



```
public class JVM {
```

```
new VirtualMachine();
```

La Máquina Virtual de Java

JVM: El corazón del ecosistema Java



Arquitectura



Lenguajes



Evolución

```
ByteCode.execute();
```

¿Qué es la JVM?

Definición

La Máquina Virtual de Java (JVM) es un programa que ejecuta código bytecode de Java independientemente de la plataforma de hardware o sistema operativo subyacente.

Funciones Principales:



Carga y ejecución de bytecode

Interpreta o compila a código nativo el bytecode de Java



Gestión de memoria

Incluye recolección de basura automática



Portabilidad

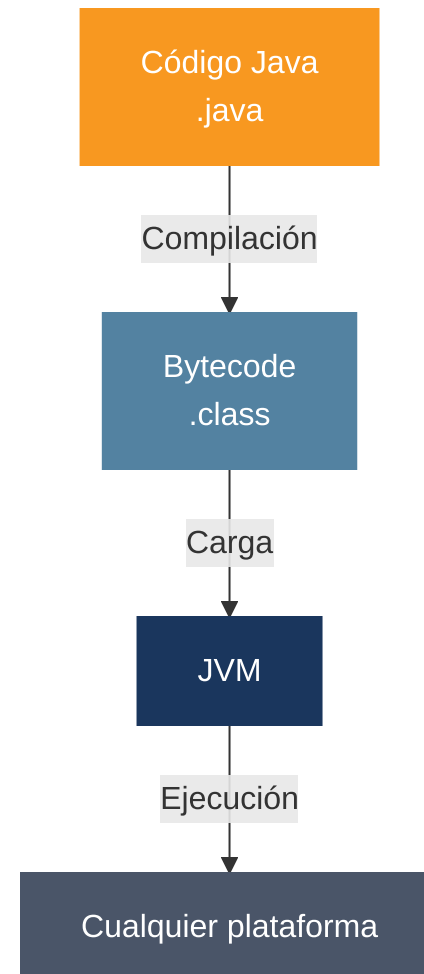
Permite "Write Once, Run Anywhere" (WORA)



Seguridad

Verificación de bytecode y sandboxing

Funcionamiento de la JVM



Una JVM para todos:



WORA: Write Once, Run Anywhere

El principal beneficio de la JVM es la capacidad de ejecutar programas Java en cualquier dispositivo sin modificación.

Propietario e Historia de la JVM

Propietario Actual

ORACLE

- ✓ Adquirió Java en 2010 al comprar Sun Microsystems
- ✓ Desarrolla la implementación oficial: Oracle HotSpot JVM
- ✓ Controla la especificación del lenguaje Java y JVM
- ✓ Gestiona el JCP (Java Community Process)

i Aunque Oracle posee la JVM, también existe OpenJDK como alternativa de código abierto.

JDK 1.0

JDK 1.1

J2SE 1.2

J2SE 1.3

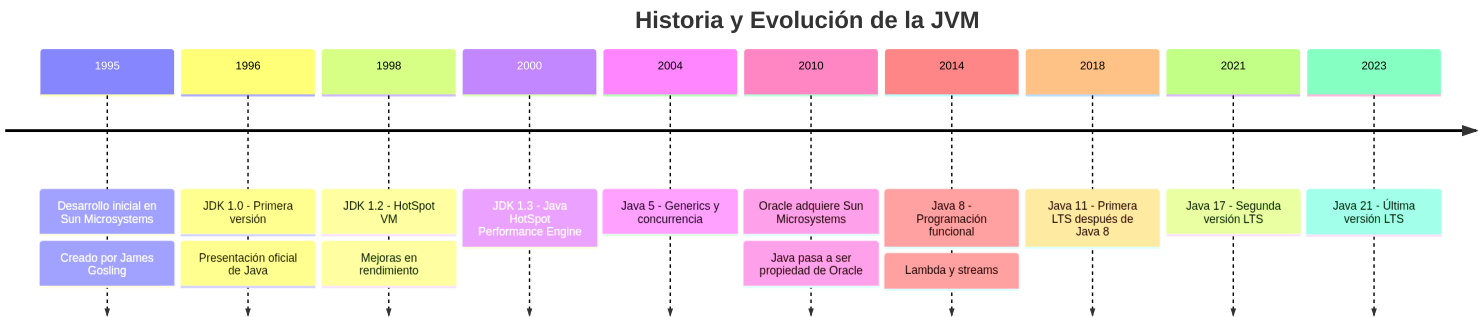
J2SE 1.4

J2SE 5.0

Java SE 6

Java SE 7

Evolución de la JVM



★ Hitos Importantes

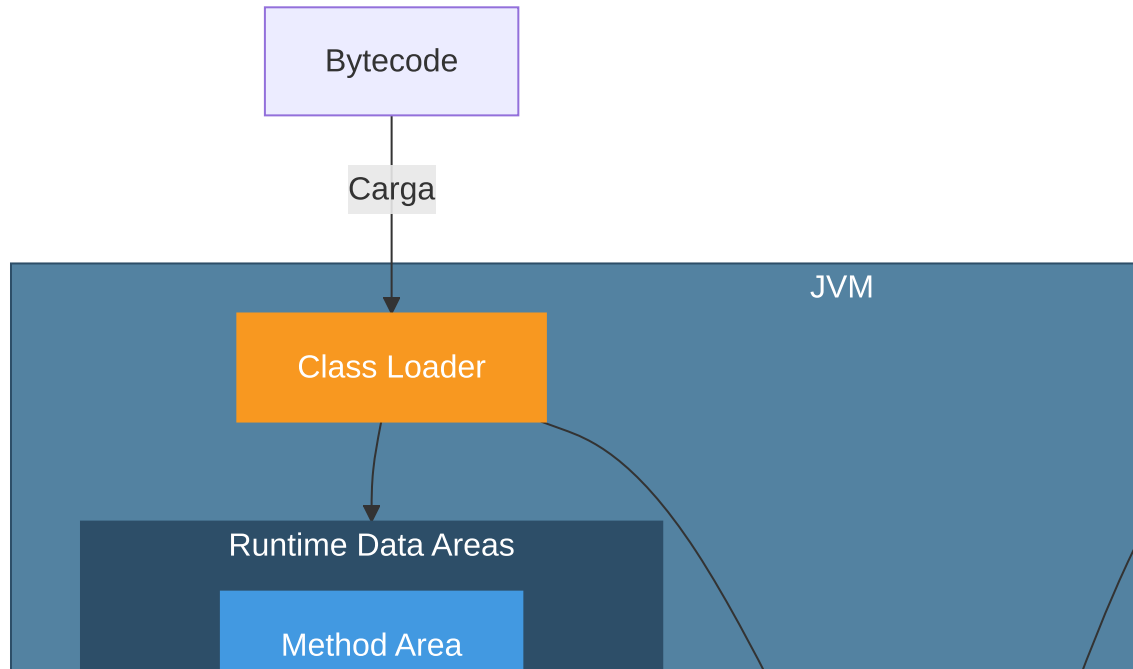
- > **1998:** Introducción de JIT (Just-In-Time) compiler
- > **2004:** Mejoras en gestión de memoria
- > **2018:** Nuevo ciclo de lanzamientos cada 6 meses

💡 Impacto Tecnológico

- > Popularizó el concepto de VM para lenguajes
- > Estableció el estándar WORA
- > Inspiró otras VMs y lenguajes de bytecode

Arquitectura de la JVM

Componentes de la JVM



¿Sabías que? La JVM es una máquina abstracta que provee un entorno de ejecución independiente del hardware y sistema operativo donde se ejecutan las aplicaciones Java.

Componentes Principales



Class Loader

Carga dinámicamente las clases Java en memoria

Loading Linking Initialization



Runtime Data Areas

Áreas de memoria para la ejecución del programa

■ Heap (Objetos)

■ Stack (Marcos)

■ Method Area

■ PC Registers



Execution Engine

Ejecuta el bytecode Java

⚡ Interpreter: Línea por línea

🧠 JIT Compiler: Optimiza ejecución


🗑️ Garbage Collector: Limpia memoria



Native Method Interface (JNI)

Permite la interacción con bibliotecas nativas escritas en C/C++

Lenguajes que se ejecutan en la JVM


Java
Principal

Lenguaje original para la JVM, creado por Sun Microsystems (ahora Oracle)

OOP
Typed
Enterprise

K Kotlin
Android

Desarrollado por JetBrains, ahora lenguaje oficial para Android

Conciso
Interoperable
Moderno

S Scala
Big Data

Combina programación orientada a objetos y funcional

Funcional
Spark
Inmutable

G Groovy
Scripting

Lenguaje dinámico con sintaxis similar a Java pero más concisa

Scripts
Grails
DSL

C Clojure
Lisp

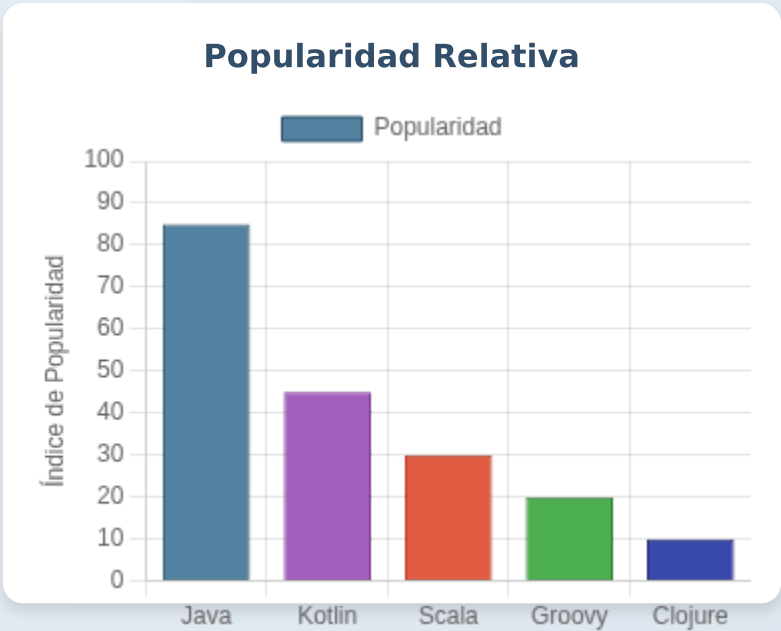
Dialecto moderno de Lisp para la JVM

Funcional
Inmutable
Concurrente

... Otros Lenguajes

Ceylon
Jython

JRuby
Frege



- ### ¿Por qué varios lenguajes?
- ✓ Diferentes paradigmas de programación
 - ✓ Sintaxis más moderna y concisa
 - ✓ Reutilización del ecosistema Java
 - ✓ Casos de uso específicos

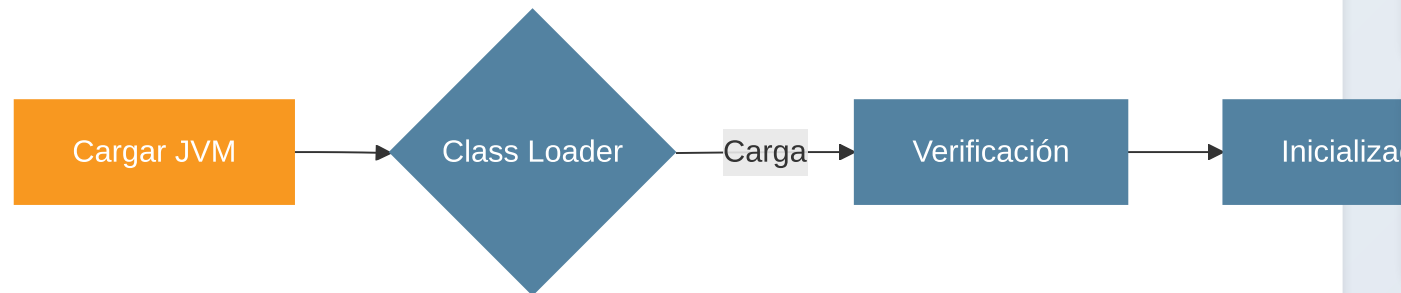
</> Interoperabilidad

Todos estos lenguajes compilan a bytecode de JVM, permitiendo:

↔ Usar librerías Java
⚙ Integración entre lenguajes
✂ Mismas herramientas JVM

Ciclo de Vida de la JVM

Ciclo de Vida Completo



1. Carga de Clases

Localiza y carga archivos .class en memoria

Loading

Linking

Initialization



2. Verificación

Comprueba la validez del bytecode por seguridad

Garantiza que el código no comprometerá la integridad de la JVM



3. Ejecución

Interpreta o compila JIT el bytecode Java

⚡ Interpreter

🔧 JIT Compiler



4. Recolección de Basura (GC)

Gestión automática de memoria durante la ejecución

Minor GC

Major GC

Full GC



5. Finalización

Cierre seguro de la JVM cuando el programa termina

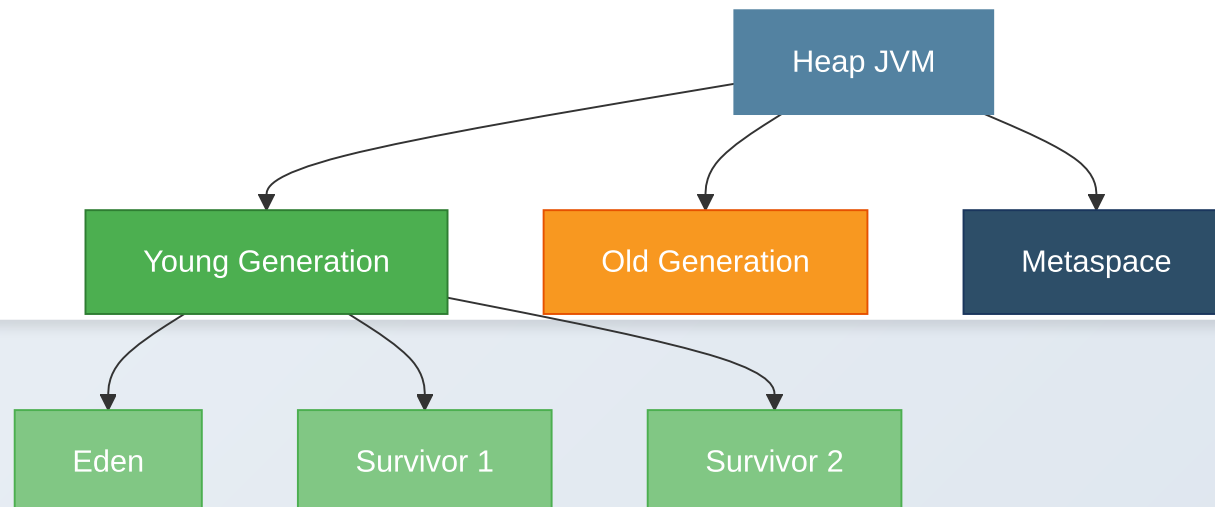
Libera recursos del sistema y finaliza todos los hilos




Consejos de Optimización

- ✓ Ajustar el tamaño del heap para evitar GC frecuentes
- ✓ Elegir el algoritmo de GC adecuado para tu aplicación
- ✓ Monitorizar el rendimiento con Java VisualVM o JConsole

Gestión de Memoria




Implementaciones de la JVM


Oracle HotSpot JVM
Estándar

La implementación de referencia de Oracle, optimizada para aplicaciones de servidor.

- ✓ Alto rendimiento
- ✓ Ampliamente probada
- ✓ JIT avanzado
- ✓ GC optimizado

Licencia: Oracle BCL (para uso comercial requiere licencia de pago)


OpenJDK
Open Source

Implementación de código abierto de la plataforma Java, base para muchas distribuciones.

- ✓ Código abierto
- ✓ Altamente compatible
- ✓ Gran comunidad
- ✓ Actualizaciones frecuentes

Licencia: GNU General Public License v2


Eclipse OpenJ9
Cloud

Antiguamente IBM J9, optimizada para entornos cloud y con menor huella de memoria.


- ✓ Arranque rápido
- ✓ Menor memoria
- ✓ Ideal para cloud
- ✓ Cache inteligente

Licencia: Eclipse Public License 2.0 + GNU GPL v2


GraalVM


JVM universal de Oracle con compilación nativa y poliglota

Nativo Poliglota Microservicios


Azul Zing/Zulu

JVM de alto rendimiento con recolector de basura pauseless

Baja latencia GC mejorado Empresarial


Amazon Corretto

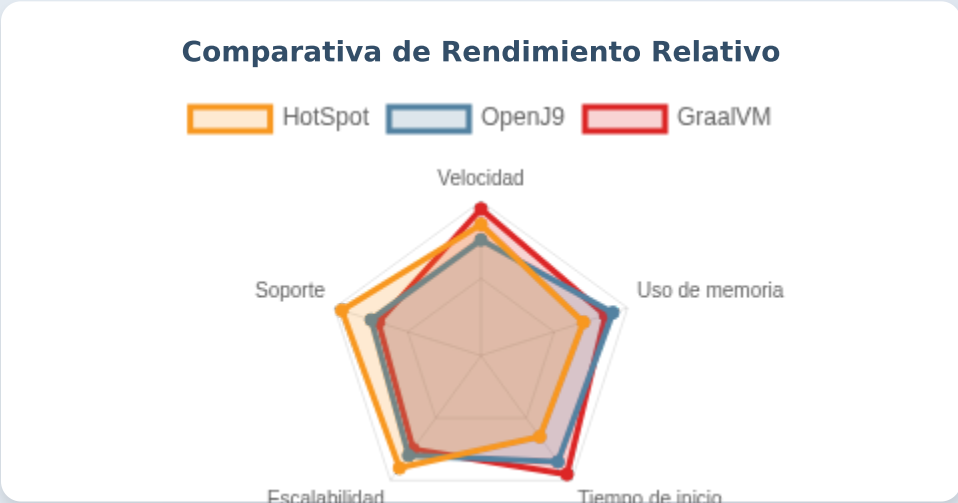
Distribución de Amazon basada en OpenJDK para AWS y producción

AWS LTS Gratuito


JRockit (Legacy)

JVM de BEA/Oracle, ahora fusionada en HotSpot JVM

Legacy Servidores Fusionado



¿Cómo elegir una JVM?

- ✓ **Empresas:** HotSpot, Azul Zing (si se requiere baja latencia)
- ✓ **Cloud:** OpenJ9, Amazon Corretto (en AWS)
- ✓ **Microservicios:** GraalVM para compilación nativa
- ✓ **Desarrollo:** OpenJDK o distribuciones gratuitas

Conclusiones y Recursos Adicionales

✓ Lo que hemos aprendido



La JVM es el corazón de Java

Permite ejecutar bytecode Java en cualquier plataforma, haciendo realidad el "Write Once, Run Anywhere".



Propiedad de Oracle

Adquirida con la compra de Sun Microsystems en 2010, con OpenJDK como alternativa de código abierto.



Arquitectura sofisticada

Class Loader, Runtime Data Areas, Execution Engine y el recolector de basura trabajan juntos para ejecutar aplicaciones de forma eficiente.



Múltiples lenguajes

Además de Java, la JVM ejecuta Kotlin, Scala, Groovy, Clojure y otros lenguajes, ampliando su ecosistema.

📖 Recursos para seguir aprendiendo



Documentación oficial

Oracle JVM Specification y Oracle Java Tutorials

docs.oracle.com/javase/specs/jvms



Cursos y videos

Udemy, Coursera, YouTube: "Java Virtual Machine Fundamentals"

coursera.org/courses?query=java



Libros recomendados

- "Java Virtual Machine Specification" - Oracle
- "Inside the Java Virtual Machine" - Bill Venner
- "Java Performance: The Definitive Guide" - S. Oaks



Herramientas para explorar

- Java VisualVM - Monitoreo y rendimiento
- JConsole - Herramienta de gestión JVM
- JMC (Java Mission Control) - Análisis profundo

¡Gracias!

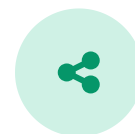
La comprensión de la JVM es fundamental para convertirse en un desarrollador Java experimentado



Programar



Aprender



Compartir