# Tema 1:
# Elementos de un programa informático

## 1º Diseño de Aplicaciones Web

# T1. Elementos de un programa informático

1. **Problemas, Algoritmos y Programas**
2. **Fases de la programación**
3. **Paradigmas de la programación**
4. **Lenguajes de programación**
5. **Java**
6. **Tipos de datos**
7. **Variables e identificadores**
8. **Declaración en inicialización**
9. **Operadores y expresiones**
10. **Conversión de tipos**
11. **Comentarios**

# 1. Problemas, Algoritmos y Programas

- En la vida real…

- **Observación de una situación o problema**

- **Pensamos en una o varias posibles soluciones**

- **Aplicamos la solución que estimamos que es más adecuada**

# 1. Problemas, Algoritmos y Programas

- En la vida real…

- **Observación de una situación o problema**

- **Pensamos en una o varias posibles soluciones**

- **Aplicamos la solución que estimamos que es más adecuada**

- En **programación**…

- **Análisis del problema**: el problema tiene que ser definido y comprendido para que pueda ser analizado con detalle

- **Diseño o desarrollo de algoritmos**: procedimiento paso a paso para solucionar el problema dado.

- **Resolución del algoritmo elegido en la computadora**: consiste en convertir el algoritmo en programa, ejecutarlo y comprobar que soluciona el problema

# 1. Problemas, Algoritmos y Programas

- Problema: Encontrar las palabras repetidas de una lista (Análisis).

- Pensamos en una o varias soluciones (Algoritmos):

# 1. Problemas, Algoritmos y Programas

- Problema: Encontrar las palabras repetidas de una lista (Análisis).

- Pensamos en una o varias soluciones (Algoritmos):

    - **Algoritmo 1:**

    - Ordenar la lista alfabéticamente.
    - Recorrer la lista,
        - Si dos elementos consecutivos son iguales, entonces estaban repetidos, escribirlos.

    - **Algoritmo 2:**

    - Recorrer la lista,
        - para cada elemento comprobar (recorriendo de nuevo la lista) si está repetido y entonces escribirlo.

- Aplicamos la solución que consideremos más adecuada (Programa).

# 1. Problemas, Algoritmos y Programas

- Dado un **problema** P un **algoritmo** A es un conjunto de **reglas** o instrucciones, que definen cómo resolver P en un tiempo finito

- Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan.

- Un algoritmo:
  - debe ser **preciso** e indicar el orden de realización paso a paso
  - debe estar **definido**, si se ejecuta dos o más veces, ante la misma entrada debe obtener el mismo resultado
  - debe ser **finito**, debe tener un número finito de pasos

- Un **programa** es un algoritmo escrito con una notación precisa (lenguaje de programación) que pueda ser **ejecutado** por un procesador

# 1. Problemas, Algoritmos y Programas

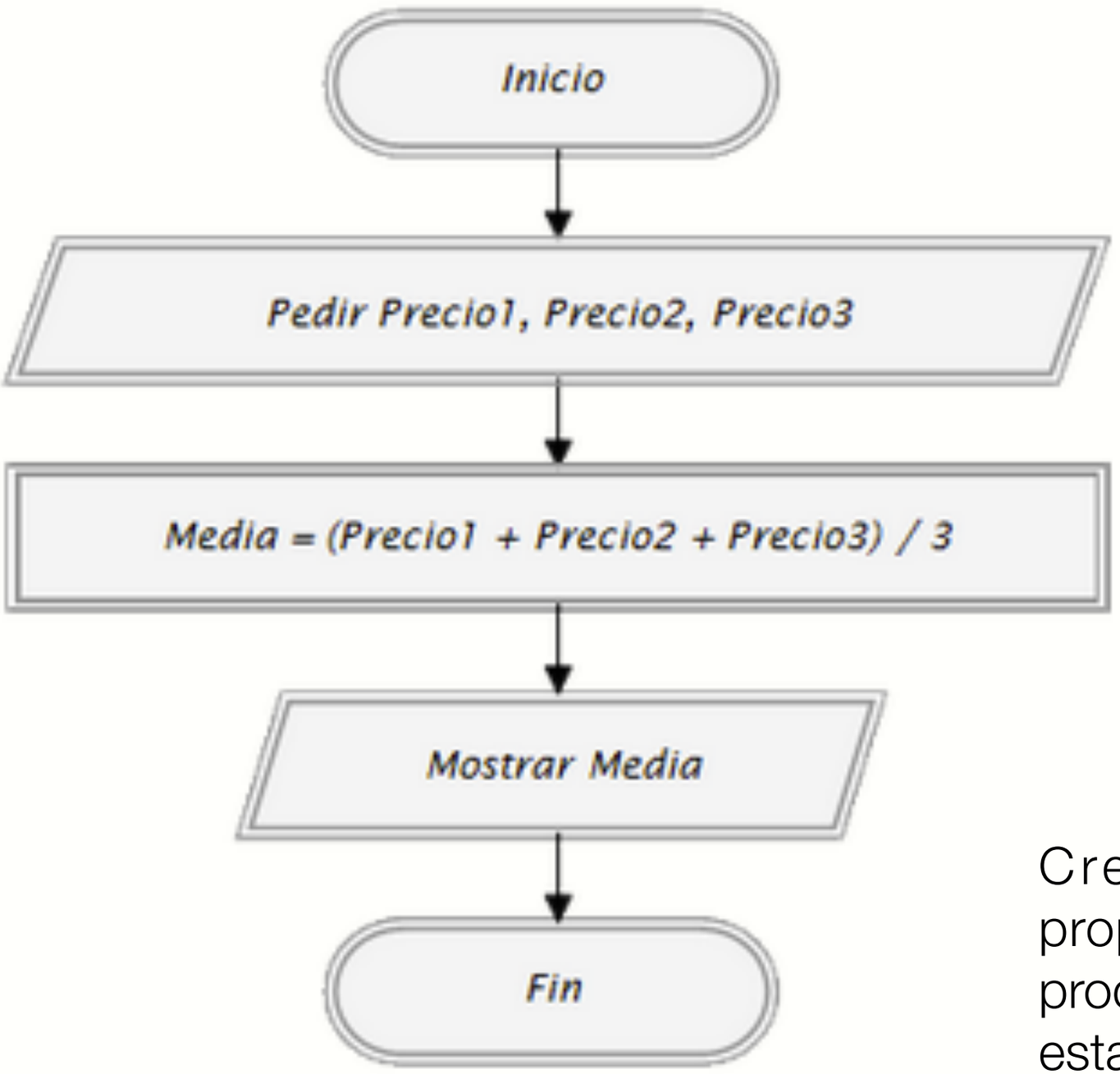
- Cuando los problemas son complejos, es necesario descomponer estos en subproblemas más simples —> diseño descendente o diseño modular

- Para representar gráficamente los algoritmos existen herramientas:

  - **Diagramas de flujo**: utiliza símbolos gráficos para la representación del algoritmo.

  - **Pseudocódigo**: se basa en el uso de palabras en el lenguaje natural, constantes, variables, instrucciones y otras estructuras de programación que expresan de forma escrita la solución del problema

# 1. Problemas, Algoritmos y Programas





Crear el diagrama de flujo que proporcione el precio medio de un producto a partir del precio en tres establecimientos distintos.

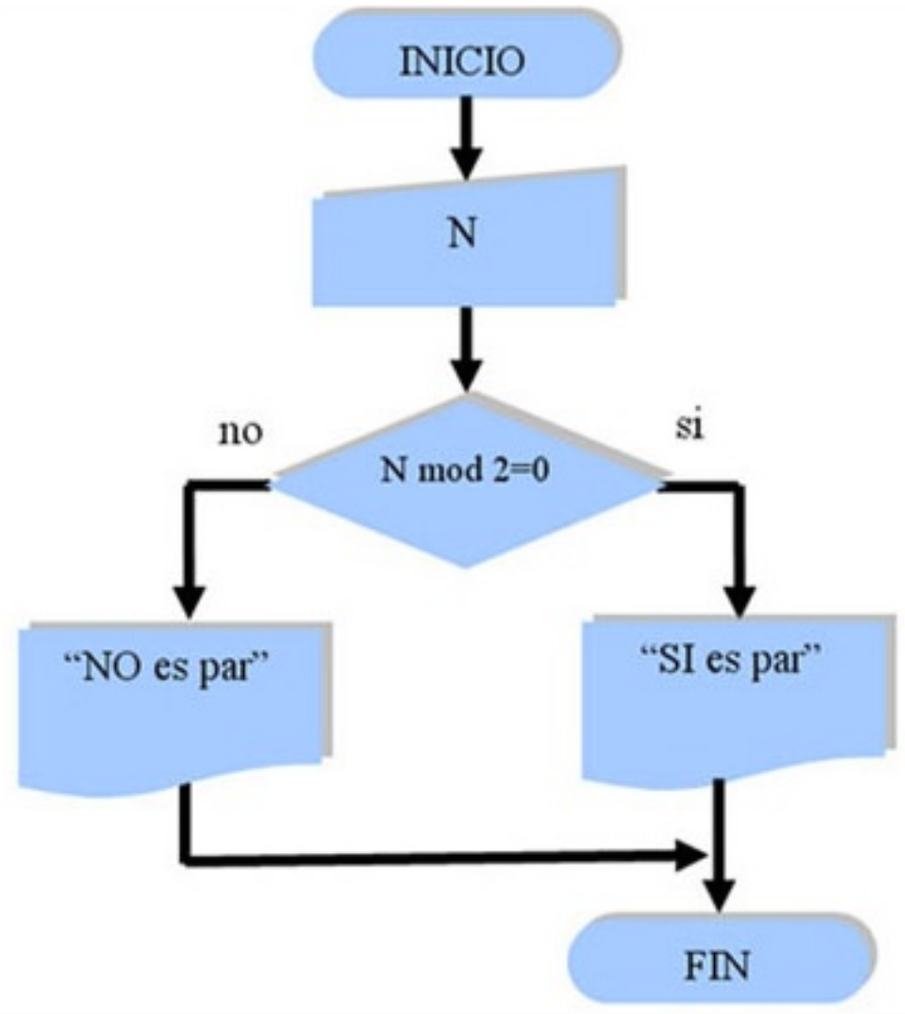
# 1. Problemas, Algoritmos y Programas


Crear el diagrama de flujo que proporcione el precio medio de un producto a partir del precio en tres establecimientos distintos.

# 1. Problemas, Algoritmos y Programas

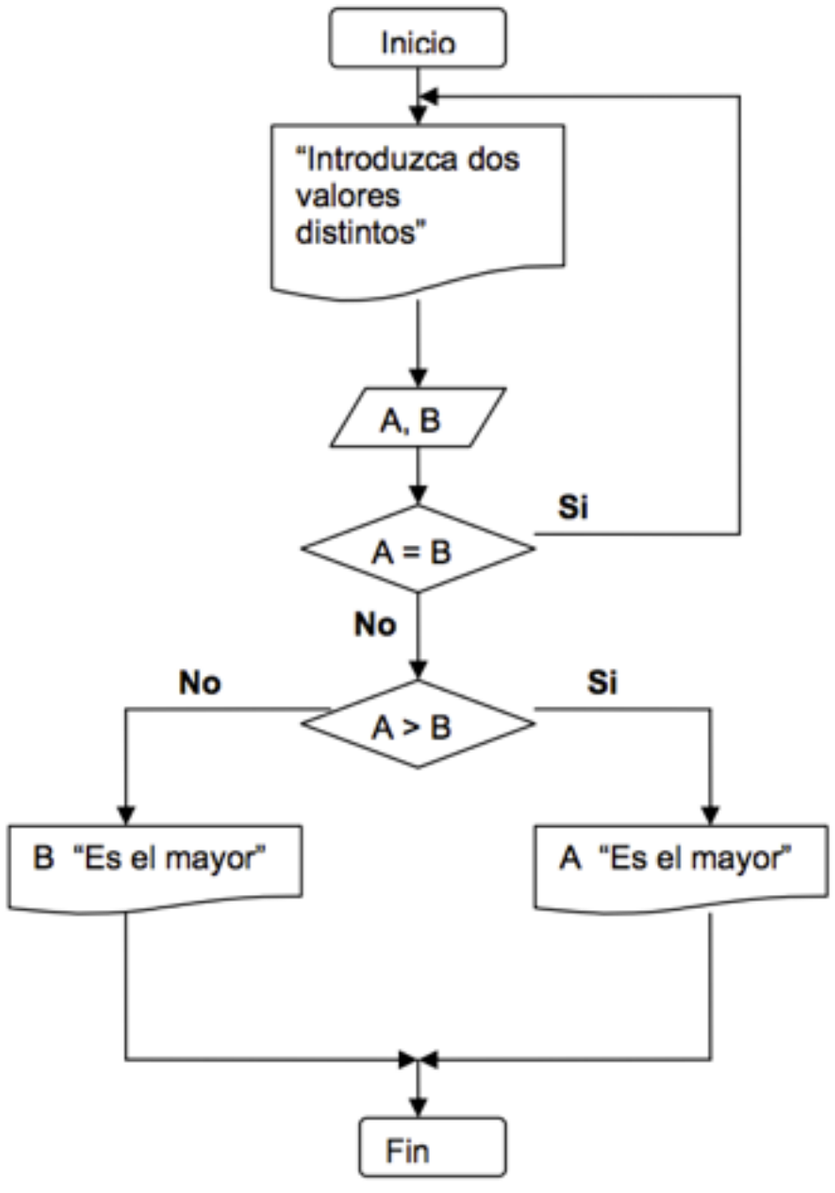
Desarrolle un algoritmo que permita leerán valor y determinar si es par o impar

# 1. Problemas, Algoritmos y Programas




Desarrolle un algoritmo que permita leerán valor y determinar si es par o impar

# 1. Problemas, Algoritmos y Programas

| Pseudocódigo | Diagrama de Flujo |
|---|---|
| **1. Inicio**<br>2. Inicializar variables: A = 0, B = 0<br>3. Solicitar la introducción de dos valores distintos<br>4. **Leer** los dos valores<br>5. Asignarlos a las variables A y B<br>6. **Si** A = B **Entonces** vuelve a 3 porque los valores deben ser distintos<br>**7. Si** A>B **Entonces**<br>**Escribir** A, "Es el mayor"<br>8. **De lo contrario**: **Escribir** B, "Es el mayor"<br>**9. Fin_Si**<br>**10. Fin** |  |

Desarrolle un algoritmo que permita leer dos valores distintos, determinar cual de los dos valores es el mayor y escribirlo.

# 1. Problemas, Algoritmos y Programas

Realizar el pseudocódigo de un programa que permita calcular el área de un rectángulo. Se debe introducir la base y la altura para poder realizar el cálculo.

```
Programa: área
Entorno: BASE, ALTURA, AREA son número enteros
Algoritmo:
        escribir “Introduzca la base y la altura”
        leer BASE, ALTURA
        calcular AREA = BASE * ALTURA
        escribir “El área del rectángulo es “AREA
Finprograma
```

# 2. Fases de la programación

- El proceso de creación de software puede dividirse en fases:
  - Fase de resolución del problema
  - Fase de implementación
  - Fase de explotación y mantenimiento

# 2. Fases de la programación

- **Fase de resolución del problema**

  - **Análisis**:

    - Especificación de **requisitos**.

    - Se especificarán los procesos y estructuras de datos que se van a emplear.

    - El análisis inicial ofrecerá una idea general de lo que se solicita, realizando posteriormente refinamientos que tendrás que dar respuesta a las siguientes cuestiones:

      - ¿Qué datos son necesarios para resolver el problema? **INPUTS**

      - ¿Cual es la información se obtendrá con la resolución del problema? **OUTPUTS**

  - **Diseño**:

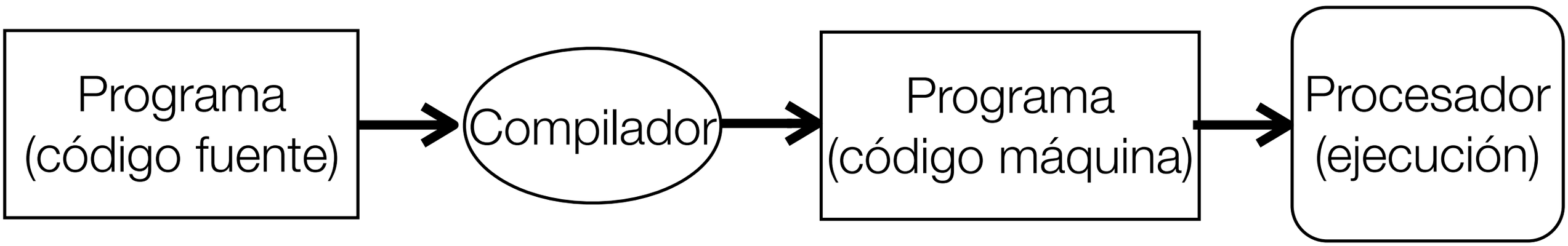
    - Se descompone la aplicación en **módulos**/operaciones.

    - Se describe la secuencia **lógica de instrucciones** capaz de resolver el problema planteado (algoritmo)

    - Todo diseño requerirá de una prueba o **traza** del algoritmo.

# 2. Fases de la programación

- **Implementación**:
  - **Codificación o construcción:**
    - Traducir los resultado de la etapa anterior a un lenguaje de programación —> código fuente /programa.
    - Pruebas de cada módulo (unitarias), pruebas de que los módulos funcionan bien entre ellos (pruebas de interconexión), y pruebas de que todos funcionan en conjunto correctamente (pruebas de integración).
      - Compilación: Es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a un lenguaje que la máquina es capaz de interpretar.
  - **Ejecución**: testear el programa y comprobar si su funcionamiento es correcto. Documentar el código.

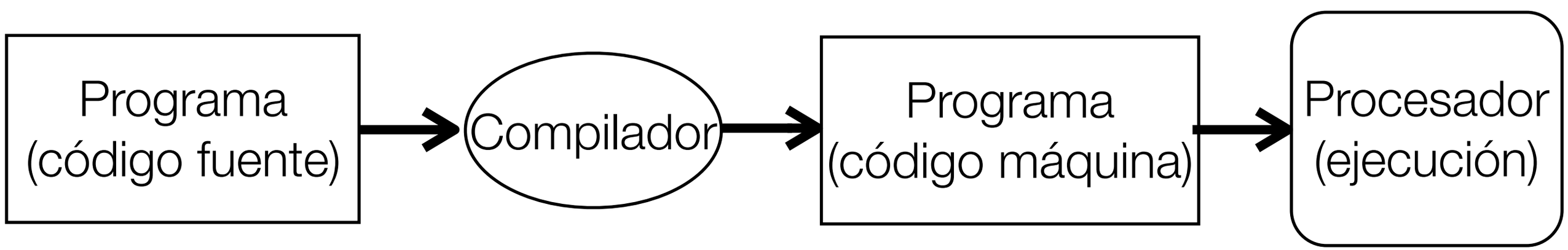


# 2. Fases de la programación

- **Explotación**:

  - Cuando el programa ya está instalado en el sistema y está siendo de utilidad para los usuarios decimos que se encuentra en fase de explotación.

  - Periódicamente será necesario realizar evaluaciones y, si es necesario llevar a cabo modificaciones para que el programa se adapte o actualice a nuevas necesidades —> mantenimiento del software.



# 3. Paradigmas de programación

- **Paradigma de programación**: es un modelo básico para el diseño y la implementación de programas. Determinará cómo será el proceso de diseño y la estructura final del programa.

**declarativa**

Las sentencias que utilizan describen el programa pero no las instrucciones para solucionarlo. Lenguaje **SQL**

**funcional**

Considera al programa como una función matemática. No existe el concepto de variable. Lenguaje **LISP**

**lógica**

Se especifica qué hacer y no cómo hacerlo. Lenguaje **Prolog**

**imperativa**

Detallan cómo hacer las cosas y llevarán al programa a través de distintos estados.

**convencional**

Basado en estructuras de salto que modificaban el flujo del programa

**estructurada**

Permite el uso del concepto de función (programación modular)

**orientada a objetos**

Objetos tienen características (propiedades) y con ellos pueden realizarse acciones (métodos).

# 3. Paradigmas de programación

- Existen múltiples paradigmas, incluso puede haber lenguajes de programación que no se clasifiquen únicamente dentro de uno de ellos.

- Existen distintos enfoques para reducir la dificultad del mantenimiento de la aplicaciones, mejorar el rendimiento y mejorar la calidad de los programas.

# 4. Lenguajes de programación

- **Lenguaje** de **programación**: Conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas.

    - **Gramática** del lenguaje: reglas aplicables al conjunto de símbolos y palabras especiales para la construcción de sentencias correctas.

- **Léxico**: Es el conjunto finito de símbolos y palabras especiales. Es el vocabulario del lenguaje.

- **Sintaxis**: Son las posibles combinaciones de símbolos y palabras especiales.

- **Semántica**: Es el significado de cada construcción del lenguaje.

# 4. Lenguajes de programación

- Los lenguajes se pueden clasificar en función de lo cerca que estén al lenguaje humano o al lenguaje de los computadores.

Están diseñados para que su ejecución se realice a través de un intérprete. El intérprete traduce y ejecuta directamente las instrucciones.

lenguaje interpretado — Python, PHP, JavaScript

lenguaje pseudo-interpretado/ pseudo-compilado — Java

**Lenguajes de alto nivel**. Las instrucciones que forman parte de estos lenguajes utilizan palabras y signos reconocibles por el programador. Independientes del hw.

lenguaje compilado — Pascal, Fortan, Algol, C

códigos de operación que describen una operación elemental del procesador. **Lenguaje de bajo nivel**, depende del hw donde son ejecutados.

lenguaje ensamblador

lenguaje utilizado directamente por el procesador. Conjunto de instrucciones codificadas en binario.

lenguaje máquina

# 4. Lenguajes de programación

- Los lenguajes se pueden clasificar en función de lo cerca que estén al lenguaje humano o al lenguaje de los computadores.

Están diseñados para que su ejecución se realice a través de un intérprete. El intérprete traduce y ejecuta directamente las instrucciones.

lenguaje interpretado — Python, PHP, JavaScript

lenguaje pseudo-interpretado/ pseudo-compilado — Java

**Lenguajes de alto nivel**. Las instrucciones que forman parte de estos lenguajes utilizan palabras y signos reconocibles por el programador. Independientes del hw.

lenguaje compilado — Pascal, Fortan, Algol, C

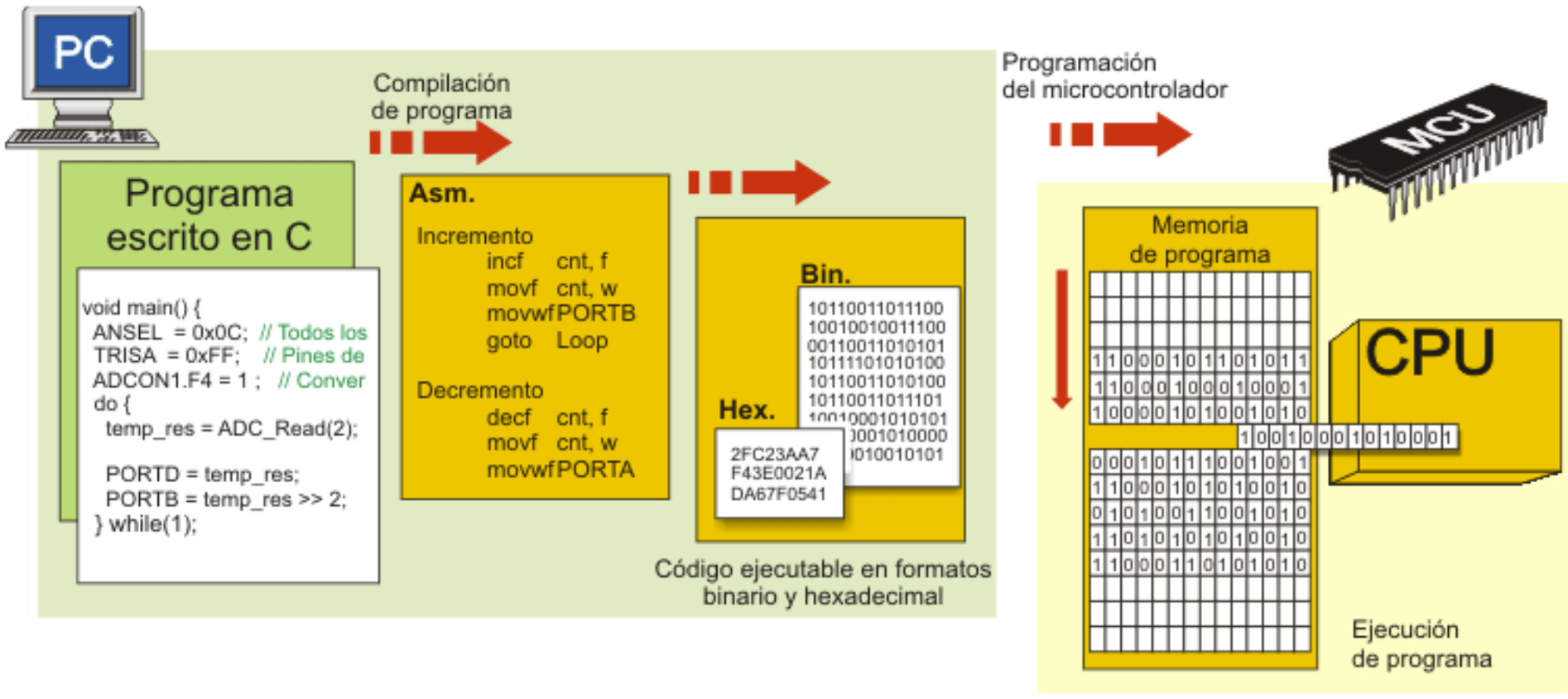
códigos de operación que describen una operación elemental del procesador. **Lenguaje de bajo nivel**, depende del hw donde son ejecutados.

lenguaje ensamblador

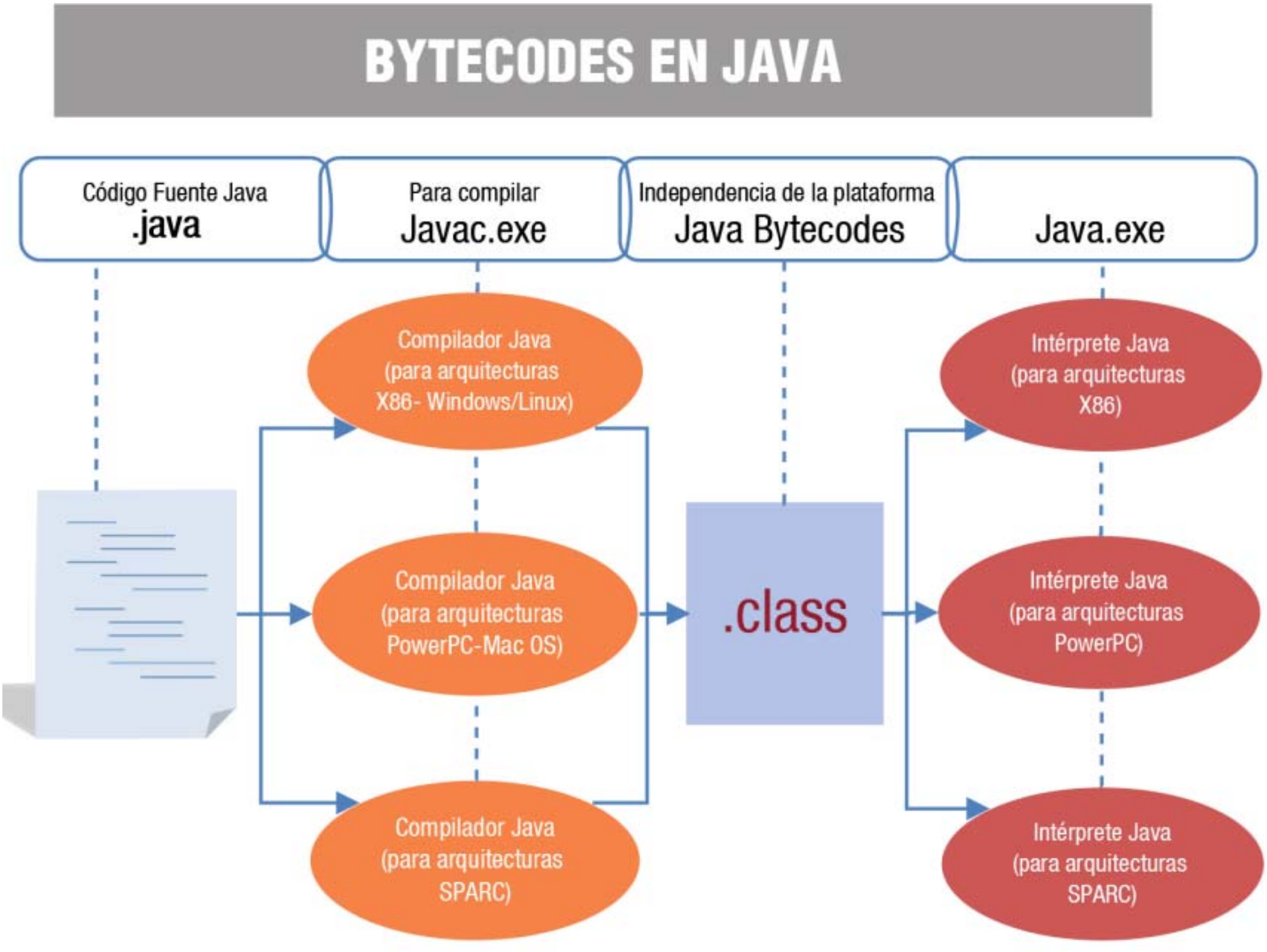
lenguaje utilizado directamente por el procesador. Conjunto de instrucciones codificadas en binario.

lenguaje máquina

# 4. Lenguajes de programación



# 5. Java


BYTECODES EN JAVA
Código Fuente Java
.java
Para compilar
Javac.exe
Independencia de la plataforma
Java Bytecodes
Java.exe
Compilador Java
(para arquitecturas
X86- Windows/Linux)
Compilador Java
(para arquitecturas
PowerPC-Mac OS)
Compilador Java
(para arquitecturas
SPARC)
.class
Intérprete Java
(para arquitecturas
X86)
Intérprete Java
(para arquitecturas
PowerPC)
Intérprete Java
(para arquitecturas
SPARC)

# 5. Java

- **Características** principales del lenguaje Java:
    - El código generado por el compilador es independiente de la arquitectura.
    - Está orientado a objetos.
    - Su sintaxis es similar a C y C++
    - Dispone de un amplio conjunto de bibliotecas.
    - Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
    - La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema

# 5. Java

- Empezar a trabajar: http://www.etnassoft.com/biblioteca/programacion-basica-en-java/
  - Kit de desarrollo de Java **JDK** (Java Development Kit) http://java.sun.com/javase/downloads:
    - **Compilador**: javac
    - **Entorno de ejecución**: JRE (Java Run Environment) incluye la máquina virtual de Java (JVM) + bibliotecas

| Versión del SDK para la versión estándar de Java | Nombre que se le da al kit de desarrollo |
|---|---|
| 1.1 | JDK 1.1 |
| 1.2 | J2SE 1.2 |
| 1.3 | J2SE 1.3 |
| 1.4 | J2SE 1.4 |
| 1.5 | J2SE 1.5 |
| 1.6 | Java SE 6 |
| 1.7 | Java SE 7 |

- Actualmente hay 3 **ediciones** de la plataforma **Java 2**:
  - **J2SE**: Entorno relacionado con la creación de aplicaciones y applets en lenguaje Java.
  - J2EE: entorno para la creación de aplicaciones Java empresariales y del lado del servidor.
  - J2ME: entorno para la creación de aplicaciones Java para dispositivos móviles.
- Desde la versión **1.2** se habla de **Java 2**. Desde la versión **1.6** se ha abandonado la terminología Java 2 y ahora se habla de **Java 6** y **Java 7** para las versiones 1.6 y 1.7 del kit de desarrollo. Cada **versión** tiene varias **revisiones**, así la versión 1.6.7 del SDK indica versión 6 de Java, revisión 7.

# 5. Java

- Fichero: HolaMundo.java

```
/**
* Estructura general de un programa en Java
*/

public class HolaMundo
{

    public static void main(String args[ ]) {
        //Esto es un comentario de una línea
        System.out.println("¡Hola Mundo!");
    }
}
```

Clase principal que incluye todos los demás elementos del programa.

Representa al programa principal desde el que se llevará a cabo la ejecución del programa

Todos los programas Java tienen un método main.

```
>>javac HolaMundo.java

>>java HolaMundo
```

¡OJO! Java distingue entre mayúsculas y minúsculas

# 6. Tipos de datos

- **Tipo de dato:** es una especificación de los valores que son válidos para una variable y de las operaciones que se pueden realizar.

  - **Tipos de datos sencillos o primitivos**: representan valores simples que vienen predefinidos en el lenguaje; contienen valores únicos, como por ejemplo un carácter o un número.

  - **Tipos de datos de referencia**: se definen con un nombre o referencia que contiene la dirección en memoria de un valor o un grupo de valores. Ej. los vectores o arras, o las clases.

# 6. Tipos de datos

- **Tipos de datos sencillos o primitivos**:

| TIPOS DE DATOS PRIMITIVOS | | | | |
|---|---|---|---|---|
| **Tipo** | **Descripción** | **Bytes** | **Rango** | **Valor por default** |
| **byte** | Entero muy corto | 1 | -128 a 127 | 0 |
| **short** | Entero corto | 2 | -32,768 a 32,767 | 0 |
| **int** | Entero | 4 | -2,147,483,648 a 2,147,483,647 | 0 |
| **long** | Entero largo | 8 | -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 | 0L |
| **float** | Numero con punto flotante de precisión individual con hasta 7 dígitos significativos | 4 | +/-1.4E-45 (+/-1.4 times $10^{-45}$) a +/-3.4E38 (+/-3.4 times $10^{38}$) | 0.0f |
| **double** | Numero con punto flotante de precisión doble con hasta 16 dígitos significativos | 8 | +/-4.9E-324 (+/-4.9 times $10^{-324}$) a +/-1.7E308 (+/-1.7 times $10^{308}$) | 0.0d |
| **char** | Carácter Unicode 2 | \u0000 a \uFFFF | '\u0000' | |
| **boolean** | Valor Verdadero o Falso | 1 | true o false | false |

# 6. Tipos de datos

- **Tipos de datos sencillos o primitivos**:
  - **char** es considerado por el compilador como un tipo **numérico**, ya que los valores que guarda son el **código Unicode** correspondiente al carácter que representa, no el carácter en sí, por lo que puede **operarse** con caracteres como si se tratara de números **enteros**.

# 6. Tipos de datos

int precio = 42; // Entero tipo int. Un número sin punto decimal se interpreta normalmente como int.
int importe_acumulado = 210; // Entero tipo int
String profesor = “Ernesto Juárez Pérez”; // Tipo String
String aula = “A-44”; // Tipo String
int capacidad = 1500; // Entero tipo int
boolean funciona = true; // Tipo boolean
boolean esVisible = false; // Tipo boolean
float diametro = 34.25f; // Tipo float. Una f o F final indica que es float.
float peso = 88.77; // Tipo double. Un número con punto decimal se interpreta normalmente como double.
short edad = 19; // Entero tipo short
long masa = 178823411L; // Entero tipo long. Una l o L final indica que es long.
char letra1 = ‘h’; // Tipo char (carácter). Se escribe entre comillas simples.

# 7. Variables e Identificadores

- **Variables:** zona en la memoria del computador con un valor que puede ser almacenado para ser utilizado más tarde. Las variables vienen determinadas por:

    - nombre (**identificador**), que permite acceder al valor que contiene en memoria.

    - un tipo de dato, que especifica qué clase de información guarda la variable en esa zona de memoria.

    - un rango de valores que puede admitir dicha variable.

- Identificador: secuencia sin espacios de letras y dígitos.

# 7. Variables e Identificadores

- **Normas de estilo para nombrar variables:**

  - Java distingue las mayúsculas de las minúsculas. Ej. alumno y Alumno son variables diferentes.

  - No se puede utilizar el valor de los literales booleanos (true o false)  ni el valor nulo (null)

  - Los identificadores deben ser lo más descriptivos posibles. Ej. variable que guarda los datos de un cliente: manejadorCliente mejor que CI (poco descriptivo).

# 7. Variables e Identificadores

- **Normas de estilo para nombrar variables:**

| Identificador | Convención | Ejemplo |
|---|---|---|
| **nombre de variable** | Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas | `numAlumnos, suma` |
| **nombre de constante** | En letras mayúsculas, separando las palabras con el guión bajo, por convenio el guión bajo no se utiliza en ningún otro sitio | `TAM_MAX, PI` |
| **nombre de una clase** | Comienza por letra mayúscula | `String, MiTipo` |
| **nombre de función** | Comienza con letra minúscula | `modifica_Valor, obtiene_Valor` |

# 7. Variables e Identificadores

- **Palabras reservadas:**

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

# 7. Variables e Identificadores

- **Tipos de Variables:**

    - **Variables**: sirven para almacenar los datos durante la ejecución del programa. Su valor puede cambiar varias veces a lo largo del programa.

        - **Variables miembro**: son las variables que se crean dentro de una clase fuera de los métodos.

        - **Variables locales**: variables que se crean y usan dentro de un método o, en general, dentro de cualquier bloque de código.

    - **Constantes** o **variables finales**: su valor no cambia a lo largo del programa.

# 7. Variables e Identificadores

```
public class EjemploVariables {
      final double PI = 3.1415926536;          //PI es una constante
      int x;                                   //x es una variable miembro
                                               //de la clase ejemplo variable



      int obtenerX(int x){                     //x es un parámetro
            int valorAntiguo = this.x;         //valor antiguo es una var. local
            return valorAntiguo;

      }

      public static void main(String[ ] args){



      }//fin del método main

}//fin de la clase EjemploVariables
```

# 8. Declaración e inicialización

**Tipo** + **Identificador**

int **numAlumnos** = 15;

double **radio** = 3.14, **importe** = 102.95;

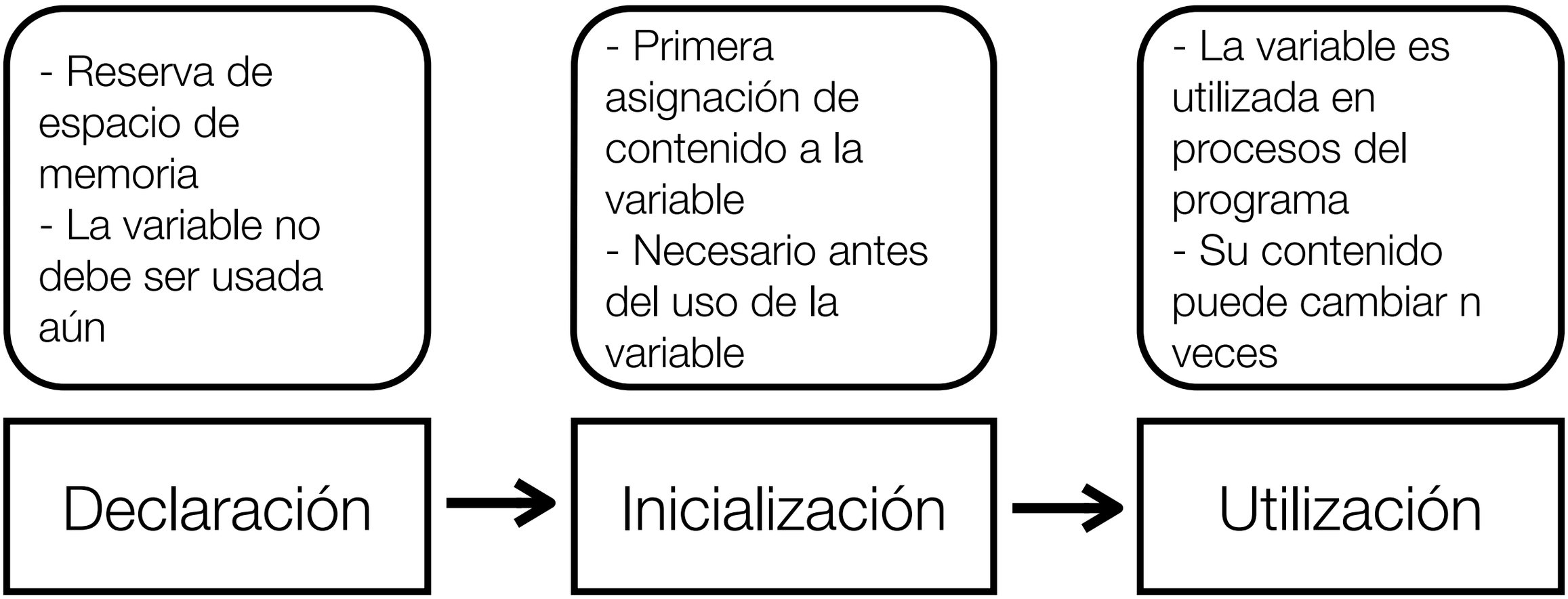
final double **PI** = 3.1415926536;

Si  al declarar una variable no le damos valor, el compilador le asigna un valor por defecto:

- **variables miembro** si se inicializan automáticamente si no les damos valor. Numérico —> 0, Carácter —> nulo, si son boolean —>false, tipo reverenciado —> null.

- **variables locales** no se inicializan automáticamente. Debemos asignarles nosotros un valor antes de ser usadas

# 8. Declaración e inicialización



# 8. Declaración e inicialización

```
int p;
double q = p;
```

```
int p;
if(...)
p = 5;
int q=p;
```

# 8. Declaración e inicialización

**Tipo** + **Identificador**

int[ ] **arrayDeEnteros**;

Cuenta **cuentaCliente**;

Cuando el conjunto de datos utilizado tiene características similares se suelen agrupar en estructuras —> Datos estructurados: arrays, listas, árboles, etc.

Además de los ocho tipos de datos primitivos, Java proporciona un tratamiento especial a los textos o cadenas de caracteres mediante el tipo **String**. En realidad se trata de objetos (son tipos referenciados) pero se pueden utilizar de forma sencilla como si fueran variables de tipos primitivos:

String **mensaje;**

**mensaje**=“El primer programa”;

# 8. Declaración e inicialización

**enum Identificador** {lista de valores que puede tomar}

**enum Días** {Lunes,Martes, Miercoles, Jueves, Viernes, Sabado, Domingo }

Para acceder a cada elemento del tipo enumerado se utiliza el nombre de la variable seguido de un punto y el valor de la lista:

Dias diaactual = Dias.Martes;

# 9. Operadores y expresiones

- **Operadores aritméticos**
- **Operadores de asignación**
- **Operador condicional**
- **Operadores de relación**
- **Operadores lógicos**
- **Operaciones con cadenas**
- **Precedencia de operadores**

# 9. Operadores y expresiones

- **Operadores aritméticos**

| Operador | Operación Java | Expresión Java | Resultado |
|---|---|---|---|
| - | Operador unario de cambio de signo | -10 | -10 |
| + | Adición | 1.2 + 9.3 | 10.5 |
| - | Sustracción | 312.5 – 12.3 | 300.2 |
| * | Multiplicación | 1.7 * 1.2 | 1.02 |
| / | División (entera o real) | 0.5 / 0.2 | 2.5 |
| % | Resto de la división entera | 25 % 3 | 1 |

- El resultado de este tipo de expresiones depende de los operandos que utilicen:

| Tipo de los operandos | Resultado |
|---|---|
| **Un operando de tipo long y ninguno real (float o double)** | long |
| **Ningún operando de tipo long ni real (float o double)** | int |
| **Al menos un operando de tipo double** | double |
| **Al menos un operando de tipo float y ninguno double** | float |

# 9. Operadores y expresiones

```
public class OpAritmeticos {
    public static void main(String[] args) {
        int i,j;
        double a,b;
        i = 7;
        j = 3;
        System.out.println("* Operandos enteros: i = " + i + " ; j = " + j);
        System.out.println(" Operador suma: i + j = " + (i+j));
        System.out.println(" Operador resta: i - j = " + (i-j));
        System.out.println(" Operador producto: i * j = " + (i*j));
        System.out.println(" Operador division: i / j = " + (i/j));
        System.out.println(" Operador resto: i % j = " + (i%j));
        a = 12.5;
        b = 4.3;
        System.out.println("* Operandos reales: a = " + a + " ; b = " + b);
        System.out.println(" Operador suma: a + b = " + (a+b));
        System.out.println(" Operador resta: a - b = " + (a-b));
        System.out.println(" Operador producto: a * b = " + (a*b));
        System.out.println(" Operador division: a / b = " + (a/b));
        System.out.println(" Operador resto: a % b = " + (a%b));
    }
}
```

# 9. Operadores y expresiones

- **Operadores aritméticos**

  - Otro tipo de operadores  son los unarios **incrementales** y **decrementales**.

  - Se pueden utilizar con notación **prefija**, si aparece antes que el operando, o con notación **postfija**, si aparece después del operando.

| Tipo operador | Expresión Java | |
|---|---|---|
| ++ (incremental) | Prefija: | Postfija: |
| | x=3; | x=3; |
| | y=++x; | y=x++; |
| | // x vale 4 e y vale 4 | // x vale 4 e y vale 3 |
| --(decremental) | 5-- // el resultado es 4 | |

# 9. Operadores y expresiones

```
class opIncrementales {
    public static void main(String[] args) {
        int i,j; // Variables enteras. Podrian ser reales o char
        i = 7;
        System.out.println("* Operando entero: i = " + i + ";");
        System.out.println(" Operador ++: j = i++; ");
        j = i++;
        System.out.println(" // i vale " + i + "; j vale " + j);
        i = 7;
        System.out.println(" i = " + i + ";");
        System.out.println(" j = ++i; ");
        j = ++i;
        System.out.println(" // i vale " + i + "; j vale " + j);
        i = 7;
        System.out.println("* Operando entero: i = " + i + ";");
        System.out.println(" Operador --: j = i--; ");
        j = i--;
        System.out.println(" // i vale " + i + "; j vale " + j);
        i = 7;
        System.out.println(" i = " + i + ";");
        System.out.println(" j = --i; ");
        j = --i;
        System.out.println(" // i vale " + i + "; j vale " + j);
    }
}
```

# 9. Operadores y expresiones

- **Operadores de asignación**

  - El principal operador de esta categoría es el operador asignación =, que permite al programa darle un valor a una variable.

  - Java proporciona otros operadores de asignación combinados con los operadores aritméticos, que permiten abreviar o reducir ciertas expresiones.

| Operador | Ejemplo en Java | Expresión equivalente |
|---|---|---|
| += | op1 += op2 | op1 = op1 + op2 |
| -= | op1 -= op2 | op1 = op1 - op2 |
| *= | op1 *= op2 | op1 = op1 * op2 |
| /= | op1 /= op2 | op1 = op1 / op2 |
| %= | op1 %= op2 | op1 = op1 % op2 |

# 9. Operadores y expresiones

```
public class opAsignacion {
    public static void main(String[] args) {
        int i,j;
        double x;
        char c;
        boolean b;
        String s;
        i = 15;
        j = i;
        x = 12.345;
        c = 'A';
        b = false;
        s = “Hola";
        System.out.println("i = " + i);
        System.out.println("j = " + j);
        System.out.println("x = " + x);
        System.out.println("c = " + c);
        System.out.println("b = " + b);
        System.out.println("s = " + s);
 }
}
```

# 9. Operadores y expresiones

```
public class OpCombinados {
 public static void main(String[] args) {
      int i,j; // Variables enteras. Podrian ser reales
      i = 7;
      j = 3;
      System.out.println("* Operandos enteros: i = "+ i +" ; j = "+ j);
      i += j;
      System.out.println(" Suma combinada: i += j " + " // i vale " + i);
      i = 7;
      i -= j;
      System.out.println(" Resta combinada: i -= j " + " // i vale " + i);
      i = 7;
      i *= j;
      System.out.println(" Producto combinado: i *= j " + " // i vale " + i);
      i = 7;
      i /= j;
      System.out.println(" Division combinada: i /= j " + " // i vale " + i);
      i = 7;
      i %= j;
      System.out.println(" Resto combinada: i %= j " + " // i vale " + i);
 }
}
```

# 9. Operadores y expresiones

- **Operadores condicional**

  - El operador condicional **?**: sirve para evaluar una condición y devolver un resultado en función de si es verdadera o falsa dicha condición. Es el único operador ternario de Java, y como tal, necesita tres operandos para formar una expresión.

```
condición ? exp1 : exp2
(x>y)       ?   x   :  y
```

- El valor que devolverá el operador condicional será:

  - exp1 —> si la condición es cierta
    exp2 —> si la condición es falsa

# 9. Operadores y expresiones

```
public class opCondicional {
        public static void main(String[] args) {
                int i,j,k;
                i = 1;
                j = 2;
                k = i > j ? 2*i : 3*j+1;
                System.out.println("i = " + i);
                System.out.println("j = " + j);
                System.out.println("k = " + k);
                i = 2;
                j = 1;
                k = i > j ? 2*i : 3*j+1;
                System.out.println("i = " + i);
                System.out.println("j = " + j);
                System.out.println("k = " + k);
        }
}
```

# 9. Operadores y expresiones

- **Operadores de relación**

| Operador | Ejemplo en Java | Significado |
|---|---|---|
| == | op1 == op2 | op1 igual a op2 |
| != | op1 != op2 | op1 distinto de op2 |
| > | op1 > op2 | op1 mayor que op2 |
| < | op1 < op2 | op1 menor que op2 |
| >= | op1 >= op2 | op1 mayor o igual que op2 |
| <= | op1 <= op2 | op1 menor o igual que op2 |

# 9. Operadores y expresiones

```
public class OpRelacionales {
    public static void main(String[] args) {
        int i,j;
        i = 7;
        j = 3;
        System.out.println("* Operandos enteros: i = "+ i +" ; j = "+ j);
        System.out.println(" Operador igualdad: i == j es " + (i==j));
        System.out.println(" Operador desigualdad: i != j es " + (i!=j));
        System.out.println(" Operador mayor que: i > j es " + (i>j));
        System.out.println(" Operador menor que: i < j es " + (i<j));
        System.out.println(" Operador mayor o igual que: i >= j es " + (i>=j));
        System.out.println(" Operador menor o igual que: i <= j es " + (i<=j));
    }
}
```

# 9. Operadores y expresiones

```
public class OpRelacionales {
    public static void main(String[] args) {
        //clase scanner para la petición de datos
        Scanner teclado = new Scanner(System.in);
        int x;
        int y;
        String cadena;
        boolean resultado;
        System.out.println("Introducir primer número:");
        x = teclado.nextInt();//pedimos el primer número
        System.out.println("Introducir segundo número:");
        y = teclado.nextInt();//pedimos el segundo número
        //realizamos las comparaciones
        cadena=(x==y)?"iguales":"distintos";
        System.out.print("Los números %d y %d son %s\n", x, y, cadena);
        resultado=(x!=y);
        System.out.println("x != y  //es "+resultado);
        resultado=(x<y);
        System.out.println("x < y //es "+resultado);
        resultado=(x>y);
        System.out.println("x > y  //es "+resultado );
 }
}
```

# 9. Operadores y expresiones

- **Operadores lógicos**

| Operador | Ejemplo en Java | Significado |
|---|---|---|
| ! | !op | Devuelve **`true`** si el operando es **`false`** y viceversa. |
| & | op1 & op2 | Devuelve **`true`** si op1 y op2 son **`true`** |
| \| | op1 \| op2 | Devuelve **`true`** si op1 u op2 son **`true`** |
| ^ | op1 ^ op2 | Devuelve **`true`** si sólo uno de los operandos es **`true`** |
| && | op1 && op2 | Igual que &, pero si op1 es **`false`** ya no se evalúa op2 |
| \|\| | op1 \|\| op2 | Igual que \|, pero si op1 es **`true`** ya no se evalúa op2 |

# 9. Operadores y expresiones

```
public class OpBooleanos {
    public static void main(String [] args) {
        System.out.println("Demostracion de operadores logicos");
        System.out.println("Negacion: ! false es : " + (! false));
        System.out.println(" ! true es : " + (! true));
        System.out.println("Suma: false | false es : " + (false | false));
        System.out.println(" false | true es : " + (false | true));
        System.out.println(" true | false es : " + (true | false));
        System.out.println(" true | true es : " + (true | true));
        System.out.println("Producto: false & false es : " + (false & false));
        System.out.println(" false & true es : " + (false & true));
        System.out.println(" true & false es : " + (true & false));
        System.out.println(" true & true es : " + (true & true));
    }
}
```

# 9. Operadores y expresiones

- **Operaciones con cadenas**

  **Creación**. Como hemos visto en el apartado de literales, podemos crear una variable de tipo String simplemente asignándole una **cadena de caracteres encerrada entre comillas dobles**.

  **Obtención de longitud.** Si necesitamos saber la longitud de un String, utilizaremos el método **length()**.

  **Concatenación**. Se utiliza el operador **+** o el método **concat()** para concatenar cadenas de caracteres.

# 9. Operadores y expresiones

- **Operaciones con cadenas**

  **Comparación**. El método **equals()** nos devuelve un valor booleano que indica si las cadenas comparadas son o no iguales. El método **equalsIgnoreCase()** ignora las mayúsculas de las cadenas a considerar.

  **Obtención de subcadenas**. Podemos obtener cadenas derivadas de una cadena original con el método **substring()**, al cual le debemos indicar el inicio y el fin de la subcadena a obtener.

  **Cambio a mayúsculas/minúsculas**. Los métodos **toUpperCase()** y **toLowerCase()** devuelven una nueva variable que transforma en mayúsculas o minúsculas, respectivamente, la variable inicial.

  **Convertir un tipo de dato primitivo** (in, long, float, etc.) a una variable de tipo **String**. **Valueof().**

# 9. Operadores y expresiones

```
public class ejemploCadenas{
        public static void main(String[] args)
        {
                String cad1 = “Ciclo DAW”;
                String cad2 = “Programación”;
                System.out.printf(“La cadena cad1 es: %s y cad2 es: %s”, cad1, cad2);
                System.out.printf(“\nLa longitud de cad1: %d”, cad1.length());
                //concatenación de cadenas (concat o bien operador +)
                System.out.printf(“\nConcatenación: %s”, cad1.concat(cad2));
                //comparación de cadenas
                System.out.printf(“\ncad1.equals(cad2) es %b”, cad1.equals(cad2));
                System.out.printf(“\ncad1.equalsIgnoreCase(cad2) es %b”,
cad1.equalsIgnoreCase(cad2));
                System.out.printf(“\ncad1.compareTo(cad2) es %d”, cad1.compareTo(cad2
                //obtención de subcadenas
                System.out.printf(“\ncad1.substring(0,5) es %s”, cad1. substring(0,5));
                //pasar a minúsculas
                System.out.printf(“\ncad1.toLowerCase() es %s”, cad1.toLowerCase());
                System.out.println();
        }
}
```

# 9. Operadores y expresiones

- **Precedencia de operadores:**

  - La **multiplicación**, **división** y **resto** de una operación se evalúan primero. Si dentro de la misma expresión tengo varias operaciones de este tipo, empezaré evaluándolas de **izquierda** a **derecha**.

  - La **suma** y la **resta** se aplican **después** que las anteriores. De la misma forma, si dentro de la misma expresión tengo varias sumas y restas empezaré evaluándolas de **izquierda** a **derecha**.

# 10. Conversión de tipos

- Las conversiones de tipo se realizan para **hacer que el resultado de una expresión sea del tipo que nosotros deseamos**.

  - **Conversiones automáticas.** Cuando a una variable de un tipo se le asigna un valor de otro **tipo numérico con menos bits** para su representación, se realiza una conversión automática. En ese caso, el valor se dice que es **promocionado al tipo más grande** (el de la variable). Por ejemplo, de **int** a **long** o de **float** a **double**.

  - **Conversiones explícitas.** Cuando hacemos una conversión **de** un **tipo con más bits** a un **tipo con menos bits**. En estos casos debemos **indicar** que queremos hacer la **conversión** de manera **expresa**, ya que se puede producir una **pérdida** de **datos** y hemos de ser conscientes de ello. Este tipo de conversiones se realiza con el operador **cast**.

# 10. Conversión de tipos

```
int a;
byte b;
a = 12;                 // no se realiza conversión alguna
b = 12;                 // se permite porque 12 está dentro
                        // del rango permitido de valores para b
b = a;                  // error, no permitido (incluso aunque
                        // 12 podría almacenarse en un byte)
byte b = (byte) a;      // Correcto, forzamos conversión explícita
```

# 10. Conversión de tipos

| Tabla de Conversión de Tipos de Datos Primitivos | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Tipo destino | | | | | | | |
| | | boolean | char | byte | short | int | long | float | double |
| Tipo origen | boolean | - | N | N | N | N | N | N | N |
| | char | N | - | C | C | CI | CI | CI | CI |
| | byte | N | C | - | CI | CI | CI | CI | CI |
| | short | N | C | C | - | CI | CI | CI | CI |
| | int | N | C | C | C | - | CI | CI* | CI |
| | long | N | C | C | C | C | - | CI* | CI* |
| | float | N | C | C | C | C | C | - | CI |
| | double | N | C | C | C | C | C | C | - |

**N:** Conversión no permitida (un boolean no se puede convertir a ningún otro tipo y viceversa).
**CI:** Conversión implícita o automática. Un **asterisco** indica que puede haber posible **pérdida** de **datos**.
**C:** Casting de tipos o conversión explícita.
El **asterisco** indica que puede haber una **posible pérdida de datos**, por ejemplo al convertir un número de tipo int que usa los 32 bits posibles de la representación, a un tipo float, que también usa 32 bits para la representación, pero 8 de los cuales son para el exponente.

# 10. Conversión de tipos

- **Reglas de promoción de tipos de datos:**
    - Cuando en una expresión hay datos o variables de distinto tipo, el compilador realiza la promoción de unos tipos en otros, para obtener como resultado el tipo final de la expresión.
    - Si uno de los operandos es de tipo **double**, el otro es convertido a double.
- En cualquier otro caso:
        - Si el uno de los operandos es **float**, el otro se convierte a **float**
        - Si uno de los operandos es **long**, el otro se convierte a **long**
        - Si no se cumple ninguna de las condiciones anteriores, entonces ambos operandos son convertidos al tipo **int**.

# 10. Conversión de tipos

- **Conversión de números en Coma flotante (float, double) a enteros (int) :**

  - Cuando convertimos números en coma flotante a números enteros, la parte decimal se trunca (redondeo a cero). Si queremos hacer otro tipo de **redondeo**:

    - Math.round(num): Redondeo al **siguiente número entero**.

    - Math.ceil(num): Mínimo **entero** que sea **mayor o igual a num**.

    - Math.floor(num): Entero **mayor**, que sea **inferior o igual a num**

# 10. Conversión de tipos

- **Conversión de números en Coma flotante (float, double) a enteros (int) :**

```
double num=3.5;
x=Math.round(num);          // x = 4
y=Math.ceil(num);           // y = 4
z=Math.floor(num);          // z = 3
```

# 10. Conversión de tipos

- **Conversiones entre caracteres (char) y enteros (int) :**

  - Como un tipo char lo que guarda en realidad es el código Unicode de un carácter, los caracteres pueden ser considerados como números enteros sin signo.

```
int num;
char c;
num = (int) ‘A’;                    //num = 65
c = (char) 65;                      // c = ‘A’
c = (char) ((int) ‘A’ + 1);         // c = ’B’
```

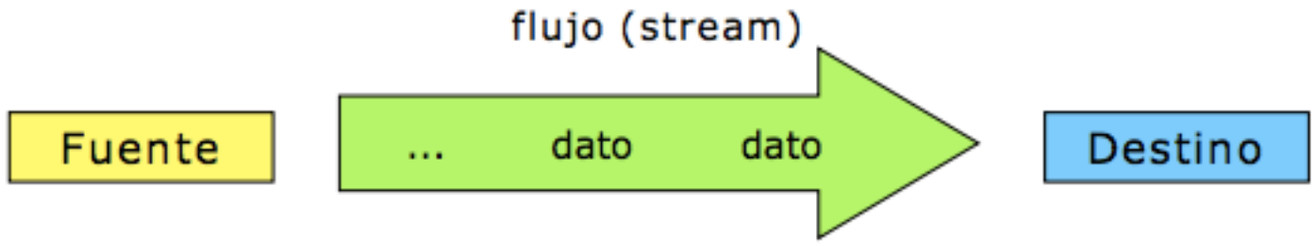
# 10. Conversión de tipos

- **Conversiones de tipo con cadenas de caracteres (String):**

```
num=Byte.parseByte(cad);
num=Short.parseShort(cad);
num=Integer.parseInt(cad);
num=Long.parseLong(cad);
num=Float.parseFloat(cad);
num=Double.parseDouble(cad);
```

# 11. Entrada y Salida con Java

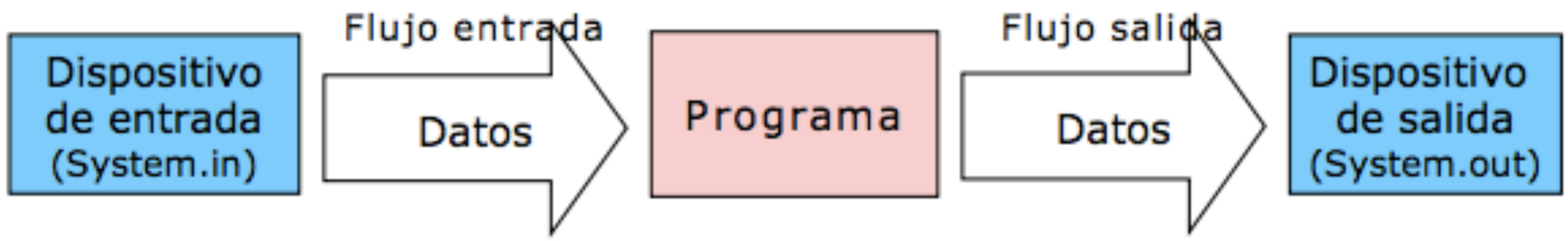
- En Java se define la abstracción de stream (flujo) para tratar la comunicación de información entre el programa y el exterior.




- Los flujos actúan como interfaz con el dispositivo o clase asociada.
  - Operación independiente del tipo de datos y del dispositivo
  - Mayor flexibilidad (p.e. redirección, combinación)
  - Diversidad de dispositivos (fichero, pantalla, teclado, red, …)
  - Diversidad de formas de comunicación
  - Modo de acceso: secuencial, aleatorio
  - Información intercambiada: binaria, caracteres, líneas

# 11. Entrada y Salida con Java

- En Java se accede a la E/S estándar a través de campos estáticos de la clase java.lang.System.
  - **System.in** implementa la entrada estándar
  - **System.out** implementa la salida estándar
  - **System.err** implementa la salida de error



# 11. Entrada y Salida con Java

- **Flujos Éstandar:**
  - **System.in**
    - Instancia de la clase InputStream: flujo de bytes de entrada
    - Metodos
      - **read()** permite leer un byte de la entrada como entero
      - skip(n ) ignora n bytes de la entrada
      - available() número de bytes disponibles para leer en la entrada
  - **System.out**
    - Instancia de la clase PrintStream: flujo de bytes de salida
    - Métodos para impresión de datos
      - **print()**, **println()**
      - flush() vacía el buffer de salida escribiendo su contenido
- **System.err**
  - Funcionamiento similar a System.out
  - Se utiliza para enviar mensajes de error (por ejemplo a un fichero de log o a la consola)

# 11. Entrada y Salida con Java

```
public static void main(String args[]) {
        int edad;
        String nombre;
        Scanner reader = new Scanner(System.in);
        System.out.println(“Introduzca su nombre: ");
        nombre= reader.nextLine();
        System.out.println(“Introduzca su edad: ");
        edad= reader.nextInt();
        System.out.println(“Su nombre es: ”+ nombre);
        System.out.println(“Su edad es: ”+edad);
}
```

# 12. Comentarios y documentación

```
// Esta es una línea comentada.

/* Aquí empieza el bloque comentado
y aquí acaba */

/** Los comentarios de documentación se comentan de este
modo */
```

# 12. Comentarios y documentación

- Una tarea importante en la generación de código es su documentación. El código no debe únicamente ejecutarse sin errores si no que además debe estar bien documentado. Java facilita esta tarea utilizando ciertas etiquetas en los comentarios de documentación.

**@author** [Nombre y Apellidos del autor]

**@version** [Información de la versión]

**@description**

**@param** [nombreDelParametro] [Descripción]

@return [Descripción del parámetro devuelto]

@exception [Excepción lanzada]

@see [Referencia cruzada]

@deprecated [Comentario de porque es obsoleto]

# 12. Comentarios y documentación

- Los comentarios de **autor** y **versión** se aplican sólo a las **clases**.

- Los comentarios de **parámetros**, **retorno** y **excepciones** se aplican sólo a los métodos.

- Los comentarios de referencias cruzadas y obsolescencias se pueden aplicar a clases, métodos y atributos.

# 12. Comentarios y documentación

```
/**
* Esta clase define un punto en un espacio de dos dimensiones.
* @author Óscar Belmonte Fernández
* @version 1.0, 27 de Octubre de 2004
*/
public class Punto {
        protected float x; /** @param x Coordenada x del punto */
        protected float y;
        /**
        * Constructor por defecto
        */
public Punto() {
        x = 0.0f;
        y = 0.0f;
}
/**
* Constructor con argumentos.
* @param x La coordenada 'x' del punto.
* @param y La coordenada 'y' del punto.
*/
public Punto(float x, float y) {
        this.x = x;
        this.y = y;
}
```

# 12. Comentarios y documentación

```
/**
* Con esta función se recupera el valor de la coordenada solicitada
* @param coordenada La coordenada que se solicita 'x' o 'y'
* @return El valor de la coordenada
* @deprecated Esta función se eliminará en próximas versiones
* @see #getX()
* @see #getY()
*/
public float get (String coordenada) {
	if(coordenada.equals("x")) return x;
	else if(coordenada.equals("y")) return y;
}

/**
* Esta función devuelve el valor de la coordenada 'x'
* @return El valor de la coordenada 'x'
*/
public float getX() {
	return x;
}
```

# 12. Comentarios y documentación

**javadoc** -d Documentacion Punto.java