メソッド

ここまでには文字列を表示する System.out.print()やキーボードから整数を入力する stdIn.nextInt()などを用いてプログラムを作成してきた。これらはメソッドと呼ばれるプログラムを構成する部品である。メソッドとは Java や C++などのオブジェクト指向プログラミング言語で利用されている概念であり、他の言語での関数やサブルーチンに相当するが、特定のクラスに属しているという特徴がある。

メソッドは自分で定義して利用することもできる。プログラムで繰り返し利用される処理をメソッドとして作成してしまえば、それ以降のプログラムや他のプログラムでそのまま、あるいは一部を修正して利用することが出来るので、再利用性の高いプログラムを作成することができる。また、処理単位でメソッドを作成するため、プログラムの可読性（読みやすさ）が高まる。これらのメリットはプログラムの保守性（管理のしやすさ）に関わるため、大事な要因である。今回はメソッドの作り方（宣言）と使い方（呼び出し）について説明していく。

簡単なメソッドを用いたプログラム

以下のプログラムはメソッドを用いて２つの整数の和を求め、結果を出力するプログラムである。

プログラム例１

```
public class Prog08_01
{
        static int add(int a, int b)    // メソッドの宣言
        {
                int c;
                c = a + b;
                return c;
        }

        public static void main(String[] args)
        {
                int a = 3, b = 4, c;
                c = add(a, b);          // メソッドの呼び出し
                System.out.println("a + b = " + c);
        }
}
```

実行結果

```
a + b = 7
```

このプログラムの場合にどのような順番で実行されるかを下図に示す。

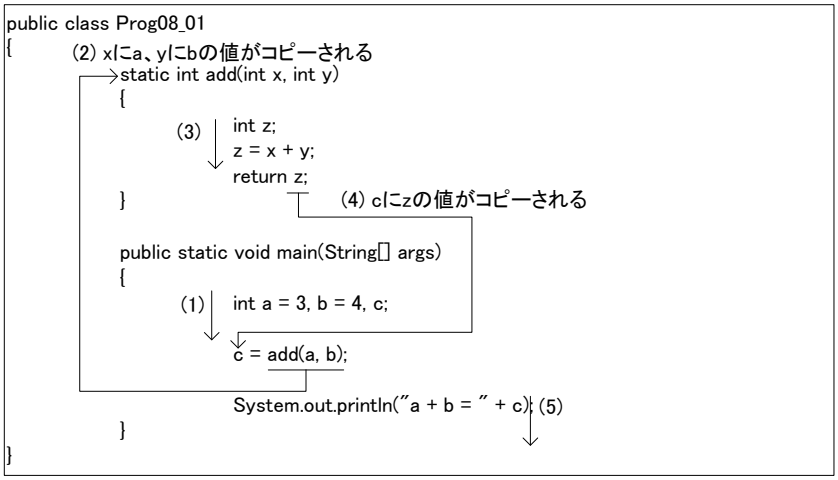


図１　プログラムの流れ

（1）プログラムの開始場所はいつでも main( )メソッドから始まる。順番に実行され、add(a,b)が実行される。これは「メソッドを呼び出す」ともいう。
（2）add( )メソッドに実行が移る。この際、add( )メソッド呼び出し時の a と b の値がそれぞれ add( )メソッドの x と y にそれぞれコピーされる。
（3）add( )メソッドが順番に実行される。
（4）main( )メソッドに処理が戻る。その際、add( )メソッドの z の値が main( )メソッドの変数 c に代入される。
（5）main( )メソッドの残りの処理が実行される。

add(a, b)や int add(int x, int y)ので用いられている a、b、x、y はメソッドに与えるパラメータであり、これらを「引数」という。特に main( )メソッドで add( )メソッドの呼び出しに使っている add(a, b)の a と b は実際に処理に使っている値であることから「実引数」といい、それに対して add( )メソッドの定義に用いている int x と int y は「仮引数」という。また add( )メソッド終了時に return z;と書いているが、これは main( )メソッドに z の値を戻している処理である。このことから、z を「返却値」という。

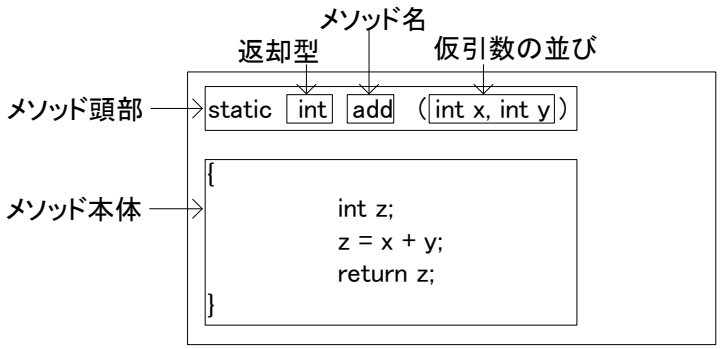
メソッドの宣言

メソッドの宣言は以下の書式に従う。

書式

```
static　返却型　メソッド名（仮引数の型　引数名, …, 仮引数の型 引数名）
{
        メソッド本体
}
```

使用例（プログラム例１の抜粋）



※メソッド名や引数名は自由に付けられる。返却型や引数の型、個数は必要に応じて変更する必要がある。

メソッドは一般的に引数によって値を受け取り、return で返却値を返す。しかし引数や返却値が必要無いメソッドも存在する。この場合、以下の例のように「引数が無い」ならば仮引数の並びは空白とし、「返却値が無い」場合には返却型に「void 型」を使う。

```
static void func()
```

例えば、返却値が無く、引数が int 型で二つ必要な場合には以下のようになる。

```
static void func(int s, int t)
```

もう一つ例を示すと、返却値が int 型であり、引数が無い場合には以下のようになる。

```
static int func()
```

次に return 文について説明を加えていく。return 文はメソッドにおいて、値を返す必要がある時、以下の書式で用いる。

書式

```
return 値; または、return 式;
```

値を返す必要のないときには「return;」と記述できるが、メソッドの最後でリターンする場合は省略するのが通常である。

使用例

```
static void func()
{
        処理
        return;          ⇒       省略する
}
```

ただし、この書き方はメソッドの途中で呼び出し元に処理を戻す場合には便利である。

使用例

```
static void func()
{
        処理
        if(条件式) return;
        処理
}
```

また、メソッド内で return 文を複数書くことも可能である。例えば引数で渡された変数の値の正負を判定し、絶対値を返却値として返すプログラムの場合、以下のようなプログラムが考えられる。

プログラム例２

```
public class Prog08_02
{
        static double abs(double x)
        {
                if(x >= 0) return x;
                else       return -x;
        }

        public static void main(String[] args)
        {
                double a = -3.0;
                System.out.println(a + "の絶対値は" + abs(a) + "です。");
        }
}
```

出力例

```
-3.0 の絶対値は 3.0 です。
```

使用例

いくつかのプログラムを示す。プログラム中の下線部の処理について、メソッドを用いたプログ

ラムに変更する例を示していく。

プログラム例３

```
public class Prog08_03
{
        public static void main(String[] args)
        {
                System.out.println("メソッドの練習");
        }
}
```

変更後

```
public class Prog08_03
{
        static void func()
        {
                System.out.println("メソッドの練習");
        }

        public static void main(String[] args)
        {
                func();
        }
}
```

このプログラムの場合には出力処理をメソッドで記述する。引数と返却値は必要ないため、メソッドの頭部は「static void func()」となる。

プログラム例４

```
public class Prog08_04
{
        public static void main(String[] args)
        {
                int n = 10;
                System.out.println(n);
        }
}
```

変更後

```
public class Prog08_04
{
        static void func(int s)
        {
                System.out.println(s);
        }

        public static void main(String[] args)
        {
                int n = 10;
                func(n);
        }
}
```

このプログラムの場合も出力処理をメソッドで記述する。変数 n を出力に使っているため、この値は引数を用いてメソッドに渡す必要がある。返却値は必要ないため、メソッドの頭部は「static void func(int s)」のようになる。

プログラム例５

```
public class Prog08_05
{
        public static void main(String[] args)
        {
                int a = 5, b = 1, c;
                c = a * b;
                System.out.println(c);
        }
}
```

変更後

```
public class Prog08_05
{
        static int func(int x, int y)
        {
                int z;
                z = x * y;
                return z;
        }

        public static void main(String[] args)
        {
                int a = 5, b = 1, c;
                c = func(a, b);
                System.out.println(c);
        }
}
```

このプログラムの場合では 2 つの変数を用いて、乗算を行いその結果を出力する。出力には変数を用いているので、その処理で得られる値が必要となる。この場合メソッドの引数は整数型で 2 つ、返却値として整数型が必要となる。よって、メソッドの頭部は「static int func(int x, int y)」のようになる。

もう少し具体的な使用例を示す。次のプログラムは 2 倍角の公式 $\sin 2\theta = 2\sin\theta\cos\theta$ において、角度を 0～360 度まで 5 度ずつ変化させて両辺の計算結果を表示させている。Math.sin()、Math.cos()、Math.toRadians ()のメソッドはそれぞれ sin の計算、cos の計算、度からラジアンに変換を行う。Math クラスは String クラスと同様に import 宣言を行わなくても自動的に import される。

プログラム例６

```
public class Prog08_06
{
        static double function1(double x)
        {
                double y;
                y = Math.sin(2.0 * x);
                return y;
        }
```

```
        static double function2(double x)
        {
                double y;
                y = 2.0 * Math.sin(x) * Math.cos(x);
                return y;
        }
        public static void main(String[] args)
        {
                int i;
                double theta, y1, y2;
                for(i = 0; i <= 360; i += 5)
                {
                        theta = Math.toRadians(i);      // 角度をラジアンに変換
                        y1 = function1(theta);
                        y2 = function2(theta);
                        System.out.println(i + "¥t" + y1 + "¥t" + y2);
                }
        }
}
```

数学関数を利用する場合、Math.sin()のようにMath.と付けて書く必要がある。Math.は静的インポート宣言を使うと省略できる。プログラムの最初の部分に「import static java.lang.Math.*;」としてインポートを行うとsin()やcos()のようにメソッド名だけで書くことができる。このようなimport宣言についての詳しい情報は教科書のp.363「第11章　パッケージ」に、静的インポート宣言についてはp.370に紹介されている。

有効範囲

ここまでに変数はメソッドの中で宣言して利用してきたが、メソッドの外で宣言することもできる。以下のプログラムはメソッドの中と外で同じ変数名 x で宣言を行っている。これらはその識別子（名前）の通用する範囲である「有効範囲（スコープ）」が異なる。

プログラム例７

```
import java.util.Scanner;

public class Prog08_07
{
        static int x = 700;                                          クラス有効範囲

        static void printX()
        {
                System.out.println("x = " + x);          // ④
        }

        public static void main(String[] args)
        {
                System.out.println("x = " + x);          // ①

                int x = 800;                                         メソッド有効範囲

                System.out.println("x = " + x);          // ②
                System.out.println("Prog08_07.x = " + Prog08_07.x);      // ③

                printX();
        }
}
```

出力結果

```
x = 700
x = 800
Prog08_07.x = 700
x = 700
```

このプログラムはコメントの①～④の順番にプログラムが実行される。動作の説明を以下に述べる。

① この時点では main メソッド内に x は宣言されていない。そのため、メソッドの外で宣言された「static int x = 700;」によっての 700 が出力される。
② この時点では main メソッド内において、「int x = 800;」と初期化されている。よって 800 が出力される。クラス有効範囲とブロック有効範囲に同一の変数 x が存在する場合、局所変数である「int x = 800;」が優先される。
③ クラス有効範囲で宣言した「static int x = 700;」の値を表示する場合にはクラス名の Prog08_07 を付けて、「Prog08_07.x」として利用する。
④ printX メソッドで出力を行う。メソッド間で値を受け渡す場合には引数を用いる必要があるが、このメソッドでは引数が無い。そのため、printX 内で有効となる変数 x はクラス有効範囲「static int x = 700;」の値である。

演習

プログラム例１～７を作成して、出力を確認しなさい。

課題５

kadai5_1　プログラム例６を参考として、もう１つの２倍角の公式$\cos 2\theta = \cos^2\theta - \sin^2\theta$を確認するプログラムを作成しなさい。
kadai5_2　３つ整数値の最小値を求めて返却するメソッドを作成し、実行結果を表示するプログラムを作成しなさい。
kadai5_3　1 から n までの全整数の和を求めて返却するメソッドを作成し、実行結果を表示するプログラムを作成しなさい。
kadai5_4　a 以上 b 未満の乱数を生成して、その値を返却するメソッドを作成し、実行結果を表示するプログラムを作成しなさい。メソッド内部では乱数を生成する標準ライブラリを呼び出すこと。b の値が a より小さい場合には a と b の値を交換する処理をメソッド内で行うこと。

課題５には続きがある。提出締め切りは後で示す。