|  |  |  |
| --- | --- | --- |
|  | Universidad católica de santa maría |  |
| Escuela profesional de Ingeniería de sistemas |

**TESTING, IMPLANTACIÓN Y MANTENIMIENTO DE SISTEMAS – 04**

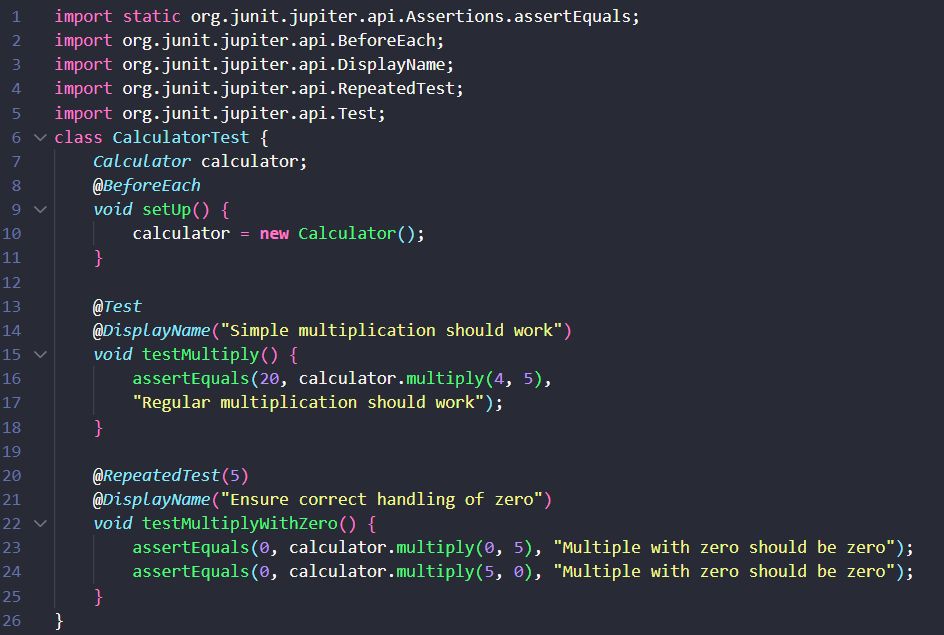
**Introducción al Software Testing**

**Cobertura AUTOMÁTICA de Código**

|  |  |  |
| --- | --- | --- |
| **CÓDIGO/DNI** | **APELLIDOS Y NOMBRES** | **FECHA** |
| 2021810041/72440510 | Pezo Vizcarra Adriel Francesco | 16/05/2024 |

**ACTIVIDADES**

1. **Crea una clase llamada "TestCalculator" que pruebe las cuatro operaciones básicas (suma, resta, multiplicación y división) de una calculadora. Define métodos de prueba para cada operación y verifica que los resultados sean correctos.**



1. **Crea una clase llamada "TestForm" que pruebe la funcionalidad de un formulario HTML sencillo. Define métodos de prueba para cada campo del formulario, incluyendo campos de texto, selectores de opciones, botones de radio y casillas de verificación. Verifica que los valores ingresados por el usuario se almacenen correctamente y que se realicen las validaciones de entrada adecuadas.**

**EJERCICIOS**

1. **Crear una clase de prueba para una función que encuentra el número más grande en una lista de números. La prueba debe verificar que el resultado es el esperado.**
2. **Crear una clase de prueba para una función que calcula la raíz cuadrada de un número. La prueba debe verificar que el resultado es el esperado.**
3. **Crea una clase de prueba llamada "TestForm" que pruebe la funcionalidad de un formulario HTML sencillo (4 campos como mínimo entre numeros, letras , correos electrónicos, etc ) . Defina un método de prueba para cada campo. Verifica que los valores ingresados por el usuario se almacenen correctamente y que se realicen las validaciones de entrada adecuadas.**

**CUESTIONARIO**

1. **¿Por qué es importante escribir pruebas unitarias?**

Hay 2 razones que destacan la importancia de las pruebas unitarias:

* Aseguran el Funcionamiento Correcto del Código: Las pruebas unitarias permiten al desarrollador verificar que cada función y método del código funciona correctamente. Esto es esencial para garantizar que el código cumpla con los requisitos y funcione como se espera [1].
* Facilitan la Identificación y Corrección de Errores: Al escribir casos de prueba para todas las funciones y métodos, las pruebas unitarias permiten identificar rápidamente cualquier error que surja como resultado de cambios en el código. Esto ahorra tiempo y dinero a largo plazo al facilitar la corrección de errores de manera oportuna [1].

1. **¿Qué se debe tener en cuenta al escribir pruebas unitarias?**

Hay 3 aspectos importantes a la hora de crear pruebas unitarias:

* Aislamiento de Dependencias: Es fundamental seguir el Principio de Dependencias Explícitas y utilizar la Inyección de Dependencias para evitar dependencias innecesarias en las pruebas unitarias. Mantener las pruebas unitarias en un proyecto separado de las pruebas de integración también garantiza que el proyecto de pruebas unitarias no tenga referencias a paquetes de infraestructura ni dependencias de estos [2].
* Evitar varias actuaciones: Al escribir las pruebas, intente incluir solo una actuación por prueba. Entre los enfoques comunes para usar solo una actuación se incluyen los siguientes [2].
* Consistencia y Claridad: Es vital seguir un esquema claro al nombrar las pruebas unitarias y mantener la consistencia en su estructura. Además, es importante realizar pruebas con un solo código a la vez para garantizar que las pruebas unitarias sean independientes y no se vean afectadas por cambios o mejoras en los requisitos [3].

1. **¿Cuál es la función de la anotación @Test en JUnit?**

La anotación @Test en JUnit se utiliza para marcar un método como un método de prueba. Esto indica a JUnit que el método anotado con @Test debe ejecutarse como parte de las pruebas unitarias. Es una forma de especificar que un método en particular es una prueba que debe ser ejecutada por el framework de pruebas JUnit. Esta anotación es fundamental para la creación y ejecución de pruebas unitarias en Java utilizando JUnit [4]. En JUnit 4, los casos de prueba se crean mediante anotaciones, por lo que es necesario utilizar Java 5 o posterior. Al marcar un método con la anotación @Test, se indica que ese método debe ser ejecutado como una prueba por JUnit. Esta anotación es esencial para implementar pruebas en JUnit 4. Además, la anotación @Test reemplaza la necesidad de heredar de clases específicas o cumplir con ciertas convenciones de nombres, lo que simplifica la creación de pruebas en comparación con versiones anteriores de JUnit [5].

1. **¿Qué es una aserción en JUnit y para qué se utiliza?**

Una aserción en JUnit es una declaración que verifica si una condición es verdadera durante la ejecución de un programa. Se utiliza para verificar el comportamiento. En caso de incumplir dichas aserciones el método de prueba fallará identificando el error producido [6].

Las aserciones en JUnit se utilizan dentro de los métodos de prueba anotados con @Test. Estas aserciones permiten verificar el comportamiento del código y, en caso de que una aserción falle, el método de prueba fallará identificando el error producido. Además, en JUnit 5, es posible agrupar aserciones con assertAll(), lo que informará cualquier aserción fallida dentro del grupo con un MultipleFailuresError.

1. **¿Qué ventajas tiene la automatización de pruebas unitarias con JUnit?**

La automatización de pruebas unitarias con JUnit ofrece varias ventajas significativas:

Detección Temprana de Errores: La automatización de pruebas unitarias con JUnit permite detectar errores en el código de manera temprana, lo que facilita su corrección antes de que se propaguen a otras partes del sistema [6].

Ahorro de Tiempo y Recursos: Al automatizar las pruebas unitarias con JUnit, se ahorra tiempo y recursos, ya que las pruebas pueden ejecutarse de forma rápida y repetitiva, lo que facilita la identificación de posibles problemas de manera eficiente [7].

Mejora de la Calidad del Software: La automatización de pruebas unitarias con JUnit contribuye a mejorar la calidad del software al garantizar que las diferentes partes del código funcionen como se espera, lo que resulta en un software más confiable y robusto.

Facilidad de Integración en IDEs: JUnit se integra fácilmente en los principales Entornos de Desarrollo Integrado (IDEs), lo que facilita su uso y adopción por parte de los desarrolladores [6].

**REFERENCIAS**

1. ▷ ¿Qué son las pruebas unitarias y cómo llevar una a cabo? (s.f.). Yeeply. <https://www.yeeply.com/blog/tendencias-habilidades/que-son-pruebas-unitarias/>
2. Procedimientos recomendados para escribir pruebas unitarias - .NET. (s.f.). Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/core/testing/unit-testing-best-practices>
3. LA IMPORTANCIA DE LAS PRUEBAS UNITARIAS PARA COMPROBAR FRAGMENTOS DE CÓDIGO - Cero ideas. (s.f.). Cero ideas. <https://ceroideas.es/la-importancia-de-las-pruebas-unitarias-para-comprobar-fragmentos-de-codigo/>
4. JUnit | Marco de Desarrollo de la Junta de Andalucía. (s.f.). Junta de Andalucía - Portal oficial. <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/248>
5. Ejemplo sencillo con JUnit. (s.f.). Ejemplos de Java y C/Linux. <https://old.chuidiang.org/java/herramientas/test-automaticos/ejemplo-junit.php>
6. Testing con JUnit 4 - Introducción a las pruebas automatizadas. (s.f.). Néstor Almeida. <https://www.nestoralmeida.com/testing-con-junit-4/>
7. 1. Qué es JUnit y cómo lo configuro. (2016, 8 de febrero). makigas. <https://www.makigas.es/series/junit/los-problemas-de-las-pruebas-manuales>