جامـــــعـة المـــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قـــســـــــــــم الانظمة الطبية الذكية

Lecture: (6)

Basic Computation Part III

Subject: Computer Programming (I)
Level: First
Lecturer: Dr. Maytham N. Meqdad

Display Variables

The `println()` method is often used to display variables.

To combine both text and a variable, use the `+` character:

Example

```
String name = "John";
System.out.println("Hello " + name);
```

You can also use the `+` character to add a variable to another variable:

Example

```
String firstName = "John ";
String lastName = "Doe";
String fullName = firstName + lastName;
System.out.println(fullName);
```

For numeric values, the `+` character works as a mathematical operator (notice that we use `int` (integer) variables here):

Example

```
int x = 5;
int y = 6;
System.out.println(x + y); // Print the value of x + y
```

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- Then we use the `println()` method to display the value of x + y, which is **11**

Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

Example

```
int x = 100 + 50;
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Example

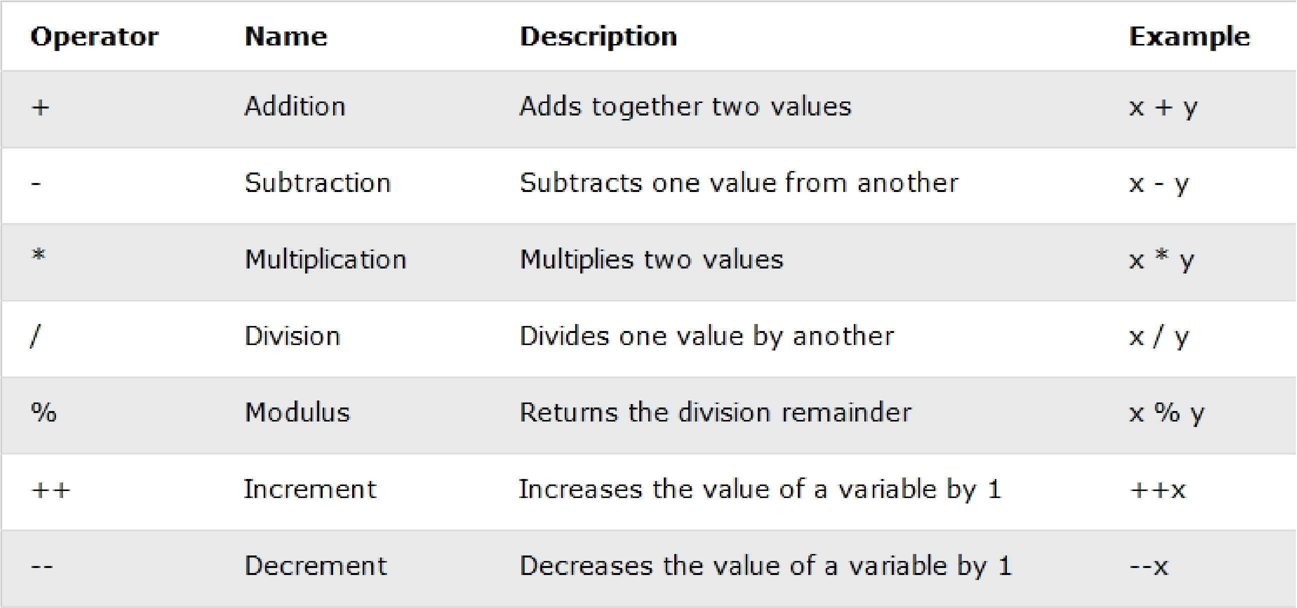
```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.



Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x - y);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x * y);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 12;
    int y = 3;
    System.out.println(x / y);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 2;
    System.out.println(x % y);
  }
}
```

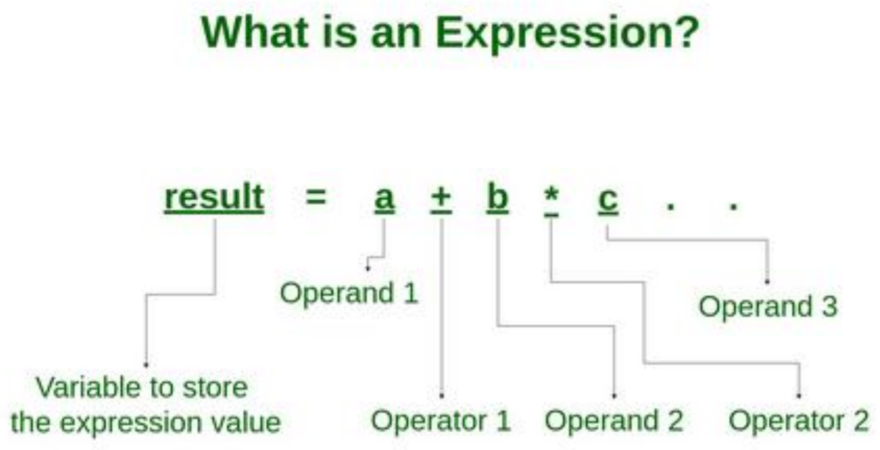
```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    ++x;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    --x;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x + y);
  }
}
```

Java Expressions

In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression. In the java programming language, an expression is defined as follows.



In the above definition, an **operator** is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.
Operands are the values on which the operators perform the task. Here operand can be a direct value or variable or address of memory location.

Expression Types

In the java programming language, expressions are divided into THREE types. They are as follows.

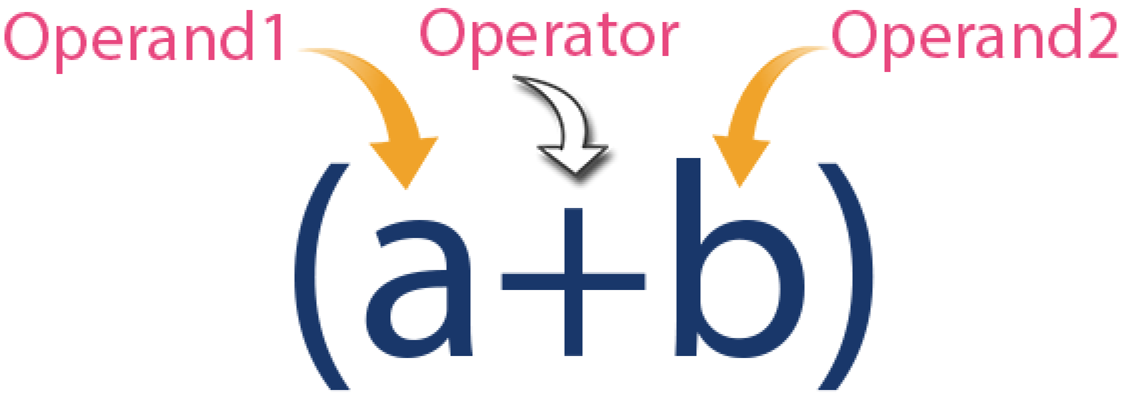
- **Infix Expression**
- **Postfix Expression**
- **Prefix Expression**

The above classification is based on the operator position in the expression.

Infix Expression

The expression in which the operator is used between operands is called infix expression.
The infix expression has the following general structure.

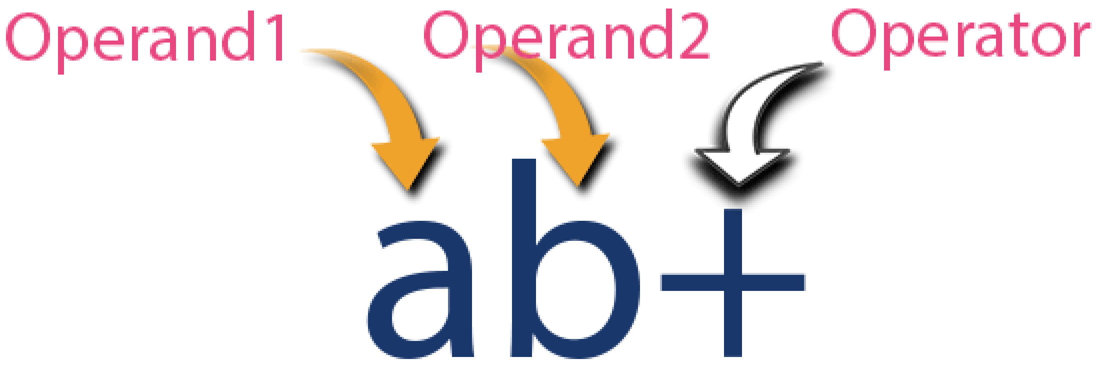
Example



Postfix Expression

The expression in which the operator is used after operands is called postfix expression.
The postfix expression has the following general structure.

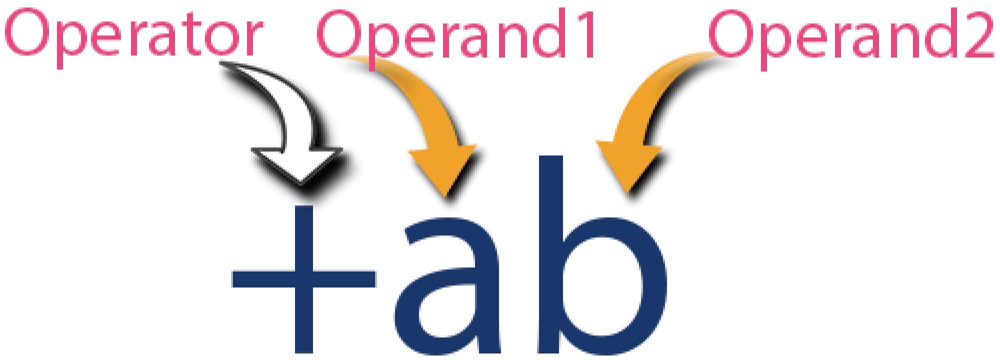
Example



Prefix Expression

The expression in which the operator is used before operands is called a prefix expression. The prefix expression has the following general structure.

Example



The Precedence of Arithmetic Operators

Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

Example

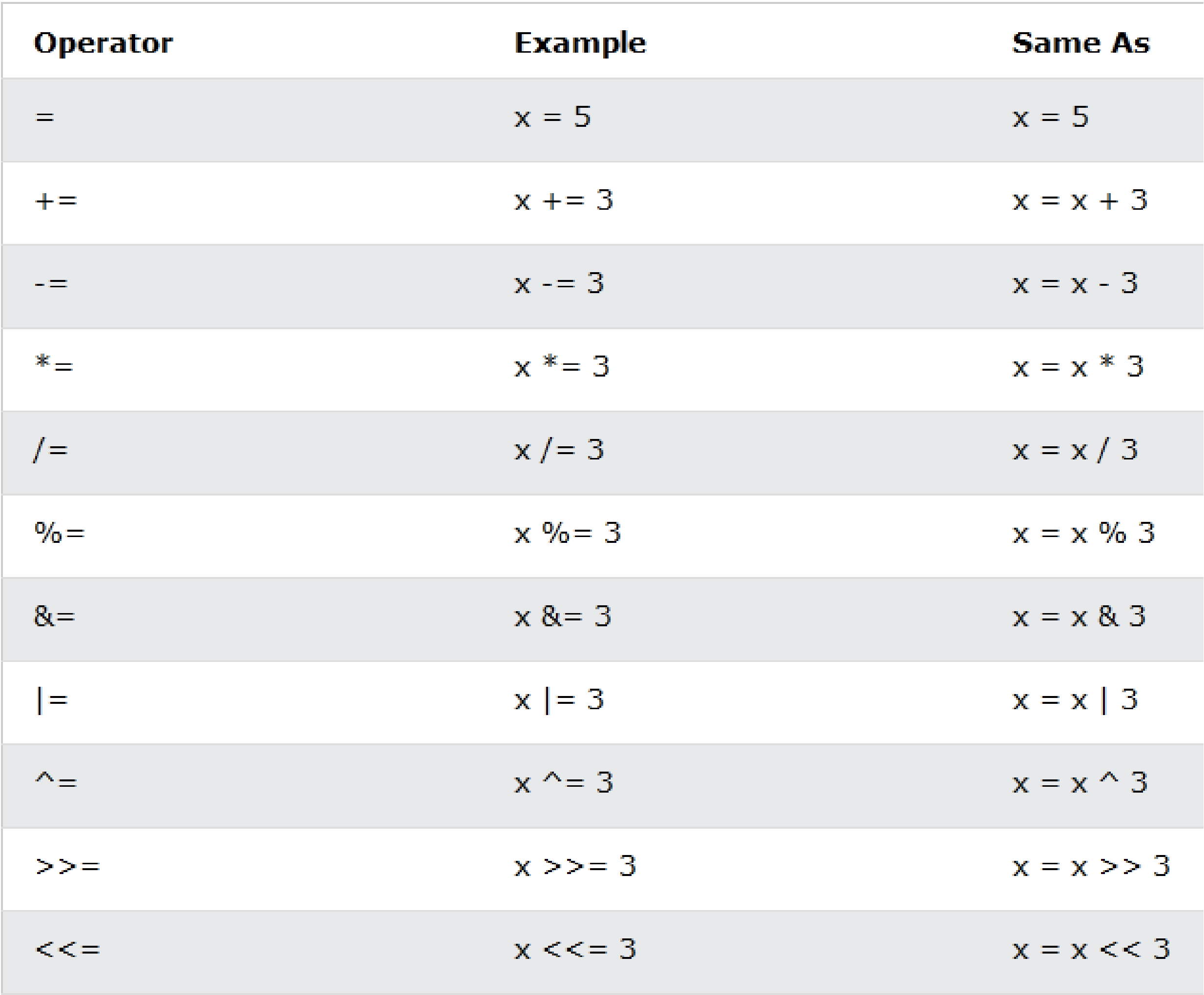
```
int x = 10;
```

The **addition assignment** operator (+=) adds a value to a variable:

Example

```
int x = 10;
x += 5;
```

A list of all assignment operators:



Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x += 3;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x -= 3;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x *= 3;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    double x = 5;
    x /= 3;
    System.out.println(x);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x %= 3;
    System.out.println(x);
  }
}
```

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  `byte -> short -> char -> int -> long -> float -> double`
- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  `double -> float -> long -> int -> char -> short -> byte`

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

Example

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double
    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

The Math Class – Library Functions

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

Math.max(*x*,*y*)

The `Math.max(x,y)` method can be used to find the highest value of *x* and *y*:

```
public class Main {

  public static void main(String[] args) {

    System.out.println(Math.max(5, 10));

  }

}
```

Math.min(*x*,*y*)

The `Math.min(x,y)` method can be used to find the lowest value of *x* and *y*:

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.min(5, 10));
  }
}
```

Math.sqrt(*x*)

The `Math.sqrt(x)` method returns the square root of *x*:

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.sqrt(64));
  }
}
```

Output: 8

Math.abs(*x*)

The `Math.abs(x)` method returns the absolute (positive) value of *x*:

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.abs(-4.7));
  }
}
```

Output: 4.7

Random Numbers

`Math.random()` returns a random number between 0.0 (inclusive), and 1.0 (exclusive):

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.random());
  }
}
```

To get more control over the random number, for example, if you only want a random number between 0 and 100, you can use the following formula:

```
public class Main {
  public static void main(String[] args) {
    int randomNum = (int)(Math.random() * 101);  // 0 to 100
    System.out.println(randomNum);
  }
}
```

All Math Methods

A list of all Math methods can be found in the table below:

Method	Description	Return Type
abs(x)	Returns the absolute value of x	double|float|int|long
acos(x)	Returns the arccosine of x, in radians	double
asin(x)	Returns the arcsine of x, in radians	double
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians	double
atan2(y,x)	Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta).	double
cbrt(x)	Returns the cube root of x	double
ceil(x)	Returns the value of x rounded up to its nearest integer	double
copySign(x, y)	Returns the first floating point x with the sign of the second floating point y	double
cos(x)	Returns the cosine of x (x is in radians)	double
cosh(x)	Returns the hyperbolic cosine of a double value	double
exp(x)	Returns the value of E^x	double
expm1(x)	Returns e^x -1	double
floor(x)	Returns the value of x rounded down to its nearest integer	double
getExponent(x)	Returns the unbiased exponent used in x	int
hypot(x, y)	Returns sqrt(x^2 +y^2) without intermediate overflow or underflow	double
IEEEremainder(x, y)	Computes the remainder operation on x and y as prescribed by the IEEE 754 standard	double
log(x)	Returns the natural logarithm (base E) of x	double
log10(x)	Returns the base 10 logarithm of x	double
log1p(x)	Returns the natural logarithm (base E) of the sum of x and 1	double
max(x, y)	Returns the number with the highest value	double|float|int|long
min(x, y)	Returns the number with the lowest value	double|float|int|long
nextAfter(x, y)	Returns the floating point number adjacent to x in the direction of y	double|float
nextUp(x)	Returns the floating point value adjacent to x in the direction of positive infinity	double|float

pow(x, y)	Returns the value of x to the power of y	double
random()	Returns a random number between 0 and 1	double
round(x)	Returns the value of x rounded to its nearest integer	int
rint(x)	Returns the double value that is closest to x and equal to a mathematical integer	double
signum(x)	Returns the sign of x	double
sin(x)	Returns the sine of x (x is in radians)	double
sinh(x)	Returns the hyperbolic sine of a double value	double
sqrt(x)	Returns the square root of x	double
tan(x)	Returns the tangent of an angle	double
tanh(x)	Returns the hyperbolic tangent of a double value	double
toDegrees(x)	Converts an angle measured in radians to an approx. equivalent angle measured in degrees	double
toRadians(x)	Converts an angle measured in degrees to an approx. angle measured in radians	double
ulp(x)	Returns the size of the unit of least precision (ulp) of x	double|float

Note: All Math methods are `static`.