

第19课

生成点 构造方法

构造器 this 与初始化

三个入口，一条初始化路线

默认出生

`this("Steve")`

指定名字

`this(name, 20)`

完整构造器

校验并写入字段

会写与类同名的构造器；能用 `this` 区分字段和参数；能在对象出生时完成校验；能用构造器重载提供三种合法出生方式。

前置背包：类、字段、实例方法、参数、重载和对象引用。今天要让每个对象从出生第一秒就处于合法状态。

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 构造器负责出生体检

先 new 再逐个写字段，容易漏掉名字或留下负生命。构造器把必要数据放进创建过程。

new 调用构造器创建并初始化对象

new

选择匹配的构造器

构造器

初始化新对象

字段

保存出生状态

验证

保证创建完成时合法

L19Example01.java • 完整独立示例

```
class Player {  
    String name;  
    int health;  
    Player(String name, int health) {  
        this.name = name == null || name.isBlank() ? "Steve" : name.trim();  
        this.health = Math.max(1, Math.min(20, health));  
    }  
    Player(String name) { this(name, 20); }  
    Player() { this("Steve"); }  
}  
  
public class L19Example01 {  
    public static void main(String[] args) {  
        Player alex = new Player("Alex", 20);  
        System.out.println(alex.name + ", " + alex.health);  
    }  
}
```

运行核对

Alex,20

识别法：构造器与类同名，没有返回类型。写成 void Player 会变成普通方法，不会在 new 时自动调用。

02 构造器语法地图

构造器写在类体中。new 后面的实参必须匹配某个构造器参数列表。

类名 参数和构造器体缺一不可

类名

构造器必须同名

参数列表

必须与实参匹配

返回类型

构造器没有返回类型

无参

显式声明构造器后不会自动补无参

L19Example02.java • 完整独立示例

```
class Mob {  
    String name; int health;  
    Mob(String name, int health) {  
        this.name = name; this.health = health;  
    }  
}  
  
public class L19Example02 {  
    public static void main(String[] args) {  
        Mob zombie = new Mob("小僵", 20);  
        System.out.println(zombie.name);  
    }  
}
```

运行核对

小僵

自动变化：一旦声明了带参数构造器，Java 不再自动提供无参构造器。new Mob() 会找不到入口。

03 this 指向当前对象

构造器常让参数名与字段名一致。此时短名字表示参数，this.字段 表示正在创建的对象字段。

参数名遮住字段名时 this 打开对象箱子

this.name

当前对象字段

name

本次传入的参数

正确赋值

字段接收参数值

name=name

参数赋给自己，字段没变

L19Example03.java • 完整独立示例

```
class Player {  
    String name;  
    Player(String name) { name = name; }  
}  
public class L19Example03 {  
    public static void main(String[] args) {  
        Player p = new Player("Alex");  
        System.out.println(p.name);  
    }  
}
```

运行核对

null

这里 `name = name` 两边都指参数，字段仍为 `null`。改成 `this.name = name` 后，输出应是 `Alex`；编译成功也可能藏着逻辑错误。

读等号：等号右边的 `name` 是送来的材料；左边的 `this.name` 是对象里的格子。

04 出生时完成校验

构造器是守住对象合法状态的第一道门。先定义规则，再选择修正值或抛出异常。初学阶段可使用安全默认值。

非法材料在构造器入口被修正或拒绝

坏名字

替换为Steve

有效名字

去除两端空白

生命

修正到1到20

策略

所有入口保持一致

L19Example04.java • 完整独立示例

```
class Player {
    String name;
    int health;
    Player(String name, int health) {
        this.name = name == null || name.isBlank() ? "Steve" : name.trim();
        this.health = Math.max(1, Math.min(20, health));
    }
    Player(String name) { this(name, 20); }
    Player() { this("Steve"); }
}

public class L19Example04 {
    public static void main(String[] args) {
        Player p = new Player("", -8);
        System.out.println(p.name + ", " + p.health);
    }
}
```

运行核对

Steve,1

先定策略：校验失败可以修正、拒绝或报告。一个类应使用一致策略，不要有时偷偷改、有时突然崩溃。

05 构造器重载

构造器可以重载，为常用创建方式提供简洁入口。参数列表必须不同。

构造器委托：最终只保留一份校验



L19Example05.java • 完整独立示例

```
class Player {
    String name;
    int health;
    Player(String name, int health) {
        this.name = name == null || name.isBlank() ? "Steve" : name.trim();
        this.health = Math.max(1, Math.min(20, health));
    }
    Player(String name) { this(name, 20); }
    Player() { this("Steve"); }
}

public class L19Example05 {
    public static void main(String[] args) {
        Player a = new Player("Alex", 12);
        Player b = new Player("Builder");
        Player c = new Player();
        System.out.println(a.name + "," + a.health);
        System.out.println(b.name + "," + b.health);
        System.out.println(c.name + "," + c.health);
    }
}
```

运行核对

Alex,12
Builder,20
Steve,20

不是三个类：三个构造器都创建 Player，只是出生时提供的材料不同。

06 this 构造器链

this(...) 调用本类另一个构造器，必须是构造器第一条语句。这样默认值和校验只维护一份。

构造器委托：最终只保留一份校验



故障实验：在本课程使用的 Java 21 中，this(...) 必须是构造器第一条语句。下面故意保留错误，请先预测，再尝试修复。

BadConstructorOrder.java • 故意错误示例

```
class Player {  
    Player(String name) { }  
    Player() {  
        System.out.println("出生");  
        this("Steve");  
    }  
}  
public class BadConstructorOrder {  
    public static void main(String[] args) { }  
}
```

单一路线：让所有重载最终走向最完整构造器。修一次校验，三个生成点一起变安全。

07 合法状态与测试

设计中希望一直保持的规则称为不变条件。本课先保证 Player 刚创建完成时，名字不能空，出生时生命必须在1到20之间。

对象一出生就应满足不可破坏的规则

名字
创建后非空

生命
出生时1到20

坐标
项目内-100到100

长期保证
还需封装防止任意修改

验收表

测试	创建方式	预期
完整	<code>new Player("Alex", 12)</code>	Alex 12
默认生命	<code>new Player("Alex")</code>	Alex 20
全默认	<code>new Player()</code>	Steve 20
坏名字	<code>new Player("", 8)</code>	Steve 8
坏生命	<code>new Player("Alex", 99)</code>	Alex 20

边界值：生命测试至少包含0、1、20、21。边界两侧比只测10更容易抓到错误。

08 构造器故障诊所

读到 “constructor cannot be applied” 时，比较 new 的实参数量和类型，再查看构造器声明。

编译错误通常来自名字 参数或 this 顺序

void类名

普通方法，不是构造器

多实参

没有匹配入口

this顺序

Java21中必须首句

漏字段

可能留下默认值

故障实验：没有与三个实参匹配的构造器。下面故意保留错误，请先预测，再尝试修复。

void Player 是普通方法。这里仍有自动提供的无参构造器，所以 new Player() 能执行，却不会自动调用这个普通方法。

急诊顺序：核对类名；确认没有返回类型；匹配参数；检查 this(...) 是否第一句；最后验证每个字段都完成初始化。

BadConstructorCall.java • 故意错误示例

```
class Player {  
    Player(String name) { }  
}  
public class BadConstructorCall {  
    public static void main(String[] args) {  
        Player p = new Player("Alex", 20, 3);  
    }  
}
```

08 构造器故障诊所（续）

L19Example08B.java • 完整独立示例

```
class Player {  
    String name;  
    void Player(String name) { this.name = name; }  
}  
public class L19Example08B {  
    public static void main(String[] args) {  
        Player p = new Player();  
        System.out.println(p.name);  
    }  
}
```

运行核对

null

09 挑战 实现 三种合法出生

挑战 程序叫 PlayerSpawn。三个构造器分别支持完整出生、指定名字和完全默认。

构造器链把所有出生方式送进同一套校验

完整玩家

名字、生命与坐标

指定名字

其余使用默认值

默认玩家

Steve与默认状态

坏材料

按统一策略修正

验收：解释四次 new 分别调用哪个构造器；确认坏材料被修正；再增加边界坐标-100和100。

09 挑战 实现 三种合法出生（续）

PlayerSpawn.java • 完整独立示例

```
public class PlayerSpawn {
    static class Player {
        String name;
        int health;
        int x;
        int z;

        Player(String name, int health, int x, int z) {
            this.name = name == null || name.isBlank() ? "Steve" : name.trim();
            this.health = Math.max(1, Math.min(20, health));
            this.x = Math.max(-100, Math.min(100, x));
            this.z = Math.max(-100, Math.min(100, z));
        }
        Player(String name) {
            this(name, 20, 0, 0);
        }
        Player() {
            this("Steve");
        }
        void describe() {
            System.out.println(name + " 生命=" + health
                + " 出生点=(" + x + ", " + z + ")");
        }
    }
    public static void main(String[] args) {
        Player a = new Player("Alex", 18, 12, -7);
        Player b = new Player("Builder");
        Player c = new Player();
        Player d = new Player("", -5, 999, -999);
        a.describe(); b.describe(); c.describe(); d.describe();
    }
}
```

运行核对

Alex 生命=18 出生点=(12,-7)
 Builder 生命=20 出生点=(0,0)
 Steve 生命=20 出生点=(0,0)
 Steve 生命=1 出生点=(100,-100)

规则只保证构造完成时的出生状态。目前字段仍可被外部代码修改；要让对象整个使用过程中都守住规则，还需要下一课的封装。出生时生命1到20是本项目约定；受伤后的对象可以另设0表示倒下。

10 拓展任务与通关复盘

作业需要为每个构造器写至少两个测试，并说明对象的不变条件。

三档任务扩建生成点

木镐

Mob增加构造器

铁镐

重载与生命边界

钻石镐

限制随机出生区域

复盘

出生规则与this

三档作业

- 木镐：给第17课 Mob 加构造器，创建时必须提供名字和生命。
- 铁镐：增加重载构造器与参数校验，生命保持0到20。
- 钻石镐：设计随机出生坐标，但保证不落在指定危险范围。

验收表

自评	能	还要练
我能识别构造器	☐	☐
我会使用this	☐	☐
我会测试出生规则	☐	☐

下一课：第20课给对象加锁。private 会保护字段，公开方法则像箱子按钮一样提供受控操作。

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>