

Parte I: Programación en un lenguaje orientado a objetos

1. Introducción a los lenguajes de programación

- Lenguajes de alto nivel. El proceso de compilación. El ciclo de vida del software. Concepto de algoritmo. Concepto de clase y objeto. Diagramas de clases. Estructura de un programa. Estructura de un método.

2. Datos y expresiones

3. Estructuras algorítmicas

4. Datos Compuestos

5. Entrada/salida

1.1. Lenguajes de programación

Las instrucciones de un programa son códigos numéricos almacenados en la memoria del computador

Ejemplo de lenguaje máquina para el microprocesador 68000: suma de dos enteros:

Dirección	Código Binario	Código Ensamblador	Alto Nivel
\$1000	0011101000111000	MOVE.W \$1200,D5	Z=X+Y
\$1002	0001001000000000		
\$1004	1101101001111000	ADD.W \$1202,D5	
\$1006	0001001000000010		
\$1008	0011000111000101	MOVE.W \$D5,\$1204	
\$100A	0001001000000100		

Programación del computador

La programación mediante códigos numéricos se conoce como ***lenguaje máquina***

- es muy compleja

Por ello se necesitan lenguajes de programación más cercanos a los programadores

Lenguajes de alto y bajo nivel

Necesitamos escribir programas en un lenguaje más cómodo para los humanos

- lenguaje de ***bajo nivel*** o ensamblador
 - cada instrucción corresponde a una instrucción de lenguaje máquina
 - es dependiente de la máquina
 - teóricamente más eficientes
- lenguajes de ***alto nivel***
 - instrucciones más abstractas y avanzadas
 - son lenguajes independientes de la máquina
 - en la práctica, mucho más productivos

Notas:

Para hacer accesible la programación de un computador a cualquier persona, existen ***lenguajes de programación*** que tienen como función presentar al usuario el computador de acuerdo con un modelo abstracto (**informático**) sencillo e independiente de su estructura electrónica interna:

Los lenguajes tienen dos categorías

- **Lenguaje ensamblador o de bajo nivel.** Cada instrucción de lenguaje ensamblador se corresponde con una instrucción de lenguaje máquina, pero en lugar de codificarse mediante números se codifica mediante símbolos alfanuméricos, más fáciles de recordar. En teoría se puede conseguir más eficiencia, pues podemos usar toda la potencia ofrecida por la máquina. En la práctica, programar en lenguaje ensamblador es tedioso, difícil, con mucha facilidad para cometer errores y, por tanto, poco productivo
- **Lenguajes de alto nivel.** Permiten programar utilizando un lenguaje más próximo al humano, e independiente de la máquina. Ejemplos de este tipo de lenguajes son el Java, FORTRAN, C, BASIC, Pascal, Ada, etc. Los lenguajes de alto nivel producen un código un poco menos eficiente que el ensamblador, pero más sencillo de escribir y mantener, con menos errores, y mucho más productivo.

Traductores: ensambladores, compiladores e intérpretes

Son programas que traducen un programa de aplicación escrito en un lenguaje de programación, a un programa en lenguaje máquina:

- lenguaje ensamblador: se traduce mediante un programa ensamblador
- lenguajes de alto nivel: se traducen mediante compiladores e intérpretes
 - los compiladores traducen el programa de aplicación antes de que éste se ejecute
 - los intérpretes van traduciendo el programa de aplicación a medida que se va ejecutando

Notas:

El vendedor de un computador, acompaña el equipo físico con un conjunto de programas de ayuda, desarrollados por él o por personas especializadas, que tienen como función presentar al usuario el computador de acuerdo con un modelo abstracto (**informático**) sencillo e independiente de su estructura electrónica interna:

- ensambladores, compiladores e intérpretes
- sistema operativo
- entorno de desarrollo y algunos programas de aplicación

Los ensambladores, compiladores e intérpretes traducen un programa escrito en un lenguaje más o menos cómodo para el usuario, a lenguaje máquina. Según el lenguaje utilizado, las herramientas son:

- **Lenguajes de bajo nivel:** se utiliza un programa ensamblador, que traduce los símbolos alfanuméricos a código máquina, por medio de algoritmos muy simples.
- **Lenguajes de alto nivel.** La traducción a lenguaje máquina se hace mediante compiladores (que traducen el programa escrito en lenguaje de alto nivel de forma completa antes de su ejecución) e intérpretes (que traducen cada instrucción mientras se ejecuta el programa). Los intérpretes son más lentos en la ejecución, ya que tienen que hacer el trabajo de traducción cada vez que se ejecuta el programa. Hoy en día es más habitual usar compiladores.

Los lenguajes de programación de alto nivel han sido definidos como una solución intermedia entre los **lenguajes naturales humanos** y los lenguajes **máquina de los computadores**. Están bien definidos, en el sentido de que la tarea que se puede expresar con ellos no es ambigua y por lo tanto pueden ser traducidos en un programa máquina concreto, de forma automatizada y por el propio (aunque no necesariamente) computador que va a realizar la tarea.

Ejemplos de lenguajes de alto nivel

Los inicios:

- **Fortran**: 1956, para cálculo científico
 - estándares: 1966, 1977, 1990, 1997, 2003, 2008
- **Cobol**: 1960, para aplicaciones de gestión
 - estándar actual: 2002
- **Lisp**: 1958, para inteligencia artificial
 - estandarizado por ANSI en 1994 (common LISP)
 - estandarizado por ISO en 1997, actualizado en 2007 (ISLISP)
 - tiene un dialecto importante llamado *Scheme*
- **Basic**: 1964, para docencia, interpretado
 - Visual Basic - Visual Studio 2015 (Microsoft)

Notas:

Existen muchos lenguajes de alto nivel de propósito general, sus principales diferencias se encuentran en que poseen un conjunto de órdenes mas adecuado para expresar tareas de un tipo concreto de problema o porque corresponden a distintos niveles de evolución de los computadores:

FORTRAN (FORmula TRANslation). Su nombre evidencia la orientación matemática de uno de los lenguajes de alto nivel mas antiguos, que aún perduran. J. Backus lo desarrolló en 1956. Aunque ha perdido terreno frente a los lenguajes mas modernos, todavía es ampliamente utilizado en aplicaciones científicas de grandes cálculos numéricos, porque probablemente, es el lenguaje con mayor número de librerías, desarrolladas y comprobadas por mucha gente, a lo largo de su historia.

COBOL (COmmon Business Oriented Language). Se trata del lenguaje que ha alcanzado una mayor resonancia en las tareas de gestión. Su desarrollo fue promovido por el Departamento de Defensa de Los EEUU, en 1960. El lenguaje ha sufrido muchas extensiones, y ha sido actualizado recientemente.

LISP (LISt Processing). El Massachusetts Institute of Technology creó, en 1959, este lenguaje de alto nivel orientado a aplicaciones de inteligencia artificial. La programación de procesos recurrentes (edificados sobre datos procesados en los pasos anteriores) es uno de los puntos fuertes del LISP.

BASIC (Beginners All-purpose Symbolic Instruction Code). Nació entre 1964 y 1965 en el Dartmouth College como una herramienta para la enseñanza. Con el tiempo han ido proliferando los dialectos y versiones, hasta el punto de que es raro el fabricante que no desarrolla un dialecto para sus propios equipos. Fue muy popular por su sencillez, pero tiene carencias importantes.

Ejemplos de lenguajes (cont.)

Hacia la programación estructurada:

- **Pascal**: 1969, para docencia, programación estructurada
 - evolucionó hacia Object Pascal (Delphi) y otras variantes
- **C**: 1972, para programación del software del sistema
 - estandarizado en 1990, 1999, y 2011
- **Ada 83**: 1983, para sistemas de alta integridad, incluyendo sistemas de tiempo real
 - estandarizado en 1983

Notas:

PASCAL (En honor del matemático francés Blaise Pascal). Es un lenguaje de programación desarrollado por el profesor Nicklaus Wirth, en 1969, en el Instituto Federal de Tecnología de Zurich partiendo de los fundamentos del ALGOL. Fue uno de los primeros lenguaje que incorporaron los conceptos de programación estructurada. Aunque fue muy popular, la dificultad para partir el programa en módulos y la falta de estandarización han hecho decaer su uso.

C. Es un lenguaje de programación desarrollado por la Bell Laboratories, en principio para trabajar con el sistema operativo UNIX. Quizás por ello, la popularidad del 'C' es muy alta. Es un lenguaje que, al mismo tiempo que permite una programación en alto nivel, permite una gran aproximación a la máquina. Muchos lo consideran un lenguaje intermedio entre alto y bajo nivel. Como estos últimos, presenta alta eficiencia y escasa fiabilidad. Es fácil cometer errores en C.

Ejemplos de lenguajes (cont.)

Lenguajes de programación orientada a objetos:

- **Smaltalk**: 1980, para programación orientada a objetos
 - estandarizado en 1998
- **C++**: 1987, extensión mejorada del C que incorpora programación orientada a objetos
 - estandarizado en 1998, estandarizado de nuevo en 2003, 2011 y 2014
- **Java**: 1995, para programación orientada a objetos en sistemas distribuidos (red Internet)
 - versión actual: Java 8 (2014)
 - ofrecido como software libre en 2007
- **Ada 95, Ada 2005, Ada 2012**: versiones mejoradas del anterior, incluyendo programación orientada a objetos
 - estandarizado en 1995, 2005 y 2012

Notas:

SMALTALK: Lenguaje de programación orientada a objetos puro. Es muy ineficiente con respecto a lenguajes procedurales como el C o el Ada, pero es cómodo de usar y de programar en él.

JAVA: Lenguaje derivado del C en cuanto a sintaxis, pero más parecido al Ada 95 en cuanto a las comprobaciones que hace el compilador y soporte de la programación concurrente. Está pensado para su ejecución en sistemas distribuidos (internet). Existe un código intermedio, bien definido, que puede intercambiarse entre computadores diferentes para luego ser traducido y ejecutado.

C++: Extensión del lenguaje C que mejora algunos de sus inconvenientes, y añade construcciones de programación orientada a objetos. Entre las mejoras destacan una mayor comprobación de los tipos de datos por parte del compilador, las excepciones, y las plantillas genéricas.

ADA (En honor de Lady Augusta ADA Byron). El ADA es un lenguaje inspirado en el PASCAL, que fue promovido por el Departamento de Defensa de Los EEUU. El objetivo de su desarrollo era conseguir un lenguaje con posibilidades de convertirse en un estándar universal y que facilitara la ingeniería de software y el mantenimiento de los programas. Entre sus campos de aplicación se incluyen los sistemas de tiempo real y los sistemas de alta integridad. El 1995 se revisó el lenguaje para mejorarlo y para añadirle construcciones de programación orientada al objeto. En 2005 se finalizó una nueva versión y en 2012 se está completando una nueva.

Ejemplos de lenguajes (cont.)

Otros lenguajes de programación:

- **C#** (C sharp): orientado a objetos y orientado a componentes; sintaxis basada en C++
 - introducido por Microsoft en 2000, estandarizado por ISO en 2003 y 2006 (versión 2.0)
 - versión actual: 7.0 (2017)
- **Objective-C**: es un superconjunto de C al que se añade la programación orientada a objetos de Smalltalk
 - se origina en los años 80
 - se utiliza como lenguaje de programación principal de Apple para OS X e iOS a partir de 1996
 - versión actual: 2.0 (2007)
 - En 2014 Apple ha creado a partir de él *Swift*, un lenguaje para construir programas más fiables. Versión actual 4.0 (2017)

Ejemplos de lenguajes (cont.)

Otros lenguajes de programación:

- **GO**: lenguaje *sencillo* con programación concurrente y seguridad en el acceso a memoria
 - Creado por Google en 2009
 - versión actual: 1.9, 2017

Ejemplos de lenguajes (cont.)

Lenguajes de guiones o “scripts”:

- **PHP**: desarrollado para hacer páginas Web dinámicas, soportado por la mayoría de servidores Web
 - desarrollado en 1995, versión actual 7.1 (2017)
- **Perl**: interpretado, con tipos dinámicos, no es orientado a objetos. Tiene facilidades de manipulación de textos
 - desarrollado en 1987, versión actual 5.26 (2017)
- **Python**: habitualmente interpretado, orientado a objetos, con tipos dinámicos; hace énfasis en la legibilidad
 - es de finales de los 80; versiones actuales 2.7 y 3.6 (2017)
 - Python 3 es incompatible con Python 2 y por ello subsisten ambas versiones

Ejemplos de lenguajes (cont.)

Lenguajes de guiones o "scripts" (continuación):

- **JavaScript**: interpretado, orientado a objetos, soportado por muchos navegadores Web. Sintaxis inspirada en Java
 - creado en 1995, propiedad de Oracle; versión actual 1.8.5 (2010)
 - estandarizado en 1997 como ECMAScript; versión actual 8 (2017)
- **Ruby**: Lenguaje de "scripts", interpretado, orientado a objetos, con tipos dinámicos. Sintaxis inspirada en Perl y Smalltalk
 - creado en 1993, versión 1.8 estandarizada en 2012 (ahora retirada)
 - versión actual 2.4 (2017)
- **R**: Lenguaje para cálculo estadístico y representación gráfica. Es orientado a objetos
 - creado en 1997, no estandarizado.
 - versión actual 3.4 (2017)

Notas:

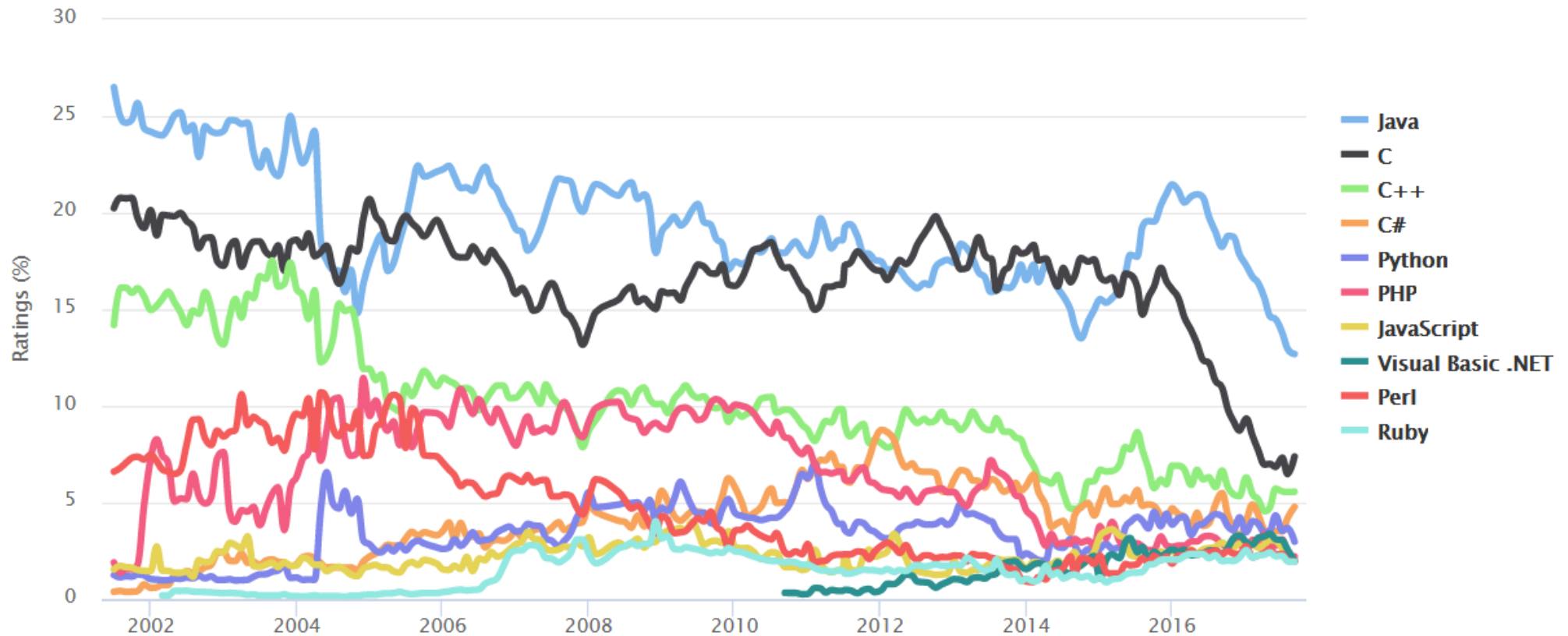
Un lenguaje de “scripts” o de guiones es un lenguaje de programación para controlar la ejecución de otras aplicaciones. Suelen ser interpretados. Tienen mucho éxito para programar aplicaciones web.

Un lenguaje con tipos dinámicos es aquel en el que la comprobación del tipo de dato que se está utilizando se hace durante la ejecución, en lugar de hacerse de manera estática durante la compilación.























Ránking de lenguajes de programación

TIOBE Programming Community Index

Source: www.tiobe.com



Ránking de lenguajes de programación

Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

Notas:

Fuentes:

<http://www.tiobe.com/tiobe-index/>

- Fecha: Sep 2017

<https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>

- Fecha: 2017

El lenguaje Java

Desarrollado por la empresa Sun Microsystems (ahora Oracle) en 1995

Recibe una amplia aceptación

Objetivos generales:

- **WORA**: (**W**rite **O**nce, **R**un **A**n anywhere)
- Portabilidad sin necesidad de recompilar
- Programación distribuida
- Orientado a objetos
 - abstracción de datos
 - modularidad, encapsulado, y ocultamiento de información
 - herencia y polimorfismo

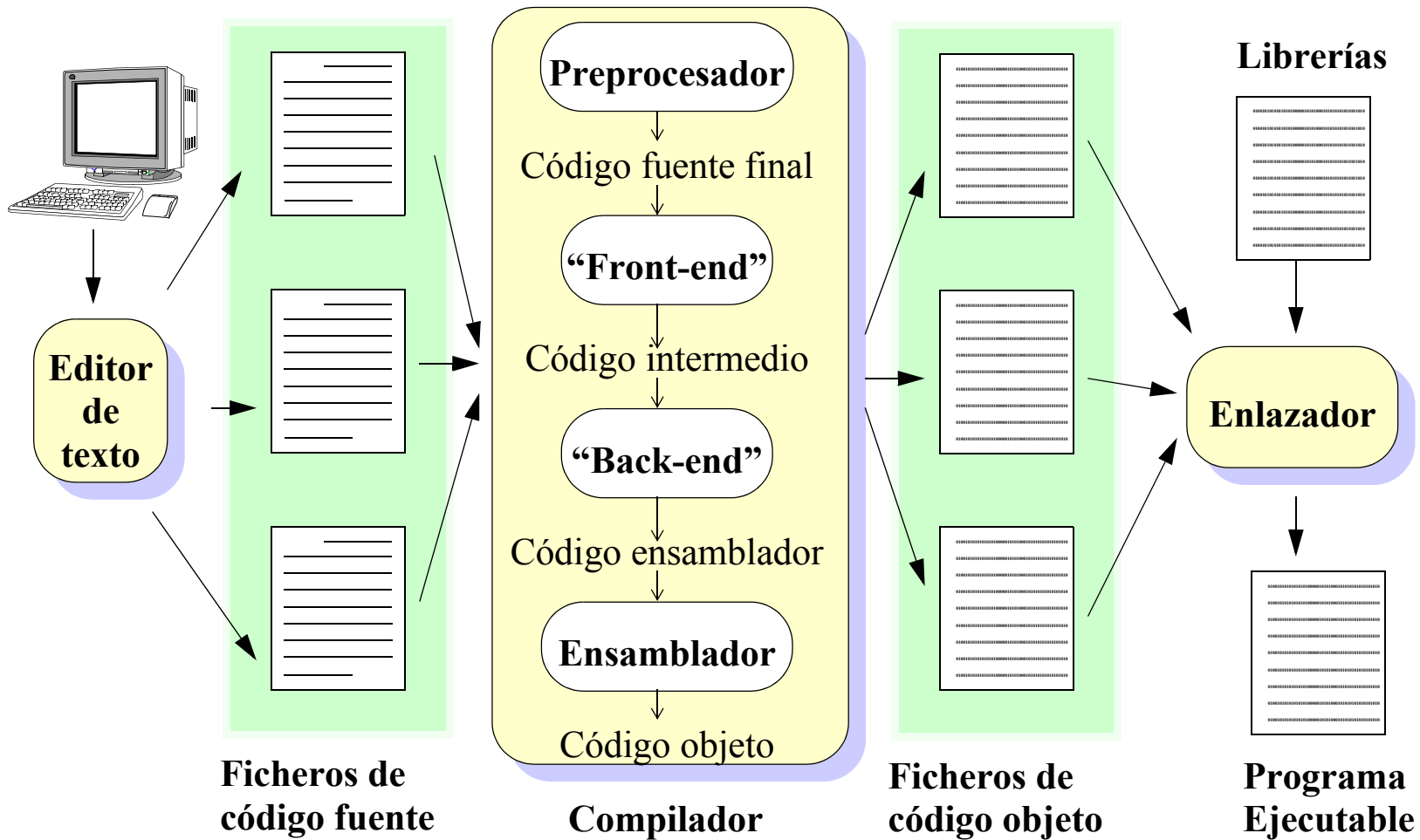
El lenguaje Java (cont.)

Objetivos generales (cont.):

- Más fiable y seguro que C o C++:
 - memoria dinámica automática, que evita los punteros explícitos
 - tipificación estricta
 - comprobación automática de tamaños de variables
- Sintaxis similar a la del C/C++
- Concurrencia integrada en el lenguaje
- Excepciones declaradas
- Interfaz gráfica integrada en el lenguaje

La versión actual es la 1.8 (Java 8)

1.2. El proceso de compilación



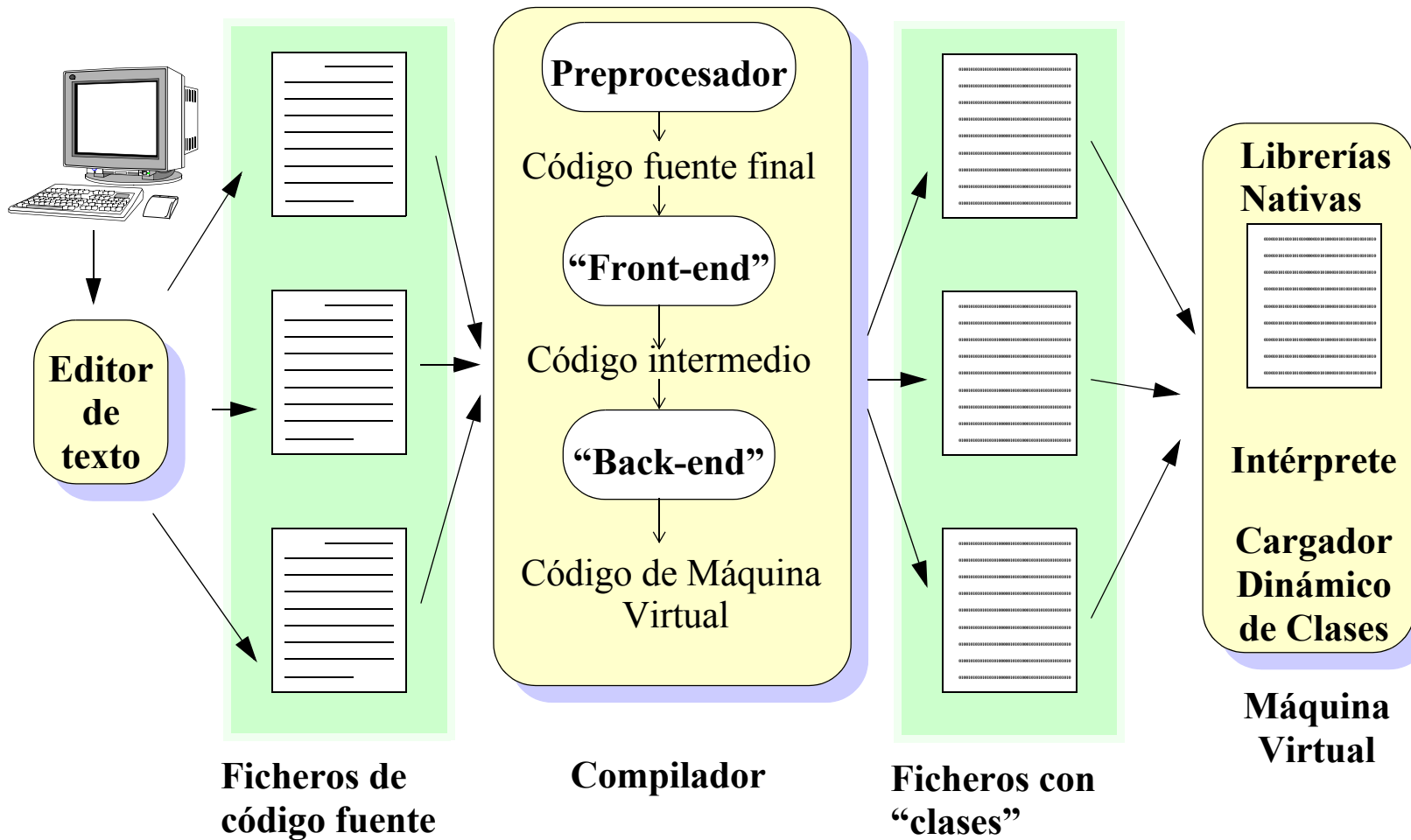
Notas:

Un compilador puede definirse como una herramienta automática de traducción que lee un programa escrito en un lenguaje (el lenguaje fuente) y lo traduce a un programa equivalente en otro lenguaje (el lenguaje objeto). En el proceso de traducción el compilador notifica al usuario de la presencia de errores en el programa fuente.

La variedad de compiladores que pueden aparecer es muy alta. Existen miles de lenguajes fuente. Igualmente ocurre con los lenguajes objeto: pueden ser otros lenguajes de programación, o el lenguaje máquina de cualquier computador entre un microprocesador o un supercomputador. En cualquier caso las tareas que debe realizar son las mismas.

Además del compilador también son necesarios otros programas para crear un programa ejecutable: el preprocesador, el ensamblador, el enlazador, y el cargador. En la figura de arriba se muestra un proceso de compilación típico.

Compilación para máquina virtual



Notas:

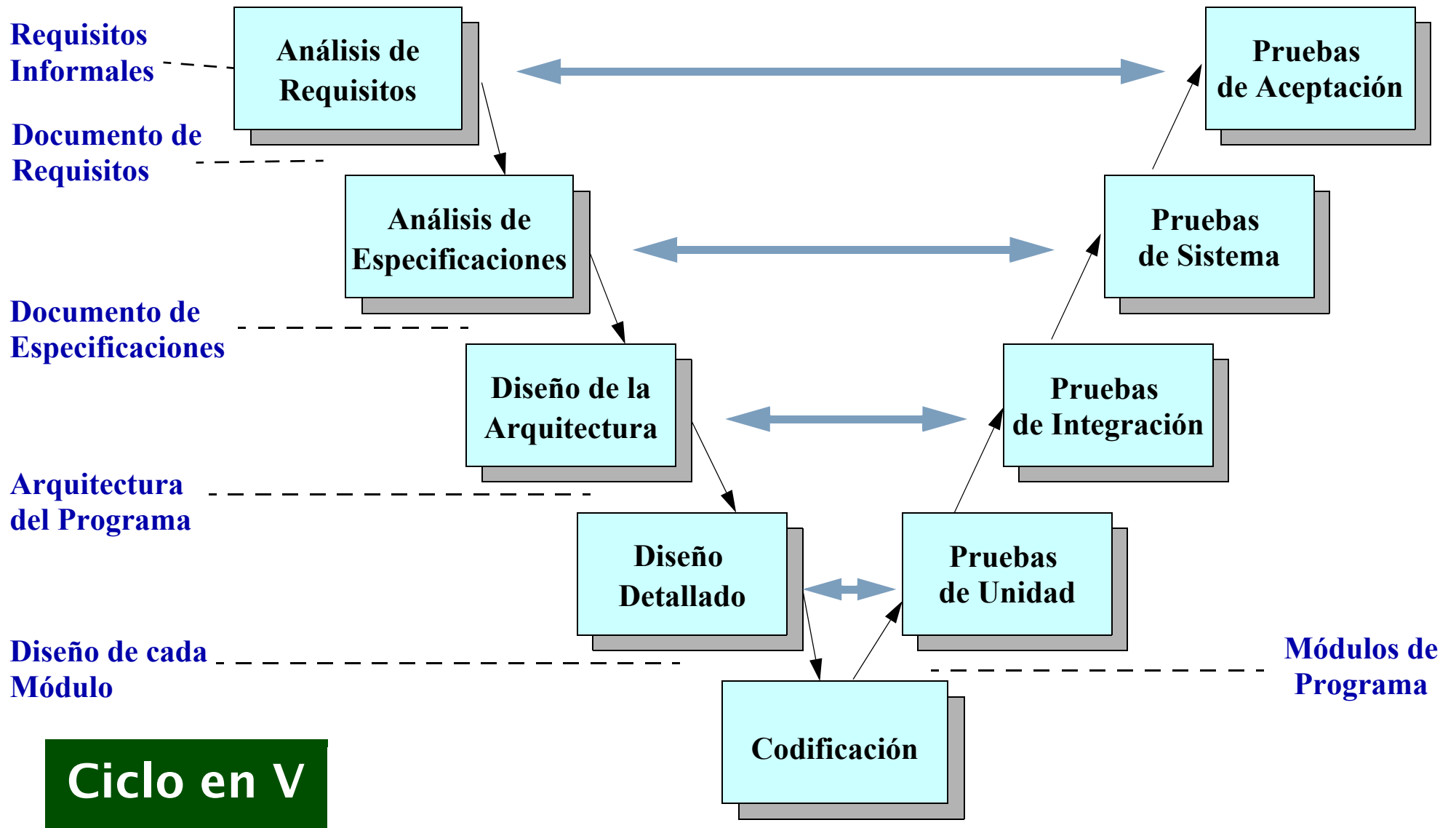
La arquitectura del entorno de ejecución de programas Java se basa en una máquina virtual, que se ejecuta en el computador para interpretar las instrucciones del programa del usuario descritas en un código intermedio especial, llamado código de máquina virtual Java (el Java Byte Code).

La idea principal de esta arquitectura es la de "Escribir una vez, ejecutar en cualquier sitio". El compilador de Java no genera código máquina de un computador concreto, sino un código especial, que luego es interpretado por otro programa, llamado máquina virtual, que existe en cada computador en el que se desea ejecutar el programa Java. De este modo, un programa Java se puede ejecutar indistintamente en cualquier computador que disponga de esa máquina virtual, sin necesidad de recompilarlo.

Adicionalmente, en la arquitectura Java los programas no se enlazan antes de su ejecución, sino que se utiliza un enlazado dinámico. Cuando se hace una llamada a una operación de un módulo (clase) que no está cargado en la máquina virtual, ésta se encarga de buscar ese módulo y cargarlo en ese momento en la máquina virtual.

Desde el programa del usuario se pueden utilizar operaciones "nativas", suministradas por la máquina virtual, escritas generalmente en código máquina, y que pueden acceder a los dispositivos hardware del computador. El resultado es: programas muy portables, muy dinámicos, aunque poco eficientes.

1.3. El ciclo de vida del software

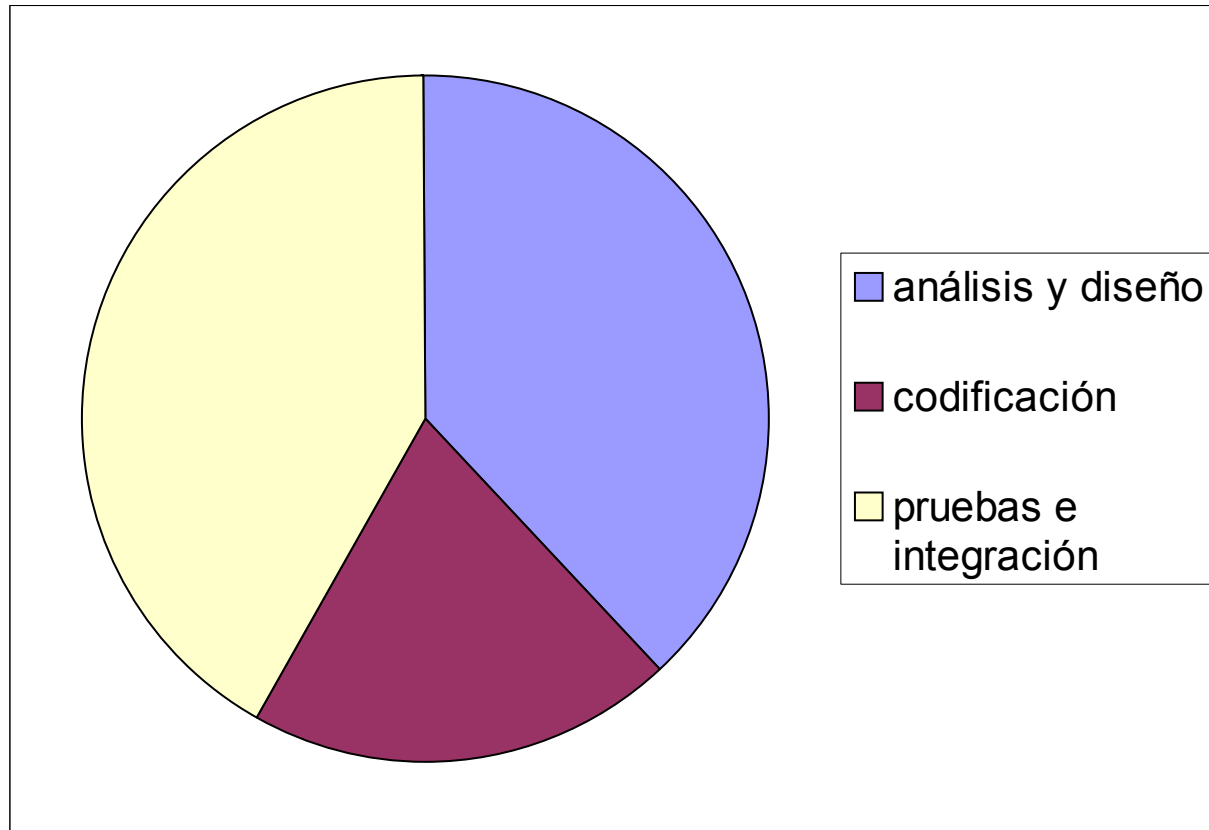


Notas:

Los pasos que se siguen generalmente a la hora de desarrollar un programa son los siguientes:

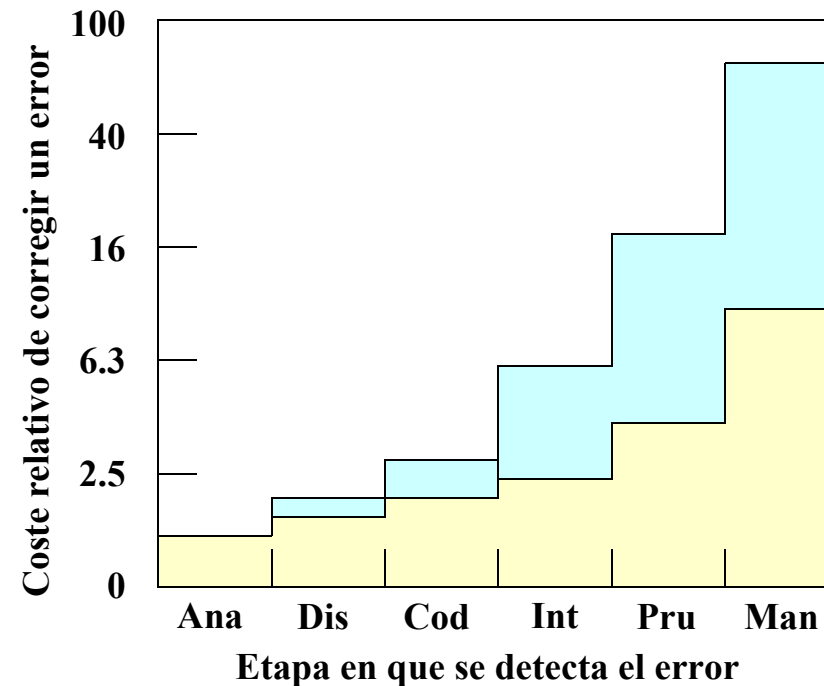
- **Análisis de requisitos:** Se define el problema a resolver y todos los objetivos que se pretenden, pero sin indicar la forma en la que se resuelve.
- **Especificación:** Se determina la forma en la que se resolverá el problema, pero sin entrar aún en su implementación informática. Se determina asimismo la interfaz con el usuario.
- **Diseño de la arquitectura del programa:** Se divide el problema en módulos, se especifica lo que hace cada módulo, así como las interfaces de cada uno de ellos.
- **Diseño detallado de los módulos:** Para cada módulo se diseñan detalladamente las estructuras de datos y los algoritmos a emplear.
- **Codificación:** Se escribe el programa en el lenguaje de programación elegido.
- **Pruebas de módulos:** Se prueban los módulos del programa aisladamente y se corrigen los fallos hasta conseguir un funcionamiento correcto.
- **Integración y Prueba de sistema:** Se unen todos los módulos, y se prueba el funcionamiento del programa completo.
- **Prueba de aceptación:** Instalación en el lugar definitivo y aceptación por parte del cliente

Distribución del Esfuerzo de desarrollo



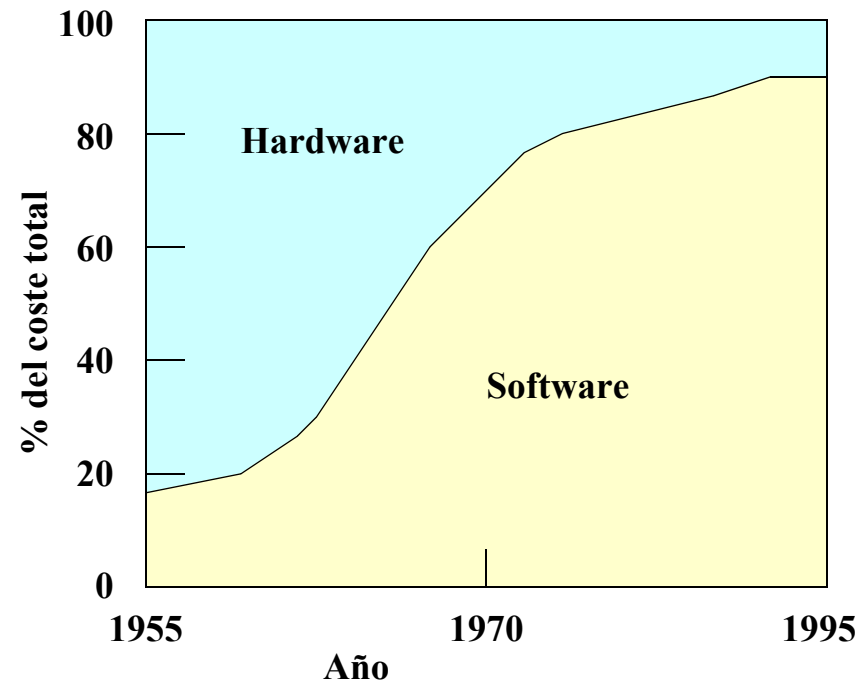
Coste de los sistemas informáticos

Los errores software tienen un alto coste: efecto y corrección



Coste de los sistemas informáticos

Relación entre el coste de HW y SW



1.4. Concepto de algoritmo

Un algoritmo es:

- una secuencia finita de instrucciones,
- cada una de ellas con un claro significado,
- que puede ser realizada con un esfuerzo finito
- y en un tiempo finito

El algoritmo se diseña en la etapa de diseño detallado y se corresponde habitualmente con el nivel de operación o método

Los programas se componen habitualmente de muchas clases que contienen algoritmos, junto con datos utilizados por ellos

- los datos y algoritmos relacionados entre sí se encapsulan en la misma clase

Ejemplo de algoritmo

Programa que intenta mantener una habitación a una temperatura deseada (± 0.5 grados)

- dispone de un radiador que se puede encender y apagar,
- y un termómetro

Variables:

- `tempDeseada`: magnitud real ($^{\circ}\text{C}$)
- `tempAmbiente`: magnitud real ($^{\circ}\text{C}$)
- `estadoRadiador`: encendido o apagado

Estado del programa:

- valor de `tempDeseada`, `tempAmbiente`, y `estadoRadiador` en cada instante

Ejemplo (cont.)

Algoritmo

```
repetir continuamente lo siguiente  
  si hace frío encender el radiador  
  si hace calor apagar el radiador  
  esperar un rato  
fin repetir
```

El algoritmo se repite continuamente (hasta que el usuario apague el sistema)

Ahora debemos refinar este algoritmo para expresarlo en términos de las variables del sistema

Ejemplo (cont.)

Algoritmo refinado

```
repetir continuamente lo siguiente
  si tempAmbiente < tempDeseada - 0.5 entonces
    // hace frío
    estadoRadiador := encendido
  fin si
  si tempAmbiente > tempDeseada + 0.5 entonces
    // hace calor
    estadoRadiador := apagado
  fin si
  esperar 1 minuto
fin repetir
```

Observar que si no hace ni frío ni calor el radiador se queda como estaba

A observar

Hemos descrito el algoritmo mediante la técnica llamada pseudocódigo, que tiene

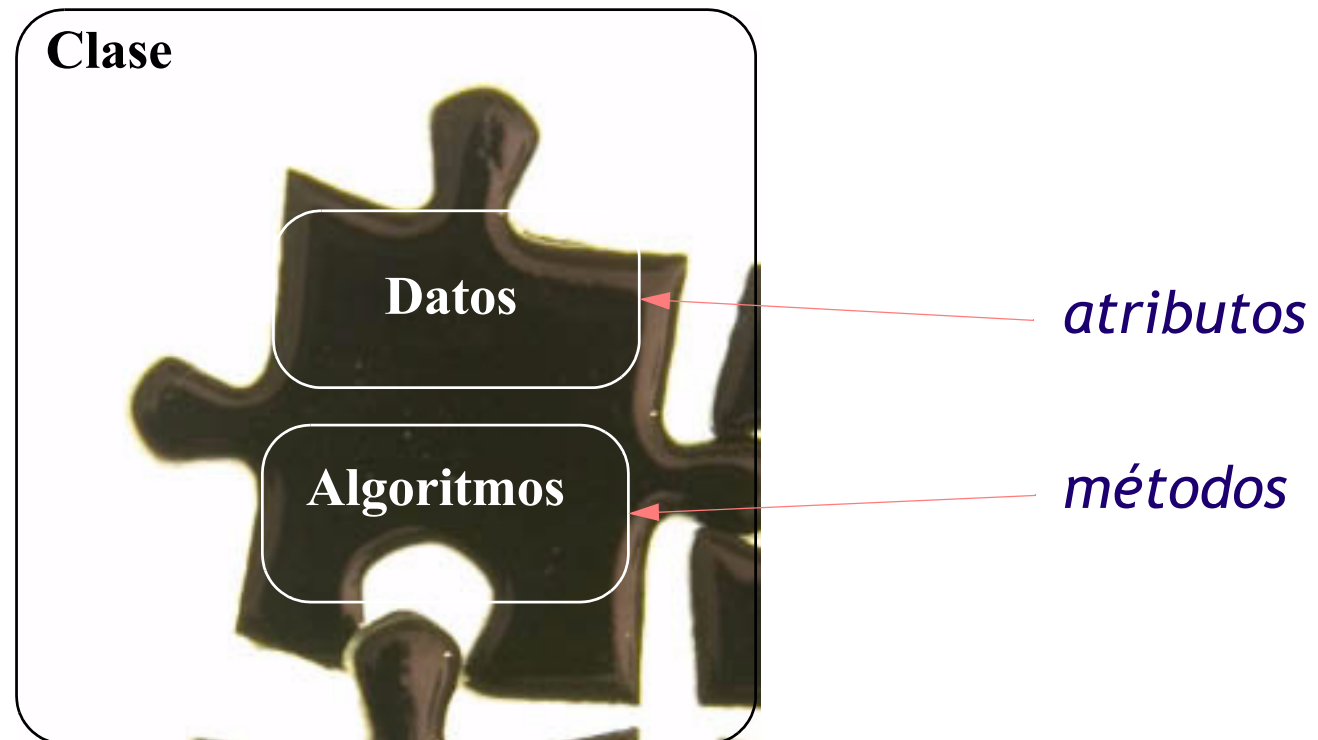
- instrucciones de control presentes en todos los lenguajes
 - si** condición **entonces**
hacer cosas
 - fin si**
- obsérvese el uso del sangrado para determinar el ámbito de aplicación de cada instrucción de control
- cálculos
- acciones expresadas sin el formalismo de los lenguajes

El propósito es que el pseudocódigo refinado sea sencillo, y directamente traducible a código en cualquier lenguaje

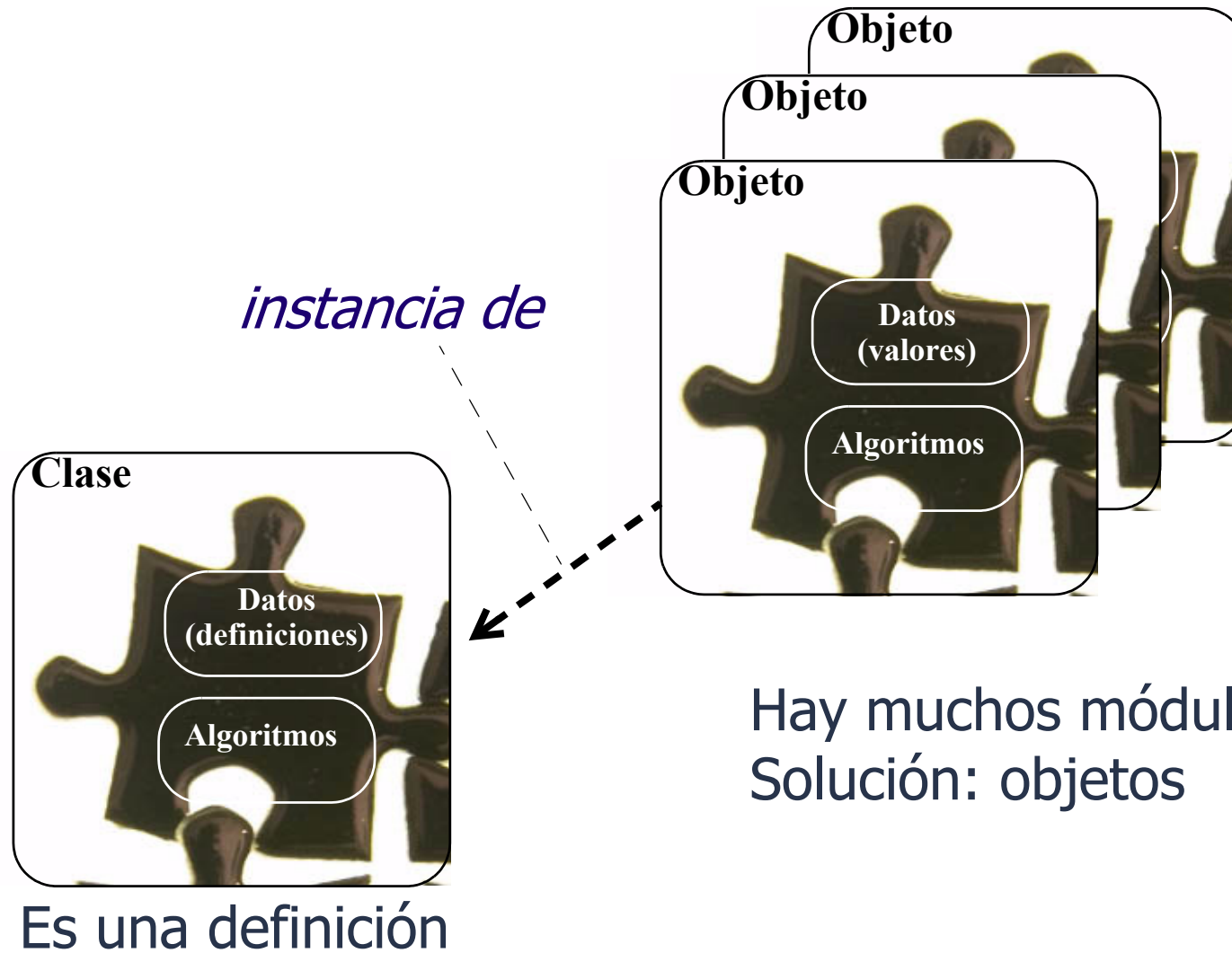
1.5. Concepto de clase y objeto

Los programas Java se construyen mediante *clases*

Una *clase* representa una definición de un módulo de programa



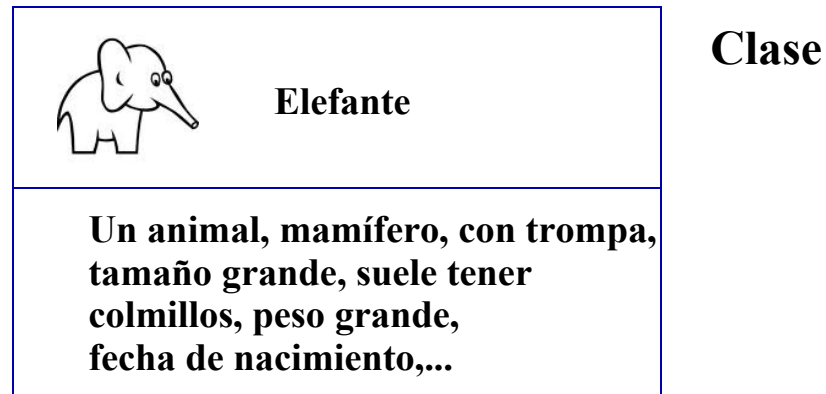
Objetos:



Hay muchos módulos parecidos
Solución: objetos

Diferencia entre clase y objeto

En la clase se *definen* las características comunes de un conjunto de objetos

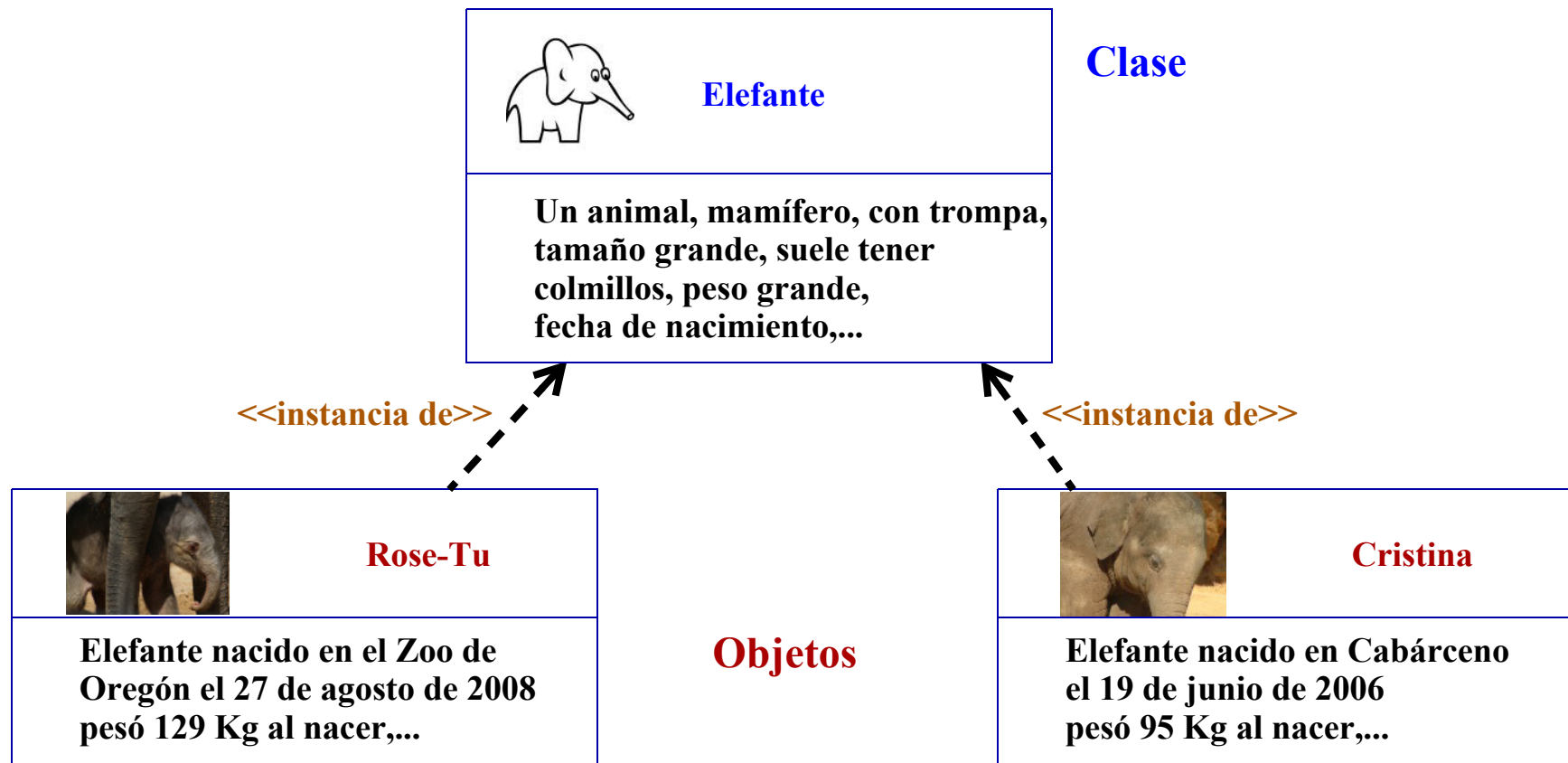


Define las características de todos los elefantes



Diferencia entre clase y objeto

Las instancias de una clase son *objetos* concretos que tienen las características de la clase y valores concretos de sus datos



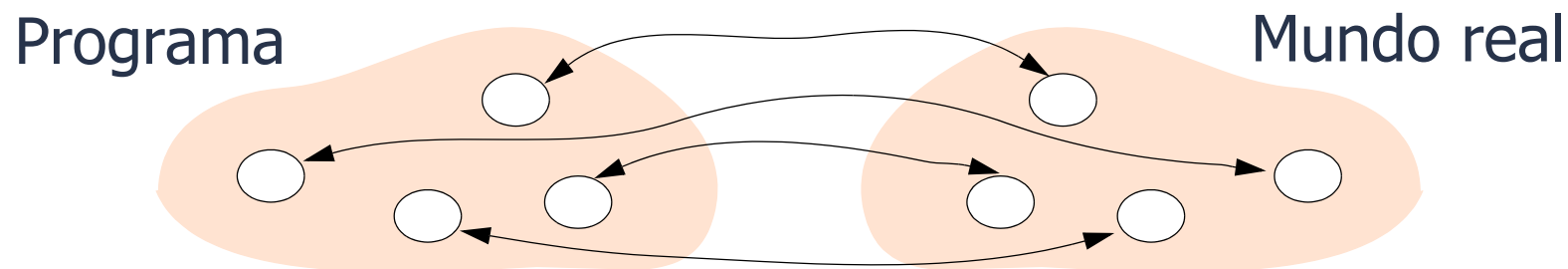
Concepto de clase y objeto

Un **objeto** es un elemento de programa que se caracteriza por:

- **atributos**: son datos contenidos en el objeto y determinan su estado
- **operaciones** o **métodos**: son acciones con las que podemos solicitar información del objeto, o modificarla
 - Secuencias de instrucciones que operan con los atributos
 - y pueden invocar operaciones de otros objetos

Ambos, atributos y métodos, se **definen** en la clase

Se intenta siempre corresponder los objetos de un programa con objetos del problema que éste resuelve



Ejemplo, círculo que pintamos en una pantalla

Los círculos se representan con una clase de objetos, y cada objeto es un círculo concreto

- **Atributos** del círculo:
 - el radio
 - las coordenadas (X,Y) del centro
 - el color
- **Operaciones** a realizar con el círculo:
 - crear un círculo nuevo
 - pintar el círculo
 - cambiar el color
 - cambiar el radio
 - cambiar las coordenadas del centro
 - obtener el radio o el área

1.6. Diagrama de Clases

Define

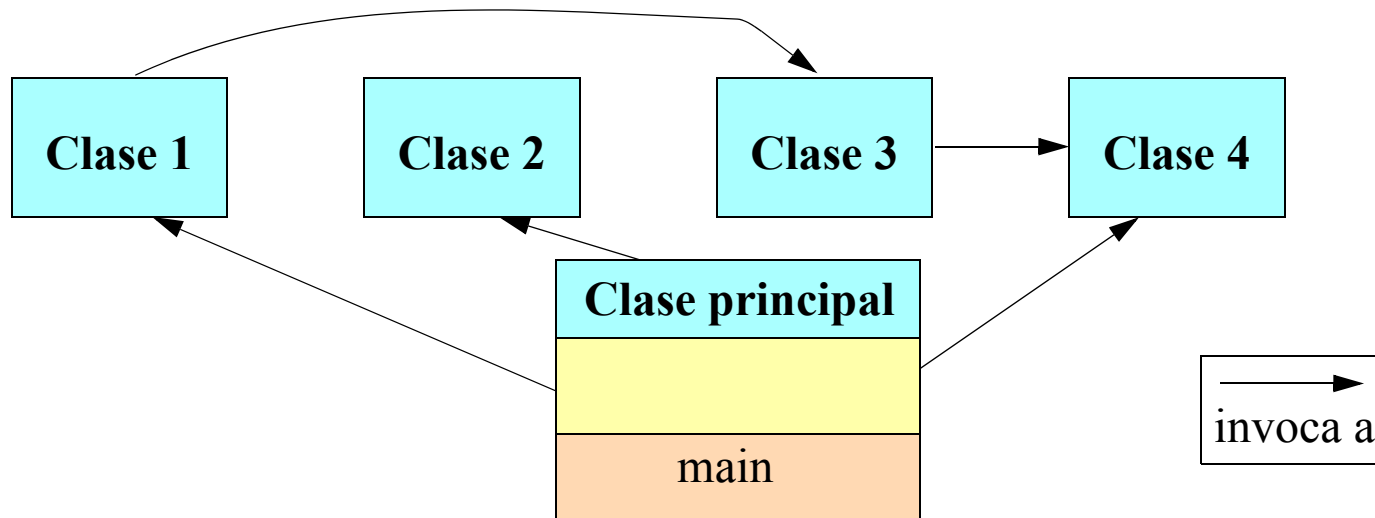
- **nombre** de la clase
- **atributos**: nombres y tipos
- **métodos**: nombre, parámetros (con nombre y tipo de cada uno), y valor retornado (tipo)
 - pueden ser constructores, que inicializan el objeto y se llaman como la clase

Circulo
-real centroX -real centroY -real radio -Color col
+Circulo(real x, real y real r, Color col) +cambiaRadio(real r) +cambiaCentro(real x, real y) +cambiaColor(Color col) +real radio() +real area() +dibuja()

1.7. Estructura de un programa

Un programa java es un conjunto de clases, donde una de ellas es especial en dos aspectos

- contiene una operación llamada `main`
- habitualmente no se crean objetos de esta clase
 - el sistema operativo invoca el método `main` al ejecutar el programa



Estructura de una clase

El caso más sencillo es el de un programa con una sola clase, que tiene esta estructura:

```
import clases importadas;  
public class Nombre {  
    atributos  
    operaciones  
}
```

Las clases importadas son aquellas pertenecientes a paquetes externos

- un ***paquete*** es una colección de clases

En una clase principal, una de las operaciones se llama **main**

1.8. Estructura de un método

Las operaciones o métodos (como `main`) tienen la siguiente estructura:

```
descriptores valor_retornado nombre (argumentos)
{
    declaraciones;

    instrucciones;
}
```

Las declaraciones declaran datos (variables o constantes), clases y objetos que se van a usar en las instrucciones

Las instrucciones representan cálculos que se hacen con esos datos.

Ejemplo de programa

Por ejemplo, el siguiente programa pone un mensaje en la pantalla:

```
/**
 * Programa que pone un mensaje en pantalla
 */
public class Hola {

    /**
     * Este es el método principal
     */
    public static void main(String[] args) {
        // No hay declaraciones
        System.out.println("Hola, Que tal estás?");
    }
}
```


Explicación:

- `public`: el método se puede usar desde fuera
- `static`: el método pertenece a la clase (no a los objetos de la clase)
- `void`: no retorna nada
- `String[] args`: es el argumento, datos que se pasan a la operación
- `System`: es una clase predefinida que representa al computador
- `out`: es un objeto de la clase `System`, predefinido: representa la pantalla
- `println`: método para poner un texto en la pantalla

Algunos detalles

- formato libre
- ***sangrado***: margen del texto, que se usa para remarcar la estructura del programa
- ***comentarios*** de *documentación*
 - multilinea, con principio y final (`/** */`)
 - *recomendable* para todas las clases, métodos y atributos *públicos*
- ***comentarios internos***
 - multilinea, con principio y final (`/* */`)
 - hasta fin de línea (`//`)
- uso de ";" para finalizar declaraciones e instrucciones
- uso de {} para definir los contenidos de clases y métodos

Algunos detalles

- ***nombre de la clase***: primera letra con mayúscula y demás minúsculas, excepto inicios de palabra
- las declaraciones e instrucciones pueden mezclarse en cualquier orden
 - pero es habitual poner primero las declaraciones, y luego las instrucciones