****

**PROGRAMACIÓN ORIENTADA A OBJETOS**

GRUPO #3

**INTEGRANTES:**

* Guamán Byron
* Mena Paul
* Mortensen Eduardo
* Portilla Diego
* Salazar Edwin

**NRC:**

* 3682

**NOMBRE DEL PROFESOR:**

* Lascano Edison

**¿Qué es Quick Start to Unit Test?**

La prueba unitaria de su código es una parte esencial del ciclo de vida del desarrollo de software. También forma la base para las pruebas de regresión cada vez que se agregan nuevas funcionalidades o se modifican funcionalidades existentes; Los casos de prueba unitaria aseguran que el sistema aún se comporte como se esperaba en términos de la funcionalidad anterior. La velocidad de prueba es un factor importante. Si el proceso de pasar la prueba es lento, no pasarán de manera constante y se perderán los beneficios que nos brindan. Tan bueno como el alcance de su código. El alcance del código indica la cantidad de código de aplicación que se probará. Por lo tanto, una cobertura baja significa que la mayor parte del código no se ha probado.

**Ventajas**

Las pruebas de unidad son fundamentales para el desarrollo de software, dándonos las siguientes ventajas:

* Aumenta la confianza en la confiabilidad de su base de código
* Refuerza la inversión del patrón de inyección de control / dependencia
* Empuja al desarrollador a escribir un buen código (el código incorrecto es a menudo difícil de probar e interpretar)
* Protege sus requisitos y evita que otros los rompan accidentalmente.

**Cómo hacer pruebas unitarias**

Para realizar pruebas unitarias, los desarrolladores escriben una sección de código para probar una función específica en la aplicación de software. Los desarrolladores también pueden aislar esta función para probar de manera más rigurosa, lo que revela dependencias innecesarias entre la función que se está probando y otras unidades para que las dependencias puedan eliminarse. Los desarrolladores generalmente usan el marco UnitTest para desarrollar casos de prueba automatizados para pruebas unitarias.

Las pruebas unitarias son de dos tipos:

* Manual
* Automatizado

Las pruebas unitarias suelen estar automatizadas, pero aún pueden realizarse manualmente. La Ingeniería de Software no favorece a uno sobre el otro, pero se prefiere la automatización. Un enfoque manual para las pruebas unitarias puede emplear un documento instructivo paso a paso.

Bajo el enfoque automatizado-

Un desarrollador escribe una sección de código en la aplicación solo para probar la función. Más tarde comentarán y finalmente eliminarían el código de prueba cuando se implemente la aplicación.

Un desarrollador también podría aislar la función para probarla de manera más rigurosa. Esta es una práctica de prueba unitaria más completa que implica copiar y pegar código en su propio entorno de prueba que en su entorno natural. Aislar el código ayuda a revelar dependencias innecesarias entre el código que se está probando y otras unidades o espacios de datos en el producto. Entonces, estas dependencias pueden eliminarse.

Un codificador generalmente usa un UnitTest Framework para desarrollar casos de prueba automatizados. Usando un marco de automatización, el desarrollador codifica criterios en la prueba para verificar la exactitud del código. Durante la ejecución de los casos de prueba, el marco registra los casos de prueba fallidos. Muchos marcos también marcarán e informarán automáticamente, en resumen, estos casos de prueba fallidos. Dependiendo de la gravedad de una falla, el marco puede detener las pruebas posteriores.

El flujo de trabajo de las pruebas unitarias es

1) Crear casos de prueba

2) Revisar / reelaborar

3) Línea de base

4) Ejecutar casos de prueba.

**Técnicas de prueba unitaria**

Las técnicas de prueba unitaria se clasifican principalmente en tres partes que son las pruebas de caja negra que implican la prueba de la interfaz de usuario junto con la entrada y la salida, las pruebas de caja blanca que implican probar el comportamiento funcional de la aplicación de software y las pruebas de caja gris que se utilizan para ejecutar la prueba suites, métodos de prueba, casos de prueba y realización de análisis de riesgo.

Las técnicas de cobertura de código utilizadas en las pruebas unitarias se enumeran a continuación:

• Cobertura de estados de cuenta

• Cobertura de decisiones

• Cobertura de sucursales

• Cobertura de condición

• Cobertura de máquina de estado finito

**¿Para qué se usa?**

Utilizamos las pruebas unitarias para probar el código bajo prueba por lo que la unidad de código puede ser un método y gracias a estas no habrá necesidad de una intervención manual y así podamos automatizar su ejecución. Estas pruebas deben ser capaces de repetirse las veces que quiera La velocidad de prueba es un factor importante. Si el proceso de pasar la prueba es lento, no pasarán de manera constante y se perderán los beneficios que nos brindan. Tan bueno como el alcance de su código. El alcance del código indica la cantidad de código de aplicación que se probará. Por lo tanto, una cobertura baja significa que la mayor parte del código no se ha probado.

Las pruebas unitarias deben poder ejecutarse independientemente del estado del entorno. La prueba debe aprobarse en todas las computadoras del equipo de desarrollo. La ejecución de una prueba no puede afectar la ejecución de otra. Después de ejecutar la prueba, el entorno debe permanecer igual que antes de la prueba. Para evitar dependencias entre módulos, debe simular diferentes relaciones que puedan existir entre módulos. Es importante tener una idea clara del propósito de la prueba. Cualquier desarrollador debe tener una idea clara de para qué sirven las pruebas y cómo funcionan. Esto solo se logra si el código de prueba se considera código de aplicación. Recuerde que si bien estos son buenos rasgos experimentales, no siempre es posible o necesario seguir todas estas reglas, y la experiencia será: Conducirnos al mismo logro.

**Pruebas unitarias, de integración y de rendimiento**

* Una prueba unitaria es un fragmento de código escrito por un desarrollador que ejecuta una funcionalidad específica en el código que se va a probar y afirma un determinado comportamiento o estado. El porcentaje de código que se prueba mediante pruebas unitarias generalmente se denomina cobertura de prueba. Una prueba unitaria tiene como objetivo una pequeña unidad de código, por ejemplo, un método o una clase. Las dependencias externas deben eliminarse de las pruebas unitarias, por ejemplo, reemplazando la dependencia con una implementación de prueba o un objeto (simulado) creado por un marco de prueba. Las pruebas unitarias no son adecuadas para probar interfaces de usuario complejas o interacción de componentes. Para ello, conviene desarrollar pruebas de integración.
* Una prueba de integración tiene como objetivo probar el comportamiento de un componente o la integración entre un conjunto de componentes. El término prueba funcional a veces se utiliza como sinónimo de prueba de integración. Las pruebas de integración verifican que todo el sistema funciona según lo previsto, por lo que están reduciendo la necesidad de pruebas manuales intensivas. Este tipo de pruebas le permiten traducir sus historias de usuario en un conjunto de pruebas. La prueba se parecería a la interacción esperada del usuario con la aplicación.
* Las pruebas de rendimiento se utilizan para comparar componentes de software repetidamente. Su propósito es garantizar que el código bajo prueba se ejecute lo suficientemente rápido incluso si está bajo una carga alta.

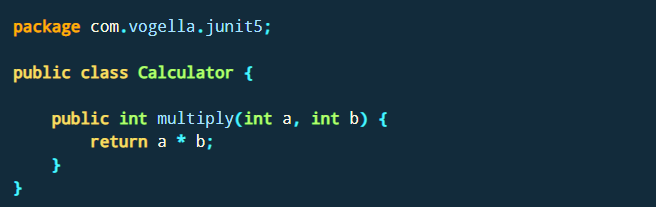
**¿Cómo definir una prueba en JUnit?**

Una prueba JUnit es un método contenido en una clase que solo se usa para probar.

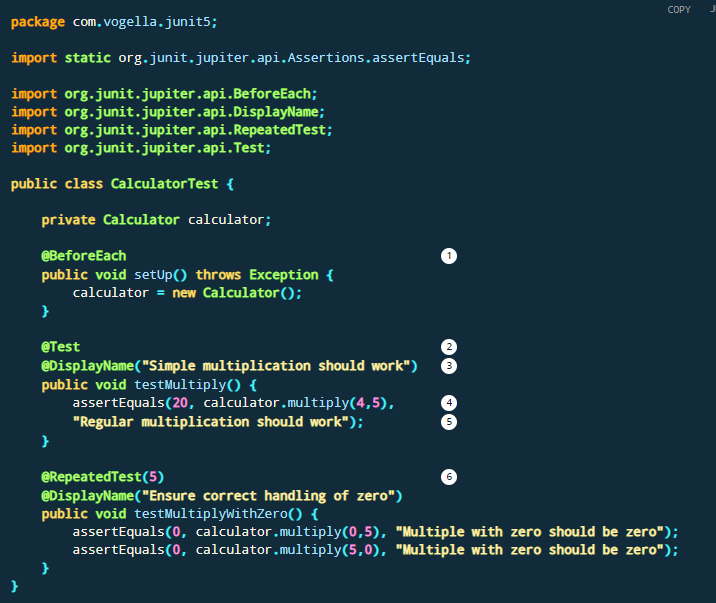
**Ejemplo de prueba JUnit 5**

El siguiente ejemplo demuestra el uso de una prueba JUnit.

Suponga que tiene la siguiente clase que desea probar.



Una clase de prueba para la clase anterior podría tener el siguiente aspecto.



1. Se ejecuta antes de cada prueba
2. Define un método de prueba
3. Nombre de la prueba que mostrará el corredor de pruebas
4. declaración de aserción que valida el valor esperado y real
5. mensaje de error en caso de que la prueba falle
6. prueba que se ejecuta varias veces, en este ejemplo 5 veces

**Anotaciones importantes de JUnit**

JUnit usa anotaciones para, marcar métodos como métodos de prueba y configuración. La siguiente tabla ofrece una descripción general de las anotaciones más importantes en JUnit 5.



**Bibliografía:**

* Studio, V., Edition, P., & Studio, V. (n.d.). *Pruebas unitarias*.
* **GmbH, V. V. L.-. (2007). *Unit Testing with JUnit 5 - Tutorial*. Lars Vogel.** [**https://www.vogella.com/tutorials/JUnit/article.html**](https://www.vogella.com/tutorials/JUnit/article.html)

**Questionnaire**

**¿Which one of the next options below is correct about the advantages of unit tests?**

1. Increase confidence in the reliability of your code base
2. It pushes the developer to write bad code
3. It weakens your requirements and prevents others from accidentally breaking them.

Answer: a

**¿True or False about the advantages of unit tests, It pushes the developer to write good code?**

1. True
2. False

Answer: a

**The unit testing workflow is:**

1) Create test cases

2) Review / rework

3) Baseline

4) Run test cases.

**Unit tests are of two types:**

1) Handbook

2) Automated

**Write 2 advantages over unit tests**

1. Increase confidence in the reliability of your code base
2. Reinforces the inversion of the control / dependency injection pattern

**Indicate which of these is not a unit test characteristic**

1. It must be able to be executed without the need for manual intervention
2. They have to be able to be repeated as many times as you want.
3. Problems and time spent on integration are drastically reduced
4. The execution of one test cannot affect the execution of another

**¿What is the objective of the unit integration test?**

It aims to test the behavior of a component or the integration between a set of components.

**¿What tools are used to carry out unit tests?**

* xUnit
* Junit
* NUnit
* PHPUnit

**¿Name three code coverage techniques used in unit tests?**

• Coverage of account statements

• Coverage of decisions

• Branch coverage

**¿Name two advantages of unit tests?**

* Aumenta la confianza en la confiabilidad de su base de código
* Refuerza la inversión del patrón de inyección de control / dependencia