Assignment
Operators and Loops

Q.1 In Java, conditional operators are used to perform operations based on specific conditions. These include the ternary operator and logical operators that evaluate conditions. Here's a detailed explanation:

1. Ternary Operator (`? :`)

- The ternary operator is a shorthand for `if-else` statements.

  Syntax:

  condition ? value_if_true : value_if_false;

  It evaluates a condition:

- If `true`, it returns the value after the `?`.
- If `false`, it returns the value after the `:`.

2. Logical Operators

Logical operators are used to combine or invert conditional expressions.

Types:

1. **AND (&&)**:
   - Returns `true` if **both** conditions are true.
   -

Example:
int a = 10, b = 20;

if (a > 5 && b > 15) {

System.out.println("Both conditions are true");

}

2 **OR (`||`)**:

- Returns `true` if **at least one** condition is true.
- **Example**
  int a = 10, b = 5

```
if (a > 15 || b > 0) {

   System.out.println("At least one condition is true");

}
```

3 NOT (!):

- Inverts the value of a boolean expression.
- Example

```
boolean isJavaFun = true;

if (!isJavaFun) {

   System.out.println("Java is not fun");

} else {

   System.out.println("Java is fun!");

}
```

3. Relational/Comparison Operators

These are often used in conditional expressions to compare two values.

- == (Equal to)
- ! = (Not equal to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

Example:

```
int a = 10, b = 20;

if (a < b) {

   System.out.println("a is less than b");

}
```

Q.2 1. Unary Operators

- **Definition**: Operate on **one operand**.
- **Examples in Java**:
    - **Unary Plus (`+`)**: Indicates a positive value.
    - **Unary Minus (`-`)**: Negates a value.
    - **Increment (`++`)**: Increases a value by 1.
    - **Decrement (`--`)**: Decreases a value by 1.
    - **Logical NOT (`!`)**: Inverts a boolean value.

2. Binary Operators

- **Definition**: Operate on **two operands**.
- **Examples in Java**:
    - **Arithmetic Operators**: `+`, `-`, `*`, `/`, `%`
    - **Relational/Comparison Operators**: `==`, `!=`, `>`, `<`, `>=`, `<=`
    - **Logical Operators**: `&&`, `||`
    - **Bitwise Operators**: `&`, `|`, `^`, `<<`, `>>`, `>>>`
    - **Assignment Operators**: `=`, `+=`, `-=`, `*=`, `/=`, etc.

3. Ternary Operator

- **Definition**: Operates on **three operands**.
- **Example in Java**: The ternary conditional operator (`?:`).

Q.3 The `switch` case is a control statement in Java used to execute one block of code out of multiple options based on the value of a variable or expression. It is an alternative to a series of `if-else-if` statements when there are multiple conditions to evaluate.

Syntax of `switch` Case

```
switch(expression) {

    case value1:

        // Code to execute if expression == value1

        break; // Optional, prevents fall-through

    case value2:

        // Code to execute if expression == value2

        break;

    ...

    default:
```

```
    // Code to execute if no case matches
}
```

Q.4 In Java, arithmetic operations follow the standard operator precedence rules. These rules determine the order in which operations are performed in an expression.

Operator Precedence Table

Precedency		Associativity
`( )`	**Parentheses**	**Left to right**
`+, -` **(Unary)**	**Unary plus and minus**	**Right to left**
`++, --`	**Pre/post-increment/decrement**	**Right to left**
`*, /, %`	**Multiplication, division, modulo**	**Left to right**
`+, -`	**Addition and subtraction**	**Left to right**

Q.5 Conditional statements in Java are used to perform actions based on certain conditions or expressions. They allow your program to make decisions, enabling different actions depending on the conditions evaluated to `true` or `false`. These statements are the building blocks for controlling the flow of execution in a Java program.

Types of Conditional Statements in Java

1. **`if` Statement**
    - The `if` statement allows you to execute a block of code only if a specified condition evaluates to `true`.
      Syntax:

      ```
      if (condition) {

          // Block of code to be executed if the condition is true

      }
      ```

2. **`if-else` Statement**

- The `if-else` statement provides an alternative block of code to execute if the condition is `false`.

Syntax:

```
if (condition) {

    // Block of code to be executed if the condition is true

} else {

    // Block of code to be executed if the condition is false

}
```

3.**`else-if` Ladder**

- The `else-if` ladder allows you to test multiple conditions in sequence.
- If one condition evaluates to `true`, its associated block of code is executed. If not, the next condition is evaluated, and so on.

Syntax:

```
if (condition1) {
    // Block of code for condition1
} else if (condition2) {
    // Block of code for condition2
} else {
    // Block of code if no condition is true
}
```

4.`switch` Case Statement

- The `switch` statement is used to execute one out of many code blocks based on the value of an expression. It's an alternative to multiple `if-else-if` statements when you need to evaluate a single expression against different constant values.

Syntax:

```
switch (expression) {
    case value1:
        // Code to execute if expression == value1
        break;
    case value2:
        // Code to execute if expression == value2
        break;
    default:
        // Code to execute if no case matches
}
```

Q.6 `if-else` Statement

- The `if-else` statement provides an alternative block of code to execute if the condition is `false`.

Syntax:
if (condition) {

// Block of code to be executed if the condition is true

} else {

// Block of code to be executed if the condition is false

}

Q.7 In Java, **iterative statements** (also known as **looping statements**) are used to repeatedly execute a block of code based on a condition. Java has three primary types of iterative (looping) statements:

1. `for` Loop

- **Definition**: The `for` loop is used when the number of iterations is known beforehand. It's ideal for iterating over a range of values or a collection of items.
- **Syntax**:
  for (initialization; condition; update) {

  // Code to be executed repeatedly

  }

2. `while` Loop

- **Definition**: The `while` loop is used when you want to execute a block of code as long as a condition is true. The condition is evaluated before each iteration.
- **Syntax**:

  while (condition) {

  // Code to be executed repeatedly

  }

3. `do-while` Loop

- **Definition**: The `do-while` loop is similar to the `while` loop, but the condition is evaluated after the loop body is executed. This guarantees that the loop will run **at least once** before checking the condition.
- **Syntax**:

```
do {

    // Code to be executed repeatedly

} while (condition);
```

Q.8 The differences between these two loop are:-

Loop Type	Condition Check	Best Use Case
`for` loop	Condition checked before each iteration	When the number of iterations is known ahead of time.
`do-while` loop	Condition checked after each iteration	When you need the loop to run at least once, regardless of the condition.

Q.9 A program to print numbers from 1 to 10.

```
public class Num
{
public static void main(String args[ ])
{ for(int i=1;i<11;i++)
{
System.out.print(i);
System.out.print(“ “);
}
}
}
```