
Mockito

¿Qué es Mockito?

Mockito es un framework de test orientado a Test Double.

¿Qué es Mockito?

Mockito es un framework de test orientado a Test Double.

CONCEPTOS CLAVE!

¿Qué es Mockito?

Mockito es un framework de test orientado a Test Double

CONCEPTOS CLAVE!

SUT (System under test)

Cuando estamos escribiendo tes unitarios el SUT es la clase que estamos testeando.

DOC (Depended-on Component)

Son las dependencias conocidas por el SUT.

¿Qué es Mockito?

Mockito es un framework de test orientado a Test Double

¿QUE ES UN TEST DOUBLE?

CONCEPTOS CLAVE!

SUT (System under test)

Cuando estamos escribiendo tes unitarios el SUT es la clase que estamos testeando.

DOC (Depended-on Component)

Son las dependencias conocidas por el SUT

¿Qué es Mockito?

Mockito es un framework de test orientado a Test Double.

¿QUE ES UN TEST DOUBLE?

Un Test double es realizar un test unitario en el cual cualquier conocimiento hacia afuera del SUT sea un objeto falso.

Existen distintos tipos de comportamiento para los Test Doubles (Stub, Mock, Spy, Dummy).

CONCEPTOS CLAVE!

SUT (System under test)

Cuando estamos escribiendo tes unitarios el SUT es la clase que estamos testeando.

DOC (Depended-on Component)

Son las dependencias conocidas por el SUT.

Para mas info **xUnit Test Patterns: Refactoring Test Code. Gerard Meszaros.**

1.
Instalación!

Instalación!

Ultima versión

Para esta versión lo único que se necesita es descargarla e importarla.

www.mockito.org

Instalación!

Ultima versión

Para esta versión lo único que se necesita es descargarla e importarla.

www.mockito.org

Versión disponible en GoogleCode

Para esta versión lo único que se necesita es descargarla e importarla.
https://code.google.com/p/mockito/


Fun with
mock!!!

Fun with Mock!

Creación

```
ClassName varName = mock(ClassName.class);
@Mock private ClassName varName;
```

Set un valor

```
when(methodCall).thenReturn(value)
                .thenReturn(value, value)
                .thenThrow(throwableClasses)
```

Verificar

```
verify(mock).methodCall
verify(mock, VerificationMode).methodCall
verifyNoMoreInteractions(mock)
verifyZeroInteractions(mock)
```

VerificationMode

atLeastOne()	times(int)
atLeast(int)	timeout(long)
atMost(int)	never()

http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html

Stub
Proporciona respuestas pre-programadas a las llamadas realizadas durante la prueba.

Dummy
Son objetos que se pasan por allí, pero nunca se utilizan realmente. Por lo general, sólo son utilizadas para rellenar listas de parámetros.

Mock
Es un Stub con la capacidad de poder verificarse.

Spy
Es un Mock pero que llama realmente a los métodos del objeto real.

Ejemplo TDD

Dominio - Calculadora de Arrays

Se nos pide desarrollar una calculadora capaz de operar dos listas y que retorne una tercera, ésta aplicada con un operador determinado.

- Los operadores pueden ser: suma, resta y multiplicación (tener en cuenta que en algún momento tiene que ser posible agregar un operador más).
- Suponer que las listas son del mismo tamaño.

```
public class TestSuma{
        @Mock private List<Integer> mockList1;
        @Mock private List<Integer> mockList2;
        private Suma suma;

        @Before
        public void setUp(){
                MockitoAnnotations.initMocks(this);
                suma = new Suma();
        }
        @Test
        public void testOperar() {
                when(mockList1.size()).thenReturn(1);
                when(mockList1.get(anyInt())).thenReturn(1);
                when(mockList2.get(anyInt())).thenReturn(2);
                List<Integer> ret = Arrays.asList(3);

                assertEquals(suma.operar(mockList1, mockList2), ret);
        }

        @Test
        public void testOperacion(){
                assertEquals(5, suma.operacion(3,2));
        }

}
```

```
public class Operacion{

    public List<Integer> operar(List<Integer> list1, List<Integer> list2){
        List<Integer> ret = new ArrayList<Integer>();
        for(int i = 0; i < list1.size(); i++){
            ret.add(this.operacion(list1.get(i), list2.get(i)));
        }
        return ret;
    }

    public abstract Integer operacion(Integer numero1, Integer numero2);

}

public class Suma extends Operacion{

    @Override
    public Integer operacion(Integer numero1, Integer numero2) {
        return numero1 + numero2;
    }

}
```

```
public class TestCalculadoraDeArray{

        private CalculadoraDeArray calculadora;
        @Mock private Operacion mockOperacion;

        @Before
        public void setUp(){
                calculadora = new CalculadoraDeArray();
                MockitoAnnotations.initMocks(this);
        }

        @Test
        public void testCalcular() {
                calculadora.calcular(mockOperacion, anyList(), anyList());
                verify(mockOperacion).operar(anyList(), anyList());
        }

}
```

```
public class CalculadoraDeArray{

	public List<Integer> calcular(Operacion operacion, List<Integer> list1, List<Integer> list2) {
		return operacion.operar(list1, list2);
	}

}
```

Gracias!

¿Alguna pregunta?