

第16课

阶段项目 背包管理员

综合基础语法项目

两组数组，一份背包记录

格子	物品	数量
0	木头	12
1	圆石	32
2	null	0

会把变量 条件 循环 方法和数组装进一个可运行项目；能设计菜单循环和安全输入；能完成查看 添加 删除 查找四项功能；能用测试表验收背包管理员。
前置背包：第1至15课的变量、输入、条件、循环、方法与数组。今天不是再学一块新砖，而是把旧砖盖成能住人的程序。

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：把示例放对地方

本章使用 JDK 21，不需要预览功能。每个普通示例都给出完整程序，按标注的文件名保存。多个示例各放在自己的文件夹，避免辅助类同名。

终端命令：把中文占位符替换成实际名称

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

类名区分大小写。完整挑战代码若分成多段，请按顺序接在同一个文件中，不要为续段另建类。运行核对只列程序输出；需要输入的地方另给输入序列。

学习动作：先写下你的预测，运行后比较，再只改一个条件。标有“故意错误”的实验用于认识 Bug；它们与可直接运行的示例有清楚区分。

01 项目地图与验收条件

前十五课的知识今天组队打 挑战。挑战 不是末影龙，而是一个会不停问“请选择菜单”的控制台程序。

最终程序能查看、添加、删除、查找物品，并且遇到错误输入不会原地爆炸。

背包管理员先画地图再开工

菜单

重复接收操作

配对数组

同格保存名字与数量

规则方法

查看、添加、删除、
查找

验收

正常、边界、错误三
类

完成标准

- 菜单会重复显示，选择0才退出。
- 物品名和数量使用相同索引保存。
- 添加和删除都检查数量与背包边界。
- 至少准备正常、边界、错误三类测试。

项目纪律：每完成一个功能就运行一次。不要把四项功能同时写完再祈祷编译器心情好。

02 背包的数据模型

一个格子要保存物品名和数量。当前阶段还没学习自定义物品类，所以先用两个等长数组配对。

同一索引是一对数据

索引	items[i]	counts[i]
0	木头	12
1	圆石	32
2	null	0

L16Example02.java • 完整独立示例

```
public class L16Example02 {
    public static void main(String[] args) {
        String[] items = {"木头", "圆石", null};
        int[] counts = {12, 32, 0};
        int slot = 1;
        System.out.println(items[slot] + " x" + counts[slot]);
    }
}
```

运行核对

圆石 x32

配对规则：items[i] 和 counts[i] 必须一起修改。只换名字不换数量，会出现“空气 x64”的灵异箱子。

03 菜单循环控制程序

菜单型程序通常不知道玩家要操作几次，所以使用 while。每轮显示菜单、读取选择、执行功能，再回到入口。

菜单循环：0 才是出口



菜单在每一轮显示；退出设 `running = false`，输入结束也安全退出。

L16Example03.java • 完整独立示例

```

public class L16Example03 {
    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        boolean running = true;
        while (running && scanner.hasNextLine()) {
            String choice = scanner.nextLine().trim();
            switch (choice) {
                case "1" -> System.out.println("打开背包");
                case "0" -> running = false;
                default -> System.out.println("没有这个按钮");
            }
        }
        System.out.println("已退出");
    }
}
    
```

运行核对

打开背包
没有这个按钮
已退出

退出钥匙：退出分支必须改变循环条件。只打印“再见”却不改 `running`，菜单会礼貌地把你请回来。

04 安全读取整数

Scanner 的 `nextLine` 先得到文本，再尝试转换为整数。输入“钻石”时，程序应给提示，而不是用异常把玩家踢出世界。

输入先过检查站再进入背包

文本

读取并去除边缘空白

`parseInt`

尝试转成int

正整数

成功且大于0才接受

非法

-1报告失败

L16Example04.java • 完整独立示例

```
public class L16Example04 {
    static int positive(String text) {
        try {
            int n = Integer.parseInt(text.trim());
            return n > 0 ? n : -1;
        } catch (NumberFormatException ex) { return -1; }
    }
    public static void main(String[] args) {
        System.out.println(positive(" 12 "));
        System.out.println(positive("钻石"));
        System.out.println(positive("0"));
        System.out.println(positive("999999999999"));
    }
}
```

运行核对

```
12
-1
-1
-1
```

输入边界：空文本、0、负数和字母都属于错误输入。先拦住它们，再让背包方法工作。

05 查看背包

查看功能需要打印索引，所以普通 for 比增强 for 更合适。空格统一显示为“空位”。

普通 for 同时拿到格子编号和内容

有物品

打印名字与数量

空位

统一显示空位

索引

显示0到length减1

数组

必须等长且配对合法

L16Example05.java • 完整独立示例

```
public class L16Example05 {
    static void show(String[] items, int[] counts) {
        for (int i = 0; i < items.length; i++) {
            String text = items[i] == null ? "空位" : items[i] + " x" + counts[i];
            System.out.println(i + ": " + text);
        }
    }
    public static void main(String[] args) {
        String[] items = {"木头", null};
        int[] counts = {12, 0};
        show(items, counts);
    }
}
```

运行核对

0: 木头 x12
1: 空位

验收：长度为5的背包必须输出0到4共五行。漏一格，通常是循环边界写成了 `length - 1`。

06 查找物品

搜索方法返回索引，让添加、删除和查询功能都能复用。目标名称先检查 null。

搜索循环遇到目标就返回索引

target

先排除null

equals

按文字内容比较

找到

返回首个索引

未找到

返回-1

L16Example06.java • 完整独立示例

```
public class L16Example06 {
    static final int MAX_STACK = 64;
    static int find(String[] items, String target) {
        if (target == null) return -1;
        for (int i = 0; i < items.length; i++)
            if (target.equals(items[i])) return i;
        return -1;
    }
    public static void main(String[] args) {
        String[] items = {"木头", "圆石", null};
        System.out.println(find(items, "圆石"));
        System.out.println(find(items, "钻石"));
        System.out.println(find(items, null));
    }
}
```

运行核对

```
1
-1
-1
```

安全比较：使用 `target.equals(items[i])`。即使背包格是 null，也不会召唤 `NullPointerException`。

07 添加与堆叠

添加物品有两条路线：背包已有该物品时增加数量；没有时放进第一个空格。每格最多64个。

先找同类物品 再找空格

已存在

使用已有格子

新物品

找第一个空格

余量

先检查64减当前数量

成功

同步写入名字和数量

马老师编程
www.teacher-ma.de

09 / 21

07 添加与堆叠 (续)

L16Example07.java • 完整独立示例

```
public class L16Example07 {
    static final int MAX_STACK = 64;
    static int find(String[] items, String target) {
        if (target == null) return -1;
        for (int i = 0; i < items.length; i++)
            if (target.equals(items[i])) return i;
        return -1;
    }
    static int empty(String[] items) {
        for (int i = 0; i < items.length; i++)
            if (items[i] == null) return i;
        return -1;
    }
    static boolean add(String[] items, int[] counts, String item, int n) {
        if (item == null || item.isBlank() || n <= 0) return false;
        int slot = find(items, item);
        if (slot < 0) slot = empty(items);
        if (slot < 0 || n > MAX_STACK - counts[slot]) return false;
        items[slot] = item;
        counts[slot] += n;
        return true;
    }
    public static void main(String[] args) {
        String[] items = {"木头", null};
        int[] counts = {12, 0};
        System.out.println(add(items, counts, "木头", 5));
        System.out.println(add(items, counts, "木头", Integer.MAX_VALUE));
        System.out.println(counts[0]);
    }
}
```

运行核对

```
true
false
17
```

规则：每种物品只占一个格子，每格最多64个。用 $n > 64 - \text{counts}[\text{slot}]$ 先检查余量，避免 $\text{counts}[\text{slot}] + n$ 先溢出再逃过检查。自动分成多堆是拓展任务。

先算后改：先确认新数量不超过64，再写回数组。不要先塞进去，再发现箱盖关不上。

08 删除与清空格子

删除方法先找到物品，再确认请求数量合法。减到0时必须同时清空物品名。

数量归零时名字和数量一起清空

查找

未找到就拒绝

数量

大于0且不超库存

扣除

检查通过后再修改

清空

归零时名字也变null

不欠材料：库存不能变成负数。删除13个木头而背包只有12个时，方法应返回false且数据不变。

马老师教编程
www.teacher-ma.de

11 / 21

08 删除与清空格子（续）

L16Example08.java • 完整独立示例

```
public class L16Example08 {
    static final int MAX_STACK = 64;
    static int find(String[] items, String target) {
        if (target == null) return -1;
        for (int i = 0; i < items.length; i++)
            if (target.equals(items[i])) return i;
        return -1;
    }
    static boolean remove(String[] items, int[] counts, String item, int n) {
        int slot = find(items, item);
        if (slot < 0 || n <= 0 || n > counts[slot]) return false;
        counts[slot] -= n;
        if (counts[slot] == 0) items[slot] = null;
        return true;
    }
    public static void main(String[] args) {
        String[] items = {"木头"};
        int[] counts = {12};
        System.out.println(remove(items, counts, "木头", 13));
        System.out.println(counts[0]);
        System.out.println(remove(items, counts, "木头", 12));
        System.out.println(items[0] + ":" + counts[0]);
    }
}
```

运行核对

```
false
12
true
null:0
```

09 方法分工

main 负责菜单与流程，功能方法负责数据规则。这样出错时能快速判断是哪一位“村民”交错了货。

main 当调度员 方法当专业村民

显示
show负责输出

查找
find返回索引

修改
add和remove维护规则

输入
main读取并安排操作

main 只选择功能；读取流程负责收集名称和数量；add/remove 只维护数组。完整菜单代码在本章挑战实现中，直接调用真实功能，不留下未定义的 addFlow。

验收表

方法	输入	返回或效果
show	两个数组	打印每格状态
find	物品数组、名称	索引或-1
add	数组、名称、数量	成功或失败
remove	数组、名称、数量	成功或失败
positive	一行文本	正整数或-1

单一职责：方法名像工具箱标签。看到 addFlow 就应知道它处理添加流程，而不是顺手修改天气。

10 三类测试

项目不是“能运行一次”就通关。每项功能至少准备一个正常测试、一个边界测试和一个错误测试。

测试台照出正常 边界和错误

正常

功能能完成

边界

恰满、恰空

错误

非法数量与未知命令

回归

修复后重跑原测试

验收表

功能	输入	预期
添加	木头 5	数量增加5
添加	圆石 40 到已有32	拒绝 超过64
删除	木头 0	拒绝 数据不变
查找	不存在的钻石	返回-1
菜单	x	提示没有按钮

回归测试：修好一个 Bug 后重跑旧测试。苦力怕有时会在你补墙时从另一扇门进来。

11 项目故障诊所

项目变大后，不要凭感觉乱改。先记录输入和现象，再定位负责该规则的方法。

从现象追到负责的方法

死循环

退出未改变条件

配对错

名字与数量不同步

越界

索引不合法

超量

检查失败必须不改数据

L16Example11.java • 完整独立示例

```
public class L16Example11 {
    public static void main(String[] args) {
        String[] items = {"木头"};
        int[] counts = {12};
        counts[0] = 0;
        System.out.println(items[0] + " x" + counts[0]);
    }
}
```

运行核对

木头 x0

这不是编译错误，却破坏“空位对应 null 和0”的约定。数量减到0时，还要把同格物品名设为 null。退出菜单也一样：打印“再见”不等于修改 running。

急诊顺序：先复现；打印 choice、slot、amount；确认数组长度；最后做最小修改并重跑测试。

12 挑战 蓝图

挑战 程序叫 InventoryManager。完整版本包含可重复操作的控制台菜单，查看、添加、删除、查找都有真实实现。先完成数据方法，再接入菜单。

背包管理员的四个核心功能

先实现

查看与查找

再实现

添加与删除

接菜单

命令调用真实功能

验收

失败时两个数组不变

验收顺序

- 显示初始背包。
- 添加5个木头并确认数量。
- 加入3个钻石并确认占用空格。
- 删除全部圆石并确认格子清空。
- 尝试添加导致超过64的数量，必须失败。

验收表

规则	合法范围	失败表现
索引	0到length-1	不访问数组
数量	大于0	返回false
堆叠	最多64	数据不变
空格	至少一个null	返回false

先搭骨架：先写 show 与 find 并运行，再添加 add，最后写 remove。每次只让一个新方法加入队伍。

13 挑战 实现

新建 InventoryManager.java。程序不依赖真实 Minecraft 接口，可直接在 Java 21 控制台编译运行。

完整程序把五种基础语法连成一条链

基础

数组与查找

规则

余量检查与清空

菜单

while与switch

输入

异常处理与结束检查

验收：按给定输入先添加5个木头，再查看背包，木头应为17。继续手动添加3个钻石、删除32个圆石、尝试超量添加；每次都核对返回提示与背包状态。

InventoryManager.java • 第1/3段，顺序接在同一文件

```
public class InventoryManager {
    static final int MAX_STACK = 64;
    static int find(String[] items, String target) {
        if (target == null) return -1;
        for (int i = 0; i < items.length; i++)
            if (target.equals(items[i])) return i;
        return -1;
    }
    static int empty(String[] items) {
        for (int i = 0; i < items.length; i++)
            if (items[i] == null) return i;
        return -1;
    }
    static boolean add(String[] items, int[] counts, String item, int n) {
        if (item == null || item.isBlank() || n <= 0) return false;
        int slot = find(items, item);
        if (slot < 0) slot = empty(items);
        if (slot < 0 || n > MAX_STACK - counts[slot]) return false;
        items[slot] = item;
        counts[slot] += n;
        return true;
    }
    static boolean remove(String[] items, int[] counts, String item, int n) {
        int slot = find(items, item);
        if (slot < 0 || n <= 0 || n > counts[slot]) return false;
        counts[slot] -= n;
        if (counts[slot] == 0) items[slot] = null;
        return true;
    }
}
```

13 挑战 实现 (续)

InventoryManager.java • 第2/3段, 顺序接在同一文件

```
}
static void show(String[] items, int[] counts) {
    for (int i = 0; i < items.length; i++) {
        String text = items[i] == null ? "空位" : items[i] + " x" + counts[i];
        System.out.println(i + ": " + text);
    }
}
static int positive(String text) {
    try {
        int n = Integer.parseInt(text.trim());
        return n > 0 ? n : -1;
    } catch (NumberFormatException ex) { return -1; }
}
public static void main(String[] args) {
    String[] items = {"木头", "圆石", null, null, null};
    int[] counts = {12, 32, 0, 0, 0};
    java.util.Scanner input = new java.util.Scanner(System.in);
    boolean running = true;
    while (running) {
        System.out.println("1查看 2添加 3删除 4查找 0退出");
        if (!input.hasNextLine()) break;
        String choice = input.nextLine().trim();
        switch (choice) {
            case "0" -> running = false;
            case "1" -> show(items, counts);
            case "2", "3" -> {
                System.out.println("物品名: ");
                if (!input.hasNextLine()) { running = false; break; }
            }
        }
    }
}
```

13 挑战 实现 (续)

InventoryManager.java • 第3/3段, 顺序接在同一文件

```

        String item = input.nextLine().trim();
        System.out.println("正整数数量: ");
        if (!input.hasNextLine()) { running = false; break; }
        int n = positive(input.nextLine());
        boolean ok = choice.equals("2")
            ? add(items, counts, item, n)
            : remove(items, counts, item, n);
        System.out.println(ok ? "操作成功" : "操作拒绝, 背包不变");
    }
    case "4" -> {
        System.out.println("物品名: ");
        if (!input.hasNextLine()) { running = false; break; }
        int slot = find(items, input.nextLine().trim());
        System.out.println(slot < 0 ? "没找到" : "索引: " + slot);
    }
    default -> System.out.println("没有这个按钮");
}
}
System.out.println("背包已关闭");
}
}

```

运行核对

1查看 2添加 3删除 4查找 0退出

物品名:

正整数数量:

操作成功

1查看 2添加 3删除 4查找 0退出

0: 木头 x17

1: 圆石 x32

2: 空位

3: 空位

4: 空位

1查看 2添加 3删除 4查找 0退出

背包已关闭

运行核对使用输入: 2、木头、5、1、0, 每项单独一行。菜单可以反复操作; 输入流结束也退出。方法约定两个数组非 null、等长, 且初始名字与数量配对合法。

14 拓展任务与阶段复盘

这是基础语法阶段的 挑战 关。提交源代码、运行结果、测试表和一条 Bug 记录。

三档扩建让背包真正像项目

木镐

重现基本菜单

铁镐

补齐测试与空位统计

钻石镐

研究多个堆叠

复盘

记录数据规则

三档作业

- 木镐：按完整菜单重现查看、添加、退出；提交输入与输出记录。
- 铁镐：完成删除与查找的边界测试，新增空格统计，确认错误输入不会改数据。
- 钻石镐：允许每种物品拥有多个堆叠，自动寻找未满64的格子。

验收表

自评	能	还要练
我能拆分项目方法	☐	☐
我能维护配对数组	☐	☐
我会写三类测试	☐	☐

下一课：第17课进入生物设计院。我们会用 class 把状态和行为装进同一张生物蓝图。

自己的验收记录

运行之前写预期，运行之后记录实际结果。正常案例说明程序会做事；边界案例说明它知道何时停；错误案例说明它能拒绝不合理的请求。

把证据写进记录

类别	你选择的输入	预期与实际
正常	一个常用值或一次成功操作	是否逐行一致？
边界	最小值、最大值及相邻值	有没有少一格、多一步？
错误	非法数量、缺失对象或错误语法	是否明确拒绝？状态是否保持？

- 把一个 Bug 写成：触发条件、现象、原因、最小修复。
- 整理一个值得记住的规则，用自己的话解释，不能只复制定义。
- 改变一个示例，保留测试记录，让结果证明你的理解。

技术核对依据：Java SE 21 语言规范，作用域与访问控制（第6章）、类与构造器（第8章）、方法调用（第15章）。 <https://docs.oracle.com/javase/specs/jls/se21/html/index.html>