Hoy en día para la realización de pruebas unitarias en distintos lenguajes se han desarrollado un conjunto de herramientas denominado arquitectura xUnit. Para el caso particular del lenguaje Java se ha desarrollado una instancia del mismo llamada Junit. La misma es de código abierto y fue creada por Kent Beck y Erich Gamma y actualmente atraviesa por su versión 5, específicamente la 5.8.2 El concepto fundamental en esta herramienta es el caso de prueba (*test case*), y la suite de prueba (*test suite*). Los casos de prueba son clases o módulos que disponen de métodos para probar los métodos de una clase o módulo concreta/o. Así, para cada clase que quisiéramos probar definiríamos su correspondiente clase de caso de prueba. El método de prueba invoca lo que se va a probar y luego se utilizan aserciones para probar resultar esperados. Mediante las suites podemos organizar los casos de prueba, de forma que cada suite agrupa los casos de prueba de módulos que están funcionalmente relacionados. Por cada clase hay una clase de pruebas que contiene al menos un método de prueba por cada método de la clase. De esta forma, construimos programas que sirven para probar nuestros módulos, y que podremos ejecutar de forma automática. A medida que la aplicación vaya avanzando, se dispondrá de un conjunto importante de casos de prueba, que servirá para hacer pruebas de regresión. Eso es importante, puesto que cuando cambiamos un módulo que ya ha sido probado, el cambio puede haber afectado a otros módulos, y sería necesario volver a ejecutar las pruebas para verificar que todo sigue funcionando.

Para la implementación de los casos de prueba los IDE más comunes, como Eclipse e IntelliJ, ya tendrán la integración de prueba JUnit instalada de forma predeterminada. Si uno no está usando un IDE y quizás se basa únicamente en un sistema de compilación como Maven o Gradle, la instalación de JUnit 4/5 se maneja a través de pom.xml o build.gradle, respectivamente.

**Métodos**

A grandes rasgos, una clase de Test realizada para ser tratada por JUnit tiene una estructura con 4 tipos de métodos:

* Método setUp: Asignamos valores iniciales a variables antes de la ejecución de cada test. Si sólo queremos que se inicialicen al principio una vez, el método se debe llamar "setUpClass"
* Método tearDown: Es llamado después de cada test y puede servir para liberar recursos o similar. Igual que antes, si queremos que sólo se llame al final de la ejecución de todos los test, se debe llamar "tearDownClass"
* Métodos Test: Contienen las pruebas concretas que vamos a realizar.
* Métodos auxiliares.

**Anotaciones**

Se llama anotaciones a caracteres especiales, que se han incluido en la versión 4, para intentar simplificar más la labor del programador. Se trata de palabras clave que se colocan delante de los métodos definidos antes y que indican a las librerías JUnit instrucciones concretas.

A continuación, pasamos a ver las más relevantes:

* @RunWith: Se le asigna una clase a la que JUnit invocará en lugar del ejecutor por defecto de JUnit
* @Before: Indicamos que el siguiente método se debe ejecutar antes de cada test (precede al método setUp). Si tiene que preceder al método setUpClass, la notación será "@BeforeClass"
* @After: Indicamos que el siguiente método se debe ejecutar después de cada test (precede al método tearDown). Si tiene que preceder al método tearDownClass, la notación será "@AfterClass"
* @Test: Indicamos a Junit que se trata de un método de Test. En versiones anteriores de JUnit los métodos tenían que tener un nombre con la siguiente estructura: "Test". Con esta notación colocada delante de los métodos podemos elegir el nombre libremente.

**Funciones de aceptación/rechazo:**

Una vez hemos creado las condiciones para probar que una funcionalidad concreta funciona es necesario que un validador nos diga si estamos obteniendo el resultado esperado o no. Para esta labor se definen una lista de funciones (incluidas en la clase Assert) que se pueden ver detalladas en el javadoc de JUnit. Se detallan las más comunes:

* sertArrayEquals: Recibe como parámetro dos arrays y comprueba si son iguales. Devuelve assertionError si no se produce el resultado esperado
* sertEquals: Realiza la comprobación entre dos valores de tipo numérico. Devuelve assertionError si no se produce el resultado esperado
* sertTrue: Comprueba si una condición se cumple. Devuelve assertionError si no se produce el resultado esperado
* il: devuelve una alerta informando del fallo en el test

**Ejemplo de una clase Test**:

import org.junit.AfterClass;     
import org.junit.Assert;     
import org.junit.BeforeClass;     
import org.junit.Test;     
    
public class pruebaTest {     
    
    @BeforeClass    
    public static void setUpClass() throws Exception {     
    //Inicialización general de variables, escritura del log...     
    }     
         
    @AfterClass    
    public static void tearDownClass() throws Exception {     
    //Liberación de recursos, escritura en el log...     
    }     
         
    @Before    
    public void setUp() {     
    //Inicialización de variables antes de cada Test     
    }     
         
    @After    
    public void tearDown() {     
    //Tareas a realizar después de cada test     
    }     
         
    @Test    
    public void comprobarAccion() {     
    //Creamos el entorno necesario para la prueba     
    //Usamos alguna de las funciones arriba descritas      
    //para realizar la comprobación     
    }     
         
    public void MetodoAuxiliar() {     
    //tareas auxiliares     
    }     
         
}