

■■ Check ODD or EVEN

Bit Manipulation • Easy

■ The Hook

Every kid learns odd/even with the rule: divide by 2, check the remainder. That works, but division is slow. A CPU performing a billion such checks per second feels that drag. Here is the thing — the last bit of any integer already encodes odd/even. No division needed. One bitwise AND, one comparison, done. This is your first real taste of reading a number's DNA directly.

■ Core Intuition

Binary numbers are sums of powers of 2. Every even number is a sum of 2, 4, 8, 16... — all divisible by 2. The ones place (bit 0) contributes exactly 1 if set. So: if bit 0 is 1 → odd. If bit 0 is 0 → even. Simple.

- 5 → ...0101 → bit 0 = 1 → ODD
- 6 → ...0110 → bit 0 = 0 → EVEN
- 13 → ...1101 → bit 0 = 1 → ODD
- -4 → two's complement ends in 00 → EVEN

■■ How It Works Internally

The AND operator compares two numbers bit by bit. We use 1 (binary: ...0001) as the mask — it has only bit 0 set. ANDing any number n with 1 zeroes out every bit except bit 0. The result is either 0 (even) or 1 (odd).

```
n = 7 → binary: 0 1 1 1
mask = 1 → binary: 0 0 0 1
n & 1 → binary: 0 0 0 1 = 1 (ODD)

n = 8 → binary: 1 0 0 0
mask = 1 → binary: 0 0 0 1
n & 1 → binary: 0 0 0 0 = 0 (EVEN)
```

■ Visual — Bit Layout

For $n = 13$ (binary 1101):

Bit Position	3	2	1	0 ← check this
Value	1	1	0	1

■ Dry Run — n = 13

Step-by-step execution of isOdd(13):

- n = 13, binary = 1101
- mask = 1, binary = 0001
- n & 1 = 0001 = 1
- 1 != 0 → return true → ODD ✓

■ Java Implementation

```
public class OddEven {
    // Check if n is odd using bitwise AND
    public static boolean isOdd(int n) {
        return (n & 1) == 1; // bit 0 set → odd
    }

    // Alternatively return (n & 1) != 0 — same thing
    public static String checkOddEven(int n) {
        return (n & 1) == 1 ? "ODD" : "EVEN";
    }

    public static void main(String[] args) {
        System.out.println(checkOddEven(7)); // ODD
        System.out.println(checkOddEven(8)); // EVEN
        System.out.println(checkOddEven(-3)); // ODD (two's comp)
    }
}
```

■ Time & Space Complexity

Case	Time	Space	Why
All cases	O(1)	O(1)	Single AND instruction — no loops, no recursion

■ Common Mistakes

- Using $n \% 2 == 1$ for negative numbers — wrong! $-3 \% 2$ gives -1 in Java, not 1 . Use $(n \& 1) == 1$ or $n \% 2 != 0$.
- Forgetting that $\&$ has lower precedence than $==$. Write $(n \& 1) == 1$, not $n \& 1 == 1$.

■ Patterns & Connections

- Checking any specific bit: $(n \gg k) \& 1$
- This is the foundation of all bit-checking problems
- Used in fast hash functions, parity checks, and network protocols

■ Practice Problems

Problem	Difficulty
Check ODD or EVEN	Easy
Find if number is a multiple of 4	Easy
Count even numbers in array using bit tricks	Easy
Game of Nim (parity-based strategy)	Medium

■ Quick Revision

- ODD if last bit is 1: $(n \& 1) == 1$
- EVEN if last bit is 0: $(n \& 1) == 0$
- Works correctly for negative integers (two's complement)
- $O(1)$ time and space — no faster check exists

■ Mental Model

Think of a number as a light switch panel. The rightmost switch is the 'ones' switch. If it's ON → odd. If it's OFF → even. You never need to look at the other switches.