**Pruebas Unitarias con JUnit 5 en Java**

**Introducción**

Las pruebas unitarias son una parte fundamental del desarrollo de software moderno, ya que permiten verificar el comportamiento individual de componentes o unidades de código. JUnit es un popular framework de pruebas unitarias para Java que facilita la escritura y ejecución de estas pruebas. En este informe, exploraremos cómo realizar pruebas unitarias con JUnit 5 en Java.

**1. Anotaciones Básicas:**

* `@Test`: Esta anotación se usa para marcar un método como una prueba unitaria. JUnit ejecutará cualquier método marcado con `@Test` durante la ejecución de las pruebas.
* `@BeforeEach`: Esta anotación indica que un método debe ejecutarse antes de cada prueba (`@Test`), lo que permite inicializar los recursos necesarios para cada prueba de manera independiente.
* `@AfterEach`: A diferencia de `@BeforeEach`, `@AfterEach` se usa para indicar que un método debe ejecutarse después de cada prueba, generalmente para liberar recursos o limpiar el estado después de la prueba.

**2. Anotaciones para Configuración Global:**

* `@BeforeAll`: Marca un método que debe ejecutarse una sola vez antes de todas las pruebas en la clase. Se usa para configurar recursos que serán compartidos por todas las pruebas.
* `@AfterAll`: Esta anotación indica que un método debe ejecutarse una sola vez después de todas las pruebas en la clase, útil para realizar tareas de limpieza o liberación de recursos globales.

**3. Aserciones:**

JUnit 5 proporciona métodos estáticos en la clase `org.junit.jupiter.api.Assertions` para realizar aserciones en las pruebas. Algunos de los métodos comunes incluyen:

* `assertEquals`: Compara dos valores esperados, siendo útil para verificar la igualdad entre objetos o valores primitivos.
* `assertNotEquals`: Similar a `assertEquals`, pero verifica que los valores sean diferentes.
* `assertNotNull`: Verifica que un objeto no sea nulo.
* `assertTrue`, `assertFalse`: Verifica que una expresión sea verdadera o falsa, respectivamente.
* `assertThrows`: Verifica que una excepción específica sea lanzada durante la ejecución de un bloque de código.

**4. Anotaciones para Agrupar y Filtrar Pruebas:**

* `@Nested`: Permite agrupar pruebas dentro de una clase de prueba, lo que facilita la organización de pruebas relacionadas.
* `@DisplayName`: Se utiliza para dar un nombre legible y descriptivo a una prueba, lo que mejora la claridad en los informes de ejecución de pruebas.
* `@Tag`: Etiqueta una prueba con una palabra clave que puede usarse para filtrar o ejecutar un subconjunto específico de pruebas.

**5. Desactivación y Condicionales:**

* `@Disabled`: Anota una prueba para desactivarla temporalmente, lo que es útil cuando una prueba no está lista para ejecutarse o se está depurando.
* `@EnabledOnOs`, `@EnabledIfSystemProperty`: Permiten habilitar o deshabilitar pruebas basadas en condiciones específicas del sistema operativo o propiedades del sistema.

**6. Extensiones y Métodos de Extensión:**

JUnit 5 introduce el concepto de extensiones (`Extensions`) que permiten agregar funcionalidades adicionales a las pruebas, como manejo de datos de prueba, inyección de dependencias, entre otros. Además, JUnit 5 admite métodos de extensión (`Extension Methods`) para extender el comportamiento de las clases de prueba de manera más flexible que las herencias tradicionales.

**7. Ejecución y Reportes:**

Para ejecutar pruebas unitarias con JUnit 5, puedes utilizar herramientas de construcción como Maven o Gradle, o ejecutarlas directamente desde un entorno de desarrollo integrado (IDE) como IntelliJ IDEA o Eclipse. JUnit 5 genera informes detallados sobre la ejecución de las pruebas, lo que facilita la identificación de problemas y el seguimiento del progreso de las pruebas a lo largo del tiempo.

**Conclusión**

JUnit 5 proporciona un conjunto robusto de funcionalidades y anotaciones para escribir y ejecutar pruebas unitarias efectivas en Java, lo que contribuye significativamente a la calidad y confiabilidad del software desarrollado. Es una herramienta poderosa para escribir y ejecutar pruebas unitarias en Java. Con las anotaciones proporcionadas y las capacidades de aserción, puedes asegurarte de que tus componentes de código funcionen como se espera y mantener la calidad de tu software a lo largo del tiempo. Integrar pruebas unitarias en tu flujo de trabajo de desarrollo te ayudará a detectar y corregir errores de manera temprana, mejorando la confiabilidad y mantenibilidad de tu código.