Java Programming

Table of Contents

---

1. Basic Java Program

Program:

```
class add {
    public static void main(String[] args) {
        System.out.println("Welcome to java");
    }
}
```

Output:

```
Welcome to java
```

Explanation:

- `class add` defines a class named "add"
- `public static void main(String[] args)` is the main method where program execution begins
- `System.out.println()` prints text to the console

2. Command Line Arguments in Java

Program:

```
class command{
    public static void main(String args[])
    {
        for(int i=0; i<args.length;i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

Sample Execution:

```
java command Hello World Java
```

Output:

```
Hello
World
Java
```

Explanation:

- Command line arguments are passed as String array `args[]`
- `args.length` gives the number of arguments
- Loop iterates through all arguments and prints them

---

3. Comments in Java

Types of Comments:

1. **Single Line Comments**: `// This is a single line comment`
2. **Multi-line Comments**: `/* This is a multi-line comment */`

Java Overview (from comments):

Java is a programming language and computing platform first released by Sun Microsystems in 1995.

Used for:

- Mobile applications (specially Android apps)
- Desktop applications

- Web applications
- Web servers and application servers
- Games
- Database connection

Why Use Java?

- Works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- One of the most popular programming languages
- Large demand in job market
- Easy to learn and simple to use
- Open-source and free

---

4. Variables in Java

Program:

```
public class Variables {
    public static void main(String[] args) {
        String name = "Saravanan";
        int age = 25;
        float percent = 89.9f;
        char gender = 'M';
        boolean married = false;

        System.out.println("Name  :" + name);
        System.out.println("Age   : " + age);
        System.out.println("percent  : " + percent);
        System.out.println("Gender   : " + gender);
        System.out.println("Married   : " + married);
    }
}
```

Output:

```
Name  :Saravanan
Age   : 25
percent  : 89.9
Gender   : M
Married   : false
```

Data Types:

- **String**: Text data
- **int**: Integer numbers
- **float**: Decimal numbers (requires 'f' suffix)
- **char**: Single character (in single quotes)
- **boolean**: true/false values

---

5. Type Casting

Widening Casting (Automatic):

```
byte -> short -> char -> int -> long -> float -> double
```

Narrowing Casting (Manual):

```
double -> float -> long -> int -> char -> short -> byte
```

Program 1: Integer to Double (Widening)

```
class casting {
    public static void main(String[] args) {
        int a = 10;
        double b = a;
        System.out.println("Int  :" + a);
        System.out.println("Double  :" + b);
    }
}
```

Output:

```
Int  :10
Double  :10.0
```

Program 2: Double to Integer (Narrowing)

```
class casting {
    public static void main(String[] args) {
        double d = 25.5385;
        int c = (int)d;
        System.out.println("Double  :" + d);
        System.out.println("Int  :" + c);
    }
}
```

Output:

```
Double  :25.5385
Int  :25
```

---

6. Arithmetic Operators

Program:

```
class add {
    public static void main(String[] args) {
        int a = 123, b = 10;
        System.out.println("Addition : " + (a + b));
        System.out.println("Subtraction  : " + (a - b));
        System.out.println("Multiplication  : " + (a * b));
```

```
        System.out.println("Division : " + (a / b));
        System.out.println("Modulo : " + (a % b));
    }
}
```

Output:

```
Addition : 133
Subtraction  : 113
Multiplication  : 1230
Division : 12
Modulo : 3
```

Operators:

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `%` Modulo (remainder)

---

7. Assignment Operators

Operators Table:

Operator	Example	Equivalent
=	a = 123	a = 123
+=	a += 10	a = a + 10
-=	a -= 10	a = a - 10
*=	a *= 10	a = a * 10
/=	a /= 10	a = a / 10
%=	a %= 10	a = a % 10

Program:

```
class Assignment {
    public static void main(String[] args) {
        int a = 123;
        System.out.println(a);    // 123
        a += 10;
        System.out.println(a);    // 133
        a -= 10;
        System.out.println(a);    // 123
        a *= 10;
        System.out.println(a);    // 1230
        a /= 10;
        System.out.println(a);    // 123
        a %= 10;
        System.out.println(a);    // 3
```

```
    }
}
```

Output:

```
123
133
123
1230
123
3
```

---

8. Relational Operators

Operators Table:

Operator	Name	Example
==	Equal to	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Program:

```
class relational {
    public static void main(String[] args) {
        int a = 110, b = 109;
        System.out.println("Equal to :" + (a == b));
        System.out.println("Not equal :" + (a != b));
        System.out.println("Greater than :" + (a > b));
        System.out.println("Less than :" + (a < b));
        System.out.println("Greater than or equal :" + (a >= b));
        System.out.println("Less than or equal :" + (a <= b));
    }
}
```

Output:

```
Equal to :false
Not equal :true
Greater than :true
Less than :false
Greater than or equal :true
Less than or equal :false
```

---

9. Logical Operators

Program:

```
class Logical {
    public static void main(String[] args) {
        int m1 = 25, m2 = 75;
        System.out.println("AND && :" + (m1 >= 35 && m2 >= 35));
        System.out.println("OR || :" + (m1 >= 35 || m2 >= 35));
        System.out.println("NOT ! :" + (!(m1 > m2)));
    }
}
```

Output:

```
AND && :false
OR || :true
NOT ! :true
```

Logical Operators:

- `&&` (AND): Returns true if both conditions are true
- `||` (OR): Returns true if at least one condition is true
- `!` (NOT): Returns opposite of the condition

---

10. Conditional Operators (Ternary)

Program:

```
class conditional {
    public static void main(String args[]) {
        int a = 45, b = 35, c;
        c = a > b ? a : b;
        System.out.println("The Greatest Number is :" + c);
    }
}
```

Output:

```
The Greatest Number is :45
```

Syntax:

```
condition ? value_if_true : value_if_false
```

---

11. Unary Operators

Program:

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

```
        int a = 10;
        System.out.println(a);        // 10
        System.out.println(a++);      // 10 (Post increment)
        System.out.println(a);        // 11
        System.out.println(++a);      // 12 (Pre increment)
    }
}
```

Output:

```
10
10
11
12
```

Types:

- ++ (Increment): Adds 1 to variable
- -- (Decrement): Subtracts 1 from variable
- **Pre-increment/decrement**: Operation happens before value is used
- **Post-increment/decrement**: Operation happens after value is used

---

12. Bitwise & Shift Operators

Program:

```
class Bitwise {
    public static void main(String[] args) {
        int a = 10, b = 2;
        System.out.println("Bitwise AND & :" + (a & b));
        System.out.println("Bitwise OR | :" + (a | b));
        System.out.println("Bitwise XOR ^ :" + (a ^ b));
        System.out.println("Complement ~ :" + (~a));
        System.out.println("Right shift >> :" + (a >> b));
        System.out.println("Left shift << :" + (a << b));
    }
}
```

Output:

```
Bitwise AND & :2
Bitwise OR | :10
Bitwise XOR ^ :8
Complement ~ :-11
Right shift >> :2
Left shift << :40
```

Formulas:

- **Complement (~)**: -(N+1)
- **Right shift (>>)**: a / 2^b
- **Left shift (<<)**: a * 2^b

13. User Input in Java

Scanner Methods:

- `nextInt()` - Reads integer
- `nextFloat()` - Reads float
- `nextDouble()` - Reads double
- `next()` - Reads single word
- `next().charAt(0)` - Reads first character
- `nextLine()` - Reads entire line

Program:

```
import java.util.Scanner;

public class UserInputExample {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String Name;
        int Age;

        System.out.print("Enter your name :");
        Name = in.nextLine();

        System.out.print("Enter your age :");
        Age = in.nextInt();

        System.out.println("Hello, " + Name + "! You are " + Age + " years
old.");
    }
}
```

Sample Output:

```
Enter your name :John
Enter your age :25
Hello, John! You are 25 years old.
```

Algebraic Calculation Example:

```
import java.util.Scanner;
class getting_inputs {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a, b, c;
        System.out.println("Enter the Two values :");
        a = in.nextInt();
        b = in.nextInt();

        c = (a*a) + (b*b) + (2*a*b); // (a+b)²
        System.out.println("Result :" + c);
    }
}
```

Sample Output:

```
Enter the Two values :
3
4
Result :49
```

---

14. Keywords in Java

Java has 50 reserved keywords that cannot be used as identifiers:

```
abstract    assert      boolean     break       byte
case        catch       char        class       const
continue    default     do          double      else
enum        extends     final       finally     float
for         goto        if          implements  import
instanceof  int         interface   long        native
new         package     private     protected   public
return      short       static      strictfp    super
switch      synchronized this       throw       throws
transient   try         void        volatile    while
```

---

15. If Statements

Program:

```
import java.util.Scanner;
class if_statement {
    public static void main(String[] args) {
        int age;
        System.out.println("Enter your age :");
        Scanner in = new Scanner(System.in);
        age = in.nextInt();
        if(age >= 18) {
            System.out.println("You are eligible for Vote");
        }
    }
}
```

Sample Output:

```
Enter your age :
20
You are eligible for Vote
```

---

16. Nested If Statements

Program:

```
public class NestedIf {
```

```
    public static void main(String[] args) {
        int x = 30;
        int y = 10;

        if (x > y) {
            System.out.println("x is greater than y");

            if (x > 20) {
                System.out.println("x is also greater than 20");
            } else {
                System.out.println("x is not greater than 20");
            }
        } else if (x < y) {
            System.out.println("x is less than y");
        } else {
            System.out.println("x is equal to y");
        }
    }
}
```

Output:

```
x is greater than y
x is also greater than 20
```

Insurance Eligibility Example:

Problem: A company insures drivers if:

- Driver is married, OR
- Driver is unmarried, male & above 30 years, OR
- Driver is unmarried, female & above 25 years

```
import java.util.Scanner;
class Insurance {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the Marital status (M/U):");
        char Marital = in.next().charAt(0);

        if (Marital == 'u' || Marital == 'U') {
            System.out.println("Enter the Gender (M/F):");
            char Gender = in.next().charAt(0);
            System.out.println("Enter the Age:");
            int Age = in.nextInt();

            if ((Gender == 'M' || Gender == 'm') && Age >= 30) {
                System.out.println("You are Eligible for Insurance");
            } else if ((Gender == 'F' || Gender == 'f') && Age >= 25) {
                System.out.println("You are Eligible for Insurance");
            } else {
                System.out.println("You are not Eligible for insurance");
            }
        } else if (Marital == 'm' || Marital == 'M') {
            System.out.println("You are Eligible for Insurance");
        } else {
            System.out.println("Invalid Input");
        }
```

```
    }
}
```

---

17. Switch Statements

Calculator Example:

```
import java.util.Scanner;
class Calculator {
    public static void main(String[] args) {
        int ch, n1, n2;
        System.out.println("1.Addition");
        System.out.println("2.Subtraction");
        System.out.println("3.Multiplication");
        System.out.println("4.Division");
        System.out.println("Enter your choice : ");

        Scanner in = new Scanner(System.in);
        ch = in.nextInt();
        System.out.println("Enter two operands: ");
        n1 = in.nextInt();
        n2 = in.nextInt();

        switch (ch) {
            case 1:
                System.out.println("sum = " + (n1 + n2));
                break;
            case 2:
                System.out.println("sub = " + (n1 - n2));
                break;
            case 3:
                System.out.println("mul = " + (n1 * n2));
                break;
            case 4:
                System.out.println("division = " + (n1 / n2));
                break;
            default:
                System.out.println("Error! Select only from 1 to 4");
        }
    }
}
```

Sample Output:

```
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice :
1
Enter two operands:
15
25
sum = 40
```

Grouped Switch - Vowel Checker:

```
class VowelChecker {
    public static void main(String args[]) {
        char ch = 'E';
        switch (ch) {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U':
                System.out.println("It is a vowel");
                break;
            default:
                System.out.println("It is not a vowel");
        }
    }
}
```

Output:

```
It is a vowel
```

---

18. Loops in Java

While Loop:

```
import java.util.Scanner;
class while_loop {
    public static void main(String[] args) {
        System.out.println("Enter the Limit :");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int i = 1;
        while(i <= n) {
            System.out.println(i);
            i++;
        }
    }
}
```

Sample Output (n=5):

```
Enter the Limit :
5
1
2
3
4
5
```

Do-While Loop:

```
import java.util.Scanner;
class do_while {
    public static void main(String[] args) {
        System.out.println("Enter the Limit :");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int i = 1;
        do {
            System.out.println(i);
            i++;
        } while(i <= n);
    }
}
```

For Loop:

```
import java.util.Scanner;
class for_loop {
    public static void main(String[] args) {
        System.out.println("Enter the Limit :");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        for(int i = 1; i <= n; i++) {
            System.out.println(i);
        }
    }
}
```

---

19. Enhanced For Loop (For-Each)

Program:

```
class Enhanced_for {
    public static void main(String[] args) {
        int number[] = {10, 20, 30, 40, 50};
        for(int n : number) {
            System.out.println(n);
        }
    }
}
```

Output:

```
10
20
30
40
50
```

Explanation:

- Enhanced for loop iterates through array elements directly
- Syntax: `for(datatype variable : array)`
- No need for index management

20. Nested Loops

Star Pattern Program:

```
class Nested_forloop {
    public static void main(String[] args) {
        int i, j;
        for(i = 1; i <= 5; i++) {
            for(j = 1; j <= i; j++) {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

Output:

```
*
**
***
****
*****
```

Explanation:

- Outer loop controls number of rows
- Inner loop controls number of stars in each row
- Creates a triangular pattern

---

21. Break and Continue

Break Statement:

```
class break_example {
    public static void main(String[] args) {
        int i;
        for(i = 1; i <= 10; i++) {
            System.out.println(i);
            if (i == 8)
                break;
        }
    }
}
```

Output:

```
1
2
```

```
3
4
5
6
7
8
```

Break and Continue Together:

```
class break_continue {
    public static void main(String[] args) {
        int i;
        for(i = 1; i <= 10; i++) {
            if(i == 5)
                continue;
            System.out.println(i);
            if (i == 8)
                break;
        }
    }
}
```

Output:

```
1
2
3
4
6
7
8
```

Explanation:

- **break**: Terminates the loop completely
- **continue**: Skips current iteration and moves to next iteration
- In the example: 5 is skipped (continue), loop ends at 8 (break)

Java Pattern Programming

---

3. Pattern Programming Overview

Pattern programming improves **Analytical and Logical Thinking** skills. It involves:

- Understanding loop structures
- Managing nested loops
- Calculating spaces and characters
- Developing problem-solving approaches

Basic Pattern Structure:

```
for (outer_loop) {          // Controls rows
    for (inner_loop) {      // Controls columns/characters
        // Print logic
    }
    System.out.println();   // New line after each row
}
```

---

4. Pattern 1: Simple Star Triangle

Program:

```
class StarTriangle {
    public static void main(String[] args) {
        int i, j;
        for (i = 0; i <= 5; i++) {
            for (j = 0; j <= i; j++) {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

Output:

```
*
**
***
****
*****
******
```

Logic Explanation:

- **Outer loop (i)**: Controls number of rows (0 to 5 = 6 rows)
- **Inner loop (j)**: Prints stars from 0 to i (increasing pattern)
- **Row 0**: 1 star, **Row 1**: 2 stars, **Row 2**: 3 stars, etc.

---

5. Pattern 2: Number Repetition Triangle

Program:

```
import java.util.Scanner;
class NumberRepetition {
    public static void main(String[] args) {
        int i, j, rows;
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
```

```
        rows = in.nextInt();

        for (i = 0; i <= rows; i++) {
            for (j = 0; j <= i; j++) {
                System.out.print(i);
            }
            System.out.println("");
        }
    }
}
```

Sample Input/Output:

```
Enter the number of rows :
5

0
11
222
3333
44444
555555
```

Logic Explanation:

- **Outer loop (i)**: Row number (also the digit to print)
- **Inner loop (j)**: Repeats the row number i times
- **Row 0**: Print '0' once, **Row 1**: Print '1' twice, etc.

---

6. Pattern 3: Right-Aligned Star Triangle

Program:

```
import java.util.Scanner;
class RightAlignedTriangle {
    public static void main(String[] args) {
        int i, j, n;
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
        n = in.nextInt();

        for (i = 0; i <= n; i++) {
            for (j = 0; j <= n; j++) {
                if ((i + j) < n)
                    System.out.print(" ");
                else
                    System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

Sample Input/Output:

```
Enter the number of rows :
5
     *
    **
   ***
  ****
 *****
******
```

Logic Explanation:

- **Condition**: `(i + j) < n` determines spaces vs stars
- **Early positions**: Print spaces for right alignment
- **Later positions**: Print stars
- Creates right-aligned triangular pattern

---

7. Pattern 4: Inverted Star Triangle

Program:

```
import java.util.Scanner;
class InvertedTriangle {
    public static void main(String[] args) {
        int i, j, n;
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
        n = in.nextInt();

        for (i = 1; i <= n; i++) {
            for (j = n; j >= i; j--) {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

Sample Input/Output:

```
Enter the number of rows :
5
*****
****
***
**
*
```

Logic Explanation:

- **Outer loop (i)**: Goes from 1 to n
- **Inner loop (j)**: Decreases from n to i
- **Row 1**: n stars, **Row 2**: n-1 stars, decreasing pattern

8. Pattern 5: Sequential Number Triangle

Program:

```
import java.util.Scanner;
class SequentialNumbers {
    public static void main(String[] args) {
        int rows, i, j;
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
        rows = in.nextInt();

        for (i = 1; i <= rows; i++) {
            for (j = 1; j <= i; j++) {
                System.out.print(j);
            }
            System.out.println("");
        }
    }
}
```

Sample Input/Output:

```
Enter the number of rows :
5
1
12
123
1234
12345
```

Logic Explanation:

- **Outer loop (i)**: Controls rows (1 to rows)
- **Inner loop (j)**: Prints numbers from 1 to i
- **Row 1**: "1", **Row 2**: "12", **Row 3**: "123", etc.

---

9. Pattern 6: Continuous Number Triangle

Program:

```
class ContinuousNumbers {
    public static void main(String[] args) {
        int number = 1, i, j;

        for (i = 1; i <= 5; i++) {
            for (j = 1; j <= i; j++) {
                System.out.print(" " + number);
                number++;
            }
```

```
            System.out.println("");
        }
    }
}
```

Output:

```
 1
 2 3
 4 5 6
 7 8 9 10
 11 12 13 14 15
```

Logic Explanation:

- **Global variable**: `number` starts at 1 and increments continuously
- **Outer loop**: Controls rows
- **Inner loop**: Prints consecutive numbers
- Numbers continue across rows (not reset)

---

10. Pattern 7: Centered Number Triangle

Program:

```
import java.util.Scanner;
class CenteredTriangle {
    public static void main(String[] args) {
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();

        for (int i = 1; i <= n; i++) {
            // Print leading spaces for centering
            for (int k = 1; k <= (n - i); k++) {
                System.out.print(" ");
            }
            // Print numbers
            for (j = 1; j <= i; j++) {
                System.out.print(" " + i);
            }
            System.out.println("");
        }
    }
}
```

Sample Input/Output:

```
Enter the number of rows :
5
    1
   2 2
  3 3 3
 4 4 4 4
```

```
5 5 5 5 5
```

Logic Explanation:

- **First inner loop**: Prints `(rows - i)` spaces for centering
- **Second inner loop**: Prints row number `i` times
- Creates centered/pyramid effect

---

11. Pattern 8: Inverted Number Pyramid

Program:

```
import java.util.Scanner;
class InvertedPyramid {
    public static void main(String[] args) {
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();

        for (i = n; i >= 1; i--) {
            // Print leading spaces
            for (int k = 1; k <= (n - i); k++) {
                System.out.print(" ");
            }
            // Print numbers (2*i-1 times for pyramid effect)
            for (int j = 1; j <= 2 * i - 1; j++) {
                System.out.print(i);
            }

            System.out.println("");
        }
    }
}
```

Sample Input/Output:

```
Enter the number of rows :
5
555555555
 4444444
  33333
   222
    1
```

Logic Explanation:

- **Outer loop**: Starts from `rows` down to 1 (inverted)
- **Spaces**: Increase as we go down `(rows - i)`
- **Numbers**: Print `2*i-1` times for pyramid shape
- Creates inverted pyramid with decreasing width

Benefits of Pattern Programming

Skill Development:

1. **Logical Thinking**: Breaking complex problems into simple steps
2. **Loop Mastery**: Understanding nested loop behavior
3. **Mathematical Skills**: Using formulas for spaces and characters
4. **Debugging Skills**: Identifying and fixing pattern issues
5. **Code Optimization**: Writing efficient and clean code

Real-World Applications:

- **Graphics Programming**: Creating visual elements
- **Game Development**: Drawing game boards and elements
- **Data Visualization**: Creating charts and graphs
- **Algorithm Design**: Understanding recursive and iterative patterns
- **User Interface**: Designing text-based layouts

---

1. ASCII (American Standard Code for Information Interchange)

Overview:

ASCII is a character encoding standard that represents text in computers and communication equipment.

Key Information:

- **Total ASCII Characters**: 256 (0 to 255)
- **Standard ASCII**: 128 characters (0 to 127) - most commonly used
- **Extended ASCII**: 128 to 255

Important ASCII Values:

```
48-57   → 0-9 (Digits)
65-90   → A-Z (Uppercase Letters)
97-122  → a-z (Lowercase Letters)
32      → Space character
```

```
import java.util.Scanner;
class vote {
    public static void main(String[] args) {
        int letter = 65, i, j,rows;
        System.out.println("Enter the number of rows : ");
        Scanner in = new Scanner(System.in);
        rows = in.nextInt();
        for (i = 0; i <= rows; i++) {
            for (j = 0; j <= i; j++) {
                System.out.print((char)(letter + j)+" ");
```

```
            }
            System.out.println("");
        }
    }
}
output:
Enter the number of rows :
5
A
A B
A B C
A B C D
A B C D E
A B C D E F
```

ASCII Conversion Formula:

- **Uppercase to Lowercase**: Add 32
- **Lowercase to Uppercase**: Subtract 32
- Example: 'A' (65) + 32 = 'a' (97)

---

2. ASCII Character Display Program

Program:

```
import java.util.Scanner;
class ASCIIDisplay {
    public static void main(String[] args) {
        char a = 'z';

        // Convert lowercase 'z' to uppercase 'Z'
        System.out.print((a) + ":" + (char)(a - 32));
    }
}
```

Sample Output (Partial):

```
z:Z
```

Explanation:

- `a - 32` converts lowercase 'z' to uppercase 'Z'

Character Manipulation:

```
// Convert case
char upper = (char)(lowercase - 32);
char lower = (char)(uppercase + 32);
```

```
// Check character type
if (ch >= 'A' && ch <= 'Z') // Uppercase
if (ch >= 'a' && ch <= 'z') // Lowercase
if (ch >= '0' && ch <= '9') // Digit
```

//Factorial number

```
import java.util.Scanner;
class Factorialnum {
   public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      int i,f,n;
      System.out.print("Enter the number :");
      n = in.nextInt();
      f = 1;
      for(i=1 ;i<=n ;i++){
         f = f*i;
      }
      System.out.println("Factorial of"+n+" is :"+f);
   }
}
```

Output:

```
Enter the number :5
Factorial of5 is :120
```

//Fibonaccci

```
import java.util.Scanner;
class Fibonaccci {
   public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      int a = -1, b = 1 , i,c,n;
      System.out.print("Enter the number :");
      n = in.nextInt();//5
      for (i=1 ;i<=n ;i++){//1<=5 2<=5 3<=5  4<=5 5<=5
```

```
        c = a+b;//-1+1=0  1+0=1 0+1=1 1+1=2 1+2=3
        System.out.print(c+" ");//0 1 1 2 3
        a = b;//1 0 1 1 2
        b = c;//0 1 1 2 3
      }
   }
}
```

Output:

Enter the number :5

0 1 1 2 3

//primenum

```
import java.util.Scanner;
class primenum {
   public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      int n,f=0,i;
      System.out.print("Enter the number :");
      n = in.nextInt();
      for (i=1 ;i<=n ;i++){
         if(n % i==0){
            f++;
         }
      }
      if(f == 2){
         System.out.println(n+"is a Prime number"); }
         else{
            System.out.println(n+"It's not a Prime number"); }
}
}
```

Output:

Enter the number :7

7is a Prime number

//Primenum 1to 100

```
class primenum {
   public static void main(String[] args){
      int n,f=0,i;
      for( n=1; n<=100 ;n++){
      for (i=1 ;i<=n ;i++){
         if(n % i==0){
            f++;
         }
      }
      if(f == 2){
         System.out.println(n+"is a Prime number");
      }
      f = 0;
      }
   }
}
```

Output:

2is a Prime number

3is a Prime number

5is a Prime number

7is a Prime number

11is a Prime number

13is a Prime number

17is a Prime number

19is a Prime number

23is a Prime number

29is a Prime number

31is a Prime number

37is a Prime number

41is a Prime number

43is a Prime number

47is a Prime number

53is a Prime number

59is a Prime number

61is a Prime number

67is a Prime number

71is a Prime number

73is a Prime number

79is a Prime number

83is a Prime number

89is a Prime number

97is a Prime number

```
import java.util.Scanner;
class perfactnum {
   public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      int n,sum=0,i;
      System.out.print("Enter the number :");
      n = in.nextInt();
      for(i=1 ;i<n ;i++){
         if(n%i==0){
         sum += i;
         }
      }
      if(sum == n){
         System.out.print(n+"is a perfect number :");
          }
          else{
```

```
            System.out.print(n+"It's not a perfect number :");
        }}}
```

Java Methods

---

1. Introduction to Methods

A **method** in Java is a block of code that performs a specific task. Methods are used to:

- Make code reusable
- Organize code into logical units
- Make programs easier to read and maintain
- Reduce code duplication

Method Syntax:

```
[access_modifier] [static] return_type method_name(parameter_list) {
    // Method body
    return value; // if return_type is not void
}
```

---

6. Static vs Non-Static Methods

Static Methods

- Belong to the class, not to any instance
- Can be called without creating an object
- Cannot access non-static variables directly

Non-Static Methods

- Belong to instances of the class
- Require object creation to be called
- Can access both static and non-static variables

```
class MethodTypes {
    static int staticVar = 100;
    int nonStaticVar = 200;

    // Static method
    public static void staticMethod() {
        System.out.println("Static method called");
        System.out.println("Static variable: " + staticVar);
        // System.out.println(nonStaticVar); // Error: Cannot access non-
static
```

```
    }

    // Non-static method
    public void nonStaticMethod() {
        System.out.println("Non-static method called");
        System.out.println("Static variable: " + staticVar);      // OK
        System.out.println("Non-static variable: " + nonStaticVar); // OK
    }
}

public class MethodTypeExample {
    public static void main(String[] args) {
        // Calling static method - no object needed
        MethodTypes.staticMethod();

        // Calling non-static method - object needed
        MethodTypes obj = new MethodTypes();
        obj.nonStaticMethod();
    }
}
```

2. Types of User-Defined Methods

There are **4 main types** of user-defined methods in Java:

Type 1: No Return Type, No Parameters

- **Return Type**: `void` (returns nothing)
- **Parameters**: No parameters
- **Usage**: Performs a task without taking input or returning output

Type 2: No Return Type, With Parameters

- **Return Type**: `void` (returns nothing)
- **Parameters**: Takes input parameters
- **Usage**: Performs a task using provided input

Type 3: With Return Type, No Parameters

- **Return Type**: Returns a value (int, float, String, etc.)
- **Parameters**: No parameters
- **Usage**: Performs a task and returns a result

Type 4: With Return Type, With Parameters

- **Return Type**: Returns a value
- **Parameters**: Takes input parameters
- **Usage**: Most flexible - takes input and returns output

---

3. Complete Program Examples

Program 1: Basic Method (Type 1)

```
//Type of User Define Methods in java
class Methods
{
    //No Return without arguments
    public void add()
    {
        int a = 123;
        int b = 10;
        System.out.println("Addition : "+(a+b));
    }
}

class function {
    public static void main(String[] args) {
        Methods o = new Methods();
        o.add();
    }
}
```

Output:

```
Addition : 133
```

Explanation:

- `add()` method has `void` return type (no return)
- Takes no parameters
- Creates object `o` to call the method
- Method performs addition and prints result directly

Program 2: All Four Types of Methods

```
//Type of User Define Methods in java
class Methods
{
    //Type 1: No Return without arguments
    public void add()
    {
        int a = 123;
        int b = 10;
        System.out.println("Addition : "+(a+b));
    }

    //Type 2: No Return with Arguments
    public void sub(int x, int y){
        System.out.println("Subtraction : "+(x - y));
    }

    //Type 3: Return Without Arguments
    public int mul(){
```

```
        int a = 123;
        int b = 10;
        return a * b;
    }

    //Type 4: Return With Arguments
    public float div(int x, int y){
        return (float)x / y;  // Cast to float for decimal result
    }
}

class function {
    public static void main(String[] args) {
        Methods o = new Methods();

        // Calling Type 1 method
        o.add();

        // Calling Type 2 method
        o.sub(123, 10);

        // Calling Type 3 method
        System.out.println("Mul : " + o.mul());

        // Calling Type 4 method
        System.out.println("Division : " + o.div(123, 10));
    }
}
```

Output:

```
Addition : 133
Subtraction : 113
Mul : 1230
Division : 12.3
```

Detailed Explanation:

1. **add() Method (Type 1)**:
    - No parameters, no return value
    - Performs addition of hardcoded values
    - Prints result directly
2. **sub(int x, int y) Method (Type 2)**:
    - Takes two integer parameters
    - No return value (void)
    - Performs subtraction and prints result
3. **mul() Method (Type 3)**:
    - No parameters
    - Returns an integer value
    - Must use `return` statement
    - Result is captured and printed in main method
4. **div(int x, int y) Method (Type 4)**:
    - Takes two parameters
    - Returns a float value
    - Most flexible type of method

4. Built-in Methods in Java

Java provides many built-in methods. Here are some commonly used ones:

String Methods

```
public class StringMethods {
    public static void main(String[] args) {
        String str = "Hello World";

        System.out.println("Length: " + str.length());           // 11
        System.out.println("Uppercase: " + str.toUpperCase());   // HELLO
WORLD
        System.out.println("Lowercase: " + str.toLowerCase());   // hello
world
        System.out.println("Substring: " + str.substring(0, 5)); // Hello
        System.out.println("Replace: " + str.replace("World", "Java")); //
Hello Java
        System.out.println("Contains: " + str.contains("Hello")); // true
    }
}
```

Math Methods

```
public class MathMethods {
    public static void main(String[] args) {
        System.out.println("Max: " + Math.max(10, 20));        // 20
        System.out.println("Min: " + Math.min(10, 20));        // 10
        System.out.println("Absolute: " + Math.abs(-15));       // 15
        System.out.println("Square root: " + Math.sqrt(16));    // 4.0
        System.out.println("Power: " + Math.pow(2, 3));        // 8.0
        System.out.println("Random: " + Math.random());        // Random
decimal
    }
}
```

Array Methods

```
import java.util.Arrays;

public class ArrayMethods {
    public static void main(String[] args) {
        int[] arr = {5, 2, 8, 1, 9};

        System.out.println("Original: " + Arrays.toString(arr));

        Arrays.sort(arr);
        System.out.println("Sorted: " + Arrays.toString(arr));

        int index = Arrays.binarySearch(arr, 8);
        System.out.println("Index of 8: " + index);
    }
}
```

5. Method Overloading

Method overloading allows multiple methods with the same name but different parameters.

```
class Calculator {
    // Method 1: Two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method 2: Three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method 3: Two doubles
    public double add(double a, double b) {
        return a + b;
    }

    // Method 4: Two strings
    public String add(String a, String b) {
        return a + b;
    }
}

public class OverloadingExample {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println("Two ints: " + calc.add(5, 3));           // 8
        System.out.println("Three ints: " + calc.add(5, 3, 2));      // 10
        System.out.println("Two doubles: " + calc.add(5.5, 3.3));    // 8.8
        System.out.println("Two strings: " + calc.add("Hello", "World"));
// HelloWorld
    }
}
```

---

7. Advanced Method Concepts

Recursive Methods

Methods that call themselves:

```
class RecursiveExample {
    public static int factorial(int n) {
        if (n <= 1) {
            return 1;
        } else {
            return n * factorial(n - 1);
        }
    }
    public static void main(String[] args) {
        System.out.println("Factorial of 5: " + factorial(5)); // 120
    }
```

```
}
```

Variable Arguments (Varargs)

Methods that accept variable number of arguments:

```
class VarargsExample {
    public static int sum(int... numbers) {
        int total = 0;
        for (int num : numbers) {
            total += num;
        }
        return total;
    }

    public static void main(String[] args) {
        System.out.println("Sum of 2 numbers: " + sum(5, 3));         // 8
        System.out.println("Sum of 4 numbers: " + sum(1, 2, 3, 4));  // 10
    }
}
```

Methods in Java are essential for:

- **Code organization**: Breaking large programs into manageable pieces
- **Reusability**: Writing code once and using it multiple times
- **Maintainability**: Making code easier to update and debug
- **Readability**: Making programs easier to understand

The four types of user-defined methods provide flexibility in how you structure your programs, while built-in methods save time by providing pre-written functionality for common tasks.