MEDAC
Instituto Oficial de Formación Profesional

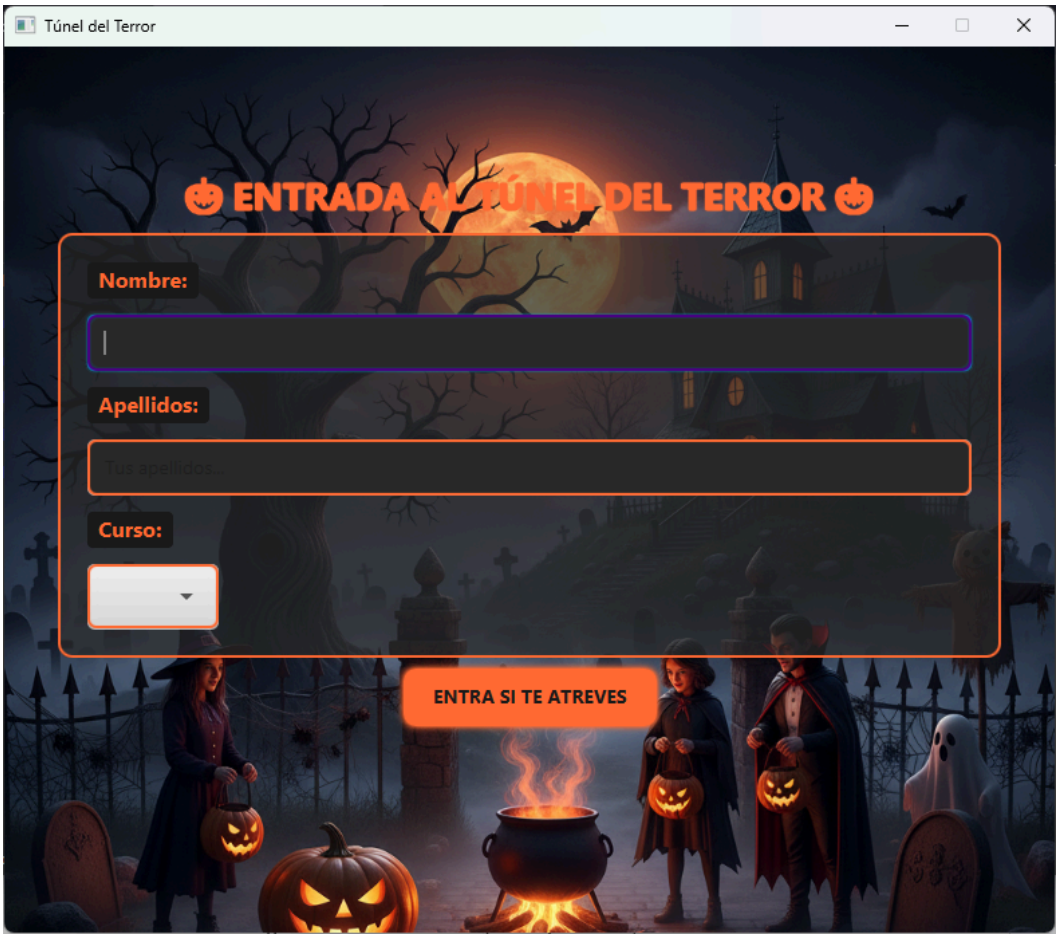
Memoria Técnica: Despliegue y Distribución de Aplicación JavaFX

Gabriel Sanchez Heredia

Índice

1. Introducción

El objetivo de esta práctica es generar un ciclo completo de distribución de software para la aplicación de escritorio "HalloWheel", desarrollada en JavaFX. Se ha transformado el proyecto desde su código fuente hasta un instalador profesional para Windows (.exe), asegurando que la aplicación sea autocontenida (no requiere que el usuario tenga Java instalado previamente).



2. Proceso de desarrollo

2.1. Generación del JAR (Maven)

Para empaquetar la aplicación, se configuró el ciclo de vida de Maven.

- **Gestión de Dependencias:** Se utilizó el archivo pom.xml para gestionar las librerías de JavaFX y otras dependencias externas.
- **Plugin Shade:** Se implementó maven-shade-plugin para crear un "Fat JAR" que incluye todas las dependencias necesarias.
- **Clase Launcher:** Para evitar conflictos con el Java Module System al iniciar la aplicación, se creó una clase intermedia Launcher.java.

pom.xml

```xml
<!-- Maven Shade Plugin: permite generar un JAR ejecutable con todas las dependencias -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>3.5.0</version>
    <executions>
        <execution>
            <phase>package</phase> <!-- Se ejecuta al empaquetar -->
            <goals>
                <goal>shade</goal> <!-- Crea un JAR "sombreado" con dependencias incluidas -->
            </goals>
            <configuration>
                <transformers>
                    <!-- Indica la clase principal en el manifest del JAR -->
                    <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                        <mainClass>org.tunelterror.di05_tunelterror.Launcher</mainClass>
                    </transformer>
                </transformers>
            </configuration>
        </execution>
    </executions>
</plugin>
```

2.2. Creación del JRE Personalizado (Modularización)

En lugar de depender de que el usuario tenga instalado el JDK completo, se optó por crear un entorno de ejecución (JRE) ligero y personalizado utilizando la herramienta jlink.

- Se combinaron los módulos base del JDK (versión 25/21) con los módulos específicos de JavaFX (versión 21).
- Se optimizó el tamaño final eliminando información de depuración (--strip-debug) y documentación (--no-man-pages), reduciendo el peso del entorno de +300MB a aproximadamente 60MB.

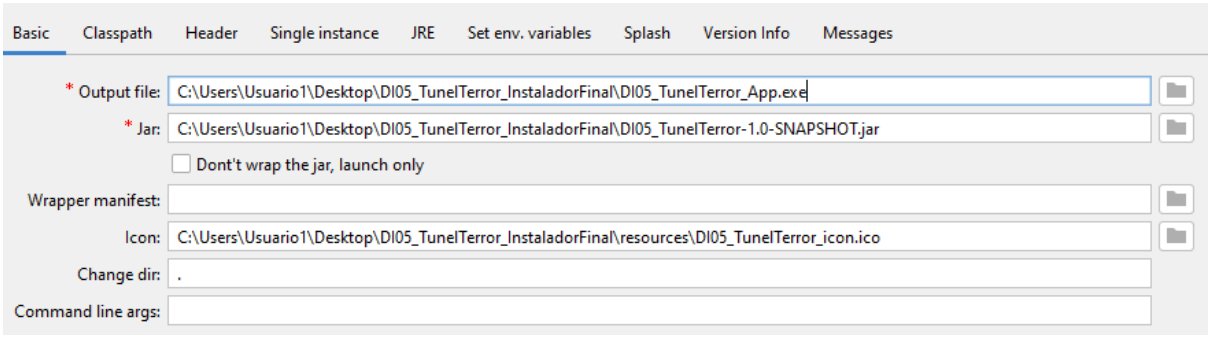
Comando utilizado:

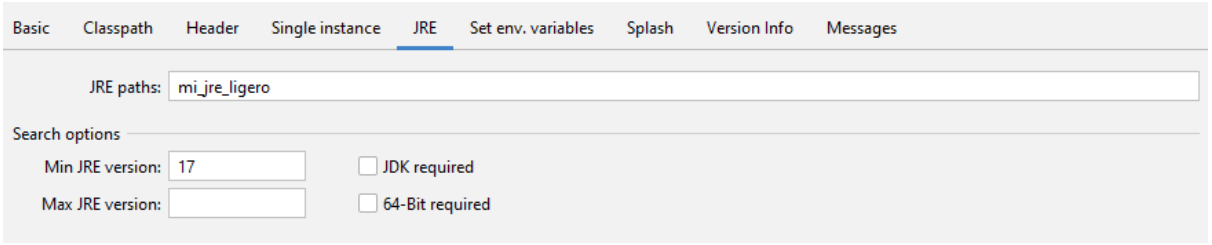
```
PS C:\Users\Usuario1\IdeaProjects\DI05_TunelTerror> jlink --module-path "Ruta_
JDK;Ruta_JavaFX" --add-modules java.base,java.desktop,javafx.controls,javafx.f
xml,javafx.graphics --output mi_jre_ligero
```

2.3. Empaquetado Ejecutable (Launch4j)

Se utilizó la herramienta Launch4j para envolver el JAR en un contenedor .exe nativo de Windows.

- **Portabilidad:** Se configuró la ruta del JRE como relativa (mi_jre_ligero), lo que permite que la carpeta del programa se pueda mover a cualquier equipo sin romper la ejecución.
- **Experiencia de Usuario:** Se configuró el modo "GUI" para suprimir la consola de comandos y se añadió un icono personalizado (.ico) para mejorar la identidad visual.

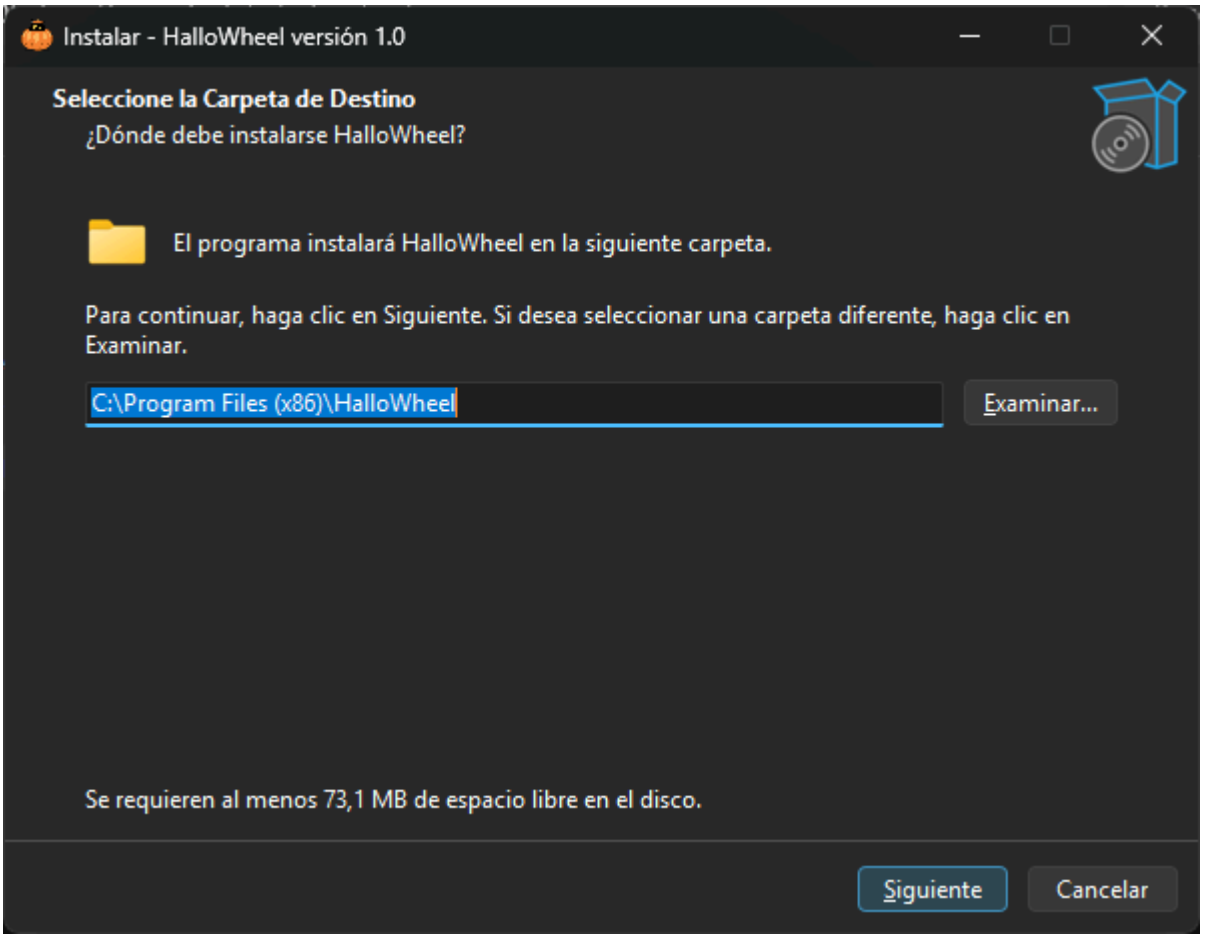




2.4. Creación del Instalador (Inno Setup)

Finalmente, se generó un asistente de instalación con Inno Setup. El script se configuró para:

1. Copiar el ejecutable y la carpeta del JRE (incluyendo subdirectorios) a Program Files.
2. Crear accesos directos en el Escritorio y Menú Inicio.
3. Incluir un desinstalador automático.
4. Personalizar las imágenes del asistente para dar una imagen corporativa.



3. Problemas encontrados y soluciones

Durante el ciclo de desarrollo y distribución de la aplicación surgieron diversos desafíos técnicos que fueron analizados y resueltos para garantizar un producto final robusto:

1. Ejecución del JAR: "JavaFX runtime components are missing"

- **Problema:** Tras generar el "Fat JAR" con Maven, al intentar ejecutarlo mediante el comando java -jar, la aplicación fallaba inmediatamente mostrando el error: "Error: JavaFX runtime components are missing, and are required to run this application".
- **Solución:** Se implementó el patrón "Launcher". Se creó una clase auxiliar (Launcher.java) que no hereda de ninguna clase de JavaFX y cuyo único propósito es invocar al método main de la aplicación real. Al configurar esta clase como el punto de entrada en el pom.xml, se engaña a la JVM para que cargue las dependencias en el orden correcto.

2. Reconocimiento del comando jlink en el entorno

- **Problema:** Al intentar generar el JRE personalizado, la terminal de Windows devolvía el error de que jlink no se reconocía como un comando interno. Esto se debía a que las variables de entorno del sistema (PATH) no apuntaban a la instalación del JDK más reciente.
- **Solución:** Se optó por ejecutar el comando utilizando la ruta absoluta al binario del JDK ("C:\Program Files\Java\jdk-25\bin\jlink.exe"), lo que permitió realizar la operación sin necesidad de modificar permanentemente la configuración del sistema operativo del equipo de desarrollo.

3. Gestión de Versiones (JDK vs JavaFX)

- **Problema:** El entorno de desarrollo contaba con una versión del JDK muy reciente (v25) mientras que los módulos de JavaFX descargados correspondían a la versión LTS (v21). Existía el riesgo de incompatibilidad al enlazar los módulos.
- **Solución:** Se verificó la compatibilidad y se configuró cuidadosamente el module-path en el comando jlink para que apuntara simultáneamente a los módulos internos del JDK 25 y a los módulos externos del SDK de JavaFX 21, logrando un runtime híbrido funcional y estable.

4. Conclusión

El resultado final es un instalador (HalloWheel_installer.exe) profesional. La aplicación cumple con el requisito de portabilidad total, ejecutándose en entornos Windows limpios sin necesidad de descargas adicionales por parte del usuario final.