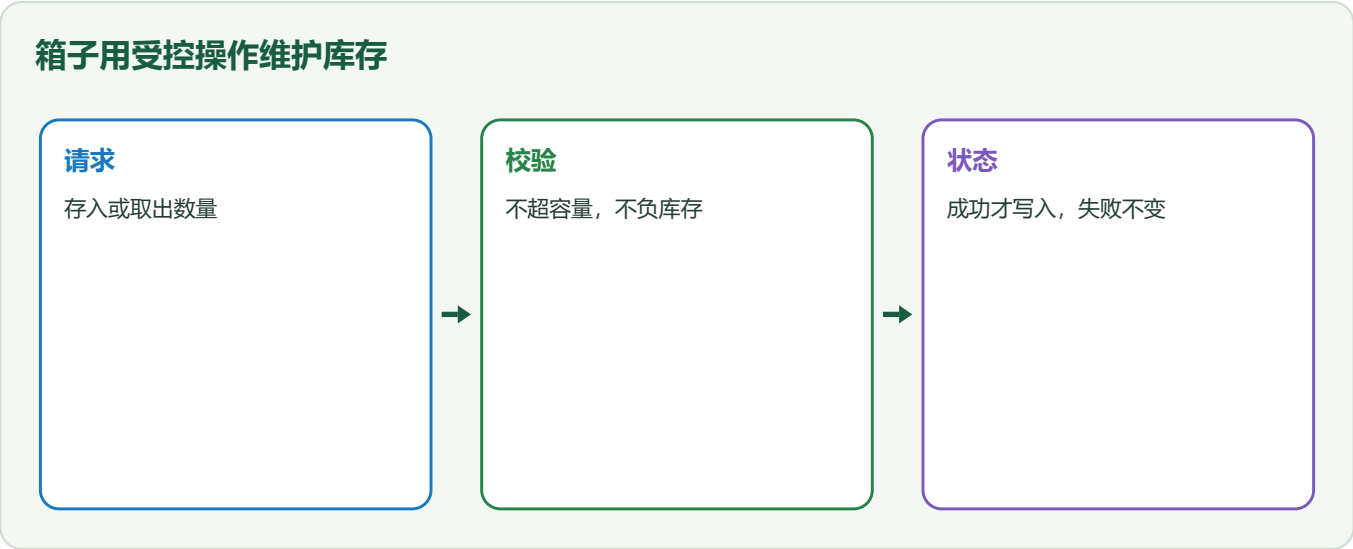


第20课

锁好你的箱子

封装 private 与校验



会用 private 保护对象字段；能通过方法提供受控读取和修改；能用校验守住容量与非负库存；能避免把内部可变数组直接交给外部代码。

前置背包：类、对象、构造器、方法参数与返回值。今天要给数据装门、装锁，还要设计真正好用的按钮。

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 封装让规则只有一个入口

public 字段像没有门的钻石仓库：任何代码都能把库存改成 -999。封装把数据和维护数据的规则放在同一个类里。

private 把字段藏进箱体

字段

private保存内部状态

方法

公开受控入口

校验

先判断合法性

规则

0到容量内的库存

L20Example01.java • 完整独立示例

```
class Chest { public int diamonds; }
public class L20Example01 {
    public static void main(String[] args) {
        Chest chest = new Chest();
        chest.diamonds = -999;
        System.out.println(chest.diamonds);
    }
}
```

运行核对

-999

这个示例故意暴露字段。程序能够运行，但负库存违反业务规则；封装要防的是这种容易发生的错误。private 是访问控制，不是文件加密或防黑客密码。

封装目标

- 外部不能随意写字段。
- 所有修改经过同一套检查。
- 类对外暴露清楚而精简的方法。
- 通过校验的公开操作应保持对象合法状态。

锁不是墙：private 不是让数据永远不可用，而是要求外部通过经过设计的方法访问。

02 private 的可见范围

本章把箱子与调用程序写成两个独立类：箱子内部可以维护 `private` 字段，调用程序使用公开方法。Java 还允许同一最外层类及其嵌套类之间访问 `private` 成员；不要用这个例外检验外部访问规则。

私有字段只在自己的类体中直接访问

箱子内部

可维护私有字段

独立外部类

不能直接写私有字段

公开方法

提供允许的操作

例外

同一外层类的嵌套成员

故障实验：独立的外部类不能直接访问 `Wallet` 的 `private` 字段。下面故意保留错误，请先预测，再尝试修复。

BadPrivate.java • 故意错误示例

```
class Wallet { private int emeralds; }
public class BadPrivate {
    public static void main(String[] args) {
        Wallet wallet = new Wallet();
        wallet.emeralds = 99;
    }
}
```

编译器守门： `private` 访问错误在编译阶段就会被发现，Bug 还没进世界就被门卫拦下。

03 getter 提供受控读取

外部若需要查看状态，可提供 getter 或更有意义的查询方法。不是每个字段都必须配一个 getter。

读取方法只交出需要的信息

getCount

读取当前库存

remainingSpace

读取剩余空间

查询

不修改状态

最少公开

只提供真正需要的信息

少开窗口：只公开调用者真正需要的信息。坐标、缓存或内部计数若不该被外部依赖，就不要顺手提供 getter。

03 getter 提供受控读取（续）

L20Example03.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class L20Example03 {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 64);
        System.out.println(chest.getCount());
        System.out.println(chest.remainingSpace());
    }
}
```

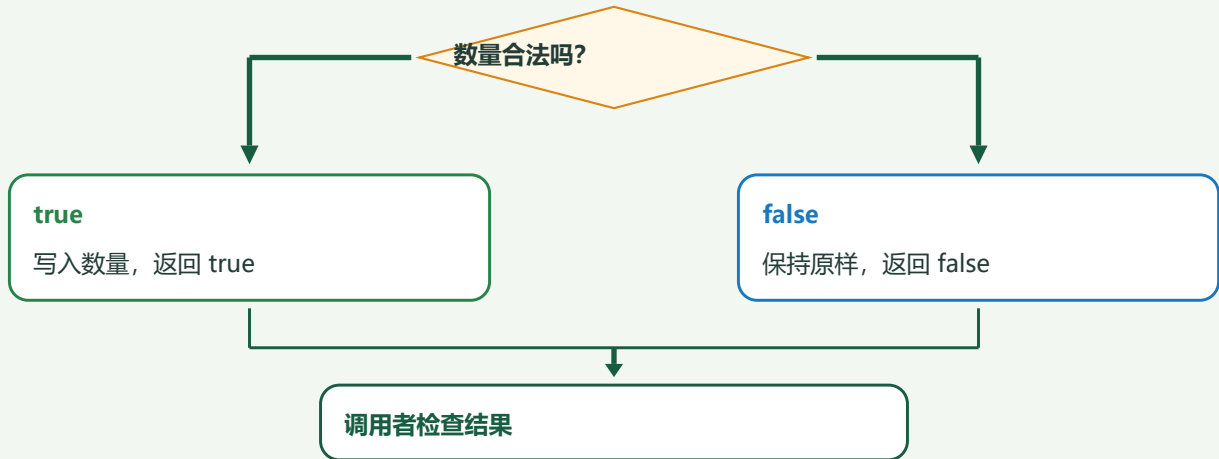
运行核对

0
64

04 行为方法比万能 setter 更安全

setCount(-999) 让调用者承担所有规则。deposit 和 withdraw 则表达真实动作，并能集中检查。

先检查，后修改



04 行为方法比万能 setter 更安全 (续)

L20Example04.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class L20Example04 {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 64);
        System.out.println(chest.deposit(12));
        System.out.println(chest.withdraw(3));
        System.out.println(chest.getCount());
    }
}
```

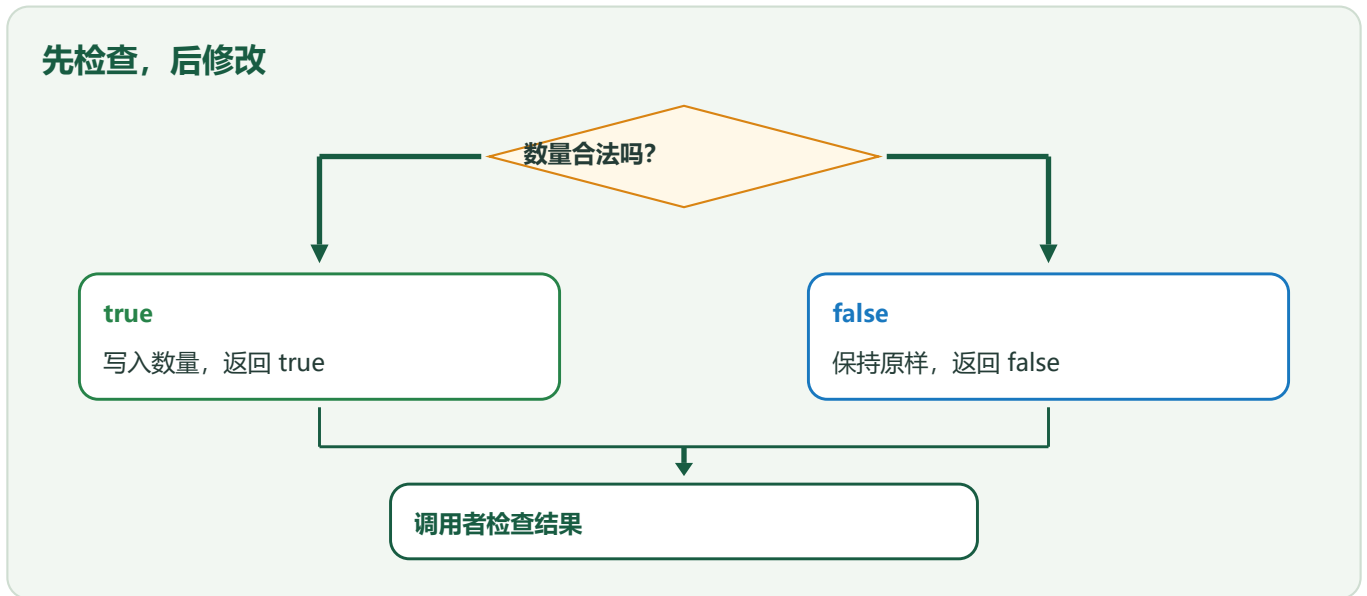
运行核对

```
true
true
9
```

方法像按钮：箱子可以有“存入5个”和“取出3个”按钮，但不必安装“把内部计数随便改成任何数”的拉杆。

05 校验守住合法范围

失败操作不应修改对象。先检查所有条件，再写字段。



05 校验守住合法范围（续）

L20Example05.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class L20Example05 {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 64);
        chest.deposit(12);
        int before = chest.getCount();
        boolean ok = chest.deposit(Integer.MAX_VALUE);
        System.out.println(!ok && chest.getCount() == before);
    }
}
```

运行核对

true

容量可能很大，`count + amount` 可能先发生 `int` 溢出。用 `amount > capacity - count` 比较剩余空间，在已满足 `0 <= count <= capacity` 的前提下，这个减法安全。这里的“成功或不变”指单次调用的契约，不涉及多线程事务。

原子动作：方法要么完整成功，要么保持原样。不要返回 `false` 却留下已经改变一半的数据。

06 boolean 汇报操作结果

修改方法返回 boolean，可以让调用者知道操作是否完成，而不必猜测字段变化。

调用者根据成功或失败决定下一步

true

表示操作完成

false

表示拒绝且不变

提示

调用者选择反馈

测试

同时检查结果与状态

马老师教编程
www.teacher-ma.de

11 / 26

06 boolean 汇报操作结果（续）

L20Example06.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class L20Example06 {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 64);
        if (chest.deposit(12)) System.out.println("已存入");
        int before = chest.getCount();
        boolean ok = chest.withdraw(999);
        System.out.println(!ok && chest.getCount() == before);
    }
}
```

运行核对

已存入
true

清楚契约：true 表示动作完成；false 表示拒绝且状态不变。把这条契约写进测试。

07 容量与非负库存

SecureChest 的核心不变条件是 $0 \leq \text{count} \leq \text{capacity}$ 。构造器先建立合法容量，之后每个方法继续守住它。

上下边界像箱子的底和盖

0

合法的空箱

capacity

合法的满箱

负数

不允许库存为负

越界

不允许超过容量

07 容量与非负库存（续）

L20Example07.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class L20Example07 {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 0);
        System.out.println(chest.remainingSpace());
        System.out.println(chest.deposit(1));
        System.out.println(chest.deposit(1));
        System.out.println(chest.withdraw(1));
    }
}
```

运行核对

```
1
true
false
true
```

07 容量与非负库存（续）

验收表

输入	是否允许	原因
存入0	否	数量必须大于0
存入到刚好满	是	等于容量合法
取出全部	是	库存可以为0
取出超过库存	否	不能为负数

边界测试：重点测试0、1、capacity和capacity+1。边界像箱盖，最容易夹住Bug的尾巴。

08 别暴露内部可变数组

private 数组若被 getter 原样返回，外部仍能通过引用修改内部内容。字段虽然 private，后门却敞开了。

把数组引用交出去等于把后门钥匙送人

内部数组

private仍是可变对象

直接返回

会共享内部引用

copyOf

新数组，元素引用浅复制

安全性

String不可变，对象仍需检查

copyOf 创建新的数组，但元素引用仍是原引用，这是浅拷贝。本例元素是不可变的 String，替换副本格子不会改原数组；若元素是可变宠物对象，修改宠物状态仍可能影响原对象。final 也只阻止数组字段改指向别的数组。

引用知识回归：第18课的共享引用在这里再次出现。封装不仅看 private，还要看有没有把可变对象引用交出去。

08 别暴露内部可变数组（续）

L20Example08.java • 完整独立示例

```
class ItemBox {
    private final String[] items = {"钻石", "火把"};
    public String[] getItemsCopy() {
        return java.util.Arrays.copyOf(items, items.length);
    }
    public String getItem(int index) {
        if (index < 0 || index >= items.length) return null;
        return items[index];
    }
}

public class L20Example08 {
    public static void main(String[] args) {
        ItemBox box = new ItemBox();
        String[] copy = box.getItemsCopy();
        copy[0] = "泥土";
        System.out.println(copy[0]);
        System.out.println(box.getItem(0));
        System.out.println(box.getItem(9));
    }
}
```

运行核对

泥土
钻石
null

09 设计箱子的公开接口

这里的“公开接口”指调用者可使用的方法集合，并非 Java 的 interface 关键字。先写调用者需要做什么，再设计最少的方法集合。

先列允许的操作 再决定字段

deposit
受控存入

withdraw
受控取出

getCount
库存查询

remainingSpace
余量查询

验收表

公开方法	输入	返回	保证
deposit	正数量	boolean	不超过容量
withdraw	正数量	boolean	不低于0
getCount	无	int	只读取库存
remainingSpace	无	int	只读取空间
describe	无	String	提供摘要

少而够用：公开方法越多，未来越难改。先提供课程需求真正使用的操作。

10 三类测试锁住规则

封装测试不仅看返回值，还要通过公开查询确认对象状态。

正常 边界和错误测试都要检查状态

正常
存入12个

边界
正好存满、正好取空

错误
满箱存入、空箱取出

状态
拒绝后不得改变

10 三类测试锁住规则（续）

L20Example10.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class L20Example10 {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 64);
        System.out.println(chest.deposit(12));
        System.out.println(chest.deposit(52));
        System.out.println(chest.deposit(1));
        System.out.println(chest.getCount());
        System.out.println(chest.withdraw(64));
        System.out.println(chest.withdraw(1));
        System.out.println(chest.getCount());
    }
}
```

运行核对

```
true
true
false
64
true
false
0
```

10 三类测试锁住规则（续）

验收表

测试	操作	预期
正常	容量64存入12	true 库存12
边界	再存入52	true 库存64
错误	满箱再存1	false 库存64
边界	取出64	true 库存0
错误	空箱取1	false 库存0

只用公开入口：测试代码也不要偷看 private 字段。通过 getCount 和 remainingSpace 验证类的公开契约。

11 封装故障诊所

编译器能发现类外直接访问 `private`，却不能自动发现 `setter` 漏校验或数组引用泄漏。

private 报错和规则漏洞要分开处理

直接访问

编译器阻止

漏校验

编译器不会代做

先改后拒绝

已经破坏状态

数组泄漏

公开了内部可变引用

L20Example11.java • 完整独立示例

```
class BrokenChest {
    private int count = 0;
    public boolean withdraw(int n) {
        count -= n;
        return count >= 0;
    }
    public int getCount() { return count; }
}
public class L20Example11 {
    public static void main(String[] args) {
        BrokenChest c = new BrokenChest();
        System.out.println(c.withdraw(3));
        System.out.println(c.getCount());
    }
}
```

运行核对

```
false
-3
```

方法已经返回 `false`，数据却变成 `-3`。 `private` 无法替你补上错误的校验顺序；应先检查 `n > 0` 且 `n <= count`，再扣减。

急诊顺序：找到所有修改字段的方法；检查是否先校验后写入；确认失败时状态不变；检查是否返回内部可变对象。

12 挑战 实现 安全箱子

挑战 程序叫 SecureChest。箱子保存一种物品、容量和数量，外部只能使用公开方法。

private 字段和公开方法共同守住库存规则

构造器

初始化容量与名字

deposit

余量足够才加库存

withdraw

库存足够才扣除

查询

通过公开方法验证

验收：确认任何公开操作后都满足0到capacity；失败操作不改变count；类外无法直接访问三个private字段。

12 挑战 实现 安全箱子（续）

SecureChest.java • 完整独立示例

```
class Chest {
    private final String item;
    private final int capacity;
    private int count;
    public Chest(String item, int capacity) {
        this.item = item == null || item.isBlank() ? "未知物品" : item.trim();
        this.capacity = Math.max(1, capacity);
    }
    public boolean deposit(int amount) {
        if (amount <= 0 || amount > capacity - count) return false;
        count += amount;
        return true;
    }
    public boolean withdraw(int amount) {
        if (amount <= 0 || amount > count) return false;
        count -= amount;
        return true;
    }
    public int getCount() { return count; }
    public int remainingSpace() { return capacity - count; }
    public String describe() { return item + " " + count + "/" + capacity; }
}

public class SecureChest {
    public static void main(String[] args) {
        Chest chest = new Chest("钻石", 64);
        System.out.println(chest.deposit(12));
        System.out.println(chest.deposit(52));
        System.out.println(chest.deposit(1));
        System.out.println(chest.describe());
        System.out.println(chest.withdraw(64));
        System.out.println(chest.withdraw(1));
        System.out.println(chest.describe());
        System.out.println(chest.remainingSpace());
    }
}
```

运行核对

```
true
true
false
钻石 64/64
true
false
钻石 0/64
64
```

Chest 是与 SecureChest 并列的独立类，放在同一个文件中，只有 SecureChest 是 public 类。字段是 private，操作和查询方法明确声明 public。这样调用者只能通过公开入口改变库存。

13 拓展任务与单元复盘

第17至20课完成了面向对象第一阶段：蓝图、引用、出生和封装。作业要同时提交接口表与测试表。

三档任务继续加固对象边界

木镐

Mob生命改为private

铁镐

维护0到20饥饿值

钻石镐

设计有状态的密码箱

复盘

封装是约束代码访问

三档作业

- 木镐：把 Mob 的 health 改为 private，提供 takeDamage、heal 和 getHealth。
- 铁镐：设计 FoodBar，饥饿值始终保持0到20，非法操作返回false。
- 钻石镐：设计三次输错会锁定的密码箱，不允许外部直接修改密码和锁定状态。

验收表

自评	能	还要练
我会使用private	☐	☐
我会设计行为方法	☐	☐
我会验证失败不改状态	☐	☐

下一课：第21课会研究 static 成员。我们将分清属于单个对象的数据和属于整个类的共享规则。

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>