



ORACLE

Academy



Java Foundations

3-2

Datos Numéricos

ORACLE
Academy



Copyright © 2022, Oracle y/o sus filiales. Oracle, Java y MySQL son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

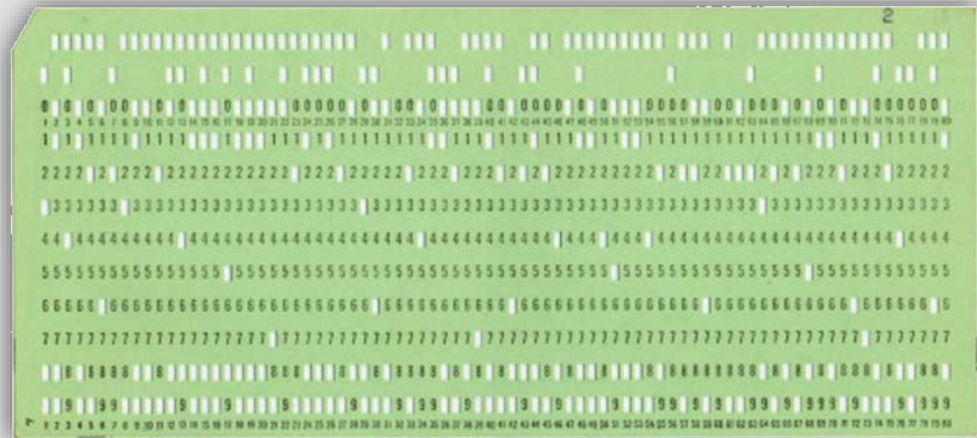
Objetivos

- En esta lección se abordan los siguientes objetivos:
 - Distinguir entre distintos tipos de datos enteros (byte, short, int, long)
 - Distinguir entre tipos de datos de coma flotante (float, double)
 - Manipular y realizar operaciones matemáticas con datos numéricos
 - Usar paréntesis y saber ordenar las operaciones



Información sobre Datos

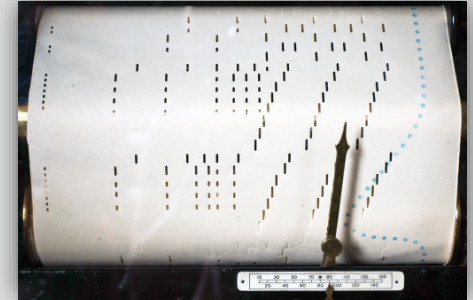
- En los comienzos de la informática, los datos se almacenaban en tarjetas perforadas



- Cada ranura tenía 2 posibles estados:
 - Perforado
 - Sin perforar

Lectura de los Datos en Tarjetas Perforadas

- Las pianolas leen tarjetas perforadas
- Una columna representa una tecla de la pianola
- La tarjeta perforada se desplaza en la pianola y va activando teclas
- Cada ranura tiene 2 estados posibles con 2 resultados posibles:



Rollo para pianola del siglo XIX

Estado	Resultado
Perforado	Reproduce la nota
Sin perforar	No reproduce la nota

Algo de Información sobre la Informática Moderna

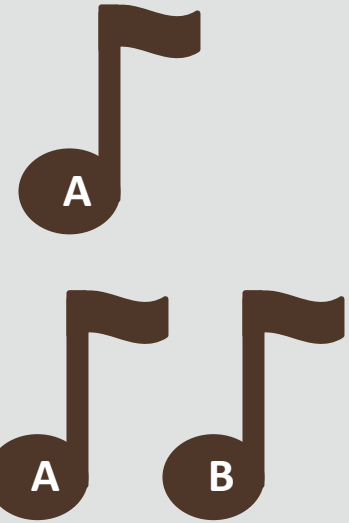
- El procesamiento actual de datos sigue teniendo que representar 2 estados:
 - Esto se interpreta como código binario: **10011101**
 - Cada **1** o **0** es lo que se denomina un bit

	Pianola	Informática actual
bit	Perforado/sin perforar	1/0
Los bits son instrucciones para...	Componentes mecánicos	El procesador
Medio	Mecánico	Electromagnético
Los bits almacenan datos relativos a...	Teclas de la pianola	Números

*Veamos esto
en más detalle*

Bits de Datos

- Una tecla de la pianola está representada por 1 bit
 - 0: No suena
 - 1: Suena
- Dos teclas requieren 2 bits
 - Hay 4 posibles combinaciones de teclas
 - Esto se puede calcular como 2^2

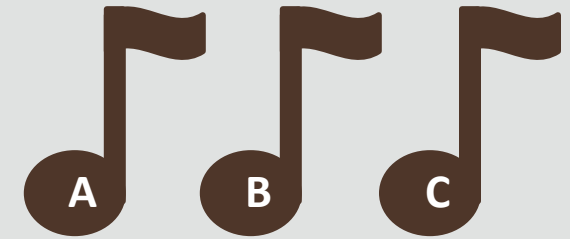


Silencio
Solo B
Solo A
Tanto A como B

Tecla A	Tecla B
0	0
0	1
1	0
1	1

Bits de Datos de Mayor Tamaño

- Tres teclas requieren 3 bits
 - Hay 8 posibles combinaciones de teclas
 - Esto se puede calcular como 2^3
- Ocho teclas requieren 8 bits
 - Hay 256 posibles combinaciones
 - Esto se puede calcular como 2^8



Tecla A	Tecla B	Tecla C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Bits y Bytes

- Ocho bits forman un byte
- Un byte de Java puede almacenar 256 posibles valores
- Los valores posibles son aquellos comprendidos entre -128 a 127
 - 128 valores por debajo de 0
 - 127 valores por encima de 0
 - 1 valor igual a 0



```
byte x = 127;
```



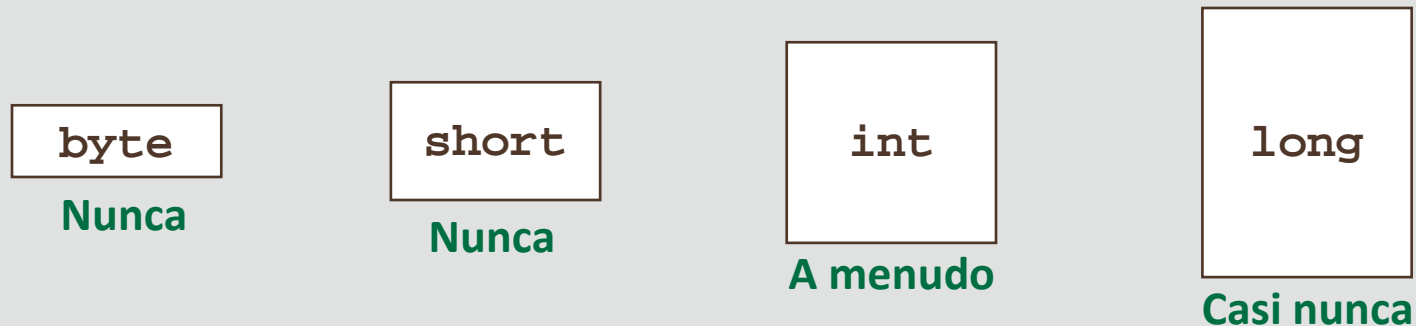
```
byte z = 128;    //Too high
```

Algunos Nuevos Tipos Primitivos Integrales

Tipo	Length	Número de valores posibles	Valor mínimo	Valor máximo
Byte	8 bits	2^8 o... 256	-2^7 o... -128	2^7-1 o... 127
short	16 bits	2^{16} o... 65.535	-2^{15} o... -32.768	$2^{15}-1$ o... 32.767
int	32 bits	2^{32} or... 4.294.967.296	-2^{31} o... -2.147.483.648	$2^{31}-1$ o... 2.147.483.647
long	64 bits	2^{64} o... 18.446.744.073.709.551 .616	-2^{63} o... -9.223.372.036. 854.775.808L	$2^{63}-1$ o... 9.223.372.036. 854.775.807L

Fíjese en la L

¿Cuándo se Debe Utilizar Cada Tipo de Dato?



- Los tipos byte y short se usan para reducir el consumo de memoria en dispositivos pequeños o más antiguos
- Pero los escritorios modernos disponen de bastante memoria
- De los 4 tipos indicados, vamos a centrarnos principalmente en el tipo int en este curso

Buscar x

```
int x = 20;  
x = 25;  
x = 5 + 3;  
  
System.out.println(x);
```

- x siempre equivale a 20...
 - Hasta que le asigne un valor diferente a x
- Es posible asignar a x un valor calculado

Values for x: ~~20~~ ~~25~~ 8

Buscar x

```
int x = 20;  
x = 25;  
x = 5 + 3;  
x = x + 1;  
x += 1;  
x++;  
System.out.println(x);
```

- Es posible asignar un nuevo valor a x en función de su valor actual:
 - Java ofrece un operador abreviado += para ello
 - Añadir 1 a una variable es tan común que Java cuenta con el operador abreviado ++

Values for x: ~~20~~ ~~25~~ ~~8~~ ~~9~~ ~~49~~ 11

Volver a Buscar x

- Es posible asignar a X el valor de otra variable:
 - Cambiar y no implica que x cambie
 - x e y son variables independientes

```
int y = 20;  
int x = y;  
y++;
```

```
System.out.println(x);  
System.out.println(y);
```

- Resultado:

x	20
y	21

Operadores matemáticos estándar

Objetivo	Operador	Ejemplo	Comentarios
Suma	+	<pre>int sum = 0; int num1 = 10; int num2 = 2; sum = num1 + num2;</pre>	Si num1 es 10 y num2 es 2, sum será 12
Resta	-	<pre>int diff = 0; int num1 = 10; int num2 = 2; diff = num1 - num2;</pre>	Si num1 es 10 y num2 es 2, diff será 8

Operadores matemáticos estándar

Objetivo	Operador	Ejemplo	Comentarios
Multiplicación	*	<pre>int prod = 0; int num1 = 10; int num2 = 2; prod = num1 * num2;</pre>	Si num1 es 10 y num2 es 2, prod será 20
División	/	<pre>int quot = 0; int num1 = 31; int num2 = 2; quot = num1 / num2;</pre>	<p>Si num1 es 31 y num2 es 6, quot será 5</p> <p>El resto se desecha</p> <p>Nota: Dividir entre cero devuelve un error</p>

¿Por qué?

Como los tipos de datos `int` son solo números enteros, se descartará el resto decimal. Verá cómo cambiar este comportamiento más adelante en esta lección.

Uso de operadores abreviados de Java para realizar asignaciones

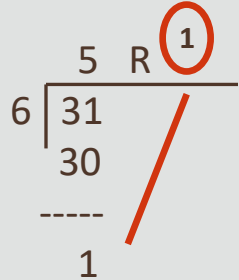
- Un operador abreviado es una forma más corta de expresar algo que ya está disponible en el lenguaje de programación Java

Objetivo	Operador	Ejemplos de operadores abreviados	Construcción equivalente	resultado
Sumar y asignar	+=	<pre>int a = 6; int b = 2; a += b;</pre>	<pre>int a = 6; int b = 2; a = a + b;</pre>	a = 8
Restar y asignar	-=	<pre>int a = 6; int b = 2; a -= b;</pre>	<pre>int a = 6; int b = 2; a = a - b;</pre>	a = 4

Uso de operadores abreviados de Java para realizar asignaciones

Objetivo	Operador	Ejemplos de operadores abreviados	Construcción equivalente	resultado
Multiplicar y asignar	<code>*=</code>	<pre>int a = 6; int b = 2; a *= b;</pre>	<pre>int a = 6; int b = 2; a = a * b;</pre>	<code>a = 12</code>
Dividir y asignar	<code>/=</code>	<pre>int a = 6; int b = 2; a /= b;</pre>	<pre>int a = 6; int b = 2; a = a / b;</pre>	<code>a = 3</code>
Obtener resto y asignar	<code>%=</code>	<pre>int a = 6; int b = 2; a %= b;</pre>	<pre>int a = 6; int b = 2; a = a % b;</pre>	<code>a = 0</code>

Operador de Módulo

Objetivo	Operador	Ejemplo	Comentarios
Resto	% módulo	<pre>num1 = 31; num2 = 6; mod = num1 % num2; mod = 1</pre>	<p>Módulo busca el resto del primer número dividido entre el segundo número</p>  <p>Módulo siempre da una respuesta con el mismo signo como primer operando</p>



Operadores de Aumento y Disminución (++ y --)

- Forma extendida:

- `age = age + 1;`

o

- `count = count - 1;`

- Forma breve:

- `age++;`

o

- `count--;`



Más Información sobre los operadores de aumento y disminución

Operador	Objetivo	Ejemplo
++	Aumento previo (++variable)	<code>int id = 6;</code> <code>int newId = ++id;</code> id es 7, newId es 7
	Aumento posterior (variable++)	<code>int id = 6;</code> <code>int newId = id++;</code> id es 7, newId es 6
--	Disminución previa (--variable)	(Se aplica el mismo principio)
	Disminución posterior (variable--)	

Operadores de Aumento y Disminución (++ y --)

```
1  int count=15;  
2  int a, b, c, d;  
3  a = count++;  
4  b = count;  
5  c = ++count;  
6  d = count;  
7  System.out.println(a + ", " + b + ", " + c + ", " + d);
```

- Resultado:

15, 16, 17, 17

Ejercicio 1, Parte 1

- Cree un nuevo proyecto y agréguele el archivo `Chickens01.java`
- Lea esta historia y calcule/imprima el número de `totalEggs` recogidos entre el lunes y el miércoles:
 - Las gallinas del granjero Paco Torres siempre ponen `eggsPerChicken` huevos a las 12 en punto del mediodía, que él recoge a lo largo del día
 - El lunes, Paco tiene `chickenCount` pollos
 - El martes por la mañana, se hace con 1 gallina más
 - El miércoles por la mañana, un zorro se come la mitad de las gallinas
 - ¿Cuántos huevos consigue recoger Paco, si empieza con...?
 - `eggsPerChicken = 5, chickenCount = 3`
 - `eggsPerChicken = 4, chickenCount = 8`

Ejercicio 1, Parte 2

- El programa debería generar los siguientes resultados:

45 Primer caso

84 Segundo caso

Engaño de la División de Enteros

- El zorro se comió a la mitad de las gallinas
- Si dividimos 9 pollos por la mitad, Java asume que $9/2 = 4$
 - Pero, en realidad, $9/2 = 4,5$
 - ¿No debería Java redondear a 5?
 - ¿Qué sucede en este caso?



División Java

- Los enteros no se redondean en Java
- Los enteros de Java se truncan, es decir, que todos los números detrás de la coma decimal se eliminan

```
int x = 9/2;  
System.out.println(x); //prints 4
```

- Necesitamos otros tipos de datos si se nos presentan situaciones como comas flotantes que requieren de una mayor precisión

Tipos primitivos de coma flotante

Tipo	Longitud Float	¿Cuándo voy a utilizar esto?
<code>float</code>	32 bits	Nunca
<code>double</code>	64 bits	A menudo

*Duplica la precisión
de un tipo float*

- Ejemplo:

```
-public float pi    = 3.141592F;  
-public double pi   = 3.141592;
```

Fíjese en la F

Doble Decepción

- El problema original:

```
int x = 9/2;  
System.out.println(x); //prints 4
```

- ¿No debería un valor double x solucionar el problema?

```
double x = 9/2;  
System.out.println(x); //prints 4.0
```

- ¡No!
- ¿Por qué no?

Doble Decepción

```
double x = 9/2;  
System.out.println(x); //prints 4.0
```

- Java resuelve la expresión, trunca el .5 y, a continuación, convierte la respuesta en un valor double
- La expresión contiene solo valores ints. Java no asigna la memoria adicional que necesita double hasta que no le quede más remedio hacerlo
 - Solución: incluir un valor double en la expresión

```
double x = 9/2.0;  
System.out.println(x); //prints 4.5
```

Una Última Nota

- Declare una variable con la palabra clave final de modo que su valor no se pueda modificar (inmutable)

```
final double PI = 3.141592;  
PI = 3.0;           //Not Allowed
```

- Java se quejará si intenta cambiar el valor de una variable final
- Reglas de nomenclatura de las variables finales:
 - Hay que poner todas las letras en mayúscula
 - Las palabras se separan mediante guiones bajos
 - MINIMUM_AGE
 - SPEED_OF_LIGHT

Ejercicio 2, Parte 1

- Cree un nuevo proyecto y agréguele el archivo `Chickens02.java`
- Lea esta historia y calcule/imprima los valores necesarios:
 - El lunes, el granjero Pedro recoge 100 huevos
 - El martes, recoge 121 huevos
 - El miércoles, recoge 117 huevos
 - ¿Cuál es el valor diario de huevos recogidos (`dailyAverage`)?
 - ¿Cuántos huevos cabría esperar como media en un mes de 30 días (`monthlyAverage`)?
 - Si se obtienen unos beneficios de 0,18 dólares por huevo, ¿qué beneficio total obtendrá Pedro (`monthlyProfit`) para todos los huevos?

Ejercicio 2, Parte 2

- El programa debería generar los siguientes resultados:

```
Daily Average:    112.66666666666667
Monthly Average:  3380.0
Profit:           $608.4
```




Los Paréntesis en las Expresiones Matemáticas

- Escribir esta expresión sin paréntesis...

```
int x = 10 +20 +30 / 3;           //x=40
```

- Es como escribir esta expresión con paréntesis:

```
int x = 10 +20 +(30 / 3);         //x=40
```

- Si desea obtener una media, utilice paréntesis como se indica a continuación:

```
int x = (10 +20 +30) / 3;         //x=20
```



Prioridad de Operadores

- A continuación se presenta un ejemplo de la necesidad de reglas de prioridad:

```
int x = 25 - 5 * 4 / 2 - 10 + 4;
```

- ¿Es la respuesta 34 o 9?
- Agregue paréntesis para aplicar la prioridad

Reglas de Prioridad

- Operadores dentro de un par de paréntesis
- Operadores de aumento y disminución (++ o --)
- Operadores de multiplicación y división, evaluados de izquierda a derecha
- Operadores de suma y resta, evaluados de izquierda a derecha
- Si aparecen sucesivamente operadores con la misma prioridad, los operadores se evalúan de izquierda a derecha

Uso de paréntesis

- Las expresiones se evalúan utilizando las reglas de prioridad
- Sin embargo, debe utilizar paréntesis para proporcionar la estructura que desea
- Ejemplos:

```
int x = (((25 - 5) * 4) / (2 - 10)) + 4;  
int x = ((20 * 4) / (2 - 10)) + 4;  
int x = (80 / (2 - 10)) + 4;  
int x = (80 / -8) + 4;  
Int x = -10 + 4;  
int x = -6;
```

Resumen

- En esta lección, debe haber aprendido lo siguiente:
 - Distinguir entre distintos tipos de dato entero (byte, short, int, long)
 - Distinguir entre tipos de dato de coma flotante (float, double)
 - Manipular y realizar operaciones matemáticas con datos numéricos
 - Usar paréntesis y saber ordenar las operaciones





ORACLE

Academy

