

## JAVA / LESSON 24 OF 40

# Coordinates, Directions and Enums

Write the legal direction and coordinate rules into the type

This chapter uses JDK 21 and does not require the preview function. | 90-110 minutes

## Make block navigator BlockNavigator

If you spell the wrong direction once, the navigation may fail. enum circles the answer, record saves the coordinates as a value; movement returns the new position, and the boundaries are explicitly specified by you. Prerequisites: classes, constructors, methods and interfaces. Today I will install armor on the coordinates to prevent them from being changed by mistake, and I will also close the entrance that spells the wrong direction.

Before you start: It is recommended to complete Lessons 19, 21-23 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=24&lang=en>

Long code lines may wrap at the > margin marker. The marker is not part of the code.

## Any text can be mixed into the direction variable

| Element          | Explanation                                           |
|------------------|-------------------------------------------------------|
| string           | Possible misspellings and inconsistent capitalization |
| Enumeration type | Delineate legal constants                             |
| null             | Still need special treatment                          |

## In this lesson

- 01. Before starting: predict first, then let the program answer
- 02. New varieties are easy to grow in the direction of the string
- 03. enum circles the legal direction
- 04. switch covers the direction branch
- 05. Enumeration to save displacement data
- 06. record install three-dimensional coordinates
- 07. Move and return to new coordinates
- 08. Equality and Boundaries of Value Objects
- 09. Challenge Blueprint Block Navigator
- 10. Challenge Implementing Block Navigation
- 11. Expansion tasks and clearance review
- 12. Own acceptance record

01 / 12

## Before starting: predict first, then let the program answer

This chapter uses JDK 21 and does not require the preview function. The story is borrowed from the block world, and the example runs on the ordinary Java console; if the values and rules are course models, they will be clearly stated.

Each common example is a complete independent file, saved according to the file name marked on the box. Place each example in a different folder to avoid duplicate names of auxiliary classes. When the complete project is divided into sections, they are connected sequentially in the same file.

**Terminal command: replace the Chinese placeholder with the actual file name and class name**

```
javac -encoding UTF-8 ExampleFile.java
java ExampleClass
```

- Prediction: Write the expected output first, or draw out the changes in references and data.
- Verification: Check line by line after running, and then only change one condition.
- Explanation: Explain whether the results come from Java rules, course contracts, or input data.

"Intentional error" is used to recognize bugs and cannot be mixed with ordinary examples. Use a special temporary directory for file experiments; do not use real archives to test coverage and deletion.

02 / 12

## New varieties are easy to grow in the direction of the string

When using String to save directions, capitalization, spaces, and spelling may create illegal states. enum fixes the legal answers in a finite set.

**Any text can be mixed into the direction variable**

| Element          | Explanation                                           |
|------------------|-------------------------------------------------------|
| string           | Possible misspellings and inconsistent capitalization |
| Enumeration type | Delineate legal constants                             |
| null             | Still need special treatment                          |

When there are only a few fixed values, enum is given priority. The compiler can block non-existent enumeration constants before running.

### L24Example01.java · Complete independent example

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

### Output

```
dongg  
false
```

03 / 12

## enum circles the legal direction

An enumeration is a type. Direction facing can only hold the value declared in Direction or null.

### Enumeration variables can only select declared constants

| Element       | Explanation      |
|---------------|------------------|
| NORTH / SOUTH | North-South axis |
| EAST / WEST   | East-West axis   |
| UP / DOWN     | Height axis      |

Enumeration constants are usually written in all uppercase letters. Variables still use CamelCase, such as facing and nextDirection.

## L24Example02.java · Complete independent example

```
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);
    }
}
```

## Output

```
EAST
6
```

Fault experiment: There is no DONGG constant in Direction, and spelling errors are exposed during compilation. Errors are intentionally left below, please anticipate them first and then try to fix them.

## BadDirection.java · Intentional error example

```
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; }
}
```

04 / 12

## switch covers the direction branch

switch directly matches enumeration constants, does not require equals, and will not be affected by spelling strings.

### Enumeration switch no longer compares strings

| Element        | Explanation                           |
|----------------|---------------------------------------|
| switch         | Covering all six directions           |
| null check     | Enter pre-branch processing           |
| Branch results | Return the corresponding Chinese name |

switch doesn't automatically handle null for you. If the null direction is allowed, a prompt should be explicitly returned or an exception should be thrown before entering the switch.

### L24Example03.java · Complete independent example

```
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 "Not selected";
        return switch (d) {
            case NORTH -> "North"; case SOUTH -> "South";
            case EAST -> "East"; case WEST -> "West";
            case UP -> "on"; case DOWN -> "Next";
        };
    }
    public static void main(String[] args) {
        System.out.println(label(Direction.NORTH));
        System.out.println(label(null));
    }
}
```

## Output

```
North  
Not selected
```

05 / 12

# Enumeration to save displacement data

Enumerations can have fields and constructors. Each direction constant saves its own three-axis increment when created.

## Six directions and three-axis displacement

| Direction | 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  |

Direction and displacement are written together, so there is no need to look for corresponding numbers everywhere when adding a new direction.

## L24Example04.java · Complete independent example

```
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);
    }
}
```

### Output

```
0,0,-1
```

06 / 12

## record install three-dimensional coordinates

Coordinates mainly carry three values and are compared by content. record will generate component accessors and equality methods; it is suitable for expressing immutable coordinates when components are immutable values such as int.

### One line declaration generates constructor accessor and comparison method

| Element          | Explanation                     |
|------------------|---------------------------------|
| record component | Save x, y, z                    |
| x() / y() / z()  | Automatic accessor              |
| equals           | Comparison by component content |

The accessor names are x(), y(), z(), not getX(). record also generates equals, hashCode, and toString.

## L24Example05.java · Complete independent example

```
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());  
    }  
}
```

### Output

```
0  
64  
0
```

The components in this example are all int, so the coordinates are immutable values. If the component of record is a mutable List, record only fixes the reference and does not automatically freeze the contents of the list.

07 / 12

## Move and return to new coordinates

The record component cannot be reassigned at will. The move method calculates and returns a new BlockPos based on the old value.

**The old position remains unchanged and the new position expresses the result.**

| Element | Explanation                          |
|---------|--------------------------------------|
| start   | The old coordinates remain unchanged |
| move    | Calculate increment by direction     |
| next    | Receive new coordinate object        |

start still represents the original coordinates. The variable next holds the result of the move, making playback, undo, and testing easier.

## L24Example06.java · Complete independent example

```
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("The height should be -64 to 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());
    }
}
```

## Output

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

08 / 12

# Equality and Boundaries of Value Objects

record solves the storage and comparison of values, but does not automatically know the legal height or movement rules in a Minecraft class.

## Records with the same content automatically equals

| Element      | Explanation                           |
|--------------|---------------------------------------|
| ==           | Comparison object identity            |
| equals       | Compare component contents            |
| Height range | The lowest is -64, the highest is 319 |

== compares whether they are the same object, and equals of record compares component contents. Coordinates generally care about content equality.

### L24Example07.java · Complete independent example

```
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("The height should be -64 to 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());
    }
}
```

## Output

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

The block coordinates in this chapter are  $-64 \leq y < 320$ : the highest available layer is 319, and 320 is the boundary. It is an exercise model that borrows the height of the main world; other dimensions and custom worlds should configure their own extents. When X/Z exceeds int, `ArithmeticException` will be thrown explicitly and will not wrap around secretly.

09 / 12

## Challenge Blueprint Block Navigator

The challenge program is called `BlockNavigator`. `Direction` saves the three-axis displacement, and `BlockPos` provides `move` and `shortText`.

### The direction carries the displacement and the coordinate returns to the new value

| Element                       | Explanation                                 |
|-------------------------------|---------------------------------------------|
| Direction and number of steps | Selected Axis of Change                     |
| High clamping                 | Stop within range beyond boundary           |
| Return new value              | The starting point can still be played back |

First write x y z step by step, and then run the program. If the final value is incorrect, only the axis that changed is checked.

### Acceptance sequence

- From (0,64,0), go 5 blocks east, 300 blocks up, and 2 blocks north.
- After height clamping, the end point should be (5,319,-2).
- 1000 blocks down, the result should be (5,-64,-2).
- The starting point remains (0,64,0).
- Negative steps and null direction return to the original position; equals for two objects with the same coordinate is true.

10 / 12

## Challenge Implementing Block Navigation

Create a new BlockNavigator.java and compile it using Java 21.

### enum and record form an immutable path system

| Element   | Explanation |
|-----------|-------------|
| Start     | 0, 64, 0    |
| End point | 5, 319, -2  |
| Bottom    | 5, -64, -2  |

Use DOWN 1000 to verify that the height stops at -64; use UP 1 to verify that the top level 319 does not cross the boundary, and then confirm that the original coordinates remain unchanged.

### BlockNavigator.java · Section 1/2, sequentially connected to the same file

```
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("The height should be -64 to 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);
```

## BlockNavigator.java · Paragraph 2/2, sequentially connected to the same file

```
        BlockPos next = start.move(Direction.EAST, 5)
            .move(Direction.UP, 300).move(Direction.NORTH, 2);
        System.out.println("Start=" + start.shortText());
        System.out.println("End point=" + next.shortText());
        System.out.println("Bottom=" + next.move(Direction.DOWN, 1000).shortText());
        System.out.println("The content is the same=" + start.equals(new BlockPos(0, 64, 0)));
    }
}
```

### Output

```
Start=(0,64,0)
End point=(5,319,-2)
Bottom=(5,-64,-2)
The content is the same=true
```

move returns the new position; empty direction or negative step number returns the original object according to the contract, and zero step number obtains the same result. The constructor rejects out-of-bounds initial height, and moves clamp the height; these are two clear business rules.

11 / 12

## Expansion tasks and clearance review

The job must submit at least four boundary tests at the same time, and indicate whether the old objects have changed.

### Third-level task expansion immutable map

| Element           | Explanation                               |
|-------------------|-------------------------------------------|
| Boundary cases    | Top, bottom and adjacent values           |
| Illegal input     | Negative number of steps, empty direction |
| Object inspection | Whether the old value is maintained       |

Lesson 25 enters the expanded ArrayList. The queue can be added or deleted dynamically, but the index still starts from 0, and the border creepers will follow.

### Third gear operation

- Wooden Pickaxe: Create a Difficulty enum and switch to return EASY NORMAL HARD's monster multiplier.

- Iron pickaxe: Add dx dy dz to Direction to complete movement in six directions.
- Diamond Pickaxe: Design TeleportPoint record, save name and coordinates, and return the new teleport point after offset.

### Acceptance comparison table

| Self-evaluation             | Can                      | Still need to practice   |
|-----------------------------|--------------------------|--------------------------|
| I will declare enum         | <input type="checkbox"/> | <input type="checkbox"/> |
| I would use record          | <input type="checkbox"/> | <input type="checkbox"/> |
| I understand immutable data | <input type="checkbox"/> | <input type="checkbox"/> |

12 / 12

## Own acceptance record

Beautiful output is the starting point, and being able to explain boundaries is considered mastery. Select one case each from normal, borderline and error, write down your expectations before running, and leave evidence after running.

### Leave a basis for judgment

| Category | The operation you selected                   | Check key points                                                       |
|----------|----------------------------------------------|------------------------------------------------------------------------|
| Normal   | A common input or a successful action        | Is the result consistent line by line?                                 |
| Border   | Range endpoints and adjacent values          | Is there one more step or one less thing?                              |
| Error    | Illegal quantity, missing object or bad data | Whether it is explicitly rejected and whether the status is maintained |

- Bug records: triggering conditions, actual phenomena, causes, and minimum repairs.
- Illustrated record: describe the starting point, end point and meaning of each arrow.
- Adaptation record: only change one rule, and modify expectations and tests at the same time.

Technical check: Java SE 21 language specification and standard library. See the language specification for class, interface, type and expression rules; see the corresponding API for the behavior of collections, Random, Files and Stream.

<https://docs.oracle.com/javase/specs/jls/se21/html/index.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>