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

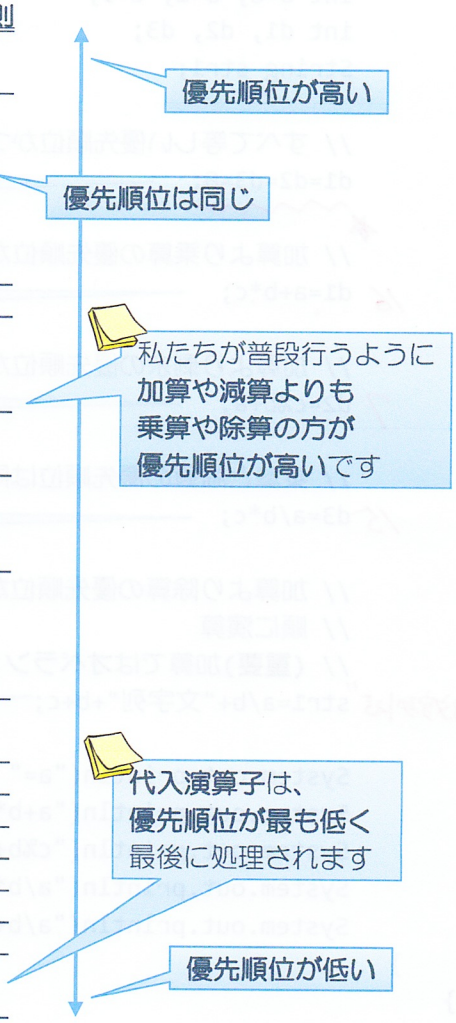
- 演算子の優先順位
- 優先順位の変更の方法
- キャスト演算子と型変換

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



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 *① b %② c /③ d;`　となります

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、
`~++a;`　の場合　`~② ++① a;`　となります

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

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

```
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、
その部分の演算を**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 +① b) *② c;** となります

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、**最も内側の括弧**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

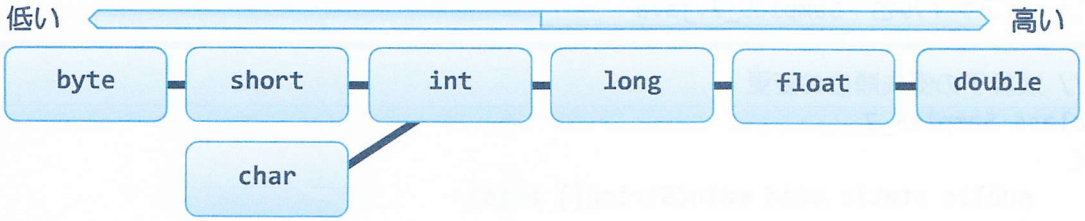
```
10/(5-3)=5
(4+17)%2-1=0
1+2=12 です。
1+2=3 です。
3*4=12 です。
```

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の拡大変換と縮小変換

高いランクの変数へ代入

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;
```

OK

低いランクの変数へ代入

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;
```

エラー

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
演算結果は一時的に型変換された値です

(型) 式

コード例　|　`(int)a`
コード例　|　`(double)10`

先ほどの例では、

```
double d = 5.5;
short s = (short)d;
```

OK

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 型に拡大変換を行う場合は、

```
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
たとえば、

```
int a;
a = (int)(3.2 + 1.9);
```

とすれば、( ) はキャスト演算子、( ) は優先順位変更で、変数 a の値は 5 です

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

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

②演算を行うときの拡大変換と縮小変換

異なるランクの変数の混在

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**拡大変換**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` → **`2.0`** `* 2.5` → **`5.0`**
`5 / 2.0` → **`5.0`** `/ 2.0` → **`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ます
        // 式の値は double 型になります
    }
}
```

int型 double型 2×3.14
(拡大) double * double 2.0×3.14
double 6.28

実行画面

```
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

```
5 / 4 は 1.0 です。
5 / 4 は 1.25 です。
```

今日の講義のまとめ

・演算子の優先順位は、式の演算の順序を決めます。たとえば、乗算演算子は加算演算子よりも優先順位が高いので、先に演算を行います。また、代入演算子や複合代入演算子の優先順位は最も低いので、最後に演算を行います。

・括弧（ ）を用いることで式の演算の順序を変更することができます。括弧（ ）の中が先に演算されます。

・型のランクとは、扱える値の範囲により昇順に並べた型の順番です。一番低いランクの型は byte 型で、一番高いランクの型は double 型です。

・値の型変換には、拡大変換と縮小変換があります。拡大変換とは、ランクの高い型への変換であり、縮小変換はランクの低い型への変換です。拡大変換では値が補われ、縮小変換では値が切り取られます。

・キャスト演算子は、値の型を一時的に変換する演算子です。

・式の演算結果の型は、オペランドの型の中で一番ランクの高い型です。整数型と整数型の割り算の演算結果は整数型になるため、必要に応じてキャスト演算子で拡大変換をします。