

**UNIVERSIDAD MARIANO GALVEZ DE
GUATEMALA CAMPUS
JUTIAPA, JUTIAPA
FACULTAD DE INGENIERIA EN SISTEMAS DE
INFORMACION**

**CURSO:
PROGRAMACION III**

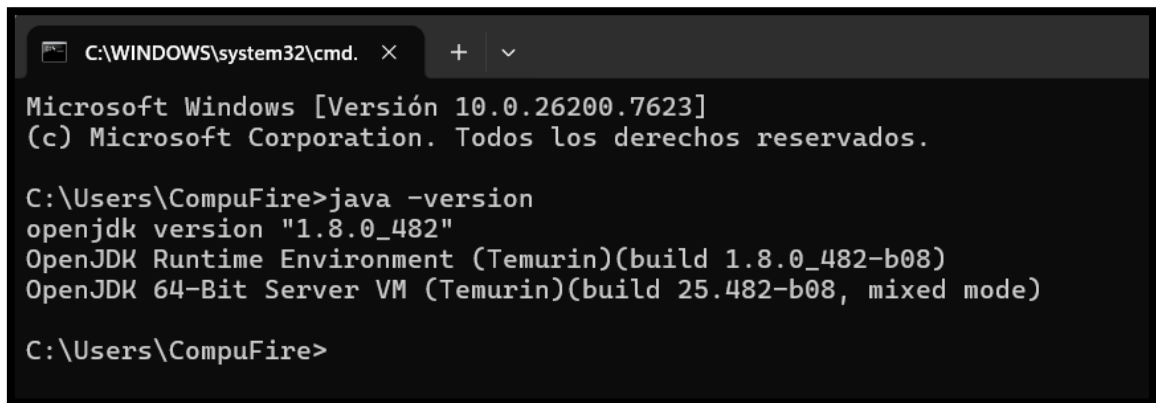
**DOCENTE:
ING. WALTER ANIBAL CORDOVA ZECEÑA**

**ALUMNO:
JEFERSÓN OSLEE CERMEÑO PINEDA**

**TAREA SOBRE:
CONFIGURACIÓN DE ENTORNO JAVA/MAVEN Y ARREGLOS EN JAVA**

FECHA: 13/02/2026

- **Java -version**



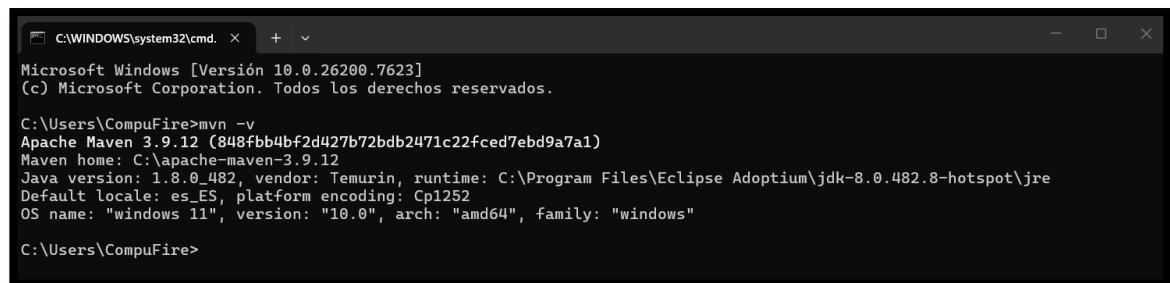
```
C:\WINDOWS\system32\cmd. x + v

Microsoft Windows [Versión 10.0.26200.7623]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\CompuFire>java -version
openjdk version "1.8.0_482"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_482-b08)
OpenJDK 64-Bit Server VM (Temurin)(build 25.482-b08, mixed mode)

C:\Users\CompuFire>
```

- **Mvn -v**



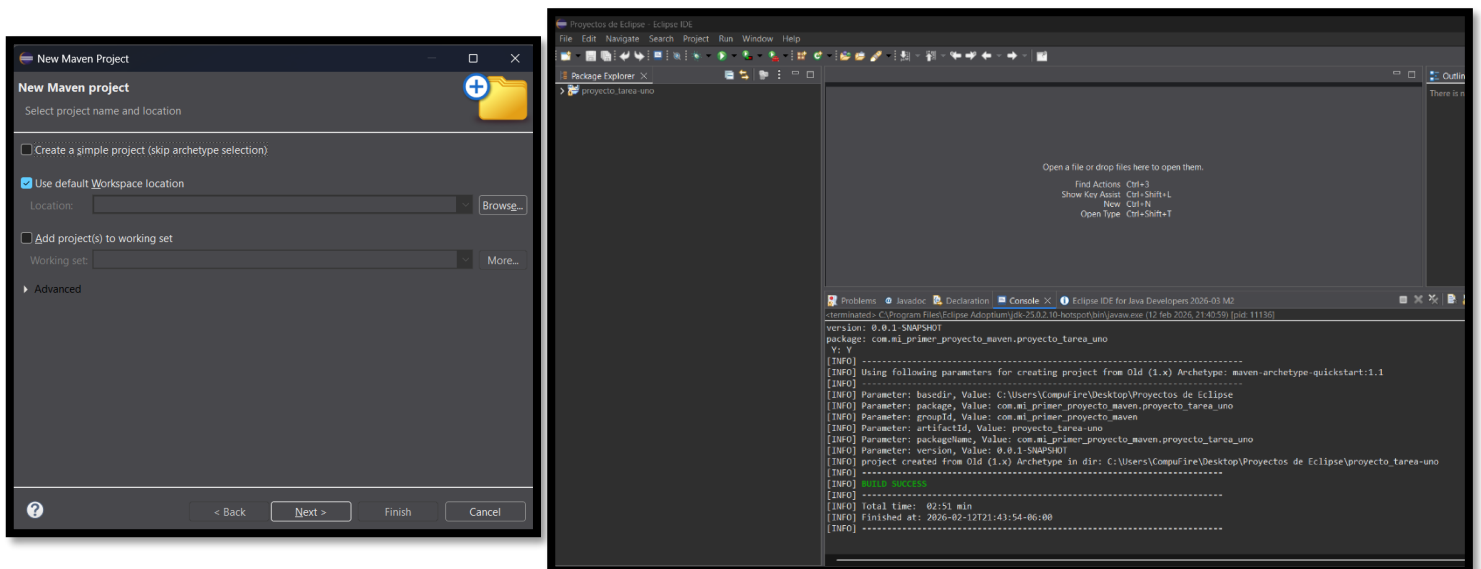
```
C:\WINDOWS\system32\cmd. x + v

Microsoft Windows [Versión 10.0.26200.7623]
(c) Microsoft Corporation. Todos los derechos reservados.

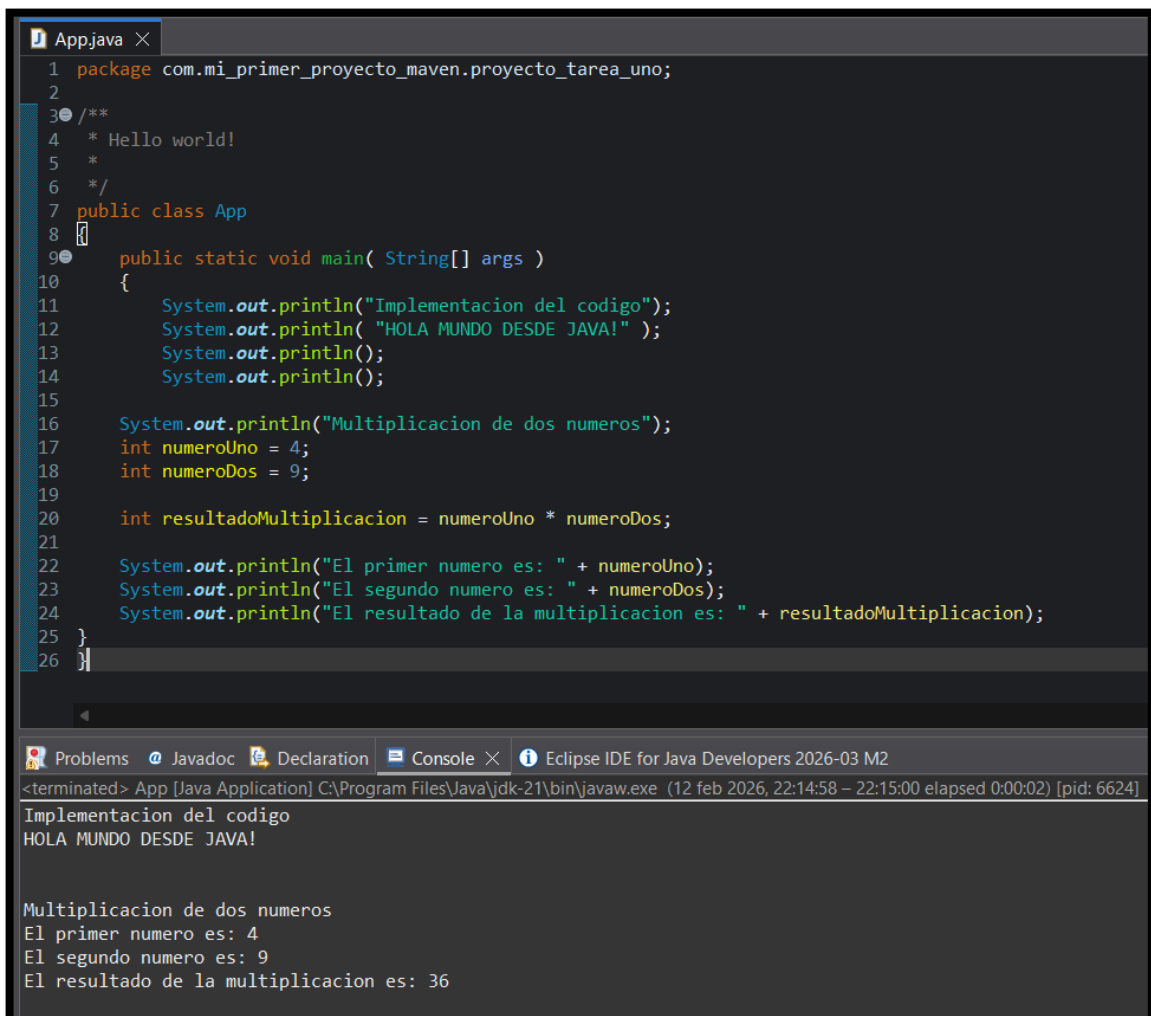
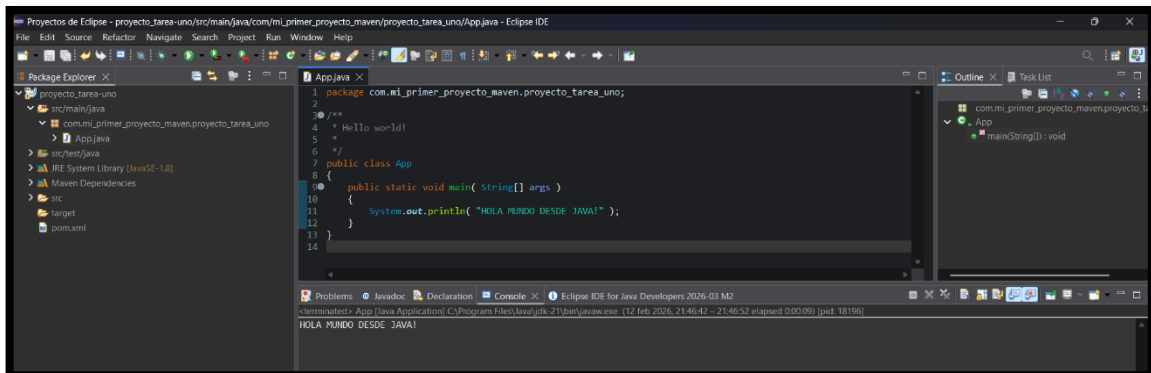
C:\Users\CompuFire>mvn -v
Apache Maven 3.9.12 (848fbb4bfb2d427b72bdb2471c22fced7ebd9a7a1)
Maven home: C:\apache-maven-3.9.12
Java version: 1.8.0_482, vendor: Temurin, runtime: C:\Program Files\Eclipse Adoptium\jdk-8.0.482.8-hotspot\jre
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\CompuFire>
```

- **Crear un Proyecto Maven**



Ejecución de los programas en consola:

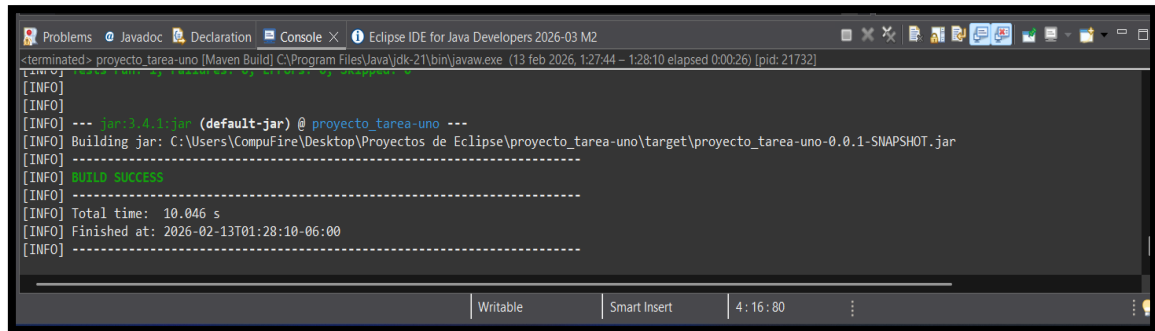


Evidencia de Ofuscación e Ingeniería Inversa

Introducción

En esta práctica se realizó un proceso de ingeniería inversa sobre un archivo ejecutable .jar generado en Java. Posteriormente, se aplicó ofuscación utilizando la herramienta ProGuard, con el objetivo de dificultar la lectura del código al ser descompilado. Se presentan evidencias comparativas antes y después del proceso.

JAR:

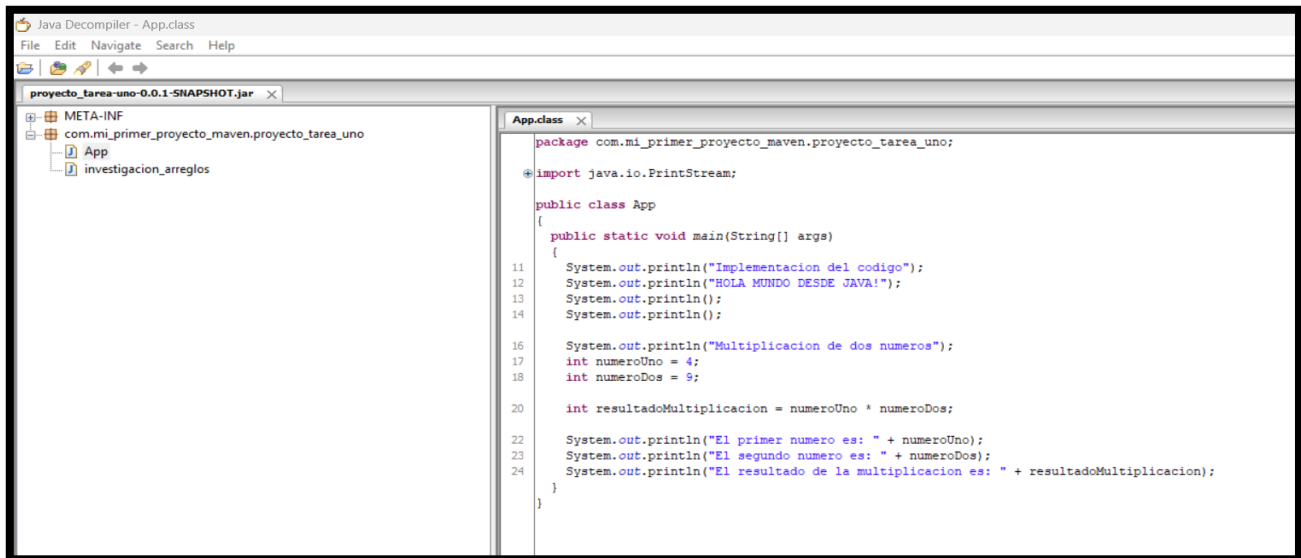


```
<terminated> proyecto_tarea-uno [Maven Build] C:\Program Files\Java\jdk-21\bin\javaw.exe (13 feb 2026, 1:27:44 - 1:28:10 elapsed 0:00:26) (pid: 21732)
[INFO]
[INFO]
[INFO] --- jar:3.4.1:jar (default-jar) @ proyecto_tarea-uno ---
[INFO] Building jar: C:\Users\CompuFire\Desktop\Proyectos de Eclipse\proyecto_tarea-uno\target\proyecto_tarea-uno-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO]
[INFO] Total time: 10.046 s
[INFO] Finished at: 2026-02-13T01:28:10-06:00
[INFO]
```

Nombre	Fecha de modificación	Tipo	Tamaño
.settings	12/02/2026 21:44	Carpeta de archivos	
src	12/02/2026 21:43	Carpeta de archivos	
target	13/02/2026 1:28	Carpeta de archivos	
.classpath	12/02/2026 21:44	Archivo CLASSPATH	2 KB
.project	12/02/2026 21:44	Archivo PROJECT	1 KB
pom	12/02/2026 21:43	Microsoft Edge HT...	1 KB

proyecto_tarea-uno-0.0.1-SNAPSHOT	13/02/2026 1:28	Archivo JAR	6 KB
-----------------------------------	-----------------	-------------	------

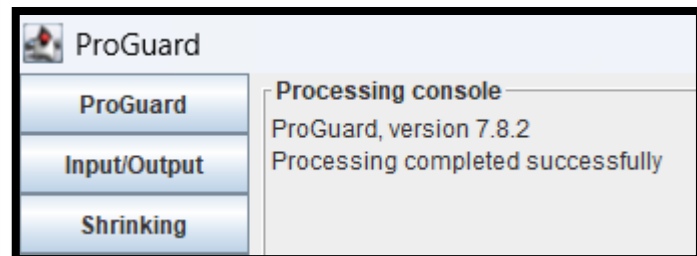
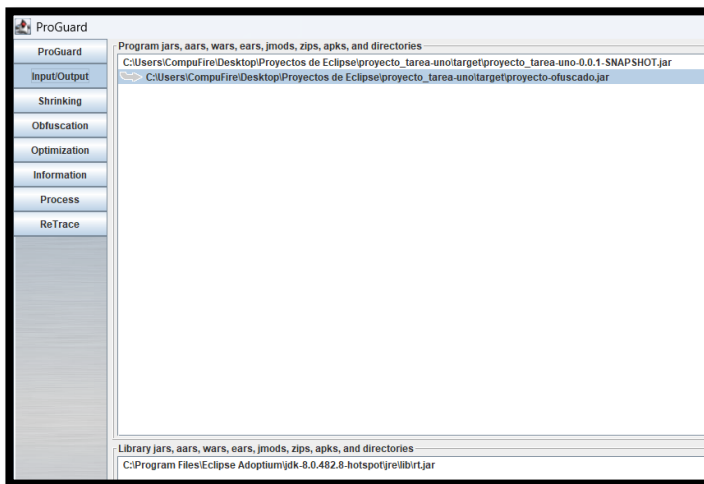
Ingeniería inversa (Antes de ofuscar).



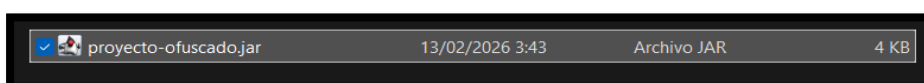
Al descompilar el archivo .jar original utilizando JD-GUI, se puede observar que el código fuente es fácilmente legible. Los nombres de clases y variables permanecen intactos, lo que demuestra que Java puede ser sometido a ingeniería inversa sin dificultad.

Proceso de Ofuscación

- Se utilizó ProGuard.

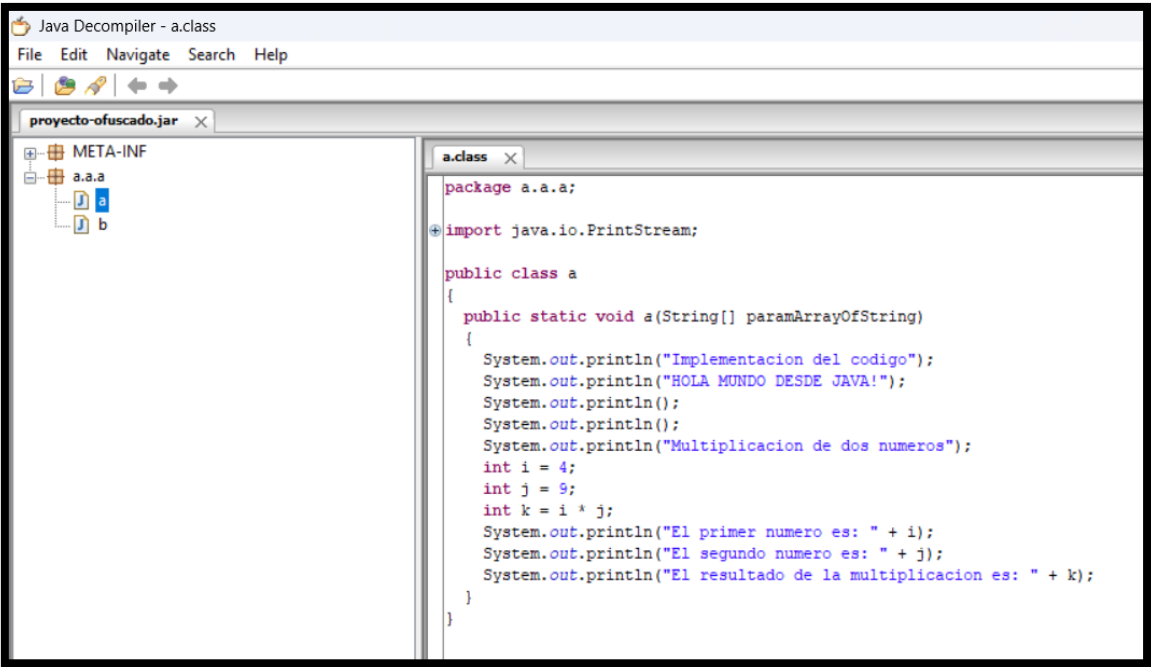


- Se generó un nuevo archivo ofuscado.jar.



- Se mantuvo la funcionalidad del programa.

Ingeniería inversa (Después de ofuscar)



```
package a.a.a;

import java.io.PrintStream;

public class a
{
    public static void a(String[] paramArrayOfString)
    {
        System.out.println("Implementacion del codigo");
        System.out.println("HOLA MUNDO DESDE JAVA!");
        System.out.println();
        System.out.println();
        System.out.println("Multiplicacion de dos numeros");
        int i = 4;
        int j = 9;
        int k = i * j;
        System.out.println("El primer numero es: " + i);
        System.out.println("El segundo numero es: " + j);
        System.out.println("El resultado de la multiplicacion es: " + k);
    }
}
```

Después de aplicar la ofuscación, al descompilar el nuevo archivo .jar, se observa que los nombres de clases y variables han sido modificados a identificadores genéricos como "a", "b", y "i", "j", "k". Esto dificulta la comprensión del código, aumentando el nivel de protección contra ingeniería inversa.

Conclusión

La ofuscación no impide completamente la ingeniería inversa, pero sí dificulta significativamente la comprensión del código fuente. Esto demuestra la importancia de aplicar técnicas de protección en aplicaciones Java cuando se desea proteger la lógica interna del programa.

Tabla de Comparación:

Antes de ofuscar	Después de ofuscar
Clase con nombre real	Clase llamada "a"
Variables descriptivas	Variables genéricas
Código fácil de entender	Código difícil de entender