

JAVA / LESSON 32 OF 40

Add Server Commands

Distributing toolkits with parameters, permissions and completions

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. | 100-120 minutes



/kit toolkit with parameter checking

A /kit command first checks the caller, permissions and parameters before putting items into the backpack. You'll take care of player, console, and full inventory situations at the same time, so failures are clearly responded to.

Front backpack: Gradle project and Paper test server of Lesson 31. Copy the project to BlockAcademyL32, keep the build script, and change the project name in settings to BlockAcademyL32.

Before you start: It is recommended to complete Lesson 31 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=32&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. First recognize who is giving the order
03. Count parameters first and then open the grid
04. The command list and permissions must be matched
05. Where to go when the backpack is full of items?
06. Press Tab to give the player a hint
07. The tool kit formula is first written on paper
08. Toolkit program first half
09. The lower part of the toolkit program
10. Walk through the failure path as well
11. Make commands easier to maintain
12. Expansion challenges and own acceptance records
13. Leave the evidence for the next adventure

01 / 13

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 / 13

First recognize who is giving the order

The villager entered the command to receive the tool kit at the console, but the program asked him for the backpack. The console has no hands and no hotbar. In the first level of the command, first identify the caller, then count the parameters, and finally open the recipe box.

Treat the command as a gated route: the backpack is modified only after the caller type, authority, number of parameters, and parameter content have all passed. Failure is also a process that requires a clear reply. Don't let players only see silence.

First recognize who is giving the order

Element	Explanation
Player	Can receive a backpack
Console	Obtained an explicit rejection
Common rules	Only change items after checking

Return value: onCommand returns true to indicate that this command has been processed, including explained failures. Returning false usually lets the server send the usage in plugin.yml.

Obtain players safely. See the instructions in this section for their positions.

```
if (!(sender instanceof Player player)) {
    sender.sendMessage("Please collect the tool kit in the game");
    return true;
}
player.sendMessage(Component.text("Backpack owner confirmed"));
```

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

Wrong cast · Intentional error, predict first and then fix

```
Player player = (Player) sender; // Console execution will cause errors
```

03 / 13

Count parameters first and then open the grid

When entering /kit miner, kit is the command name, and there is only miner in args; args[0] is not kit. When the player only types /kit, the array length is 0, and reading the 0th cell is out of bounds.

Count the parameters first and then open the grid

Step	Explanation
1. Count parameters first	Length must be 1
2. Read the 0th box again	It is a miner, not a kit
3. Finally recognize the formula	Unknown options explicitly rejected

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

Condition first: check the length first, then read the element. The more parameters, the smarter it is. There must be clear rules for the extra words.

Only accepts one parameter. See the description in this section for the position.

```
if (args.length != 1) {  
    player.sendMessage(Component.text("Usage: /kit miner or /kit builder"));  
    return true;  
}
```

Comparison that is not affected by system language · See the description in this section for the location

```
String name = args[0].toLowerCase(Locale.ROOT);
```

Checklist

Enter	Length	Result
/kit	0	Show usage
/kit miner	1	Select miner package
/kit miner extra	2	Show usage
/kit wizard	1	Prompt unknown toolkit

04 / 13

The command list and permissions must be matched

Create lesson32 package, the main class is called KitPlugin, replace plugin.yml in the resource directory. The exercise allows players to receive it by default, and the code still checks permissions to facilitate the withdrawal of a player's qualifications by the permission management system in the future.

The command list and permissions must be matched

Element	Explanation
command kit	List corresponds to getCommand
Authority node	Code checks before execution
Default rules	This chapter defaults to true

Select processing location: In this lesson, the code uniformly checks permissions and sends Chinese prompts. There is no permission written under commands.kit. default: true is the practice rule for this lesson and is not the default choice for all servers.

Complete plugin.yml of this lesson · Complete file

```
name: BlockAcademyL32
version: 1.0.0
main: de.teacherma.lesson32.KitPlugin
api-version: '1.21.11'
author: TeacherMa
website: https://www.teacher-ma.de
commands:
  kit:
    description: Get classroom tool kit
    usage: /kit <miner|builder>
permissions:
  blockacademy.kit:
    default: true
```

Pre-execution check authority · See the description in this section for the location

```
if (!player.hasPermission("blockacademy.kit")) {
    player.sendMessage(Component.text("You have not received the permission yet"));
    return true;
}
```

05 / 13

Where to go when the backpack is full of items?

addItem attempts to put the item into the backpack and returns the part that was not put in. The returned Map records the remaining items, and the key is the serial number of the input item, not the backpack slot. You can't just say "received successfully" and forget about the remaining items.

Where to go when the backpack is full of items?

Step	Explanation
1. Try to enter the package	addItem modifies backpack
2. Get the remainder	Return the items in the Map
3. Handle truthfully	Dropped at feet and prompted

Practice strategy: In this lesson, drop the remaining items at the player's feet. Players nearby may pick them up, and the dropped items may also disappear. The test is conducted in a flat open space; the official server can be changed to "vacate the position first and then receive it".

Send items and process the remaining items. See the description in this section for the location.

```
var leftovers = player.getInventory().addItem(  
    new ItemStack(Material.TORCH, 16));  
for (ItemStack item : leftovers.values()) {  
    player.getWorld().dropItemNaturally(player.getLocation(), item);  
}
```

Feedback must conform to the results. See the description in this section for the location.

```
String message = leftovers.isEmpty()  
    ? "Put all into backpack" : "The remaining items fell at your feet";  
player.sendMessage(Component.text(message));
```

06 / 13

Press Tab to give the player a hint

Command completion only recommends possible inputs, and parameters still need to be checked during actual execution. Otherwise, players can bypass rules that only exist in menus by manually entering a strange name.

Press Tab to give the player a hint

Element	Explanation
Completion prompt	Only legal names are recommended
Perform verification	Manual input still needs to be checked
Unknown prefix	Return empty list

Review Lesson 30: filter here retains matching names, and toList collects results. No additional suggestions are provided when the completion method returns an empty list.

Filter completion items from the list · See the description in this section for the location

```
String prefix = args[0].toLowerCase(Locale.ROOT);
return List.of("miner", "builder").stream()
    .filter(name -> name.startsWith(prefix)).toList();
```

Do not access parameters that do not exist. See the description in this section for the location.

```
if (args.length != 1) return List.of();
if (!sender.hasPermission("blockacademy.kit")) return List.of();
```

Checklist

Current parameters	Recommend
Empty string	miner and builder
m	miner
b	builder
z	Empty list

07 / 13

The tool kit formula is first written on paper

Today's challenge is two kits. First write down the recipe clearly, and then string together input, judgment, distribution and feedback. The practice version can be obtained repeatedly; the cooling function will be added after learning the 34th lesson.

Toolkit formula is first written on paper

Element	Explanation
miner	1 stone pickaxe and 16 torches
builder	32 wooden boards, 16 glass
Unknown name	Stop issuing after returning null

Empty results will not be issued: if items are null, it means an unknown recipe. Send instructions and return first, and then use the backpack logic. The next page and the page after that together are a complete source document.

Use switch to select formula · See the description in this section for the location

```
ItemStack[] items = switch (name) {  
    case "miner" -> new ItemStack[] {  
        new ItemStack(Material.STONE_PICKAXE),  
        new ItemStack(Material.TORCH, 16)};  
    default -> null;  
};
```

Checklist

Toolkit	Item one	Item two
miner	1 stone pick	16 torches
builder	32 oak planks	Glass 16 pieces

08 / 13

Toolkit program first half

Create new src/main/java/de/teacherma/lesson32/KitPlugin.java. Copy this page first, then the next page immediately after; do not include package names, imports, or extra braces.

Toolkit program first half

Element	Explanation
A source document	The front and rear parts are connected sequentially
Check first	Types, Permissions and Parameters
After passing	Just enter formula and distribution

Errors should be discovered as early as possible: `getCommand` is null usually indicating that the command name in the resource is not correct. `requireNonNull` will give a clear clue at startup.

KitPlugin Part 1 · This section 1/2, sequential connection

```
package de.teacherma.lesson32;

import java.util.List;
import java.util.Locale;
import java.util.Objects;
import net.kyori.adventure.text.Component;
import org.bukkit.Material;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.entity.Player;
import org.bukkit.inventory.ItemStack;
import org.bukkit.plugin.java.JavaPlugin;

public final class KitPlugin extends JavaPlugin {
    @Override
    public void onEnable() {
        var kit = Objects.requireNonNull(getCommand("kit"), "Please check plugin.yml");
        kit.setExecutor(this);
        kit.setTabCompleter(this);
    }

    @Override
    public boolean onCommand(CommandSender sender, Command command,
                            String label, String[] args) {
        if (!(sender instanceof Player player)) {
```

KitPlugin Part 1 · This paragraph 2/2, sequential connection

```
        sender.sendMessage("Please collect the tool kit in the game");
        return true;
    }
    if (!player.hasPermission("blockacademy.kit")) {
        player.sendMessage(Component.text("You have not received the permission yet"));
        return true;
    }
    if (args.length != 1) {
        player.sendMessage(Component.text("Usage: /kit miner or /kit builder"));
        return true;
    }
}
```

09 / 13

The lower part of the toolkit program

The following page follows immediately. Distribution, remaining items, and replenishment are all placed in the same class. Keep the boundaries between the two `@Override` methods, and don't let the creepers count the number of braces for you.

Toolkit program lower half

Element	Explanation
Distribution	addItem and process the rest
Feedback	Explain actual results
Completion	Filter by current prefix

Build and deploy: Use the build script of Lesson 31 and the plugin.yml of this lesson. Run `.\gradlew.bat clean build`, put the BlockAcademyL32 JAR into the test server and restart.

KitPlugin Part 2 · This paragraph 1/2, sequential connection

```
ItemStack[] items = switch (args[0].toLowerCase(Locale.ROOT)) {
    case "miner" -> new ItemStack[] {
        new ItemStack(Material.STONE_PICKAXE),
        new ItemStack(Material.TORCH, 16)};
    case "builder" -> new ItemStack[] {
        new ItemStack(Material.OAK_PLANKS, 32),
        new ItemStack(Material.GLASS, 16)};
    default -> null;
};
if (items == null) {
    player.sendMessage(Component.text("Only recognize miner and builder"));
    return true;
}
var leftovers = player.getInventory().addItem(items);
for (ItemStack item : leftovers.values()) {
    player.getWorld().dropItemNaturally(player.getLocation(), item);
}
player.sendMessage(Component.text(leftovers.isEmpty()
    ? "The tool kit has been placed in the backpack" : "The backpack is full and the
> remaining items fall at my feet"));
return true;
}

@Override
public List<String> onTabComplete(CommandSender sender, Command command,
    String alias, String[] args) {
```

KitPlugin Part 2 · This paragraph 2/2, sequential connection

```
if (!sender.hasPermission("blockacademy.kit") || args.length != 1)
    return List.of();
String prefix = args[0].toLowerCase(Locale.ROOT);
return List.of("miner", "builder").stream()
    .filter(name -> name.startsWith(prefix)).toList();
}
}
```

10 / 13

Walk through the failure path as well

First free up the backpack to do a successful test, and then actively create empty parameters, unfamiliar parameters and a full backpack. The goal of testing is to know what will happen with each input. You can't just declare that you have passed the level just because "I got a pickaxe once."

Also walk through the failed path

Element	Explanation
Normal	Both packages comply with the formula
Border	Zero parameters, full backpack
Reject	Unknown options, console, no permissions

Permission test: Allowed to receive by default. If the teacher already has permission to plug-in, he can explicitly deny blockacademy.kit and then verify that the items will not be sent; do not misjudge the default permission as the code is not checked.

Checklist

Operation	Expected
player/kit miner	1 stone pickaxe and 16 torches
player/kit builder	32 wood panels and 16 glass
Console kit miner	Prompt to receive it in the game, no exception
/kit or /kit wizard	Give clear feedback and do not send items
Collect after the backpack is filled	There are remaining items at your feet, the information is accurate
Enter /kit m and press Tab	Complete miner

11 / 13

Make commands easier to maintain

When there are more and more commands, each branch needs to write a huge distribution code, and repairing it once is like digging an entire mountain. Divide the recipe into a method, and the command method is only responsible for checking and calling.

Stop and think about it: What judgment should be completed before reading args[0]? Why can't zero parameters be remedied by the subsequent recipe branch?

Make commands easier to maintain

Element	Explanation
Clear responsibilities	Command responsible for inspection
Formula method	Only return item list
Test synchronization	Changing the rules also changes the expectations

Figure this out: Check `args.length` first. The out-of-bounds occurrence occurs when reading, and subsequent branches have not yet had a chance to be executed; return early after explaining the usage.

Extract a recipe method. See the description in this section for the location.

```
private ItemStack[] minerKit() {  
    return new ItemStack[] {  
        new ItemStack(Material.STONE_PICKAXE),  
        new ItemStack(Material.TORCH, 16)};  
}
```

The help messages must also be consistent. See the description in this section for the location.

```
player.sendMessage(Component.text("Usage: /kit miner or /kit builder"));  
// plugin.yml Medium usage Write the same selection range
```

- Change the case to MINER and expect the miner package to still be recognized.
- Changed unknown option to wizard, no expected item changes.
- Change the tool kit quantity to the new value in the recipe table, and then update the acceptance record.

12 / 13

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	Implement name greetings
Iron Pickaxe	Complete two kinds of packages and failure records
Diamond Pickaxe	Increased independent cooldown per player

Next stop: Lesson 33: Use events to manage which blocks can be mined.

Official extended reading: jd.papermc.io/paper/1.21.11/org/bukkit/command/CommandExecutor.html

- Wooden Pickaxe: Implement /hello, sending greetings with names to players; the console can also get reasonable prompts.
- Iron Pickaxe: Complete two tool kits, submit empty parameters and full backpack test records.
- Diamond Pickaxe: After learning the 34th expansion, receive the command to add a 30-second cooldown for each player, and verify that the two players do not affect each other.
- Acceptance requirements: Use players and consoles to test separately, and complete scenarios of no permissions, zero parameters, unknown formulas, and full backpacks; items will not be issued if the confirmation fails.

Checklist

I can complete it independently	Mastered	Practice again
First check the sender and parameters	☐	☐
Can handle the remaining items in the backpack	☐	☐
Able to explain authority and completion	☐	☐

13 / 13

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/>