

JAVA / LESSON 36 OF 40

Let Server Owners Configure It

Let configurations be checked and become new rules

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



Verifiable and rereadable rule configuration platform

Configuration is not to insert any random text into a digital grid. First check the type, scope and world and replace qualified candidate configurations with new rules; when reading fails, the last valid configuration continues to work.

Front-end backpack: Gradle project in Lesson 31, commands in Lesson 32, and exception handling in Lesson 28. Continue to use Java21 and ordinary Paper1.21.11. The new project name is BlockAcademyL36.

Before you start: It is recommended to complete Lessons 28, 31-35 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=36&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. There is no need to rebuild the oven for one more piece of bread
03. Template files and files in use
04. Check the type first and then check the size
05. New rules: Change shift only after all the new rules are passed
06. Concentrate the inspection in the rule class
07. Item list of configuration station
08. Configure the upper half of the main class
09. Configure the lower half of the main class
10. Zero reward and full backpack are also considered normal situations
11. Change three into four with your own hands
12. Let the error indicate which line to fix
13. Give the seconds to the next event
14. Expansion challenges and own acceptance records
15. Leave the evidence for the next adventure

01 / 15

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 / 15

There is no need to rebuild the oven for one more piece of bread

The server owner just wants to change the reward bread from 3 pieces to 4 pieces, and A Fang is already preparing to reprogram the entire plug-in. Profiles are like oven knobs: when the rules allow for adjustments, turn the knob to change the setting, don't tear down the house just yet.

The configuration template provides default values, the disk file saves the actual settings, and the memory saves the currently effective rules. When reloading, the candidate configurations are first completely read and verified, and only the old rules in memory are replaced after all are qualified.

No need to rebuild the oven for one more piece of bread

Element	Explanation
Template default value	Starting point for first installation
Disk configuration	Files edited by the server owner
Memory rules	The settings that are currently in effect

The works of this lesson: complete /rules show, /rules snack, /rules reload. Snack is a repeatable test command authorized by the administrator, not a novice gift package that each person can receive only once.

Put into src/main/resources/config.yml · See the instructions in this section for the location

```
welcome-message: "Welcome to Block Academy!"
starter-bread: 3
ore-rush:
  duration-seconds: 60
  world: "world"
```

- starter-bread controls the number of test rewards; 0 means closed.
- ore-rush.world specifies a world that allows bread testing.
- duration-seconds is for viewing first, and then the same idea is used to control the game duration.

03 / 15

Template files and files in use

The config.yml in the JAR is like a blank recipe book; the one in the plugins folder is the book that the server owner is filling in. saveDefaultConfig only copies the default file when the target file does not exist, and will not erase the teacher's changes every day when the computer is turned on.

Template files and files in use

Element	Explanation
resources/config.yml	Default template distributed with JAR
Plug-in data directory	Current server configuration
saveConfig	Write back, not reread

Don't press the wrong save key: saveConfig writes the memory configuration back to the disk, not "rereading" it. In this lesson, the candidate configuration is read after manually editing the file, and saveConfig is not called in reload.

Prepare the default file when enabled. See the description in this section for the location.

```
saveDefaultConfig();
```

Two different positions · See the description in this section for the position

```
src/main/resources/config.yml  
plugins/BlockAcademyL36/config.yml
```

Checklist

File	Responsibilities	How to take effect after the change
plugin.yml	Plug-in identity and command	Rebuild and restart
config.yml in JAR	Default value for first installation	Rebuilding affects new installations
Data directory config.yml	Current server rules	This lesson uses /rules reload

04 / 15

Check the type first and then check the size

The number 3, the text "3" and the decimal 3.5 look like relatives, but they cannot all live in the integer grid. Just clamping the negative numbers to 0 still doesn't explain the input "many pieces of bread". We check the type first and then the agreed scope.

Check the type first and then the size

Element	Explanation
Type	Integer is not equal to alphanumeric
Range	Bread 0 to 64
world	The name corresponds to the loaded world

Leave a space after the colon: YAML levels are indented with spaces, without tabs. The above three starter-breads are three independent experiments, do not copy them as duplicate keys at the same time.

Three values that look similar · See the instructions in this section for their positions

```
starter-bread: 3      # Integer Qualified
starter-bread: "3"    # Text Reject
starter-bread: 3.5    # Decimal Reject
```

Read nested keys · See the description in this section for the location

```
Object value = config.get("ore-rush.duration-seconds");
```

Checklist

Configuration items	Agreement in this lesson
welcome-message	Non-empty text, up to 80 Java characters
starter-bread	Integer 0 to 64
ore-rush.duration-seconds	Integer 1 to 3600, unit second
ore-rush.world	Non-empty name and world loaded

05 / 15

New rules: Change shift only after all the new rules are passed

If the bread is first changed to 64, and then it is discovered that the world name is misspelled, the server is in a half-new and half-old rule. We first read a candidate configuration, check all fields and worlds, and finally replace the rules in use with only one assignment.

Stop and think about it: the reward amount is legal, but the active seconds are written in text. Can I apply the reward first and then report the seconds error?

The new rules will only change shifts if all are qualified

Step	Explanation
1. Read candidate files	Keep the old rules unchanged
2. Complete verification	Both the field and the world are qualified
3. One shift change	rules point to new records

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

Think through this step: reject the entire application and continue to use the old rules to avoid half a set of new rules and half a set of old rules. The log indicates the specific configuration key and expected type.

Check the world before applying · See the description in this section for the location

```
RuleSettings next = RuleSettings.read(candidate);
if (getServer().getWorld(next.world()) == null)
    throw new IllegalArgumentException("World not loading");
rules = next;
```

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

Don't change half of it first · Intentional mistakes, predict first and then fix them

```
// Wrong idea: update rewards first, then check the world
// If the second step fails, the first step has taken effect.
```

06 / 15

Concentrate the inspection in the rule class

Create a new RuleSettings.java in the same package. Record is Java's data record type: data is listed in parentheses, and Java automatically provides welcome(), bread() and other reading methods. The following method returns a record that has passed the field check; whether the world exists is checked by the plugin.

Concentrate inspections in rule classes

Element	Explanation
Read-only records	String and int storage rules
text verification	Non-empty text and length
integer check	Integer types and ranges

Know first and then use: This lesson only treats record as a read-only data box. The field is String or int, and the field value will not be changed after creation; it will be replaced with a new box when new configuration is applied.

Complete RuleSettings.java · Complete file

```
package de.teacherma.lesson36;

import org.bukkit.configuration.file.FileConfiguration;

public record RuleSettings(String welcome, int bread,
                           int seconds, String world) {
    public static RuleSettings read(FileConfiguration c) {
        String welcome = text(c, "welcome-message");
        int bread = integer(c, "starter-bread", 0, 64);
        int seconds = integer(c, "ore-rush.duration-seconds", 1, 3600);
        String world = text(c, "ore-rush.world");
        return new RuleSettings(welcome, bread, seconds, world);
    }
    private static String text(FileConfiguration c, String key) {
        Object value = c.get(key);
        if (!(value instanceof String s) || s.isBlank() || s.length() > 80)
            throw new IllegalArgumentException(key + " Yes. 1 to 80 non-empty literal of word");
        return s;
    }
    private static int integer(FileConfiguration c, String key, int min, int max) {
        Object value = c.get(key);
        if (!(value instanceof Integer n) || n < min || n > max)
            throw new IllegalArgumentException(key + " Requires integer " + min + ".." + max);
        return n;
    }
}
```

07 / 15

Item list of configuration station

The project follows the build.gradle.kts and Java21 settings of Lesson 31, and the rootProject.name of settings.gradle.kts is changed to BlockAcademyL36. The two Java files are placed in the lesson36 package; the two yml files are placed in the resources directory.

Item list of configuration table

Element	Explanation
Two Java files	Rule class and main class
plugin.yml	Identity, Commands and Permissions
config.yml	Default settings for first time installation

First determine the management authority: the authority node defaults to op. Ordinary players cannot receive unlimited bread on their own, nor can they change configurations; during testing, please ask a teacher to practice on

the local server.

Complete plugin.yml · Complete file

```
name: BlockAcademyL36
version: 1.0.0
main: de.teacherma.lesson36.RuleDeskPlugin
api-version: '1.21.11'
author: TeacherMa
website: https://www.teacher-ma.de
commands:
  rules:
    description: View and test classroom configurations
    usage: /rules <show|reload|snack>
permissions:
  blockacademy.rules.admin:
    default: op
```

Java file directory · See the description in this section for the location

```
src/main/java/de/teacherma/lesson36/
  RuleSettings.java
  RuleDeskPlugin.java
```

08 / 15

Configure the upper half of the main class

Create a new RuleDeskPlugin.java, put this page first, and then the next page. candidate.load will hand over reading or YAML format issues to catch; do not mistake the empty configuration after failure as a valid new rule.

Configure the upper half of the main class

Element	Explanation
Prepare for configuration	Copy only if it does not exist
Load candidates	Capture read, write or parse failure
Initialization command	Only continue if the startup configuration is qualified.

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

```
package de.teacherma.lesson36;

import java.io.File;
import java.io.IOException;
import java.util.Objects;
import java.util.logging.Level;
import org.bukkit.Material;
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.inventory.ItemStack;
import org.bukkit.plugin.java.JavaPlugin;

public final class RuleDeskPlugin extends JavaPlugin {
    private RuleSettings rules;
    @Override
    public void onEnable() {
        saveDefaultConfig();
        if (!loadRules()) {
            getServer().getPluginManager().disablePlugin(this);
            return;
        }
        Objects.requireNonNull(getCommand("rules")).setExecutor(this);
    }
}
```

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

```
    }
    private boolean loadRules() {
        try {
            var candidate = new YamlConfiguration();
            candidate.load(new File(getDataFolder(), "config.yml"));
            RuleSettings next = RuleSettings.read(candidate);
            if (getServer().getWorld(next.world()) == null)
                throw new IllegalArgumentException("World not loading: " + next.world());
            rules = next;
            return true;
        } catch (IOException | InvalidConfigurationException
            | IllegalArgumentException ex) {
            getLogger().log(Level.WARNING, "Configuration not applied, please check config.yml", ex);
            return false;
        }
    }
}
```

09 / 15

Configure the lower half of the main class

Continue to the previous page, retaining all curly braces. show displays the current rules; reload tries to change the rules; snack tests the actual rewards. The unknown parameters are only explained and cannot be easily distributed to a cart of bread.

Configure the lower half of the main class

Element	Explanation
show	See the current effective rules
reload	Renewal rules only after passing the test
snack	Demonstration rewards by configuration

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

```
}
@Override
public boolean onCommand(CommandSender sender, Command command,
                        String label, String[] args) {
    if (!sender.hasPermission("blockacademy.rules.admin")) {
        sender.sendMessage("This classroom test requires teacher authorization");
        return true;
    }
    if (args.length != 1) {
        sender.sendMessage("Usage: /rules show or reload or snack");
        return true;
    }
    switch (args[0]) {
        case "reload" -> sender.sendMessage(loadRules()
            ? "The new configuration has taken effect" : "Configuration failed, continue to
> use the last valid rule");
        case "show" -> {
            sender.sendMessage(rules.welcome());
            sender.sendMessage("Bread.=" + rules.bread() + ", Duration=" + rules.seconds()
                + "seconds, world=" + rules.world());
        }
        case "snack" -> {
            if (!(sender instanceof Player p)) {
                sender.sendMessage("Please test the bread in the game");
                return true;
            }
        }
    }
}
```

Main Class Part 2-This paragraph 2/2, sequential connection

```
        if (!p.getWorld().getName().equals(rules.world())) {
            p.sendMessage("Please go to the world specified by the configuration first");
            return true;
        }
        if (rules.bread() == 0) { p.sendMessage("Bread reward has been closed"); return true;
> }

        var left = p.getInventory().addItem(new ItemStack(Material.BREAD, rules.bread()));
        left.values().forEach(item -> p.getWorld().dropItemNaturally(p.getLocation(), item));
        p.sendMessage(left.isEmpty() ? "The bread has been put into the backpack" : "The
> remaining bread is at your feet");
    }
    default -> sender.sendMessage("Parameters can only be show, reload, snack");
}
return true;
}
}
```

10 / 15

Zero reward and full backpack are also considered normal situations

0 pieces of bread means that the teacher has turned off the reward, and there is no need to create a pile of items with a quantity of 0. Even if the backpack is full, it is not the end of the world for the program: addItem will return the part that cannot be loaded. Practice example: Place the remaining items at the player's feet and prompt them accordingly.

Zero rewards and full backpacks are also considered normal situations

Element	Explanation
Quantity 0	Rewards disabled; do not create an item stack with quantity 0
Backpack can accommodate	Normal packet entry
The backpack cannot fit	Only process the remaining part

Test site: Test full backpack in flat open space. Rewards on the ground may be picked up by others or disappear; formal gameplay should have separate rules for collection, cooling and overflow. snack only demonstrates reading configuration.

Process the closed state first. See the description in this section for the location.

```
if (rules.bread() == 0) {  
    p.sendMessage("Bread reward has been closed");  
    return true;  
}
```

Only put the remaining part on the ground. See the instructions in this section for the location.

```
left.values().forEach(item ->  
    p.getWorld().dropItemNaturally(p.getLocation(), item));
```

11 / 15

Change three into four with your own hands

Build and install first, enter the game run/rules show, and confirm that the bread is 3. Only modify the starter-bread of the data directory to 4, save the file, and then execute /rules reload. Finally, use snack to verify that there is actually one more piece, instead of just trusting the success prompt on the screen.

Change three into four with your own hands

Step	Explanation
1. Change disk file	Bread 3 is changed to 4
2. Reread explicitly	Execute rules reload
3. Verify actual reward	Show and snack are checked together

Record evidence: save three sets of results "before modification, after executing reload, and actual receipt". Older JAR versions and wrongly edited files can also cause "I obviously changed it but it didn't change".

Build and practice commands · See the description in this section for the location

```
.\gradlew.bat clean build  
/rules show  
/rules reload  
/rules snack
```

Checklist

Modification	Expected
3 changed to 4	The new configuration takes effect, and the reward is 4 yuan
Set to 0	Close the prompt and do not send items
Set to 65 or negative number	Reread failed, old rules remain unchanged
Change to "4"	Reject literal numbers
The world name is spelled incorrectly	Deny, old world rules still valid

12 / 15

Let the error indicate which line to fix

It is not enough to report an error saying "configuration failed". The rule check writes the key name into the exception information; the main class records the exception and stack, and the chat bar only gives brief feedback. Find the specific configuration key first, and then let someone with administrative rights decide when to reread it.

Let the error indicate which line to fix

Element	Explanation
Format error	Check for indented colon quotes
Rule error	Exception points to specific key
Loading error	Keep logs and old rules

Save old files: Copy a small configuration for backup before editing. Don't swallow exceptions directly to eliminate red letters; clearing the log does not mean that the problem disappears.

Keep abnormal clues · See the description in this section for the location

```
getLogger().log(Level.WARNING, "Configuration not applied, please check config.yml", ex);
```

Checklist

Phenomenon	Check first
YAML parsing error	Indentation, colon, quotation marks
Requires integer 0..64	Whether the field is an integer and within the range
World not loading	World name and server actual world
Command not available after initial startup	Whether the enable log shows configuration failure
The value remains the same after reload	Whether a failure prompt appears and whether the wrong file is corrected

13 / 15

Give the seconds to the next event

Lesson 35: Write 60 in the code. Now when starting the activity, copy the verified seconds to the local variable. If I change the configuration after the current game starts, should it affect the next game, or should I change the current game immediately? Write the rules clearly first, then write the code.

Give the seconds to the next event

Element	Explanation
Snapshot of this field	Opening copy verified seconds
Change configuration on the way	This site is still in accordance with the original contract
Next game	Adopt new value that takes effect later

Scale boundary: practicing synchronous rereading of small files is easy to understand, but I/O will occupy the main thread. Large configurations should parse independent data asynchronously and then return to the main thread to verify the world and replace it. The world cannot be accessed directly in the asynchronous thread.

Take a snapshot at the beginning. See the description in this section for the location.

```
int thisRoundSeconds = rules.seconds();
// Use this value to initialize the countdown of this field
// Don't do it in every tick Re-read the file.
```

Expectations for both games · Locations are explained in this section

At the beginning of the first game: 60 seconds
Modify during operation to: 90 seconds
The first game is maintained 60 seconds, next field use 90 seconds

- The rules plug-in for this lesson only displays the duration and will not automatically start the competition.
- Lesson 38 will create a competition plug-in that actually uses duration configuration.
- Reading small configurations only occurs at startup and when the administrator explicitly rereads; do not read the disk every time you mine.

14 / 15

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	Change welcome message
Iron Pickaxe	Prove that bad configurations do not overwrite old rules
Diamond Pickaxe	Design new fields and boundaries

Next stop: Lesson 37: Add an identity tag to the miner's compass that does not depend on the display name.
Official extended reading: docs.papermc.io/paper/dev/plugin-configurations/

- Wooden Pickaxe: Configure your own welcome message for show, and explain the difference between templates and effective files.
- Iron Pickaxe: Complete 0, 64, 65, negative numbers, alphanumeric and wrong world name tests, the old rules remain unchanged after failure.
- Diamond Pickaxe: Design the three-level bread reward configuration, first write the field and range description, and then add a read check.
- Acceptance requirements: Test missing keys, error types, negative numbers and boundary values; after reloading a bad configuration, prove that the last valid setting is still in effect.

Checklist

I can complete it independently	Mastered	Practice again
Can find the config.yml being used	☐	☐
Type range and world will be checked	☐	☐
Can explain when candidate rules are replaced	☐	☐

15 / 15

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/>