

第10课

挖到钻石才停

while、do-while 与工具耐久模拟器



不知道要挖几块，也要知道怎样停。读懂前置与后置检查，给搜索加上次数护栏，用固定随机种子重放结果，完成有停止原因的耐久模拟器。

阅读方式：先猜结果，再运行核对；每次只改一个条件。

使用 Java 11 或更新版本。故事与插画借用方块世界，练习运行在普通 Java 控制台。

01 钻石没出现，镐子先打了喷嚏



史蒂夫宣布再挖7块。第7块后面只有圆石，于是他改成70块。镐子听完，耐久条先晕了过去：目标在哪里，事先不知道；可工具能坚持多久，不能不管。

把任务说成“只要还没找到目标，而且没有超过上限，就继续尝试”。while 会反复检查这句条件。终点未知，不等于没有停止规则。

- while 为什么可能一轮也不执行？
- 循环中哪个状态必须变化，才会靠近出口？
- 随机消耗怎样重复测试，怎样提前安全停机？

本课用控制台模拟搜索和工具耐久。钻石位置、耐久数值与随机消耗都是练习模型，不会读取存档或修改实际游戏。

02 while: 先过门，再动手

每轮开始前，while 先检查 boolean 条件。true 才进入循环体，执行完再回到条件。第一轮前已经为false，循环体就执行0次。

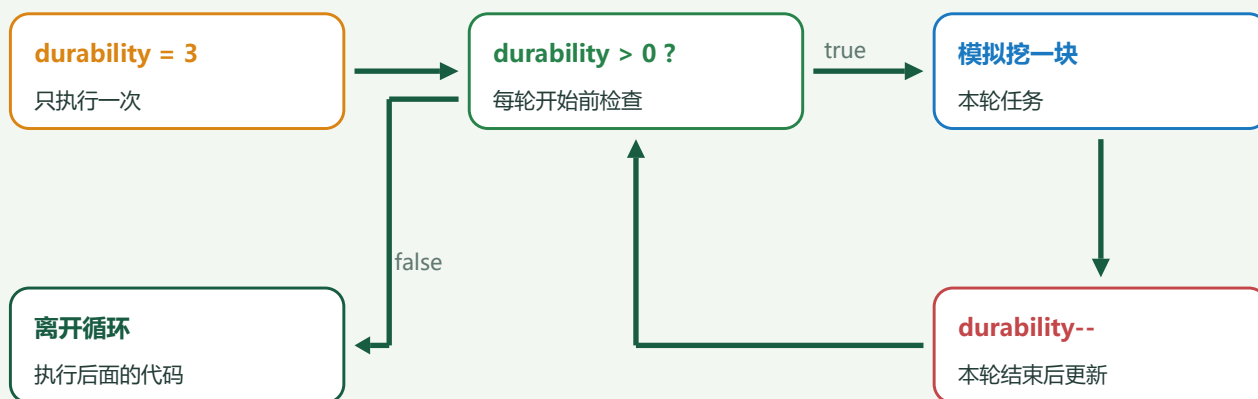
片段：放在 main 中，单独运行

```
int durability = 3;
while (durability > 0) {
    System.out.println("挖一下，动作前耐久=" + durability);
    durability--;
}
System.out.println("镐子坏了");
```

运行核对

挖一下，动作前耐久=3
挖一下，动作前耐久=2
挖一下，动作前耐久=1
镐子坏了

先检查，动作和更新后再回去



打印1之后才减到0，下一次检查才离开。本课短片段分别放进 main 运行。while 也能用于次数明确的任务；for 与 while 的选择主要看哪种形式更清楚。

03 让状态向停止条件移动

片段：放在 main 中，单独运行

```
int durability = 0;
while (durability > 0) {
    System.out.println("这一行不会出现");
}
System.out.println("先修工具");
```

运行核对

先修工具

每轮不一定只消耗1点。消耗3点时，耐久8会经过5、2，然后跨过0。Math.max(0, 新值) 将显示值夹到0，不让耐久条出现负数。

片段：放在 main 中，单独运行

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

运行核对

剩余耐久=5
剩余耐久=2
剩余耐久=0

追踪表：写清动作前后

轮次	动作前	直接减3	显示的新耐久
1	8	5	5
2	5	2	2
3	2	-1	0
下一次	0	条件为false	不再挖掘

删掉 durability 的更新，条件就不会朝停止方向变化。循环不是自己知道何时下班，它只会反复检查你写下的状态。

04 找不到目标，也要有安全出口

先用确定位置研究搜索，避免随机结果干扰理解。设第7块有钻石：每挖一次，块数加1，再判断是不是目标。最多20块，两项继续条件必须同时满足。

片段：放在 main 中，单独运行

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

运行核对

挖了7块，找到=true

两道护栏：任意一项失败就停止

!found
还没找到，才需要继续搜索。

blocks < 20
仍在次数上限内，才允许继续。

用 && 连接
成功或到上限，都会离开循环。

一次只改 diamondAt

目标位置	实际块数	停止结果
1	1	找到=true
7	7	找到=true
20	20	找到=true
30	20	找到=false，到上限

若写成 ||，找到后仍可能继续；目标在上限之外时也会越过护栏。上限保证停机，不保证一定找到目标。

05 do-while：先做一次，再检查

把条件放在末尾，循环体就至少执行一次。适合必须先展示的菜单或先发生的一次尝试。最后的 while (条件) ; 有一个必需的分号。

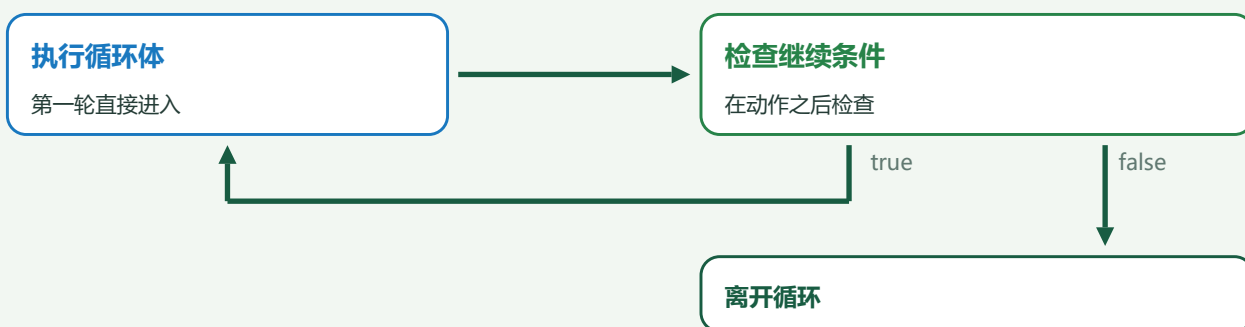
片段：放在 main 中，单独运行

```
int choice = 3;
do {
    System.out.println("1 挖矿 2 修理 3 退出");
} while (choice != 3);
```

运行核对

1 挖矿 2 修理 3 退出

至少一轮：先执行，再检查



片段：放在 main 中，单独运行

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

运行核对

修理后=3
修理后=6
修理后=9

第一个例子只演示一次显示，没有读取选择。要让菜单重复，循环里还要更新 choice；真正的交互菜单在课后挑战中完成。

06 先检查，还是先动作？

差别在第一次检查的时机

while

先检查再做，允许0轮。工具已损坏时，不应继续挖。

do-while

先做再检查，至少1轮。退出菜单也要先让玩家看见。

片段：放在 main 中，单独运行

```
int torches = 0;
while (torches > 0) {
    System.out.println("放火把");
    torches--;
}
System.out.println("结束");
```

运行核对

结束

片段：放在 main 中，单独运行

```
int items = 0;
do {
    System.out.println("检查箱子");
} while (items > 0);
```

运行核对

检查箱子

不是哪一种更高级，而是第一轮动作应不应该先发生。别为了保证“至少一次”，把已经损坏的工具交给 do-while 强行挖一下。

do { ... } while (条件); 的末尾分号属于语法；while (条件); 后再写代码块，分号却会制造空循环。这两个位置的作用完全不同。

07 Random：让变化可以重放

伪随机可以模拟不同消耗。测试时使用固定种子10，相同种子加上相同调用顺序，会得到相同序列。这样遇到 Bug，能重放同一组消耗。

片段：放在 main 中，单独运行

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

运行核对

3

片段：放在 main 中，单独运行

```
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++;
}
```

运行核对

3
2
2
2
1

随机范围：上界不包含

表达式	可能结果
<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

同一个 Random 放在循环外创建一次。每轮重新 `new Random(10)`，会反复拿到序列的第一个数。想让游玩结果变化可改为 `new Random()`，测试时仍保留固定种子。

08 停不下来的循环急诊室



读代码查原因，不运行失控反例

- 条件变量没有更新：durability 保持5，durability > 0 就一直成立。
- while 后多了分号：循环体变成空语句，后面的更新根本进不去。
- && 写成 ||：只要有一项为true就继续，成功和上限都失去独立停机作用。
- 安全上限计数没有更新：看似写了上限，实际永远到不了它。

片段：放在 main 中，单独运行

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

运行核对

已挖1块
已挖2块
已挖3块

发现控制台异常刷屏，先用停止按钮或 Ctrl+C 结束程序。再核对条件、更新、方向、分号与上限，每次只修一处。计数器也要在合理范围内，别用溢出来当出口。

09 预测变化，也预测停止原因

把循环的出口一起写进答案

预测

列出每轮开始与结束的状态。



运行

核对实际序列，不只看最后一行。



解释

指出哪一项继续条件先失效。

片段：放在 main 中，单独运行

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

片段：放在 main 中，单独运行

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

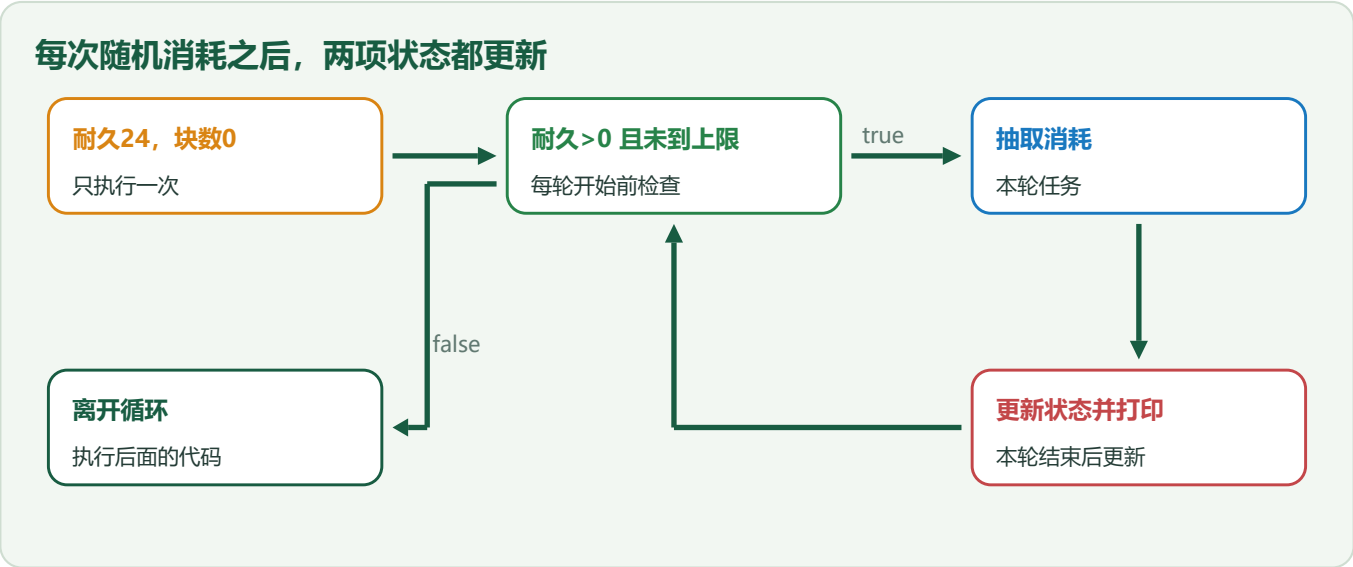
片段：放在 main 中，单独运行

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

核对：第一题逐行输出5、3、1、0，耐久归零；第二题输出3:false，次数上限先到；第三题输出一行“检查”，随后条件为false。

10 项目蓝图：工具耐久模拟器

用24点耐久模拟挖掘，每轮随机消耗1到4点，最多50块。打印轮次、消耗、剩余耐久和耐久条，再明确说明为什么停止。



先写契约，再写代码

项目	本课规则
允许设定	耐久和上限均为0到100的整数
随机	循环外 new Random(10), 每轮抽取一次
显示	Math.max 保证耐久不低于0
耐久条	每2点显示一个 #, 只作简易比例提示
结束	耐久为0报告损坏, 否则报告安全上限

这是工具消耗项目，不模拟随机钻石掉落。前面的钻石搜索已演示成功出口；若扩展本项目，也加入 !found，并写清“找到、损坏、上限”同时发生时的提示优先级。

11 模拟器代码：初始化与消耗

新建 `DurabilitySimulator.java`。完整程序分两页，依次连在同一文件内。每轮只抽取一次消耗，方便和固定序列逐项对照。

DurabilitySimulator.java • 上半部分，下一页继续

```
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("耐久和上限必须在0到100之间");
        } else {
            System.out.println("=== 工具耐久模拟 ===");
            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("第%02d块 消耗%d 剩余%02d %s\n",
                    blocksMined, cost, durability, bar);
            }
        }
    }
}
```



`String.repeat` 是第5课见过的工具，`durability / 2` 用整数除法，所以耐久1时没有 `#`，并不代表已经损坏；判断损坏仍用 `durability == 0`。

12 模拟器收尾：说清停机原因

DurabilitySimulator.java • 紧接上一页

```
        if (durability == 0) {
            System.out.println("工具损坏，停止挖掘");
        } else {
            System.out.println("触发安全上限，带着工具回家");
        }
        System.out.println("共挖掘: " + blocksMined + "块");
    }
}
```

两项条件都可能令循环结束。如果最后耐久刚好归零，同时也达到上限，本项目优先报告工具损坏。先把提示优先级写清楚，就不会把结果交给运气决定。

标准版24点耐久，前五轮可手算

块数	本次消耗	剩余耐久	耐久条长度
1	3	21	10
2	2	19	9
3	2	17	8
4	2	15	7
5	1	14	7

固定种子10的标准程序第11块归零，最后报告“工具损坏，停止挖掘”和“共挖掘：11块”。反复运行两次，应得到相同的完整输出。

13 验收：证明两扇出口都能用

只改一个常量，观察不同出口

测试设定	块数	剩余耐久	停止原因
耐久0, 上限50	0	0	工具损坏
耐久1, 上限50	1	0	工具损坏
耐久24, 上限50	11	0	工具损坏
耐久24, 上限3	3	17	安全上限
耐久24, 上限0	0	24	安全上限
耐久1, 上限1	1	0	优先报告损坏
耐久-1或上限101	不挖掘	不统计	报告范围错误

验证的是规则，不是“看起来没崩”

- 每轮的消耗都在1到4之间，剩余耐久永不为负。
- 总块数等于实际打印的挖掘记录数。
- 上限3时，恰好打印3条，不多挖第4块。
- 标准版重复运行，所有行一致；更换种子后也守住范围和上限。

不要照着“删掉计数器再看看”去运行失控循环。静态检查能看出上限无效；需要实验时，用另一个独立、明确的小上限约束次数。

14 给未知终点装上可靠护栏



木镐任务：从十倒数到零

只用 while 输出10到0，再打印“发射”。共11个数字；另测起点0和1，解释为什么 ≥ 0 才能包含最后的0。

铁镐任务：饥饿值下降模拟

初始20，每轮随机下降2到5，固定种子10，显示值夹到0。打印轮次、消耗与剩余值，并设安全上限；改成上限2时必须提前停止。

钻石镐任务：可以重复的菜单

Scanner 配合 do-while：1挖矿、2修理、3退出。挖矿前查耐久，修理最多到20；未知输入提示重试。用字符串选择可避免数字解析错误，菜单须至少显示一次。

验收菜单：3只显示一次；1、3先挖再退出；2、3先修再退出；0、abc能提示并继续；工具耐久0时不能挖。
下一课学习方法，把这些动作整理成独立技能。

技术核对：Oracle Java 语言规范第14章；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>