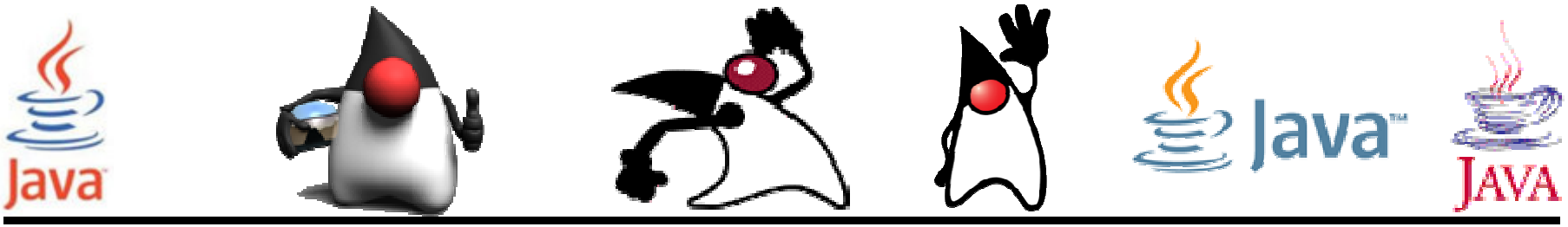


Programación básica en Java

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Describir las partes básicas de un programa en Java.
- Presentar los elementos lexicales del lenguaje de programación Java.
- Diferenciar entre literales, tipos de datos primitivos, identificadores, tipos de variables, y operadores.
- Desarrollar programas simples válidos.

Índice

- Análisis del primer programa Java
- Comentarios
- Elementos lexicales
- Tipos de datos primitivos
- Variables
- Impresión de variables
- Operadores y expresiones

Análisis del primer programa Java

```
/**
 * Ejemplo HolaMundo
 * Imprime el mensaje "Hola, Mundo!"
 */
public class HolaMundo {
  public static void main(String[] args) {
    // Imprime el mensaje "Hola, Mundo!"
    System.out.println("Hola, Mundo!");
  }
}
```

Análisis del primer programa Java

```
/**
 * Ejemplo HolaMundo
 * Imprime el mensaje "Hola, Mundo!"
 */
```

- Las cuatro primeras líneas del código fuente son comentarios.
- Un comentario:
  - sirve para documentar parte del código.
  - no es parte del programa, pero se usa con el propósito de documentarlo.
  - es una buena práctica de programación añadir comentarios al código.

Análisis del primer programa Java

```
/**
 * Ejemplo HolaMundo
 * Imprime el mensaje "Hola, Mundo!"
 */
public class HolaMundo {
```

- Indica el nombre de la clase que es HolaMundo.
- En Java, todo el código debe ser colocado dentro de una declaración de clase.
- El especificador de acceso a la clase public, indica que la clase es accesible desde otras clases de otros paquetes (los paquetes son colecciones de clases).

Análisis del primer programa Java

```
/**
 * Ejemplo HolaMundo
 * Imprime el mensaje "Hola, Mundo!"
 */
public class HolaMundo {
```

- La llave indica el inicio de un bloque.
- En este código la llave se ha colocado en la misma línea que la declaración de clase, pero puede ponerse en la siguiente línea.

Análisis del primer programa Java

```
public class HolaMundo {
  public static void main(String[] args) {
```

- Indica el nombre de un método en HolaMundo que es el método principal.
- El método main es el punto de inicio de un programa Java.
- Todos los programas en Java, excepto los applets, empiezan con el método main.
- Es necesario colocar toda la declaración exactamente.

Análisis del primer programa Java

```
public class HolaMundo {
  public static void main(String[] args) {
    // Imprime el mensaje "Hola, Mundo!"
```

- Otro tipo de comentario Java.

Análisis del primer programa Java

```
public class HolaMundo {
  public static void main(String[] args) {
    // Imprime el mensaje "Hola, Mundo!"
    System.out.println("Hola, Mundo!");
```

- La instrucción System.out.println() imprime el texto especificado dentro de los paréntesis en la pantalla .
- System.out se refiere a la salida estándar.
- El método println() imprime su argumento como una cadena de caracteres con un salto de línea final.
- Toda instrucción termina con un ;

Análisis del primer programa Java

```
/**
 * Ejemplo HolaMundo
 * Imprime el mensaje "Hola, Mundo!"
 */
public class HolaMundo {
  public static void main(String[] args) {
    // Imprime el mensaje "Hola, Mundo!"
    System.out.println("Hola, Mundo!");
  }
}
```

- Las dos últimas llaves se usan para cerrar el bloque correspondiente al método main y la clase principal respectivamente.

Directivas de programación

- Los ficheros de programas Java siempre deben terminar con la extensión .java
- Los nombres de los ficheros deben coincidir con el nombre de la clase pública. Por ejemplo, si la clase pública es HolaMundo, se debe guardar en un fichero HolaMundo.java
- Se deben introducir comentarios en el código explicando lo que hace una clase o método.

Comentarios en Java

- Los comentarios son notas introducidas en el código con el propósito de documentación.
- El texto de los comentarios no es parte del programa y no afecta el flujo de ejecución del programa.
- Hay tres tipos de comentarios en Java:
  - Comentario estilo C o de párrafo.
  - Comentario estilo C++ o de línea.
  - Comentario especial Javadoc.

Comentarios en Java

- *Comentario estilo C o de párrafo:*
  - Se les llama también multilínea
  - Todo texto encerrado entre /* y */ que puede estar en una o varias líneas se trata como comentario.
  - Ejemplo:

```
/* este es un ejemplo de un
   comentario estilo C o multilinea */
```

Comentarios en Java

- *Comentario estilo C++ o de línea:*
  - Empiezan con // y se extienden hasta el final de la línea.
  - Todo el texto que sigue a // se trata como comentario.
  - Ejemplo:

```
// este es un comentario estilo C++ o de linea
```

Comentarios en Java

- *Comentario documentación Java o Javadoc:*
  - Los comentarios Javadoc se usan para generar la documentación en HTML a partir de los programas Java. Se utilizan para documentar clases, campos y métodos.
  - Se pueden crear comentarios Javadoc empezando la línea con /** y finalizando con */. De la misma forma que los comentarios estilo C se pueden extender por varias líneas. Puede contener etiquetas para añadir información a los comentarios
  - Se usa la utilidad de generación automática de documentación llamada ***javadoc.***

Javadoc

- Las etiquetas más utilizadas son:

Tag	Descripción
@author	Nombre del desarrollador.
@deprecated	Indica que el método o clase es obsoleto y que no se recomienda su uso.
@param	Definición de un parámetro de un método.
@return	Describe el valor devuelto de un método.
@see	Asocia con otro método o clase.
@since	Fecha o versión en que apareció por primera vez la característica.
@throws	Excepción lanzada por el método
@version	Versión del método o clase.

Javadoc - ejemplo

```
/**
 * Clase para representar círculos sobre el plano.
 * Un círculo se define por su radio y las
 * coordenadas de su  centro.
 * @version 1.2, 2/09/10
 * @author Pedro Corcuera
 */
public class Circulo {
   protected double x,y; // coordenadas del centro
   protected double r;  // radio del círculo
  /**
   * Crea un círculo a partir de su origen y radio.
   * @param x Coordenada x del centro del círculo.
   * @param y Coordenada y del centro del círculo.
   * @param r Radio del círculo. (>= 0).
   */
    public Circulo(double x, double y, double r) {
        this.x=x; this.y = y; this.r = r;
    }
```

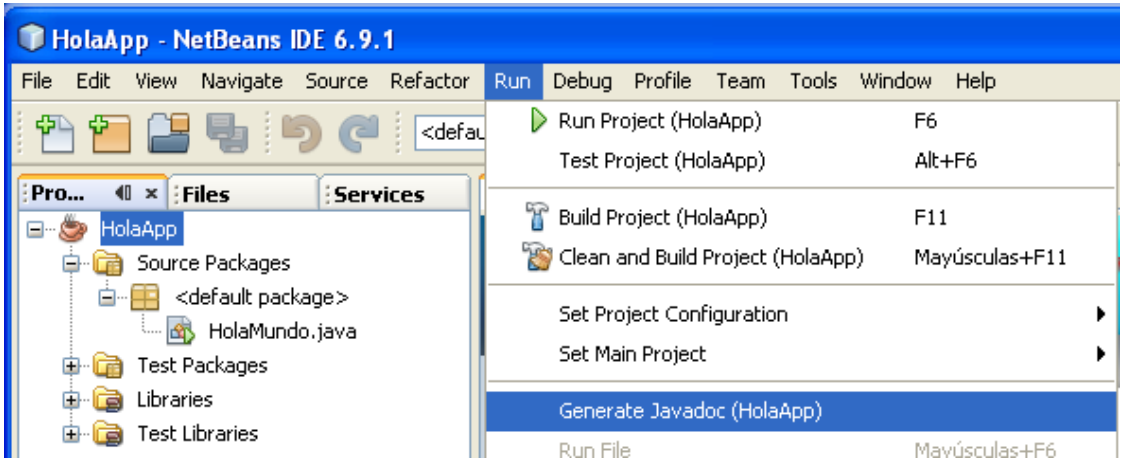
Javadoc - ejemplo

```
    /**
     * Cálculo del área de este círculo.
     * @return El área (mayor o igual que 0) del círculo.
     */
     public double area() {
        return Math.PI*r*r;
     }

     /**
      * Indica si un punto está dentro del círculo.
      * @param px componente x del punto
      * @param py componente y del punto
      * @return true si el punto está dentro del círculo o false en otro caso.
      */
     public boolean contiene(double px, double py) {
        /* Calculamos la distancia de (px,py) al centro del círculo (x,y),
           que se obtiene como raíz cuadrada de (px-x)^2+(py-y)^2 */
        double d = Math.sqrt((px-x)*(px-x)+(py-y)*(py-y));

        // el círculo contiene el punto si d es menor o igual al radio
        return  d <= r;
     }
}
```

Javadoc

- Para generar la documentación (formato HTML) se puede utilizar:
  - Modo comando:

    > javadoc Circulo.java
  - La mayoría de entornos de programación disponen de un menú para ejecutar javadoc. Por ejemplo en NetBeans:



Instrucciones Java

- Instrucción
  - una o más líneas de código terminada por un punto y coma (;).
  - Ejemplo:

```
System.out.println("Hola, Mundo!");
```

Bloques Java

- Bloque
  - es una o más instrucciones encerradas entre llaves ({ }) que agrupa las instrucciones como una unidad.
  - las instrucciones de bloque se pueden anidar indefinidamente.
  - se permite cualquier cantidad de espacios en blanco.
  - Ejemplo:

```
public static void main(String[] args) {
   // Imprime el mensaje "Hola, Mundo!"
   System.out.println("Hola, Mundo!");
}
```

Directivas de programación

- En la creación de bloques, se puede colocar la llave de apertura en línea con la instrucción. Ejemplo:

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

- o se puede colocar la llave en la siguiente línea:

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

- Se deben *indentar* las instrucciones dentro de un bloque para mostrar su nivel de profundidad.

```
public static void main(String[] args) {
   System.out.println("Hola, Mundo!");
}
```

Elementos lexicales – Conjunto de caracteres

- Los programas en Java se escriben en ***Unicode***.
- Unicode es un estándar internacional de conjuntos de caracteres (16 bits) que contiene los códigos de caracteres de la mayoría de lenguajes utilizados en el mundo.
- El conjunto de caracteres ASCII (7 bits) equivale a los primeros 128 caracteres de Unicode (ISO-8859-1).

Elementos lexicales - Identificadores

- Los identificadores se usan para nombrar clases, métodos, variables y todo lo que requiera un nombre.
- Un identificador en Java empieza con una letra (nunca un dígito!) y puede estar seguido de letras y dígitos. Debe ser diferente a las *palabras reservadas* del lenguaje.
- En Java *letras* incluye los caracteres en los alfabetos de todos los lenguajes de Unicode, el guión bajo (_) y el signo dólar ($).
- Java distingue entre mayúsculas y minúsculas.

Directivas de programación

- Para nombres de las clases poner en mayúscula la primera letra del nombre de la clase. Ejemplo:

  `EsteEsUnEjemploDeNombreDeClase`

- Para nombres de métodos y variables poner en minúscula la primera letra del nombre. Ejemplo:

  `esteEsUnEjemploDeNombreDeMetodo`

- En caso de identificadores multipalabra poner en mayúscula sólo la primera letra de cada palabra.
- Evitar usar guión bajo al inicio de un identificador.

Palabras reservadas en Java

- Las palabras reservadas son identificadores predefinidos reservados por Java para un propósito específico.
- No se puede usar las palabras reservadas como nombres de variables, clases, métodos,…etc.

Palabras reservadas en Java

abstract	double	instanceof	static
assert	else	int	strictfp
boolean	enum	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto*	protected	transient
const*	if	public	try
continue	implements	return	void
default	import	short	volatile
do			while

* const y goto no se usan

true, false y null son literales

Literales en Java

- Literales son símbolos que no cambian – son constantes.
- Los diferentes tipos de literales en Java son:
  - Literales Booleanos (boolean)
  - Literales Enteros (integer)
  - Literales Punto Flotantes (floating point)
  - Literales Caracter (character)
  - Literales de Cadenas de Caracteres (String)

Literales booleanos

- Los literales tipo boolean consisten de dos valores: **`true`** y **`false`**.

Literales enteros

- Los literales enteros pueden escribirse en tres formatos:
  - decimal (base 10): empiezan con un dígito decimal diferente a cero seguido por dígitos decimales (p.e. 30).
  - hexadecimal (base 16): empiezan con `0x` o `0X`, seguido de dígitos hexadecimales (p.e. 0x1E y 0XC1E).
  - octal (base 8): empieza con `0`, seguido de dígitos octales (p.e. 036).

Literales punto flotante

- Los literales en punto flotante representan decimales con parte decimal.
- Se escriben con un número decimal con una parte exponencial opcional (notación científica):

```
digitos.[digitos][(e|E)Enteroconsigno]
```

- Ejemplos:
  - 3.1416 314.16E-2
  - 583.45 5.8345e2

Literales caracter

- Los literales caracter representan caracteres Unicode (16 bits). Se escriben entre comillas simples ' '
- Los caracteres ASCII se pueden escribir directamente, p.e. 'c'.
- Los caracteres no ASCII se pueden escribir con códigos hexadecimales u octales.
  - En código hexadecimal, \u es seguido por cuatro dígitos hexadecimales. Ejemplos: '\u00E6' '\u5496' '\u03b1'
  - En código octal, \ es seguido por uno o tres dígitos octales. Ejemplos: '\377' '\040'

Literales caracter – secuencias de escape

- Hay caracteres especiales, conocidos como secuencias de escape (*escape sequences*), que se escriben:

Descripción	*Escape Sequence*	*Unicode*
Backspace	\b	\u0008
Tab	\t	\u0009
Linefeed	\n	\u000A
Carriage return	\r	\u000D
Form feed	\f	\u000C
Backslash	\\	\u005C
Single Quote	\'	\u0027
Double Quote	\"	\u0022

Literales Cadenas de caracteres

- Los literales cadenas de caracteres o String representa secuencias de caracteres, incluyendo las secuencias de escape, encerradas por comillas dobles.
- Ejemplos:

Literal String	Valor
"Un literal cadena"	Un literal cadena
"\"Una nota \" "	"Una nota"
"\u00E6 \u0027"	æ '

Tipos de datos primitivos

- El lenguaje de programación Java define ocho tipos de datos primitivos:
  - boolean (para lógica)
  - char (para textos)
  - byte
  - short
  - int
  - long (integral)
  - double
  - float (punto flotante)

Tipos de datos primitivos - boolean

- Un tipo de dato booleano representa dos estados: true y false.
- Un ejemplo es:

```
boolean resultado = true;
```

- El ejemplo anterior declara una variable resultado como de tipo boolean y le asigna el valor de true.
- No es compatible con tipos enteros (como en C).
- Las condiciones de las instrucciones de control (p.e. if, while) se esperan que sean de tipo boolean.

Tipos de datos primitivos - char

- Un tipo de dato char representa un caracter simple Unicode (65536 símbolos).
- Debe tener el literal encerrado entre comillas simples ' '.
- Ejemplos:

  'a' \\ letra a

  '\t' \\ tabulador

  '\'' \\ comilla simple

  '\"' \\ comilla doble

Tipos de datos primitivos - String

- String **no** es un tipo de dato primitivo, es una **clase**.
- Un String o cadena de caracteres representa un tipo de dato que contiene varios caracteres.
- Tiene la parte literal encerrada entre comillas dobles "cadena"
- Ejemplo:

  String mensaje = "Hola Mundo!"

Tipos de datos primitivos - Integer

- Los tipos de datos integrales o enteros (**byte**, **short**, **int** y **long**) en Java representan enteros de diferentes tamaños.
- Se puede usar tres formas: decimal, octal o hexadecimal para su representación.
- El tipo de dato integral por defecto es int.
- Se puede definir el valor largo añadiendo la letra l o L.
- Ejemplos:

  2 077 0xBAC 10L

Tipos de datos primitivos – Integer

- Tamaño y rango de enteros:

Tipo	Tamaño (bit)	Valor Mínimo	Valor Máximo
byte	8	-128 (-2^7)	127 (2^7-1)
short	16	-32768 (-2^{15})	32767 (2^{15}-1)
int	32	-2147483648 (-2^{31})	2147483647 (2^{31}-1)
long	64	-9223372036854775808 (-2^{63})	9223372036854775807 (2^{63}-1)

Tipos de datos primitivos – Punto flotante

- Los tipos de datos de punto flotante (**float** y **double**) en Java representan números en general.
- Los literales en punto flotante incluyen una parte decimal o uno de los siguientes:
  - E o e // valor exponencial
  - F o f // (float)
  - D o d // (double)
- El tipo de dato punto flotante por defecto es double.
- Ejemplos:

  3.1415 6.02E23 2.718F 123.4E35d

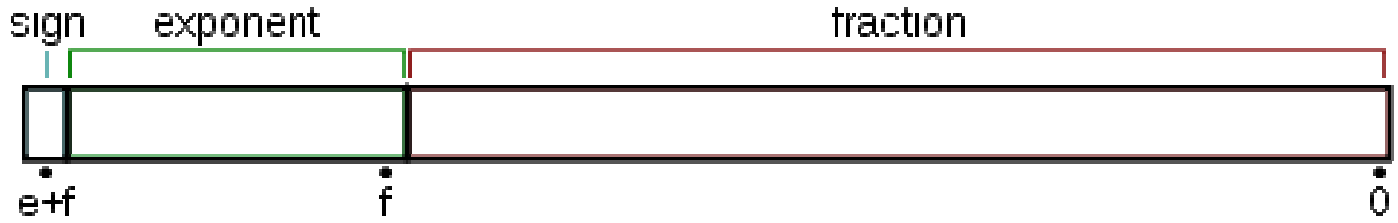
Tipos de datos primitivos – Punto flotante

- Los tipos de datos de punto flotante se almacenan según IEEE-754:
  - Signo
  - Exponente despl.
  - Mantisa

Partes de un número en punto flotante -5:

Signo	Mantisa	Base ^{exponente}
-1	5	10^{0}

- El signo usa el bit más significativo.
- El exponente emplea un byte en formato desplazado.
- La mantisa ocupa 23 bits (float) y 52 bits (double).



Tipos de datos primitivos – Punto flotante

- Tamaño y rango de tipos reales:

Tipo	Tamaño (bit)	Rango
float	32	$+ / - 3.4 \cdot 10^{38}$
double	64	$+ / - 1.8 \cdot 10^{308}$

Tipos de datos primitivos – Almacenamiento por tipo (bytes)

- **Integer Types**
  - `byte:`
  - `short:`
  - `int:`
  - `long:`
- **Floating Point Types**
  - `float:`
  - `double:`
- **Other Types**
  - `boolean:`
  - `char:`

Variables

- Una variable es una posición de memoria donde se almacena un valor (estado de un objeto).
- Tiene un nombre para facilitar el acceso.
- Una variable tiene asociado un:
  - **Tipo**: el tipo indica el tipo de valor que la variable puede almacenar.
  - **Nombre**: el nombre de la variable debe seguir las reglas para los identificadores.
- Java soporta dos clases de tipos: tipos primitivos y tipos referencia.

Variables – declaración e inicialización

- Sintaxis para declarar una variable:

  <tipo> <nombre> [= valor inicial];

  – Nota: los valores encerrados entre < > son valores requeridos, mientras que los encerrados entre [ ] son opcionales.

- Ejemplos:

```
int tanqueN1 = 6;
double tanqueVolumen = 12.0;
boolean resultado;
char opcion = 'c';
```

Variables – Directivas de programación

- Usar nombres descriptivos para las variables, empezando siempre por una letra en minúscula.
- Es recomendable inicializar las variables en el momento de la declaración. Los valores iniciales por defecto de los diferentes tipos son:

Tipo	Valor inicial por defecto
Integer	0
Floating-point	0.0
Char	\u0000
Boolean	False
Reference	null

Compatibilidad de tipos y conversión

- ***Compatibilidad de tipos***: un tipo T_1 es compatible con tipo T_2 si un valor del tipo T_1 puede aparecer donde sea que un valor del tipo T_2 se espera y viceversa.
- ***Conversión de tipos***: es la conversión de valores de un tipo a valores de otro tipo.
- ***Ampliación y estrechamiento de tipos numéricos***: convertir un tipo numérico de rango pequeño a un tipo numérico de mayor rango es ampliación. Lo opuesto es estrechamiento.

Compatibilidad de tipos y conversión

- Tamaños y rangos de tipos numéricos ordenados de menor a mayor:
  byte short int long float double
- La ampliación de tipos se realiza de manera implícita, mientras que el estrechamiento puede resultar en pérdida de precisión. Por ello, es necesario realizar un *cast* explícito (sino el compilador produce error).
- Ejemplo:

```
int i = 10; long m = 10000L; double d = Math.PI;
i = (int) m; m = i; m = (long) d; d = m;
```

Impresión simple del valor de variables

- Para imprimir el valor de una variable se puede usar cualquiera de las instrucciones:

  System.out.println()

  Añade una nueva línea al final del dato impreso

  System.out.print()

  No añade una nueva línea al final del dato impreso

Impresión simple del valor de variables - Ejemplos

- *Código 1*:

```
System.out.print("Hola");
System.out.print("Mundo");
```

Salida:

```
HolaMundo
```

- *Código 2*:

```
System.out.println("Hola");
System.out.println("Mundo");
```

Salida :

```
Hola
Mundo
```

Impresión simple del valor de variables - Ejemplo

```
public class ImprimeVariable {
   public static void main(String[] args) {
      int valor = 10;
      char x = 'A';
      System.out.println(valor);
      System.out.println("Valor de x = " + x);
   }
}
```

- *Resultado en pantalla*:

  10

  Valor de x = A

Impresión con formato de variables

- Emula la impresión con formato printf() de C.
- Sintaxis del método:
  System.out.printf (String de formato, Objectos... args)
- El String de formato puede contener especificadores de formato cuya sintaxis es:

  %[argument_index$][flags][width][.precision]conversion_char

  El carácter de conversión puede ser:

f para puntos flotantes	d para enteros
o para octales	e para notación científica
g notación punto flotante general	s para cadenas de caracteres

Tipos de formato de impresión

Format Types		
Code	Type	Example
d	Decimal integer	123
f	Fixed floating-point	12.30
e	Exponential floating-point	1.23e+1
g	General floating-point (exponential notation is used for very large or very small values)	12.3
s	String	Tax:

- Se puede incluir texto dentro de las comillas:

```
System.out.printf("Precio por litro: %10.2f", precio);
```

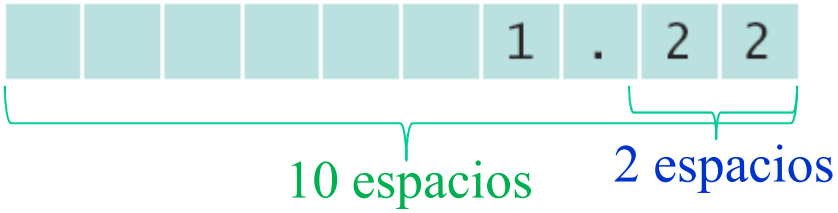
Flags para formatos de impresión

- Se puede usar flags de formatos para cambiar la manera en que el texto y los valores numéricos se muestran:

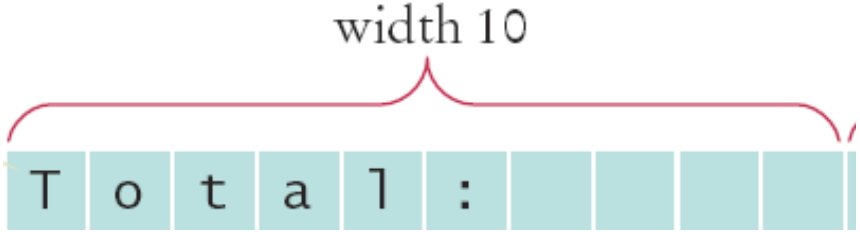
Format Flags		
Flag	Meaning	Example
-	Left alignment	1.23 followed by spaces
0	Show leading zeroes	001.23
+	Show a plus sign for positive numbers	+1.23
(	Enclose negative numbers in parentheses	(1.23)
,	Show decimal separators	12,300
^	Convert letters to uppercase	1.23E+1

Impresión con formato de variables

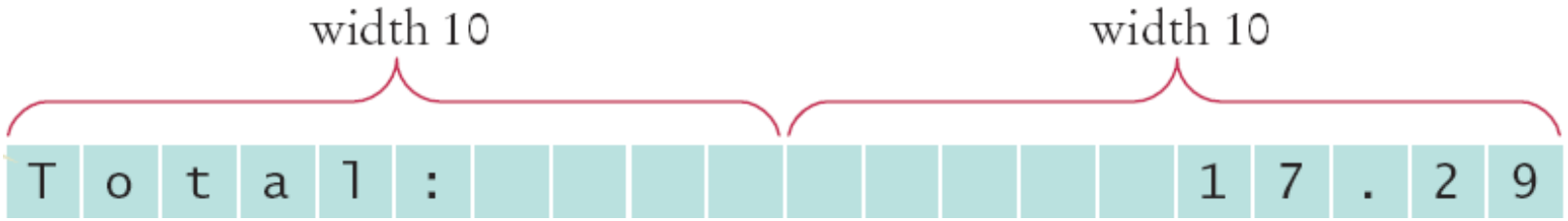
```
System.out.printf("%10.2f", precio);
```




```
System.out.printf("%-10s", "Total:");
```




```
System.out.printf("%-10s%10.2f", "Total:", precio);
```



Impresión con formato de variables - Ejemplo

```
public class PrintfDemo {
  public static void main(String[] args) {
    double q = 1.0/3.0; System.out.printf ("1.0/3.0 = %5.3f %n", q);
    System.out.printf ("1.0/3.0 = %7.5f %n", q);
    q = 1.0/2.0; System.out.printf ("1.0/2.0 = %09.3f %n", q);
    q = 1000.0/3.0; System.out.printf ("1000/3.0 = %7.2e %n", q);
    q = 3.0/4567.0; System.out.printf ("3.0/4567.0 = %7.2e %n", q);
    q = -1.0/0.0; System.out.printf ("-1.0/0.0 = %7.2e %n", q);
    q = 0.0/0.0; System.out.printf ("0.0/0.0 = %5.2e %n", q);
    System.out.printf ("pi = %5.3f, e = %5.4f %n", Math.PI, Math.E);
    double r = 1.1;
    System.out.printf ("C = 2 * %1$5.5f * %2$4.1f, "+
                       "A = %2$4.1f * %2$4.1f * %1$5.5f %n",
                       Math.PI, r);
  }
}
```

Impresión con formato de variables – Resultado del Ejemplo

- *Salida*:

  1.0/3.0 = 0,333
  1.0/3.0 = 0,33333
  1.0/2.0 = 00000,500
  1000/3.0 = 3.33e+02
  3.0/4567.0 = 6.57e-04
  -1.0/0.0 = -Infinity
  0.0/0.0 = NaN
  pi = 3.142, e = 2.7183
  C = 2 * 3.14159 * 1.1, A = 1.1 * 1.1 * 3.14159

- Enlace a opciones de printf:
  http://download-llnw.oracle.com/javase/tutorial/java/data/numberformat.html

Variables primitivas y variables referencia

- Java dispone de dos tipos de variables que se corresponden con los tipos empleados en su declaración:
  - Variables primitivas
  - Variables referencia
- Variables primitivas:
  - Variables declaradas con tipos de datos primitivos.
  - Almacenan los datos en la ubicación de la memoria asignada a la variable.

Variables primitivas y variables referencia

- Los tipos de datos de referencia son class, interface o array.
- Variables referencia:
  - Variables que almacenan la dirección de memoria de otro variable: apuntan a otra dirección de memoria donde se encuentra los datos.
  - Cuando se declara una variable de una cierta clase, se está declarando una variable referencia al objeto con esa clase.
- Las referencias en Java se implementan como punteros de 32 bits.

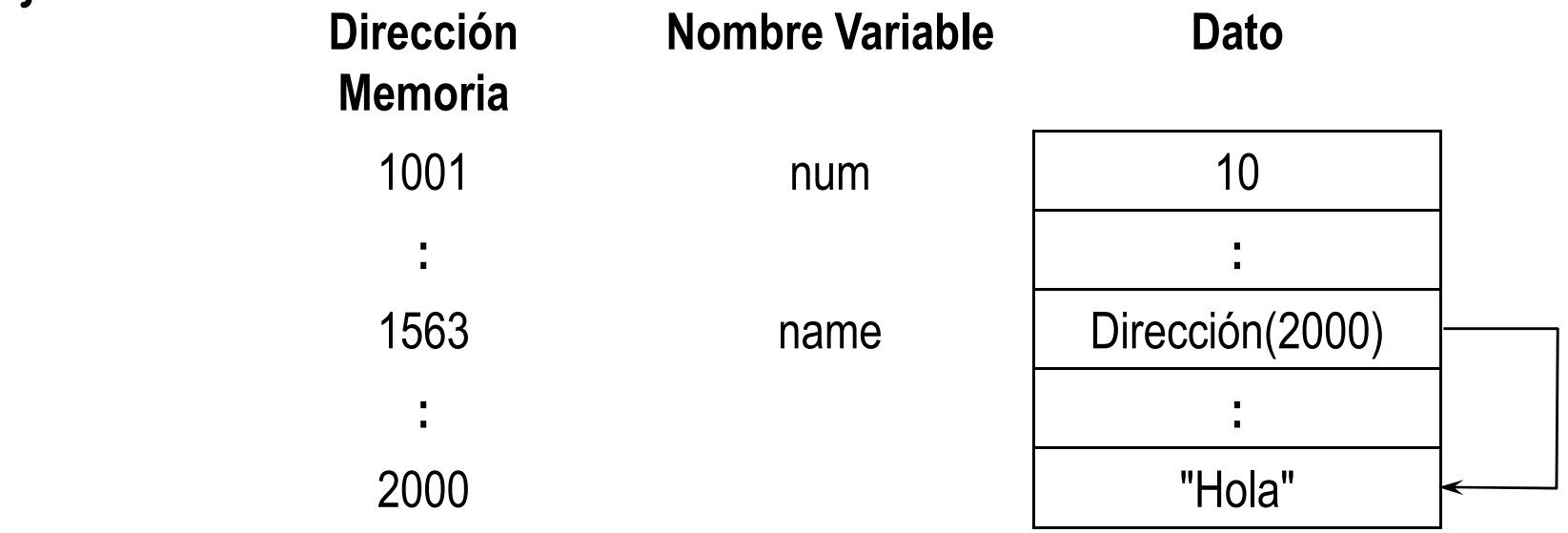
Variables referencia: diferencia con punteros C y C++

- Las variables referencia Java se diferencian de los punteros C y C++ en:

C y C++	Java
Los punteros se pueden modificar mediante cast, aritmética de punteros o asignación de valores.	Java no permite estas operaciones y prohíbe la manipulación directa de variables referencia
Los punteros hacen referencia a segmentos de memoria que se separan dinámicamente. El programador es responsable de gestionar la asignación y liberación de memoria.	Las variables referencia apuntan al espacio de memoria que es dinámicamente asignado por el garbage-collector heap. Los programadores están liberados de la responsabilidad de gestionar la memoria.

Variables primitivas y variables referencia Ejemplo

```
public class TiposVariables {
   public static void main(String[] args) {
      int num = 10;  // tipo primitivo
      String name = "Hola"; // tipo referencia
   }
}
```



Operadores y expresiones

- Java ofrece operadores similares a C, C++ y C#.
- La combinación de **operandos** (variables, palabras reservadas, literales, llamadas a métodos y campos) y **operadores** permiten formar ***expresiones***.
- Una clasificación de los operadores es:
  - Operadores aritméticos
  - Operadores relacionales
  - Operadores lógicos
  - Operadores condicionales
  - Operadores de asignación

Operadores y expresiones

- La evaluación de una expresión da como resultado un valor de un determinado tipo de dato.
- La *precedencia* indica el orden de evaluación cuando hay varios operadores en la misma expresión y la *asociatividad* cuando tienen la misma precedencia.
- Todos los operadores binarios, excepto los de asignación, se asocian de izquierda a derecha.
- Los operadores de asignación se asocian de derecha a izquierda.

Operadores y expresiones: precedencia

Precedencia	Expresión	Tipos de Operandos	Descripción
1.	exp++ exp––	Numérico Numérico	Incremento postfijo; resultado es el valor antes Decremento postfijo; resultado es el valor antes
2.	++exp ––exp +exp –exp ~exp !exp	Numérico Numérico Numérico Numérico Integer, boolean Boolean	Incremento prefijo; resultado es el valor después Decremento prefijo; resultado es el valor desp. Positivo unario Negativo unario Complemento bits Negación lógica
3.	exp_1 * exp_2 exp_1 / exp_2 exp_1 % exp_2	Numérico Numérico Numérico	Multiplicación División Resto, módulo
4.	exp_1 + exp_2 exp_1 – exp_2	Numérico String Numérico	Suma Concatenación de String Resta

Operadores y expresiones: precedencia

Precedencia	Expresión	Tipos de Operandos	Descripción
5.	exp_1 << exp_2 exp_1 >> exp_2 exp_1 >>> exp_2	Integer Integer Integer	Desplazamiento izquierda, relleno de 0s Desplazamiento derecha con signo Desplazamiento derecha sin signo, relleno de 0s
6.	exp_1 < exp_2 exp_1 > exp_2 exp_1 <= exp_2 exp_1 >= exp_2	Numérico Numérico Numérico Numérico	Menor que Mayor que Menor que o igual a Mayor que o igual a
7.	exp_1 == exp_2 exp_1 != exp_2	Cualquiera Cualquiera	Igual de comparación Diferente
8.	exp_1 & exp_2	Integer, boolean	And bits
9.	exp_1 ^ exp_2	Integer, boolean	Or exclusivo bits (xor)
10.	exp_1 | exp_2	Integer, boolean	Or inclusivo bits
11.	exp_1 && exp_2	Boolean	AND lógico

Operadores y expresiones: precedencia

Precedencia	Expresión	Tipos de Operandos	Descripción
12.	exp_1 || exp_2	Boolean	O lógico
13.	exp_1 ? exp_2 : exp_3	exp_1:Boolean $exp_{2/3}$,:Cualquiera	Expresión condicional
14.	var = exp var += exp var –= exp var *= exp var /= exp var %= exp var <<= exp var >>= exp var >>>= exp var &= exp var ^= exp var |= exp	Cualquiera Numérico, String Numérico Numérico Numérico Numérico Integer Integer Integer Integer, boolean Integer, boolean Integer, boolean	Asignación: var op= exp equivale a: var = (var) op (exp) con la excepción que var se evalúa una sola vez

Operadores aritméticos

- Se usan con operandos de tipos entero y punto flotante.

Operador	Uso	Descripción
*	op_1 * op_2	Multiplicación
/	op_1 / op_2	División
%	op_1 % op_2	Resto, módulo. Equivalencia x % y == x – (x / y) * y
+	op_1 + op_2 + op	Suma Signo positivo
–	op_1 – op_2 – op	Resta Signo negativo

Operadores aritméticos

- En las operaciones de división y módulo, si ambos operandos son enteros hay que tener cuidado de no perder “precisión”.

```
int first = 7, second = 4, answer;
answer =  first / second;  // answer es 1
```

  - El resultado es un entero, se pierde la parte fraccionaria.
- Para hallar el resto de la división entera se usa el operador módulo %

```
int first = 7, second = 4, answer, remainder;
answer =  first / second;
remainder =  first % second;  // remainder es 3
```

Operaciones con potencias y raíces

- Java no tiene operadores para potencias y raíces. En su lugar ofrece métodos de la clase Math. Ejemplo:

$$b \times \left(1 + \frac{r}{100}\right)^n$$

Java

```
b * Math.pow(1 + r / 100, n)
```

- Otros métodos son:

Método	Descripción
Math.sin(x)	seno de x (en radianes)
Math.cos(x)	coseno de x (en radianes)
Math.tan(x)	tangente de x
Math.log10(x)	logaritmo decimal $\log_{10}(x)$, x > 0
Math.abs(x)	valor absoluto | x |

Operaciones aritméticas

- Las operaciones con enteros nunca se desbordan. Si el valor excede el rango de su tipo se extiende por el módulo del rango.
- En expresiones x/y y x%y se produce una excepción de tipo ArithmeticException cuando y es 0.
- Las operaciones con punto flotante siguen la norma IEEE-754-1985. Una ventaja es que no se genera una excepción bajo ninguna circunstancia (el programa no aborta cuando se produce una división por cero).

Operaciones aritméticas

- La norma IEEE-754 define dos números mágicos: Infinity y NaN (not a number).
- Reglas que gobiernan la multiplicación y división:
  - Si ninguno de los operandos es NaN el resultado es:

x	y	x / y	x * y
Finito	±0.0	± ∞	±0.0
Finito	± ∞	±0.0	± ∞
±0.0	±0.0	NaN	±0.0
± ∞	Finito	± ∞	± ∞
± ∞	± ∞	NaN	± ∞
±0.0	± ∞	±0.0	NaN

  - Si algún operando es NaN el resultado es NaN.

Concatenación de cadenas (+)

- El tipo **String** sirve para declarar e inicializar cadenas de caracteres:

```
String nombre = "Esteban"
```

- La clase String dispone de métodos. Ejemplo:

```
int n = nombre.length();  // n = 7
```

- El operador + también se puede usar para concatenar dos cadenas. Si uno de los operandos es una cadena y el otro es de otro tipo, éste último se convertirá a una representación de cadena y se concatenará al operando cadena.

Concatenación de cadenas (+)

- 'Sumar' un String al final de otro:

```
String fName = "Harry"; String lName = "Morgan";
String name = fName + lName;  // HarryMorgan
```

- Añadir un espacio entre los dos:

```
String name = fName + " " + lName; // Harry Morgan
```

- Concatenar un valor numérico a una variable String:

```
String a = "Agent";  int n = 7;
String bond = a + n; // Agent7
```

- Concatenar Strings y valores numéricos dentro de println:

```
System.out.println("El precio total es " + total);
```

Operadores de incremento/decremento

- Se aplican con operandos de tipos entero y punto flotante. Pueden ser postfijos o prefijos.

Operador	Uso	Descripción
++	op++	Incrementa op en 1. El valor de la expresión es el valor de op antes del incremento
++	++op	Incrementa op en 1. El valor de la expresión es el valor de op después del incremento
––	op––	Decrementa op en 1. El valor de la expresión es el valor de op antes del decremento
––	––op	Decrementa op en 1. El valor de la expresión es el valor de op después del decremento

Operadores de incremento/decremento

- Ejemplos:

```
int i = 10;
int j = 3;
int k = 0;
k = ++j + i;  //resultado: k = 4 + 10 = 14
```

```
int i = 10;
int j = 3;
int k = 0;
k = j++ + i; //resultado: k = 3 + 10 = 13
```

Operadores relacionales

- Comparan dos valores y determinan la relación entre esos valores. El resultado es boolean (true o false).

Operador	Uso	Descripción
>	$op_1 > op_2$	op_1 es mayor que op_2
>=	$op_1 >= op_2$	op_1 es mayor o igual a op_2
<	$op_1 < op_2$	op_1 es menor que op_2
<=	$op_1 <= op_2$	op_1 es menor o igual a op_2
==	$op_1 == op_2$	op_1 y op_2 son iguales
!=	$op_1 != op_2$	op_1 y op_2 no son iguales (diferentes)

Operadores relacionales

- Los operadores == (igual de comparación) y != (diferente) se pueden aplicar a cualquier tipo de operando.
- Los operadores <, <=, > y >= se aplican sólo a tipos numéricos.

Operadores lógicos

- Se aplican a operandos de tipo boolean. El resultado es boolean (true o false).

x1	x2	x1 && x2 AND	x1 || x2 OR	!x1 NOT
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Operadores lógicos booleanos

- Se aplican a operandos de tipo integer o boolean.

x	y	~x complemento	x & y AND	x | y OR inclusivo	x ^ y OR exclusivo
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE

Operadores lógicos && y & (boolean)

- La diferencia entre los operadores lógicos AND es que && realiza una evaluación parcial o "corto circuito" mientras que & no.
- Así en la expresión `exp1 && exp2`
  - && evaluará la expresión exp1 y retorna `false` de forma inmediata si exp1 es `false`.
  - Si exp1 es `false`, el operador nunca evaluará la expresión exp2 porque el resultado de la expresión es `false` sin importar el valor de exp2.
- Por el contrario, el operador & siempre evaluará exp1 y exp2 antes de devolver un valor.

Operadores lógicos || y | (boolean)

- La diferencia entre los operadores lógicos OR es que || realiza una evaluación parcial o "corto circuito" mientras que | no.
- Así en la expresión `exp1 || exp2`
  - || evaluará la expresión exp1 y retorna `true` de forma inmediata si exp1 es `true`.
  - Si exp1 es `true`, el operador nunca evaluará la expresión exp2 porque el resultado de la expresión es `true` sin importar el valor de exp2.
- Por el contrario, el operador | y ^ siempre evaluará exp1 y exp2 antes de devolver un valor.

Operadores de desplazamiento bits

- Se aplican a operandos de tipo integer.

Expresión	Descripción
x << k	Desplaza los bits en x k lugares a la izquierda rellenando por la derecha con 0 bits. El resultado es $x \cdot 2^k$
x >> k	Desplaza los bits en x k lugares a la derecha rellenando por la izquierda con el bit más alto (signo). El resultado es $x / 2^k$
x >>> k	Desplaza los bits en x k lugares a la derecha rellenando por la izquierda con 0 bits

Operador condicional (? :)

- El operador condicional es *ternario* y por ello requiere de tres expresiones como operandos.
- La sintaxis de una expresión que usa el operador condicional es:

  exp1 ? exp2 : exp3

  donde: exp1 es una expresión boolean, exp2 y exp3 pueden ser de cualquier tipo
- Resultado: el valor de la expresión condicional es exp2 si exp1 es true y exp3 si exp1 es false.

Operador condicional (? :): ejemplo

```
public class OperadorCondicional {
   public static void main(String[] args) {
      String status = "";
      int grade = 80;
      // Obtiene el estado de un estudiante
      status = (grade >= 60)?"Apto":"No apto";
      // Imprime status
      System.out.println(status);
   }
}
```

- Resultado:

  Apto

Operadores de asignación

- El operador de asignación simple es = o uno de:

  += -= *= /= %=

  <<= >>= >>>= &= ^= |=

- La expresión de asignación: var op= exp

  es equivalente a: var = (var) op (exp)

  Con la excepción que var se evalúa una sola vez en el primer caso.

Ejemplos

- Programas de las transparencias:

  http://personales.unican.es/corcuerp/Java/Labs/codigo/HolaMundo.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/Circulo.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/PrintfDemo.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/OperadorCondicional.java

- Algunos programas para experimentar son:

  http://personales.unican.es/corcuerp/Java/Labs/codigo/Literales.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/OperadoresAritmeticos.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/OperadoresRelacionales.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/OperadoresLogicos.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/OperadoresBits.java

  http://personales.unican.es/corcuerp/Java/Labs/codigo/OperadoresAsignacion.java