In Java, the short-circuit operators && (logical AND) and || (logical OR) are used to perform logical operations on boolean expressions. The short-circuit operators have an interesting behavior where the second operand is not evaluated if the result can be determined based on the value of the first operand.

Short-Circuit Logical AND (&&):

The short-circuit logical AND operator (&&) evaluates the left operand first. If the left operand is false, the overall result is false, and the right operand is not evaluated because the result is already determined. However, if the left operand is true, the right operand is evaluated, and the overall result is determined by the value of the right operand.
Example:

```
boolean a = true;
boolean b = false;
boolean result = a && b;
```

In this example, since a is true, the right operand b is evaluated. The overall result is false because b is false.

Short-Circuit Logical OR (||):

The short-circuit logical OR operator (||) also evaluates the left operand first. If the left operand is true, the overall result is true, and the right operand is not evaluated. If the left operand is false, the right operand is evaluated, and the overall result is determined by the value of the right operand.
Example:

```
boolean a = false;
boolean b = true;
boolean result = a || b;
```

In this example, since a is false, the right operand b is evaluated. The overall result is true because b is true.

The short-circuit operators are particularly useful in scenarios where evaluating the right operand may lead to an error or unnecessary computation when the result can be determined based on the left operand alone. They help improve efficiency and prevent unnecessary evaluation.

In Java, the single & (bitwise AND) and | (bitwise OR) operators are used to perform bitwise operations on integer types. Unlike the short-circuit logical operators && and ||, the single & and | operators do not short-circuit and always evaluate both operands.

Bitwise AND (&):

The bitwise AND operator (&) performs a bitwise AND operation between the corresponding bits of the operands. It returns a value where each bit is set to 1 only if the corresponding bits of both operands are 1. Otherwise, the bit is set to 0.
Example:

```
int a = 10;    // Binary: 1010
int b = 6;     // Binary: 0110
int result = a & b;
```

In this example, the bitwise AND operation is performed on a and b. The result will be 2 because the corresponding bits at each position are 1 and 0, which evaluates to 0, except for the second position where both bits are 1.

Bitwise OR (|):

The bitwise OR operator (|) performs a bitwise OR operation between the corresponding bits of the operands. It returns a value where each bit is set to 1 if at least one of the corresponding bits of the operands is 1. Otherwise, the bit is set to 0.
Example:

```
int a = 10;    // Binary: 1010
int b = 6;     // Binary: 0110
int result = a | b;
```

In this example, the bitwise OR operation is performed on a and b. The result will be 14 because the corresponding bits at each position have at least one 1 bit.

Note that the single & and | operators are different from the short-circuit logical operators && and ||. The single & and | operators operate on individual bits, while the short-circuit logical operators operate on boolean values and provide short-circuiting behavior.

```java
package com.hdfc.operator;

public class ShortCircuitOpeartorTest {


    public static void main(String[] args) {

        /**
         *  Short circuit Opearator : && (AND) and || (OR)
         *
         *   && -> if first expression is false then second is not evaluated.
         * true && true = true
         * true && false = false
         * false && true = false
         * false && false = false
         *
         *  || (OR) Operator
         *  if first expression is true then second will not be evaluated.
         *
         *  true || true =  true
         *  false || true = true
         *  true || false = true
         *  false || false = false
         *
         *
         *  Single & operator
         *  : all the expression are evaluated
         * true && true = true
         * true && false = false
         * false && true = false
         * false && false = false
         *
         *  |  (OR) Operator
         *   all the expression are evaluated
         *  true || true =  true
         *  false || true = true
         *  true || false = true
         *  false || false = false
         *
         */

        int a = 10;
        int b = 20;
        if (a > 100 && ++b > 5) {
            System.out.println("condition");
        }
        System.out.println(b);


        int a2 = 10;
        int b2 = 20;
        if (a2 > 100 || ++b2 > 5) {
            System.out.println("condition");
        }
        System.out.println(b2);


        int a3 = 10;
        int b3 = 20;
        if (a < 100 & ++b > 5) {
            System.out.println("condition");
        }
        System.out.println(b3);
        int a4 = 10;
        int b4 = 20;
        if (a4 > 100 | ++b4 > 5) {
            System.out.println("condition");
        }
        System.out.println(b4);



    }
}
```