



**Universidad de El Salvador**

**Facultad Multidisciplinaria de Occidente**

**Departamento de Ingeniería y Arquitectura**

---

**Asociación de Estudiantes de Ingeniería y Arquitectura**

**Curso: Programación 2.5**

## ***Guía de Trabajo***

### ***Pruebas Unitarias con JUnit***

#### ***Prueba Unitaria***

En programación, una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo en diseño estructurado o en diseño funcional una función o un procedimiento, en diseño orientado a objetos una clase. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido entonces el estado final es válido.

Para que una prueba unitaria tenga la calidad suficiente se deben cumplir los siguientes requisitos:

- Automatizable

No debería requerirse una intervención manual. Esto es especialmente útil para integración continua.

- Completas

Deben cubrir la mayor cantidad de código.

- Repetibles o Reutilizables

No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.

- Independientes

La ejecución de una prueba no debe afectar a la ejecución de otra.

- Profesionales

Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

Aunque estos requisitos no tienen que ser cumplidos al pie de la letra, se recomienda seguirlos o de lo contrario las pruebas pierden parte de su función.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

- Fomentan el cambio

Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.

- Simplifica la integración

Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.

- Documenta el código

Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.

- Separación de la interfaz y la implementación

Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro, a veces usando objetos mock (mock object) para simular el comportamiento de objetos complejos.

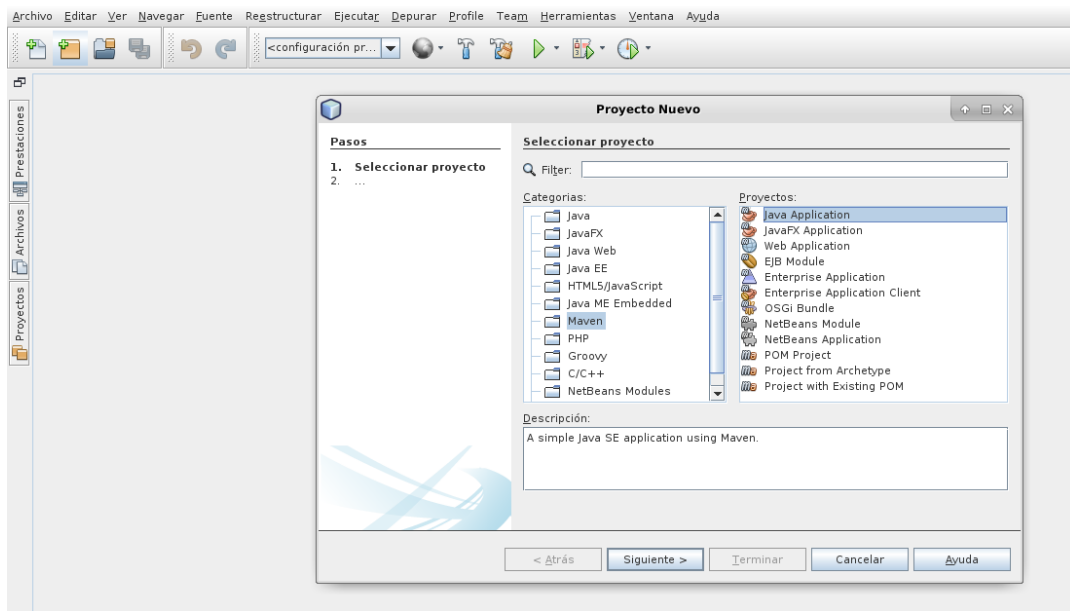
- Los errores están más acotados y son más fáciles de localizar

Dado que tenemos pruebas unitarias que pueden desenmascararlos.

## ***JUnit***

JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

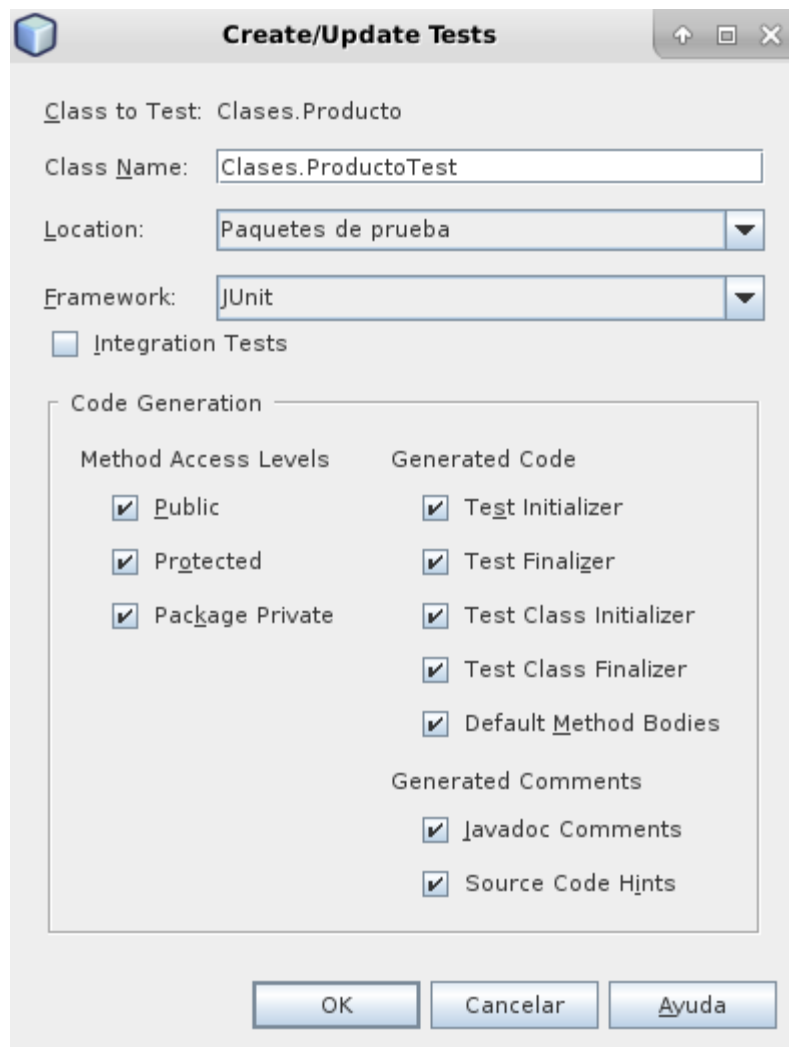
## 1. Crear un nuevo proyecto



## 2. Crear un paquete llamado Clases, luego crear la clase Producto.java

```
1 package Clases;
2
3 /**
4  *
5  * @author joker
6  */
7 public class Producto {
8
9     public String nombre;
10    public double precio;
11    public String categoria;
12
13    public String getNombre() {
14        return nombre;
15    }
16
17    public void setNombre(String nombre) {
18        this.nombre = nombre;
19    }
20
21    public double getPrecio() {
22        return precio;
23    }
24
25    public void setPrecio(double precio) {
26        this.precio = precio;
27    }
28
29    public String getCategoria() {
30        return categoria;
31    }
32
33    public void setCategoria(String categoria) {
34        this.categoria = categoria;
35    }
36
37    public double obtenerNuevoPrecio(double precioAnterior, double incremento){
38        double total= precioAnterior+incremento;
39        return total;
40    }
41
42    public double obtenerImpuesto(double precioVenta){
43        double impuesto=precioVenta*0.13;
44        return impuesto;
45    }
46
47 }
48
```

3. Ir a la pestaña de herramientas, y seleccionar Create/Update Tests.



Se crearan por defecto algunos Test en la clase creada. Se identifican por tener la notación @Test.

```
43 | @Test
44 | public void testGetNombre() {
45 |     System.out.println("getNombre");
46 |     Producto instance = new Producto();
47 |     String expectedResult = "";
48 |     String result = instance.getNombre();
49 |     assertEquals(expResult, result);
50 |     // TODO review the generated test code and remove the default call to fail.
51 |     fail("The test case is a prototype.");
52 | }
53 |
```

## *JUnit Asserts*

JUnit proporciona métodos estáticos en la clase Assert para probar ciertas condiciones. Estos métodos de afirmación típicamente comienzan con assert y le permiten especificar el mensaje de error, el esperado y el resultado real. Un método afirmación compara el valor real devuelto por una prueba para el valor esperado, y se produce una `AssertionException` si la prueba de comparación falla.

Algunos métodos estáticos del Assert:

- `assertArrayEquals()`
- `assertEquals()`
- `assertTrue()`
- `assertFalse()`
- `assertNull()`
- `assertNotNull()`
- `assertSame()`
- `assertNotSame()`
- `assertThat()`

4. Modificar los Test creados para comprobar el funcionamiento de nuestros métodos.

```
/**
 * Test of setNombre/getNombre method, of class Producto.
 */
@Test
public void testGetNombre() {
    String nombre="Pizza";
    Producto instance = new Producto();
    instance.setNombre(nombre);
    String result = instance.getNombre();
    assertEquals(nombre, result);
}
```

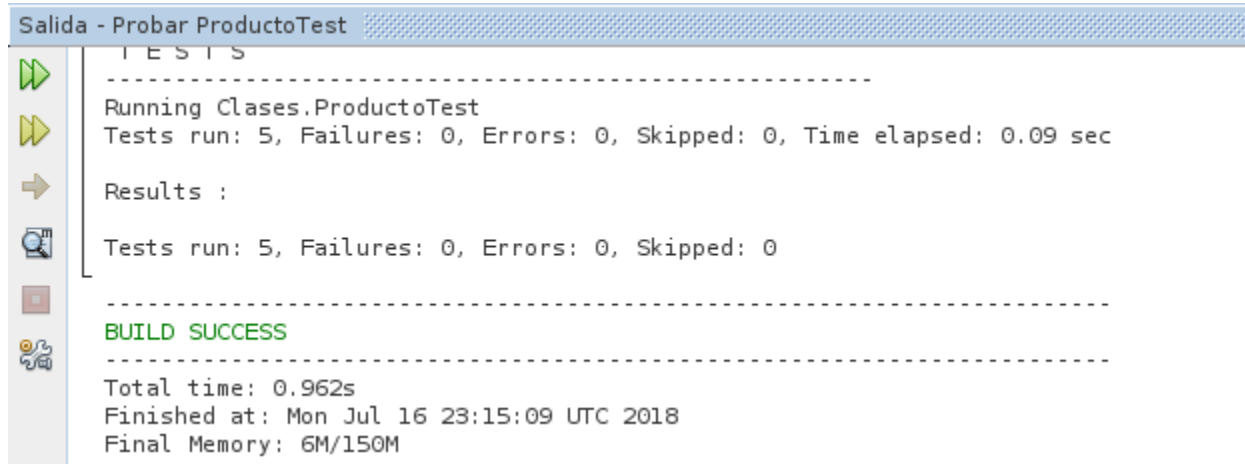
```
/**
 * Test of setPrecio/getPrecio method, of class Producto.
 */
@Test
public void testGetPrecio() {
    Double precio=2.75;
    Producto instance = new Producto();
    instance.setPrecio(precio);
    double result = instance.getPrecio();
    assertNotNull(result);
}

/**
 * Test of setCategoria/getCategoria method, of class Producto.
 */
@Test
public void testGetCategoria() {
    Producto instance = new Producto();
    String categoria= "bebida";
    instance.setCategoria(categoria);
    String result = instance.getCategoria();
    assertEquals(categoria, result);
}

/**
 * Test of obtenerNuevoPrecio method, of class Producto.
 */
@Test
public void testObtenerNuevoPrecio() {
    double precioAnterior = 1.0;
    double incremento = 0.5;
    Producto instance = new Producto();
    double expectedResult = 1.5;
    double result = instance.obtenerNuevoPrecio(precioAnterior, incremento);
    assertEquals(expectedResult, result, 0.0);
}

/**
 * Test of obtenerImpuesto method, of class Producto.
 */
@Test
public void testObtenerImpuesto() {
    double precioVenta = 10.0;
    Producto instance = new Producto();
    double expectedResult = 1.30;
    double result = instance.obtenerImpuesto(precioVenta);
    assertEquals(expectedResult, result, 0.1);
}
```

5. Seleccionar la clase ProductoTest.java, hacer click derecho y luego seleccionar Probar archivo.



```
Salida - Probar ProductoTest
TESTS
-----
Running Clases.ProductoTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.09 sec

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

-----
BUILD SUCCESS
-----
Total time: 0.962s
Finished at: Mon Jul 16 23:15:09 UTC 2018
Final Memory: 6M/150M
```

Se obtendrá esa salida, comprobando los resultados de nuestra prueba, si se desea se puede ver el resultado en la ventana.

