

Guía de pruebas unitarias con JUnit

[Java: JUnit](#)

[Creación de pruebas unitarias.](#)

[Anotaciones base de JUnit](#)

[Aserciones base de JUnit](#)

[Pruebas Unitarias Parametrizadas](#)

[Cómo Escribir una Prueba Unitaria para Java Usando Eclipse y JUnit.](#)

[2.1. Primer Paso](#)

[2.2. Segundo Paso](#)

[2.3. Tercer Paso](#)

[2.4. Cuarto Paso](#)

[2.5. Quinto Paso](#)

1. Java: Junit

JUnit es el framework de referencia para la implementación de pruebas unitarias en Java, y alrededor de éste se han generado un amplio conjunto de librerías, en general para ampliar las funcionalidades estándar ofrecidas por JUnit. Aquí vamos a cubrir los fundamentos de pruebas unitarias con JUnit version 4.X.

Creación de pruebas unitarias.

Para crear una prueba unitaria, solo hay que crear una nueva clase Java y el método en el que se programará la prueba debe lucir de la siguiente forma:

```
public class ClasePrueba {  
    @Test  
    public void test() {  
        fail("Not yet implemented");  
    }  
}
```

Para mayor detalle sobre el proceso de creación de pruebas unitarias con JUnit y Eclipse, vease [Cómo Escribir una Prueba Unitaria para Java Usando Eclipse y JUnit.](#)

Anotaciones base de Junit

A continuación se presenta un resumen de las anotaciones base de Junit que permiten la definición y configuración de las pruebas.

Annotation	Description
@Test public void method()	@Test es la anotación base que identifica que un método dentro de la clase es un método de pruebas.
@Before public void method()	Este método se ejecuta antes de cada test. Puede ser utilizado para realizar el proceso de preparación del entorno de pruebas, por ejemplo, inicializando la instancia de la clase bajo pruebas o clases requeridas.
@After public void method()	Este método se ejecuta después de cada test. Usualmente es utilizado para limpiar el entorno; por ejemplo, liberando recursos como conexiones, borrando datos temporales o eliminando estructuras costosas en memoria.
@BeforeClass public static void method()	<p>Los métodos marcados con esta anotación se ejecutan una única vez, antes del inicio de la ejecución de las pruebas, y deben ser definidos como estáticos. Aunque no es usual, es posible tener más de un método BeforeClass en una clase de pruebas, aunque estos deberán ser independientes entre si ya que no hay una definición de orden de ejecución de los mismos.</p> <p>Estos métodos son utilizados para realizar actividades costosas en tiempo/recursos, comunmente obtener una conexión a base de datos o preparar entornos de datos complejos requeridos por todas las pruebas.</p>
@AfterClass public static void method()	<p>Estos métodos también son ejecutados una única vez, al final de todas las pruebas. Las consideraciones a tener en cuenta son las mismas que con los métodos BeforeClass.</p> <p>Estos métodos son utilizados para realizar operaciones de limpieza o restauración de entorno.</p>
@Ignore	Como su nombre sugiere, esta anotación indica que el método de prueba debe ser ignorado. Esta definición es útil cuando, por ejemplo, el código base ha cambiado y las pruebas aún no ha sido ajustadas o si hay alguna dependencia a un sistema externo que se encuentra indisponible.
@Test (expected = Exception.class)	Esta anotación es útil para aquellos casos en que queremos probar que nuestro código lance una excepción bajo ciertas condiciones. Por tanto, si el método de prueba está marcado con esta anotación fallará si no se recibe la excepción esperada.
@Test(timeout=100)	Esta anotación permite realizar o definir pruebas con requerimientos de tiempo específicos. Por ejemplo en una prueba de integración, tenemos la limitación de que una consulta se realice en menos de X tiempo. La prueba fallará si se supera el timeout (en milisegundos) definido.

Aserciones base de Junit

La validación de los resultados de una prueba se hacen mediante un conjunto de métodos disponibles en la clase Assert de Junit. A continuación se listan las aserciones más comúnmente utilizadas, teniendo presente que los métodos con parámetros entre corchetes '[']' significa que el método tiene versión con y sin dicho parámetro.

Statement	Description
<code>assertTrue([message], boolean condition)</code>	Valida que la condición o variable boolean sea verdadera.
<code>assertEquals([String message], expected, actual)</code>	Valida que dos valores son iguales. Tenga presente el orden de los parámetros: primero el valor esperado, luego el calculado.
<code>assertEquals([String message], expected, actual, tolerance)</code>	Valida que dos valores de punto flotante (double o float) son iguales a un nivel de tolerancia dado en términos de número de posiciones decimales.
<code>assertNull([message], object)</code>	Valida que el objeto es nulo.
<code>assertNotNull([message], object)</code>	Valida que el objeto no es nulo.
<code>assertSame([String], expected, actual)</code>	Valida que las dos variables referencian al mismo objeto o instancia. En otras palabras, valida que <code>obj1==obj2</code> .
<code>assertNotSame([String], expected, actual)</code>	Valida que los dos objetos referencian diferentes instancias. Podría ser útil para validar que dos objetos son "lógicamente" iguales (equals) pero diferentes en instancia (<code>==</code>).

Pruebas Unitarias Parametrizadas

Uno de los escenarios más comunes a los que nos enfrentamos a la hora de diseñar e implementar pruebas unitarias es la necesidad de probar una funcionalidad con diferentes parámetros de entrada para verificar diferentes respuestas o salidas. Una aproximación inicial a este problema usualmente tiene las siguientes características:

- Múltiples métodos de prueba con duplicación de código para la preparación de los parámetros y la ejecución de la prueba. O bien...
- Un método que realiza el procesamiento de los parámetros y la ejecución de la funcionalidad y múltiples métodos de prueba que invocan el método anterior y realizan el assert del resultado obtenido.

En cualquiera de los escenarios mencionados hay en mayor o menor medida, duplicación de código. Además, la mantenibilidad de las pruebas se hará cada vez más compleja en la medida que vayamos añadiendo más y más escenarios de prueba.

Por lo anterior, para estos escenarios consideramos recomendable usar una librería que extiende la funcionalidad básica de Junit y permite tener métodos de prueba parametrizados.

Para usar esta librería se debe:

- Vincular el jar,
 - Al script de gradle añadir la siguiente dependencia:

```
testCompile "pl.pragmatists:JUnitParams:1.0.2"
```

- Si el proyecto no usa Gradle, descargar el jar desde la dirección <https://code.google.com/p/junitparams/downloads/list> y añadirlo al classpath del proyecto en Eclipse.
- Para que la clase de pruebas pueda manejar pruebas parametrizadas, se debe ejecutar con un Runner de Junit provisto por la librería. Se configura añadiendo la siguiente anotación a la clase correspondiente:
`@RunWith(JUnitParamsRunner.class)`
- Para indicar que la prueba recibe parámetros, el método de prueba se debe anotar de la siguiente forma:
`@Parameters()`
- JUnitParams permite pasar los parámetros de varias formas:
 - En la misma anotación `@Parameters({"John",47,true})`
 - Mediante un método que retorna un array de Objetos
`@Parameters(method="fullNameData")`
 - Mediante el procesamiento de un archivo CSV,
`@FileParameters(value="classpath:sample.csv",
mapper=PersonMapper.class)`

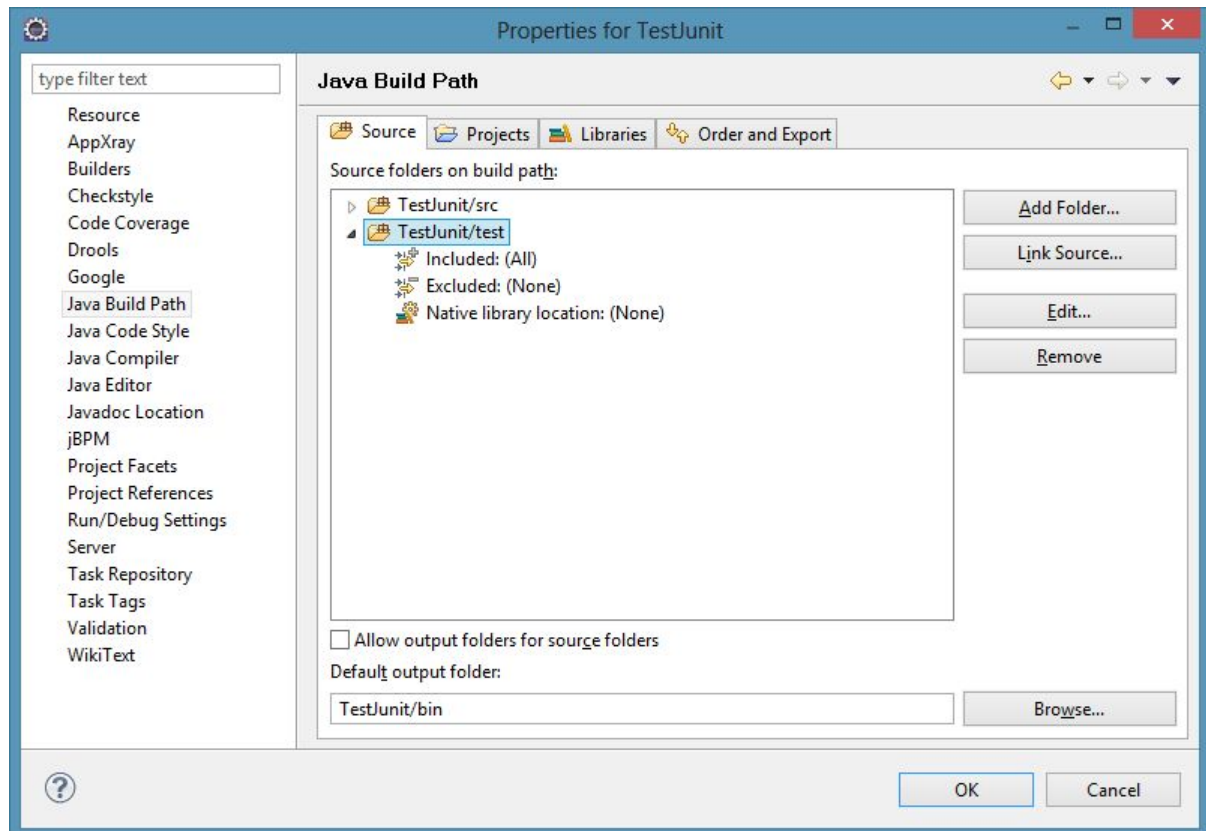
Hay algunas variaciones a estas formas, éstas están documentadas en el sitio de la librería <https://github.com/Pragmatists/junitparams> mediante un ejemplo práctico.

En el repositorio de la gerencia técnica se ha subido un ejemplo básico del uso de la librería. La URL es la siguiente:

http://svn.ceiba.com.co/repositorios/ceiba_ejemplos_pruebas/trunk/03.%20Construccion/test_with_parameters

2. Cómo Escribir una Prueba Unitaria para Java Usando Eclipse y Junit.

Recomendación: Tenga una carpeta de test exclusiva para las pruebas, tal como se ilustra en la siguiente imagen.



2.1. Primer Paso

El primer paso para crear la prueba unitaria es adicionar JUNIT como librería, para hacerlo realizamos lo siguiente.

En las propiedades de proyecto en la opción “Java Build Path” seleccionamos la pestaña “Libraries”, Elegimos la opción “Add Library”, (ver Imagen 1), en el Wizard emergente seleccionamos “JUnit”, luego “Next”,(ver Imagen 2), en la lista de “JUnit library versión” seleccionamos JUnit 4 y presionamos Finish,(ver Imagen 3), A continuación encuentra la secuencia de pantallas y el resultado esperado,(ver Imagen 4).

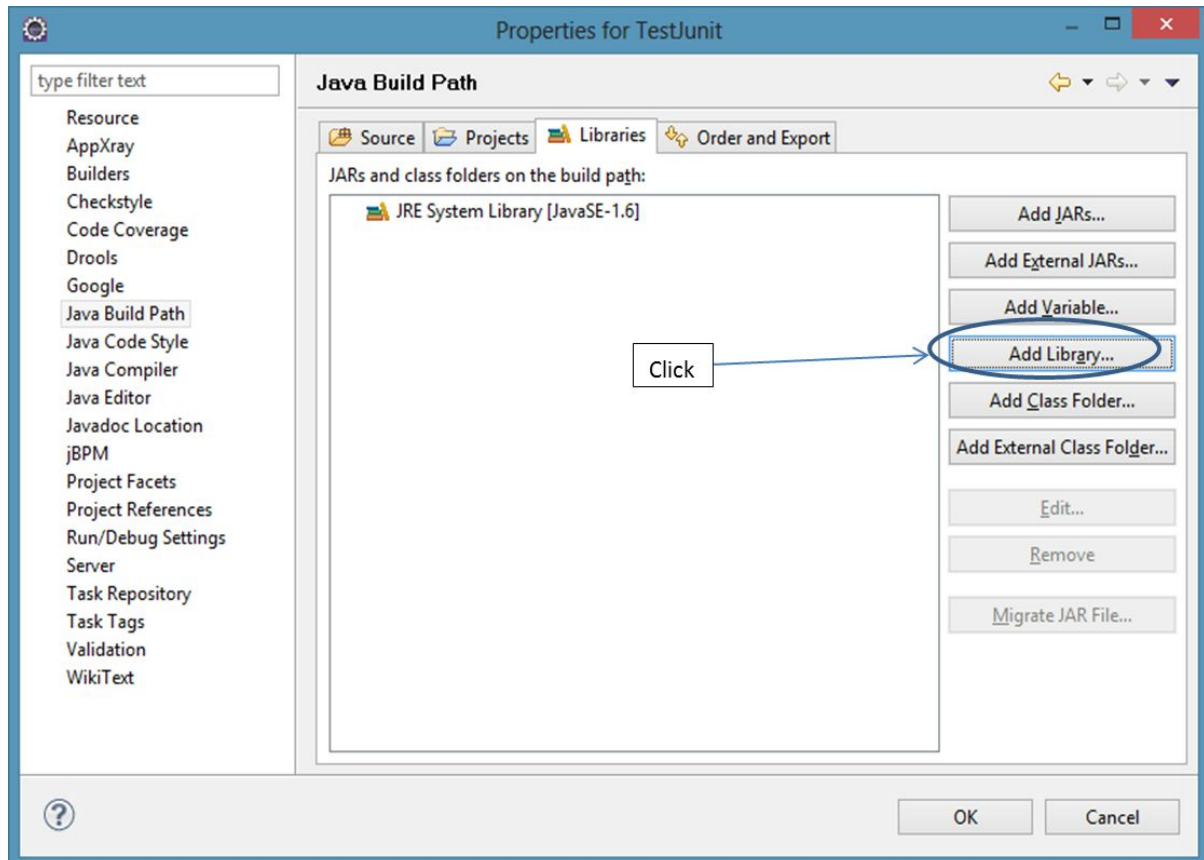


Imagen 1

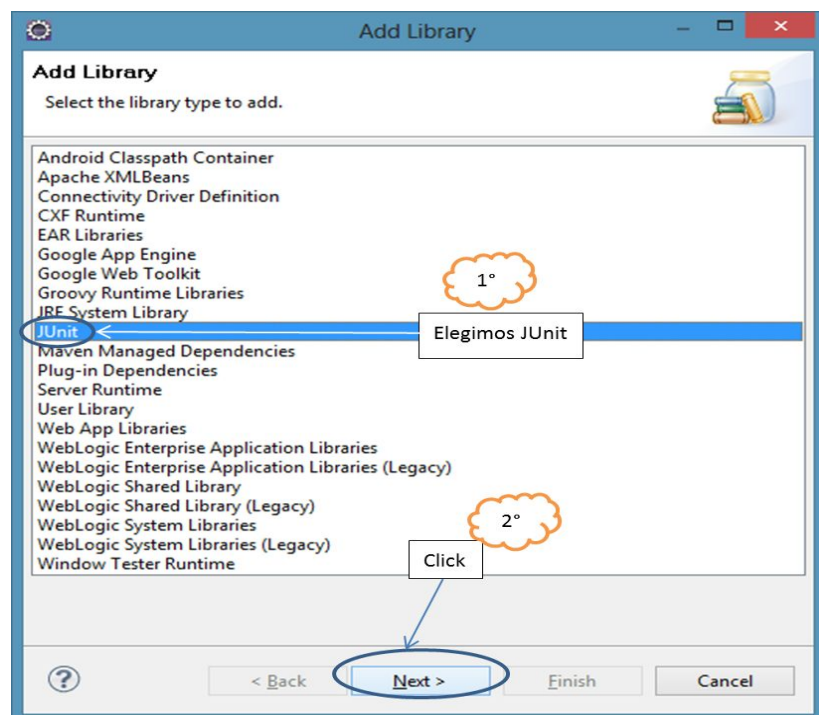


Imagen 2

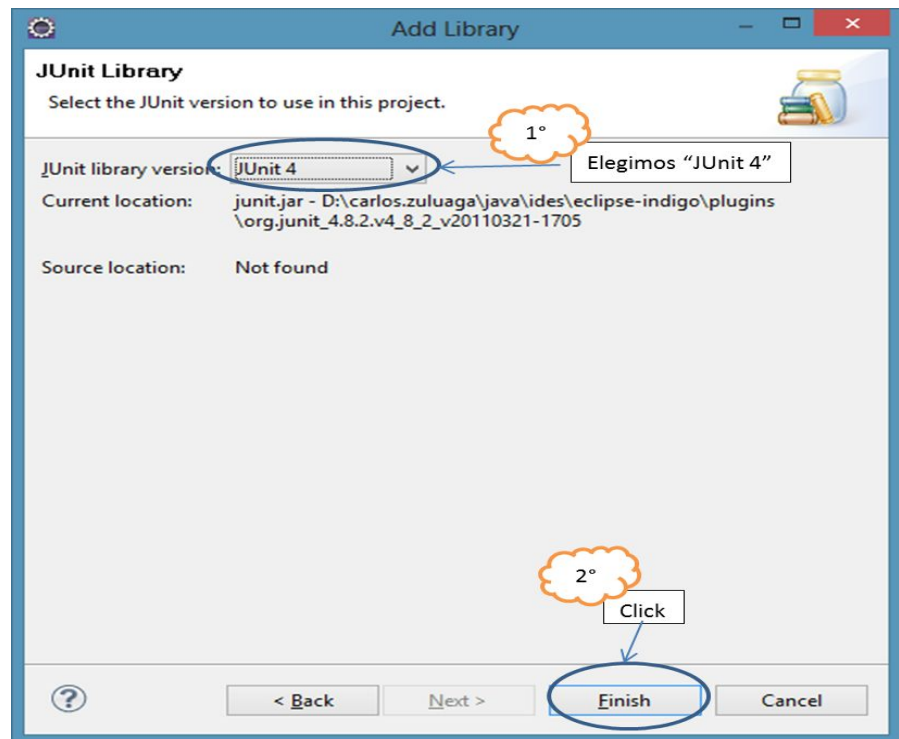
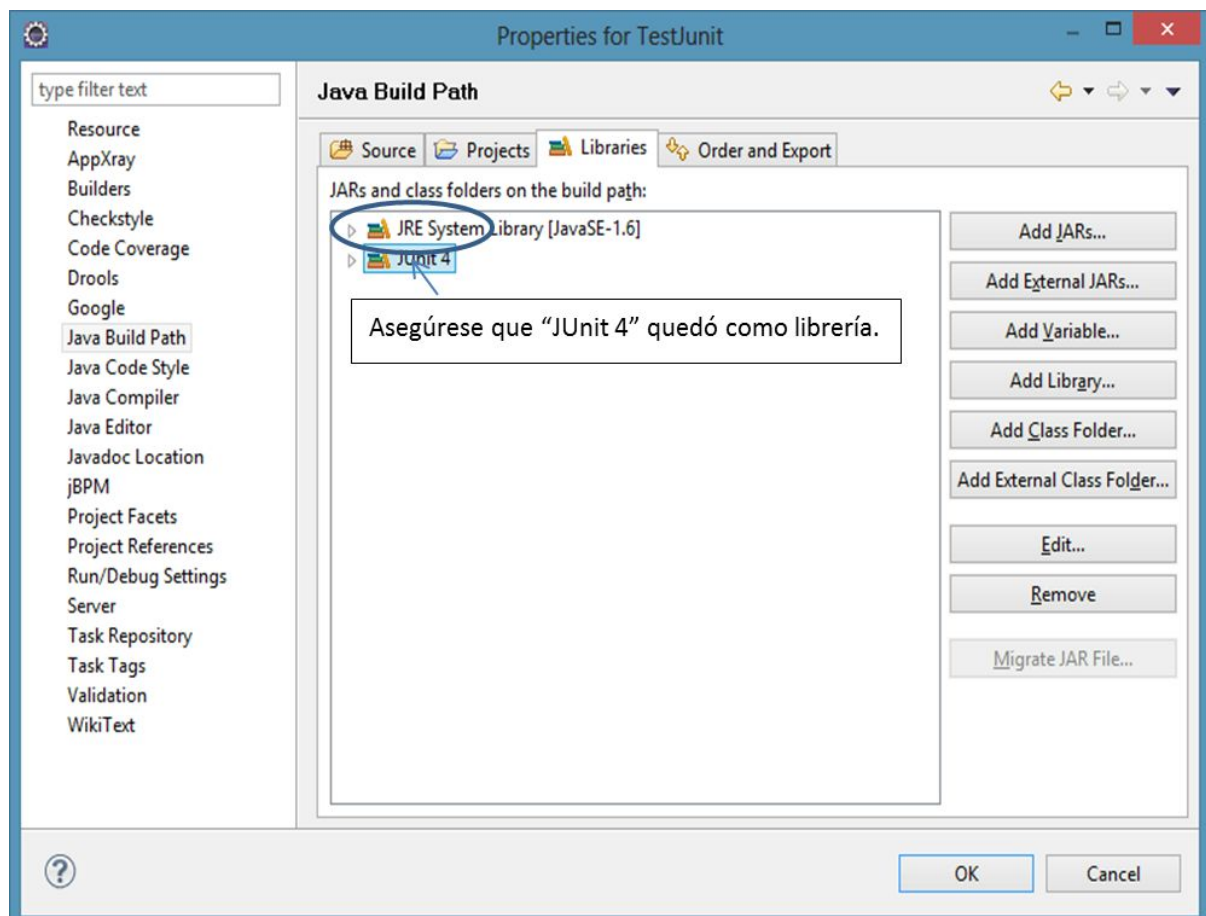


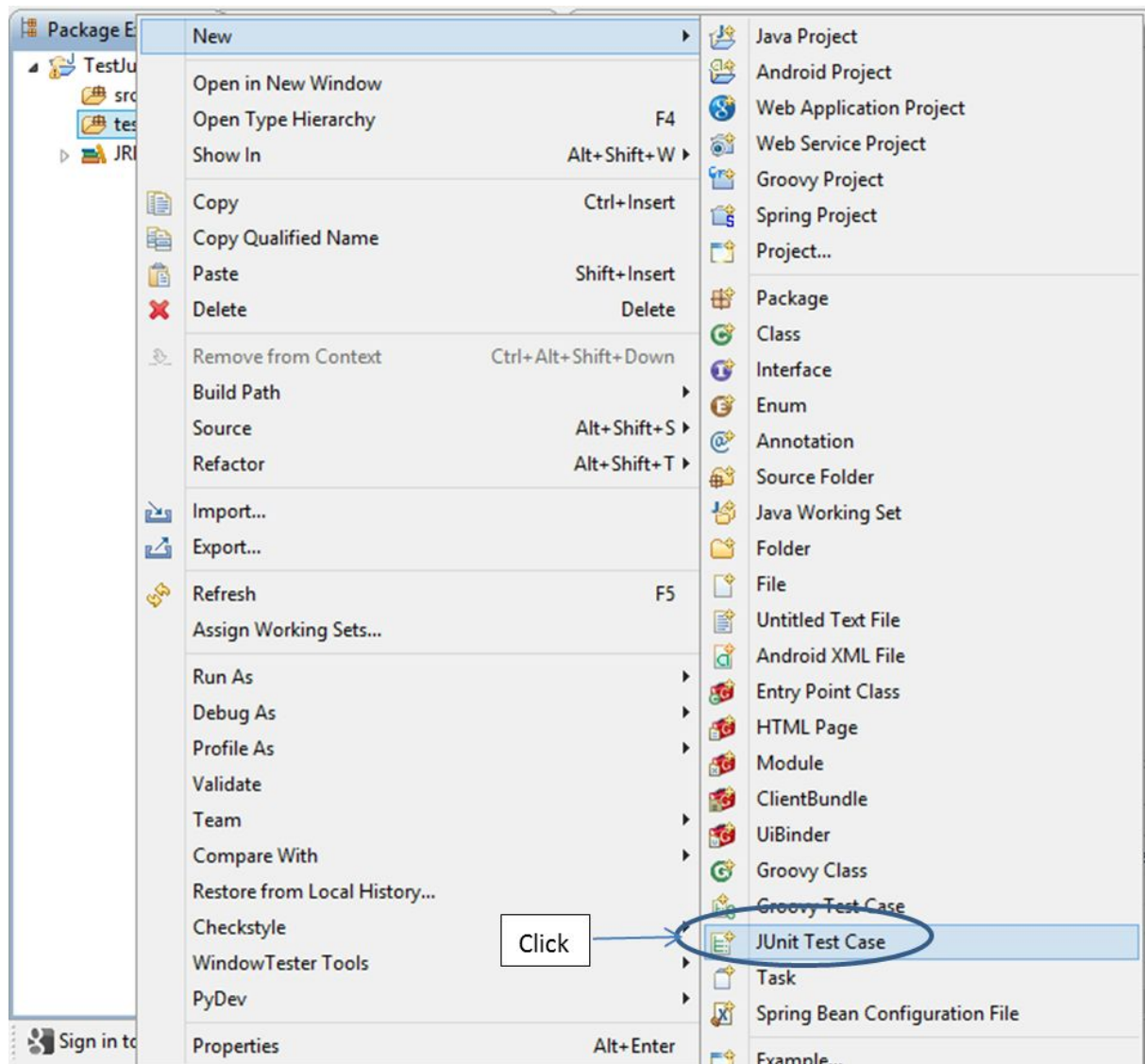
Imagen 3



2.2. Segundo Paso

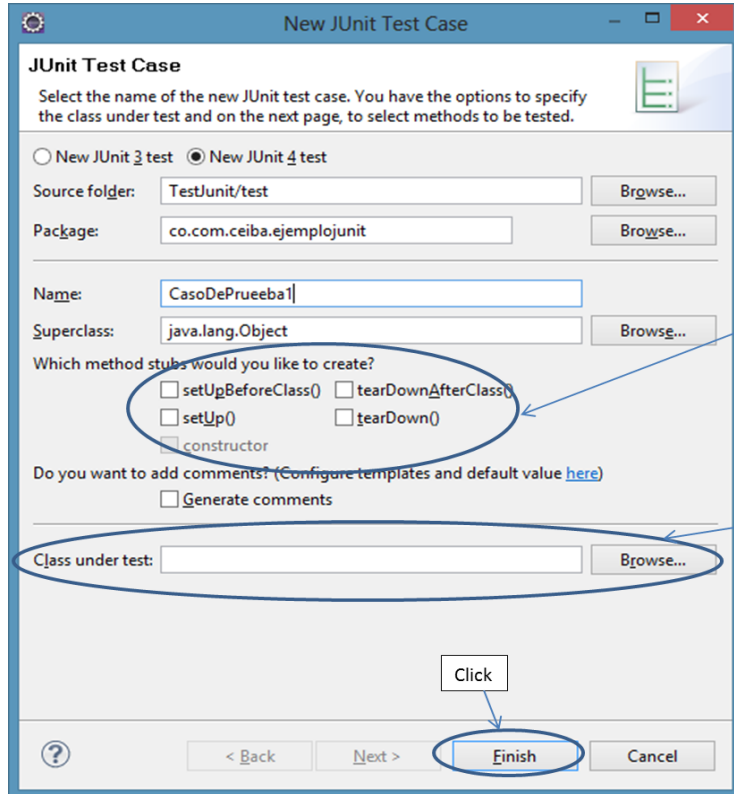
El Siguiente Paso que debemos realizar es la creación de una clase de pruebas para esto realizamos lo siguiente.

Sobre la carpeta de test damos click derecho allí seleccionamos la opción “New ->” y “JUnit Test Case”.



2.3. Tercer Paso

A Continuación llenamos el formulario de creación de clase de prueba como corresponde. Y seleccionamos Finish



JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments: (Configure templates and default value [here](#))
☐ Generate comments

Class under test:

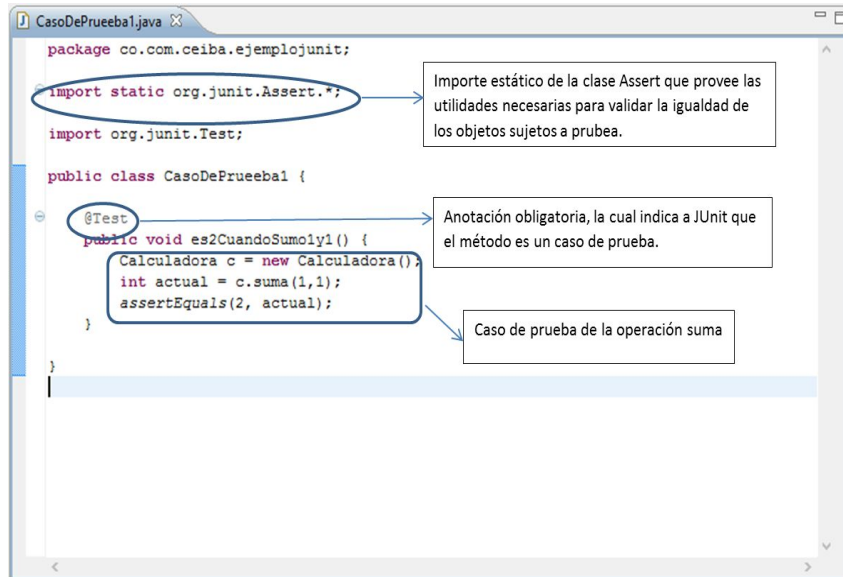
Métodos de inicialización y finalización de los casos de prueba; Son utilizados para preparar los Mock que se requieren en la prueba, por ejemplo simular un contexto ejb con EJB3Unit.

Los métodos setUp y tearDown se ejecutan antes y después de la ejecución de una prueba y se recomienda usarlos para destruir los objetos utilizados en las pruebas.

Class Under Test: Este es un campo opcional y ayuda a la creación de los casos de prueba, se basa en una clase existente y da como resultado una plantilla de caso de prueba por cada método en la clase seleccionada.

2.4. Cuarto Paso

Al finalizar ya tenemos la clase donde se van a escribir los casos de prueba. Para escribir un caso de prueba basta con utilizar la anotación Test sobre un método de la clase, en este método se escribe el caso de prueba y se realiza la verificación de la prueba; para la verificación de la prueba se utilizan los métodos assert importado estáticamente de la clase Assert de junit, a continuación se presenta un ejemplo de un caso de prueba que valida que la operación suma de la clase calculadora retorna 2 cuando se suma 1 y 1.



2.5. Quinto Paso

Finalmente para realizar la ejecución de las pruebas realice lo siguiente: Click derecho en la clase “Run ->” y “JUnit Test Case”.

