Java

what is programing language?

A **programming language** is a formal language used to give instructions to a computer. It allows humans to write code that a computer can understand and execute to perform specific tasks like calculations, data processing, web development, and more.

There are two main types of programming languages:

1. **Low-level languages**
    - Close to machine code
    - readable , executable , understandable
    - Example: Binary (0, 1)
2. **High-level languages**
    - Easier for humans to read , write , understandable , executable
    - Examples: Python, Java, C++, JavaScript
3. **Mid level languages**

which can be readable , undersatndable by humans and computers

example : assembly lang

Mnemonics:

add , mov , sub

Assembler :

converts Mid level lang to Low level lang

Example in Python (high-level):

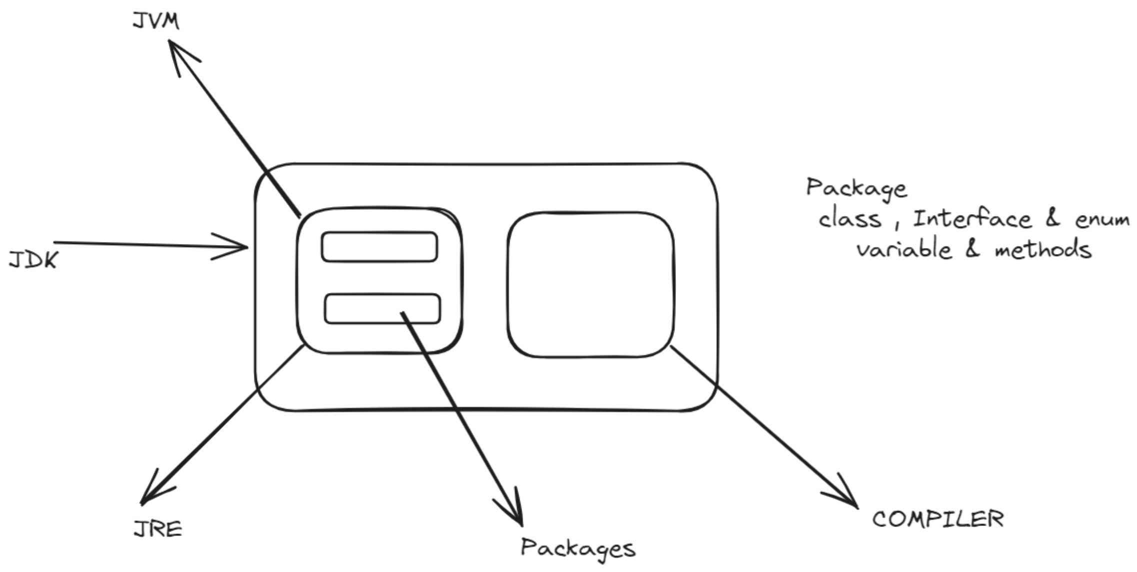
```
print("Hello, world!")
```

This line tells the computer to display the text “Hello, world!” on the screen.

Features of java :

1. provides high security .
2. has rich in-built libraries .
3. robust .
4. platform independent .
5. high level language .

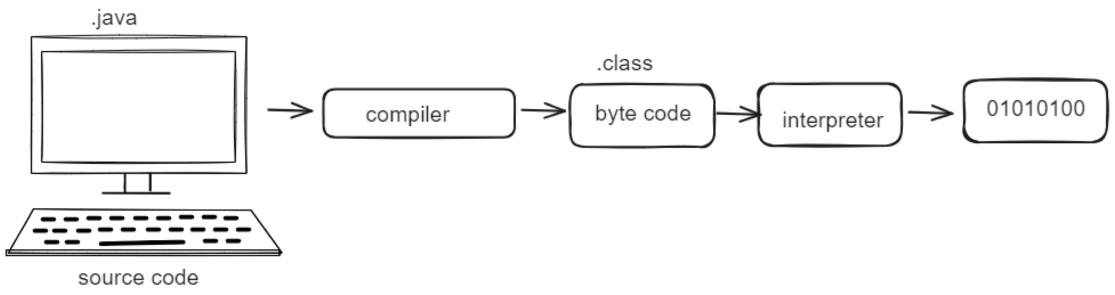
JDK



Important components of Java:

- class

- Interface
- Enums



Tokens :

- smallest unit of any programming language
- **keywords** - predefined words in java which design to perform specific taks

Here are some common keywords in Java:

```
// Control Flow Keywords
if
else
while
for
switch
break
continue
return

// Data Type Keywords
int
double
boolean
char
void

// Object-Oriented Keywords
```

```
class
public
private
protected
static
new
this
extends
implements

// Exception Handling
try
catch
finally
throw
```

These keywords are reserved words that have special meaning in Java and cannot be used as identifiers.

- **Literals** ->

In programming, **literals** are **fixed values** written directly in the code that represent data.

Common Types of Literals:

1. **Numeric Literals**
    - Integer: `10`, `5`
    - Float: `3.14`, `0.001`
2. **String Literals**
    - Text inside quotes: `"Hello"`, `'World'`
3. **Boolean Literals**
    - `True`, `False`
4. **Character Literals** (used in some languages like C, Java)
    - `'A'`, `'9'`
5. **Null or None Literals**

- Python: `None`
- JavaScript: `null`
- Java: `null`

Examples in Python:

```
python
CopyEdit
x = 10                # 10 is a numeric literal
name = "Alice"        # "Alice" is a string literal
is_valid = True       # True is a boolean literal
value = None          # None is a null literal
```

Literals are the **actual data values** assigned to variables.

- **value / data**
- **Identifiers ->**

An **identifier** in programming is the **name used to identify** variables, functions, classes, objects, and other user-defined elements.

Examples of Identifiers:

```
name = "Alice"           # 'name' is an identifier
def greet():             # 'greet' is an identifier
class Student:           # 'Student' is an identifier
```

Rules for Identifiers:

1. Can contain **letters (A-Z, a-z)**, **digits (0-9)**, and **underscores (_)**.
2. **Must not begin with a digit** (e.g., `1name` is invalid).
3. **Cannot use keywords** (e.g., `if`, `while`, `class`, etc.).
4. Are **case-sensitive** (`Name` and `name` are different).

Identifiers help you label and access different parts of your program in a meaningful way.

Java Program to Find Factors of a Number

```
import java.util.Scanner;

public class FactorFinder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        System.out.println("Factors of " + num + " are:");

        for (int i = 1; i <= num; i++) {
            if (num % i == 0) {
                System.out.println(i);
            }
        }
    }
}
```

Operators :

- in java we have predefined symbols which are understandable by all the software in the JRE.
- operations done on operands .
- **Operands** : are thoes on which operators perform operations (data or literals) .

Characterstics of operators :

1. will return value and type of the data .

2. an operator has precedence (priority) .
3. assosiativity .(direction of executaion either from left to right or right to left) .

    Q. 10+2+2*10

    approch 1: L → R : 10+2 =12 +2=14 *10 =140

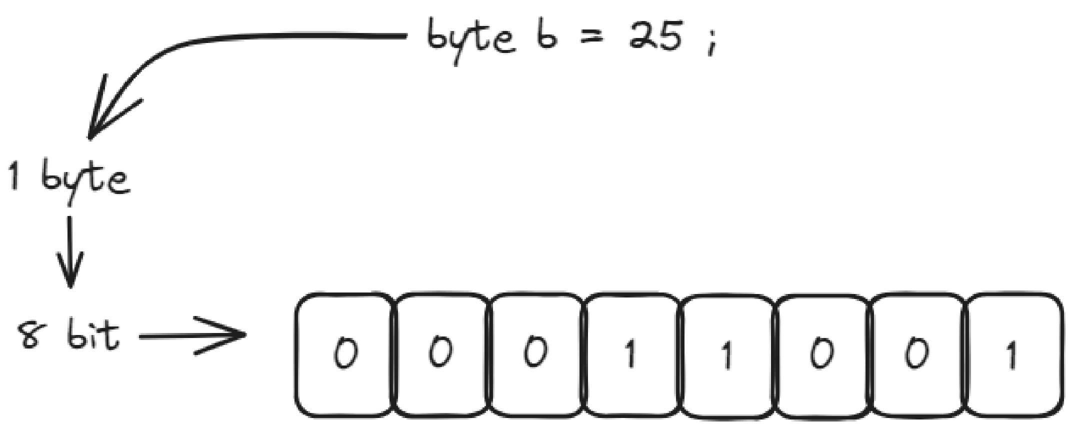
    approch 2 : R→ L : 2*10 =20 +2=22+10=32

Data types :

- in java we have 8 pre defined data types (primitive data types) which follows the literals .

type	data type	default value	size	range
num	byte	0	1 byte	
num	short	0	2 byte	
num	int	0	4 byte	
num	long	0	8 byte	
num	float	0 of	4 byte	
num	double	0 od	8 byte	
character	char	\ u 0000 (null character)	2 byte	
bool	Boolean	Flase	1 byte	

Conversion from decimal to binaryj



Xylem , phloem Problem:

Given a 4-digit number, check if the sum of the first and last digits is equal to the sum of the middle two digits.

For example, for the number `1234`:

- First digit = `1`
- Last digit = `4`
- Middle digits = `2` and `3`

Sum of first and last digits: `1 + 4 = 5`

Sum of middle digits: `2 + 3 = 5`

Since both sums are equal, this is a valid case.

Java Code Solution:

```
import java.util.Scanner;

public class XylemPhloemProblem {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Input the 4-digit number
        System.out.print("Enter a 4-digit number: ");
```

}

```java
        int num = scanner.nextInt();

        // Check if the input is a 4-digit number
        if (num < 1000 || num > 9999) {
            System.out.println("Please enter a valid 4-digit number.");
        } else {
            // Extract the digits
            int firstDigit = num / 1000;  // First digit
            int lastDigit = num % 10;     // Last digit
            int secondDigit = (num / 100) % 10; // Second digit
            int thirdDigit = (num / 10) % 10;  // Third digit

            // Calculate the sum of first and last digits, and the sum of middle digits
            int sumFirstLast = firstDigit + lastDigit;
            int sumMiddle = secondDigit + thirdDigit;

            // Check if the sums are equal
            if (sumFirstLast == sumMiddle) {
                System.out.println("The sum of the first and last digits equals the sum of the middle digits.");
            } else {
                System.out.println("The sum of the first and last digits does not equal the sum of the middle digits.");
            }
        }

        // Close the scanner object
        scanner.close();
    }
}
```

Explanation:

1. The program first reads a 4-digit number from the user.
2. It checks whether the number is valid (i.e., between 1000 and 9999).
3. It then extracts the first digit, second digit, third digit, and last digit using integer division and modulo operations.
4. The sums of the first and last digits, and the middle two digits, are calculated.
5. Finally, it compares the two sums and prints a message based on whether they are equal.

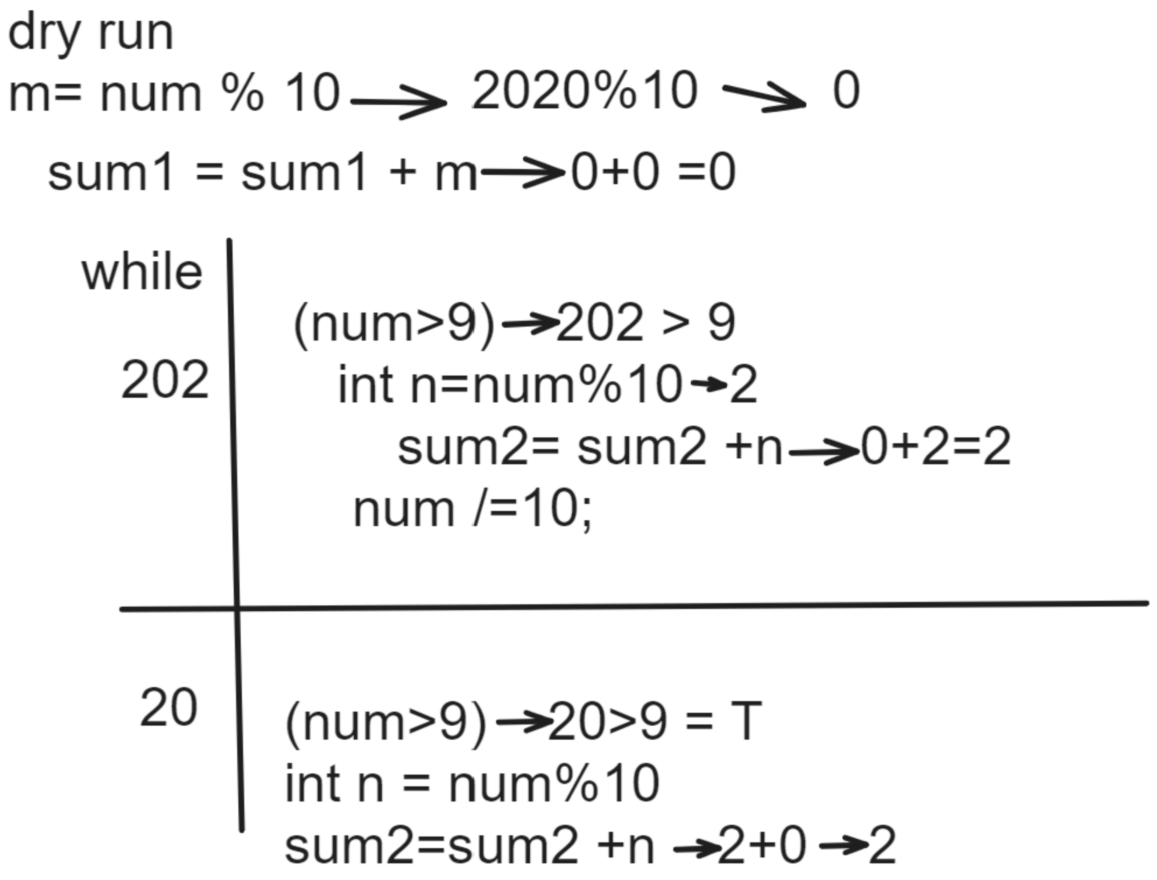
Example Execution:

```
Enter a 4-digit number: 1234
The sum of the first and last digits equals the sum of the
middle digits.

Enter a 4-digit number: 4321
The sum of the first and last digits does not equal the sum
of the middle digits.
```

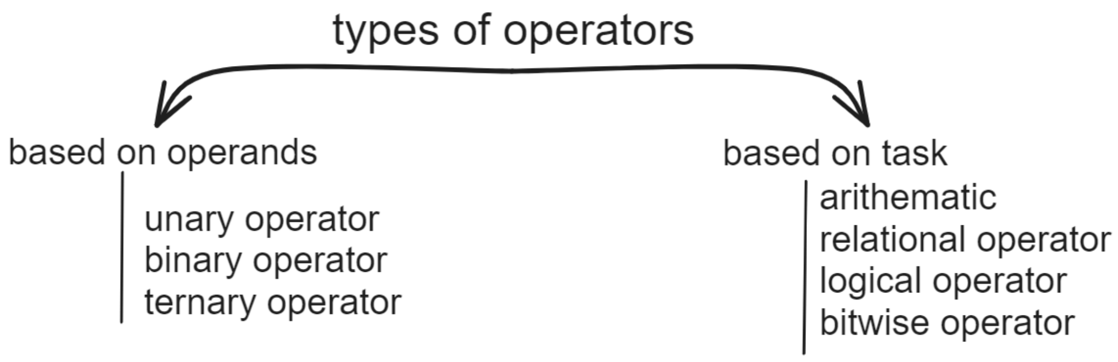
second method

```
public static void main (String args[]){
       int num = 2020 , sum1 = 0 , sum2 = 0 ;
       int m = num%10;
       sum1+=m;
       num/=10;
       while(num>9) {
          int n = nu%10;
          sum2+=n;
          num/=10;
          }
          sum1+=num;
```

```
           if (sum1 == sum2){
                 System.out.println("The sum of the first a
nd last digits equals the sum of the middle digits.");
                 }
           else{
               System.out.println("The sum of the first and
last digits not equals the sum of the middle digits.");

               }
          }
```



Types of operators :



Arithmetic operator :

- **+** —> addition ——→ polymorphic —| addition and concatenation
- - —> sub
- * –> mul
- % ——> modulo(reminder)
- / ——> div (quotient)

```
char = 'a' , b = 29 ;
string s = 'java' ;
System.out.println(a+s);
System.out.println(a+b);



o/p -----> ajava
    -----> 126 ----> a= asci value of a is 97 + 29 = 126


    char a='a' ,b='a' ;
    system.out.println(a+b);

    op: 97+97 =194
    char a='a';
    string s="java";
    boolean c = true ;
    system.out.println(2*10+s+a+10+s+c);


    op:  20javaa10javatrue ----> 2*10 =20 + java = 20java +
a = 20javaa + 10 = 20javaa10 + java = 20javaa10java + true
= 20javaa10javatrue
```

% : gives us reminder

/ : gives us qutiont

Relational operator : compare b/w two variables

- >
- <
- ≥
- ≤
- ==
- ≠

Assignment operator (=) :

assign the data to the variable

- int i=10 ;

Compound assignment operator:

combination of arithmetic and assignment operator

- +=
- -=
- *=
- /=
- %=

increment & decrement :

- increasing / decreasing the value a variable by 1
- pre increment / decrement & post increment / decrement
- pre : [symbol]variable
- post: variable[symbol]

Conditional operator :

syntax : condition ? statement 1 : statement 2 ;

when ever condition is True statement 1 will execute

when ever condition is False statement 2 will execute

```
class Main{
    public static void main(String args[]){
        int i = 10 ;
        int j=5;
        boolean c=i>j;
        int res = c?(i+j):(i-j);
        System.out.println(res);

    }

}
```

Logical operators :

- and

- or
- not

Logical operators are used to combine or manipulate boolean values (`true` or `false`). They are commonly used in programming and logic-based expressions. The three main logical operators are:

1. AND (`&&`)

- Returns `true` if **both** operands are `true`.
- Example:

  `true && true` → `true`

  `true && false` → `false`

2. OR (`||`)

- Returns `true` if **at least one** operand is `true`.
- Example:

  `true || false` → `true`

  `false || false` → `false`

3. NOT (`!`)

- Reverses the boolean value.
- Example:

  `!true` → `false`

  `!false` → `true`

XOR (Exclusive OR) is a logical operator that returns:

- `true` **only if exactly one** of the operands is `true`
- `false` if both operands are the same (either both `true` or both `false`)

Truth Table for XOR:

A	B	A ⊕ B (XOR)
true	true	false
true	false	true
false	true	true
false	false	false

Bit - wise operator :

Bitwise Operators perform operations bit by bit on binary representations of integers. They're useful in low-level programming, embedded systems, graphics, encryption, etc.

Here are the main **bitwise operators**:

Operator	Name	Description	Example (`a = 5`, `b = 3`)
`&`	AND	1 if both bits are 1	`5 & 3` → `1`
`	`	OR	1 if at least one bit is 1
`^`	XOR	1 if **only one** bit is 1	`5 ^ 3` → `6`
`~`	NOT	Inverts all bits (1's complement)	`~5` → `-6`
`<<`	Left Shift	Shifts bits to the left, adding zeros on the right	`5 << 1` → `10`
`>>`	Right Shift	Shifts bits to the right, keeping the sign bit	`5 >> 1` → `2`
`>>>`	Unsigned Right Shift	Shifts bits right, filling zeros, ignores sign (JavaScript only)	`-5 >>> 1` → large positive

---

Binary Example

Let’s take:

- `a = 5` → `0101` (binary)

- `b = 3` → `0011` (binary)

Operations:

- `a & b` → `0001` → `1`
- `a | b` → `0111` → `7`
- `a ^ b` → `0110` → `6`
- `~a` → `1010` (in 2's complement: `6`)
- `a << 1` → `1010` → `10`
- `a >> 1` → `0010` → `2`

Control flow statement :

Control flow statements determine **the order in which code is executed** in a program. They allow your code to make decisions, repeat tasks, or jump to specific parts.

]

🔹 1. Conditional Statements

✅ `if`, `else if`, `else`

```java
int age = 20;

if (age < 18) {
    System.out.println("Minor");
} else if (age < 60) {
    System.out.println("Adult");
} else {
    System.out.println("Senior");
}
```

✅ `switch`

```
int day = 2;

switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    default:
        System.out.println("Another day");
}
```

🔹 2. Looping Statements

✅ `for` loop

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

✅ `while` loop

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

✅ `do...while` loop

```
int i = 0;
do {
    System.out.println(i);
    i++;
} while (i < 5);
```

🔹 3. Jump Statements

✅ `break`

Used to **exit** a loop or `switch` early.

```
for (int i = 0; i < 5; i++) {
    if (i == 3) break;
    System.out.println(i); // Prints 0, 1, 2
}
```

✅ `continue`

Used to **skip** the current iteration.

```
for (int i = 0; i < 5; i++) {
    if (i == 2) continue;
    System.out.println(i); // Prints 0, 1, 3, 4
}
```

✅ `return`

Used to **exit a method**.

```
public static void greet() {
    System.out.println("Hello");
    return;
}
```

Number problem :

```
import java.util.*;
class Main{
    public static void main(String args[]){
        Scanner sc =new Scanner(System.in);
        int i;
        System.out.println("enter the number = ");
        i=sc.nextInt();
        if(i>0){
            if((i%2)==0){
                System.out.println("the number is positive
and even");

            }
            else{
                System.out.println("the number is positive
and odd");
            }
        }
        else if(i==0){
            System.out.println("the given number is 0");
        }
        else{
            if((i%2)==0){
                System.out.println("the number is negative
and even");
            }
            else{
                System.out.println("the number is negative
and odd");
            }
        }
    }
```

program

1. Checks if the input character is an **alphabet or number**.
2. If it's an alphabet, it further checks whether it's a **vowel or consonant**.

✅ Java Program:

```
import.util.Scanner;

public class CharChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a single character: ");
        char ch = scanner.next().charAt(0);

        if (Character.isLetter(ch)) {
            System.out.println(ch + " is an alphabet.");

            // Convert to lowercase for easier comparison
            char lowerChar = Character.toLowerCase(ch);

            if (lowerChar == 'a' || lowerChar == 'e' || low
erChar == 'i' || lowerChar == 'o' || lowerChar == 'u') {
                System.out.println(ch + " is a vowel.");
            } else {
                System.out.println(ch + " is a consonan
t.");
            }

        } else if (Character.isDigit(ch)) {
            System.out.println(ch + " is a number.");
        } else {
            System.out.println(ch + " is neither an alphabe
t nor a number.");
```

```
        }

        scanner.close();
    }
}
```

🧪 Sample Output:

```
Enter a single character: A
A is an alphabet.
A is a vowel.
```

Switch case :

```
import java.util.*;

public class switch1 {
    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter your choice: ");
        int choice = in.nextInt();

        switch (choice) {
            case 1:
                System.out.println("You entered 1");
                break;

            case 2:
                System.out.println("You entered 2");
                break;

            default:
                System.out.println("Invalid choice");
                break;
        }
        in.close(); // Good practice to close the scanner
    }
}
✅ Sample Output:
enter  your choice:
2
You entered 2
```

✅ 1. Count, Sum, Product of Digits

```
import java.util.Scanner;
public class DigitStats {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = 0, sum = 0, product = 1;
        System.out.print("Enter a number: ");
        int num = sc.nextInt(), temp = num;

        while (temp > 0) {
            int digit = temp % 10;
            count++;
            sum += digit;
            product *= digit;
            temp /= 10;
        }

        System.out.println("Count: " + count + ", Sum: " +
sum + ", Product: " + product);
    }
}
```

✅ 2. Autobiographical Number

A number is autobiographical if the first digit is the count of 0s, second digit is the count of 1s, etc.

```java
public class Autobiographical {
    public static void main(String[] args) {
        String num = "1210";
        int[] count = new int[10];

        for (int i = 0; i < num.length(); i++) {
            int digit = num.charAt(i) - '0';
            count[digit]++;
        }

        boolean isAuto = true;
        for (int i = 0; i < num.length(); i++) {
            if ((num.charAt(i) - '0') != count[i]) {
                isAuto = false;
                break;
            }
        }

        System.out.println(num + " is " + (isAuto ? "autobi
ographical" : "not autobiographical"));
    }
}
```

✅ 3. Spy Numbe

Sum of digits equals product of digits.

```java
import java.util.Scanner;
public class SpyNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int sum = 0, prod = 1;
        int n = sc.nextInt(), temp = n;
        while (temp > 0) {
            int d = temp % 10;
            sum += d;
            prod *= d;
            temp /= 10;
        }

        System.out.println(n + " is " + (sum == prod ? "a S
py number" : "not a Spy number"));
    }
}
```

✅ 4. Beautiful Number

A number is beautiful if the sum of its digits is present in the number itself.

```
public class BeautifulNumber {
    public static void main(String[] args) {
        int num = 123;
        int sum = 0, temp = num;
        while (temp > 0) {
            sum += temp % 10;
            temp /= 10;
        }

        System.out.println(num + " is " + (String.valueOf(n
um).contains(String.valueOf(sum)) ? "Beautiful" : "Not Beau
tiful"));
    }
}
```

✅ 5. Super Number

A number is super if the sum of its digits is divisible by each digit.

```
public class SuperNumber {
    public static void main(String[] args) {
        int num = 132;
        int temp = num, sum = 0;
        boolean isSuper = true;

        while (temp > 0) {
            int d = temp % 10;
            if (d == 0) {
                isSuper = false;
                break;
            }
            sum += d;
            temp /= 10;
        }

        temp = num;
        while (temp > 0) {
            int d = temp % 10;
            if (sum % d != 0) {
                isSuper = false;
                break;
            }
            temp /= 10;
        }

        System.out.println(num + " is " + (isSuper ? "Supe
r" : "Not Super"));
    }
}
```

✅ 6. Handsome Number

A number is handsome if the sum of its digits is a prime number.

```
public class HandsomeNumber {
    static boolean isPrime(int n) {
        if (n < 2) return false;
        for (int i = 2; i <= n/2; i++)
            if (n % i == 0) return false;
        return true;
    }

    public static void main(String[] args) {
        int num = 135;
        int sum = 0, temp = num;
        while (temp > 0) {
            sum += temp % 10;
            temp /= 10;
        }

        System.out.println(num + " is " + (isPrime(sum) ?
"Handsome" : "Not Handsome"));
    }
}
```

✅ 7. Deficient, Perfect, Abundant

```
public class NumberType {
    public static void main(String[] args) {
        int num = 12, sum = 0;

        for (int i = 1; i < num; i++)
            if (num % i == 0)
                sum += i;
        if (sum == num)
            System.out.println("Perfect number");
        else if (sum > num)
            System.out.println("Abundant number");
        else
            System.out.println("Deficient number");
    }
}
```

✅ 8. Prime or Composite

```
public class PrimeCheck {
    public static void main(String[] args) {
        int num = 13;
        boolean isPrime = true;

        if (num <= 1)
            isPrime = false;
        else {
            for (int i = 2; i <= num/2; i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        System.out.println(num + " is " + (isPrime ? "Prim
e" : "Composite"));
    }
}
```

✅ 9. Palindrome

```
public class PalindromeCheck {
    public static void main(String[] args) {
        int num = 121, rev = 0, temp = num;
        while (temp > 0) {
            rev = rev * 10 + temp % 10;
            temp /= 10;
        }
        System.out.println(num + " is " + (rev == num ? "Palindrome" : "Not Palindrome"));
    }
}
```

✅ 10. Neon Number

Square of number, and sum of digits of square = number.

```
public class NeonCheck {
    public static void main(String[] args) {
        int num = 9;
        int sq = num * num;
        int sum = 0;
        while (sq > 0) {
            sum += sq % 10;
            sq /= 10;
        }
        System.out.println(num + " is " + (sum == num ? "Neon" : "Not Neon"));
    }
}
```

✅ 11. Xylem/Phloem

If the sum of first and last digit equals the sum of middle digits → Xylem, else Phloem.

```java
public class XylemPhloem {
    public static void main(String[] args) {
        int num = 1234, temp = num;
        int first = 0, last = temp % 10, middleSum = 0;
        temp /= 10;

        while (temp >= 10) {
            middleSum += temp % 10;
            temp /= 10;
        }
        first = temp;

        System.out.println(num + " is " + ((first + last) =
= middleSum ? "Xylem" : "Phloem"));
    }
}
```

✅ 12. Harshad Number

Number divisible by the sum of its digits.

```java
public class HarshadNumber {
    public static void main(String[] args) {
        int num = 18, sum = 0, temp = num;
        while (temp > 0) {
            sum += temp % 10;
            temp /= 10;
        }

        System.out.println(num + " is " + (num % sum == 0 ?
"Harshad" : "Not Harshad"));
    }
}
```

✅ 13. Armstrong Number

Sum of each digit raised to the power of total digits equals the number.

```
public class ArmstrongNumber {
    public static void main(String[] args) {
        int num = 153, temp = num, sum = 0, count = 0;

        while (temp > 0) {
            count++;
            temp /= 10;
        }

        temp = num;
        while (temp > 0) {
            int d = temp % 10;
            sum += Math.pow(d, count);
            temp /= 10;
        }

        System.out.println(num + " is " + (sum == num ? "Ar
mstrong" : "Not Armstrong"));
    }
}
```

Methods in java :

1. it represents the Block of statements which is used to perform the simple or complex task.
2. it is also called as function member

Purpose :

to achive

- code re-useablity .
- code optimization
- code readability
- business logics implementation

Syntax

```
[access modifier][modifier] return_type methodName([formal_
arguments]){

//statements

}
```

Access modifier

1. it represents the Accessibility if the Members of the programs
2. it can be pre-fixed to he Class , Interface , Variable , Method , Initialzers and Constructor.

Types

1. private
2. public
3. protected

Modifers :

1. it represents the functionality of the Members of the program
2. it can pre - fixed to the Classs,interface,variable, and initializers

types

1. abstract
2. default
3. final
4. native
5. static
6. strictfy
7. synchronized
8. transient
9. volatile

Return

1. it is a keyword
2. it is a Branching statement

purpose : it is used to Return the type of data form the called Method to the Caller method

Return Type:

it represents the type of data that the Metod has to return to the Caller

Types:

1. void.
2. primitive data type.
3. non primitive data type.

Types of methods

based on modifier

1. static method .
2. non static method .

based on Availability

1. pre-defined Method
2. User-Defined Method

based on Formal Arguments

1. no Arguments Method
2. Parameterized Method

Method call Statement

it represents the statement which is used to call the method

types

1. static Method Call Statement
2. Non Static Method Call statement

Static Method call Statement

it represents the Statement which is used to call the static method

Syntax

```
className.staticMethodName([actual_arguments]);
```

Non static Method Call Statement

it represents the Statement which is used to call the Non Static Method.

Syntax

```
className objectName = new className([actual_arguments]);
objectName.nonStaticMethodName([actual_arguments]);
```

note

1. for each and every Static Method call Statement, static method will get executed.

2. for each and every non static method call statement , non static method will get executed.

```
class Test{
  public static void display(char c){
       System.out.println(c);
       }
  }
```

```
class javaprg{
public static void execution(){
Test.display('j');
 }
}
```

```
class Main{
public static void main(String args[]){
   JavaProgram.execution();
   }
}
```

```
class JavaProgram{
public static void execution(){
  System.out.println("Hello world");
  }
}
```

Class Scanner

Hard coding

1. it represents the variable initilization by the java programmer during the compliation of the java Application
2. it is also called as static input

note:

while achiving Hard coding , if any mismatch occurs then at that time the java complier will generate the compilaation error.

Soft coding

1. it represents the variable initialization by the User during the execution of the java application.
2. it is also called as Dynamic input.

note

while achieving Soft coding, if any mismatch occurs, then at that time the java virtual machine will generate the Execution Error.

Scanner Class

1. it is a pre Defined class which is present within the pre defined module ie. java.base nd package ie. java.util
2. it consists of the contructors and the methods.

purpose of Scanner Class

to achive soft coidng

Different steps to work on Scanner Class

1. import statement
2. object creation statement
3. method call statement

Module

java.base

package

java.util

Entity

Class Scanner

Class Declaration

```
public final class Scanner
extends Object
implements Iterator , Closeable
```

Contructor Declaration

public Scanner (InputStream source)

Method Decleration

1. public byte nextByte()
2. public short nextShort()
3. public int nextInt()
4. public long nextLong()
5. public float nextFloat()
6. public double nextDouble()
7. public boolean nextBoolean()
8. publicString next()
9. public String nextLine()
10. public void close()

Methods example (arithmetic operators) :

```
import java.util.Scanner;
public class Arithematic {
    public static int sub(int a, int b) {
        int result = a - b;
        return result;
    }
    public static int add(int a ,int b){
        int result =a+b;
        return result;
    }
    public static int multi(int a , int b){
        int result=a*b;
        return result;
    }
    public static int div(int a , int b){
        int result=a/b;
        return result;
    }
    public static void display(int a,int d,int p, int div,i
nt num1,int num2){
        System.out.println("the addition of "+num1+" and "+
num2+"is = "+a);
        System.out.println("the difference of "+num1+" and
"+num2+"is = "+d);
        System.out.println("the product of "+num1+" and "+n
um2+"is = "+p);
        System.out.println("the qutiont of "+num1+" and "+n
um2+"is = "+div);
    }
    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        System.out.println("enter the num1");
        int num1=in.nextInt();
        System.out.println("enter the num2");
        int num2=in.nextInt();
        int add=add(num1,num2);
        int difference = sub(num1,num2);
        int product= multi(num1,num2);
        int qutiont = div(num1,num2);
        display(add,difference,product,qutiont, num1,num2);
    }
}
```

Multiplication table

```
import java.util.*;
public class mutiplicationTable {
    public static void main(String [] args){
        Scanner in = new Scanner(System.in);
        System.out.println("enter a nuber");
        int n= in.nextInt();
        for(int i =1 ; i<=10 ; i++){
            System.out.println(n+"*"+i+"="+(n*i));
        }
    }
}
```

```
public class helloSir {
    public static void main(String[] args) {
        System.out.println("I feel truly fortunate to be le
arning Core Java under the guidance of Sivaraj Hirematt Si
r.");
        System.out.println("His teaching is not only knowle
dgeable but also deeply inspiring.");
        System.out.println("Sir has a unique way of breakin
g down complex topics into simple, understandable concept
s.");
        System.out.println("His sessions are always engagin
g, filled with real-life examples and clear explanations");
        System.out.println("that make learning Java both ef
fective and enjoyable.");
        System.out.println("What I admire the most is his p
atience and dedication—");
        System.out.println("he ensures that every student u
nderstands the topic thoroughly before moving ahead.");
        System.out.println("Learning from Sivaraj Sir has m
ade me more confident in my Java skills,");
        System.out.println("and I am genuinely grateful for
the opportunity to be his student.");
    }
}
```

pattren printing :

```
class pattern_printing {

    public static void closed() {
        for (int i = 0; i < 5; i++) {
            for (int j = 1; j <= 5; j++) {
                System.out.print(" * ");
            }
            System.out.println();
        }
    }

    public static void open() {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 5; j++) {
                if (i == 1 || j == 1 || i == 5 || j == 5) {
                    System.out.print(" * ");
                } else {
                    System.out.print("   "); // print space
s to keep structure
                }
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        open();
        System.out.println();
        closed();
    }
}
```

Class and Object

Class :

1. it represents the blueprint or the prototype or the templte of the object
2. it consists of variables and the methods
3. it is considered as the imaginary entity according to both real world and software application
4. it is possible for the java prg to create number of classes

Object

1. it represents the instance of class
2. it consists of the variables and the methods
3. the variables are used to represent the state of the object
4. the methods are used to represent the behaviour of the object
5. it is considered as the real entity and the imagenary entity according to the real world and the software application respectively
6. it is possible for the java programmer to create the n number of objects for a single class.

Static

1. it is a keyword
2. it is a modifier
3. it can be pre-fixed to the class, interface , variable , initializers and Method
4. technically , static means Shared Memory Location
5. generally , Static means single copy.

Static members

the members of the class which has been declared as static by pre-fixing using the static keyword.

Different types

1. static variable
2. static method
3. static initializer

Properties

1. static members will be stored within the class static area
2. static members can be accessed by using the class name

Non static

1. technically , non static means UnShared Memory Location
2. Generally, Non Static means Multiple Copies

Different types

1. non static varaible
2. non static method
3. non static initializer
4. constructor

Properties

1. non static members will be stored within the heap area
2. non static members can be accessed by using the object name

Static variable and non static variable

```
syntax
[access_modifier] [modifier] data_type varable_name;
```

properties

- Static variable and non static variable will holds the default data according to the data type.

static initializer and non static method

```
syntax
[access_modifier] [modifier] return_type method_name([forma
l_arguments]){
  //statements
  }
```

Properties

1. for each and every static method call statement , static method will get executed .
2. for each and every non static method call statement, non static method will get executed.
3. static method is used to achieve static variable initialization
4. non static method is used to achieve non static variable initialization

Static initialization and non static initializer

```
synatx
[modifier]{
  //statements
  }
```

Properties

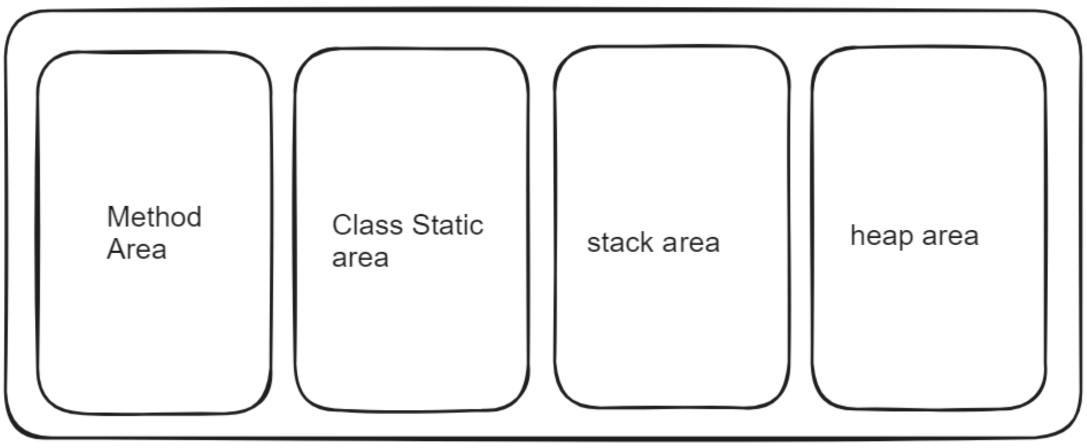
1. static initializer is also called static block or static context.
2. non Static initializer is also called non static block or non static context.

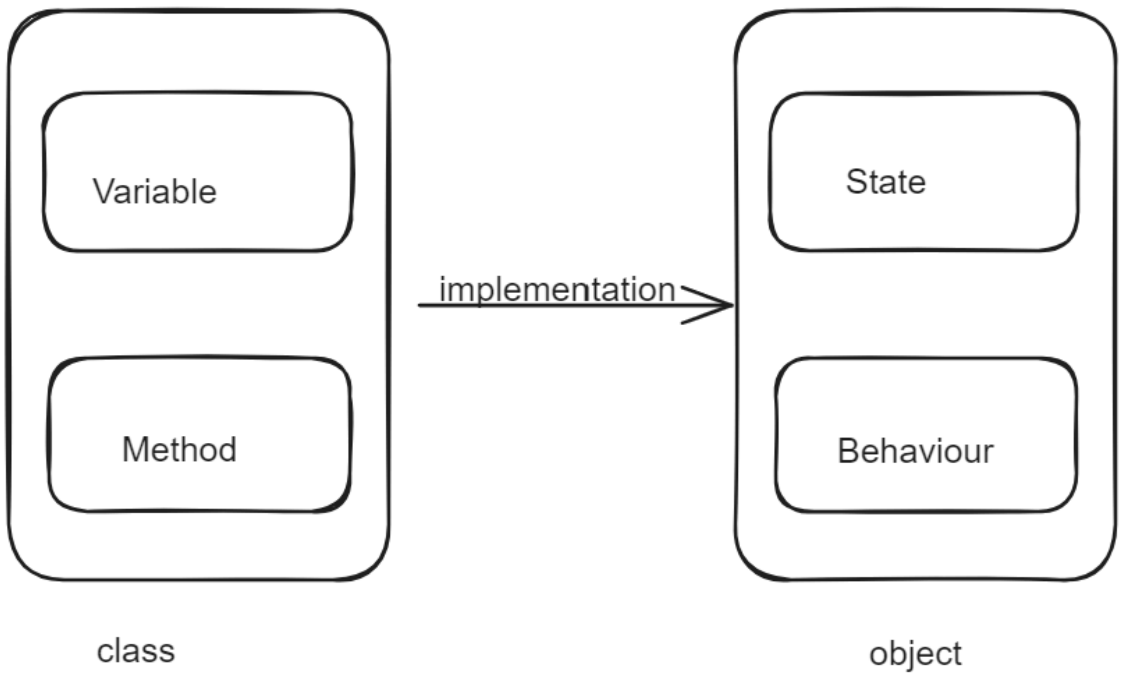
3. for each and every class loading process , static block will get executed single time in a top to bottom approach.
4. for each and every object loading process , non static block will get executed multiple times in a top to bottom approach .
5. class loading process always occurs first
6. object loading process always occurs second .

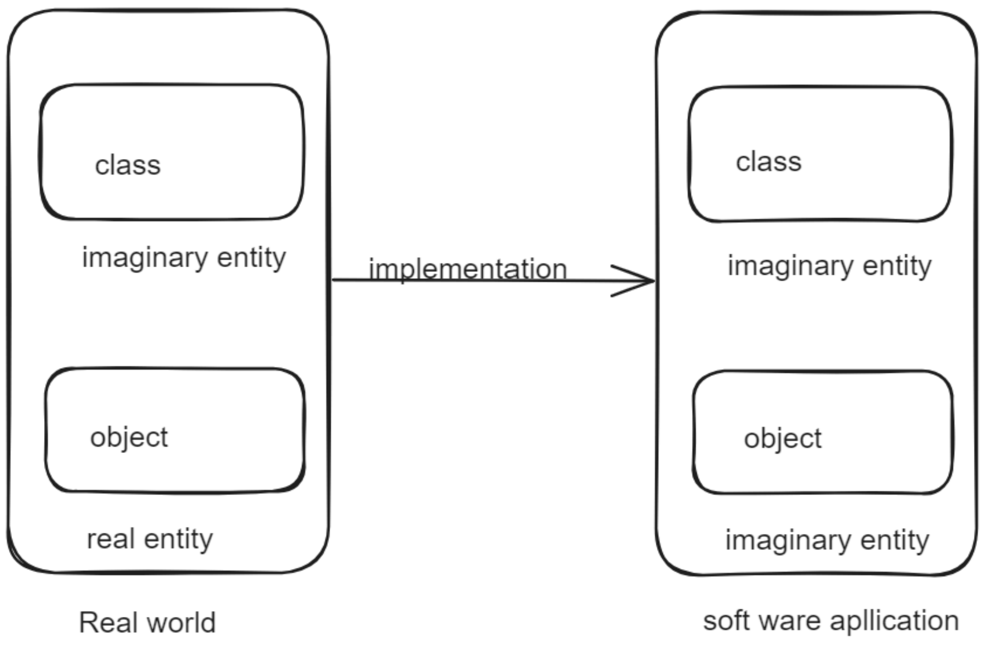
Purpose

1. static initializer is used to achieve static variable initialization .
2. non static initializer is used to achieve non static variable initialization .

RAM :







Constructor

it represents the block of the statement which is used to preform the task

Purpose of constructor

1. code re-useability
2. code Optimization
3. code readability
4. to load all the non static members of the class into the object
5. to initialize all the non static variable of the class into the object

```
Syntax
[access_modifier] className([formal_arguments)]
 {
   // statement
   }
```

Different types of constructors

1. no argument constructor

```
class classname{                                      class cla
ssname{
                      ---> java complier                 classn
ame(){
     }
super();
                                                              }
                                                              }
```

2. Based on java programmer
- pre-defined constructor
- user defined constructor

3. based on formal arguments
- no arguments constructor

- parameterized constructor

Constructor call statement

it represents the statement which is used to call the constructor

Different types of constructor call statement

1. no arguments constructor call statement
2. parameterized constructor call ststement

no arguments constructor call statement

it represents the statement which is used to call no arguments constructor

```
syntax
className objectName = new className([actual_arguments]);
```

Purpose of no argument constructor

1. to load all the non static members of the class into object

Parameterized constructor call statement

it represents the statement which is used to call the parameterized constructor

```
syntax
className objectName = new className([actual_arguments]);
```

Purpose of parameterized constructor

1. to load all the non static members of the class into the object
2. to initialize all the non static variables of the class into the object.

Note

1. if the java programmer does not generate either of the argument or the parameterized constructor , then at that time the java compiler generate the no argument constructor
2. if the java programmer generates either of the no arguments or the parameterized constructor , then at that time the java complier does not generate the no argument constructor
3. for each and every constructor call statement , constructor will get executed
4. constructor call statement is a sub part of the object creation statement

THIS

1. itis a keyword
2. it is a predefined non static non primitive variable
3. it will holds the current class type object address

purpose of this

1. it is used to differentiate the Global Variable and the local variable whenever they are having the same name.
2. it is used to access the current class type non static member
3. it has too be used always within the non static context

Global variable

it represents the variable declaration inside the class or interface declaration but outside the method or constructor deceleration or implementation

Local variable

it represents the variable declaration either along with the method or constructor declaration or within the method or constructor implementation

Note

1. global variable will holds the default data according to the data types . therefore , global variable can be used without initialization
2. Local variable will not holds the default data according to the data types . therefore local variable cannot be used without initialization

object oriented programming system

it represents the process of Scientific and systematic way of designing and developing the software application as a solution according to the real world problem as per the customer requirements

Principles

1. analysis
2. design
3. implementation

Concepts

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

Encapsulation

it represents the process of binding the variables with the methods of the same class.

Purpose

to achieve Data hiding

Data hiding

1. it represents the process of restricting the direct access and providing the indirect access to the variable by using the methods of the same class
2. it is considered as the data visibility , According to the java complier
3. it is considered as Data accessibility , According to the java programmer

Purpose

to achieve data security

steps to achieve data security

1. generate and declare the variables as private by pre fixing using the private keyword
2. generate and declare the variables as public by pre fixing using the public keyword

Private

1. it is a keyword
2. it is an Access Modifier
3. it is having the lowest visibility and the highest security

public

1. it is a keyword
2. it is an access modifier
3. it is having the highest visibility and the lowest security