# Basic Question and Answer 2

1. What is Operator and why we need operators?

* In programming, operators are special symbols that perform specific operations on values called operands. Think of them as verbs that tell the computer what to do with the data.
* We need operators because they are fundamental for:
* Performing calculations: Operators like +, -, \*, / allow us to do math.
* Making comparisons: Operators like >, <, == help us check relationships between values.
* Assigning values: The = operator lets us store data in variables.
* Logical operations: Operators like AND, OR, NOT enable complex decision-making in programs.

1. List down all the operators available in Java?

* Arithmetic Operators: Perform basic mathematical operations (+, -, \*, /, %, ++, --).
* Assignment Operators: Assign values to variables (=, +=, -=, \*=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=).
* Relational Operators: Compare two values (==, !=, >, <, >=, <=).
* Logical Operators: Combine or modify Boolean conditions (&&, ||, !).
* Bitwise Operators: Perform operations on individual bits of data (&, |, ^, ~, <<, >>, >>>).
* Unary Operators: Operate on a single operand (+, -, ++, --, !).
* Ternary Operator: A shorthand for if-else statements (condition ? value\_if\_true : value\_if\_false).

1. What is the difference between Relational & Logical Operators?

| **Aspect** | **Relational Operators** | **Logical Operators** |
| --- | --- | --- |
| **Purpose** | Compare values to establish a relationship (e.g., greater, less, equal). | Combine or invert Boolean expressions. |
| **Examples** | ==, !=, >, <, >=, <= | && (AND), ` |
| **Result** | Returns a Boolean value (true or false) based on comparison. | Returns a Boolean value based on logical operations. |
| **Usage** | Used for comparison between two values. | Used for decision-making with multiple conditions. |
| **Operands** | Works on numerical or comparable data types. | Works on Boolean expressions. |

1. What is the purpose of 'new' operator?

* The new operator in Java is used to create new objects or instances of a class. It allocates memory for the object on the heap and initializes the object by calling its constructor.
* Purpose of new Operator:

1. Object Creation: The primary purpose of new is to create an instance of a class.

Example: MyClass obj = new MyClass();

1. Memory Allocation: It allocates memory on the heap for the new object.
2. Constructor Invocation: It initializes the object by invoking the constructor of the class.

Example: new MyClass(parameters);

1. Array Creation: It is also used to create arrays in Java.

Example: int[] arr = new int[5];

1. What is the purpose of 'dot(.)' operator?

* The dot (.) operator is used to access members (such as properties, methods, or fields) of an object, structure, or class in programming languages like Java, C++, Python, and JavaScript.
* Purpose:

1. Access Properties: To retrieve or modify the attributes of an object.

Example: person.name (accesses the name property of the person object).

1. Invoke Methods: To call functions associated with an object or class.

Example: object.methodName() (calls a method named methodName on the object).

1. Namespace Resolution: In some languages, the dot operator resolves a namespace or package.

Example: java.util.Scanner (refers to the Scanner class in the java.util package).

1. What is the purpose of instance of operator?
2. What is the difference between "=" and "==" operators?

| **Feature** | **= (Assignment Operator)** | **== (Equality Operator)** |
| --- | --- | --- |
| **Functionality** | Used to assign a value to a variable. | Compares two values for equality. |
| **Data Types** | Works with variables of any type (int, double, object, etc.). | Compares primitives or object references. |
| **Return Type** | Does not return a value. | Returns a boolean (true or false). |
| **Example** | int x = 5; | if (x == 5) |

1. What is the purpose of Control Statements & List down all the control statements available in java?

* Control statements in Java determine the flow of program execution based on conditions or repetitions.
* Types of Control Statements:

1. Conditional Statements:
   1. if: Executes a block of code if the condition is true.
   2. if-else: Executes one block if the condition is true, otherwise another block.
   3. else if: Used to check multiple conditions.
   4. switch: Executes one block of code out of several cases based on a variable.
2. Looping Statements:
   1. for: Executes a block of code a specific number of times.
   2. while: Executes a block as long as a condition is true.
   3. do-while: Similar to while, but ensures at least one execution.
3. Jumping Statements:
4. break: Terminates a loop or switch statement.
5. continue: Skips the current iteration of a loop and moves to the next.
6. return: Exits a method and optionally returns a value.
7. Write a java program on if - else if - else ladder

* public class IfElseIfExample {

public static void main(String[] args) {

int marks = 85;

if (marks >= 90) {

System.out.println("Grade: A");

} else if (marks >= 75) {

System.out.println("Grade: B");

} else if (marks >= 50) {

System.out.println("Grade: C");

} else {

System.out.println("Grade: F");

}

}

}

1. Write a java program on 'switch' case

* public class SwitchExample {

public static void main(String[] args) {

int day = 3;

switch (day) {

case 1: System.out.println("Monday"); break;

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

case 3: System.out.println("Wednesday"); break;

default: System.out.println("Invalid day");

}

}

}

1. What is the difference between while & do-while loops

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| **Condition Check** | Condition is checked before the loop executes. | Condition is checked after executing the loop body once. |
| **Execution Guarantee** | May not execute if the condition is initially false. | Always executes the loop body at least once. |
| **Use Case** | Used when the number of iterations is not fixed, but the loop may not execute at all. | Used when the loop body must execute at least once, regardless of the condition. |

1. Write a java program on 'while' loop

* public class WhileExample {

public static void main(String[] args) {

int i = 1;

while (i <= 5) {

System.out.println(i);

i++;

}

}

}

1. What is the difference between 'while' loop & 'for' loop

| **Feature** | **while Loop** | **for Loop** |
| --- | --- | --- |
| **Syntax** | Simplified syntax for loops with unknown iterations. | Combines initialization, condition, and increment in a single line. |
| **Structure** | Initialization and increment are handled outside the loop condition. | All loop control elements are included in the loop header. |
| **Use Case** | Preferred when the condition depends on external factors. | Preferred for loops with a fixed number of iterations. |

1. Write a java program on 'for' loop

* public class ForExample {

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

System.out.println(i);

}

}

}

1. Write a java program on 'nested for loop'

* public class NestedForExample {

public static void main(String[] args) {

for (int i = 1; i <= 3; i++) {

for (int j = 1; j <= 3; j++) {

System.out.println("i=" + i + ", j=" + j);

}

}

}

}

1. What is the difference between 'break' and 'continue' & 'return' keywords

| **Aspect** | **break** | **continue** | **return** |
| --- | --- | --- | --- |
| **Purpose** | Exits the nearest enclosing loop (for, while, do-while) or switch statement immediately. | Skips the current iteration of a loop and continues with the next iteration. | Exits the current method and optionally returns a value to the calling method. |
| **Scope** | Can be used within loops or switch statements. | Can only be used within loops. | Can only be used in methods. |
| **Effect** | Ends the loop or switch entirely, and control moves to the next statement after the loop or switch. | Skips the remaining code in the loop for the current iteration and jumps to the next iteration. | Ends the method execution and transfers control back to the caller. |
| **Usage Example** | if (x == 5) break; | if (x == 5) continue; | return x; |
| **Execution Control** | Terminates the loop or switch. | Skips the current iteration. | Ends the method execution and optionally passes back a value. |

1. Write a java program using 'break' keyword

* public class BreakExample {

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

if (i == 3) break;

System.out.println(i);

}

}

}

1. Write a java program using 'continue' keyword

* public class ContinueExample {

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

if (i == 3) continue;

System.out.println(i);

}

}

}