Methods

Example

Suppose that you need to find the sum of integers from 1 to 10, from 20 to 37, and from 35 to 49, respectively. You may write the code as follows:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 37; i++)
  sum += i;
System.out.println("Sum from 20 to 37 is " + sum);

sum = 0;
for (int i = 35; i <= 49; i++)
  sum += i;
System.out.println("Sum from 35 to 49 is " + sum);
```

Using Methods

```
1  public static int sum(int i1, int i2) {
2    int result = 0;
3    for (int i = i1; i <= i2; i++)
4      result += i;
5
6    return result;
7  }
8
9  public static void main(String[] args) {
10   System.out.println("Sum from 1 to 10 is " + sum(1, 10));
11   System.out.println("Sum from 20 to 37 is " + sum(20, 37));
12   System.out.println("Sum from 35 to 49 is " + sum(35, 49));
13 }
```

Eclipse

```
1  public class Meth{
2
3  public static void main(String[] args) {
4
5   System.out.println("Sum from 1 to 10 is " + sum(1, 10));
6   System.out.println("Sum from 20 to 37 is " + sum(20, 37));
7   System.out.println("Sum from 35 to 49 is " + sum(35, 49));
8
9   }
10
11 public static int sum(int i1, int i2) {
12       int result = 0;
13      for (int i = i1; i <= i2; i++)
14       result = result + i;
15
16       return result;
17       }
18
19 }
20
21
```

Problems @ Javadoc Declaration Console

erminated> Meth [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Nov 15, 2022, 8:08:51 A

```
um from 1 to 10 is 55
um from 20 to 37 is 513
um from 35 to 49 is 630
```

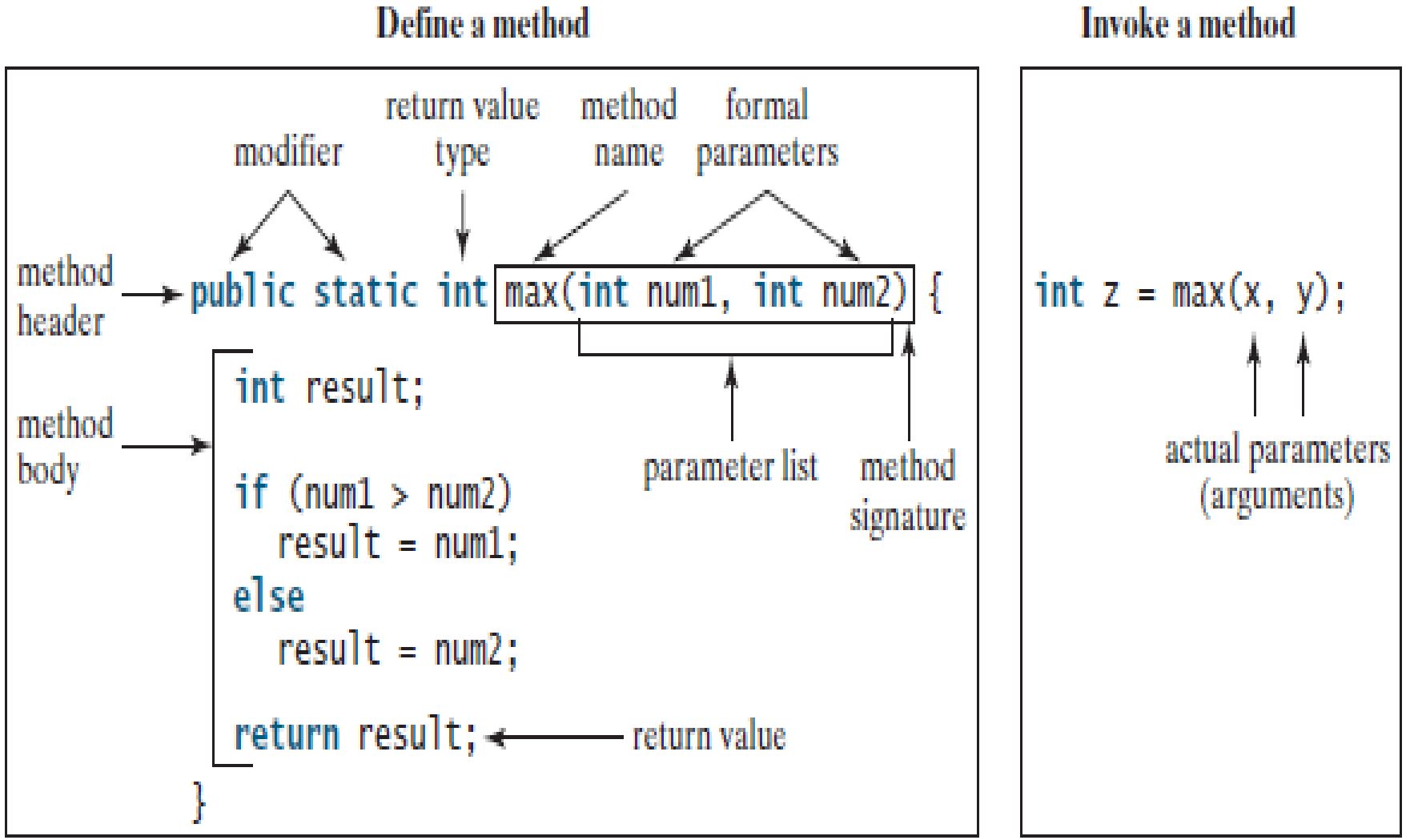
Method Definition

A method is a collection of statements that are grouped together to perform an operation.

A method definition consists of its *method name, parameters, return value type, and body.*

Method Definition

Example of a method defined to find the larger between two integers.



Method Definition

If a method performs desired operations without returning a value. In this case, the **return value type** is the keyword **void**.

If a method returns a value, it is called a *value-returning method;* otherwise it is called a *void method.*

```
public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.print("Enter two numbers: ");
              int x = input.nextInt();
              int y = input.nextInt();
 System.out.println("The larger number is " + Math.max(x, y));
 }
```

```
public static int max(int num1, int num2) {
        int result;
        if (num1 > num2)
                result = num1;
        else
                result = num2;
        return result;
        }
```

CAUTION

A return statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

(a)

Should be

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

(b)

To fix this problem, delete *if (n < 0)* in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

Example

What is wrong in the following program?

```
class Test {
    public static void main(String[] args) {
        System.out.println(print(7));

    }

    public static int print(int x){

        if (x>2)
          return 3;
        if (x>5)
         return 8;
    }
}
```

missing return statement

Passing Parameters

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

Suppose you invoke the method using

nPrintln(“Hello”, 5);

What is the output?

Suppose you invoke the method using

nPrintln(5, “Hello”);

What is the output?

Overloading Methods

Overloading methods enables you to define the methods with the same name as long as their signatures are different.

Overloading the `max` Method

```
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

public static int max(int num1, int num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

LISTING 6.9 TestMethodOverloading.java

```
1  public class TestMethodOverloading {
2    /** Main method */
3    public static void main(String[] args) {
4      // Invoke the max method with int parameters
5      System.out.println("The maximum of 3 and 4 is "
6        + max(3, 4));
7
8      // Invoke the max method with the double parameters
9      System.out.println("The maximum of 3.0 and 5.4 is "
10       + max(3.0, 5.4));
11
12     // Invoke the max method with three double parameters
13     System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
14       + max(3.0, 5.4, 10.14));
15   }
16
17   /** Return the max of two int values */
18   public static int max(int num1, int num2) {
19     if (num1 > num2)
20       return num1;
21     else
22       return num2;
23   }
24
25   /** Find the max of two double values */
26   public static double max(double num1, double num2) {
27     if (num1 > num2)
28       return num1;
29     else
30       return num2;
31   }
32
33   /** Return the max of three double values */
34   public static double max(double num1, double num2, double num3) {
35     return max(max(num1, num2), num3);
36   }
37 }
```

Note

Overloaded methods must have different parameter lists. You cannot overload methods based on different modifiers or return types

```
The maximum of 3 and 4 is 4
The maximum of 3.0 and 5.4 is 5.4
The maximum of 3.0, 5.4, and 10.14 is 10.14
```

Question and Answer

Can you invoke the **max** method with an **int** value and a **double** value, such as **max(2, 2.5)**? If so, which of the **max** methods is invoked?

The answer to the first question is yes. The answer to the second question is that the **max** method for finding the maximum of two **double** values is invoked. The argument value **2** is automatically converted into a **double** value and passed to this method.

The Java compiler finds the method that best matches a method invocation

✓ Check Point

What is wrong in the following program?

```
public class Test {
  public static void method(int x) {
  }

  public static int method(int y) {
    return y;
  }
}
```

Methods public static void method(int x) and public static int method(int y) have the same signature method(int).

✓ Check Point

Given two method definitions,

```
public static double m(double x, double y)
public static double m(int x, double y)
```

tell which of the two methods is invoked for:

```
a.      double z = m(4, 5);
b.      double z = m(4, 5.4);
c.      double z = m(4.5, 5.4);
```

(a) invokes the second method.
(b) invokes the second.
(c) invokes the first method.

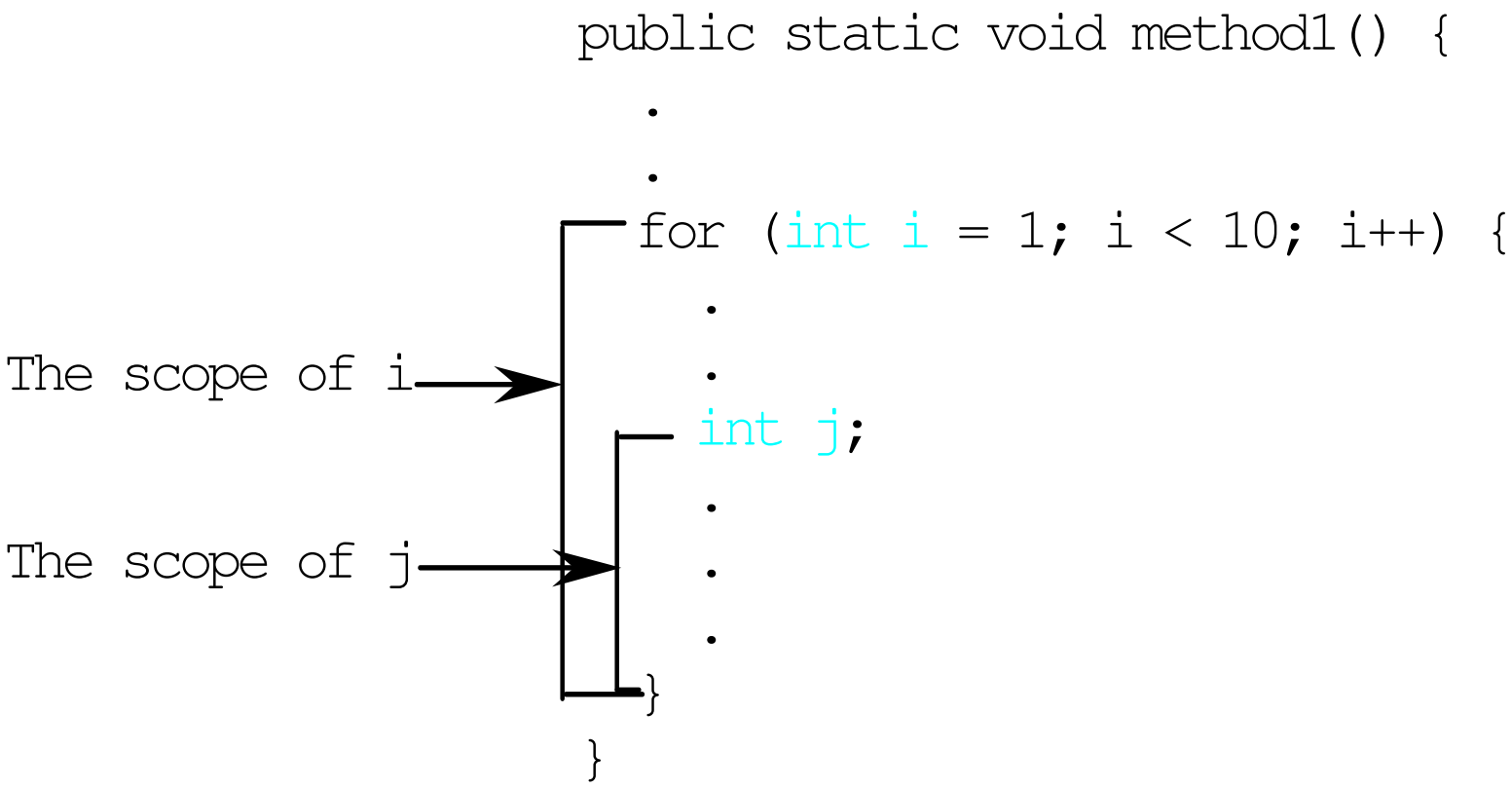
Scope of Local Variables

A local variable: a variable defined inside a method.

The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.

A local variable must be declared before it can be used.

Scope of Local Variables



Scope of Local Variables

You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.

Scope of Local Variables, cont.

It is fine to declare i in two non-nesting blocks

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

It is wrong to declare i in two nesting blocks

```
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum += i;
  }
}
```

Scope of Local Variables

A variable declared in the initial action part of a for loop header has its scope in the entire loop. But a variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

Benefits of Methods

- Write a method once and **reuse** it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

Mathematical Functions

Java provides many useful methods in the **Math** class for performing common mathematical functions.

The `Math` Class

- Class constants:
  - `PI` `Math.PI(3.141592653589793)`
  - `E` `Math.E (2.718281828459045)`
- Class methods:
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods

The `Math` Class

4.2.1 Trigonometric Methods

The `Math` class contains the following methods as shown in Table 4.1 for performing trigonometric functions:

TABLE 4.1 Trigonometric Methods in the Math Class

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

Trigonometric Methods

The parameter for `sin`, `cos`, and `tan` is an angle in radians. The return value for `asin`, `acos`, and `atan` is a degree in radians in the range between $-\pi/2$ and $\pi/2$. One degree is equal to $\pi/180$ in radians, 90 degrees is equal to $\pi/2$ in radians, and 30 degrees is equal to $\pi/6$ in radians.

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

Examples:

```
Math.sin(0) returns 0.0
Math.sin(Math.PI / 6) returns 0.5
Math.sin(Math.PI / 2) returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6) returns 0.866
Math.cos(Math.PI / 2) returns 0
Math.cos(Math.toRadians(30)) returns 0.86602540
```

The `Math` Class

TABLE 4.2 Exponent Methods in the Math Class

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

Exponent Methods

- **`exp(double a)`**

  Returns `e` raised to the power of `a`.

- **`log(double a)`**

  Returns the natural logarithm of `a`.

- **`log10(double a)`**

  Returns the 10-based logarithm of `a`.

- **`pow(double a, double b)`**

  Returns `a` raised to the power of `b`.

- **`sqrt(double a)`**

  Returns the square root of `a`.

Examples:

```
Math.exp(1) returns 2.71
Math.log(2.71) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(3.5, 2.5) returns
   22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 3.24
```

Math.pow

```
public class Expo {

public static void main(String[] args) {

   double x = 2.5;
   double y = 2;

        System.out.println(Math.pow(x, y));
}
}
```

Rounding Methods

- **`double ceil(double x)`**

  x rounded up to its nearest integer. This integer is returned as a double value.

- **`double floor(double x)`**

  x is rounded down to its nearest integer. This integer is returned as a double value.

- **`double rint(double x)`**

  x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double (i.e., 3.5).

- **`int round(float x)`**

  Return (int)Math.floor(x+0.5).

- **`long round(double x)`**

  Return (long)Math.floor(x+0.5).

Rounding Methods Examples

```
Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math.rint(2.1) returns 2.0
Math.rint(2.0) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3   // Returns int
Math.round(2.0) returns 2    // Returns long
Math.round(-2.0f) returns -2 // Returns int
Math.round(-2.6) returns -3 // Returns long
```

Rounding Methods Examples

```
Math.rint(2.0)     returns 2.0
Math.rint(2.1)     returns 2.0
Math.rint(2.2)     returns 2.0
Math.rint(2.3)     returns 2.0
Math.rint(2.4)     returns 2.0
Math.rint(2.5)     returns 2.0
Math.rint(2.6)     returns 3.0
Math.rint(-2.0)    returns -2.0
Math.rint(-2.5)    returns -2.0
Math.rint(-2.6)    returns -3.0
Math.rint(3.5)     returns 4.0
Math.rint(-3.5)    returns -4.0
Math.rint(2.501)   returns 3.0
Math.rint(-2.501) return -3.0
```

min, max, and abs

- `max(a, b)` and `min(a, b)`

  Returns the maximum or minimum of two parameters.

- `abs(a)`

  Returns the absolute value of the parameter.

- `random()`

  Returns a random `double` value in the range [0.0, 1.0).

Examples:

```
Math.max(2, 3) returns 3
Math.max(2.5, 3) returns 3.0
Math.min(2.5, 3.6) returns 2.5
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1
```

The random Method

Generates a random double value greater than or equal to 0.0 and less than 1.0 (0 <= Math.random() < 1.0).

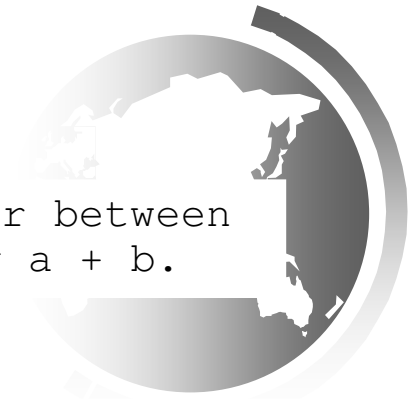
Examples:

```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

In general,

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.



Arithmetic Expressions

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

In Java, it will be translated to

```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

Operator Precedence

- Operators contained within pairs of parentheses are evaluated first. Parentheses can be nested, in which case the expression in the inner parentheses is evaluated first.

- When more than one operator is used in an expression, the following operator precedence rule is used to determine the order of evaluation.

  - Multiplication, division, and remainder operators are applied first. If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.

  - Addition and subtraction operators are applied last. If an expression contains several addition and subtraction operators, they are applied from left to right.

Operator Precedence