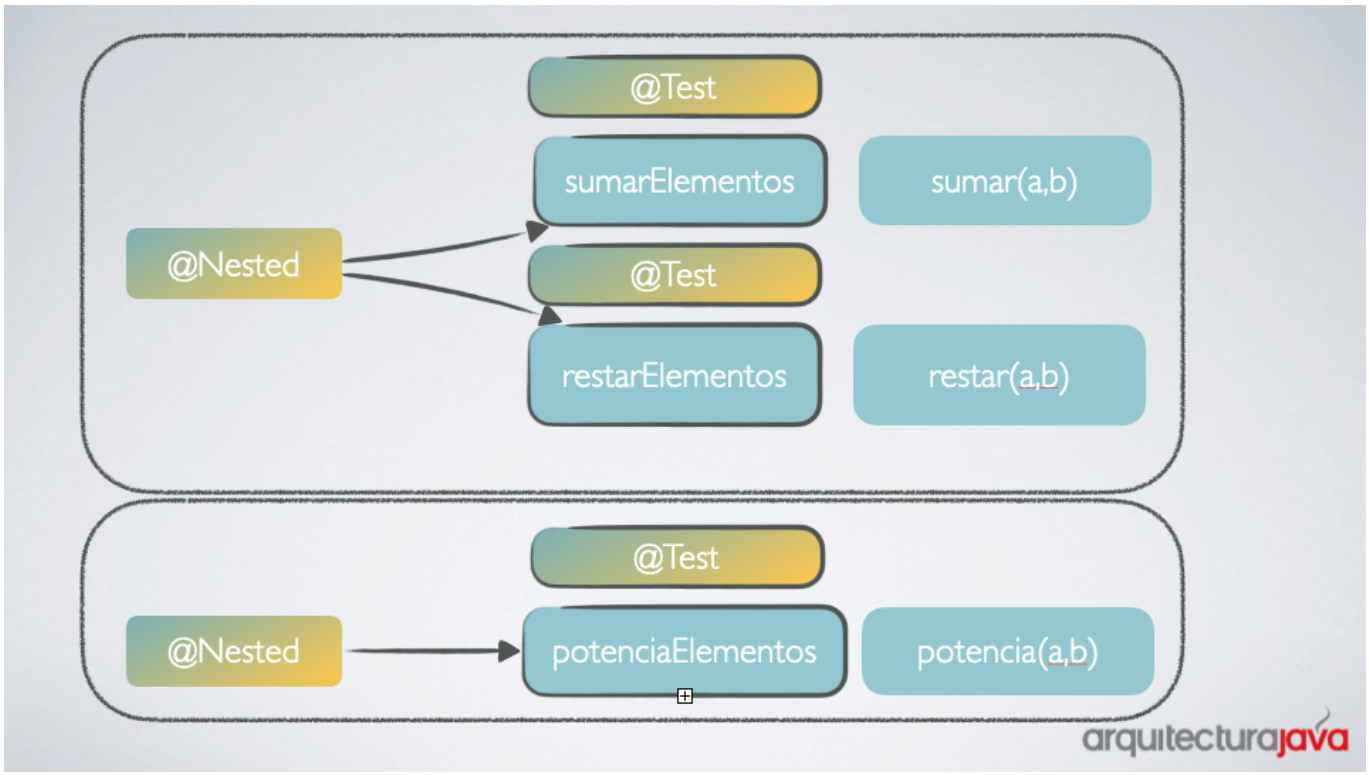
Bienvenido al nuevo Curso de Junit 5 en el cual abordaremos las características nuevas más importantes de este framework.



1. Fundamentos de JUnit 5: Desde la configuración inicial hasta la estructura básica de una prueba

En esta sección, aprenderás los conceptos clave para comenzar con JUnit 5, el framework más utilizado para escribir pruebas unitarias en Java. Exploraremos:

- Configuración inicial de JUnit 5: Cómo integrar JUnit 5 en tu proyecto, tanto si estás utilizando Maven como Gradle. Veremos cómo configurar las dependencias necesarias y cómo asegurarte de que el entorno esté correctamente preparado para ejecutar pruebas.
- Estructura básica de una prueba en JUnit 5: Revisaremos la anatomía de una prueba unitaria, desde la creación de los métodos de prueba hasta el uso de anotaciones. Compararemos cómo JUnit 5 introduce un enfoque más modular y flexible en comparación

con versiones anteriores como JUnit 4.

- Nuevas anotaciones de JUnit 5:
  - @DisplayName: Cómo utilizar esta anotación para mejorar la legibilidad de los nombres de tus pruebas, haciendo que los resultados sean más claros y comprensibles.
  - @Nested: Veremos cómo agrupar pruebas relacionadas lógicamente utilizando clases anidadas, lo que permite una mejor organización y encapsulación de los test cases.
  - @Disabled: Esta anotación nos permite desactivar temporalmente pruebas, ya sea porque están en desarrollo o debido a que hay características aún no implementadas.

2. Pruebas Parametrizadas y Repetitivas

JUnit 5 introduce nuevas formas de ejecutar pruebas, que son particularmente útiles para evitar la duplicación de código y mejorar la cobertura de casos de prueba:

- Pruebas Parametrizadas: Exploraremos cómo configurar pruebas que puedan ser ejecutadas con múltiples conjuntos de datos. Esto es ideal para verificar cómo un mismo método o funcionalidad se comporta bajo diferentes condiciones. Aprenderás a utilizar anotaciones como @ParameterizedTest y fuentes de parámetros como @ValueSource.
- Pruebas Repetitivas: JUnit 5 también permite la creación de pruebas que se ejecutan varias veces de forma automática mediante la anotación @RepeatedTest. Esta característica es útil para asegurarse de que el comportamiento de un método es consistente a través de múltiples ejecuciones, detectando posibles problemas intermitentes.

3. Assertions Complejos

JUnit 5 amplía el conjunto de *Assertions* disponibles, permitiendo la creación de pruebas más precisas y detalladas:

- Mejora en el uso de Assertions: Revisaremos los assertions tradicionales como assertEquals, assertTrue y assertFalse, pero también introduciremos nuevas opciones como assertAll, que te permite agrupar varias validaciones dentro de un mismo bloque de prueba, garantizando que todas se ejecuten antes de reportar fallos.
- Assertions con mensajes personalizados: Una característica poderosa que JUnit 5 introduce

es la capacidad de incluir mensajes más detallados cuando las pruebas fallan, lo que facilita identificar problemas sin necesidad de revisar el código de prueba. También exploraremos cómo usar lambdas en las afirmaciones, mejorando el rendimiento cuando las fallas no son inmediatas.
- Assertions para excepciones y temporales: Veremos cómo manejar situaciones en las que esperamos que una operación lance una excepción mediante el uso de assertThrows, así como también cómo validar tiempos de ejecución con assertTimeout.

4. Organización de las Pruebas y Construcción de Tests Dinámicos

JUnit 5 facilita la organización y estructura de tus pruebas, haciéndolas más fáciles de mantener y escalar:

- Estrategias para organizar las pruebas: Además del uso de clases anidadas, veremos cómo podemos usar etiquetas (`@Tag`) para agrupar pruebas por categorías, permitiendo ejecutar solo subconjuntos de pruebas específicos (por ejemplo, pruebas rápidas, pruebas de integración, etc.).
- Construcción de pruebas dinámicas: JUnit 5 introduce el concepto de tests dinámicos mediante DynamicTest. Con esta funcionalidad, es posible generar pruebas de manera programática durante la ejecución, lo que es especialmente útil para pruebas que dependen de datos que no están disponibles en tiempo de compilación o para casos en los que el conjunto de pruebas no es fijo.