Ｊａｖａプログラミング Ｉ

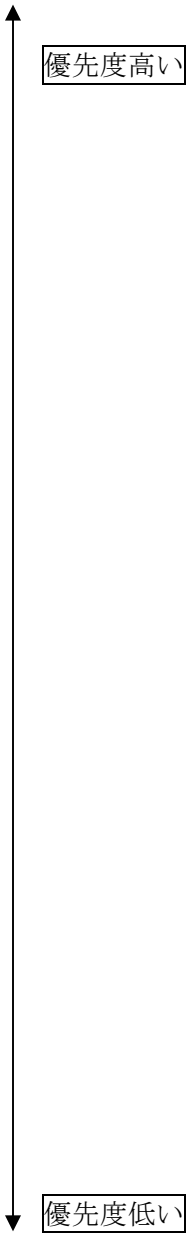
５回目　演算子の優先順位と変数の型変換
２００６年5月１５日（月）

●　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番
但し、“()”で優先順位を変更することができる

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	↑
~	補数	右	同じ優先度
+	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
例えば、a * b % c / d;の場合 → (((a * b) % c) / 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
例えば、~++a;の場合 → (~(++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 1
class Sample5_1
{
    public static void main(String[] args)
    {
        int a=6, b=2, c=5;
        int d1, d2, d3;

        // 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; // d1=(d2=(d3=0));

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

        // シフト演算より加算の優先順位が高い
        d2=1+a<<1;   // d2=((1+a)<<1);

        // 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;   // d3=((a/b)*c);

        System.out.println("a=" + a + ", b=" + b + ", c=" + c);
        System.out.println("a+b*c=" + d1 + ", 1+a<<1=" + d2 + ", a/b*c=" + d3 );
    }
}
```

実行画面

```
>java Sample5_1
a=6, b=2, c=5
a+b*c=16, 1+a<<1=14, a/b*c=15
 -- Press any key to exit (Input "c" to continue) --
```

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２
class Sample5_2
{
	public static void main(String[] args)
	{
		// 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
1+2=12 です。
1+2=3 です。
3*4=12 です。
 -- Press any key to exit (Input "c" to continue) --
```

● 変数の型変換

変数の型変換　　変数の型を変換すること
例えば、double 型から int 型、byte 型から int 型

キャスト演算子“( )”　　式の値を( )内で指定した型に変換する

```
( 型 ) 式;
```

例えば、(double)10、(int)a など

大きなサイズの型に変換（※）　　値は拡張される
例えば、int 型 2 → double 型 2.0

小さなサイズの型に変換　　値は切り捨てられる
例えば、double 型 2.5 → int 型 2
int 型 256 → byte 型 0

※大きなサイズの型に変換する場合はキャスト演算子“( )”を省略することができる

ソースコード例

ソースファイル名：Sample5_3.java

```
// 変数の型変換
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 = (double)b; // ※大きな型への変換であるため( )を省略して c = b;とできる
		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;
		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;
		System.out.println("int 型" + b + " -> byte 型" + a);
	}
}
```

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

● 演算時の型変換

演算時の型変換　　演算前に一方のオペランドが大きなサイズの型に変換されること
演算後の式の値は大きなサイズのオペランドの型になる

例えば、2 * 2.5 → 2.0 * 2.5 → 5.0、5 / 2.0 → 5.0 / 2.0 → 2.5 など

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

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 型に型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。”);
	}
}
```

実行画面

```
>java Sample5_5
5 / 4 は 1.0 です。
5 / 4 は 1.25 です。
 -- Press any key to exit (Input "c" to continue) --
```