

JAVA / LESSON 38 OF 40

Project: The Ore Sprint

Combining commands, events and timing into a game

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. | 150-180 minutes



A complete round of registration, timing, scoring and settlement

Connect the parts of the first seven lessons into an ore sprint race. The rules class manages status and scores, and the plug-in connects commands, events and timings; after the game ends, the scoring entrance is closed first, and then the results are read and the cleaning tasks are completed.

Front Backpack: Lessons 31-37. The independent project name is OreRush, the package name is de.teacherma.orerush, and we continue to use Java21 and ordinary Paper1.21.11. This lesson can be completed in two sessions.

Before you start: It is recommended to complete Lessons 31-37 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=38&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. Make an appointment with the rules of the competition first
03. Status determines what can be done now
04. Registration list and score sharing player key
05. Filter events first and then calculate the ore score
06. Only one task ticket is received per game
07. Close the scoring first and then announce the results
08. Prepare two classes and two resource files
09. Complete competition rules class
10. The first part of the main class of the competition
11. Competition main class second part
12. The third part of the main class of the competition
13. The fourth part of the main class of competition
14. Complete the entire round from registration
15. Expansion challenges and own acceptance records
16. Leave the evidence for the next adventure

01 / 16

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 / 16

Make an appointment with the rules of the competition first

Before the game started, A Fang had already dug out a bag of coal and urged the referee to add points. The referee turned over the status card: you can only register during the assembly, and the scores will be recorded during the game. After the settlement, you can't modify the score of this round by swinging the pickaxe.

First write the rules as status changes: WAITING and other registrations, RUNNING to accept valid scoring, ENDING to freeze and settle. Pure Java class management rules, the Paper main class connects commands, events and tasks, and each side has checkable responsibilities.

Make an appointment with the rules of the game first

Element	Explanation
Player registration to check scores	join and score
Administrator starts and stops	start and stop
Scoring Contract	World survival mode is specified and the event is not canceled

Practice scope: This is the minimum playable version of MVP supervised by the teacher, which does not include automatic mining area reset, cross-server, anti-brushing and physical rewards. Only use in droppable practice test worlds; do not pickaxe with important survival saves.

Four commands for players and teachers · See the instructions in this section for their positions

```
/orerush join      # Player registration
/orerush score     # Query the score of this bureau
/orerush start     # Teacher starts
/orerush stop      # Teacher cancels and clears registration
```

- The three types of ores include ordinary ores and deep slate ores. Points are scored based on the number of blocks allowed to be destroyed, not the number of dropped items.
- Only events in designated worlds, survival modes, and registered players are calculated.
- Registration and scores will be cleared after one round; registration must be re-registered for the next round.

03 / 16

Status determines what can be done now

WAITING is like a gathering area, allowing registration; RUNNING is like a mine in a competition, allowing scoring; ENDING locks the scoring first, and then sorts out the results. State is not a sign written on the wall, but a value that is checked by the program before each action.

Three states: start, return to zero and return to waiting

Element	Explanation
WAITING	Registration is allowed, waiting to start
RUNNING	Only score points for registered players
ENDING	Stop scoring and read the results

Transition	Trigger
WAITING → RUNNING	Start
RUNNING → ENDING	Timer reaches zero
ENDING → WAITING	Reset
RUNNING → WAITING	Stop / reset

Add the return arrow: reset and return to WAITING after settlement is completed. When drawing, draw an arrow for ENDING back to the gathering area, so that there will be an entrance to the next game.

Extra points can only be added during the game. See the description of this section for the position.

```
if (state != State.RUNNING) return;
```

Checklist

Action	WAITING	RUNNING	ENDING
Registration	Allow	Reject	Reject
Start	Some people allow it	Reject	Reject
Bonus points	Reject	Only allowed after registration	Reject
reset	Clear	Cancel and Clear	Clear after settlement

04 / 16

Registration list and score sharing player key

UUID is used as a key, and a score of 0 means that you have registered but have not yet scored. The absence of this key means that you have not registered, and the two cannot be confused. names is saved as the name during registration and is only used for result display; the program will not hold on to a Player object for a long time.

Registration list and score sharing player key

Element	Explanation
The same UUID	Name and score share identity key
Initial 0 points	Created only after successful registration
Name Snapshot	Only used for performance display

Offline processing: The results of this game will not be deleted when offline; continue with the same UUID after coming back. By default, the practice has a small number of people. At least one person can sign up to start. It does not force all applicants to be online; the rules must be told to the players.

Avoid repeated registration and clearing · See the instructions in this section for the location

```
if (state != State.WAITING || scores.containsKey(id)) return false;
scores.put(id, 0);
return true;
```

Distinguish between unregistered and zero points. See the description in this section for the location.

```
Integer score = round.snapshot().get(id);
String message = score == null ? "You have not registered yet" : "Score of this game: " + score;
```

05 / 16

Filter events first and then calculate the ore score

A stone can also generate a destruction event, but it does not have to be given a diamond medal. The listener first filters the competition status, world and survival mode; the rules class then checks the registration. The material name is given to a pure Java method and gets 0, 3, 5 or 10.

Stop and think about it: Should you broadcast the leaderboard first when you finish, or leave the RUNNING state first? What if another mining event is received during settlement?

Filter events first and then calculate the ore score

Step	Explanation
1. Filter events	Game status, world, mode
2. Obtain ore score	Iron 3, Gold 5, Diamond 10
3. Write in those who have registered	Not registered or 0 points are not written in

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

Figure out this step: turn off scoring first, then generate and broadcast the results. In this way, new events during the settlement period will not change the score of the game, and can also prevent repeated settlements.

Unified score entrance · See the description in this section for the location

```
int points = GameRound.points(event.getBlock().getType().name());
```

Two types of deep slate ores must also be recognized. See the description in this section for their locations.

```
GameRound.points("DEEPSLATE_DIAMOND_ORE"); // 10
GameRound.points("DEEPSLATE_IRON_ORE");    // 3
GameRound.points("STONE");                  // 0
```

06 / 16

Only one task ticket is received per game

Repeated tasks are created only after successful start. GameRound.start refuses to start again, so if you press start twice, two referees will not appear, and the seconds will be reduced twice. Wait for 20 ticks for the first callback and then subtract 1.

Only one task ticket is received per game

Element	Explanation
Start successfully	Arrange only after the status check passes
Every 20 ticks	Callback an update
The task is unique	Refuse to start again

Game rhythm: ideally, the interval of 60 times at 20TPS is about 60 seconds. The actual time will be longer when the server is stuck. This version uses a tick countdown and does not promise accurate wall clock timing.

Check first and then create the task · See the instructions in this section for the location

```
if (!round.start(duration)) {  
    sender.sendMessage("Cannot start now");  
    return true;  
  
}  
// Called after passing runTaskTimer
```

The order and position in the callback are described in this section.

```
if (round.tick()) { finish(); return; }  
int left = round.remaining();
```

07 / 16

Close the scoring first and then announce the results

The last tick changes the status to ENDING before calling finish. Cancel the task during settlement, read the score snapshot, display the scores from high to low, and finally clean up. Otherwise, when the referee calls for the second person, the first person may secretly dig another piece.

Turn off the scoring first and then announce the results

Step	Explanation
1. Turn off scoring	Return to zero and enter ENDING
2. Read snapshot	Sort by score and UUID
3. Unified reset	Cancel the task and clear the office

Don't hand out rewards conveniently: there is no reward code in this version. To issue physical rewards, you also need to deal with repeated settlements, disconnections, and full backpacks. You can't just stuff a line after the broadcast to issue diamonds.

The status changes in the rule class. See the description in this section for the location.

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

The score snapshot is not afraid of clearing the original table. See the description in this section for the location.

```
var result = round.snapshot();
round.reset();
// result The previous results are still retained and cannot be modified..
```

- Tie scorers have the same score, and the display order is stably arranged with UUID strings. It is not claimed that the person who displays it first wins.
- The results of this game are only broadcast, and the history is not saved; stop is a cancellation, and no awards are given.
- Plug-in deactivation also cancels the task and clears the memory state.

08 / 16

Prepare two classes and two resource files

Create Gradle project OreRush, using build.gradle.kts from Lesson 31. Java files are placed in src/main/java/de/teacherma/orerush. The following two ymls are placed in src/main/resources. After the first startup, modify plugins/OreRush/config.yml, and it will take effect after stopping the server and restarting it.

Prepare two classes and two resource files

Element	Explanation
GameRound.java	Number rules and status
OreRushPlugin.java	Command events and scheduling
Two yml	Identity permissions and gameplay configuration

Teacher's pre-class preparation: prepare a small amount of three types of ores in the world, away from birth protection, give students appropriate pickaxes and switch to survival mode. This lesson does not automatically generate mines, nor does it limit specific mining areas in the world.

Complete plugin.yml · Complete file

```
name: OreRush
version: 1.0.0
main: de.teacherma.orerush.OreRushPlugin
api-version: '1.21.11'
author: TeacherMa
website: https://www.teacher-ma.de
commands:
  orerush:
    description: Sign up and manage ore sprint
    usage: /orerush <join|score|start|stop>
permissions:
  blockacademy.orerush.admin:
    default: op
```

Complete config.yml · Complete file

```
world: "world"
duration-seconds: 60
```

09 / 16

Complete competition rules class

Save the following completely as GameRound.java. This class does not import Paper and only handles status, UUIDs, and numbers; you will test it directly in Lesson 39 without having to start the game every time.

Complete competition rules class

Element	Explanation
Registration contract	Only in WAITING and not repeated
Start contract	Someone, legal duration, free
Scoring Contract	Registered and scored positive points in the competition

Complete GameRound.java · This paragraph 1/2, sequential connection

```
package de.teacherma.orerush;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;
public final class GameRound {
    public enum State { WAITING, RUNNING, ENDING }
    private State state = State.WAITING;
    private int remaining;
    private final Map<UUID, Integer> scores = new HashMap<>();
    public State state() { return state; }
    public int remaining() { return remaining; }
    public boolean join(UUID id) {
        if (state != State.WAITING || scores.containsKey(id)) return false;
        scores.put(id, 0); return true;
    }
    public boolean start(int seconds) {
        if (state != State.WAITING || scores.isEmpty() || seconds < 1 || seconds > 3600)
            return false;
        scores.replaceAll((id, old) -> 0);
        remaining = seconds; state = State.RUNNING; return true;
    }
    public void add(UUID id, int points) {
        if (state == State.RUNNING && scores.containsKey(id) && points > 0)
            scores.computeIfPresent(id, (key, old) -> old + points);
    }
}
```

Complete GameRound.java · This paragraph 2/2, sequential connection

```
public boolean tick() {
    if (state != State.RUNNING) return false;
    if (--remaining > 0) return false;
    state = State.ENDING; return true;
}
public Map<UUID, Integer> snapshot() { return Map.copyOf(scores); }
public void reset() {
    state = State.WAITING; remaining = 0; scores.clear();
}
public static int points(String material) {
    if (material == null) return 0;
    return switch (material) {
        case "DIAMOND_ORE", "DEEPSLATE_DIAMOND_ORE" -> 10;
        case "GOLD_ORE", "DEEPSLATE_GOLD_ORE" -> 5;
        case "IRON_ORE", "DEEPSLATE_IRON_ORE" -> 3;
        default -> 0;
    };
}
```

10 / 16

The first part of the main class of the competition

The next four pages synthesize OreRushPlugin.java in order. This page imports dependencies, declares shared status, and reads configuration; if the YAML itself cannot be read, the plug-in is disabled and the log is retained.

The first part of the main class of the competition

Element	Explanation
Plug-in field	A rule and a task
Candidate Configuration	Type, Duration and World Verification
Registration entrance	Only receive command events if the configuration is qualified

Main class part 1 · This paragraph 1/2, sequential connection

```
package de.teacherma.orerush;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import java.util.UUID;
import net.kyori.adventure.text.Component;
import org.bukkit.GameMode;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.configuration.InvalidConfigurationException;
import org.bukkit.configuration.file.YamlConfiguration;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
import org.bukkit.event.EventPriority;
import org.bukkit.event.Listener;
import org.bukkit.event.block.BlockBreakEvent;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.scheduler.BukkitTask;

public final class OreRushPlugin extends JavaPlugin implements Listener {
    private final GameRound round = new GameRound();
    private final Map<UUID, String> names = new HashMap<>();
```

Main Class Part 1·This paragraph 2/2, sequential connection

```
    private BukkitTask task;
    private String worldName;
    private int duration;
    @Override
    public void onEnable() {
        saveDefaultConfig();
        var config = new YamlConfiguration();
        try { config.load(new File(getDataFolder(), "config.yml")); }
        catch (IOException | InvalidConfigurationException ex) {
            getLogger().log(java.util.logging.Level.SEVERE, "Configuration read failed", ex);
            getServer().getPluginManager().disablePlugin(this); return;
        }
    }
```

11 / 16

Competition main class second part

Continue with the same file. Check the type and scope of the configuration, register commands and events. Join and score must be performed by the player; registration must also confirm the player's world and survival mode.

The second part of the main class of the competition

Element	Explanation
join	Players register in designated worlds
score	Check your own score
start and stop	Management rights protection

Failure will not be corrected secretly: when the world does not exist or the duration exceeds the range, let the teacher correct the configuration. This lesson does not automatically replace the wrong world with another world, nor does it start the game quietly.

Main class part 2·sequentially connected to the same source file

```

        worldName = config.getString("world", "");
        duration = config.getInt("duration-seconds", 0);
        if (!config.isString("world") || !config.isInt("duration-seconds")
            || getServer().getWorld(worldName) == null || duration < 1 || duration > 3600) {
            getLogger().severe("Invalid configuration: world Need to be loaded, duration-seconds Need
> to be 1 to 3600 integer");
            getServer().getPluginManager().disablePlugin(this); return;
        }
        Objects.requireNonNull(getCommand("orerrush")).setExecutor(this);
        getServer().getPluginManager().registerEvents(this, this);
    }
    @Override
    public boolean onCommand(CommandSender sender, Command command,
                            String label, String[] args) {
        if (args.length != 1) { sender.sendMessage("/orerrush join|score|start|stop"); return true; }
        String action = args[0];
        if (action.equals("join") || action.equals("score")) {
            if (!(sender instanceof Player p)) {
                sender.sendMessage("Please operate in the game to register and check scores.");
            }
            return true;
        }
        UUID id = p.getUniqueId();
        if (action.equals("score")) {
            Integer score = round.snapshot().get(id);
            p.sendMessage(score == null ? "You have not registered yet" : "Score of this game: "
> + score);
        } else if (!p.getWorld().getName().equals(worldName)
            || p.getGameMode() != GameMode.SURVIVAL) {
            p.sendMessage("Please use survival mode to register in the competition world");
        } else if (round.join(id)) {
            names.put(id, p.getName()); p.sendMessage("Registration successful, waiting for the
> teacher to start");
        } else p.sendMessage("Already registered, or the competition is in progress");
        return true;
    }

```

12 / 16

The third part of the main class of the competition

This paragraph deals with the teacher's start and stop. When start is rejected, no new tasks will be created; stop cancels the game and clears the registration. The scheduled callback only performs one subtraction and a small amount of message judgment.

The third part of the main class of the competition

Element	Explanation
Start	Create a task after success
Countdown	Both the first time and the interval are 20 ticks
Event	Only send qualified materials into the rules

Why use synchronization tasks: Timing callbacks and ordinary block events are executed sequentially on the Paper main thread of this lesson. Do not move round modifications to asynchronous threads at will, otherwise HashMap and status may compete.

Main class part 3:sequentially connected to the same source file

```

        if (!action.equals("start") && !action.equals("stop")) {
            sender.sendMessage("Parameters can only be join, score, start, stop"); return true;
        }
        if (!sender.hasPermission("blockacademy.orerush.admin")) {
            sender.sendMessage("Only the teacher can start or stop the game"); return true;
        }
        if (action.equals("stop")) {
            reset(); announce("This office has been canceled and the registration has been cleared");
> return true;
        }
        if (getServer().getWorld(worldName) == null || !round.start(duration)) {
            sender.sendMessage("Unable to start: Check world, registrants and race status"); return
> true;
        }
        announce("Ore sprint begins, total " + duration + " seconds!");
        task = getServer().getScheduler().runTaskTimer(this, () -> {
            if (round.tick()) { finish(); return; }
            int left = round.remaining();
            if (left % 10 == 0 || left <= 5) announce("Remaining " + left + " seconds");
        }, 20L, 20L);
        return true;
    }

```

13 / 16

The fourth part of the main class of competition

Complete the block events, performance sorting and cleaning methods, and finally close the main class. finish is only called by the zeroed tick, and all finishing paths can cancel the task; Map snapshot is used to read scores, and the original table is still managed by the rule class.

The fourth part of the main class of the competition

Element	Explanation
Settlement	Read independent score snapshot first
Broadcast	Even points also have a certain order
Clean up	Tasks, registration, names are unified and concluded

Build and install: Run `.\gradlew.bat clean build` to get `build/libs/OreRush-1.0.0.jar`. After stopping the server, install it into plugins and restart; do not put in two versions of this plugin at the same time.

Main Class Part 4-Sequentially connected to the same source file

```

@EventHandler(priority = EventPriority.MONITOR, ignoreCancelled = true)
public void onBreak(BlockBreakEvent event) {
    if (round.state() != GameRound.State.RUNNING) return;
    if (!event.getBlock().getWorld().getName().equals(worldName)) return;
    if (event.getPlayer().getGameMode() != GameMode.SURVIVAL) return;
    round.add(event.getPlayer().getUniqueId(),
        GameRound.points(event.getBlock().getType().name()));
}
private void finish() {
    if (task != null) task.cancel();
    task = null;
    announce("Time is up, settle the score of this game!");
    round.snapshot().entrySet().stream().sorted((a, b) -> {
        int order = Integer.compare(b.getValue(), a.getValue());
        return order != 0 ? order : a.getKey().toString().compareTo(b.getKey().toString());
    }).forEach(entry -> announce(names.getDefault(entry.getKey(), "Unknown player")
        + ": " + entry.getValue() + " points"));
    reset();
}
private void announce(String text) { getServer().broadcast(Component.text(text)); }
private void reset() {
    if (task != null) task.cancel();
    task = null; round.reset(); names.clear();
}
@Override
public void onDisable() { reset(); }
}

```

14 / 16

Complete the entire round from registration

Invite two students to sign up separately and let the teacher start. Alex digs a diamond and Steve digs two iron ores. The ending points should be 10 and 6 respectively. Do the fault route again: do not register, start again, continue digging after the end, and see if the score is blocked.

Complete the entire round from registration

Element	Explanation
Alex Diamond	10 points
Steve iron ore two pieces	6 points
Dig again after finishing	No longer written to this office

Fairness boundary: This lesson does not distinguish between natural ores and ores placed by players, and there is no complete anti-cheating. Only play in the test field prepared and supervised by the teacher; public servers also require mining area isolation, block source and reward auditing.

Checklist

Operation	Expected
start when no one signs up	Reject
Start or join again during the game	Reject, do not superimpose tasks
Digging two types of Diamond Ores in survival mode	10 points per block
Unregistered or other world mining	No scoring
Destruction is canceled by the protected plug-in	No scoring
Dig again after natural end	No points, need to re-register
Stop or stop serving during the game	Tasks are cleared and results are not retained

15 / 16

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 a game completely
Iron Pickaxe	Modify score and retest
Diamond Pickaxe	Add read-only status query

Next step: Lesson 39 uses logs and automatic inspection to remove boundary bugs from the mine.

Official extended reading: docs.papermc.io/paper/dev/scheduler/

- Wooden Pickaxe: Make a test list for the three ores and their dark slate versions, confirming that ordinary stone is 0 points.
- Iron Pickaxe: Complete a two-player game, repeat start, stop and restart tests, and record ten acceptance results.
- Diamond Pickaxe: Design a team or combo, select only one new function; write down clearly when to add points and when to clean up, and then implement it.
- Acceptance requirements: Complete a two-player game, record registration, repeated starts, unregistered mining, event cancellation, mining after the end and reset of the second game.

Checklist

I can complete it independently	Mastered	Practice again
Can explain three game states	☐	☐
Will connect command events and scheduling	☐	☐
Able to navigate normal and failed paths	☐	☐

16 / 16

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/>