

4回目 演算子

■ 今日の講義で学ぶ内容 ■

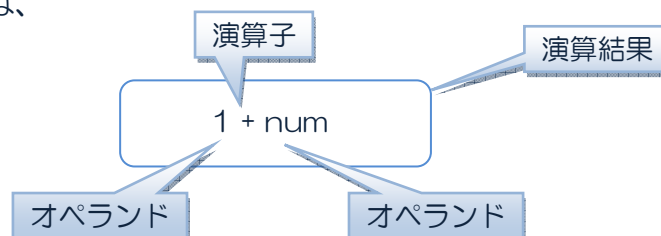
- ・ 演算と式
- ・ 様々な演算子
- ・ 代表的な演算子の使用例

演算子とオペランド

演算子 演算の種類
例えば、+、-、*、/、=

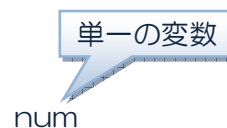
オペランド 演算の対象
例えば、5（値）、num（変数）

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
式は演算結果をもちます
例えば、



⚠ 関数呼び出しもオペランドにおくことができます
※関数については後の講義で説明します

単一の変数も式と呼びます
例えば、



⚠ 単一の関数も式と呼びます
戻り値をもたない関数呼び出しなど
式が演算結果をもたない場合もあります
※関数については後の講義で説明します

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);
    }
}
```

実行画面

```
>java Sample4_1
15 と 3 の四則計算：
和 = 18
差 = 12
積 = 45
商 = 5
-- Press any key to exit (Input "c" to continue) --
```

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)		(インクリメント・デクリメント演算子)	
+	加算 (文字列連結) ※ 1	++	インクリメント (単項)
-	減算	--	デクリメント (単項)
*	乗算	(関係演算子)	
/	除算	>	より大きい
%	剰余	>=	以上
(ビット論理演算子)		<	未満
&	ビット論理積	<=	以下
	ビット論理和	==	等しい
^	ビット排他的論理和	!=	等しくない
(シフト演算子)		(論理演算子)	
<<	左シフト	!	論理否定 (単項)
>>	右シフト	&&	論理積
>>>	符号なし右シフト		論理和
(その他)		(条件演算子)	
+	プラス (単項)	?:	条件 (三項演算子)
-	マイナス (単項)	(代入演算子)	
~	1 の補数 (単項)	=	

演算子は、オペランドの数により単項演算子または二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)



演算子+は、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、演算子+のオペランドの種類によりいずれかに決まります

- ・ 文字列連結 いずれかまたは両方のオペランドが文字列のとき
- ・ 加算 それ以外

例えば、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
        String str1, str2;
        int num1, num2;

        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。");
        str1 = br.readLine();
        num1 = Integer.parseInt(str1);
        System.out.println("2つ目の整数を入力してください。");
        str2 = br.readLine();
        num2 = Integer.parseInt(str2);

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

        // 商の計算
        div = (num1 - mod) / num2;
        // ※ここで括弧 ( ) は演算の優先度を表す
        // 演算の優先度の詳細は次回の講義で説明する

        // 結果の出力
        System.out.println(num1 + "/" + num2 + "=" + div + " 余り" + mod);
        // ※複数の加算を含む式の場合、演算を左から右へ向って順に実行される
        // オペランドによる加算の機能の選択もこの順に従い行われる
        // 演算の優先度の詳細は次回の講義で説明する
    }
}
```

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 余り 3
-- Press any key to exit (Input "c" to continue) --
```

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	++	値を1増やします
a++; 後置インクリメント	変数 a を演算結果として処理した後、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 を演算結果として処理します	
デクリメント演算子	--	値を1減らす
a--; 後置デクリメント	変数 a を演算結果として処理した後、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 を演算結果として処理します	

ソースコード例

ソースファイル名：Sample4_3_1.java

```
// 前置・後置インクリメント
class Sample4_3_1
{
    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("他の演算子を伴い後置・前置インクリメントすると、");
        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);

        // 単独で用いる場合は同じ意味
        a1++;
        ++a2;
        System.out.println("単独で後置・前置インクリメントすると、");
        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_1
a1=0
a2=0
他の演算子を伴い後置・前置インクリメントすると、
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すると、
後置インクリメント a1++;
前置インクリメント ++a2;
a1=2
a2=2
-- Press any key to exit (Input "c" to continue) --
```



次の場合、前置・後置の違いがどのように演算結果に表れるでしょうか？

ソースファイル名：Sample4_3_2.java

```
// 前置・後置インクリメント (2)
class Sample4_3_2
{
    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;
        int num = 5;

        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 = num + a1++;
        c = num + ++a2;

        System.out.println("後置インクリメント(b = num + a1++;) b=" + b);
        System.out.println("前置インクリメント(c = num + ++a2;) c=" + c);
        System.out.println("a1=" + a1);
        System.out.println("a2=" + a2);
    }
}
```



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

複合的な代入演算子 ■ = 演算■と代入を同時に実行します

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;

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。
-- Press any key to exit (Input "c" to continue) --
```

シフト演算子

シフト演算子 <<, >>, >>> 指定ビットずらし

a << b; 左シフト演算子 bビット分 a のビット列を左へずらし、右を 0 で埋める

a >> b; 右シフト演算子 bビット分 a のビット列を右へずらし、
a が正の場合は 0 で、負の場合は 1 で左を埋める

 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 >>> b; 符号なし右シフト演算子 bビット分 a のビット列を右へずらし 0 で左を埋める

 符号なし右シフト演算子は、変数内の 2 進データをそのまま取り出したいときによく利用されます

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
-- Press any key to exit (Input "c" to continue) --
```

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  00000000000000101(b)
シフト後  00000000000010100(b)
-- Press any key to exit (Input "c" to continue) --
```