



CICLO 2

[FORMACIÓN POR CICLOS]


Automatización de Pruebas Unitarias **USANDO JUNIT**



Ingeni@
Soluciones TIC



UNIVERSIDAD
DE ANTIOQUIA
Facultad de Ingeniería




Como sabemos, existen diferentes niveles de pruebas. Estos niveles se distinguen entre sí dependiendo de si buscan realizar verificación o validación, así como si buscan evaluar el sistema completo o solo alguno de sus componentes. Entre dichos niveles encontramos las pruebas unitarias, con las cuales se verifica el funcionamiento de un componente, clase o función de manera aislada respecto al resto de la aplicación. La naturaleza de este tipo de pruebas ha llevado a que sean ampliamente automatizadas.

La **automatización de pruebas** consiste en el uso de herramientas de software para controlar la ejecución de pruebas, la comparación de resultados obtenidos vs. resultados esperados, la definición de precondiciones y otras funciones de control y de reporte de pruebas. Así pues, esta cuenta con algunos beneficios potenciales, tales como:

- Reducir el costo de realización de pruebas.
- Reducir los errores humanos al probar.
- Reducir las diferencias en la calidad de las pruebas que se puede presentar entre diferentes individuos.
- Reducir significativamente el costo de pruebas de regresión, es decir, de realizar la misma prueba de manera repetida a medida que el sistema de software evoluciona.

Para obtener estos beneficios, es necesario entender que al realizar pruebas no basta simplemente con ejecutar partes de una aplicación al azar ingresando determinados valores. Generalmente, el proceso de pruebas (o de *testing*) es un conjunto de actividades planificadas, donde se usan técnicas, herramientas y criterios que buscan que las pruebas, efectivamente, aseguren la calidad de la aplicación o sistema de software.

Es por esto por lo que se parte de un plan de pruebas. Un **plan de pruebas** comprende las pruebas de diferentes niveles que se realizarán en una aplicación o sistema de software. También, incluye diferentes criterios que se deben tener en cuenta a la hora de realizar dichas pruebas. El plan de pruebas suele estar compuesto por uno o varios conjuntos de pruebas. Un **conjunto de pruebas** (o *set* de pruebas), no es más que un conjunto de casos de prueba de un nivel de pruebas determinado (p.e., un conjunto de pruebas unitarias). Finalmente, dentro de la planeación del proceso de *testing*, tenemos el caso de prueba.



Casos de prueba

Un **caso de prueba** especifica y, en algunos casos, documenta una prueba que se le realiza a un componente o aplicación de software. Por lo general, un caso de prueba tiene los siguientes componentes:

- Los **valores de entrada** (del caso de prueba), son los valores de entrada o parámetros necesarios para la ejecución del caso de prueba en cuestión.
- Los **resultados esperados** son los valores, datos de salida y/o funcionamiento esperados de parte del componente o sistema si este se comporta como se espera de acuerdo a los valores de entrada.
- Por su parte, los **valores prefijados** (o *prefix values*) son las entradas o parámetros necesarios para establecer el sistema o componente de software en un estado apropiado previo al inicio del caso de prueba, es decir, antes de recibir los valores de entrada mencionados arriba.
- Finalmente, los **valores postfijados** (o *postfix values*) son las entradas a ingresar al sistema después de que reciba los valores de entrada, y que son necesarios para devolver el sistema a un estado deseado.

Diferentes estándares y marcos de trabajo de calidad de software proponen maneras diferentes de documentar los casos de prueba. Sin embargo, en esta lectura sólo debemos tener en cuenta los componentes listados arriba, de cara a su automatización. Es así como surge el concepto de *script* de prueba ejecutable. Un **script de prueba** es un caso de prueba preparado de tal forma que pueda ser ejecutado automáticamente usando un software de pruebas, produciendo un reporte de resultados. A algunos de estos softwares de pruebas se les denomina *frameworks* de automatización de pruebas.

JUnit

Un **framework de automatización de pruebas** es un conjunto de conceptos y herramientas que soportan y posibilitan la automatización de pruebas, siendo dependientes del lenguaje de programación. Es decir, existen *frameworks* de pruebas diferentes para cada lenguaje. Por lo general, la mayor parte de dichos *frameworks* soportan:

- **Aserciones** para evaluar los resultados obtenidos vs. los resultados esperados.
- La posibilidad de **compartir datos comunes** entre diferentes pruebas.
- **Sets o conjuntos de pruebas**, para organizar y ejecutar pruebas más fácilmente.
- La posibilidad de ejecutar pruebas, tanto mediante línea de comandos, como mediante una interfaz gráfica de usuario.

Existe una gran variedad de *frameworks* de pruebas para el lenguaje Java, asociados a diferentes niveles (p.e., Cucumber para pruebas de aceptación o JMeter para pruebas de carga). En cuanto a las pruebas unitarias, el *framework* de automatización más popular es JUnit.

JUnit es un *framework* de automatización de pruebas de código abierto para Java. Al ser de código abierto, es desarrollado, mantenido y mejorado por una comunidad de desarrolladores no comercial alrededor del mundo. Se usa para escribir y ejecutar pruebas unitarias y de integración, automatizadas y repetibles. Además, puede ser usado por separado, sin embargo, es muy común que se utilice con la ayuda del IDE o de un servidor de integración continua.



Con JUnit se pueden probar un método, un objeto o la interacción entre varios objetos. A esto nos referimos arriba cuando decimos que permite automatizar pruebas unitarias y de integración. Para cualquiera de los casos, una prueba se implementa mediante un **método de prueba**. Por otro lado, tenemos las **clases de prueba** que incluyen uno o varios métodos de prueba. Además de los métodos de prueba, una clase de prueba puede incluir métodos que preparen el estado del objeto a probar antes y/o después de realizarse las pruebas (implementando los *prefix values* y los *postfix values*).

En JUnit, las pruebas se hacen, principalmente, mediante aserciones, una **aserción** es una instrucción en la cual chequeamos que se cumpla una condición en la ejecución del código. Si dicha condición se cumple (y la aserción es verdadera), la prueba “pasa”. Si la condición no se cumple, la prueba “no pasa”. JUnit cuenta con diferentes aserciones, disponibles en la documentación oficial del *framework*¹.

Ejemplo: Pruebas a una Calculadora

Para automatizar pruebas unitarias usando JUnit, primero es necesario agregar la dependencia de JUnit en el proyecto con la ayuda del IDE. La forma de instalación dependerá del IDE, de la versión del IDE y del JDK (*Java Development Kit*). En versiones del JDK inferiores a la 8, será necesario usar JUnit 4. En versiones del JDK a partir de la 8 se podrá usar JUnit 5. En caso de que el proyecto se esté configurando usando Maven, bastará con agregar la siguiente dependencia en el archivo de configuración pom.xml². Recordemos que este archivo siempre deberá estar situado en la carpeta raíz del proyecto, junto a la carpeta *src*.

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.1.0</version>
  <scope>test</scope>
</dependency>
```

¹Documentación oficial de JUnit 5: <https://junit.org/junit5/docs/current/user-guide/>

²Un ejemplo del archivo pom.xml para JUnit 5 y el JDK 11 se encuentra aquí: https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/pom.xml

Como se mencionó arriba, las pruebas unitarias se usan para evaluar un método o una clase, en el presente ejemplo tenemos la clase `Calculadora`, que cuenta con dos métodos, uno que devuelve el cuadrado de un número dado, y otro que devuelve el cubo de un número. Tal como se puede apreciar a continuación³. La clase mencionada se encuentra en el paquete `co.edu.udea.udea_ruta2_ciclo2.pruebas`, que se deberá crear previamente en la sección *Source packages* dentro del explorador de proyectos (donde se crean los paquetes normalmente).

```
package co.edu.udea.udea_ruta2_ciclo2.poo.
pruebas;

public class Calculadora {

    public Double cuadrado(Double numero) {
        return numero * numero;
    }

    public Double cubo(Double numero) {
        return numero * numero * numero;
    }

}
```

Y abajo, se puede apreciar la clase de prueba `TestCalculadora`, que explicaremos en total detalle a continuación⁴. La clase mencionada se encuentra también en el paquete `co.edu.udea.udea_ruta2_ciclo2.pruebas`, pero en este caso ese paquete se creará en la sección *Test packages* del explorador de proyectos. La existencia de esta sección permite que los paquetes del código fuente ejecutable y los paquetes de las pruebas se gestionen por separado.

³El código fuente de la clase `Calculadora` se puede encontrar acá: https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/poo/pruebas/Calculadora.java

⁴El código fuente de la clase `TestCalculadora` se puede encontrar acá: https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/test/java/co/edu/udea/udea_ruta2_ciclo2/poo/pruebas/TestCalculadora.java

```
package co.edu.udea.udea_ruta2_ciclo2.poo.
pruebas;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.
Assertions.*;

public class TestCalculadora {

    //Pruebas unitarias para el cuadrado
    @Test
    public void testCuadradoPositivo() {
        Calculadora calc = new Calculadora();
        assertEquals(25.0, calc.cuadrado(5.0),
0.000000001);
    }

    @Test
    public void testCuadradoCero() {

Calculadora calc = new Calculadora();
        assertEquals(0.0, calc.cuadrado(0.0),
0.000000001);
    }

    @Test
    public void testCuadradoUno() {
        Calculadora calc = new Calculadora();
        assertEquals(1.0, calc.cuadrado(1.0),
0.000000001);
    }

    @Test
    public void testCuadradoNegativo() {
        Calculadora calc = new Calculadora();
        assertEquals(16.0, calc.
cuadrado(-4.0), 0.000000001);
    }
}
```

```
//Pruebas unitarias para el cubo
@Test
public void testCuboPositivo() {
    Calculadora calc = new Calculadora();
    assertEquals(125.0, calc.cubo(5.0),
0.000000001);
}

@Test
public void testCuboCero() {
    Calculadora calc = new Calculadora();
    assertEquals(0.0, calc.cubo(0.0),
0.000000001);
}

@Test
public void testCuboUno() {
    Calculadora calc = new Calculadora();
    assertEquals(1.0, calc.cubo(1.0),
0.000000001);
}

@Test
public void testCuboMenosUno() {
    Calculadora calc = new Calculadora();
    assertEquals(-1.0, calc.cubo(-1.0),
0.000000001);
}

@Test
public void testCuboNegativo() {
    Calculadora calc = new Calculadora();
    assertEquals(-64.0, calc.cubo(-4.0),
0.000000001);
}
}
```


El código anterior consta de una **clase de prueba** compuesta por diferentes **métodos de prueba**. Si lo observamos bien, encontraremos lo siguiente.

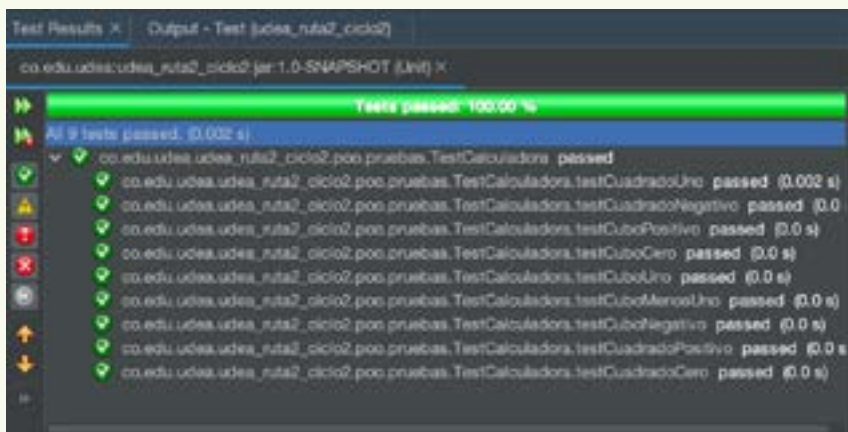
- Dos cláusulas `import`: con la primera importamos la anotación `@Test`, con la segunda importamos todos los métodos estáticos de la clase `Assertions`. Dichos métodos son las aserciones que se usan posteriormente en los métodos de prueba.
- Cada uno de los métodos de la clase son métodos de prueba. Para indicar que un método es de prueba, se pone la anotación `@Test` justo en la línea anterior al método. Esto posibilitará que, al correr las pruebas, el *framework* sepa cuáles debe ejecutar.
- Cada método tiene un nombre descriptivo respecto a lo que queremos probar en cada caso.
- Dentro de cada uno de los métodos de prueba, hay una aserción. Si la aserción se cumple, la prueba “pasa”, y si no se cumple, la prueba “no pasa”.

Si bien JUnit soporta diferentes tipos de aserciones⁵, en el ejemplo previo se usa solamente `assertEquals`. Esta verifica si un método u operación devuelve un valor *igual* a un valor esperado y consta de un método que, como observamos, tiene tres parámetros:

- El primer parámetro, con el valor que esperamos que devuelva el método de la clase `Calculadora` que estamos llamando, de acuerdo con un valor de entrada.
- El segundo, donde hacemos el llamado al método de la clase `Calculadora` que queremos probar, pasándole un valor de entrada.
- El tercero, que consiste en un margen de error aceptable a la hora de evaluar si lo devuelto por el método evaluado es igual al valor esperado.

En resumen, en el ejemplo anterior se prueban los métodos de la clase `Calculadora` usados para hallar el cuadrado y el cubo de un número. En cada caso pasamos a los métodos a evaluar valores de entrada y definimos las aserciones contra valores conocidos, con un margen de error.

Para ejecutar las pruebas y verificar que la clase `Calculadora` las pasa, basta con hacer clic derecho en el proyecto y hacer clic a la opción *Test* del menú contextual. En el panel respectivo aparecerá un breve reporte de las pruebas que pasaron y las que no pasaron. De haber pasado todas (como debería de ser en este caso), en Apache NetBeans el reporte debería tener el siguiente aspecto.



⁵Las diferentes aserciones en JUnit 5 se pueden encontrar acá: <https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html>