

//even odd

```
public static int checkEvenOrOdd(int num) {  
    if (num % 2 == 0) {  
        return 2; // Return 2 if even  
    } else {  
        return 1; // Return 1 if odd  
    }  
}
```

//odd

```
public static int checkOdd(int n) {  
    if (n % 2 != 0) {  
        return 2; // Odd  
    } else {  
        return 1; // Even  
    }  
}
```

//even or odd

```
public class OddEvenChecker {
```

```
    public static int checkOddEven(int number) {  
        if (number % 2 != 0) {  
            return 2; // Odd  
        } else {  
            return 1; // Even  
        }  
    }
```

//last digit

```
public class LastDigitFinder {

    public static int getLastDigit(int number) {
                return Math.abs(number % 10); // Handles
                negative numbers too
    }

//second last digit

public static int getSecondLastDigit(int number) {
        number = Math.abs(number); // Handle negative numbers
        if (number < 10) {
                return -1; // No second last digit exists
    }
        return (number / 10) % 10;
}

//sum of last digits of given number

public class SumOfDigits {
public class LastDigitSum {

    public static int lastadd(int input1, int input2) {
                int lastDigit1 = Math.abs(input1) % 10;
                int lastDigit2 = Math.abs(input2) % 10;
                return lastDigit1 + lastDigit2;
    }

//is N exact multiple of M

public class EvenCounter {
```

```
public static int countEven(int a, int b, int c, int d, int e) {  
    int count = 0;  
  
    if (a % 2 == 0) count++;  
    if (b % 2 == 0) count++;  
    if (c % 2 == 0) count++;  
    if (d % 2 == 0) count++;  
    if (e % 2 == 0) count++;  
  
    return count;  
}  
//from the 5 given numbers how many are even or odd>
```

```
public static int countByType(int input1, int input2, int input3, int input4, int input5, String  
type) {  
    int count = 0;  
  
    if (type.equalsIgnoreCase("odd")) {  
        if (input1 % 2 != 0) count++;  
        if (input2 % 2 != 0) count++;  
        if (input3 % 2 != 0) count++;  
        if (input4 % 2 != 0) count++;  
        if (input5 % 2 != 0) count++;  
    } else if (type.equalsIgnoreCase("even")) {  
        if (input1 % 2 == 0) count++;  
        if (input2 % 2 == 0) count++;  
        if (input3 % 2 == 0) count++;  
        if (input4 % 2 == 0) count++;  
        if (input5 % 2 == 0) count++;  
    }  
}
```

```
    }

    return count;
}
```

Let me know if you'd like to test it with user input!

Ask ChatGPT

You said:

is prime

ChatGPT said:

To check whether a number is prime, we can write a simple function in Java. Here's a clean and efficient way to do it:

Java Method: Check if a Number is Prime

java

[Copy code](#)

```
public class PrimeCheck {

    public static boolean isPrime(int num) {
        if (num <= 1) return false; // 0 and 1 are not prime
```

```
for (int i = 2; i <= Math.sqrt(num); i++) {  
    if (num % i == 0)  
        return false; // divisible by a number other than 1 and itself  
    }  
    return true;  
}  
  
public static void main(String[] args) {  
    int number = 29; // test any number here  
    if (isPrime(number)) {  
        System.out.println(number + " is a Prime number");  
    } else {  
        System.out.println(number + " is NOT a Prime number");  
    }  
}
```

 **Explanation:**

A prime number is a number greater than 1 that has no divisors other than 1 and itself.

Loop from 2 to sqrt(num) (optimization).

If any number divides it exactly → it's not prime.

 **Sample Output:**

csharp

[Copy code](#)

29 is a Prime number

Let me know if you want this to work with multiple inputs or as a reusable utility.

You said:

factorial of a number

ChatGPT said:

Here's a simple Java program to calculate the factorial of a number:

Java Code: Factorial of a Number

java

[Copy code](#)

```
public class Factorial {  
  
    public static long factorial(int n) {  
        if (n < 0) {  
            throw new IllegalArgumentException("Factorial is not defined for negative  
numbers.");  
        }  
  
        long result = 1;  
        for (int i = 2; i <= n; i++) {  
            result *= i;  
        }  
    }  
}
```

```
    return result;  
}  
  
public static void main(String[] args) {  
    int num = 5; // Change this value to test with other numbers  
    long fact = factorial(num);  
    System.out.println("Factorial of " + num + " is: " + fact);  
}  
}
```

 **Explanation:**

factorial(5) = $5 \times 4 \times 3 \times 2 \times 1 = 120$

Returns long to handle larger results (up to ~20!).

 **Output for num = 5:**

csharp

[Copy code](#)

Factorial of 5 is: 120

Let me know if you want a recursive version, or want to read the input from the user using Scanner.

You said:

give another way like fact*act-1

ChatGPT said:

You're referring to the recursive way of calculating factorial using the relation:

$$\text{fact}(n) = n * \text{fact}(n - 1)$$

 **Java Code: Recursive Factorial**

java

[Copy code](#)

```
public class RecursiveFactorial {
```

```
    public static long factorial(int n) {
```

```
        if (n < 0) {
```

```
            throw new IllegalArgumentException("Factorial is not defined for negative numbers.");
```

```
    }
```

```
    if (n == 0 || n == 1) {
```

```
        return 1; // Base case
```

```
    }
```

```
    return n * factorial(n - 1); // Recursive case
```

```
}
```

```
public static void main(String[] args) {
```

```
    int num = 5; // Change this to test other inputs
```

```
    long result = factorial(num);
```

```
    System.out.println("Factorial of " + num + " is: " + result);
```

```
    }  
}
```

💡 How It Works:

```
factorial(5)  
→ 5 * factorial(4)  
→ 5 * 4 * factorial(3)  
→ ... until factorial(1) → returns 1
```

✓ Output for 5:

csharp

[Copy code](#)

Factorial of 5 is: 120

Let me know if you want to handle large numbers with BigInteger or add user input using Scanner.

You said:

nth fibanocci

ChatGPT said:

Here's how you can find the nth Fibonacci number in Java.

✓ Method 1: Recursive Approach

java

Copy code

```
public class FibonacciRecursive {  
  
    public static int fibonacci(int n) {  
        if (n <= 1) return n; // Base cases: 0 or 1  
        return fibonacci(n - 1) + fibonacci(n - 2); // Recursive formula  
    }  
  
    public static void main(String[] args) {  
        int n = 6; // Find the 6th Fibonacci number  
        System.out.println("Fibonacci number at position " + n + " is: " + fibonacci(n));  
    }  
}
```



Note:

Simple but slow for large n due to repeated calls.

Method 2: Iterative (Efficient) Approach

java

Copy code

```
public class Fibonacciterative {
```

```
    public static int fibonacci(int n) {
```

```
        if (n <= 1) return n;
```

```
        int a = 0, b = 1, fib = 1;
```

```
        for (int i = 2; i <= n; i++) {
```

```
            fib = a + b;
```

```
a = b;  
b = fib;  
}  
  
return fib;  
}  
  
public static void main(String[] args) {  
    int n = 6; // You can change this  
    System.out.println("Fibonacci number at position " + n + " is: " + fibonacci(n));  
}
```

 **Fibonacci Sequence:**

Copy code

0, 1, 1, 2, 3, 5, 8, 13, ...

fib(0) = 0

fib(1) = 1

fib(2) = 1

fib(3) = 2

...

Let me know if you want to generate the first N Fibonacci numbers instead!

You said:

Default

CODE EXECUTION DETAILS

Time: 118 ms

Memory: 57820 kb

TEST CASE INFORMATION

Input

1

Expected Output

0

Actual Output

-1

CONSOLE OUTPUT

STANDARD ERROR/WARNING

None

ChatGPT said:

Thanks for sharing the issue!

You have a test case where:

Input: 1

Expected Output: 0

Actual Output: -1

This indicates that your code is returning -1 for input 1, while it should return 0.



Problem Diagnosis

This likely means your Fibonacci function is 1-based, but the test case expects 0-based indexing.



Fix: Adjust the base cases

If the 1st Fibonacci number should be 0, and the 2nd should be 1, then your code should be:

java

Copy code

```
public class FibonacciFix {  
  
    public static int fibonacci(int n) {  
        if (n <= 0) return -1; // invalid case  
        if (n == 1) return 0; // 1st Fibonacci = 0  
        if (n == 2) return 1; // 2nd Fibonacci = 1  
  
        int a = 0, b = 1, fib = 1;  
        for (int i = 3; i <= n; i++) {  
            fib = a + b;  
            a = b;  
            b = fib;  
        }  
        return fib;  
    }  
  
    public static void main(String[] args) {  
        int n = 1; // Example input  
        System.out.println(fibonacci(n)); // Should print 0  
    }  
}
```

 **Explanation:**

fibonacci(1) → returns 0

fibonacci(2) → returns 1

fibonacci(3) → returns 1

fibonacci(4) → returns 2

and so on...

Let me know if your platform uses 0-based or 1-based indexing — I'll adjust accordingly.

You said:

nth prime numer

ChatGPT said:

To find the nth prime number in Java, you can use a method that keeps checking numbers for primality until it finds the nth one.

 **Java Code: Find the nth Prime Number**

java

[Copy code](#)

```
public class NthPrimeFinder {  
  
    public static boolean isPrime(int num) {  
        if (num <= 1) return false;  
        for (int i = 2; i <= Math.sqrt(num); i++) {  
            if (num % i == 0)  
                return false;  
        }  
        return true;  
    }  
}
```

```
        }

    return true;
}

public static int findNthPrime(int n) {

    if (n <= 0) return -1; // Invalid input

    int count = 0;
    int number = 1;

    while (count < n) {

        number++;
        if (isPrime(number)) {

            count++;
        }
    }

    return number;
}
```

```
public static void main(String[] args) {

    int n = 6; // Change this to find nth prime
    int nthPrime = findNthPrime(n);
    System.out.println("The " + n + "th prime number is: " + nthPrime);
}
```

 **Sample Outputs:**

n = 1 → Output: 2

n = 2 → Output: 3

n = 3 → Output: 5

n = 6 → Output: 13

Let me know if you want to print the first N primes instead or improve efficiency for large n.

You said:

give it simple jus to find the nth primeliek where in between 1 to 10000 only and if n is 10 then return 29

ChatGPT said:

Sure! Here's a simple Java program that finds the nth prime number (where n is between 1 to 10000).

If n = 10, it will return 29 ✓

 **Simple Java Code: Find nth Prime (Within 1 to 10000)**

java

Copy code

```
public class NthPrimeSimple {
```

```
public static boolean isPrime(int num) {
    if (num <= 1) return false;

    for (int i = 2; i <= Math.sqrt(num); i++) {
        if (num % i == 0)
            return false;
    }

    return true;
}

public static int nthPrime(int n) {
    int count = 0;
    int number = 1;

    while (count < n) {
        number++;
        if (isPrime(number)) {
            count++;
        }
    }

    return number;
}

public static void main(String[] args) {
    int n = 10; // You can change this to test other positions
    int result = nthPrime(n);
    System.out.println("The " + n + "th prime number is: " + result);
}
```

```
}
```

 Output for n = 10:

csharp

[Copy code](#)

The 10th prime number is: 29

Let me know if you'd like it to validate input (e.g., `n > 0 && n <= 10000`) or return -1 for invalid cases.

You said:

ChatGPT said:

Thanks! Based on the screenshot and your constraints (N between 1 and 1000), here's the completed code for the NthPrime method in Java:

 Final Code (To Paste in Your Platform):

java

[Copy code](#)

```
int NthPrime(int input1)
```

```
{
```

```
    int count = 0;
```

```
    int number = 1;
```

```
    while (count < input1) {
```

```
number++;

if (isPrime(number)) {
    count++;
}

}

return number;
}

boolean isPrime(int num) {
    if (num <= 1) return false;
    for (int i = 2; i <= Math.sqrt(num); i++) {
        if (num % i == 0)
            return false;
    }
    return true;
}
```

 Example:

NthPrime(10) returns 29 

NthPrime(13) returns 41 

This code meets the expected behavior shown in the problem. Let me know if you want it optimized or adapted for large n values.

You said:

numbers o prime in a speciic rangw

ChatGPT said:

To count how many prime numbers exist in a specific range (say between start and end), here's a simple and clear Java method:

 **Java Code: Count Primes in a Range**

java

[Copy code](#)

```
int countPrimesInRange(int start, int end) {  
    int count = 0;  
    for (int i = start; i <= end; i++) {  
        if (isPrime(i)) {  
            count++;  
        }  
    }  
    return count;  
}
```

```
boolean isPrime(int num) {  
    if (num <= 1) return false;  
    for (int i = 2; i <= Math.sqrt(num); i++) {  
        if (num % i == 0) return false;  
    }  
    return true;
```

```
}
```

 Example:

java

[Copy code](#)

```
System.out.println(countPrimesInRange(10, 50)); // Output: 10
```

 Prime numbers between 10 and 50 are: 11, 13, 17, 19, 23, 29, 31, 37, 41, 43

Let me know if you want it to print the primes too or use the Sieve of Eratosthenes for better performance on large ranges.

You said:

all digit count

ChatGPT said:

If you want to count the total number of digits in a given number (or multiple numbers), here's a simple Java method:

 Java Code: Count Total Digits in a Number

java

[Copy code](#)

```
int countDigits(int num) {  
    int count = 0;  
    num = Math.abs(num); // To handle negative numbers
```

```
if (num == 0) return 1;  
while (num > 0) {  
    num /= 10;  
    count++;  
}  
return count;  
}
```

 Example:

java

[Copy code](#)

```
System.out.println(countDigits(12345)); // Output: 5
```

```
System.out.println(countDigits(-890)); // Output: 3
```

```
System.out.println(countDigits(0)); // Output: 1
```

Let me know if you want to:

Count each digit's frequency (e.g., how many 1s, 2s, etc.)

Count digits in an array of numbers

Handle input as a string or multiple integers

You said:

unique digit count

ChatGPT said:

To count the number of unique digits in a given number (e.g., in 112233, the unique digits are 1, 2, and 3 → count is 3), here's a simple and clear Java implementation:

 **Java Code: Unique Digit Count**

java

Copy code

```
int countUniqueDigits(int num) {  
    boolean[] digits = new boolean[10]; // for digits 0-9  
    num = Math.abs(num); // handle negative numbers  
  
    while (num > 0) {  
        int digit = num % 10;  
        digits[digit] = true;  
        num /= 10;  
    }  
  
    int count = 0;  
    for (boolean seen : digits) {  
        if (seen) count++;  
    }  
  
    return count;  
}
```

 **Example:**

java

Copy code

```
System.out.println(countUniqueDigits(112233)); // Output: 3  
System.out.println(countUniqueDigits(9876543210)); // Output: 10  
System.out.println(countUniqueDigits(1111)); // Output: 1
```

Let me know if you want to:

Count unique digits across multiple numbers,

Work with a number in string format (e.g., String numStr = "112233"), or

Return the actual unique digits, not just their count.

You said:

give it simpel oen

ChatGPT said:

Sure! Here's a simple and short Java version to count unique digits in a number:

 Simple Java Code:

java

[Copy code](#)

```
int countUniqueDigits(int num) {
```

```
    int[] freq = new int[10];
```

```
num = Math.abs(num);

while (num > 0) {
    int digit = num % 10;
    freq[digit] = 1;
    num /= 10;
}

int count = 0;
for (int i = 0; i < 10; i++) {
    count += freq[i];
}

return count;
}
```

◆ Example:

java

Copy code

```
System.out.println(countUniqueDigits(112233)); // Output: 3
System.out.println(countUniqueDigits(12345)); // Output: 5
System.out.println(countUniqueDigits(111)); // Output: 1
```

Let me know if you want to do the same for a String input too.

You said:

non repeating digit count

ChatGPT said:

Here's a simple Java method to count non-repeating (non-duplicate) digits in a number:

 **Java Code: Non-Repeating Digit Count**

java

Copy code

```
int countNonRepeatingDigits(int num) {  
    int[] freq = new int[10];  
    num = Math.abs(num);  
  
    // Count frequency of each digit  
    while (num > 0) {  
        int digit = num % 10;  
        freq[digit]++;  
        num /= 10;  
    }  
  
    // Count digits that appear exactly once  
    int count = 0;  
    for (int i = 0; i < 10; i++) {  
        if (freq[i] == 1) {  
            count++;  
        }  
    }  
}
```

```
    return count;  
}
```

◆ Example Usage:

java

[Copy code](#)

```
System.out.println(countNonRepeatingDigits(121345)); // Output: 3 (digits 3, 4, 5)
```

```
System.out.println(countNonRepeatingDigits(112233)); // Output: 0
```

```
System.out.println(countNonRepeatingDigits(987)); // Output: 3
```

Let me know if you also want the digits themselves, not just the count.

You said:

digit sum :sum o all digits

ChatGPT said:

Here's a simple Java method to calculate the digit sum — that is, the sum of all digits in a number:

 [Java Code: Sum of All Digits](#)

java

[Copy code](#)

```
int digitSum(int num) {  
    num = Math.abs(num); // Handle negative numbers
```

```
int sum = 0;

while (num > 0) {
    sum += num % 10;
    num /= 10;
}

return sum;
}
```

◆ Example Usage:

java

Copy code

```
System.out.println(digitSum(1234)); // Output: 10 (1+2+3+4)
System.out.println(digitSum(9876)); // Output: 30 (9+8+7+6)
System.out.println(digitSum(-123)); // Output: 6 (1+2+3)
```

Let me know if you want a recursive version or want to include it as part of a class or input-driven program!

You said:

no not like for eaxmple if 12345 is digits then values is like(1+2+3+4+5=15 then 1+5=6

ChatGPT said:

Ah, you're asking for the digit sum until a single digit remains — also called the digital root.

 **Java Code: Final Digit Sum (Digital Root)**

java

Copy code

```
int finalDigitSum(int num) {  
    num = Math.abs(num); // Handle negative numbers  
  
    while (num >= 10) {  
        int sum = 0;  
        while (num > 0) {  
            sum += num % 10;  
            num /= 10;  
        }  
        num = sum;  
    }  
  
    return num;  
}
```

◆ Example:

java

Copy code

```
System.out.println(finalDigitSum(12345)); // Output: 6 (1+2+3+4+5=15 → 1+5=6)
```

```
System.out.println(finalDigitSum(9876)); // Output: 3 (9+8+7+6=30 → 3+0=3)
```

Let me know if you want a version using recursion or mathematical optimization (digital root = $1 + (\text{num} - 1) \% 9$ for $\text{num} > 0$).

You said:

CODE EXECUTION DETAILS

Time: 122 ms

Memory: 57820 kb

TEST CASE INFORMATION

Input

-99999

Expected Output

-9

Actual Output

-99999

CONSOLE OUTPUT

STANDARD ERROR/WARNING

None

Mettl Online Assessment © 2021-2031

WiFi Signal

Need Help? Contact us (Please add country code while dialing)

ChatGPT said:

Thanks for sharing the test case details!

You're trying to compute the digit sum until a single digit remains, and the expected output for negative numbers is also negative — e.g., input -99999 → output -9.



Fix:

In the earlier version, we used `Math.abs(num)`, which turns the number positive. But from your test case, we now know:

! If the input is negative, the final output should also be negative.

Correct Java Code (Preserve sign):

java

Copy code

```
int finalDigitSum(int num) {
```

```
    if (num == 0) return 0;
```

```
    int sign = num < 0 ? -1 : 1;
```

```
    num = Math.abs(num);
```

```
while (num >= 10) {  
    int sum = 0;  
    while (num > 0) {  
        sum += num % 10;  
        num /= 10;  
    }  
    num = sum;  
}  
  
return num * sign;  
}
```

◆ Example:

java

Copy code

```
System.out.println(finalDigitSum(12345)); // Output: 6  
System.out.println(finalDigitSum(-99999)); // Output: -9  
System.out.println(finalDigitSum(0)); // Output: 0
```

Let me know if you want this using recursion or as a one-liner using math.

You said:

CODE EXECUTION DETAILS

Time: 127 ms

Memory: 57688 kb

TEST CASE INFORMATION

Input

-99999

Expected Output

-9

Actual Output

99999

CONSOLE OUTPUT

STANDARD ERROR/WARNING

None

Need Help? Contact us (Please add country code while dialing)

ChatGPT said:

The issue is that the current logic is stripping the negative sign but not restoring it at the end — hence returning 99999 instead of -9.

Here's a simple and correct version that works exactly as required, including for negative inputs like -99999 → -9.

 Final Correct Java Code:

java

[Copy code](#)

```
public static int finalDigitSum(int num) {  
    if (num == 0) return 0;  
  
    int sign = num < 0 ? -1 : 1;  
    num = Math.abs(num);  
  
    while (num >= 10) {  
        int sum = 0;  
        while (num > 0) {  
            sum += num % 10;  
            num /= 10;  
        }  
        num = sum;  
    }  
  
    return num * sign;  
}
```

◆ Sample Test Outputs:

java

Copy code

```
System.out.println(finalDigitSum(12345)); // Output: 6
```

```
System.out.println(finalDigitSum(-99999)); // Output: -9
```

```
System.out.println(finalDigitSum(0)); // Output: 0
```

```
System.out.println(finalDigitSum(18)); // Output: 9
```

```
System.out.println(finalDigitSum(-1234)); // Output: -1
```

Let me know if you also want this done in recursive or one-liner format.

You said:

sum of even digit

ChatGPT said:

Here's a simple Java method to calculate the sum of even digits in a given number:

 **Java Code: Sum of Even Digits**

java

Copy code

```
public static int sumOfEvenDigits(int num) {
```

```
    num = Math.abs(num); // Handle negative numbers
```

```
    int sum = 0;
```

```
while (num > 0) {  
    int digit = num % 10;  
    if (digit % 2 == 0) {  
        sum += digit;  
    }  
    num /= 10;  
}  
  
return sum;  
}
```

◆ Example Outputs:

java

Copy code

```
System.out.println(sumOfEvenDigits(123456)); // Output: 2 + 4 + 6 = 12  
System.out.println(sumOfEvenDigits(-28471)); // Output: 2 + 8 + 4 = 14  
System.out.println(sumOfEvenDigits(111)); // Output: 0  
System.out.println(sumOfEvenDigits(0)); // Output: 0
```

Let me know if you want this modified for odd digits, recursive approach, or input from the user.

```
//is prime  
public class PrimeCheck {
```

```
public static boolean isPrime(int num) {  
    if (num <= 1) return false; // 0 and 1 are not prime  
  
    for (int i = 2; i <= Math.sqrt(num); i++) {  
        if (num % i == 0)  
            return false; // divisible by a number other than 1 and itself  
    }  
    return true;  
}
```

Pub

```
//facorial of a number
```

```
public class Factorial {
```

```
    public static long factorial(int n) {  
        if (n < 0) {  
            throw new IllegalArgumentException("Factorial is not defined for negative  
numbers.");  
        }  
  
        long result = 1;
```

```
        for (int i = 2; i <= n; i++) {  
            result *= i;  
        }  
  
        return result;  
    }
```

```
//give another way like fact*act-1

public static long factorial(int n) {

    if (n < 0) {

        throw new IllegalArgumentException("Factorial is not defined for negative
numbers.");
    }

    if (n == 0 || n == 1) {

        return 1; // Base case
    }

    return n * factorial(n - 1); // Recursive case
}

//nth fibanocci

public static int fibonacci(int n) {

    if (n <= 1) return n; // Base cases: 0 or 1

    return fibonacci(n - 1) + fibonacci(n - 2); // Recursive formula
}

//nth prime numer

public class NthPrimeSimple {

    public static boolean isPrime(int num) {

        if (num <= 1) return false;

        for (int i = 2; i <= Math.sqrt(num); i++) {

            if (num % i == 0)

                return false;
        }

        return true;
    }
}
```

```
}
```

```
public static int nthPrime(int n) {  
    int count = 0;  
    int number = 1;  
  
    while (count < n) {  
        number++;  
        if (isPrime(number)) {  
            count++;  
        }  
    }  
  
    return number;  
}  
  
public static void main(String[] args) {  
    int n = 10; // You can change this to test other positions  
    int result = nthPrime(n);  
    System.out.println("The " + n + "th prime number is: " + result);  
}  
}  
  
//numbers o prime in a speciic rangw  
nt countPrimesInRange(int start, int end) {  
    int count = 0;  
    for (int i = start; i <= end; i++) {  
        if (isPrime(i)) {  
            count++;  
        }  
    }  
}
```

```
    }

    return count;
}

boolean isPrime(int num) {

    if (num <= 1) return false;

    for (int i = 2; i <= Math.sqrt(num); i++) {

        if (num % i == 0) return false;
    }

    return true;
}

//all digit count

int countDigits(int num) {

    int count = 0;

    num = Math.abs(num); // To handle negative numbers

    if (num == 0) return 1;

    while (num > 0) {

        num /= 10;

        count++;
    }

    return count;
}

//unique digit count

int countUniqueDigits(int num) {

    int[] freq = new int[10];

    num = Math.abs(num);

    while (num > 0) {

        int digit = num % 10;
```

```
freq[digit] = 1;
num /= 10;
}

int count = 0;
for (int i = 0; i < 10; i++) {
    count += freq[i];
}

return count;
}

//non repeating digit count

int countNonRepeatingDigits(int num) {

    int[] freq = new int[10];
    num = Math.abs(num);

    // Count frequency of each digit
    while (num > 0) {
        int digit = num % 10;
        freq[digit]++;
        num /= 10;
    }

    // Count digits that appear exactly once
    int count = 0;
    for (int i = 0; i < 10; i++) {
        if (freq[i] == 1) {
            count++;
        }
    }
}
```

```
}

return count;

}

//digit sum :sum o all digits

public static int finalDigitSum(int num) {

    if (num == 0) return 0;

    int sign = num < 0 ? -1 : 1;
    num = Math.abs(num);

    while (num >= 10) {

        int sum = 0;
        while (num > 0) {

            sum += num % 10;
            num /= 10;
        }
        num = sum;
    }

    return num * sign;
}

//sum of even digit

public static int sumOfEvenDigits(int num) {

    num = Math.abs(num); // Handle negative numbers
    int sum = 0;

    while (num > 0) {

        int digit = num % 10;
        if (digit % 2 == 0) sum += digit;
        num /= 10;
    }

    return sum;
}
```

```
if (digit % 2 == 0) {  
    sum += digit;  
}  
  
num /= 10;  
  
}  
  
return sum;  
}  
  
//digit sum:od sum  
  
public static int sumOfOddDigits(int num) {  
    num = Math.abs(num); // Handle negative numbers  
  
    int sum = 0;  
  
    while (num > 0) {  
        int digit = num % 10;  
  
        if (digit % 2 != 0) {  
            sum += digit;  
        }  
  
        num /= 10;  
    }  
  
    return sum;  
}  
  
//digit sum:sum of even or odd digit in N  
  
public class DigitSum {  
  
    public static int digitSum(int n, String type) {  
        n = Math.abs(n); // Handle negative input  
  
        int sum = 0;
```

```

while (n > 0) {
    int digit = n % 10;
    if (type.equalsIgnoreCase("even") && digit % 2 == 0) {
        sum += digit;
    } else if (type.equalsIgnoreCase("odd") && digit % 2 != 0) {
        sum += digit;
    }
    n /= 10;
}

return sum;
}

public static void main(String[] args) {
    System.out.println("Even digit sum: " + digitSum(123456, "even")); // 2+4+6 = 12
    System.out.println("Odd digit sum: " + digitSum(123456, "odd")); // 1+3+5 = 9
}
//is palindrome?

public static boolean isPalindrome(int n) {
    int original = n;
    int reversed = 0;

    while (n > 0) {
        int digit = n % 10;
        reversed = reversed * 10 + digit;
        n /= 10;
    }
}

```

```
        return original == reversed;

//create pin using gamma,etta,

public class CreatePin {
    public static void main(String[] args) {
        String w1 = "alpha";
        String w2 = "eta";
        String w3 = "gamma";

        // Extract first letters
        char c1 = w1.charAt(0);
        char c2 = w2.charAt(0);
        char c3 = w3.charAt(0);

        // Get lengths
        int l1 = w1.length();
        int l2 = w2.length();
        int l3 = w3.length();

        // Combine to form a PIN
        String pin = "" + c1 + l1 + c2 + l2 + c3 + l3;

        System.out.println("Generated PIN: " + pin);
    }
}

// weight of hill pattern?

import java.util.Scanner;
```

```
public class HillPatternWeight {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Enter number of rows (n): ");  
        int n = sc.nextInt();  
  
        // Weight = n * n (sum of first n odd numbers)  
        int weight = n * n;  
  
        System.out.println("Weight of the hill pattern is: " + weight);  
    }  
}  
  
//return secodn word in uppercase  
import java.util.Scanner;  
  
public class SecondWordUppercase {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Enter a sentence: ");  
        String input = sc.nextLine();  
  
        String[] words = input.trim().split("\\s+");  
  
        if (words.length < 2) {  
            System.out.println("No second word found.");  
        } else {  
            System.out.println("Second word in uppercase: " + words[1].toUpperCase());  
        }  
    }  
}
```

```

    }

}

//is palindorme(String)

public static boolean isPalindrome(String str) {

    str = str.replaceAll("\s+", "").toLowerCase(); // Remove spaces and convert to
lowercase

    int left = 0, right = str.length() - 1;

    while (left < right) {

        if (str.charAt(left) != str.charAt(right)) {

            return false;
        }

        left++;
        right--;
    }

    return true;
}

//weight of string

public static int getStringWeight(String str) {

    int weight = 0;

    str = str.toLowerCase(); // Make case-insensitive

    for (char ch : str.toCharArray()) {

        if (ch >= 'a' && ch <= 'z') {

            weight += (ch - 'a' + 1); // a=1, b=2, ..., z=26
        }
    }

    return weight;
}

```

```
//most reqnet digit

import java.util.*;

public class MostFrequentDigit {

    public static void main(String[] args) {

        int number = 1223377755;

        int mostFrequent = findMostFrequentDigit(number);

        System.out.println("Most Frequent Digit: " + mostFrequent);

    }

    public static int findMostFrequentDigit(int number) {

        int[] freq = new int[10];

        number = Math.abs(number); // handle negative numbers

        while (number > 0) {

            int digit = number % 10;

            freq[digit]++;

            number /= 10;

        }

        int maxFreq = 0;

        int mostFrequentDigit = -1;

        for (int i = 0; i < 10; i++) {

            if (freq[i] > maxFreq) {

                maxFreq = freq[i];

                mostFrequentDigit = i;

            }

        }

    }

}
```

```
    return mostFrequentDigit;  
}  
}
```