

## 5回目 演算子の優先順位と変数の型変換

## ■ 今日の講義で学ぶ内容 ■

- 演算子の優先順位
- 優先順位の変更の方法
- キャスト演算子と型変換

## 演算子の優先順位

## 演算子の優先順位

式を計算するときの**演算の順序**です

例えば、 $a = b * c + d;$  では乗算を先に計算するというルールです

(主な演算子の優先順位)

演算子	名前	結合規則
++	後置インクリメント	左
--	後置デクリメント	左
!	論理否定	右
~	1の補数(反転)	右
+	プラス	右
-	マイナス	右
++	前置インクリメント	右
--	前置デクリメント	右
()	キャスト	右
*	乗算	左
/	除算	左
%	剰余	左
+	加算(文字列連結)	左
-	減算	左
<<	左シフト	左
>>	右シフト	左
>>>	符号なし右シフト	左
>	より大きい	左
>=	以上	左
<	未満	左
<=	以下	左
==	等価	左
!=	非等価	左
&	ビット論理積	左
^	ビット排他的論理和	左
	ビット論理和	左
&&	論理積	左
	論理和	左
?:	条件	右
=	代入	右
+=, -= など	複合代入演算	右

優先順位が高い

優先順位は同じ

私たちが普段行うように  
加算や減算よりも  
乗算や除算の方が  
優先順位が高いです

代入演算子は、  
優先順位が最も低く  
最後に処理されます

優先順位が低い

## 左 結 合

優先順位が等しい場合、**左側から順に**演算をする規則  
たとえば、  
 **$a * b \% c / d$** ; の場合  **$a *^{\textcircled{1}} b \%^{\textcircled{2}} c /^{\textcircled{3}} d$** ; となる

## 右 結 合

優先順位が等しい場合、**右側から順に**演算をする規則  
たとえば、  
 **$\sim ++a$** ; の場合  **$\sim^{\textcircled{2}} ++^{\textcircled{1}} a$** ; となる

## ソースコード例

ソースファイル名 : Sample5\_1.java

```
// 演算子の優先順位
class Sample5_1
{
    public static void main(String[] args)
    {
        int a=6, b=2, c=5;
        int d1, d2, d3;
        String str1;

        // すべて等しい優先順位かつ右結合の演算子なので右側から順に演算
        d1=d2=d3=0;

        // 加算より乗算の優先順位が高い
        d1=a+b*c;

        // 加算より剰余の優先順位が高い
        d2=c%b+a;

        // 乗算と除算の優先順位は同じかつ左結合の演算子なので左側から順に演算
        d3=a/b*c;

        // 加算より除算の優先順位が高いかつ加算は左結合の演算子なので左側から
        // 順に演算
        // (重要)加算ではオペランドに文字列がある場合は、文字列連結となる
        str1=a/b+ "文字列" +b+c;

        System.out.println("a=" + a + ", b=" + b + ", c=" + c);
        System.out.println("a+b*c = " + d1);
        System.out.println("c%b+a = " + d2);
        System.out.println("a/b*c = " + d3);
        System.out.println("a/b+¥"文字列¥"+b+c = " + str1);
    }
}
```

代入演算子"="は演算結果を持ちます

- 右辺の値を左辺に代入します
  - 式の演算結果は代入された値です
- 例えば、

```
int d;
System.out.println(d=6);
```

とすると、画面には  
**6**  
と出力されて **d** には **6** が代入されます

$d1 =^{\textcircled{3}} d2 =^{\textcircled{2}} d3 =^{\textcircled{1}} 0$ ;

$d1 =^{\textcircled{3}} a +^{\textcircled{2}} b *^{\textcircled{1}} c$ ;

$d2 =^{\textcircled{3}} c \%^{\textcircled{1}} b +^{\textcircled{2}} a$ ;

$d3 =^{\textcircled{3}} a /^{\textcircled{1}} b *^{\textcircled{2}} c$ ;

$str1 =^{\textcircled{5}} a /^{\textcircled{1}} b +^{\textcircled{2}} \text{"文字列"} +^{\textcircled{3}} b +^{\textcircled{4}} c$ ;

加算演算子+

文字列連結として機能  
した場合の演算結果は、  
連結された文字列です。

## 実行画面

```
>java Sample5_1
a=6, b=2, c=5
a+b*c = 16
c%b+a = 7
a/b*c = 15
a/b+"文字列"+b+c = 3 文字列 25
```

## “( )”による演算子の優先順位の変更

### 式のグループ化

式を“( )”で囲みグループにすることにより、その部分を他より先に行わせることができます

たとえば、

$(a + b) * c$ ; の場合  $(a +^{\textcircled{1}} b) *^{\textcircled{2}} c$ ; となる



括弧は、入れ子（括弧の中にさらに括弧）にもできます  
この場合、内側の括弧から演算が行われます

### ソースコード例

ソースファイル名：Sample5\_2.java

```
// 演算子の優先順位の変更
class Sample5_2
{
    public static void main(String[] args)
    {
        // 括弧内の演算が先にされ、その後は優先順位に従い除算が行われる
        int a=10/(5-3);
        System.out.println("10/(5-3)=" + a);

        // 括弧内の演算が先にされ、その後は優先順位に従い剰余、減算と進む
        int b=(4+17)%2-1;
        System.out.println("(4+17)%2-1=" + b);

        // すべて等しい優先順位かつ左結合の演算子なので左側から順に演算
        System.out.println("1+2=" + 1+2 + "です。"); // 期待通りの結果が得られない
        System.out.println("1+2=" + (1+2) + "です。"); // ( )により優先順位を変更

        // 加算より乗算の優先順位が高い
        System.out.println("3*4=" + 3*4 + "です。");
    }
}
```

## 実行画面

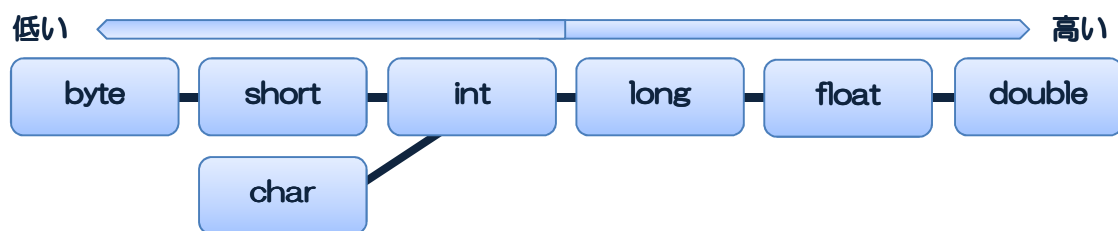
```
>java Sample5_2  
10/(5-3)=5  
(4+17)%2-1=0  
1+2=12 です。  
1+2=3 です。  
3*4=12 です。
```

 演算子の優先順位をすべて覚えるのは大変です  
意図的に“( )”を用いて優先順位を指定するとよいでしょう

## 値の型変換

### 型のランク

型は**値の表現範囲**より次のように**ランク付け**されています  
これを**型のランク**といいます




### 値の型変換

値の**型を変換**することです  
たとえば、double 型から int 型、byte 型から int 型の変換です

型変換は**型のランク**にもとづき区別され処理されます

### 高いランクへの型変換 (拡大変換といいます)

値は**拡張**されます  
たとえば、int 型 2 → double 型 2.0

 低いほうの型で表現されない情報は0で補完されます

### 低いランクへの型変換 (縮小変換といいます)

値は**切り捨て**られます  
たとえば、double 型 2.5 → int 型 2

 低いほうの型で表現される情報だけ残ります

## 型変換はいつ行われる？ (1) 変数に値を代入するとき (2) 演算を行うとき

### (1) 変数に値を代入するとき

#### 高いランクの型の変数へ代入

自動的に型の**拡大変換**が行われます  
たとえば、  
`int i = 5;`  
`double d = i;`



#### 低いランクの型の変数へ代入

自動的な**縮小変換**は行われません！！

例えば、  
`double d = 5.5;`  
`short s = d;`



縮小変換では、情報が失われます。

プログラマがこれを承知して行っているかどうか Java が判断できないため、エラーを出します。

このようなコードをコンパイルしたら「**精度が落ちている可能性**」という**コンパイルエラー**がでます

キャスト演算子により**明示的に型変換**を行います

#### キャスト演算子 “( )”

式の値を ( ) 内で指定した型に**一時的に型変換**します  
たとえば、  
`(double) 10`  
`(int) a`

( 型 ) 式

先ほどの例では、

`double d = 5.5;`  
`short s = (short) d;`  
とすれば、**明示的に**型の**縮小変換**が行われ、  
変数 `s` に `5` が代入されます



キャスト演算は、**一時的な型変換**ですので  
このとき、変数 `d` の値は `5.5` のままです

一般に、**キャスト演算子**は型の**拡大変換**にも使えます  
たとえば、  
`int` 型の値を `double` 型に**キャスト**する場合は、  
`int i = 5;`  
`double d = (double) i;`  
とすれば、**明示的に**型の**拡大変換**が行われます



キャスト演算子と演算子の優先順位変更の記号は、  
どちらも ( ) です

きちんと区別して間違えないようにしましょう！！

★キャスト演算子の場合は括弧の中には**型**を書きます

★優先順位変更の場合は括弧の中には**式**を書きます

## ソースコード例

ソースファイル名：Sample5\_3.java

```
// 代入時の型変換
class Sample5_3
{
    public static void main(String[] args)
    {
        byte a;
        int b;
        double c;

        // int 型から double 型への型変換
        b = 2;
        c = b; // 高いランクの型への変換
        System.out.println("int 型" + b + " -> double 型" + c);

        // double 型から int 型への型変換
        c = 2.5;
        b = (int)c; // 低いランクの型への変換 キャスト演算子必要
        // キャスト演算は一時的な型変換なので変数 c そのものの値は変化しない
        System.out.println("double 型" + c + " -> int 型" + b);

        // int 型から byte 型への型変換
        b = 256;
        a = (byte)b; // 低いランクの型への変換 キャスト演算子必要
        System.out.println("int 型" + b + " -> byte 型" + a);
    }
}
```



$256_{(10)} = 1,0000,0000_{(2)}$

この値を byte 型（8ビット）で切り取ると、  
 $0000,0000_{(2)}$  となります

## 実行画面

```
>java Sample5_3
int 型 2 -> double 型 2.0
double 型 2.5 -> int 型 2
int 型 256 -> byte 型 0
```

## (2) 演算を行うとき

**演算時の型変換** 異なるランクの型が同じ式に混在する場合、  
演算前に一方のオペランドが**ランクの高い型に一時的に型変換**されます

演算後の式の**演算結果**はオペランドの型の中で**ランクの高い型**になります

例えば、  
 $2 * 2.5 \rightarrow \underline{2.0} * 2.5 \rightarrow \underline{5.0}$   
 $5 / 2.0 \rightarrow \underline{5.0} / 2.0 \rightarrow \underline{2.5}$

 **byte 型**と **short 型**、**char 型**のオペランドは、  
**演算前**に一時的にランクの高い **int 型**へ**拡大変換**されます

### ソースコード例

ソースファイル名：Sample5\_4.java

```
// 演算時の型変換 1
class Sample5_4
{
    public static void main(String[] args)
    {
        int diameter=2;
        double pi=3.14;

        // 円周の計算
        System.out.println("直径が" + diameter + "cm の円の");
        System.out.println("円周は" + (diameter*pi) + "cm です。");

        // int 型*double 型であるため、
        // int 型変数は double 型に変換されて演算される
        // 式の値は double 型になる
    }
}
```

### 実行画面

```
>java Sample5_4
直径が 2cm の円の
円周は 6.28cm です。
```

## ソースコード例

ソースファイル名：Sample5\_5.java

```
// 演算時の型変換 2
class Sample5_5
{
    public static void main(String[] args)
    {
        int num1=5;
        int num2=4;
        double div;

        div = num1/num2;
        System.out.println("5 / 4 は" + div + "です。");

        // int 型/int 型であるため型変換はされず、式の値は int 型になる
        // このため期待通りの答えが得られない
        // 一方の変数を double 型にキャスト演算子を用いて拡大変換することで
        // 演算は double 型で行われる
        div = ((double)num1)/num2;
        System.out.println("5 / 4 は" + div + "です。");
    }
}
```

### 整数同士の除算に要注意！！

整数値や整数型の変数のみからなる足し算や引き算の式では答えは整数ですね

しかし、  
除算を含む場合は小数が出る場合がありますので、演算結果の型の決まり方に注意！！

## 実行画面

```
>java Sample5_5
5 / 4 は 1.0 です。
5 / 4 は 1.25 です。
```