Taller Práctico de Calidad de Software en Java (Pruebas Unitarias)

## Objetivo

Aprender a diseñar, implementar y ejecutar pruebas unitarias en Java utilizando JUnit, siguiendo buenas prácticas de aseguramiento de calidad de software.

## Requisitos Previos

* + Conocimientos básicos de Java.
  + Tener instalado JDK 11+.
  + Tener un IDE como IntelliJ IDEA o Eclipse.
  + Conocer el uso básico de Maven o Gradle.

## Actividades Prácticas

**Actividad 1: Configuración del Entorno con JUnit**

**Objetivo:** Preparar un proyecto Java para ejecutar pruebas unitarias.

## Instrucciones:

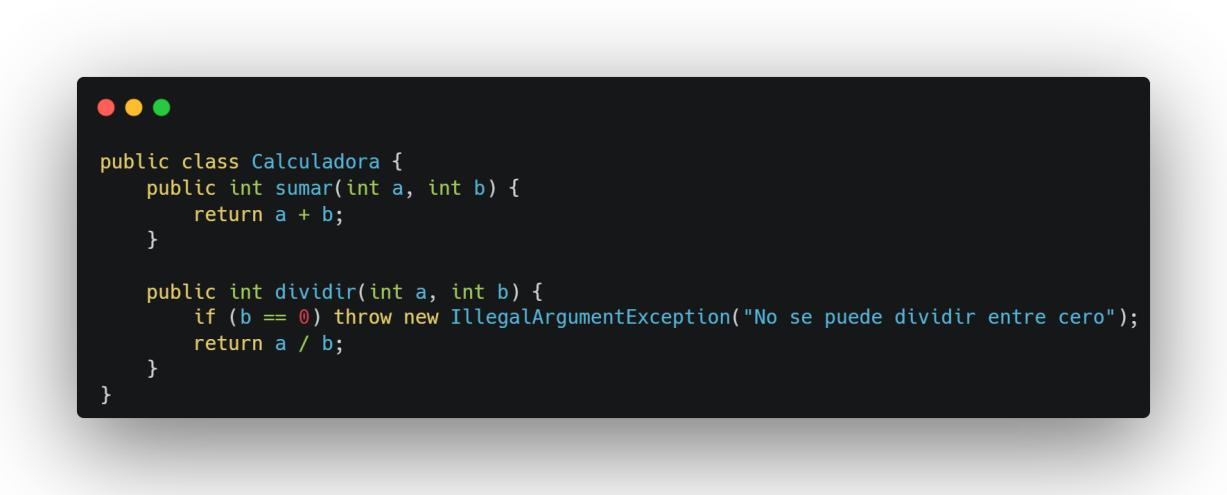
1. Crea un nuevo proyecto Maven.
2. Sigue las instrucciones de este video para entender o instalar Junit <https://www.youtube.com/watch?v=4wdQ_Xdg_l0>
3. Verifica que tu IDE reconoce el framework de pruebas.

## Actividad 2: Crear Clase y Método a Probar

**Objetivo:** Tener una funcionalidad sencilla para testear.

## Instrucciones:

1. En src/main/java, crea la clase Calculadora:



# Actividad 3: Crear Casos de Prueba Unitarios

**Objetivo:** Diseñar y ejecutar pruebas usando JUnit.

## Instrucciones:

1. En src/test/java, crea la clase CalculadoraTest:



# Actividad 4: Ejecutar y Analizar Resultados

**Objetivo:** Verificar que todas las pruebas pasen y analizar fallos.

## Instrucciones:

1. Ejecuta las pruebas desde tu IDE o usando Maven:



1. Observa el reporte de resultados.
2. Si alguna prueba falla, corrige el código y vuelve a ejecutar.

## Buenas Prácticas

* + Un caso de prueba debe ser independiente de otros.

Rl: Lo son, estan separados por proposito, Cada prueba en PlatilloCrudTest inicializa su propio PlatilloCrud en @BeforeEach, por lo que no dependen unas de otras.

* + Nombrar las pruebas de forma descriptiva.

Rl: Hecho, ya esta en su lado derecho la descripcion de lo que hace el test nombrado

test\_crear\_platillo\_valido\_debe\_agregarse\_al\_menu() y test\_crear\_platillo\_sin\_nombre\_debe\_lanzar\_excepcion()  
 Cada nombre indica qué se prueba y qué se espera.

* + Probar tanto casos esperados como escenarios de error.

Casos esperados: agregar platillos válidos.

Escenarios de error: crear platillos sin nombre o sin descripción.

* + Mantener el código de pruebas limpio y legible.

Rl: Lo está, Esta separado por Componentes

## Recursos

* + JUnit 5 Documentation
  + Mockito para pruebas con dependencias simuladas.
  + AssertJ para aserciones más expresivas.

## Proyecto de Aula

**Objetivo:** Aplicar los conceptos y técnicas aprendidas a un proyecto real del curso.

## Instrucciones:

1. Elige un módulo o funcionalidad de tu proyecto de aula (por ejemplo: módulo de usuarios, carrito de compras, gestión de inventario).

Rl: Elejí el modulo de platillos

1. Diseña al menos 5 casos de prueba unitaria para validar las funciones principales de ese módulo.

### **Casos de Prueba (mínimo 5)**

1. **Crear un platillo válido debe agregarse al menú**
   1. Entrada: ("Tacos", 25.0, "Tacos de carne")
   2. Resultado esperado: el menú tiene 1 platillo.
2. **Crear un platillo sin nombre debe lanzar excepción**
   1. Entrada: ("", 30.0, "Sopa de verduras")
   2. Resultado esperado: excepción con mensaje "Los campos obligatorios no pueden estar vacíos.".
3. **Crear un platillo sin descripción debe lanzar excepción**
   1. Entrada: ("Pizza Margarita", 40.0, "")
   2. Resultado esperado: excepción con mensaje "Los campos obligatorios no pueden estar vacíos.".
4. **El menú inicial debe estar vacío**
   1. Entrada: ninguna.
   2. Resultado esperado: menu.isEmpty() == true.
5. **Agregar varios platillos debe guardarlos todos en el menú**
   1. Entrada: dos platillos distintos.
   2. Resultado esperado: el menú contiene 2 platillos.
6. Implementa las pruebas en Java utilizando JUnit siguiendo la estructura y ejemplos vistos.
7. Ejecuta las pruebas y documenta los resultados en un archivo o reporte.
8. Presenta el código de las pruebas y el reporte al instructor para su retroalimentación.