

# 1º CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

## UD 2. elementos básicos del lenguaje Java.

Módulo: Programación



Centro de Enseñanza  
Gregorio Fernández

# Identificadores

- Los **identificadores** son elementos textuales que se utilizan para nombrar los elementos de un programa (clases, variables, métodos, ...)
- En Java, un identificador puede ser una secuencia de:
  - Dígitos (no para la primera letra)
  - Letras
  - Caracteres de subrayado: \_
  - Caracteres dólar: \$
- Los identificadores no pueden ser palabras reservadas.
- Pueden tener cualquier longitud.
- Java es case-sensitive por lo que los siguientes identificadores son distintos:  
ALFA ≠ alfa ≠ ALFa ≠ alFA





# Identificadores

HolaMundo	✓	HolaMundo01	✓
Hola_Mundo	✓	01HolaMundo	✗
Hola Mundo	✗	i	✓
\$HolaMundo	✓	\$	✓
_Hola_Mundo	✓	_	✓
Hola\$Mundo	✓	_\$	✓
Hola-Mundo	✗	\$\$	✓
\$_\$HolaMundo	✓	_1	✓



Centro de Enseñanza  
Gregorio Fernández

# Palabras reservadas

- Son palabras que para Java tienen algún significado especial.
- No pueden usarse como identificadores.
- Ejemplos:

```
int char = 'C'    //Error de compilación  
class void { }    //Error de compilación
```

## Signos de puntuación

;  
! % \$ \* ( ) - + = { } ~ ^ | [ ] \ ' \_ < > ? , . / "

Como separadores podemos utilizar:





- › Tabulaciones
- › Espacios en blanco
- › Retornos de carro



Centro de Enseñanza  
Gregorio Fernández



# Paquetes (packages)

- Son una forma especial de agrupar clases con cierta relación.
- Algunos de los paquetes predefinidos más importantes en Java son:
-  **java.lang**: contiene el núcleo del lenguaje Java. Algunas clases contenidas en este paquete son: System, String, Integer, Math, Object, Exception, Error.
-  **java.io**: contiene clases para trabajar con la entrada/salida. Algunas clases contenidas en este paquete son: BufferedReader, InputStreamReader, FileInputStream, FileOutputStream.
-  **java.util**: contiene clases de utilidad, como las colecciones, el modelo de eventos, utilidades para trabajar con fechas y horas, una clase para generar números aleatorios, la clase StringTokenizer, etc.
-  **java.applet, java.awt y javax.swing**: paquetes de clases para trabajar con applets y aplicaciones gráficas basadas en ventanas.
- A parte de los paquetes predefinidos, el programador puede crear sus propios paquetes.



Centro de Enseñanza  
Gregorio Fernández

# Tipos de datos

- Un tipo de dato es el **conjunto de valores** que puede tomar una variable.
- Los tipos de datos básicos (**primitivos**) en Java son (todos en minúsculas):

Grupo	Tipo	Tamaño	Rango	Ejemplo
Caracteres	char	2 bytes	Caracteres en Unicode	'C'
Enteros	byte	1 byte	-128 a 127	-15
	short	2 bytes	-32.768 a 32.767	1024
	int	4 bytes	-2.147.483.648 a 2.147.483.649	42235
	long	8 bytes	$-9 * 10^{18}$ a $9 * 10^{18}$	58.262.143.218
Reales	float	4 bytes	$-3,4 * 10^{38}$ a $3,4 * 10^{38}$	10,5
	double	8 bytes	$-1,79 * 10^{308}$ a $1,79 * 10^{308}$	0,00045
Lógicos	boolean	1 bit	true, false	true



Centro de Enseñanza  
Gregorio Fernández



# Tipos de datos

## I - Enteros

- Los tipos **byte**, **short**, **int** y **long** sirven para almacenar datos enteros.
- Los enteros son números sin decimales.
- A las variables se pueden asignar enteros normales o enteros octales y hexadecimales. Los octales se indican anteponiendo un cero al número, los hexadecimales anteponiendo 0x.
- Ejemplos de declaración de variables enteras:

```
int numero=16; //16 decimal
numero=020;    //20 octal=16 decimal
numero=0x10;   //10 hexadecimal=16 decimal
```

- Java considera los números enteros de tipo **int** (32 bits) salvo si al final se le coloca la letra **L**, se entenderá entonces que es de tipo **long**.

<code>byte b1=1; //Correcto</code>	<code>long l1=2000000000; //Correcto</code>
<code>byte b2=130; //Incorrecto</code>	<code>long l2=3000000000; //Incorrecto</code>
<code>short s1=30000; //Correcto</code>	<code>long l3=3000000000L; //Correcto</code>
<code>short s2=35000; //Incorrecto</code>	
<code>int i1=2000000000; //Correcto</code>	
<code>int i2=3000000000; //Incorrecto</code>	



Centro de Enseñanza  
Gregorio Fernández

# Tipos de datos

## I - Enteros

- No se acepta *en general* asignar variables de distinto tipo. Sí se pueden asignar valores de variables enteras a variables enteras de un tipo superior (por ejemplo asignar un valor **int** a una variable **long**).

Pero al revés no se puede porque se perdería precisión.

```
int i=12;  
byte b=i; //Error de compilación
```

- La solución hacer un **cast** ó **casting**. Esta operación permite convertir valores de un tipo a otro:

```
int i=12;  
byte b= (byte) i; //No hay problema por el (cast)
```

- Hay que tener en cuenta en estos **casting** que si el valor asignado sobrepasa el rango del elemento, el valor convertido no tendrá ningún sentido:

```
int i=1200;  
byte b=(byte) i; //El valor de b no tiene sentido
```



Centro de Enseñanza  
Gregorio Fernández



# Tipos de datos

## I - Enteros

- Java utiliza 32 bits (int = 4 bytes) para realizar cualquier operación aritmética entre enteros de tipo **byte**, **short** e **int**. Es decir, promociona todos los operandos de tamaño  $\leq 32$  bits a tipo **int** para operar con la mayor precisión posible.
- Cuando alguno de los operandos es de tipo **long** las operaciones se hacen con 64 bits (long = 8 bytes), es decir, se promocionan los operandos a tipo **long**.
- Para realizar todas estas operaciones Java convierte automáticamente los operandos a **int** o **long**, y devuelve un tipo de dato **int** o **long**.
- Así, las siguientes sentencias dan error:

```
short a,b,c;  
a=1;  
b=2;  
c=a+b //Error de compilación: posible pérdida de precisión.
```

- ¿Solución?



Centro de Enseñanza  
Gregorio Fernández

# Tipos de datos

## II – Coma flotante

- Los números con decimales (reales) se almacenan en los tipos **float** y **double**.
- Los decimales no son almacenados de forma exacta por eso siempre hay un posible error.
- En los decimales de coma flotante se habla por tanto de precisión. Es mucho más preciso el tipo **double** (15 dígitos de precisión) que el tipo **float** (7 dígitos).
- A un valor literal (como 1.5 por ejemplo), se le puede indicar con una **F** al final del número que es float (1.5F por ejemplo) o una **D** para indicar que es double.
- Si no se indica nada, Java considera siempre que es double, por lo que al usar tipos float hay que convertir los literales para que no se produzcan errores de compilación.

```
float f=1.0;    //Error de compilación
```

```
float f=1.0F;   //Ok
```

```
double d=1.0;   //Ok
```

- Como se ve el comportamiento es distinto al caso de los enteros.



Centro de Enseñanza  
Gregorio Fernández



# Tipos de datos

## III – Booleanos

- Los valores booleanos (o lógicos) sirven para indicar si algo es verdadero (**true**) o falso (**false**).
- Si a un valor booleano no se le da un valor inicial se toma como valor inicial el valor **false**.
- Normalmente los tipos de datos booleanos se utilizan para las sentencias de selección (**if**) o estructuras de control (**bucles**).
- Ejemplos:

```
boolean encontrado = true; //Ok
boolean salir="false";      //Error de compilación
                             false ≠ "false"
boolean fin=falso;          //Error de compilación
boolean otro=0;             //Error de compilación
boolean mayor=(2>1);        //Ok
```



Centro de Enseñanza  
Gregorio Fernández

# Tipos de datos

## IV – Caracteres

- Java almacena los caracteres con 2 bytes.
- Podemos almacenar en un dato de tipo **char** cualquier carácter del código Unicode ([www.unicode.org](http://www.unicode.org)), que incluye caracteres de distintas lenguas.
- Los 128 primeros caracteres del Unicode son los del carácter ASCII.
- Internamente los caracteres almacenan su código Unicode asociado. Así, podemos asignar a un dato de tipo char un valor numérico.
- Las dos declaraciones siguientes de variables son iguales:

```
char letra;  
letra='A'; //Los caracteres van entre comillas simples  
letra=65;  //El código Unicode de la A es el 65  
System.out.println(letra); //Imprime 'A'
```



Centro de Enseñanza  
Gregorio Fernández



# Tipos de datos

## IV – Caracteres

- Como en realidad los datos de tipo **char** son números, podemos realizar operaciones aritméticas con ellos. Por ejemplo, los caracteres ASCII en mayúsculas y minúsculas difieren en 32:

ASCII('A') = 65       $\Rightarrow$       ASCII('a') = 97

ASCII('B') = 66       $\Rightarrow$       ASCII('b') = 98

- Por tanto, podríamos obtener a partir de las letras mayúsculas las correspondientes letras minúsculas de la siguiente forma:

```
char letra_A = 'A';  
char letra_a = letra_A + 32;  
System.out.println(letra_a)      //Imprime 'a'
```



Centro de Enseñanza  
Gregorio Fernández

# Tipos de datos

## IV – Caracteres

- También hay una serie de caracteres especiales llamados **secuencias de escape**, que van precedidos por el símbolo `\`:

Secuencia de escape	Significado
<code>\b</code>	Retroceso
<code>\t</code>	Tabulador
<code>\n</code>	Nueva línea
<code>\r</code>	Retorno de carro
<code>\\</code>	Backslash
<code>\"</code>	Dobles comillas
<code>\'</code>	Dobles comillas
<code>\udddd</code>	Las cuatro letras d, son un número en hexadecimal. Representa el carácter Unicode cuyo código es representado por las dddd.





# Tipos de datos

## V – El tipo void

- No se pueden definir variables de tipo **void**.
- Sólo se puede utilizar como tipo de retorno de un método para indicar que no devuelve nada.

```
void v; //Error  
  
void public static entradaDatos() {  
    ...  
}
```



Centro de Enseñanza  
Gregorio Fernández

# Constantes

- Una **constante** es un valor que no cambia durante la ejecución de un programa.
- Podemos considerar dos tipos de constantes en Java:
  - **Constantes literales.**
  - **Constantes declaradas.**



Centro de Enseñanza  
Gregorio Fernández



# Constantes

## I – Constantes literales

- Se introducen directamente en el texto del programa.
- Pueden ser:
  - a) **Constantes enteras.** Por ejemplo: 123456
  - b) **Constantes reales.** Ejemplos: 82.347, .63, 0.63, 83., 83.0
  - c) **Constantes carácter.** Caracteres particulares del conjunto de caracteres Unicode.

Hay diferentes formas de especificar las constantes carácter en Java:

- Entre comillas simples: 'A', 'b'
- En octal o hexadecimal (indicando su código Unicode en octal o hexadecimal):

```
ASCII(Σ) = 22810 = 3448
```

```
char sigma = '\344'; //octal
```

```
char sigma = '\u00E4' //hexadecimal
```

- Caracteres especiales. Se indican escapándolos con un backslahs '\.



Centro de Enseñanza  
Gregorio Fernández

# Constantes

## I – Constantes literales

- Recordemos que se pueden realizar operaciones aritméticas sobre datos de tipo **char**.
- Analicemos el siguiente código:

```
char c;  
c='T' + 5; //Error
```

- ¿Solución?
- El proceso inverso no presenta problemas:

```
int j=0;  
j='p'; //ASCII('p')=80  
System.out.println(j); //Imprimirá 80
```



Centro de Enseñanza  
Gregorio Fernández



# Constantes

## I – Constantes literales

a) **Constantes de cadena.** Secuencia de caracteres almacenados entre comillas dobles.

Ejemplos:

`""` ⇒ cadena vacía

`" "` ⇒ espacio en blanco

`"1"` ⇒ cadena con el 1 (no es el número 1)

Las constantes de cadena se concatenan con el operador "+":

`"ABC" + "DEF" + "GHI" ⇔ "ABCDEFGHI"`

Podemos incluir en las cadenas de caracteres secuencias de escape: `"Día de Navidad:\n\t25 de Diciembre"`

que da lugar a: Día de Navidad:

25 de Diciembre

Para concatenar cadenas en varias líneas lo haremos de la siguiente forma:

`"cadena de " +`  
`"varias " +`  
`"líneas"`

gf

Centro de Enseñanza  
Gregorio Fernández

# Constantes

## II – Constantes declaradas

- Se declaran con el modificador **final**.

```
final byte MESES=12;  
final char ARROBA='@';  
final String MES7="Julio";  
final boolean ENCONTRADO=true;
```

- Si intentamos modificarlas dará lugar a un error de compilación:

```
MESES=13; //Error de compilación
```

- Es recomendable poner los nombres de las constantes en mayúsculas.





# Variables

- Las **variables** son los contenedores de los datos que utiliza un programa.
- Cada variable ocupa un espacio en la memoria RAM del ordenador para almacenar un dato determinado.
- Tienen un nombre (un **identificador**).



Centro de Enseñanza  
Gregorio Fernández

# Variables

## I – Declaración

- Antes de poder utilizar una variable se debe declarar.
- En Java las variables se pueden declarar en cualquier parte del programa.

- Sintaxis: **tipo nombre\_variable = valor\_inicial**

- Ejemplos: 

```
int dias=365;
boolean decision;
decision=false;
```

- También se puede declarar varias variables en la misma línea:

```
int dias=365, semanas;
```

- Al declarar una variable se puede incluso utilizar una expresión:

```
int a=13, b=18;
int c=a+b;
```



Centro de Enseñanza  
Gregorio Fernández



# Variables

## I – Ámbito

- El ámbito de una variable se refiere a la **zona del programa** donde puede utilizarse la variable, donde es “visible”.
- Según el ámbito tenemos **variables locales** y **variables de clase** (globales).



Centro de Enseñanza  
Gregorio Fernández

# Entrada y Salida estándar

- **java.lang.System** es una clase que poseen multitud de pequeñas clases relacionadas con la configuración del sistema.
- Entre ellas están:
  - **Clase in.** Representa la entrada estándar (normalmente el teclado)
  - **Clase out.** Representa a la salida estándar (normalmente la pantalla).



Centro de Enseñanza  
Gregorio Fernández



# Salida estándar

- Algunos de los métodos más importantes de **System.out** para mostrar datos por pantalla son:
  - **print()**: imprime en pantalla una cadena de caracteres.
  - **println()**: print() + salto de línea.



Centro de Enseñanza  
Gregorio Fernández

# Entrada estándar

## I – System.in

- Su método principal es **read()** que lee un carácter del teclado y lo almacena en un dato de tipo **int**. De forma que para pasarlo a **char** tengo que hacer un cast (int  $\Rightarrow$  char).

```
char c;  
c=(char)System.in.read();
```

- Normalmente no se quiere leer sólo un carácter sino una línea entera, pero la clase *System* a la que pertenece el objeto **in** no tiene ningún método para ello, pero si la clase *BufferedReader*, que dispone del método **readLine()** que lee una línea de texto en forma de String.
- Para poder hacer esto es necesario crear un objeto *BufferedReader* a partir de uno *BufferedInputStream*. Pero no se puede hacer directamente, hay que realizar un paso intermedio a través de un objeto *InputStreamReader*.

Centro de Enseñanza  
Gregorio Fernández



# Entrada estándar

## I – System.in

- Es decir, habría que hacer lo siguiente:

```
BufferedInputStream fuente= System.in;  
InputStreamReader flujo=new InputStreamReader(fuente);  
BufferedReader teclado= new BufferedReader(flujo);
```

- Normalmente se hace en un solo paso:

```
BufferedReader teclado = new BufferedReader (  
    new InputStreamReader(System.in));
```

- Una vez hecho esto ya podría leer líneas desde el teclado:

```
teclado.readLine()
```



# Entrada estándar

## I – System.in

- Hay que tener en cuenta que el método **readLine()** (como todos los métodos de lectura) puede provocar excepciones de tipo *IOException* por lo que habrá que capturar dicho error:

```
String texto=""; //almacena las cadenas introducidas por teclado

try {

    //Obtención del objeto BufferedReader
    BufferedReader teclado=new BufferedReader(
        new InputStreamReader(System.in));

    //Leemos una línea desde el teclado
    texto=teclado.readLine();

} catch(IOException e){
    System.out.println("Error");
}

System.out.println(texto);
```



Centro de Enseñanza  
Gregorio Fernández



# Entrada estándar

## I – System.in

- Lo primero que tenemos que hacer para poder utilizar los métodos indicados es importar a nuestros programas el paquete **java.io.\***.

```
import java.io.*
```

- Además, siempre que utilicemos métodos para leer datos desde el teclado, debemos encerrar al código en un bloque **try-catch** para tratar los errores de tipo *IOException*.



Centro de Enseñanza  
Gregorio Fernández

# Entrada estándar

## II – Scanner

- La clase Scanner facilita la entrada de datos en nuestros programas.
- Se encuentra dentro del paquete `java.util`, por lo que para poder utilizarla habrá que importarla.

```
import java.util.Scanner;
```

- En primer lugar, lo que hay que hacer es crear un objeto Scanner asociado al teclado:

```
Scanner sc = new Scanner(System.in);
```

- Una vez hecho esto podemos leer datos por teclado. Para ello usaremos los métodos `nextX()` donde `X` indica en tipo de datos a leer.
- Ejemplos:

```
System.out.print("Introduzca un número entero: ");  
int i = sc.nextInt();  
System.out.print("Introduzca un número real: ");  
double d = sc.nextDouble();  
System.out.print("Introduzca un texto: ");  
String s = sc.nextLine();
```



Centro de Enseñanza  
Gregorio Fernández



# Entrada estándar

## III – Línea de comandos

- Toda aplicación Java debe comenzar con la llamada al método **main()** de la clase principal.
- El método **main()** es opcional, excepto para la clase principal.
- La JVM al ejecutar un programa lo que hace en primer lugar es llamar al método **main()**.
- Podemos pasar datos (argumentos) al programa desde la línea de comandos.
- Si recordamos la forma del **main()**:

```
public static void main(String [] args)
```

- Como parámetro del main tenemos una estructura de datos (array) llamada *args* que recoge los datos pasados por línea de comandos al ejecutar el programa y los almacena como cadenas (String).

- En el ejemplo: 

```
java prueba 1 2 3
```

```
args[0] almacena el 1er argumento ⇨ 1
```

```
args[1] almacena el 2o argumento ⇨ 2
```

```
args[2] almacena el 3er argumento ⇨ 3
```

- De tal forma, que si dentro del programa quiero utilizar los valores pasados como argumentos al programa, no tengo más que acceder al elemento correspondiente del array **args** indicado la posición del elemento a recuperar.



Centro de Enseñanza  
Gregorio Fernández

# Operadores y expresiones

- **Expresión:** secuencia de operadores y operandos que realizan un cálculo y devuelven un valor.
- Ejemplo:  $3 + x + 5 * z$
- **Operadores:** pueden ser **binarios**, actúan sobre dos operandos, como la suma (+) o **unarios**, que actúan sobre sólo un operando, por ejemplo el operador lógico NOT (!).
- Los tipos de operadores que vamos a ver son:
  - ☐ Operador de asignación
  - ☐ Operadores aritméticos
  - ☐ Operadores relacionales
  - ☐ Operadores lógicos



Centro de Enseñanza  
Gregorio Fernández



# Operadores y expresiones

## I – Operador de asignación

- Sirve para dar valor a las variables.
- Se representa por el signo igual “=”.
- Es un operador binario.
- Sintaxis: **variableA=variableB;**
- Hay otros 5 operadores de asignación que se suelen emplear como notación abreviada:

Operador	Operación aritmética	Ejemplo	Equivalencia	Tipo
+=	Suma	m += n	m = m + n	Binario
-=	Resta	m -= n	m = m - n	
*=	Multiplicación	m *= n	m = m * n	
/=	Cociente	m /= n	m = m / n	
%=	Resto	m %= n	m = m % n	



# Operadores y expresiones

## II – Operadores aritméticos

- Permiten realizar operaciones aritméticas básicas.
- Hay dos tipos de operadores aritméticos:
- Operadores aritméticos **binarios**:

Operador	Operación aritmética	Ejemplo
+	Suma	3+2
-	Resta	1-4
*	Multiplicación	4.5*2.1
/	Cociente	3/2
%	Resto	4%3

- Operadores aritméticos **unarios**:

Operador	Operación aritmética	Ejemplo
+	Mas	+2
-	Menos	-4
++	Incremento (pre y post)	4++ (post) ++4 (pre)
--	Decremento (pre y post)	4-- (post) --4 (pre)



# Operadores y expresiones

## II – Operadores aritméticos

- ¿Cuál es la salida del siguiente código?

```
int a=2;
int b=3;
int z=0;
z=a++;
System.out.println(z);
System.out.println(a);

z=++a;
System.out.println(z);
System.out.println(a);

z=b--;
System.out.println(z);
System.out.println(b);

z=--b;
System.out.println(z);
System.out.println(b);
```



Centro de Enseñanza  
Gregorio Fernández

# Operadores y expresiones

## II – Operadores aritméticos

- ¿Y la de éste?

```
int n=5,t;  
t=++n*--n;  
System.out.println("n= " + n + ",t = " + t);  
System.out.println(++n + " " + ++n + " " + ++n);
```

```
int m=5,u;  
u=m++*m--;  
System.out.println("m= " + m + ",u = " + u);  
System.out.println(m++ + " " + m++ + " " + m++);
```



Centro de Enseñanza  
Gregorio Fernández



# Operadores y expresiones

## II – Operadores relacionales

- Permiten realizar comparaciones.
- El resultado devuelto es de tipo booleano (true o false).
- Se emplean normalmente para sentencias de selección y bucles.
- Son binarios.
- Tienen menor prioridad que los operadores aritméticos.

Operador	Ejemplo	Resultado true si...
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 es igual que op2
!=	op1 != op2	op1 es distinto que op2

# Operadores y expresiones

## II – Operadores relacionales

- Ejemplos:

```
boolean mayorDeEdad, menorDeEdad;  
int edad = 21;  
mayorDeEdad = (edad >= 18);  
menorDeEdad = (!mayorDeEdad);
```



Centro de Enseñanza  
Gregorio Fernández



# Operadores y expresiones

## III – Operadores lógicos

- Operan sobre operandos de tipo boolean, ya sean variables o expresiones relacionales.
- Devuelven un valor booleano.

Operador	Nombre	Ejemplo
&&	AND (Y lógico)	op1 && op2
	OR (O lógico)	op1    op2
!	NOT (Negación)	!op1
^	XOR (O exclusivo)	op1 ^ op2

- Tablas de verdad:

AND			OR			NOT		XOR		
A/B	T	F	A/B	T	F	A	!A	A/B	T	F
T	T	F	T	T	T	T	F	T	F	T
F	F	F	F	T	F	F	T	F	T	F

*gf*

Centro de Enseñanza  
Gregorio Fernández

# Operadores y expresiones

## III – Operadores lógicos

- Ejemplos:

```
boolean carnetConducir=true;
int edad=20;
boolean puedeConducir = (edad>=18) && carnetConducir;
boolean nieva = true, llueve=false, graniza=false;
boolean malTiempo = nieva || llueve || graniza;
```

- Si la variable “horas” vale 150 y la variable “ventas” vale 60, ¿cuál será la salida del siguiente código? :

```
if ((ventas>50) || (horas<100)) {
    System.out.println("1");
}

if ((ventas>50) && (horas<100)) {
    System.out.println("2");
}

if (!(ventas>50) || (horas<100)) {
    System.out.println("3");
}

if (!((ventas>50) || (horas<100))) {
    System.out.println("4");
}
```



Centro de Enseñanza  
Gregorio Fernández



# Operadores y expresiones

## III – Operadores lógicos

- Se evalúan de izquierda a derecha, por lo que si uno de los operandos determina que la expresión tomará un valor, Java no sigue evaluando el resto de operandos.
- Es lo que se llama **evaluación en cortocircuito**.
- Por ejemplo, el siguiente código no dará error por división por cero:

```
int x=0;  
if (x==0 || (7<1/x)) {  
    System.out.println("¿Esto se imprime?");  
}
```

- Este comportamiento mejora el rendimiento de los programas.
- Si se desea que se evalúen todas las condiciones, se pueden utilizar las versiones “simples” de los operadores AND y OR, es decir, & y |.



Centro de Enseñanza  
Gregorio Fernández

# Operadores y expresiones

## IV – Operador ?

- Devuelve un valor u otro dependiendo de que se cumpla una condición o no.
- Sirve para reemplazar a la sentencia if-else en algunas ocasiones.
- Operador ternario.
- Sintaxis: **expresión\_booleana ? valor1 : valor2**
- Funcionamiento: se evalúa la expresión, si el resultado es verdadero se devuelve el valor1, sino, se devuelve el valor2.
- Como vemos el operador ? siempre devuelve un valor, y ambos deben ser del mismo tipo.
- Ejemplo: `paga=(edad>18)? 6000:3000;`
- Error típico: `(edad>18)? paga=6000: paga=3000;`



Centro de Enseñanza  
Gregorio Fernández



# Precedencia y asociatividad

- **Precedencia:** el orden en que se evalúan las operaciones en expresiones con más de un operador.
- **Asociatividad:** orden de evaluación de las operaciones de la misma prioridad.

Prioridad	Operadores	Asociatividad
1	() [] .	
2	++ -- (pre)	D-I
3	++ -- (post)	
4	+ - ~ !	D-I
5	* / %	
6	+ - (binarios)	
7	< > <= >=	
8	== !=	
9	&	
10	^	
11		
12	&&	
13		
14	? :	D-I
15	= += -= *= /= %= &=  =	D-I

# Conversiones de tipos

- Java puede realizar dos tipos de conversiones de datos, a fin de realizar operaciones de asignación sin que se produzcan errores de incompatibilidad de tipos.
  - **Conversiones implícitas** (automáticas)
  - **Conversiones explícitas**



Centro de Enseñanza  
Gregorio Fernández



# Conversiones de tipos

## I – Implícitas

- a) Conversiones numéricas siempre que no haya pérdida de información.

```
int i=12;  
double x=4.0;  
x=i; //conversión implícita int->double
```

- b) Cuando se combinan tipos en expresiones.

```
x = x + i; //i (int) se convierte a double
```

- c) Cuando se pasan argumentos a métodos siempre que no se pierda precisión.



Centro de Enseñanza  
Gregorio Fernández

# Conversiones de tipos

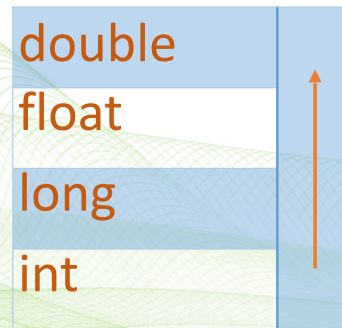
## I – Implícitas

- Algunas **reglas** de las conversiones implícitas:

- Los operadores de tipo char, byte o short se transforman en **int**.
- Las constantes enteras se consideran de tipo int. Las reales de tipo double.

Así `float f=1.0` es erróneo, debe escribirse `float f=1.0F` ó `float f=(float)1.0`.

- Si hay mezcla de tipos en una expresión, la conversión se realizará de la siguiente forma:



Centro de Enseñanza  
Gregorio Fernández



# Conversiones de tipos

## I – Implícitas

- Ejemplo:

```
long l;  
int i;  
double d;  
float f;  
  
d= f*i+l*2;  
i=l*5;  
f=5.0*d+7.0;
```



Centro de Enseñanza  
Gregorio Fernández

# Conversiones de tipos

## II – Explícitas

- Podemos **forzar** la conversión de tipos mediante el **casting**.
- Para ello se emplea el operador “()”.
- Dentro de los paréntesis se pone el tipo de dato al que se quiere convertir.
- Ejemplo:

```
int i;  
float f=3.45F;  
i=(int)f; //se pierden los decimales
```

- Existen limitaciones. Por ejemplo, no se puede hacer casting a boolean, ni a cadenas de caracteres (String).



Centro de Enseñanza  
Gregorio Fernández





# Actividades



## Actividades Tema2



Centro de Enseñanza  
Gregorio Fernández