

FABRIC 1.20 / JDK 21

LESSON 01

Run your first Fabric mod

Open the project, enter a world, and find your own startup message.



The real test world: grass-block steps created by the template when the player joins.

WHAT YOU WILL MAKE

Run the supplied template and add a startup message that you verify in a real game launch.

90-110 minutes | [simon-minecraft-template-1.20](#)

Get your workspace ready today. In lesson 2, you will register a new block of your own.

Before you start: Comfortable with a keyboard, mouse and folders

teacher-ma.de/minecraft-mod-lesson.php?lesson=1&lang=en

01 / 11

Run it first. Then change one line.

You do not need to memorize Java or build a mod project from scratch.

Use the project template supplied by your teacher. First, run it unchanged to establish a working baseline. Then add just one log message to check that your Java code really runs. Keep the mod ID, block code and dependency versions unchanged.

IntelliJ IDEA	Edit and locate files and read the Run console. The IDE is separate from the Minecraft game.
JDK 21	Provides the Java development and runtime tools. Use the JDK required by this template; installing only a JRE is not enough.
Gradle Wrapper	Uses the build-tool configuration bundled with the project to download dependencies, compile the code and launch the development client.
Fabric	Fabric Loader discovers and loads mods. Fabric API provides interfaces used when developing mods.

1. Open the project / 2. Check the JDK / 3. Run it unchanged / 4. Find the log / 5. Edit and test again

Time and evidence

Allow about 90-110 minutes for the lesson, plus the initial downloads. To finish, show the original entrypoint log, a working test world, and Lesson 1 ready! after restarting.

Predict before you start: does opening the game prove that the line of Java you changed has run?

02 / 11

Open the whole project

Extract the ZIP into a normal folder before opening it.

Example template project root

```
C:\TeacherMa\Mod\simon-minecraft-template-1.20
```

In IntelliJ, choose File → Open and select the project folder. Your folder may be elsewhere, but its root must contain build.gradle and gradlew.bat. Opening only a .java file does not load the complete Gradle project.

First time opening it	Let IntelliJ load the Gradle project. Wait for dependency downloads and synchronization to finish.
Returning to the project	Open the same project from Recent Projects. Do not create a second, empty project with New Project.
Signs that sync worked	The Gradle tool window and project tasks appear, and widespread red errors in the source disappear.
Downloads still running	The first sync needs an internet connection. Download time depends on the network. If it fails, read the specific error instead of repeatedly clicking Run.

Folder map: edit src, inspect run, leave build alone

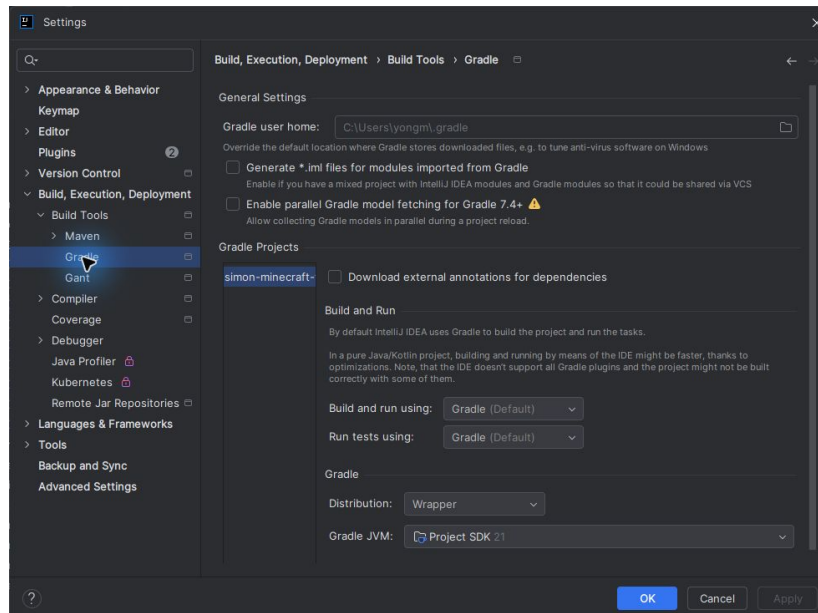
```
project root/  
  build.gradle, gradle.properties, gradlew.bat  
  src/main/java/           Java source  
  src/main/resources/      JSON and textures  
  run/                     game data and logs  
  build/                   generated output
```

Edit the source and resources under src. Files under build are generated output, not your original source files.

03 / 11

Check the template's JDK 21 settings

Runtime requirements and compilation targets are different. Check both settings.



Actual settings: Distribution uses Wrapper; Gradle JVM points to Project SDK 21.

Project SDK

Open File → Project Structure → Project and choose an installed JDK 21 as the SDK.

Gradle JVM

Open Settings → Build, Execution, Deployment → Build Tools → Gradle. Choose JDK 21, or a Project SDK that points to 21. Keep Distribution set to Wrapper.

Keep the template's versions

In this supplied template, fabric.mod.json declares java >=21. The options.release=17 setting in build.gradle is a compilation target; it does not mean this template can launch on Java 17. Keep the template versions and configure the runtime tools to use JDK 21.

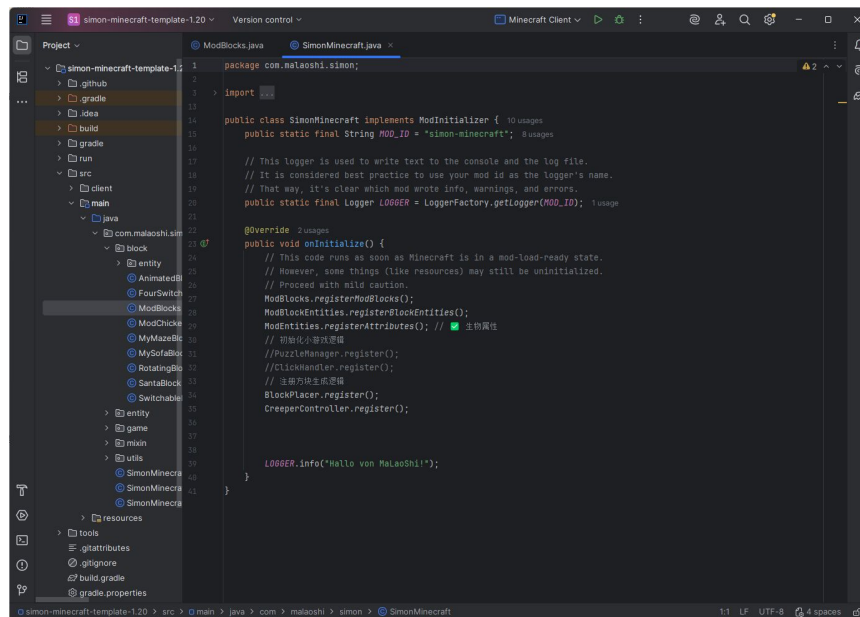
04 / 11

Find the Java entrypoint

Look first. Do not replace the entire class.

Open this file

```
src/main/java/com/malaoshi/simon/SimonMinecraft.java
```



Actual IntelliJ screenshot: the SimonMinecraft entrypoint used in this lesson.

Two existing lines in different places: locate them only

```
public static final String MOD_ID = "simon-minecraft";
LOGGER.info("Hallo von MaLaoShi!");
```

These lines are in different parts of the file. MOD_ID identifies the mod; LOGGER.info writes a log message. Find the braces of onInitialize() and locate the existing startup message near the bottom of that method.

Keep the existing template behavior

Calls involving ModBlocks, ModBlockEntities and BlockPlacer already exist. You do not need to understand every detail yet. Keep all registration and event calls. Do not overwrite the class or the whole onInitialize() method with this excerpt.

05 / 11

How Fabric finds the entrypoint

Open `src/main/resources/fabric.mod.json`. Read it without changing it.

Reading excerpt from `fabric.mod.json`, not the complete file

```
{
  "id": "simon-minecraft",
  "environment": "*",
  "entrypoints": {
    "main": [
      "com.malaoshi.simon.SimonMinecraft"
    ],
    "client": [
      "com.malaoshi.simon.SimonMinecraftClient"
    ]
  }
}
```

This is an excerpt for reading, not a complete configuration file. Keep the original `schemaVersion`, `version`, `depends`, `mixins` and all other fields. Do not edit the JSON in this lesson.

main	Points to the common entrypoint implementing <code>ModInitializer</code> . This also loads when the client starts.
client	Points to a client-only entrypoint implementing <code>ClientModInitializer</code> , for features such as rendering. It is separate from the common entrypoint.
Fully qualified class name	The package <code>com.malaoshi.simon</code> and class name <code>SimonMinecraft</code> combine to form the full class name in <code>main</code> .

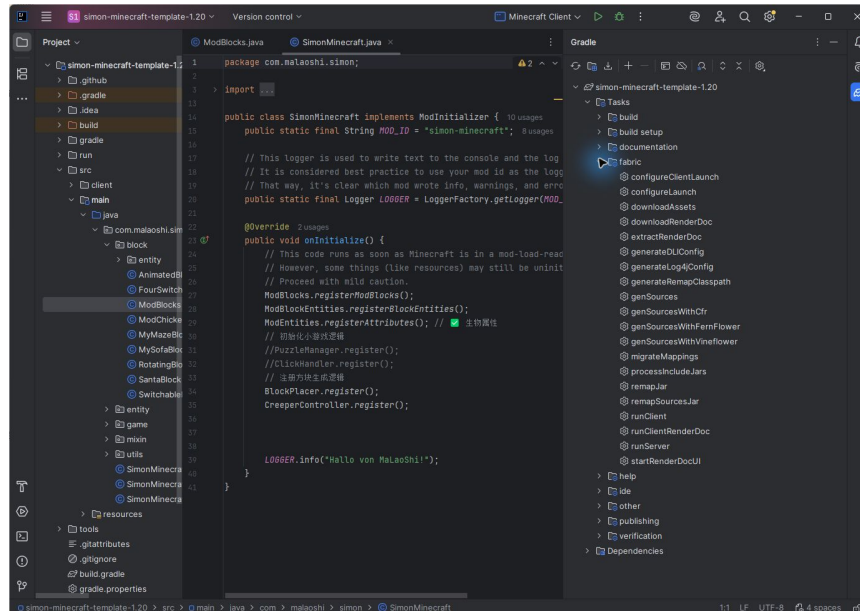
Check the namespace; keep it unchanged

The JSON `id` and Java `MOD_ID` are both `simon-minecraft`. Resources also use this namespace. Changing just one location can leave entrypoints, registrations and resources inconsistent.

06 / 11

Launch runClient

Gradle → project name → Tasks → fabric → runClient



Actual IntelliJ Gradle panel: find runClient in the fabric group.

Double-click runClient, wait for compilation and resource loading, then switch to the Minecraft window. The task continues running while the game is open. This is normal; do not repeatedly launch extra clients.

Alternative: PowerShell in the project root

```
.\gradlew.bat runClient
```

Choose one launch method

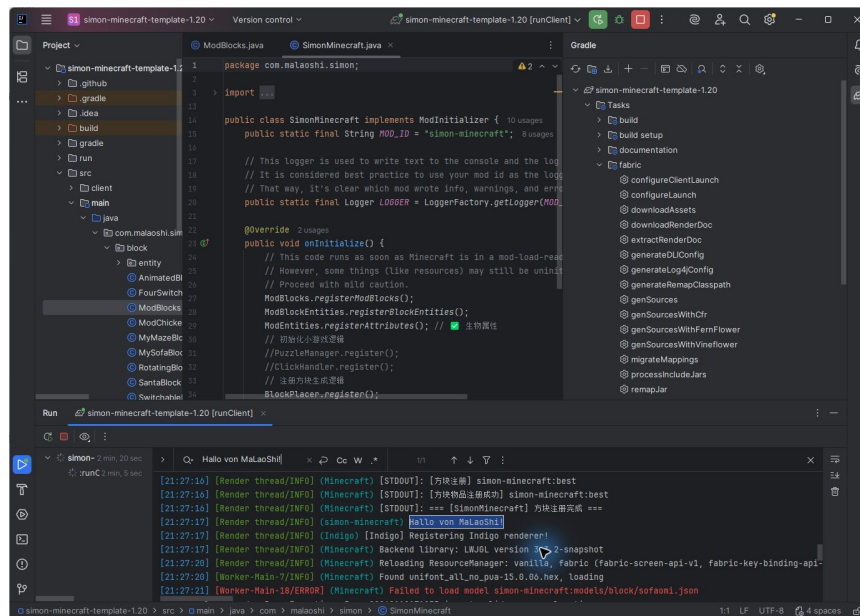
Run the command above only from the project root. You can also select an existing Minecraft Client run configuration in the IDE and click Run. Do not run SimonMinecraft.java as an ordinary Java main program.

07 / 11

Find the original startup message

Open the Run console at the bottom and use Ctrl+F to search for the full message.

Search the console for Hallo von MaLaoShi!. Find the line from simon-minecraft before entering a world. If output is scrolling quickly, use search rather than scanning by eye.



Actual Run console: the original startup message has been found.

Actual log content, with timestamp and thread prefix omitted

(simon-minecraft) Hallo von MaLaoShi!

INFO

An information-level message, not an error. Log lines may also include a timestamp, thread name and source.

simon-minecraft

The logger name, which helps identify which mod produced the message.

What the log proves

It proves execution reached this line. It does not prove that every block model or texture is complete.

Where are the logs?

Read the IDE Run console while the game is running, or open run/logs/latest.log in the project. `LOGGER.info` does not automatically send a message to in-game chat.

08 / 11

Enter a separate test world

Use Creative mode, cheats enabled, Peaceful difficulty and a Superflat world.



Actual gameplay: a new lesson world entered after launching from IntelliJ.

Create

Choose Singleplayer → Create New World and name it Lesson01Test. Keep experiments separate from important existing saves.

Observe

Move with W/A/S/D, look with the mouse and jump with Space. This template automatically creates grass-block steps when a player joins.

Record

Record the world name and screenshot location. F2 saves screenshots to run/screenshots. These are actual game captures, not 3D illustrations.

The steps come from the template

The steps are generated by the existing BlockPlacer in the template, not by code added today. Entering the world and finding the log completes the baseline check. Record and fix any missing experimental models separately.

09 / 11

Add your own startup message

Exit the game normally, then return to SimonMinecraft.java.

Inside the existing onInitialize(): keep the old line and add the new one

```
LOGGER.info("Hallo von MaLaoShi!");  
LOGGER.info("Lesson 1 ready!");
```

Keep the first line. Add the second line below it, before the closing brace of onInitialize(). Do not create another class, add imports or delete other registration calls.

LOGGER.info(...)	Calls the info method on the template's existing logger and writes the text in parentheses to the log.
"Lesson 1 ready!"	The text inside double quotes is a string. Use straight ASCII double quotes, not typographic curly quotes.
Semicolon ;	Ends this Java statement. A missing semicolon causes a compilation error, usually highlighted in red.
Case and placement	Use uppercase LOGGER and lowercase info. Put the new line inside the method braces, not outside the class.

Predict, then launch

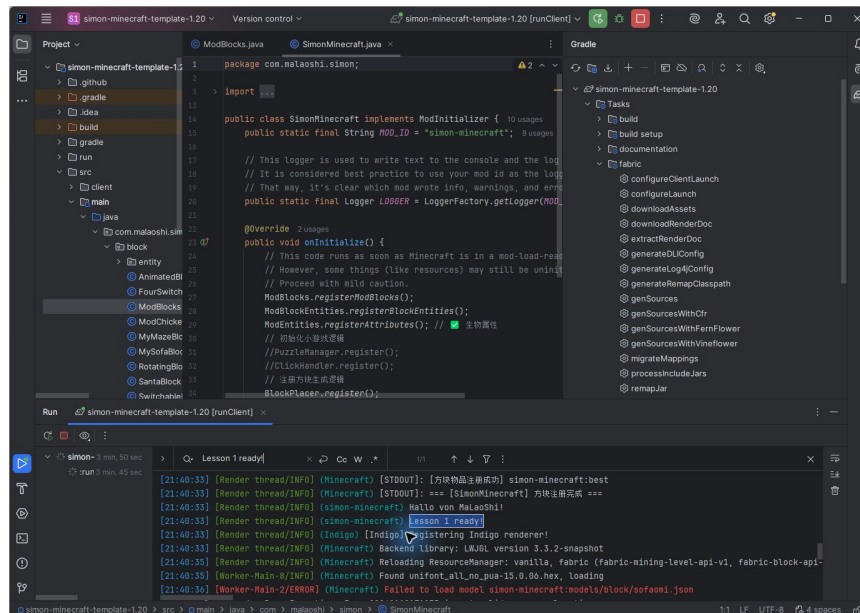
This line changes only the startup log. It does not change a block's appearance, create a new block or send chat text. Save the file, run runClient again and search for Lesson 1 ready!.

Try it: change the string to your own English startup message. Keep the method call, quotes and semicolon intact.

10 / 11

Restart and verify the change

After the second launch, search for Lesson 1 ready!



Actual console from the second launch: the newly added Java message has run.

Actual second-launch log, with the timestamp prefix omitted

(simon-minecraft) Lesson 1 ready!

After editing Java

Close the old client, save the source and run `runClient` again to recompile and load the change.

F3+T

Reloads client resources. It does not recompile Java or rerun the entrypoint.

Leaving and rejoining a world

Usually keeps the same client process running. It is not the same as restarting the mod entrypoint.

runClient and build test different things

`runClient` checks the launch and gameplay. `build` checks compilation, checks and artifact generation. After exiting the game, run `.\gradlew.bat build` from the project root and look for **BUILD SUCCESSFUL**. A successful build does not replace testing in the game.

11 / 11

Troubleshooting and completion checklist

Locate the stage that failed. Avoid changing several versions or files at once.

No Gradle tasks	Check that you opened the root folder containing build.gradle. Check the Gradle sync result.
Dependency download fails	Read the specific network or repository error. Restore connectivity and sync again; do not start by upgrading every dependency.
Wrong Java version	Check Project SDK and Gradle JVM. Command-line launches also need a JDK that matches the template requirements.
New message missing	Check that you edited onInitialize() in this project. Close the old client, save and restart.
Compilation error or crash	Start with errors that show a source line number. Follow the exception chain to the relevant Caused by message and keep the full log.
A model is missing	The original handout's screenshots show a pre-existing missing sofaomi model; that run could still enter the world. Investigate other ERROR messages on their own merits.

Complete these five checks before moving on

1. Find the project root and src.
2. Explain how main points to the entrypoint class.
3. Find the original startup message.
4. Enter a separate test world.
5. Find your own message after restarting, and explain the difference between restarting Java and reloading resources.

Next lesson: reuse this project and launch method to register a separate starter_block. Give it a working inventory icon and a Chinese display name, then obtain and place it.

Official references

Use these official references for concepts and settings. This lesson follows the versions in the supplied template.

Fabric: Project structure and entrypoints

Fabric: fabric.mod.json

JetBrains: Gradle JVM selection