Operators: An operator is a symbol that tells the compiler to perform specific mathematical or logic function.

1. Arithmetic Operator:
2. Relational Operator
3. Logical Operator
4. Conditional Operator
5. Assignment Operator
6. Bitwise Operator
7. Shift Operator

Arithmetic Operators:

+	Addition
-	Substraction
*	Multiplication
/	Division
%	Modulus
Unary Operator:	
In++	Post Increment
De--	Post Decrement

++In	Pre Increment
--De	Pre Decrement
+=	Addition assignment
-=	Substraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment

Program:

```java
public class ArithmeticOperator {
     public static void main(String[] args)
     {
       int a = 9;
       int b = 2;
       int add = a+b;
       int subs = a-b;
       int multi = a*b;
       int divi = a/b;
       int modulo = a%b;
       System.out.println("Addition of 9 + 2 = " + add);
       System.out.println("Substraction of 9 - 2 = " + subs);
       System.out.println("Multiplication of 9 * 2 = " + multi);
       System.out.println("Division of 9 / 2 = " + divi);
       System.out.println("Modulus of 9 % 2 = " + modulo);
    }
}
```

Increment(++) & Decrement Operator(--) :

The increment operator (++) increases its operand by one AND The decrement operator (- -) decreases its operand by one.

Example: x = x + 1 → x++; x = x - 1 → x—;

a++: Post increment -> Next Statement
++a; Pre increment -> Same statement
a--; Post decrement -> Next Statement
--a; Pre decrement -? Same statement

Program:

```java
public class IncOrDec2 {
  public static void main(String[] args) {
     int x = 21;
     x++;
    System.out.println(x++); //Print = 22, Store = 23
    System.out.println(++x); //Print = 24, Store = 25
    System.out.println(x--); //Print = 24, Store = 23
    System.out.prinltn(--x); //Print = 22, Store = 20
     }
}
```

Relational Operator: The relational operators determines the relationship that one operand has to the other. Specifically, they determine equality and ordering

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

→The outcome of these operation is a Boolean value.

The relationship operators are most frequently used in the expression that control the if statement and the various loop statements. Any type in Java, including Integers, Floating-point numbers, characters, and Boolean can be compared using the equality test, ==, and the inequality test, !=.

Remember: A single equal sign is the assignment operator, and two equal signs is used to denoted equality.

```
if(!equal)... // Valid in C/C++, but not valid in Java.
    This statement must be written like:
if (done == 0) // This is Java-Style
if (done != 0)
```

Program:

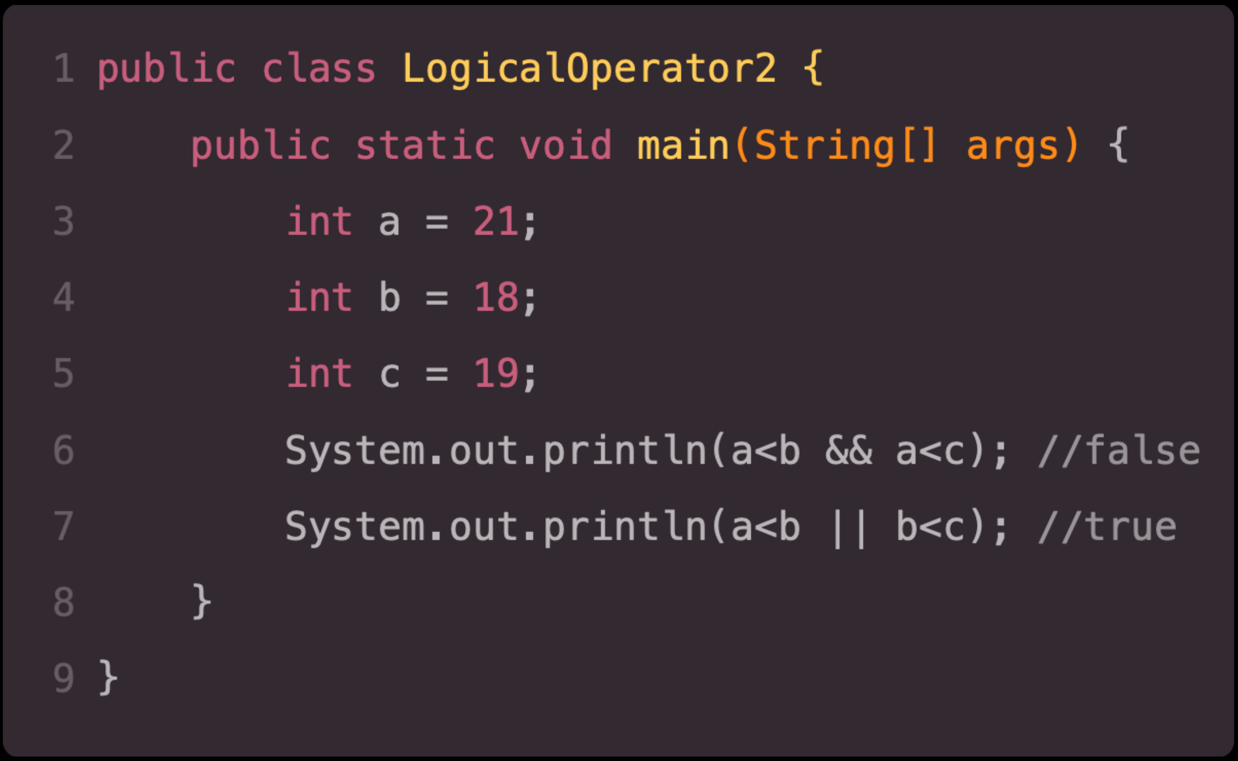
```java
// Writing a program on Relational Operator
public class RelationalOperator {
    public static void main(String[] args)
    {
      int dhruv = 21;
      int lenka = 19;
      System.out.println(dhruv == lenka);
      System.out.println(dhruv != lenka);
      System.out.println(dhruv > lenka);
      System.out.println(dhruv < lenka);
      System.out.println(dhruv >= lenka);
      System.out.println(dhruv <= lenka);
    }
}
```

Logical Operator: The logical operators combine two boolean values to form a resultant boolean value. The logical (&&) operator doesn't
check the second condition if the first condition is false. It checks second condition only when first condition is true. The logical (||) operator doesn't check second condition if the first condition is true. It checks second condition only when first condition is false.

Operator	Result
&&	Logical AND
||	Logical OR
!	Logical NOT

A	B	A | B	A & B
FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE

A	B	A ^ B	!A
FALSE	FALSE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	TRUE
TRUE	TRUE	FALSE	FALSE



```java
public class LogicalOperator2 {
    public static void main(String[] args) {
        int a = 21;
        int b = 18;
        int c = 19;
        System.out.println(a<b && a<c); //false
        System.out.println(a<b || b<c); //true
    }
}
```

Bit-wise Operator:

The logical ! Operator inverts the Boolean state: ! true == false and !false == true.

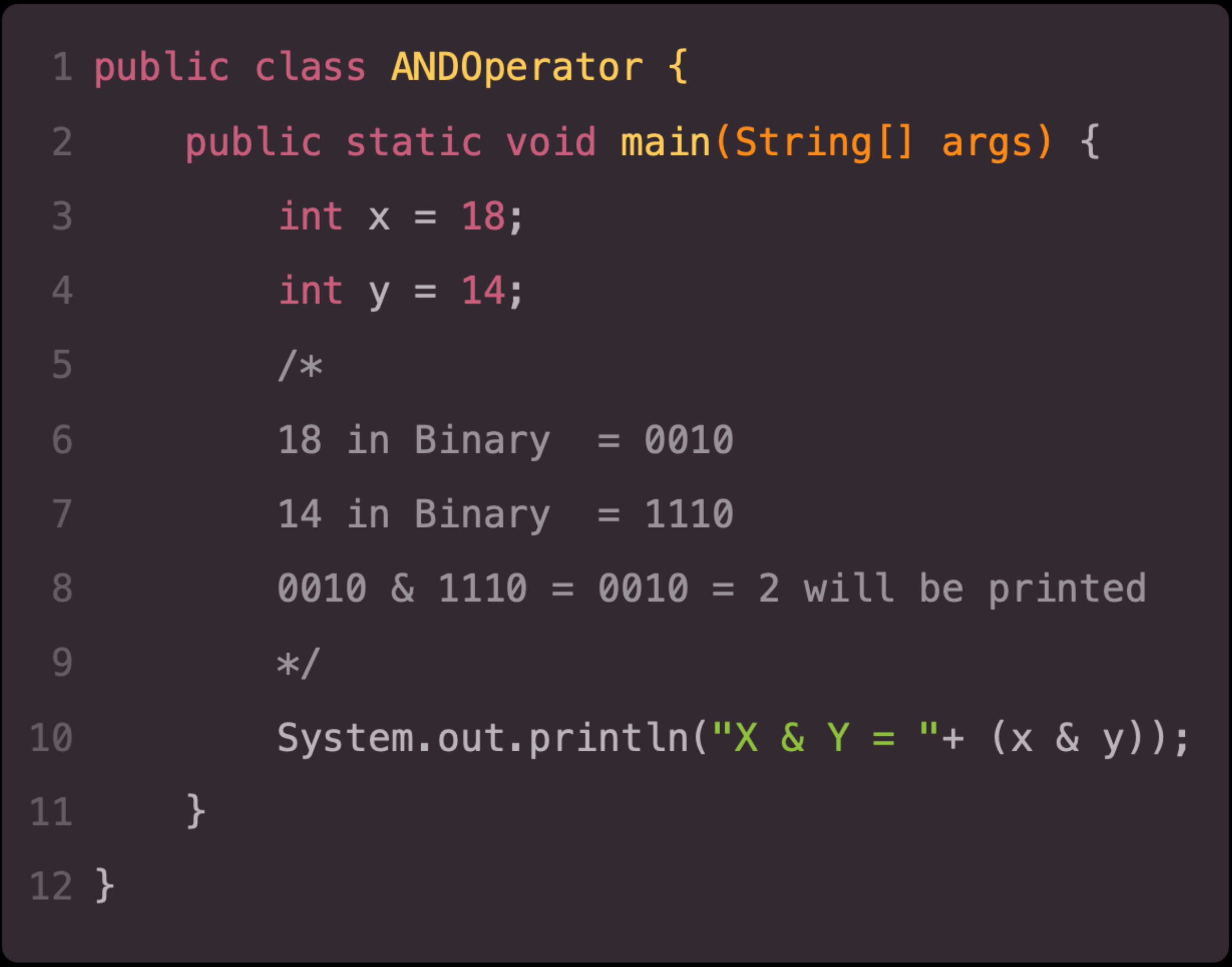
Operator	Result
~	Bitwise Compliment or Not
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
<<	Bitwise Left Shift Operator
>>	Bitwise Right Shift Operator
>>>	Unsigned Right Shift Operator

BITWISE AND Operator:

It returns 1 if and only if both bits are 1, else return 0.

A	B	A&B
0	0	0
1	0	0
0	1	0
1	1	1

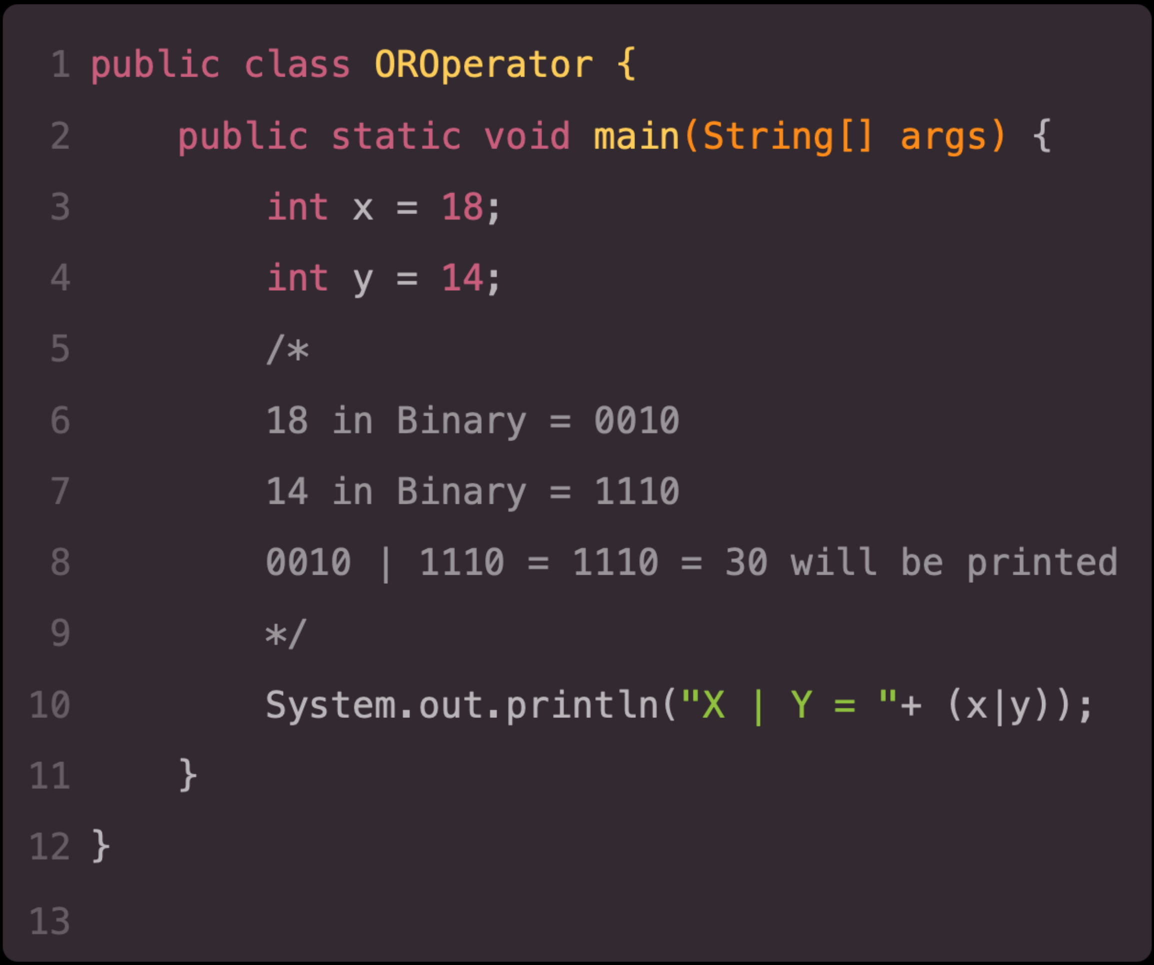
Program:



```java
public class ANDOperator {
    public static void main(String[] args) {
        int x = 18;
        int y = 14;
        /*
        18 in Binary  = 0010
        14 in Binary  = 1110
        0010 & 1110 = 0010 = 2 will be printed
        */
        System.out.println("X & Y = "+ (x & y));
    }
}
```

BITWISE OR Operator: It returns 1 if either of the bit is 1, else returns 0.

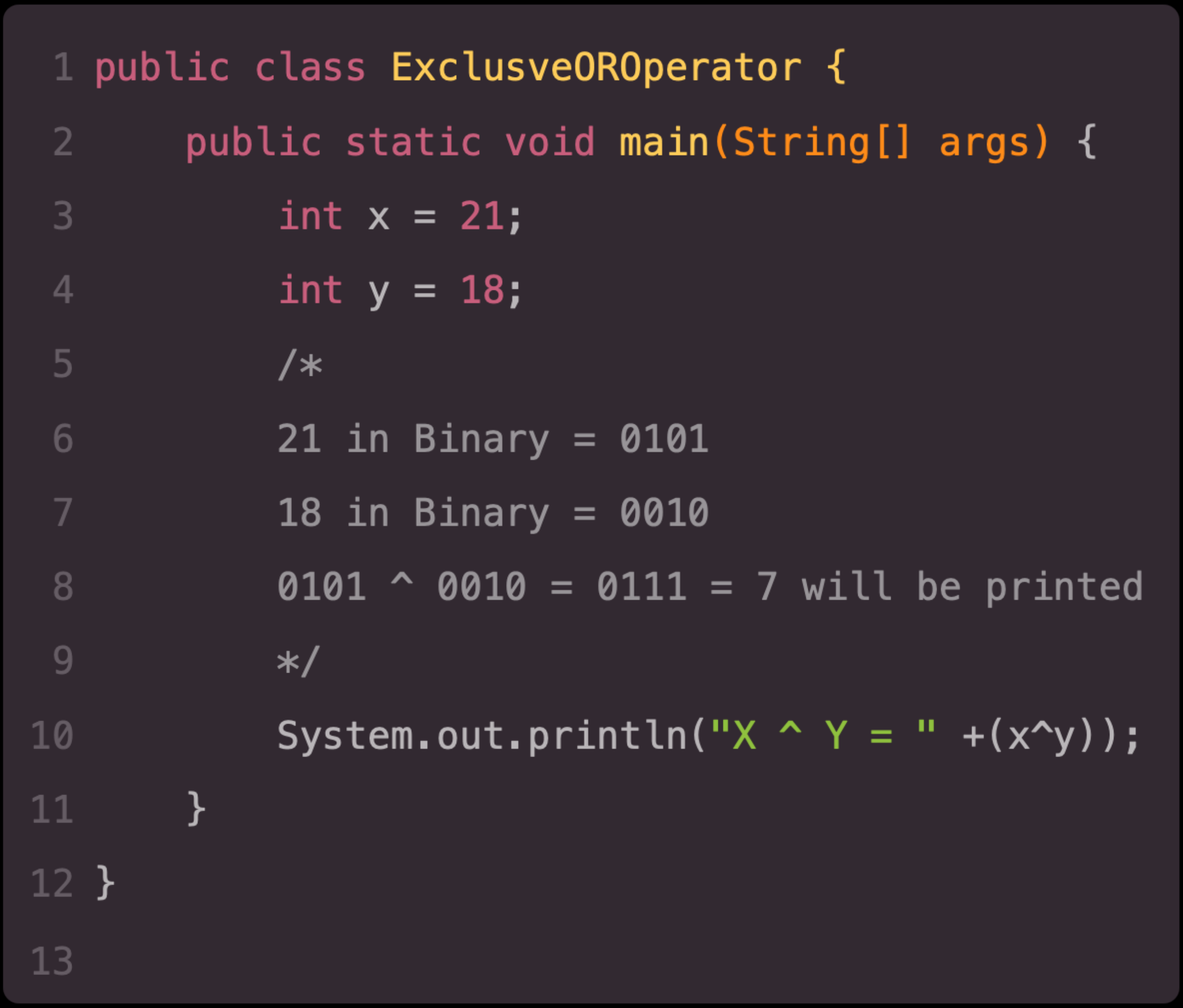
A	B	A|B
0	0	0
1	0	1
0	1	1
1	1	1



```java
public class OROperator {
    public static void main(String[] args) {
        int x = 18;
        int y = 14;
        /*
        18 in Binary = 0010
        14 in Binary = 1110
        0010 | 1110 = 1110 = 30 will be printed
        */
        System.out.println("X | Y = "+ (x|y));
    }
}
```

BITWISE Exclusive Operator(^): It will returns 0 if both bits are the same, else will returns 1.

A	B	A^B
0	0	0
1	0	1
0	1	1
1	1	0



```java
public class ExclusveOROperator {
    public static void main(String[] args) {
        int x = 21;
        int y = 18;
        /*
        21 in Binary = 0101
        18 in Binary = 0010
        0101 ^ 0010 = 0111 = 7 will be printed
        */
        System.out.println("X ^ Y = " +(x^y));
    }
}

```

BITWISE Compliment Operator(~): It returns the inverse or complement of the bit. It makes every 0 a 1 and every 1 a 0.

x	~x
0	1
1	0

Left Shift Operator: The Left Shift Operator is a type of Bitwise Shift Operator, which performs operations on the binary bits. It is a binary operator that requires two operands to shift or move the position of the bits to the left side and add zeroes to the empty space created at the right side shifting the bits.

Method:

1. Suppose, 10<<2, We convert 10 into binary and remove 2 bits from left and fill empty spaces as zero(0) in right bits.
2. Then, We convert Binary to Decimal

Shortcut Method:

Left Shift Operator (Value * 2^n Multiplication)

- Suppose 10<<2 = 10*2*2 = 40

```java
// Left Shift Operator (Multiplication)
public class LeftShiftOperator {
    public static void main(String[] args) {

        System.out.println(21<<2);
        System.out.println(18<<3);
        System.out.println(19<<4);
        System.out.println(20<<5);
    }
}

```

Right Shift Operator: The Right Shift Operator is a type of bitwise shift operator used to move the bits at the right side. Like the left side operator, the right shift operator also requires two operands to shift the bits at the right side and then insert the zeroes at the empty spaces created at the left side after shifting the bits.

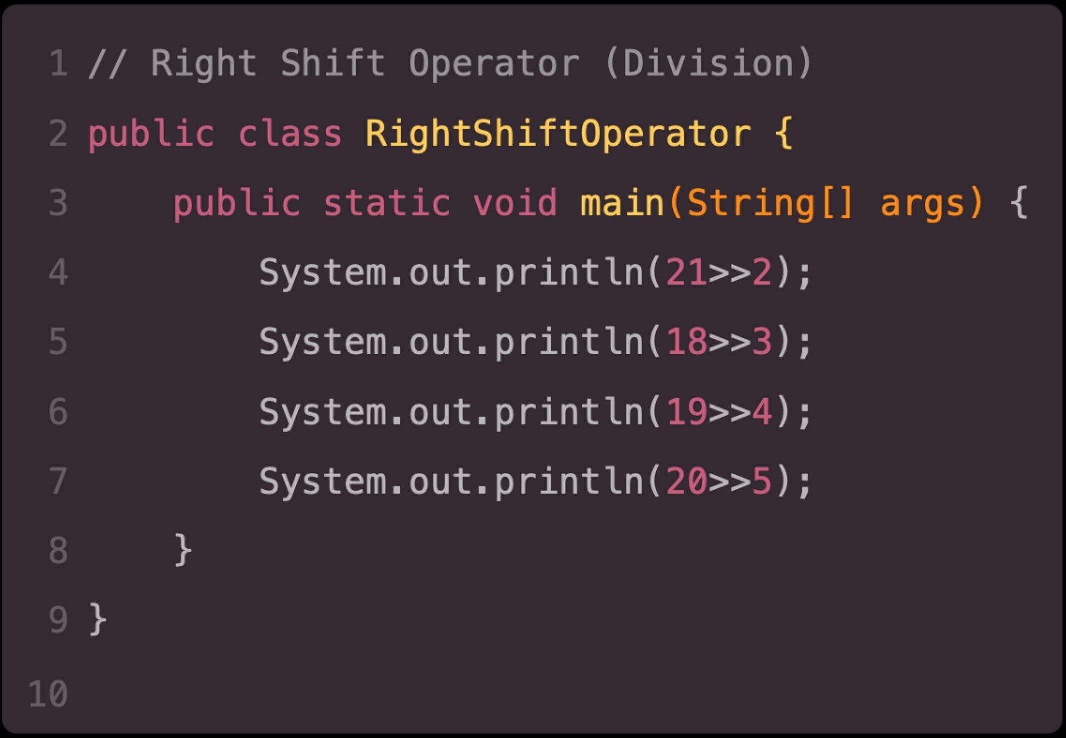
Method:

1. Suppose 10>>2, We convert left side operand to binary and remove two bits from the right and fill empty spaces as zero(0) in left bits.
2. Then, We convert Binary to Decimal

Shortcut Method:

Right Shift Operator(Value / 2^n Division)

- Suppose 10>>2 = 10/2*2 = 10/4 = 2

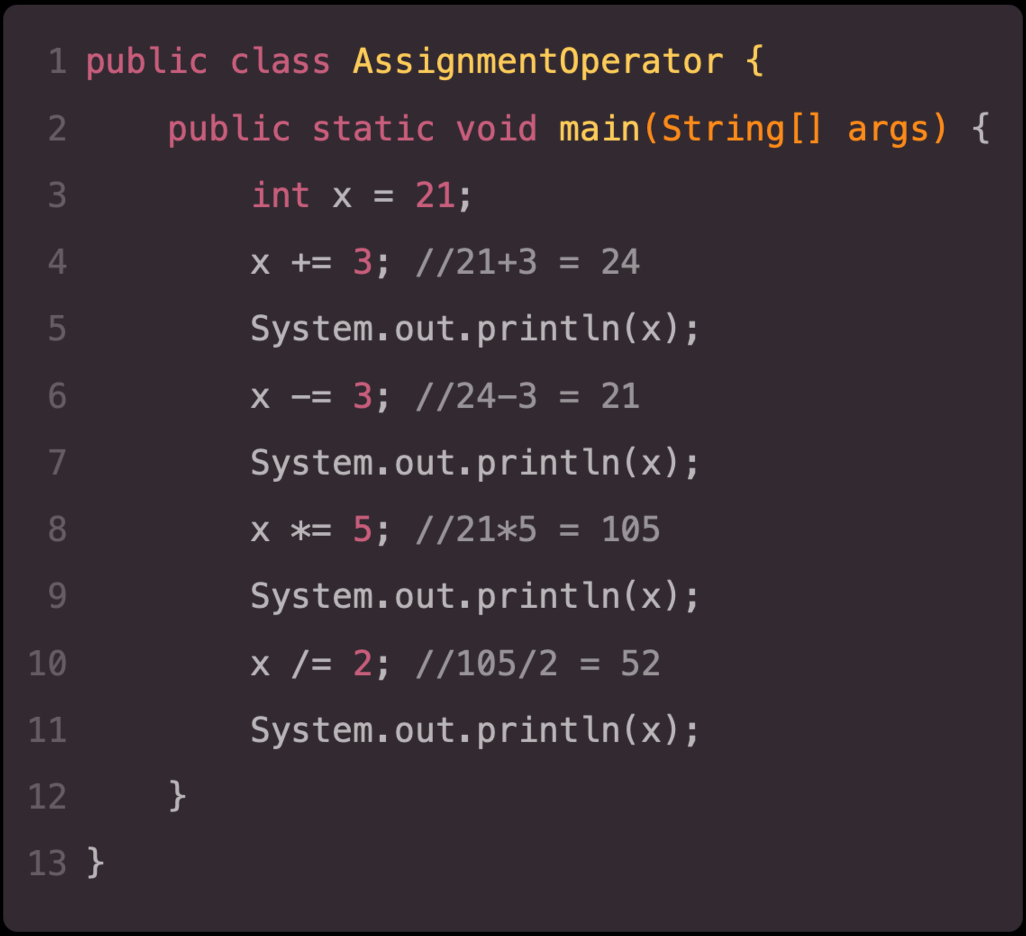


```java
// Right Shift Operator (Division)
public class RightShiftOperator {
    public static void main(String[] args) {
        System.out.println(21>>2);
        System.out.println(18>>3);
        System.out.println(19>>4);
        System.out.println(20>>5);
    }
}

```

Assignment Operator: Assignment operators are used in Java to assign values to variables.
It assigns the value on its right to the variable on its left.
Ex:
int age = 21;

+=	Addition assignment
-=	Substraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment



```java
public class AssignmentOperator {
    public static void main(String[] args) {
        int x = 21;
        x += 3; //21+3 = 24
        System.out.println(x);
        x -= 3; //24-3 = 21
        System.out.println(x);
        x *= 5; //21*5 = 105
        System.out.println(x);
        x /= 2; //105/2 = 52
        System.out.println(x);
    }
}
```

Ternary Operator(?) : Java Ternary Operator is used as one line replacement for if-then else statement. It is only conditional operator which takes three operands.

Ex:

variable = expression ? expression1 : expression2

- If the expression is true , expression1 is assigned to the variable.
- If the expression is false, expression2 is assigned to the variable.

```java
public class TernaryOperator {
    public static void main(String[] args)
    {
        int dhruvage = 21;
        int ayushage = 18;
        int minAge = (dhruvage>ayushage ? dhruvage : ayushage);
        System.out.println(minAge);
    }
}
```

Java Output: For output in Java we can write:

A. System.out.print();
B. System.out.printf();
C. System.out.println();

Where:

System is a class,

out is a public static field: it accept output data

Java Input or Scanner Class: Scanner class is used to collect user input and Java class is a part of Java.util package.

→ Scanner read text from standard input and return it to program.

In order to use the object of scanner, we need to import:

Step 1: import java.util.Scanner
Step 2: Create an object of the Scanner class. We can use the object to take input from the user.

```java
// Creating a object of the Scanner
Scanner input = new Scanner(System.in);

// Taking input from the user
int number = input.nextInt();
```

Data Type Method:

```java
// Data Type Methods
boolean nextBoolean()
byte nextByte()
short nextShort()
int nextInt()
long nextLong()
float nextFloat()
double nextDouble()
string next(), nextLine()
char next().charAt(), nextLine().charAt()
```

Program for taking Integer Input:

```java
// Program upon taking int type input.
import java.util.Scanner;
public class Input {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.print("Enter a Inter Value =  ");
        int number = input.nextInt();
        System.out.println("You entered = " + number);

        // closing the scanner object
        input.close();
    }
}
```

Control Statement: Java provides statements that can be used to control the flow of Java code, Such statements are called Control Statement.

OR

Java programs control statements can be divided into three categories: Selection, Iteration and Jump. Selection statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable. Iteration statements enable program execution to repeat one or more statements (that is, iteration statements from loops). Jump statements allow your program to execute in a nonlinear fashion.

Selection Statement: Check/Filter

1. If
2. Switch

Iteration Statement: Repeat/Loop

1. For Loop
2. Do While Loop
3. While Loop

Jump Statement: Stop/Skip

1. break
2. Continue

IF Statements:

1. Simple IF Statements
2. IF-Else Statements
3. If-else-If Ladder
4. Nested If-Statement

1. Simple IF Statement: If statement tests the condition and executes the if block if condition is true.

Syntax:

```
// Syntax of Simple IF Statement
if(condition) {
 statements // Executed when condtion is true.
}
```

2. IF-Else Statement: In IF-Else statements, If condition is true then statement under the if will executed, if condition is false then statement under the else will be executed.

Syntax:

```
// Syntax of IF-Else  Statement
if(condition) {
 statements // Executed when condtion is true.
} else {
 statements // Executed when condition is false.
}
```

3. IF-Else-If-Laddar Statements: The IF-Else-If Statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax:

```
// IF-Else-If-Ladder Statements
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

4. Nested-If Statements: In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

OR

In simple words, in nested-if statements contain one more if or if..else statements under another if or if..else statements.

Syntax:

```
// Syntex of Nested-IF Statements
    if(condition 1) {
    statement 1; //executes when condition 1 is true
    if(condition 2) {
    statement 2; //executes when condition 2 is true
    }
    else{
    statement 2; //executes when condition 2 is false
    }
    }
```

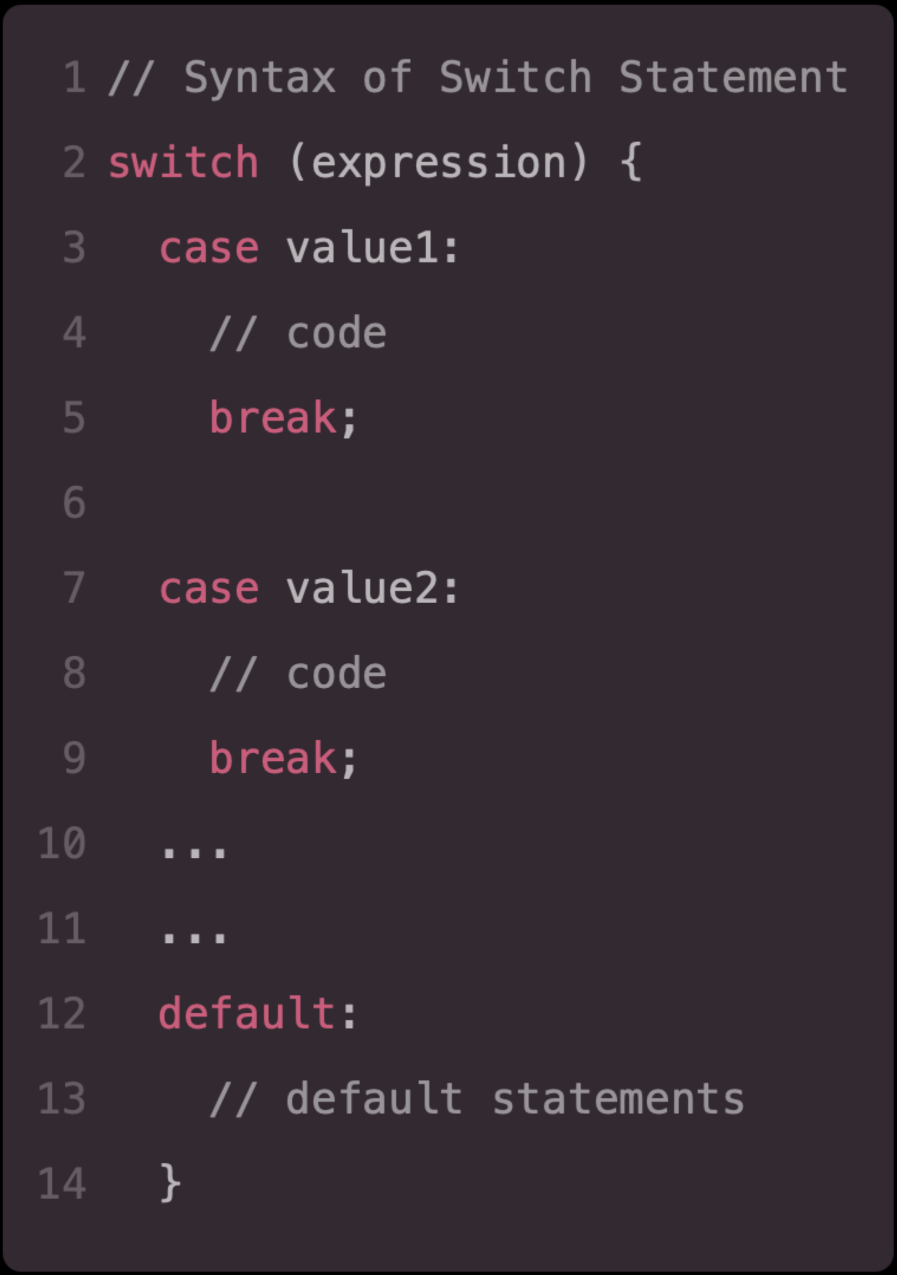
Switch Statement: The Java switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement.

Point to remember for writing Switch Statement:

1. One or 'n' number of case values for a switch expression.
2. The case value must be of switch expression type only. The case value must be literal or constant. It doesn't allow variables.
3. The case value must be unique. In case of duplicate value, it renders complex-time error.

4. The Java switch expression must be of byte, Short, int, long and string.
5. Each case statements can have a break statements which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
6. The case value can have a default label which is optional.

Syntax:



```
// Syntax of Switch Statement
switch (expression) {
  case value1:
    // code
    break;

  case value2:
    // code
    break;
  ...
  ...
  default:
    // default statements
  }
```

Iteration Statement/Loops: A Loop executes the sequence of statements many times until the started condition becomes false.

OR

A loop repeatedly executes the same set of instruction until a termination condition is met. The purpose of the loop is to repeat the same code a number of times.

Simple for Loops:

1. Initialization: It is the initial condition which is executed ones when the loop starts.
   It is an optical condition.
2. Condition: If condition is true then, the body of loop will be executed, if false then, the loop is terminated. It is an optical condition.
3. Increment: It increments or decrements the variable value. It is an optical condition.
4. Statement: The statement of the loop is executed each time until the second condition become false.

There are 3 types of Loops: While, Do..While..For Loop

While Loop: It repeats a statement or block while its statement is true.

Syntax:

```java
while(condition){
// statement or body of loop
}
```

* Condition should be boolean type.

Working:

True: condition-t->statements->condition-t->statements

False: condition-f-?=>stop

```java
public class BookPrograms {
public static void main(String[] args) {
int i = 100;
int j = 200;
while(++i < --j);
System.out.println("Midpoint of 100 & 200 = "+i);

  }
}

```

How this program will work: The value of ‘i’ is incremented, and the value of ‘j’ is decremented. These values are then compared with one another. If the new value of ‘i’ is still less than than the new value of ‘j' , then the stop repeats. If ‘i’ is equal to or greater than ‘j’, the loop stops. Upon exit from the loop, ‘i’ will hold a value that is midway between the original values of ‘i’ and ‘j’.(This procedure only works when ‘i’ is less than ‘j’ to begin with.) As we can see, there is no need for a loop body; all the action occurs within the conditional expression, itself. In professionally written java code, short loops are frequently coded without bodies when the controlling expression can handle all of the details itself.

Do While Loop: The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

Why we use Do..WhileLoop: In our previous program, if the conditional expression controlling a while loop is initially false, then the body of the loop will not be executed at all. However, Sometimes it is desirable to execute the body of a loop at least once, even if the conditional expression is false to begin with. In simple words: Sometime you would like to test the termination expression at the end of the loop rather than at the beginning.

Syntax:

```
// Do..While Loop Syntax
do {
// Statement or body of the loop
} while (condition);
```

Working:

True:statement->condition-t->statements->condition-t->

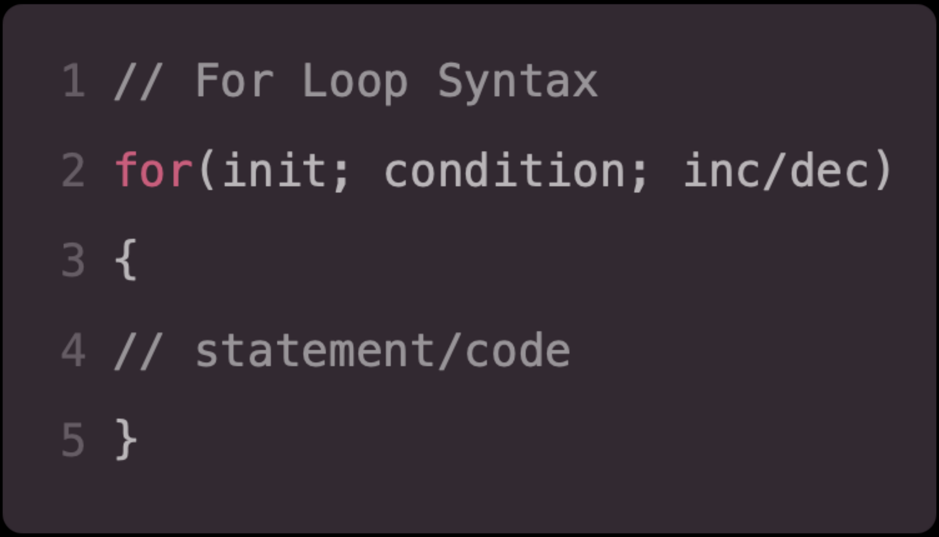
False: statements->condition-f->stop

For Loop: For loop is used to run a block of code for a certain number of times.

OR

For loop is used to iterate a part of the program several times. If the number of iteration is fixed.

Syntax:



```
// For Loop Syntax
for(init; condition; inc/dec)
{
// statement/code
}
```

Note: If only one statement is being repeated, there is no need for the curly braces.

Jump Statement: There are three jump statements who supports Java:
Break, continue, return.

Break Statement: When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

There are three uses of Break Statement:

1. It terminates a statement sequence in a switch statement.
2. It can be used to exit a loop.
3. It can be used a "civilized" form of goto.

```java
// Using break with for loop
public class BreakExample2 {
    public static void main(String[] args) {
        for(int i=1; i<100; i++)
        {
            if(i==10)
            break;
            System.out.println("i: "+ i);
        }
            System.out.println("Loop Complete");
    }
}

```

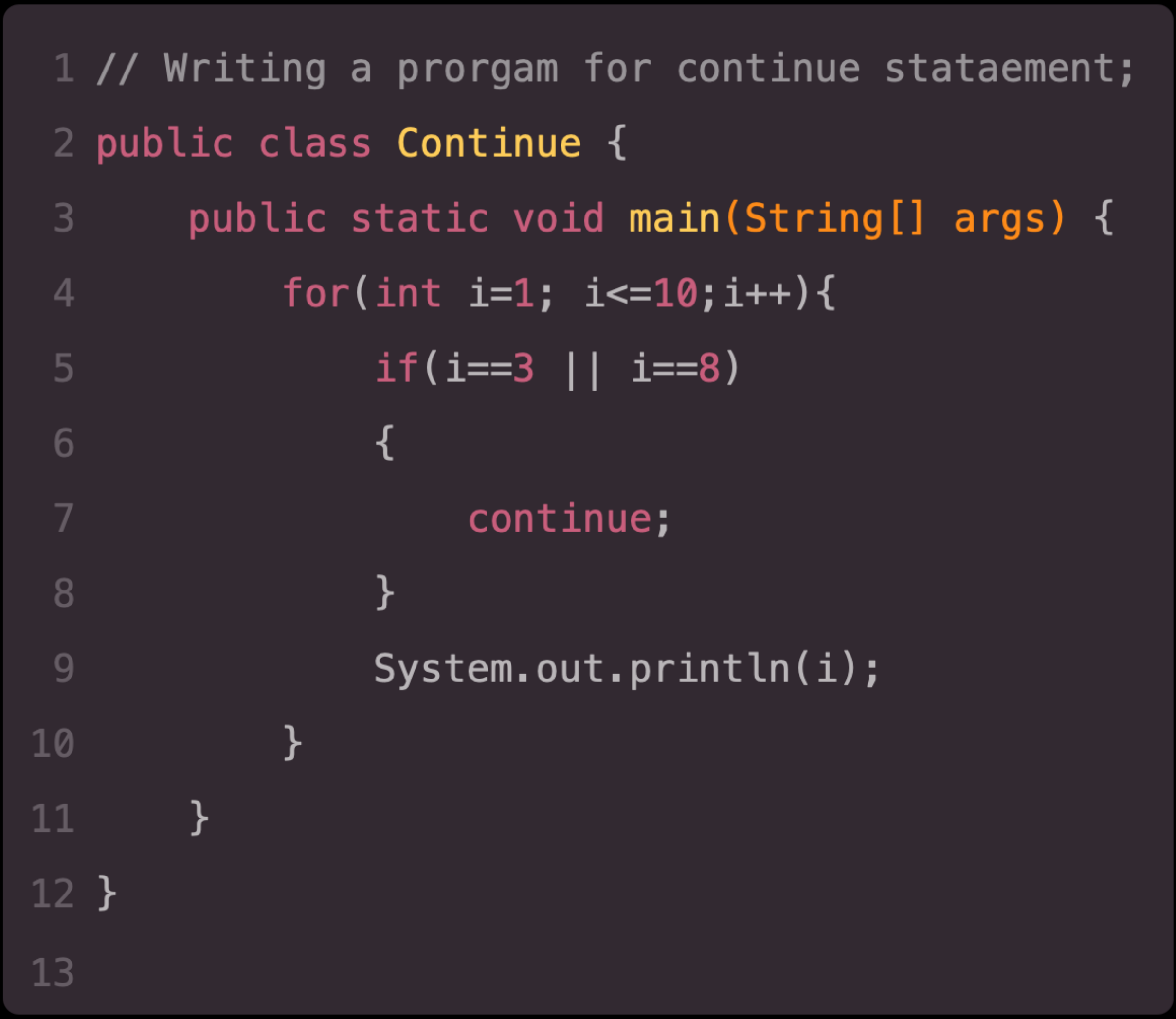
Continue Statement: The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop

✧ The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

Syntax:

```java
// Continue Syntax
jump-statement;
continue;
```

Example of Continue Statement:



```java
// Writing a prorgam for continue stataement;
public class Continue {
    public static void main(String[] args) {
        for(int i=1; i<=10;i++){
            if(i==3 || i==8)
            {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

Function OR Method: A function or method is a collection of statements designed to perform a specific task. It provides the reusability of code.

Why we use Function ?

Ans: Because, We can write a function or method once and use it many times, we do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

OR

We can build DRY(Don't Repeat Yourself) code, reusing the code we already wrote.

* We already use a function or method, when we write some codes that is main().

```
public class Main {
  access_modifier return_type method() {
       // Statements or Code
       return statements;
    }
 }
```

Method Declaration: The method declaration provides information about method attributes, such as visibility, return-type, name and arguments.

```
public int sum (int a, int b)
```

Where:

public: Access Specifier or Modifier

int: Return Type

sum: Method Name

(int a, int b): Parameter

Access Specifier or Modifier is the access types of the method. It specifies the visibility of the method.

There are four types of Access Modifier:

Public: The method is accessible by all classes when we use public modifier.

Private: When we use a private access modifier, the method is accessible only in the classes in which it is defined.

Protected: When we use protected access modifier, the method is accessible within the same package or subclasses in a different package.

Default: When we do not use any access modifier in the method declaration, Java uses default access modifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void etc..

Method Name: It is a unique name that is used to define the name of a method. It must be verb and start with a lowercase letter.
Single-word method name: addition(), substraction()
Multi-word method name: additionOperator()

Parameter: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name.
Arguments:

There are two types of Function or Method:

1. Standard Library Function (Predefined)
2. User- Defined Function

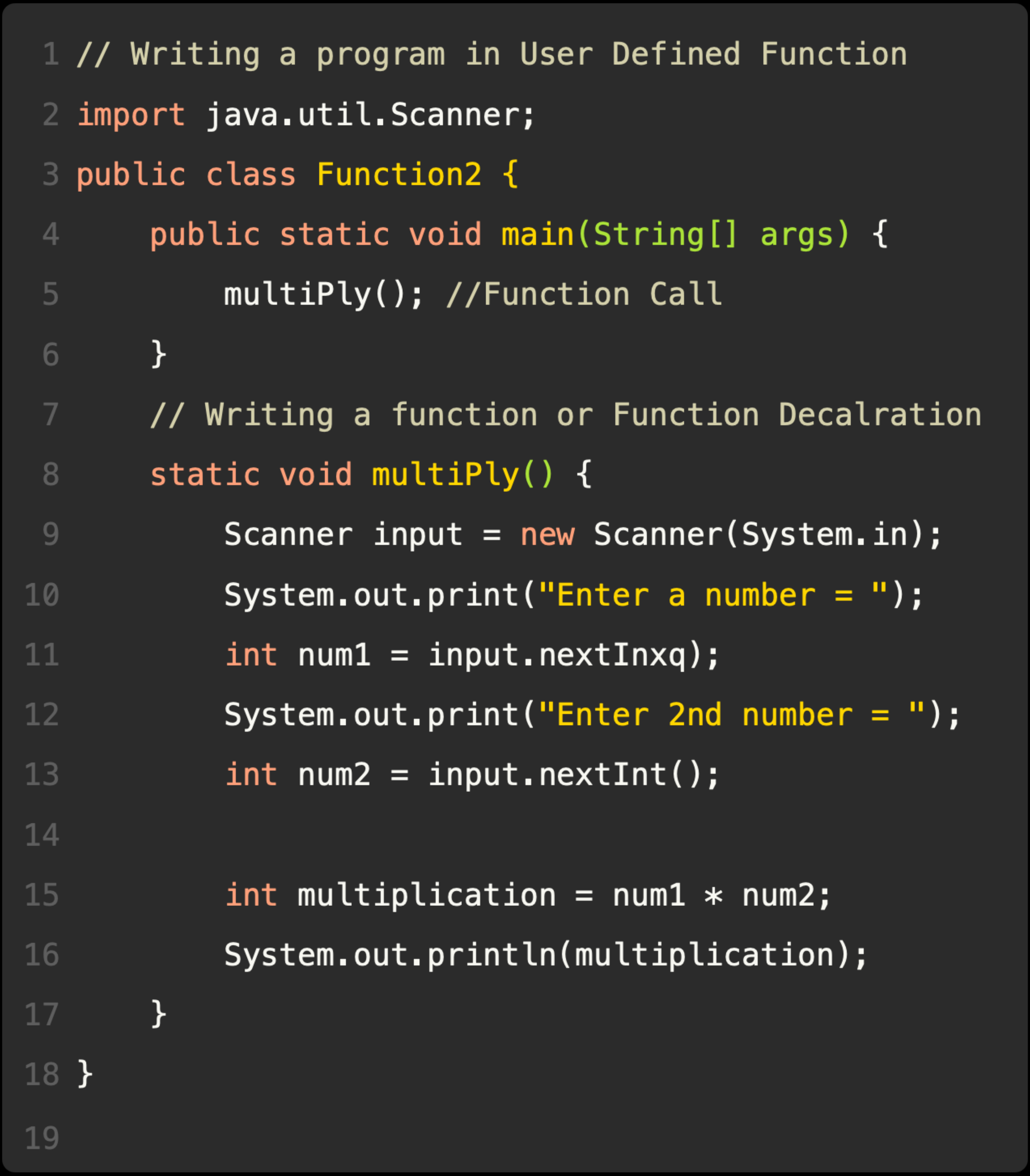
1. Standard Library Function (Predefined):
   Predefined functions or methods are method that is already defined in Java class libraries like in C or C++ etc...

* We can directly use these methods just by calling them in the program at any point. Some Predefined methods are length(), equals(), compareTo(), sqrt() etc.. When we call any of the Predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library. Each and every predefined method is defined inside a class. Such as print() method is defined in the java.io.PrintStream class.

2. User-Defined Function: The function or method that is written by the user or programmer is known as User-Defined Function.

```java
import java.lang.Math;
//Program on Predefind function or method
public class Function {
    public static void main(String[] args) {
        System.out.println(Math.max(21, 19));
        System.out.println(Math.min(21, 19));
    }
}
```

In the above program, we have used predefined functions like main(), print(), max(), min(), We have use these methods without declaration because they are predefined. The print() method is used for PrintStream class and the max() method is a method of the Math class that returns the greater of two numbers, and min() method is a method of the Math class that returns the smaller of two numbers.



```java
// Writing a program in User Defined Function
import java.util.Scanner;
public class Function2 {
    public static void main(String[] args) {
        multiPly(); //Function Call
    }
    // Writing a function or Function Decalration
    static void multiPly() {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a number = ");
        int num1 = input.nextInxq);
        System.out.print("Enter 2nd number = ");
        int num2 = input.nextInt();

        int multiplication = num1 * num2;
        System.out.println(multiplication);
    }
}

```

Function Overloading:

Two or more methods may have the same name if they differ in parameters. These methods are called overloaded methods and this feature is called Method overloading.

OR

If a class has multiple methods having same name but different in parameters, it is known as Method overloading.

Ex:

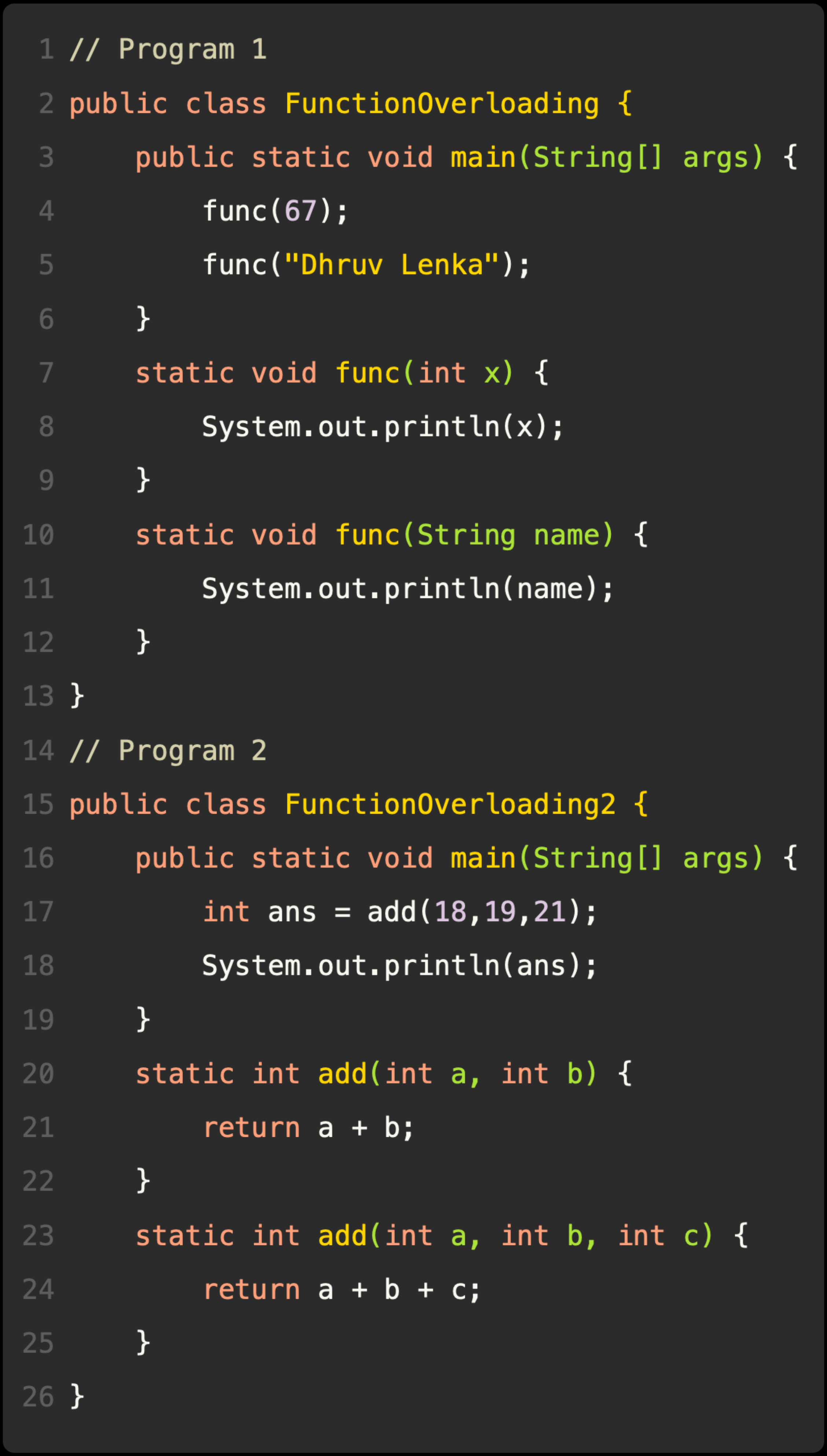
```
void add() {.....}
void add(int a, int b) {.....}
```

In above Ex: You can see we write same name but different parameters.

Note: Overloaded methods may have the same or different return types, but they must differ in parameters.

Advantage:

Method overloading increases the readability of the program.



```java
// Program 1
public class FunctionOverloading {
    public static void main(String[] args) {
        func(67);
        func("Dhruv Lenka");
    }
    static void func(int x) {
        System.out.println(x);
    }
    static void func(String name) {
        System.out.println(name);
    }
}
// Program 2
public class FunctionOverloading2 {
    public static void main(String[] args) {
        int ans = add(18,19,21);
        System.out.println(ans);
    }
    static int add(int a, int b) {
        return a + b;
    }
    static int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

Scope: Variables are only accessible inside the region they are created. This is called Scope.

Shadowing: If you declared an instance variable and also you declared a local variable having same name whenever you print or access it in the method. The value of local variable will be printed(Instance variable will be shadowed

Program

```java
public class FunctionShadowing {
    static int x = 21;
    public static void main(String[] args) {
      //  System.out.println(x);
        int x;
        x = 19;
        System.out.println(x);
        //fun();

    }
    static void fun() {
        System.out.print(x);
    }
}
```