LENGUAJES DE ALTO NIVEL

Sergio López y David Lázaro

I.- PHYTON

Es un lenguaje de programación habitualmente comparado con TCL, Perl, Scheme, Java y Ruby. Actualmente, Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. La última versión estable del lenguaje es actualmente (Septiembre de 2006) la 2.5 .Guido van Rossum, más conocido como Guido, creó Python, un lenguaje de programación de scripting, la "oposición leal" a Perl, lenguaje con el cual mantiene una rivalidad amistosa. Los usuarios de Python consideran a éste mucho más limpio y elegante para programar.

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender Python). También hay módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, sockets y hasta interfaces a GUI (interfaz gráfica con el usuario) como Tk, GTK, Qt entre otros...

Python es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa. También es una calculadora muy útil.

El nombre del lenguaje proviene de la afición de su creador original, Guido van Rossum, por los geniales humoristas británicos Monty Python. El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño.

(1) Filosofía

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos:Programación orientada a objetos, programación estructurada, programación funcional y programación orientada a aspectos. Otros muchos paradigmas más están soportados mediante el uso de extensiones. Python usa tipado dinámico de datos y reference counting para el manejo de memoria. Una característica importante del Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa.

Otro objetivo del diseño del lenguaje era la facilidad de extensión. Nuevos módulos pueden ser fácilmente escritos en C o C++. Python puede ser utilizado como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable. Aunque el diseño de Python es de alguna manera hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia del Lisp como puede ser el Scheme.

II.- Ada

Ada es un lenguaje de programación estructurado y fuertemente tipado de forma estática que fue diseñado por Jean Ichbiah de CII Honeywell Bull por encargo del Departamento de Defensa de los Estados Unidos (DoD). Es un lenguaje multipropósito, orientado a objetos y concurrente, pudiendo llegar desde la facilidad de Pascal hasta la flexibilidad de C++.

Fue diseñado con la seguridad en mente y con una filosofía orientada a la reducción de errores comunes y difíciles de descubrir. Para ello se basa en un tipado muy fuerte y en chequeos en tiempo de ejecución (desactivables en beneficio del rendimiento). La sincronización de tareas se realiza mediante la primitiva rendezvous.

Ada se usa principalmente en entornos en los que se necesita una gran seguridad y confiabilidad como la defensa, la aeronáutica (Boeing o Airbus), la gestión del tráfico aéreo (como Indra en España) y la industria aeroespacial entre otros.

El lenguaje fue diseñado bajo encargo del DoD. Durante los años 1970, este departamento tenía proyectos en una infinidad de lenguajes y estaba gastando mucho dinero en software. Para solucionarlo se buscó un lenguaje único que cumpliese unas ciertas normas recogidas en el documento Steelman. Después de un estudio de los lenguajes existentes en la época se decidió que ninguno las cumplía totalmente, por lo que se hizo un concurso público al que se presentaron cuatro equipos, cuyas propuestas se nombraron con un color: Rojo (Intermetrics), Verde (CII Honeywell Bull), Azul (SofTEch) y Amarillo (SRI International). Finalmente en mayo de 1979 se seleccionó la propuesta Verde diseñada por Jean Ichbiah de CII Honeywell Bull, y se le dio el nombre de Ada. Esta propuesta era un sucesor de un lenguaje anterior de este equipo llamado LIS y desarrollado durante los años 1970.

El nombre se eligió en conmemoración de lady Ada Augusta Byron (1816-1852) Condesa de Lovelace, hija del poeta Lord George Byron, a quien se considera la primera programadora de la Historia, por su colaboración y relación con Charles Babbage, creador de la máquina analítica.

El lenguaje se convirtió en un estándar de ANSI en 1983 (ANSI/MIL-STD 1815) y un estándar ISO en 1987 (ISO-8652:1987).

El DoD y los ministerios equivalentes de varios países de la OTAN exigían el uso de este lenguaje en los proyectos que contrataban (el *Ada mandate*). La obligatoriedad en el caso de Estados Unidos terminó en 1997, cuando el DoD comenzó a usar productos COTS (*commercial off the shelf*).Ada ha sido utilizado por la NASA en sistemas VAX.

- La sintaxis, inspirada en Pascal, es bastante legible incluso para personas que no conozcan el lenguaje. Es un lenguaje que no escatima en la longitud de las palabras clave, en la filosofía de que un programa se escribe una vez, se modifica decenas de veces y se lee miles de veces (legibilidad es más importante que rapidez de escritura).

- Identificadores y palabras claves son equivalentes sea cual sea el uso de mayúsculas y minúsculas, es decir es un lenguaje *case-insensitive*.
- En este caso, todo el programa es un único procedimiento, que puede contener subprogramas (procedimientos o funciones) (en este caso: la función `Ack`).
- Cada sentencia se cierra con **`end`** *`qué_cerramos`*. Es un modo de evitar errores y facilitar la lectura. No es necesario hacerlo en el caso de subprogramas, aunque todos los manuales lo aconsejan y casi todos los programadores de Ada lo hacen.
- El operador de asignación es `:=`, el de igualdad `=`. A los programadores de C y similares les puede confundir este rasgo inspirado en Pascal.
- La sintaxis de atributos predefinidos es `Objeto'Atributo` (o `Tipo'Atributo`) (nota: esto sólo aplica a atributos predefinidos por el lenguaje, ya que no es el concepto de atributo típico de OOP).
- Se distingue entre "procedimientos" (subrutinas que no devuelven ningún valor pero pueden modificar sus parámetros) y "funciones" (subrutinas que devuelven un valor y no modifican los parámetros). Muchos lenguajes de programación no hacen esta distinción. Las funciones de Ada favorecen la seguridad al reducir los posibles efectos colaterales, pues no pueden tener parámetros **`in out`**.

III.- BASIC

Es una familia de lenguajes de programación. Fue originalmente ideado como una herramienta de enseñanza, se diseminó entre las microcomputadores caseras en la década de 1980, y sigue siendo popular hoy en día en muchos dialectos bastante distintos del original.

BASIC es el acrónimo de ***B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode*[1] (en español 'código de instrucciones simbólicas de propósito general para principantes') y está ligado al nombre de un trabajo sin publicar del coinventor del lenguaje, Thomas Kurtz (el nombre no está relacionado con la serie de C. K. Ogden, *Basic English*).

Antes de mediados de la década de 1960, las computadoras eran herramientas sumamente caras que eran utilizadas únicamente para propósitos especiales, ejecutando una sola "tarea" a la vez. Sin embargo, durante esa década, los precios comenzaron a bajar al punto que incluso las pequeñas empresas podían costearlas. La velocidad de las máquinas se incrementó al punto que a menudo quedaban ociosas porque no había suficientes tareas para ellas. Los lenguajes de programación de aquellos tiempos estaban diseñados como las máquinas en las que corrían: para propósitos específicos como el procesamiento de fórmulas. Como las máquinas para una sola tarea eran caras, se consideraba que la velocidad de ejecución era la característica más importante de todas. En general, todas eran difíciles de utilizar, y aportaban poca estética.

Fue en aquellos tiempos que el concepto de sistema de Tiempo compartido comenzó a popularizarse. En uno de estos sistemas, el tiempo de procesamiento de la computadora principal se dividía, y a cada usuario se le otorgaba una pequeña porción en una secuencia. Las máquinas eran lo suficientemente rápidas como para engañar a la mayoría de usuarios, dándoles la ilusión de que disponían de una máquina entera solo

para ellos. En teoría la distribución del tiempo entre los usuarios redujo considerablemente el costo de la computación, ya que una sola máquina podía ser compartida, al menos en teoría, entre cientos de usuarios.

(1) Nacimiento y primeros años

El lenguaje BASIC original fue inventado en 1964 por John George Kemeny (1926-1993) y Thomas Eugene Kurtz (1928-) en el Dartmouth College. En los años subsiguientes, mientras que otros dialectos de BASIC aparecían, el BASIC original de Kemeny y Kurtz era conocido como *BASIC Dartmouth*.

BASIC fue diseñado para permitir a los estudiantes escribir programas usando terminales de computador de tiempo compartido. BASIC estaba intencionado para facilitar los problemas de complejidad de los lenguajes anteriores, con un nuevo lenguaje diseñado específicamente para la clase de usuarios que los sistemas de tiempo compartido permitían: un usuario más sencillo, a quien no le interesaba tanto la velocidad, sino el hecho de ser capaz de usar la máquina. Los diseñadores del lenguaje también querían que permaneciera en el dominio público, lo que contribuyó a que se diseminara.

Los ocho principios de diseño de BASIC fueron:

1. Ser fácil de usar para los principiantes.
2. Ser un lenguaje de propósito general.
3. Permitir que los expertos añadieran características avanzadas, mientras que el lenguaje permanecía simple para los principiantes.
4. Ser interactivo.
5. Proveer mensajes de error claros y amigables.
6. Responder rápido a los programas pequeños.
7. No requerir un conocimiento del hardware de la computadora.
8. Proteger al usuario del sistema operativo.

El lenguaje fue en parte basado en FORTRAN II y en parte en Algol 60, con adiciones para hacerlo apropiado para tiempo compartido y aritmética de matrices, BASIC fue implementado por primera vez en la mainframe GE-265[2], que soportaba múltiples terminales. Contrario a la creencia popular, era un lenguaje compilado al momento de su introducción. Casi inmediatamente después de su lanzamiento, los profesionales de computación comenzaron a alegar que BASIC era muy lento y simple. Tal argumento es un tema recurrente en la industria de las computadoras.

Aún así, BASIC se expandió hacia muchas máquinas, y se popularizó moderadamente en las minicomputadores como la serie DEC PDP y la Data General Nova. En estos casos, el lenguaje era implementado como un intérprete, en vez de un compilador, o alternativamente, de ambas formas.

(2) Crecimiento explosivo

Sin embargo, fue con la introducción de la Microcomputador Altair 8800 en 1975 que BASIC se diseminó ampliamente. La mayoría de lenguajes de programación eran

demasiado grandes para caber en la pequeña memoria que la mayoría de usuarios podía pagar para sus máquinas, y con el lento almacenamiento que era la cinta de papel, y más tarde la cinta de audiocasete (los discos magnéticos aún no existían), y la falta de editores de texto adecuados, un lenguaje pequeño como BASIC era una buena opción. Uno de los primeros en aparecer fue Tiny BASIC, una implementación simple de BASIC escrita originalmente por el Dr. Li-Chen Wang, y portada más tarde a la Altair por Dennis Allison, a petición de Bob Albrecht (quien después fundó el *Dr. Dobbs Journal*). El diseño de Tiny BASIC y el código fuente completo fue publicado en 1976 en DDJ.

En 1975 Microsoft (entonces constaba de dos personas: Gates y Allen) lanzó Altair BASIC. Luego comenzaron a aparecer bajo licencia versiones para otras plataformas, y millones de copias y variantes pronto estarían en uso. Se convirtió en uno de los lenguajes estándar en la Apple II. Para 1979 Microsoft estaba negociando con varios vendedores de microcomputadores, incluyendo a IBM, para licenciar un intérprete de BASIC para sus computadores. Una versión se incluyó en los chips ROM de las PCs IBM, para PCs sin discos, y en las que disponían de unidad de diskettes el BASIC era iniciado automáticamente si es que no se colocaba ningún diskette de arranque de sistema operativo.

Mientras que las nuevas compañías intentaban seguir los pasos del éxito de Altair, IMSAI, North Star, y Apple, creando la revolución de la computadora casera. BASIC se convirtió en una característica estándar para casi todas las computadores caseras; la mayoría venía con un intérprete de BASIC en ROM (algo hecho por primera vez por la Commodore PET en 1977). Pronto había muchos millones de computadores corriendo BASIC alrededor del mundo, un número mucho más grande que el de todos los usuarios de otros lenguajes juntos. Muchos programas, especialmente los de la Apple II e IBM PC, dependían de la presencia del intérprete de BASIC de Microsoft y no correrían sin éste; por lo que Microsoft usó la licencia de copyright en los intérpretes de BASIC para influir en las negociaciones con los vendedores de computadores.

El BASIC fue también el lenguaje prefijado en los computadores caseros europeos de la década de los 80 como el ZX Spectrum, MSX o el Commodore 64, haciendo muchas veces la función de intérprete y sistema operativo primitivo ya que venían implementados en ROM.

(3) Madurez

En este período se crearon versiones de BASIC nuevas y más poderosas. Microsoft vendió varias versiones de BASIC para MS-DOS/PC-DOS, incluyendo BASICA, GW-BASIC (una versión compatible con BASICA que no necesitaba la ROM de IBM), y Quick BASIC. El fabricante de Turbo Pascal, Borland, publicó Turbo BASIC 1.0 en 1985 (versiones sucesoras aún se venden bajo el nombre de PowerBASIC por otra compañía). Aparecieron varias extensiones de BASIC para computadores caseras, típicamente con gráficos, sonido, y comandos DOS, así como facilidades para Programación estructurada. Otros lenguajes usan la sintaxis de BASIC como base para otros sistemas totalmente diferentes, como por ejemplo GRASS.

Sin embargo a finales de la década de 1980 las computadores nuevas eran mucho más complejas, e incluían características (como la Interfaz gráfica de usuario) que hacían a

BASIC menos apropiado para programarlas. Al mismo tiempo las computadores progresaban de ser un interés para aficionados a herramientas usadas principalmente para ejecutar aplicaciones escritas por otros, y la programación en sí se fue haciendo menos importante para la creciente mayoría de usuarios. BASIC comenzó a desvanecerse, aunque numerosas versiones aún estaban disponibles.

La suerte de BASIC dio un giro nuevamente con la introducción de Visual Basic de Microsoft. Aunque es algo difícil considerar este lenguaje como BASIC (a pesar de que usa muchas palabras clave conocidas de BASIC) se ha convertido en uno de los lenguajes más usados en la plataforma Windows, y se dice que representa del 70 al 80% del desarrollo comercial de aplicaciones. Visual Basic for Applications (VBA) fue añadido a Microsoft Excel 5.0 en 1993 y al resto de la línea de productos de Microsoft Office en 1997. Windows 98 incluyó un intérprete de VBScript. La versión más reciente de Visual Basic es llamada VB.NET. La suite OpenOffice.org incluye una variante de BASIC menos poderosa que su contraparte de Microsoft.

(4) El Lenguaje

Sintaxis

Una sintaxis muy mínima de BASIC sólo necesita los comandos LET, PRINT, IF y GOTO. Un intérprete que ejecuta programas con esta sintaxis mínima no necesita una pila. Algunas de las primeras implementaciones eran así de simples. Si se le agrega una pila, se pueden agregar también ciclos FOR anidados y el comando GOSUB. Un intérprete de BASIC con estas características necesita que el código tenga números de línea.

Los dialectos modernos de BASIC MIUN han abandonado los números de línea, y la mayoría (o todos) han añadido control de flujo estructurado y los constructores de declaración de datos como los de otros lenguajes como C y Pascal:

- do
- loop
- while
- until
- exit
- on ... goto
- gosub
- switch
- case

Variantes recientes como Visual Basic han introducido características orientadas a objetos, y hasta herencia en la última versión. La administración de memoria es más fácil que con muchos otros lenguajes de programación procedimentales debido al Recolector de basura (a costas de la velocidad de ejecución).

(5) Procedimientos y Control de Flujo

BASIC no tiene una biblioteca externa estándar como otros lenguajes como C. En cambio, el intérprete (o compilador) contiene una biblioteca incorporada de procedimientos intrínsecos. Estos procedimientos incluyen la mayoría de herramientas que un programador necesita para aprender a programar y escribir aplicaciones sencillas, así como funciones para realizar cálculos matemáticos, manejar cadenas, entrada de la consola, gráficos y manipulación de archivos.

Algunos dialectos de BASIC no permiten que los programadores escriban sus propios procedimientos. Los programadores en cambio deben escribir sus programas con un gran número de enunciados GOTO para hacer las ramificaciones del programa. Esto puede producir un código fuente muy confuso, comúnmente referido como *Código espagueti*. Los enunciados GOSUB ramifican el programa a una especie de subrutinas sin parámetros o variables locales. La mayoría de versiones de BASIC modernas, como Microsoft QuickBASIC han añadido soporte completo para subrutinas, funciones y programación estructurada. Este es otro área donde BASIC difiere de otros muchos lenguajes de programación. BASIC, como Pascal, hace una distinción entre un procedimiento que no devuelve un valor (llamado subrutina) y un procedimiento que lo hace (llamado función). Muchos otros lenguajes (como C) no hacen esa distinción y consideran todo como una función (algunas que devuelven un valor "*void*" [vacío]).

Mientras que las funciones que devuelven un valor son una adición relativamente reciente a los dialectos de BASIC, muchos de los primeros sistemas soportaban la definición de funciones matemáticas en línea, con DEF FN ("*DEFine FunctioN*" [DEFinir FunciÓN]). El Dartmouth BASIC original también soportaba funciones al estilo de Algol, así como subrutinas desde sus primeros tiempos.

(6) Tipos de Datos

BASIC es reconocido por tener muy buenas funciones para manipular cadenas. Los primeros dialectos ya tenían un juego de funciones fundamentales (LEFT$, MID$, RIGHT$) para manipular cadenas fácilmente. Como las cadenas son utilizadas en aplicaciones diarias, esta era una ventaja considerable sobre otros lenguajes al momento de su introducción.

El Dartmouth BASIC original soportaba únicamente datos de tipo numérico y cadenas. No había un tipo entero. Todas las variables numéricas eran de punto flotante. Las cadenas eran de tamaño dinámico. Los arreglos de ambos, números y cadenas, eran soportados, así como matrices (arreglos de dos dimensiones).

Cada dialecto moderno de BASIC posee al menos los tipos de datos entero y cadena. Los tipos de datos son generalmente distinguidos por un posfijo, los identificadores de cadenas terminan con $ (signo de dólar), mientras que los enteros no lo hacen. En algunos dialectos, las variables deben ser declaradas (con DIM) antes de usarse; otros dialectos no lo requieren, pero tienen la opción para hacerlo (típicamente usando una directiva como OPTION EXPLICIT). Muchos dialectos también soportan tipos adicionales, como enteros de 16 y 32 bits y números de punto flotante. Adicionalmente

algunos permiten la utilización de tipos definidos por el usuario, similar a los "records" de Pascal, o las "structs" de C.

La mayoría de dialectos de BASIC además de los arreglos de tipos primitivos, también soportan arreglos otros tipos. En algunos, los arreglos deben ser declarados antes de que puedan usarse (con el enunciado DIM). Es común también el soporte para arreglos de dos y más dimensiones.

```
DIM miArregloDeEnteros(100) AS INTEGER
DIM miListaDeNombres(50) AS STRING
```

Dependiendo del dialecto de BASIC y del uso del enunciado OPTION BASE, los valores pueden estar en el rango miArregloDeEnteros(0) a miArregloDeEnteros(100), de miArregloDeEnteros(1) a miArregloDeEnteros(100), o de miArregloDeEnteros(LimiteInferior) a miArregloDeEnteros(LimiteSuperior).

(7) Disponibilidad y variantes del dialecto

BASIC está disponible para casi todas las plataformas de microprocesador existentes. Una implementación gratuita que cumple con estándares y es multiplataforma es Bywater BASIC (bwBASIC). El intérprete está escrito en C y viene bajo la licencia GNU. Está diseñado para programas de consola de texto, así que no incluye soporte para crear interfaces gráficas de usuario (GUIs). Un BASIC gratuito que incluye soporte para GUI, es similar a Visual Basic y se ejecuta en Windows y Linux es Phoenix Object Basic.

Las versiones de compiladores más conocidos son la línea de productos de Quick BASIC de Microsoft, y QBASIC (una versión que no genera programas ejecutables). Algunas versiones de Visual Basic también son compiladores, aunque Microsoft ha mantenido a Visual Basic al menos mínimamente compatible con incluso las primeras versiones de sus BASICs.

Otras versiones incluyen PowerBASIC de PowerBASIC, así como True BASIC de True BASIC, que cumple con los últimos estándares oficiales de BASIC. (True BASIC Inc. fue fundada por los creadores originales de Dartmouth BASIC.)

REALbasic es una variante disponible para Mac OS Classic, Mac OS X, Microsoft Windows y Linux. Una variante de un dialecto simple de BASIC para la parrot virtual machine muestra como se implementa un intérprete de BASIC en un lenguaje parecido al ensamblador. PureBasic es una variante con una sintaxis sencilla, pero que produce ejecutables rápidos y pequeños para Windows y Linux. También puede compilar instrucciones en ensamblador incrustadas entre el código. SmallBASIC es un dialecto que corre en muchas plataformas (Win32, DOS, Linux y PalmOS) y viene bajo la licencia GNU (GPL).

Ejemplo 1: Saludo

```
PRINT "Hola"
```

Ejemplo 2: BASIC original no estructurado (Applesoft BASIC)

```
 10 INPUT "¿Cuál es su nombre? "; U$
 20 PRINT "Hola "; U$
 30 INPUT "¿Cuántos asteriscos desea? "; N
 35 S$ = ""
 40 FOR I = 1 TO N
 50 S$ = S$ + "*"
 55 NEXT I
 60 PRINT S$
 70 INPUT "¿Desea más asteriscos? "; A$
 80 IF LEN(A$) = 0 GOTO 70
 90 A$ = LEFT(A$, 1)
100 IF (A$ = "S") OR (A$ = "s") THEN GOTO 30
110 PRINT "Adiós ";
120 FOR I = 1 TO 200
130 PRINT U$; " ";
140 NEXT I
150 PRINT
```

Ejemplo 3: BASIC estructurado moderno

```
INPUT "¿Cuál es su nombre?"; NombreUsuario$
PRINT "Hola "; NombreUsuario$
DO
   INPUT "¿Cuántos asteriscos desea?"; NoAsteriscos
   Asteriscos$ = ""
   Asteriscos$ = REPEAT$("*", NoAsteriscos)
   PRINT Asteriscos$
   DO
      INPUT "¿Desea más asteriscos?"; Respuesta$
   LOOP UNTIL Respuesta$ <> ""
LOOP WHILE UCASE$(LEFT$(Respuesta$, 1)) = "S"
PRINT "Adiós";
FOR I = 1 TO 200
   PRINT NombreUsuario$; " ";
NEXT I
PRINT
```

IV.- PASCAL

Es un lenguaje de programación desarrollado por el profesor suizo Niklaus Wirth a finales de los años 60. Su objetivo era crear un lenguaje que facilitara el aprendizaje de la programación a sus alumnos. Sin embargo con el tiempo su utilización excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo.

Pascal se caracteriza por ser un lenguaje de programación estructurado fuertemente tipificado. Esto implica que:

1. El código esta dividido en porciones fácilmente legibles llamadas *funciones o procedimientos*. De esta forma *Pascal* facilita la utilización de la *programación estructurada* en oposición al antiguo estilo de *programación monolítica*.
2. El *tipo de dato* de todas las variables debe ser declarado previamente para que su uso quede habilitado.

El nombre de Pascal fue escogido en honor al matemático Blaise Pascal.

(1) Características únicas

A diferencia de lenguajes de programación descendientes de C, Pascal utiliza el símbolo := para la asignación en vez de =. Si bien el segundo es más conciso, la práctica ha demostrado que muchos usuarios utilizan el símbolo de igualdad para comparar valores en lugar del comparador de C que es el símbolo ==. Esta sintaxis conduce a muchos errores o *bugs* difíciles de rastrear en código C. Dado que Pascal no permite dentro de expresiones y utiliza sintaxis distinta para asignaciones y comparaciones, no sufre estos errores.

Otra diferencia importante es que en Pascal, el tipo de una variable se fija en su definición; la asignación a variables de valores de tipo incompatible no están autorizadas (En C, en cambio, el compilador hace el mejor esfuerzo para dar una interpretación a casi todo tipo de asignaciones). Esto previene errores comunes donde variables son usadas incorrectamente porque el tipo es desconocido. Esto también evita la necesidad de notación húngara, esto es prefijos que se añaden a los nombres de las variables y que indican su tipo.

(2) Implementaciones

Las primeras versiones del compilador de Pascal, entre ellas la más distribuida fue UCSD Pascal, traducían el lenguaje en código para una máquina virtual llamada máquina-P. La gran ventaja de este enfoque es que para tener un compilador de Pascal en una nueva arquitectura de máquina solo hacía falta reimplementar la máquina-P. Como consecuencia de esto, solo una pequeña parte del interprete tenía que ser reescrita hacia muchas arquitecturas.

En los años 1980, Anders Hejlsberg escribió el compilador Blue Label Pascal para la Nascom-2. Más tarde fue a trabajar para Borland y reescribió su compilador que se convirtió en Turbo Pascal para la IBM PC. Este nuevo compilador se vendió por $49, un precio orientado a la distribución masiva.

El económico compilador de Borland tuvo una larga influencia en la comunidad de Pascal que comenzó a utilizar principalmente en el IBM PC. En busca de un lenguaje estructurado muchos aficionados al PC reemplazaron el BASIC por este producto. Dado que Turbo Pascal, solo estaba disponible para una arquitectura, traducía directamente hacia el código máquina del Intel 8088, logrando construir programas que se ejecutaban mucho más rápidamente que los producidos en los esquemas interpretados.

Durante los años 1990, estuvo disponible la tecnología para construir compiladores que pudieran producir código para diferentes arquitecturas de hardware. Esto permitió que los compiladores de Pascal tradujeran directamente al código de la arquitectura en que corrieran.

Con Turbo Pascal versión 5, Borland, agregó programación orientada a objetos a Pascal.

Sin embargo, Borland después decidió mejorar esa extensión del lenguaje introduciendo su producto Delphi, diseñado a partir de *estándar Object Pascal* propuesto por Apple como base. Borland también lo llamó Object Pascal en las primeras versiones, pero cambio el nombre a lenguaje de programación Delphi en sus últimas versiones.

(3) Ejemplo de código

```
program raiz(input, output);
(*
  Obtener la raiz cuadrada de un numero real x.
  Se supone que x no es negativo.
*)

var x, y: real;

begin
  writeln('** Calcular la raiz cuadrada de x **');
  writeln;
  write('Entrar x (> 0): '); readln(x);
  y := sqrt(x);
  writeln;
  writeln('La raiz cuadrada de ', x, ' es ', y);
  writeln; writeln('** Fin **')
  readln; (* Espera a que el usuario pulse enter para salir del
programa *)
end.
```

(4) Compiladores disponibles públicamente

Varios compiladores de Pascal están disponibles para el uso del público en general:

- Compilador GNU Pascal (GPC), escrito en C, basado en GNU Compiler Collection (GCC). Se distribuye bajo licencia GPL.
- Free Pascal está escrito en Pascal (el compilador está creado usando FreePascal), es un compilador estable y potente. También distribuido libremente bajo la licencia GPL. Este sistema puede mezclar código Turbo Pascal con código Delphi, y soporta muchas plataformas y sistemas operativos.
- Turbo Pascal fue el compilador Pascal dominante para PCs durante los años 1980 y hasta principios de los años 1990, muy popular debido a sus magníficas extensiones y tiempos de compilación sumamente cortos. Actualmente, versiones viejas de Turbo Pascal (hasta la 5.5) están disponibles para descargarlo gratuito desde el sitio de Borland (es necesario registrarse)
- Delphi es un producto tipo RAD (Rapid Application Development) de Borland. Utiliza el lenguaje de programación Delphi, descendiente de Pascal, para crear aplicaciones para la plataforma Windows. Las últimas versiones soportan compilación en la plataforma .NET.
- Kylix es la versión más nueva de Borland reiterando la rama de Pascal de sus productos. Es descendiente de Delphi, con soporte para el sistema operativo Linux y una librería de objetos mejorada (CLX). El compilador y el IDE están

disponibles para uso no comercial. Actualmente este proyecto está descontinuado.

V.- JAVA

Es un lenguaje de programación orientado a objetos desarrollado por James Gosling y sus compañeros de Sun Microsystems al inicio de la década de 1990. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es ejecutado (usando normalmente un compilador JIT), por una máquina virtual Java.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos mucho más simple y elimina herramientas de bajo nivel como punteros.

Java está sólo lejanamente emparentado con JavaScript, aunque tengan nombres similares y compartan una sintaxis al estilo de C algo parecida.

(1) Historia

La plataforma Java y el lenguaje Java empezaron como un proyecto interno de Sun Microsystems en diciembre de 1990. Patrick Naughton, ingeniero de Sun, estaba decepcionado con el estado de C++ y la API de C y sus herramientas. Mientras consideraba migrar a NeXT, Naughton recibió la oferta de trabajar en una nueva tecnología, y así comenzó el proyecto *Stealth*.

El Proyecto Stealth fue rebautizado, es decir, vuelto a bautizar, como *Green Project* (o *Proyecto Verde*) cuando James Gosling y Mike Sheridan se unieron a Naughton. Con la ayuda de otros ingenieros, empezaron a trabajar en una pequeña oficina en Sand Hill Road en Menlo Park, California. Intentaban desarrollar una nueva tecnología para programar la siguiente generación de *dispositivos inteligentes*, en los que Sun veía un campo nuevo a explotar.

El equipo pensó al principio usar C++, pero se descartó por varias razones. Al estar desarrollando un sistema empotrado con recursos limitados, C++ no es adecuado por necesitar mayor potencia además de que su complejidad conduce a errores de desarrollo. La ausencia de un *recolector de basura* (*garbage collector*) obligaba a los programadores a manejar manualmente el sistema de memoria, una tarea peligrosa y proclive a fallos. El equipo también se encontró con problemas por la falta de herramientas portables en cuanto a seguridad, programación distribuida, y programación concurrente. Finalmente abogaban por una plataforma que fuese fácilmente portable a todo tipo de dispositivo.

Bill Joy había concebido un nuevo lenguaje que combinase lo mejor de *Mesa* y C. En un escrito titulado *Further* (más lejos), proponía a Sun que sus ingenieros crearan un entorno *orientado a objetos* basado en C++. Al principio Gosling intentó modificar y ampliar C++, a lo que llamó C++ ++ --, pero pronto descartó la idea para crear un

lenguaje completamente nuevo, al que llamó *Oak*, en referencia al roble que tenía junto a su oficina.

El equipo dedicó largas horas de trabajo y en el verano de 1992 tuvieron lista algunas partes de la plataforma, incluyendo el Sistema Operativo Green, el lenguaje Oak, las librerías y el hardware. La primera prueba, llevada a cabo el 3 de Septiembre de 1992, se centró en construir una PDA (*Personal Digital Assistant* o Asistente Digital Personal) llamada *Star7*[1], que contaba con una interfaz gráfica y un asistente apodado "Duke" para guiar al usuario.

En noviembre de ese mismo año, el Proyecto Verde se convirtió en **FirstPerson, Inc**, una división propiedad de Sun Microsystems, y el equipo se trasladó a Palo Alto (California). El interés se centró entonces en construir dispositivos interactivos, hasta que Time Warner publicó una solicitud de oferta para un adaptador de televisión. Es decir, un aparato que se sitúa entre la televisión y una fuente de señal externa y que adapta el contenido de ésta (video, audio, páginas Web, etc.) para verse en la pantalla. Entonces, Firsperson cambió de idea y envió a Warner una propuesta para el dispositivo que deseaban. Sin embargo, la industria del cable consideró que esa propuesta daba demasiado control al usuario, con lo que FirstPerson perdió la puja a favor de Silicon Graphics Incorporated. Un trato con la empresa 3DO para el mismo tipo de dispositivo tampoco llegó a buen puerto. Viendo que no había muchas posibilidades en la industria de la televisión, la compañía volvió al seno de Sun.

(2) Java e Internet

En junio y julio de 1994, tras una sesión maratónica de tres días entre Jonh Gaga, James Gosling, Joy Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador Web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

Ese año renombraron el lenguaje como **Java** tras descubrir que "Oak" era ya una marca comercial registrada para adaptadores de tarjetas gráficas. El término Java fue acuñado en una cafetería frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: **J**ames Gosling, **A**rthur **V**an Hoff, y **A**ndy Bechtolsheim. Otros abogan por el siguiente acrónimo, **J**ust **A**nother **V**ague **A**cronym ("sólo otro acrónimo ambiguo"). La hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes (el *número mágico*) de los archivos .class que genera el compilador, son en hexadecimal, 0xCAFEBABE.

En octubre de 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen,

Vicepresidente Ejecutivo de Netscape, que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. [1] Dos semanas más tarde la primera versión de Java fue publicada.

(3) En la Web

En el cliente

La capacidad de los navegadores Web para ejecutar applets de Java ha asegurado la continuidad del uso de Java por el gran público. Flash está más extendido para animaciones interactivas y los desarrolladores estén empezando a usar la tecnología AJAX también en este campo. Java suele usarse para aplicaciones más complejas como la zona de juegos de Yahoo, Yahoo! Games, o reproductores de video (ej. [2]).

En el servidor

En la parte del servidor, Java es más popular que nunca, con muchos sitios empleando páginas JavaServer, conectores como Tomcat para Apache y otras tecnologías Java.

En el PC de escritorio

Aunque cada vez la tecnología Java se acerca más y más al PC de sobremesa, las aplicaciones Java han sido relativamente raras para uso doméstico, por varias razones.[3]

- Las aplicaciones Java pueden necesitar gran cantidad de memoria física.
- La Interfaz Gráfica de Usuario (GUI) no sigue de forma estricta la *Guía para la Interfaz Humana'* (Human Interface Guidelines), así como tampoco aquella a la que estamos habitualmente acostumbrados. La apariencia de las fuentes no tiene las opciones de optimización activadas por defecto, lo que hace aparecer al texto como si fuera de baja calidad.
- Las herramientas con que cuenta el JDK no son suficientemente potentes para construir de forma simple aplicaciones potentes. Aunque el uso de herramientas como Eclipse, un IDE con licencia GNU de alta calidad, facilita enormemente las tareas de desarrollo.
- Hay varias versiones del Entorno en Tiempo de Ejecución de Java, el JRE. Es necesario tener instalada la versión adecuada. El paquete JRE puede ser de tamaño considerable, 7Mbytes, lo que puede ser un inconveniente a la hora de descargarlo e instalarlo.
- Las aplicaciones basadas en la Web están tomando la delantera frente a aquellas que funcionan como entidades independientes. Las nuevas técnicas de programación producen aplicaciones basadas en un modelo en red cada vez más potentes.

Sin embargo hay aplicaciones Java cuyo uso está ampliamente extendido, como los NetBeans, el entorno de desarrollo (IDE) Eclipse, y otros programas como LimeWire y Azureus para intercambio de archivos. Java también es el motor que usa MATLAB para el renderizado de la interfaz gráfica y para parte del motor de cálculo. Las aplicaciones

de escritorio basadas en la tecnología Swing y SWT (Standard Widget Toolkit) suponen una alternativa a la plataforma .Net de Microsoft.

(4) Disponibilidad del JRE de Java

Una versión del JRE (Java Runtime Environment) está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en su sistema operativo, ya que Windows XP fue lanzado en 2001. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría de las distribuciones de Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de applets de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

(5) Historial de versiones

El lenguaje Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la librería estándar. Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa *Java Specification Requests* (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la *Java Language Specification* (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

- **JDK 1.0** (23 de enero de 1996) — Primer lanzamiento.

comunicado de prensa

- **JDK 1.1** (19 de febrero de 1997) — Principales adiciones incluidas: comunicado de prensa
    - una reestructuración intensiva del modelo de eventos AWT (Abstract Windowing Toolkit)
    - clases internas (inner classes)
    - JavaBeans
    - JDBC (Java Database Connectivity), para la integración de bases de datos
    - RMI (Remote Method Invocation)

- **J2SE 1.2** (8 de diciembre de 1998) — Nombre clave *Playground*. Esta y las siguientes versiones fueron recogidas bajo la denominación **Java 2** y el nombre "J2SE" (Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Otras mejoras añadidas incluían: comunicado de prensa
    - la palabra reservada (keyword) `strictfp`
    - reflexión o reflection
    - la API gráfica (Swing) fue integrada en las clases básicas

    - la máquina virtual (JVM) de Sun fue equipada con un compilador JIT (Just in Time) por primera vez
    - Java Plug-in
    - Java IDL, una implementación de IDL (Interfaz para Descripción de Lenguaje) para la interoperabilidad con CORBA
    - Colecciones (Collections)

- **J2SE 1.3** (8 de mayo de 2000) — Nombre clave *Kestrel*.

Los cambios más notables fueron:comunicado de prensa lista completa de cambios

    -
    - la inclusión de la máquina virtual de HotSpot JVM (la JVM de HotSpot fue lanzada inicialmente en abril de 1999, para la JVM de J2SE 1.2)
    - RMI fue cambiado para que se basara en CORBA
    - JavaSound
    - se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de librerías principales (anteriormente disponible como una extensión)
    - Java Platform Debugger Architecture (JPDA)

- **J2SE 1.4** (6 de febrero de 2002) — Nombre Clave *Merlin*. Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59. Los cambios más notables fueron: press releasefull list of changes
    - Palabra reservada `assert` (Especificado en JSR 41.)
    - Expresiones regulares modeladas al estilo de las expresiones regulares Perl
    - Encadenación de excepciones Permite a una excepción encapsular la excepción de bajo nivel original.
    - non-blocking NIO (New Input/Output) (Especificado en JSR 51.)
    - Logging API (Specified in JSR 47.)
    - API I/O para la lectura y escritura de imágenes en formatos como JPEG o PNG
    - Parser XML integrado y procesador XSLT (JAXP) (Especificado en JSR 5 y JSR 63.)
    - Seguridad integrada y extensiones criptográficas (JCE, JSSE, JAAS)
    - Java Web Start incluido (El primer lanzamiento ocurrió en Marzo de 2001 para J2SE 1.3) (Especificado en JSR 56.)

- **J2SE 5.0** (30 de septiembre de 2004) — Nombre clave: *Tiger*. (Originalmente numerado 1.5, esta notación aún es usada internamente.[4]) Desarrollado bajo JSR 176, Tiger añadió un número significativo de nuevas características press release
    - Plantillas (genéricos) — provee conversion de tipos (type safety) en tiempo de compilación para colecciones y elimina la necesidad de la mayoría de conversion de tipos (type casting). (Especificado por JSR 14.)
    - Metadatos — también llamados anotaciones, permite a estructuras del lenguaje como las clases o los métodos, ser etiquetados con datos

adicionales, que puedan ser procesados posteriormente por utilidades de proceso de metadatos. (Especificado por JSR 175.)
    - Autoboxing/unboxing — Conversiones automáticas entre tipos primitivos (Como los `int`) y clases de envoltura primitivas (Como Plantilla:Javadoc:SE). (Especificado por JSR 201.)
    - Enumeraciones — la palabra reservada `enum` crea una typesafe, lista ordenada de valores (como `Dia.LUNES`, `Dia.MARTES`, etc.). Anteriormente, esto solo podía ser llevado a cabo por constantes enteras o clases construidas manualmente (enum pattern). (Especificado por JSR 201.)
    - Varargs (número de argumentos variable) — El último parámetro de un método puede ser declarado con el nombre del tipo seguido por tres puntos (e.g. `void drawtext(String... lines)`). En la llamada al método, puede usarse cualquier número de parámetros de ese tipo, que serán almacenados en un array para pasarlos al metodo.
    - Enhanced `for` bucle — La sintaxis para el bucle `for` se ha extendido con una sintaxis especial para iterar sobre cada miembro de un array o sobre cualquier clase que implemente Plantilla:Javadoc:SE, como la clase estándar Plantilla:Javadoc:SE, de la siguiente forma:

```
void displayWidgets (Iterable<Widget> widgets) {
    for (Widget w : widgets) {
        w.display();
    }
}
```

Este ejemplo itera sobre el objeto `Iterable widgets`, asignando, en orden, cada uno de los elementos a la variable `w`, y llamando al método `display()` de cada uno de ellos. (Especificado por JSR 201.)

- **Java SE 6** — Nombre clave *Mustang*. Estando en 2006 está en desarrollo bajo la JSR 270. Una versión Beta fue lanzada el 15 de febrero de 2006 nota de prensa y está disponible en http://java.sun.com/javase/6/. Se espera otra beta en verano de 200A con la versión final en otoño de 200n. Las nuevas versiones, que incorporan mejoras y errores resueltos, son lanzadas aproximadamente cada semana. En esta versión, Sun cambió el nombre "J2SE" por **Java SE** y eliminó el ".0" del número de versión.[5]

(6) Filosofía

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, los programadores de Java a veces recurren a extensiones como CORBA (Common Object Request Broker Architecture), Internet Communications Engine o OSGi respectivamente.

(7) Orientado a Objetos

La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que use estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y librerías de objetos.

(8) Independencia de la plataforma

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Es lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, '''write once, run everywhere'''.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode)—instrucciones máquina

simplificadas específicas de la plataforma Java. Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (VM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran librerías adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean “compatibles”. Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ‘’dependientes’’ de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaba una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o ‘’’compilación al vuelo’’’), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una ‘’’recompilación dinámica’’’ en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran

número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o “Escríbelo una vez, ejecútalo en todas partes” por “Escríbelo una vez, depúralo en todas partes”)

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

(9) El recolector de basura

Un argumento en contra de lenguajes como C++ es que los programadores se encuentran con la carga añadida de tener que administrar la memoria de forma manual. En C++, el desarrollador debe asignar memoria en una zona conocida como *heap* (montículo) para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente solicitada, o si no lo hace en el instante oportuno, puede llevar a una *fuga de memoria*, ya que el sistema operativo piensa que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual *cuelgue*.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura (o *automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios—es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y frecuentemente más rápida que en C++.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

(10) Críticas

Harold dijo en 1995 que Java fue creado para abrir una nueva vía en la gestión de software complejo, y es por regla general aceptado que se ha comportado bien en ese aspecto. Sin embargo no puede decirse que Java no tenga grietas, ni que se adapta completamente a todos los estilos de programación, todos los entornos, o todas las necesidades.

(11) General

- Java no ha aportado capacidades estándares para aritmética en punto flotante. El estándar IEEE 754 para "Estándar para Aritmética Binaria en Punto Flotante" apareció en 1985, y desde entonces es el estándar para la industria. Y aunque la aritmética flotante de Java *(cosa que cambió desde el 13 de Noviembre de 2006, cuando se abrió el código fuente y se adoptó la licencia GPL, aparte de la ya existente)* se basa en gran medida en la norma del IEEE, no soporta aún algunas características. Más información al respecto puede encontrarse en la sección final de enlaces externos.
- A raíz de la naturaleza propietaria de Java, la supuesta inflexibilidad para cambiar, y un creciente estrechamiento de líneas alrededor del sector corporativo, se dice que Java es "el nuevo COBOL". Aunque puede ser una afirmación exagerada, hace referencia a una preocupación real sobre el panorama de futuro de Java.
- El recolector de basura de Java sólo gestiona la memoria, pero el instante en que tiene lugar su tarea no puede controlarse manualmente. Por tanto, aquellos objetos que reservan recursos externos deben ser desalojados de memoria a mano (usando el mecanismo del lenguaje "finally"), o deben mantenerse hasta que acabe la ejecución del programa.

(12) El lenguaje

- En un sentido estricto, Java no es un lenguaje absolutamente orientado a objetos, a diferencia de, por ejemplo, Ruby o Smalltalk. Por motivos de eficiencia, Java ha relajado en cierta medida el paradigma de orientación a objetos, y así por ejemplo, no todos los valores son objetos.
- Por el contrario, los programadores de C++ pueden caer en la confusión con Java, porque en éste los tipos primitivos son siempre variables automáticas, y los objetos siempre residen en el montículo (heap), mientras que en C++ ambos casos están en manos del programador, usando el operador new.
- El código Java puede ser a veces redundante en comparación con otros lenguajes. Esto es en parte debido a las frecuentes declaraciones de tipos y conversiones de tipo manual (casting). También se debe a que no se dispone de operadores sobrecargados, y a una sintaxis relativamente simple. Sin embargo, J2SE 5.0 introduce elementos para tratar de reducir la redundancia, como una nueva construcción para los bucles '''foreach'''.
- A diferencia de C++, Java no dispone de operadores de sobrecarga definidos por el usuario. Sin embargo esta fue una decisión de diseño que puede verse como una ventaja, ya que esta característica puede hacer los programas difíciles de leer y mantener.

- Java es un lenguaje basado en un solo paradigma.

- Java no permite herencia múltiple como otros lenguajes. Sin embargo el mecanismo de los interfaces de Java permite herencia múltiple de tipos y métodos abstractos.
- El soporte de Java para patrones de texto y la manipulación de éste no es tan potente como en lenguajes como Perl, Ruby o PHP, aunque J2SE 1.4 introdujo las expresiones regulares.

(13) Apariencia

La apariencia externa (el ‘’’look and feel’’’) de las aplicaciones GUI (Graphical User Interface) escritas en Java usando la plataforma Swing difiere a menudo de la que muestran aplicaciones nativas. Aunque el programador puede usar el juego de herramientas AWT (Abstract Windowing Toolkit) que genera objetos gráficos de la plataforma nativa, el AWT no es capaz de funciones gráficas avanzadas sin sacrificar la portabilidad entre plataformas; ya que cada una tiene un conjunto de APIs distinto, especialmente para objetos gráficos de alto nivel. Las herramientas de Swing, escritas completamente en Java, evitan este problema construyendo los objetos gráficos a partir de los mecanismos de dibujo básicos que deben estar disponibles en todas las plataformas. El inconveniente es el trabajo extra requerido para conseguir la misma apariencia de la plataforma destino. Aunque esto es posible (usando GTK+ y el Look-and-Feel de Windows), la mayoría de los usuarios no saben cómo cambiar la apariencia que se proporciona por defecto por aquella que se adapta a la de la plataforma. Mención merece la versión optimizada del Java Runtime que ha desarrollado Apple y que incluye en su sistema operativo, el Mac OS X. Por defecto implemente su propio look-and-feel (llamado Aqua), dando a las aplicaciones Swing ejecutadas en un Macintosh una apariencia similar a la que tendría si se hubiese escrito en código nativo.

(14) Rendimiento

El rendimiento de una aplicación está determinado por multitud de factores, por lo que no es fácil hacer una comparación que resulte totalmente objetiva. En tiempo de ejecución, el rendimiento de una aplicación Java depende más de la eficiencia del compilador, o la JVM, que de las propiedades intrínsecas del lenguaje. El bytecode de Java puede ser interpretado en tiempo de ejecución por la máquina virtual, o bien compilado al cargarse el programa, o durante la propia ejecución, para generar código nativo que se ejecuta directamente sobre el hardware. Si es interpretado, será más lento que usando el código máquina intrínseco de la plataforma destino. Si es compilado, durante la carga inicial o la ejecución, la penalización está en el tiempo necesario para llevar a cabo la compilación.

Algunas características del propio lenguaje conllevan una penalización en tiempo, aunque no son únicas de Java. Algunas de ellas son el chequeo de los límites de arrays, chequeo en tiempo de ejecución de tipos, y la indirección de funciones virtuales.

El uso de un recolector de basura para eliminar de forma automática aquellos objetos no requeridos, añade una sobrecarga que puede afectar al rendimiento, o ser apenas apreciable, dependiendo de la tecnología del recolector y de la aplicación en concreto.

Las JVM modernas usan recolectores de basura que gracias a rápidos algoritmos de manejo de memoria, consiguen que algunas aplicaciones puedan ejecutarse más eficientemente.

El rendimiento entre un compilador JIT y los compiladores nativos puede ser parecido, aunque la distinción no está clara en este punto. La compilación mediante el JIT puede consumir un tiempo apreciable, un inconveniente principalmente para aplicaciones de corta duración o con gran cantidad de código. Sin embargo, una vez compilado, el rendimiento del programa puede ser comparable al que consiguen compiladores nativos de la plataforma destino, inclusive en tareas numéricas. Aunque Java no permite la expansión manual de llamadas a métodos, muchos compiladores JIT realizan esta optimización durante la carga de la aplicación y pueden aprovechar información del entorno en tiempo de ejecución para llevar a cabo transformaciones eficientes durante la propia ejecución de la aplicación. Esta recompilación dinámica, como la que proporciona la máquina virtual HotSpot de Sun, puede llegar a mejorar el resultado de compiladores estáticos tradicionales, gracias a los datos que sólo están disponibles durante el tiempo de ejecución.

Java fue diseñado para ofrecer seguridad y portabilidad, y no ofrece acceso directo al hardware de la arquitectura ni al espacio de direcciones. Java no soporta expansión de código ensamblador, aunque las aplicaciones pueden acceder a características de bajo nivel usando librerías nativas (JNI, Java Native Interfaces).

(15) Sintaxis

La sintaxis de Java se deriva en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones), y todo en Java reside en alguna clase.

Aplicaciones autónomas

```
// Hola.java
public class Hola
{
    public static void main(String[] args) {
        System.out.println("Hola, mundo!");
    }
}
```

Este ejemplo necesita una pequeña explicación.

- Todo en Java está dentro de una clase, incluyendo programas autónomos.
- El código fuente se guarda en archivos con el mismo nombre que la clase que contienen y con extensión ".java". Una clase (`class`) declarada pública (`public`) debe seguir este convenio. En el ejemplo anterior, la clase es `Hola`, por lo que el código fuente debe guardarse en el fichero "Hola.java"
- El compilador genera un archivo de clase (con extensión ".class") por cada una de las clases definidas en el archivo fuente. Una clase anónima se trata como si su nombre fuera la concatenación del nombre de la clase que la encierra, el símbolo "$", y un número entero.

- Los programas que se ejecutan de forma independiente y autónoma, deben contener el método `"main()"`.
- La palabra reservada `"void"` indica que el método main no devuelve nada.
- El método main debe aceptar un array de objetos tipo String. Por acuerdo se referencia como `"args"`, aunque puede emplearse cualquier otro identificador.
- La palabra reservada `"static"` indica que el método es un método de clase, asociado con la clase en vez de a instancias de la misma. El método main debe ser estático o '''de clase'''.
- La palabra reservada **`public`** significa que un método puede ser llamado desde otras clases, o que la clase puede ser usada por clases fuera de la jerarquía de la propia clase. Otros tipos de acceso son `"private"` o `"protected"`.
- La utilidad de impresión (en pantalla por ejemplo) forma parte de la librería estándar de Java: la clase '''System''' define un campo público estático llamado '''out'''. El objeto `out` es una instancia de '''PrintStream''', que ofrece el método '''println(String)''' para volcar datos en la pantalla (la salida estándar).
- Las aplicaciones autónomas se ejecutan dando al entorno de ejecución de Java el nombre de la clase cuyo método main debe invocarse. Por ejemplo, una línea de comando (en Unix o Windows) de la forma `java -cp . Hola` ejecutará el programa del ejemplo (previamente compilado y generado "Hola.class") . El nombre de la clase cuyo método main se llama puede especificarse también en el fichero "MANIFEST" del archivo de empaquetamiento de Java (.jar).

(16) Applets

Las applets de Java son programas incrustados en otras aplicaciones, normalmente una página Web que se muestra en un navegador.

```
// Hola.java
import java.applet. Applet;
import java.awt. Graphics;

public class Hola extends Applet {
    public void paint(Graphics gc) {
        gc.drawString("Hola, mundo!", 65, 95);
    }
}
<!-- Hola.html -->
<html>
  <head>
    <title>Applet Hola Mundo</title>
  </head>
  <body>
    <applet code="Hola" width="200" height="200">
    </applet>
  </body>
</html>
```

La sentencia **`import`** indica al compilador de Java que incluya las clases **java.applet. Applet** y **java.awt. Graphics**, para poder referenciarlas por sus nombres, sin tener que anteponer la ruta completa cada vez que se quieran usar en el código fuente.

La clase **`Hola`** extiende (`extends`) a la clase `Applet`, es decir, es una subclase de ésta. La clase `Applet` permite a la aplicación mostrar y controlar el estado del applet. La clase `Applet` es un componente del AWT (Abstract Windowing Toolkit), que permite al applet mostrar una inteterfaz gráfica de usuario o GUI (Graphical User Interface), y responder a eventos generados por el usuario.

La clase `Hola` sobrecarga el método **paint(Graphics)** heredado de la superclase contenedora (`Applet` en este caso), para acceder al código encargado de dibujar. El método `paint()` recibe un objeto **`Graphics`** que contiene el contexto gráfico para dibujar el applet. El método **`paint()`** llama al método drawString(String, int, int) del objeto **`Graphics`** para mostrar la cadena de caracteres **Hola, mundo!** en la posición (65, 96) del espacio de dibujo asignado al applet.

La referencia al applet es colocada en un documento HTML usando la etiqueta **`<applet>`**. Esta etiqueta o tag tiene tres atributos: **`code="Hola"`** indica el nombre del applet, y **`width="200" height="200"`** establece la anchura y altura, respectivamente, del applet. Un applet también pueden alojarse dentro de un documento HTML usando los elementos `object`, o `embed`, aunque el soporte que ofrecen los navegadores Web no es uniforme.[6][7]

(17) Servlets

Los servlets son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.

```
// Hola.java
import java.io.*;
import javax.servlet.*;

public class Hola extends GenericServlet {
    public void service(ServletRequest request, ServletResponse
response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("Hola, mundo!");
        pw.close();
    }
}
```

Las sentencias **`import`** indican al compilador de Java la inclusión de todas las clases públicas e interfaces de los paquetes **java.io** y **javax.servlet** en la compilación.

La clase **`Hola`** extiende (**`extends`**), es heredera de la clase **GenericServlet**. Esta clase proporciona la interfaz para que el servidor le pase las peticiones al servlet y el mecanismo para controlar el ciclo de vida del servlet.

La clase `Hola` sobrecarga el método **service(ServletRequest, ServletResponse)**, definido por la interfaz **servlet** para acceder al manejador de la petición de servicio. El método `service()` recibe un objeto de tipo **ServletRequest** que contiene la petición del cliente y un objeto de tipo **ServletResponse**, usado para generar la respuesta que se

devuelve al cliente. El método `service()` puede *lanzar* (**`throws`**) excepciones de tipo ServletException e IOException si ocurre algún tipo de anomalía.

El método **setContentType(String)** en el objeto respuesta establece el tipo de contenido MIME a "text/html", para indicar al cliente que la respuesta a su petición es una página con formato HTML. El método **getWriter()** del objeto respuesta devuelve un objeto de tipo **PrintWriter**, usado como una *tubería* por la que viajarán los datos al cliente. El método **println(String)** escribe la cadena "Hola, mundo!" en la respuesta y finalmente se llama al método **close()** para cerrar la conexión, que hace que los datos escritos en la **tubería** o stream sean devueltos al cliente.

(18) Aplicaciones con ventanas

Swing es la librería para la interfaz gráfica de usuario avanzada de la plataforma Java SE.

```
// Hola.java
import javax.swing.*;

public class Hola extends JFrame {
    Hola() {
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        add(new JLabel("Hola, mundo!"));
        pack();
    }

    public static void main(String[] args) {
        new Hola().setVisible(true);
    }
}
```

Las sentencias **`import`** indican al compilador de Java que las clases e interfaces del paquete **javax.swing** se incluyan en la compilación.

La clase **`Hola`** extiende (**`extends`**) la clase **javax.swing.JFrame**, que implementa una ventana con una barra de título y un control para cerrarla.

El constructor **`Hola()`** inicializa el marco o frame llamando al método **setDefaultCloseOperation(int)** heredado de JFrame para establecer las operaciones por defecto cuando el control de cierre en la barra de título es seleccionado al valor WindowConstants.DISPOSE_ON_CLOSE. Esto hace que se liberen los recursos tomados por la ventana cuando es cerrada, y no simplemente ocultada, lo que permite a la máquina virtual y al programa acabar su ejecución. A continuación se crea un objeto de tipo JLabel con el texto "Hola, mundo!", y se añade al marco mediante el método **add(Component)**, heredado de la clase **Container**. El método **pack()**, heredado de la clase **Window**, es invocado para dimensionar la ventana y distribuir su contenido.

El método **`main()`** es llamado por la JVM al comienzo del programa. Crea una instancia de la clase **`Hola`** y hace la ventana sea mostrada invocando al método **setVisible(boolean)** de la superclase (clase de la que hereda) con el parámetro a true. Véase que, una vez el marco es dibujado, el programa no termina cuando se sale del método **`main()`**, ya que el código del que depende se encuentra en un hilo de ejecución

independiente ya lanzado, y que permanecerá activo hasta que todas las ventanas hayan sido destruidas.

(19) Recursos

JRE

El **JRE** (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

(20) Componentes

- Librerías de Java, que son el resultado de compilar el código fuente desarrollado por quien implementa la JRE, y que ofrecen apoyo para el desarrollo en Java. Algunos ejemplos de estas librerías son:
  - Las librerías centrales, que incluyen:
    - Una colección de liberías para implementar estructuras de datos como listas, arrays, árboles y conjuntos.
    - Librerías para análisis de XML.
    - Seguridad.
    - Liberías de internacionalización y localización.
  - Librerías de integración, que permiten la comunicación con sistemas externos. Estas librerías incluyen:
    - La API para acceso a bases de datos JDBC (Java DataBase Conectivity).
    - La interfaz JNDI (Java Naming and Directory Interface) para servicios de directorio.
    - RMI (Remote Method Invocation) y CORBA para el desarrollo de aplicaciones distribuidas.
  - Liberías para la interfaz de usuario, que incluyen:
    - El conjunto de herramientas nativas AWT (Abstract Windowing Toolkit), que ofrece componentes GUI (Graphical User Interface), mecanismos para usarlos y manejar sus eventos asociados.
    - Las librerías de Swing, construidas sobre AWT pero ofrecen implementaciones no nativas de los componentes de AWT.
    - APIs para la captura, procesamiento y reproducción de audio.
- Una implementación dependiente de la plataforma en que se ejecuta de la máquina virtual de Java (JVM), que es la encargada de la ejecución del código de las librerías y las aplicaciones externas.
- Plugins o conectores que permiten ejecutar applets en los navegadores Web.
- Java Web Start, para la distribución de aplicaciones Java a través de Internet.

- Documentación y licencia.

(21) APIS

Sun define tres plataformas en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus APIs (Application Program Interface) de forma que pertenezcan a cada una de las plataformas:

- Java ME (Java Platform, Micro Edition) o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.
- Java SE (Java Platform, Standard Edition) o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.
- Java EE (Java Platform, Enterprise Edition) o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Las clases en las APIs de Java se organizan en grupos disjuntos llamados **paquetes**. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información sobre los paquetes que ofrece cada plataforma puede encontrarse en la documentación de ésta.

El conjunto de las APIs es controlado por Sun Microsystems junto con otras entidades o personas a través del programa JCP (Java Community Process). Las compañías o individuos participantes del JCP pueden influir de forma activa en el diseño y desarrollo de las APIs, algo que ha sido motivo de controversia.

En 2004, IBM y BEA apoyaron públicamente la idea de crear una implementación de código abierto (open source) de Java, algo a lo que Sun, a fecha de 2006, se ha negado.

Bibliografía:

Búsqueda en:

www.wikipedia.com (diferentes enlaces):

- phyton
- ada
- basic
- pascal
- java

Índice: