



# **JAVA**

---

## **Características**

Jesús Tomás Gironés  
Universidad Politécnica de Valencia

# Inice de la exposición

- ¿Por que Java?
- Un poco de historia
- Características:
  - ☞ Simple y familiar
  - ☞ Interpretado
  - ☞ Orientado a objetos
  - ☞ Seguro
  - ☞ Robusto
  - ☞ Independiente de la plataforma
  - ☞ Distribuido y dinámico
  - ☞ Multi-thread
- Fases de implantación
- Conclusiones

Comenzaré con una introducción, donde se comentarán las circunstancias que han hecho aparecer al lenguaje Java.

Revisión histórica sobre los orígenes de Java.

Luego haré una descripción de las características más importantes de Java que lo convierten en algo muy diferente al resto de los lenguajes de programación.

Y para terminar destacaremos en las conclusiones los puntos de mayor interés

# El panorama actual de las aplicaciones en red

- Tradicionalmente el desarrollo de aplicaciones ha estado ligado al procesador y al S.O.
- En las empresas suelen existir diferentes tipos de plataformas
- Aplicaciones dependientes de la plataforma
- Se trata de atrapar al cliente en una plataforma concreta

Tradicionalmente el desarrollo de aplicaciones informáticas en general, está estrechamente ligado al sistema operativo donde correrá la aplicación; y al sistema operativo del ordenador.

En las empresas actuales suelen convivir diferentes tipos de plataformas, cada una incompatible con las demás.

¿Como conseguir escribir aplicaciones distribuidas, que corran en todos los sistemas operativos? y de una manera sencilla.

Actualmente es preciso reescribir las aplicaciones para cada sistema operativo, cosa nada sencilla. O por el contrario usar lenguajes interpretados, sumamente ineficientes. Los problemas son tantos que los programas se suelen escribir para una única plataforma.

Esta circunstancia suele ser utilizada por los fabricantes de hardware y software, para atrapar a los clientes en una plataforma determinada.

# La revolución WWW

- Solución parcial a estos problemas
- Causas
  - ☞ Interfaz universal y estandarizado
  - ☞ Independiente de la plataforma
  - ☞ Grandes posibilidades de difusión
- Limitaciones
  - ☞ Solo datos, no programas
  - ☞ Información estática
  - ☞ Cliente poco flexible

La aparición y rápida expansión del WWW ha permitido que se solucione en parte este problema.

Las empresas pueden proporcionar todo tipo de datos, a sus empleados o clientes, sin preocuparse de la plataforma final.

En gran éxito de este planteamiento se debe:

- Todo el mundo conoce el interfaz, (periodo de aprendizaje cero)
- Éste se encuentra disponible para todas las plataformas y ya está instalado en la mayoría de máquinas.
- La red Internet permite una gran difusión de la información.

Sin embargo este sistema presenta graves limitaciones:

- Solo permite acceder a datos y no a programas
- La información es estática (por lo menos por lo que al cliente se refiere, con el interfaz CGI se puede conseguir paliar parcialmente este problema haciendo que el servidor ejecute programas)
- Los clientes se limitan a visualizar información, por lo que están muy limitados en sus capacidades.

NOTA: En la actualidad se han superado muchos de estos problemas

# La revolución Java

- Aprovecha ventajas de WWW
  - ☞ Interfaz universal
  - ☞ Independiente de la plataforma
- Trata de paliar sus problemas
  - ☞ Permite la ejecución de programas (“applets”)
  - ☞ Cliente flexible
  - ☞ Protocolos interoperativos
- Java puede llegar mucho más lejos

El gran éxito de Java se fundamenta en que aprovecha las ventajas del WWW, pero va a paliar algunas de sus limitaciones, permitiendo la ejecución de programas dentro de páginas web, convirtiendo a los clientes en verdaderamente flexibles (dejando de ser meros visualizadores de información), Permite además, que los navegadores puedan adoptar nuevos protocolos de manera dinámica.

Pero Java puede llegar mucho más lejos de simplemente paliar las deficiencias de WWW; como veremos al final de la exposición una vez analizadas todas sus características.

## Un poco de historia

- 1990 Sun está interesada en desarrollar aplicaciones distribuidas con redes heterogéneas
- 1991 Bill Joy crea un lenguaje nuevo el Oak (precursor de Java).
- 1992 Las primeras aplicaciones no tienen éxito
- 1993 Aparece el primer navegador gráfico (Mosaic)
- 1994 Se empieza a experimentar sobre las posibilidades de Oak en Internet
- 1995 Sun anuncia Java y JotJava

Antes de describir con más detalle en que consiste Java, paso a realizar una revisión histórica.

En primer lugar hay que destacar que Java ha sido Sun Microsystems.

1990 Sun está interesada en desarrollar aplicaciones distribuidas con redes heterogéneas, para pequeños dispositivos electrónicos domésticos.

1991 Primero se pensó en utilizar C++, aunque pronto se vio que no era un lenguaje adecuado. Bill Joy crea un lenguaje nuevo el "Oak" (precursor de Java).

1992 Se utiliza en aplicaciones como el control completo de electrodomésticos o vídeo bajo demanda.

1993 Aparece el primer navegador gráfico (Mosaic).

1994 Se empieza a experimentar sobre las posibilidades de Oak para Internet. Se desarrolla un navegador capaz de interpretar el lenguaje (JotJava)

1995 Sun anuncia Java y JotJava. Distribución Libre. Rápida difusión.

1996 Acuerdo con Netscape que garantiza que su navegador permita la ejecución de "applets" (pequeños programas en Java incrustados en páginas Web)

# Características de Java

- Simple y familiar
- Orientado a objetos
- Independiente de la plataforma
- Interpretado
- Seguro
- Robusto
- Distribuido y dinámico
- Multi-thread

A continuación vamos a destacar las características que definen a Java y lo diferencian de otros lenguajes de programación, haciendolo especialmente interesante para el desarrollo de aplicaciones en red.

# Simple y familiar

- Basado en C
- Desarrollado desde cero
- Extremadamente simple
  - ☞ Rápido aprendizaje
  - ☞ Simplificación de la programación
  - ☞ Reducción del número errores

Java está basado en en el lenguaje C, de esta manera resulta muy familiar para los millones de programadores que ya conocen este lenguaje.

Desarrollado desde cero, sin concesiones de compatibilidad, lo que ha permitido la definición de un lenguaje sumamente sencillo y coherente, pero sin perder potencia.

Se han eliminado aquellas características más confusas o menos utilizadas de C (no hay punteros, macros, registros, ni definición de tipos.)

Por lo tanto se consigue un lenguaje:

- rápido aprendizaje
- simplificación del proceso de programación
- reducción del número errores (50%)



# Orientado a objetos

- Orientado a objetos desde la base
- Todo en Java son objetos
- Incorpora las características:
  - ☞ Encapsulación
  - ☞ Herencia
  - ☞ Polimorfismo
  - ☞ Enlace dinámico
- Pero no otras de menor utilidad:
  - ☞ Herencia múltiple
  - ☞ Sobrecarga de operadores

En línea con las modernas prácticas de ingeniería del software, Java está orientado a objetos desde su base. Se ha demostrado en la práctica, que utilizando esta técnica de programación se reduce el tiempo de desarrollo y se obtiene un producto de mejor calidad. Especialmente útil en un entorno distribuido cliente/servidor.

Todo en Java son objetos (con excepción de los tipos elementales)

- Cualquier definición de datos ha de ser un objeto
- Toda función ha de estar dentro de un objeto
- Las librerías de funciones son objetos

Incorpora las características más importantes de este paradigma:

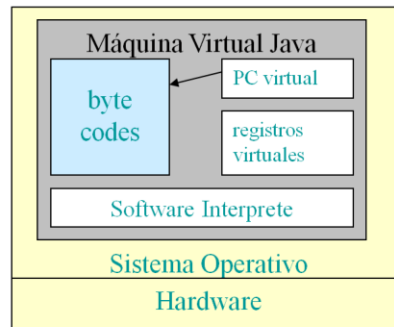
- Encapsulación: La información es ocultada y modularizada
- Herencia: Definición de nuevas clases a partir de las existentes (ej. La clase *automóvil* a partir de *vehículo*)
- Polimorfismo: Un objeto es considerado de la clase a la que pertenece, pero también de las clases de las que desciende. Podemos trabajar con objetos como si fuera de la clase *vehículo*, aunque sea una instancia de de una clase más específicas, como *automóvil* o *camión*.
- Enlace dinámico: Los objetos pueden venir de cualquier sitio, posiblemente a través de la red.

Pero no otras de menor utilidad:

- Herencia múltiple: Una clase definida a partir de más de una clase
- Sobrecarga de operadores: Un operador como el "+" puede ser redefinido para operar dos objetos cualesquiera (suma de matrices)

# Independiente de la plataforma

- Otras propuestas limitadas
- Solución: Arquitectura Neutra
- Máquina Virtual Java
- Compilador Java genera bytecodes



El objetivo principal que se quería conseguir con Java, era que un programa pudiera ser ejecutado en cualquier sistema operativo o procesador. (Independiente de la plataforma)

Existen diversas alternativas para salvar este problema

- grandes binarios que se adaptan a la arquitectura
- lenguajes interpretados o de scripts

Soluciones parciales y además costosas o poco eficientes.

Solución Java: Definir una nueva arquitectura independiente de las ya existentes y por tanto totalmente neutra.

Cuando se compila un programa en Java, no se genera instrucciones de código máquina de ningún procesador concreto, si no que se va a generar unas instrucciones destinadas a una máquina virtual. (instrucciones conocidas como bytecodes)

Cuando queramos ejecutar dicho programa en un procesador concreto, esta máquina virtual tendrá que ser emulada.

A este nivel de abstracción todas las máquinas tienen la misma arquitectura, solucionando todos los problemas de portabilidad.

## Independiente de la plataforma (II)

### ■ Solución problema presentación

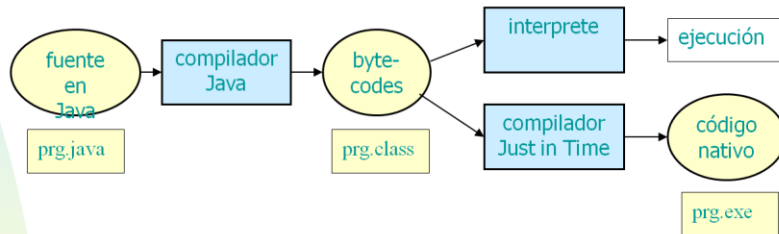
#### tipos de datos en Java:

enteros	byte	8 bits	complem. a 2
	short	16 bits	"
	int	32 bits	
	long	64 bits	
reales	float	32 bits	IEEE 754
	double	64 bits	
caracteres	char	16 bits	Unicode

El hecho de que un programa en Java siempre crea que se está ejecutando en la misma máquina, nos soluciona directamente uno de los problemas habituales, el problema que trataba de solucionar el nivel de "Presentación" (cómo se codifican los datos).

# Interpretado

- Los bytecodes han de ser interpretados



- Ventajas frente a otros intérpretes

- ☞ Código compacto
- ☞ Eficiente
- ☞ Código confidencial

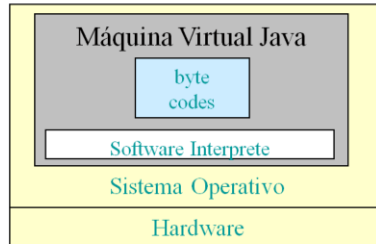
La arquitectura neutra de Java, hace que los programas no puedan ser ejecutados directamente por ningún compilador. Es necesario simular esta plataforma mediante un proceso de simulación

Sin embargo no se sigue el esquema típico de un programa interpretado. Escritura/compilación/interpretación.

Si se quiere mejorar el tiempo de ejecución, también se puede proceder a la compilación de los bytecodes en código nativo de la plataforma (Compilación *Just in Time*).

# Seguro

- Soporta seguridad “sandboxing”



- Verificación bytecodes
- Otras medidas de seguridad:
  - ☞ Cargador de clases
  - ☞ Restricción en el acceso a la red

La seguridad resulta crucial en el desarrollo de aplicaciones distribuidas, por lo que Java extrema las medidas de seguridad.

Se ha de garantizar que cuando se trae un programa de la red, este no pueda destruir datos, contener un virus o curiosear en la máquina local.

- La primera línea de defensa la pone la “M.V.J.”, un programa se ejecuta siempre dentro, por lo que se pueden limitar los accesos a la máquina real. Por ejemplo se puede impedir el acceso al sistema de ficheros local
- Un compilador Java asegura que el código fuente no viole las normas de seguridad. El problema es que no puede asegurarse que todo el mundo utilice compiladores de este tipo. Para solucionar este problema los bytecodes son verificados antes de su ejecución.

Entre otras cosas se comprueba: no existen punteros, no se violan restricciones de acceso, las clases se utilizan correctamente, ...

Otras medidas de seguridad:

- Cuando se carga una nueva clase, nunca puede sustituir a otra ya cargada, o hacer referencia a una clase local o a otra cargada de otro origen. Cuando se importa una clase, se sitúa en un espacio de nombres separado asociado con su origen.
- Java incorpora un paquete de red con la interfaz para manejar varios protocolos de red. Podemos configurar varios niveles de seguridad. Impedir que se use nuestro ordenador como puente.

Puede repercutir negativamente en la eficiencia. “Tenemos las manos atadas”

# Robusto

- La ejecución dentro de la M.V.J. impide bloquear el sistema
- La asignación entre tipos es muy estricta
- La gestión de memoria siempre la realiza el sistema
- Chequeo del código tanto en tiempo de compilación como de ejecución

La ejecución dentro de la M.V.J. impide bloquear el sistema

La asignación entre tipos es muy estricta, no podemos asignar un *int* a un *char* como hacemos en C.

La gestión de memoria la hace siempre el sistema, no el programador, lo que lleva a aplicaciones más fiables y seguras.

El código es chequeado tanto en tiempo de compilación como de ejecución

# Distribuido y dinámico

- Diseñado para una ejecución remota y distribuida
- Sistema dinámico
  - ☞ Clase enlazada cuando es requerida
  - ☞ Pueden ser cargada por red
- Dinámicamente extensible
  - ☞ Diseñado para adaptarse a entornos en evolución

Como se ha comentado los módulos que componen una aplicación se estructuran a base de clases. Estas clases pueden cargarse remotamente por la red.

(Se utiliza un sistema de nombres únicos similar al DNS)

Las clases son cargadas dinámicamente en tiempo de ejecución, cuando son requeridas.

Gracias a Java los nuevos navegadores también incorporan cualidades dinámicas. Los protocolos y formatos de representación pueden manipularse por medio de applets Java, que se van incorporando al navegador a medida que se necesitan.

Si el navegador encuentra un objeto que no es capaz de manejar, pedirá al servidor de dicho objeto, el código Java necesario para su representación.

# Multi-thread

- Solución sencilla y elegante a la multiprogramación
- Un programa puede lanzar varios hilos de ejecución o threads
- No son nuevos procesos, comparten código y variables con el principal
- De forma simultanea se pueden atender varias tareas

Incorpora de manera sencilla y elegante posibilidad de multiprogramación. Un programa puede lanzar varios hilos de ejecución (threads o procesos ligeros). No son nuevos procesos, dado que comparten el código y las variables del programa principal.

Cada thread puede encargarse de una tarea distinta, traer una imagen, realizar una animación, reproducir un sonido, ...



# Conclusiones

## ■ Ventajas:

- ☞ Mucho más que un lenguaje, define un entorno completo para el desarrollo de aplicaciones distribuidas
- ☞ Sencillez
- ☞ Programas eternos y universales

## ■ Inconvenientes:

- ☞ interpretado: ejecución poco eficiente
- ☞ no permite acceso directo al hardware

Podríamos destacar como conclusiones las siguientes características:

Java presenta una serie de ventajas que pueden convertirlo en una alternativa seria:

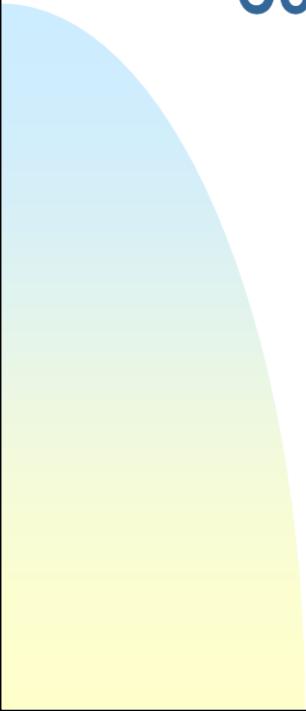
- Java ha sido pensado desde su base, para el desarrollo de aplicaciones distribuidas, por lo que incorpora desde su base soluciones a muchos de los problemas más habituales (seguridad,)
- Pueden ser ejecutadas de forma independiente a la arquitectura
- Basado en la arquitectura Cliente/Servidor reduce a la nada el mantenimiento de los clientes

También podemos destacar los siguientes inconvenientes:

- Interpretado: Ejecución poco eficiente
- Las fuertes medidas de seguridad nos limitan el acceso directo al hardware del sistema, no pudiendole sacar un máximo rendimiento del mismo. Por ejemplo está muy limitada la posibilidad de desarrollar juegos que movieran grandes gráficos a toda velocidad.

(Estos Inconvenientes cada vez tienen menor importancia, con el aumento de las prestaciones del hardware de los ordenadores)

# Conclusiones



*“Tu ordenador nunca será el mismo...  
Nunca más los programas de tu  
máquina determinarán las funciones  
que puedes realizar. **La red es el  
ordenador.** Los ordenadores se  
están convirtiendo en periféricos de  
Internet y la Web.”*

*George Gilder*