



La palabra clave assert de Java y un ejemplo

Escrito por picodotdev el 13/02/2015, actualizado el 14/02/2015.

[blog-stack](#) [java](#) [planeta-codigo](#) [programacion](#)

[Enlace permanente](#) [Comentarios](#)

La palabra clave o reservada `assert` ^{afirmar} sirve para aseverar que en un **determinado momento del código una determinada condición debe ser cierta**. Está disponible en [Java](#) desde la versión 1.4 pero al menos yo con bastantes años de experiencia en programación en este lenguaje aún no he usado de forma amplia y posiblemente le pase a mucha de la gente y aún así hemos sobrevivido durante todo este tiempo.

Sin embargo, puede resultarnos bastante útil. Una de las situaciones en que puede ayudarnos es **para descubrir una condición no válida en el momento del `assert` y no donde se produce una excepción en otro punto del código que puede no ser la causa real del error**. Por ejemplo, supongamos que un **método privado no acepta un parámetro con valor `null`, una variable no puede ser `null` o una colección no ha de estar vacía** por poner solo unos pocos ejemplos de condiciones, si **en un punto del código estamos seguros que es un error que esa condición sea falsa podemos hacer que el programa falle con una excepción ahí y no más tarde a consecuencia de que las condiciones no se cumplieran**. Otra forma en la que nos ayudan los `assert` es como documentación, en vez de poner un comentario o en el javadoc indicando una **condición que se ha de cumplir podemos ponerlo con un `assert`**. Normalmente se usan en:



- **Precondiciones**: en **métodos privados que el llamador ha de cumplir**.
- **Postcondiciones**: para **verificar el resultado prometido por el método**.
- **Class invariants**: para **validar el estado de una clase según está definido en su contrato, siempre se debe cumplir independientemente de las operaciones que se realicen**.
- **Código no alcanzable en tiempo de ejecución**: **partes del programa que se espera que no sea alcanzable, como cláusulas `else` o `default` en sentencias `switch`**.

Y no deben usarse para:

- No se deben usar para **comprobar argumentos en métodos públicos**: los `asserts` pueden habilitarse o deshabilitarse, comprobar los argumentos se considera parte de las responsabilidades del método y su especificación.
- No se deben usar para **realizar tareas**: ya que los `asserts` pueden deshabilitarse las tareas dejarían de ejecutarse y de proporcionar la funcionalidad del programa.

Nos pueden entrar dudas de cuando emplear un `assert` y cuando un `if` o una excepción. Las excepciones se encargan de hacer que el programa sea robusto controlando las situaciones inesperadas pero posibles, los `assert` se encargan de que el programa sea correcto. Los `assert` deberían ser usados para **asegurar algo, mientras que las excepciones deberían usarse para comprobar algo que podría ocurrir**. Un `assert` termina la ejecución (ya que no se suele capturar la excepción que se produce) mientras que una excepción permite al programa continuar

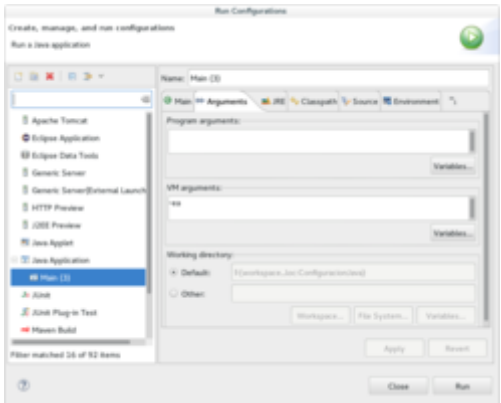
con la ejecución. Los **asserts no deben ser sustitutos** de **condiciones de validación** que **debería hacer** el **programa** en **métodos públicos** de una **clase**. Los **assert** son una **herramienta** en **tiempo de desarrollo**, las **excepciones** además son una **herramienta** para la **ejecución** en **producción**.

Un pequeño ejemplo de los *asserts* podría ser el siguiente en la que en el método *nextNumber* hay una postcondición según la cual el método debe devolver un número entero entre 0 y 9 (incluidos):

```
1  package io.github.picodotdev.blogbitix.asserts;
2
3  import java.util.Random;
4
5  public class Main {
6
7      private Random random;
8
9      public Main() {
10         random = new Random();
11     }
12
13     /**
14      * Devuelve un número entero entre 0 y 9.
15      */
16     public int nextNumber() {
17         int i = random.nextInt(40);
18
19         // Si el cálculo del número fuese más complejo incluyendo un assert
20         // podemos asegurar en tiempo de desarrollo en esta postcondición
21         // el valor generado por este método.
22         // La línea de código anterior según el contrato del método debería ser:
23         // int i = random.nextInt(10);
24         assert i >= 0 && i < 10: String.format("El número devuelto no cumple la postcondición (%d)", i);
25
26         return i;
27     }
28
29     public static void main(String[] args) {
30         Main main = new Main();
31         System.out.println(main.nextNumber());
32     }
33 }
```

Main.java hosted with ❤ by GitHub view raw

Un **assert** cuya **expresión** se **evalúa** como **falso produce** una **excepción** del tipo `java.lang.AssertionError` pero **para ello** se **han** de **habilitar** en **tiempo de ejecución** como el **parámetro -ea** de la **máquina virtual**. En **eclipse** podemos cambiarlo en la configuración de ejecución del programa en la pestaña *Arguments* y *VM arguments* tal como se ven en la siguiente captura de pantalla:



En la primera de las siguientes capturas de pantalla puede verse como el programa se ejecuta sin producir una excepción a pesar de no cumplirse el *assert* del método *nextNumber* ya que los *asserts* no fueron activados, en la segunda captura activando los *assert* se lanza una excepción al no cumplirse la postcondición.



En el recomendable artículo [Programming With Assertions](#) se comenta de forma más detallada y amplia el funcionamiento y uso adecuado de la palabra clave *assert* de Java.

El funcionamiento de los *assert* en [Groovy](#) es distinto. En groovy los *assert* no pueden deshabilitarse, están siempre habilitados y por tanto no hace falta usar el parámetro *-ea* de la máquina virtual que empleamos en Java, no es un *bug* es una *feature*. Por el contrario, en Java los *asserts* se consideran una herramienta en tiempo de desarrollo o depuración y por tanto podemos habilitarlos mientras desarrollamos y no habilitarlos en producción, una de las razones es que los *asserts* pueden suponer una penalización de rendimiento si las comprobaciones son costosas en tiempo o carga de CPU cosa que no queremos en producción donde el código ya se considera correcto.

Referencia:

- [Programming With Assertions](#)
- [Correct use Java “assert” keyword](#)
- [When to use an assertion and when to use an exception](#)
- [Is Groovy’s assert a good idea for production code, unlike Java’s assert?](#)
- [Java: Should I assert or throw AssertionError?](#)

Artículos relacionados:

- [Convertir fechas y husos horarios en Java](#)
- [Cómo hacer un substring de una cadena HTML](#)
- [Ejercicios \(katas\) para mejorar habilidades de programación practicando](#)
- [8+ libros para mejorar como programadores](#)
- [Cómo filtrar contenido HTML de forma segura](#)

115
Shares

Tweet

Share

Share

Share

1 Comentario

Blog Bitix

 Acceder ▾

 Recomendar

 Tweet

 Compartir

Ordenar por los más antiguos ▾



Únete a la conversación...

INICIAR SESIÓN CON



O REGISTRARSE CON DISQUS 

Nombre



hernan • hace 5 meses

no entendi niuna wea

 |  • Responder • Compartir ›

TAMBIÉN EN BLOG BITIX

Películas sobre tecnología o informática, series, documentales, vídeos, libros, ...

2 comentarios • hace 10 meses



picodotdev — Por supuesto, gracias por el aporte. Me veré la el documental que indicas :), un saludo.

Introducción al JavaScript de ECMAScript 6

3 comentarios • hace 2 años



gerardo pacheco — muchas gracias!! saludos!!

Sustituir caldera de gas por un termo eléctrico Fleck Duo 7 50

6 comentarios • hace un año



picodotdev — Hola Rafael, el termo queda bastante pegado a la pared, si el enchufe lo tienes justo detrás con 3cm creo que te va a ser un ...

Tú con tu Mac, yo con mi GNU/Linux

7 comentarios • hace 7 meses



picodotdev — Era este <https://elblogdepicodev.blo...> y lo compre directamente online en la página de Sony pero ...

 Suscríbete

 Añade Disqus a tu sitio webAñade Disqus Añadir

 Política de privacidad de DisqusPolítica de privacidadPrivacidad

Parece que tienes activado un
bloqueador de anuncios