**Truth Tables**

Truth tables let us combine different values and have different solutions to do based on the outcomes. Truth tables evaluate Boolean expressions. We combine inputs and produce an output based on which logical operators we are using.

|  |  |  |
| --- | --- | --- |
| It's cold | It's snowing | It's cold and it's snowing |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

**Logical Operators:**

**AND - &&**: combines two boolean values and returns a boolean which is true *if and only if* both of its operands are true

If you have an expression of the form *A && B*then Java evaluates *A*first. If *A*is true, then Java evaluates *B.*If *A*is false, then Java does not evaluate *B* (Short-circuit behavior)*.*

Referred to as the Conditional AND.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A && B** |
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

**OR - ||**: combines two boolean variables or expressions and returns a result that is true *if either or both* of its operands are true

If you have an expression of the form *A || B*then Java evaluates *A*first. If *A*is true, then Java does not evaluate *B.*If *A*is false, then Java does evaluate *B* (Short-circuit behavior)*.*

Referred to as the Conditional OR.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A || B** |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

**NOT - !**:  *reverses* the value of a boolean expression. Thus if b is true !b is false. If b is false !b is true. Unary logical negation.

|  |  |
| --- | --- |
| **A** | **!A** |
| T | F |
| F | T |

**Ternary Conditional Operator (**Shorthand for the if … else statement**)**

The conditional operator **? :**uses the boolean value of one expression to decide which of the two other expressions should be evaluated.

exp1 ? exp2 : exp3

If exp1 is true, then exp2 is chosen. If exp1 is false then exp3 is chosen.

**Bitwise Operators**

Evaluated left to right. Applies to Boolean and integer types.

A bitwise operation operates on one or more bit patterns or binary numerals at the level of their individual bits, results are on a bit-by-bit basis.

Both operands are always evaluated.

**EXCLUSIVE OR - ^**: “*one or the other, but not both*”

(does not short-circuit, looks at each side)

This is usefule because it can tell whether each bit is different)

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A ^ B** |
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | F |

**INCLUSIVE OR - |**: (any F = a True) “*when both are not T*”

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A | B** |
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | T |

**Bitwise AND - &**: (any T = a False) “*if and only if, both are F*”

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A & B** |
| T | T | F |
| T | F | F |
| F | T | F |
| F | F | T |

**Bitwise COMPLEMENT OPERATOR- ~:** flips the bits of the variable, T to F or F to T

**Shift Operators**

Shift operators are applied only to integer types.

x << k : shift the bits in x, k places to the left

x >> k : shift the bits in x, k places to the right filling in with the highest bit on the left hand side

x >>> k : shift the bits in x, k places to the right and fill in with 0