

Práctica 2: Introducción al desarrollo

5.1. Relación entre Software y Hardware	2
5.2. Código Fuente, Código Objeto y Ejecutable	4
5.3. Generación de Código Intermedio para Máquinas Virtuales	8
5.4. Clasificación de Lenguajes de Programación	8
5.5. Herramientas de Desarrollo	9

5.1. Relación entre Software y Hardware

C (Compilado)

Memoria RAM: Tu programa ejecutable se carga directamente en la memoria RAM. Las variables se almacenan en el stack y el heap. Tienes control directo sobre las direcciones de memoria a través de punteros.

Procesador (CPU): El procesador ejecuta las instrucciones de máquina de tu programa de forma nativa, una tras otra. Esto le da una velocidad y eficiencia máximas.

Periféricos: Funciones como `printf()` realizan llamadas al sistema (system calls), pidiendo al sistema operativo que interactúe con el hardware (ej., el controlador de vídeo) para mostrar texto en la pantalla.

Python (Interpretado)

Memoria RAM: El intérprete de Python se carga en la RAM. Él gestiona toda la memoria por ti, creando objetos y variables en el heap y liberándola automáticamente con un recolector de basura (garbage collector). No accedes a la memoria directamente.

Procesador (CPU): La CPU ejecuta las instrucciones del intérprete, no tu código Python directamente. El intérprete lee tu código y le dice a la CPU qué operaciones nativas debe realizar. Es una capa de indirección.

Periféricos: Funciones como `print()` le piden al intérprete que realice la llamada al sistema operativo correspondiente para mostrar la salida en la pantalla.

Java (Máquina Virtual)

Memoria RAM: La Máquina Virtual de Java (JVM) se carga en la RAM y reserva un bloque de memoria para sí misma. Dentro de este bloque, gestiona el stack, el heap y el metaspace de forma automática, incluyendo su propio recolector de basura.

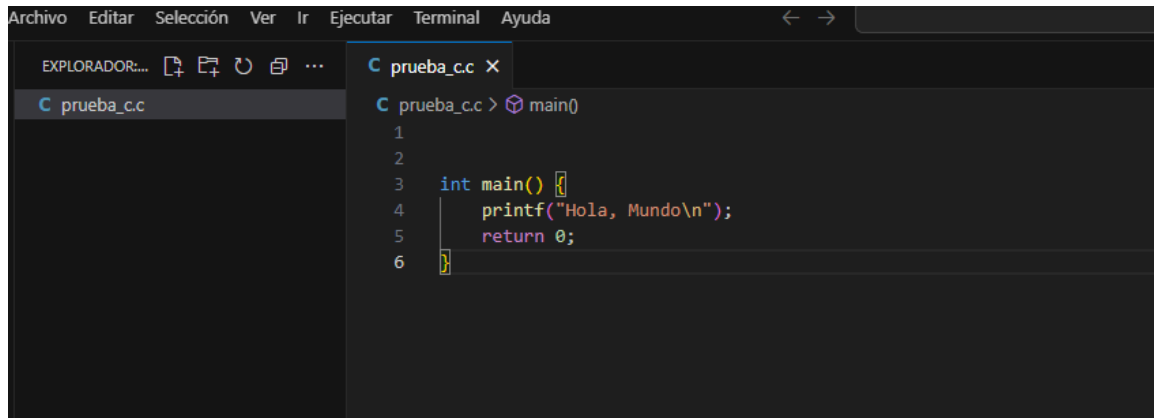
Procesador (CPU): La CPU ejecuta las instrucciones de la JVM. La JVM traduce el bytecode (código intermedio) de tu programa a instrucciones de máquina nativas, a menudo utilizando una compilación Just-In-Time (JIT) para optimizar el rendimiento de las partes más usadas.

Periféricos: Métodos como `System.out.println()` invocan código nativo dentro de la JVM, que es el encargado de realizar la llamada al sistema operativo para interactuar con la pantalla.

5.2. Código Fuente, Código Objeto y Ejecutable

C (Compilado)

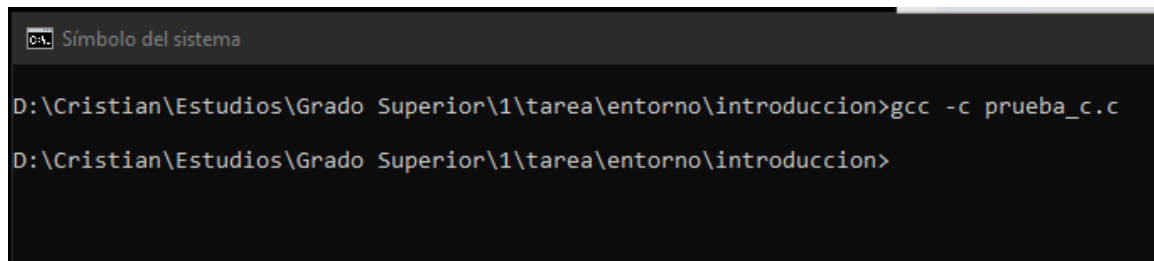
El código fuente (prueba_c.c) se transforma en un ejecutable en dos pasos:



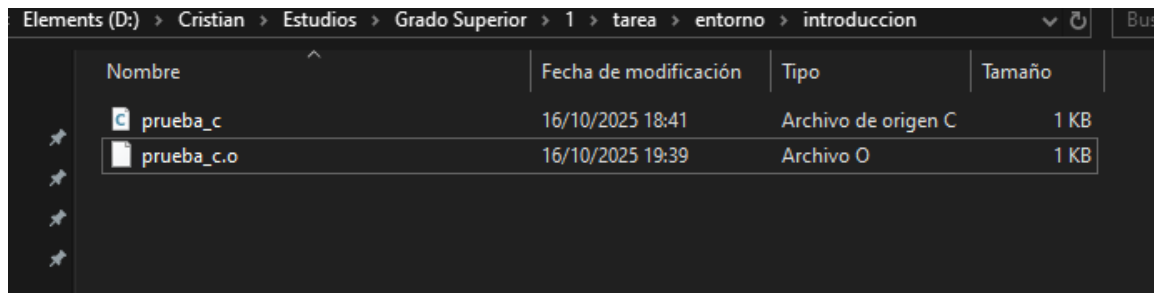
The screenshot shows a code editor with a menu bar (Archivo, Editar, Selección, Ver, Ir, Ejecutar, Terminal, Ayuda) and a toolbar. The left pane shows the Explorer with 'prueba_c.c' selected. The right pane shows the code for 'prueba_c.c' with a 'main()' function call at the top. The code is as follows:

```
1
2
3 int main() {
4     printf("Hola, Mundo\n");
5     return 0;
6 }
```

Compilación: El comando `gcc -c prueba_c.c` traduce el código fuente a código máquina, generando un archivo intermedio llamado código objeto (programa.o). Este archivo aún no es ejecutable.



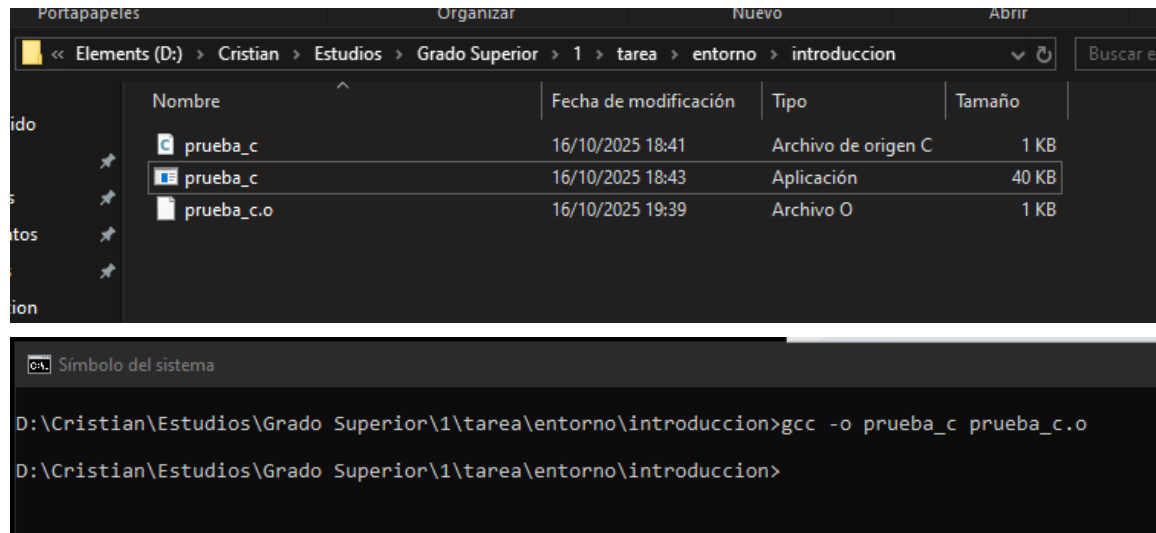
The screenshot shows a terminal window with the title 'Símbolo del sistema'. The command prompt is 'D:\Cristian\Estudios\Grado Superior\1\tarea\entorno\introduccion>'. The command entered is `gcc -c prueba_c.c`. The prompt returns to 'D:\Cristian\Estudios\Grado Superior\1\tarea\entorno\introduccion>'.



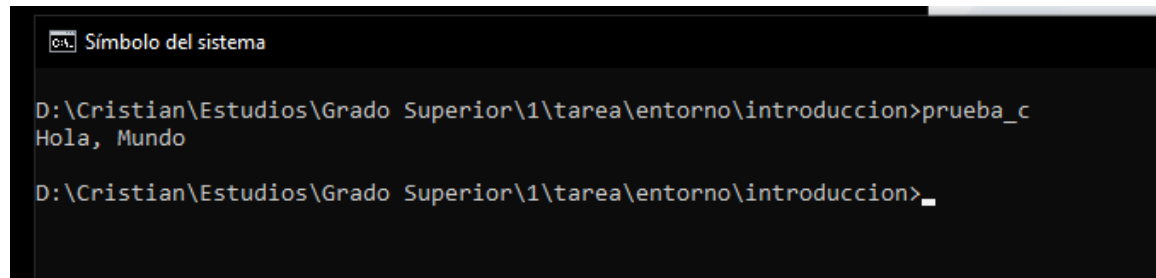
The screenshot shows a file explorer window with the path 'Elements (D:) > Cristian > Estudios > Grado Superior > 1 > tarea > entorno > introduccion'. The table below lists the files in the directory:

Nombre	Fecha de modificación	Tipo	Tamaño
prueba_c	16/10/2025 18:41	Archivo de origen C	1 KB
prueba_c.o	16/10/2025 19:39	Archivo O	1 KB

Enlazado (Linking): El comando `gcc -o prueba.o prueba_c` une tu código objeto con el código de las bibliotecas que necesita (como las de entrada/salida) para crear un archivo ejecutable final (`prueba_c`).

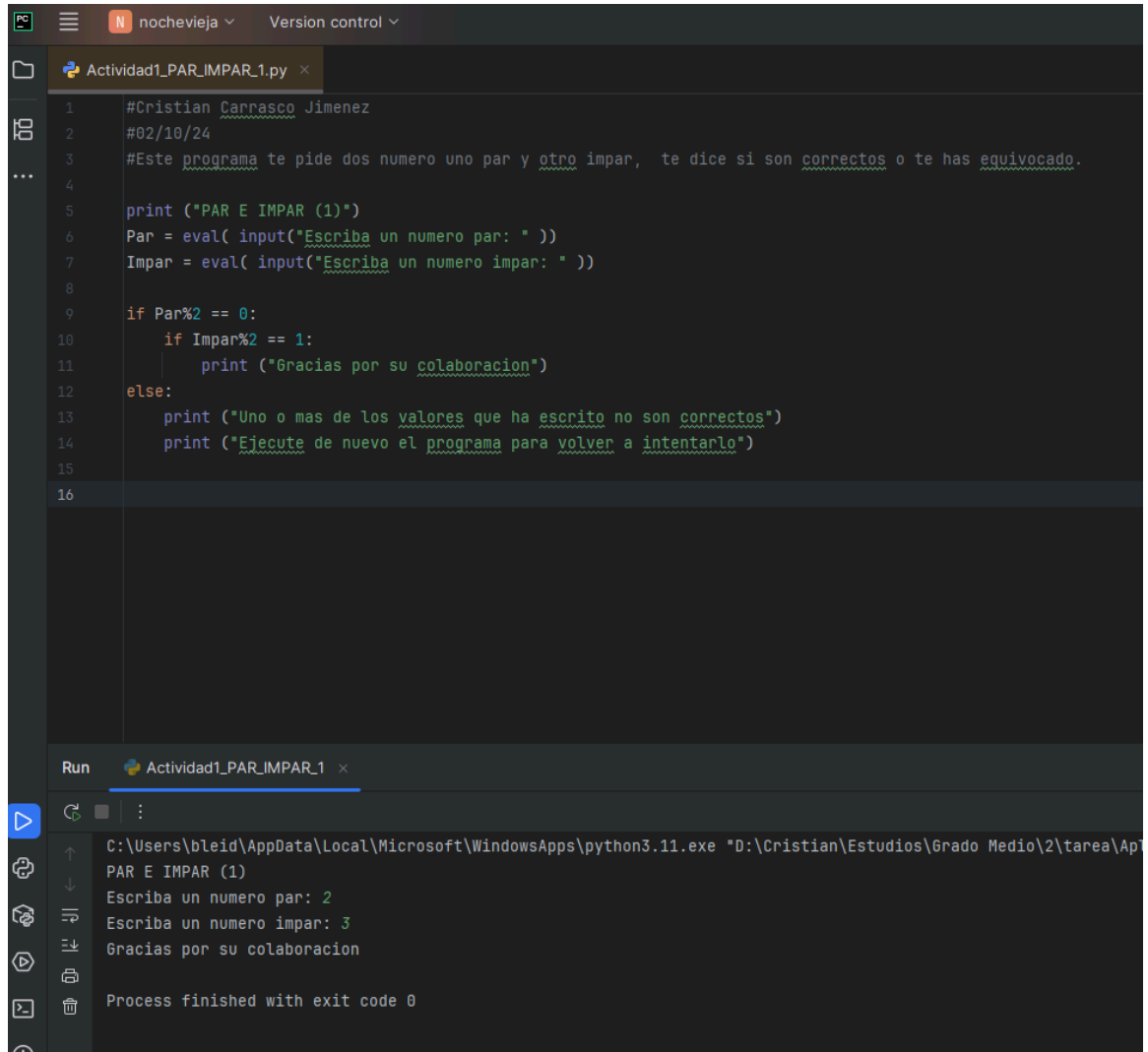


Y ya podremos ejecutar nuestro programa.



Python (Interpretado)

El código fuente (script.py) se ejecuta directamente por el intérprete (python script.py). No se genera un archivo de código objeto o ejecutable que tú gestionas. Internamente Python puede crear un archivo de bytecode (.pyc) en una carpeta `__pycache__`, pero esto es un proceso automático y transparente para el desarrollador.



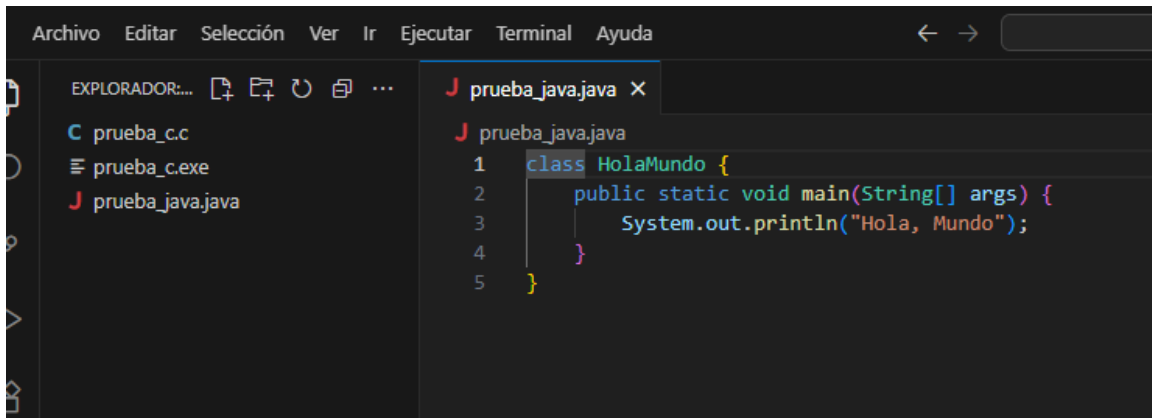
The screenshot shows a Python IDE with a file named `Actividad1_PAR_IMPAR_1.py`. The code is as follows:

```
1 #Cristian Carrasco Jimenez
2 #02/10/24
3 #Este programa te pide dos numero uno par y otro impar, te dice si son correctos o te has equivocado.
4
5 print ("PAR E IMPAR (1)")
6 Par = eval( input("Escriba un numero par: " ))
7 Impar = eval( input("Escriba un numero impar: " ))
8
9 if Par%2 == 0:
10     if Impar%2 == 1:
11         print ("Gracias por su colaboracion")
12 else:
13     print ("Uno o mas de los valores que ha escrito no son correctos")
14     print ("Ejecute de nuevo el programa para volver a intentarlo")
15
16
```

Below the editor, the `Run` panel shows the execution output:

```
Run Actividad1_PAR_IMPAR_1 x
C:\Users\bleid\AppData\Local\Microsoft\WindowsApps\python3.11.exe "D:\Cristian\Estudios\Grado Medio\2\tarea\Ap
PAR E IMPAR (1)
Escriba un numero par: 2
Escriba un numero impar: 3
Gracias por su colaboracion
Process finished with exit code 0
```

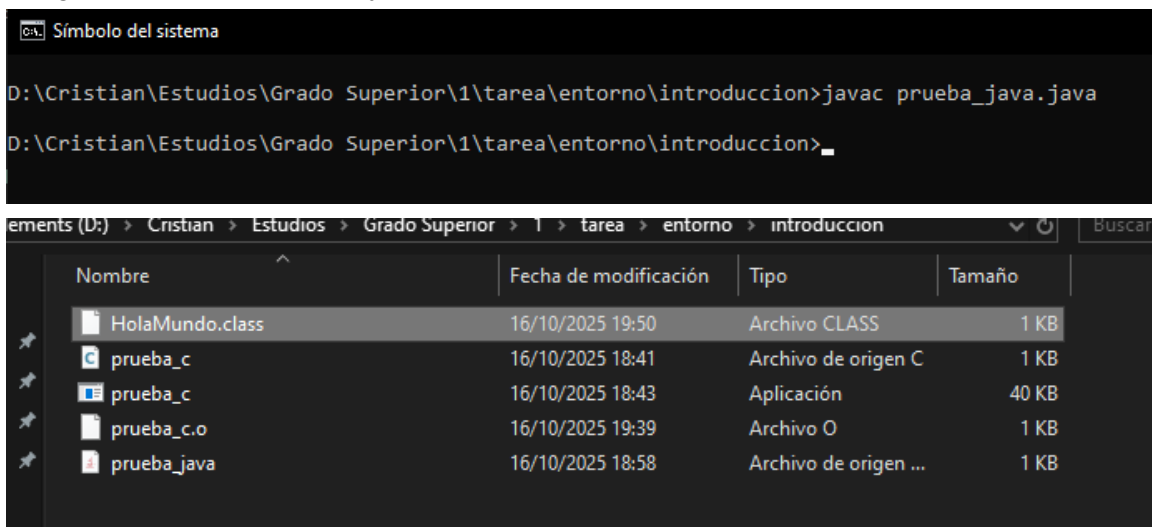
Java (Máquina Virtual)



The screenshot shows an IDE window with the file explorer on the left and the code editor on the right. The file explorer shows a project named 'prueba_java' with files 'prueba_c.c', 'prueba_c.exe', and 'prueba_java.java'. The code editor shows the following Java code:

```
1 class HolaMundo {  
2     public static void main(String[] args) {  
3         System.out.println("Hola, Mundo");  
4     }  
5 }
```

Compilación a código intermedio: El código fuente (prueba_java.java) se compila con el comando `javac prueba_java.java`. Esto no genera código máquina, sino un archivo de código intermedio llamado bytecode (HolaMundo.class).



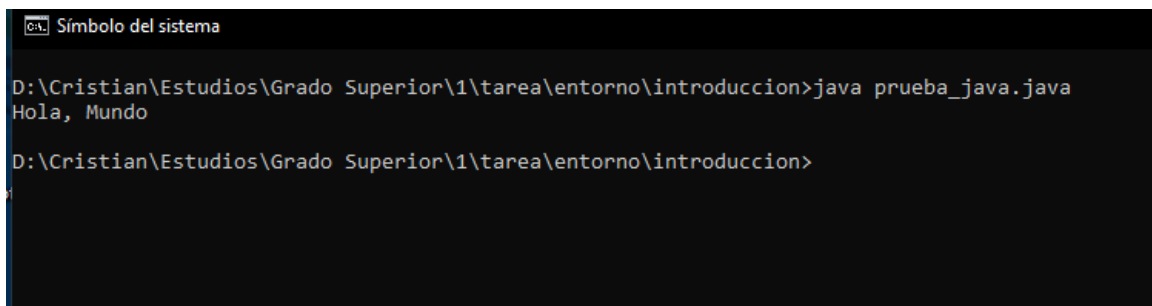
The top part of the screenshot shows a terminal window with the following commands and output:

```
C:\> Símbolo del sistema  
D:\Cristian\Estudios\Grado Superior\1\tarea\entorno\introduccion>javac prueba_java.java  
D:\Cristian\Estudios\Grado Superior\1\tarea\entorno\introduccion>_
```

The bottom part of the screenshot shows a file explorer window displaying the following files:

Nombre	Fecha de modificación	Tipo	Tamaño
HolaMundo.class	16/10/2025 19:50	Archivo CLASS	1 KB
prueba_c	16/10/2025 18:41	Archivo de origen C	1 KB
prueba_c	16/10/2025 18:43	Aplicación	40 KB
prueba_c.o	16/10/2025 19:39	Archivo O	1 KB
prueba_java	16/10/2025 18:58	Archivo de origen ...	1 KB

Ejecución por la VM: Este archivo .class es ejecutado por la Máquina Virtual de Java (JVM) con el comando `java prueba_java.java`. La JVM es la que traduce el bytecode a instrucciones que el procesador entiende.



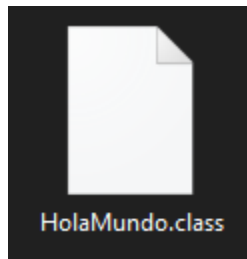
The screenshot shows a terminal window with the following commands and output:

```
C:\> Símbolo del sistema  
D:\Cristian\Estudios\Grado Superior\1\tarea\entorno\introduccion>java prueba_java.java  
Hola, Mundo  
D:\Cristian\Estudios\Grado Superior\1\tarea\entorno\introduccion>
```

5.3. Generación de Código Intermedio para Máquinas Virtuales

Java

Proceso de Generación: El compilador de Java (javac) analiza tu código fuente (.java), verifica que sea correcto y lo traduce a un conjunto de instrucciones universales e independientes de la plataforma, llamado bytecode. Este código se guarda en un archivo .class. Este archivo es portable y puede ejecutarse en cualquier dispositivo que tenga una JVM.



Rol de la Máquina Virtual (JVM): La JVM actúa como un "ordenador virtual" que se ejecuta sobre tu sistema operativo real.

5.4. Clasificación de Lenguajes de Programación

Característica	C	Python	Java
Modo de Ejecución	Compilado	Interpretado	Híbrido
Nivel de Abstracción	Medio-Bajo	Muy Alto	Alto
Paradigma	Imperativo	Multi-paradigma	Orientado a Objetos

5.5. Herramientas de Desarrollo

C (Compilado)

- Sistema Operativo: Linux (o WSL en Windows) por su acceso nativo a herramientas de desarrollo.
- IDE: Visual Studio Code (VS Code) con la extensión C/C++.
- Compilador: GCC (GNU Compiler Collection), usado desde la terminal.
- Depurador: GDB (GNU Debugger), integrado en la interfaz de VS Code.
- Sistema de Versiones: Git.
- Otras herramientas: Make, para automatizar la compilación de proyectos complejos con múltiples archivos mediante un Makefile.

Python (Interpretado)

- Sistema Operativo: Windows, macOS o Linux.
- IDE: Visual Studio Code (VS Code) con la extensión de Python.
- Intérprete: La implementación estándar CPython, invocada desde la terminal (python o python3).
- Depurador: El depurador visual integrado en VS Code.
- Sistema de Versiones: Git.
- Otras herramientas: pip (gestor de paquetes) para instalar librerías y venv para crear entornos virtuales que aíslan las dependencias de cada proyecto.

Java (Máquina Virtual)

- Sistema Operativo: Windows, macOS o Linux (gracias a la portabilidad de la JVM).
- IDE: Visual Studio Code (VS Code) con el Extension Pack for Java.
- Compilador/Intérprete: El JDK (Java Development Kit), que incluye javac (compilador) y java (lanzador de la JVM).
- Depurador: El depurador de Java integrado en VS Code.
- Sistema de Versiones: Git.
- Otras herramientas: Maven o Gradle. Son sistemas de gestión de dependencias y construcción que automatizan la compilación, la descarga de librerías y el empaquetado del proyecto (ej., en un archivo .jar)