５回目　演算子の優先順位と変数の型変換

●　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番
例えば、a = b * c + d; のとき乗算を先に行うというルールのこと

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	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 *① b %② c /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
例えば、~++a;の場合 → ~② ++①a;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;          d1 =③ d2 =② d3 =① 0;

        // 加算より乗算の優先順位が高い
        d1=a+b*c;            d1 =③ a +② b *① c ;

        // 加算より剰余の優先順位が高い
        d2=c%b+a;            d2 =③ c %① b +② 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;            d3 =③ a /① b *② c;

        // 加算より乗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、文字列結合となる
        str1=a/b+"文字列"+b+c;   str1 =⑤ a /① b +② "文字列" +③ b +④ 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);
    }
}
```

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

● （）による演算子の優先順位の変更

式のグループ化　　式の部分を( )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る

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

● 値の型変換

値の型変換　　値のデータ型を変換すること
例えば、double 型から int 型、byte 型から int 型

型変換は型のランクにもとづき区別され処理される

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る
高い ←――――――――――――――――→ 低い
double - float - long - int - short - byte

高いランクの型へ変換　　値は拡張される
例えば、int 型 2 → double 型 2.0

低いランクの型へ変換　　値は切り捨てられる
例えば、double 型 2.5 → int 型 2

○ 変数に値を代入するときの型変換

高いランクの型の変数へ代入　自動的に型変換は行われ、値は拡張される
例えば、
int i = 5;
double di = i;　← ○ エラーにならない

低いランクの型の変数へ代入　自動的には型変換は行われない
例えば、
float fi = 5.0;
short si = fi;　← × コンパイルエラーとなる

キャスト演算子により明示的に型変換を行う必要がある

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する
例えば、(double)10、(int)a など

```
( 型 ) 式;
```

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);
	}
}
```

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) –
```

○　演算を行うときの型変換

演算時の型変換　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る
演算後の式の結果はランクの高い型をもつ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

※ short 型と byte 型のオペランドをもつ場合の注意
このオペランドは演算前に一時的にランクの高い int 型へ型変換される

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) --
```