

第27课

随机战利品表

用 Set 去重，用区间解释概率

同一成就获得两次仍只登记一次

Set

保存唯一成员

第一次 add

改变集合，返回 true

重复 add

不改变，返回 false

宝箱里的钻石为什么没有每二十次准时出现？把概率拆成一百格，固定随机种子重播实验，先验边界，再看次数和比例。

前置背包：if 分支、循环、ArrayList 和 HashMap。今天让宝箱不再连续开出七根木棍。

阅读方式：先猜结果，再运行核对；每次只改一个条件。

本章使用 JDK 21，无需预览功能。故事与插画借用方块世界，练习运行在普通 Java 控制台。

开始前：先预测，再让程序回答

本章使用 JDK 21，无需预览功能。故事借用方块世界，示例运行在普通 Java 控制台；数值和规则若是课程模型，都会明确说明。

每个普通示例都是完整独立文件，按框上标注的文件名保存。各示例放在不同文件夹，避免辅助类重名。完整项目分段时，按顺序接在同一个文件中。

终端命令：中文占位符换成实际文件名与类名

```
javac -encoding UTF-8 示例文件名.java  
java 示例类名
```

- 预测：先写预期输出，或画清引用和数据的变化。
- 验证：运行后逐行核对，再只改一个条件。
- 解释：说清结果来自 Java 规则、课程合同还是输入数据。

“故意错误”用于认识 Bug，不能与普通示例混用。文件实验使用专门临时目录；不要拿真实存档测试覆盖和删除。

01 Set 只保留不重复元素

Set 像成就墙。它关心有没有某项记录，不关心第几个，也不允许同一内容重复占位置。

同一成就获得两次仍只登记一次

Set

保存唯一成员

第一次 add

改变集合，返回 true

重复 add

不改变，返回 false

第一次 add 改变了集合，所以返回 true；第二次没有变化，所以返回 false。

L27Example01.java • 完整独立示例

```
import java.util.*;
public class L27Example01 {
    public static void main(String[] args) {
        Set<String> achievements = new HashSet<>();
        System.out.println(achievements.add("首次挖矿"));
        System.out.println(achievements.add("首次挖矿"));
        System.out.println(achievements.size());
    }
}
```

运行核对

```
true
false
1
```

02 添加 查询与删除

HashSet 的常用操作都围绕成员资格。它没有 `get(0)`，因为集合不承诺位置。

contains 回答是否拥有 remove 取消记录

contains

检查成员资格

remove

移除指定成员

无索引

没有 `get(0)`

需要顺序和重复项用 List；只关心唯一成员和快速判断时用 Set。

L27Example02.java • 完整独立示例

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

运行核对

```
true
true
false
```

故障实验：Set 没有按索引 `get` 的方法。下面故意保留错误，请先预测，再尝试修复。

BadSetIndex.java • 故意错误示例

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

03 集合运算先复制

Set 可以比较两队玩家或两张掉落表。注意这些方法会直接修改调用它们的集合。

并集 交集 差集都会修改接收者

交集 retainAll

只保留两边都有的

并集 addAll

包含任意一边的

差集 removeAll

留下仅在原集合的

若后面还需要原集合，先 `new HashSet<>(source)`。副本像存档点，实验失败也不会毁掉原世界。

L27Example03.java • 完整独立示例

```
import java.util.*;
public class L27Example03 {
    public static void main(String[] args) {
        Set<String> a = new HashSet<>(Set.of("钻石", "面包"));
        Set<String> b = Set.of("面包", "铁锭");
        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("面包")));
        System.out.println(either.size());
        System.out.println(onlyA.equals(Set.of("钻石")));
        System.out.println(a.size());
    }
}
```

运行核对

```
true
3
true
2
```

04 Random 的上界不包含

`random.nextInt(100)` 可能得到0到99，共100种整数，永远不会得到100。

`nextInt bound` 生成0到bound减1

`nextInt(100)`

得到0到99

`nextInt(6)+1`

得到1到6

上界

不包含且必须为正数

`nextInt(n)` 的范围是 $0 \leq \text{value} < n$ 。看到概率和随机索引时先写这条公式。

L27Example04.java • 完整独立示例

```
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);
    }
}
```

运行核对

true

`nextInt` 要求上界为正数。骰子是 `nextInt(6) + 1`；`nextInt(5) + 1` 永远没有6。`Random` 用于练习模拟，不是密码或抽奖防作弊工具。

05 概率区间必须连续

把0到99排成100格。每格只能属于一种奖励，所有格子都必须有归属。

100格分成5、20、75格



钻石: 0-4

5格 = 5%

铁锭: 5-24

20格 = 20%

面包: 25-99

75格 = 75%

第二个条件写 `roll < 25` 表示累计到25格，不是再加25格。边界点4、5、24、25必须逐个测试。

L27Example05.java • 完整独立示例

```
public class L27Example05 {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll应为0到99");
        if (roll < 5) return "钻石";
        if (roll < 25) return "铁锭";
        return "面包";
    }
    public static void main(String[] args) {
        for (int roll : new int[]{0, 4, 5, 24, 25, 99})
            System.out.println(roll + ": " + pickRoll(roll));
    }
}
```

运行核对

0: 钻石
4: 钻石
5: 铁锭
24: 铁锭
25: 面包
99: 面包

06 固定种子复现随机测试

随机程序若每次都不同，Bug 可能像末影人一样瞬移。测试时传入固定种子，失败过程就能重播。

同一种子产生同一串伪随机数

相同种子

提供相同起点

相同调用顺序

保证随机序列可重播

改变调用

后续序列会错位

固定种子只为复现测试。真正开宝箱时可使用默认构造器或从外部传入 Random。

L27Example06.java • 完整独立示例

```
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);
    }
}
```

运行核对

true

同一种 Random、相同种子、相同方法调用顺序才会复现。若某处额外抽了一次，后续结果会错位；固定种子不是“让概率变成固定数量”。

07 模拟1000次观察频率

5%不表示每20次必出一次钻石。它表示大量独立尝试中，长期比例通常接近5%。

次数越多比例通常越接近设计值

次数总和

必须等于模拟次数

长期比例

通常接近设计概率

一次实验

数量不必恰好50

测试总次数必须等于1000；每种数量不得为负。不要断言钻石一定恰好50个。

L27Example07.java • 完整独立示例

```
import java.util.*;

public class L27Example07 {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll应为0到99");
        if (roll < 5) return "钻石";
        if (roll < 25) return "铁锭";
        return "面包";
    }

    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("模拟次数应为0到100万");
        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));
    }
}
```

运行核对

1000
true

08 用 Set 保证奖励不重复

连续抽取三种不同奖励时，Set 能自动拒绝重复。但若候选只有两种，要求三种就会无限循环。

抽到新奖励才放进最终列表

目标数量

不能超过候选种类

尝试上限

避免长时间循环

结果数量

达到上限可能未集齐

先确认目标数量不超过不同候选数量，再加最大尝试次数作为第二道保险。

达到尝试上限时可能只拿到部分奖励。调用者必须检查 `result.size()` 是否达到 `wanted`；设置上限防止卡住，不保证集齐。本课拒绝目标数超过三种候选。

08 用 Set 保证奖励不重复（续）

L27Example08.java • 完整独立示例

```
import java.util.*;
public class L27Example08 {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll应为0到99");
        if (roll < 5) return "钻石";
        if (roll < 25) return "铁锭";
        return "面包";
    }
    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("检查目标数量和尝试上限");
        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());
    }
}
```

运行核对

3
0

09 挑战 蓝图战利品模拟器

挑战 程序叫 LootSimulator。它用种子2026重复模拟，输出三种奖励数量、总数和发现过的种类数。

固定种子模拟1000次并统计

先验边界

0、4、5、24、25、99

再定种子

2026可重播

后查总数

不依赖 Set 顺序

先单独验证概率边界，再运行1000次。选择器错了，模拟次数再多也只会更自信地错。

验收条件

- roll只能是0到99。
- 钻石区间5格，铁锭20格，面包75格。
- 三种数量相加必须等于1000。
- 相同种子重复运行输出一致。
- Set 中最多出现三种奖励。

验收对照表

边界点	奖励
0	钻石
4	钻石
5	铁锭
24	铁锭
25	面包
99	面包

10 战利品模拟程序

新建 LootSimulator.java, 使用 Java 21 编译运行。

Random Set 和 Map 协作

simulate

按种子重复试验

counts

按奖励名计数

稳定打印

明确钻石、铁锭、面包

把循环改成0次，三种计数都应0；把种子改为2027，具体数量可变但总数仍为1000。

LootSimulator.java • 第1/2段，顺序接在同一文件

```
import java.util.*;
public class LootSimulator {
    static String pickRoll(int roll) {
        if (roll < 0 || roll >= 100)
            throw new IllegalArgumentException("roll应为0到99");
        if (roll < 5) return "钻石";
        if (roll < 25) return "铁锭";
        return "面包";
    }
    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("模拟次数应为0到100万");
        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("检查目标数量和尝试上限");
        Set<String> result = new HashSet<>();
        for (int attempt = 0; result.size() < wanted && attempt < limit; attempt++)
            result.add(pickLoot(random));
        return result;
    }
}
```

10 战利品模拟程序（续）

LootSimulator.java • 第2/2段，顺序接在同一文件

```
}  
    public static void main(String[] args) {  
        Map<String, Integer> counts = simulate(1000, 2026);  
        for (String item : List.of("钻石", "铁锭", "面包"))  
            System.out.println(item + "=" + counts.getOrDefault(item, 0));  
        System.out.println("总数=" + counts.values().stream()  
            .mapToInt(Integer::intValue).sum());  
        System.out.println("发现种类=" + counts.size());  
    }  
}
```

运行核对

钻石=43
铁锭=166
面包=791
总数=1000
发现种类=3

11 拓展任务与通关复盘

作业必须附概率区间表和边界测试，不接受只凭肉眼看起来挺随机。

三档任务扩建随机系统

概率表

每格恰属一种奖励

边界表

验证相邻分界点

模拟表

记录次数与频率

第28课给程序穿防爆甲。坏输入仍会到来，但异常处理能让背包管理员解释问题并继续工作。

三档作业

- 木镐：制作1到6的骰子，运行60次并统计每个点数。
- 铁镐：从奖励池抽取三个不重复物品，候选不足时给出提示。
- 钻石镐：加入保底机制，连续20次未出钻石时下一次必出。

验收对照表

自评	能	还要练
我会使用Set去重	☐	☐
我能写清概率区间	☐	☐
我会固定种子测试	☐	☐

自己的验收记录

漂亮的输出是起点，能解释边界才算掌握。把正常、边界与错误各选一个案例，运行前写预期，运行后留下证据。

为判断留下依据

类别	你选择的操作	核对重点
正常	一个常用输入或一次成功动作	结果是否逐行一致
边界	范围端点与相邻值	有没有多一步、少一项
错误	非法数量、缺失对象或坏数据	是否明确拒绝，状态是否保持

- Bug 记录：触发条件、实际现象、原因、最小修复。
- 图示记录：为每条箭头说明起点、终点和含义。
- 改编记录：只改一个规则，同时修改预期和测试。

技术核对：Java SE 21 语言规范与标准库。类、接口、类型与表达式规则见语言规范；集合、Random、Files 与 Stream 的行为见对应 API。
<https://docs.oracle.com/javase/specs/jls/se21/html/index.html>
<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>