

Tema 3. Variables: definición, tipos y uso en Java

La clase Math y las “Constantes Java”

- La clase `Math`: detalles y ejemplos de uso; problemas de Redondeo y Números Aleatorios
- “Constantes Java”: uso, definición y ejemplos

La clase `java.lang.Math`

Detalles (I): características relevantes

- Referencia: [Documentación de la clase Math - API de Java](#)
- Clase de Utilidades del estándar de Java. Proporciona...

Constantes	
Uso	Significado
<code>Math.E</code>	$e = 2.7182818284590452354$
<code>Math.PI</code>	$\pi = 3.14159265358979323846$

(ver apartado 5.2 del libro)

Métodos	
Invocación	Significado
<code>Math.abs(x)</code>	valor absoluto de x
<code>Math.max(x, y)</code>	mayor de x e y
<code>Math.min(x, y)</code>	menor de x e y
<code>Math.pow(x, y)</code>	x^y
<code>Math.log(x)</code>	$\ln(x)$
<code>Math.round(x)</code>	Redondeo al entero más próximo

NOTA: usa sus métodos anteponiendo a su identificador `Math.` (¿Y eso?)

La clase java.lang.Math

Detalles (II): métodos y ejemplos relevantes



BlueJ: ejercicios - Tema 3

```
double x = 2.0, y = 5.0;

// Exponenciales - logarítmicos:
double pot = Math.pow(x, y);           // pot = 2.05.0 = 32.0
double raiz = Math.sqrt(x);            // raiz = 1.4142135623730951
double ln = Math.log(y);               // ln = 1.6094379124341003

// Trigonómicas
double sin = Math.sin(Math.PI / 2);    // sin es 1.0
double alf = Math.arcsin(sin);         // alf = 1.5707963267948966
double tan = Math.tan(Math.PI / 2);    // tan = 1.633123935319537E16

// Matemáticas básicas:
double abs = Math.abs(-x);             // abs = 2.0
double max = Math.max(x, y);           // max = 5.0
double ceil = Math.ceil(3.76);         // ceil = 4.0
double flr = Math.floor(3.76);         // flr = 3.0
long round1 = Math.round(3.76);        // round1 = 4L
long round2 = Math.round(3.45);        // round2 = 3L
```

La clase `java.lang.Math` y el problema del Redondeo



Redondeo (clave CCDGH4ai)

Haciendo uso de la clase `Math`, completa el método `redondearA` de la Clase (de Utilidades) `miLibreria`, que redondea un `double x` a `d` cifras decimales

PISTA: ¿cómo **redondear** a **2** cifras decimales el valor real (`double`) **34.86842105263158**?

1. Usando `Math.pow`, multiplicar por **100** el valor, pues queremos redondear a **2** cifras decimales: **3486.842105263158**
2. Usando `Math.round`, redondear el valor obtenido en el **punto 1** para eliminar los decimales no deseados: **3487**, pues el primer decimal del valor obtenido en dicho punto es MAYOR O IGUAL QUE 5
3. Dividir por **100.0** el valor obtenido en el **punto 2** para obtener el resultado deseado: **34.87**

OJO: `Math.round` devuelve un `long` → Se divide por **100.0** para obtener el nº real **34.87** y no el cociente de la división entera **3487 / 100** (que es **34**)

Usa esta pista para mejorar la forma en la que se muestran los resultados los ejercicios del Capítulo 3 que figuran a continuación

La clase java.lang.Math y el problema de los **Números Aleatorios**



Aleatorio en intervalo (clave CCDGI4ai)

Haciendo uso de la clase Math, completa los dos métodos de la Clase (de Utilidades) **MiLibreria** que obtienen un valor aleatorio en un intervalo dado de, respectivamente, tipo `double` e `int`

PISTA: ¿cómo generar **aleatoriamente** un entero en el intervalo **[1, 6]** ?

1. `Math.random()` devuelve un valor real aleatorio en el intervalo **[0.0, 1.0[**
2. Para obtener un valor **entero** en el intervalo **[1, 7[**
 - a) “Manipular” el intervalo **[0.0, 1.0[** para obtener un valor **real** en **[1.0, 7.0[**
 - b) Quedarse con la **parte entera** del valor real obtenido (**truncar con (int)**)

```
x ∈ [0.0, 1.0[ ⇒ x = x * 6 + 1;  
                ⇒ x ∈ [1.0, 7.0[  
                ⇒ x ∈ [1.0, ..., 1.999...9, 2.0, ..., 2.999...9, ..., 6.0, ..., 6.999...9]  
                ⇒ (int) x ∈ [ 1, 2, ..., 6 ]
```

La clase java.lang.Math

Ejercicio propuesto: **Ejercicio nº 5 del Capítulo 3**



BlueJ: ejercicios - Tema 3

Escribe un programa Java **Ejercicio5C3** que, dada una temperatura de **31** grados celsius calcule a cuántos grados fahrenheit equivale y lo muestre por pantalla. La fórmula a utilizar es:

$$^{\circ}\text{F} = (9/5) * ^{\circ}\text{C} + 32.$$

Una vez comprobado su correcto funcionamiento, añade al programa las instrucciones necesarias para que **redondee a centésimas los valores obtenidos y los muestre por pantalla**

Modificadores `static` y `final`

¿Variables inmutables o constantes (!): modificador `final` cuyo valor **NO** puede cambiar?

¿Y si quieres usar **nemotécnicos** en vez de “**números mágicos**”?

Por ejemplo: quieres usar `PI_APROX` en vez del “número mágico” `3.14` porque vas a hacer un gran nº de cálculos donde aparece este valor y, claro, es más fácil y más seguro recordar `PI_APROX` que `3.14`

```
> double PI_APROX = 3.14;
> double radio = 12.0;
> double area = PI_APROX * radio * radio;
> area
➔ 452.15999999999997 (double)
> PI_APROX = 3.1416;
> area = PI_APROX * radio * radio;
> area
➔ 452.3904 (double)
> System.out.println("Houston, tenemos un problema");
```

variable

```
> final double PI_APROX = 3.14
> double radio = 12.0;
> double area = PI_APROX * radio * radio;
> area
➔ 452.15999999999997 (double)
> PI_APROX = 3.1416;
Error: cannot assign a value to final variable PI_APROX
> area
➔ 452.15999999999997 (double)
```

constante

Modificadores `static` y `final`

Variables inmutables o constantes (II): atributos `final`

También se puede definir como `final` un **Atributo**

Por ejemplo: usar `PI_APROX` en todos los métodos de la clase `Círculo` donde aparezca el “número mágico” `3.14` (area y perimetro)

```
public class Círculo {  
    private double radio;  
    private String color;  
    private int centroX, centroY;  
    private final double PI_APROX = 3.14;  
    // Métodos de la clase:  
    ...  
    public double area() { return PI_APROX * radio * radio; }  
    public double perimetro() { return 2 * PI_APROX * radio; }  
    ...  
}
```

Atributos: tipo *variable de instancia*

¿Tiene sentido que
una variable de instancia
sea **INMUTABLE**?

NO

Modificadores static y final

Variables de clase constantes o “Constantes Java”: atributos static final

También se puede definir como final un **Atributo**

Por ejemplo: usar **PI_APROX** en todos los métodos de la clase **Círculo** donde aparezca el “número mágico” **3.14** (area y perimetro)

```
public class Círculo {  
    private double radio;  
    private String color;  
    private int centroX, centroY;  
  
    public static final double PI_APROX = 3.14;  
  
    // Métodos de la clase:  
    public double area() { return PI_APROX * radio * radio; }  
    public double perimetro() { return 2 * PI_APROX * radio; }  
    ...  
}
```

Atributos tipo *variable de instancia*

Atributos tipo *variable de clase*

BlueJ: ejercicios - Tema 3

Abre la clase **Círculo** del proyecto para **ver qué “Constantes Java”** se han definido **y el orden** en el que lo hacen con respecto a las variables de instancia siguiendo las convenciones de código Java

Observa también **cómo se usan** dentro de los métodos de la clase, evitando los “números mágicos”

Modificadores static y final

Variables de clase constantes o “Constantes Java”: atributos static final



BlueJ: ejercicios – Tema 3

- **Limpia** el *Object Bench* del proyecto
- **Escribe en el Code Pad** del proyecto cada una de las expresiones que aparecen en el recuadro anterior **¿A qué se evalúan?**

Observa cómo se usa una “Constante Java” desde fuera de la clase *Círculo*, donde se ha definido:

nombreDeLaClase.nombreDeLaConstanteJava



Círculo.PI_APROX

Círculo.RADIO_STD

Círculo.COLOR_STD

Círculo.CENTRO_X_STD

Círculo.CENTRO_Y_STD

new Círculo(Círculo.RADIO_STD, “amarillo”, 20, 30)



Ejercicio nº 7, Capítulo 3: Test Constantes (clave CCDGL4ai)

Completa los huecos que figuran a continuación para diseñar el programa Java **TestConstantes**