1. What are the conditional operators in java?

The are used when a condition comprises more than one Boolean expression. For instance, if we want to print a number only if it is greater than 2 and less than 5, then we will use conditional operators to combine the 2 expressions. We have 3 types of conditional operators

1) Logical-AND
2) Logical-OR and
3) Temporary

2. What are the types of operators based on the number of operands?

Operators in programming are like tools we use to perform actions on data. They come in different types, and one way to classify them is by the number of values they work with. Let me break it down for us:
1. Unary Operators: Imagine you have one piece of data, like a number. Unary operators are like actions you can perform on just that one piece of data. For example, think of the negative sign (-) in math. If you have the number 5, adding a negative sign turns it into -5. That's a unary operation because it's working on just one value.
2. Binary Operators: Now, picture having two pieces of data, like two numbers. Binary operators are the actions we can perform on these pairs of data. For instance, addition (+), subtraction (-), multiplication (*), and division (/) are all binary operators. They need two numbers to perform their operation. For example, in 5 + 3, the plus sign is a binary operator because it's combining two values, 5 and 3, to give us 8.
3. Ternary Operator: This one is a bit unique. It's called "ternary" because it works with three values. In Java, the only ternary operator is the conditional operator (also known as the ternary conditional operator). It's like a compact way of writing an if-else statement. It takes three operands: a condition, a value to return if the condition is true, and a value to return if the condition is false. It's like saying, "If this condition is true, give me this value; otherwise, give me that value."

3. What is the use of switch case in java programming?

The `switch` statement in Java is used to execute different blocks of code based on the value of a variable or expression. It provides a cleaner alternative to using multiple `if-else` statements, especially when dealing with multiple possible conditions.

Here are some key points about the use of `switch` case in Java programming:

1. **Multi-branch Selection**: `switch` allows you to specify multiple possible values for a variable or expression and execute different code blocks based on which value matches.

2. **Simplicity and Readability**: Using `switch` can make code more readable and easier to understand, especially when there are many possible conditions. It can make your code cleaner and more organized compared to a series of nested `if-else` statements.

3. **Efficiency**: Under the hood, `switch` statements can be more efficient than equivalent `if-else` chains, especially when dealing with a large number of conditions. This is because the Java compiler can optimize the `switch` statement to use a jump table or lookup switch, leading to faster execution.

Here's a basic syntax of a `switch` statement in Java:

```
switch (expression) {
    case value1:
```

```
        // code to be executed if expression matches value1
        break;
    case value2:
        // code to be executed if expression matches value2
        break;
    // more cases...
    default:
        // code to be executed if none of the above cases match
}
```

In this syntax:

- `expression` is the variable or expression whose value will be compared.

- `case value: ` defines the possible values that `expression` can take.

- The `break; ` statement is used to exit the `switch` block once a matching case is found. If omitted, execution will continue to the next case, which is called "fall-through."

- `default: ` is optional and provides a default case to be executed if none of the specified cases match the value of the expression.

4. What are the priority levels of arithmetic operation in java?

In Java, arithmetic operations follow the standard order of operations, also known as operator precedence. The priority levels, from highest to lowest precedence, are as follows:

1. **Parentheses**: Operations enclosed within parentheses are evaluated first. This allows you to explicitly control the order of operations.

2. **Multiplication, Division, and Modulus**: Multiplication (`*`), division (`/`), and modulus (`%`) operations have the next highest precedence. They are evaluated left to right.

3. **Addition and Subtraction**: Addition (`+`) and subtraction (`-`) have the lowest precedence among the arithmetic operators. They are evaluated left to right.

Here's a simple example to illustrate operator precedence:

```
int result = 10 + 5 * 2; // Result will be 20, because multiplication has higher precedence than addition
```

To change the order of evaluation, you can use parentheses:

int result = (10 + 5) * 2; // Result will be 30, because addition inside parentheses is evaluated fir

Understanding operator precedence is important for writing expressions that produce the desired results in Java programs.

5. What are the conditional statements and use of conditional statements in java?

Conditional statements in Java are used to make decisions based on certain conditions. They allow a program to execute different blocks of code depending on whether a condition is true or false. The main conditional statements in Java are:

1. if statement: The if statement is used to execute a block of code only if a specified condition is true.

```
if (condition) {
   // block of code to be executed if the condition is true
}
```

2. if-else statement: The if-else statement is used to execute one block of code if the condition is true and another block of code if the condition is false.

```
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. if-else-if ladder: The if-else-if ladder allows you to check multiple conditions and execute a block of code corresponding to the first condition that is true.

```
if (condition1) {
   // block of code to be executed if condition1 is true
} else if (condition2) {
   // block of code to be executed if condition2 is true
} else if (condition3) {
   // block of code to be executed if condition3 is true
} else {
   // block of code to be executed if none of the conditions are true
}
```

6. What is the syntax of if else statements?

To initialize the if else statements we follow the following syntax:

```
if(condition){ statement
} else{
Statement
}
```

7. What are the 3 types of iterative statements in java?

In Java, there are three types of iterative statements:

for loop: The for loop provides a compact way to iterate over a range of values. It consists of initialization, condition, and iteration expression.

```
for (initialization; condition; iteration) {
// statements to be executed repeatedly
```

while loop: The while loop repeatedly executes a block of statements as long as the specified condition is true. It is used when the number of iterations is not known beforehand.
while (condition) {
// statements to be executed repeatedly
}
do-while loop: The do-while loop is similar to the while loop, but the condition is evaluated after the execution of the block of statements, ensuring that the block of statements is executed at least once.
do {
// statements to be executed repeatedly
} while (condition);

8. Write the difference between for loop and do while loop?

For Loop	Do-While Loop

Ulike while loop, in for loop we have in the for header.
Syntax
for(init-statement; condition; final expression){
statement
}
	Unlike while and for loop, do while loop tests for the condition at the end of each of each execution for the next iteration. In other words, the loop is expected at least one before the condition is checked. Other than that everything is the same as in the while loop
	Syntax
	do{statement;
	}while(condition);

9. Write a program to print numbers from 1 to 10

```java
public class Printing1to10 {
    public static void main(String[] args) {
        for(int i = 1; i <= 10; i++){
            System.out.println(i);
        }
    }
}
```