

JAVA / LESSON 04 OF 40

The Miner's Calculator

Turn "is it enough" and "is it worth it" into verifiable calculations?

Java 11+ | 90-110 minutes



Make miner income calculator

With seven pieces of coal and ten wooden sticks, how many torches can be synthesized? How many sets can 150 cobblestones hold? How to deduct the cost from the income of a mining trip? These questions do not require the computer to guess the answers, only the rules need to be written as expressions. In this lesson, we will distinguish between arithmetic, comparison and logic, and arrange boundary tests for the results.

Before you start: It is recommended to complete Lessons 1-3 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=4&lang=en>

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

In this lesson

01. First clarify the problem, then choose the operator
02. Five computing tools: + - * / %
03. Priority: Who is calculated first?
04. Complete group and remainder, transport grid and box
05. Comparison: Use one question to get true and false
06. Logic: Connect the two answers
07. Math: Give common calculations a good name
08. Errors cannot be solved just by changing the symbols
09. First design the rules for mining revenue
10. Complete work: Check first, then calculate
11. Let boundary values check your ideas

01 / 11

First clarify the problem, then choose the operator



Calculating resources before entering the mine is more confident than guessing how long you can last in the dark.

To calculate a quantity, arithmetic operations are usually required; to ask "whether it is enough" requires comparison; when multiple requirements appear at the same time, logic must be used to connect them. The results obtained by the three operations are different. Don't confuse "how much" and "whether it is enough" into the same thing.

- How many torches? Get an integer.
- Is the profit at least 50? Gets true or false.
- Profits reached the target and diamonds were dug? Still get true or false.

The pricing unit and unit price in this lesson are practice rules. Write the data in variables first, and then learn keyboard input in Lesson 6; don't mistake the "initial data of the work" for being able to interact.

02 / 11

Five computing tools: + - * / %



By breaking down each calculation step, complex results will have traceable sources.

Snippet: place inside main and run separately

```
int coal = 7, sticks = 10, blocks = 150;  
System.out.println(coal + 3);  
System.out.println(sticks - coal);  
System.out.println(coal * 4);  
System.out.println(blocks / 64);  
System.out.println(blocks % 64);
```

Output

```
10  
3  
28  
2  
22
```

Addition records acquisition, subtraction records consumption, and multiplication processes "a few per group". For non-negative item quantities, use / to count complete sets and % to count remaining quantities. The calculation here is temporarily based on the practice rule of up to 64 items per group for one item.

"Coal" + 7 gets "Coal 7", there will not be one more space out of thin air. 7 + 3 is integer addition. When you need to calculate and output, put the arithmetic part in parentheses.

03 / 11

Priority: Who is calculated first?

Commonly used operators in this lesson, from high to low

Layer	Operator	Reading Tips
Brackets	()	First clarify the grouping
Negation	!	Flip true and false
Remainder of multiplication and division	* / %	The same layer is usually from left to right
Addition and subtraction	+ -	Also pay attention to string splicing
Size comparison	> >= < <=	get boolean
Equality comparison	== !=	get boolean
Logical AND	&&	Both are satisfied
Logical OR		At least one satisfies

Snippet: place inside main and run separately

```
System.out.println(2 + 3 * 4);  
System.out.println((2 + 3) * 4);  
int total = 3 * 12 + 5;  
System.out.println(total);
```

Output

```
14  
20  
41
```

Brackets are first for the reader to see. When an expression asks you to read it a second time, put the intention clearly in parentheses. The table only lists the commonly used operations in this lesson, and is not a complete priority list of the entire Java language.

04 / 11

Complete group and remainder, transport grid and box

$$150 = 2 \times 64 + 22$$

Element	Explanation
Group 1	64
Group 2	64
Remaining	22

Snippet: place inside main and run separately

```
int blocks = 150;
int fullStacks = blocks / 64;
int leftover = blocks % 64;
System.out.println(fullStacks + " Group, remainder " + leftover);
int slotsNeeded = (int) Math.ceil(65 / 64.0);
System.out.println("65 The number of grids required for each item=" + slotsNeeded);
```

Output

```
2 Group, remainder 22
65 The number of grids required for each item=2
```

The second result is the number of inventory slots, not the number of boxes. Here it is set that 64 items of an item are placed in each grid; boxes will have multiple grids, and to calculate the number of boxes, one must also know the number of grids in each box. Let's talk about the units first before the calculation makes sense.

To at least accommodate all items, round up. First write / 64.0 to get the decimal, and then use ceil; if you do integer division first, the missing fraction will not be recovered by ceil.

05 / 11

Comparison: Use one question to get true and false

Six comparisons, check item by item

Operator	diamonds = 4 hours	Result
>	4 > 2	true
>=	4 >= 4	true
<	4 < 10	true
<=	4 <= 3	false
==	4 == 4	true
!=	4 != 0	true

Snippet: place inside main and run separately

```
int diamonds = 4;
System.out.println(diamonds > 2);
System.out.println(diamonds >= 4);
System.out.println(diamonds < 10);
System.out.println(diamonds <= 3);
System.out.println(diamonds == 4);
System.out.println(diamonds != 0);
```

Output

```
true
true
true
false
true
true
```

= is assignment, == is equality comparison. Java cannot concatenate `0 < y < 64` into a range judgment. Instead, write `y > 0 && y < 64`. String content comparison will be studied specifically in Lesson 5.

06 / 11

Logic: Connect the two answers



AND, OR and NOT combine or reverse conditions. The redstone gates in the illustration help us remember what each operator does.

First look at the two inputs, then look at the results

A	B	A && B	A B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

!true is false; !false is true.

Snippet: place inside main and run separately

```
int torches = 24;
boolean hasPickaxe = true;
boolean ready = torches >= 20 && hasPickaxe;
boolean canStay = false || true;
System.out.println(ready);
System.out.println(canStay);
System.out.println(!ready);
```

Output

```
true  
true  
false
```

&& requires both left and right sides to be satisfied; || only needs to be satisfied on at least one side. They also short-circuit: when the left-hand side of && is false and the left-hand side of || is true, the right-hand side does not have to be evaluated. Lesson 5 will use this to avoid calling methods on null.

07 / 11

Math: Give common calculations a good name

Four commonly used tools

Method	Example	Result
Math.min	Math.min(7, 10)	7
Math.max	Math.max(18, 23)	23
Math.round	Math.round(12.6)	13 (long)
Math.ceil	Math.ceil(129 / 64.0)	3.0 (double)

Snippet: place inside main and run separately

```
int batches = Math.min(7, 10);  
int torches = batches * 4;  
long distance = Math.round(12.6);  
int slots = (int) Math.ceil(129 / 64.0);  
System.out.println(torches);  
System.out.println(distance);  
System.out.println(slots);
```

Output

```
28  
13  
3
```

The torch formula will be limited by a small number of materials, so first take the smaller value of coal and sticks, and then multiply by 4. round is used for the nearest integer in this case; ceil is used for capacity calculations where no fractions can be missed.

The return type should also be checked: in this case `Math.round(double)` gets long, `Math.ceil` gets double. The final hard transfer is only used for grid numbers within a known range and cannot replace all numerical verification.

08 / 11

Errors cannot be solved just by changing the symbols



To check the calculator, you need to distinguish between grammatical errors, operational errors and rule errors.

Three situations, three different clues

- Write `diamonds == 3` as `diamonds = 3`: query becomes assignment, intent changes.
- Integer division by 0: `ArithmeticException` will be thrown at runtime.
- Treat negative numbers as the number of ores: the grammar may be legal, but it does not comply with the rules of this work.

Snippet: place inside main and run separately

```
int players = 0;
boolean canDivide = players != 0;
System.out.println(canDivide);
```

Output

```
false
```

canDivide only saves the check results and does not automatically prevent subsequent divisions. The program must be allowed to choose actions based on the results. The complete work on the next page uses an if to guard the entrance. Lesson 7 will talk about it in detail.

Capacity calculation measures at least 0, 63, 64, 65: the required number of cells should be 0, 1, 1, 2. Don't just test the 150 you're already familiar with.

09 / 11

First design the rules for mining revenue



The same record must record income and consumption at the same time, and cannot just focus on the diamonds dug.

Four Steps to Profit Calculator

Step	Explanation
1. Initial data	Ore quantity Torches and food
2. Pricing	Revenue minus cost
3. Judgment	Profit at least 50 And there are diamonds
4. Report	Data, profit With judgment result

Set the practice unit prices of coal, iron, gold, redstone, and diamond to 1, 3, 5, 2, and 12 respectively. Torches cost 1 each, and food costs 2 each. The same “practice points” are used for unit prices to avoid mixing units.

- Income: multiply the quantity of each ore by the unit price, and then add them up.
- Cost: Number of torches plus twice the amount of food.
- Profit: income minus costs; losses are allowed.
- Invalid data: If the number is negative, a prompt will be given first.

10 / 11

Complete work: Check first, then calculate

Complete program · MinerCalculator.java

```
public class MinerCalculator {
    public static void main(String[] args) {
        int coal = 20, iron = 8, gold = 3;
        int redstone = 12, diamonds = 2;
        int torchesUsed = 16, foodUsed = 4;
        boolean valid = coal >= 0 && iron >= 0 && gold >= 0
            && redstone >= 0 && diamonds >= 0
            && torchesUsed >= 0 && foodUsed >= 0;
        // First borrow a conditional goalkeeper; 7 The class will unfold it.
        if (!valid) {
            System.out.println("The quantity must be a non-negative integer");
            return;
        }
        int income = coal + iron * 3 + gold * 5
            + redstone * 2 + diamonds * 12;
        int cost = torchesUsed + foodUsed * 2;
        int profit = income - cost;
        boolean richTrip = profit >= 50 && diamonds > 0;
        System.out.println("Income=" + income + ", Cost=" + cost);
        System.out.println("Profit=" + profit);
        System.out.println("Harvest=" + richTrip);
    }
}
```

Output

```
Income=107, Cost=24
Profit=83
Harvest=true
```

The content in if is only executed when the data is invalid; return ends main here. The core task is still to understand the expressions of valid, income, cost and richTrip, and to be able to calculate the same result by hand.

Let boundary values check your ideas

The ability map of the fourth lesson

Step	Explanation
1. Count quantity	+ - * / %
2. mark boundary	Brackets and units
3. Ask whether it is true or false	Comparison operation
4. With conditions	&& !
5. Check	Normal and borderline

- Basics: 389 cobblestones, a complete set of 6, and a remainder of 5; at least 7 squares are required when each square has 64.
- Advanced: Torches are at least 20 and have a mine pickaxe, and there is at least one type of food or bed. Write the judgment in parentheses.
- Creation: Design a discount formula and use `Math.max(1, calculated price)` to set your own minimum price rule.

Test the complete work

Set all ores to 0, retain consumption: profit should be -24, harvest should be false. Set coal to -1: only output quantity hints. Adjust the cost to make the profit exactly 50, and then observe the difference between diamonds being 0 and greater than 0.

Self-test: Are / and % interchangeable? Does `ceil` count the number of boxes? The answer is no; the units and rules must be explained first. In the next lesson, String Magic in Chat, learn to organize, compare, and clean up text.

Technical reference: Expressions in Chapter 15 of the Math API and Language Specification of Java SE 11. For official documents, see docs.oracle.com.