Programación I

Introducción al Lenguaje Java

Introducción al Lenguaje Java

Un poco de historia

Java inicia a principios de los 90.

Es creado por los ingenieros de Sun Microsystems James Gosling y Patrick Naughton.

Su objetivo inicial era ser utilizado para manejar pequeños electrodomésticos.

James Gosling

Patrick Naughton

Sun microsystems

Un poco de historia

Java inicia a principios de los 90.

Es creado por los ingenieros de Sun Microsystems James Gosling y Patrick Naughton.

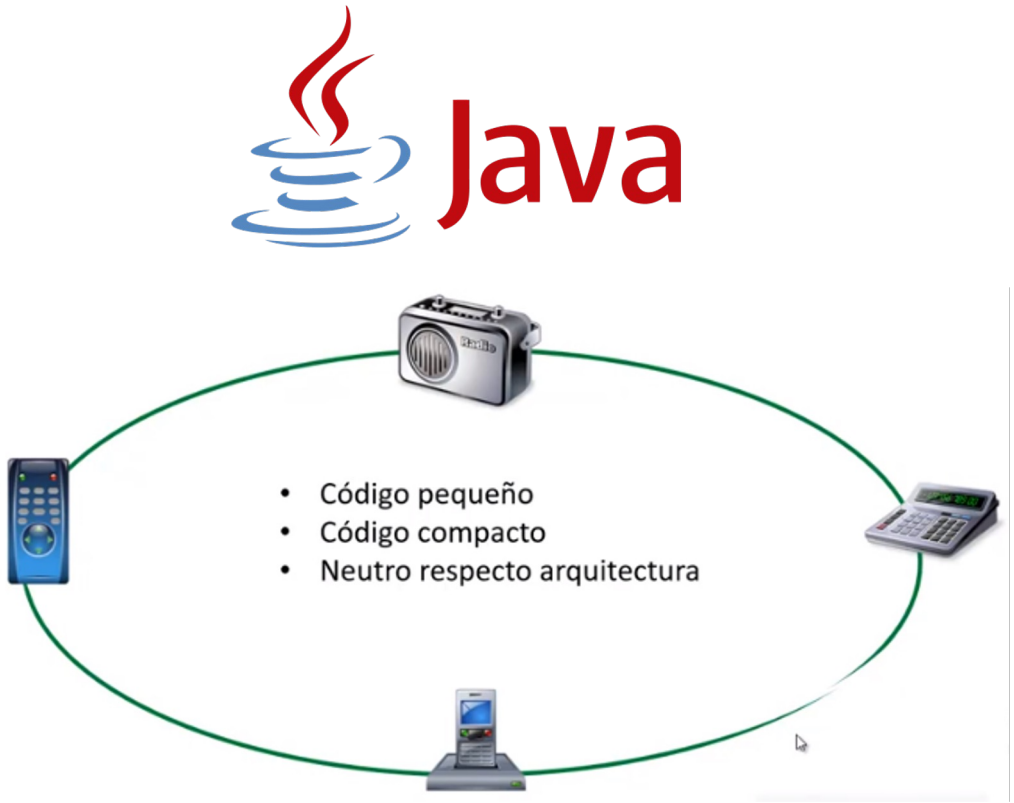
Su objetivo inicial era ser utilizado para manejar pequeños electrodomésticos.

James Gosling

Patrick Naughton

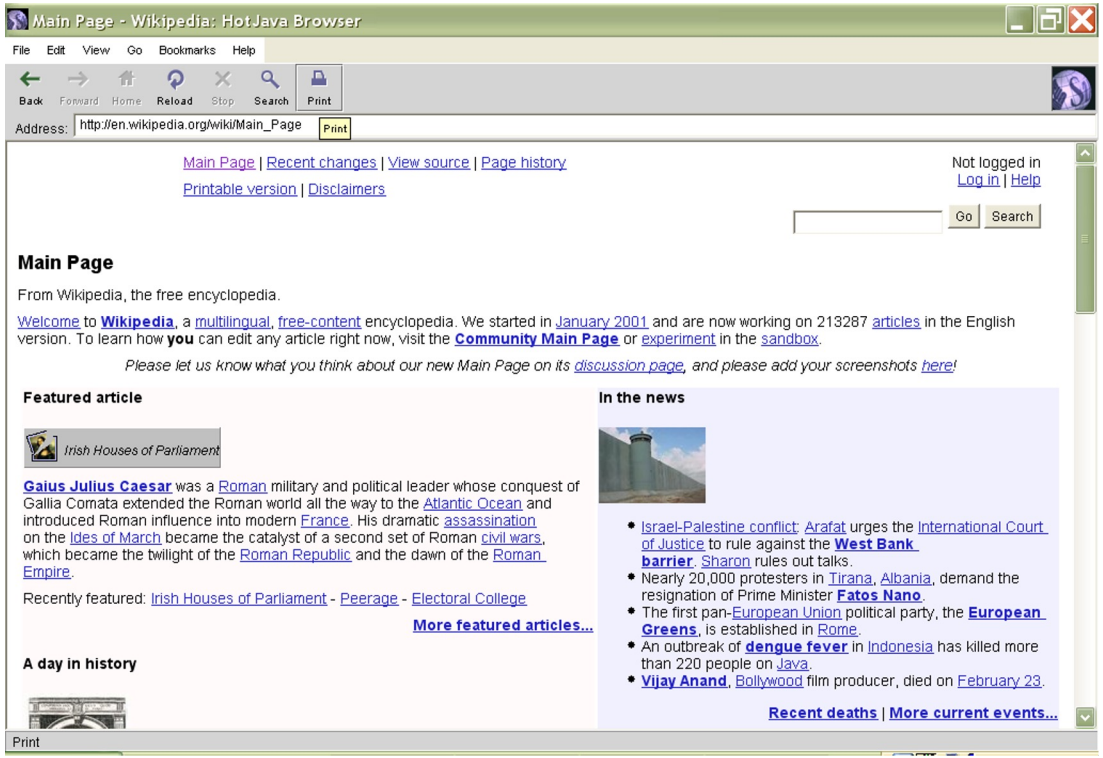
Características iniciales

Era un lenguaje simple, compacto y neutro respecto a la arquitectura (el hardware) requisito necesario para poder reutilizar el mismo software con diferentes electrodomésticos.



Hot Java

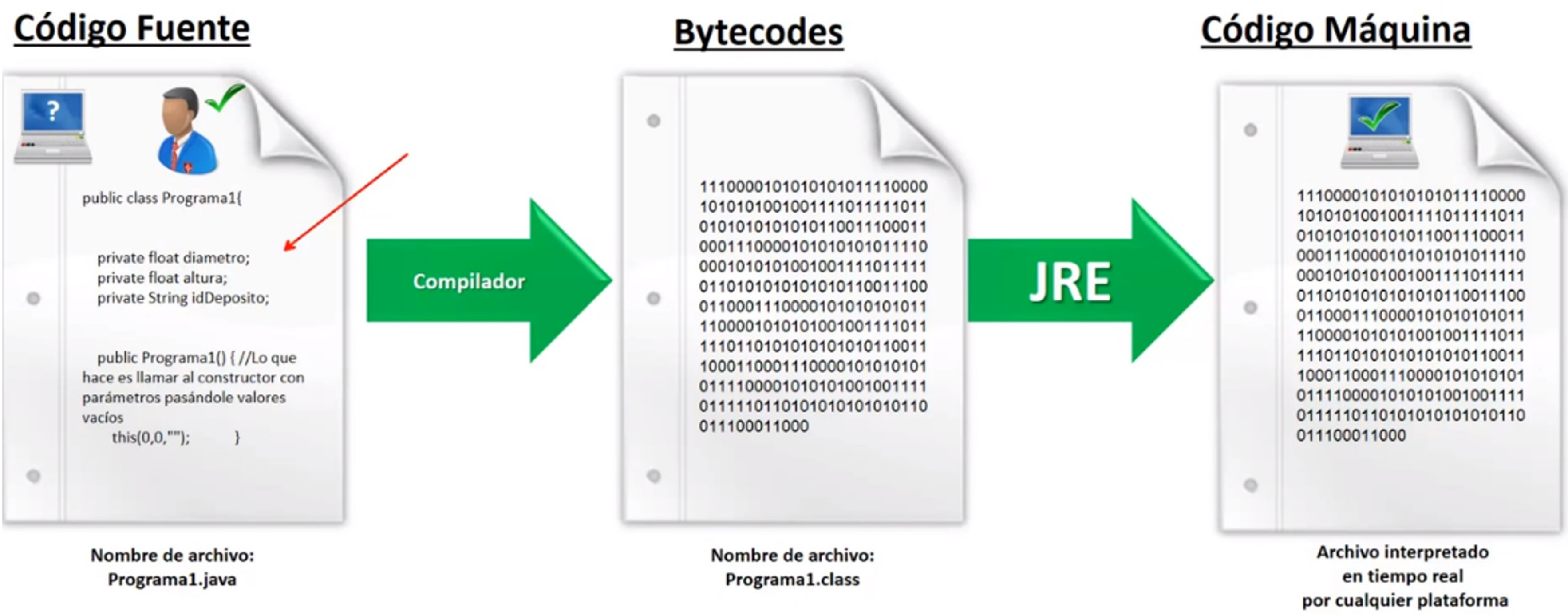
Navegador desarrollado en Java que se podía utilizar en ordenadores con diferentes plataformas (Windows, Unix, Solaris, etc) y que además tenía la característica de poder ejecutar pequeños programas denominados applets (hoy en desuso).



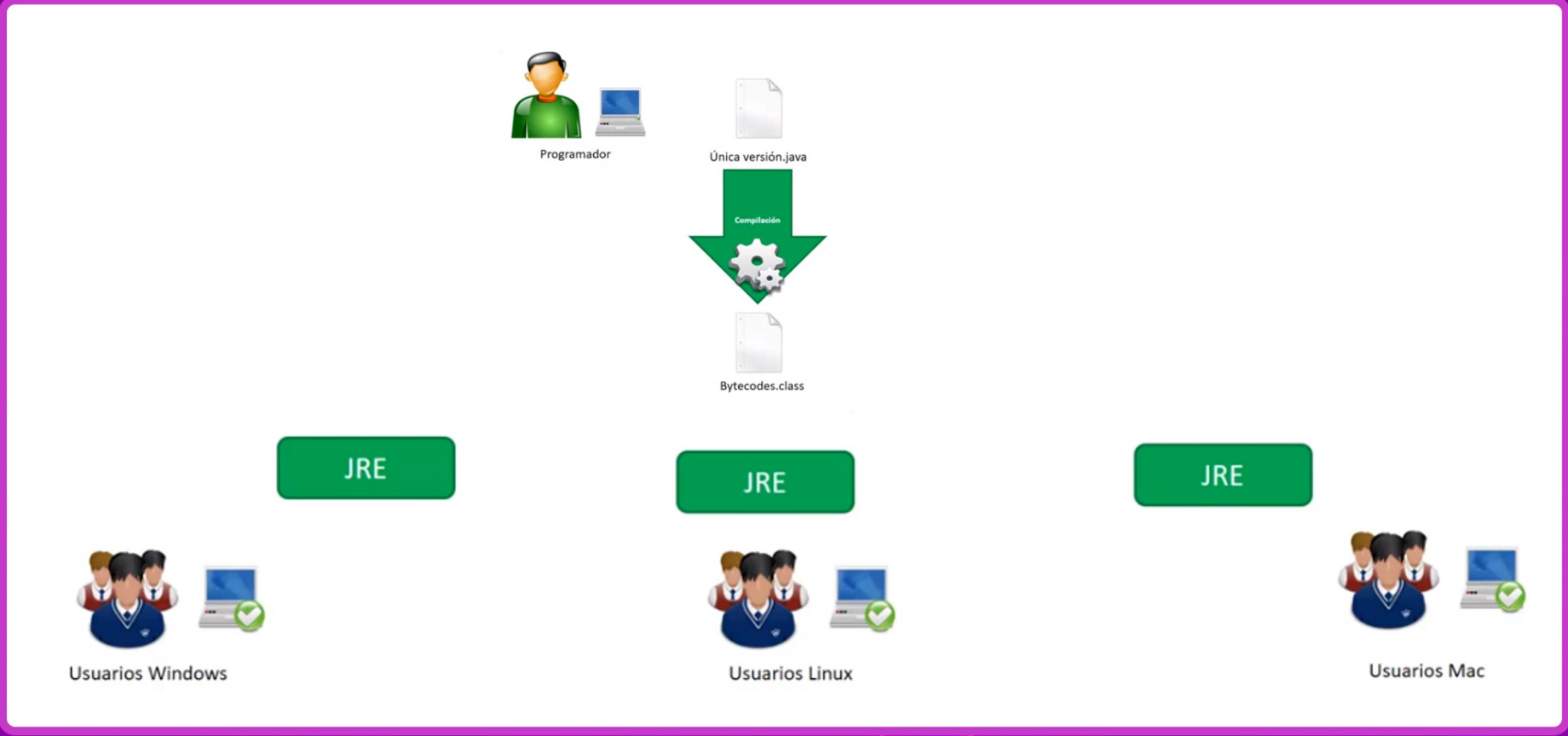
Hot Java


Programador
Versión 1
Versión 2
Versión 3
Ejecutable Windows
Ejecutable Linux
Ejecutable Mac
Usuarios Windows
Usuarios Linux
Usuarios Mac

Funcionamiento básico de JAVA



Funcionamiento básico de JAVA

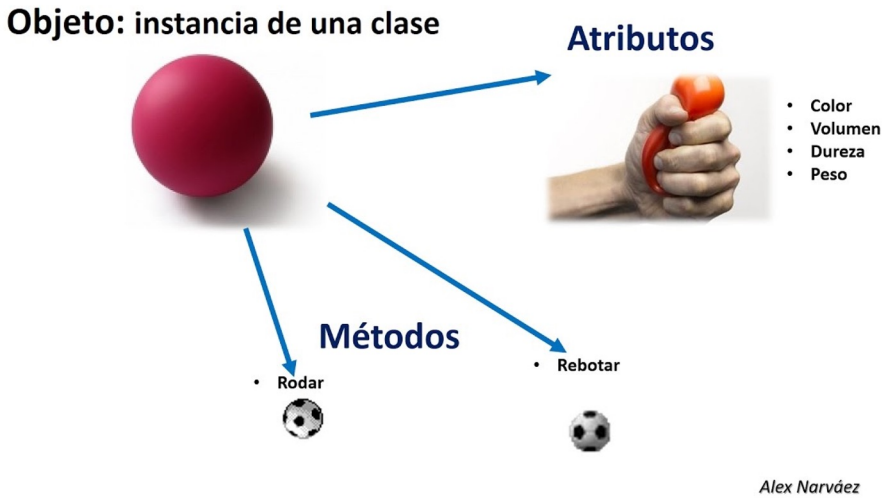


Características de Java

- Orientado a objetos
- Distribuido
- Robusto
- Seguro
- Neutro respecto a la arquitectura
- Adaptable
- Interpretado
- Alto rendimiento
- Multihilo
- Dinámico

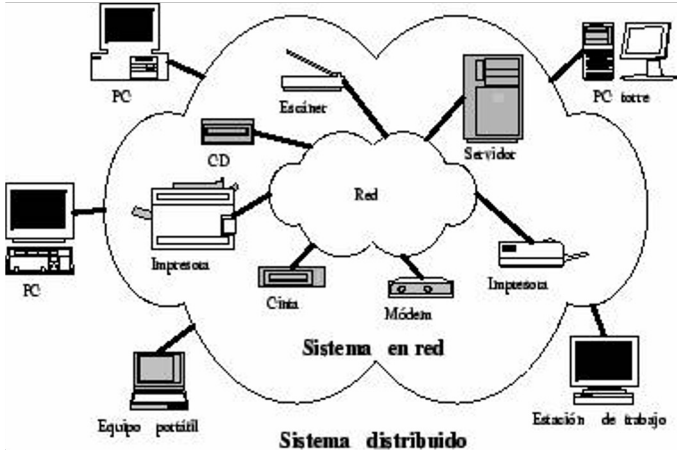
Orientado a objetos

El diseño orientado a objetos es una técnica de programación que se centra en los datos (esto es, los objetos) y en las interfaces de esos objetos.
Hoy en día no se concibe un lenguaje de programación moderno que no esté orientado a objetos.



Distribuido

Java posee una extensa biblioteca de rutinas para tratar protocolos de TCP/IP como HTTP y FTP. Las aplicaciones de Java pueden abrir objetos y acceder a objetos remotos con tanta facilidad como acceder a un sistema local de ficheros.
En la actualidad la arquitectura J2EE admite aplicaciones distribuidas a gran escala.



Robusto

Java está diseñado para escribir programas que deben resultar fiables en múltiples sentidos. Java hace hincapié en la comprobación temprana de posibles errores, efectúa después una comprobación dinámica (en el momento de la ejecución) y elimina las situaciones proclives a error.

El compilador de Java detecta muchos errores que en otros lenguajes de programación sólo sería posible detectarlos en tiempo de ejecución.

Seguro

Java está diseñado para utilizarse en entornos distribuidos o en entornos de red. Con este fin, se ha hecho mucho hincapié en la seguridad. Java permite construir sistemas libres de virus y manipulaciones.
Java se ha diseñado para evitar ciertos ataques como por ejemplo:

- Desbordar la pila de ejecución, ataque común de gusanos y virus.
- Corromper la memoria fuera de su propio proceso.
- Leer o escribir ficheros sin permiso.

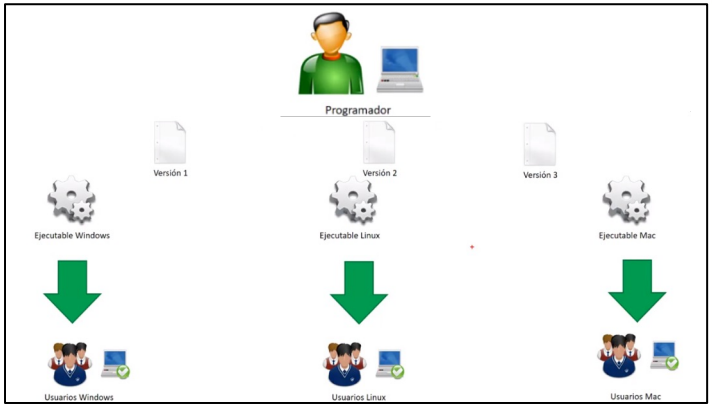
Con el tiempo se han ido añadiendo nuevas características de seguridad como por ejemplo las clases con firma digital. Esto permite saber quién ha escrito las clases.

Neutro respecto a la arquitectura

Al compilar un programa Java, se genera automáticamente un fichero de bytecodes independiente de la arquitectura de la máquina. El fichero de bytecodes será interpretado por la máquina virtual de Java (JRE).

Lo único que habrá que tener instalado en la máquina donde se desea ejecutar el programa Java es la máquina virtual de Java.

Un programa escrito en Java funcionará igual en una máquina con Windows, Linux, Unix, Solaris etc. También será independiente del hardware (procesador, placa, memoria etc.)



Adaptable

Los tamaños de los datos primitivos están especificados, así como el comportamiento de la aritmética que se les aplica

Ejemplo: en Java un int siempre es un entero de 32 bits. En C o C++ puede ser de 16 bits o de 32 bits.

Al tener un tamaño prefijado para los tipos de datos se elimina uno de los principales quebraderos de cabeza a la hora de crear adaptaciones.

Crear un programa que tenga el mismo aspecto en Windows, Macintosh o variantes de Unix es un gran esfuerzo para el programador. Java proporciona una interfaz de usuario común a todas las plataformas.

Interpretado

Al compilar un programa Java, el compilador genera un fichero de bytecodes que posteriormente será interpretado por la JRE (máquina virtual de Java).

Alto rendimiento

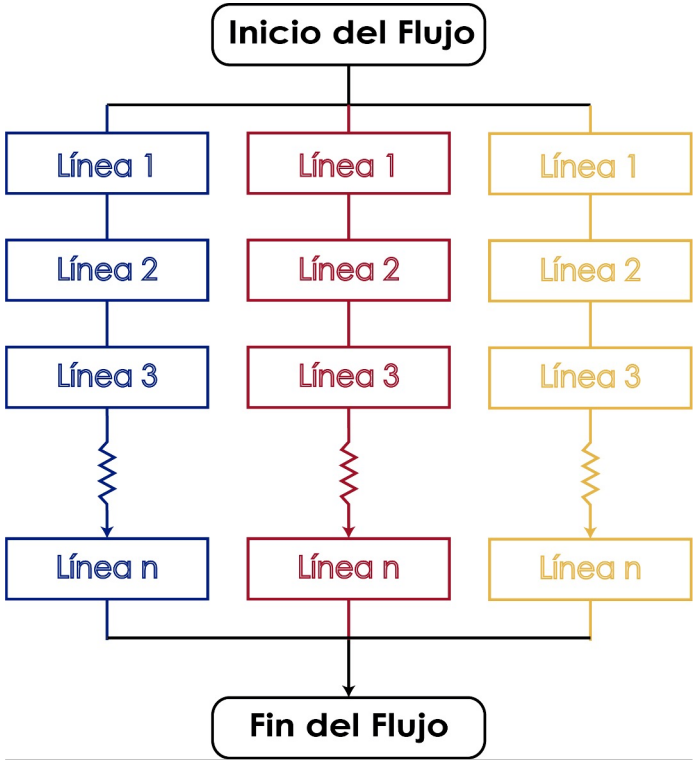
El rendimiento de los bytecodes suele ser aceptable en la mayoría de las ocasiones, pero para aquellas ocasiones en las que no resulte suficiente, existe la posibilidad de traducir sobre la marcha esos bytecodes a código máquina. Los resultados de esa traducción se guardan en memoria y cuando se vuelvan a necesitar se recurre a ellos. Esto acelera en muchas ocasiones el proceso de Interpretación de los bytecodes.

A esta característica se la denomina JIT (Just in Time), que viene a querer decir traducción sobre la marcha.

Multihilo

La ejecución multihilo consiste fundamentalmente en la ejecución de varias partes de código a la vez, de forma que a la hora de ejecutar el programa parecerá una ejecución multitarea.

Esta es una de las características más atractivas de Java.



Dinámico

Java se ha diseñado para adaptarse a un entorno cambiante. Las bibliotecas pueden añadir libremente nuevos métodos y variables de ejemplar sin que sus clientes se vean afectados.

Esta es una característica muy útil cuando se necesite añadir código a un programa en ejecución.

Tipos de aplicaciones JAVA

1. Aplicaciones de Consola
2. Aplicaciones de propósito general

Aplicaciones

```
C:\Program Files\Java\jdk1.6.0_16\bin>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:<lines,vars,source>     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are u
sed
  -classpath <path>          Specify where to find user class files and annotati
on processors
  -cp <path>                 Specify where to find user class files and annotati
on processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:<none,only>          Control whether annotation processing and/or compil
ation is done.
  -processor <class1>[,<class2>,<class3>...]Names of the annotation processors t
o run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -implicit:<none,class>     Specify whether or not to generate class files for
implicitly referenced files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release

  -target <release>          Generate class files for specific VM version
  -version                   Version information
  -help                      Print a synopsis of standard options
```