

JAVA / LESSON 39 OF 40

The Server Emergency Room

We'll find the real Bug on logs, boundaries and tests.

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. | 110-130 minutes



24 rules inspection and game acceptance records

Debugging is based on evidence. Read your own location in the stack and repeat three Bugs for extra-score scores, late endings and double counting; verify rules and connections with 24 pure Java checks and 10 game scenes, respectively.

Continue with Lesson 38 OreRush Project. The test uses Java21; the in-game check continues to use ordinary Paper1.21.11. Faulty clips should only be placed on the practice copy and do not overwrite the working version.

Before you start: Continue with the Ore Dash in Lesson 38

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=39&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. The red stack is a positioning map
03. If the player cannot be found, stop first.
04. Boundary Repair 1: Bonus points after the end
05. Boundary Repair 2 The game of surviving for one more second
06. Boundary Repair Three Two referees count seconds at the same time
07. The first half of the automatic inspection program
08. The lower part of the automatic inspection program
09. Test the game in ten scenarios
10. Find the real time-consuming before optimizing
11. Expansion challenges and own acceptance records
12. Leave the evidence for the next adventure

01 / 12

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 / 12

The red stack is a positioning map

The server throws out a long list of red letters, and Fang guesses from the bottom. The detective handed over a magnifying glass: first identify the type and cause of the anomaly, then find the first location of your project, and follow the call path to avoid getting lost in the log.

Debug first reproduce, then narrow down the conditions, and finally fix one point and retest. There are two levels of testing: pure Java checking rules, and real server checking whether events, commands and scheduling are connected correctly. Compilation approval cannot replace in-game acceptance.

The red stack is a positioning map

Element	Explanation
Exception type	Explain what type of failure
Own source files	Find the first location in the project
Reproduction steps	The same input can be retested

The line number is not a permanent coordinate: the textbook indicates that the line number is for practice and is not guaranteed to be the same as the current line number of your file. If import or comments are added, the following line numbers will change.

The teaching instruction stack is not the actual measurement log. See the description in this section for the location.

```
java.lang.NullPointerException: ...  
    at de.teacherma.orerush.OreRushPlugin.onCommand(OreRushPlugin.java:58)  
    at org.bukkit.command.PluginCommand.execute(PluginCommand.java:...) 
```

Write a fault record · See the description in this section for the location

```
Steps: Enter/orerush score  
Expected: Not registered yet  
Actual: Red letter abnormality  
Environment: Java21 Paper1.21.11 Plug-in version and complete exception are attached separately
```

03 / 12

If the player cannot be found, stop first.

Querying online players may result in null, and offline does not mean "finding an invisible Player". Check results immediately before reading names or sending messages. Not all nulls should be forced to the default object, sometimes it is more appropriate to explicitly tell the operator to fail.

If the player cannot be found, stop first.

Element	Explanation
Query object	Online search may be null
Check failed	Description offline and stop
Success object	Method is called later

Distinguish between expected failure and program defects: Player offline is an expected situation; incorrect type conversion may be a code defect. Don't cover all problems with a catch (Exception) and then tell the player "the operation was successful".

The security fragment that can be added to the management command · See the description in this section for the location

```
Player target = getServer().getPlayerExact(name);
if (target == null) {
    sender.sendMessage("Player is not online, please check name");
    return true;
}
target.sendMessage("The teacher invites you to take the test");
```

Keep the exception instead of pretending to be successful · See the description in this section for the location

```
catch (IOException ex) {
    getLogger().log(Level.WARNING, "Read failed", ex);
}
```

04 / 12

Boundary Repair 1: Bonus points after the end

The bad version only checks whether the player has signed up. When the list is still there after the game, if you continue to mine, your points will increase. Add "in game" to the condition of the extra point method, and even if the caller forgets to check, the core rules can still block extra points outside the game.

Stop and think about it: If the repeated start test of the rule class passes, can it prove that the server only creates one scheduled task? What else to observe?

Boundary Repair 1: Still bonus points after the end

Element	Explanation
WAITING	No points will be added even if you have registered
RUNNING	Write only after you have registered and scored correctly
ENDING	Entrance is closed

Figure this out: No. You also need to send the start command repeatedly in the game, observe the countdown speed, confirm that there are no residual messages after stopping, and start normally again.

Fault experiment: The errors are deliberately kept below and are only observed in the practice copy. Do not merge it into available plugins.

Bad version, forgotten status · Intentional error, predict first and then repair

```
if (scores.containsKey(id)) scores.merge(id, points, Integer::sum);
```

The location of the restored entrance is shown in this section.

```
if (state == State.RUNNING && scores.containsKey(id) && points > 0)
    scores.computeIfPresent(id, (key, old) -> old + points);
```

Inspections that can reveal problems · The location is explained in this section

```
r.add(alex, 10); // In WAITING Status
check(r.snapshot().get(alex) == 0, "waiting cannot score");
```

05 / 12

Boundary Repair 2 The game of surviving for one more second

If it is judged that remaining is equal to 0 first and then subtracted, the 2-second game will not end until the third callback. Let's use an example that's short enough to count: start with 2, change to 1 for the first time, change to 0 for the second time and settle. Don't really have to wait a minute.

Boundary Repair 2: The Game of Surviving One More Second

Step	Explanation
1. Initial 2	No callback, still competing
2. 1st tick	1 left, not over
3. The 2nd tick	Return to zero and end once

Precondition: Check whether state is RUNNING at the beginning of tick. Otherwise, if you continue to subtract after the end, the remaining time will be dug all the way to the negative basement.

Fault experiment: The errors are deliberately kept below and are only observed in the practice copy. Do not merge it into available plugins.

Bad version: judge first and then subtract · Intentional error, predict first and then fix

```
if (remaining == 0) return true;
remaining--;
return false;
```

It ends when it returns to zero. See the description in this section for the position.

```
if (--remaining > 0) return false;
state = State.ENDING;
return true;
```

Checklist

Number of callbacks	Remaining after repair	Whether to end
0	2	No
1	1	No
2	0	Yes
3	0	No, end without repeating

06 / 12

Boundary Repair Three Two referees count seconds at the same time

The bad version creates a task every time it receives a start. Press it twice and the seconds will drop as if you have taken a potion of urgency. First let GameRound.start verify the status, and only enter the scheduler if successful; when stopping, both cancel and clear the task field.

Boundary Repair 3 Two referees count seconds at the same time

Step	Explanation
1. Check status first	Repeat start rejected
2. Re-create tasks	Only successful paths are arranged
3. End cleanup	cancel and task is set to null

Before entering the next step along the arrow, the conditions of the previous step must be met; the rejected branch returns early.

Both layers must be tested: Pure Java inspection can prove that repeated start is rejected, but it cannot prove that only one task is really created in Paper. The countdown speed, silence after stop and restart must also be verified in the game.

Do not place the creation task before checking · See the description in this section for the location

```
if (!round.start(duration)) {
    sender.sendMessage("Unable to start");
    return true;
}
// Create the task below
```

Cleaning should be complete·Complete files

```
if (task != null) task.cancel();
task = null;
round.reset();
names.clear();
```

07 / 12

The first half of the automatic inspection program

Create new checks/RoundChecks.java in the OreRush project. This page continues from the next page which is a complete file and does not include the plug-in JAR. When the check fails, an AssertionError is thrown and the operation fails immediately; it will not be skipped silently like Java assert, which is turned off by default.

Automatic inspection program upper part

Element	Explanation
Air bureau and registration	Refuse no one to start and repeat registration
Start boundary	Both duration and status are verified
Scoring Boundary	Out-of-match, outsider and negative score rejection

The time is controlled by the test: only two ticks are called for a 2-second game without sleep. The test does not access the world, network or player, so it is fast and easily repeatable.

RoundChecks Part 1 · Sequentially connected to the same source file

```
import de.teacherma.orerush.GameRound;
import java.util.UUID;

public class RoundChecks {
    private static int checks;
    private static void check(boolean condition, String name) {
        checks++;
        if (!condition) throw new AssertionError(name);
    }
    public static void main(String[] args) {
        UUID alex = new UUID(0, 1), steve = new UUID(0, 2);
        GameRound r = new GameRound();
        check(!r.start(60), "empty cannot start");
        check(r.join(alex), "first join");
        check(!r.join(alex), "duplicate join");
        r.add(alex, 10);
        check(r.snapshot().get(alex) == 0, "waiting cannot score");
        check(!r.start(0), "zero duration rejected");
        check(!r.start(3601), "too long rejected");
        check(r.start(2), "start two seconds");
        check(!r.start(2), "duplicate start rejected");
        check(!r.join(steve), "late join rejected");
        r.add(alex, 10); r.add(steve, 10); r.add(alex, -5);
        check(r.snapshot().get(alex) == 10, "valid points only");
        check(!r.snapshot().containsKey(steve), "outsider cannot score");
    }
}
```

08 / 12

The lower part of the automatic inspection program

Continue to the previous page. After returning to zero, try adding points again, reset again, and check whether the old score snapshot still exists. Finally overrides deep slate ore, normal stone and null input. Outputs PASS 24 checks on success.

Automatic inspection program lower half

Element	Explanation
Time and end	The second tick returns to zero
Reset and Snapshot	Old snapshots still retain results
Material rules	Deep slate, ordinary stone and null

Description of running commands: The two commands are executed separately; long commands may be displayed in new lines in the book and remain as one when input. You can also run the main of RoundChecks directly in the IDE. Failure in the test does not mean passing the level, it needs to be located and repaired.

RoundChecks Part 2 · Sequentially connected to the same source file

```
        check(!r.tick() && r.remaining() == 1, "first tick");
        check(r.tick(), "second tick ends");
        check(r.state() == GameRound.State.ENDING, "closed before result");
        r.add(alex, 10);
        check(r.snapshot().get(alex) == 10, "ending cannot score");
        check(!r.tick(), "cannot end twice");
        var result = r.snapshot();
        r.reset();
        check(r.state() == GameRound.State.WAITING, "reset state");
        check(r.snapshot().isEmpty(), "reset enrollment");
        check(result.get(alex) == 10, "snapshot survives reset");
        check(GameRound.points("DEEPSLATE_DIAMOND_ORE") == 10, "deep diamond");
        check(GameRound.points("GOLD_ORE") == 5, "gold");
        check(GameRound.points("DEEPSLATE_IRON_ORE") == 3, "deep iron");
        check(GameRound.points("STONE") == 0, "ordinary block");
        check(GameRound.points(null) == 0, "missing material");
        System.out.println("PASS " + checks + " checks");
    }
}
```

First create a new checks/sources.txt, and write the following two source file paths relative to the project root directory, one per line. Then run the short command from the project root directory; you can also run RoundChecks.main directly from the IDE.

checks/sources.txt · Full content

```
src/main/java/de/teacherma/orerush/GameRound.java
checks/RoundChecks.java
```

Run Windows project root directory · See the instructions in this section for the location

```
javac -encoding UTF-8 -d checks/out @checks/sources.txt
java -cp checks/out RoundChecks
```

Output

```
PASS 24 checks
```

09 / 12

Test the game in ten scenarios

What is automatically checked is digital rules, and there are more command permissions, modes, worlds and other plug-ins in the game. Operate according to the table and record the "actual results" and log location; when you find a problem, only change one condition and try again, so that the evidence will not teleport like the Ender Man.

Test the game in ten scenarios

Element	Explanation
Rule Check	Numbers and status can be independently verified
Game acceptance	Permission event scheduling needs to be aligned
Two pieces of evidence	Cannot replace each other

Note the switch: The protection plug-in in Lesson 33 may block damage first. When testing normal scoring, be in a position that allows destruction; when testing cancellation events, enable corresponding protections.

Checklist

Number and action	Expected
1 No one signed up to start	Reject
2 Repeat join with the same person	Second rejection
3 Two people sign up to start	Only start one round
4 Start again immediately	Rejected, no second task
5 Mining by those who have not signed up	No scoring
6 Creative mode or different world mining	No scoring
7 Protection to cancel mining	No scoring
8 dig again after natural end	No scoring
9 stop during timing	No subsequent countdown broadcast
10 Stop the service and restart after registration starts again	Old tasks and local data are not restored

10 / 12

Find the real time-consuming before optimizing

When players say "stuck", it may be caused by network, world generation, or plug-ins. Record the time, number of people, operations and reproduction steps first, and then measure. Don't declare a for loop guilty just when you see it: looping ten points is not the same thing as writing a file for every piece of ore.

Before optimizing, find the real time-consuming

Step	Explanation
1. Reproduce first	Record operations, number of people and time
2. Measure again	Identify the real time-consuming path
3. Post-improvement	Retest the same scenario

Look at these two indicators: TPS is how many ticks are completed per second; MSPT is the time spent per tick on average or under some statistical caliber. The target 20TPS corresponds to a budget of about 50 milliseconds, but one sample or average cannot represent all lags.

Only a section of pure calculation is measured. See the description in this section for the position.

```
long started = System.nanoTime();
var result = round.snapshot();
long elapsedNs = System.nanoTime() - started;
// Single result is subject to JVM Influences such as preheating are not formal performance
> conclusions..
```

Useful but not excessive logs · See the description in this section for the location

```
getLogger().info("OreRush started, players=" + round.snapshot().size());
```

- Do not download web pages, traverse all blocks, or save entire files every time you mine.
- In the test server, you can use the performance analyzer according to Paper's official Profiling guide, first check the time-consuming calls and then optimize.
- Check the player name, address and path before sharing logs or analysis reports; underage students should not upload complete server information by themselves.

11 / 12

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	Explain your own stack
Iron Pickaxe	Repair three boundary bugs
Diamond Pickaxe	Make a read-only self-test command

Next stop: Lesson 40: Organize the accepted works into a graduation package that others can install and use.
Official extended reading: docs.papermc.io/paper/dev/reading-stacktraces/

- Wooden Pickaxe: Explain a stack of one's own project, pointing out exceptions, methods and source file locations.

- Iron Pickaxe: Reproduce and fix three bugs in this lesson in the practice copy, and submit the test results before and after the repairs.
- Diamond Pickaxe: Make a read-only self-test command to display the version, current status and number of applicants; do not output private information.
- Acceptance requirements: Keep the pre-repair recurrence steps, positioning basis and post-repair results; switch to a deskmate and record the operation, and you should still get a consistent conclusion.

Checklist

I can complete it independently	Mastered	Practice again
Errors can be reproduced with evidence	☐	☐
Will run 24 rule checks	☐	☐
Able to distinguish unit rules from game acceptance	☐	☐

12 / 12

Leave the evidence for the next adventure

Divide "can compile", "number rules pass" and "actual performance of the game is correct" into three columns. Scenarios that are not running are written to be tested; when bugs are discovered, trigger conditions, expected, actual, minimum repairs, and retest results are recorded.

Your verification record

Category	The operation you selected	Evidence left behind
Normal	A successful path for this lesson	Version and actual output
Border	The moment before the end, the maximum value or the full backpack	Compare item by item with contract
Reject	Wrong parameters, no permissions or irrelevant events	The feedback is clear and the status has not been changed by mistake.

- Explain why each arrow in this lesson points from here to there; the parallel cards have no sequential meaning.
- First change a rule, then modify the description and expectations at the same time, and retest with the same input.

Technical check: This lesson is based on Paper 1.21.11 API and Java SE 21. The reference document may be switched to a newer version, so pay attention to the version number when checking it.

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

<https://docs.papermc.io/paper/dev/>

<https://docs.oracle.com/en/java/javase/21/docs/api/>