UNIVERSIDAD DE MÁLAGA

TEMA 2

Introducción a un Lenguaje Programación. Programación Imperativa y Estructurada

Contenido del Tema

2.1. Introducción a Java
- 2.1.1. Características de Java
- 2.1.2. Ejemplo de programa en Java
- 2.1.3. Esquema general de un programa en Java (para Módulo 1)

2.2. Tipos básicos de datos
- 2.2.1. Tipos básicos en Java
- 2.2.2. Operadores
- 2.2.3. Conversiones de tipos

2.3. Identificadores, variables, constantes y asignaciones

2.4. Entrada y salida básicas

2.5. Flujo de control.

2.6. Expresiones lógicas o booleanas.

Samsung TECH INSTITUTE

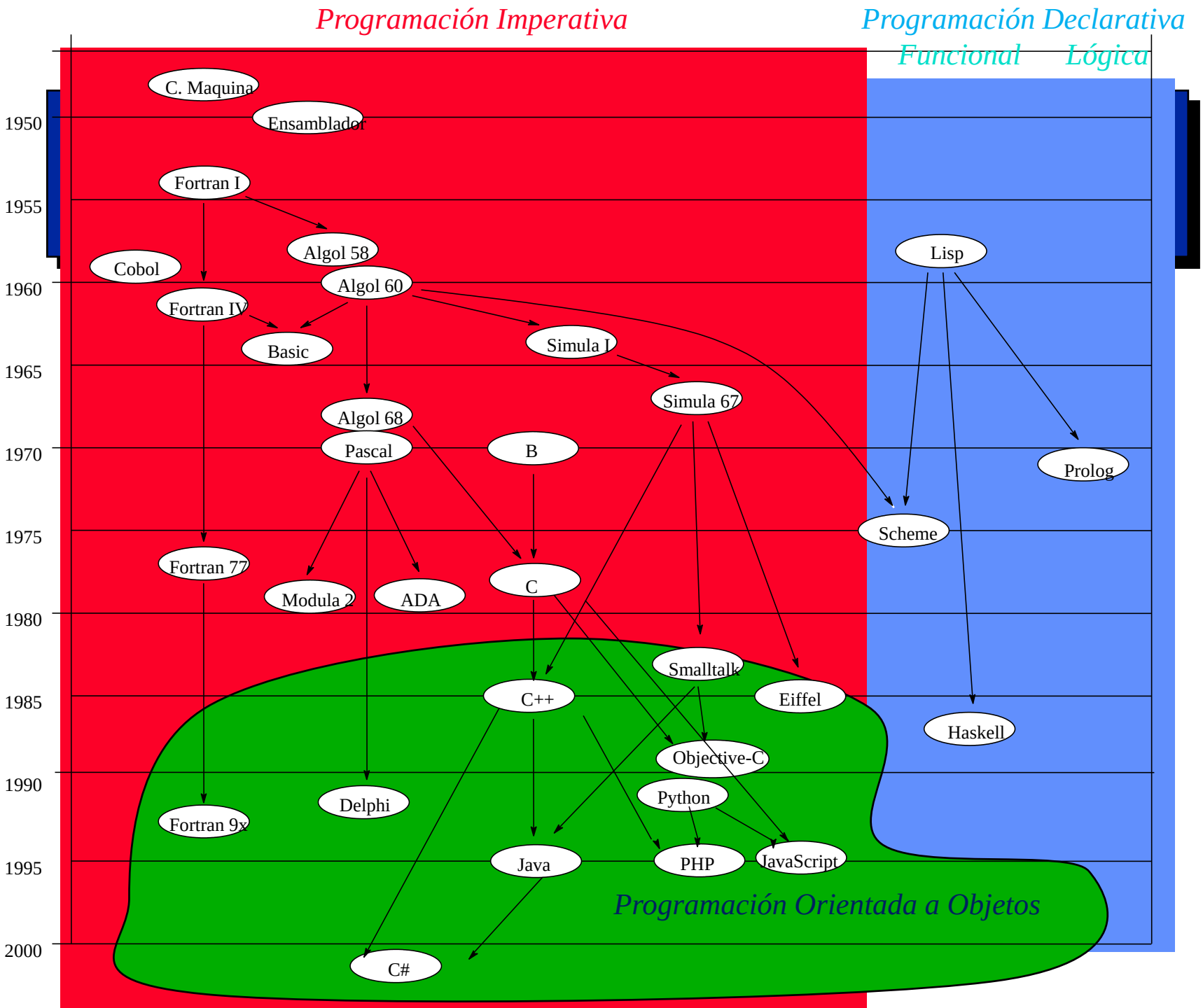
UNIVERSIDAD DE MÁLAGA

TEMA 2

Introducción a un Lenguaje Programación. Programación Imperativa y Estructurada

Contenido del Tema

Samsung TECH INSTITUTE


Programación Imperativa
Programación Declarativa
Funcional
Lógica
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
C. Maquina
Ensamblador
Fortran I
Cobol
Algol 58
Algol 60
Fortran IV
Basic
Simula I
Simula 67
Algol 68
Pascal
B
Fortran 77
Modula 2
ADA
C
C++
Smalltalk
Eiffel
Objective-C
Delphi
Python
Fortran 9x
Java
PHP
JavaScript
C#
Lisp
Prolog
Scheme
Haskell
Programación Orientada a Objetos

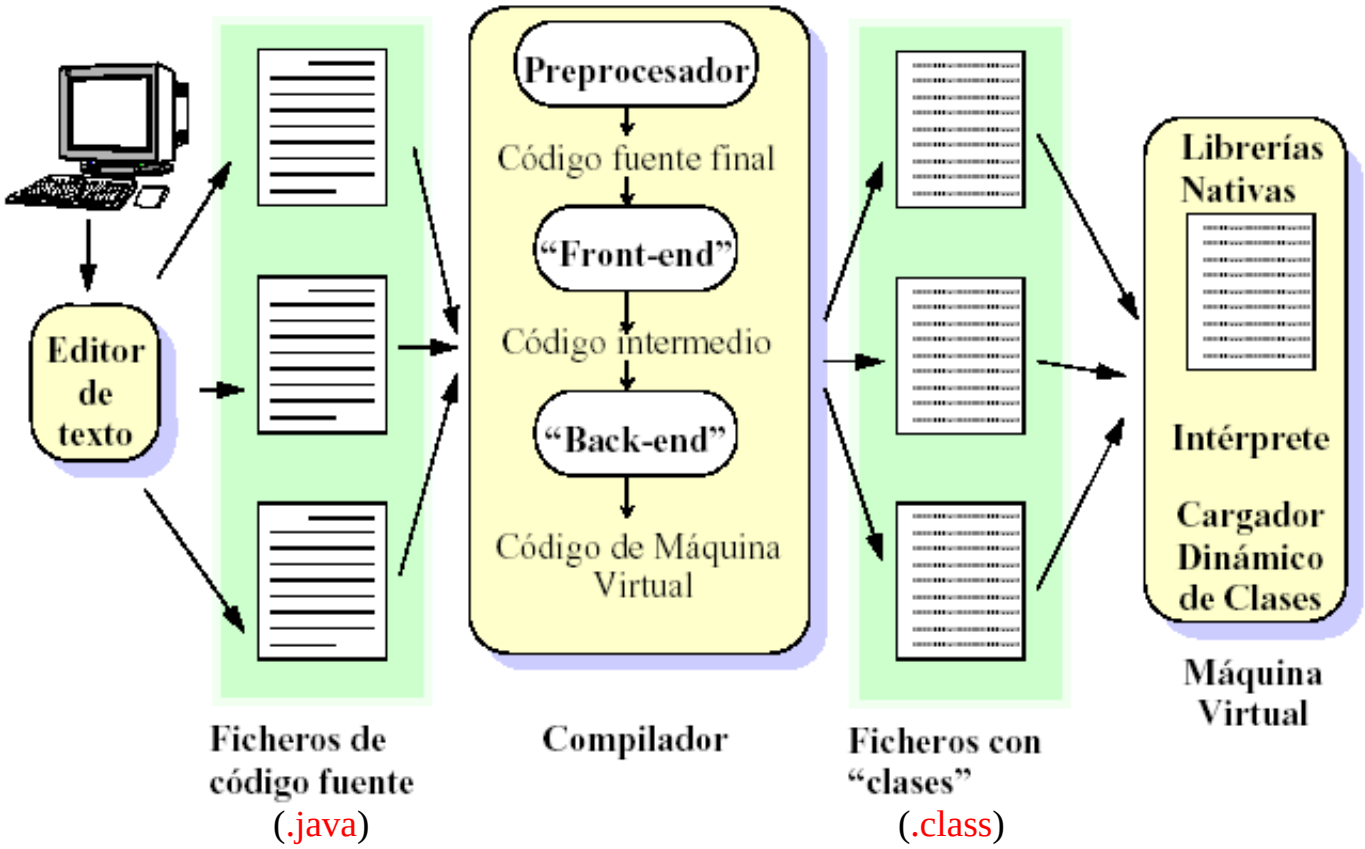
UNIVERSIDAD DE MÁLAGA

Características de Java

- Desarrollado por Sun. Aparece en 1995
- Basado en C++ (y algo en Smalltalk) eliminando:
  - definiciones de tipos de valores y macros,
  - punteros y aritmética de punteros,
  - necesidad de liberar memoria.
- Fiable y seguro:
  - memoria dinámica automática (no punteros explícitos)
  - comprobación automática de tamaño de variables
- Orientado a objetos con: (se estudiará en el Módulo 2)
  - herencia simple y polimorfismo de datos,
  - redefinición de métodos y vinculación dinámica.
  - concurrencia integrada en el lenguaje
  - interfaz gráfica integrada en el lenguaje
- Compilado "especial"
  - ficheros fuente **`.java`** se convierten en ficheros *bytecode* **`.class`**
- Interpretado
  - ficheros **`.class`** son interpretados por la máquina virtual de Java (JVM)

UNIVERSIDAD DE MÁLAGA

Características de Java



Compilación e Interpretación

Características de Java

Programa en Java

- Formado por una o varias clases (el concepto de clase se profundizará en el Módulo 2 del curso).
- Existirá una clase pública distinguida que contiene el método de clase:

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

que desencadena la ejecución del programa.

- Las demás clases pueden estar definidas *ad hoc* (por el programador) o pertenecer a una biblioteca de clases.

Ejemplo de Programa en Java

```java
import java.util.Scanner;

// Programa que convierte pulgadas a centímetros
public class PulgadasCentimetros {

        static final double CTMS_PULGADA = 2.54;

        public static void main(String[] args) {

                double centimetros,pulgadas;
                Scanner teclado = new Scanner(System.in);

                System.out.println("Programa para convertir pulgadas a centímetros:\n");
                System.out.print("Introduzca la medida en pulgadas: ");
                pulgadas = teclado.nextDouble();
                centimetros = pulgadas * CTMS_PULGADA;
                System.out.println("Esa medida en centímetros es: " + centimetros);
                teclado.close();
        }
}
```

Ejemplo de Programa en Java

```
import java.util.Scanner;

// Programa que convierte pul
public class PulgadasCentimet

        static final double

        public static void m

                double centimetros,pulgadas;
                Scanner teclado = new Scanner(System.in);

                System.out.println("Programa para convertir pulgadas a centímetros\n");
                System.out.print("Introduzca la medida en pulgadas: ");
                pulgadas = teclado.nextDouble();
                centimetros = pulgadas * CTMS_PULGADA;
                System.out.println("Esa medida en centímetros es: " + centimetros);
                 teclado.close();
        }
 }
```

Ejecución

Programa para convertir pulgadas a centímetros

Introduzca la medida en pulgadas: **42**
Esa medida en centímetros es: 106.68

Esquema general de un programa en Java (Módulo 1)

```
Importación de elementos de biblioteca necesarios

public class NombreClase {

    Declaración de constantes

    public static void main(String[] args) {

      Declaraciones de variables

      Secuencia de sentencias (también se pueden declarar
                               variables dentro)
    }
    Declaración de métodos auxiliares (Tema 3)
}
```

Fichero: NombreClase.java

Tipos Básicos de Datos

En los algoritmos se manipulan datos.

Características de los datos.

- Nombre.
- Tipo.
- Valor:
  - Variables.
  - Constantes.

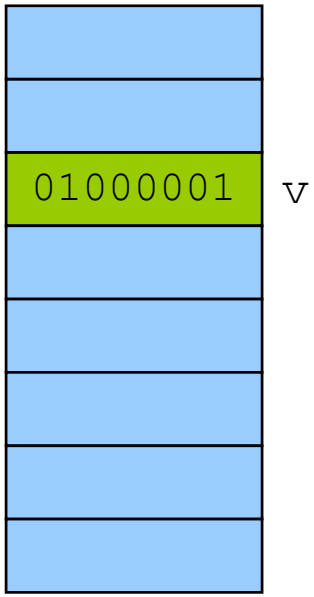
Tipos Básicos de Datos

El tipo de un dato determina

- El **conjunto de valores** que puede tomar.
- Las **operaciones** que se le pueden aplicar.
- El **espacio** que será necesario reservar **en memoria** para albergarlo.
- La **interpretación** del valor almacenado en memoria.

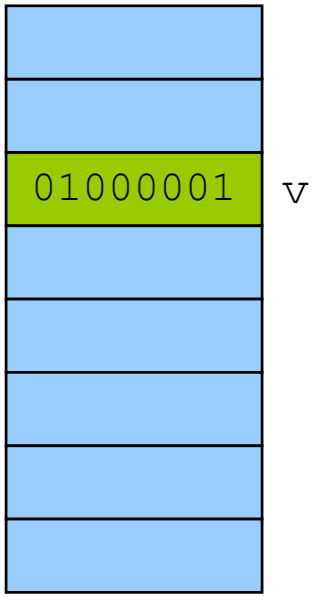
Tipos Básicos de Datos

- ¿Qué contiene la variable v?



Tipos Básicos de Datos

- ¿Qué contiene la variable v?

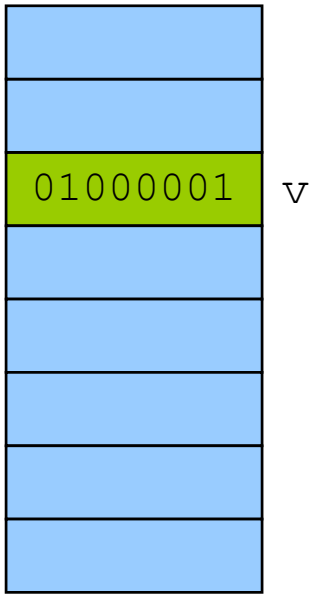


¿Si v es de tipo entero?

Entonces vale ...

Tipos Básicos de Datos

- ¿Qué contiene la variable v?

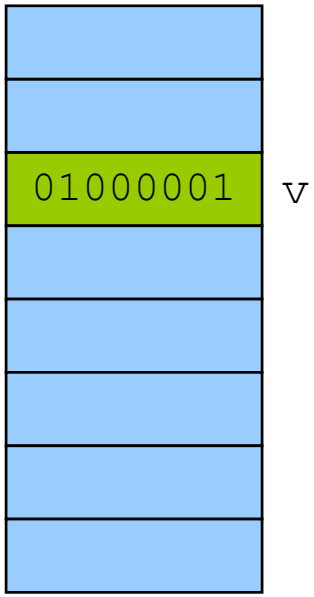


¿Si v es de tipo entero?

Entonces vale 65

Tipos Básicos de Datos

- ¿Qué contiene la variable v?



¿Si v es de tipo entero?

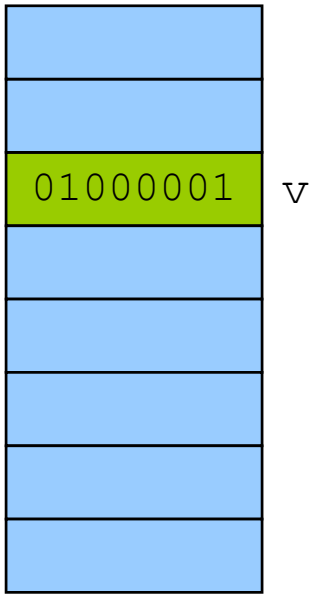
Entonces vale 65

¿Si v es de tipo carácter?

Entonces vale ...

Tipos Básicos de Datos

- ¿Qué contiene la variable v?



¿Si v es de tipo entero?

Entonces vale 65

¿Si v es de tipo carácter?

Entonces vale 'A'

Pues 01000001 (65) es el valor del carácter 'A' en la tabla ASCII y en Unicode.

UNIVERSIDAD DE MÁLAGA

Tipos Básicos en Java

- Sólo existen los siguientes tipos básicos:

`byte` (entero de 8 bits)	**`short`** (entero de 16 bits)
`int` (entero de 32 bits)	**`long`** (entero de 64 bits)
`float` (decimal de 32 bits)	**`double`** (decimal de 64 bits)
`char` (Unicode de 16 bits)	**`boolean`** (**`true`**, **`false`**)

- El número de bits dedicado es independiente de las plataformas sobre las que se ejecuten los programas.
- No se pueden definir más tipos básicos.

UNIVERSIDAD DE MÁLAGA

Tipos Básicos en Java

byte short int long

- Valores: números enteros
- Operadores aritméticos : +, -, *, /, %
- Operadores relacionales: ==, >, <, >=, <=, !=
- Representación: secuencia de dígitos (precedida por el signo – para los negativos)
- Cardinalidad: 2^N (N es el número de bits dedicados a su representación interna en memoria)
- Rango: $[-2^{N-1}..2^{N-1}-1]$

UNIVERSIDAD DE MÁLAGA

Tipos Básicos en Java

float double

- Valores: números reales
- Operadores aritméticos : +, -, *, /
- Operadores relacionales: ==, >, <, >=, <=, !=
- Representación: secuencia de dígitos (precedida por el signo – para los negativos) con el obligado punto decimal y con un posible factor de escala

0.0 1.5 -1.50 1.5E2 2.34E-2

UNIVERSIDAD DE MÁLAGA

Tipos Básicos en Java

char

- Valores: caracteres Unicode de 16 bits
- Operadores relacionales: ==, >, <, >=, <=, !=
- Representación: carácter entre apóstrofes. Ej. ‘a’

Tipos Básicos en Java

boolean

- Valores: valores verdadero y falso (*true* y *false*)
- Operadores relacionales: ==,!=
- Operadores booleanos : &&, ||, !

Expr.1	Expr.2	`&&`	`||`	`!Expr.1`
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Tipos Básicos en Java

Problemas derivados de la implementación de los tipos numéricos.

- El número de bits utilizados para representar los números enteros y reales es limitado.
- Si intentamos almacenar un valor cuya representación en binario excede el número de bits reservado para el tipo de datos de dicho valor se produce un DESBORDAMIENTO (OVERFLOW).
- Ejemplo:

  Si trabajamos con enteros y N=5 bits:

  7 0111

  10 1010 + El resultado en realidad sería 1 y no 17

  ----

  17 10001

Tipos Básicos en Java

Problemas derivados de la implementación de los tipos numéricos.

- En los reales, además del problema anterior, se produce la denominada PÉRDIDA DE PRECISIÓN: no se pueden representar todos los dígitos decimales debido a que el número de bits utilizados para ello es limitado.
- Ejemplo:

  Si la precisión es de 4 dígitos y las variables X, Y, Z tienen los valores 9.900, 1.000 y -0.999 respectivamente:

  (X+Y)+Z = 10.900 + (-0.999) = 9.910

  X+(Y+Z) = 9.900 + 0.001 = 9.901
- Nunca debemos realizar comparaciones de igualdad (==) o distinto (!=) con reales, debido a la pérdida de precisión.

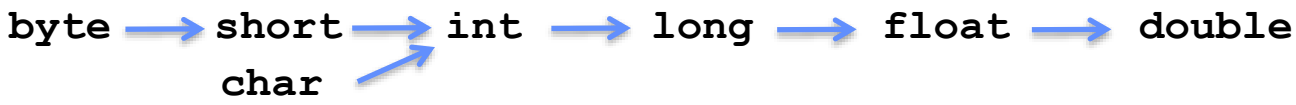
Operadores

De Mayor a Menor Precedencia

! -	unario	asociativo de derecha a izquierda
* / %	binarios	asociativos de izquierda a derecha
+ -	binarios	asociativos de izquierda a derecha
< <= > >=	binarios	asociativos de izquierda a derecha
== !=	binarios	asociativos de izquierda a derecha
&&	binario	asociativo de izquierda a derecha
||	binario	asociativo de izquierda a derecha
=	binario	asociativo de derecha a izquierda

Conversiones de tipos

- Se producen conversiones de tipos de forma ***implícita*** en ciertos contextos.
  - Siempre a ***tipos más amplios*** siguiendo la ordenación:



UNIVERSIDAD DE MÁLAGA

Conversiones de tipos

- La conversión implícita se produce en los siguientes contextos:
  - **Asignaciones** (el tipo de la expresión se promociona al tipo de la variable de destino)
  - **Invocaciones de métodos** (los tipos de los parámetros reales se promocionan a los tipos de los parámetros formales)
  - **Evaluación de expresiones aritméticas** (los tipos de los operandos se promocionan al del operando con el tipo más general y, como mínimo se promocionan a `int`)
  - **Concatenación de cadenas** (los valores de los argumentos se convierten en cadenas)

Conversiones de tipos

- Se permiten conversiones ***explícitas*** en sentido contrario mediante la construcción:

  (<tipo>) <expresión>

  Sólo se comprueban durante la ejecución

```
(int) 3.6
```

Identificadores

- Un *identificador* (nombre) es una secuencia arbitraria de caracteres Unicode: letras, dígitos, subrayado, …

  double centimetros;

- No debe comenzar por dígito ni coincidir con alguna palabra reservada.
- Los identificadores dan nombre a:

  variables, constantes, métodos, clases e interfaces.

- Por convenio:
  - Nombres de variables y métodos en minúsculas. Si son compuestos, las palabras no se separan y comienzan con mayúscula.

    long valorMaximo;

  - Nombres de clase igual, pero comenzando con mayúscula.

    class Principal

  - Nombres de constantes todo en mayúsculas. Si son compuestos, las palabras se separan con subrayados.

    static final double CTMS_PULGADA

Variables

```
tipo nombre; // valor inicial indefinido
```

```
tipo nombre1, nombre2;
```

```
tipo nombre = valor_inicial;
```

```
tipo nombre1 = valor_inicial, nombre2;
```

```
char c; // c con valor inicial indefinido

int contador = 0; // contador tiene valor inicial 0
```

- El compilador producirá un error si se intenta utilizar una variable no inicializada en algún punto donde se requiere su valor (en Módulo 2 se verán variables que se inicializan automáticamente a un valor predefinido)

Constantes

- Una variable se puede declarar como constante precediendo su declaración con la etiqueta **`final`** (en este Módulo 1 también se pondrá la etiqueta **`static`**):

```
static final int MINIMO = 5;
```

- El valor al que se inicializa puede ser un literal, otra constante o una expresión constante.

```
static final int MAXIMO = MINIMO * 2;
```

- Cualquier intento de cambiar el valor de una constante después de su inicialización produce un error en tiempo de compilación

Sentencia de asignación

```
variable = expresión
```

```
centimetros = pulgadas * CTMS_PULGADA;
```

- Una *expresión* es una combinación de
  - literales,
  - variables,
  - constantes,
  - operadores,
  - Invocación o llamadas a métodos (envíos de mensajes)

Sentencias de asignación especiales

```
++variable;          // variable = variable + 1;
--variable;          // variable = variable - 1;
variable++;          // variable = variable + 1;
variable--;          // variable = variable - 1;
variable += expresion; // variable = variable + expresion;
variable -= expresion; // variable = variable - expresion;
variable *= expresion; // variable = variable * expresion;
variable /= expresion; // variable = variable / expresion;
variable %= expresion; // variable = variable % expresion;
```

E/S básicas

- Ya hemos visto lo simple que es escribir datos por pantalla:

  `System.out.println`

  `System.out.print`

```
System.out.println("Esa medida en centímetros es: " + centimetros);
```

- Con **`System.out`** accedemos a un elemento de la clase `System` conocido como el flujo de salida estándar (texto por la pantalla).

E/S básicas

- De la misma forma , existe un **`System.in`** para el flujo de entrada estándar (texto desde el teclado)
- Pero Java no fue diseñado para este tipo de entrada textual desde el teclado (modo consola).
- Por lo que **`System.in`** nunca ha sido simple de usar para este propósito.

E/S básicas

- Afortunadamente, existe una forma fácil de leer datos desde la consola:

  clase **`Scanner`**

- Al construir un objeto **`Scanner`**, se le pasa como argumento **`System.in`**:

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

- En el Módulo 2 se abordarán en profundidad los conceptos de clase y objeto

E/S básicas

- La clase **`Scanner`** dispone de métodos para leer datos de diferentes tipos (por defecto los separadores son los espacios, tabuladores y nueva línea, aunque se pueden establecer otros (se verá en el Módulo 2)):
  - `nextDouble()` lee y devuelve el siguiente número real (`double`)
  - `nextInt()` lee y devuelve el siguiente entero (`int`)
  - ...

Consultar API

E/S básicas

- Produce **NoSuchElementException** si no hay más elementos que leer
- Produce **InputMismatchException** si el dato a leer no es el esperado.
  - Por ejemplo si se utiliza `nextInt()` y lo siguiente no es un entero

Al final del tema se abordarán algunos aspectos sobre excepciones. Se estudiarán más a fondo en el Módulo 2

E/S básicas

- La clase **`Scanner`** también dispone de métodos para consultar si el siguiente dato disponible es de un determinado tipo:
  - `hasNextDouble()` devuelve *true* si el siguiente dato es un `double`
  - `hasNextInt()` devuelve *true* si el siguiente dato es un `int`
  - ...

UNIVERSIDAD DE MÁLAGA

E/S básicas

- Existe una operación para “cerrar” el objeto Scanner: `close()`

- La clase Scanner no sólo sirve para leer de teclado.
- Se pueden construir objetos Scanner sobre cadenas de caracteres y sobre objetos de otras clases de entrada (se verá en el Módulo 2).

UNIVERSIDAD DE MÁLAGA

Flujo de Control

- **Flujo de Control: orden de ejecución de las sentencias**
- **Con las sentencias estudiadas hasta ahora (asignación, entrada y salida):**
  - **Flujo de Control = ORDEN FÍSICO (disposición de las sentencias)**
  - Recurso insuficiente:
    - No permite resolver problemas que exigen una toma de decisión.

      Ej: leer dos números y determinar cuál es el mayor
    - No permite la repetición de un conjunto de acciones un número de veces indeterminado.

      Ej: suma de números leídos por teclado hasta la introducción de 0

Flujo de Control

Necesidad de sentencias o estructuras de control nuevas

Ejecutar unas sentencias u otras, según se cumpla o no una determinada condición → **Estructura selectiva**

Ejecutar un número de veces unas sentencias → **Estructura iterativa**

Capacidad de realizar preguntas sobre datos del programa → **Expresiones lógicas**

Usan las expresiones lógicas

UNIVERSIDAD DE MÁLAGA

Expresiones Lógicas

- También llamadas Condicionales o Booleanas.

- Expresión (pregunta) que una vez evaluada puede ser:

  true

  false

- Uso fundamental:
  - Sentencias de selección
  - Sentencias de iteración

UNIVERSIDAD DE MÁLAGA

Expresiones Lógicas

- Una expresión lógica se puede formar:
  - Usando una variable booleana
  - Usando operadores relacionales
  - Usando operadores booleanos
  - Usando ambos operadores

UNIVERSIDAD DE MÁLAGA

Expresiones Lógicas

Variable Booleana

- Declaración: `boolean vble;`
- Sólo puede contener **`true`** o **`false`**.
- Se le asigna un valor mediante el operador asignación (=).

```
vble = true;
```

- Se puede realizar una pregunta (expresión lógica) utilizando únicamente una variable booleana.

```
vble
```

UNIVERSIDAD DE MÁLAGA

Expresiones Lógicas

Operadores Relacionales

- `==, !=, <, <=, >, >=`
- Permiten comparar dos datos del mismo tipo.
- El resultado de la comparación será **`true`** o **`false`**
- Ejemplos:

```
int n, m, x;
3 > 8
n >= 4
n*m <= x+18
```

- Si **`boolean`** `v` ¿es posible realizar la asignación: `v = n >= 4`?

UNIVERSIDAD DE MÁLAGA

Expresiones Lógicas

Operadores Booleanos

- `&&, ||, !`
- Se aplican sobre expresiones lógicas.
- El resultado será **`true`** o **`false`**
- Ejemplos:

```
boolean v,w;
v && w
!w
```

UNIVERSIDAD DE MÁLAGA

Expresiones Lógicas

Operadores Booleanos (y Relacionales)

- Ejemplos:

```
boolean v;
int x, y, z;

x == 3 &&  z != 4

!(v && (x <= y))
```

Expresiones Lógicas

Equivalencias

```
boolean A,B;
int X,Y;
```

- `!(A==B)` ↔ `A!=B`
- `!(A && B)` ↔ `!A || !B`
- `!(A || B)` ↔ `!A && !B`
- `!(!B)` ↔ `B`
- `!(X>Y)` ↔ `X<=Y`
- `!(X<Y)` ↔ `X>=Y`
- `B==true` ↔ `B`
- `B==false` ↔ `!B`

Expresiones Lógicas

Una expresión lógica se evalúa en CORTOCIRCUITO

Se evalúa de izquierda a derecha y llegado el punto en el que ya se conoce el resultado de la expresión completa (el resto de la misma no tiene influencia en el resultado), no se continúa evaluando

Ejemplo:

```
(j>0) && (100/j<2)
```

Expresiones Lógicas Cuestiones

a) Si **boolean** `valor`, ¿cómo se puede poner de otra forma la expresión lógica `valor==`**`false`**?

Solución : !valor

b) ¿y `valor==`**`true`**?

Solución: valor

c) boolean `prueba,valor;`

¿podemos hacer `prueba = valor ==` **`false`**?

¿a qué es equivalente?

Solución: prueba = !valor

Expresiones Lógicas Cuestiones

d) Si **boolean** `valor;`
int `x,y;`

¿`(x<y) && (y<x)` es equivalente a?

Solución: false

¿`(x<=y) || (y<=x)` es equivalente a?

Solución: true

e) Si **boolean** `x,y` ¿qué relación hay entre I) y II)?

I) `x != y`

II) `(x || y) && !(x && y)`

Solución: son equivalentes

UNIVERSIDAD DE MÁLAGA

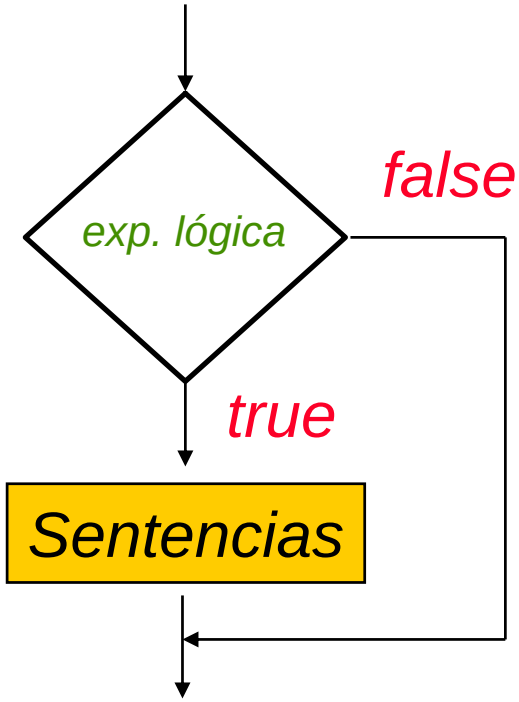
Estructuras de Selección

- Sirven para ejecutar unas sentencias u otras, según se cumpla o no una determinada condición (expresión lógica) o bien según sea el valor de una determinada expresión.
- 3 tipos de sentencias de selección:

Sentencias de selección simple
Sentencias de selección doble
Sentencias de selección múltiple

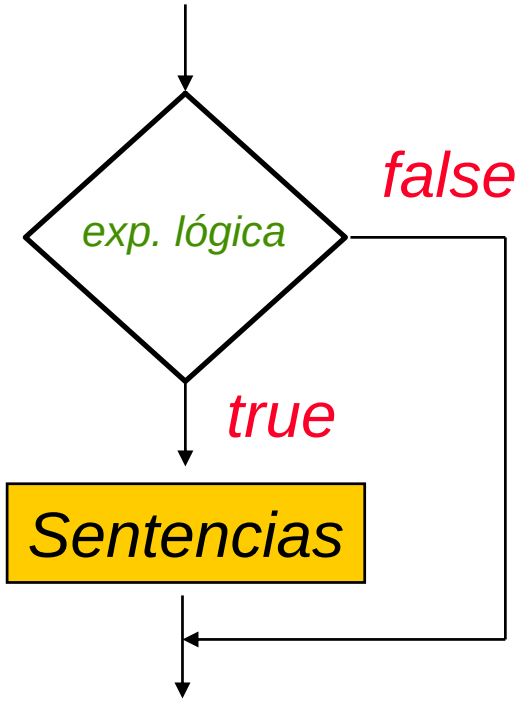
Estructuras de Selección

Sentencia de selección simple




```
if (expresión lógica) {
   Sentencias
}
```

Estructuras de Selección



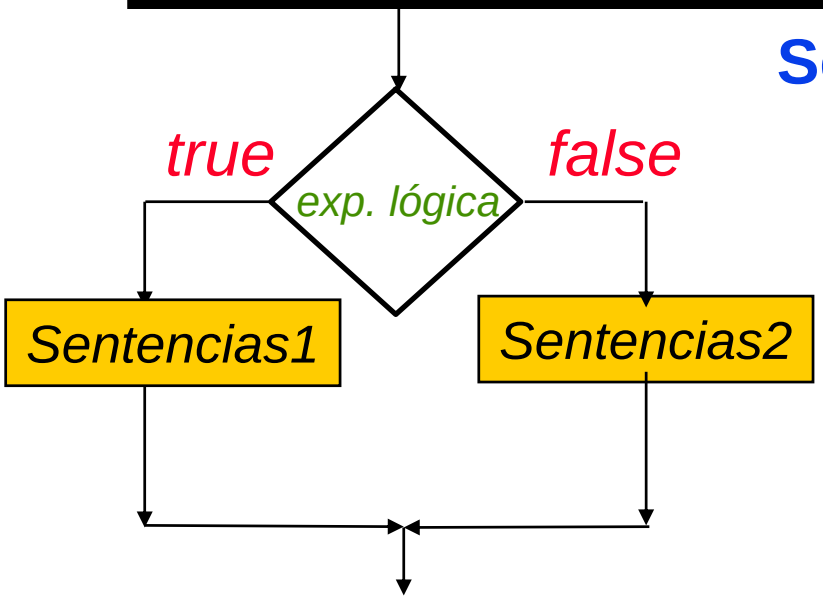
Sentencia de selección simple

```
if (expresión lógica) {
   Sentencias
}
```

Ejemplo: **valor absoluto de un número**

```
num = teclado.nextInt();
if (num < 0) {
   num = -num;
}
System.out.println(num);
```

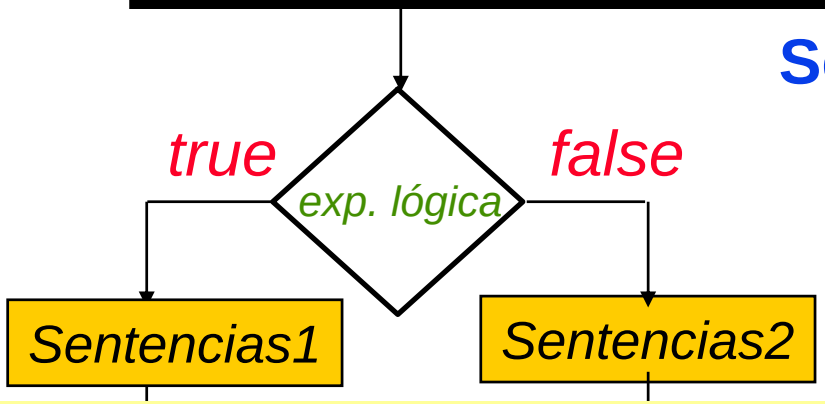
Estructuras de Selección



Sentencia de selección doble

```
if (expresión lógica) {
   Sentencias1
} else {
   Sentencias2
}
```

Estructuras de Selección



Sentencia de selección doble

```
if (expresión lógica) {
   Sentencias1
} else {
```

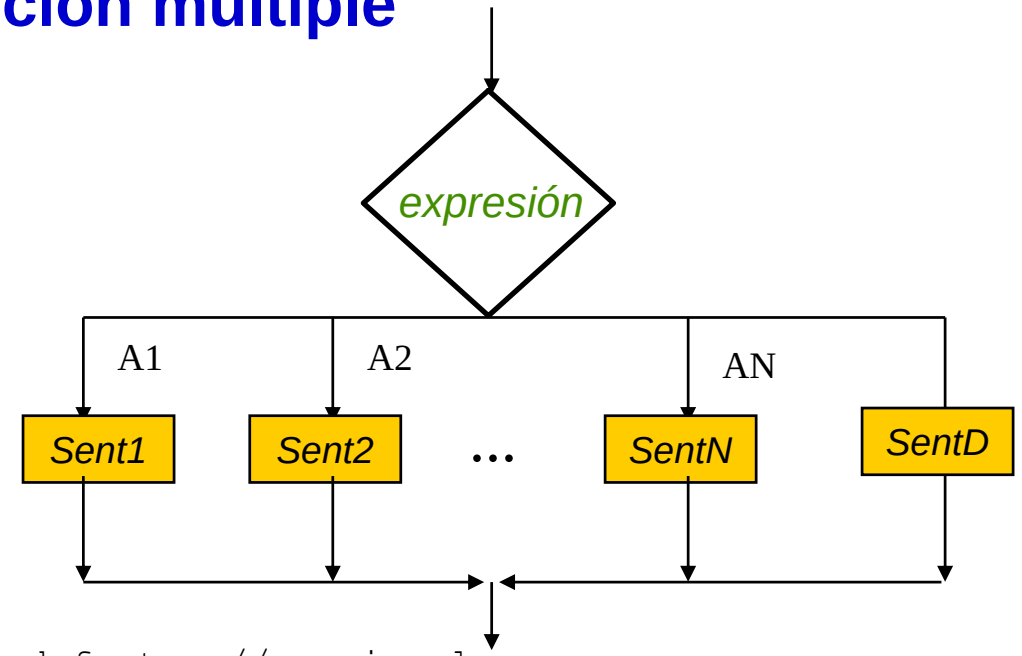
Ejemplo: **ver si un número es divisible por otro**

```
num1 = teclado.nextInt();
num2 = teclado.nextInt();
if (num1 % num2 == 0) {
   System.out.println(num1+" es divisible por "+num2);
} else {
   System.out.println(num1+" no es divisible por "+num2);
}
```

Estructuras de Selección

Sentencia de selección múltiple

```
switch (expresión) {
    case alternat1 :   Sentencias 1
                       break;
    case alternat2 :   Sentencias 2
                       break;
    case alternat3 :   Sentencias 3
                       break;
    …
    case alternatN :   Sentencias N
                       break;
    default:           Sentencias por defecto  // opcional
                       break;
}
```



- `expresión` debe ser de tipo `char`, `byte`, `short` o `int`
- A partir de Java 1.7 también se admiten cadenas de caracteres (`String`)

Estructuras de Selección

Ejemplo: **Realizar una operación**

```
char operador;
int resultado,a,b;

.....
switch (operador) {
    case'+' : resultado = a+b;
              break;
    case'-' : resultado = a-b;
              break;
    case'*' : resultado = a*b;
              break;
    case'/' : resultado = a/b; // suponemos b != 0
              break;
    default: .....
             break;
}
.....
```

Estructuras de Selección

Anidamientos

```
if ((1 <= dia) && (dia <= 7)) {
      System.out.println("Día valido");
      if  (dia <= 5) {
            System.out.println("Día laborable");
      } else {
            System.out.println("Día no laborable");
      }
} else {
      System.out.println("Día no valido");
}
```

Estructuras de Selección

Anidamientos

```
if (nota == 10.0) {
    System.out.println("Matrícula de Honor");
} else {
    if (nota >= 9.0) {
            System.out.println("Sobresaliente");
    } else {
            if (nota >= 7.0) {
                    System.out.println("Notable");
            } else {
                    if (nota >= 5.0) {
                            System.out.println("Aprobado");
                    } else {
                            System.out.println("Suspenso");
                    }
            }
    }
}
```

UNIVERSIDAD DE MÁLAGA

Estructuras de Selección

Anidamientos

```
if (nota == 10.0) {
   System.out.println("Matrícula de Honor");
} else if (nota >= 9.0) {
   System.out.println("Sobresaliente");
} else if (nota >= 7.0) {
   System.out.println("Notable");
} else if (nota >= 5.0) {
   System.out.println("Aprobado");
} else {
   System.out.println("Suspenso");
}
```

Estructuras de Iteración

También se denominan bucles

Utilidad: repetir una serie de sentencias un número de veces, que dependerá de una determinada condición de control (expresión lógica).

El número de repeticiones o iteraciones puede ser:

1. No predeterminado. Depende del efecto de las sentencias ejecutadas en las distintas iteraciones
2. Fijo o predeterminado. No depende de las sentencias ejecutadas en cada iteración. Se conoce con anterioridad a iniciar las iteraciones.

Estructuras de Iteración

También se denominan bucles

Utilidad: repetir una serie de sentencias un número de veces, que dependerá de una determinada condición de control (expresión lógica).

El número de repeticiones o iteraciones puede ser:

1. No predeterminado. Depende del efecto de las sentencias [redacted] as iteraciones

   while do ... while

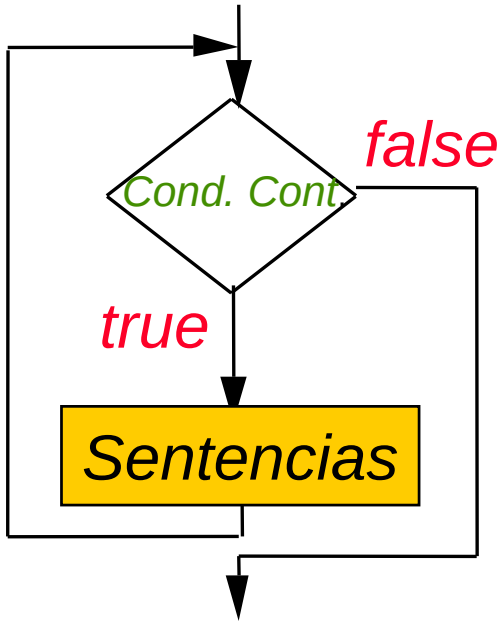
2. Fijo o predeterminado. No depende de las sentencias ejecutadas [redacted] a iteración. Se conoce con anterioridad [redacted] las iteraciones.

   for

Estructuras de Iteración

Sentencia de Iteración *while*

```
while (condición de control) {
      Sentencias      // Cuerpo del bucle
}
```



Estructuras de Iteración

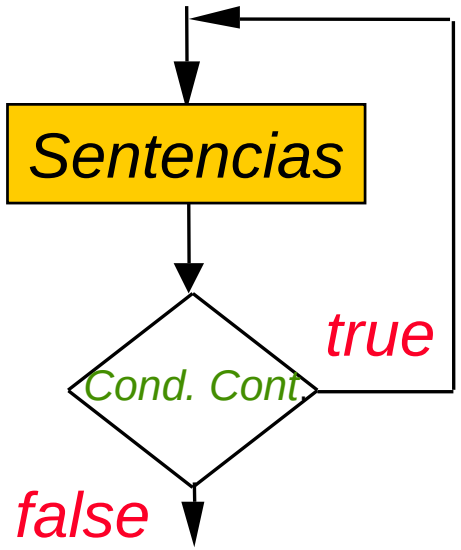
Ejemplo: **Cálculo del Máximo Común Divisor de dos números naturales (> 0) mediante *Algoritmo de Euclides*.**

```
System.out.print("Introduzca dos números: ");
num1 = teclado.nextInt();
num2 = teclado.nextInt();
while (num1 != num2) {
   if (num1 > num2) {
        num1 = num1 - num2;
   } else {
        num2 = num2 - num1;
   }
}
System.out.println("Su MCD es: " + num1);
```

Estructuras de Iteración

Sentencia de Iteración *do while*

```
do {
      Sentencias      // Cuerpo del bucle
} while (condición de control);
```



Estructuras de Iteración

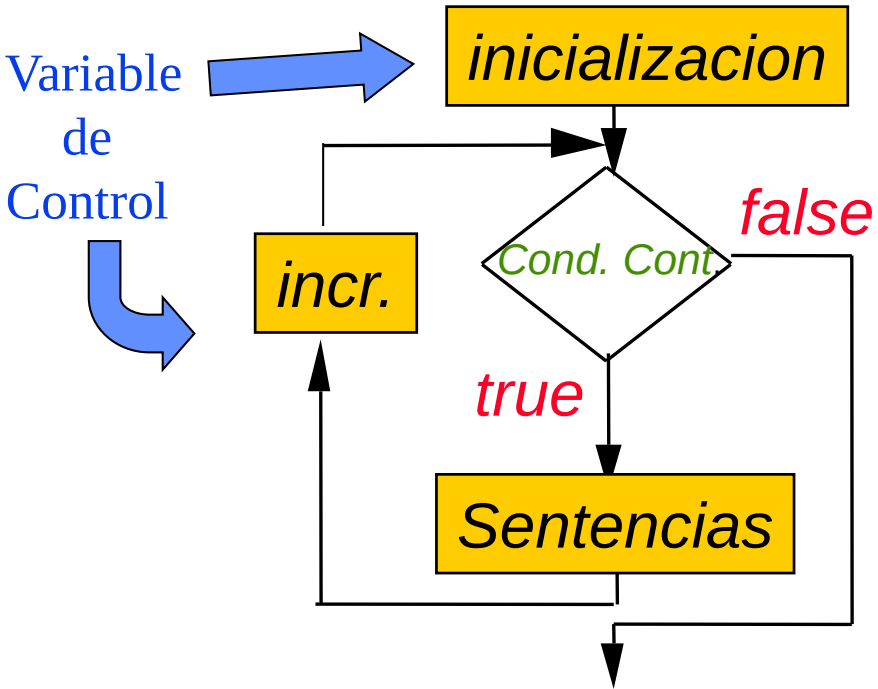
Ejemplo: **Leer dos números naturales para hacer su división entera, asegurándonos de que los dos son naturales y el divisor no sea cero**

```
do {
    System.out.print("Introduzca dividendo: ");
    dividendo = teclado.nextInt();
} while (dividendo < 0); // no negativo
do {
   System.out.print("Introduzca divisor: ");
   divisor = teclado.nextInt();
} while (divisor <= 0); // ni negativo ni 0
```

Estructuras de Iteración

Sentencia de Iteración *for*

```
for (inicializacion; cond. de control; incr.) {
      Sentencias        // Cuerpo del bucle
}
```



Estructuras de Iteración

Ejemplo: **Suma de los N primeros números naturales (sin contar el cero)**

```
import java.util.Scanner;

public class Principal {

        public static void main(String[] args) {
                Scanner teclado = new Scanner(System.in);
                int suma, n;
                do {
                    System.out.print("Introduzca N (> 0): ");
                    n = teclado.nextInt();
                } while (n <= 0);
                suma = 0;
                for (int cont = 1; cont <= n; cont++) {
                    suma = suma + cont;
                }
                System.out.println("La suma es: " + suma);
                teclado.close();
        }
}
```

declaración

UNIVERSIDAD DE MÁLAGA

Estructuras de Iteración

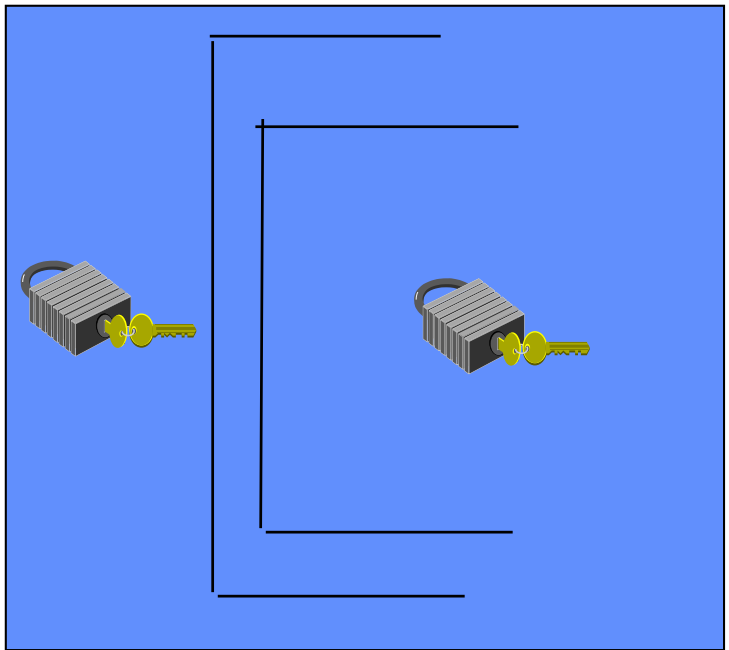
Para el diseño correcto de un bucle se deben considerar los siguientes aspectos:

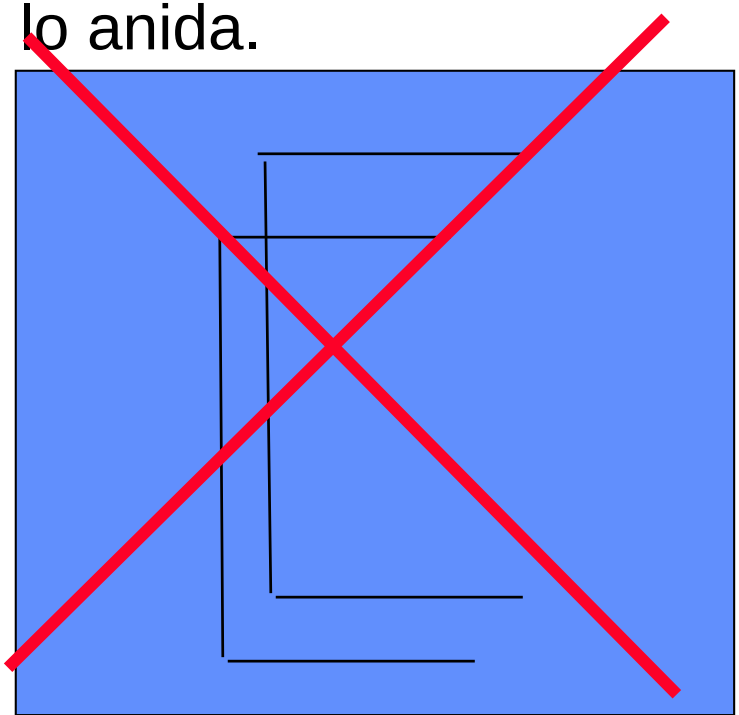
- Diseño del flujo de control, determinando:
  - a) Condición de terminación del bucle.
  - b) La forma de inicializar y actualizar esa condición.
- Diseño del procesamiento interior del bucle, determinando:
  - a) El proceso que se repite
  - b) La forma de iniciarlo y actualizarlo.

Estructuras de Iteración

Anidamientos

- Los bucles también pueden anidarse.
- El comienzo y final de un bucle anidado deben estar dentro del mismo bucle que lo anida.

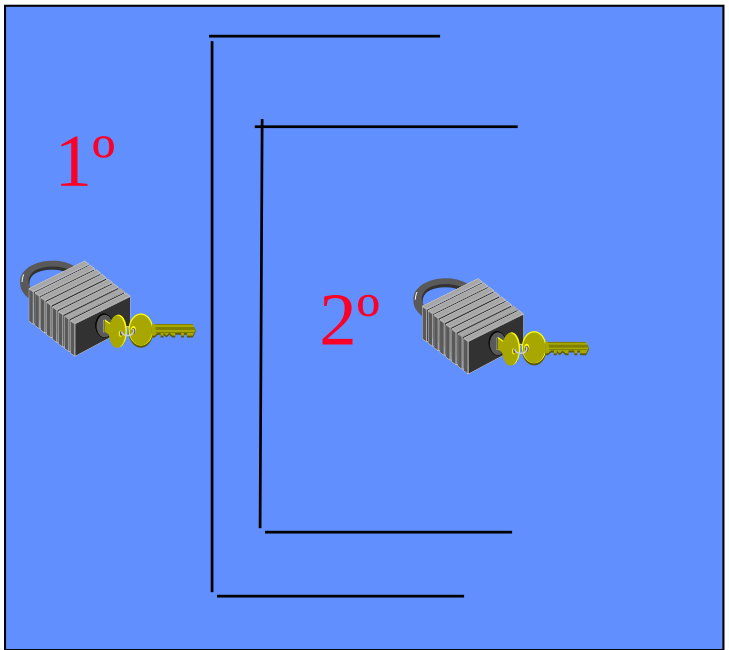




Estructuras de Iteración

Anidamientos

- Al diseñar una estructura iterativa anidada debemos empezar diseñando la más externa y terminar con las más internas.



UNIVERSIDAD DE MÁLAGA

Control de Errores. Excepciones

- Durante la ejecución de un programa se pueden dar situaciones inesperadas o errores inesperados que:
  - Desencadenarán la finalización del programa

    Ej. División por cero
  - Provocarán un malfuncionamiento del programa

    Ej. Lectura de un número negativo cuando se intenta leer un natural

Control de Errores. Excepciones

- El programador puede utilizar las estructuras de selección y de iteración para controlar algunos de estos errores:

```
valor = numerador/denominador;
```

Control de Errores. Excepciones

- El programador puede utilizar las estructuras de selección y de iteración para controlar algunos de estos errores:

```
if (denominador == 0) {
   System.out.println("Error: división por cero");
}  else {
   valor = numerador/denominador;
}
```

UNIVERSIDAD DE MÁLAGA

Control de Errores. Excepciones

- El programador puede utilizar las estructuras de selección y de iteración para controlar algunos de estos errores:

```
int num;

System.out.println("Introduzca valor (>=0): ");
num = teclado.nextInt();
```

Control de Errores. Excepciones

- El programador puede utilizar las estructuras de selección y de iteración para controlar algunos de estos errores:

```
int num;

do {
   System.out.println("Introduzca valor (>=0): ");
   num = teclado.nextInt();
}  while (num < 0);
```

Control de Errores. Excepciones

- La utilización de **excepciones** es otro mecanismo para controlar los errores inesperados

```
if (denominador == 0) {
   System.out.println("Error: división por cero");
}  else {
   valor = numerador/denominador;
}
```

Control de Errores. Excepciones

- La utilización de **excepciones** es otro mecanismo para controlar los errores inesperados

El programador "lanza" la excepción

```
if (denominador == 0) {
    throw new Exception("Error división por cero\n");
}
valor = numerador/denominador;
```

Control de Errores. Excepciones

- La utilización de **excepciones** es otro mecanismo para controlar los errores inesperados

El sistema "lanza" una excepción si no se introduce un número entero

```
int num;

do {
    System.out.println("Introduzca valor (>=0): ");
    num = teclado.nextInt();
}  while (num < 0);
```

UNIVERSIDAD DE MÁLAGA

Control de Errores. Excepciones

- En el Módulo 2 se abordará el mecanismo para llevar a cabo la captura y tratamiento de excepciones lanzadas tanto por el programador como por el sistema.

Nota Final del tema

Un algoritmo que use tan sólo las estructuras de control tratadas en este tema (Selección e Iteración), se denomina estructurado.

Bohm y Jacopini demostraron que todo problema que pueda solucionarse con un número finito de pasos puede resolverse usando únicamente estas estructuras.

Esta es la base de la programación estructurada**.**