JASMINE + KARMA

Jasmine

- Es un framework de código abierto para pruebas unitarias en JavaScript .
- Está fuertemente influenciado por otros marcos de prueba de unidades, como: ScrewUnit, JSSpec, JSpec y RSpec
- Características
  - Se integra fácilmente con Angular / Ionic
  - Admite pruebas asincrónicas.
  - Hace uso de 'espías' para implementar dobles de prueba
  - Admite pruebas de código front-end a través de una extensión front-end de Jasmine llamada Jasmine-jQuery.

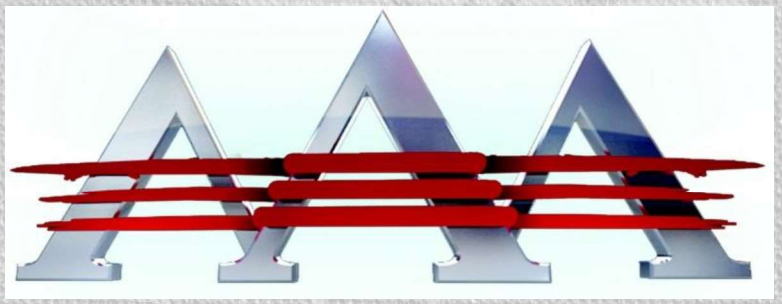
Karma

- Es una herramienta que permite la ejecución de código fuente (es decir, JavaScript) contra navegadores reales a través de la CLI.
- Proporciona herramientas útiles que nos facilitan llamar a nuestras pruebas de Jasmine mientras redactamos el código.

Estructura de un caso de prueba

- Describe
  - Permite definir bloques de pruebas relacionadas entre sí. Se pueden concatenar varios “describe” si es necesario para la estructura de tests.
- It
  - Cada elemento it es una prueba. Se pueden definir todas las pruebas (it) que sean necesarias dentro de un elemento describe. Las pruebas pueden estar divididas entre varios elementos de tipo “describe”. Esto es útil si queremos diferenciar varios bloques de pruebas en un mismo fichero.

Estructura de un caso de prueba



Estructura de un test

- Preparar y configurar el estado necesario para realizar el test
- Realizar la llamada al método que se quiere testear
- Verificar el resultado obtenido

Arrange **A**ct **A**ssert

Estructura de un caso de prueba

- Los métodos de test se crean siguiendo el patrón AAA:
  - Arrange (Preparar)
  - Act (Actuar)
  - Assert (Afirmar)

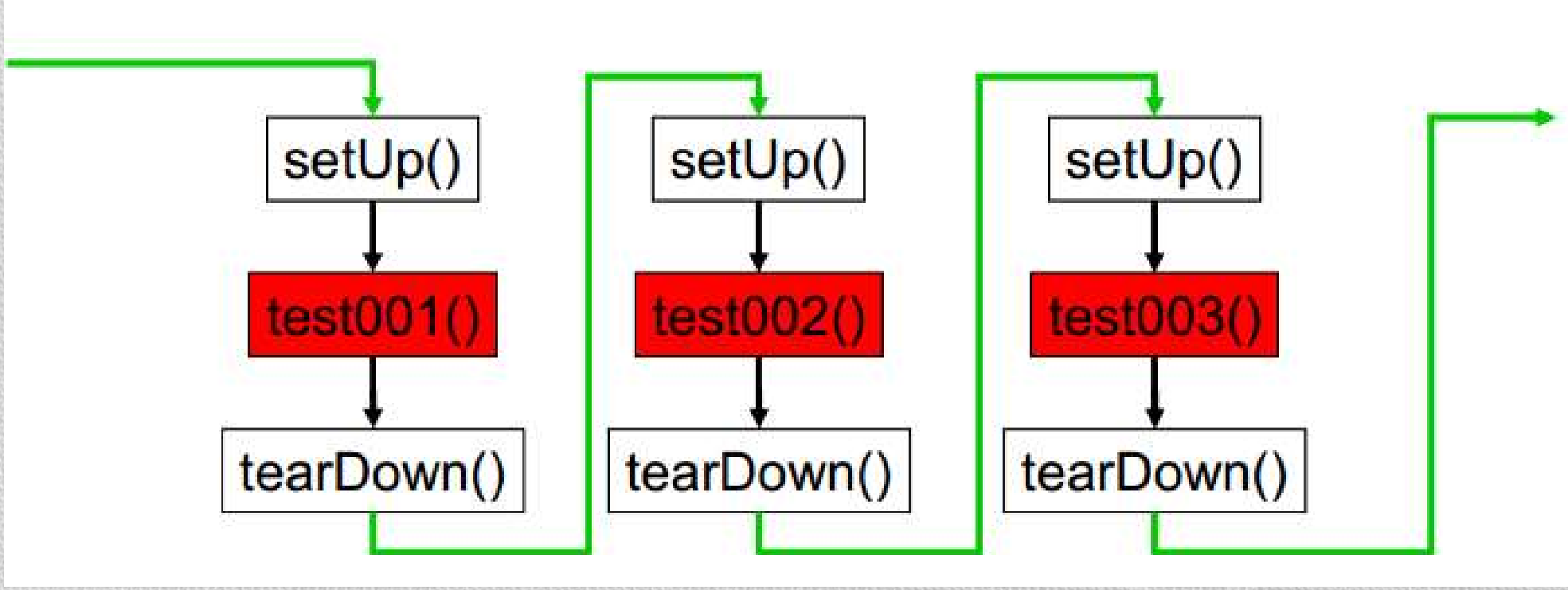
```
describe('Sumar', function() {

  it('10 + 15 debe ser 25', function() {
    //Arrange
    let calc = new Calculadora();
    //Act
    var res = calc.sumar( 10, 15);
    //Assert
    expect( res ).toBe( 25 );
  });
});
```

Hooks

- Las clases de prueba tienen que programarse eficientemente y refactorizarse cuando sea necesario.
- Una de las características principales de Mocha es la capacidad de ejecutar ciertos fragmentos de código antes o después de ciertas pruebas. Mediante las que podemos inicializar o finalizar elementos comunes, evitar duplicidades, preparar datos, etc.
  - Before
  - BeforeEach
  - AfterEach
  - After

```
describe('Calculadora', function () {
  let calc;
  describe('Sumar', function () {
    beforeEach(function () {
      calc = new Calculadora();
    });
    it('10 + 15 debe ser 25', function () {
      var res = calc.sumar(10, 15);
      expect(res).toBe(25);
    });
    it("-10 + -15 debe ser -25", function () {
      var res = calc.sumar(-10, -15);
      expect(res).toBe(-25);
    });
    afterEach(function () {
      calc = null;
    });
  });
});
```

Hooks


setUp()
test001()
tearDown()
setUp()
test002()
tearDown()
setUp()
test003()
tearDown()

Assertions

- Para realizar validaciones en nuestros métodos de prueba, se utilizan las condiciones de aceptación.

expect(object).toExist([message])	Afirma que el objeto dado es verdad
expect(object).toNotExist([message])	Afirma que el objeto dado es falso
expect(object).toBe(value, [message])	Afirma que el objeto es estrictamente igual al valor usando el operador =
expect(object).toNotBe(value, [message])	Afirma que el objeto NO es estrictamente igual al valor usando el operador =
expect(object).toEqual(value, [message])	Afirma que el objeto es estrictamente igual al valor usando la función is-equal
expect(object).toNotEqual(value, [message])	Afirma que el objeto NO es estrictamente igual al valor usando la función is-equal
expect(block).toThrow([error], [message])	Afirma que el bloque dado arroja un error. El argumento de error puede ser un constructor (para probar usando instanceof), o una cadena / RegExp para probar contra error.message.

Chaining Assertions

- Cada aserción devuelve un objeto Expectation, por lo que puede encadenar aserciones juntas.

```
expect(3.14)
.toExist()
.toBeLessThan(4)
.toBeGreaterThan(3)
```

Timeouts

- Se pueden realizar pruebas de rendimiento verificando que un test no exceda un tiempo límite de ejecución. Esto se puede hacer para una prueba, para un bloque de pruebas o para todas las pruebas.

```
describe('Calculadora', function() {
  describe('Sumar', function() {
    this.timeout(500);
     it('10 + 15 debe ser 25', function(done) {
          setTimeout(done, 300);
          let calc = new Calculadora();
          var res = calc.sumar( 10, 15);
          expect( res ).toBe( 25 );
       });
   it('-10 + -15 debe ser -25', function(done) {
          setTimeout(done, 200);
          let calc = new Calculadora();
          var res = calc.sumar( -10, -15);
          expect( res ).toBe( -25 );
       });
   });
});
```

```
describe('Calculadora', function() {
  describe ('Sumar', function() {
  it('10 + 15 debe ser 25', function(done) {
          this.timeout(500);
          setTimeout(done, 300);
          let calc = new Calculadora();
          var res = calc.sumar( 10, 15);
          expect( res ).toBe( 25 );
       });
    it('-10 + -15 debe ser -25', function() {
          let calc = new Calculadora();
          var res = calc.sumar( -10, -15);
          expect( res ).toBe( -25 );
       });
   });
});
```