

JAVA / LESSON 40 OF 40

Release Your Block Legend

Give a playable work to the person who uses it for the first time

Environment: JDK 21, ordinary Paper 1.21.11, client of the same version. Server plugins are loaded by Paper; only explicitly labeled rule tests use main. | 120-150 minutes



Graduation packs with versions, installation instructions and test evidence

The starting point for publishing is that another person can install it according to the instructions. Organize the version, JAR, source code, configuration and test records, practice updates and rollbacks, and then spend five minutes explaining the rules and known limitations of the work.

By default, OreRush continues in Lessons 38 and 39. You can also select learned tasks, pets, or tool items.

The release of this lesson refers to the local work package handed over to test partners, and does not require public upload.

Before you start: Continue with the tested projects from Lessons 38-39

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=40&lang=en>

Long code lines may wrap at the > margin marker. The marker is not part of the code.

In this lesson

01. Before starting: first recognize the running position
02. Choose a work that can explain clearly
03. Separate what must be done from what must be done later
04. Put the code where you can find it
05. Give an accurate version of the release package
06. Configuration is not a score archive
07. Let the instructions answer the first round of questions for you
08. Release records and write changes without writing magic
09. Use clean test clothes to accept the installation
10. If the update fails, you need to know how to go back.
11. five-minute demonstration, let students play first
12. The graduation score depends on the evidence and the explanation.
13. Expansion challenges and own acceptance records
14. The next adventure of the graduation work

01 / 14

Before starting: first recognize the running position

The first 30 lessons lay the foundation for ordinary Java programs. At this stage, the code is connected to the Paper server: the Minecraft client is responsible for entering the world, the server loads the plug-in, and the events and commands hand over player actions to your methods.

This book uses JDK 21 and Paper 1.21.11, and the client version is also consistent. The latest version on the download page may have changed, select it by version number. This book uses normal plugin.yml and the Bukkit API; no claims are made to support Folia at this stage.

The short examples in the picture frame are partial fragments. Read the location description of this section first, and do not save each as a plug-in. The file name will be clearly given for the complete file; when the same file is divided into multiple segments, they will be connected by number, and the package name, import and outer class will only appear once.

- Prediction: Write success conditions and expected messages first.
- Run: After successful construction, install and restart on the dedicated local test server.
- Verification: Ordinary players, management identities and consoles take corresponding paths respectively.

The scene illustrations help to understand the story; the illustrations explain the real rules and are not used as IDE or game actual measurement screenshots. After making Java modifications, rebuild, stop the server, replace it with the new JAR, and start it again; do not treat the performance of the old JAR as the result of the new code.

02 / 14

Choose a work that can explain clearly

A Fang threw a JAR to his deskmate: "Graduation!" The deskmate asked about the version, command, configuration and installation location, so Afang had to put away the celebratory fireworks first. A release is a complete delivery that others can use.

The release package puts the program together with instructions for use: supported versions, installation methods, command permissions, configuration, testing, and known limitations. Use a clean test server to install it according to your own instructions. You can find the steps that people who are familiar with the project are most likely to miss.

Choose a work that can explain clearly

Element	Explanation
Select a direction	Small but complete works
Write clear boundaries	Necessary items and temporarily unavailable items
Explanation completed	Show real test evidence

Don't treat options as finished products: I'll take OreRush, which I've already written, as an example. Tasks, pets, and tools are only optional directions and must be implemented and tested by yourself; choosing more directions will not automatically add points.

Checklist

Direction	Minimum deliverable gameplay	Major projects on hold
Mini games	Registration, timing, scoring, settlement	Cross-server league
Task system	Accept tasks, progress, and receive rewards at once	Complex plot editor
Pet companion	Summon, follow, withdraw	Complete combat AI
Utility tools	One command solves one problem	Universal management panel

03 / 14

Separate what must be done from what must be done later

Write a range card for 1.0.0. On the card, write both "what you can do" and "what you can't do." In this way, when students ask "Where is my champion after restarting?", you can point out that only the memory data of this game is currently retained, instead of temporarily pretending that the champion was stolen by the creeper.

Separate what must be done from what will be done later

Element	Explanation
Game play completed	Registration, timing, scoring, settlement
Clear restrictions	Does not include historical list and complete cheating prevention
Restart contract	This office is cleared and the configuration is retained.

Definition of completion: at least one complete two-player game, ten game scene records, and 24 rule checks passed. Projects that have not been run are marked as "to be tested" and cannot be checked for installation and clearance.

OreRush 1.0.0 range card · See the description in this section for the location

Must: join, score, start, stop Four subcommands
 Must: Iron ore 3 points, gold mine 5 Points, diamonds 10 points, including deep slate version
 Must: The world and 1 to 3600 seconds configuration, administrator starts and stops
 Must: Naturally end the display of results, cancel or deactivate the cleaning task
 Does not contain: History list, physical rewards, automatic mines, complete anti-cheating
 Restart: Clear the results and registration of this bureau, disk config.yml Reserved

- Ordinary players can only register and inquire; administrators can start or cancel.
- In the survival mode of the designated world, registered players are allowed to destroy the target ore before scoring.
- After verification in the prepared dedicated test field, the practice version cannot be declared as the anti-cheating system of the public competitive server.

04 / 14

Put the code where you can find it

Lesson 38 only has two class, so there is no need to split it into twenty empty boxes in order to appear advanced. GameRound manages numbers and status; OreRushPlugin connects to Paper. First clarify the responsibilities, and then consider packages such as command, listener, and data when the real complexity appears.

Put the code where it can be found

Element	Explanation
Number rules	GameRound independent testing
Server wiring	OreRushPlugin connects Paper
Delivery Instructions	Source code, resources, Wrapper and records

Boundaries are useful like walls: GameRound doesn't read Player, write files, or manipulate blocks. This way the automatic check in Lesson 39 can run independently. Don't spin up an entire server just to test a scoring method.

The actual delivery structure of this project · See the description in this section for the location

```
OreRush/  
  build.gradle.kts  
  settings.gradle.kts  
  gradlew gradlew.bat gradle/wrapper/  
  src/main/java/de/teacherma/orerush/  
    GameRound.java  
    OreRushPlugin.java  
  src/main/resources/  
    plugin.yml  
    config.yml  
  checks/RoundChecks.java  
  README.md  
  CHANGELOG.md  
  TEST-RESULTS.md
```

05 / 14

Give an accurate version of the release package

The file name, plugin.yml and instructions should be the same version. Use the same build script from Lesson 31 and check only the fields below. We agree that 1.0.1 represents minor fixes, and 1.1.0 represents compatible new features; the version number is an agreement, not a magic guarantee of compatibility.

Stop and think about it: After the update, the game still behaves like the old version. Should I change the code first or check the installation package first? How to prove that the test is a new build?

Give the release package an accurate version

Element	Explanation
Build version	build.gradle.kts
Plug-in version	plugin.yml
Delivery version	The JAR file name is consistent with the description

Figure out this step: first check the plug-in version, build product file name and installation location, and then look at the startup log. Compare file hashes when necessary to avoid repeatedly testing new source code on old JARs.

The three positions are modified respectively. See the instructions in this section for the positions.

```
// settings.gradle.kts
rootProject.name = "OreRush"
// build.gradle.kts
version = "1.0.0"
```

For the version field in plugin.yml, see the description in this section for its location.

```
version: 1.0.0
```

Build the project root directory and view the content · See the description in this section for the location

```
.\gradlew.bat clean build
jar tf build/libs/OreRush-1.0.0.jar
```

- There should be plugin.yml, config.yml and .class files of the two main program classes in the JAR.
- Use the Gradle Wrapper of the project, retaining the wrapper directory and startup script.
- A successful build only means that the build is successful, and does not mean that the plug-in permissions, world conditions or two-player gameplay have been tested.

06 / 14

Configuration is not a score archive

The configuration explains how the game is played, and the results explain what happened in the game. They are not the same account. The current plug-in saves config.yml, but registration and results are only in memory; after stopping the server, it will not automatically resurrect the previous game, and it cannot resume the countdown from a task number.

Configuration is not a score archive

Element	Explanation
Configure	Explain how to play the game
This office data	Explain what happened in this game
Future History	Clearly mark it before it is implemented

What to do if you want to add to the history list: Go back to the data storage ideas of Lesson 29, clarify the format version, when to write, and what to do if you read it incorrectly. Don't rewrite the entire history file every time you break a block.

The draft of future historical documents has not yet been implemented. See the description in this section for the location.

```
schema-version: 1
# records Game numbers, players can be saved in the future UUID and score.
# First design verification, backup and failure handling, and then write persistence code.
```

Checklist

Data	Current save location	After restarting
World and duration	Disk config.yml	Read and verify again
Registration and scores for this bureau	GameRound Memory	Clear
Timing tasks	Paper runtime	Cancel, do not restore
Historical results	Currently not implemented	No history to read

07 / 14

Let the instructions answer the first round of questions for you

README is an instruction manual for the work, not a password for the author himself. Ask a classmate who has not participated in the development to follow the installation; wherever he stops, that step is usually worth adding. The following is a rewriteable skeleton, and the actual test fields must be filled in by yourself.

Let the instructions answer the first round of questions for you

Element	Explanation
Installation steps	First time users can follow the instructions
Command authority	Distinguish between players and administrators
True record	Tested, failed and untested are separated

The permission name must be checked: blockacademy.orerush.admin here must be consistent with the plugin.yml and code of Lesson 38. When copying someone else's description, it is easiest to move in other people's permission names as well.

README.md reference text · See the description in this section for the location

```
# OreRush 1.0.0
Purpose: Classroom mining scoring competition under teacher supervision.
Environment: Java21, Normal Paper1.21.11;Not supported Folia.
Installation: Stop taking, will JAR put plugins, Check log after startup.
Configure: plugins/OreRush/config.yml;Stop and restart after modification.
Command: /orerush join, score;Administrator use start, stop.
Management permissions: blockacademy.orerush.admin, Default op.
Rules: Iron 3, Kim. 5, diamond 10 points, all containing deep slate version.
Conditions: Registered, survival mode, designated world, event not canceled.
Result: Natural end of broadcasting the score of this game;stop Cancelled, no awards will be given.
Restart: Clear registration and scores, do not restore this game; configuration retained.
Limits: No history list, no rewards, no mining areas or ore sources to prevent cheating.
Test record: Write your own date, environment, passed items and failed items.
Feedback: Submit recurrence steps and desensitized logs to the teacher.
```

08 / 14

Release records and write changes without writing magic

"Better, faster, stronger" sounds like an enchantment advertisement, but it doesn't tell players what has been changed. Release records list new additions, fixes, and known limitations; test records additionally hold actual evidence. Put unfinished improvements in the planning area and don't pretend to be completed.

Release records write changes without writing magic

Element	Explanation
CHANGELOG	Write only implemented changes
TEST-RESULTS	Leave this real evidence
Next edition plan	Unfinished items do not pretend to be published functions

Honest records also belong to engineering capabilities: the supporting code of this textbook has passed compilation and 24 rule checks, but your modified version needs to be retested. In-game scenes cannot be replaced by actual execution as "written in a book".

CHANGELOG.md writing example · See the description in this section for the location

```
# 1.0.0
New: Registration, score query, administrator start and stop.
New: Six kinds of ore materials are scored, including three kinds of deep slate ores.
New: Configurable world and game duration.
Limits: Matches are not saved across server restarts and do not award physical items.
```

TEST-RESULTS.md record format · See the description in this section for the location

```
Date and environment: Fill in according to this real test
Use case number | Expected results | Actual results | Through/Failure/To be tested
Failure item supplement: Reproduction steps, log position, and retest after repair
Build version: With final delivery JAR Be consistent
```

09 / 14

Use clean test clothes to accept the installation

Your old server may hide old configurations, other permission plug-ins and JARs left last time. Please prepare a separate local test directory and re-install it; when you need to test the protection interaction, explicitly add the protection plug-in. The test environment must also be explainable.

Use clean test clothes to accept the installation

Element	Explanation
Clean installation	No old JARs or old configurations interfering
Normal game	Two players complete a game
Boundary regression	24 rules and ten game scenarios

Acceptance in the dedicated local test server: Do not open the public network port for assignments, upload the complete server or copy real player information. The server license and operating conditions are confirmed by the teacher. This book does not accept the agreement for you.

Checklist

Checkpoint	What counts as seeing something?
Environment	Java and Paper versions comply with instructions
Plug-in loading	No startup exception, command has been registered
First configuration	Generate correct data directory and default file
Ordinary players	Cannot use start or stop
Normal gameplay	Two people sign up, score, and end can be reproduced
Automatic inspection	Rerun and get PASS 24 checks
Boundary Check	The ten scenes of Lesson 39 are recorded one by one
Restart	The old office is cleared and the configuration is still there

10 / 14

If the update fails, you need to know how to go back.

When updating a railway switch, let the train stop first. Stop the server and wait for exit, back up the current JAR and corresponding plug-in data directory, and then replace it with a new version. If the startup fails, keep the log first, and restore the matching old files after stopping the service. You cannot change them randomly during operation.

If the update fails, you need to know how to go back.

Step	Explanation
1. Stop service backup	JAR and data are saved in sets
2. Replace with new version	Restart reading log
3. Failure rollback	Restore matching old version and verify again

Clearly mark the identity of the work: This is a third-party practice plug-in made by learners. Do not claim to be officially certified by Minecraft or Paper. Publishing to a public platform is an independent decision next and is not an automatic action in this lesson.

Local delivery package suggestions · See the description in this section for the location

```
release/  
  OreRush-1.0.0.jar  
  README.md  
  CHANGELOG.md  
  TEST-RESULTS.md  
source/ # Source code, build files, wrapper, Inspection Procedure
```

PowerShell checks the transferred files. See the description in this section for the location.

```
Get-FileHash .\OreRush-1.0.0.jar -Algorithm SHA256
```

- SHA256 consistency can be used to check file copies and is not equivalent to plug-in security authentication.
- Backup and rollback require actual practice; if the data format is changed in the future, it is necessary to verify whether the old program can be read.
- Do not package entire servers, account information, or private logs; check sources and permissions before publicly using material.

five-minute demonstration, let students play first

The demonstration does not require pronouncing each line of import as a spell. Let the audience know the goal first, then show a play and an action that was correctly rejected. You can shorten the configuration before the demonstration and explain that it is a demonstration setting; showing evidence when an accident occurs is more professional than pretending that nothing happened.

Five-minute demonstration, let students play first

Element	Explanation
Experience first	Let the audience see a game
Let's talk about the rules again	Interpret a correct rejection
Final evidence	Show a real repair

Prepare reproducible backup solutions: save tested versions and short action cards. If the live server is unavailable, the logs and test results can be explained, but it should be made clear that the live game demonstration is not completed this time.

- Why use UUID as key instead of just player name?
- Why enter ENDING first and then broadcast the results?
- What is retained and what is cleared on restart? How to prove that there is no second timed task?

Checklist

time	Demonstration content
1st minute	Explain the gameplay and applicable environment in one sentence
2nd minute	Two people signed up, and the teacher started a short game
3rd minute	Mining score check, showing natural settlement
4th minute	Repeated start or unregistered mining is blocked
The 5th minute	Explain GameRound responsibilities and a real repair

12 / 14

The graduation score depends on the evidence and the explanation.

Scoring is not about counting lines of code: copying 10,000 lines but not being able to tell the status is not as good as making a reliable little plug-in yourself. Use the same sheet to check the work and ask trial partners to give feedback on reproducibility. It's recommended to be specific about the action, don't just say "it's a little weird".

The graduation score depends on the evidence and the explanation.

Element	Explanation
Operation and Programming	Can be used and have clear responsibilities
Experience and testing	Feedback is accurate and reproducible
Expression and improvement	Explain limitations and select a piece of feedback

Feedback is not a list of commands: collect three opinions and choose the one that most affects the experience. Explain the reasons for adopting or suspending it. Don't change the plugin into an ender dragon that bakes bread just to please everyone.

Turn fuzzy feedback into action · See the description in this section for the location

```
Blurred: Why always 0 points?
Concrete: No join Just dig diamonds and query to get the unregistered prompt.
Judgment: It belongs to the rule rejection; you can remind you to register earlier in the
> description..
Next step: First change the prompt, then ask students to go through the process again.
```

Checklist

Dimensions and scores	Observe the evidence
Run for 30 minutes	Can be installed, normal gameplay is completed, failure does not destroy the state
Programming 25 minutes	Responsibilities are clear, and conditions, collections, methods and events are used correctly
Player friendly 20 points	Command feedback, permissions, configuration and instructions are easy to understand
Test 15 points	Normal and boundary tests have real records, and retest after repair
Expression 10 points	Can explain key design, known limitations and next steps

13 / 14

Expansion challenges and own acceptance records

Leave the source code, actual running results and a problem record. Explain it in your own words first, and then run the verification; where the results are different from the predictions, they are often the most worthy of discussion.

Expansion tasks and clearance review

Element	Explanation
Wooden Pickaxe	Complete local graduation package
Iron Pickaxe	Select only one improvement and accept it
Diamond Pickaxe	Use three pieces of feedback to formulate the next version

Next stop: The main line of 40 lessons is completed here. Pick a specific problem and go through the next round of adventure of “design, code, test, and illustrate” independently.

Official extended reading: docs.papermc.io/paper/dev/project-setup/

- **Wooden Pickaxe:** Submit a minimum playable work, independent JAR, source code, instructions, real test records, and give a five-minute demonstration.
- **Iron Pickaxe:** Select only one improvement from configuration, permission feedback or history records, and complete normal, boundary and failed tests.

- Diamond Pickaxe: Invite three students to try it out, choose one piece of feedback to formulate the 1.1.0 plan; only after it is implemented and accepted will it be included in the published functions.
- Acceptance requirements: Complete a 5-minute demonstration and independent installation by deskmates; submit version list, tested scenarios, untested projects and executable rollback steps.

Checklist

I can complete it independently	Mastered	Practice again
Able to deliver works that others can install	☐	☐
Ability to explain core rules and known limitations	☐	☐
Feedback and testing will be used to drive the next improvement	☐	☐

14 / 14

The next adventure of the graduation work

The end point of Lesson 40 is a work that others can install, check, and give feedback on. Pick a real problem: maybe a clearer rejection prompt, a more stable state cleanup, or a contracted data save. Write it as an acceptance condition for the next version.

Next edition plan

Problem	First define the result	Leave evidence again
Improved experience	At which step do new readers no longer get stuck?	Let another trial user go through the process again
Repair defects	The same input is processed according to the contract	Keep regression results before and after repair
Add function	Clearly write boundaries and failure strategies	Published changes will be included after implementation

Technical reference: This lesson uses Paper 1.21.11 and Java 21.

Version API: <https://jd.papermc.io/paper/1.21.11/>

Project establishment and release structure: <https://docs.papermc.io/paper/dev/project-setup/>