

JAVA / LESSON 27 OF 40

Random Loot Tables

Use Set to remove duplicates and use interval to explain probability

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

Make a loot simulator LootSimulator

Why don't the diamonds in the treasure chest appear on time every twenty times? Divide the probability into one hundred grids, fix the random seed and replay the experiment, the prior boundary, and then look at the number and proportion.

Prerequisites: if branch, loop, ArrayList and HashMap. Today, the treasure chest will no longer open seven sticks in a row.

Before you start: It is recommended to complete Lessons 25-26 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=27&lang=en>

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

The same achievement is obtained twice and still only registered once.

Element	Explanation
Set	Save unique member
First add	Change the collection and return true
Repeat add	No change, return false

In this lesson

01. Before starting: predict first, then let the program answer
02. Set only retains non-repeating elements
03. Add query and delete
04. Set operation is copied first
05. The upper bound of Random does not include
06. The probability interval must be continuous
07. Fixed seed recurrence random test
08. Simulate 1000 observation frequencies
09. Use Set to ensure that rewards are not repeated
10. Challenge Blueprint Loot Simulator
11. loot simulation program
12. Expansion tasks and clearance review
13. Own acceptance record

01 / 13

Before starting: predict first, then let the program answer

This chapter uses JDK 21 and does not require the preview function. The story is borrowed from the block world, and the example runs on the ordinary Java console; if the values and rules are course models, they will be clearly stated.

Each common example is a complete independent file, saved according to the file name marked on the box. Place each example in a different folder to avoid duplicate names of auxiliary classes. When the complete project is divided into sections, they are connected sequentially in the same file.

Terminal command: replace the Chinese placeholder with the actual file name and class name

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

- Prediction: Write the expected output first, or draw out the changes in references and data.
- Verification: Check line by line after running, and then only change one condition.
- Explanation: Explain whether the results come from Java rules, course contracts, or input data.

"Intentional error" is used to recognize bugs and cannot be mixed with ordinary examples. Use a special temporary directory for file experiments; do not use real archives to test coverage and deletion.

02 / 13

Set only retains non-repeating elements

Set is like an achievement wall. It cares whether there is a certain record, does not care about the number, and does not allow the same content to occupy the position repeatedly.

The same achievement is obtained twice and still only registered once.

Element	Explanation
Set	Save unique member
First add	Change the collection and return true
Repeat add	No change, return false

The first time add changes the collection, so it returns true; the second time there is no change, so it returns false.

L27Example01.java · Complete independent example

```
import java.util.*;

public class L27Example01 {
    public static void main(String[] args) {
        Set<String> achievements = new HashSet<>();
        System.out.println(achievements.add("First mining"));
        System.out.println(achievements.add("First mining"));
        System.out.println(achievements.size());
    }
}
```

Output

```
true
false
1
```

03 / 13

Add query and delete

Common operations on HashSet revolve around membership. It doesn't have get(0) because collections don't commit to positions.

contains Answer whether it has remove Cancel the record

Element	Explanation
contains	Check membership
remove	Remove specified member
No index	No get(0)

Use List when order and duplicate items are required; use Set when you only care about unique members and quick judgment.

L27Example02.java · Complete independent example

```
import java.util.*;
public class L27Example02 {
    public static void main(String[] args) {
        Set<String> loot = new HashSet<>(Set.of("diamond", "Bread."));
        System.out.println(loot.contains("diamond"));
        System.out.println(loot.remove("diamond"));
        System.out.println(loot.contains("diamond"));
    }
}
```

Output

```
true
true
false
```

Fault experiment: Set does not have a get method by index. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadSetIndex.java · Intentional error example

```
import java.util.*;
public class BadSetIndex {
    public static void main(String[] args) {
        Set<String> loot = new HashSet<>();
        System.out.println(loot.get(0));
    }
}
```

04 / 13

Set operation is copied first

Set can compare two teams of players or two drop tables. Note that these methods directly modify the collection on which they are called.

Union, intersection and difference will modify the receiver

Element	Explanation
Intersection retainAll	Only keep those on both sides
Union addAll	Including any side
Difference set removeAll	Leave only those in the original set

If the original set is needed later, first new HashSet<>(source). The copy is like a save point. If the experiment fails, the original world will not be destroyed.

L27Example03.java · Complete independent example

```
import java.util.*;
public class L27Example03 {
    public static void main(String[] args) {
        Set<String> a = new HashSet<>(Set.of("diamond", "Bread."));
        Set<String> b = Set.of("Bread.", "Iron Ingot");
        Set<String> both = new HashSet<>(a); both.retainAll(b);
        Set<String> either = new HashSet<>(a); either.addAll(b);
        Set<String> onlyA = new HashSet<>(a); onlyA.removeAll(b);
        System.out.println(both.equals(Set.of("Bread.")));
        System.out.println(either.size());
        System.out.println(onlyA.equals(Set.of("diamond")));
        System.out.println(a.size());
    }
}
```

Output

```
true
3
true
2
```

05 / 13

The upper bound of Random does not include

random.nextInt(100) may get 0 to 99, a total of 100 integers, and will never get 100.

nextInt bound generates 0 to bound minus 1

Element	Explanation
nextInt(100)	Get 0 to 99
nextInt(6)+1	Get 1 to 6
Upper bound	Does not contain and must be a positive number

The range of nextInt(n) is $0 \leq \text{value} < n$. Write this formula first when you see probability and random index.

L27Example04.java · Complete independent example

```
import java.util.*;
public class L27Example04 {
    public static void main(String[] args) {
        Random random = new Random(2026);
        boolean inRange = true;
        for (int i = 0; i < 100; i++) {
            int dice = random.nextInt(6) + 1;
            inRange &= dice >= 1 && dice <= 6;
        }
        System.out.println(inRange);
    }
}
```

Output

```
true
```

nextInt requires the upper bound to be a positive number. Dice is `nextInt(6) + 1`; `nextInt(5) + 1` is never 6. Random is for practice simulation, not a password or lottery anti-cheat tool.

06 / 13

The probability interval must be continuous

Arrange 0 to 99 into 100 grids. Each grid can only belong to one reward, and all grids must be assigned.

100 grids are divided into 5, 20, and 75 grids

Item	Roll range	Chance
Diamond	0-4	5%
Iron	5-24	20%
Bread	25-99	75%

100 possible values: $0 \leq \text{roll} < 100$. The value 100 is excluded.

The second condition is written `roll < 25`, which means that the total has reached 25 squares, not 25 more squares. Boundary points 4, 5, 24, and 25 must be tested one by one.

L27Example05.java · Complete independent example

```
public class L27Example05 {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll should be 0 to 99");
        if (roll < 5) return "diamond";
        if (roll < 25) return "Iron Ingot";
        return "Bread.";
    }
    public static void main(String[] args) {
        for (int roll : new int[]{0, 4, 5, 24, 25, 99})
            System.out.println(roll + ": " + pickRoll(roll));
    }
}
```

Output

```
0: diamond
4: diamond
5: Iron Ingot
24: Iron Ingot
25: Bread.
99: Bread.
```

07 / 13

Fixed seed recurrence random test

If the random program is different every time, the bug may teleport like the Enderman. By passing in a fixed seed during testing, the failed process can be replayed.

The same seed generates the same string of pseudo-random numbers

Element	Explanation
Same seed	Provide the same starting point
Same calling sequence	Ensure that the random sequence can be replayed
Change call	Subsequent sequences will be misaligned

Fixed seeds are only for recurrence testing. When actually opening the treasure chest, you can use the default constructor or pass Random in from the outside.

L27Example06.java · Complete independent example

```
import java.util.*;
public class L27Example06 {
    public static void main(String[] args) {
        Random a = new Random(2026), b = new Random(2026);
        boolean same = true;
        for (int i = 0; i < 20; i++) same &= a.nextInt(100) == b.nextInt(100);
        System.out.println(same);
    }
}
```

Output

```
true
```

The same Random, the same seed, and the same method calling sequence will reappear. If an extra draw occurs somewhere, the subsequent results will be misaligned; fixing the seed does not mean "making the probability a fixed number."

08 / 13

Simulate 1000 observation frequencies

5% does not mean that a diamond will come out every 20 times. It means that among a large number of independent attempts, the long-term proportion is usually close to 5%.

The more times the ratio is usually closer to the design value

Element	Explanation
Sum of times	Must be equal to the number of simulations
Long-term ratio	Usually close to the design probability
An experiment	The quantity does not have to be exactly 50

The total number of tests must equal 1000; each quantity must not be negative. Don't assert that there must be exactly 50 diamonds.

L27Example07.java · Complete independent example

```
import java.util.*;
public class L27Example07 {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll should be 0 to 99");
        if (roll < 5) return "diamond";
        if (roll < 25) return "Iron Ingot";
        return "Bread.";
    }
    static String pickLoot(Random random) {
        return pickRoll(java.util.Objects.requireNonNull(random).nextInt(100));
    }
    static Map<String, Integer> simulate(int trials, long seed) {
        if (trials < 0 || trials > 1_000_000)
            throw new IllegalArgumentException("The number of simulations should be 0 to 100 Ten
> thousand");
        Random random = new Random(seed);
        Map<String, Integer> counts = new HashMap<>();
        for (int i = 0; i < trials; i++)
            counts.merge(pickLoot(random), 1, Integer::sum);
        return counts;
    }
    public static void main(String[] args) {
        Map<String, Integer> counts = simulate(1000, 2026);
        int total = counts.values().stream().mapToInt(Integer::intValue).sum();
        System.out.println(total);
        System.out.println(counts.values().stream().allMatch(n -> n >= 0));
    }
}
```

Output

```
1000
true
```

09 / 13

Use Set to ensure that rewards are not repeated

When three different rewards are drawn continuously, Set can automatically reject repetitions. But if there are only two candidates, requiring three will lead to an infinite loop.

New rewards will be drawn before being put into the final list

Element	Explanation
Target quantity	Cannot exceed candidate class
Try upper limit	Avoid long cycles
Number of results	The upper limit may not be collected yet

First confirm that the number of targets does not exceed the number of different candidates, and then add the maximum number of attempts as a second insurance.

When the upper limit of attempts is reached, only partial rewards may be obtained. The caller must check whether `result.size()` reaches wanted; setting an upper limit to prevent getting stuck does not guarantee that it will be collected. This lesson rejects targets with more than three candidates.

L27Example08.java · Complete independent example

```
import java.util.*;
public class L27Example08 {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll should be 0 to 99");
        if (roll < 5) return "diamond";
        if (roll < 25) return "Iron Ingot";
        return "Bread.";
    }
    static String pickLoot(Random random) {
        return pickRoll(java.util.Objects.requireNonNull(random).nextInt(100));
    }
    static Set<String> collectUnique(Random random, int wanted, int limit) {
        if (random == null || wanted < 0 || wanted > 3 || limit < 0)
            throw new IllegalArgumentException("Check target number and attempt upper limit");
        Set<String> result = new HashSet<>();
        for (int attempt = 0; result.size() < wanted && attempt < limit; attempt++)
            result.add(pickLoot(random));
        return result;
    }
    public static void main(String[] args) {
        Set<String> loot = collectUnique(new Random(2026), 3, 100);
        System.out.println(loot.size());
        System.out.println(collectUnique(new Random(2026), 3, 0).size());
    }
}
```

Output

```
3
0
```

10 / 13

Challenge Blueprint Loot Simulator

The challenge program is called LootSimulator. It repeats the simulation with seed 2026 and outputs the three reward quantities, the total number and the number of discovered species.

Fixed seed simulation 1000 times and statistics

Element	Explanation
Priori boundary	0, 4, 5, 24, 25, 99
Reseed	Replayable in 2026
Post-check total number	Does not depend on Set order

First verify the probability boundary individually and then run it 1000 times. If the selector is wrong, the more you simulate it, the more confident it will become.

Acceptance conditions

- roll can only be 0 to 99.
- There are 5 spaces for diamonds, 20 spaces for iron ingots, and 75 spaces for bread.
- The sum of the three quantities must equal 1000.
- The output of repeated runs with the same seed is consistent.
- There are at most three rewards in a Set.

Acceptance comparison table

Boundary point	Reward
0	diamond
4	diamond
5	Iron Ingot
24	Iron Ingot
25	Bread.
99	Bread.

11 / 13

loot simulation program

Create a new LootSimulator.java, compile and run using Java 21.

Random Set and Map collaboration

Element	Explanation
simulate	Repeat the test by seed
counts	Count by reward name
Stable printing	Clear diamonds, iron ingots, bread

Change the cycle to 0 times, and all three counts should be 0; change the seed to 2027, the specific number is variable but the total number is still 1,000.

LootSimulator.java · Section 1/2, sequentially connected to the same file

```
import java.util.*;
public class LootSimulator {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll should be 0 to 99");
        if (roll < 5) return "diamond";
        if (roll < 25) return "Iron Ingot";
        return "Bread.";
    }
    static String pickLoot(Random random) {
        return pickRoll(java.util.Objects.requireNonNull(random).nextInt(100));
    }
    static Map<String, Integer> simulate(int trials, long seed) {
        if (trials < 0 || trials > 1_000_000)
            throw new IllegalArgumentException("The number of simulations should be 0 to 100 Ten
> thousand");
        Random random = new Random(seed);
        Map<String, Integer> counts = new HashMap<>();
        for (int i = 0; i < trials; i++)
            counts.merge(pickLoot(random), 1, Integer::sum);
        return counts;
    }
    static Set<String> collectUnique(Random random, int wanted, int limit) {
        if (random == null || wanted < 0 || wanted > 3 || limit < 0)
            throw new IllegalArgumentException("Check target number and attempt upper limit");
        Set<String> result = new HashSet<>();
        for (int attempt = 0; result.size() < wanted && attempt < limit; attempt++)
            result.add(pickLoot(random));
        return result;
    }
}
```

LootSimulator.java · Section 2/2, sequentially connected to the same file

```
    }
    public static void main(String[] args) {
        Map<String, Integer> counts = simulate(1000, 2026);
        for (String item : List.of("diamond", "Iron Ingot", "Bread."))
            System.out.println(item + "=" + counts.getOrDefault(item, 0));
        System.out.println("Total number=" + counts.values().stream()
            .mapToInt(Integer::intValue).sum());
        System.out.println("Discovery type=" + counts.size());
    }
}
```

Output

```
diamond=43
Iron Ingot=166
Bread.=791
Total number=1000
Discovery type=3
```

12 / 13

Expansion tasks and clearance review

The assignment must be accompanied by a probability interval table and boundary test. It is not acceptable to just look random to the naked eye.

Three-level task expansion random system

Element	Explanation
Probability table	Each grid belongs to exactly one kind of reward
Boundary table	Verify adjacent dividing points
Analog table	Recording times and frequency

Lesson 28: Wear explosion-proof armor for the program. Bad input will still come, but exception handling allows the backpack administrator to explain the problem and continue working.

Third gear operation

- Wooden Pickaxe: Make dice from 1 to 6, run 60 times and count each point.
- Iron Pickaxe: Draw three non-duplicate items from the reward pool, and give a prompt when there are insufficient candidates.
- Diamond Pickaxe: Add a guarantee mechanism. If a diamond is not produced for 20 consecutive times, it will be produced next time.

Acceptance comparison table

Self-evaluation	Can	Still need to practice
I will use Set to remove duplicates	☐	☐
I can write down the probability interval	☐	☐
I will fix the seed test	☐	☐

13 / 13

Own acceptance record

Beautiful output is the starting point, and being able to explain boundaries is considered mastery. Select one case each from normal, borderline and error, write down your expectations before running, and leave evidence after running.

Leave a basis for judgment

Category	The operation you selected	Check key points
Normal	A common input or a successful action	Is the result consistent line by line?
Border	Range endpoints and adjacent values	Is there one more step or one less thing?
Error	Illegal quantity, missing object or bad data	Whether it is explicitly rejected and whether the status is maintained

- Bug records: triggering conditions, actual phenomena, causes, and minimum repairs.
- Illustrated record: describe the starting point, end point and meaning of each arrow.
- Adaptation record: only change one rule, and modify expectations and tests at the same time.

Technical check: Java SE 21 language specification and standard library. See the language specification for class, interface, type and expression rules; see the corresponding API for the behavior of collections, Random, Files and Stream.

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

<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>