

第23课

行为契约

抽象类打骨架，接口约定能力

共同状态已经写好 差异行为交给子类

Actor

抽象共同骨架

name / label

已完成的部分

role()

留给子类回答

蝙蝠和鞘翅玩家并非同一家族，却都能飞。用接口约定“会什么”，用抽象类保存共同骨架，让能力合同小而清楚。

前置背包：继承、多态和方法重写。今天不按长相组队，而是按能力签合同。

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 抽象类留下必须完成的空格

若所有角色都有名字，却没有统一的 `role` 实现，父类就不该假装给出一个无意义答案。抽象类允许写好共同部分，并明确留下未完成的方法。

共同状态已经写好 差异行为交给子类

Actor

抽象共同骨架

name / label

已完成的部分

role()

留给子类回答

`new Actor()` 会编译失败，因为抽象方法还没有实现。必须创建完成合同的具体子类。

故障实验：抽象类 `Actor` 不能直接 `new`；它还留下 `role` 的实现任务。下面故意保留错误，请先预测，再尝试修复。

BadAbstract.java • 故意错误示例

```
abstract class Actor {
    protected final String name;
    Actor(String name) { this.name = name; }
    abstract String role();
    public String label() { return role() + " " + name; }
}
public class BadAbstract {
    public static void main(String[] args) { Actor a = new Actor("未完成"); }
}
```

02 抽象方法强制子类回答

抽象方法像试卷上的必答题。子类若仍是具体类，就必须给出正确签名的方法体。

没有方法体 但名字参数和返回类型完整

具体子类

必须完成抽象方法

@Override

核对方签名

Bat

提供蝙蝠角色

返回类型、方法名和参数必须匹配。保留 @Override，让编译器替你核对答题卡。

L23Example02.java • 完整独立示例

```
abstract class Actor {
    protected final String name;
    Actor(String name) { this.name = name; }
    abstract String role();
    public String label() { return role() + " " + name; }
}
class Bat extends Actor {
    Bat(String name) { super(name); }
    @Override String role() { return "蝙蝠"; }
}
public class L23Example02 {
    public static void main(String[] args) {
        Actor a = new Bat("夜影");
        System.out.println(a.label());
    }
}
```

运行核对

蝙蝠 夜影

故障实验：非抽象 Bat 未完成 role()，编译器拒绝它。下面故意保留错误，请先预测，再尝试修复。

02 抽象方法强制子类回答（续）

BadRole.java • 故意错误示例

```
abstract class Actor {  
    protected final String name;  
    Actor(String name) { this.name = name; }  
    abstract String role();  
    public String label() { return role() + " " + name; }  
}  
class Bat extends Actor { Bat(String name) { super(name); } }  
public class BadRole { public static void main(String[] args) { } }
```

03 接口只描述会不会

接口适合约定跨家族能力。蝙蝠是动物，鞘翅玩家是玩家，但两者都能实现 Flyable；调用者只需知道对方会飞。

不要求同一家族 只要求相同能力

Flyable

约定飞行能力

Bat

实现飞行

ElytraPlayer

也实现飞行

extends 回答“是什么”，implements 回答“会什么”。两个问题不要混在一条继承链里。

L23Example03.java • 完整独立示例

```
interface Flyable { String fly(); }
class Bat implements Flyable {
    @Override public String fly() { return "蝙蝠飞行"; }
}
class ElytraPlayer implements Flyable {
    @Override public String fly() { return "玩家滑翔"; }
}
public class L23Example03 {
    public static void main(String[] args) {
        Flyable[] team = {new Bat(), new ElytraPlayer()};
        for (Flyable member : team) System.out.println(member.fly());
    }
}
```

运行核对

蝙蝠飞行
玩家滑翔

本章介绍的是接口的抽象实例方法。接口还可以声明 default、static、private 方法；“只描述合同”是一种设计用途，不是 Java 禁止方法体。

04 实现方法必须公开

接口方法默认是 public。实现类必须用 public 提供同样可见的入口。

接口承诺是公开入口

接口抽象方法

默认 public

实现方法

不能缩小可见性

@Override

核对同一合同

本节接口的抽象实例方法默认是 public；实现方法也必须公开。接口可以包含 default、static 与 private 方法，不能把这些方法一概说成抽象方法。

故障实验：接口 fly() 是 public；实现方法不能缩小可见性。下面故意保留错误，请先预测，再尝试修复。

BadPublic.java • 故意错误示例

```
interface Flyable { String fly(); }  
class Bat implements Flyable { String fly() { return "飞行"; } }  
public class BadPublic { public static void main(String[] args) { } }
```

在 String fly() 前补上 public，并加 @Override。接口字段是常量，不能替代各个对象自己的可变状态。

05 一个类可以签多份合同

Java 类只能直接继承一个父类，却可以实现多个接口。一个飞行炸弹可以同时遵守 Flyable 和 Explodable。

继承只有一个父类 接口可以有多个

一个父类

类只能直接继承一个

多个接口

能力可以组合

FlyingMine

同时会飞和引爆

接口像可组合的能力卡。类只领取自己真正能完成的合同，不必为了多继承制造复杂家谱。

L23Example05.java • 完整独立示例

```
interface Flyable { String fly(); }
interface Explodable { String explode(); }
class FlyingMine implements Flyable, Explodable {
    @Override public String fly() { return "飞向目标"; }
    @Override public String explode() { return "引爆"; }
}
public class L23Example05 {
    public static void main(String[] args) {
        FlyingMine mine = new FlyingMine();
        System.out.println(mine.fly());
        System.out.println(mine.explode());
    }
}
```

运行核对

飞向目标
引爆

06 接口引用调用能力

方法参数写接口类型，就能接收任何实现者。这是多态从家族关系扩展到能力关系。

工具方法只依赖所需合同

接口参数

只要所需能力

任意实现者

可接受相同触发

null

本课约定安全跳过

只需要 fly 的方法就接收 Flyable，不必接收更庞大的 Actor 或具体 Bat。

L23Example06.java • 第1/2段，顺序接在同一文件

```
interface Flyable { String fly(); }
interface Tradable { String trade(String item); }
interface Explodable { int blastPower(); String explode(); }
abstract class Actor {
    protected final String name;
    Actor(String name) { this.name = name; }
    abstract String role();
    public String label() { return role() + " " + name; }
}
class Bat extends Actor implements Flyable {
    Bat(String name) { super(name); }
    @Override String role() { return "蝙蝠"; }
    @Override public String fly() { return name + " 飞向高处"; }
}
class Villager extends Actor implements Tradable {
    Villager(String name) { super(name); }
    @Override String role() { return "村民"; }
    @Override public String trade(String item) {
        return item == null || item.isBlank()
            ? name + " 拒绝空交易" : name + " 收下" + item.strip();
    }
}
class Creeper extends Actor implements Explodable {
    Creeper(String name) { super(name); }
    @Override String role() { return "苦力怕"; }
    @Override public int blastPower() { return 3; }
    @Override public String explode() { return name + " 爆炸"; }
}
```

06 接口引用调用能力（续）

L23Example06.java • 第2/2段，顺序接在同一文件

```
public class L23Example06 {
    static void triggerFlight(Flyable target) {
        if (target != null) System.out.println(target.fly());
    }
    static void triggerTrade(Tradable target, String item) {
        if (target != null) System.out.println(target.trade(item));
    }
    static void triggerExplosion(Explodable target) {
        if (target != null)
            System.out.println(target.explode() + " 强度=" + target.blastPower());
    }
    public static void main(String[] args) {
        triggerFlight(new Bat("夜影"));
        triggerTrade(new Villager("村长"), "绿宝石");
        triggerExplosion(new Creeper("绿豆"));
        triggerFlight(null);
    }
}
```

运行核对

夜影 飞向高处
村长 收下绿宝石
绿豆 爆炸 强度=3

07 小接口比万能杂物袋清楚

把飞行、交易、爆炸塞进一个 MegaAbility，会迫使村民实现不会的 fly。小接口更容易理解和组合。

每份合同围绕一个稳定能力

Flyable

只负责飞行

Tradable

只负责交易

Explodable

只负责爆炸

问每个方法是否总被同一批类使用。若答案不是，就把合同拆成更小的能力接口。

07 小接口比万能杂物袋清楚（续）

L23Example07.java • 完整独立示例

```
interface Flyable { String fly(); }
interface Tradable { String trade(String item); }
interface Explodable { int blastPower(); String explode(); }
abstract class Actor {
    protected final String name;
    Actor(String name) { this.name = name; }
    abstract String role();
    public String label() { return role() + " " + name; }
}
class Bat extends Actor implements Flyable {
    Bat(String name) { super(name); }
    @Override String role() { return "蝙蝠"; }
    @Override public String fly() { return name + " 飞向高处"; }
}
class Villager extends Actor implements Tradable {
    Villager(String name) { super(name); }
    @Override String role() { return "村民"; }
    @Override public String trade(String item) {
        return item == null || item.isBlank()
            ? name + " 拒绝空交易" : name + " 收下" + item.strip();
    }
}
class Creeper extends Actor implements Explodable {
    Creeper(String name) { super(name); }
    @Override String role() { return "苦力怕"; }
    @Override public int blastPower() { return 3; }
    @Override public String explode() { return name + " 爆炸"; }
}
public class L23Example07 {
    public static void main(String[] args) {
        Tradable merchant = new Villager("村长");
        System.out.println(merchant.trade(""));
    }
}
```

运行核对

村长 拒绝空交易

村民只签 Tradable；不需要假装会飞。方法名称相同并不自动成为接口实现，还要 implements 并满足签名。

08 抽象类与接口比较

抽象类可保存实例状态、构造器和共同实现；接口强调调用者可依赖的能力。一个类常常继承抽象父类，同时实现若干接口。

家族骨架和能力合同解决不同问题

抽象类

状态、构造器、共同实现

接口

调用者依赖的能力

组合使用

继承骨架并实现接口

Bat extends Actor implements Flyable 同时回答它是什么和它会什么。

验收对照表

问题	抽象类	接口
表达	同一家族骨架	跨家族能力
实例字段	可以	不保存普通实例状态
构造器	可以	没有
数量	只能继承一个	可实现多个
典型例子	Actor	Flyable

09 合同测试与故障诊所

接口错误常在编译阶段暴露，行为错误仍要通过测试发现。

编译器检查签名 运行测试检查行为

签名匹配

参数不能遗漏

空交易

明确返回拒绝

爆炸强度

本课约定3

先看是否遗漏 public，再核对方法签名和 @Override，最后运行每份合同的行为测试。

故障实验：trade() 少了 String 参数，不能完成 trade(String) 合同。下面故意保留错误，请先预测，再尝试修复。

BadTrade.java • 故意错误示例

```
interface Tradable { String trade(String item); }
class Villager implements Tradable {
    public String trade() { return "交易"; }
}
public class BadTrade { public static void main(String[] args) { } }
```

验收对照表

测试	输入	预期
飞行	Bat	返回飞行描述
交易	空物品名	返回拒绝提示
爆炸	Creeper	强度为3
空引用	null	安全提示或跳过

10 挑战 蓝图三种能力

挑战 程序叫 AbilityContracts。Bat 会飞，Villager 会交易，Creeper 会爆炸；三者都继承 Actor 并实现各自能力。

抽象角色提供名字 三份接口提供能力入口

Bat

飞行合同

Villager

交易合同

Creeper

爆炸合同

先声明三个接口和 Actor，再逐个实现具体类，最后写触发方法。每签完一份合同就编译一次。

验收条件

- Actor 不能直接 new。
- 三个具体类都实现 role。
- 接口实现方法全部 public。
- 触发方法使用接口参数。
- 交易空字符串时返回拒绝提示。

11 挑战 实现能力合同

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

飞行 交易 爆炸通过接口统一触发

先声明

三个接口与 Actor

再实现

逐个完成具体类

后触发

接口参数统一调用

让 Bat 再实现第二接口；去掉 public 观察编译错误；确认触发方法不依赖具体类。

AbilityContracts.java • 第1/2段, 顺序接在同一文件

```
interface Flyable { String fly(); }
interface Tradable { String trade(String item); }
interface Explodable { int blastPower(); String explode(); }
abstract class Actor {
    protected final String name;
    Actor(String name) { this.name = name; }
    abstract String role();
    public String label() { return role() + " " + name; }
}
class Bat extends Actor implements Flyable {
    Bat(String name) { super(name); }
    @Override String role() { return "蝙蝠"; }
    @Override public String fly() { return name + " 飞向高处"; }
}
class Villager extends Actor implements Tradable {
    Villager(String name) { super(name); }
    @Override String role() { return "村民"; }
    @Override public String trade(String item) {
        return item == null || item.isBlank()
            ? name + " 拒绝空交易" : name + " 收下" + item.strip();
    }
}
class Creeper extends Actor implements Explodable {
    Creeper(String name) { super(name); }
    @Override String role() { return "苦力怕"; }
    @Override public int blastPower() { return 3; }
    @Override public String explode() { return name + " 爆炸"; }
}
```

11 挑战 实现能力合同（续）

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

```
public class AbilityContracts {
    static void triggerFlight(Flyable target) {
        if (target != null) System.out.println(target.fly());
    }
    static void triggerTrade(Tradable target, String item) {
        if (target != null) System.out.println(target.trade(item));
    }
    static void triggerExplosion(Explodable target) {
        if (target != null)
            System.out.println(target.explode() + " 强度=" + target.blastPower());
    }

    public static void main(String[] args) {
        Bat bat = new Bat("夜影");
        Villager villager = new Villager("村长");
        Creeper creeper = new Creeper("绿豆");
        System.out.println(bat.label());
        triggerFlight(bat);
        triggerTrade(villager, "绿宝石");
        triggerTrade(villager, " ");
        triggerExplosion(creeper);
        triggerFlight(null);
    }
}
```

运行核对

蝙蝠 夜影
夜影 飞向高处
村长 收下绿宝石
村长 拒绝空交易
绿豆 爆炸 强度=3

12 拓展任务与通关复盘

每份接口都要附上一句合同说明，写清输入、返回值和失败处理。

三档任务扩展能力合同

合同说明

写清输入与输出

失败规则

明确拒绝或跳过

编译与行为

两种检查都需要

第24课把方向限制为 enum，把坐标装进 record。移动不会偷偷改旧坐标，而是返回一个新位置。

三档作业

- 木镐：设计 Swimmable，让海豚和玩家都能实现 swim。
- 铁镐：让一个类同时实现 Flyable 与 Tradable，并用两个接口引用调用。
- 钻石镐：设计红石设备接口，分别实现门、灯和陷阱，并写统一触发器。

验收对照表

自评	能	还要练
我能区分抽象类和接口	☐	☐
我会使用 implements	☐	☐
我会设计小接口	☐	☐

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>