> *1. WAP to `count digit` of a Number.*

```java
    public static int countDigit(int n)
    {
        int count=0;
        while(n!=0)
        {
            count++;
            n/=10;
        }
        return count;
    }
```

Method Tracing: `countDigit(int n)`

Purpose

This method counts the number of digits in a given integer `n`.

Initial Setup

- `count` is initialized to `0`
- Input `n` is processed in a loop until it becomes `0`

Tracing Steps

Example 1: `n = 12345`

Iteration	n (before)	Condition (n != 0)	count++	n /= 10 (after)	count (after)
1	12345	true	1	1234	1
2	1234	true	2	123	2
3	123	true	3	12	3
4	12	true	4	1	4
5	1	true	5	0	5
6	0	false (loop ends)	-	-	5
Final Return Value: 5					

Example 2: `n = 0`

Iteration	n (before)	Condition (n != 0)	count++	n /= 10 (after)	count (after)
1	0	false (loop ends)	-	-	0

Final Return Value: `0`

Example 3: `n = -789` **(Negative Number)**

Iteration	n (before)	Condition (n != 0)	count++	n /= 10 (after)	count (after)
1	-789	true	1	-78	1
2	-78	true	2	-7	2
3	-7	true	3	0	3
4	0	false (loop ends)	-	-	3

Final Return Value: `3`

Key Observations

1. The loop continues until `n` becomes `0`
2. Each iteration:
   - Increments `count` by 1
   - Divides `n` by 10 (integer division)
3. Works for negative numbers (treats them the same as positives)
4. Returns `0` when input is `0` (edge case)
5. Time Complexity: $O(\log_{10} n)$ (number of digits in `n`)

Edge Cases

- `n = 0` → returns `0`
- `n = 1` → returns `1`
- `n = -1` → returns `1`
- Maximum/Minimum integer values work normally

> *2. WAP to find `digital sum` of a digit.*

```
public static int digitSum(int n)
{            int sum =0;
    while(n!=0)
    {
        sum+=n%10;
        n/=10;
    }
    return sum;
}
```

Method Tracing: `digitSum(int n)`

Purpose

This method calculates the sum of all digits in a given integer `n`.

Initial Setup

- `sum` is initialized to `0`
- Input `n` is processed in a loop until it becomes `0`

Tracing Steps

Example 1: `n = 12345`

Iteration	n (before)	Condition (n != 0)	n%10 (digit)	sum += digit	n /= 10 (after)	sum (after)
1	12345	true	5	0 + 5 = 5	1234	5
2	1234	true	4	5 + 4 = 9	123	9
3	123	true	3	9 + 3 = 12	12	12
4	12	true	2	12 + 2 = 14	1	14
5	1	true	1	14 + 1 = 15	0	15
6	0	false (loop ends)	-	-	-	15

Final Return Value: `15`

Example 2: `n = 0`

Iteration	n (before)	Condition (n != 0)	n%10 (digit)	sum += digit	n /= 10 (after)	sum (after)
1	0	false (loop ends)	-	-	-	0

Final Return Value: `0`

Example 3: `n = -789` **(Negative Number)**

Iteration	n (before)	Condition (n != 0)	n%10 (digit)	sum += digit	n /= 10 (after)	sum (after)
1	-789	true	-9	0 + (-9) = -9	-78	-9
2	-78	true	-8	-9 + (-8) = -17	-7	-17
3	-7	true	-7	-17 + (-7) = -24	0	-24
4	0	false (loop ends)	-	-	-	-24

Final Return Value: `-24`

(Note: For negative numbers, the sum will also be negative)

Key Observations

1. The loop continues until `n` becomes `0`
2. Each iteration:
   - Extracts the last digit using `n%10`
   - Adds the digit to `sum`
   - Removes the last digit using `n /= 10`
3. Handles negative numbers (digits contribute negatively to the sum)

4. Returns `0` when input is `0` (edge case)
5. Time Complexity: O(log$_{10}$ n) (number of digits in `n`)

Edge Cases

- `n = 0` → returns `0`
- `n = 9` → returns `9`
- `n = -9` → returns `-9`
- Single-digit numbers return the digit itself
- Works with maximum/minimum integer values

> *3. WAP to reverse a Digit of Number.*

```
public static int reverseDigit(int n)
{
    int revNum=0;
    while(n!=0)
    {
        revNum=revNum*10+n%10;
        n/=10;
    }
    return revNum;
}
```

Method Tracing: `reverseDigit(int n)`

Purpose

This method reverses the digits of a given integer `n` (e.g., 1234 → 4321).

Initial Setup

- `revNum` is initialized to `0`
- Input `n` is processed in a loop until it becomes `0`

Tracing Steps

Example 1: `n = 1234`

Iteration	n (before)	Condition (n != 0)	n%10 (digit)	revNum = revNum*10 + digit	n /= 10 (after)	revNum (after)
1	1234	true	4	0*10 + 4 = 4	123	4
2	123	true	3	4*10 + 3 = 43	12	43
3	12	true	2	43*10 + 2 = 432	1	432
4	1	true	1	432*10 + 1 = 4321	0	4321
5	0	false (loop ends)	-	-	-	4321

Final Return Value: `4321`

Example 2: `n = 100`

Iteration	n (before)	Condition (n != 0)	n%10 (digit)	revNum = revNum*10 + digit	n /= 10 (after)	revNum (after)
1	100	true	0	0*10 + 0 = 0	10	0
2	10	true	0	0*10 + 0 = 0	1	0
3	1	true	1	0*10 + 1 = 1	0	1
4	0	false (loop ends)	-	-	-	1

Final Return Value: `1`
(Note: Leading zeros in the original number are dropped in the reversal)

Example 3: `n = -123` **(Negative Number)**

Iteration	n (before)	Condition (n != 0)	n%10 (digit)	revNum = revNum*10 + digit	n /= 10 (after)	revNum (after)
1	-123	true	-3	0*10 + (-3) = -3	-12	-3
2	-12	true	-2	-3*10 + (-2) = -32	-1	-32
3	-1	true	-1	-32*10 + (-1) = -321	0	-321
4	0	false (loop ends)	-	-	-	-321

Final Return Value: `-321`
(Preserves the negative sign while reversing digits)

Key Observations

1. **Digit Extraction**: `n%10` gets the last digit
2. **Number Construction**: `revNum*10 + digit` appends the digit
3. **Termination**: Loop exits when `n` becomes `0`
4. **Handling Negatives**: Maintains sign while reversing digits
5. **Leading Zeros**: Drops leading zeros from original number
6. **Time Complexity**: O($\log_{10}$ n) (number of digits in `n`)

Edge Cases

Input (n)	Output	Notes
0	0	Returns 0 immediately
5	5	Single-digit unchanged
-5	-5	Single negative-digit unchanged
1200	21	Trailing zeros become leading

Integer.MAX_VALUE (2147483647)	7463847412	*May cause integer overflow*

Special Note

- **Overflow Risk**: For large reversed numbers (e.g., reversing 2147483647 gives 7463847412 which exceeds `Integer.MAX_VALUE`), the result may be incorrect due to integer overflow. This implementation doesn't handle overflow cases.