
# Tema 5. Pruebas en aplicaciones web

Programación web

Boni García

Curso 2017/2018

# Índice

# Índice

# Introducción

## Verificación y validación

- La **evaluación en el software** (también conocido como control de calidad o verificación y validación, V&V) es la fase del ciclo de vida destinada a:
  - Evaluar la **calidad** del software asegurando que el producto desarrollado cumple los **requisitos** funcionales y no funcionales establecidos (especificación) y las expectativas del consumidor (cliente/usuario)
  - Identificar los posibles **defectos** (conocidos de forma genérica como ***bugs***)
- No hay un consenso claro del alcance de verificación y validación, por eso se suele agrupar bajo el termino V&V. La definición clásica es:
  - Verificación: *Hacer bien las cosas* (cumplir la especificación)
  - Validación: *Hacer las cosas bien* (cumplir expectativas)

# Introducción

## Verificación y validación

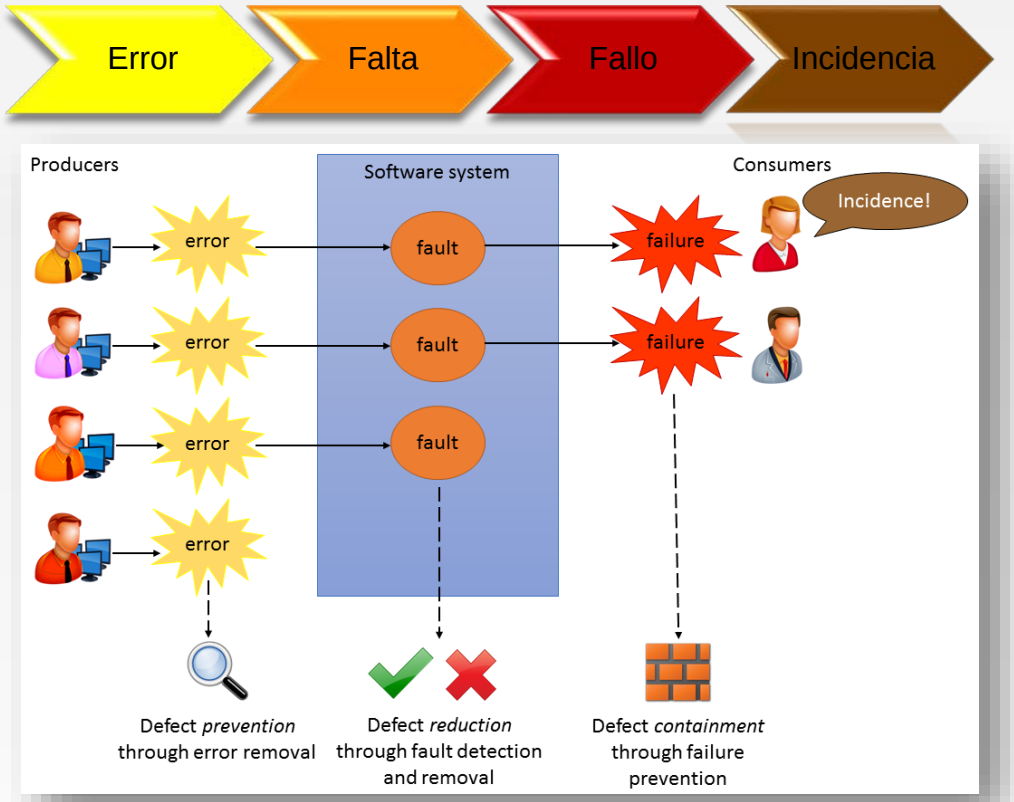
- Hay dos tipos de técnicas dentro de V&V:

1. **Análisis** (estático): evaluación del software (ya sea código, modelos, documentación, etc.) sin ejecutar el propio software
   - Revisión de pares: Collaborator, Crucible, Gerrit …
   - Análisis automático (*linters*): SonarQube, Checkstyle, FindBugs, PMD …

2. **Pruebas** (dinámico): evaluación del software observando un resultado *esperado* de la *ejecución* de una *parte o todo el sistema* (caso de prueba) y dar un *veredicto* sobre la misma

# Introducción

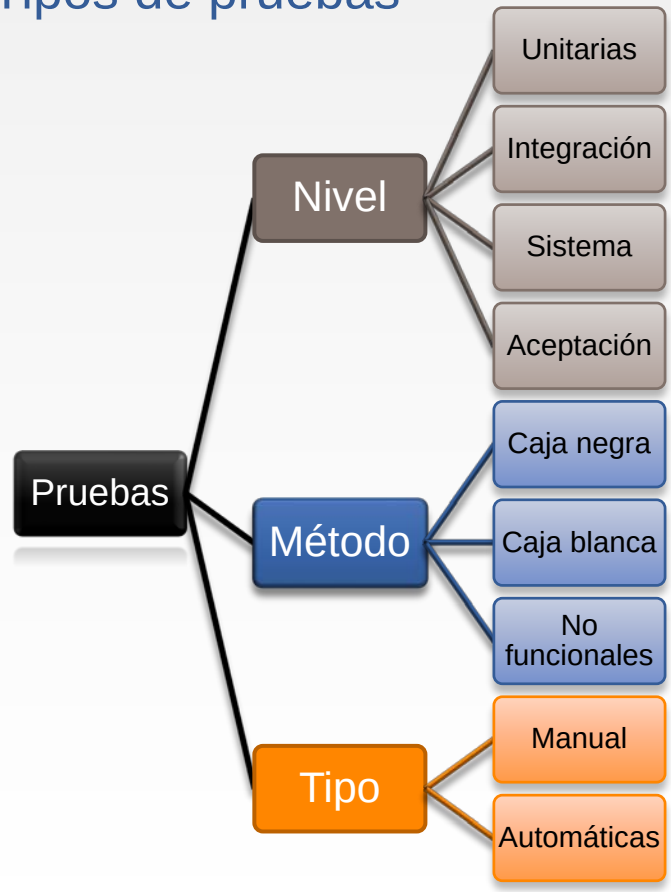
## Verificación y validación




Taxonomía de defectos

# Introducción

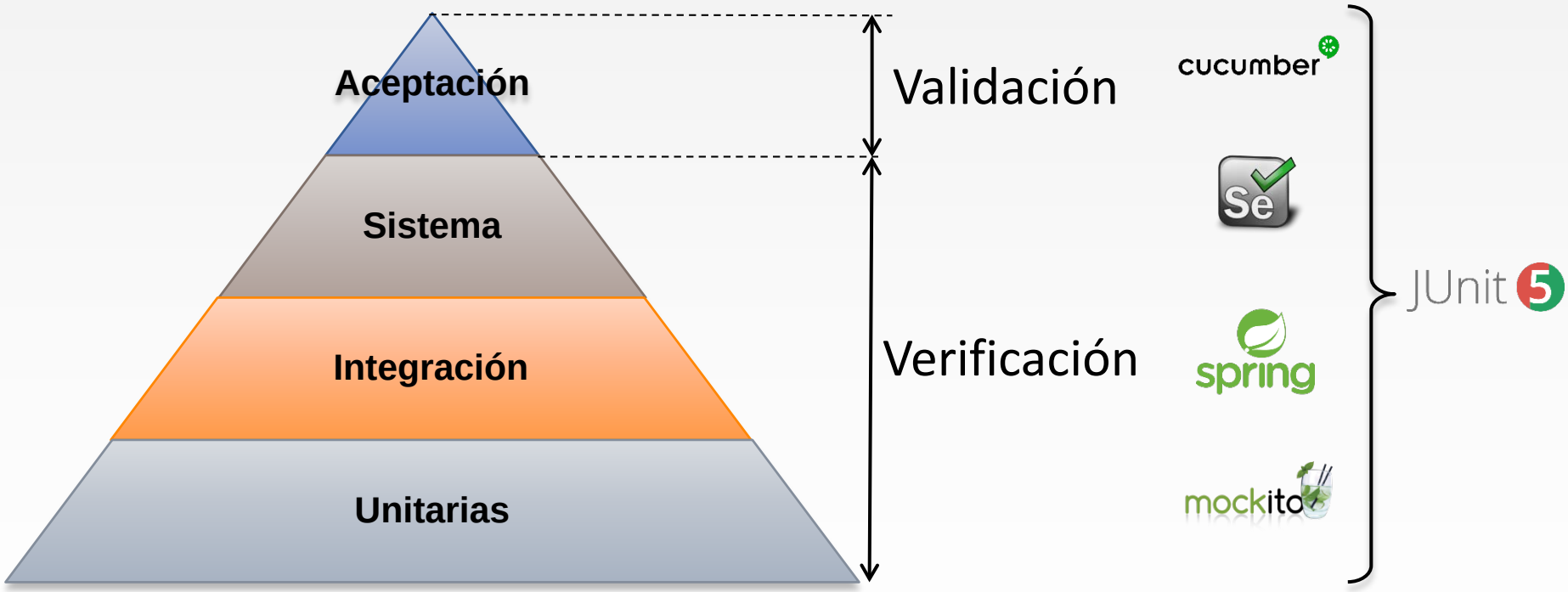
## Tipos de pruebas




- Pruebas unitarias (componentes aislados)
- Pruebas de integración (diferentes componentes)
- Pruebas de sistema (todos los componentes)
- Pruebas de aceptación (pruebas de usuario)
- Pruebas de caja negra (funcionales)
- Pruebas de caja blanca (estructurales)
- Pruebas no funcionales (rendimiento, seguridad,…)
- Pruebas manuales
- Pruebas automáticas (*frameworks* de prueba)

# Introducción

## Tipos de pruebas



# Introducción

## Automatización de pruebas

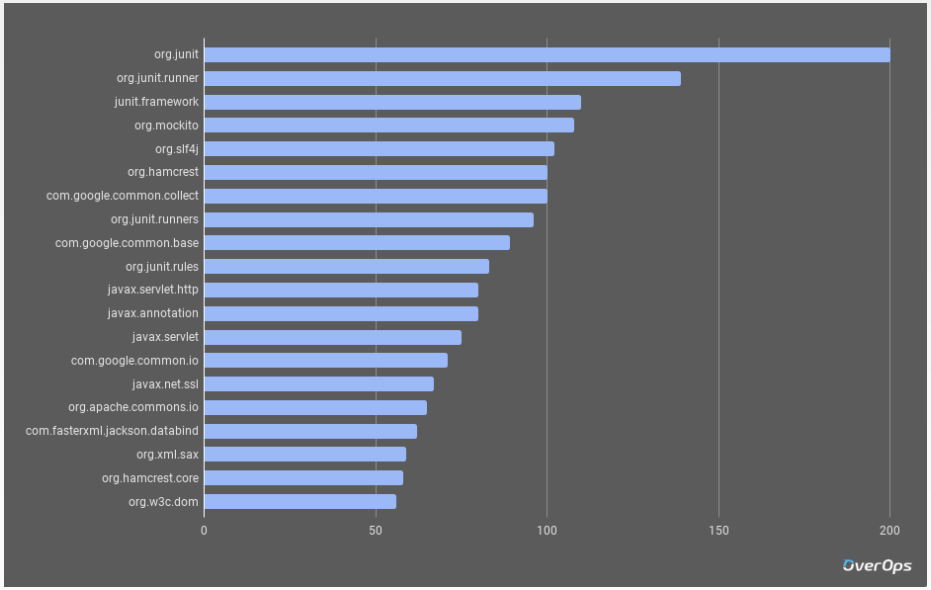
- Las pruebas manuales son muy **costosas**
- La **automatización** de pruebas ayuda a reducir dichos esfuerzos y se define como:

  "*La aplicación e implementación de tecnología software durante todo el ciclo de pruebas para mejorar la eficacia y eficiencia del mismo*"
- La automatización de pruebas es mucho más efectiva cuando se implementa usando un **framework** de pruebas:
  - JUnit, Selenium, Mockito, Cucumber, …

# Índice

# JUnit

- JUnit es el **framework de pruebas** más utilizado en la comunidad Java y uno de los más influyentes en la ingeniería de software en general (precursor de la familia xUnit)
- La última versión de JUnit 4 fue liberada en diciembre de 2014
- JUnit 4 tiene importantes limitaciones que han propiciado el rediseño completo del framework en **JUnit 5**




The Top 100 Java libraries on GitHub (by OverOps)

JUnit 5

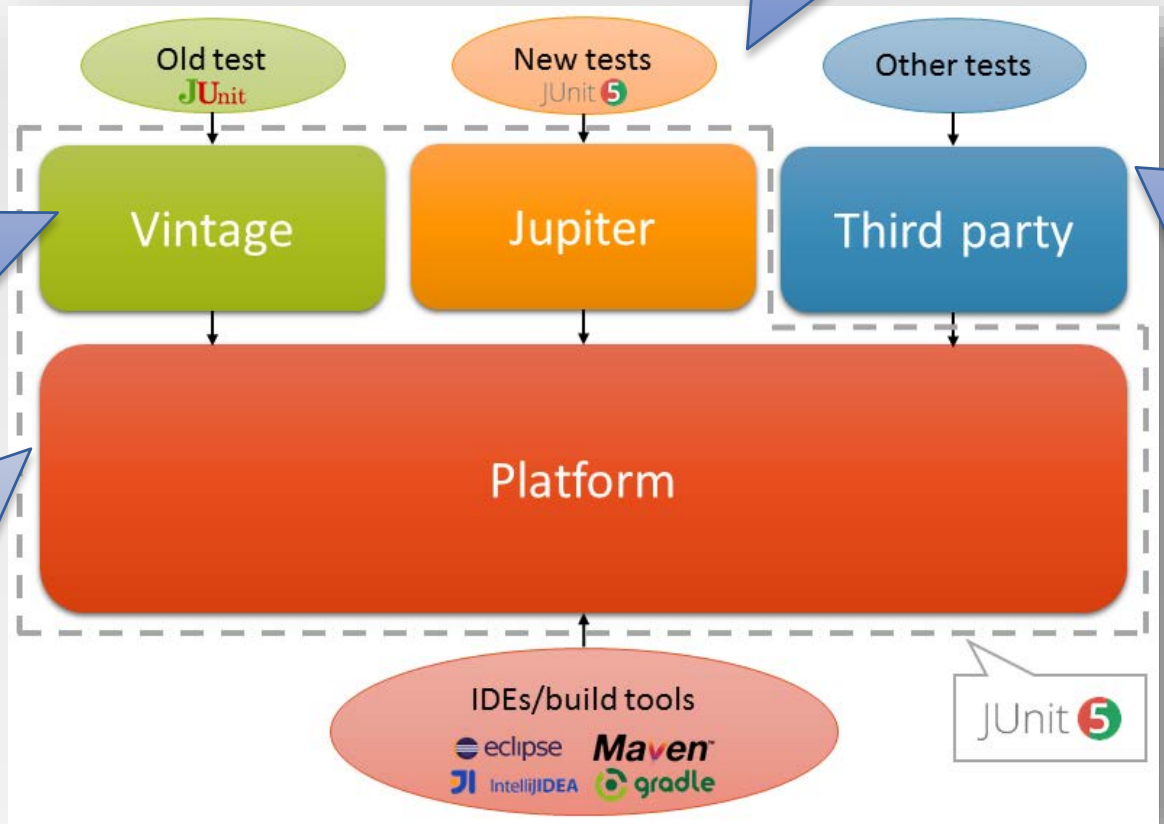
http://junit.org/junit5/

# JUnit

***Jupiter*** es componente que implementa el nuevo modelo de programación y extensión en JUnit 5

## Arquitectura de JUnit 5

Test de versiones anteriores de JUnit (3 y 4) serán ejecutados a través del componente ***Vintage***




Otros frameworks pueden ejecutar sus propios tests realizando una extensión de la plataforma

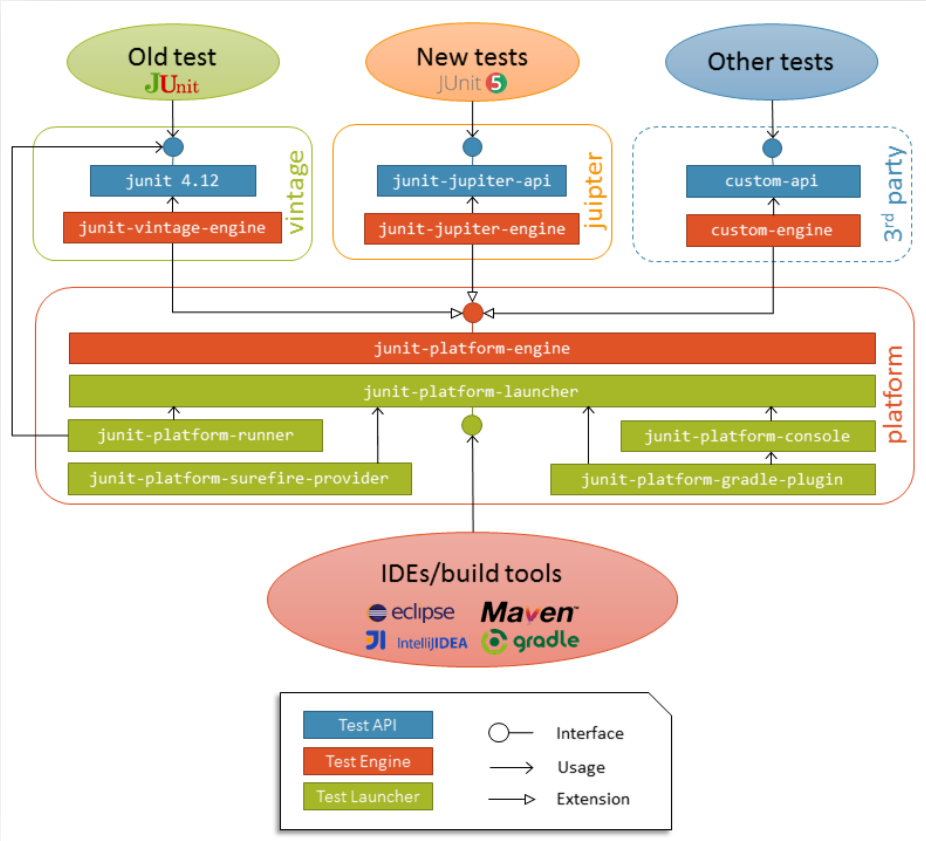
La plataforma JUnit (***Platform***) es un componente que actúa de ejecutor genérico para pruebas que se ejecutan en la JVM

# JUnit

## Arquitectura de JUnit 5

- Hay tres tipos de módulos:

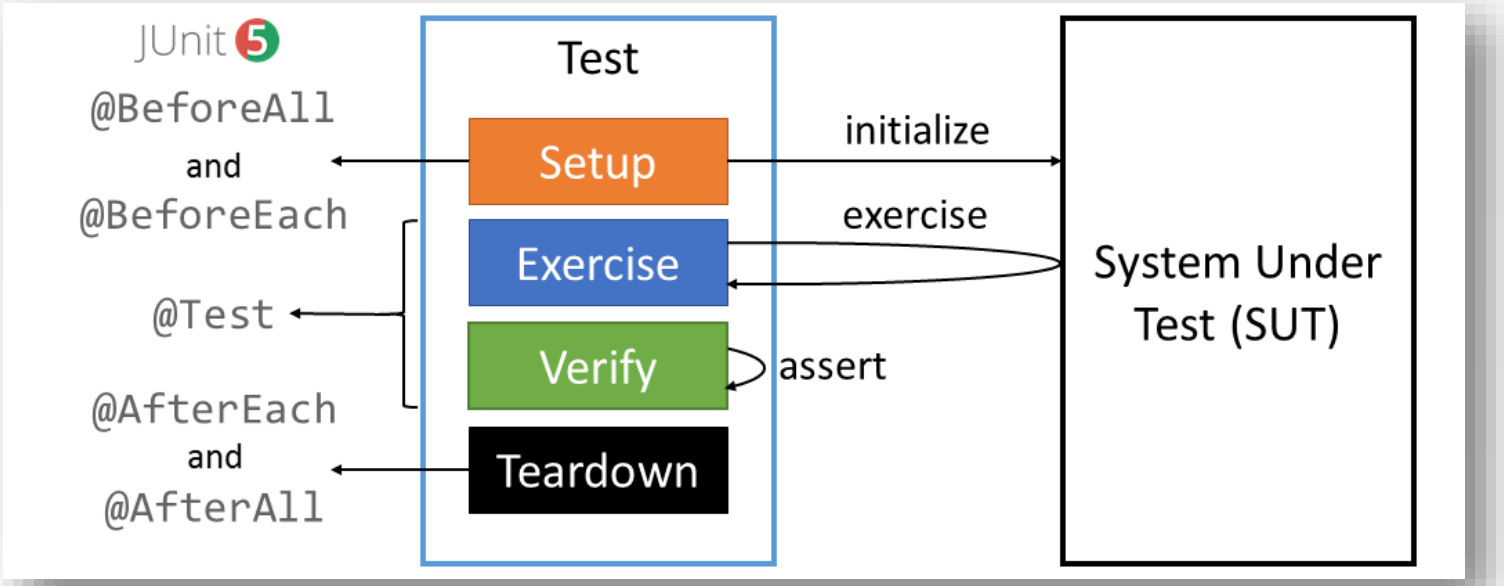
1. **Test API**: Módulos usados por *testers* para implementar casos de prueba
2. **Test Engine SPI**: Módulos extendidos para un *framework* de pruebas Java para la ejecución de un modelo concreto de tests
3. **Test Launcher API**: Módulos usados por *clientes programáticos* para el descubrimiento de tests



# JUnit

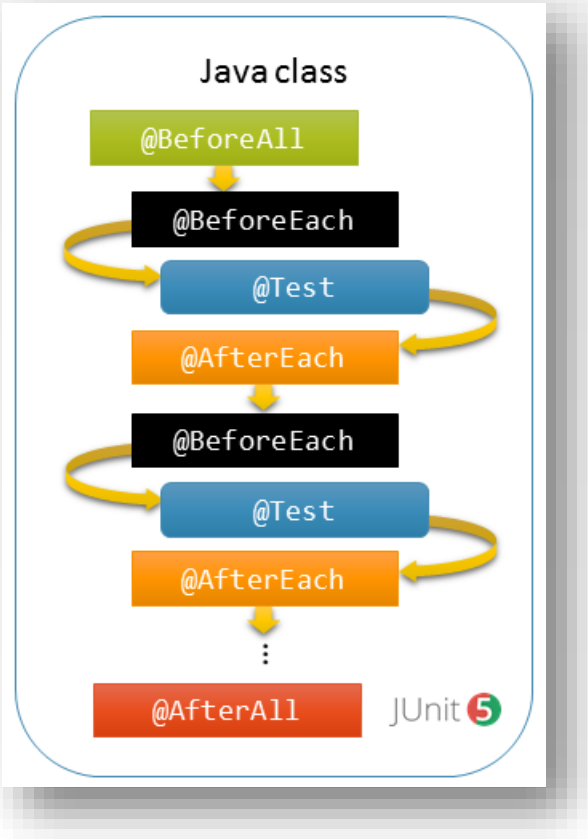
## Ciclo de vida

- El **ciclo de vida** de un caso de prueba en JUnit 5 es como sigue:



# JUnit

## Ciclo de vida




```java
import org.junit.jupiter.api.*;

public class JUnitTest {

    @BeforeAll
    static void setupAll() {
        // Setup all tests
    }

    @BeforeEach
    void setup() {
        // Setup each test
    }

    @Test
    void test1() {
        // Exercise and verify
    }

    @Test
    void test2() {
        // Exercise and verify
    }

    @AfterEach
    void teardown() {
        // teardown each test
    }

    @AfterAll
    static void teardownAll() {
        // teardown all tests
    }

}
```

# JUnit

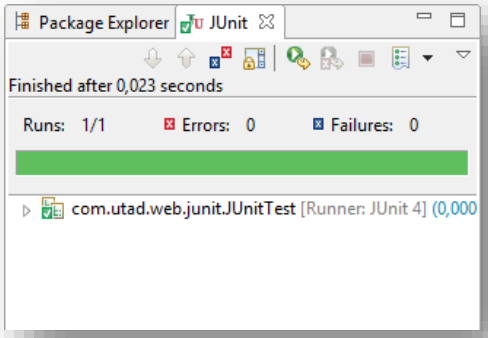
## Ejecución de tests

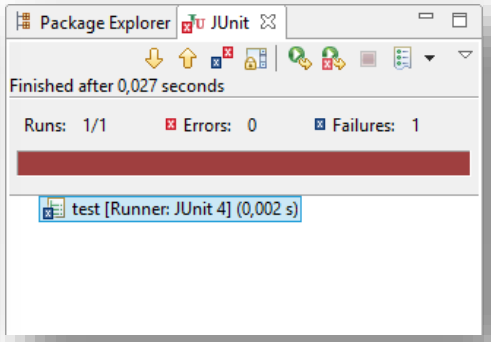
- Actualmente se pueden **ejecutar** test de JUnit 5 de diferentes formas:

1. Mediante herramienta de gestión y construcción de proyectos Java (*build tools*)
   - Maven
   - Gradle
   - Ant



2. Mediante un Entorno de Desarrollo Integrado (IDE)
   - IntelliJ IDEA 2016.2+
   - Eclipse 4.7+
   - Visual Studio 2017+



# JUnit



## Ejecución de tests

- Dependencias JUnit 5 en Maven:

```
    <properties>
        <junit.jupiter.version>5.1.1</junit.jupiter.version>
        <junit.platform.version>1.1.1</junit.platform.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.jupiter.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-launcher</artifactId>
            <version>${junit.platform.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

# JUnit



## Ejecución de tests

- Dependencias JUnit 5 en Maven:

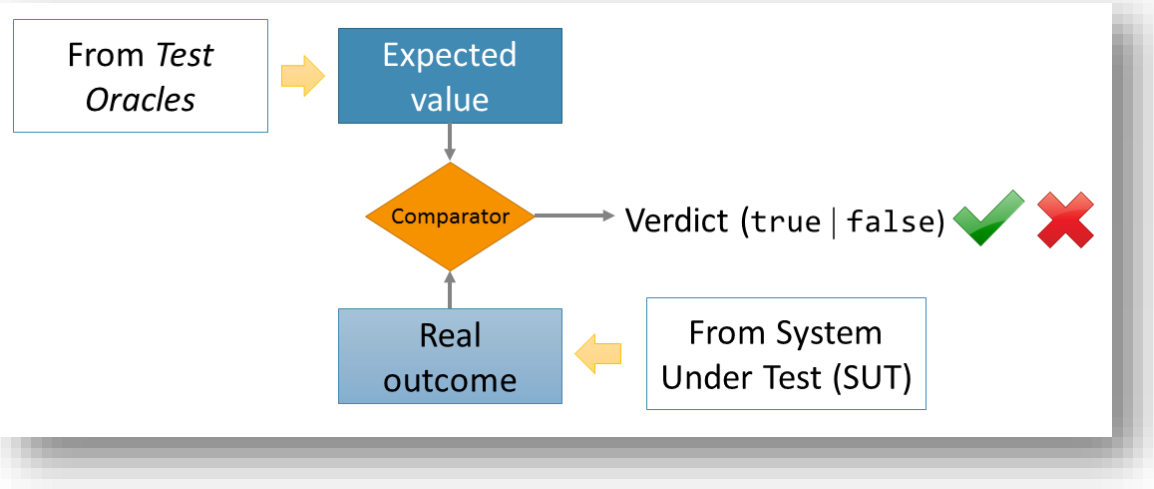
```
<properties>
    <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
</properties>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>${maven-surefire-plugin.version}</version>
            <dependencies>
                <dependency>
                    <groupId>org.junit.platform</groupId>
                    <artifactId>junit-platform-surefire-provider</artifactId>
                    <version>${junit.platform.version}</version>
                </dependency>
            </dependencies>
        </plugin>
    </plugins>
</build>
```

El plugin `maven-surefire-plugin` es necesario para ejecutar las pruebas desde línea de comandos (`mvn test`)

# JUnit

## Aserciones

- Una **aserción** (o predicado) está formada por
  - Datos esperados, obtenidos de lo que se conoce como oráculo (típicamente la especificación del SUT)
  - Datos reales, obtenidos de ejercitar el sistema bajo pruebas (SUT)
  - Un operador lógico que compara ambos valores



# JUnit

## Aserciones

- Las aserciones básicas en Jupiter son las siguientes:

| Aserción | Descripción |
|---|---|
| `fail` | Hace fallar un test proporcionando un mensaje de error u excepción |
| `assertTrue` | Evalúa si una condición es cierta |
| `assertFalse` | Evalúa si una condición es false |
| `assertNull` | Evalúa si un objeto es null |
| `assertNotNull` | Evalúa si un objeto no es null |
| `assertEquals` | Evalúa si un objeto es igual a otro |
| `assertNotEquals` | Evalúa si un objeto no es igual a otro |
| `assertArrayEquals` | Evalúa si un array es igual a otros |
| `assertIterableEquals` | Evalúa si dos objetos iterables son iguales |
| `assertLinesMatch` | Evlúa si dos listas de String son iguales |
| `assertSame` | Evalúa si un objeto es el mismo que otro |
| `assertNotSame` | Evalúa si un objeto no es el mismo que otro |

Métodos estáticos de la clase `Assertions`

# JUnit



## Aserciones

- Algunos ejemplos de aserciones en JUnit 5:

```
import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.junit.jupiter.api.Assertions.assertTrue;


boolean booleanCondition = true;
Object object1 = null;
Object object2 = new Object();
Object[] array1 = {};
Object[] array2 = {};

assertTrue(booleanCondition);
assertFalse(!booleanCondition);

assertArrayEquals(array1, array2);

assertNull(object1);
assertNotNull(object2);
```

# JUnit



- Ejemplo: pruebas unitarias para la clase `ArrayList` de Java:

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.ArrayList;
import java.util.List;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class ArrayListTest {
    List<String> list;
    final String[] data = { "data1", "data2", "data3" };

    @BeforeEach
    void setup() {
        list = new ArrayList<String>();
        for (String s : data) {
            list.add(s);
        }
    }

    @AfterEach
    void teardown() {
        list.clear();
    }

    @Test
    void testContent() {
        for (int i = 0; i < data.length; i++) {
            // Exercise
            String expectedContent = data[i];
            String realContent = list.get(i);

            // Verify
            assertEquals(expectedContent, realContent);
        }
    }

    @Test
    void testSize() {
        // Exercise
        int expectedSize = data.length;
        int realSize = list.size();

        // Verify
        assertTrue(realSize == expectedSize);
    }

}
```

# Índice

# Selenium

## Introducción

- **Selenium** es un **framework** que permite la **automatización** de **pruebas** para aplicaciones **web**
- Diseñado inicialmente en 2004 por Jason Huggins
- El nombre fue elegido como burla de la herramienta comercial de pruebas Mercury (actualmente HP Unified Functional Testing)

"*Selenium is a key mineral which protects the body from Mercury toxicity*"


http://www.seleniumhq.org/

# Selenium

## Introducción

- Selenium está formado por tres componentes:

| Proyecto | | Descripción |
|---|---|---|
| Selenium IDE |  | Plugin Firefox que permite grabación y reproducción de aplicaciones web |
| Selenium WebDriver |  | Control programático de ejecución web con navegadores locales |
| Selenium Grid |  | Permite ejecutar Selenium WebDriver en máquinas remotas |

# Selenium

## WebDriver

- Selenium WebDriver permite manejar un navegador web programáticamente
- Compatibilidad:
  - Navegadores: Chrome, Firefox, Internet Explorer, Opera, Safari, Edge
  - Navegadores móviles: Android, iOS, Windows Phone
  - Navegadores "*headless*": HtmlUnit, PhantomJS
  - Sistemas operativos: Windows, Linux, Mac OS X
  - Lenguajes: C#, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, R, Ruby

http://docs.seleniumhq.org/projects/webdriver/

# Selenium

## WebDriver

- Estructura típica de un caso de prueba Java con Selenium WebDriver:

1. Importar la dependencia de `selenium-java` en nuestro proyecto
2. Resolver binario necesario para controlar el navegador
   - *chromedriver* para Chrome
   - *geckodriver* para Firefox
   - ...
3. Instanciar un objeto `WebDriver` (para Chrome, Firefox, etc)
4. Abrir una página web (URL)
5. Localizar elementos (`WebElement`)
6. Interactuar con elementos (hacer click, leer atributos, etc)
7. Verificar que se cumplen las condiciones esperadas (aserciones)

# Selenium

## WebDriver

- Estructura típica de un caso de prueba con Selenium WebDriver:

1. Importar la dependencia de `selenium-java` en nuestro proyecto:

```
<dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-java</artifactId>
   <version>3.11.0</version>
   <scope>test</scope>
</dependency>
```

2. Resolver binario necesario para controlar el navegador:

```
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
```

3. Instanciar un objeto `WebDriver` (para Chrome, Firefox, etc):

```
WebDriver driver = new FirefoxDriver();
WebDriver driver = new ChromeDriver();
WebDriver driver = new OperaDriver();
WebDriver driver = new InternetExplorerDriver();
WebDriver driver = new SafariDriver();
```

# Selenium

## WebDriver

- Estructura típica de un caso de prueba con Selenium WebDriver:

4. Abrir una página web (URL):

```
driver.get("http://en.wikipedia.org/wiki/Main_Page");
```

5. Localizar elementos (`WebElement`)

```
// Locate single element
WebElement webElement1 = driver.findElement(By.id("id"));
WebElement webElement2 = driver.findElement(By.name("name"));
WebElement webElement3 = driver.findElement(By.className("class"));
WebElement webElement4 = driver.findElement(By.cssSelector("cssInput"));
WebElement webElement5 = driver.findElement(By.linkText("text"));
WebElement webElement6 = driver.findElement(By.partialLinkText("partial text"));
WebElement webElement7 = driver.findElement(By.tagName("tag name"));

// Locate single element list
List<WebElement> webElements = driver.findElements(By...);
```

# Selenium

## WebDriver

- Estructura típica de un caso de prueba con Selenium WebDriver:

6. Interactuar con elementos (hacer click, leer atributos, etc). Por ejemplo:

```
webElement1.click();
webElement1.clear();
webElement1.sendKeys("text");

String text = webElement1.getText();
String href = webElement1.getAttribute("href");
String css = webElement1.getCssValue("css");
Dimension dim = webElement1.getSize();

boolean enabled = webElement1.isEnabled();
boolean selected = webElement1.isSelected();
boolean displayed = webElement1.isDisplayed();
```

# Selenium

## WebDriver

- Estructura típica de un caso de prueba con Selenium WebDriver:

7. Verificar que la web bajo pruebas cumple las condiciones esperadas (aserciones). Por ejemplo:

```
WebDriverWait wait = new WebDriverWait(driver, 30); // seconds

wait.until(ExpectedConditions.elementToBeClickable(By.id("id1")));
wait.until(ExpectedConditions.elementToBeSelected(By.id("id2")));
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("id3")));
wait.until(ExpectedConditions.textToBePresentInElementLocated(By.tagName("body"), "text"));
wait.until(ExpectedConditions.titleIs("Page title"));
```

- También podemos usar JUnit para dar un veredicto sobre el test, pero en ese caso hay que gestionar las esperas manualmente

# Selenium

## Selenium-Jupiter

- Los pasos 1 a 3 se pueden hacer de forma automática usando la extensión **Selenium-Jupiter** para JUnit 5

```
<dependency>
   <groupId>io.github.bonigarcia</groupId>
   <artifactId>selenium-jupiter</artifactId>
   <version>2.1.1</version>
   <scope>test</scope>
</dependency>
```


https://bonigarcia.github.io/selenium-jupiter/

# Selenium

## Selenium-Jupiter

Ejemplo usando Chrome como navegador



```
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeTest {

    @Test
    void testChrome(ChromeDriver driver) {
        driver.get("http://en.wikipedia.org/wiki/Main_Page");
        driver.findElement(By.id("searchInput")).sendKeys("Software");
        driver.findElement(By.id("searchButton")).click();

        boolean containsText = driver.findElement(By.tagName("body")).getText()
                .contains("Computer software");
        assertTrue(containsText);
    }

}
```

# Selenium

## Selenium-Jupiter



Ejemplo usando Firefox como navegador

```
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class FirefoxTest {

    @Test
    void testFirefox(FirefoxDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertTrue(
                driver.getTitle().contains("JUnit 5 extension for Selenium"));
    }

}
```

# Índice

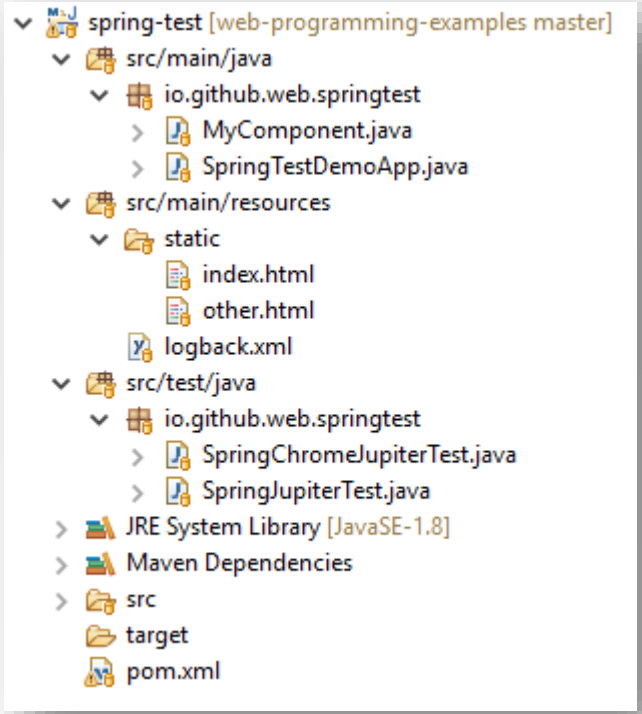
# Spring Test

## Introducción

- La dependencia de pruebas en Spring Boot se llama `spring-boot-starter-test`
- Esta dependencia permite cargar un contexto de aplicación Spring desde casos de prueba
- Proporciona además diferentes utilidades para pruebas
- Spring 5 (Spring Boot 2) proporciona una extensión para JUnit 5
  - `@ExtendWith(SpringExtension.class)`

# Spring Test

## Ejemplo




Fork me on GitHub

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.jupiter.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-launcher</artifactId>
            <version>${junit.platform.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>selenium-jupiter</artifactId>
            <version>${selenium-jupiter.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

# Spring Test

## Ejemplo

```
@SpringBootApplication
public class SpringTestDemoApp extends WebMvcConfigurerAdapter {

   public static void main(String[] args) {
      SpringApplication.run(SpringTestDemoApp.class, args);
   }
}
```

```
import org.springframework.stereotype.Component;

@Component
public class MyComponent {

    public String sayHello() {
        return "Hello world";
    }
}
```

```
<!DOCTYPE html>
<html>
<head>
<title>Spring Boot Test</title>
</head>

<body>
<h1>Home page</h1>
Go to <a href="other.html">another</a> page.
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Spring Boot Test</title>
</head>

<body>
<h1>Other page</h1>
Hello!
</body>
</html>
```

# Spring Test

## Ejemplo

Test de integración

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@ExtendWith(SpringExtension.class)
@SpringBootTest
public class SpringJupiterTest {

    @Autowired
    MyComponent myComponent;

    @Test
    void test() {
        String hello = myComponent.sayHello();
        assertEquals(hello, "Hello world");
    }

}
```

# Spring Test

## Ejemplo

Test de sistema (*end-to-end*)

```
@ExtendWith({ SeleniumExtension.class, SpringExtension.class })
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class SpringChromeJupiterTest {

    @LocalServerPort
    int serverPort;

    @Test
    public void test(ChromeDriver driver) {
        // Open system under test
        driver.get("http://localhost:" + serverPort);

        // Verify first page title
        String firstPageTitle = driver.getTitle();
        String expectedFirstPageTitle = "Spring Boot Test - Page 1";
        assertEquals(expectedFirstPageTitle, firstPageTitle);

        // Click on link
        driver.findElement(By.linkText("another")).click();

        // Verify second page caption
        String secondPageCaption = driver.findElement(By.id("caption"))
                .getText();
        String expectedSecondPageTitle = "Other page";
        assertEquals(expectedSecondPageTitle, secondPageCaption);
    }

}
```

# Spring Test

## Ejemplo

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.1.RELEASE)

2018-04-14 16:21:55.401  INFO 9580 --- [           main]
i.g.web.springtest.SpringChromeTest      : Starting SpringChromeTest on LAPTOP-
T9O4060I with PID 9580 (started by boni in D:\projects\web-programming-
examples\spring-test-selenium)
...
2018-04-14 16:21:59.038  INFO 9580 --- [           main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 53452 (http)
2017-04-14 16:21:59.044  INFO 9580 --- [           main]
i.g.web.springtest.SpringChromeTest      : Started SpringChromeTest in 3.936
seconds (JVM running for 6.166)
Starting ChromeDriver 2.37.544315 (62ebf098771772160f391d75e589dc567915b233) on
port 29914
Only local connections are allowed.
...
2018-04-14 16:22:02.915  INFO 9580 --- [       Thread-3]
ationConfigEmbeddedWebApplicationContext : Closing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationCon
text@293a5f75: startup date [Fri Apr 14 16:21:55 CEST 2017]; root of context
hierarchy
```