第6章　演算子

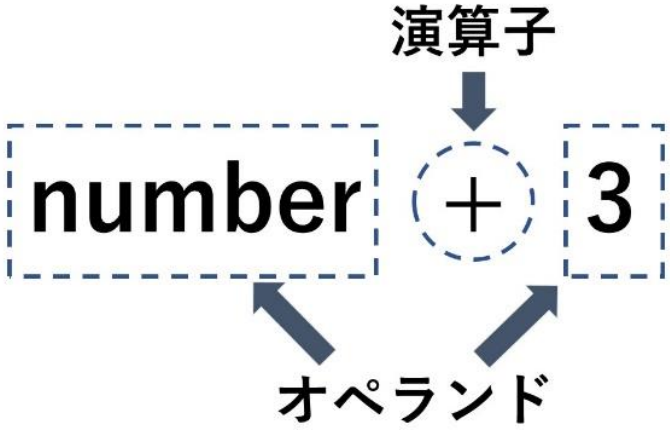
目次

- 演算子
- 演算子の種類
- 演算子の優先順位
- 型変換

演算子：演算子の種類

プログラム中で計算などの処理（演算）を行う際には、演算の種類を表す記号「演算子（operator）」を使用する。

演算子は1個以上の情報に対して演算処理を行う。このとき、演算の対象となる情報を「オペランド（operand）」という。



演算子の種類

記号	名前	記号	名前
+	加算、文字列連結	>	より大きい
-	減算	>=	以上
*	乗算	<	より小さい
/	除算	<=	以下
%	剰余	==	等価
+	プラス（単項）	!=	非等価
-	マイナス（単項）	!	論理否定
++	インクリメント	&&	論理積
--	デクリメント	||	論理和
=	代入演算子	?:	条件演算子（三項演算子）
(型)	キャスト演算子	new	new演算子

算術演算子

加算、減算、乗算、除算、剰余の5つは、「算術演算子」と呼ばれる。
これらの演算子は数値の計算（四則演算）に使われる。

```
5 - 3
```

【Sample0601 算術演算子を利用する】を作成しましょう

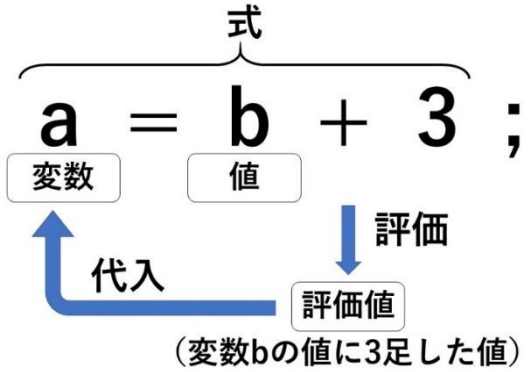
Sample0601のポイント

- 「%（剰余）」は割り算の余りを計算するための演算子
- 「-」、「/」、「%」は左右のオペランドの位置を変えると演算結果が変わる
- 「/」と「%」を使う際は、分母側のオペランドとして数値0を指定しない。エラーが発生する。

式の評価

演算子とオペランドの組み合わせを「式」という。

「+」：左右のオペランドを使用して計算を行い、結果を求める。
結果を求める処理のことを「評価」、
得られた結果の値を「評価値」と呼ぶ。
「=」：右辺の評価値を左辺の変数に代入する機能を持つ。



=（代入演算子）の意味

算数や数学の世界：
右辺と左辺の値が等しい

プログラミングの世界：
右辺の評価値を左辺に代入する

【Sample0602】を作成しましょう

Sample0602のポイント

①と②の式では、右辺の評価値を、左辺の変数に代入している。

①の式では、変数num1とnum2の加算を行い、その評価値を変数sumに代入している。

```
//①
int sum = num1 + num2;
```

Sample0602のポイント

②の式では、変数num1の値に1加算して、その評価値を同じくnum1に代入している。「=」は「右辺の評価値を左辺の変数に代入する」という意味であるため、②のように記述しても問題ない。

```
②
num1 = num1 + 1;
```

演算子の種類:文字列連結演算子

「+」には加算以外に、文字列を連結する文字列連結演算子の機能もある。

```
System.out.println("num1+num2は" + (num1 + num2) + "です。");
```

オペランド2つのどちらかが文字列の場合、「+」は文字列連結演算子として機能する。

「+」演算子は数字の加算だけでなく、
文字列を連結することもできます。

インクリメントとデクリメント

ある変数の値に1加算して、同じ変数に代入する処理
→インクリメント演算子

ある変数の値から1減算して、同じ変数に代入する処理
→デクリメント演算子

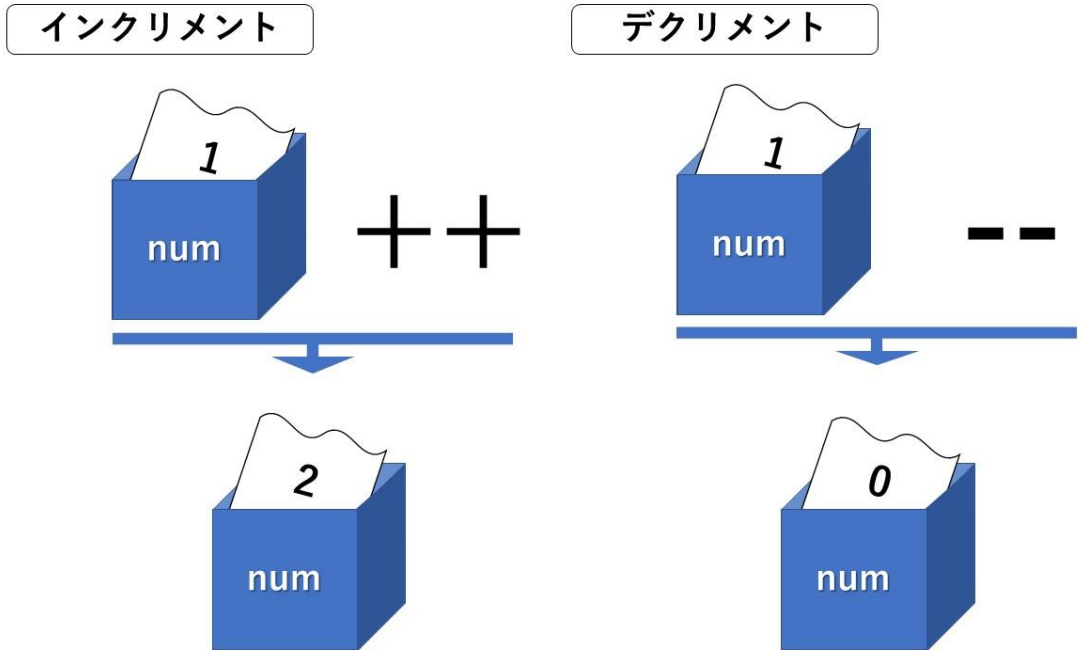
```
num = num++;     // numに1が加算される
num = num--;     // numから1が減算される
```

「+」演算子は数字の加算だけでなく、
文字列を連結することもできます。

インクリメントとデクリメント

名前	記号	特徴	記述例
インクリメント演算子	++	値を1増やす （num = num + 1）	num++
デクリメント演算子	--	値を1減らす （num = num – 1）	num--

インクリメントとデクリメント



インクリメント（デクリメント）演算子を使うと、
変数の値を1加算（減算）できます。

インクリメントとデクリメントの前置・後置

インクリメント（デクリメント）演算子は、オペランドの前と後ろのどちらも記述できる。

後ろに置いた場合：後置インクリメント演算子

```
a++
```

前に置いた場合：前置インクリメント演算子

```
++a
```

【Sample0603 インクリメントの前置・後置】を作成しましょう

前置・後置どちらも、変数の値を1増やすという機能は同じ。
場合によっては前置と後置で実行結果が変わる。

Sample0603のポイント

後置インクリメント演算子を使った場合、
変数num2の値は1となる

```
num2 = num1++;
```

```
num2の値は1です。
```

Sample0603のポイント

前置に直した場合、後置とnum2の値が変わる。

```
num2 = ++num1;
```

```
num2の値は2です。
```

Sample0603のポイント

前置と後置でインクリメント(デクリメント)演算子が実行される
タイミングが異なる。

後置の場合:

1.変数num2にnum1の値を代入する

2.num1の値を1増やす

前置の場合:

1.num1の値を1増やす

2.変数num2にnum1の値を代入する

Sample0603のポイント

インクリメント(デクリメント)演算子と他の演算子の処理を組み合わせると想定外の処理結果になる恐れがある。

開発現場では、インクリメント(デクリメント)演算子の処理は、他の演算とは切り離すように記述することが推奨されている。

```
++num1;
num2 = num1;
```

代入演算子

これまで使ってきた「=」のこと。
「右辺の値を左辺に代入する」という機能を持つ。

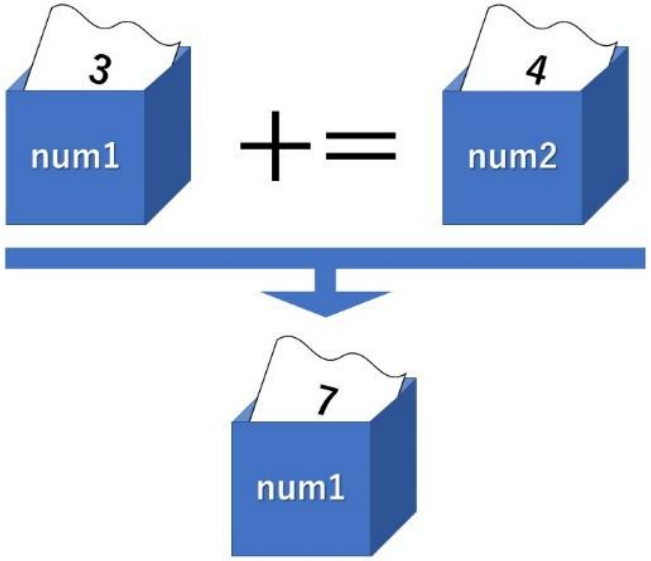
代入演算子

代入演算子の中には、四則演算と代入をまとめて実行できる複合的な演算子がある。

記号	名前	使用例	同義の処理
+=	加算代入	num1 += num2	num1 = num1 + num2
-=	減算代入	num1 -= num2	num1 = num1 - num2
*=	乗算代入	num1 *= num2	num1 = num1 * num2
/=	除算代入	num1 /= num2	num1 = num1 / num2
%=	剰余代入	num1 %= num2	num1 = num1 % num2

代入演算子

「num1 += num2」
num1の値にnum2の値を加算して、その値をnum1に代入する
「num1 = num1 + num2」と同じ結果となる。



優先順位の一覧

演算子には優先順位がある。

1つの式に複数種類の演算子が存在する場合は優先順位が高い演算子から評価される。

（例）

「3 + 5 * 2」では、「*」が「+」より先に評価される

「(4 + 3) * 2」のように式を「()」で囲むと、「()」内の演算子が優先的に評価される。

※一覧はテキスト参照※

優先順位が同じ演算子の評価順

優先順位が同じ演算子の間での評価される順番は、「結合の種類」で決まる。

各演算子は「左結合」、「右結合」のいずれかの種類に分類される。

優先順位が同じ演算子の評価順

左結合の場合、同じ優先度の演算子が存在する場合、左に記述された演算子から優先して評価される。

```
num1 + num2 + 5
```

+は左結合のため、num1とnum2の加算が先に評価され、
次にその評価値と5が加算される。

優先順位が同じ演算子の評価順

右結合の場合、同じ優先度の演算子が存在する場合、右に記述された演算子から優先して評価される。

```
num1 = num2 = 5
```

=は右結合のため、5をnum2に代入した後、

num2の値がnum1に代入される。

【Sample0604 優先順位をチェックする】

演算子の結合について、
文字列連結と組み合わせる場合は注意が必要

Sample0604のポイント

「2 * 10」の計算結果は期待通り「20」と出力されている。
「2 + 10」の計算結果は「210」と想定外の値となっている。

```
2*10は20です。
2+10は210です。
```

Sample0604のポイント

加算演算子の「+」と文字列連結演算子の「+」は同じ優先順位。
下記のように1つの式に混在すると、一番左側の「＋（文字列連結演算子）」が優先的に実行される。

```
"2+10は" + 2 + 10 + "です。"
```

①文字列「2+10は」＋整数値「2」＝文字列「2+10は2」
②文字列「2+10は2」＋整数値「10」＝文字列「2+10は210」

Sample0604のポイント

算術演算を優先して評価させたい場合は、下記のように記述する。

「( )」で算術演算の式を囲むことで、算術演算が優先的に評価される。

```
”2+10は” + (2 + 10) + ”です。”
```

```
2+10は12です。
```

【Sample0605 大きい型に代入する】を作成しましょう

代入する値の型より、変数の型のサイズが大きい場合、代入するとどのような結果になるか。

Sample0605のポイント

処理の途中でint型の変数inum1、inum2の値をdouble型の変数dnum1、dnum2に代入している。

dnum1とdnum2の値を出力すると、それらの値が浮動小数点型であることがわかる。

```
身長は170cmです。
体重は65kgです。
身長は170.0cmです。
体重は65.0kgです。
```

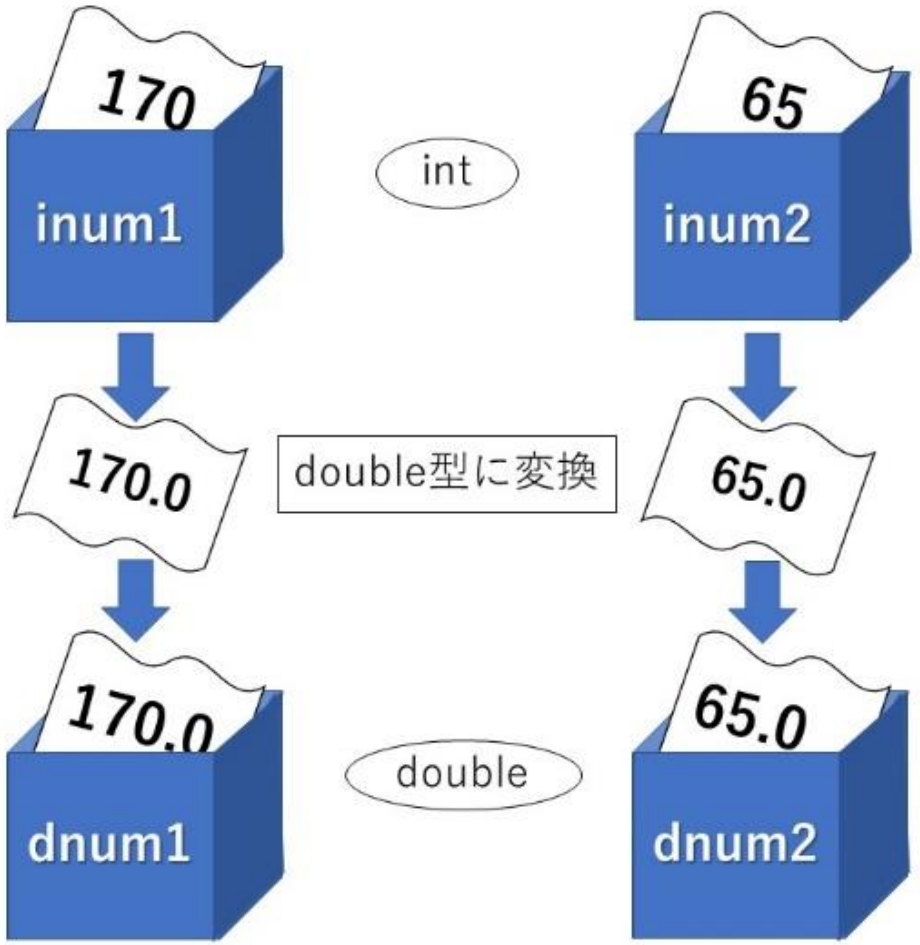
浮動小数点

Sample0605のポイント

代入する値の型よりも変数の型のサイズが大きい場合、その変数に代入すると、自動的に変数の型に合わせて値の型が変換される。この処理のことを型変換と呼ぶ。

```
double dnum1 = inum1;
double dnum2 = inum2;
```

Sample0605のポイント



【Sample0606 小さい型に代入する】を作成しましょう

Sample0606のポイント

小さいサイズの型に代入するには、「明示的な型変換（キャスト）」を行う必要がある。

キャストにはキャスト演算子(「( )」を利用する。

```
（変換先の型名） 式
```

Sample0606のポイント

キャスト演算子は、式の型を「( )」内で指定した型に強制的に型変換できる。

サンプルソースを以下のように編集すると、実行できるようになる。

```
int inum1 = (int) dnum1;
int inum2 = (int) dnum2;
```

Sample0606のポイント

int型にキャストした際には小数点以下の値は切り捨てられる。

```
身長は170.5cmです。
体重は65.3kgです。
身長は170cm です。
体重は65kg です。
```

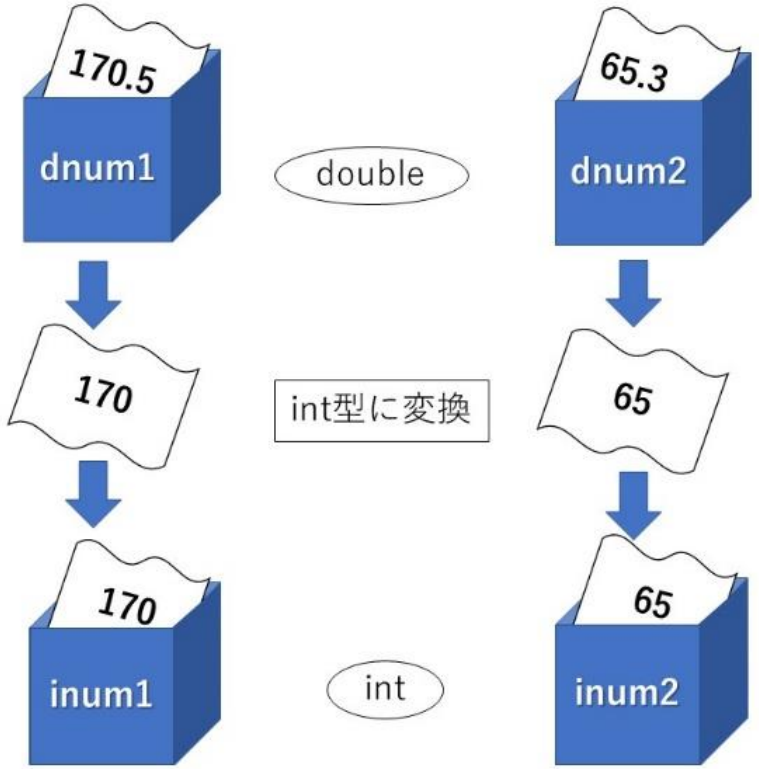
int型にキャスト後の値

Sample0606のポイント

キャスト演算子はサイズが小さい型から大きい型への変換にも利用できる。

```
int inum = 170;
double dnum = (double) inum;
```

Sample0606のポイント



キャスト演算子で明示的な型変換が行えます。

【Sample0607 異なる型同士で演算する】を作成しましょう

Sample0607のポイント

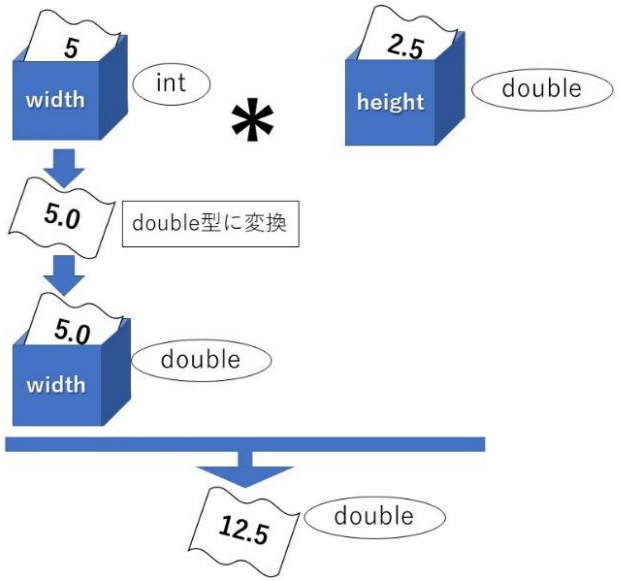
int型とdouble型の数値を算術演算子で評価している。
出力結果から、その評価値はdouble型であることが分かる。

```
四角形の面積は12.5cm2です。
```

Sample0607のポイント

異なる型の値で評価した場合、小さいサイズの型を大きいサイズの型に変換してから演算を行う。

そのため、サンプルコードでの評価値はint型よりもサイズが大きいdouble型となる。



【Sample0608 型変換のタイミングの確認】を作成しましょう

演算を行う際にはどのタイミングで型変換が発生するかを理解しておくことが重要。

Sample0608のポイント

「6 ÷ 4」の評価値は「1.0」となっている。

```
6÷4 は1.0です。
```

Sample0608のポイント

以下の順序で処理が行われる。

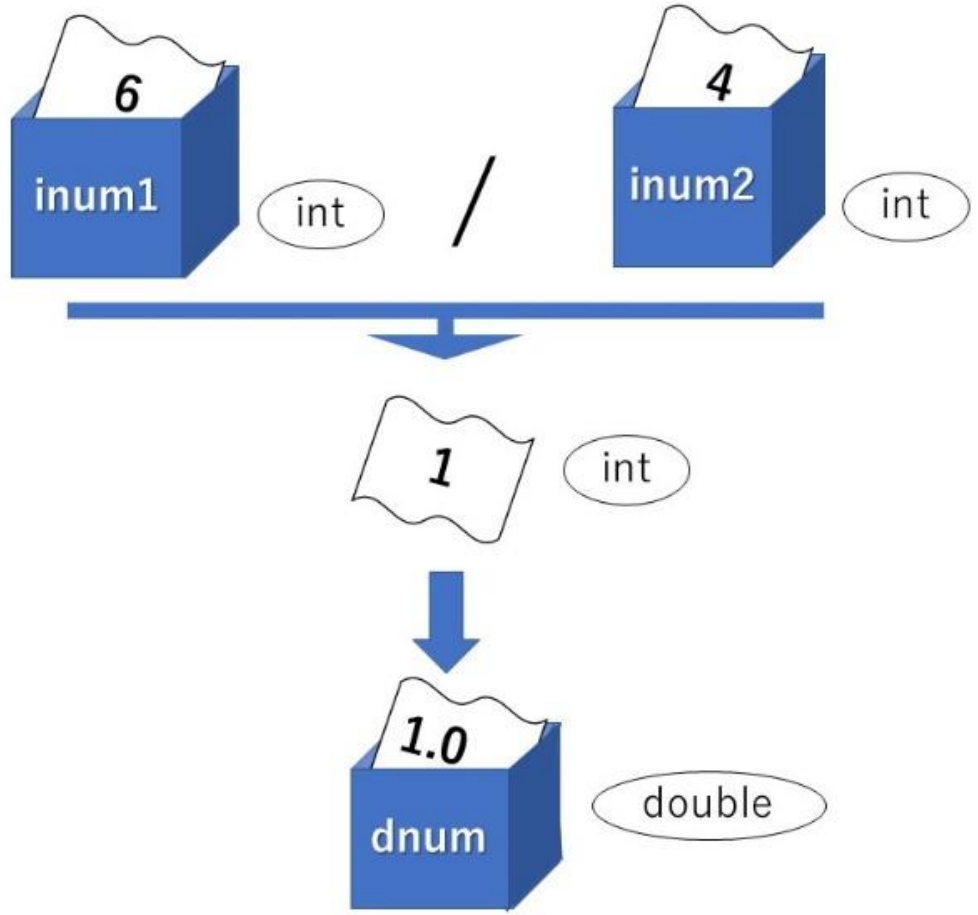
①変数num1とnum2の除算

num1とnum2はどちらもint型であるため、「6 ÷ 4」の評価値は1の位までしか求められない。評価値は「1」となる。

②評価値「1」をdouble型の変数dnumに代入する。

double型はint型よりもサイズが大きいため、暗黙的な型変換が発生し、変数dnumには「1.0」が代入される。

Sample0608のポイント



Sample0608のポイント

期待する結果「1.5」を求めるためには…
①int型の変数のいずれかをdouble型に型変換
②除算

```
double dnum = (double) inum1 / inum2;
```

6÷4は1.5です。

章のまとめ

- Javaの式の多くは、演算子とオペランドの組み合わせによってつくられています。
- +演算子は文字列を連結する働きをします。
- インクリメント（デクリメント）演算子を使うことで、変数の値を1加算（減算）できます。
- 複合的な代入演算子を利用することで、四則演算と代入を簡潔に記述できます。
- 演算子の優先順位を間違えると、誤った実行結果になることがあります。
- キャスト演算子を使ってサイズが小さい型の変数に、サイズが大きい型の値を代入できます。