cucumbers
16     Then I should have <left> cucumbers
17
18     | start | eat | left |
19     | 20    | 10  | 10   |
20     | 15    | 3   | 12   |
```

TIDY PREVIEW **JAVA STEPS** RUBY STEPS

```
1 package my.package.name
2
3 import cucumber.api.PendingException;
4 import cucumber.api.java.en.Given;
5 import cucumber.api.java.en.When;
6 import cucumber.api.java.en.Then;
7 import cucumber.api.junit.Cucumber;
8 import org.junit.runner.RunWith;
9
10 @RunWith(Cucumber.class)
11 public class MyStepDefinitions {
12
13     @Given("^there are 12 cucumbers$")
14     public void there_are_12_cucumbers() throws Throwable {
15         throw new PendingException();
16     }
17 }
```

- Escribe los tests de la clase de números complejos como features Gherkin usando Tidy Gherkin
  - Cuando cualquier número complejo es sumado a  $(0+0i)$  el resultado es el mismo número
  - Dado un número, cada una de sus partes puede ser obtenida de forma individual

- Introducción
- Gherkin
- **Cucumber**
- Karate

- Es un framework para ejecutar especificaciones **Gherkin**
- Originalmente escrito en **Ruby**, actualmente tiene versiones para **Java** y **JavaScript**



- Para **ejecutar** una **funcionalidad Gherkin** (*feature*) es necesario implementar **código Java de pegamento** (*glue code*) que “**interprete**” los pasos usando clases del código
- Ese código Java **leerá** los ficheros **.feature** para cargar los **valores** y los **ejemplos** de las plantillas
- Las especificaciones se ejecutan como tests de **TestNG** que pasarán sólo si el software se comporta como se espera

- **Feature**

```
ejem5/src/test/resources/es/codeurjc/test/cucumber/calc.feature
```

**Feature:** Calculator

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

**Scenario:** Add two numbers -2 & 3

**Given** I have a calculator

**When** I add -2 and 3

**Then** the result should be 1

**Scenario:** Add two numbers 10 & 15

**Given** I have a calculator

**When** I add 10 and 15

**Then** the result should be 25

- Ejecución de la feature como TestNG

```
ejem5/src/test/java/es/codeurjc/test/cucumber/CalculatorTest.java
```

```
package es.codeurjc.test.cucumber;  
  
import cucumber.api.CucumberOptions;  
import cucumber.api.testng.AbstractTestNGCucumberTests;  
  
@CucumberOptions(monochrome = true)  
public class CalculatorTest extends AbstractTestNGCucumberTests{  
  
}
```



- **pom.xml**

```
<dependencies>
  <dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-testng</artifactId>
    <version>1.2.5</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.2.5</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-library</artifactId>
    <version>1.3</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Ejecución de los tests

Runs: 6/6 (8 skipped) ✖ Errors: 0 ✖ Failures: 0

es.codeurjc.test.cucumber.CalculatorTest [Runner: ...] Failure Trace

- Feature: Calculator (0.001 s)
  - Scenario: Add two numbers -2 & 3 (0.000 s)
    - Given I have a calculator (0.000 s)
    - When I add -2 and 3 (0.000 s)
    - Then the result should be 1 (0.000 s)
  - Scenario: Add two numbers 10 & 15 (0.000 s)
    - Given I have a calculator (0.000 s)
    - When I add 10 and 15 (0.000 s)
    - Then the result should be 25 (0.000 s)

JUNIT: Test ignorados  
porque no tienen  
"pegamento"

TESTNG: Test ejecutados  
sin funcionalidad

All Tests Failed Tests Summary

- Default suite ( 1/0/0/0 ) (0.08 s)
  - Default test ( 0.08 s)
    - es.codeurjc.test.cucumber.CalculatorTest
      - feature (0.08 s)
        - Calculator (0.08 s)

```
2 Scenarios
6 Steps
0m0.000s
```

You can implement missing steps with the snippets below:

```
@Given("^I have a calculator$")
public void i_have_a_calculator() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I add -(\\d+) and (\\d+)$")
public void i_add_and(int arg1, int arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^the result should be (\\d+)$")
public void the_result_should_be(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I add (\\d+) and (\\d+)$")
public void i_add_and(int arg1, int arg2) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

- Para ejecutar las pruebas de aceptación hay que implementar el **código de pegamento** (*glue code*)
- El test de TestNG:
  - Busca el código de pegamento en su **mismo paquete** (pero es configurable)
  - Busca los .feature en la carpeta con la **misma ruta del paquete** (pero es configurable)

```
ejem6/src/test/java/es/codeurjc/test/cucumber/CalculatorTest.java
```

```
@CucumberOptions(  
    plugin = {"pretty"},  
    features = { "classpath:es/codeurjc/test/cucumber" },  
    glue = {"es.codeurjc.test.cucumber" },  
    monochrome = true)  
public class CalculatorTest extends AbstractTestNGCucumberTests{}
```

- Código pegamento

ejem6/src/test/java/es/codeurjc/test/cucumber/CalculatorRunSteps.java

```
public class CalculatorRunSteps {  
  
    private int total;  
  
    private Calculator calculator;  
  
    @Before  
    private void init() {  
        total = -999;  
    }  
  
    @Given("^I have a calculator$")  
    public void initializeCalculator() throws Throwable {  
        calculator = new Calculator();  
    }  
  
    @When("^I add (-?\\d+) and (-?\\d+)$")  
    public void testAdd(int num1, int num2) throws Throwable {  
        total = calculator.add(num1, num2);  
    }  
  
    @Then("^the result should be (-?\\d+)$")  
    public void validateResult(int result) throws Throwable {  
        assertThat(total, Matchers.equalTo(result));  
    }  
}
```

- Ejecución de los test

The screenshot displays the Cucumber test runner interface. At the top, it shows the execution status: **Runs: 6/6**, **Errors: 0**, and **Failures: 0**, accompanied by a green progress bar. Below this, the test hierarchy is expanded, showing the package `es.codeurjc.test.cucumber.CalculatorTest`. Underneath, two features are listed: **Feature: Calculator (0.008 s)** and **Scenario: Add two numbers 10 & 15 (0.002 s)**. The first feature contains three steps: **Given I have a calculator (0.002 s)**, **When I add -2 and 3 (0.003 s)**, and **Then the result should be 1 (0.000 s)**. The second feature contains three steps: **Given I have a calculator (0.000 s)**, **When I add 10 and 15 (0.000 s)**, and **Then the result should be 25 (0.002 s)**. A green box labeled **JUNIT** has an arrow pointing to the first step of the first feature. A green box labeled **TESTNG** has an arrow pointing to the **Default suite** entry in the **All Tests** tab at the bottom right. The **All Tests** tab shows a hierarchy: **Default suite ( 1/0/0/0 ) (0.142 s)** > **Default test ( 0.142 s)** > **es.codeurjc.test.cucumber.CalculatorRunSteps** > **feature (0.142 s)** > **Calculator (0.142 s)**.

Runs: 6/6 Errors: 0 Failures: 0

es.codeurjc.test.cucumber.CalculatorTest [Runner: ...] Failure Trace

Feature: Calculator (0.008 s)

Scenario: Add two numbers -2 & 3 (0.005 s)

- Given I have a calculator (0.002 s)
- When I add -2 and 3 (0.003 s)
- Then the result should be 1 (0.000 s)

Scenario: Add two numbers 10 & 15 (0.002 s)

- Given I have a calculator (0.000 s)
- When I add 10 and 15 (0.000 s)
- Then the result should be 25 (0.002 s)

JUNIT

TESTNG

All Tests Failed Tests Summary

Default suite ( 1/0/0/0 ) (0.142 s)

- Default test ( 0.142 s)
- es.codeurjc.test.cucumber.CalculatorRunSteps
- feature (0.142 s)
- Calculator (0.142 s)

- Ejecución de los test

Feature: Calculator

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

Scenario: Add two numbers -2 & 3

Given I have a calculator

When I add -2 and 3

Then the result should be 1

```
# es/codeurjc/test/cucumber/calc.feature:6
# CalculatorRunSteps.initializeCalculator()
# CalculatorRunSteps.testAdd(int,int)
# CalculatorRunSteps.validateResult(int)
```

Scenario: Add two numbers 10 & 15

Given I have a calculator

When I add 10 and 15

Then the result should be 25

```
# es/codeurjc/test/cucumber/calc.feature:11
# CalculatorRunSteps.initializeCalculator()
# CalculatorRunSteps.testAdd(int,int)
# CalculatorRunSteps.validateResult(int)
```

2 Scenarios (2 passed)

6 Steps (6 passed)

0m0.068s

- Cuando tenemos varios escenarios similares, es mejor tener una plantilla (scenario outline)

```
ejem7/src/test/resources/es/codeurjc/test/cucumber/calc.feature
```

Feature: Calculator

As a user

I want to use a calculator to add numbers

So that I don't need to add myself

Scenario Outline: Add two numbers <num1> & <num2>

Given I have a calculator

When I add <num1> and <num2>

Then the result should be <total>

Examples:

num1	num2	total
-2	3	1
10	15	25
99	-99	0
-1	-10	-11



- **Steps con expresiones regulares**
  - Para que los steps sean flexibles y puedan ejecutar diferentes escenarios se tienen que implementar usando expresiones regulares
  - Las expresiones regulares
    - Definen el patrón que deben cumplir los steps
    - Capturan los valores para que se puedan usar en la implementación

<https://agileforall.com/just-enough-regular-expressions-for-cucumber/>

- **Steps con expresiones regulares**

- Inicio y fin del texto `^I'm logged in$`
- Cero o más caracteres `.*`
- Uno o más caracteres `.+`
- Cero o más dígitos `\d*` (en Java `\\d*`)
- Uno o más dígitos `\d+` (en Java `\\d+`)
- Cero o más caracteres entre comillas `"[^"]*" (Java "\\ "[^\\"]*"\\ ")`
- Caracter opcional `a?`

- Grupos para capturar valores
  - Entre paréntesis se capturan los valores

```
When I'm logged as an admin  
When I'm logged as a user
```

```
@When("^I'm logged in as an? (.*)$")  
public void whenLogged(String role) throws Throwable {...}
```

- Con ?: no se capturan los valores

```
When I'm logged as an admin  
When I'm logged as a user
```

```
@When("^(?:I'm logged|I log) in as an? (.*)$")  
public void whenLogged(String role) throws Throwable {...}
```

- Implementa el código de pegamento para ejecutar los tests de los números complejos del ejercicio 8

- Introducción
- Gherkin
- Cucumber
- **Karate**



- Es una herramienta de testing que combina:
  - Automatización de test de API REST
  - Mocks
  - Testing de rendimiento
- **Extiende la sintaxis** de Cucumber, pero evitandonos escribir el código “pegamento”.

<https://github.com/intuit/karate>



**Scenario:** create and retrieve a cat

**Given** url 'http://myhost.com/v1/cats'

**And** request { name: 'Billie' }

**When** method post

**Then** status 201

**And** match response == { id: '#notnull', name: 'Billie' }

**Given** path response.id

**When** method get

**Then** status 200

JSON is 'native'  
to the syntax

Intuitive DSL  
for HTTP

Payload  
assertion in  
one line

Second HTTP  
call using  
response data



- ¿Cómo lo añadimos a nuestro proyecto?
- La librería de Karate + TestNG esta obsoleta
- Aún así, podemos trabajar desde TestNG con la librería de Junit 5.

```
<dependency>
  <groupId>com.intuit.karate</groupId>
  <artifactId>karate-apache</artifactId>
  <version>0.9.4</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>com.intuit.karate</groupId>
  <artifactId>karate-junit5</artifactId>
  <version>0.9.4</version>
  <scope>test</scope>
</dependency>
```





## • Ejemplo de Karate en la aplicación de Items

ejem8/src/test/java/es/codeurjc/test/rest/items.feature

Feature: items end-point

Background:

- \* url 'http://localhost:8080'
- \* configure logPrettyRequest = true
- \* configure logPrettyResponse = true

Scenario: create and retrieve a item

Given path 'items/'

And request { description: 'Leche', checked: true }

When method post

Then status 201

And match response == { id: '#number', description: 'Leche', checked: true }

Definimos la URL base

Cuerpo de la petición  
POST

Comprobamos la  
respuesta

No sabemos el 'id' de la respuesta,  
comprobará que 'id' es un número



- Ejemplo de Karate en la aplicación de Items

ejem8/src/test/java/es/codeurjc/test/rest/ItemsControllerJUnit5Test.java

```
package es.codeurjc.test.rest;

import com.intuit.karate.junit5.Karate;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DefinedPort)
class ItemsControllerTest {

    @Karate.Test
    public Karate testSample() {
        return new Karate().feature("items").relativeTo(getClass());
    }
}
```

Nombre del fichero *.feature* (Debe estar en el mismo directorio)

- Ejemplo de Karate en la aplicación de Items

ejem8/src/test/java/es/codeurjc/test/rest/ItemsControllerTestNGTest.java

```
package es.codeurjc.test.rest;
```

```
import static org.testng.Assert.assertTrue;
```

```
import org.springframework.boot.test.context.SpringBootTest;
```

```
import org.springframework.test.context.testng.AbstractTestNGSpringContextTests;
```

```
import org.testng.annotations.Test;
```

```
import com.intuit.karate.Results;
```

```
import com.intuit.karate.Runner;
```

Classpath de la carpeta dónde  
buscará los ficheros .features

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
```

```
class ItemsControllerTestNGTest extends AbstractTestNGSpringContextTests{
```

```
    @Test
```

```
    public void testParallel() throws InterruptedException {
```

```
        Results results = Runner.path("classpath:es/codeurjc/test/rest/").parallel(1);
        assertTrue(results.getFailCount() == 0);
```

```
    }
```

```
}
```

Debemos usar el Runner genérico de  
Karate.



- Comprobación de datos

- Podemos validar el formato de de cada uno de los campos de la respuesta.

```
* def anuncio = { id: 3, contenido: "Vendo moto", pais: 'ES', activo: true }  
* match anuncio == { id: '#number', contenido: '#present', pais: '#regex [A-Z]{2}', activo: '#boolean' }
```

- **#number** → Comprobar que es un número
- **#present** → Comprobar que la clave está presente
- **#regex STR** → Comprueba que cumple la expresión regex STR
- **#boolean** → Comprueba que es un valor booleano

Mas validadores: <https://github.com/intuit/karate#fuzzy-matching>



- ¿Qué hacemos si queremos utilizar variables de Java en nuestro archivo .feature?
  - Es necesario incluir un **fichero Javascript** que haga de 'puente'
  - Este fichero debe estar situado en en la ruta **/src/test/java/** y nombrarse como **karate-config.js**



- Utilizar variables de Java en Karate

ejem9/src/test/java/es/codeurjc/test/rest/ItemsControllerJUnit5Test.java

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class ItemsControllerTest {

    @LocalServerPort
    int port;

    @BeforeEach
    public void setPort(){
        System.setProperty("demo.server.port", port+"");
    }

    @Karate.Test
    public Karate testSample() {
        return new Karate().feature("items").relativeTo(getClass());
    }
}
```

Definimos una propiedad en el sistema

\* Aplicable al test con TestNG



- Utilizar variables de Java en Karate

ejemg/src/test/java/karate-config.js

```
function fn() {  
  var port = karate.properties['demo.server.port'];  
  var config = {  
    'targetUrlBase': 'http://127.0.0.1:' + port  
  };  
  return config;  
}
```

Recuperamos la propiedad (Karate puede leer las propiedades declaradas en Java)

Podemos pasarle al archivo *.feature* las variables declaradas en el objeto config



- Utilizar variables de Java en Karate

ejemg/src/test/java/es/codeurjc/test/rest/items.feature

Feature: items end-point

Background:

- \* url baseUrl
- \* configure logPrettyRequest = true
- \* configure logPrettyResponse = true

Utilizamos la variable  
definida en el archivo de  
configuración

Scenario: create and retrieve a item

Given path 'items/'

And request { description: 'Leche', checked: true }

When method post

Then status 201

And match response == { id: '#number', description: 'Leche', checked: true }





- Ejercicio 10
  - Para la aplicación de anuncios, usar Karate para crear el siguiente escenario
    - Crear, recibir y borrar anuncio

**TIP:** Podemos obtener el id del objeto creado en otra petición para reutilizarlo:

```
....
```

```
* def id = response.id
```

```
Given path '/', id
```

```
...
```