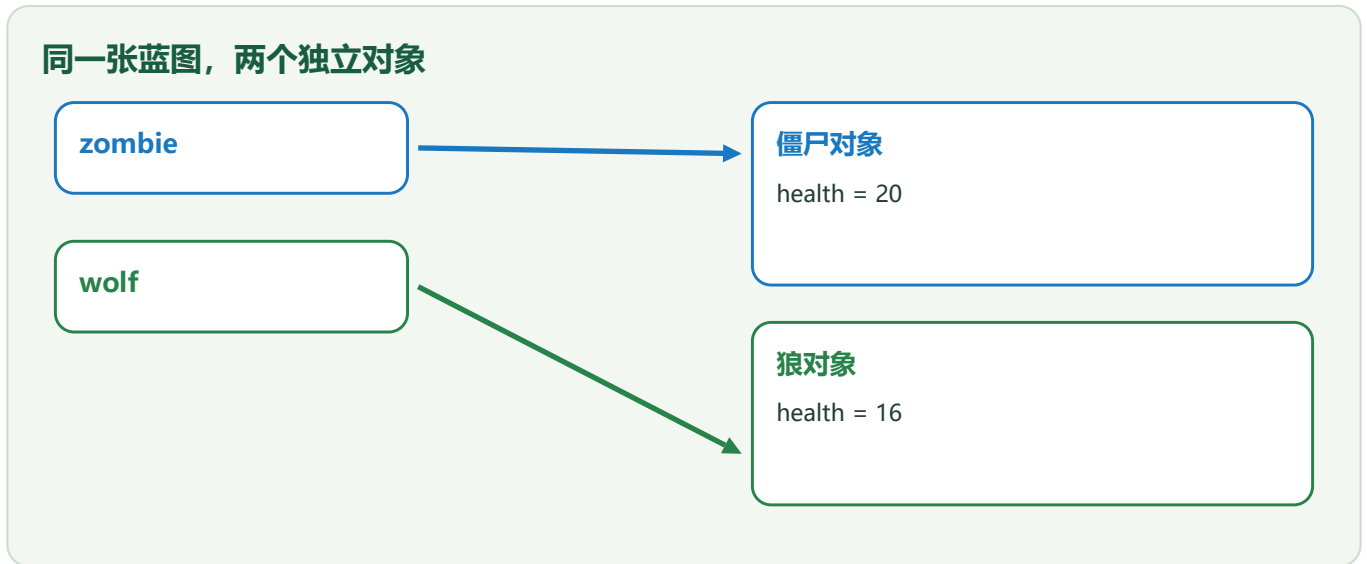


第17课

用 class 设计生物蓝图

类 字段与实例方法



会区分类和对象；能用字段保存每只生物的状态；能写实例方法让对象行动；能创建多个互不干扰的 Mob 并完成竞技场测试。

前置背包：变量、方法、参数、返回值和作用域。今天把散落的数据和动作搬进同一张蓝图。

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 蓝图不是生物本身

如果每只僵尸都用一组散落变量表示，代码会像村庄袭击一样乱。类把相关数据和动作放进一个新类型。

class 像工作台配方 对象才是成品

类

定义共同结构

new

创建具体对象

变量

保存对象引用

状态

实例字段分别保存

L17Example01.java • 完整独立示例

```
class Wolf { String name; int health; }
public class L17Example01 {
    public static void main(String[] args) {
        Wolf wolf = new Wolf();
        wolf.name = "小灰";
        wolf.health = 16;
        System.out.println(wolf.name);
    }
}
```

运行核对

小灰

四个问题

- 类定义了什么共同结构？
- new 创建了什么？
- 对象变量保存的是什么？
- 两只对象的字段会不会自动共享？

读法：class Wolf 是蓝图；new Wolf() 创建一只具体狼；wolf 是找到这只狼的变量。

02 类的四块结构

类名通常使用大驼峰命名。字段写在类体内、方法外，实例方法可以直接访问同一个对象的字段。

类体把字段和方法围在花括号里

class Mob

声明新类型

字段

名字与生命

实例方法

操作当前对象

花括号

类体容纳成员

L17Example02.java • 完整独立示例

```
class Mob {
    String name;
    int health;
    void describe() { System.out.println(name + " 生命=" + health); }
}

public class L17Example02 {
    public static void main(String[] args) {
        Mob mob = new Mob();
        mob.name = "小僵";
        mob.health = 20;
        mob.describe();
    }
}
```

运行核对

小僵 生命=20

位置地图：字段属于对象状态，不要写进某个方法里；方法属于类，不要在另一个方法里面声明。

03 字段保存对象状态

字段描述对象现在是什么样。使用点号选择具体对象的字段。

每个对象都有自己的状态箱

僵尸

自己的名字与生命

狼

另一份名字与生命

同一蓝图

相同的字段结构

两次new

得到独立对象

L17Example03.java • 完整独立示例

```
class Mob { String name; int health; }
public class L17Example03 {
    public static void main(String[] args) {
        Mob zombie = new Mob();
        zombie.name = "小僵"; zombie.health = 20;
        Mob wolf = new Mob();
        wolf.name = "小灰"; wolf.health = 16;
        System.out.println(zombie.health + wolf.health);
    }
}
```

运行核对

36

独立背包：修改 `wolf.health` 不会改变 `zombie.health`。它们共享蓝图，不共享字段数据。

04 实例方法操作相关状态

把生物规则放进生物类，调用处就不必到处手动修改 health。

对象让自己的方法修改自己的字段

takeDamage

只接受正伤害

health

扣血但不低于0

实例调用

点号左边选择对象

下一步

封装限制直接修改

L17Example04.java • 完整独立示例

```
class Mob {
    String name;
    int health;
    void takeDamage(int n) {
        if (n > 0) health = Math.max(0, health - n);
    }
    int heal(int n) {
        if (n <= 0) return 0;
        int gained = Math.min(n, 20 - health);
        health += gained;
        return gained;
    }
    boolean isAlive() { return health > 0; }
    void describe() { System.out.println(name + " 生命=" + health); }
}

public class L17Example04 {
    public static void main(String[] args) {
        Mob zombie = new Mob(); zombie.health = 20;
        Mob wolf = new Mob(); wolf.health = 16;
        zombie.takeDamage(7);
        wolf.takeDamage(3);
        System.out.println(zombie.health + "," + wolf.health);
    }
}
```

运行核对

13,13

责任归位：health 的合法规则由 Mob 方法维护。调用者只说“受到7点伤害”，不负责手动夹到0。

05 多个对象互不干扰

创建三个 Mob，再分别受伤和治疗。观察同一方法在不同对象上产生不同结果。

同一蓝图生成一支对象军团

对象a

受到5点伤害

对象b

没有受到伤害

对象c

生命降到0

结论

一次调用只操作目标对象

L17Example05.java • 完整独立示例

```
class Mob {
    String name;
    int health;
    void takeDamage(int n) {
        if (n > 0) health = Math.max(0, health - n);
    }
    int heal(int n) {
        if (n <= 0) return 0;
        int gained = Math.min(n, 20 - health);
        health += gained;
        return gained;
    }
    boolean isAlive() { return health > 0; }
    void describe() { System.out.println(name + " 生命=" + health); }
}

public class L17Example05 {
    public static void main(String[] args) {
        Mob a = new Mob(); a.health = 20;
        Mob b = new Mob(); b.health = 20;
        Mob c = new Mob(); c.health = 16;
        a.takeDamage(5);
        c.takeDamage(20);
        System.out.println(a.health + "," + b.health + "," + c.health);
    }
}
```

运行核对

15,20,0

蓝图复用：类只写一次，new 可以执行多次。程序得到很多结构一致、状态独立的对象。

06 方法参数与返回值

实例方法和 static 方法一样可以有参数与返回值。区别是它还自动连接到当前对象。

实例方法也能收材料并交回结果

heal参数

请求增加量

实际增加

不超过剩余空间

返回值

告诉调用者实际增加量

isAlive

只回答当前是否存活

这里明确把生命上限设为20。先算还能增加多少，再相加，避免 `health + n` 溢出。第20课会让字段只能经过方法修改；目前直接写字段仍可能破坏规则。

查询方法：isAlive 只读取状态并返回答案；takeDamage 会修改状态。名字应让读者看出差别。

06 方法参数与返回值（续）

L17Example06.java • 完整独立示例

```
class Mob {
    String name;
    int health;
    void takeDamage(int n) {
        if (n > 0) health = Math.max(0, health - n);
    }
    int heal(int n) {
        if (n <= 0) return 0;
        int gained = Math.min(n, 20 - health);
        health += gained;
        return gained;
    }
    boolean isAlive() { return health > 0; }
    void describe() { System.out.println(name + " 生命=" + health); }
}

public class L17Example06 {
    public static void main(String[] args) {
        Mob wolf = new Mob(); wolf.health = 16;
        System.out.println(wolf.heal(10));
        System.out.println(wolf.health);
        System.out.println(wolf.isAlive());
    }
}
```

运行核对

```
4
20
true
```

07 设计 Mob 的责任地图

好类不是把所有代码塞进新花括号。字段和方法应围绕同一种对象的责任。

状态和行为要围绕同一个对象

name

保存对象名字

health

保存当前生命

规则方法

受伤、治疗与查询

边界

不让单只生物管理全世界

验收表

成员	类型	责任
name	String 字段	保存名字
health	int 字段	保存当前生命
takeDamage	实例方法	扣血并守住下限
heal	实例方法	回血并守住上限
isAlive	实例方法	回答是否存活

别越界：天气、世界时间和所有玩家列表不属于单只 Mob。别让一只鸡负责太阳下山。

马老师教编程
www.teacher-ma.de

10 / 16

08 类与对象故障诊所

最常见错误是把类名当对象、在 static main 中直接访问实例字段，或声明变量后忘记创建对象。

static 和实例是两条不同轨道

类名访问

实例字段需要对象

只声明

局部引用仍未初始化

错误static

可能意外共享状态

检查

从new与点号左边开始

故障实验：实例字段需要具体对象，不能用类名直接读写。下面故意保留错误，请先预测，再尝试修复。

BadInstance.java • 故意错误示例

```
class Mob { int health; }
public class BadInstance {
    public static void main(String[] args) { Mob.health = 10; }
}
```

故障实验：局部变量 mob 未初始化；声明引用没有创建对象。下面故意保留错误，请先预测，再尝试修复。

BadObjectInit.java • 故意错误示例

```
class Mob { int health; }
public class BadObjectInit {
    public static void main(String[] args) {
        Mob mob;
        mob.health = 20;
    }
}
```

急诊顺序：先找 new；再看点号左边是谁；最后确认字段或方法是否错误写成 static。

09 挑战 蓝图 Mob 竞技场

挑战 程序叫 MobArena。Mob 保存名字、生命和攻击力；attack 让目标受伤，describe 显示当前状态。

两只生物用自己的方法完成战斗

创建

两只独立生物

攻击

调用目标受伤方法

存活检查

倒下者不再参与

测试

伤害恰好等于生命

验收条件

- 创建僵尸和铁傀儡两个对象。
- 双方轮流攻击，生命不低于0。
- 攻击已倒下目标时不继续扣血。
- 输出能看出两个对象状态互不干扰。

验收表

测试	操作	预期
正常	僵尸攻击傀儡	傀儡生命减少
边界	伤害等于剩余生命	生命变0
错误	负数攻击力	按0处理
状态	修改一只生命	另一只不变

最小版本：先完成 describe 与 takeDamage，再写 attack。方法越少，第一轮调试越安静。

10 挑战 实现 Mob 竞技场

新建 MobArena.java。内部 Mob 类用于本课演示，运行后检查每轮战斗输出。

类把状态和行为收进同一张蓝图

状态

名字、生命、攻击力

行为

受伤、攻击、描述

main

安排战斗顺序

结果

两只对象生命分别变化

验收：解释为什么 zombie 和 golem 的 health 独立；把攻击力改为负数测试；再增加一只蜘蛛。

MobArena.java • 第1/2段，顺序接在同一文件

```
public class MobArena {
    static class Mob {
        String name;
        int health;
        int attackPower;

        void takeDamage(int amount) {
            health = Math.max(0, health - Math.max(0, amount));
        }
        void attack(Mob target) {
            if (target != null && isAlive() && target.isAlive()) target.takeDamage(attackPower);
        }
        boolean isAlive() { return health > 0; }
        void describe() {
            System.out.println(name + " 生命=" + health);
        }
    }

    static Mob create(String name, int health, int power) {
        Mob mob = new Mob();
        mob.name = name;
        mob.health = Math.max(0, health);
        mob.attackPower = Math.max(0, power);
        return mob;
    }

    public static void main(String[] args) {
        Mob zombie = create("小僵", 20, 5);
```

10 挑战 实现 Mob 竞技场 (续)

MobArena.java • 第2/2段, 顺序接在同一文件

```
        Mob golem = create("铁傀儡", 30, 12);
        zombie.attack(golem);
        golem.attack(zombie);
        zombie.describe();
        golem.describe();
        golem.attack(zombie);
        zombie.describe();
        golem.describe();
    }
}
```

运行核对

小僵尸 生命=8
铁傀儡 生命=25
小僵尸 生命=0
铁傀儡 生命=25

实现把辅助 Mob 写成 static 嵌套类, 让这一文件可直接运行。static 只修饰嵌套类声明; health 没有 static, 仍然由每只对象独立保存。练习攻击力与生命均为自定义数值。

11 拓展任务与通关复盘

作业需提交类图草稿、源代码、运行结果和一个故障记录。

三档任务扩建生物设计院

木镐

设计两只羊

铁镐

双对象轮流攻击

钻石镐

宠物信任与行为

复盘

类、对象与实例字段

三档作业

- 木镐：设计 Sheep 类，包含颜色、生命和叫声方法，创建两只羊。
- 铁镐：让两个 Mob 互相攻击，记录每轮生命。
- 钻石镐：设计可驯服 Pet 类，包含信任值、喂食和跟随行为。

验收表

自评	能	还要练
我能区分类与对象	☐	☐
我会写实例字段	☐	☐
我会让方法维护状态	☐	☐

下一课：第18课研究对象变量真正保存的引用。两张变量纸条可能指向同一只宠物，也可能谁都没指向。

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>