**Operators**

**Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Result** | |
| + | Addition |  |
| – | Subtraction (also unary minus) |  |
| \* | Multiplication |  |
| / | Division |  |
| % | Modulus |  |
| ++ | Increment |  |
| += | Addition assignment |  |
| –= | Subtraction assignment |  |
| \*= | Multiplication assignment |  |
| /= | Division assignment |  |
| %= | Modulus assignment |  |
| –– | Decrement |  |

**BitWise Operators**

|  |  |
| --- | --- |
| Operator | Result |
| ~ | Bitwise unary NOT |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| >> | Shift right |
| >>> | Shift right zero fill |
| << | Shift left |
| &= | Bitwise AND assignment |
| |= | Bitwise OR assignment |
| ^= | Bitwise exclusive OR assignment |
| >>= | Shift right assignment |
| >>>= | Shift right zero fill assignment |
| <<= | Shift left assignment |

**The Bitwise Logical Operators**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **A| B** | **A& B** | **A^ B** | **~A** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

**Relational Operators**

|  |  |
| --- | --- |
| **Operator** | **Result** |
| == | Equal to |
| != | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

**Boolean Logical Operators**

|  |  |
| --- | --- |
| **Operator** | **Result** |
| & | Logical AND |
| | | Logical OR |
| ^ | Logical XOR (exclusive OR) |
| || | Short-circuit OR |
| && | Short-circuit AND |
| ! | Logical unary NOT |
| &= | AND assignment |
| |= | OR assignment |
| ^= | XOR assignment |
| == | Equal to |
| != | Not equal to |
| ?: | Ternary if-then-else |

**Short-Circuit Logical Operators**

if (denom != 0 && num / denom > 10)

if(c==1 & e++ < 100) d = 100;

**The Assignment Operator**

=

**The ? Operator (ternary)**

Java includes a special *ternary* (three-way) *operator* that can replace certain types of if-then-else statements. This operator is the **?**. It can seem somewhat confusing at first, but the **?** can be used very effectively once mastered. The **?** has this general form:

*expression1* **?** *expression2* **:** *expression3*

ratio = denom == 0 ? 0 : num / denom;

**Operator Precedence**

|  |  |  |  |
| --- | --- | --- | --- |
| **Highest** |  |  |  |
| () | [] | . |  |
| ++ | –– | ~ | ! |
| \* | / | % |  |
| + | – |  |  |
| >> | >>> | << |  |
| > | >= | < | <= |
| == | != |  |  |
| & |  |  |  |
| ^ |  |  |  |
| | |  |  |  |
| && |  |  |  |
| || |  |  |  |
| ?: |  |  |  |
| = | op= |  |  |
| **Lowest** |  |  |  |

**Control Statements**

**Java’s Selection Statement**

1. **if**
2. **Nested ifs**

if(i == 10) {

if(j < 20) a = b;

if(k > 100) c = d; // this if is

else a = c; // associated with this else

}

else a = d;

1. **The if-else-if ladder**

if(condition) statement;

else if(condition) statement;

else if(condition) statement;

.

.

. else

statement;

Example Program:

class IfElse {

public static void main(String args[]) {

int month = 4; // April

String season;

if(month == 12 || month == 1 || month == 2)

season = "Winter";

else if(month == 3 || month == 4 || month == 5)

season = "Spring";

else if(month == 6 || month == 7 || month == 8)

season = "Summer";

else if(month == 9 || month == 10 || month == 11)

season = "Autumn";

else

season = "Bogus Month";

System.out.println("April is in the " + season + ".");

}

}

1. **Switch**

The switch statement is Java’s multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative than a large series of if-else-if statements. Here is the general form of a switch statement:

switch (expression) { case value1:

// statement sequence

break; case value2:

// statement sequence

break; .

. .

case valueN:

// statement sequence

break; default:

// default statement sequence }

The *expression* must be of type **byte**, **short**, **int**, or **char**; each of the *values* specified in the **case** statements must be of a type compatible with the expression. (An enumeration value can also be used to control a **switch** statement. Enumerations are described in Chapter 12.) Each **case** value must be a unique literal (that is, it must be a constant, not a variable). Duplicate **case** values are not allowed.

The **switch** statement works like this: The value of the expression is compared with each of the literal values in the **case** statements. If a match is found, the code sequence following that **case** statement is executed. If none of the constants matches the value of the expression, then the **default** statement is executed. However, the **default** statement is optional. If no **case** matches and no **default** is present, then no further action is taken.

The **break** statement is used inside the **switch** to terminate a statement sequence. When a **break** statement is encountered, execution branches to the first line of code that follows the entire **switch** statement. This has the effect of “jumping out” of the **switch**.

Here is a simple example that uses a **switch** statement:

Example:

// Java Program to check the size

// using the switch...case statement

class Main {

public static void main(String[] args) {

int number = 44;

String size;

// switch statement to check size

switch (number) {

case 29:

size = "Small";

break;

case 42:

size = "Medium";

break;

// match the value of week

case 44:

size = "Large";

break;

case 48:

size = "Extra Large";

break;

default:

size = "Unknown";

break;

}

System.out.println("Size: " + size);

}

}

Example: Season with Switch

// An improved version of the season program.

class Switch {

public static void main(String args[]) {

int month = 4;

String season;

switch (month) {

case 12:

case 1:

case 2:

season = "Winter";

break;

case 3:

case 4:

case 5:

season = "Spring";

break;

case 6:

case 7:

case 8:

season = "Summer";

break;

case 9:

case 10:

case 11:

season = "Autumn";

break;

default:

season = "Bogus Month";

}

System.out.println("April is in the " + season + ".");

}

}

**Iteration Statements**

1. **While**

while(condition) { // body of loop

}

1. **Do-while**

int i=10;

do{

System.out.println("Hello");

}while(i>11);

1. **For**
   1. **For variations**

**Jump Statements**

1. **Break**
2. **Continue**

class Continue {

public static void main(String args[]) {

for(int i=0; i<10; i++) {

System.out.print(i + " ");

if (i%2 == 0) continue;

System.out.println("");

} }

}

1. **Return**

class Return {

public static void main(String args[]) {

boolean t = true;

System.out.println("Before the return.");

if(t) return; // return to caller

System.out.println("This won't execute.");

}

}

**Arrays**

**One-Dimensional Arrays**

A *one-dimensional array* is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is

*type var-name*[ ];

int a[]= {1,2,3,4};

int b[] = new int[4];

int []c= {1,2,3,4};

int[] d= {1,2,3,4};

A diagram of a graph

Description automatically generated

A diagram of a number of marks

Description automatically generated with medium confidence

A diagram of a number of squares

Description automatically generated

Program: Find maximum and minimum in an array

**Multidimensional Arrays**

int twoD[][] = new int[3][3];

A diagram of a number of columns

Description automatically generated