

JAVA / LESSON 10 OF 40

Keep Mining Until You Find Diamonds

while, do-while and tool durability simulator

Java 11+ | 90-110 minutes



Making Tool Durability Simulator

I don't know how many pieces to dig, but I also need to know how to stop. Understand the pre- and post-checks, add frequency guardrails to the search, replay the results with a fixed random seed, and complete a durable simulator with a stop reason.

Before you start: It is recommended to complete Lessons 1-9 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=10&lang=en>

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

In this lesson

01. The diamond did not appear, the pickaxe sneezed first
02. while: Pass the door first, then take action
03. Let the state move to the stop condition
04. If the target cannot be found, there must be a safe exit
05. do-while: do it once, then check
06. Check first or act first?
07. Random: Make changes replayable
08. Unstoppable circulation emergency room
09. Predict changes and also predict stop reasons
10. Project Blueprint: Tool Durability Simulator
11. Simulator code: initialization and consumption
12. Simulator ending: explain the reasons for shutdown
13. Acceptance: Prove that both exits can be used
14. Install reliable guardrails on unknown end points

01 / 14

The diamond did not appear, the pickaxe sneezed first



Steve announces that he will mine 7 more blocks. When the seventh reveals only cobblestone, he changes the target to 70. His pickaxe's durability bar nearly faints. We do not know where the target is, but we still have to account for how long the tool can last.

Say the task as "As long as the target has not been found and the upper limit has not been exceeded, continue trying." while will check this condition repeatedly. The end point is unknown, which does not mean that there is no stopping rule.

- Why is it possible that while is not executed in one round?
- Which state in the loop must change to get closer to the exit?
- How to repeatedly test random consumption and how to shut down safely in advance?

This lesson uses the console to simulate search and tool durability. Diamond positions, durability values and random consumption are all practice models and will not read archives or modify the actual game.

02 / 14

while: Pass the door first, then take action

Before each round starts, while first checks the boolean condition. true before entering the loop body, and then return to the condition after execution. It was already false before the first round, so the loop body is executed 0 times.

Snippet: place inside main and run separately

```
int durability = 3;
while (durability > 0) {
    System.out.println("Dig it, and wait for a while before action=" + durability);
    durability--;
}
System.out.println("The pickaxe is broken");
```

Output

```
Dig it, and wait for a while before action=3
Dig it, and wait for a while before action=2
Dig it, and wait for a while before action=1
The pickaxe is broken
```

Check first, act and update before going back

Step	Action
1 · Once	durability = 3
2 · Check	durability > 0 ?
3 · If true	Simulate digging a piece
4 · Update; return to step 2	durability--

Exit the loop when the condition is false.

After printing 1, it is reduced to 0, and the next check is left. The short fragments of this lesson are put into main to run. while can also be used for tasks with clear times; the choice between for and while mainly depends on which form is clearer.

03 / 14

Let the state move to the stop condition

Snippet: place inside main and run separately

```
int durability = 0;
while (durability > 0) {
    System.out.println("This line will not appear");
}
System.out.println("Prerequisite tools");
```

Output

```
Prerequisite tools
```

Each round does not necessarily only consume 1 point. When 3 points are consumed, Durability 8 will pass through 5, 2, and then cross 0. `Math.max(0, new value)` clamps the displayed value to 0 to prevent negative numbers from appearing on the durability bar.

Snippet: place inside main and run separately

```
int durability = 8;
while (durability > 0) {
    durability = Math.max(0, durability - 3);
    System.out.println("Remaining durability=" + durability);
}
```

Output

```
Remaining durability=5
Remaining durability=2
Remaining durability=0
```

Tracking table: write down the actions before and after

Rounds	Before action	Directly subtract 3	New Durability Displayed
1	8	5	5
2	5	2	2
3	2	-1	0
Next time	0	The condition is false	No more digging

Delete the update of durability and the condition will not change in the stopping direction. The loop does not know when to get off work, it only repeatedly checks the status you wrote down.

04 / 14

If the target cannot be found, there must be a safe exit

First use the determined position to study the search to avoid random results interfering with understanding. Assume that there is a diamond in the 7th block: every time it is dug, the number of blocks is increased by 1, and then it is judged whether it is the target. The maximum is 20 yuan, and both continuation conditions must be met at the same time.

Snippet: place inside main and run separately

```
int blocks = 0, diamondAt = 7;
boolean found = false;
while (!found && blocks < 20) {
    blocks++;
    found = blocks == diamondAt;
}
System.out.println("Dig" + blocks + "block, found=" + found);
```

Output

```
Dig7block, found=true
```

Two guardrails: stop if any one fails

Element	Explanation
!found	If you haven't found it yet, you need to continue searching.
blocks < 20	Only if it is still within the upper limit of the number of times, it is allowed to continue.
Connect with &&	If it succeeds or reaches the upper limit, it will leave the loop.

Only change diamondAt at a time

Target position	Actual number of blocks	Stop results
1	1	found=true
7	7	found=true
20	20	found=true
30	20	Found=false, to the upper limit

If it is written as `||`, it is still possible to continue after finding it; the target will also cross the guardrail when it is outside the upper limit. The upper limit guarantees shutdown, but does not guarantee that the target will be found.

05 / 14

do-while: do it once, then check

Put the condition at the end, and the loop body will be executed at least once. Good for a menu that must be shown first or a try that happens first. The final while (condition); has a required semicolon.

Snippet: place inside main and run separately

```
int choice = 3;
do {
    System.out.println("1 Mining  2 Repair  3 Exit");
} while (choice != 3);
```

Output

```
1 Mining  2 Repair  3 Exit
```

At least one round: execute first, then check

Step	Explanation
1. Execute the body	Run immediately on the first pass
2. Then check the condition	true → repeat the body; false → exit

Snippet: place inside main and run separately

```
int durability = 0;
do {
    durability += 3;
    System.out.println("After repair=" + durability);
} while (durability < 9);
```

Output

```
After repair=3
After repair=6
After repair=9
```

The first example only demonstrates a display without reading selection. To make the menu repeat, the choice needs to be updated in the loop; the real interactive menu will be completed in the after-class challenge.

06 / 14

Check first or act first?

The difference lies in the timing of the first inspection

Element	Explanation
while	Check first and then do it, 0 rounds are allowed. You should not continue digging when the tool is damaged.
do-while	Do it first and then check, at least 1 round. Exiting the menu must also be visible to the player first.

Snippet: place inside main and run separately

```
int torches = 0;
while (torches > 0) {
    System.out.println("Set a torch");
    torches--;
}
System.out.println("Over.");
```

Output

```
Over.
```

Snippet: place inside main and run separately

```
int items = 0;
do {
    System.out.println("Check box");
} while (items > 0);
```

Output

```
Check box
```

It is not which one is more advanced, but whether the first round of actions should occur first. Don't give the damaged tool to do-while to force dig in order to ensure "at least once".

The semicolon at the end of `do { ... } while (condition);` belongs to the syntax; `while (condition);` is written after the code block, but the semicolon will create an empty loop. The functions of these two positions are completely different.

07 / 14

Random: Make changes replayable

Pseudo-random can simulate different consumption. A fixed seed of 10 is used during testing. The same seed and the same calling sequence will result in the same sequence. In this way, if a bug is encountered, the same set of consumption can be replayed.

Snippet: place inside main and run separately

```
java.util.Random random = new java.util.Random(10);
int cost = 1 + random.nextInt(4);
System.out.println(cost);
```

Output

```
3
```

Snippet: place inside main and run separately

```
java.util.Random random = new java.util.Random(10);
int count = 0;
while (count < 5) {
    int cost = 1 + random.nextInt(4);
    System.out.println(cost);
    count++;
}
```

Output

```
3
2
2
2
1
```

Random range: upper bound does not include

Expression	Possible results
<code>random.nextInt(4)</code>	0, 1, 2, 3
<code>1 + random.nextInt(4)</code>	1, 2, 3, 4
<code>2 + random.nextInt(4)</code>	2, 3, 4, 5

The same `Random` is created once outside the loop. New `Random(10)` is used every round, and the first number of the sequence will be obtained repeatedly. If you want the game results to change, you can change it to new `Random()` and keep the fixed seed during testing.

08 / 14

Unstoppable circulation emergency room



Read the code to check the cause, and do not run out-of-control counterexamples

- The condition variable is not updated: durability remains 5, and `durability > 0` is always true.

- There is an extra semicolon after while: the loop body becomes an empty statement, and subsequent updates cannot enter at all.
- && is written as ||: As long as one item is true, continue. Both success and upper limit lose the independent shutdown function.
- The safe upper limit count is not updated: it seems that the upper limit is written, but in fact it can never be reached.

Snippet: place inside main and run separately

```
int blocks = 0;
while (blocks < 3) {
    blocks++;
    System.out.println("Digged" + blocks + "Block");
}
```

Output

```
Digged1Block
Digged2Block
Digged3Block
```

If you find that the console screen is abnormally refreshed, first use the stop button or Ctrl+C to end the program. Then check the conditions, updates, directions, semicolons and upper limits, and only fix one point at a time. The counter must also be within a reasonable range, and do not use overflow as an outlet.

09 / 14

Predict changes and also predict stop reasons

Write the exit of the loop into the answer

Step	Explanation
1. Projections	List the starting and ending status of each round.
2. Run	Check the actual sequence, not just the last line.
3. Explanation	Point out which continuation condition fails first.

Snippet: place inside main and run separately

```
int durability = 7;
while (durability > 0) {
    durability = Math.max(0, durability - 2);
    System.out.println(durability);
}
```

Snippet: place inside main and run separately

```
int blocks = 0;
boolean found = false;
while (!found && blocks < 3) {
    blocks++;
}
System.out.println(blocks + ":" + found);
```

Snippet: place inside main and run separately

```
int energy = 0;
do {
    System.out.println("Check");
} while (energy > 0);
```

Check: the first question outputs 5, 3, 1, 0 line by line, and returns to zero permanently; the second question outputs 3: false, and the upper limit of times is reached first; the third question outputs "check" once, and then the condition is false.

10 / 14

Project Blueprint: Tool Durability Simulator

Simulate mining with 24 points of durability, randomly consuming 1 to 4 points in each round, up to 50 blocks. Print the rounds, consumption, remaining durability, and durability bars, and clearly state why you stopped.

After each random consumption, both statuses are updated

Step	Action
1 · Once	Durability 24, number of blocks 0
2 · Check	Durability>0 and not reaching the upper limit
3 · If true	Extraction consumption
4 · Update; return to step 2	Update status and print

Exit the loop when the condition is false.

Write the contract first, then write the code

Item	Rules for this lesson
Allow settings	Durability and upper limit are both integers from 0 to 100
Random	new Random(10) outside the loop, extracted once in each round
Display	Math.max ensures durability is not less than 0
Durable strip	Display a # every 2 points, only a simple proportion prompt
Over.	If the durability is 0, damage is reported, otherwise the safety upper limit is reported.

This is a tool consumption item and does not simulate random diamond drops. The previous diamond search has demonstrated successful export; if you expand this project, also add !found, and write down the prompt priority when "find, damage, upper limit" occurs at the same time.

11 / 14

Simulator code: initialization and consumption

Create new DurabilitySimulator.java. The complete program is divided into two pages, connected in the same file. Only one consumption is extracted in each round, making it easy to compare item by item with the fixed sequence.

DurabilitySimulator.java · First half, continued on next page

```
import java.util.Random;

public class DurabilitySimulator {
    public static void main(String[] args) {
        final int START_DURABILITY = 24;
        final int SAFETY_LIMIT = 50;
        int durability = START_DURABILITY, blocksMined = 0;
        Random random = new Random(10);
        if (START_DURABILITY < 0 || START_DURABILITY > 100
            || SAFETY_LIMIT < 0 || SAFETY_LIMIT > 100) {
            System.out.println("Durability and upper limit must be within 0 to 100 between");
        } else {
            System.out.println("=== Tool Durability Simulation ===");
            while (durability > 0 && blocksMined < SAFETY_LIMIT) {
                int cost = 1 + random.nextInt(4);
                blocksMined++;
                durability = Math.max(0, durability - cost);
                String bar = "#".repeat(durability / 2);
                System.out.printf("No.%02d Block Consumption%d Remaining%02d %s%n",
                                blocksMined, cost, durability, bar);
            }
        }
    }
}
```



String.repeat is a tool seen in Lesson 5. Durability / 2 uses integer division, so there is no # when the durability is 1, which does not mean it has been damaged; still use durability == 0 to judge damage.

12 / 14

Simulator ending: explain the reasons for shutdown

DurabilitySimulator.java · [Next to the previous page](#)

```
        if (durability == 0) {
            System.out.println("Tool damaged, stop digging");
        } else {
            System.out.println("Trigger the safety upper limit and go home with the tools");
        }
        System.out.println("Digging together: " + blocksMined + "Block");
    }
}
```

Both conditions may cause the loop to end. If the durability just returns to zero in the end and reaches the upper limit at the same time, this project will give priority to reporting tool damage. By writing down the prompt priorities clearly first, you won't leave the outcome to luck.

The standard version has 24 points of durability, and the first five rounds can be calculated by hand.

Number of blocks	This consumption	Remaining durability	Durable strip length
1	3	21	10
2	2	19	9
3	2	17	8
4	2	15	7
5	1	14	7

The 11th block of the standard procedure of fixed seed 10 is reset to zero, and finally reports "Tool damaged, stop digging" and "Total excavation: 11 blocks". Running it twice should give you the same complete output.

13 / 14

Acceptance: Prove that both exits can be used

Only change one constant and observe different exits

Test settings	Number of blocks	Remaining durability	Stop reason
Durability 0, upper limit 50	0	0	Tool damaged
Durability 1, upper limit 50	1	0	Tool damaged
Durability 24, upper limit 50	11	0	Tool damaged
Durability 24, upper limit 3	3	17	Safety upper limit
Durability 24, upper limit 0	0	24	Safety upper limit
Durability 1, upper limit 1	1	0	Prioritize reporting of damage
Durability -1 or upper limit 101	No digging	No statistics	Report range error

What is verified is the rules, not "it doesn't look broken"

- The consumption of each round is between 1 and 4, and the remaining durability is never negative.
- The total number of blocks is equal to the number of mining records actually printed.
- When the upper limit is 3, exactly 3 pieces will be printed, and the fourth block will not be mined more.
- The standard version is run repeatedly, and all rows are consistent; the range and upper limit are also maintained after changing the seeds.

Don't run a runaway loop by following the "delete counter and try again" approach. Static inspection can show that the upper limit is invalid; when experiments are needed, use another independent and clear small upper limit to constrain the number of times.

14 / 14

Install reliable guardrails on unknown end points



Wooden pickaxe task: counting down from ten to zero

Just use while to output 10 to 0, and then print "emit". There are 11 numbers in total; also measure the starting points 0 and 1, and explain why ≥ 0 can include the last 0.

Iron Pickaxe Task: Hunger Value Decline Simulation

Initial 20, randomly dropped from 2 to 5 in each round, fixed seed 10, display value clamped to 0. Print the round, consumption and remaining value, and set a safety upper limit; when changing to upper limit 2, it must be stopped in advance.

Diamond Pickaxe Quest: Repeatable Menu

Scanner cooperates with do-while: 1 mining, 2 repair, 3 exit. Check durability before mining, and repair up to 20; unknown input prompts to try again. Using string selection avoids numeric parsing errors and the menu must be shown at least once.

Acceptance menu: 3 is only displayed once; 1, 3 dig first and then exit; 2, 3 repair first and then exit; 0, abc can prompt and continue; the tool cannot dig when the durability is 0. The next lesson is about learning how to organize these actions into independent skills.

Technical check: Oracle Java Language Specification Chapter 14; java.util.Random API.

<https://docs.oracle.com/javase/specs/jls/se21/html/jls-14.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Random.html>