

Tema 2. Elementos básicos de programación en Java

CHRISTOPHER EXPÓSITO IZQUIERDO

AIRAM EXPÓSITO MÁRQUEZ

ISRAEL LÓPEZ PLATA

MARÍA BELÉN MELIÁN BATISTA

JOSÉ MARCOS MORENO VEGA



Programación en Java

- Lenguaje compilado, concurrente y orientado a objetos
- A pesar de ser compilado, posee un intérprete que lo hace multiplataforma
- Es uno de los lenguajes con mayor utilización en el mundo
- Posee un gran número de librerías y frameworks



Programación en Java

- En java los ficheros de código fuente tienen extensión ***.java**
- Estos ficheros se compilan, generando un fichero ***.class** o ***.jar**
- Los ficheros resultantes de la compilación se pasan al intérprete de Java, el cual los ejecuta



Programación en Java

- Un programa en Java se estructura como un conjunto de **clases**, las cuales se verán en detalle en el tema sobre Programación Orientada a Objetos
- Es recomendable crear un fichero de código por cada clase, ambos con el mismo nombre
- Todo programa Java empieza por la función **main**
- A partir del main, las sentencias de Java se ejecutan en orden secuencial, es decir, de la parte superior del fichero a la parte inferior

Programación en Java

```
3 public class EjemploSencilloJava { Clase
4
5     /* Comentario
6     Ejemplo sencillo en Java de un programa que suma 2 + 2
7     */
8     public static void main(String[] args) {
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma); Punto de entrada
11    }
12 }
```

Delimitadores

- Toda sentencia Java viene delimitada por el carácter ;
- Para delimitar bloques de código se utilizan los caracteres { y }
 - {. Abre el bloque de código
 - }. Cierra el bloque de código
- Un bloque de código puede ser utilizado para delimitar:
 - Una clase
 - Una función
 - Un bucle o condicional

Delimitadores

```
3 public class EjemploSencilloJava {
4
5     /*
6     Ejemplo sencillo en Java de un programa que suma 2 + 2
7     */
8     public static void main(String[] args) {
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma);
11    }
12 }
```

Delimitadores de bloque

Delimitadores de sentencias

Palabras reservadas

- Al igual que la mayoría de lenguajes de programación, Java posee un gran número de palabras reservadas
- Cada palabra reservada tiene una funcionalidad dentro del lenguaje Java
- No se permite la utilización de una palabra reservada para nombrar los elementos de Java

Palabras reservadas

Palabras reservadas

```
3 public class EjemploSencilloJava {
4
5     /*
6      * Ejemplo sencillo en Java de un programa que suma 2 + 2
7      */
8     public static void main(String[] args) {
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma);
11    }
12 }
```

Palabras reservadas

Lista de palabras reservadas

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

Identificadores

- Un identificadores es un nombre con el que se hace referencia a:
 - Una función
 - Una clase
 - El contenido de una variable o una constante
- A la hora de definir un identificador, se debe tener en cuenta lo siguiente:
 - Se forma por una secuencia de letras y dígitos (incluido _)
 - No puede contener ni espacios en blanco ni caracteres especiales como +, -, *, /, ,, etc.
 - El primer carácter debe ser una letra o _
 - Distingue entre mayúsculas y minúsculas
 - Conviene incluir nombres que indiquen lo que representan o su utilidad

Identificadores

Identificadores

```
3 public class EjemploSencilloJava {
4
5     /*
6         Ejemplo sencillo en Java de un programa que suma 2 + 2
7     */
8     public static void main(String[] args) {
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma);
11    }
12 }
```

Comentarios

- Su objetivo es explicar o aclarar una parte del código
- Son ignorados por el compilador
- Existen 2 tipos:
 - `/* */` Cuando el comentario ocupa mas de una línea
 - `//` Cuando el comentario sólo ocupa una línea
- Se debe procurar incluir sólo los comentarios mínimos para la comprensión del código

Comentarios

```
3 public class EjemploSencilloJava { Clase
4
5     /* Comentario
6      Ejemplo sencillo en Java de un programa que suma 2 + 2
7      */
8     public static void main(String[] args) { Punto de entrada
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma);
11    }
12 }
```

Tipos de datos

- Todo dato manejado en Java debe tener un tipo
- El tipo puede ser:
 - Explícito. Indicado por el programador
 - Implícito. El compilador interpreta el tipo automáticamente
- El tipo de datos sirve al compilador para:
 - Conocer el espacio de memoria a reservar para ese dato
 - Conocer una serie de directrices a la hora de manejarlo

Tipos de datos

```
3 public class EjemploSencilloJava {
4
5     /*
6     Ejemplo sencillo en Java de un programa que suma 2 + 2
7     */
8     public static void main(String[] args) {
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma);
11    }
12 }
```

Declaración explícita

Declaración implícita

Tipos de datos. Numéricos

Nombre	Tipo	bytes	Rango
byte	Entero	1 byte	-128 a 127
short	Entero	2 bytes	-32768 a 32767
int	Entero	4 bytes	-2147483648 a 2147483647
long	Entero	8 bytes	-9223372036854775808 a 9223372036854775807
float	Decimal	4 bytes	1.40129846432481707e-45 a 3.40282346638528860e+38
double	Decimal	8 bytes	4.94065645841246544e-324 a 1.79769313486231570e+308

Tipos de datos. Otros

Nombre	Representa	bytes	Ejemplo
boolean	Verdadero o Falso	1 byte	true, false
char	Carácter	2 bytes	'a', 'b', 'c', '6',
String	Cadena de caracteres	4 bytes	"cadena", "ejemplo"

- Los caracteres soportan cualquier carácter ASCII. Esto implica:
 - Algunos caracteres no pueden ser representados (tildes, ñ, etc)
 - El carácter `\` representa el carácter de escape. Ejemplos:
 - `\n`: Salto de línea
 - `\'`: Comilla
 - `\\`: Carácter `\`

Variables

- Elemento del lenguaje que permite almacenar valores que pueden cambiar a lo largo del tiempo o las ejecuciones
- Se definen de la forma **tipo nombre**
- En Java es obligatorio indicar el tipo de la variable, así como inicializarla
- Ejemplos:
 - `int suma = 0;`
 - `double numDecimal = 3.8;`
 - `boolean esCierto = true;`
 - `char caracterSimple = 'c';`
 - `String cadena = "Ejemplo";`

Operadores

- Carácter o grupo de caracteres especiales que actúa sobre una, dos o mas variables y/o constantes para obtener un resultado
- Se pueden combinar dando lugar a expresiones, pudiendo usar paréntesis para modificar su precedencia
- Para dar un valor a una variable, se utiliza el operador de asignación =
- Ejemplo:
 - `int a = 3;` → a vale 3

Operadores aritméticos

- Operadores cuyo resultado es una operación aritmética
- Pueden hacer use de una o varias variables o constantes
- Al devolver un valor numérico, suelen utilizarse junto con el operador de asignación de valor a una variable =, para que esa variable tome como valor el resultado de la operación aritmética
- Los operandos de los operadores aritméticos deben ser de tipo numérico, a excepción del + que tiene utilidad en los operandos tipo String

Operadores aritméticos

Nombre	Símbolo	Descripción	Ejemplo
Suma	+	Realiza la suma entre los valores a ambos lados del operador	<pre>int a = 2+3; double b = a + 5.3; String c = "a = " + a + " - " + "b = " + b;</pre>
Resta	-	Realiza la resta entre los valores a ambos lados del operador	<pre>int a = 2-3; double b = a - 0.8;</pre>
Multiplicación	*	Realiza la multiplicación entre los valores a ambos lados del operador	<pre>int a = 2*3; double b = 2.2*a;</pre>
División	/	Realiza la división cuyo dividendo se encuentra a la derecha del operador y divisor a la izquierda	<pre>int a = 10 / 4; double b = 5 / a</pre>
Módulo	%	Obtiene el resto de la división cuyo dividendo se encuentra a la derecha del operador y divisor a la izquierda	<pre>int a = 11 % 4</pre>
Incremento Decremento	++ --	Equivale a incrementar o decrementar 1 a la variable del operador. Operador unario.	<pre>int a = 2; int b = a++; int c = b--;</pre>

Operadores aritméticos

Nombre	Símbolo	Descripción	Ejemplo
Suma	+	Realiza la suma entre los valores a ambos lados del operador	<pre>int a = 2+3; double b = a + 5.3; String c = "a = " + a + " - " + "b = " + b; Resultado: a = 5 b = 10.3 c = "a = 5 - b = 10.3"</pre>
Resta	-	Realiza la resta entre los valores a ambos lados del operador	<pre>int a = 2-3; double b = a - 0.8; Resultado: a = -1 b = -1.8</pre>
Multiplicación	*	Realiza la multiplicación entre los valores a ambos lados del operador	<pre>int a = 2*3; double b = 2.2*a; Resultado: a = 6 b = 13.2</pre>
División	/	Realiza la división cuyo dividendo se encuentra a la derecha del operador y divisor a la izquierda	<pre>int a = 10 / 4; double b = 5 / a Resultado: a = 2 b = 2.5</pre>
Módulo	%	Obtiene el resto de la división cuyo dividendo se encuentra a la derecha del operador y divisor a la izquierda	<pre>int a = 11 % 4 Resultado: a = 2</pre>
Incremento Decremento	++ --	Equivale a incrementar o decrementar 1 a la variable del operador. Operador unario.	<pre>int a = 2; int b = a++; int c = b--; Resultado: a = 2 b = 3 c = 2</pre>

Operadores relacionales

- Operadores que permiten la comparación de varios elementos
- Su resultado es un valor **boolean** (true o false)
- Tienen gran utilidad en condicionales y bucles, los cuales se explican en temas posteriores

Operadores relacionales

Nombre	Símbolo	Descripción	Ejemplo
Igual que	==	Compara si 2 elementos son iguales	2 == 3; 'a' == 'a';
Menor que	<	Indica si el operador de la izquierda es menor estricto que el de la derecha	2 < 3; 'a' < 'b';
Mayor que	>	Indica si el operador de la izquierda es mayor estricto que el de la derecha	2 > 3; 'a' > 'a';
Menor o igual que	<=	Indica si el operador de la izquierda es menor o igual que el de la derecha	2.3 <= 2.4 'a' <= 'a'
Mayor o igual que	>=	Indica si el operador de la izquierda es mayor o igual que el de la derecha	3.2 >= 3.2 'c' >= 'a'
Distinto de	!=	Compara si 2 elementos son diferentes	2 != 2; 'a' != 'b';

Operadores relacionales

Nombre	Símbolo	Descripción	Ejemplo
Igual que	==	Compara si 2 elementos son iguales	2 == 3; Resultado: false 'a' == 'a'; Resultado: true
Menor que	<	Indica si el operador de la izquierda es menor estricto que el de la derecha	2 < 3; Resultado: true 'a' < 'b'; Resultado: true
Mayor que	>	Indica si el operador de la izquierda es mayor estricto que el de la derecha	2 > 3; Resultado: false 'a' > 'a'; Resultado: false
Menor o igual que	<=	Indica si el operador de la izquierda es menor o igual que el de la derecha	2.3 <= 2.4 Resultado: true 'a' <= 'a' Resultado: false
Mayor o igual que	>=	Indica si el operador de la izquierda es mayor o igual que el de la derecha	3.2 >= 3.2 Resultado: true 'c' >= 'a' Resultado: true
Distinto de	!=	Compara si 2 elementos son diferentes	2 != 2; Resultado: false 'a' != 'b'; Resultado: true

Conversiones de tipos

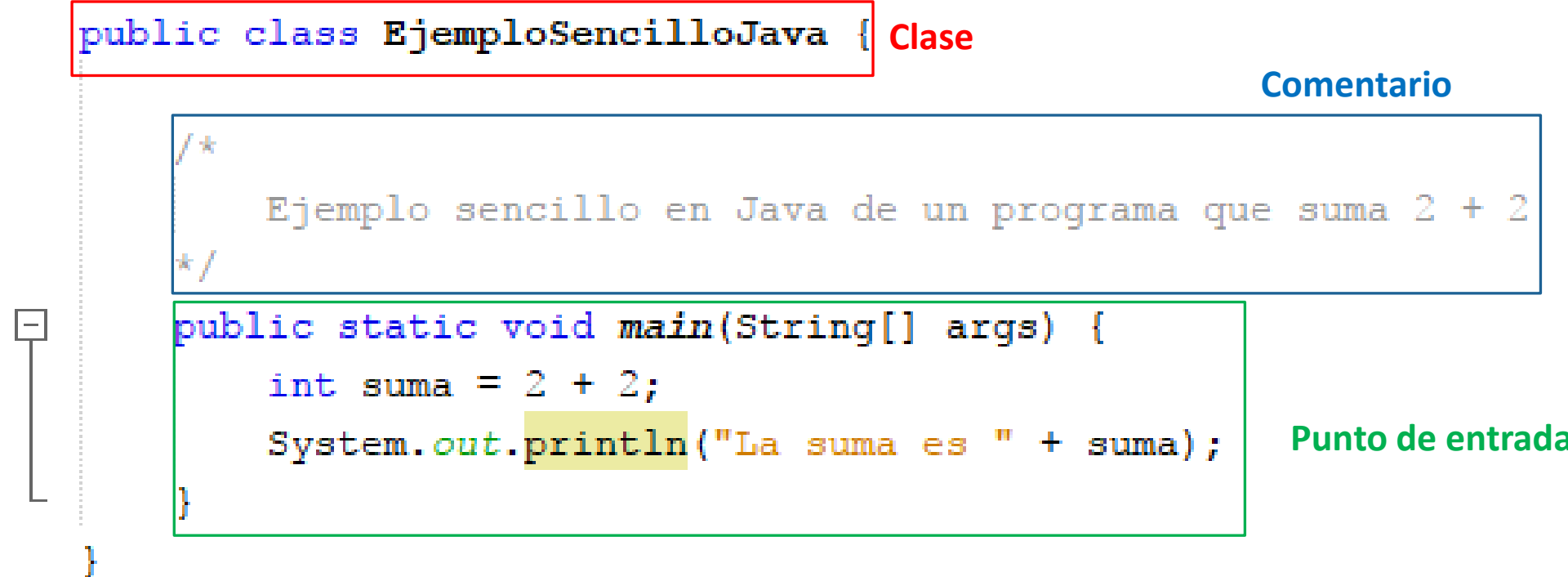
- Java permite operar entre variables o constante de distinto tipo, siempre que éstas sean compatibles
- Dentro de una operación, intenta convertir todos los operadores al tipo del resultado
- A este tipo de conversión, se le conoce como conversión implícita
- Ejemplo:
 - `int a = 3.35;` → a vale 3
 - `String c = 'a'` → c vale la cadena "a"

Conversiones de tipos

- También se permite la conversión de tipos explícita, donde el programador indica a que tipo desea convertir la variable o constante
- Para hacer una conversión explícita, se usa el operador paréntesis (**tipo**)
- Ejemplo:
 - `int a = (int) 3.35;` → a vale 3
 - `double c = (double) 3` → c vale 3.0

Primer programa en Java

```
3 public class EjemploSencilloJava { Clase
4
5     /* Comentario
6     Ejemplo sencillo en Java de un programa que suma 2 + 2
7     */
8     public static void main(String[] args) {
9         int suma = 2 + 2;
10        System.out.println("La suma es " + suma); Punto de entrada
11    }
12 }
```



Primer programa en Java

- Para escribir un programa en Java, a partir del punto de entrada (función main) se escriben secuencialmente todas las sentencias del programa
- Java ejecutará todas las sentencias del programa en orden estricto
- Los entornos de desarrollo actuales (IDEs) crean automáticamente la clase principal con la función main incorporada

Primer programa en Java

- Escribir un programa en Java que, dado un lado, calcule el área de un cuadrado:

```
public class Prueba {  
  
    public static void main(String[] args) {  
        int lado = 2;  
  
    }  
  
}
```

Primer programa en Java

- Escribir un programa en Java que, dado un lado, calcule el área de un cuadrado:

```
public class Prueba {  
  
    public static void main(String[] args) {  
        int lado = 2;  
        int area = lado*lado;  
    }  
  
}
```


Primer programa en Java

- Escribir un programa en Java que, dado ancho y largo, calcule el perímetro y el área de un rectángulo. Además, debe sumar ambos resultados:

```
public class Prueba {  
  
    public static void main(String[] args) {  
        int ancho = 5;  
        int largo = 10;  
  
    }  
  
}
```

Primer programa en Java

- Escribir un programa en Java que, dado ancho y largo, calcule el perímetro y el área de un rectángulo. Además, debe sumar ambos resultados:

```
public class Prueba {  
  
    public static void main(String[] args) {  
        int ancho = 5;  
        int largo = 10;  
        int area = ancho * largo;  
        int perimetro = ancho*2 + largo*2;  
        int total = area + perimetro;  
    }  
  
}
```

Primer programa en Java

- Escribir un programa en Java que, dados 3 caracteres, concatene ambos:

```
public class Prueba {  
  
    public static void main(String[] args) {  
        char a = 'x';  
        char b = 'y';  
        char c = 'z';  
  
    }  
  
}
```

Primer programa en Java

- Escribir un programa en Java que, dados 3 caracteres, concatene ambos:

```
public class Prueba {  
  
    public static void main(String[] args) {  
        char a = 'x';  
        char b = 'y';  
        char c = 'z';  
        String concatena = a + b + c;  
    }  
  
}
```

Entrada y Salida

- Para mostrar un texto (incluido valores de variables) por pantalla, se hace uso de la sentencia **System.out**. Existen 2 formas básicas:
 - **System.out.println(cadena)**: Muestra la cadena de caracteres y realiza un salto de línea
 - **System.out.print(cadena)**: Igual que el anterior, pero sin hacer un salto de línea
- La cadena que se pasa por parámetro puede contener variables, las cuales son transformadas automáticamente por Java en un String
- Ejemplos:
 - `System.out.println("Hola clase");` → "Hola clase"
 - `int res = 3; System.out.println("El resultado es = " + res);` → "El resultado es = 3"
 - `int a = 2; int b = 3; System.out.println(a + " es menor que " + b + " = " + (a < b));` → "2 es menor que 3 = true"

Entrada y Salida

- Para obtener un texto por teclado, se utiliza la clase Scanner. Se declara como sigue:

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

- Una vez tenemos la variable `sc`, utilizamos diferentes funciones dependiendo del tipo de datos que queremos leer.

- Ejemplos:

```
System.out.println("Por favor ingrese su nombre");  
String nombre = sc.nextLine();  
  
System.out.println("Bienvenido " + nombre + ". Por favor ingrese su edad");  
sc.nextInt();
```

Buenas prácticas

- Se debe utilizar el tipo adecuado para la operación que quieres realizar
- Lo habitual suele ser:
 - int → Para números enteros
 - double → Para números con decimales
 - char → Para un solo carácter
 - String → Para cadenas de caracteres
 - boolean → Para resultados true/false
- Utilizar el tipo incorrecto puede provocar errores o hacer el software menos optimizado

Buenas prácticas

- Los nombres de las variables, funciones o clases deben describir su utilidad
- Con ello se facilita la lectura del código y se comprende mejor lo que se ha hecho
- Ejemplos:
 - `int area = ancho * alto; //Nombre correcto`
 - `int xx = ancho * alto; //Nombre incorrecto`
- No poner mas comentarios de los estrictamente necesarios. Sólo cuando se quiere describir una parte relativamente compleja
- No es correcto comentar todas las líneas del código

Buenas prácticas

- Hay que ser ordenado a la hora de escribir el código
- Utilizar la indentación adecuada.
 - Dentro de cada bloque, se debe incluir 3 espacios o un tabulador a la derecha
 - Con ello se permite ver mejor los límites del bloque

```
int a = 3;
for (int i = 0; i < 10; i++) {
    if (i < a) {
        System.out.println("I es menor");
    } else {
        System.out.println("I es mayor");
    }
    int suma = a + i;
    System.out.println("La suma es " + suma);
}
```

Correcto

```
int a = 3;
for (int i = 0; i < 10; i++) {
    if (i < a) {System.out.println("I es menor");}
    else {
        System.out.println("I es mayor");}
    int suma = a + i;
    System.out.println("La suma es " + suma);
}
```

Incorrecto

Buenas prácticas

- **Pensar antes de teclear.** Piensa primero el algoritmo a implementar. Ahorra mucho tiempo de codificación
- **Probar el algoritmo.** Se debe probar el código para garantizar su calidad
 - Piensa en todos los posibles casos
 - Si el código no funciona para esos casos, arréglalo
 - Prueba casos raros
- **Conocer en detalle del código creado.** Si no se comprende una parte del código creado, se debe estudiar. Evita futuros errores con esa sección del código