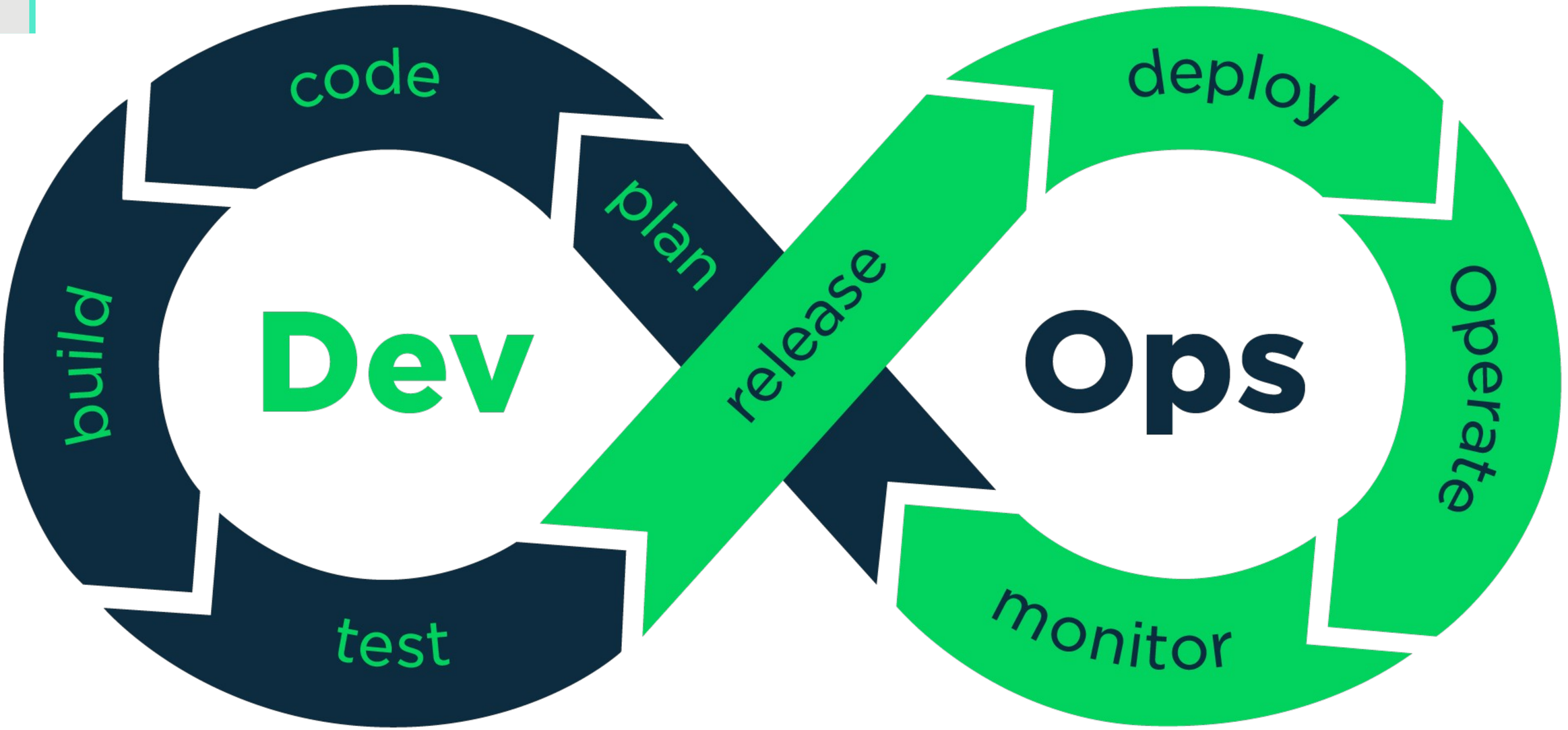
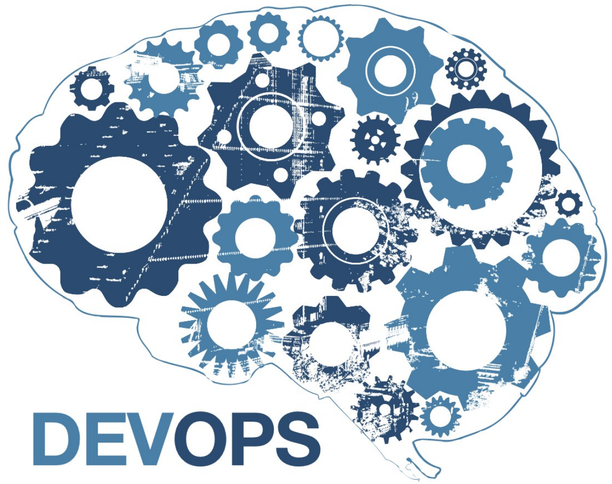

code
plan
build
Dev
test
release
deploy
Ops
operate
monitor




INTERMEDIO

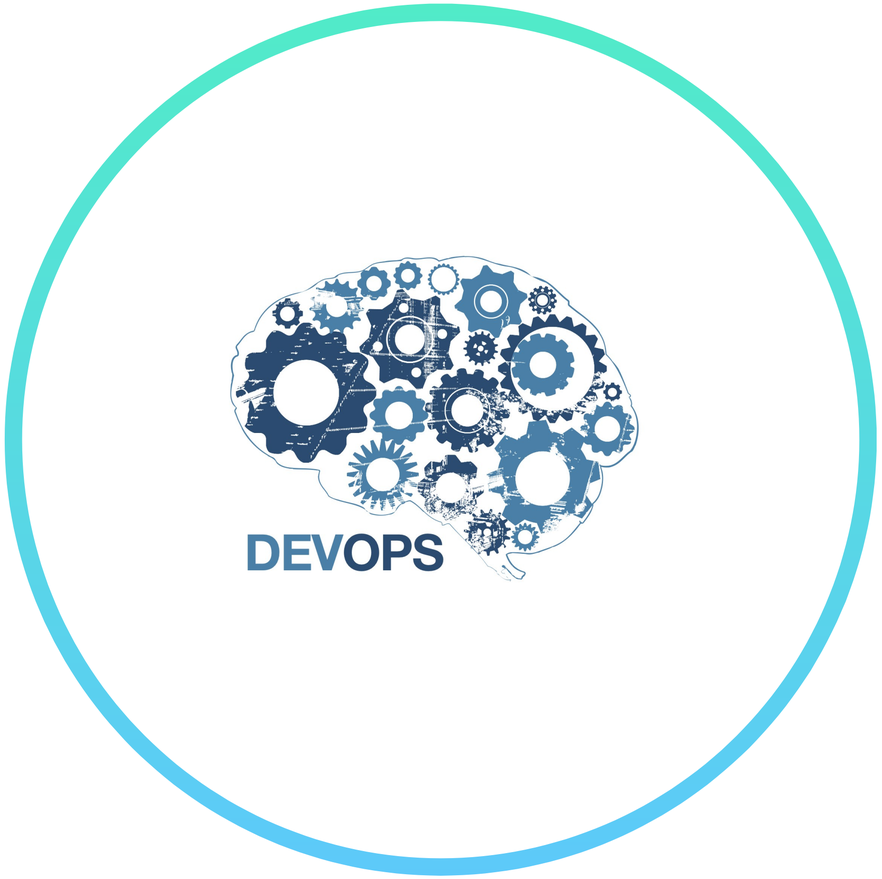
Unidad 02

# Junit/Mockito

CURSOS DE DESARROLLO

# Objetivos

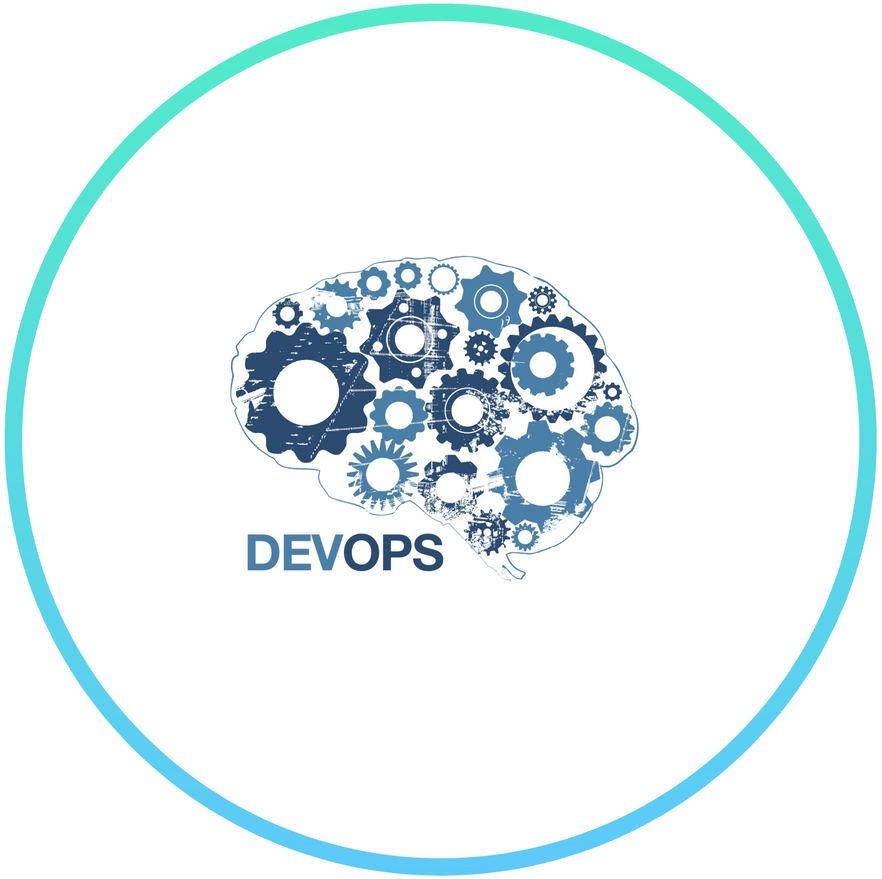
¿Qué voy a a aprender?




- Saber cómo escribir pruebas de unidad
- Saber cómo lanzar pruebas de unidad
- Saber cómo realizar pruebas de integración
- Saber cómo aislar un componente en las pruebas de integración

# Contenidos

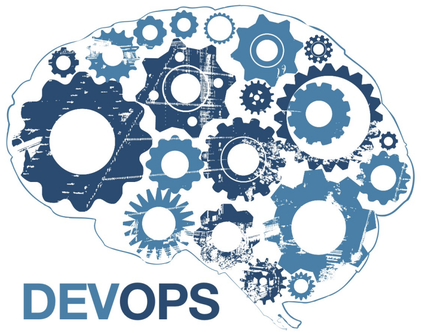
¿Cómo voy a aprenderlo?




1. Introducción
2. Estructura de un fichero
3. Fixtures
4. Test Suite
5. Asserts
6. Matchers
7. Mockito
8. Mock() y @Mock
9. Stubbing
10. Número de llamadas a función
11. Lanzando Excepciones
12. Spy

# Introducción

¿Por dónde empezamos?

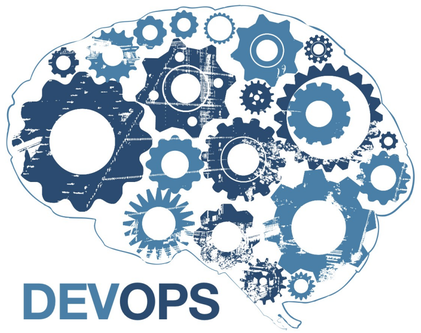



JUnit es el estándar de facto en las pruebas de Unidad Java

Además dispone de Integraciones con la mayor parte de Frameworks de Pruebas

# Introducción

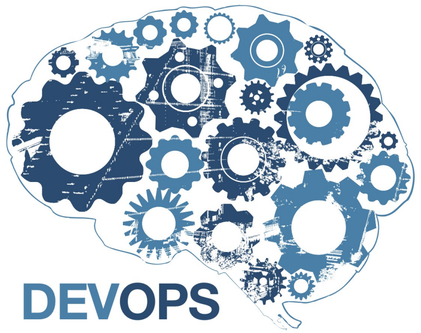
¿Por dónde empezamos?




```
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>
```

# Introducción

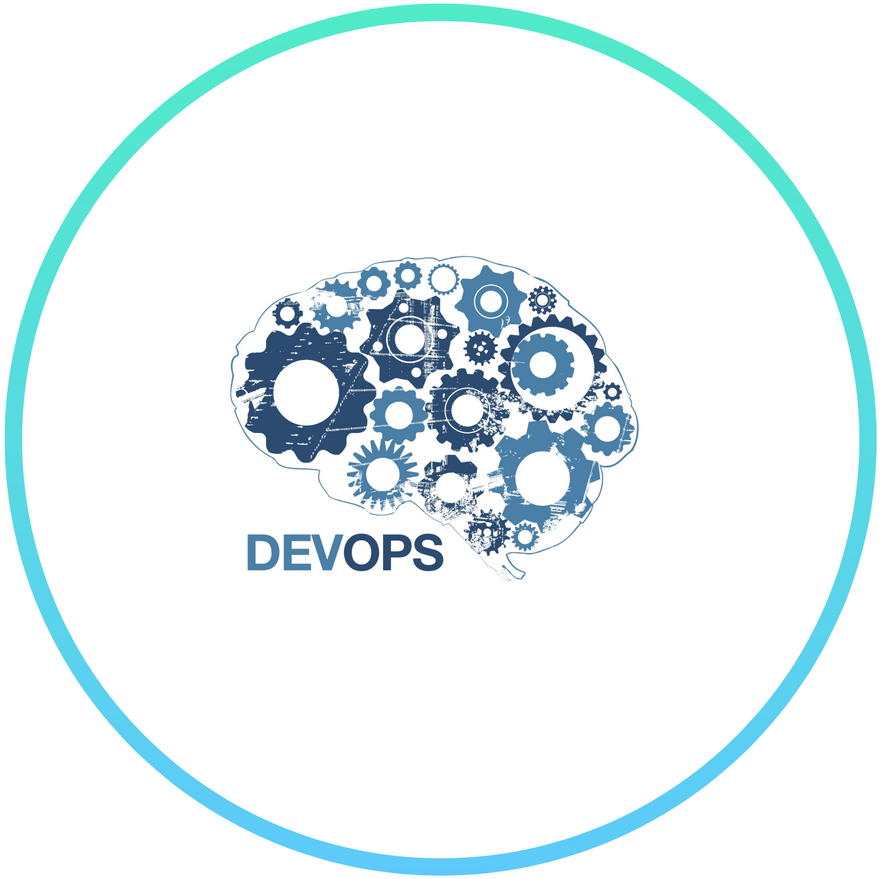
¿Por dónde empezamos?




Para lanzar las pruebas con maven ejecutaremos
mvn test

# Estructura de Fichero

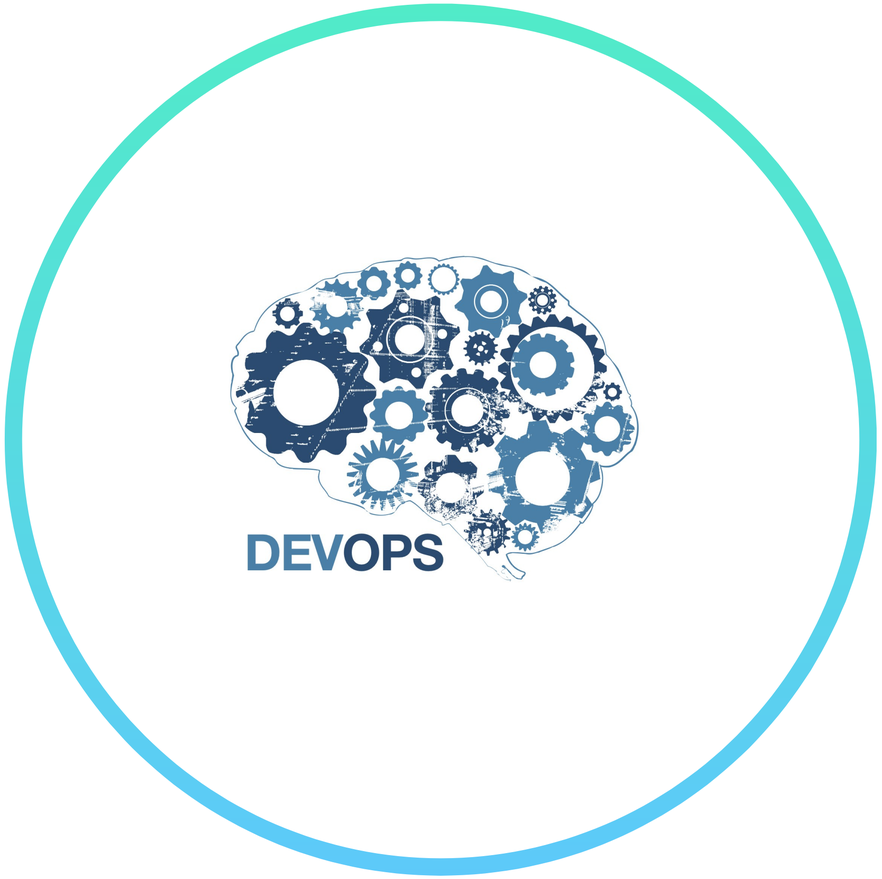
¿Cómo va el fichero?




```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class CalculatorTest {
 @Test
 public void evaluatesExpression() {
 Calculator calculator = new Calculator();
 int sum = calculator.evaluate("1+2+3");
 assertEquals(6, sum);
 }
}
```

# Fixtures

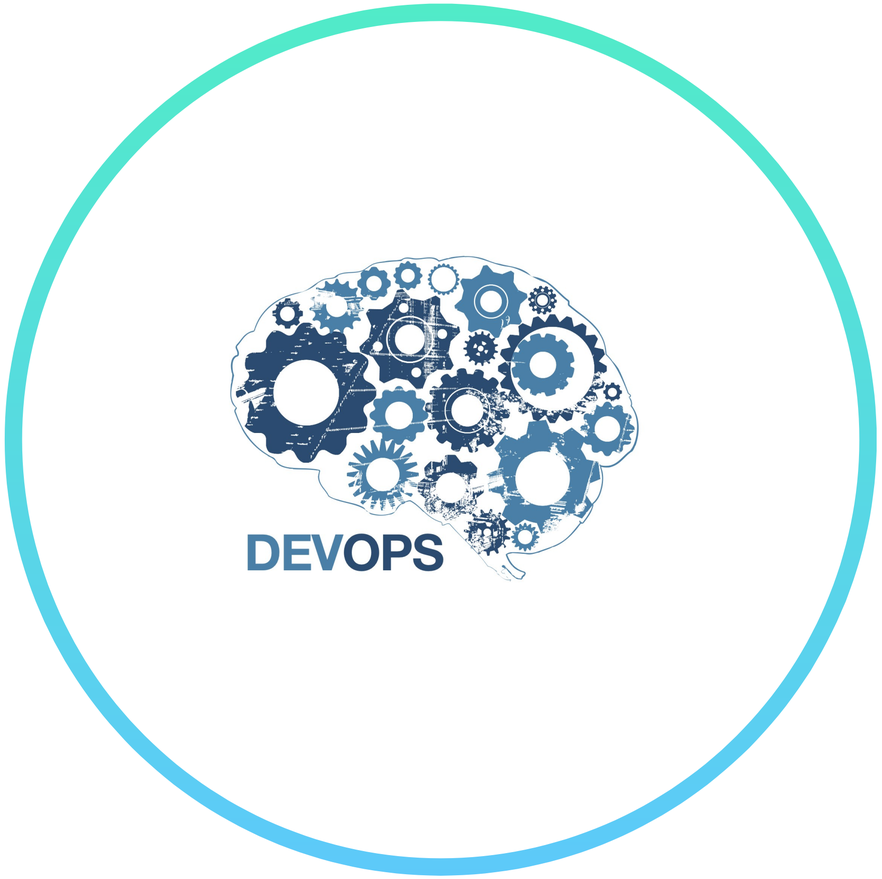
¿Cómo se ejecutan las pruebas?




Las Fixtures nos van a permitir marcar cuando deben ejecutarse los métodos definidos en la clase de pruebas

# Fixtures

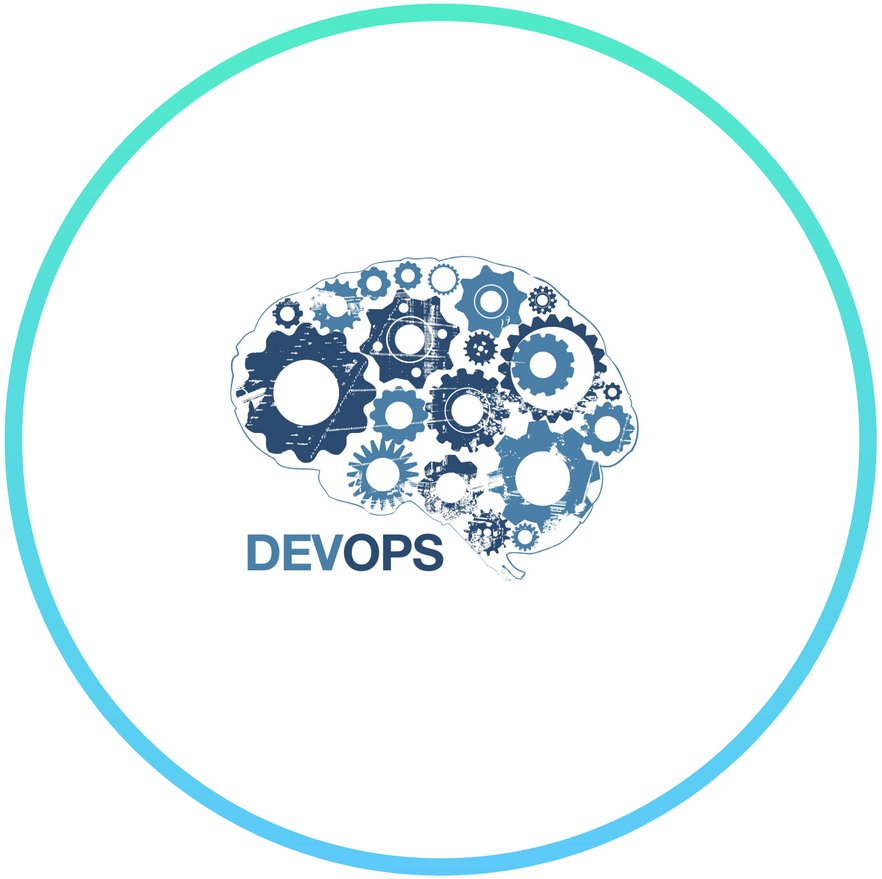
¿Cómo se ejecutan las pruebas?




@BeforeClass setUpClass // antes de la clase
@Before setUp // antes de cada test
@After tearDown //después de cada test
@Test test2() // test normal
@Test test1() // test normal
@AfterClass tearDownClass //después de la clase

# Test Suite

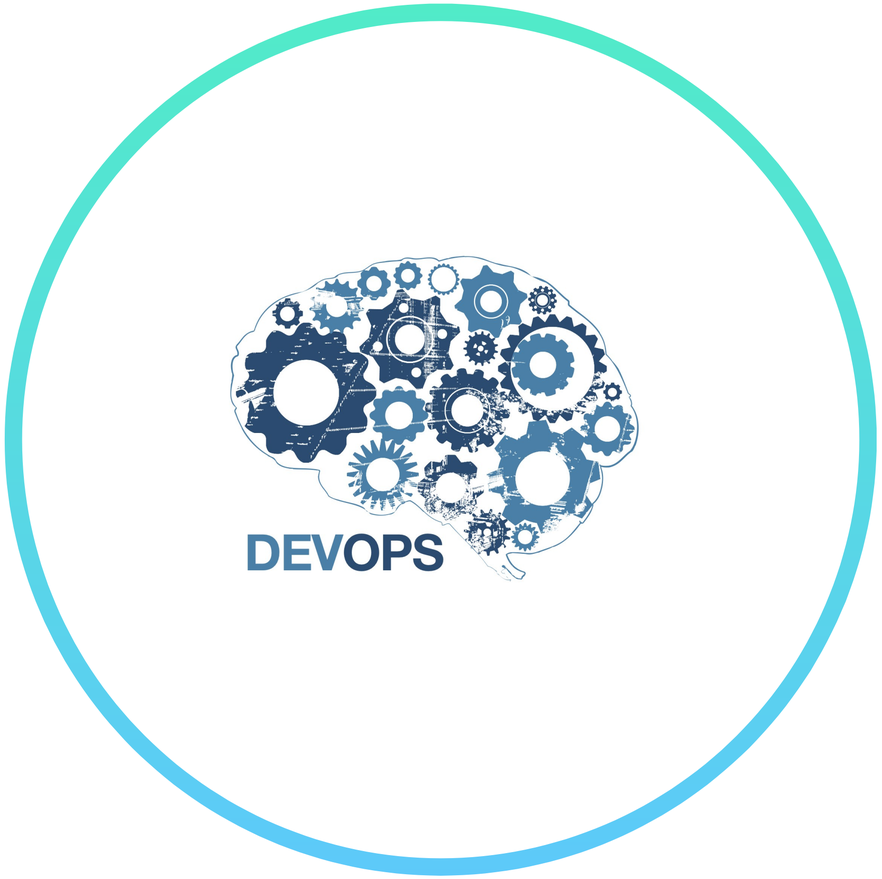
¿Cómo se agrupan las pruebas?




@Suite nos va a permitir gestionar Test de manera unificada, para agrupar varias clases

# Test Suite

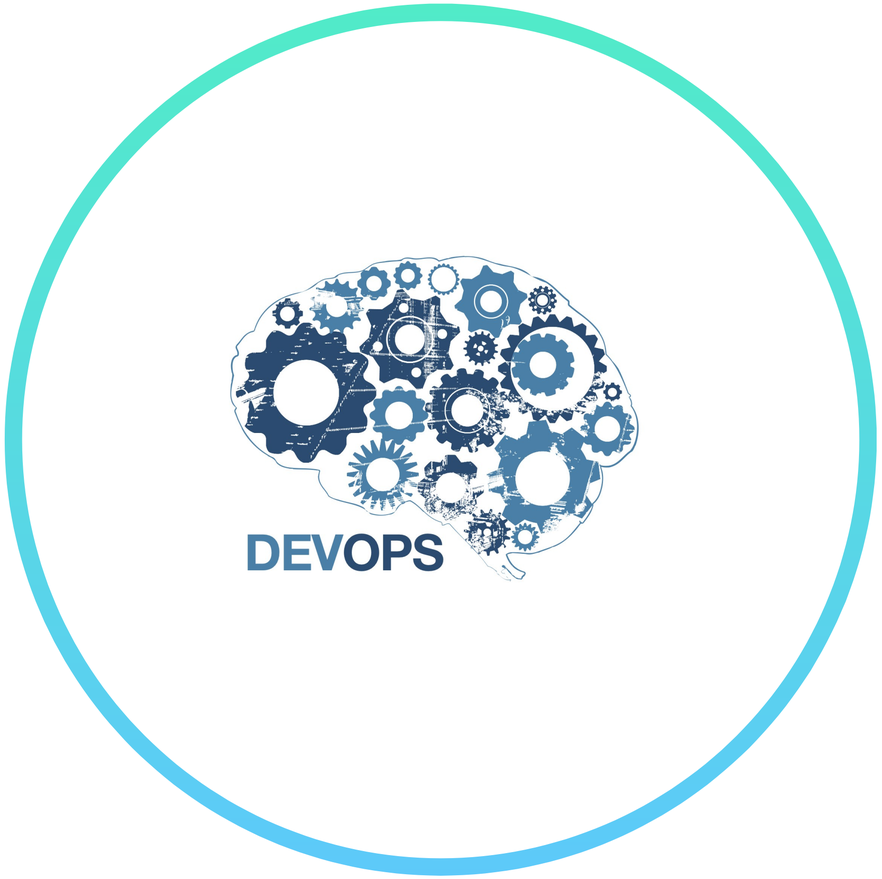
¿Cómo se agrupan las pruebas?




```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses({
 SuiteOne.class,
 SuiteTwo.class
})
public class TestSuite {
}
```

# Asserts

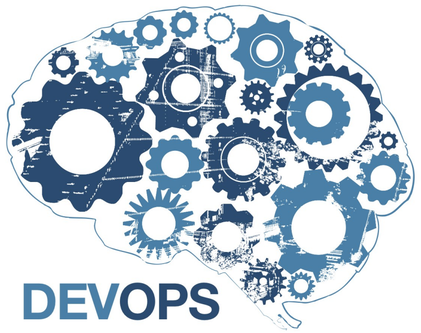
¿Cómo se comprueban los datos?




Permiten comprobar si se cumplen algunas condiciones
https://github.com/junit-team/junit/wiki/Assertions

# Asserts

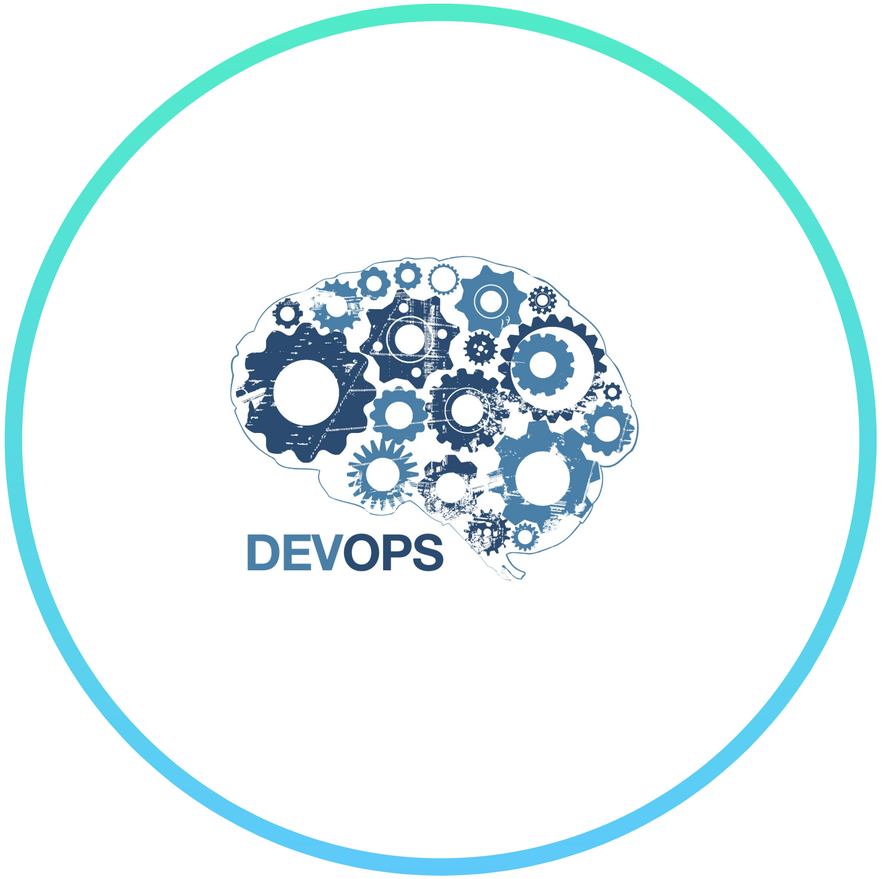
¿Cómo se comprueban los datos?




assertEquals
assertTrue
assertFalse
AssertNull
assertNotNull
assertSame
assertNotSame
assertArrayEquals
assertThat

# Matchers

¿Cómo se comprueban los datos?



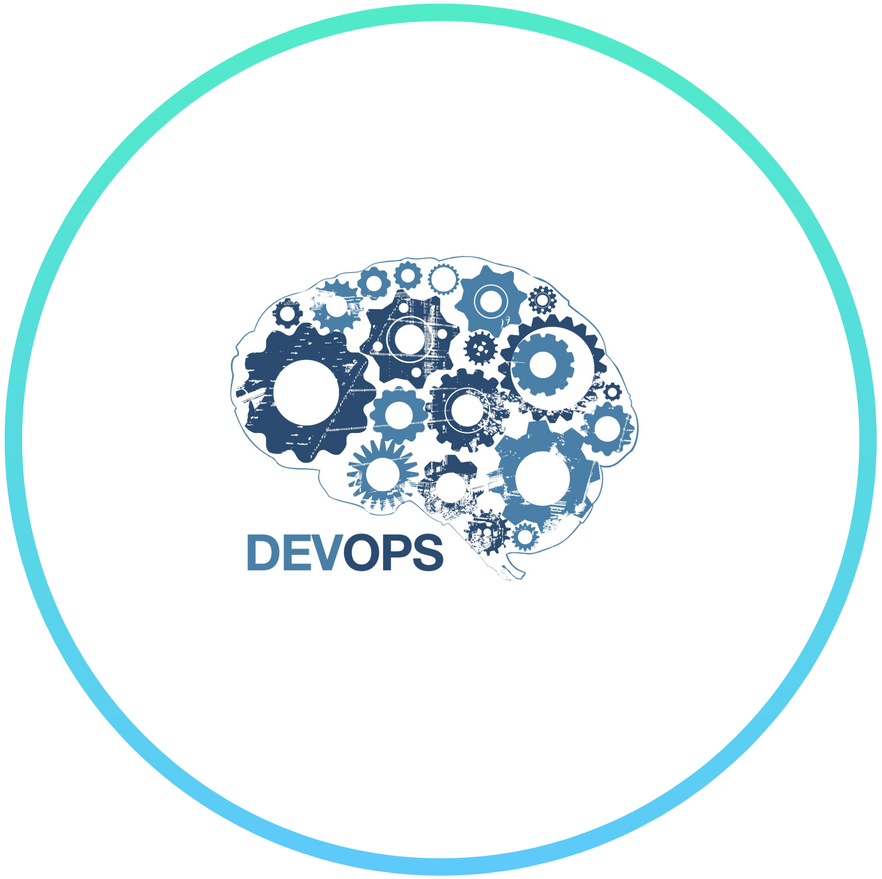

Permite realizar

comparaciones entre Objetos

https://objectpartners.com/2013/09/18/the-benefits-ofusing-assertthat-over-otherassert-methods-in-unit-tests/

# Mockito

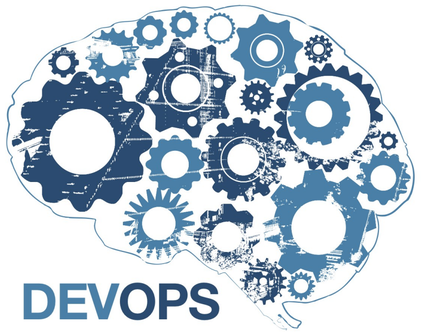
¿Porqué un mojito integra?




Dentro de las pruebas de integración siempre es necesario simular uno de los objetos relacionados para ver cómo se comporta uno con el otro

# Mockito

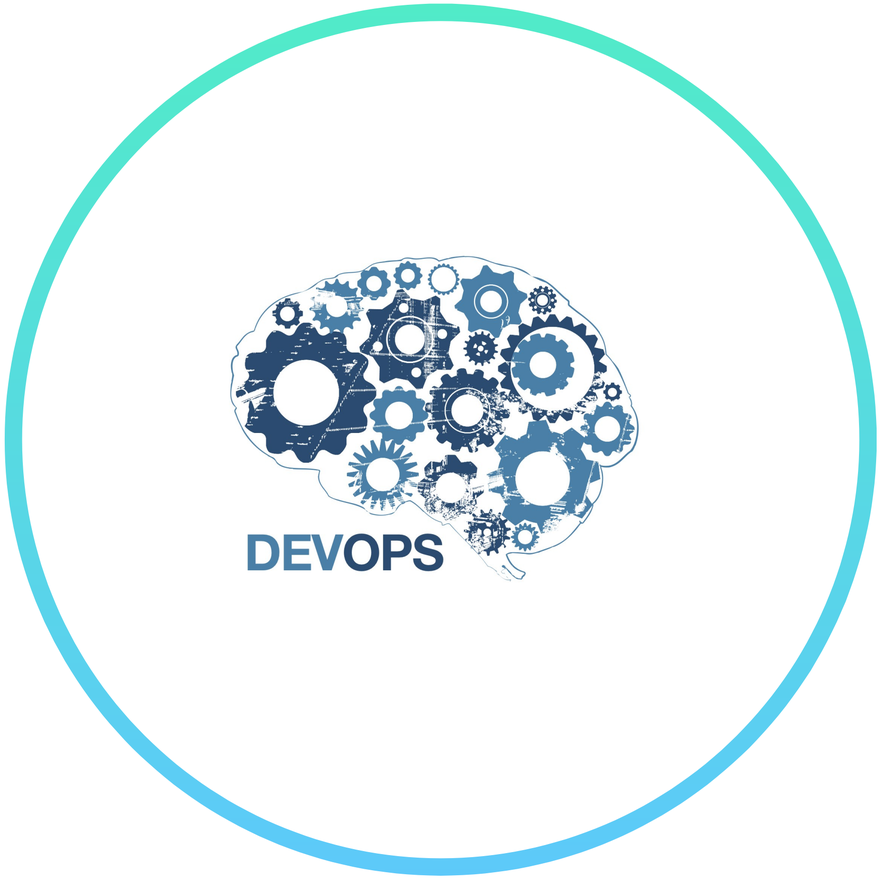
¿Porqué un mojito integra?




Para simular estos objetos se utilizan las bibliotecas de Mock
Una de las principales es Mockito
http://mockito.org/

# Mockito

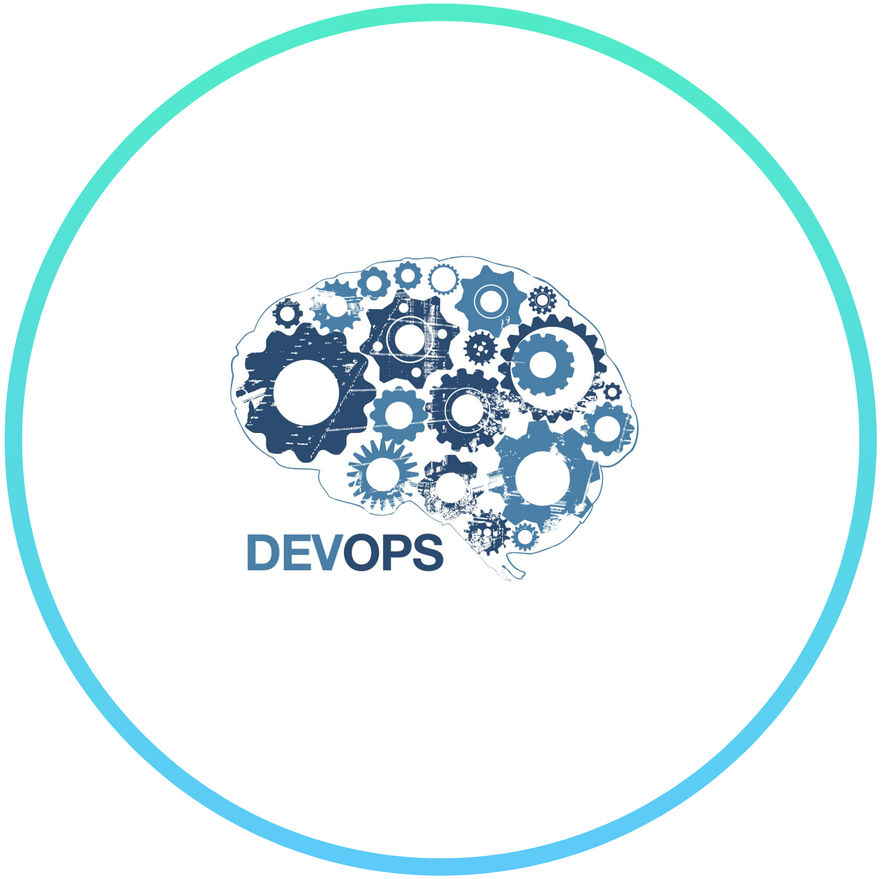
¿Porqué un mojito integra?




```
<dependency>
<groupId>org.mockito</groupId>
<artifactId>mockito-all</artifactId>
<version>1.9.5</version>
</dependency>
```

# Mock() y @Mock

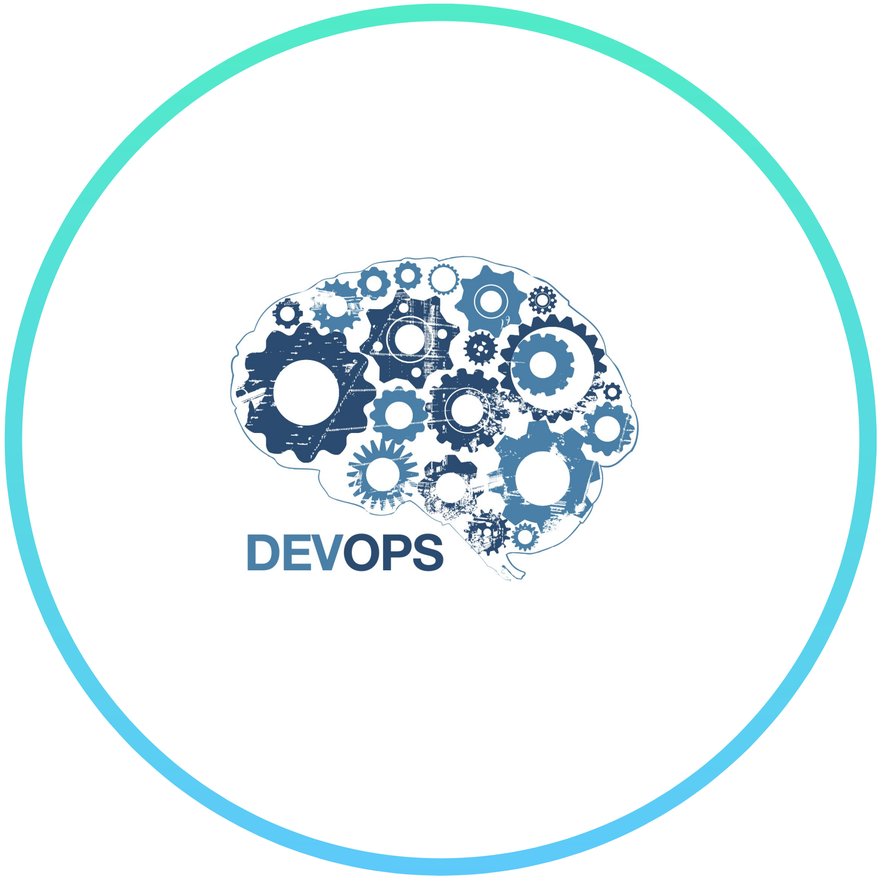
¿Quien engaña a una clase?




Tratan de crear objetos ficticios para simular el comportamiento de una Clase definida

# Mock() y @Mock

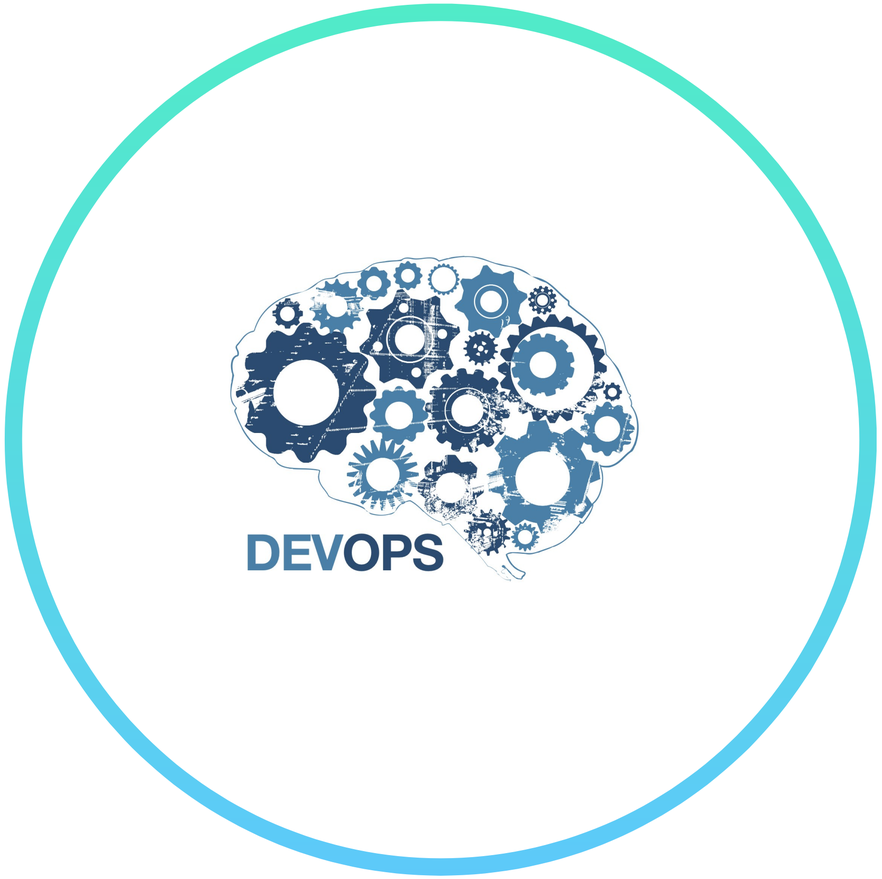
¿Quien engaña a una clase?




```
List mockedList = mock(List.class);
mockedList.add("one");
verify(mockedList).add("one");
```

# Mock() y @Mock

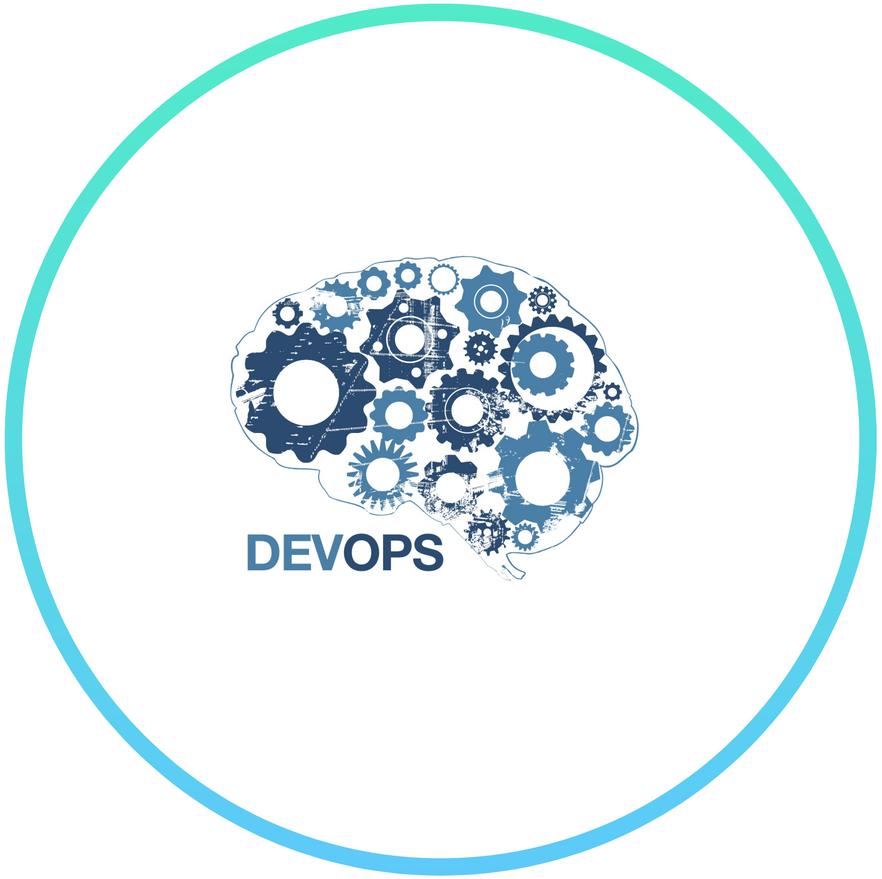
¿Quien engaña a una clase?




```
@Mock
 private LinkedList<String> listado;
 @Before
 public void initMocks() {
 MockitoAnnotations.initMocks(this);
 }
 @Test
 public void testAnnotation(){
 listado.add("Uno");
 verify(listado).add("Uno");
 }
```

# Stubbing

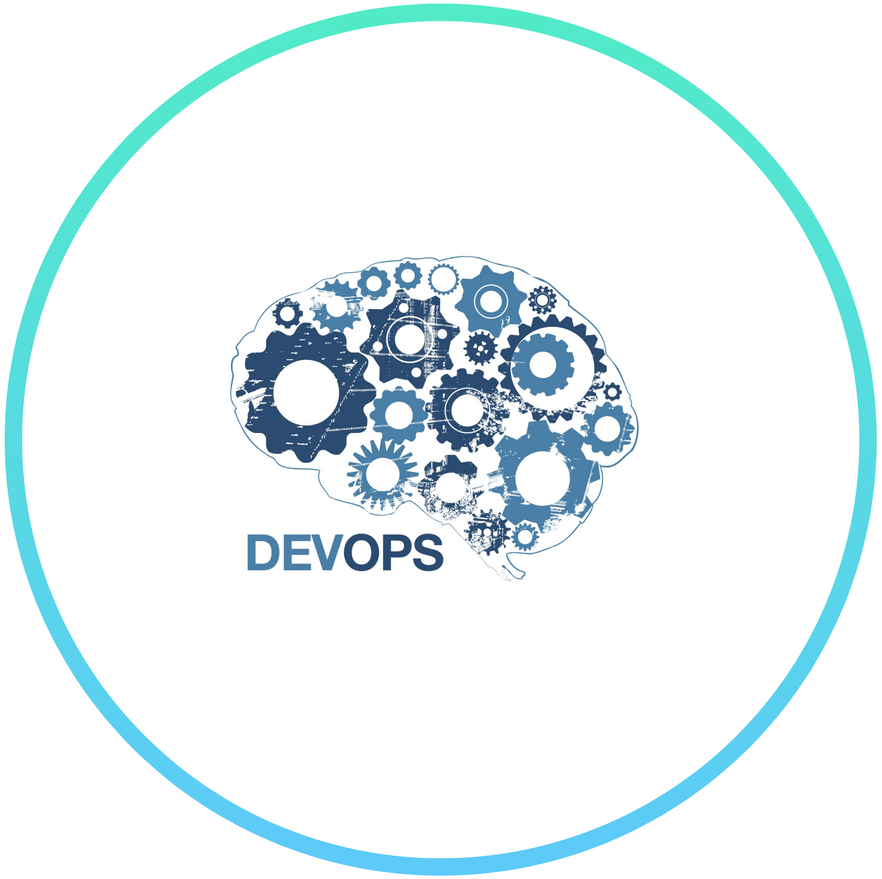
¿Quien engaña a un método?




Es un método que permite la simulación de la llamada a métodos del objeto Mock

# Stubbing

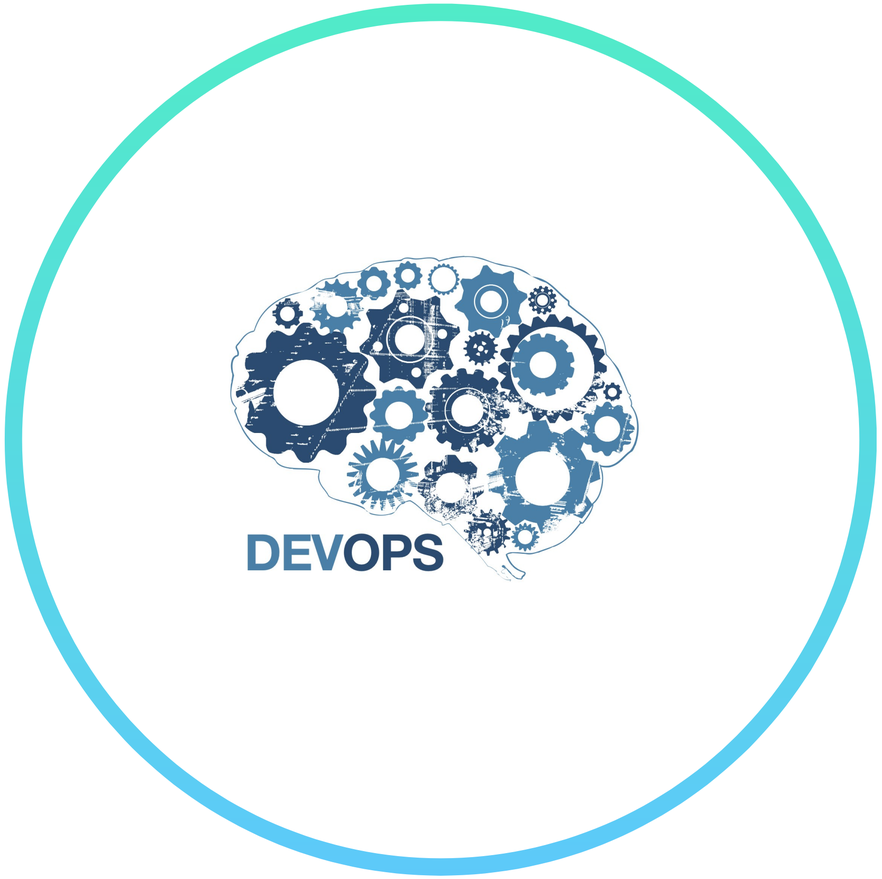
¿Quien engaña a un método?




```
//You can mock concrete classes, not only interfaces
LinkedList mockedList = mock(LinkedList.class);
//stubbing
when(mockedList.get(0)).thenReturn(“first");
//following prints "first"
System.out.println(mockedList.get(0));
//following prints "null" because get(999) was not stubbed
System.out.println(mockedList.get(999));
verify(mockedList).get(0);
```

# Número de llamadas

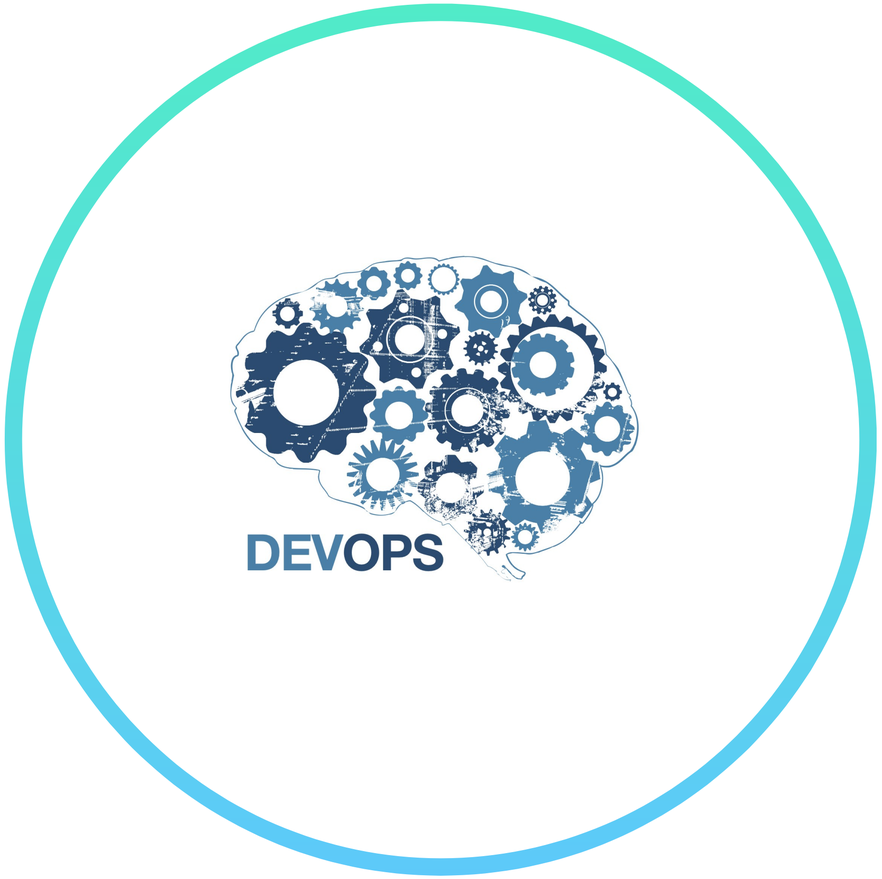
¿Cuantas veces llama el cartero?




Hay veces que es necesario saber si se ha llamado un determinado número de veces a un método para eso se puede utilizar otro parámetro al verify

# Número de llamadas

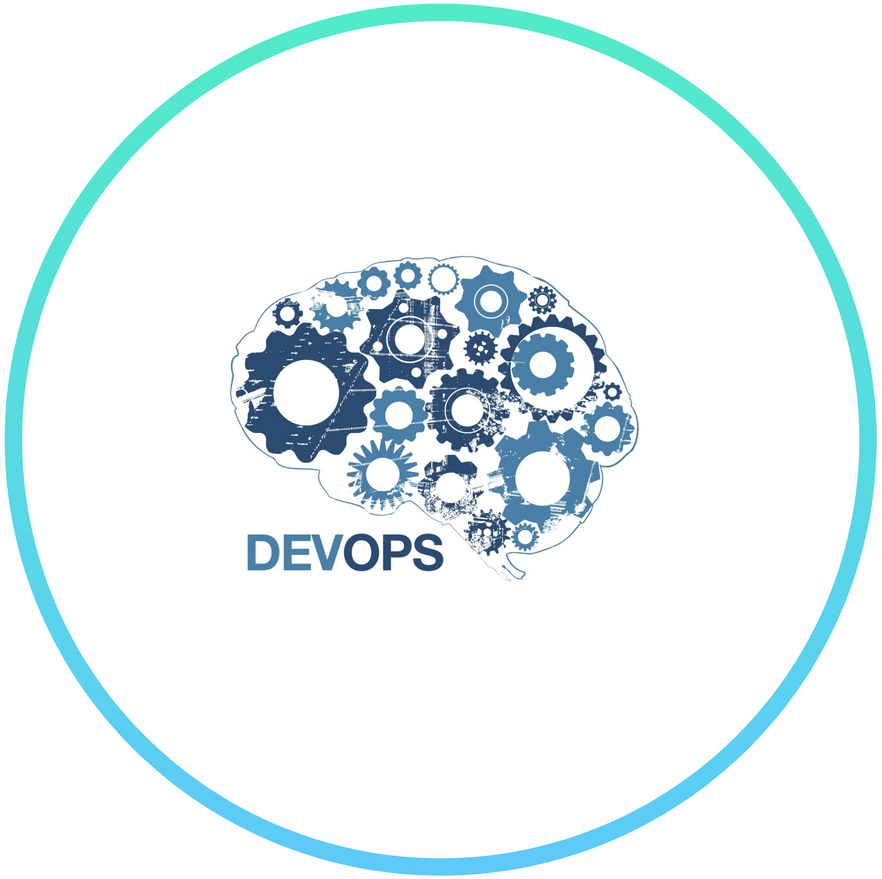
¿Cuantas veces llama el cartero?




```
//using mock
mockedList.add(“once");
verify(mockedList, times(1)).add("once");
mockedList.add("twice");
mockedList.add(“twice");
verify(mockedList, atLeastOnce()).add(“twice");
//verify that method was never called on a mock
verify(mockedList, never()).add("two");
```

# Lanzando Excepciones

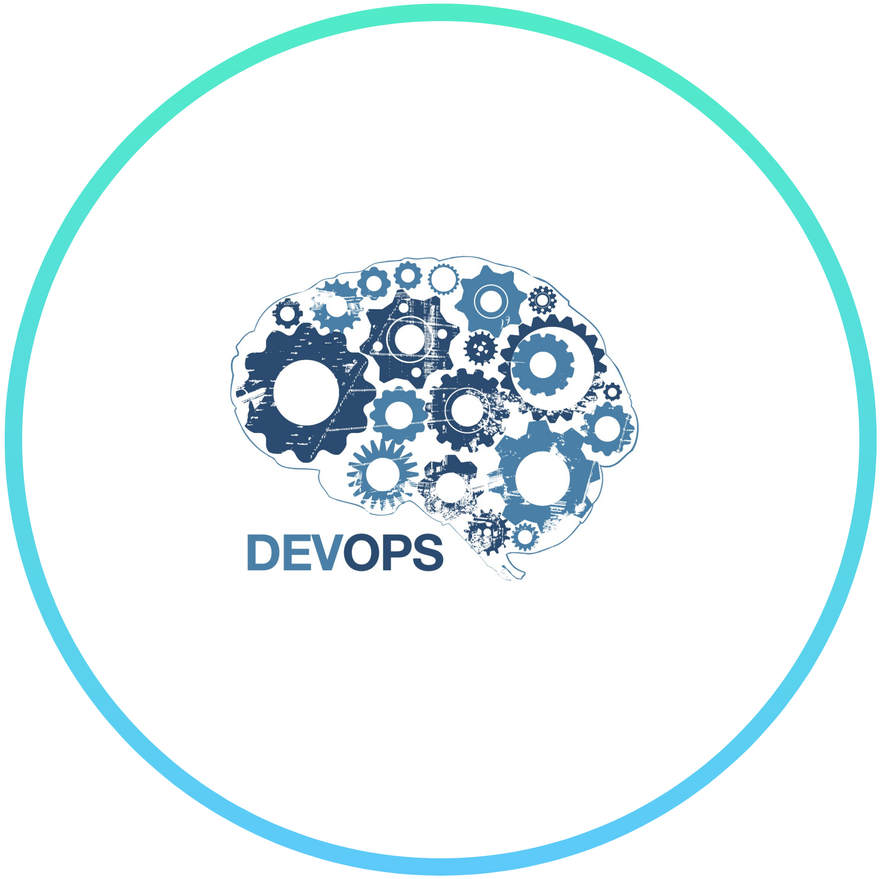
¡It’s a TRAP!




```
@Test(expected = RuntimeException.class)
 public void testWithException(){
 LinkedList mockedList = mock(LinkedList.class);
 doThrow(new
RuntimeException()).when(mockedList).clear();
 //following throws RuntimeException:
 mockedList.clear();
}
```

# Spy

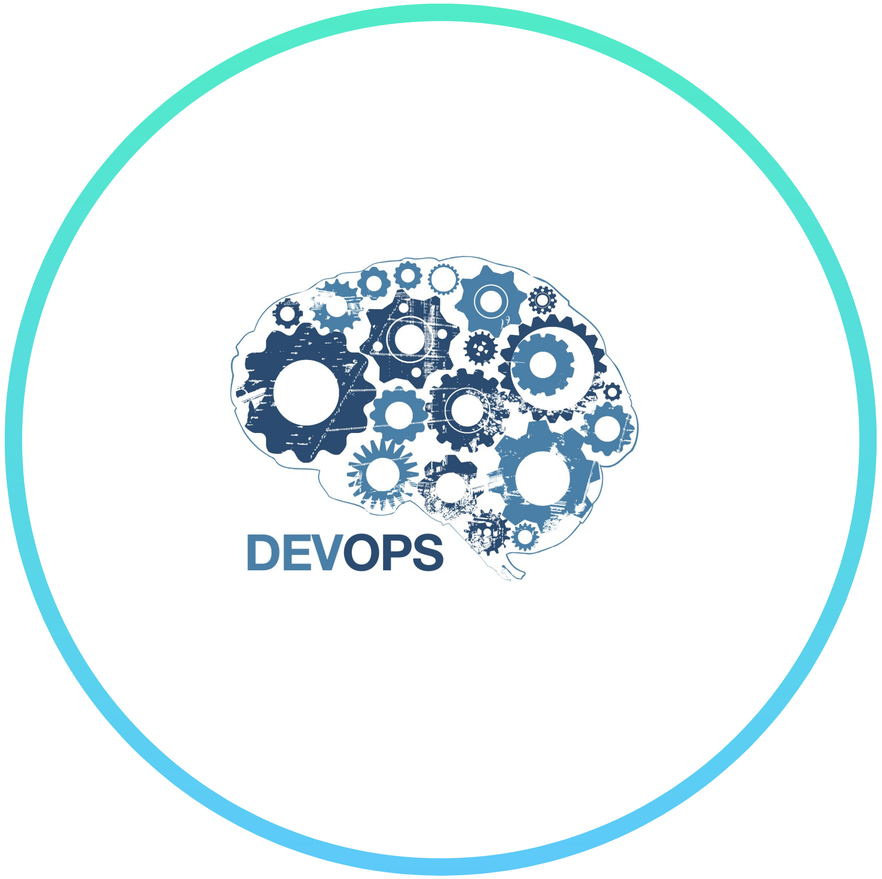
¡Saca el clonador de llaves!




Al espiar un objeto permite utilizar los métodos reales definidos en el mismo

# Spy

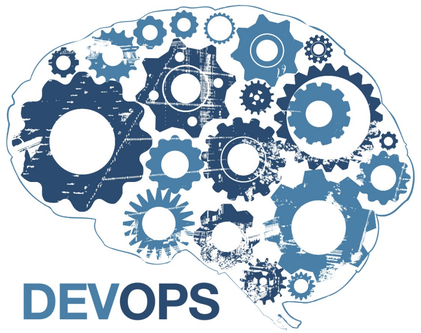
¡Saca el clonador de llaves!




```
List list = new LinkedList();
 List spy = spy(list);
 //optionally, you can stub out some methods:
 when(spy.size()).thenReturn(100);
 //using the spy calls *real* methods
 spy.add("one");
 spy.add("two");
 //prints "one" - the first element of a list
 System.out.println(spy.get(0));
 //size() method was stubbed - 100 is printed
 System.out.println(spy.size());
```

# Conclusiones

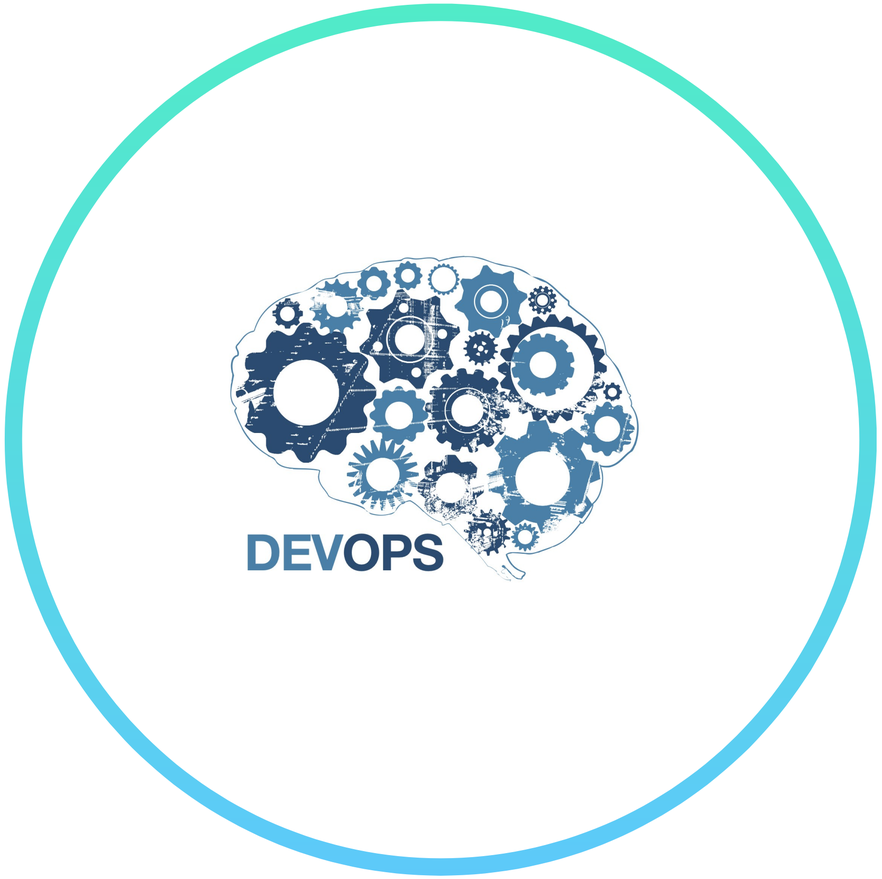
¿Qué podemos sacar en claro?




- Hemos definido pruebas
- Hemos visto cómo lanzar las pruebas
- Hemos codificado pruebas de unidad
- Hemos visto cómo aislar un componente para probar otro
- Hemos visto cómo escribir pruebas de integración

# Referencias

¿Fuentes de información?




Cosecha propia :)

# CURSOS DE DESARROLLO
# DAVID VAQUERO
# LICENCIA CC-BY-SA-NC 4.0

info@cursosdedesarrollo.com



http://cursosdedesarrollo.com