Introducción a las Aserciones en Java

Las **aserciones** (`assert`) en Java son una herramienta útil para verificar que ciertas condiciones se cumplen durante la ejecución del programa. Se utilizan principalmente como una ayuda para la depuración y prueba de código, permitiendo detectar errores en tiempo de ejecución.

Una aserción en Java se define con la palabra clave `assert`, seguida de una **expresión booleana**. Si la expresión evalúa a `false`, el programa lanza un `AssertionError`.

int edad = -5;

assert edad >= 0 : "La edad no puede ser negativa";

La sintaxis básica de assert en Java es la siguiente:

assert condición : "Mensaje de error opcional";

Las aserciones en Java son útiles principalmente durante el desarrollo y las pruebas, pero no deben ser usadas para validar condiciones de entrada o errores en producción. En producción, se prefieren métodos de validación más robustos, como lanzar excepciones personalizadas o usar herramientas de validación específicas.

Por defecto, las aserciones están **deshabilitadas** en Java. Para habilitarlas, debes ejecutar el programa con la opción `-ea` (Enable Assertions):

// desde la consola cmd

java -ea AsercionesEjemplo.java

Método 2: Configurar `launch.json` en VS Code

Si usas **depuración en VS Code**, puedes habilitar las aserciones en el archivo `launch.json`:

1. Ve a **Ejecutar y depurar** (`Run & Debug` en inglés).
2. Haz clic en **Crear un archivo `launch.json`** si no tienes uno.
3. Agrega o edita la configuración para incluir la opción `-ea`:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "java",
      "request": "launch",
      "name": "Ejecutar con aserciones",
      "mainClass": "AsercionesEjemplo",
      "vmArgs": "-ea"
    }
  ]
}
```

Diferencia con JUnit y otras herramientas

- **`assert` de Java**: Es más simple y se utiliza principalmente para comprobar condiciones internas durante el desarrollo. No proporciona muchos detalles adicionales, como lo haría un marco de pruebas.
- **JUnit**: Ofrece un conjunto mucho más amplio de métodos de aserción (como `assertEquals`, `assertTrue`, `assertThrows`, etc.), y se utiliza para pruebas unitarias más formales, donde se necesita realizar una verificación más estructurada y detallada.

1. Aserciones de igualdad:

- **assertEqual(actual, expected)**: Verifica que dos valores sean iguales.
- **assertNotEqual(actual, expected)**: Verifica que dos valores no sean iguales.
- **assertSame(actual, expected)**: Verifica que dos referencias de objeto sean la misma (en algunas bibliotecas).
- **assertNotSame(actual, expected)**: Verifica que dos referencias de objeto no sean la misma.

2. Aserciones de identidad:

- **assertTrue(expression)**: Verifica que la expresión dada sea `true`.
- **assertFalse(expression)**: Verifica que la expresión dada sea `false`.
- **assertNull(value)**: Verifica que el valor sea `null`.
- **assertNotNull(value)**: Verifica que el valor no sea `null`.

3. Aserciones de tipo:

- **assertIsInstance(obj, class)**: Verifica que el objeto sea una instancia de la clase dada.
- **assertNotIsInstance(obj, class)**: Verifica que el objeto no sea una instancia de la clase dada.

4. Aserciones de comparación:

- **assertGreater(a, b)**: Verifica que `a` sea mayor que `b`.
- **assertGreaterEqual(a, b)**: Verifica que `a` sea mayor o igual que `b`.
- **assertLess(a, b)**: Verifica que `a` sea menor que `b`.
- **assertLessEqual(a, b)**: Verifica que `a` sea menor o igual que `b`.

5. Aserciones de contenido:

- **assertIn(item, collection)**: Verifica que el ítem esté presente en la colección.
- **assertNotIn(item, collection)**: Verifica que el ítem no esté presente en la colección.
- **assertIsSubset(subset, superset)**: Verifica que un conjunto sea un subconjunto de otro.
- **assertIsSuperset(superset, subset)**: Verifica que un conjunto sea un superconjunto de otro.

6. Aserciones de excepción:

- **assertRaises(exception, callable, *args, **kwargs)**: Verifica que una excepción sea lanzada cuando se llama a una función con los argumentos dados.

7. Aserciones de exactitud:

- **assertAlmostEqual(a, b)**: Verifica que dos números sean casi iguales (con un margen de error pequeño, útil para números con decimales).
- **assertNotAlmostEqual(a, b)**: Verifica que dos números no sean casi iguales.

8. Aserciones de longitud:

- **assertLength(obj, length)**: Verifica que un objeto (como una lista o cadena) tenga la longitud esperada.