

4回目 演算子

■ 今日の講義で学ぶ内容 ■

- 演算子とオペランド、式
- 様々な演算子
- 代表的な演算子の使用例

演算子とオペランド

演算子

演算の種類です

例えば、+、-、*、/



掛け算の記号は×ではなく、*（アスタリスク）を使います
割り算の記号は÷ではなく、/（スラッシュ）を使います

オペランド

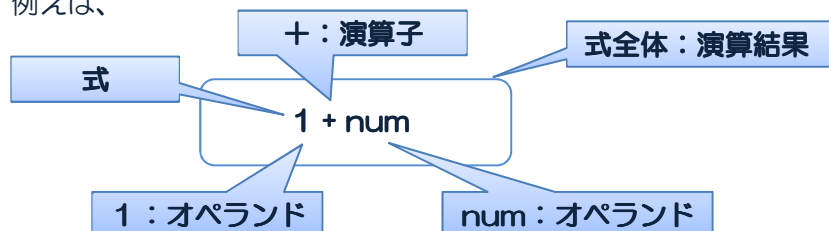
演算の対象です

例えば、5（値）、num（変数）

式

演算子とオペランドの組み合わせにより構成される数式です
式は演算結果をもちます

例えば、



単一の変数も式と呼びます

例えば、



一般に式は演算結果を持ちますが、演算結果をもたない特殊な式もあります。Java プログラミング II で解説します

ソースコード例

ソースファイル名 : Sample4_1.java

```
// 変数を用いた式
class Sample4_1
{
    public static void main(String[] args)
    {
        // 変数宣言と初期化
        int num1 = 15;
        int num2 = 3;

        // num1 と num2 の四則計算
        int add;
        add = num1 + num2;
        int sub;
        sub = num1 - num2;
        int mul;
        mul = num1 * num2;
        int div;
        div = num1 / num2;

        // 計算結果の出力
        System.out.println(num1 + "と" + num2 + "の四則計算：");
        System.out.println("和 = " + add);
        System.out.println("差 = " + sub);
        System.out.println("積 = " + mul);
        System.out.println("商 = " + div);
    }
}
```

掛け算は、×ではなく
*(アスタリスク)を使います

割り算は、÷ではなく
/(スラッシュ)を使います

実行画面

```
>java Sample4_1
15 と 3 の四則計算：
和 = 18
差 = 12
積 = 45
商 = 5
```

演算子の種類

演算子は、算術演算子、ビット論理演算子、シフト演算子、インクリメント・デクリメント演算子、関係演算子、論理演算子、条件演算子、代入演算子など多彩な分類をもちます

(算術演算子)

+	加算 (文字列連結)
-	減算
*	乗算
/	除算
%	剰余

(ビット論理演算子)

&	ビット論理積
	ビット論理和
^	ビット排他的論理和

(シフト演算子)

<<	左シフト
>>	右シフト
>>>	符号なし右シフト

(その他)

+	プラス (単項)
-	マイナス (単項)
~	1 の補数「反転」 (単項)

(インクリメント・デクリメント演算子)

++	インクリメント (単項)
--	デクリメント (単項)

(関係演算子)

>	より大きい
>=	以上
<	未満
<=	以下
==	等しい
!=	等しくない

(論理演算子)

!	論理否定 (単項)
&&	論理積
	論理和

(条件演算子)

?:	条件 (三項演算子)
----	------------

(代入演算子)

=	
---	--

n項演算子

演算子はオペランドの数により単項、二項、三項演算子と呼ばれます

・単項演算子

オペランドが1つ
例えば、++ (インクリメント)

・二項演算子

オペランドが2つ
例えば、+、-、*、/ (四則計算)

・三項演算子

オペランドが3つ
Java では唯一であり、?: (条件演算子)

加算演算子+

加算と文字列連結の2つの役割をもちます

2つの役割はオペランドの種類によりどちらかに決まります

・文字列連結

いずれかまたは両方のオペランドが String 型のとき

・加算

それ以外



“Hello”などの文字列リテラルは String 型とみなされます



次のコードを実行すると、右のように出力されます

System.out.println("ABC"+"DEF");	→	ABCDEF
System.out.println("ABC"+50);	→	ABC50
System.out.println(20+50);	→	70

ソースコード例

ソースファイル名：Sample4_2.java

```
// 剰余付き割り算プログラム
import java.io.*;

class Sample4_2
{
    public static void main(String[] args) throws IOException
    {
        // キーボード準備
        BufferedReader br;
        br = new BufferedReader(new InputStreamReader(System.in));

        // 整数用の変数
        int num1, num2, tmp;
        int div;    // 商
        int mod;    // 余り

        // 整数の入力
        System.out.println("#剰余付き割り算#¥n2つの整数を入力してください。");
        System.out.println("1つ目の整数を入力してください。");
        num1 = Integer.parseInt(br.readLine());
        System.out.println("2つ目の整数を入力してください。");
        num2 = Integer.parseInt(br.readLine());

        // 剰余の計算
        mod = num1 % num2; // %は余りを計算する演算子

        // 商の計算
        tmp = num1 - mod;
        div = tmp / num2;

        // 結果の出力
        System.out.println(num1 + " ÷ " + num2 + " = " + div + " あまり " + mod);
    }
}
```

実行画面

```
>java Sample4_2
#剰余付き割り算#
2つの整数を入力してください。
1つ目の整数を入力してください。
11
2つ目の整数を入力してください。
4
11 ÷ 4 = 2 あまり 3
```

インクリメント・デクリメント演算子

インクリメント演算子

++

オペランドの変数の値を1増やします
演算結果は次のようになります


a++ 後置インクリメント 変数 a を演算結果とした後、a の値を 1 増やします
++a 前置インクリメント 変数 a の値を 1 増やした後、a を演算結果とします

デクリメント演算子

--

オペランドの変数の値を1減らします
演算結果は次のようになります

a-- 後置デクリメント 変数 a を演算結果とした後、a の値を 1 減らします
--a 前置デクリメント 変数 a の値を 1 減らした後、a を演算結果とします

 演算結果が出されるタイミングに注意しましょう

ソースコード例

ソースファイル名：Sample4_3.java

```
// 前置・後置インクリメント
class Sample4_3
{
    public static void main(String[] args)
    {
        // 変数の宣言と初期化
        int a1 = 0, a2 = 0;
        int b = 0, c = 0;

        System.out.println("a1=" + a1);
        System.out.println("a2=" + a2);

        // 前置・後置インクリメントの違い
        b = a1++;
        c = ++a2;
        System.out.println("後置・前置インクリメントをします");
        System.out.println("後置インクリメント (b=a1++) b=" + b);
        System.out.println("前置インクリメント (c=++a2) c=" + c);
        System.out.println("a1=" + a1);
        System.out.println("a2=" + a2);

        // 単独で用いると1増えるだけ
        a1++;
        ++a2;
        System.out.println("単独で後置・前置インクリメントもできます");
        System.out.println("後置インクリメント a1++;");
        System.out.println("前置インクリメント ++a2;");
        System.out.println("a1=" + a1);
        System.out.println("a2=" + a2);
    }
}
```

実行画面

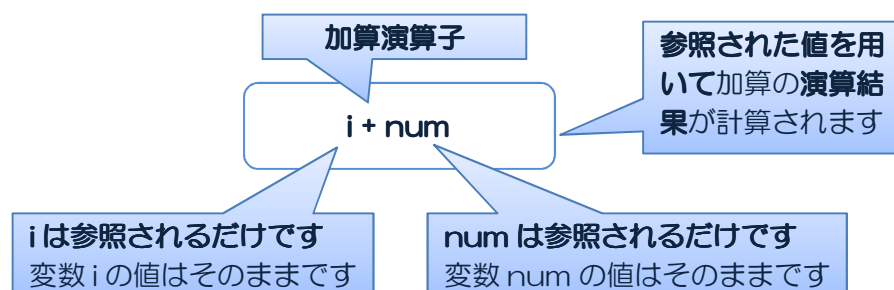
```
>java Sample4_3
a1=0
a2=0
後置・前置インクリメントをします
後置インクリメント(b=a1++;) b=0
前置インクリメント(c=++a2;) c=1
a1=1
a2=1
単独で後置・前置インクリメントもできます
後置インクリメント a1++;
前置インクリメント ++a2;
a1=2
a2=2
```



インクリメント・デクリメント演算子では、
その**オペランドの変数の値そのものが演算前と演算後で変化**します

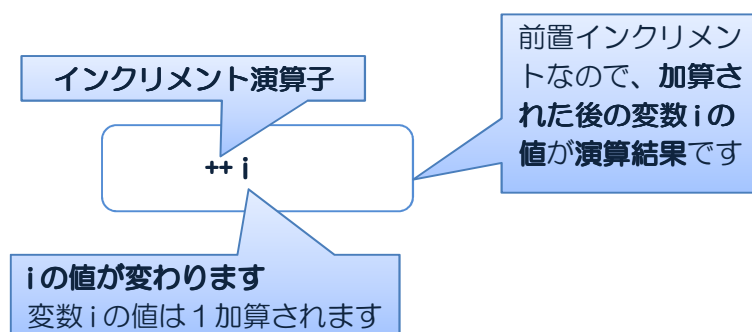
一般に、式を計算するとき、オペランドの値を用いて演算結果を計算します
オペランドの変数の値は参照されるだけで、新しい値が代入されることはありません

例えば、



例外として、**インクリメント・デクリメント演算子**と**代入演算子**（次に説明）では
オペランドの変数に新しい値が代入されます

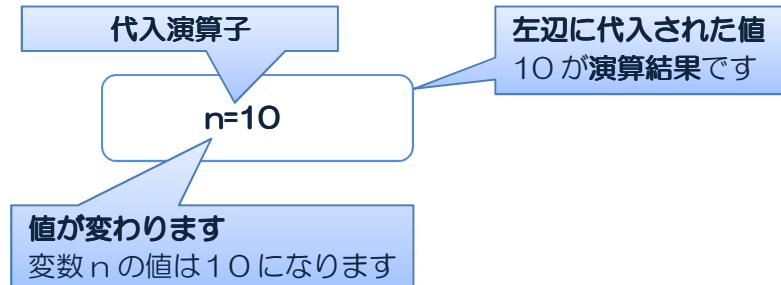
例えば、



代入演算子

代入演算子

= 右辺の値を左辺の変数に代入します
演算結果は左辺に代入された値です



たとえば、

```
int n;
System.out.println(n=10);
```

 とすると、10 が画面に出力されます

代入演算子の目的は**代入**です
 代入演算子自身の演算結果を利用することは少ないです

複合的な 代入演算子

■= 左辺と右辺を用いて**演算■**をした後、**結果を左辺に代入**します
 演算結果は左辺に代入された値です

■には次のような演算子を入れて利用できます

機能的には
a ■= b
 と
a = a ■ b
 は同じです

a += b	加算+代入	a = a + b
a -= b	減算+代入	a = a - b
a *= b	乗算+代入	a = a * b
a /= b	除算+代入	a = a / b
a %= b	剰余+代入	a = a % b
a &= b	論理積+代入	a = a & b
a = b	論理和+代入	a = a b
a ^= b	排他的論理和+代入	a = a ^ b
a <<= b	左シフト+代入	a = a << b
a >>= b	右シフト+代入	a = a >> b
a >>>= b	符号なし右シフト+代入	a = a >>> b

たとえば、

```
int n=5;
System.out.println(n+=10);
```

 とすると、15 が画面に出力されます

複合的な代入演算子の目的は**演算を伴った代入**です
 複合的な代入演算子自身の演算結果を利用することは少ないです

ソースコード例

ソースファイル名：Sample4_4.java

```
// 複合的な代入演算子を用いた総計処理
class Sample4_4
{
    public static void main(String[] args)
    {
        int sum=0; // 総計用の変数、ゼロで初期化

        // 処理内容のメッセージ
        System.out.println("3つの整数の総計を求めます。");
        System.out.println("1つ目の整数は3");
        sum += 3;
        System.out.println("2つ目の整数は5");
        sum += 5;
        System.out.println("3つ目の整数は2");
        sum += 2;

        // 総計の表示
        System.out.println("総計は" + sum + "です。");
    }
}
```

実行画面

```
>java Sample4_4
3つの整数の総計を求めます。
1つ目の整数は3
2つ目の整数は5
3つ目の整数は2
総計は 10 です。
```


シフト演算子

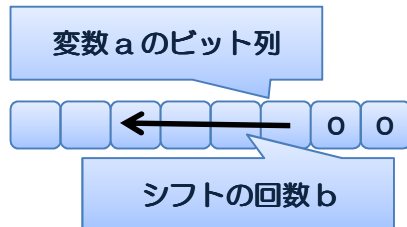
シフト演算子

《、》、《》、《》


指定ビットシフトします

演算結果は指定ビットシフトした値です

$a \ll b$ 左シフト演算子



b ビット分 a のビット列を左へシフトし、
右を 0 で埋めます

 シフト演算子のオペランドの変数の値は
加算や減算などの算術演算子と同様に
演算前と演算後で変化しません

$a \gg b$ 右シフト演算子



b ビット分 a のビット列を右へシフトし、
a が正の場合は 0 で、負の場合は 1 で左を埋めます

 右シフト演算子は左側を 0 又は 1 で埋めることに
よりシフトする値の正負を保持します

$a \ggg b$ 符号なし右シフト演算子



b ビット分 a のビット列を右へシフトし、
0 で左を埋めます

ソースコード例

ソースファイル名: Sample4_5.java

```
// シフト演算
class Sample4_5
{
    public static void main(String[] args)
    {
        int i=2, ans;

        // シフト演算例
        ans = i << 1; // 1 ビット左へ (2倍)
        System.out.println(ans);
        ans = i >> 1; // 1 ビット右へ (0.5倍)
        System.out.println(ans);
    }
}
```

実行画面

```
>java Sample4_5
4
1
```

ソースコード例

ソースファイル名 : Sample4_6.java

```
// シフト演算の働き
class Sample4_6
{
    public static void main(String[] args)
    {
        int i;
        short num=5;

        // ビット列出力
        System.out.print("シフト前 ");
        for(i=0;i<16;i++)
            System.out.print(0x0001&(num>>(15-i)));
        System.out.println("(b)");

        // シフト演算
        num <<= 2;

        // ビット列出力
        System.out.print("シフト後 ");
        for(i=0;i<16;i++)
            System.out.print(0x0001&(num>>(15-i)));
        System.out.println("(b)");
    }
}
```

実行画面

```
>java Sample4_6
シフト前  000000000000000101(b)
シフト後  00000000000010100(b)
```