**Academia Java – JAVA**

***Problema de Números***

Tabla de contenido

NÚMEROS Y ESTÁTICOS 3

1. Métodos matemáticos: lo más cerca de métodos globales 4

1.1. Diferencias entre los métodos estáticos y no estáticos 4

1.2. ¿Qué significa tener una clase con métodos estáticos? 6

# NÚMEROS Y ESTÁTICOS

Hacer los cálculos.

Cuando los hacemos siempre hay por hacer que simple operaciones aritméticas, no? A lo mejor requerimos valores absolutos, redondear un número, o encontrar el mayor de dos números. Puedes querer imprimirlos a dos decimales, o usar comas en números grandes para que sean más fáciles de leer.

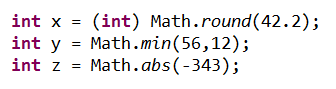
¿Y el uso de fechas? La cantidad de formatos que usas para imprimir fechas es variado, o incluso manipulas las fechas para cosas como, "añadir tres semanas a la fecha de hoy".

Y ¿cuándo quieres convertir una cadena en un número? ¿O convertir un número en una cadena? Hay suerte. La API de Java está llena de herramientas que nos brindan métodos que ayudan a trabajar fácilmente con los números y son faciles de usar. Sin embargo, la mayoría de ellos son estáticos, por lo que vamos a empezar por aprender lo que significa para una variable o un método ser estáticos, incluyendo las constantes en Java –variables estáticas finales.

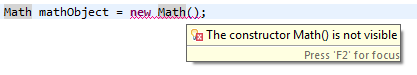
# Métodos matemáticos: lo más cerca de métodos globales

Excepto que no hay *nada* global en Java. Pero piensa en esto: qué pasaria si tienes un método con comportamiento que no depende del valor de una variable de instancia. Por ejemplo, el método round() de la clase Math, siempre hace lo mismo: redondea un número de punto flotandote (el argumento del método) al entero más cercano. Si tuvieras muchas instancias de la clase Math, y ejecutas round(345.66), obtendrás el valor de 42. Cada vez. En otras palabras, los métodos actuan de acuerdo a los argumentos y no se afectan por el estado de la instancia de la variable. El único valor que cambia la forma como el método round() cambia es el argumento que se pasa.

De esta forma, los métodos de la clase Math no usan valores de variables de instancia. Y esto se debe a que son “estáticos”, entonces no necesitas tener una instancia de Math. Todo lo que requieres es la clase Math.



Es más! Si intentas instanciar la clase, mira lo que sucede:



# Diferencias entre los métodos estáticos y no estáticos

Java es orientado a objetos, pero a veces te das con casos donde necesitas generar métodos utilitarios (como los métodos de la clase Math), donde no necesitas una instancia de una clase. La palabra clave **static** permite a un método ejecutarse sin instanciar la clase a la cual pertenece. Un método estático significa “el comportamiento no depende de una variable de instancia, entonces no es necesario un objeto instanciado. Solo la clase”.

|  |  |
| --- | --- |
| **Métodos no estáticos** | **Métodos estáticos** |
|  |  |

# ¿Qué significa tener una clase con métodos estáticos?

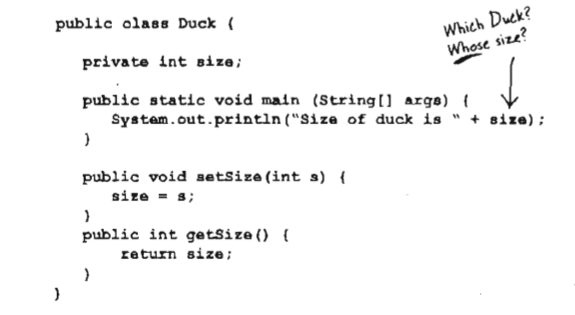
Frecuentemente, pero no siempre, una clase con métodos estáticos no es inicializada.

Si recuerdas, en los capítulos anteriores, vimos como el modificador abstract en una clase, hacía imposible que esta pudiera ser instanciada.

Pero puedes usar también otra palabra reservada para restringir instanciar una clase y es marcar el constructor con prívate. Recuerda, que un método marcada con private significa que solo el código de misma clase podrá acceder a él. Un método marcado igualmente con prívate es esencialmente lo mismo. Nadie podrá decir new a tu clase fuera de ella. Así es como trabaja la clase Math, por ejemplo. El constructor es privado, por ello no puedes instanciarla. El compilador sabe que no puedes acceder de esta forma a su constructor privado.

Esto no significa que una clase con uno o más métodos estáticos nunca debería ser instanciada. De hecho, cada clase que tiene un método main y que usamos para probar o testear clases tiene dicho método estático. Asi que somos libres para combinar static y métodos no static en una clase.

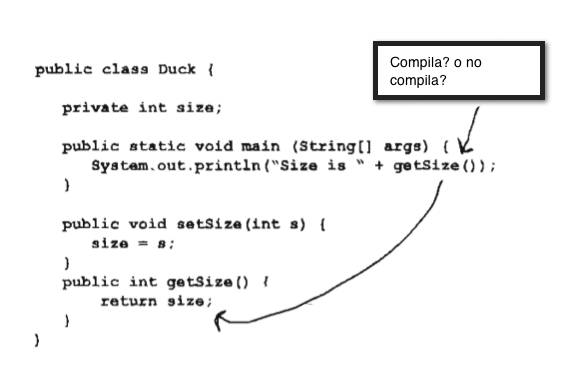
# Métodos static no pueden usar variables no estáticas



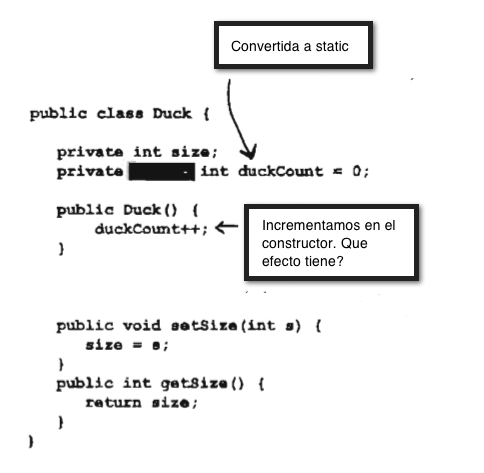
COMPILA o NO COMPILA???

**Si probamos o intentamos usar una variable de instancia dentro de un método estático, el compilador nos dice “como saben las variables de instancia de lo que estas hablando?” Porque?? Puesto que si tienes diez objetos Duck en el heap, un método estático no sabe nada, a cerca de, ellos.**

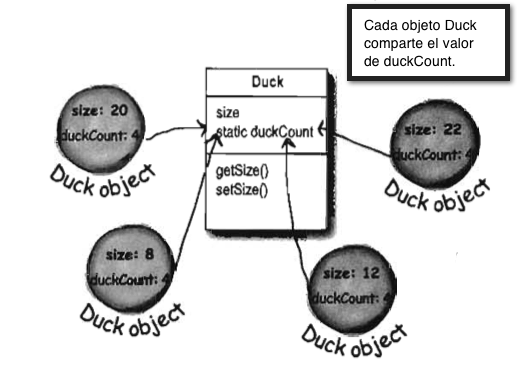
# Métodos static no pueden usar métodos no estáticos tampoco.



# Variable STATIC: El valor es igual para todas las variables de instancia



El efecto en todas las variables de instancia es:



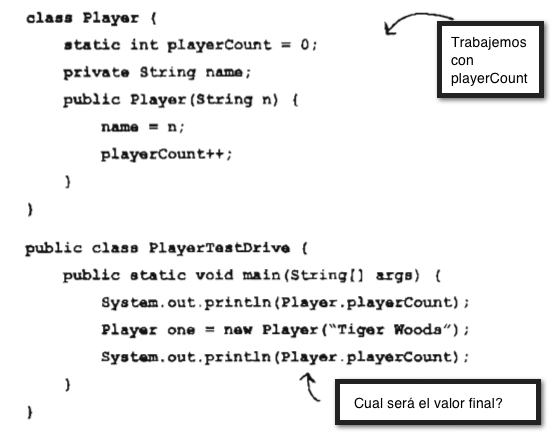
**Las variables static son compartidas.**

**Todas las instancias de la misma clase comparten una simple copia de las variables static.**

****

# Inicializando una variable static

**TODAS LAS VARIABLES STATIC EN UNA CLASE SON INICIALIZADAS ANTES DE QUE CUALQUIER OBJETO DE ESA CLASE SEA CREADA.**



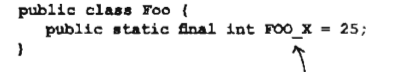
# Las variables static final son constantes

Toda variable marcada como final significa que es una vez inicializada, esta no puede ser cambiada. En otras palabras el valor de un static final variable permanece igual a lo largo de la vida de la clase. Si observamos la clase Math del Java API, veremos que tiene

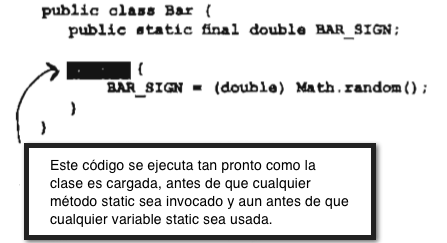
**public static final doublé PI = 3.14159;**

**Para inicializar una variable static final**

Se puede hacer en el momento de la declaración de esta:



En un static initializer:

****

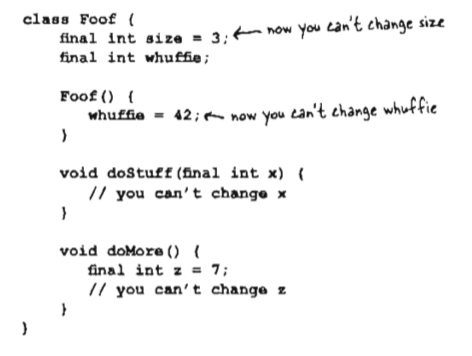
# final no es solo para variables static

**Una variable final significa que no le podemos cambiar su valor.**

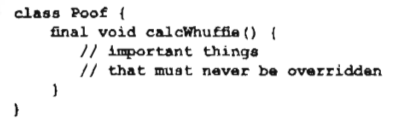
**Un método final significa que no podemos sobrescribir dicho método.**

**Una clase marcada como final, significa que no podemos heredar o hacer una subclase a partir de ella.**

**Ejemplo: Variables final no static**

****

**Método final:**

****

**Clase Final**

****

# Clase Math

Ver Math.random()

Ver Math.abs()

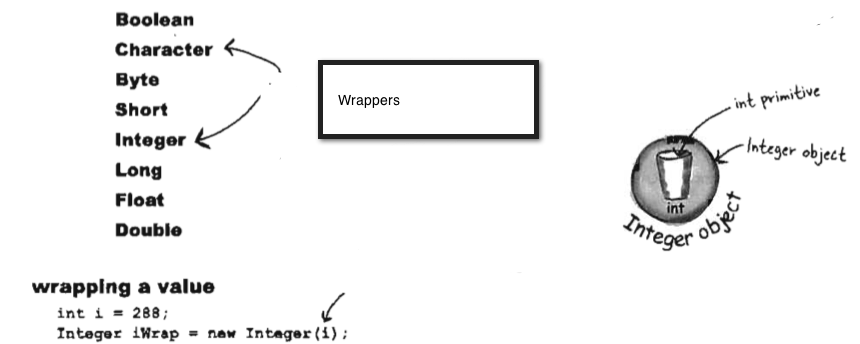
Ver Math.round()

Ver Math.min()

Ver Math.max()

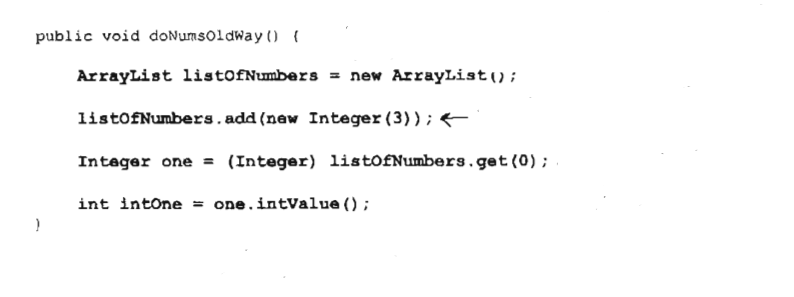
# Wrappers

**Cuando necesitas tratar a un primitivo como objeto, utilizar el wrapper para este.**

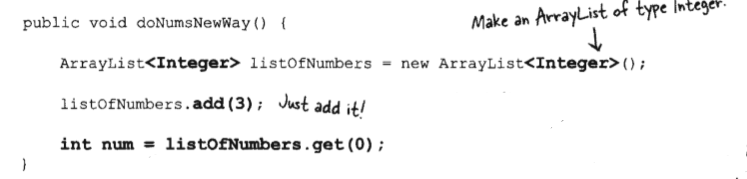
****

**Antes de Java 5.0 teníamos que hacer todo el trabajo**

**Sin autoboxing**

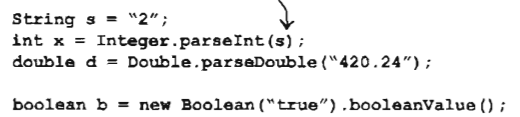
****

**Con autoboxing**

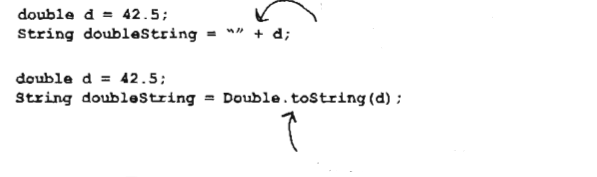
****

# Los Wrappers tienen métodos static útiles

**Convertir String a primitivo**

****

**Primitivo a String**

****

# Formateo de números

