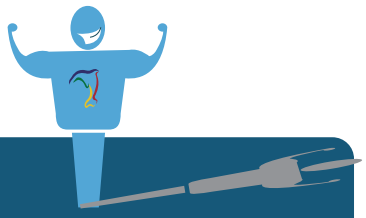
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

# ¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

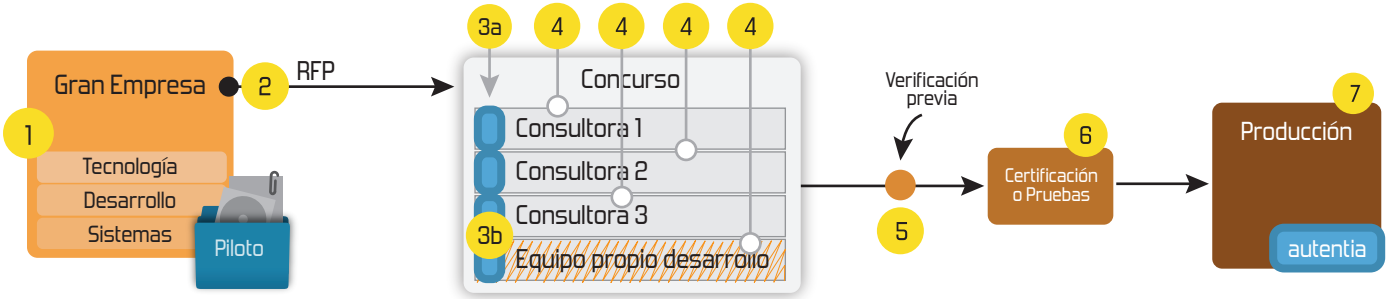
## 1. Desarrollo de componentes y proyectos a medida



## 2. Auditoría de código y recomendaciones de mejora

## 3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.




## 4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
**www.adictosaltrabajo.com**

Para más información visítenos en:
**www.autentia.com**

**Tutorial desarrollado por: Carlos García Pérez**

**Puedes encontrarme en Autentia**
**Somos expertos en Java/J2EE**
**Contacta en info@autentia.com**



# JUnit 4. Pruebas de Software Java

## Introducción

Yo trabajo en Autentia, una empresa en donde velamos por la calidad de nuestros desarrollos en todas las fases del ciclo de vida del software. Una fase importantísima es la **Fase de Pruebas.**

JUnit es un framework para realizar y automatizar pruebas de aplicaciones Java. Es decir, JUnit se sitúa en la fase de pruebas dentro del ciclo de Ingeniería del Software.

Este documento no trata de ser una referencia teórica sobre pruebas. Se centra en introducir al lector en el uso y las nuevas características de la versión JUnit 4 respecto a las anteriores.

Si quería remarcar dos conceptos teóricos:

a. Nunca se puede asegurar que una aplicación funcionará correctamente siempre, pues es **inviable económicamente** realizar pruebas de todos los componentes individuales y de todos los componentes como conjunto.
b. **Las pruebas deben ser diseñadas para descubrir fallos** y no para demostrar que el software funciona. Siendo más razonable diseñar pruebas de aquellas partes en donde la probabilidad de fallo es mayor.
c. **Las pruebas deben ser diseñadas por personas distintas a las personas que desarrollan** el componente que se desea probar. De todos es bien sabido que conocemos los fallos de nuestros componentes y si el componente es ejecutado por las personas que lo desarrollan cuando lo enseñan, no van a ser tan ingenuos de hacer que falle.

## Instalación de JUnit 4

JUnit 4 puede descargarse desde la siguiente dirección http://www.junit.org.

JUnit se distribuye como un fichero comprimido, lo descomprimimos y añadimos el fichero **junit.jar** al CLASSPATH de nuestro sistema.

Para realizar pruebas con JUnit 4.0, se **requiere una versión de Java 5.0 o superior.**

## Ejemplo

A continuación presentamos una clase sobre la que vamos a realizar pruebas utilizando JUnit 4. La clase aporta una seria de funcionalidad sobre un array de cadenas de caracteres:

```
String getMaxLength()
```

Devuelve la cadena que más caracteres tenga.

`int getTotalLength()`

Devuelve la suma de la longitud de todas las cadenas de caracteres.

`int getIndexOf(String) throws java.util.NoSuchElementException`

Devuelve la posición que ocupa una determinada cadena o lanza una java.util.NoSuchElementException si la cadena no existe.

```
package com.autentia.examples.junit4.example1;

import java.util.*;

/**
 * Clase con métodos de utilidad sobre Arrays de cadenas de caracteres
 * @author Carlos García. Autentia Real Business Solutions.
 * @see http://www.autentia.com
 */
public class StringArrayUtils {

	// Atributos de la clase
	private java.util.Vector elements;

	/**
	 * Constructor
	 */
	public StringArrayUtils(String[] data){
		// Verificamos que la lista tenga valores
		if ((data == null) || (data.length == 0))
			throw new IllegalArgumentException();

		this.elements = new Vector();
		for (String element : data){
			elements.addElement(element);
		}
	}

	/**
	 * @return Devuelve la cadena que tiene más caracteres
	 */
	public String getMaxLength(){
		String max = "";

		for (String element : elements){
			if (element.length() > max.length()){
				max = element;
			}
		}

		return max;
	}

	/**
	 * @return Devuelve la suma de la longitud de todas las cadenas
	 */
	public int getTotalLength(){
		int total = 0;

		for (String d : elements){
			total += d.length();
		}

		return total;
	}

	/**
	 * @param searched Cadena buscada
	 * @return Devuelve la posición de un elemento dentro del array
	 * @throws java.util.NoSuchElementException Si el elemento no existe en la lista
	 */
	public int getIndexOf(String searched) throws java.util.NoSuchElementException {
		int pos = 0;

		// Comprobamos que el argumento sea válido
		if (searched == null){
			throw new IllegalArgumentException();
		}

		// Recorremos la información hasta encontrar el elemento
		for (String d : elements){
			if (d.equals(searched)){
				return pos;
			}
			pos++;
		}

		// El elemento no existia, lanzamos la excepción
		throw new java.util.NoSuchElementException(searched);
	}
}
```

Bueno ya tenemos la clase sobre la que vamos a realizar las pruebas de unidad, ahora vamos a diseñar otra clase con las pruebas a realizar.

```
package com.autentia.examples.junit4.example1;

import java.io.*;
import java.util.Vector;

import junit.framework.Assert;
import junit.framework.JUnit4TestAdapter;
```

```
import org.junit.runner.JUnitCore;
import org.junit.*;

/**
 * Realiza pruebas funcionales sobre la clase calculadora
 * @author Carlos García. Autentia Real Business Solutions
 */
public class JUnitTestStringArrayUtils {

          private static StringArrayUtils stringUtils;

          /**
           * Leemos desde un fichero, los datos de prueba con los que vamos a realizar las pruebas
           */
          @BeforeClass
          public static void inicioClase() {
                    // Leemos las cadenas con las que vamos a realizar las pruebas desde un fichero de texto
                    try {
                              FileReader reader = new FileReader("string.txt");
                              BufferedReader buffer = new BufferedReader(reader);
                              String line = buffer.readLine();
                              Vector velements = new Vector();
                              while (line != null) {
                                        velements.addElement(line);
                                        line = buffer.readLine();
                              }

                              // Creamos el String[] a partir de los datos leidos del fichero
                              String[] elements = new String[velements.size()];
                              velements.copyInto(elements);

                              JUnitTestStringArrayUtils.stringUtils = new StringArrayUtils(
                                                  elements);

                              // Liberamos recursos
                              velements.removeAllElements();
                              reader.close();
                    } catch (IOException ex) {
                              // No se dará
                    }

          }

          /**
           * Este método liberaría los recursos reservados en BeforeClass
           */
          @AfterClass
          public static void finClase() {
                    // Para este ejemplo no hacemos nada, pero exponemos el método por
                    // motivos didácticos exclusivamente
          }

          /**
           * Este método se ejecuta para cada prueba ANTES de invocar el código de cada prueba
           */
          @Before
          public void testStart() {
                    // Para este ejemplo no hacemos nada, pero exponemos el método por
                    // motivos didácticos exclusivamente
          }

          /**
           * Este método se ejecuta para cada prueba DESPUÉS de invocar el código de cada prueba.
           */
          @After
          public void testEnd() {
                    // Para este ejemplo no hacemos nada, pero exponemos el método por
                    // motivos didacticos exclusivamente
          }

          /**
           * Verificamos que en caso de recibir un null como argumento en el
           * constructor la clase lanza una IllegalArgumentException
           */
          @Test(expected = java.lang.IllegalArgumentException.class)
          public void initTest() {
                    new StringArrayUtils(null);
          }

          /**
           * Verificamos que la cadena más larga sea la cadena "Tres"
           */
          @Test
          public void getLengthTest() {
                    Assert.assertEquals("Tres", JUnitTestStringArrayUtils.stringUtils.getMaxLength());
          }

          /**
           * Prueba sobre el método que devuelve la suma total de todas las cadenas
           * almacenadas Suponemos que el calculo del tamaño total es un método
           * crítico que debe realizarse antes de 25 milisegundos
           */
          @Test(timeout = 25)
          public void getTotalLengthTest() {
                    Assert.assertEquals(10, JUnitTestStringArrayUtils.stringUtils.getTotalLength());
          }

          /**
           * Prueba sobre el método que devuelve la posición de una cadena.
```

```
	 * Verificamos que si le pasamos null como argumento lanza la excepción
	 * correcta
	 */
	@Test(expected = java.lang.IllegalArgumentException.class)
	public void getIndexOfTest() {
		JUnitTestStringArrayUtils.stringUtils.getIndexOf(null);
	}

	/**
	 * Prueba sobre el método que devuelve la posición de una cadena Verificamos
	 * que si le pasamos una cadena que no existe como argumento lanza la
	 * excepción correcta
	 */
	@Test(expected = java.util.NoSuchElementException.class)
	public void getIndexOfTest2() {
		Assert.assertEquals(0, JUnitTestStringArrayUtils.stringUtils.getIndexOf("EsteElementoNoExiste"));
	}

	/**
	 * Prueba sobre el método que devuelve la posición de una cadena.
	 * Verificamos que si le pasamos una cadena que existe devuelve la posición correcta
	 */
	@Test
	public void getIndexOfTest3() {
		Assert.assertEquals(1, JUnitTestStringArrayUtils.stringUtils.getIndexOf("Dos"));
	}

	@Ignore("Este test no se hace, se expone como ejemplo")
	@Test
	public void ignore() {
		// Código que compone la prueba
		// ...
		// ...
	}

	/**
	 * @return Para mantener compatibilidad con las herramientas que tratan con versiones anteriores de JUnit
	 */
	public static junit.framework.Test suite() {
		return new JUnit4TestAdapter(JUnitTestStringArrayUtils.class);
	}

	/**
	 * Lanza las pruebas sin compatibilidad hacia atrás, es decir se requiere Java 5 y JUnit4 instalado
	 */
	public static void main(String[] args) {
		JUnitCore.runClasses(JUnitTestStringArrayUtils.class);
	}
}
```

`BeforeClass:`

Sólo puede haber un método con este marcador.
Este método será invocado una vez al principio del lanzamiento de todas las pruebas. Se suele utilizar para inicializar los atributos comunes a todas las pruebas o para realizar acciones que tardan un tiempo considerarse en ejecutarse.

`AfterClass:`

Sólo puede haber un método con este marcador. Este método será invocado un sóla vez cuando finalizen todas las pruebas.

`Before:`

Los métodos marcados con está anotación, serán invocados antes de iniciarse cada prueba.

`After:`

Los métodos marcados con está anotación, serán invocados después de ejecutarse cada prueba.

`Test:`

Representa un test que se debe ejecutar.
Puede tener dos tipos de modificadores:

- Test(timeout=X) que significa que el test será válido si se ejecuta en un tiempo máximo de X milisegundos.
- Test(expected = java.util.NoSuchElementException.class). Que significa que el test será válido si se lanza una determinada excepción.

`Ignore:`

Los métodos marcados con esta anotación no serán ejecutados. Se suelen poner para desactivar las pruebas que no pueden ser realizadas por algún motivo.
Puede tener un parámetro que indica un texto que se visualizará en la la plataforma donde se ejecuten los test.
Ejemplo: Ignore("Se desactiva esta prueba hasta que tengamos privilegios de conexión con la base de datos.");

# Conclusiones

Al igual que la gran mayoría de las tecnologías del mundo Java, JUnit evoluciona para aportar nuevas mejoras y hacernos la vida más fácil a los programadores, diseñadores de pruebas, etc.

Ahora JUnit incorpora las siguientes ventajas y modificaciones respecto a versiones anteriores

- Las clases que contienen los métodos que representan las pruebas ya no tienen que ser subclases de junit.framework.TestCase
- Los métodos que representan las pruebas a realizar ya no tienen que tener el prefijo "test" en su nombre sino que se indican con la anotación @Test
- Sustitución del método 'setUp' por la anotación o anotaciones @Before.
- Sustitución del método 'tearDown' por la anotación o anotaciones @After.
- Se pueden realizar pruebas en donde el tiempo de ejecución es crítico. De manera que una prueba falle si tarda más de X milisegundos en ser ejecutada.
- Se pueden realizar pruebas en donde se debe controlar que una excepción es lanzada.

- Se pueden desactivar pruebas, a través de la anotación @Ignore.

Puedes opinar sobre este tutorial aquí

# Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

**¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?**

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
**Autentia = Soporte a Desarrollo & Formación**



# Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

| Subscribirse a Novedades | |
|---|---|
| ***e-mail*** | |
| | Enviar |

# Otros Tutoriales Recomendados (También ver todos)

| Nombre Corto | Descripción |
|---|---|
| Pruebas Web con JWebUnit | Os mostramos como automatizar las pruebas de caja negra (desde el punto de vista de usuario final) de vuestro Web con el Framework gratuito JWebUnit. Esta técnica es perfecta para crear test de regresión de aplicaciones Web complejas. |
| Soporte de Asserts en Java 1.4.x | Os mostramos como utilizar los asserts en Java (disponibles a partir de la versión 1.4) |
| Test con JUnit | Cuando se hacen desarrollo profesionales, no basta con hacer los programas, hay que asegurarse de que van a funcionar. Una de las técnicas más seguras es crear aplicaciones que incluyan el código para autoprobarse. Os mostramos como usar JUnit |
| Decompilar Java | Os mostramos como recuperar el fuente de vuestro código a partir de los ficheros compilados .class |
| Java en tu movil con J2ME | Os enseñamos como construir una aplicación Java capaz de correr en tu Movil gracias a J2ME |
| Novedades en Java 1.5 | Ya está disponible la versión Beta del J2SDK 1.5. Os mostramos algunas de las nuevas características introducidas en el lenguaje Java: Clases genéricas, enumeraciones, bucles simplificados, etc. |
| Construir un Servidor Web en Java | En este tutorial os enseñamos los principios de las aplicaciones multi-hilo a través de la creación de un servidor web básico en Java. Podremos ver en un ejemplo real el uso de sockets, threads, excepciones, etc. |
| Técnicas básicas y poco comentadas en Java | Os mostramos como realizar algunas cosas simples en Java: Formateo de decimales y enteros, gestión de preferencias y comparación entre objetos de nuevas clases |
| Pruebas de Rendimiento y Funcionales Web | Jose María Toribio, nos enseña en este tutorial como podemos utilizar la aplicación gratuita JMeter para realizar pruebas de rendimiento y funcionales (vitales para la regresión y reingeniería) sobre nuestras aplicaciones Web |
| | Os mostramos como enviar ficheros a un servidor Web y manipularlos en un servlet en el |

Upload de ficheros en Java servidor, gracias a APIs de apache