Unidad 4

Principios del lenguaje JAVA

Objetivos

- Conocer la historia y fundamentos de Java.
- Conocer la estructura general de un de un programa en Java
- Conocer las herramientas básicas: tipos de datos, constantes, operadores, etc.
- Usar los operadores conociendo su orden de prioridad y asociatividad.
- Saber construir expresiones válidas.
- Conocer las conversiones de tipos de datos.
- Conocer las funciones de E/S y las clases que las contiene.
- Conocer y utilizar las estructuras de control de flujo en Java.
  - De selección
  - De iteración
- Aprender y usar el tratamiento adecuado de excepciones.
- Conocer un entorno de desarrollo.
- Escribir y probar código.
- Crear y probar programas.
- Comentar y documentar código.

Contenidos

- Introducción
  - Historia.
  - Características generales.
  - Aplicaciones Java
  - JVM y JDK
- Estructura general de un programa en Java.
- Algunos elementos básicos: Comentarios, Delimitadores Identificadores, Palabras Claves.
- Tipos de datos elementales.
  - Modificadores de tipo
- Constantes.
- Variables
- Operadores.
- Expresiones
- Funciones básicas de entrada/salida.
- Sentencias de control
- Estructura secuencial.
- Estructura de selección simple *if*.
- Estructuras Repetitivas
  - while, do while.
  - for
- Estructura de selección múltiple *switch*.
- Proposiciones especiales
- Geany
  - Instalación
  - Edición, compilación, ejecución.

Introducción: Historia
De dónde procede Java

- En los ochenta aún reinaban los lenguajes de alto nivel estructurados
- Su problema: cuanto mayor es el problema a resolver más difícil es la solución con estos lenguajes
- Solución: Se adaptó la POO a los lenguajes existentes

Historia: De dónde procede Java

- En especial fue famoso el lenguaje C++ que adaptó el C a la POO
- Durante mucho tiempo fue el lenguaje más utilizado (aún lo es en muchos entornos)
- Otras adaptaciones:
  - Pascal→Turbo Pascal→Delphi
  - Basic→QuickBasic→Visual Basic

Historia: De dónde procede Java

- Ventajas de C++
  - Añadir soporte de POO (incluida la herencia múltiple)
  - Creación de potentes bibliotecas por parte de los desarrolladores (Por ejemplo MFC Microsoft Foundation Classes)
  - Es muy veloz

Historia: De dónde procede Java

- Desventajas de C++
  - C++ es compilado y se produce un ejecutable válido sólo para una plataforma concreta
  - Es inseguro
  - No es apropiado para la web

Historia: De dónde procede Java

- En 1991 se crea Oak en Sun Microsystems
- En 1995 aparece Java mejorando Oak
- Su sintaxis se basa en C++
- Sin embargo su funcionamiento es diferente

Introducción: Características

Transmission Control Protocol/ Internet Protocol

- Ventajas
  - Su sintaxis es similar a C y C++
  - Más seguro (No hay punteros)
  - Totalmente orientado a objetos
  - Muy preparado para aplicaciones TCP/IP
  - Implementa excepciones de forma nativa (Tratamiento de errores)
  - Permite multihilos
  - Tipos de datos más robustos
  - Es independiente de la plataforma

Introducción: Características

- Seguridad
  - La máquina virtual puede decidir no ejecutar el código si detecta instrucciones inseguras
  - Tiene varios verificadores de código que se aseguran de que el código es válido

Introducción. Aplicaciones Java

- Aplicaciones de consola. Para mostrar en la consola del sistema
- Aplicaciones gráficas. Haciendo uso de los objetos para gráficos
- Applets. Aplicaciones embebidas dentro de una página web que se ejecutan en el cliente
- Servlets. Aplicaciones embebidas en una página web que se ejecutan en el servidor

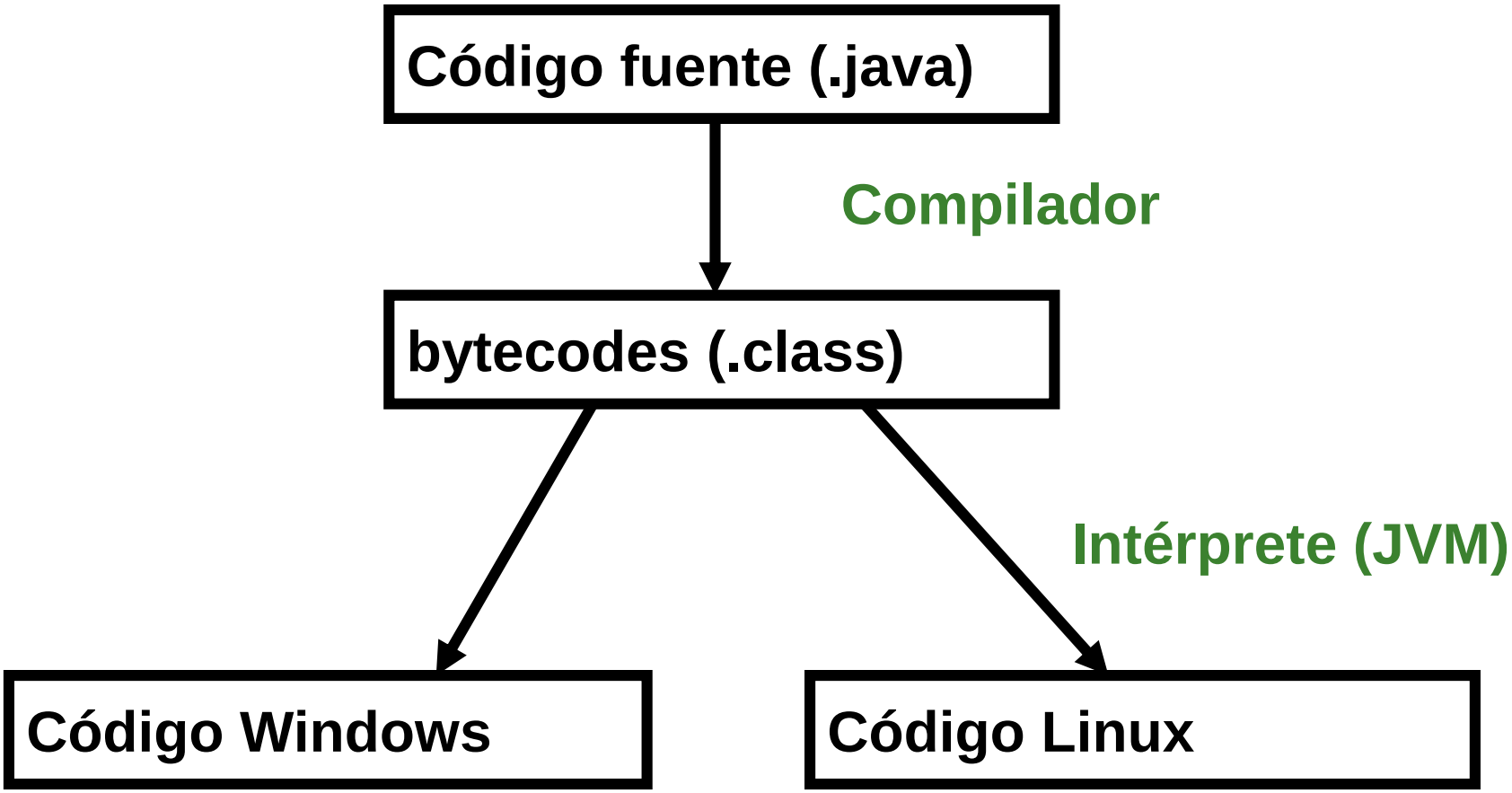
Plataformas de Java

- J2SE. Es la plataforma Java Estándar (cuando se habla de Java a secas, se entiende que nos referimos a esta plataforma)
- J2EE. Versión “enterprise”, empresarial y orientada al lado del servidor.
- J2ME. Versión para dispositivos portátiles

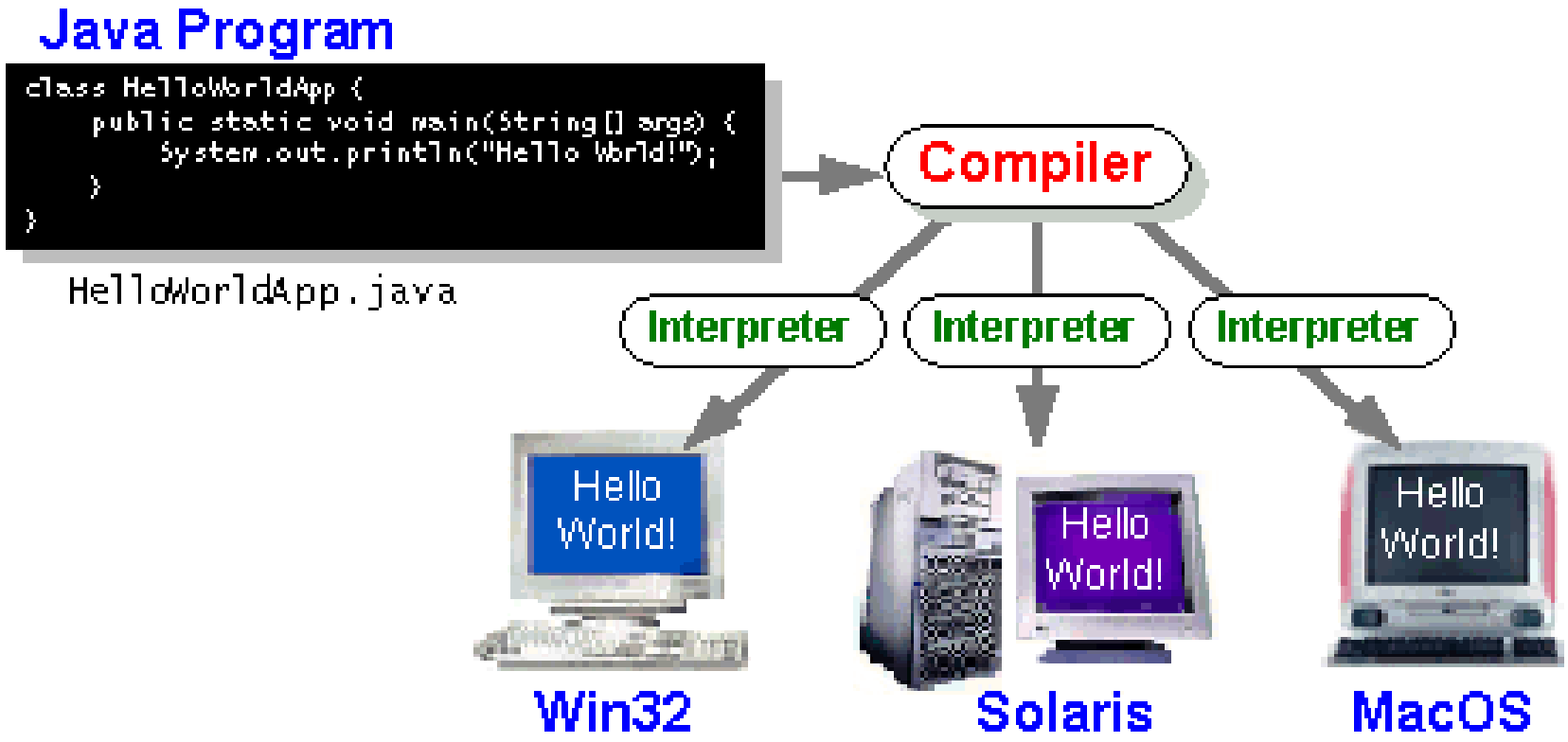
La máquina virtual Java (JVM)

- Java es un lenguaje compilado e interpretado
- El código fuente en Java se compila en forma de *bytecodes* que es un código semicompilado
- El resultado es un archivo *class*
- Este archivo luego es interpretado utilizando un software llamado JVM

La máquina virtual Java (JVM)



La máquina virtual Java (JVM)



El JDK

- JDK = *Java Developer Kit*, kit de desarrollo de Java
- Es el software que permite realizar el compilado y ejecución de los programas Java
- Es gratuito e incluye visores de código, depuradores y otras utilidades
- Se descarga de java.sun.com

Fases en la creación de un programa

- **Fase I: Edición**
  - Se crea un programa con la ayuda de un editor
  - Se almacena en un fichero con extensión *.java* (Por ejemplo, *MiPrograma .java*)
- **Fase II: Compilación**
  - El compilador lee el código Java (*MiPrograma .java*)
  - Si se detectan errores sintácticos, el compilador nos informa (*warnings*).
  - Si no hay errores se generan y almacenan los *bytecodes* en un fichero *.class* (*MiPrograma .class*)
- **Fase III: Cargado o enlazado de clases**
  - El cargador de clases lee el fichero *.class* e incluye las librerías necesarias.
- **Fase IV: Verificación de bytecodes**
  - El verificador de *bytecodes* comprueba que son válidos.
- **Fase V: Interpretación de bytecodes o compilación JIT(just-in-time)**

Estructura general de un Prog. JAVA

/*Ejemplo1: Escribir un programa que presente el texto Hola mundo en pantalla*/

```
import java.io.*;//no es necesario
public class Hello
{
 public static void main(String[] args)
    {
       System.out.println("Hola mundo");
    }
}
```

```
/*
Estructura general de una Clase en Java
*/
// import necesarios
public class NombreDeClase
{
   //Declaración de los atributos de la clase
   //Declaración de los métodos de la clase

  //el método main no lo tienen todas las clases
   //El método main, dónde empieza la ejecución
   public static void main(String[] args)
    {
      //declaración de variables del método
      //sentencias del método
    }
}
```

•public para que sea accesible por códigos externos a la clase.

•static permite que main() sea llamado por el intérprete de java antes de que se cree el objeto

•El * se usa si se quiere incluir varias (o todas) las clases del mismo paquete .

•El archivo fuente debe llamarse EXACTAMENTE IGUAL que la clase principal

Elementos básicos

- **Comentarios del programa**
  - `// Comentario de una linea`
  - `/* Comentario`
    `de varias lineas */`
  - `/**`
    `Comentario de documentación . Se verán más adelante`
    `*/`
- **Signos de puntuación o delimitadores**.
  - **;** **terminador** de sentencia.
  - **,** separa dos elementos consecutivos de una lista.
  - **( )** Delimita expresiones de grupo, condicionales, y listas de parámetros.
  - **{ }** Delimita un bloque de instrucciones o una lista de valores iniciales.
  - **[ ]** En la definición de arrays, delimita sus dimensiones.

Ejemplo:

```
{
    int var = 25, array [3]= {1,2,3};
    for (i = 0; i<3; i++)
        System.out.println("valor : "+ array [i]);
    System.out.println("valor  var: "+ var);
}
```

Identificadores

- Seguirán las reglas estudiadas en la UNIDAD 02
- Deben comenzar por una letra, _ o $
- El tamaño máximo depende del compilador
- ¡Ojo!
  - Son Case sensitive
  - No permitidas ni palabras reservadas ni nombres de funciones de Java
- Admite todos los caracteres de cualquier idioma. (Unicode)

Palabras reservadas en Java

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

Además las palabras null, false y true que sirven como constantes no se pueden usar como nombres de identificadores.

Tipos de datos

	Tipo	Byte	Rango	Ejemplos
Lógico	**boolean**	1	Sólo admite true o false	false, true
carácter	**char**	2	Unicode. Desde `'\u0000'` a `'\uffff'` inclusive. Esto es desde 0 a 65535	'a', 'A', '0', '*', ...
Entero	**byte**	1	Desde-128 a 127	0, 1, 5, -120, ...
Entero	**short**	2	Desde -32,768 a 32,767	0, 1, 5, -120, ...
Entero	**int**	4	Desde -2,147,483,648 a 2,147,483,647	0, 1, 5, -120, ...
Entero	**long**	8	Desde -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	0, 1, 5, -120, ...
Real	**float**	4	Desde 1.40239846e–45f a 3.40282347e+38f)	121.2 45
Real	**double**	8	Desde 4.94065645841246544e–324d a 1.7976931348623157e+308d	1111221.2 22222
	void		Sin valor	

Variables

- Formato de declaración e inicialización
  tipo lista_variables [=cte];

Pueden ser inicializadas al declararlas o cuando se necesite a lo largo del programa.

```
char seguir, continuar
='s';
long int contador;
float sueldo = 2590.54;
.................
Contador = 0
```

- ¿Dónde se declaran?
  - Las locales: Declaradas y conocidas dentro de la función o bloque que las usa.
  - Las globales: static (se verán más adelante)

Constantes

- Literales: se usan directamente. Ej: 4, L87, 3.14, ‘a’, “ hola”
- Simbólicas: deben ser declaradas

  ***final* tipo constante = cte;**

```
final char car = ‘s’;
final long  NUMMAX=52000;
final float IVA = 5.4;
```

Operadores aritméticos

*	producto
/	división
%	resto de división entera
+	suma
–	resta (como operador unario es el cambio de signo)

```
int a = 1, b = 2;
int c = a + b;
short s = 1;
int d = s+c; // s se convierte a int
float f = 1.0 + a; // a se convierte a float
c = a%b; //almacena 1 en c
c = a/b; //almacena cero entero en c
f = a/b; // almacena cero real en f
f = (float) a/b; //almacena 1.5 en f
```

Operadores relacionales

<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual
!=	Distinto
==	Igual

Operadores lógicos

&&	AND
||	OR
!	NOT

Otros operadores

=	Asignación simple
++	Incremento
--	Decremento

+=	Suma y asignación
-=	Diferencia y asignación
*=	Producto y asignación
/=	División y asignación

```
int a = 1;
a+=1   //a = a + 1;
```

Incremento++ decremento--

Ambos pueden ser usados en pre y en post

```
a = 5;
b = ++ a;
```

En la salida : a vale 6 y b vale 6

```
a = 5;
b =  a ++;
```

En la salida : a vale 6 y b vale 5

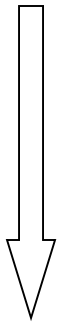
```
int var1 = 3;
    var1 ++;
    int var1 = 3, var2 = 0;
    var2 = ++var1;    // Se incrementa var1 y luego se asigna a var2;
    var2 = var1++;     // Se asigna a var2 y luego se incrementa var1;

    int nota = 17, pendientes = 10;
    nota++;     // es idéntico a nota = nota + 1;
    pendientes--;   // es idéntico a pendientes = pendientes - 1;
```

Prioridad de los operadores

Operador	Asociatividad
( )	Izquierda a derecha, primero los internos
! ++ -- - (unario)	derecha a izquierda
* / %	izquierda a derecha
+ - (binario)	izquierda a derecha
< <= > >=	izquierda a derecha
== !=	izquierda a derecha
&&	izquierda a derecha
||	izquierda a derecha
=	Derecha a izquierda

Mayor Precedencia



Menor Precedencia

En Java pueden añadirse en la escritura del código cuantos **espacios y paréntesis** se deseen para añadir legibilidad y claridad, sin que aumente el tiempo de ejecución, el compilador los ignora.

Expresiones

- Definición:
  - Una expresión se forma con identificadores, valores constantes.
  - Una expresión bien formada tiene siempre un tipo y una vez evaluada devuelve un valor de ese tipo.
- Se evalúan según:
  - Reglas de precedencia de los operadores que intervienen.
  - Teniendo en cuenta los paréntesis si los hay.
  - Y los tipos de datos de los operandos.

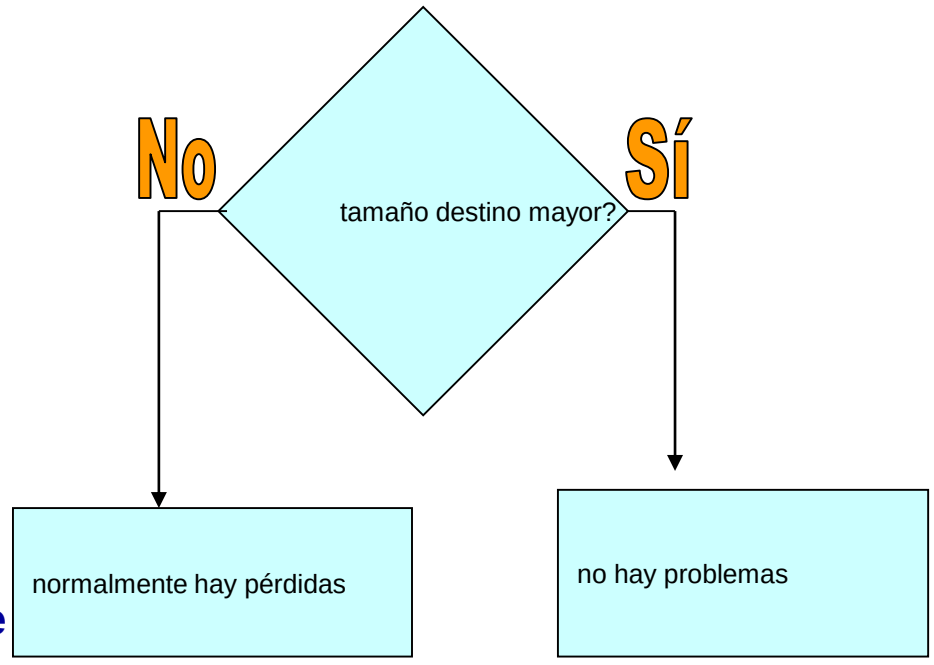
Conversiones de tipos primitivos

En Java pueden mezclarse en una expresión objetos de distintos tipos de datos, por lo que deben realizarse conversiones de tipo.

a) Conversiones automáticas:

- **En expresiones: al de mayor tamaño.**
- **En asignaciones: al tipo de la izquierda.**

byte
▼
char
▼
short
▼
int
▼
long
▼
float
▼
double



b) Conversiones forzadas:

(tipo) expresión

int a = 5, b = 2; float resul, resul2;
resul = a / b; ***// resul = 2.0***
resul2 = ***(float)*** *a / b;* ***// resul2= 2.5***

sin pérdida:

float a = 375;
int b = ‘ H’;
long c = - 45;
double d = 9.6;
float e = 250;

con pérdida:

char a = 450450;
int b = 52074,5;
int c = 75.45;
float d = 1.58 e + 62;

Conversiones de tipos primitivos

- La siguiente tabla muestra las conversiones de tipo que no producen pérdida de información:

no produce pérdida ⟶ si se convierte a

Tipo original	Tipo convertido
`byte`	`short`, `char`, `int`, `long`, `float`, `double`
`short`	`int`, `long`, `float`, `double`
`char`	`int`, `long`, `float`, `double`
`int`	`long`, `float`, `double`
`long`	`float`, `double`
`float`	`double`

Sentencia o proposición

- **Simple**: contienen una única instrucción y acaban en **;**

  Sintaxis:

  Sentencia-expresión;

  - **Tipos**
    - **De declaración**

      int x;
    - **De asignación**

      media = total/numAlumno;
    - **De llamada a función**

      System.in.read();
    - **De incremento y decremento**

      j++;

- **Compuesta o bloque**:

  {

  lista-de-sentencias

  }

```
for (cont = 0, suma =0; cont < = 10; cont++)
{
      System.out.println("cont vale:  " + cont
                                "\n y suma: "+ suma);
      suma = suma + cont;
}
```

Salida estándar

- **System.out.print ("cadena1”+arg1+arg2+”cadena2”+ arg3,..., argn);**
- **System.out.println ("cadena1”+arg1+arg2+”cadena2”+ arg3,..., argn);**

No necesita importar ninguna librería, aunque es de buen estilo de programación el incluirla.

```
//Nombre Programa: ejemploSalida.java
import java.io.*;
public class ejemploSalida
 {

    public static void  main(String [] Args)
       {
        System.out.println(“Hola caracola");
        System.out.println(“Adios blanca flor");
       }
}
```

Secuencias de escape

Cada cadena de System.out.print--( ) puede contener:

- **Texto**
- **secuencias de escape**

Secuencias de escape o Caracteres de control

- \n -----> Nueva Línea.
- \t -----> Tabulador horizontal.
- \r -----> Retorno de Carro.
- \f -----> Comienzo de Pagina.
- \b -----> Borrado a la Izquierda.
- \\ -----> El carácter barra inversa (\ ).
- \' -----> El carácter comilla simple (').
- \" -----> El carácter comilla doble (").

Entrada estándar: – I –

- Necesitaremos la librería **io**
- Necesitaremos activar el manejo de **excepciones**.

Escribid y ejecutad el siguiente programa y observad los resultados

```
import java.io.*;
public class inout
 {
        public static void main (String[] args) throws
                                java.io.IOException
        {
        final double PI = 3.1416, area;
        double radio;
        radio = System.in.read();
        area = PI*radio*radio;
        System.out.println("valor leido: "+(char)radio+ '?'+
                        radio + "\narea calculada: " + area);
        }
}
```

Quita el cast y ejecuta de nuevo y observa los nuevos resultados

Entrada estándar: – II –

- **Crearemos un objeto Scanner**
- **Leeremos los datos del objeto guardándolos en variables.**
- **Diremos el tipo de dato esperado.**

Tipo	Método a invocar
byte	teclado.nextByte();
short	teclado.nextShort();
int	teclado.nextInt();
long	teclado.nextLong();
float	teclado.nextFloat();
double	teclado.nextDouble();
boolean	teclado.nextBoolean();
string	teclado.nextLine();
char	charAt(índice) //para sacar un carácter de una cadena

Entrada estándar: – III–

```
import java.util.Scanner;
import java.io.*;
public class inout
 {
       public static void main (String[] args)throws
                                   java.io.IOException
      {
        final double PI = 3.1416;
        double radio, altura;
        Scanner teclado = new Scanner (System.in);
        System.out.print("Introduzca los datos del cilindro:"+'\n'+"Radio:  ");
        radio =teclado.nextDouble();
        System.out.print("Altura:  ");
        altura = teclado.nextDouble();
        System.out.print("el área del cilindro es:  ");
        System.out.println(PI*radio*radio*altura);
        radio = System.in.read();
        System.out.println("valor leido: " + (char)radio  + ','+ radio);
        teclado.close();
       }

}
```

Nota: con **new** se pide memoria o se abre una corriente y con **close** se devuelve o se cierra

Ejecuta este código y observa la salida

Estructura alternativa

if (expresión-condicional)
proposición1
[else]
proposición2

¿Dónde acaba el if? ¿y el else?

```
if (numero >0)
 {
   System.out.print("El número es positivo");
   System.out.println(numero);
 }
else
   System.out.print("El número es positivo");
 ......
```

Operador condicional:

(condicion?cierto: falso)

Los if pueden anidarse (Apartado 13.1, 2, pag 31), pero …¡OJO!

```
System.out.print("El número mayor es el de valor: ");
System.out.print( a > b ? a : b);
```

Estructuras iterativas

- while

  while (expresión-condicional)

  proposición1

```
acertado = false;
while(acertado == false)
{
 .....
   if (numero == numero secreto)
      acertado = true;
 .....
}
```

- do-while

  do

  proposición1

  while (expresión-condicional);

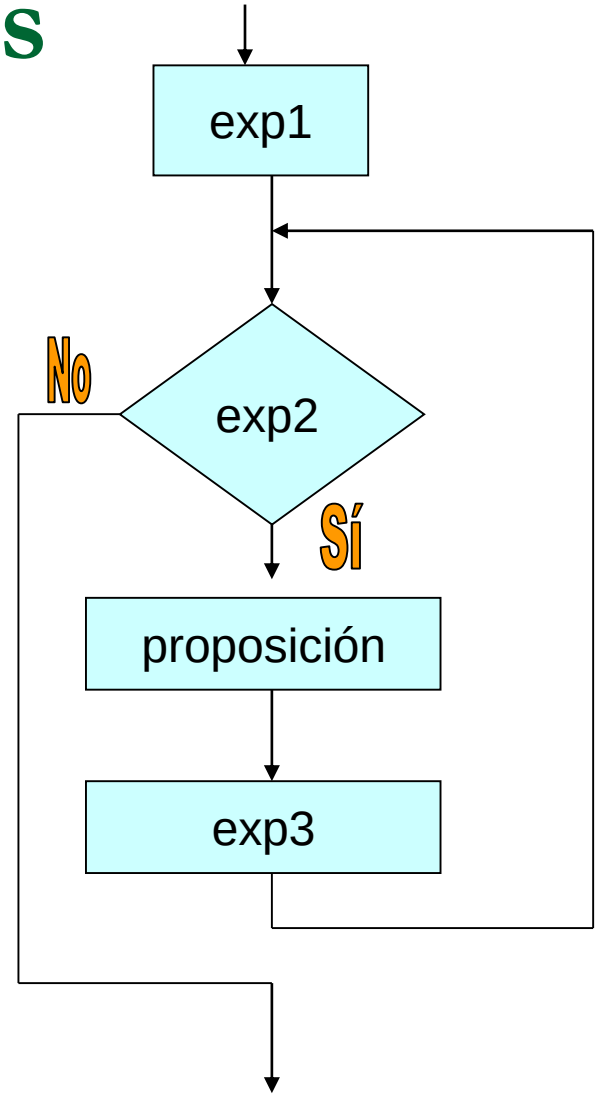
```
acertado = false;
do{
 .....
   if (numero == numero secreto)
      acertado = true;
 .....
} while(acertado == false);
```

Estructuras iterativas

- for

for ([exp1]; [exp2]; [exp3])
proposición

for (variable:estructura)
proposición



Ejemplos:

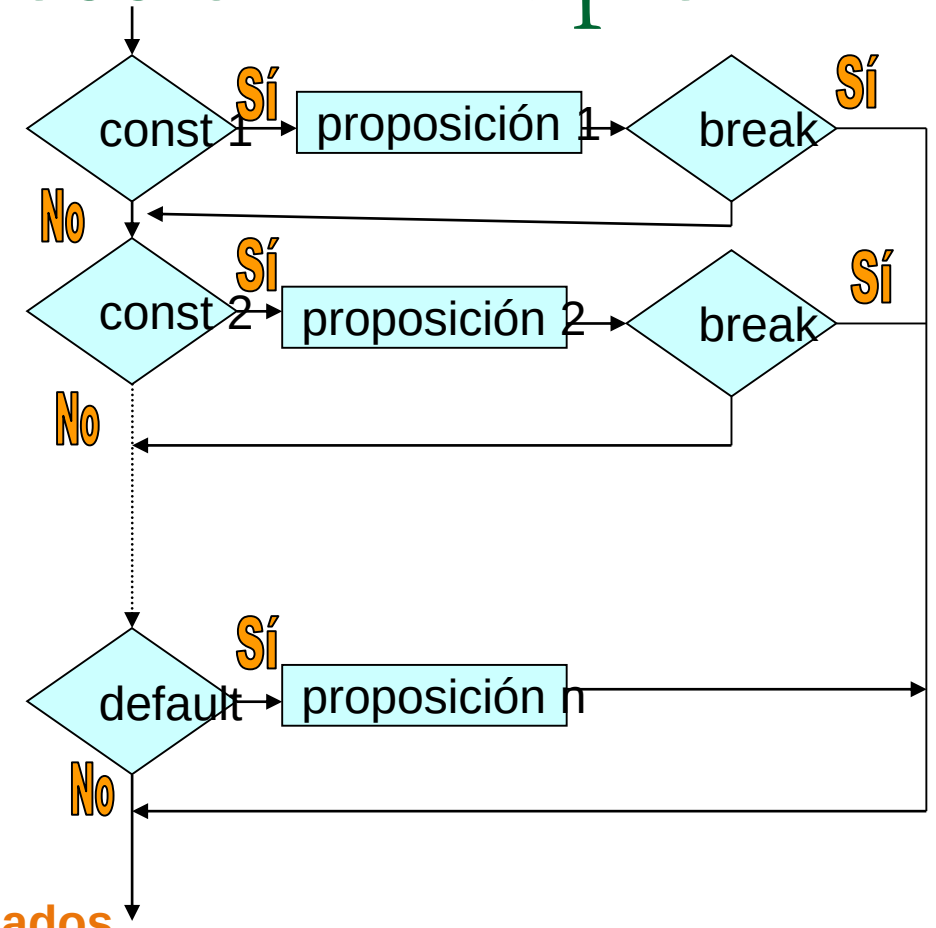
```
for (;;); //bucle infinito.
 ……..
for ( i =0; i > 10 && ind ==‘v’; i++)
 {
       ……
 }

for (String a : miArrayDeString)
{
        System.out.println(a);
}
```

Este es el bucle for mejorado (Java 6)

Estructura de selección múltiple

```
switch (expresión)
{
    case const 1:
        proposición1
        [break;]
    case const 2:
        proposición2
        [break;]
    …
    case const n:
            proposición n
            [break;]
    [default:]
            proposición
} /* Fin de switch. */
```



Los switch pueden usarse anidados

Si no existe algún **break** seguirá ejecutando proposiciones sin evaluar las constantes, hasta encontrar el siguiente **break** o la llave de cierre del **switch.**

Sentencias de salto

break;

sólo usaremos break en switch.

Salida de la estructura que lo contiene. Por ejemplo, se utiliza para detener completamente un bucle, pero

continue;

Usada normalmente en bucles, provoca la siguiente iteración cuando ocurre algún suceso.

goto

Aunque *goto* es una palabra reservada, actualmente Java no soporta la sentencia *goto*.

Consejos de implementación

- A partir de ahora se complica el diseño y la implementación, por lo que cuando implementemos iremos usando la técnica de Scarlet O'Hara.
- Daremos nombre a los módulos y en el código Java aparecerán como comentarios y además se visualizará en pantalla "en construcción"

"No basta adquirir conocimientos, es preciso usarlos".

Tulio Marco Cicerón

(Escritor, orador y político romano (106-43 a.C.))