# CREACIÓN DE MODS EN MINECRAFT

## Java y Forge

Digitalización aplicada a los sectores productivos

Este documento recoge el proceso de creación de modificaciones (mods) para el videojuego Minecraft, utilizando el lenguaje de programación Java y el entorno Forge.
se explican los pasos necesarios para configurar el entorno de desarrollo, escribir código, integrar recursos gráficos y probar el mod en el juego.

Javier López Gutiérrez

# Índice:

# 1. Introducción:

## ¿Qué es Minecraft?

Minecraft es un videojuego tipo sandbox que permite a los jugadores explorar, construir y modificar un mundo abierto compuesto por bloques. Desde su lanzamiento, ha logrado una gran popularidad no solo por su estilo visual y libertad creativa, sino también por su gran comunidad y la posibilidad de personalizar el juego mediante modificaciones, conocidas como *mods*.

## ¿Qué son los Mods?

Los *mods* son extensiones del juego que permiten añadir nuevos bloques, objetos, funciones, criaturas, o incluso cambiar por completo la mecánica del juego. Se desarrollan principalmente utilizando el lenguaje de programación Java y herramientas como Forge, que facilita la integración de los cambios en el entorno de Minecraft.

## ¿Por qué se debería aprender a crear Mods?

Aprender a crear *mods* no solo resulta útil para entender mejor cómo funciona internamente un videojuego tan complejo como Minecraft, sino que también representa una oportunidad práctica para aplicar conceptos de programación en un entorno creativo y motivador. A lo largo de este proyecto he podido profundizar en el uso de Java, familiarizarme con estructuras de un proyecto real y comprender la importancia de seguir una lógica clara a la hora de desarrollar funcionalidades. Crear un mod es un proceso que exige organización, atención al detalle y una buena base técnica, por lo que ha sido una experiencia muy enriquecedora tanto a nivel personal como académico.

# 2. Herramientas necesarias:

## Java JDK (Versión 17):

El JDK (Java Development Kit) es el conjunto de herramientas necesario para programar en Java. Incluye el compilador y otras utilidades para desarrollar y ejecutar aplicaciones. Minecraft y Forge están desarrollados en Java, por lo que es imprescindible tener una versión compatible. En este proyecto he utilizado Java 17, ya que es la versión requerida por Forge 1.19.2. Instalarlo correctamente fue uno de los primeros pasos, y también aprendí a configurar la variable de entorno `JAVA_HOME`.

## Eclipse:

Eclipse es un entorno de desarrollo integrado (IDE) que permite programar en Java de forma más organizada. Incluye herramientas para escribir, compilar y depurar el código. Inicialmente se ha usado junto con Java 23 para poder practicar y entender las estructuras de proyectos.

## Visual Studio Code:

Aunque Eclipse fue el IDE principal, también utilicé Visual Studio Code en algunas ocasiones cuando se tenía que trabajar con java 17, especialmente para editar archivos de configuración como `build.gradle` o los archivos JSON del mod. VS Code es un editor más ligero que permite trabajar cómodamente con distintos tipos de archivos gracias a sus extensiones.

## Minecraft Forge (Versión 1.19.2):

Forge es una de las herramientas más importantes del proyecto. Se trata de un conjunto de bibliotecas y una API que permite crear mods compatibles con Minecraft. En este caso, utilicé Forge 1.19.2, una versión estable y actual. Gracias a Forge, pude acceder a eventos del juego, registrar nuevos bloques e ítems, y conectarme con las funciones internas de Minecraft de forma más segura y controlada.

## Gradle (Versión 7.6):

Gradle es una herramienta de automatización de proyectos que se encarga de compilar el código, gestionar dependencias y construir el proyecto. En el contexto de los mods, Gradle se usa para preparar el entorno, descargar las librerías necesarias y compilar el mod. Al principio desconocía completamente esta herramienta, pero ahora sé cómo usar los comandos básicos como `gradlew genEclipseRuns` o `gradlew build`.

Estas herramientas forman la base de todo el proyecto, y aprender a configurarlas correctamente ha sido una parte fundamental del proceso. Aunque al principio resultaba confuso debido a la correcta utilización de todas las herramientas con su debida versión, con cada error fui entendiendo cómo se relacionan entre sí y cómo cada una cumple un rol importante en el desarrollo del mod.

# 3. Preparación del Entorno de Desarrollo:

Uno de los pasos más importantes del proyecto ha sido la preparación del entorno de desarrollo. Al tratarse de un proyecto de programación en Java con múltiples herramientas involucradas, era necesario seguir una serie de pasos para poder empezar a trabajar con Forge de forma correcta.

## Configuración inicial del proyecto con Forge:

Para comenzar, descargué el *MDK* (Mod Development Kit) de Forge en su versión 1.19.2 desde su página oficial. Este paquete contiene los archivos necesarios para empezar un proyecto de modding. Tras descomprimirlo, abrí una terminal en la carpeta del proyecto y ejecuté el comando:

*gradlew genEclipseRuns*

Este comando genera los archivos de configuración específicos para Eclipse, incluyendo los *launch configurations* necesarios para ejecutar Minecraft directamente desde el entorno. Luego, ejecuté:

*gradlew eclipse*

## Estructura del proyecto:

Una vez cargado el proyecto en Eclipse, encontré varias carpetas y archivos organizados de forma específica. Esta estructura tiene un propósito claro dentro del desarrollo de mods, y aprendí a ubicar cada elemento según su función:

- *src/main/java:* aquí se encuentra el código fuente del mod. Todas las clases Java que definen bloques, ítems, eventos o lógica personalizada se guardan en esta carpeta. Aprendí a organizar el código en *paquetes* (packages) para mantenerlo limpio y estructurado.
- *src/main/resources:* contiene todos los recursos del mod, como archivos de configuración (.toml), archivos de idioma (.json), modelos de los ítems o bloques, texturas y demás archivos necesarios para que Minecraft entienda cómo mostrar e interpretar el contenido añadido.
- *build.gradle:* este archivo es esencial para definir las dependencias del proyecto, el nombre del mod, la versión, el grupo, y otra configuración técnica que Gradle utiliza para compilar el mod.
- *settings.gradle:* especifica el nombre del proyecto.
- gradlew y gradlew.bat: son los *wrappers* de Gradle que permiten ejecutar comandos sin necesidad de tener Gradle instalado de forma global en el sistema.

# 4. Creación de un Mod:

En la creación de mods para Minecraft, uno de los primeros conceptos fundamentales que aprendí fue el uso de la anotación `@Mod`. Esta anotación es esencial, ya que marca la clase principal del mod y le indica a Forge cómo inicializar y cargar el mod dentro de Minecraft.

El uso de la anotación *@Mod*

La anotación *@Mod* permite definir el nombre y la versión del mod, entre otras propiedades, lo que facilita la identificación y gestión del mod en el entorno de Forge. Además, se puede utilizar para registrar eventos que deben ser ejecutados durante las fases de carga del juego, como la inicialización.

```java
package com.pocioncaidasuave;

import com.mojang.logging.LogUtils;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.eventbus.api.IEventBus;
import
net.minecraftforge.fml.javafmlmod.FMLJavaModLoadingContext;
import org.slf4j.Logger;

@Mod(MiMod.MOD_ID)
public class MiMod {
    public static final String MOD_ID = "pocioncaidasuave";
    public static final Logger LOGGER =
LogUtils.getLogger();

    public MiMod() {
        IEventBus modEventBus =
FMLJavaModLoadingContext.get().getModEventBus();
        ModItems.register(modEventBus);
        ModEffects.register(modEventBus);
    }
}
```

Este archivo es la clase principal, la cual se marca con la anotación *@Mod*. El Mod se llama *"pocioncaidasuave",* el código define su identificador (`MOD_ID`) como una constante. Este identificador es muy importante porque le permite a Forge gestionar los mod y asociar recursos como texturas, sonidos, y configuraciones a los mod durante la carga del juego.

# 5. Contenido en el mod

Durante la realización del mod, he trabajado con varias clases Java distribuidas en la estructura típica de un proyecto Forge, que he ido comprendiendo progresivamente a medida que desarrollaba el proyecto.

## Estructura del directorio:

El proyecto se encuentra ubicado en la siguiente ruta principal:

*src/main/java/com/pocioncaidasuave*

En Visual Studio Code, esta estructura no siempre se muestra como una carpeta ramificada en árbol de forma visual (por ejemplo, *src > main > java > com > pocioncaidasuave*).

En cambio, aparece como una única carpeta con el nombre completo del paquete, es decir, *com.pocioncaidasuave.* Esto se debe a cómo el propio VS Code interpreta y muestra los paquetes de Java dentro de la carpeta *java*.

Sin embargo, internamente, sí están organizados como subcarpetas que respetan la convención del nombre del paquete.

## Clases y su funcionalidad:

A continuación, explico brevemente el propósito de cada clase incluida en mi mod y la relación que tienen entre ellas:

MiMod.java: Es la clase principal del mod. En ella se inicializa el mod y se registran los distintos elementos, como los ítems *(ModItems)* y los efectos *(ModEffects).* Contiene el @Mod, que indica que esta clase es el punto de entrada del mod.

ModEffects.java: Se encarga de registrar los efectos personalizados del mod utilizando *DeferredRegister.* En este caso, registra el nuevo efecto "absorción de caída" con una categoría beneficiosa y un color específico. Este efecto es el corazón funcional del mod.

PotionBase.java: Define el comportamiento concreto del efecto "absorción de caída". Extiende la clase *MobEffect* y sobreescribe métodos como *applyEffectTick()* para anular la distancia de caída del jugador, evitando que reciba daño. También se asegura de que el efecto se aplique en cada *tick* del juego mientras dure la poción.

ModItems.java: Aquí se registra la poción como un nuevo ítem dentro del juego. Se define el ítem con sus propiedades básicas y se relaciona con el efecto previamente definido. Es decir, gracias a esta clase, la poción está disponible en el inventario del jugador y se puede usar.

ModRecipe.java: Aunque en este proyecto básico aún no he profundizado en la lógica completa de recetas, esta clase está preparada para incluir y registrar las recetas de creación de la poción en el juego, como por ejemplo utilizando una mesa de pociones o ingredientes personalizados.

## Relación entre las clases:

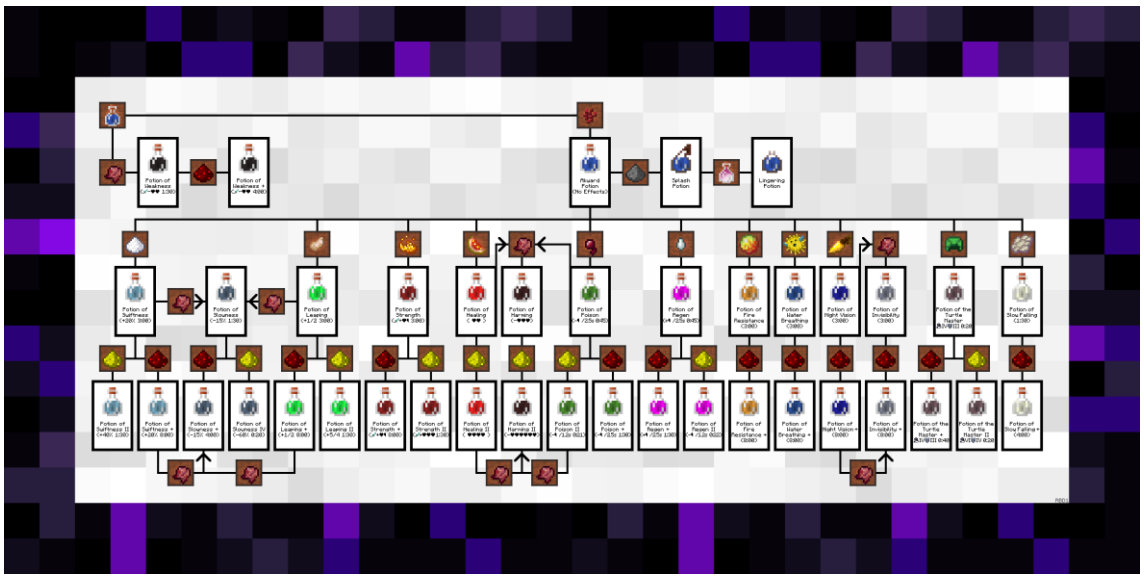
La clase MiMod es el punto de entrada y de coordinación. Se apoya en las demás clases para registrar y activar los contenidos del mod.

ModEffects y PotionBase trabajan juntas: una registra el efecto y la otra define su comportamiento.

ModItems añade al juego el ítem físico que el jugador puede usar, y al consumirlo, activa el efecto de la PotionBase.

ModRecipe está pensada para integrar futuras mecánicas de creación, haciendo el mod más completo.

Este análisis me ha permitido no solo comprender mejor cómo organizar un proyecto Java, sino también cómo interactúan las distintas clases y cómo se puede estructurar una aplicación modular.



# 6. Problemas comunes y cómo solucionarlos:

Durante el proceso de desarrollo de mods para Minecraft utilizando Forge, he encontrado diversos problemas comunes que se presentan, especialmente cuando se trabaja con versiones de Java, Forge, y Gradle. A continuación, describiré los problemas más frecuentes que encontré, cómo los solucioné y las buenas prácticas que he aprendido para evitarlos.

## Errores Por mal uso de versiones:

Uno de los problemas más frecuentes en el desarrollo del proyecto de Mods para Minecraft es la incompatibilidad de versiones entre Forge**,** Java**,** y Gradle**.** Al principio, me encontré con varios errores al intentar compilar el mod, principalmente debido a diferencias entre las versiones de Forge y la versión de Java que estaba utilizando.

En la asignatura de programación utilizamos Java 23, pero al investigar más sobre el desarrollo de mods con Forge 1.19.2, descubrí que la versión más compatible para este entorno es Java 17. Esto se debe a que las versiones más recientes de Forge suelen ser más estables con versiones específicas de Java y Gradle.

**Para solucionarlo:**

- **Revisé la documentación de Forge** para asegurarme de que la versión de Java que estaba utilizando fuera compatible con la versión de Forge que quería usar.
- En mi caso, para Forge 1.19.2, se recomienda Java 17. Por lo tanto, decidí usar esa versión para evitar errores de incompatibilidad.
- **Revisé la configuración de Gradle**. En un principio instalé la última versión, tras ir probando otras versiones Gradle 7.6**,** resultó ser la versión más adecuada para este entorno, ya que las versiones más nuevas de Gradle a veces no son totalmente compatibles con versiones anteriores de Forge.

**Crash por mal registro:**

Otro error común fue el **crash del juego** debido a un **mal registro de ítems, bloques o efectos**. Esto sucedió cuando intentaba registrar un ítem o efecto, pero no lo hacía correctamente, lo que causaba un **crash** al iniciar el juego. Por ejemplo, olvidé registrar ciertos objetos o efectos dentro del **event bus** de Forge.

Para solucionarlo:

- Me aseguré de **registrar todos los ítems, bloques, y efectos correctamente** en el **evento de registro** (`IEventBus`), como se muestra en los ejemplos de código de los tutoriales.

- Verifiqué que todos los métodos de registro (como `ModItems.register()` o `ModEffects.register()`) fueran llamados en el momento adecuado, específicamente cuando se cargaba el mod.

**Buenas prácticas para evitar errores:**

A medida que fui avanzando en el desarrollo de mi mod, fui aprendiendo varias buenas prácticas que me ayudaron a evitar los errores más comunes y hacer que el desarrollo fuera más fluido:

1. **Mantener una versión compatible de Java y Forge:** Como mencioné anteriormente, la elección de **Java 17** y **Forge 1.19.2** me permitió evitar problemas de compatibilidad. Siempre es recomendable trabajar con versiones de herramientas que estén comprobadas como estables y compatibles entre sí.
2. **Seguir tutoriales y documentación oficial:** Durante los primeros meses, seguí tutoriales y documentación de **Forge** para asegurarme de que estaba haciendo todo correctamente. Esto me permitió entender bien cómo debía registrar los elementos y efectos, y cómo manejar las rutas de recursos.
3. **Realizar pruebas frecuentes:** Para evitar complicaciones, probé el mod con frecuencia. Al realizar cambios, probaba el mod en el juego inmediatamente para asegurarme de que los ítems y efectos se registraran correctamente y que no se produjeran fallos o crashes.
4. **Revisar los logs de errores:** Si el mod se bloqueaba o no funcionaba correctamente, revisaba los **logs de errores** generados por Minecraft para identificar el origen del problema. Los logs a menudo ofrecen información precisa sobre qué causó el error y cómo solucionarlo.
5. **Usar la estructura de carpetas correcta:** Asegurarse de que los archivos estuvieran bien organizados en las carpetas `src/main/java` y `src/main/resources` fue fundamental para evitar problemas de carga de recursos.

# 7. Impacto y aplicaciones:

Aprender a desarrollar mods para Minecraft me ha resultado una experiencia útil desde el punto de vista académico. Aunque pueda parecer un entorno de juego, el modding de Minecraft representa una introducción práctica al desarrollo de software real.

Por un lado, saber hacer mods me ha permitido poner en práctica conceptos fundamentales de programación orientada a objetos, como la creación de clases, métodos, herencia, y encapsulamiento. Además, trabajar con eventos, registros y estructuras de paquetes me ha ayudado a entender cómo se organizan y gestionan los proyectos.

Además, he aprendido a trabajar con herramientas del entorno profesional como Eclipse, Gradle, y el uso de librerías externas (como Forge), lo cual me ha acercado al funcionamiento real del desarrollo de software. También he tenido que enfrentarme a problemas de versiones, compatibilidad y estructura de proyecto, lo que me ha ayudado a desarrollar habilidades de resolución de errores y lectura de documentación técnica.

Desde una perspectiva educativa y profesional, este tipo de proyectos pueden ser un punto de partida para explorar campos como el desarrollo de videojuegos, la programación en Java avanzada. Además, puede servir como una base para aprender sobre sistemas de compilación, uso de control de versiones (como Git) y colaboración en proyectos más grandes, habilidades muy valoradas en el mundo laboral.

# 8. Conclusión:

Realizar este proyecto me ha servido no solo para descubrir el mundo del desarrollo de mods para Minecraft, sino también para afianzar y ampliar mis conocimientos en Java y en programación en general. A lo largo de los tres trimestres, he podido progresar desde la fase de exploración y aprendizaje, hasta llegar a desarrollar un mod funcional.

He aprendido a trabajar con herramientas y tecnologías que no habíamos utilizado directamente en clase, como Forge, Gradle y la gestión de recursos dentro de un entorno de desarrollo real. También he aprendido a leer documentación técnica, adaptar ejemplos y resolver errores por mi cuenta, lo cual considero uno de los aprendizajes más valiosos del proyecto.

Si bien mi conocimiento de Java sigue siendo básico, este trabajo me ha motivado a seguir practicando y explorando nuevas formas de aplicar lo aprendido en programación a proyectos más creativos y personales. Me gustaría seguir profundizando en el estudio de java y, si es posible, conectar más adelante estos conocimientos con otras asignaturas del ciclo.

En definitiva, este proyecto ha sido una forma de complementar el estudio y repaso de Java, enfrentándome a un reto real que me ha permitido crecer como estudiante y como futuro desarrollador.

# 9. Bibliografía y recursos:

Durante el desarrollo de este proyecto, he consultado diversas fuentes para comprender mejor tanto el lenguaje Java como la estructura y funcionamiento de los mods en Minecraft utilizando Forge. A continuación, se listan los principales recursos utilizados:

**Documentación oficial y herramientas**

Forge Documentación (1.19.2) – Guía y documentación oficial para desarrollar mods con Forge.

Gradle Documentación (v7.6) – Información sobre el sistema de automatización de compilación usado en el proyecto.

Minecraft Wiki – Información general sobre el juego y elementos del entorno como efectos de pociones.

Java SE Development Kit 17 – Entorno de desarrollo necesario para trabajar con Forge 1.19.2.

Vídeos y tutoriales Series de tutoriales en YouTube sobre modding en Forge 1.19.2.

Foros y comunidades de desarrollo, como Stack Overflow y los foros de Forge, que fueron clave para resolver errores específicos.

**Herramientas utilizadas**

Eclipse IDE for Java Developers – Entorno de desarrollo donde se ha programado el mod.

Visual Studio Code – Editor de texto usado para visualizar y editar archivos del proyecto.

Minecraft Launcher – Para ejecutar y probar el mod en el entorno real del juego.

Blockbench – Herramienta opcional explorada para modelado 3D de elementos en Minecraft (aunque no se ha implementado en este proyecto).