Gobierno de Canarias
Servicio Canario de Empleo

Centro GASTRONÓMICO del Atlántico

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL
SEPE

Módulo 1: Programación en Java 2023

Tema 4 Operadores y la Clase Math

Tema 4

Contenido

1. Tipos Primitivos de Datos

1.1. Objetivos

- Describir los operadores (aritméticos, incrementales, de relación, lógicos y de asignación) y los tipos de dato primitivos sobre los que actúan.
- Evaluar expresiones que empleen datos primitivos, operadores y paréntesis.
- Construir expresiones que empleen combinaciones de datos primitivos, operadores y paréntesis.

Un **operador** lleva a cabo operaciones sobre uno (operador unario), dos (operador binario) o tres (operador ternario) datos u operandos de tipo primitivo devolviendo un valor determinado también de un tipo primitivo. El tipo de valor devuelto tras la evaluación depende del operador y del tipo de los operandos. Por ejemplo, los operadores aritméticos trabajan con operandos numéricos, llevan a cabo operaciones aritméticas básicas y devuelven el valor numérico correspondiente. Los operadores se pueden clasificar en distintos grupos según se muestra en los siguientes apartados.

1.2. Operador de asignación

El **operador asignación** '=' es un operador binario que asigna el valor del término de la derecha al operando de la izquierda. El operando de la izquierda suele ser el identificador de una variable. El término de la derecha es, en general, una expresión de un tipo de dato compatible; en particular puede ser una constante u otra variable. Como caso particular, y a diferencia de los demás operadores, este operador no se evalúa devolviendo un determinado valor.

Operador	Descripción	Ejemplo de expresión	Resultado
=	Operador de asignación	n = 4	n vale 4

No debe confundirse el operador asignación (=) con el operador relacional de igualdad (==) que se verá más adelante. Además, Java dispone de otros operadores que combinan la asignación con otras operaciones (operadores aritméticos combinados).

En el siguiente código se muestran algunos ejemplos de uso del operador asignación con datos de distintos tipos:

```
/**
* Ejemplos de uso del operador asignación
*/
public class OperadorAsignacion {
	public static void main(String[] args) {
		// Declaración de variables
		int i,j;
		float x;
		double y;
		char c;
```

```
            boolean b;
            String s;
            // Asignación de valores
            i = 15;
            j = i;
            x = 335.57F;
            y = 12.345;
            c = 'A';
            b = false;
            s = "Hola";
            // Imprimir valor por pantalla
            System.out.println("i = " + i);
            System.out.println("j = " + j);
            System.out.println("x = " + x);
            System.out.println("y = " + y);
            System.out.println("c = " + c);
            System.out.println("b = " + b);
            System.out.println("s = " + s);
      }
}
```

Salida por pantalla del ejemplo anterior:

```
$ javac OperadorAsignacion.java
$ java OperadorAsignacion
i = 15
j = 15
x = 335.57
y = 12.345
c = A
b = false
s = Hola
```

1.3. Operadores aritméticos

El lenguaje de programación Java tiene varios operadores aritméticos para los datos numéricos enteros y reales. En la siguiente tabla se resumen los siguientes operadores aritméticos:

Operador	Descripción	Ejemplo de expresión	Resultado
-	Operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 – 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2	0.25

%	Resto de la división entera	20 % 7	6

El resultado exacto depende de los tipos de operando involucrados. Es conveniente tener en cuenta las siguientes peculiaridades:

- El resultado es de tipo long si, al menos, uno de los operandos es de tipo long y ninguno es real (float o double).
- El resultado es de tipo int si ninguno de los operandos es de tipo long y tampoco es real (float o double).
- El resultado es de tipo double si, al menos, uno de los operandos es de tipo double.
- El resultado es de tipo float si, al menos, uno de los operandos es de tipo float y ninguno es double.
- El formato empleado para la representación de datos enteros es el complemento a dos. En la aritmética entera no se producen nunca desbordamientos (overflow) aunque el resultado sobrepase el intervalo de representación (int o long).
- La división entera se trunca hacia 0. La división o el resto de dividir por cero es una operación válida que genera una excepción ArithmeticException que puede dar lugar a un error de ejecución y la consiguiente interrupción de la ejecución del programa.
- La aritmética real (en coma flotante) puede desbordar al infinito (demasiado grande, overflow) o hacia cero (demasiado pequeño, underflow).
- El resultado de una expresión inválida, por ejemplo, dividir infinito por infinito, no genera una excepción ni un error de ejecución: es un valor NaN (Not a Number).

```
/**
* Ejemplos de uso de los operadores aritméticos básicos
*/
public class OperadiresAritmeticos {
	public static void main(String[] args) {
		int i,j;
		i = 7;
		j = 3;
		System.out.println("OPERANDOS ENTEROS: i = " + i + " ; j = " + j);
		System.out.println("Operador suma: i + j = " + (i+j));
		System.out.println("Operador resta: i - j = " + (i-j));
		System.out.println("Operador producto: i * j = " + (i*j));
		System.out.println("Operador division: i / j = " + (i/j));
		System.out.println("Operador resto: i % j = " + (i%j));
		double a,b;
		a = 12.5;
		b = 4.3;

		System.out.println("OPERANDOS REALES: a = " + a + " ; b = " + b);

		System.out.println("Operador suma: a + b = " + (a+b));
		System.out.println("Operador resta: a - b = " + (a-b));
		System.out.println("Operador producto: a * b = " + (a*b));
		System.out.println("Operador division: a / b = " + (a/b));
		System.out.println("Operador resto: a % b = " + (a%b));
	}

}
```

```
Salida por pantalla del programa anterior:
OPERANDOS ENTEROS: i = 7 ; j = 3
Operador suma : i + j = 10
```

```
Operador resta : i - j = 4
Operador producto: i * j = 21
Operador division: i / j = 2
Operador resto : i % j = 1
OPERANDOS REALES : a = 12.5 ; b = 4.3
Operador suma : a + b = 16.8
Operador resta : a - b = 8.2
Operador producto: a * b = 53.75
Operador division: a / b = 2.906976744186047
Operador resto : a % b = 3.9000000000000004
```

1.4. Operadores aritméticos incrementales

Los operadores aritméticos incrementales son operadores unarios (un único operando). El operando puede ser numérico o de tipo char y el resultado es del mismo tipo que el operando. Estos operadores pueden emplearse de dos formas dependiendo de su posición con respecto al operando.

Operador	Tipo	Descripción	Ejemplo de expresión	Resultado
++	Incremento		4++	5
		i++ primero se utiliza la variable y luego se incrementa su valor	a = 5; b = a++;	a vale 6 y b vale 5
--	Decremento		4--	**3**
		i++ primero se utiliza la variable y luego se incrementa su valor	a = 5; b = a--;	a vale 4 y b vale 5
		++i primero se incrementa el valor de la variable y luego se utiliza	a = 5; b = --a;	a vale 4 y b vale 4

Estos operadores suelen sustituir a veces al operador asignación y también suelen aparecer en bucles for.

```
/**
* Ejemplo de uso de los operadores incrementales
*/
class OperadoresIncrementales {
        public static void main(String[ ] args) {
                int i,j; // Variables enteras. Podrian ser reales o char
                i = 7;
                System.out.println("Operando entero: i = " + i + ";");
```

```
                System.out.println("Operador ++: j = i++; ");
                j = i++;
                System.out.println("// i vale " + i + "; j vale " + j);
                i = 7;
                System.out.println("i = " + i + ";");
                System.out.println("j = ++i; ");
                j = ++i;
                System.out.println("// i vale " + i + "; j vale " + j);
                i = 7;
                System.out.println("\nOperando entero: i = " + i + ";");
                System.out.println("Operador --: j = i--; ");
                j = i--;
                System.out.println("// i vale " + i + "; j vale " + j);
                i = 7;
                System.out.println("i = " + i + ";");
                System.out.println("j = --i; ");
                j = --i;
                System.out.println("// i vale " + i + "; j vale " + j);
        }
}
```

Salida por pantalla del ejemplo anterior:

```
Operando entero: i = 7;
Operador ++ : j = i++;
// i vale 8; j vale 7
i = 7;
j = ++i;
// i vale 8; j vale 8
Operando entero: i = 7;
Operador -- : j = i--;
// i vale 6; j vale 7
i = 7;
j = --i;
// i vale 6; j vale 6
```

1.5. Operadores aritméticos combinados

Combinan un operador aritmético con el operador asignación. Como en el caso de los operadores aritméticos pueden tener operandos numéricos enteros o reales y el tipo específico de resultado numérico dependerá del tipo de éstos. En la siguiente tabla se resumen los diferentes operadores de esta categoría.

Operador	Descripción	Ejemplo de expresión	Resultado
+=	Suma combinada	a += b	a = a + b
-=	Resta combinada	a -= b	a = a - b
***=**	Producto combinado	a *= b	a = a * b
/=	División combinada	a /= b	a = a / b
%=	Resto combinado	a %= b	a = a % b

Ejemplo de programa que emplea operadores combinados:

```
/**
* Ejemplo de uso de los operadores aritmeticos combinados
*/
public class OperadoresCombinados {
        public static void main(String[ ] args) {
                int i,j; // Variables enteras. Podrian ser reales
                i = 7;
                j = 3;
                System.out.println("Operandos enteros: i = "+ i +" ; j = "+ j);
                i += j;
                System.out.println("Suma combinada: i += j " + " // i vale " + i);
                i = 7;
                i -= j;
                System.out.println("Resta combinada: i -= j " + " // i vale " + i);
                i = 7;
                i *= j;
                System.out.println("Producto combinado: i *= j " + " // i vale " + i);
                i = 7;
```

```
                i /= j;
                System.out.println("Division combinada: i /= j " + " // i vale " + i);
                i = 7;
                i %= j;
                System.out.println("Resto combinada: i %= j " + " // i vale " + i);
        }
}
```

Salida por pantalla del ejemplo anterior:

Operandos enteros : i = 7 ; j = 3

Suma combinada : i += j // i vale 10

Resta combinada : i -= j // i vale 4

*Producto combinado: i *= j // i vale 21*

Division combinada: i /= j // i vale 2

Resto combinada : i %= j // i vale 1

1.6. Operadores de relación

Realizan comparaciones entre datos compatibles de tipos primitivos (numéricos, carácter y booleanos) teniendo siempre un resultado booleano. Los operandos booleanos sólo pueden emplear los operadores de igualdad y desigualdad. En la siguiente tabla se resumen los diferentes operadores de esta categoría:

Operador	Descripción	Ejemplo de expresión	Resultado
==	Igual que	7 == 38	False
!=	Distinto que	'a' i= 'k'	True
<	Menor que	'G' < 'B'	False
>	Mayor que	'b' > 'a'	True
<=	Menor o igual que	7.5 <= 7.38	False
>=	Mayor o igual que	38 >= 7	True

Todos los valores numéricos que se comparan con NaN dan como resultado false excepto el operador '!=' que devuelve true. Esto ocurre incluso si ambos valores son NaN.

Ejemplo de programa que emplea operadores relacionales:

```
/**
* Ejemplo de uso de los operadores relacionales
*/
public class OperadoresRelacionales {
        public static void main(String[ ] args) {
                int i,j;
                i = 7;
                j = 3;
                System.out.println("OPERANDOS ENTEROS: i = "+ i +" ; j = "+ j);
                System.out.println("Operador igualdad: i == j es " + (i==j));
                System.out.println("Operador desigualdad: i != j es " + (i!=j));
                System.out.println("Operador mayor que: i > j es " + (i>j));
                System.out.println("Operador menor que: i < j es " + (i<j));
                System.out.println("Operador mayor o igual que: i >= j es " + (i>=j));
                System.out.println("Operador menor o igual que: i <= j es " + (i<=j));
        }
}
```

Salida por pantalla del programa anterior:

```
OPERANDOS ENTEROS : i = 7 ; j = 3
Operador igualdad : i == j es false
Operador desigualdad : i != j es true
Operador mayor que : i > j es true
Operador menor que : i < j es false
Operador mayor o igual que: i >= j es true
Operador menor o igual que: i <= j es false
```

1.7. Operadores lógicos o booleanos

Realizan operaciones sobre datos booleanos y tienen como resultado un valor booleano.

Operador	Descripción	Ejemplo de expresión	Resultado
!	Negación – NOT (unario)	!false	true
		!(5==5)	false
|	Suma lógica – OR (binario)	true | false	true
		(5 == 5) | (5 < 4)	true
^	Suma lógica exclusiva – XOR (binario)	true ^ false	true
		(5 == 5) ^ (5 < 4)	true
&	Producto lógico – AND (binario)	true & false	false
		(5 == 5) & (5 < 4)	false
||	Suma lógica con cortocircuito	true || false	true
		(5==5) || (5 < 4)	true
&&	Producto lógico con cortocircuito	false && true	false
		(5 == 5) && (5 < 4)	false

- **Suma lógica con corticircuito**: si el primer operando es true entonces el segundo se salta y el resultado es true.
- **Producto lógico con cortocircuito**: si el primer operando es false entonces el segundo se salta y el resultado es false.

Para mejorar el rendimiento de ejecución del código es recomendable emplear en las expresiones booleanas el operador '&&' en lugar del operador '&'. En este caso es conveniente situar la condición más propensa a ser falsa en el término de la izquierda. Esta técnica puede reducir el tiempo de ejecución del programa. De forma equivalente es preferible emplear el operador '||' al operador '|'. En este caso es conveniente colocar la condición más propensa a ser verdadera en el término de la izquierda.

```
/**
* Ejemplo de uso de los operadores lógicos
*/
public class OperadoresBooleanos {
	public static void main(String [] args) {
		System.out.println("Demostracion de operadores logicos");
		System.out.println("Negacion: ! false es : " + (! false));
		System.out.println(" ! true es : " + (! true));
```

```
            System.out.println("Suma: false | false es : " + (false | false));
            System.out.println(" false | true es : " + (false | true));
            System.out.println(" true | false es : " + (true | false));
            System.out.println(" true | true es : " + (true | true));
            System.out.println("Producto: false & false es : " + (false & false));
            System.out.println(" false & true es : " + (false & true));
            System.out.println(" true & false es : " + (true & false));
            System.out.println(" true & true es : " + (true & true));
      }
}
```

Salida por pantalla del programa anterior:

```
Demostracion de operadores logicos
Negacion: ! false es : true
          ! true es : false
Suma: false | false es : false
      false | true es : true
      true | false es : true
      true | true es : true
Producto: false & false es : false
          false & true es : false
          true & false es : false
         true & true es : true
```

1.8. El operador condicional

Este operador ternario tomado de C/C++ permite devolver valores en función de una expresión lógica. Sintaxis:

expresionLogica ? expresion_1 : expresion_2

Si el resultado de evaluar la expresión lógica es verdadero, devuelve el valor de la primera expresión, y en caso contrario, devuelve el valor de la segunda expresión.

Operador	Descripción	Ejemplo de expresión	Resultado
?:	Operador condicional	a = 4;	
		b = (a == 4 ¿ a+5: b-6);	b vale 9
		b = (a > 4 ¿ a * 7: a + 8);	b vale 12

La sentencia de asignación:

valor = (expresionLogica ? expresion_1 : expresion_2);

como se verá más adelante es equivalente a:

if (expresionLogica)

valor = expresion_1;

else

valor = expresion_2

```
/**
* Ejemplo de uso del operador condicional.
*/
public class OperadorCondicional {
	public static void main(String[] args) {
		int i,j,k;
		i = 1;
		j = 2;
		k = i > j ? 2*i : 3*j+1;
		System.out.println("i = " + i);
		System.out.println("j = " + j);
		System.out.println("k = " + k);
		i = 2;
		j = 1;
		k = i > j ? 2*i : 3*j+1;
		System.out.println("i = " + i);
		System.out.println("j = " + j);
		System.out.println("k = " + k);
	}
}
```

Salida por pantalla del programa anterior:

i = 1

j = 2

k = 7

i = 2

j = 1

k = 4

1.9. Operador concatenación de cadenas

El operador concatenación '+', es un operador binario que devuelve una cadena resultado de concatenar las dos cadenas que actúan como operandos. Si sólo uno de los operandos es de tipo cadena, el otro operando se convierte implícitamente en tipo cadena.

Operador	Descripción	Ejemplo de expresión	Resultado
+	Operador de concatenación	"Hola" + " Juan"	Hola Juan

1.10. Separadores

Existen algunos caracteres que tienen un significado especial en el lenguaje Java.

Separador	Descripción
()	Permiten modificar la prioridad de una expresión, contener expresiones para el control de flujo y realizar conversiones de tipo. Por otro lado, pueden contener la lista de parámetros o argumentos, tanto en la definición de un método como en la llamada al mismo.
{}	Permiten definir bloques de código y ámbitos y contener los valores iniciales de las variables array.
[ ]	Permiten declarar variables de tipo array (vectores o matrices) y referenciar sus elementos.
;	Permite separar sentencias.
,	Permite separar identificadores consecutivos en la declaración de variables y en las listas de parámetros. También se emplea para encadenar sentencias dentro de un bucle for.
.	Permite separar el nombre de un atributo o método de su instancia de referencia. También separa el identificador de un paquete de los de los subpaquetes y clases.

1.11. Expresiones

Una expresión es una combinación de operadores y operandos que se evalúa generándose un único resultado de un tipo determinado.

1.11.1. Prioridad entre operadores

Si dos operadores se encuentran en la misma expresión, el orden en el que se evalúan puede determinar el valor de la expresión. En la siguiente tabla se muestra el orden o prioridad en el que se ejecutan los operadores que se encuentren en la misma sentencia. Los operadores de la misma prioridad se evalúan de izquierda a derecha dentro de la expresión.

Prioridad	Operador	Tipo de operador	Operación
1	++	Aritmético	Incremento previo o posterior (unario)
	--	Aritmético	Decremento previo o posterior (unario)
	+, -	Aritmético	Suma unaria, Resta unaria
	!	Booleano	Negación (unario)

2	(tipo)	Cualquiera	
3	*, /, %	Aritmético	Multiplicación, división, resto
4	+, -	Aritmético	Suma, resta
	+	Cadena	Concatenación de cadenas
5	<, <=	Aritmético	Menor que, menor o igual que
	>, >=	Aritmético	Mayor que, mayor o igual que
	Instanceof	Objeto, tipo	Comparación de tipos
6	==	Primitivo	Igual (valores idénticos)
	!=	Primitivo	Distinto (valores idénticos)
	==	Objeto	Igual (referencia al mismo objeto)
	!=	Objeto	Distinto (referencia a distintos objetos)
7	&	Booleano	Producto booleano
8	^	Booleano	Suma exclusiva booleana
9	|	Booleano	Suma booleana
10	&&	Booleano	AND condicional
11	||	Booleano	OR condicional
12	?:	Booleano	Operador condicional (ternario)
13	=	Variable	Asignación
14	*=, /=, %=	Variable	Asignación con operación
	+=, -=		
	<<=, >>=		
	>>>=		

1.12. La clase Math

En cuanto a las funciones matemáticas en Java, las funciones disponibles vienen definidas en la clase Math. Hay muchas funciones disponibles. Se puede consultar la lista completa en la documentación oficial del API de Java:

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/Math.html

La función Math se encuentra dentro del paquete java.lang.

A continuación, se muestra un listado de las funciones más importantes:

Función matemática	Significado	Ejemplo de uso	Resultado
abs	Valor absoluto	double x = Math.abs(-2,3);	x = 2.3;
sqrt	Raíz cuadrada	double x = Math.sqrt(9);	x = 3.0;
pow	Potencia	double x = Math.pow(2,3);	x = 8.0;
round	Redondeo	double x = Math.round(2.5);	x = 3;
floor	Redondeo al entero menor	double x = Math.floor(2.5);	x = 2.0;
ceil	Redondeo al entero mayor	double x = Math.ceil(2.5);	x = 3.0;
random	Número aleatorio	double x = Math.random();	x = 0.20614522323378;
max	Mayor de 2 números	double x = Math.max(2.5, 9.3);	x = 9.3;

Destacar que las funciones matemáticas, al pertenecer a la clase Math, se invocan siempre de la siguiente manera:

Math.funcion(argumentos)

Ejemplo de uso de algunas de funciones de la clase Math:

```
public class EjemploClaseMath {
        public static void main(String[ ] args) {
        double operador1 = 25.5;
        double operador2 = 15.21;
        System.out.println(Math.ceil(operador1)); // Devuelve 26.0
        System.out.println(Math.floor(operador2)); // Devuelve 15.0
        // Devuelve 2.474435537975361E21
        System.out.println(Math.pow(operador1, operador2));
        System.out.println(Math.max(operador1, operador2)); // Devuelve 25.5
        // Devuelve 5.049752469181039
        System.out.println(Math.sqrt(operador1));
    }
}
```

El resultado de ejecutar el programa anterior es:

```
26.0
15.0
2.474435537975361E21
25.5
5.049752469181039
```

Si vamos a trabajar con constantes físicas o matemáticas, resulta de interés la instrucción final para la declaración de constantes. La ventaja de declarar una constante en vez de una variable, consiste en que la constante no puede variar en el transcurso del programa. Por tanto, se impide que por error pueda tener un valor no válido en un momento dado. Una declaración tipo de constante podría ser la siguiente:

final double pi = 3.14159265358979;

Sin embargo, Java tiene una constante propia para definir la constante matemática PI: Math.PI

A continuación, se muestra un listado de constantes de la clase Math:

Constante	Tipo	Descripción	Valor
PI	double	Devuelve el valor de PI	3.14159265358979
E	double	Devuelve el valor E	2.718281828459045

El siguiente programa, muestra el uso de la función Math.PI en la conversión de un ángulo sexagesimal a radianes:

```
public class Radianes {
        public static void main(String args[ ]) {
                double sexagesimal = 30;
                double radianes = Math.PI/180 * sexagesimal;
                System.out.println("Ángulo en radianes: " + radianes);
        }
}
```

El resultado de ejecutar el programa anterior es:

Ángulo en radianes: 0.5235987755982988

1.12.1. Diferencia entre round, ceil y floor

Las funciones round, ceil y floor se usan para obtener un entero próximo a un número decimal y tienen similitudes, de hecho en algunos casos devuelven el mismo resultado. Sin embargo también tienen diferencias que es interesante conocer. Estas tres funciones se aplican sobre valores numéricos decimales y retornan un valor numérico que en el caso de round es un entero long, mientras que en el caso de floor y ceil retornan un valor de tipo double coincidente o equivalente con un entero.

- Método round: redondea siempre al entero más próximo, por ejemplo 2.6 redondea a 3 mientras que -2.6 redondea a -3. Si el decimal está exactamente entre dos valores se redondea al entero superior más próximo (por ejemplo 2.5 redondea a 3 y -2.5 redondea a -2).
- Método floor: devuelve el entero menor, por ejemplo 2.9 quedaría en 2.0 y -2.9 quedaría en -3.0. También 2.1 quedaría en 2.0 y -2.1 quedaría en -3.0.
- Método ceil: devuelve el entero mayor, por ejemplo 2.9 quedaría en 3.0 y -2.9 quedaría en -2.0. También 2.1 quedaría a 3.0 y -2.1 quedaría en -2.0.

En cada programa deberemos determinar qué método es el adecuado para obtener los resultados deseados. Por ejemplo, si tenemos que redondear cantidades de dinero parece más lógico utilizar round. En cambio, si estamos trabajando con edades de una persona en años utilizando decimales, podremos razonar de otra manera. La edad de una persona es un valor positivo (no es posible que tome valores negativos). Una persona decimos que tiene x años mientras no cumpla x+1 años, de modo que en todo el periodo intermedio decimos que tiene x años.

Por ejemplo, si cumplo 35 años el 4 de febrero de 2017, desde el 4 de febrero de 2016 hasta el 3 de febrero de 2017 diré que tengo 35 años. Pero en un programa que trabaje con decimales, en el punto intermedio entre estas dos fechas tendría 35.50 años. Si quiero obtener el valor que representa la edad a partir del valor decimal, lo lógico será utilizar el método floor porque nos devolverá 35 para todos los valores decimales entre 35 y 36, tal y como expresamos en el lenguaje natural las edades. En este caso tanto round como ceil no ofrecerían un resultado adecuado a nuestros intereses.

```
public class Redondeo {
double x=0;
public static void main(String[ ] args) {
System.out.println("x = Math.round(2.6); dará como resultado : "+ Math.round(2.6));
System.out.println("x = Math.round(-2.6); dará como resultado : "+ Math.round(-2.6));
System.out.println("x = Math.round(2.4); dará como resultado : "+ Math.round(2.4));
System.out.println("x = Math.round(-2.4); dará como resultado : "+ Math.round(-2.4));
System.out.println("x = Math.round(2.5); dará como resultado : "+ Math.round(2.5));
System.out.println("x = Math.round(-2.5); dará como resultado : "+ Math.round(-2.5)+"\n");
System.out.println("x = Math.ceil(2.6); dará como resultado : "+ Math.ceil(2.6));
System.out.println("x = Math.ceil(-2.6); dará como resultado : "+ Math.ceil(-2.6));
```

```
System.out.println("x = Math.ceil(2.4); dará como resultado : "+ Math.ceil(2.4));
System.out.println("x = Math.ceil(-2.4); dará como resultado : "+ Math.ceil(-2.4));
System.out.println("x = Math.ceil(2.5); dará como resultado : "+ Math.ceil(2.5));
System.out.println("x = Math.ceil(-2.5); dará como resultado : "+ Math.ceil(-2.5)+"\n");
System.out.println("x = Math.floor(2.6); dará como resultado : "+ Math.floor(2.6));
System.out.println("x = Math.floor(-2.6); dará como resultado : "+ Math.floor(-2.6));
System.out.println("x = Math.floor(2.4); dará como resultado : "+ Math.floor(2.4));
System.out.println("x = Math.floor(-2.4); dará como resultado : "+ Math.floor(-2.4));
System.out.println("x = Math.floor(2.5); dará como resultado : "+ Math.floor(2.5));
System.out.println("x = Math.floor(-2.5); dará como resultado : "+ Math.floor(-2.5));
}
}
```

El resultado de ejecutar el programa anterior es el siguiente:

```
x = Math.round(2.6); dará como resultado : 3
x = Math.round(-2.6); dará como resultado : -3
x = Math.round(2.4); dará como resultado : 2
x = Math.round(-2.4); dará como resultado : -2
x = Math.round(2.5); dará como resultado : 3
x = Math.round(-2.5); dará como resultado : -2
x = Math.ceil(2.6); dará como resultado : 3.0
x = Math.ceil(-2.6); dará como resultado : -2.0
x = Math.ceil(2.4); dará como resultado : 3.0
x = Math.ceil(-2.4); dará como resultado : -2.0
x = Math.ceil(2.5); dará como resultado : 3.0
x = Math.ceil(-2.5); dará como resultado : -2.0
x = Math.floor(2.6); dará como resultado : 2.0
x = Math.floor(-2.6); dará como resultado : -3.0
x = Math.floor(2.4); dará como resultado : 2.0
x = Math.floor(-2.4); dará como resultado : -3.0
x = Math.floor(2.5); dará como resultado : 2.0
x = Math.floor(-2.5); dará como resultado : -3.0
```

1.13. Formatear los datos de salida

El método **System.out.printf** se usa para dar formato a los datos que se imprimen por pantalla en Java. Este problema se nos plantea por ejemplo cuando queremos mostrar un número de tipo float o double con un número determinado de decimales y no con los que por defecto muestra Java.

A partir de la versión Java 5 se incorporan los métodos format y printf que permiten aplicar un formato a la salida de datos por pantalla. Ambos realizan la misma función, tienen exactamente el mismo formato y emulan la impresión con formato printf() de C.

Si queremos mostrar el número 12.3698 de tipo double con dos decimales:

System.out.printf("%.2f %n", 12.3698);

- %: indica que en esa posición se va a escribir un valor. El valor a escribir se encuentra a continuación de las comillas.
- .2: indica el número de decimales.
- f: indica que el número es de tipo float o double.
- %n: indica un salto de línea. Equivale a \n. Con printf podemos usar ambos para hacer un salto de línea.

La salida por pantalla de la sentencia anterior es:

12,37

Otra fórmula más elegante para dar formato a un número es utilizar la clase **DecimalFormat**:

```
DecimalFormat df = new DecimalFormat();
df.setMaximumFractionDigits(3);
System.out.println(df.format(12.3698));
```

- **setMaximunFractionsDigits**: método que indica el número de decimales a mostrar.
- **format**: formatea un número utilizando el valor definido previamente por *setMaxumunFractionDigits*.

La salida por pantalla de las sentencias anteriores es:

12,37

2. Bibliografía

- Material suministrado por CGA.
- https://www.discoduroderoer.es/metodos-de-la-clase-math-de-java/
- http://dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/UT3/ActividadesAlumnos/java7/paginas/pag13.html
- https://programandojava.webnode.es/products/a18-la-clase-math/
- http://www.manualweb.net/java/operadores-asignacion-aritmeticos-java/
- http://dis.um.es/~bmoros/Tutorial/parte4/cap4-2.html