

Chapter 4. 変数、定数と式の評価

4.1. 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" という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。

注1 離散数、連続数とは、数学上での分類の仕方です。

4.2. 定数

■型と定数の記述の仕方

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。

★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には、`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
```

★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
```

★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る。

```
float    1.40129846432481707e-45～3.40282346638528860e+38
double   4.94065645841246544e-324～1.78769313486231570e+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

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。

通常の見方		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、なんとなくお任せの（注1）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
```

★文字の定数（型名 char）

1文字で表せる英数字、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ーション(')で囲みます。

例：'a' 'A' '8' '*' ' ' 'Ã' 'あ' '蛸' 'ð' '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。

\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
```

★文字列定数（型名 String）

複数の文字の連なりを表現するために用います。これは、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\n"

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");             // その通りに表示されるはず、、、
```

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。

注1 整数や実数の表示の桁数を制御したいときは、java.textパッケージにあるDecimalFormatクラスを利用します。

4.3. 変数

■変数とは？

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。

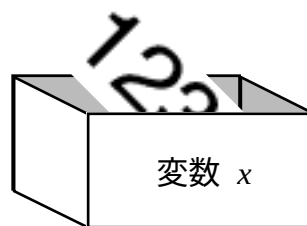


図4-1 変数の箱

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。

■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。場合によって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 (<code>\u0000</code>)
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 後の章で述べる、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。

■変数の値の参照

変数が式の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
```

■変数の自己参照による代入

例えば、以下の記述があるとしましょう。この代入分の羅列は、変数の値を変えていますが、その変数が持っていた過去の値とは、まったく関係なしに、新しい値をどんどん代入してしまっています。

```
x = 30;
x = 20;
x = 10;
```

一方、次のように変数の過去の値を参照して、その変数の新しい値を計算することができます。

```
x = x - 10;
```

これは、今までの数学の常識から考えれば、かなりおかしい記述です。しかしながら、Javaの代入の=というのは、数学の等価を示す=とまったく異なる働きをします。=の左辺と右辺ではまったく意味が異なります。

▼式の左辺と右辺

=の左辺に変数如果出现したら
…変数が新しく保持する値を=の右辺の式で設定することを意味する。

=の右辺に変数如果出现したら
…変数が今まで保持していた値が参照される。

例 $i = i + 5;$
…変数 i が保持していた値に5を足して、新しい i の値とします。

```
x = 10 / x;
```

…10を変数xが今まで保持していた値で割って、新しいxの値とします。

```
y = ( 20 * y ) - z;
```

…うっ、自分で考えてみなさい、、、

★再帰代入文と演算子

なお、同一の変数に代入する代入文（これを再帰代入文と呼びます）には、次のような省略形が用意されています。優先順位に注意して例を見てください。

+=	例：y += 45;	→	y = y + 45;
-=	例：z -= 5 * x;	→	z = z - 5 * x;
*=	例：w *= x + 15;	→	w = w * (x + 15);
/=	例：r /= y - 45;	→	r = r / (y - 45);
%=	例：p %= p / 2;	→	p = p % (p / 2);

■プログラムの状態遷移

プログラムが進行していくに連れ、変数が保持している値が変わっていくことに注意しましょう（水は絶えず流れていて、しかも元の水にあらず）。

変数が一定の値にあることを「状態（State）」と呼んでいます。この状態が、プログラムの進行にあわせて変わっていくことを「状態遷移（State Transition）」と呼びます。

```
public class Transient {  
    public void main() {  
        int x;  
  
        x = 10;  
        x = x + 11;  
        x = x - 78;  
        x = 32 / 4;  
    }  
}
```



// x の値がどんどん変わっていく

図4-2 プログラムの状態遷移

プログラムがある状態にあるということは、プログラムがもつ変数の状態が一定であるということです。変数の値が状態遷移するということは、プログラムが状態遷移するということと等しいのです。

それでは、次のプログラムは、変数の値はどのように変わっていくのでしょうか？実際に入力して、変数の値の移り変わりを確かめてみてください。

```
public class TransitionTester {  
    public static void main( String a [] ) {  
        int x;  
        x = 34; x += 5;  
        x = x * x;  
        x %= 47; x -= 23;  
        System.out.println( x );  
    }  
}
```

// 変数xの保持する値を表示する

■変数の宣言と有効範囲

変数は、必要になった時点で宣言 (Declaration) して導入することができます。とにかく、値を代入する前には、宣言しておいてください。また、値を参照する前には変数に値を代入しておいてください。

```
int x;  
x = 34;
```

// この行以降は、xが有効
// この行以降は、xを参照可能

```

int y;                // この行以降は、yが有効
y = x * 12;           // この行以降は、yを参照可能

```

前に紹介したように、同じ型の変数は、カンマで区切って、複数同時に宣言することができますから、プログラムの途中で、一つの宣言文で一挙に複数の変数を導入することができます。

```

int x, y, width, height;    // この行以降は、x, y, width, heightが有効

```

変数の有効範囲（Scope）は、宣言してから、その宣言文が含まれているブロックが終了するまでです。ブロックとは、波括弧（braces） `{ }` で囲まれた範囲のことを指します。

```

{ int x;                // xが有効になった
                          // この外側のブロックでは、xだけが有効である
  { int y;              // yが有効になった
                          // この内側のブロックでは xもyも有効である
  }

                          // 内側のブロックが終了するとyが消えてしまった....
}

                          // 外側のブロックが終了すると、xも消えてしまった...

```

この有効範囲の規則を違反して変数を使うと、まともなコンパイラに叱られてしまいます。例えば、次のプログラムでは、メソッドの処理を定義したブロックの中で変数 `independent` を定義していますが、これをブロックの外側で使うことはできません。

```

public class ProgramWithProblems {
    public static void main( String [] args ){
        int independent;
        independent = 5;
    }
    System.out.println( independent );    // independentはここでは無効
}

```

■変数をいくつ使うべきか？

式を用いて様々な計算を表すことができるので、実際に使うべき変数の数はそれほど多くありません。例えば、以下の左右の例は、同じ処理をしていますが、右側は一つの変数だけでパラメータを計算しています。

```

int x, y, z;           int x;
x = 34;                 x = 34;
y = x * 43;             gc.drawString( "Hello", x*43, x*44 );
z = x + y;
gc.drawString( "Hello", y, z );

```

プログラムを書き慣れていない人は、いくつもいくつも余分な変数を使いがちです。必要以外に変数を用意するのは、少し考えものです。しかしながら、下の例のように、明示的に変数が保持している値の内容を区別したいときには、それぞれの情報を別々の変数に分けた保持した方がよいでしょう。

```

int x, y;                // xとyは、それぞれx座標、y座標を示している
x = 60; y = 30;
gc.drawString( "Good Morning", x+20, y + 40 );

```

同一の変数を途中から別の用途に使うこともできます。しかし、変数を何の用途に使っているのかを自分で混乱しないようにしなければいけません。慣れないうちは、避けた方が無難かも知れません。

```

int count;              // 個数を覚えておく変数を用意した
count = 10;             // まずは、Orangeが10個であることを覚えていた
:                       // その後、いろいろありまして、、、

```

```
count = 5;
```

```
// ここからは、Pineappleが5個あることを示すことにした
```

4.4. 式と評価

■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

式 - 式

式 * 式

式 / 式

式 % 式

(式)

足し算

引き算

掛け算

割り算

余り

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

y

x

28

x - 28

(x - 28)

y * (x - 28)

589 + y * (x - 28)

「定数そのもの」だから式

「変数そのもの」だから式

「変数そのもの」だから式

「定数そのもの」だから式

「式 - 式」に合っているので式

「(式)」に合っているので式

「式 * 式」に合っているので式

「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。

```
3 x + 5 y
```

■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。
- ・同じ優先順位の場合は左から右に（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。

$$\begin{array}{r} y = (10 + x) * 5 - 15; \\ \downarrow \\ 10 + 20 \\ \downarrow \\ 30 \quad * \quad 5 \\ \downarrow \\ 150 \quad - \quad 15 \\ \downarrow \\ 135 \end{array}$$

■整数にまつわる特殊な演算

★整数除算と剰余算

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

剰余算は、整数でも実数でも行なうことが可能です。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

■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、

```
gc.drawString( 45, 100, 100 );           // 整数であり、文字列ではない
gc.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。整数や実数を表示するのはどうしたらいいのでしょうか？

■整数と文字列の変換

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、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"
```

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" );
gc.drawString( "体重" + x * 4 + "kg ", 100, 20 );
showStatus( "value of variable x is :" + x );
```

4.5. まとめと課題

■まとめ

★式

代入文やメソッド呼出しのときに式が記述できる

★型

式、変数、定数には、型がついている。
論理型、整数型、実数型、文字型、文字列型などがある。

★変数

宣言文で変数を導入する
代入文による値の代入
式中での変数の値の参照
再帰代入文

★定数

内部表現によって、表される範囲が異なる

★演算子

加法演算子と乗法演算子がある
文字列の足し算は、整数や実数なども自動的に変換してくれて、かなり便利

★式の評価

式の書式に則っていなければならない
優先順位に従って式が評価される
整数除算や剰余算は注意して使う

■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。

4-2.

整数除算と実数の除算の結果を比較しなさい。

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。