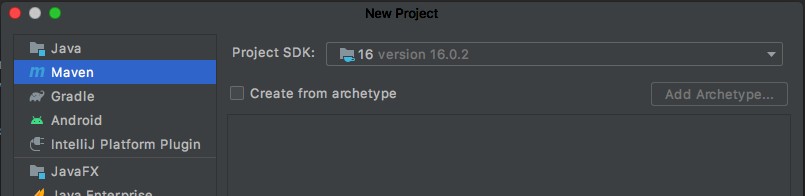
Pruebas unitarias jUnit.

# Configuración IntelliJ con jUnit y Maven.

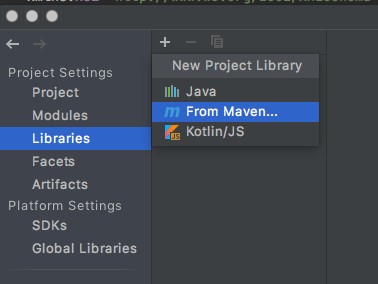
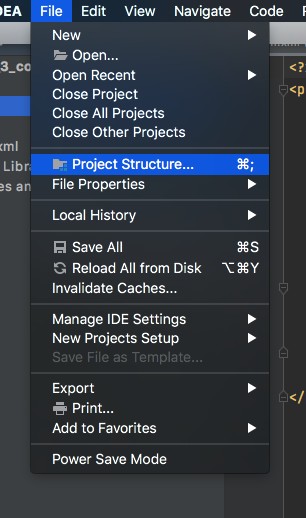
En esta guía vamos a ver cómo configurar jUnit en IntelliJ. Además, introduciremos los primeros ejemplos de pruebas unitarias. Toda la información de instalación, está extraída de <https://www.jetbrains.com/help/idea/testing.html>

En primer lugar, crearemos un Proyecto en IntelliJ usando Maven como herramienta de construcción de Software.



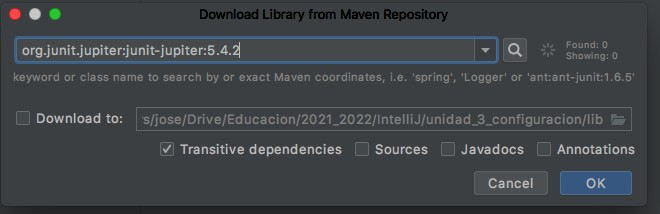
A continuación, accederemos a las opciones del proyecto para añadir las librerías necesarias. Iremos a

*Project Structure* y haremos click sobre *Libraries -> From Maven*.

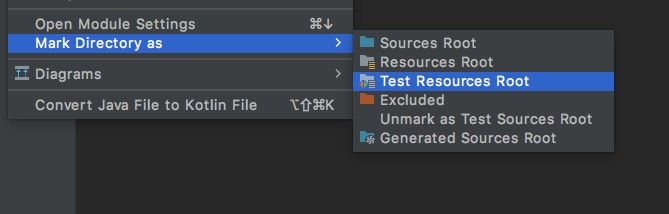
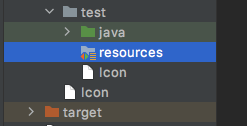


En el formulario nos solicitarán la librería a instalar. Introduciremos el valor

org.junit.jupiter:junit-jupiter:5.4.2 y pulsaremos OK**.**



En la carpeta *src/test* crearemos un directorio llamado *resources* y lo marcaremos como **Test Resources Root** (botón derecho -> Mark directory as). A partir de ahora, los Tests que creemos se guardarán en el directorio de tests del proyecto**.**



# Primeros tests unitarios con jUnit.

Para empezar a trabajar, vamos a crear una clase en nuestro proyecto llamada *Mayor.* Esta clase tendrá dos métodos:

* *Mayor( int a, int b) :* Pasamos dos números enteros y nos devuelve un entero con el mayor.
* *EsMayor(int a, int b) :* Pasamos dos números enteros y devuelve un booleano si el primero es mayor que el segundo.

public class Mayor {

public static int Mayor(int a, int b ) { int mayor = a;

if ( b > a ) mayor = b;

return mayor;

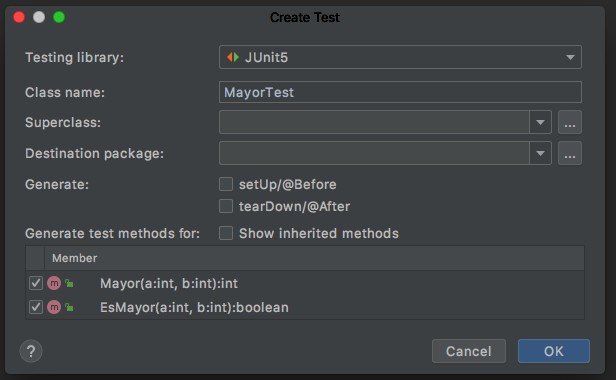
}

public static boolean EsMayor(int a, int b) { return a > b;

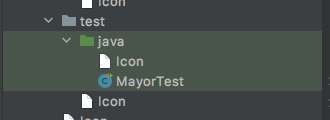
}

}

Una vez creada la clase y guardada en el nuestro directorio *src*, vamos a pulsar *Alt+Enter* sobre la línea de declaración de la clase y se nos abrirá un menú para crear nuestros tests. El formulario nos pregunta si queremos hacer tests sobre los dos métodos, los seleccionaremos y daremos a OK.



Se nos creará una nueva clase llamada MayorTest en el directorio de tests/java.



Es en esta clase donde trabajaremos todos nuestros tests unitarios. Dichos tests, los habremos diseñado previamente haciendo un análisis de Prueba de Caja Blanca o Caja Negra. Es decir, determinaremos todos los posibles valores que pueden dar lugar a errores en nuestro código.

La idea principal es crear tests para cada método y que su ejecución nos indique si han pasado los tests de manera correcta. Así validamos que nuestro código hace lo que realmente debe hacer.

Para ello, usaremos funciones concretas de jUnit. Para el siguiente ejemplo, sólo necesitaremos

*assertEquals*, *assertTrue* y *assertFalse*.

* ***assertEquals( valor esperado, valor devuelto) :*** El resultado del test será correcto si el valor esperado que le indicamos es igual al valor devuelto por nuestra función.
* ***assertTrue( resultado de la condición) :*** Si tenemos un método que devuelve un booleano y su resultado esperado es True, pasará el test.
* ***assertFalse (resultado de la condición) :*** Si tenemos un método que devuelve un booleano y el resultado esperado es False, pasará el test.

Para comprender mejor todo esto, fíjate en el código de *MayorTest.*

class MayorTest { @Test

void mayor() {

int mayorUltimo = Mayor.Mayor(3, 4); assertEquals(4, mayorUltimo);

int mayorPrimero = Mayor.Mayor(4,3); assertEquals(4, mayorPrimero);

int mayorIguales = Mayor.Mayor(4,4); assertEquals(4, mayorIguales);

int mayorNegativos = Mayor.Mayor(-3, -4); assertEquals(-3, mayorNegativos);

}

@Test

void esMayor() { assertTrue(Mayor.EsMayor(4,3)); assertTrue(Mayor.EsMayor(-3, -4)); assertFalse(Mayor.EsMayor(3,4)); assertFalse(Mayor.EsMayor(4,4)); assertFalse(Mayor.EsMayor(-4, -3));

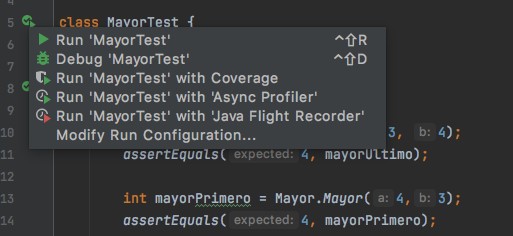
}

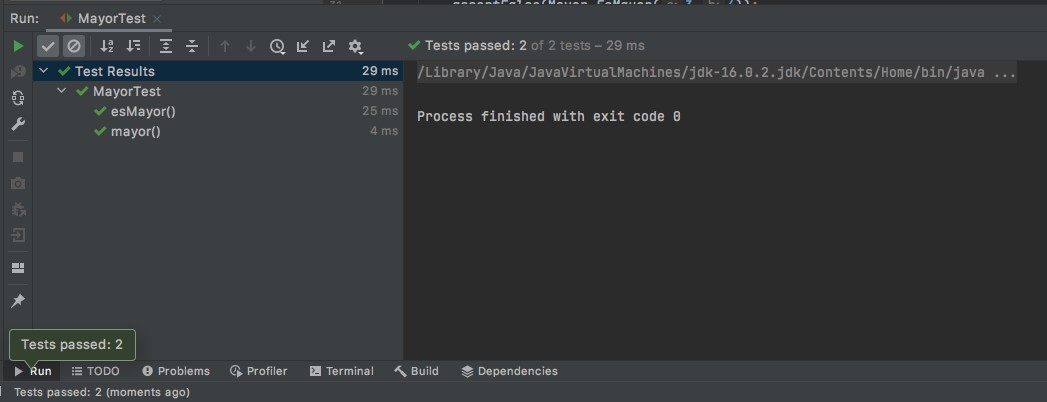
}

El primer test (*void mayor*) realiza pruebas con *assertEquals* para el método *Mayor(a,b)* de la clase *Mayor*. Dichas pruebas están definidas para comprobar distintos casos: *el último es mayor, el primero es mayor, mayor de dos números iguales y mayor con números negativos.*

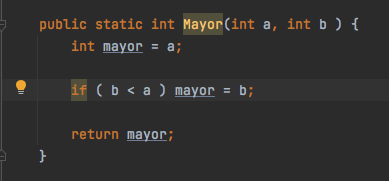
El segundo test (void esMayor) realiza pruebas con *assertTrue* y *assertFalse* para el método *EsMayor(a,b)* de la clase *Mayor*. Las pruebas comprueban: *el primero es mayor, el mayor de dos negativos* (ambas esperan **True**) *y luego el segundo es mayor, son iguales el segundo es mayor de dos negativos* (todas esperan **False**)*.*

Llega el momento de lanzar los tests. Para ello lanzamos la ejecución de la clase MayorTest y observamos los resultados. Si todo ha ido bien, los test habrán pasado y nuestro código habrá superado la validación.





Vamos a hacer una pequeña modificación a nuestro código para que no valide los tests de Mayor(a,b).



En este caso, si lanzamos los tests observaremos que en los resultados no se obtiene lo esperado. Por tanto, no pasará nuestros tests de validación.

La consola nos muestra información sobre el valor esperado y el obtenido. Aprovecharemos esto para modificar nuestro código.

