

第24课

坐标 方向与枚举

把合法方向和坐标规则写进类型

任意文字都能混进方向变量

字符串

可能拼错和大小写不一致

枚举类型

圈定合法常量

null

仍需专门处理

拼错一次方向，就可能让导航失灵。enum 把答案圈定，record 把坐标保存为值；移动返回新位置，边界由你明文规定。

前置背包：类、构造器、方法和接口。今天要给坐标装上防误改护甲，也要把拼错方向的入口关掉。

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 字符串方向容易长出新品种

用 String 保存方向时，大小写、空格和拼写都可能制造非法状态。enum 把合法答案固定在一个有限集合里。

任意文字都能混进方向变量

字符串

可能拼错和大小写不一致

枚举类型

圈定合法常量

null

仍需专门处理

当取值只有固定几种时，优先考虑 enum。编译器能在运行前挡住不存在的枚举常量。

L24Example01.java • 完整独立示例

```
public class L24Example01 {  
    public static void main(String[] args) {  
        String facing = "dongg";  
        System.out.println(facing);  
        System.out.println(facing.equals("EAST"));  
    }  
}
```

运行核对

```
dongg  
false
```

02 enum 圈出合法方向

枚举是一种类型。Direction facing 只能保存 Direction 中声明的值或 null。

枚举变量只能选择声明过的常量

NORTH / SOUTH

南北轴

EAST / WEST

东西轴

UP / DOWN

高度轴

枚举常量通常使用全大写。变量仍使用小驼峰，例如 facing 和 nextDirection。

L24Example02.java • 完整独立示例

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

public class L24Example02 {
    public static void main(String[] args) {
        Direction facing = Direction.EAST;
        System.out.println(facing);
        System.out.println(Direction.values().length);
    }
}
```

运行核对

EAST
6

故障实验：Direction 中没有 DONGG 常量，拼写错误在编译时暴露。下面故意保留错误，请先预测，再尝试修复。

02 enum 圈出合法方向（续）

BadDirection.java • 故意错误示例

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

public class BadDirection {
    public static void main(String[] args) { Direction d = Direction.DONGG; }
```

03 switch 覆盖方向分支

switch 直接匹配枚举常量，不需要 equals，也不会受拼写字符串影响。

枚举 switch 不再比较字符串

switch

覆盖全部六方向

null 检查

进入分支前处理

分支结果

返回对应中文名

switch 不会自动替你处理 null。若允许空方向，应在进入 switch 前明确返回提示或抛出异常。

L24Example03.java • 完整独立示例

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

public class L24Example03 {
    static String label(Direction d) {
        if (d == null) return "未选择";
        return switch (d) {
            case NORTH -> "北"; case SOUTH -> "南";
            case EAST -> "东"; case WEST -> "西";
            case UP -> "上"; case DOWN -> "下";
        };
    }

    public static void main(String[] args) {
        System.out.println(label(Direction.NORTH));
        System.out.println(label(null));
    }
}
```

运行核对

北
未选择

04 枚举保存位移数据

枚举可以拥有字段和构造器。每个方向常量创建时保存自己的三轴增量。

六个方向与三轴位移

方向	dx	dy	dz
NORTH	0	0	-1
SOUTH	0	0	1
EAST	1	0	0
WEST	-1	0	0
UP	0	1	0
DOWN	0	-1	0

方向与位移写在一起，新增方向时不必到处寻找对应数字。

L24Example04.java • 完整独立示例

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

public class L24Example04 {
    public static void main(String[] args) {
        Direction d = Direction.NORTH;
        System.out.println(d.dx + "," + d.dy + "," + d.dz);
    }
}
```

运行核对

0,0,-1

05 record 装下三维坐标

坐标主要携带三个值，并按内容比较。record 会生成组件访问器和相等方法；当组件都是 int 等不可变值时，它适合表达不可变坐标。

一行声明生成构造器访问器和比较方法

record 组件

保存 x、y、z

x() / y() / z()

自动访问器

equals

按组件内容比较

访问器名是 x()、y()、z()，不是 getX()。record 还会生成 equals、hashCode 和 toString。

L24Example05.java • 完整独立示例

```
record BlockPos(int x, int y, int z) { }  
public class L24Example05 {  
    public static void main(String[] args) {  
        BlockPos p = new BlockPos(0, 64, 0);  
        System.out.println(p.x());  
        System.out.println(p.y());  
        System.out.println(p.z());  
    }  
}
```

运行核对

```
0  
64  
0
```

本例组件都是 int，因此坐标是不可变值。若 record 的组件是可变 List，record 只固定引用，不会自动冻结列表内容。

06 移动返回新坐标

record 组件不能随意重新赋值。移动方法根据旧值计算并返回一个新 BlockPos。

旧位置保持不变 新位置表达结果

start

旧坐标保持不变

move

按方向计算增量

next

接收新的坐标对象

start 仍表示原坐标。变量 next 保存移动结果，这让回放、撤销和测试更容易。

06 移动返回新坐标（续）

L24Example06.java • 完整独立示例

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

record BlockPos(int x, int y, int z) {
    BlockPos {
        if (y < -64 || y > 319)
            throw new IllegalArgumentException("高度应为-64到319");
    }
    BlockPos move(Direction direction, int steps) {
        if (direction == null || steps < 0) return this;
        long nx = (long) x + (long) direction.dx * steps;
        long ny = (long) y + (long) direction.dy * steps;
        long nz = (long) z + (long) direction.dz * steps;
        int nextY = (int) Math.max(-64L, Math.min(319L, ny));
        return new BlockPos(Math.toIntExact(nx), nextY,
            Math.toIntExact(nz));
    }
    String shortText() { return "(" + x + "," + y + "," + z + ")"; }
}

public class L24Example06 {
    public static void main(String[] args) {
        BlockPos start = new BlockPos(0, 64, 0);
        BlockPos next = start.move(Direction.EAST, 5);
        System.out.println(start.shortText());
        System.out.println(next.shortText());
    }
}
```

运行核对

(0,64,0)
(5,64,0)

07 值对象的相等与边界

record 解决值的存储与比较，但不会自动知道 Minecraft 课程中的合法高度或移动规则。

内容相同的 record 自动 equals

==

比较对象身份

equals

比较组件内容

高度范围

最低-64，最高319

== 比较是否同一对象，record 的 equals 比较组件内容。坐标通常关心内容相等。

07 值对象的相等与边界（续）

L24Example07.java • 完整独立示例

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

record BlockPos(int x, int y, int z) {
    BlockPos {
        if (y < -64 || y > 319)
            throw new IllegalArgumentException("高度应为-64到319");
    }
    BlockPos move(Direction direction, int steps) {
        if (direction == null || steps < 0) return this;
        long nx = (long) x + (long) direction.dx * steps;
        long ny = (long) y + (long) direction.dy * steps;
        long nz = (long) z + (long) direction.dz * steps;
        int nextY = (int) Math.max(-64L, Math.min(319L, ny));
        return new BlockPos(Math.toIntExact(nx), nextY,
            Math.toIntExact(nz));
    }
    String shortText() { return "(" + x + "," + y + "," + z + ")"; }
}

public class L24Example07 {
    public static void main(String[] args) {
        BlockPos a = new BlockPos(0, 319, 0);
        BlockPos b = new BlockPos(0, 319, 0);
        System.out.println(a == b);
        System.out.println(a.equals(b));
        System.out.println(a.move(Direction.UP, 1).shortText());
        System.out.println(a.move(Direction.DOWN, 1000).shortText());
    }
}
```

运行核对

```
false
true
(0,319,0)
(0,-64,0)
```

本章方块坐标采用 $-64 \leq y < 320$ ：最高可用层是319，320是边界。它是借用主世界高度的练习模型；其他维度和自定义世界应配置自己的范围。X/Z 超出 int 时会明确抛出 ArithmeticException，不会偷偷绕回。

08 挑战 蓝图方块导航器

挑战 程序叫 BlockNavigator。Direction 保存三轴位移，BlockPos 提供 move 与 shortText。

方向携带位移 坐标返回新值

方向与步数

选定变化的轴

高度夹紧

超过边界停在范围内

返回新值

起点仍可回放

先逐步写出 x y z，再运行程序。若最终值不对，只检查发生变化的那一轴。

验收顺序

- 从 (0,64,0) 出发，向东5格、向上300格、向北2格。
- 高度夹紧后，终点应为 (5,319,-2)。
- 向下1000格，结果应为 (5,-64,-2)。
- 起点仍保持 (0,64,0)。
- 负数步数和 null 方向返回原位置；两个同坐标对象 equals 为 true。

09 挑战 实现方块导航

新建 BlockNavigator.java, 使用 Java 21 编译。

enum 与 record 组成不可变路径系统

起点

0、64、0

终点

5、319、-2

底部

5、-64、-2

用 DOWN 1000 验证高度停在-64；用 UP 1 验证顶层319不越界，再确认原坐标不变。

BlockNavigator.java • 第1/2段, 顺序接在同一文件

```
enum Direction {
    NORTH(0, 0, -1), SOUTH(0, 0, 1),
    EAST(1, 0, 0), WEST(-1, 0, 0),
    UP(0, 1, 0), DOWN(0, -1, 0);
    final int dx, dy, dz;
    Direction(int dx, int dy, int dz) {
        this.dx = dx; this.dy = dy; this.dz = dz;
    }
}

record BlockPos(int x, int y, int z) {
    BlockPos {
        if (y < -64 || y > 319)
            throw new IllegalArgumentException("高度应为-64到319");
    }
    BlockPos move(Direction direction, int steps) {
        if (direction == null || steps < 0) return this;
        long nx = (long) x + (long) direction.dx * steps;
        long ny = (long) y + (long) direction.dy * steps;
        long nz = (long) z + (long) direction.dz * steps;
        int nextY = (int) Math.max(-64L, Math.min(319L, ny));
        return new BlockPos(Math.toIntExact(nx), nextY,
            Math.toIntExact(nz));
    }
    String shortText() { return "(" + x + ", " + y + ", " + z + ")"; }
}

public class BlockNavigator {
    public static void main(String[] args) {
        BlockPos start = new BlockPos(0, 64, 0);
```

09 挑战 实现方块导航（续）

BlockNavigator.java • 第2/2段，顺序接在同一文件

```
        BlockPos next = start.move(Direction.EAST, 5)
            .move(Direction.UP, 300).move(Direction.NORTH, 2);
        System.out.println("起点=" + start.shortText());
        System.out.println("终点=" + next.shortText());
        System.out.println("底部=" + next.move(Direction.DOWN, 1000).shortText());
        System.out.println("内容相同=" + start.equals(new BlockPos(0, 64, 0)));
    }
}
```

运行核对

起点=(0, 64, 0)
终点=(5, 319, -2)
底部=(5, -64, -2)
内容相同=true

move 返回新位置；空方向或负步数按合同返回原对象，零步数得到内容相同的结果。构造器拒绝越界初始高度，移动则夹紧高度；这是两种明确的业务规则。

10 拓展任务与通关复盘

作业要同时提交至少四个边界测试，并说明旧对象是否发生变化。

三档任务扩展不可变地图

边界案例

顶层、底层及相邻值

非法输入

负步数、空方向

对象检查

旧值是否保持

第25课进入会扩容的 ArrayList。队列可动态增删，但索引仍从0开始，边界苦力怕也会跟来。

三档作业

- 木镐：创建 Difficulty 枚举，并用 switch 返回 EASY NORMAL HARD 的怪物倍率。
- 铁镐：给 Direction 增加 dx dy dz，完成六个方向移动。
- 钻石镐：设计 TeleportPoint record，保存名字和坐标，并返回偏移后的新传送点。

验收对照表

自评	能	还要练
我会声明 enum	☐	☐
我会使用 record	☐	☐
我理解不可变数据	☐	☐

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>