

FrameworkJUnit

Santa Tecla
parqueNaturalSantaTecla@gmail.com
Version 0.0.1

Índice

Justificación: ¿Por qué?

Definición: ¿Qué?

Objetivos: ¿Para qué?

Descripción: ¿Cómo?

Pruebas Unitarias

Clases de Pruebas

Método de Pruebas

Reusabilidad

Excepciones del SUT

Clase de Pruebas Parametrizada

Expiración del Tiempo de Ejecución

Conjuntos de Pruebas

Pruebas de Componentes

Pruebas de Integración

Pruebas de Sistema

Bibliografía

MetACLases

Anotaciones

Aserciones

Justificación: ¿Por qué?

- **No es necesario ningún framework para la gestión de pruebas automáticas:**
 - Instanciar objetos que contengan el SUT
 - Enviar mensajes para ejecutar el SUT
 - Elevar una excepción si se verifica que la respuesta del SUT no es igual a la esperada.
- **... pero es muy conveniente para facilitar y acelerar su gestión**

<i>Ejemplos sin Junit</i>	
• Producción: <u>ticTacToe.utils.Coordinate</u> (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/main/java/ticTacToe/utils/Coordinate.java)	• 1
• Test: <u>auxiliar.withoutJUnit.CoordinateTest</u> (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/auxiliar/withoutJUnit/ClosedIntervalTest.java)	• 1

Definición: ¿Qué?

- **Framework sencillo** desarrollado por **Kent Beck** y **Erich Gamma** para escribir pruebas repetibles.
 - Es una instancia de **XUnit**, arquitectura para *framework* de pruebas unitarias: **CppUnit**, **PHPUnit**,...
 - Basado en: **MetACLases**, **Anotaciones** y **Aserciones**
 - Incluye:
 - **Visualizadores de resultados**, *runners*, en modo **_texto**, **gráfico** ...
 - **Plugins** para principales *IDEs*, **Eclipse**, **NetBeans**, **VisualStudio**, ...
 - **Integración con gestores de proyectos**: *Ant*, *Maven*, *Gradle*, ...

Objetivos: ¿Para qué?

- Gestionar eficientemente el código de pruebas unitarias, de componentes, integración y sistemas para estilos arquitectónicos como MVP-PM
 - Facilitar la reusabilidad específica del código de pruebas
 - Facilitar la legibilidad, escritura/lectura, con funciones estáticas especializadas para frente a la sentencia *assert* original del lenguaje
 - Agrupar jerarquias de Conjuntos de Clases de Pruebas (*@TestSuite*) que automatiza la ejecución de la totalidad de las Pruebas de Regresión
 - Categorizar de Clases de Pruebas (*@Categories*) que automatiza la ejecución de subconjuntos de la totalidad de los Casos de Prueba, para las pruebas **Alfa**, **Beta** o **Humo**

Descripción: ¿Cómo?

Pruebas Unitarias

Clases de Pruebas

- Las **Clases de Pruebas** son clases normales (**POJO**) del lenguaje con atributos, métodos públicos y privados, miembros estáticos... relacionadas con otra clases como las del SUT que se ejercite, auxiliares (*Factory*, *Builder*,...) sin ninguna limitación excepto que su **nombre debe terminar con el sufijo "Test"**

Ejemplos: patrón Factory Method en pruebas	
<u>ticTacToe.models.InheritBoardTest</u> (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/InheritBoardTest.java)	<u>ticTacToe.mod</u>
<ul style="list-style-type: none">La Organización de Clases de Pruebas mantiene el Código de Producción separado del Código de Pruebas	
para el Código de Producción con un Jerarquia de Paquetes de Paquetes de la Arquitectura del Software	el Código de Pruebas Unitarias con una Jerarquia de Paquetes paralela a la Arquitectura del Software del Código de Producción
<ul style="list-style-type: none">src/main/java<ul style="list-style-type: none">xxx.yyy.zzz.Axxx.yyy.zzz.B...	<ul style="list-style-type: none">src/test/java<ul style="list-style-type: none">xxx.yyy.zzz.ATestxxx.yyy.zzz.BTest...
src/main/resources	src/test/resources
para los Recursos de Producción : imágenes de UI, ficheros de configuración,... con la estructura que se considere más adecuada	para los Recursos de Pruebas : ficheros de datos para alimentar las pruebas,... con una Jerarquia de Paquetes paralela a la jerarquia Recursos de Producción para facilitar biunivocamente la localización de recursos de una prueba dada

Ejemplos	
<u>Producción</u> (https://github.com/MasterCloudApps/1.ADCS.pruebas/tree/master/src/main)	<u>Pruebas</u> (https://github.com/MasterCloudApps/1.ADCS.pruebas/tree/master/src/test)

Método de Pruebas

Declaración	Ejecución
<div><pre><code>@Test 1 public void test <nombre>() { 2 ... }</code></pre></div> <div><p>1 decorado con la anotación @Test</p><p>2 siempre público sin parametros y no devuelve nada</p><ul style="list-style-type: none">el nombre debe determinar de forma inequívoca lo que se está probando del SUT, alcanzando la longitud que sea necesaria comenzando por el prefijo "test".Por ejemplo, <i>testLenght</i>.En caso de colisión por sobrecarga del método del SUT, añadir al nombre los nombres de los parámetros que los distinguen.<ul style="list-style-type: none">Por ejemplo, <i>testIncludesInt</i> y <i>testIncludesClosedInterval</i>.</div>	<ul style="list-style-type: none">Cuando se ejecuta una Clase de Pruebas, por cada Método de Prueba (@Test), en un orden" desconocido";<ul style="list-style-type: none">se crea un objeto de la Clase de Pruebas, a través de reflexion, metaclassesse le pasa el mensaje correspondiente al Metodo de PruebaCuando ocurre un fallo en una aserción en el código de pruebas, se aborta su ejecucionCuando ocurre un error en una sentencia del código de pruebas o del código de producción, si no se captura específicamente con la sentencia <i>try/catch/finally</i>, se aborta su ejecuciónEn cualquiera de los dos casos anteriores y cuando la prueba pasa, se continúa la ejecución con otro Método de Pruebas sobre otro objeto de la Clase de Pruebas

Cuerpo del Método de Prueba	Una triple A (<i>arrange/act/assert</i>) y una parte opcional final:
Preparacion (<i>Arrange/Seup</i>)	para la creación, relación, gestión de recursos (ficheros de datos, bases de datos,...),... de los objetos del SUT
Acción (<i>Act</i>)	para el ejercicio del SUT por el camino establecido en la prueba
Aserción (<i>Assert</i>)	para la comprobación de que el resultado esperado coincida con el resultado obtenido
Demolición (<i>TearDown</i>)	en caso necesario, para liberar los recursos que fueron necesarios y devolverlos al estado anterior facitando la idependencia de otros Métodos de Pruebas poder reutilizarlos en otro Metodos de Pruebas evitando su continua re-creación(no recomendado!!!)

Ejemplos:

ticTacToe.models.TurnTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/TurnTest.java)	tic
ticTacToe.models.TurnWithAttributesTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/TurnWithAttributeTest.java)	tic

Reusabilidad

- Para ejecutar automáticamente el código común de los **Métodos de Prueba**, a través de anotaciones en métodos:
 - **@Before**, para la preparación (**Arrange**) de un conjunto igual de SUT, accesorios, recursos, ...
 - **@After**, para la demolición (**TearDown**) de un conjunto igual de SUT, accesorios, recursos, ...
 - **Ambos son de instancia sin parámetros y no devuelven nada**
- Para ejecutar automáticamente el código previo y posterior a la ejecución de todos los **Metodos de Pruebas**:
 - **@Beforeclass**, para la preparación (**Arrange**) del mismo conjunto de SUT, accesorios, recursos, ...
 - **@Afterclass**, para la demolición (**TearDown**) del mismo conjunto de SUT, accesorios, recursos, ...
 - **Ambos son estáticos sin parámetros y no devuelven nada**

Ejemplos:

auxiliar.junit.ExecutionReusabilityTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/auxiliar/junit/ExecutionReusabilityTest.java)	ticTacT
---	-------------------------

Aserciones Avanzadas

Generación	Descripción
• 1ª Generación	<ul style="list-style-type: none"> • Del lenguaje de programación Java <pre>assert <expresión>;</pre>
• 2ª Generación	<ul style="list-style-type: none"> • Del <i>framework</i> JUnit (Baeldug (https://www.baeldung.com/junit-assertions), buen tutor!) <pre>assertEquals(<expected>, <actual>); assertArrayEquals(<expected>, <actual>); assertNull(<expected>, <actual>); assertNotNull(<expected>, <actual>); assertSame(<expected>, <actual>); assertNotSame(<expected>, <actual>); assertTrue(<expected>, <actual>); assertFalse(<expected>, <actual>); fail(<expected>, <actual>);</pre> <ul style="list-style-type: none"> • todas las aserciones <ul style="list-style-type: none"> ◦ aceptan como argumentos, el esperado según los requisitos y el actual obtenido en el ejercicio del SUT, todos los tipos primitivos, Objects y arrays de éstos. ◦ tienen un parámetro opcional inicial de tipo String para especificar un mensaje: <pre>assertEquals(<message>, <expected>, <actual>);</pre>
• 3ª Generación	<ul style="list-style-type: none"> • Del <i>framework</i> Hamcrest (Baeldug (https://www.baeldung.com/java-junit-hamcrest-guide), buen tutor!) ◦ Es un ejemplo paradigmático del patrón Intérprete: dado un lenguaje, define una representación para su gramática junto con un intérprete del lenguaje ... implantación sin métodos static (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/auxiliar/junit/Matchers.java) y con métodos static (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/auxiliar/junit/StaticMatchers.java) ▪ Matcher, el "Emparejador", es la clase padre de la jerarquía del lenguaje de consulta: <i>is</i>, <i>hasProperty</i>, <i>samePropertyValuesAs</i>, <i>empty</i>, <i>equalTo</i>, <i>arrayWithSize</i>, ... <pre>assertThat(<expected>, Matcher matcher); assertThat(Matcher matcher, <actual>);</pre>

- Objetivos de Hamcrest

• Aumentar la legibilidad		
<pre>assertEquals(expected, actual)</pre>	JAVA	<pre>assertThat(actual, is(equalTo(expected)));</pre>
• Mejorar los mensajes		
<pre>assertTrue(expected.contains(actual));</pre>	JAVA	<pre>assertThat(actual, containsString(expected));</pre>
• Imponer seguridad de tipos		
<pre>assertEquals("abc", 123);</pre>	JAVA	<pre>assertThat(123, is("abc"));</pre>
• Aumentar la flexibilidad		

- Objetivos de Hamcrest	
<pre>assertTrue("test".contains("x") && "test".contains("y"));</pre>	JAVA
<pre>assertThat("test", allOf(containsString("x"), containsString("y")));</pre>	JAVA
Ejemplos:	auxiliar.junit.AssertThatTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/auxiliar/junit/AssertThatTest.java)

Excepciones del SUT

Con excepciones	Con la anotación <code>@Test</code>
<pre>@Test public void testX() { try{ //arrange //act with XException fail(); ¹ } catch(XException e) { ² } catch (Exception e) { fail(); ¹ } }</pre> <p>¹ Tras el ejercicio del SUT que eleva la excepción, capturar la excepción esperada sin ninguna acción pero</p> <p>² fallando en ausencia de la excepción o si fuera de otro tipo</p>	<pre>@Test(expected = XException) ¹ public void testX() { //arrange //act with XException }</pre> <p>¹ Especificando la clase de excepción esperada en su atributo <code>expected</code></p>

Ejemplos:
ticTacToe.models.CoordinateWithExceptionTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/CoordinateWithExceptionTest.java)
ticTacToe.models.SetBoardWithExceptionTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/SetBoardWithExceptionTest.java)

Clase de Pruebas Parametrizada

- En muchas clases de pruebas **se escribe y se reescribe la creación de muchos objetos** del SUT para ser ejercitados en distintos métodos de pruebas
- Las **Clases de Pruebas Parametrizadas** buscan **reutilizar los mismos accesorios del SUT escritos** en todos los métodos de pruebas de la clase para cada elemento de una colección de accesorios del SUT definida **separada y extensible**.

Ejemplos:		
Repitiendo código	ticTacToe.utills.CoordinateTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateTest.java)	ticTacToe.utills.CoordinateTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateTest.java)
Sin repetir código	ticTacToe.utills.CoordinateWithoutParametrizedTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateWithoutParametrizedTest.java) con ticTacToe.utills.CoordinateObjectTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateObjectTest.java)	ticTacToe.utills.CoordinateWithoutParametrizedTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateWithoutParametrizedTest.java) con ticTacToe.utills.CoordinateObjectTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateObjectTest.java)
Parametrizada	ticTacToe.utills.CoordinateParametrizedTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateParametrizedTest.java)	ticTacToe.utills.CoordinateParametrizedTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateParametrizedTest.java)
Parametrizada sin Constructor	ticTacToe.utills.CoordinateParametrizedWithoutConstructorTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateParametrizedWithoutConstructorTest.java)	ticTacToe.utills.CoordinateParametrizedWithoutConstructorTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/CoordinateParametrizedWithoutConstructorTest.java)

Expiración del Tiempo de Ejecución

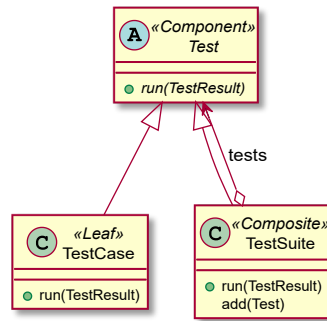
- Para **Pruebas de Rendimiento** de los **Requisitos no funcionales**.
 - Por ejemplo, el cálculo de la existencia de las Tres en Raya no puede exceder los 5msg
 - la anotación `@Test` incorpora el atributo **timeout** inicializado con el valor del **tiempo máximo en milisegundos que se concede al método para su ejecución**.
 - Una vez **expirado el tiempo**, la prueba falla.
 - En caso contrario, **dependerá del código de la prueba**

Ejemplos:	
ticTacToe.models.EfficiencySetBoardTest (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/EfficiencySetBoardTest.java)	tic7

Conjuntos de Pruebas

Jerarquías de Pruebas

- Un **Conjunto de Pruebas** es una clase que permite la **ejecucion de un conjunto** de **Clases de Pruebas**, mediante el patrón Composite donde las hojas son los *TestCase* y los compuestos son *TestSuite*
- Regla de Estilo:**
 - incorpora una **Conjunto de Pruebas** denominado **AllTest** en **cada paquete** de clases de prueba, incluyendo a todas las **Clases de Pruebas Unitarias** del paquete y
 - los **Conjuntos de Pruebas AllTest** de todos **subdirectorios**, de tal manera que **src/test/java/<proyecto>/AllTest** dispase la ejecucion de todas las pruebas acumuladas (**Pruebas de Regreción**)



```

@RunWith(Suite.class) 1
@SuiteClasses({
    <Clase1Test>.class, 3
    <Clase2Test>.class, 3
    <ConjuntoPruebas3Test>.class,
    3
    <ConjuntoPruebas4Test>.class
    3
})
class
<ConjuntoDePruebas>Test{}
  
```

Ejemplos:

- [src.test.java.ticTacToe](https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/AllTests.java)
(https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/AllTests.java)
- [src.test.java.ticTacToe.models](https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/AllTests.java)
(https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/models/AllTests.java)
- [src.test.java.ticTacToe.utills](https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/AllTests.java)
(https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/ticTacToe/utills/AllTests.java)

- Anotación **@RunWith(Suite.class)** para indicar que es un Conjunto de Pruebas
- Anotación **@SuiteClasses** cuyo argumento define los contenidos del Conjunto de Pruebas
- Clase de Pruebas, *TestCase* o *TestSuite*, añadidas al Conjunto de Pruebas

Categorías de Pruebas

- Para poder **configurar sub-conjuntos de pruebas** como en el caso de pruebas alfa, beta, humo,...

- Definir una **interfaz vacía por cada categoría**
- Decorar el **Conjunto de Pruebas a configurar** con anotaciones:
 - @RunWith(Categories.class)**
 - @IncludeCategory** ({<Categoría1>.class [, <Categoría2>.class, ...]) y/o
 - @ExcludeCategory** ({<Categoría1>.class [, <Categoría2>.class, ...])
 - cuyo argumento, en ambos casos, es la lista de clases de categoría incluidas y excluidas respectivamente
 - el **framework ejecutará todos los métodos de prueba de las categorías especificadas como incluidas que no sean de las categorías excluidas**
- Decorar los **métodos de Prueba seleccionados de cada categoría** con la anotación
 - @Category** ({<Categoría1>.class [, <Categoría2>.class, ...]})
 - cuyo argumento es la **lista de clases de categoría a las que pertenece el método**

```

public interface SlowTest {}

@RunWith(Categories.class)
@IncludeCategory(SlowTest.class)
@SuiteClasses({
    AllTests.class })
public class SlowSuiteTest {}
  
```

```

public class XTest {

    @Test
    @Category({SmokeTest.class, SlowTest.class})
    public void testOne() {
        System.out.println("testOne of XTest of
pPackage, SmokeTest!!! SlowTest!!!");
    }

    @Test
    public void testTwo() {
        System.out.println("testTwo of XTest of
pPackage");
    }

}
  
```

Ejemplos:**`src.test.java.auxiliar.junit.categories`****`(https://github.com/MasterCloudApps/1.ADCS.pruebas/tree/master/src/test/java/auxiliar/junit/categories)`**

Pruebas ignoradas

- Mediante la anotación ***@Ignored*** acompañando al **Metodo de Prueba**, el *framework* **no ejecutará** dicho caso de prueba

Pruebas de Componentes

- ¿Todas las Clases de Pruebas mostradas anteriormente son unitarias? Existe un debate:
 - No! Estrictamente son aquellas que **NO incorporan más de una clase en el SUT**, sin ejercitar DOC's
 - Por ejemplo: *Coordinate Test* no porque incorpora el enumerado *Direction*; *BoardTest* no porque incorpora objetos de la clase *Coordinate*; ... sin dobles, se ejercita más de una clase!!!
 - Entonces deberían ser consideradas como **Pruebas de Componentes**...casi todas! O se ponen dobles!
 - Si! Relajadamente porque **no salen del ámbito del componente**: no acceden a clases de otros paquetes, no son de componentes
 - Entonces deberían ser consideradas como **Pruebas de Unidad**! No existen **Pruebas de Componentes**!
 - Si! Relajadamente porque **no salen del código de producción**: no acceden a recursos externos (por ejemplo, ficheros, base de datos, servicios,...) a través de otros componentes software (por ejemplo, librerías, protocolos,...), no son de integración
 - Entonces no existen **Pruebas de Componentes**! Se consideran todas unitarias!
 - Solución "practica": muchos desarrolladores no contemplan la existencia de **Pruebas de Componentes** específicamente y las consideran junto con las **Pruebas Unitarias** sin distinción

Elemplos:

[ticTacToe.models.GameTest](https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/components/GameTest.java)

(<https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/components/GameTest.java>)

[ticTacToe.models.ControllerModelIntegrationTest](https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/components/ControllerModelIntegrationTest.java)

(<https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/components/ControllerModelIntegrationTest.java>)

Pruebas de Integración

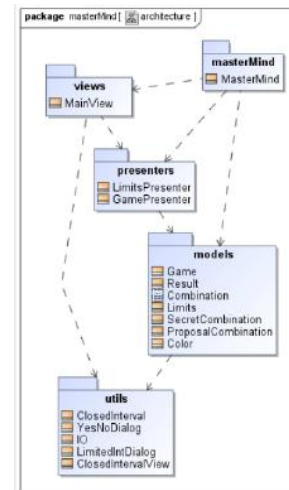
- Serán aquellas que en el ejercicio del SUT se colabora con un DOC desarrollado en otro componente (por ejemplo: *jar* de una librería, un servicio de bases de datos, ...)

Ejemplos:	<u>integration.DecisionTreeBroadTest</u> (https://github.com/MasterCloudApps/1.ADCS.pruebas/blob/master/src/test/java/integration/DecisionTreeBoardTest.java)
-----------	--

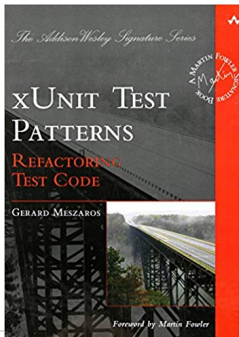

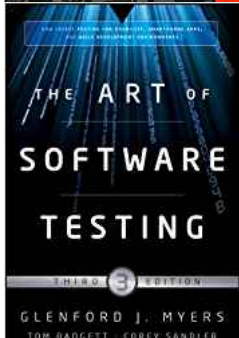
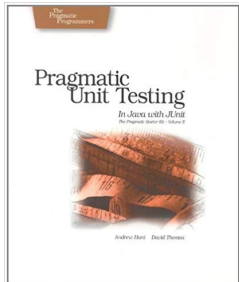
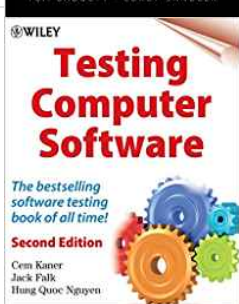

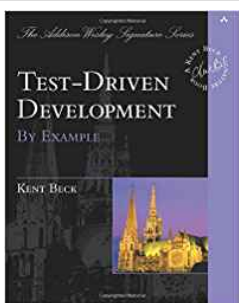
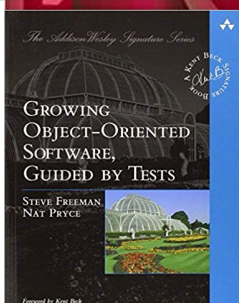
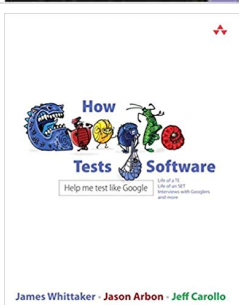
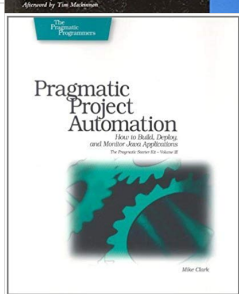
- Si no interesa que sea una **Prueba de Integración**(p.e. para no decelerar las **Pruebas de Regresión**), habría que incorporar un **doble para el DOC *BufferedReader*, *FileReader* y *IOException*** y, así convertirla en **Prueba Unitaria/de Componente**

Pruebas de Sistema

- **Arquitectura Modelo/VistaPresentador con Presentador de Modelo:**
 - **Modelo (Model)** con la responsabilidad de manejar los datos y la funcionalidad del negocio
 - **Presentador (Presenter)** con la responsabilidad total del estado y lógica de la presentación y solicitar funcionalidades de los modelos (*Plain Old Java Object*, POJO)
 - **Pruebas de los Presentadores ejercitan el sistema un ~9X%**
 - **Vista (View)** con la responsabilidad de manejar los controles de interfaz, la sincronización de la presentación pero eliminando el estado y la lógica de la presentación *Humple View pattern*

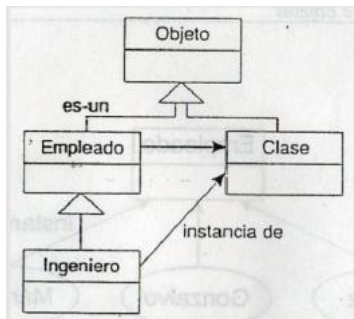


Bibliografía

Obra, Autor y Edición	Portada	Obra, Autor y Edición	Portada
<ul style="list-style-type: none"> • <i>xUnit Test Patterns: Refactoring Test Code</i> <ul style="list-style-type: none"> ◦ Gerard Meszaros ◦ Addison-Wesley Educational Publishers Inc; Edición: 01 (21 de mayo de 2007) 		<ul style="list-style-type: none"> • <i>Effective Unit Testing</i> <ul style="list-style-type: none"> ◦ Lasse Koskela ◦ Manning Publications Company; Edición: 1 (16 de febrero de 2013) 	
<ul style="list-style-type: none"> • <i>The Art of Software Testing</i> <ul style="list-style-type: none"> ◦ Glenford J. Myers ◦ John Wiley & Sons Inc; Edición: 3. Auflage (16 de diciembre de 2011) 		<ul style="list-style-type: none"> • <i>Pragmatic Unit Testing in Java with JUnit</i> <ul style="list-style-type: none"> ◦ Andy Hunt, Dave Thomas ◦ The Pragmatic Programmers (1674) 	
<ul style="list-style-type: none"> • <i>Testing Computer Software</i> <ul style="list-style-type: none"> ◦ Cem Kaner, Jack Falk, Hung Q. Nguyen ◦ Wiley John + Sons; Edición: 2nd (12 de abril de 1999) 		<ul style="list-style-type: none"> • <i>Diseño Ágil con TDD</i> <ul style="list-style-type: none"> ◦ Ble Jurado, Carlos ◦ Lulu.com (18 de enero de 2010) 	
<ul style="list-style-type: none"> • <i>Test Driven Development</i> <ul style="list-style-type: none"> ◦ Kent Beck ◦ Addison Wesley; Edición: 01 (8 de noviembre de 2002) 		<ul style="list-style-type: none"> • <i>Growing Object-Oriented Software, Guided by Tests</i> <ul style="list-style-type: none"> ◦ Steve Freeman ◦ Addison-Wesley Professional (12 de octubre de 2009) 	
<ul style="list-style-type: none"> • <i>How Google Tests Software</i> <ul style="list-style-type: none"> ◦ James A. Whittaker, Jason Arbon, Jeff Carollo ◦ Addison-Wesley Educational Publishers Inc; Edición: 01 (2 de abril de 2012) 		<ul style="list-style-type: none"> • <i>Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Apps</i> <ul style="list-style-type: none"> ◦ Mike Clark ◦ The Pragmatic Programmers (1900) 	

Metaclases

- Es una **clase** cuyas instancias son **clases** que permiten la **reflexión** del código: un código que manipula como información paquetes, clases, método, atributos, objetos y mensajes
 - Ejemplo: [auxiliar.withoutJUnit.MetaClazzeExample](#)
 - Ejemplo: [auxiliar.withoutJUnit.MetaClazzeInspectorExample](#)



Anotaciones

- Anotación es un mecanismo para añadir metadatos al código fuente Java que están disponibles en tiempo de compilación/ejecución.
 - Ejemplos:
 - `[@Override, @SuppressWarnings#...]` para el IDE
 - `[@Entity, @Key#...]` para JPA de Swing
 - Ejemplos: [auxiliar.withoutJUnit.AnnotationExample](#)
 - Ejemplo: [auxiliar.withoutJUnit.ClassExample](#)
 - Ejemplo: [auxiliar.withoutJUnit.MetaClassesWithAnnotationsExample](#)

Aserciones

- Todas las funciones tienen una semántica similar al **assert**:
 - se comprueba una condición, **si es cierta continúa la ejecución**.
 - En caso contrario, **la prueba no pasa y se continúa la ejecución por otro método de prueba de la clase**

org.junit.Assert

```

class org.junit.Assert {
    public static void assertTrue([String msg,] boolean expected); // 1)
    public static void assertFalse([String msg,] boolean expected);
    public static void assertEquals([String msg,] Type1 expected, Type2 result,
        [double tolerance]); // 2)
    public static void assertEquals([String msg,] Object expected, Object result);
    public static void assertEquals([String msg,] Object expected, Object result);
    public static void assertEquals([String msg,] Object expected);
    public static void assertEquals([String msg,] Object expected);
    public static void assertEquals([String msg,] Type1[] expected, Type2
        result);
    public static void fail([String msg,] )
}
  
```

1) msg opcional para expresar el motivo de fallo que se concatena a "expected <expected> but was <result>".
 2) Type1 y Type2 son ambos del mismo cualquier tipo primitivo o clase de objeto

• Ejemplo: [auxiliar.JUnit.AssertExample](#)