

JAVA / LESSON 12 OF 40

Pack Recipes into Methods

Method calling and reuse

This chapter uses JDK 21 and does not require the preview function. | 90-110 minutes

Make a five-step mine preparation program

Will be able to load a section of instructions into a named method; be able to call methods from main and explain the execution sequence; be able to identify duplicate codes and refactor safely; and finally break down the mine preparation into five clear small skills.

Pre-Knapsack: Variables, Conditions, Loops, and the Pixel House of Lesson 11. The method will tidy up the existing syntax and will not summon new diamonds out of thin air.

Before you start: It is recommended to complete Lesson 11 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=12&lang=en>

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

From repeated instructions to clear processes

Step	Explanation
1. Extraction method	Define a preparation step
2. Call by name	main arrangement sequence
3. Check results	Behavior unchanged after reconstruction

In this lesson

01. Before you begin: Put the examples in the right place
02. Do you need to copy the recipe again every time?
03. Name a group of instructions
04. Define and call void method
05. Calling sequence and return route
06. A method only does one thing
07. Divide the pixel house into skills
08. Reuse methods and small tests
09. Method Failure Clinic
10. Challenge Blueprint Mine Preparation Process
11. Challenge Implementation Five Methods Queuing
12. Expansion tasks and clearance review
13. Own acceptance record

01 / 13

Before you begin: Put the examples in the right place

This chapter uses JDK 21 and does not require the preview function. A complete program is given for each common example, saved with annotated file names. Multiple examples are placed in their own folders to avoid auxiliary classes with the same name.

Terminal command: Replace the Chinese placeholder with the actual name

```
javac -encoding UTF-8 ExampleFile.java
java ExampleClass
```

Class names are case-sensitive. If the complete challenge code is divided into multiple sections, please connect them in the same file in order. Do not create separate classes for the continued sections. Run verification only lists the program output; additional input sequences are provided where input is required.

Learning action: first write down your prediction, compare it after running, and then change only one condition. Experiments marked "Intentional Bug" are used to identify bugs; they are clearly distinguished from ready-to-run examples.

02 / 13

Do you need to copy the recipe again every time?

A Fang is responsible for the pre-departure inspection. He copied "check the pickaxe, load the food, and bring the torch" three times: once during the day, once at night, and again according to the instructions of his deskmate. As a result, the number of torches was changed in two places, and the third place remained at the old version.

A method is like a recipe card with a name. Write a set of actions once and call them with method names when needed. When modifying the rules, only the recipe card needs to be modified, and there is no need to chase three copy-pasted monsters.

The workbench is repeating code

Element	Explanation
Repeat	Save the same code in multiple places
Risk	One omission was corrected and the behavior was inconsistent
Method	Define once, call multiple times

Starting with four questions

- What is the difference between method definition and invocation?
- Why is main still the birth point of the program?
- How to give a method an understandable action name?
- How to prove that the behavior has not changed after refactoring?

World Setting: This lesson still uses the Java 21 console program. Methods are responsible for organizing code and do not directly modify the real Minecraft world.

03 / 13

Name a group of instructions

First find actions that are exactly the same or have the same concept, and then move them into methods. Method names start with verbs, such as packFood, checkArmor, printDivider.

Repeated code becomes a recipe card

Element	Explanation
Record first	Save original output
Extraction	Repeated Action Move In Method
Compare again	Call twice, check six lines

L12Example02.java · Complete independent example

```
public class L12Example02 {  
    static void prepareMining() {  
        System.out.println("Check pick");  
        System.out.println("Load 16 bread");  
        System.out.println("Bring on 64 Root torch");  
    }  
    public static void main(String[] args) {  
        prepareMining();  
        prepareMining();  
    }  
}
```

Output

```
Check pick  
Load 16 bread  
Bring on 64 Root torch  
Check pick  
Load 16 bread  
Bring on 64 Root torch
```

Refactoring goal: reduce duplication without changing the output. First record the old output, then extract the method, and finally compare the old and new results line by line.

04 / 13

Define and call void method

Defining the method is just to write the recipe in the book; calling the method is to actually start working with the materials. The following classes can be compiled and run directly.

Method consists of method header and method body

Element	Explanation
static	This example can be called directly from main
void	Execute action, do not return value
Method name	Such as packFood, describe the action
Method body	Instructions within curly braces

L12Example03.java · Complete independent example

```
public class L12Example03 {  
    static void packFood() {  
        System.out.println("Load 16 bread");  
    }  
    static void checkArmor() {  
        System.out.println("Check Armor Durability");  
    }  
    public static void main(String[] args) {  
        packFood();  
        checkArmor();  
    }  
}
```

Output

```
Load 16 bread  
Check Armor Durability
```

Pronunciation: See packFood(); read as "call packFood method". The closing parentheses and semicolons cannot be left behind.

05 / 13

Calling sequence and return route

When Java encounters a method call, it will temporarily leave the current location and enter the called method. After the method is executed to the right curly brace, the program returns to the next line where it was called.

main issues a task and returns after the method is completed

Element	Explanation
main	Print and start
ringBell	Call once and print a ding
Recall	Print again
Continue	Return to main and print the mine

L12Example04.java · Complete independent example

```
public class L12Example04 {  
    static void ringBell() { System.out.println("Ding"); }  
    public static void main(String[] args) {  
        System.out.println("Departure");  
        ringBell();  
        ringBell();  
        System.out.println("Mine");  
    }  
}
```

Output

```
Departure  
Ding  
Ding  
Mine
```

Prediction skills: first only look at the calling sequence of main, and then expand each method separately. Don't scrunch up the entire code and read it.

06 / 13

A method only does one thing

doEverything() sounds powerful, but is actually like a chest stuffed with rotten flesh, diamonds, and seventeen wooden hoes. The best method is to complete a clear action and describe it with verbs.

The method name is the label outside the box

Element	Explanation
Check equipment	checkEquipment
Pack food	packFood
Select mine	chooseMine
Arrangement sequence	prepareTrip calls three steps

L12Example05.java · Complete independent example

```
public class L12Example05 {  
    static void checkEquipment() { System.out.println("Check equipment"); }  
    static void packFood() { System.out.println("Pack food"); }  
    static void chooseMine() { System.out.println("Select mine"); }  
    static void prepareTrip() {  
        checkEquipment();  
        packFood();  
        chooseMine();  
    }  
    public static void main(String[] args) {  
        prepareTrip();  
    }  
}
```

Output

```
Check equipment  
Pack food  
Select mine
```

Naming check

- Use humpback for method names, such as checkArmor.
- The name should describe the action, do not write method1.
- When the method is too long, look for small steps that can be individually named.

Judgment criteria: If you can't explain what a method does in a short sentence, it probably takes on too many tasks.

07 / 13

Divide the pixel house into skills

The pixel house in the previous lesson can be reconstructed as a "parameter-free version" first: the roof is responsible for the first two rows, and the walls are responsible for the remaining five rows and the doors and windows are judged within the grid. In the next lesson, we'll use parameters to let the same method handle different sizes.

The roof and wall are divided, and the doors and windows stay in the correct grid

Element	Explanation
printRoof	Output the first two rows of roofs
printBody	Process walls, doors, and windows frame by frame
main	Call two methods in sequence
Check	The output is consistent with Lesson 11

PixelMethods.java · Complete standalone example

```

public class PixelMethods {
    static void printRoof() {
        System.out.println("  ^^^^ ");
        System.out.println(" ^^^^^^ ");
    }
    static void printBody() {
        for (int row = 2; row < 7; row++) {
            for (int col = 0; col < 9; col++) {
                char block = ' ';
                if (col == 4 && row >= 3 && row <= 5) block = 'D';
                else if ((col == 1 || col == 7)
                        && (row == 3 || row == 4)) block = 'O';
                else if (row == 2 || row == 6 || col == 0 || col == 8)
                    block = '#';
                System.out.print(block);
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        printRoof();
        printBody();
    }
}

```

Output

```

  ^^^^
 ^^^^^^
#####
#O  D  O#
#O  D  O#
#   D   #
#####

```

Doors and windows must be determined in the corresponding grid. You cannot print the entire wall first and then add the door to a new line. This reconstructed version is consistent character by character with Lesson 11: 9 characters per line, 7 lines total.

Boundary reminder: This lesson starts with practicing organization code. The fixed-string version is good for refactoring exercises, but truly variable-size construction still requires parameters and loops.

08 / 13

Reuse methods and small tests

Multiplexing is not blind multiple calls. Each call should have a clear location, and the method itself should be individually inspectable.

Write once and call multiple places

Element	Explanation
divider	Define a dividing line
First call	Print before title
Second call	Print after title
Small test	First verify a method

L12Example07.java · Complete independent example

```
public class L12Example07 {  
    static void divider() { System.out.println("====="); }  
    public static void main(String[] args) {  
        divider();  
        System.out.println("Block Academy");  
        divider();  
    }  
}
```

Output

```
=====  
Block Academy  
=====
```

Three types of inspections

- Normal: All methods are called as planned.
- Boundary: Call only one method or call it twice.
- Error: Intentionally misspelled method name, observe compiler prompts.

Reuse does not mean hiding: method names should let readers know what is happening; don't hide ten mysterious steps into go().

09 / 13

Method Failure Clinic

Method bugs are often divided into two class: the compiler blocks them directly, or the program can run but takes one less step. The second type is more disguised, so check the call list.

Three common method bugs

Element	Explanation
Fewer calls	Compilation successful, action missing
Nested methods	Java rejects this statement
Name misspelled	Unable to find method
Repair	Only change one question at a time

L12Example08.java · Complete independent example

```
public class L12Example08 {  
    static void packFood() { System.out.println("Pack food"); }  
    public static void main(String[] args) {  
        System.out.println("Departure");  
    }  
}
```

Output

Departure

This program can be compiled, but it does not contain food: the definition method will not be executed automatically. Fixed by adding packFood(); before "departure".

Fault experiment: Java does not allow methods to be declared in the method body. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadNestedMethod.java · Intentional error example

```
public class BadNestedMethod {  
    public static void main(String[] args) {  
        static void checkArmor() { }  
    }  
}
```

Emergency sequence: first confirm that the method is written inside the class and outside other methods; then check the name and parentheses; finally check whether main is really called.

10 / 13

Challenge Blueprint Mine Preparation Process

The challenge deliverable is MiningPrep.java. The program must contain five custom methods, called by main in a reasonable order.

Five methods form a starting pipeline

Element	Explanation
checkPickaxe	Check pick
packFood	Load bread
packTorches	Load torch
wearArmor	Put on armor
printPlan	Show route

Construction Contract

- Each of the five methods has clear action names.
- Each method outputs at least one piece of verifiable information.
- main is only responsible for arranging the process and does not repeat the output inside the method.
- Run it normally once, then delete a call to create a logic error.

Acceptance and inspection forms

Steps	Method	Acceptance output
1	checkPickaxe	Pickaxe durability checked
2	packFood	Food has been loaded
3	packTorches	Torch loaded
4	wearArmor	Armor has been put on
5	printPlan	Show the mine route

Minimum version: First write two methods and call them successfully, then add them one by one. By adding methods one at a time, bugs won't be playing hide and seek in five mines at the same time.

11 / 13

Challenge Implementation Five Methods Queuing

Create new MiningPrep.java. The class name and file name are consistent. These preparation methods require no input and do not return a value. void means there is no return value; static makes them callable directly from static main. The two are different design choices.

Complete mine preparation procedure

Element	Explanation
Definition	Five methods to save instructions
Call	main arranges five calls
Execute	Return to main after each completion
Acceptance	Output five pieces of information in sequence

Acceptance: can be compiled; output five items; the calling sequence is reasonable; delete packTorches(); and then be able to point out which step is missing, and then restore and run.

MiningPrep.java · Complete standalone example

```
public class MiningPrep {
    static void checkPickaxe() {
        System.out.println("Pickaxe durability checked");
    }
    static void packFood() {
        System.out.println("Load 16 bread");
    }
    static void packTorches() {
        System.out.println("Load 64 Root torch");
    }
    static void wearArmor() {
        System.out.println("The armor has been put on");
    }
    static void printPlan() {
        System.out.println("Route: Main Mine Road -> Supply station -> Go home");
    }
    public static void main(String[] args) {
        checkPickaxe();
        packFood();
        packTorches();
        wearArmor();
        printPlan();
    }
}
```

Output

```
Pickaxe durability checked
Load 16 bread
Load 64 Root torch
The armor has been put on
Route: Main Mine Road -> Supply station -> Go home
```

12 / 13

Expansion tasks and clearance review

The job requires handing over the source code, a run result and a bug record. Just submitting screenshots doesn't count, because screenshots won't tell you where the semicolon is hiding.

The third-level task continues to organize the code

Element	Explanation
Wooden Pickaxe	Three prompting methods
Iron Pickaxe	House reconstruction, output unchanged
Diamond Pickaxe	Four-stage challenge process
Review	Explain the difference between definition and call

Third gear operation

- Wooden pick: write three prompt methods morningTip, nightTip, and dangerTip, and call them once from main.
- Iron pick: Put the roof and the wall containing doors and windows of the pixel house in Lesson 11 into the method respectively, keeping the output unchanged.
- Diamond Pickaxe: Design challenge battle process, including at least four methods of preparation, battle, second stage, and victory.

Acceptance and inspection forms

Self-evaluation	Can	Still need to practice
I can distinguish between definition and call	☐	☐
I can explain the order of execution	☐	☐
I can extract duplicate code into methods	☐	☐

Next lesson: Lesson 13 Add parameters and return values to methods. The same recipe card will be able to receive different materials and send the calculation results back to the calling place.

13 / 13

Own acceptance record

Write expectations before running, and record actual results after running. Normal cases show that the program does things; edge cases show that it knows when to stop; error cases show that it rejects unreasonable requests.

Write the evidence into the record

Category	Input of your choice	Expectation and actual
Normal	A common value or a successful operation	Is it consistent line by line?
Border	Minimum value, maximum value and adjacent value	Is there one less square and one more step?
Error	Illegal quantity, missing object or incorrect syntax	Explicitly reject? Is the status maintained?

- Write a bug as: triggering conditions, phenomena, causes, and minimal repairs.
- Organize a rule worth remembering and explain it in your own words. Don't just copy the definition.
- Change an example, keep a record of the test, and let the results prove your understanding.

Technical verification basis: Java SE 21 language specification, scope and access control (Chapter 6), classes and constructors (Chapter 8), method invocation (Chapter 15).

<https://docs.oracle.com/javase/specs/jls/se21/html/index.html>