Chapter 4.　変数、定数と式の評価

4ー1.　式とは？

■式は計算の基本

コンピュータにさまざまな動作をさせる基本は、文であることを前の章で説明しましたが、文中でさらにコンピュータにいろいろな計算をさせるための記述の仕方として式（expression）があります。式は、それ自体でも一つの文を構成することができるのですが、後で説明する代入文の中やメソッド呼出しのときに用いられます。式は、プログラミング言語において非常に重要な記述の仕方の一つです。式は、次のような要素から構成されています。

▼式の構成要素
- 変数（Variable）
- 定数（Constant）
- 演算子（Operator）

変数は、後で説明しますが、計算の途中結果を覚えておくための名前です。たとえば、数学で習ったようなxやyなどの変数を思い浮かべてみて下さい。定数は、数値や文字列などの値自身を示しています。例えば、12という数値や"Spring"といった文字列がこれに該当します。定数とは、値自身が変数のようにどんどんその内容を変えることがなく、一定の情報を示すことから、この名前がついています。演算子とは、プラス「+」やマイナス「–」といった記号のことです。詳細は、後で述べますが、例えば次のような足し算の式は、Javaでも用いることができます。

```
123 + 89453 + 234
```

定数に関して、12や"Spring"といったように、表現しているものが異なる値が存在しています。表現している内容を分類したものを型（Type）と呼んでいます。式の計算結果や変数などは、すべて最終的には定数になりますので、定数だけでなく、変数や式全体に関しても型で分類することになります。型には、例えば次のようなものがあります。

▼型の例

整数型	-1, 0, 1, 2といった整数（離散数）を表しています。
実数型	56.2, 0.00003といった実数（連続数）を表しています。
文字列型	"March"といった文字の連なりを表しています。

離散数、連続数とは、数学上での分類の仕方です。下記の数直線を見ますと、整数は、飛び飛びの点でしかなく、実数は連続した直線になっています。飛び飛びということで、離散という名前がついてます。つまり、整数は、たとえば1と2の間には、数が存在しないということなのです。

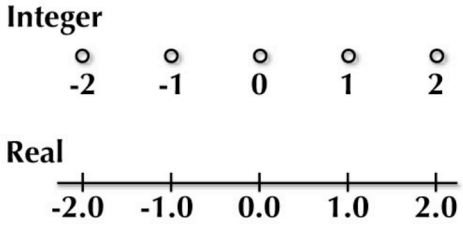

図4-1数直線

4ー2. 型と定数

式の中には、定数を記述することができます。それぞれの型の定数の値の範囲と記述例は以下の通りになっています。なお、各型に対応して、クラスが用意されています。これは、ラッパー（Wrapper）クラスと呼ばれています。クラスの方は、それほど頻繁には使わないのですが、型を変換してくれるようなメソッドが定義されており、それを利用することがあります。

４－２－１．論理値の定数（型名 boolean）

条件などが満足しているかどうかを示すのに用いるための定数です。値は以下の２つしか用意されていません。**true**は、満足している（真である）ことを示し、**false**は満足していない（偽である）ことを示します。

trueまたは**false**

論理値は、繰返しや条件分岐で条件を判定するときに用いられます。アプリケーションで端末に表示するときは、System.out.printlnメソッドなどを用いてください。

```
System.out.println( true );        //    文字列でtrueと表示されます
System.out.println( false );       //    文字列でfalseと表示されます
```

４－２－２．整数の定数（型名 byte, short, int, long）

数えることのできる数値を表現するのに用いるための定数です。数学的には、整数は、離散数（Discrete Number）とも呼ばれています。整数型は、一般にはIntegerの名前から、intという型が用意されているのですが、内部的に表現するためのビットサイズに応じて、その他の型も用意されています。各型に対して、表せる範囲が限られているので注意してください。

byte	-128〜127	// 8ビット
short	-32768〜32767	// 16ビット
int	-2147483648〜2147483647	// 32ビット
long	-9223372036854775808L〜9223372036854775807L	// 64ビット

定数の例：330,　　-522167,　　+20394

定数そのものを記述した場合は、int型の定数とみなされます。また、足し算や引き算などを行なったときに、内部の計算では、byte型、short型もint型に変換されて計算されます。強制的に64ビット長の、long型の整数にしたい場合には、最後にLかl（小文字のエル）を付けます。

long型の整数の例：456L, 80l

通常は、数は10進数で表しますが、Java言語では特殊な用途のために、8進数と16進数も表現することができます。これは、元々コンピュータが2進数を使ってすべての情報が表現されていることに起因しています。2進数は、1と0だけを用いて情報を表しますが、これだと人間には非常に読みづらいので、2進数を3桁ごとに区切った8進数、4桁ごとに区切った16進数がよく用いられます。

8進数の場合は、最初に0を付けます。数は0〜7までしか使えません。

8進数の例：0772

16進数の場合は、最初に0xを付けます。数字の0〜9以外に、文字のa〜fまでも用いることができます。A〜Fは大文字でも小文字でも構いません。文字のa〜fは、10進数に換算すると、10〜15までの数を表していま

す。

16進数の例：0xafdd, 0x568e

アプリケーションの場合は、整数を表示するのにも、System.out.printlnメソッドを用いることができます。

```
System.out.println(  45  );          //  45と表示されます
System.out.println(  -78L  );        //  -78と表示されます
System.out.println(  0x44  );        //  16進数が変換され、10進数で68と表示されます
```

４－２－３．実数の定数（型名 **float, double**）

実数は、連続数を表現するのに用います。連続数とは、数直線を考えたときに、どこを切っても更にその間の数があるというような性質（linearと呼ぶ）を持った数のことです。簡単に理解するには、小数点以下の数を持つ数（あるいはJava上では小数点がついている数）が実数と考えればよいでしょう。実数も２つの表現範囲を持つ型が用意されています。float型の実数は単精度浮動小数点実数（Floating point real number）と、double型の実数は倍精度浮動小数点実数（Double precision Floating point real number）と呼ばれています。

float　$1.40129846432481707 \times 10^{-45} \sim 3.40282346638528860 \times 10^{+38}$

double　$4.94065645841246544 \times 10^{-324} \sim 1.78769313486231570 \times 10^{+308}$

実数は、内部的には国際規格のIEEE-754と呼ばれる浮動小数点表現の規格に基づいて表現されています。floatの方は、32ビットで表現されていますし、doubleの方は64ビットで表現されています。上の範囲をみればわかるように、double型の方が、より精密な数、より大きい数を表すことができます。

通常に実数の定数を書きますと、double型の定数であると解釈されますが、明示的に型の指定をすることもできます。実数定数の最後にfかFを付けると強制的にfloat型になります。同様に、実数定数の最後にdかDを付けると強制的にdouble型になります。

型を明示した実数定数の例：	3.141459F	1.603e-45f	33.1d	0D	3F
小数点の反対側を省略した例：	-.05	34.			

実数では、指数表現が可能です。eまたはEのあとに、10のべき乗数を書きます。

指数表現を使った実数の例：	45.6e9 3	.14159E-19	5.5e+4

指数表現とは何でしょうか？知らない人のために、簡単に説明します。指数表現とは、以下のように、基数（ここでは10）の何乗というものを掛け合わせた実数の表現形式のことで、非常に大きい数や小さい数などを表現するときに用います。

通常の書き方		指数表現		Javaでの指数表現
0.00000567	→	5.67×10^{-6}	→	5.67e-6
123000000	→	1.23×10^{8}	→	1.23e8

ちなみに、このように0が少なくなるように指数表現を使って桁あわせを行なうことを、実数の正規化（Normalization）と呼んでいます。IEEE-754規格で表現されているJavaの実数定数でも、限られたビットサイズで、非常に小さな数から大きな数を表すために、内部的に正規化を行なっています。

System.out.printlnメソッドによる実数の表示は、なんとなくお任せ的で、0が一杯になるときは、指数表現されます。そうではないときは、普通に表示されます。

```
System.out.println(  45D  );          //  45.0と表示されます
System.out.println(  0.00000032  );   //  3.2E-7と表示されます
System.out.println(  55e+10  );       //  5.5E11と表示されます
```

４－２－４．文字の定数（型名 char）

1文字でで表せる英数字、ひらがな、漢字等を表現するのに用います。文字定数はシングルクォーテション(')で囲みます。

例：'a'　'A'　'8'　'*'　' '　'Å'　'あ'　'蛸'　'δ'　'Ж'

次のような特殊文字が用意されています。バックスラッシュ¥は、￥マークで表されるフォントもあります。ただし、改行、水平タブ、改ページ、バックスペース、復帰などの制御コードは、オペレーティングシステムごとに異なりますので、使用するのを避ける指導がなされています。

¥n	改行	¥t	水平タブ
¥f	改ページ	¥'	シングルクォーテーション
¥b	バックスペース	¥"	ダブルクォーテーション
¥¥	バックスラッシュ	¥r	復帰

特殊文字は、それほど使わないかも知れませんが、改行や水平タブなどは結構使う場合があります。水平タブは、文字ベースの端末（unixなどが稼働している古い端末）において、何行もあるようなテキストを表示するときに、桁の位置あわせをするときなどに用いられます。

文字の定数は、内部的には国際規格のUnicodeと呼ばれる２バイトの文字コード表現の規格に基づいて表現されている。この規格は、日本語以外に、英語、ロシア語、中国の漢字など、多くの国で使われている文字を表すことができる規格です。直接Unicodeの文字コードを使って、文字を表現したいときは¥uという接頭語をつけて、4桁の16進数で文字を表現することもできます。

例：　¥u6f22　// 実は、漢字の「漢」を示しています

端末画面が日本語表示可能な実行環境であれば、System.out.printlnメソッドは、Unicodeの文字も表示してくれるでしょう。ただし、プログラムのソーステキストの漢字などを一旦Unicodeに変換するためのコンパイラのオプション設定が必要になります。この設定については、開発環境に依存しますので、知っている人にきいてみてください。

```
System.out.println( '丸' );          //  日本語が表示可能であれば、丸と表示されます
System.out.println( '¥u6f22' );      //  日本語が表示可能であれば、漢と表示されます
```

４－２－５．文字列定数（クラス名 String）

文字列には、型がなく、その替わりにクラスとして定義されています。複数の文字の連なりを表現するために用います。複数の文字を扱うのでオブジェクトとしてみなした方が整合性が良いからでしょう。文字列は、最初から出てきましたので、もうおなじみでしょう。文字列定数は、ダブルクォーテーションで囲みます。日本語や上で示したような特殊文字も使えます。なお、下の例のように、ダブルクォーテーションマークそのものを文字列中で表現したいときは、¥"という書き方を使います。

例：　"This is an apple."

"This is a long message, ¥"WOW¥" 春に花が咲きます"

System.out.printlnメソッドでの表示は、文字のときと同じように日本語で表示したいときは、端末を示すウィンドウのフォントなどを調整したり、コンパイラでのオプション設定が必要となります。

```
System.out.println( "雨にも負ケズ、風にも負ケズ" );        //  アプリケーションで実行してね
```

文字と文字列については、また後の章で詳しく説明します。

４－３．変数

４－３－１．変数と宣言

変数とは、計算の途中結果や最終結果を覚えておくための道具です。直感的には、数値や文字などの定数を覚えておくことができる箱をイメージするとわかりやすいでしょう。

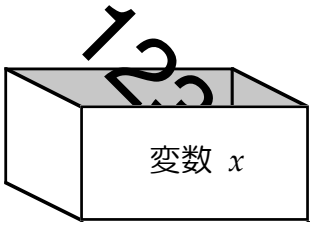

図4-2 変数の箱

変数を使うためには、予め「実は、かくかくしかじかなる変数を使わせて戴きます。以降、お見知りおきをよろしくお願い申し上げます…」というような宣言をしておかなければなりません。江戸時代のやくざ仲間での仁義を切るのと似ています。そうしないと、コンパイラが「こんな名前の変数なんか知らない！」と文句を言って、エラーにしてしまうからです。

ということで、最初に変数を使う前に宣言（Declaration）をしておきましょう。次のような書式で行ないます。

▼宣言の書式

```
型名      変数名；
```

書式で使われている型名とは、変数が覚えるデータの型を示しています。次の幾つかの型があります。

boolean	論理型	論理値を保持する変数を宣言するとき
int	整数型	整数値を保持する変数を宣言するとき
char	文字型	文字を保持する変数を宣言するとき
double	実数型	実数値を保持する変数を宣言するとき

変数の名前は任意の名前をつけてよいが、英文字で始まる必要があります（注１）。コンパイラは、３２文字ぐらいだったら、変数の名前をちゃんと識別することができます。例えば、

thisisalonglongvariableasyoulike

というような変数名をつけても構いません。変数名の途中に_（アンダーバー）も使って構いません。変数の宣言は、例えば、つぎのように行ないます。

```
int     x;          // xという名前の変数を宣言する
```

こういった宣言は、コンパイラに怒られますので、変数が使われる前に行なっておかなければなりません。上の例では、変数xが最初に現れるのは、かならず変数宣言の中です。宣言も１つの文であり、宣言文と呼ばれています。同じ型の変数を宣言するときは、カンマで区切って複数の変数を宣言することができます。例えば、次のように、同じ型の２つの変数が宣言されているとしましょう。

```
int     x;          // xという名前の変数を宣言したが、
int     y;          // yという名前の変数も欲しくなってしまった
```

これは、以下のように、一回の宣言文で宣言することができます。

```
int     x, y;       // 面倒なので一緒に宣言してしまう
```

（注１）変数の名前は、英字だけに限らずUnicodeの文字（日本の文字など）を含めて、使うことが可能ですが、ここでは英字だけを使うことと致します。

４－３－２．変数への代入

変数には、計算結果を保持するために値を代入することができます。代入（Assignment）は、代入文で行ないます。

▼代入文の書式

```
変数 = 式;
```

代入文では、まず式が評価されて単一の値（定数）として計算結果が得られます。その得られた定数値を変数が保持するのです。

例

```
x = 67;
```

…変数*x* に67が代入される。

```
y = 35 - ( 24 * 2 );
```

…変数*y* に–13が代入される。

変数は、他の式で参照される前に、まずは何らかの値が代入されていなければなりません。もし、代入し忘れて、うっかりその値を参照したとすると、たいていの場合はコンパイラに怒られます。フィールド変数の場合は許されるのですが（注１）、そのときは次の値が予め代入されているものとします。

boolean型の場合	**false**
char型の場合	Unicodeのヌル文字（\u0000）
int型の場合	0
double型の場合	0.0

＝の左辺と右辺を取り替えてはなりません。ですから、以下の代入文は、誤りです。

```
34 = x;     //誤り。コンパイラにこっぴどく叱られてしまうので注意のこと
```

注１　後の章で紹介するフィールド変数（インスタンス変数）の場合だけです。通常はローカル変数なので、コンパイラに代入されていないと叱られてしまいます。

４－３－３．変数の値の参照

変数が式中で用いられた場合、変数が保持している値に置き換えられます。

例：仮定として、整数型の変数*x*に67、実数型の変数*y*に、0.56が入っているとします。

y * 10.0
…この式の評価値は5.6となる。
y * 10.0 → 0.56 * 10.0 → 5.6

z = *x* - 47;
…zには、20が代入される。
z = *x* - 47 → *z* = 67 - 47 → *z* = 20

４－３－４．宣言時のおける初期値代入

変数を宣言する際には、同時に初期値を代入することができます。これによって、宣言と最初の代入を１つにすることができます。次のような書式で記述します。

▼宣言時の初期値代入の書式：

```
型名　変数名　＝　式；
```

例えば数値型の変数の場合は、次のように使います。

```
int      z = 10;          // 数値型の変数zを宣言し、その値を10にする
double   y = z * 67.0;    // 倍精度浮動小数点実数型の変数yを宣言し、その値をz * 67.0にする
```

上記の２番目の例 のように既に存在する変数を使っての初期化することも可能です。しかし、右辺の式中に出現する変数は、それまでに値が代入されていなければいけません。

同じ型の変数の宣言は、カンマ（, ）で区切っていくつも書いても構いません。

```
int      x = 20, y = 30;          // 数値型の変数xとyを宣言し、その値を設定する
int      w = x * 20, u = y / 20;  // 同様
```

なお、次のように記述した場合は、変数*y*は初期化されますが、変数*x*は初期化されないので注意してください。

```
int      x, y = 30;               // 数値型の変数xとyを宣言し、yの値だけ設定する
```

４－４．式と評価

４－４－１．式の記述の仕方

変数を宣言した後は、宣言された変数や定数、そして演算子を使って式を記述することができます。式は、いろいろな計算をさせるための記述で、基本的には算数や数学で習ったものと同じと考えてください。式には、次のようなものがあります。他にもいろいろあるのですが、ここでは一番簡単なものだけを挙げておきます。

▼式の書式（簡易版）

定数そのもの	
変数そのもの	
式　＋　式	足し算
式　－　式	引き算
式　＊　式	掛け算
式　／　式	割り算
式　％　式	余り
（　式　）	括弧をつけて先に計算させる

掛け算や割り算の記号が、通常のものとは少し異なることに注意しておいて下さい。例えば、この式の書式に基づいて、次のように記述できます（ここで、ｘやｙは変数の名前）。

$$589 + y * (x - 28)$$

式は必ず計算されて、計算結果が単一の値（定数）になります。この計算のことを式の評価（Evaluation）と呼んでいます。上の例であれば、変数ｘに13という値が、変数ｙに56という値が入っていれば、計算結果は、-251という値に落ち着きます。

式を考える上で、２つの側面を見なければなりません。まずは、記述されたものが式の書き方の規則に則っているかどうかです。上の例について考えてみましょう。

589	「定数そのもの」だから式
y	「変数そのもの」だから式
x	「変数そのもの」だから式
28	「定数そのもの」だから式
$x - 28$	「式　－　式」に合っているので式
$(x - 28)$	「（　式　）」に合っているので式
$y * (x - 28)$	「式　＊　式」に合っているので式
$589 + y * (x - 28)$	「式　＋　式」に合っているので式

このように、よく見てみると上の規則に則っているので式であるとコンパイラに認識されています。このような書式上の規則のことは、自然言語に倣って文法（Syntax）」と呼ばれています。文法にあっていないものは、式として認識されません。例えば、以下の記述は、Javaの式の文法としては間違いです。

$$3x + 5y$$

４－４－２．式の評価と優先順位

Javaの式を考える上で、式が構文規則に則っているかに加えて、式はどこから計算されるかという点にも気をつけなければなりません。式は、評価されて、最終的には単一の定数値になります。この評価の際には、優先順位（priority order）について考える必要があります。これは、一般の数学（算数）と同じです。まず、括弧のついているものは、先に評価されます。括弧が多重になっている場合は、一番内側から評価されます。また、掛け算、わり算、余りについては、足し算や引き算よりも先に評価されます。では、以下の式が一体どこから評価されるかについて考えてみてください。

$$2 + ((x - 3) * y) / 4$$

以下に、より正確に式の評価における優先順位の法則について記述していきます。

★整数・実数で使える演算子の優先順位

▼乗法演算子と加法演算子

乗法演算子とは、　　*（掛け算）、　/（割り算）、および　%（剰余算）
加法演算子とは、　　+（足し算）、および　–（引き算）

Javaでは、掛け算・割り算・剰余算の３つの演算のことを乗法演算と呼び、足し算・引き算のことを加法演算と呼んでいます。通常の算数と同様に、乗除剰余算が先に計算されます。乗法演算子と、加法演算子の中では、同じ優先順位を持ちます。これを理解しておきますと、式の評価は次のようにまとめることができます。

▼式の評価（Evaluation of Expression）

・変数があれば、まずその変数が保持する値に置き換えられる。
・優先順位の高い順から演算子が評価される。
・同じ優先順位のものは左から右に（乗法演算子や加法演算子の場合）評価される。
・括弧で囲まれた式は、先に評価される。
・最終的に１つの定数になるまで評価は続く。

式の評価の例：xが20の値を持っているとします。

```
y = ( 10 + x ) * 5 – 15;
        ↓
    10 + 20
      ↓
      30   * 5
          ↓
        150   – 15
            ↓
           135
```

４－４－３．メソッド呼出しと式

式は代入文の右辺に書ける以外にも、メソッド呼出しのパラメータの部分にも記述することができます。

```
gc.drawString( "Today's Menu", 34 + 5, 27 + y );
```

メソッドのパラメータの部分に記述された場合は、次の順番で実行が行なわれます。

１．パラメータの式が評価され、定数になるまで計算される
２．その定数の値をパラメータとして、メソッドが呼び出される。

この例の場合には、まず34+5を計算し、次に27+yを計算してから、drawStringメソッドが呼び出されることになります。

★パラメータの型

メソッド呼出しのパラメータとして、式や定数、あるいは変数を記述するときには、型を気にする必要があります。例えば、drawStringメソッドは、第１パラメータとして文字列を要求しています。ですから、

```
g.drawString( 45, 100, 100 );                    // 整数であり、文字列ではない
g.drawString( 32.0/11 , 100, 100 );              // 計算結果は実数であり、文字列ではない
```

のようなメソッド呼出しは、コンパイラによって間違っているとしてはねられてしまいます。45という整数の数と、"45"という数字からなる文字列は違う型に分類されます。文字列に変換して、整数や実数を表示させる方法は、次の節で説明します。

４－５．整数演算と型の変換

４－５－１．整数にまつわる特殊な演算

★整数除算

整数除算と実数除算は、Java言語では異なるものです。電卓のように、整数どうしを割ったときに、割り切れなくなったら自動的に実数に変換してくれると思ったら大間違いなのです。整数除算は、小数以下を切り捨ててしまいます。また、マイナスがついたときなども注意しましょう。

整数除算の例：

```
5 / 3        →   1
1 / 3        →   0
5 / 2 * 2    →   4
5 / – 3      →   – 1
```

★剰余算

剰余算は、整数でも実数でも行なうことが可能です。ここでは、整数の場合について扱ってみます。直感的に、整数についての剰余は小学校でもやった内容なのでわかりやすいでしょう。ただし、除数や被除数のいずれかにマイナスの符号がついていた場合には、少しやっかいです。以下の例をみて、どうして、そのような結果がでるのかコメントを見てみてください。

整数の剰余算の例：

```
68  % 12     →   8
55 % –12     →   7        // 整数除算：55/–12の結果は –4 、–12*–4 + 7→55
–9 % 5       →   – 4      // 整数除算：–9/ 5の結果は –1 、5 * –1 – 4 → –9
–26 % –5     →   – 1      // 整数除算：–26/–5の結果は 5 、–5 * 5 – 1 → – 26
```

４－５－２．整数と実数の型の変換

実際の数値計算では、整数以外に実数を基本として計算することが多いでしょう。たとえば、金銭計算などの数値情報にしても、利率などを考えると実数を使わざるを得ません。これまでは、主に整数を基本として数値計算を考えてきましたが、ここでは実数を基本とした数値計算を考えましょう。まずは、整数と実数の間の変換から考えます。

★暗黙の型変換

実数が式に現れた場合は、Javaでは評価結果は、必ず実数（しかもdouble型）になります。次の例のように、実数で答えが欲しい場合は、演算の対象となる数値を実数表記（小数点をつける）にします。

例：		
	10/8.34	答えは実数
	10/8	整数の除算で、答えは整数の1
	10/8.0	実数の1.25が答え

変数への代入としては、整数の式を実数型の変数に代入するときは、そのままで何もする必要はありません。次の例のように、実数に変換されて代入されます。

例：　　**double** $x = 10;$　　// xには10.0が代入される

★明示的な型変換

逆に、実数を整数に直すような場合は、丸め（Round）を行なわなければなりません。丸めとは小数部分を取り去る操作のことを指します。そのためには、明示的な型変換が必要となってきます。まず、式の評価結果を別の型の値に変換するための書式を紹介しましょう。

型を別の型に変換するためには、キャスト（cast）という式が用いられます。

▼キャストの書式

```
( 変換したい型 )　式
```

型によっては、変換できないものもあります。例えば、5という数はそのままでは文字列の"5"に変換することはできません。しかしながら、整数と実数の間ではキャストを使えばかなりうまく値を変換することができます。次の例は、幾つかの式を別の型に変換させてみたものです。

例：

```
(double) 5 / 2          // 結果は5.0 / 2となり2.5である。
( int ) 34. 1           // 結果は、34となる
( int ) – 45.5          // 結果は、-45となる
( long ) 45             // 結果は45Lとなる
```

例にありますように、実数から整数への変換は、切り捨てという形になります。これをJavaでは「0に近くなるように丸める」と呼んでいて、整数部だけを取り出すことになります。他の言語では、truncateあるいはtrunc関数と呼ばれているものが、この操作にあたります。

実数の式を整数型の変数に代入するときは、このキャストを用いて型の強制変換を行なう必要があります。ここで、よく間違いがちなのが優先順位です。キャストは、単項演算子の一種で、掛け算や割り算などよりも高い優先順位を持ちます。そのため、次の例にありますように、括弧をうまくつけないと整数の式になってくれません。

例：

```
int    z = 10/8.34;          // これは間違い、エラーになる
int    z = (int) 10/8.34;    // これも間違い、優先順位が違う
int    z = (int) (10/8.34); // これが正解
```

ただし、実数の式の評価結果が、整数で表せる範囲を超えた場合は、エラーになります。例えば、次のような実数を整数型にして、代入するとプログラムの実行時にエラーが発生します。

```
int    x = (int) 45.23e56;    // 45.23 × 10^56は、整数では表しきれない
```

これは実行時のエラーに分類されます。後の章で出てきますが、実行時のエラーは例外と呼ばれています。

４－５－３．数から文字列への変換

整数や実数を文字列に直すことを考える前に、まず文字列どうしの足し算を考えてみましょう。文字列定数どうしは、引き算や掛け算などはできませんが、足し算だけは可能です。これは、文字列をそのまま結合させる演算を行なうものです。

"千里の路も" + "一歩から"　→　"千里の路も一歩から"

数字どうしを文字列として足し合わせてみると、ちょっと整数の足し算とは違う結果になります。

"2345" + "6789"　→　"23456789"

ところで、Javaでは、文字列どうしだけでなくて、文字列と整数、あるいは文字列と実数なども足し算することができます。このときに、整数や実数は、文字列と足し算を行なうと自動的に文字列に変換されます。

333 + "mの高さのタワー"　→　"333mの高さのタワー"
12 * 5 + "個のみかん"　→　"60個のみかん"

これを利用して、変数が持つ値を表示させることも可能となります。例えば、変数xに10が入っていたとすると、次のように変換されます。

" x の値は" + *x* + "です"　→　"xの値は10です"

特に、何も中身のない文字列と結合すると、すぐに整数や実数をそのまま文字列に変換することができます。同じように変数xに10が入っているとしましょう。中身のない文字列はダブルクォーテーション・マークを２回続けて記述します。

"" + x　→　10
x * 10 + ""　→　100

ただし、足し算は左から行なわれていきますので、予め整数の足し算を計算しておきたいときは、次のように括弧を付ける必要があります。先ほどと同じで、変数xに10が入っていたとしましょう。

" 高さ" + *x* + 4 + "フィート"　→　"高さ104フィート"
" 高さ" + (*x* + 4) + "フィート"　→　"高さ14フィート"

これらの文字列の足し算機能を使って、整数や実数の計算結果や、変数の値を、System.out.printlnメソッド以外のshowStatusメソッドやdrawStringメソッドの中で表示することができます。

```
System.out.println( "東京にある" + 111 * 3 + "mの高さのタワー" );
System.out.println( x + "坪の大きさ" );
g.drawString( "体重" + x * 4 + "kg ", 100, 20 );
showStatus( "value of variable x is :" + x + ". This is " + true + ". " );
```

文字列から、整数あるいは実数に変換するためには、文字列の足し算や、強制的な型変換で行なうことはできません。型に対応しているラッパークラスの中のメソッドを用います。それらは、文字列を入力として受け取るプログラムで使いますので、後の章でご紹介します。

４－６．課題

課題4-1.

変数の値の推移を追ってみましょう。まず、変数*x*に１を代入します。その後、変数*x*の値を2倍して、その値をSystem.out.printlnで表示させます。この2倍して表示させる操作を32回ぐらい繰り返してみてください。最後に、変数*x*の値はどれくらいになっているでしょうか？もし、最後に変数の値が予想と異なるものになったときは、それは何故か考えなさい。アプリケーションプログラムで行ないます。クラス名は、BinaryCalcにしましょう。

（ヒント）System.out.println("xの値は" + *x*);というように、文字列と結合させて表示させましょう。

課題4-2.
整数除算と実数の除算の結果を比較しなさい。

課題4-3.
実数の剰余算の式を考え出し、符号などを変えていろいろな場合（最低、除数の符号が違う場合と被除数の符号が違う場合の組み合わせで４つの場合は必要です）の結果を確かめなさい。