El lenguaje de programación Java

Programa Java

- Un programa Java está formado por un conjunto de clases que interactúan entre sí
  - La clase es la unidad básica de programación
- La ejecución depende de la clase pública a la que se invoque con el intérprete

Programa Java

```
/*Primer programa de ejemplo*/

//definicion de clase publica o accesible
public class PrimeraAplicacion {

 // programa principal de la clase
 public static void main(String args[]){
         // escritura por pantalla
         System.out.println( "Esta es la
              primera aplicacion" );
 }
}
```


Comentarios

- Permiten documentar el código haciéndolo más legible a los programadores

```
// comentario de una sóla línea

/* Un comentario
   que aparece en varias
            líneas
   */

/** Comentario de documentación.
   La herramienta javadoc extrae los comentarios del código y
   genera html a partir de este tipo de comentarios
*/
```


Identificadores

- Permiten nombrar los distintos elementos de un programa
  - variables, objetos, clases, paquetes, interfaces
- Sintaxis:
  - Comienzan con letra (incluyendo _ y $)
  - Seguido de letras o dígitos
  - Cualquier longitud
  - Se distinguen mayúsculas de minúsculas
- Ejemplos:
  - x
  - _var1
  - MAXIMO
  - $Caracter

Palabras reservadas

- Palabras reservadas con propósito especial en el sistema y que no se pueden utilizar como identificadores

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

- Otras palabras reservadas (sin uso actual):

cast	future	generic	inner	operator	outer
rest	var				

Literales

- Valor constante de un tipo de datos
  - Booleanos, siempre en minúsculas
    - true y false
  - Numéricos enteros
    - Decimal: 12 0 134L (L para Long, l para Integer)
    - Octal: O25 (O delante)
    - Hexadecimal: Ox15 (Ox delante)
  - Numéricos reales, con punto, no con coma
    - Double: 28.4 5.3 .65 4.87d 1e3 (es 1×10^3) 78.34e-4d (la d de double es opcional al final)
    - Float: 1e2f .84f 2.F ('f' o 'F' detrás del valor)

Literales

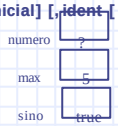
- Caracteres: un carácter UNICODE, entre comillas simples
  - 'N' '\116' (Octal) '\U00A2' (Hexadecimal)
  - Caracteres especiales:
  - \b retroceso
  - \t tabulador
  - \n salto de línea
  - \r cambio de línea
  - \" carácter comillas dobles (desinterpretado)
  - \' carácter comillas simples
  - \\ carácter barra invertida
- Cadenas de caracteres de la clase String, entre comillas dobles
  - "Hola Mundo." +" otro ejemlplo" + " +es un operador concatenador de textos"
- Referencia literal de objeto
  - null

Variables

- Unidad básica de almacenamiento de información
- Elementos cuyo valor puede cambiar durante el programa
- Declaración

  tipo identificador;
  tipo identificador [= valor_inicial] [, ident [= valor_ini]] ...;
- Ejemplos

```
int numero;
int  max= 5;
boolean sino = true;
```



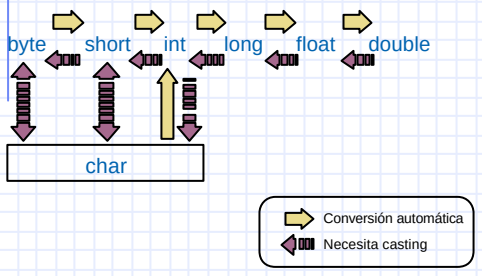
Constantes con nombre

- Variable constante
  - Variable cuyo valor nunca cambia
  - Se declaran con la palabra clave ***final***
    - **final float PI = 3.141592;**
    - **final int MAX = 255;**
    - **final int ABIERTO = 0, CERRADO = 1;**
    - **final boolean FALSO = false;**

Tipos primitivos

- Enteros (con signo): MIN_VALOR MAX_VALOR
  - byte 8 bits -128 +127
  - short 16 bits -32768 +32767
  - int 32 bits -2147483648 +2147483647
  - long 64 bits -2**63 2**63-
    - 0 1 -1 29 035 0x1d 29L
- Reales (coma flotante, IEEE 754-1985):
  - float 32 bits
  - double 64 bits
    - 24. 2.4e1 .24e2 0.0
- Lógico o booleano: 1 bit
  - boolean
    - true false
- Caracteres (ISO Unicode 1.1 de 16 bits):
  - char
    - 'a' 'A' '\n' '\t' '\u0058' → 'X'

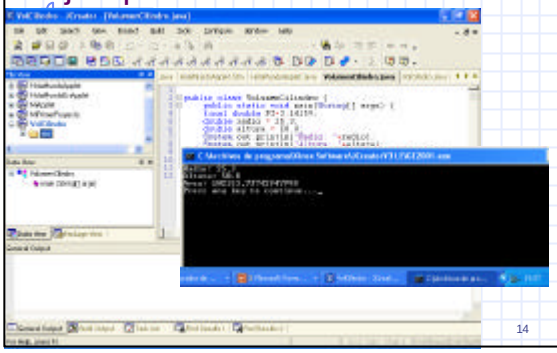
Tipos primitivos



Tipos primitivos

```
int i, j;
long x;
float f;
double d;
// ...
i = j; // mismo tipo
x = i; // OK
i = x; // NO, se perdería información
i = (int) x; // Se necesita casting
d = i; // OK
f = d; // NO
```


Ejemplo



Ejemplo

```
public class MathDemo {
 public static void main(String[] args) {
 double values[] = {11.5, 12.5, -12.5, -13.5};
 for (int i=0;i<values.length;i++) {
  double current = values[i];
  System.out.println("value: "+current);
  System.out.println("ceil: "+Math.ceil(current));
  System.out.println("floor:"+Math.floor(current));
  System.out.println("round: "+Math.round(current));
  System.out.println();
  }
 }
}
```

Expresiones

- Las expresiones son instrucciones que devuelven un valor lógico
  - Cada expresión tiene un tipo que se determina en tiempo de compilación
- Una expresión puede ser:
  - una constante (un literal): true 0 1.0
  - una variable x resultado
  - una combinación de constantes, variables, operadores y llamadas a métodos
    - x+1
    - ((x==3) || b)
    - Math.random() // genera un número aleatorio entre 0 y 1

Operadores aritméticos

- Suma: +
- Resta: -
- Multiplicación: *
- División: /
- resto: %

- Promoción automática de tipos
  - Cuando los tipos de operandos no coinciden, el operando de menor rango se convierte implícitamente al tipo de mayor rango:
    double > float > long > int > short > byte
  - La conversión descendente es obligatoria -pierde decimales-: (int)32
  - El resultado de la operación es del tipo de mayor rango
  - Al operar con byte y short, se convierten implícitamente a int

Ejemplo

```
// ejemplo de operadores aritméticos
public class PruebaAritmetica {
  public static void main (String args[]) {
    short x = 6;
    int y = 4;
    float a = 12.5f;
    float b = 7f;

    System.out.println("x es " + x + ", y es " + y);
    System.out.println("x + y = " + (x + y));
    System.out.println("x - y = " + (x - y));
    System.out.println("Div. Ent: x / y = " + (x/y));
    System.out.println("Resto: x % y = " + (x % y));
    System.out.println("a es " + a + ", b es " + b);
    System.out.println("a / b = " + (a / b));
  }
}
```

Operadores aritméticos

- Operadores aritméticos unarios
  - preincremento: ++x
  - postincremento: x++
  - predecremento : --x
  - postdecremento: x--
- Operadores de asignación
  - normal: x = y
  - adición: x += y x = x + y
  - resta: x -= y x = x - y
  - multiplicación: x *= y x = x * y
  - división: x /= y x = x / y

Expresiones booleanas

- Operadores lógicos
  - y lógico: x && y (cortocircuito)
    x & y (completo)
  - o lógico: x || y (cortocircuito)
    x | y (completo, evalunado ambos operandos)
  - negación: ! x
- Operadores relacionales: Comparaciones
  - igual: x == y (sólo con tipos básicos no objetos)
  - diferente: x != y
  - mayor que: x > y
  - menor que: x < y
  - mayor o igual que: x >= y
  - menor o igual que: x <= y

Ejemplo: año bisiesto

```
// ejemplo de operadores enteros, lógicos y relacionales
public class Bisiesto {
     // programa principal de la clase
     public static void main(String args[]) {
      int anno=2000;
      boolean  sino;
         // bisiesto si es múltiplo de 4 pero no de 100
         // o bien si es múltiplo de 400
      sino = (((anno % 4) == 0) && ((anno % 100) != 0)) ||
                                    ((anno % 400) == 0) );
      System.out.println( "Año bisiesto, valor de sino " +
                                                sino);
      anno++;
       sino = (((anno % 4) == 0) && ((anno % 100) != 0)) ||
                  ((anno % 400) == 0) );
      System.out.println( "Año bisiesto, valor de sino " +
sino);
       }
}
```


Expresiones a nivel de bits

- Operadores a nivel de bits (en enteros)
  - AND: x & y
  - OR: x | y
  - XOR: x ^ y
  - Desplazamiento izd: x << y
  - Desplazamiento der : x >> y
  - Desplazamiento der llenado de 0 a der : x >>> y
  - complemento de bits: ~x

Evaluación de expresiones

- Los operadores se evalúan de izquierda a derecha
- Precedencia de operadores:

Postfix	[] .(params) ++ --
Unarios	++ -- + - ~ !
Creación y ast	new (type)
Multiplicavos	* / %
Aditivos	+ -
Shift	<< >> >>>
Relacionales	< > <= >= instanceof
Igualdad	== !=
Bitwise AND	&
Bitwise XOR	^
Bitwise OR	|
Logical AND	&&
Logical OR	||
Condicional	?:
Asignación	= *= /= %= += -= >>= <<= >>>= &= ^= |=

Instrucciones, asignación y bloque

- Instrucción
  - Una orden que se le da al programa para realizar una tarea específica
- Asignación
  - Acción que da el valor de una expresión a una variable

  variable = expresión;

```
boolean condicion = true;        // declaración y asignación
                                    de valor inicial
int numero;                 // declaración de entero
numero = Math.sqrt(90);
// asignación de la raíz cuadrada de 90
```

Instrucciones, asignación y bloque

- Bloque de instrucciones
  - Conjunto de instrucciones que se consideran como una unidad
  - Están encerradas entre llaves { y }

Instrucción condicional `if`

```
if (expresión_booleana)
    instrucción
if (expresión_booleana)
    instrucción1
else
    instrucción2
```


Instrucción condicional `if`

- *Ejemplo:*

```
if ( x > y ){
  // si hay mas instrucciones hay que poner un bloque
  System.out.println("Dentro del if ");
  System.out.println("El mayor de x e y es x: " + x);
}
else if ( x < y )
  System.out.println("El mayor de x e y es y: " + y);
else
  System.out.println("x,y tienen mismo valor:" + x);
```


Ejemplo

```
int i =0;
i++;
if (i == 0) { }
int a[] = {1, 2, 3, 4, 5};
a.length;
i += 85;
if ( (i == 0) && (a.length != 5) )
  { }
if ( (i == 0) || (a.length != 5) )
  {  }
```


Instrucción condicional `switch`

```
switch (expresión)
{
   case expresión-constante1:
 instrucciones
   case expresión-constante2:
 instrucciones
   ...
   default: instrucciones
}
```


Instrucción condicional `switch`

- Se tiene que especificar `break` para salir de la instrucción
- La `expresión` puede ser tipo char, byte, short o int.
- Se puede usar un conjunto de instrucciones para varios casos:

```
case expresión-constantex:
case expresión-constantey:
case expresión-constantez:
instrucciones
```

Ejemplo

```
public class EjemploSwitch {
   public static void main(String[] args) {
       int mes = 2;
       switch (mes) {
           case 1:  System.out.println("Enero"); break;
           case 2:  System.out.println("Febrero"); break;
           case 3:  System.out.println("Marzo"); break;
           case 4:  System.out.println("Abril"); break;
           case 5:  System.out.println("Mayo"); break;
           case 6:  System.out.println("Junio"); break;
           case 7:  System.out.println("Julio"); break;
           case 8:  System.out.println("Agosto"); break;
           case 9:  System.out.println("Septiembre"); break
           case 10: System.out.println("Octubre"); break;
           case 11: System.out.println("Noviembre"); break
           case 12: System.out.println("Diciembre"); break
           default: System.out.println("Este mes no
existe"); break;     }   }  }
```


Bucles o Iteración: while

Ejecutan de forma repetida un bloque de instrucciones hasta que se hace falsa la condición lógica

```
while (expresión_booleana)
        instrucción
```

```
int a[] = { 1, 2, 3 ,4, 5};
int sum = 0;

int i = 0;
while ( i<a.length ){
   sum += a[i];
   ++i;
}
System.out.println("sum = " + sum);
```


Bucles o Iteración: while

```
public class Factorial {
     public static void main(String[] args) {
         int dato=9;
         long resultado=1;
         while (dato > 0) {
             resultado = dato * resultado;
             dato = dato -1;
         }
       System.out.println("El factorial es: " + resultado);
       }
  }
```


Bucles o Iteración: while

```
int a[] = { 1, 2, 3, 4, 5};
int sum = 0;

int i = 0;
while ( i < a.length ){
    sum = sum + a[i];
    i = i + 1;
}

System.out.println("sum = " + sum);
```


Bucles o Iteración: do while

- Ejecutan de forma repetida un bloque de instrucciones hasta que se hace falsa la condición lógica
- Se ejecuta por lo menos una vez

```
do
     instrucción
while (expresión_booleana)
```


Bucles o Iteración: do while

```
int a[] = { 1, 2, 3, 4, 5};
int sum = 0;
int i = 0;
do {
    sum = sum + a[i];
    i = i + 1;
    } while ( i < a.length );
System.out.println("sum = " + sum);
```

Bucles o Iteración: `for`

- Forma compacta de expresar bucles
- No sólo para un número de repeticiones fijas

```
for (inicialización;condición; incremento)
   instrucción
```


Bucles o Iteración: `for`

```
public class FactorialFor {
    public static void main(String[] args) {
        int numero=9;
        long resultado=1;
        for(int cont=1; cont <= numero; cont++){
            resultado= resultado * cont;
        }
        System.out.println("El factorial es
"+resultado);
    }
}
```

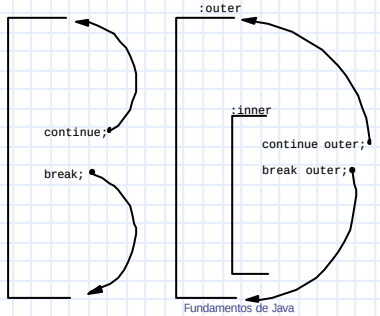

Bucles o Iteración: `for`

```
int a[] = { 1, 2, 3, 4, 5};
int sum = 0;

for (int i=0; i<a.length; i = i + 1){
   sum = sum + a[i];
}

System.out.println("sum = " + sum)
```


Otras instrucciones

- Fin de ejecución de un método
  - return ; // para los métodos void
  - return expresión ;
    - permite salir de cualquier ciclo dentro del método
    - vuelve al punto donde se invocó el método
- Modificación del flujo de programa en los bucles
  - **break**
    - permite salir del bucle
  - **continue**
    - salta a la siguiente iteración

Otras instrucciones



Argumentos de entrada

```
public static void main(String[]
  args){
 for (int i = 0; i < args.length;
  i++)
System.out.println("Parámetro " +
  + ": " + args[i]);
}
```