6 メソッド

6.1メソッドの概要

メソッドは変数と並んでJavaを学ぶ上では避けてとおれない機能の一つです。

変数がデータなら、メソッドは操作ということができるでしょう。C や C++では関数と呼ばれるものです。

通常プログラミングを行っていると、同じような作業を何度も行わなければいけなくなることがあります。ここで、例として、パケット代を合計した結果を画面上に表示したい場合について考えてみましょう。この作業はこれまでに勉強したプログラムを利用すれば、以下のようになります。

```
int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
int packetPrice = 0;
for(int i = 0; i < 8; i++){
          packetPrice += packetPriceArray[i];
}
System.out.println("今週のパケット代は"+packetPrice+"円です。");
```

しかし、異なる配列の中身を、プログラムの色々なところで合計しなければいけない場合があります。たとえば、一ヶ月のパケット代の合計と、一ヶ月の通話料の合計を表示したい場合などは、それぞれの配列を合計するために、上記のような記述をしなければなりません。

```
//まずはパケット代を計算
int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
int packetPrice = 0;
for(int i = 0; i < 8; i++){
          packetPrice += packetPriceArray[i];
}
System.out.println("今週のパケット代は"+packetPrice+"円です。");

//次に通話料を計算
int[] callPriceArray = {10,10,60,100,0,0,20};
int callPrice = 0;
for(int i = 0; i < 8; i++){
          callPrice += callPriceArray[i];
}

System.out.println("今週の通話料は"+callPrice+"円です。");
```

ちなみに，この結果は，

```
今週のパケット代は508円です。
今週の通話料は200円です。
```

となります.

このように、同じような動作を何度も行わなければいけないことが、あらかじめ分かっていた場合、何度も同じようなことを書くのは面倒くさいですよね。たとえ面倒くさくなくても何度も書き直しているうちに書き損じが発生してしまうか農政だってあります。そ

こで、同じような動作を何度も行う場合は一つにまとめておきたいと思うじゃありませんか。

というわけで、似たような動作を何度も行う場合にメソッドを利用します。

さて、それでは先ほどの作業をメソッドにした例をこちらに示しましょう。

```
public static void main(String[] args) {
        int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
        int packetPrice = addArray(packetPriceArray); //①
        System.out.println("今週のパケット代は"+packetPrice+"円です。");

        int[] callPriceArray = {10,10,60,100,0,0,20};
        int callPrice = addArray(callPriceArray); //④
        System.out.println("今週の通話料は"+callPrice+"円です。");
}

static int addArray(int[] ary){ //②
        int total = 0;
        for(int i = 0; i < ary.length; i++){
                total += ary[i];
        }
        return total; //③
}
```

さて、これを実行してみるとどうなるでしょうか？

```
今週のパケット代は 508 円です。
今週の通話料は 200 円です。
```

ちゃんと、packetPriceArray と callPriceArray の合計を表示していることが分かります。

ここの中で、「addArray」というのがメソッドになります。

プログラムは①の行

```
        int packetPrice = addArray(packetPriceArray); //①
```

の行まできたら、②へ移動します。そして，この{}ブロックの中を実行して③まで来ると，もとの①の行へと戻ります.

```
static int addArray(int[] ary){ //②
        int total = 0;
        for(int i = 0; i < ary.length; i++){
                total += ary[i];
        }
        return total;
}//③
```

同様に④へ移動すると，②へ移動してまた③までを実行してから④へ戻ります.

このように，同じ動作を繰り返したい場合は，メソッドという形でプログラムの処理をまとめてしまうことが出来るのです．まとめられた処理は，その処理が呼ばれるたびに実行されます．つまり、似たような作業を何回も行いたい場合に使うと便利なんですね。

for 文や while 文はまったく同じ作業を何度も同じ場所で行うための制御構文でしたが、メソッドは任意の場所から呼び出して処理を行うことができます。*footnote{正確にはどこからでも、というわけではありませんが・・・}*

6.2メソッド作成

メソッドには、大きく三つの機能があります。それが、**引数、返り値、処理**です。

それぞれどのように記述するのか見ていきましょう。

```
static 返値型 メソッド名(引数1, 引数2){
        処理
        return 返値
}
```

6.2.1 メソッドの宣言

まず、メソッドを記述する場合は

```
static int addArray(int[] ary){
```

のようにメソッドの宣言を記述します。

このとき、static は単なるおまじないだと思って下さい。通常のメソッドを作成するときは static と書く必要はありません。が、今回はこれがないとうまく動作しないので特別に書いています。static がどういう意味なのか、static が必要なときと必要ではないときがどんなときなのかは第 8 章で説明します。

で、static の次に**返値の変数型**を記述します。たとえば、この例だと int 型を指定していますが、この場合このメソッドを呼び出すと int 型の値が返ってくることを示します。返ってくると言うのがちょっとよく分からないかも知れませんが，プログラム上は，

```
int packetPrice = addArray(packetPriceArray);
```

この太字の部分が，返り値によって置き換えられたものとして扱われます.

つまり，今回返り値は 508 ですので，

```
int packetPrice = 508;
```

と書かれた物としてプログラムが動いていくわけです．で，その値がどんな型なのかを宣言するのがこの部分という訳です. 今回は整数 508 に変換されるので, 整数型 int を返り値として宣言します．もしここが double 型であれば実数型が返ってきますし、配列や次章以降で説明するクラス型だった場合も、それぞれの型の変数が返ってくることになります。

その次に記述されているのが**メソッド名**です。今回は配列の合計を求めるメソッドを作成しましたので、addArray という名前を付けています。メソッドも変数などと同じで、独自の名前を付けることができます。

メソッド名の次に来るのが、**引数**です。引数とはメソッドに与える変数であり、この引数の値を変化させるとそれに応じて返値が変化するようなものです。先の例でいくと、「X から y までの合計を出す」の x,y の値が引数となります。

引数は、メソッド名の直後に()で囲んで通常の変数宣言と同じように

```
(変数型名 変数名)
```

と書きます。

ちなみに、引数は返り値と同様，型が決められており、整数型の引数を求めるメソッドには整数型しか渡せませんし、実数型の引数をもつメソッドには実数型のみがわたせます。

先ほどの例ですと、配列を渡しています。

ちなみに、引数はいくつでも作成することができます。先の例では 1 つの配列を引数にしていますが、三つの実数にしたり、一つの整数と二つの実数にする事だってできちゃいます。一つ以上の変数を指定する場合は、”,”で区切ります。例えば，二つの実数型(double 型)の平均値を求めるメソッド average を作りたければ、その宣言は以下のようになるでしょう。

```
        static double average(double x, double y){
```

6.2.2 メソッドの処理

メソッド内でプログラムがどのような処理をするのかは，メソッドの宣言の後に書く{}ブロック内に記述します.

```
        static int addArray(int[] ary){
                int total = 0;
                for(int i = 0; i < ary.length; i++){
                        total += ary[i];
                }
                return total;
        }
```

メソッドの中身の書き方は通常の Java プログラムを書く場合と同じです。ただし、あらかじめ引数として与えられたものを変数として利用することができます。この例ですと、配列 ary は引数で与えられており、変数として利用できます。

ちなみに、ary の値はあとで説明する「メソッドの呼び出し」の時に決定されます。つまり、ary の値はメソッドを書いているときには何が入っているのか検討もつかず、呼び出しをする側で決めることができるわけです。

したがって，メソッドの引数として与えられる変数には何が入ってくるのか，型以外はまったく分からないということを意識してプログラムを書きましょう.

「パケット代が入ってくるはずだから，配列の値は全部正だよな～」

などと勝手に考えてプログラムを書くと，あとで負の数字が入った配列が引数に与えられたときにあわてることになってしまいます.

6.2.3 返り値

メソッドの最後に返り値を決定します。ん？さっきメソッドの宣言をしたときに返り値は決定した気がしますよね？なんでまた決定するんでしょうか？

実は、メソッドの宣言では、返り値の「型」は決定しましたが、具体的な返り値の「値」決めていませんでした。そこで、メソッドの最後に返り値の具体的な値を決定するのです。

それが、

```
                return total;
```

この部分です。

return というのは英語をかじった人ならすぐ分かりますが、「返る」という意味がありますので、まさに値を返すという意味になるのです。

ちなみに、この構文で result の値を返り値にするよ、という意味になります。この場合だと、total の値が返り値となるわけです。total は配列の数値を全部足した値ですから，これによって，このメソッドを呼ぶことで配列の全ての数値を足した値を得ることが出来るようになるわけです.

もしここで、result の代わりに、

```
return ary[0];
```

とすれば、配列の一番最初に格納されている値が返されることになります。また、

```
return 10;
```

と書いたら、どんな引数を書こうがカならず 10 が返ってくる頑固一徹なメソッドが完成します。だったら最初から 10 って書けばいいわけですが.

まあ、実際にプログラムを書くときは引数に応じて返り値が変更されることが多いです。ようするにメソッドというのは、「ある引数に対して特定の計算を行うことによって返り値を計算する」機能であるということができます。

正確に言えば，もっと複雑なことも色々出来るんですが，それについてはクラスについて学んだときに勉強しましょう.

6.2.4 返り値なしのメソッド

ところで、メソッドの中には返り値がないものがあったりします。そのメソッド内で処理が完結してしまう場合などは、返り値をつけません。

ここで，例として配列の中身を画面に表示するプログラムを考えて見ましょう.

```
for(int i = 0; i < ary.length; i++){
        System.out.println(ary[i]);
}
```

しかし，毎回 3 行もプログラムを書くのは面倒なので showArray というメソッドを作ってしまうことにします.

しかし，この処理をメソッドにしても，値を表示してしまえばお役ご免で，とくに値を返す必要性はありません．そのようなメソッドを作る場合は，メソッド宣言の返り値宣言には「返り値はないよ」という意味を示す、**void** と書きます。つまり、配列の中身すべてを表示するメソッドを作りたい場合は、

```
static void showArray(int[] ary){
        for(int i = 0; i < ary.length; i++){
                System.out.println(ary[i]);
        }
}
```

と書くことになります。このメソッドを使う場合は以下のようにします.

```
int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
showArray(packetPriceArray);
```

この実行結果は,

```
100
120
13
0
41
32
100
102
```

というわけで，配列の中身をちゃんと表示していますね.

ちなみに，このように書いた場合，返り値はありませんので，返り値を代入することは出来ません.

```
int value = showArray(packetPriceArray);
```

などと書こうとするとコンパイル時に

```
型の不一致: void から int には変換できません。
```

というエラーメッセージが出てコンパイルに失敗します.

6.3メソッドの呼び出し

6.3.1 単純なメソッドの呼び出し動作

メソッドの作り方がわかったところで、次にメソッドを利用する方に着目してみましょうか。まあ，さっきまでも呼び出しの部分もついでに書いていたので，大体分かってしまっているとは思いますが.

まずメソッドを呼び出すとどのような動きをするか理解するために、以下のようなプログラムを作ってみましょう。

```
public static void main(String[] args) {
        System.out.println("メソッドを呼び出し開始");
        showMessage();
        System.out.println("メソッドを呼び出し終了");
}


static void showMessage(){
        System.out.println("メソッドの中身だよ");
}
```

これを実行してみましょう。

```
メソッドを呼び出し開始
メソッドの中身だよ
メソッドを呼び出し終了
```

となりました。

このように、メソッドの呼び出しをおこなう場合は、呼び出したいメソッド名+（）を記述します。

```
メソッド名();
```

これによって、プログラムはメソッドに移動して、メソッドの中身を実行し、実行し終わったら元の位置に戻ってきてプログラムを再開します。

6.3.2 引数を使ったメソッドの呼び出し

次に、引数を与えるメソッドの場合を見てみましょう。

```
public static void main(String[] args) {
        System.out.println("メソッドを呼び出し開始");
        showMessage(1);
        showMessage(10);
        showMessage(50);
        System.out.println("メソッドを呼び出し終了");
}


static void showMessage(int x){
        System.out.println("引数は"+x+"だよ");
}
```

これを実行してみるとどうなるでしょうか。

```
メソッドを呼び出し開始
引数は 1 だよ
引数は 10 だよ
引数は 50 だよ
メソッドを呼び出し終了
```

メソッドでの表示が、引数によって変化していることが分かると思います。このように、引数を変化させることで、メソッド内での変数 **x** の値が変化していることが確認できました。

このように、メソッドの中身はまったく変えていないにもかかわらず、呼び出すときの引数を変化させることでメソッド内での動作を変化させることができます。

引数を持ったメソッドの呼び出しは，以下のように行います.

```
メソッド名(引数);
```

ここでは引数として直接数字を書いていますが，変数を使うことも出来ます．その場合は，

```
メソッド名(変数名);
```

と書くことになります.

6.3.3 返り値を使ったメソッドの利用

メソッドの利用最後は、返り値を受け取る場合です。

こんなプログラムを作ってみましょう。ここで使うメソッドは最初にメソッドの作成の

説明で使った，配列の中身を合計するメソッドです。

```
public static void main(String[] args) {
        int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
        int packetPrice = addArray(packetPriceArray);
        System.out.println("今週のパケット代は"+packetPrice+"円です。");

        int[] callPriceArray = {10,10,60,100,0,0,20};
        int callPrice = addArray(callPriceArray);
        System.out.println("今週の通話料は"+callPrice+"円です。");
}

static int addArray(int[] ary){
        int total = 0;
        for(int i = 0; i < ary.length; i++){
                total += ary[i];
        }
        return total;
}
```

このメソッドでは、直接メッセージは表示しませんが、返り値として配列の数字を合計した数字を返してくれます。実行結果はこうなります。

```
今週のパケット代は 508 円です。
今週の通話料は 200 円です。
```

このように、ちゃんと packetPriceArray と callPriceArray には配列の中身の合計が帰ってきました.

もう,今後はどんな配列があっても addArray メソッドさえあれば、いつでも配列の数値を合計した値を手に入れることができます。

返り値を受け取るメソッドの書き方は以下のとおりです.

```
受け取る変数名 = メソッド名(引数名);
```

以上、三つの方法でメソッドを呼び出すとプログラムがどう動くかを見てきました。メソッドの働きはなんとなくわかったでしょうか？

メソッドの最大の働きは、

似たような機能を繰り返して使う場合に、一箇所にまとめておくことができる

というものであることを覚えて置いてください。

6.3.4 メソッド呼び出しの構文・まとめ

メソッドを呼び出すときの基本構文は以下のとおりです。

まず、そのメソッド名を書いて、そのあとに()内に引数を書きます。

```
メソッド名(引数, 引数,・・・);
```

複数の引数があるメソッドを呼び出す場合は、”,”で区切ります。そして、このような記述があるとプログラムはかかれている引数をもってメソッド処理を行って、返り値を戻してきます。プログラムは，メソッドの記述の変わりに返り値が記述されているものとしてプログラムを実行していきます.

したがって,代入以外にも代入を行っていますが、

```
                if(addArray(packetPriceArray) > 2500){
                        System.out.println("使いすぎです");
                }
```

というように直接比較を行ったりすることもできます。これで，パケット通信をやりすぎたら怒ってくれるお母さんみたいなプログラムだって作れます．余計なお世話ですが．

6.4オーバーロード

メソッドには，オーバーロードという機能があります．これは，一言で言えば**同じ名前のメソッドを引数を変えて二つ作成する機能**です．

たとえば，ある配列の値の平均値を求めるプログラムを作ったとしましょう．

```
        public static void main(String[] args) {
                int[] packetPriceArray = {100,120,13,0,41,32,100, 102};
                int packetAverage = average(packetPriceArray);
                System.out.println("今週のパケット代の平均は"+packetAverage+"円です。");

                int[] callPriceArray = {10,10,60,100,0,0,20};
                int callAverage = average(callPriceArray);
                System.out.println("今週の通話料の平均は"+callAverage+"円です。");

                int[] priceArray = {packetAverage, callAverage};


        }

        static int average(int[] ary){
                int total = 0;
                for(int i = 0; i < ary.length; i++){
                        total += ary[i];
                }
                return total/ary.length;
        }
```

合計を求めるプログラムとほとんど一緒ですが，求めているものが平均であるところだけがちょっと違います．

ちなみに，実行結果は，

```
今週のパケット代の平均は 63 円です。
今週の通話料の平均は 28 円です。
```

となります．

パケット代のほうが平均すると多く使っていることが一目瞭然です．

さて，ここで一週間のパケット代と通話料の平均も出してみたいと思ったとしましょう．つまり，63 円と 28 円の平均ですね．これを計算するメソッドはどう書けばよいでしょうか？

手の一つとして，これらの値を配列にして，先ほど作った average メソッドに代入するという手もあります．

その場合こうなります.

```
int[] priceArray = {packetAverage, callAverage};
int priceAverage = average(priceArray);
System.out.println("今週のパケット代と通話料の平均は"+priceAverage+"円です。");
```

ちょっとテクニカルなことをやっていますが，この結果は・・・

```
今週のパケット代と通話料の平均は 45 円です。
```

というわけで，無事平均値が出てきました．本当にあっているか疑う方は計算機で計算してみてください．ちなみに，小数点以下は切り捨てられています.

しかし，いちいち配列に直すのはちょっと面倒ですよね．できればもっと簡単に二つの整数の平均を求めたいものです．そこで，二つの整数の平均を求めるメソッドを作ってしまいましょう.

メソッドはこんな感じになるでしょう.

```
static int メソッド名(int x, int y){
        return (x+y)/2;
}
```

単純に二つの整数型引数を足して 2 で割っているだけです．このくらいメソッドにしなくても簡単に書けそうですが，そんな細かいことは気にしない方向でお願いします.

さて，ここで問題がメソッド名になります．このメソッド，どんなメソッド名にすればよいでしょうか。

できることなら平均を求めるメソッドなので, average という名前をつけたいところです. しかしながら．すでに average という名前のメソッドは存在しています．普通に考えると同じ名前のメソッドを作ると、どっちだか分からなくなりそうですが・・・

```
static int average(int[] ary){//①
        int total = 0;
        for(int i = 0; i < ary.length; i++){
                total += ary[i];
        }
        return total/ary.length;
}

static int average(int x, int y){//②
        return (x+y)/2;
}
```

実はこのように，二つの average メソッドを同時に作ることが可能です。このように同じ名前の異なるメソッドが存在することを**メソッドのオーバーロード**といいます。

メソッドをオーバーロードする場合の注意点はタダ一つ。**引数を変えておく**ことだけです。Java は引数を見て、どちらのメソッドを呼ぶべきか自動的に判断してくれるのです。

この例ですと、配列一つを引数として average メソッドを呼んだ場合は、①の average メソッド(配列用 average メソッド)が呼ばれ、整数二つを引数として average メソッドを呼んだ場合は②の average メソッド(整数用 average メソッド)が呼ばれます。

Java 君は，たとえメソッド名が同じでも引数が異なれば異なるメソッドとして判断してくれるのです．結構賢いですね．

6.5間違えやすいメソッド

6.5.1 引数の型が同じなオーバーロード

さて，メソッド名はオーバーロードすることで，同じメソッド名で異なる引数を持つものを作ることが出来ました．

でも，いくら引数が異なれば同じ名前で作れると行っても，こんな二つのメソッドを作ることはできません。

```
        static void showMessage(int x){
                System.out.println("x の値は"+x+"だよ");
        }

        static void showMessage(int a){
                System.out.println("a の値は"+a+"だよ");
        }
```

この二つのメソッドも引数が x と a で異なるように見えますが、**引数の型は二つとも int 型**です。そのため、呼び出そうとして、

```
        showMessage(10);
```

と書いたときに、どちらのメソッドを呼び出せばよいのかは分かりませんからね。人間に分からないのですから，Java 君にだって分かるわけがありません．

したがって，このようなメソッドを二つ作ろうとすると，

```
型 FailOverLoad にメソッド showMessage(int) が重複しています。
```

というメッセージが出て，コンパイルすることができません．

一方で，引数の型が違えば，呼び出すときに与えられた型から間違いなくどちらかのメソッドだと決めることが出来ます．そのため，

```
        static void showMessage(int x){
                System.out.println("x の値は"+x+"だよ");
        }
        static void showMessage(double a){
                System.out.println("a の値は"+a+"だよ");
        }
```

という二つのメソッドは作ることが出来ます．これはすでに勉強しましたね．

というわけで、同じ名前のメソッドを作りたい場合は、必ず**引数の型を変える**ようにしましょう。

6.5.2 引数の値を変化させたら？

ある変数をメソッドに引数として渡した場合，その引数を別の値にしたときもとの変数はどうなるでしょうか？

次のようなプログラムを見てみましょう.

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

static void square(int x){
        x = x*x;
}
```

このプログラムでは，引数を二乗するメソッド square を作成しています．これで，x の値は 10×10 で 100 になっているはずですよね.

念のため結果を見てみましょうか.

```
10
```

あれ？二乗されていません．どうなっているのでしょうか？

実は，square で使われている x と main で使われている x はまったく別物なのです．同じ 10 という値を指しているので混乱してしまいそうですが・・・

二つの x は最初同じ 10 という値を指しています,

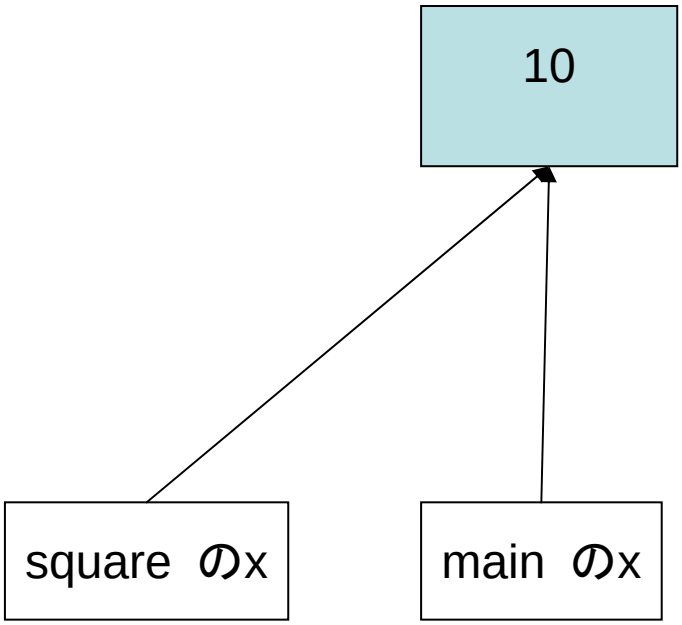

図 二つの x が指すもの

しかしながら，square メソッドの x には x ×x，つまり 100 が代入されます．代入されるということは，指すものが変化するということになりますので，square の x は 100 を指すように変更されます.

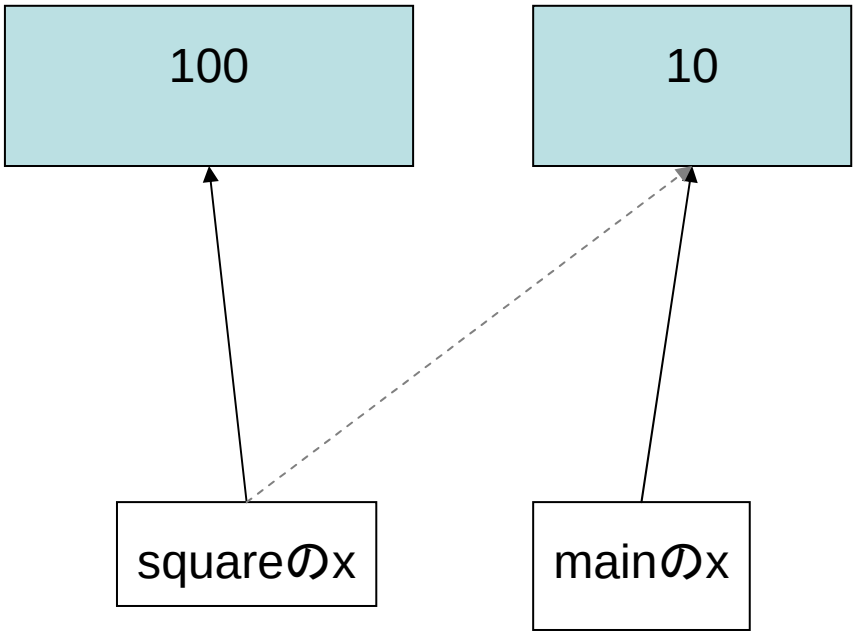

図 square の x が指すものが変化

しかし，このとき main の x が指すものは変化していません.

一見同じ x という変数なので，片方を変更すればもう片方も変更されるような気がします．しかし，実際はそうではないわけです.

square メソッド内の x は確かに, x を二乗した値 100 をさすようになりましたが，main の中の x は何も変化しませんので，10 のままとなります．これは結構間違えやすいミスですので，注意しましょう.

メソッド内で引数の変数に別の値を代入しても，もとの値は変化しません.

もし，x に二乗した値を格納したければ，以下のようにします.

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

        static int square(int x){
                return x*x;
        }
```

ここでは,x を二乗したものをメソッドの返り値として設定しています.

```
square(x);
```

は x を二乗した値，100 であるとプログラムが考えます.

つまり，

```
                x = 100;
```

と書いてあると判断するわけです.

そのため，x の値は 100 に修正されるわけですね.

結果,

```
100
```

と求めている答えを得ることができました.

メソッドの引数を変更する話はちょっと複雑なので，気をつけてください.

6.5.3 配列の場合は中身が変更されます

先ほどは，引数の値を変化させてももとの値は変化しないというお話をしました．ところが，配列の要素を変更した場合，元の配列にも影響を与えてしまうのです.

ここで，ある日のパケット代を記録していた配列に対して初日分だけパケット代が 0 円になるサービスが適用されたとしましょう.

さて，そんな素敵なサービスを実現するメソッドを作成してみます.

```
public static void main(String[] args) {
        int[] packetPriceArray = {100,120,13,5,41,32,100};
        service(packetPriceArray);
        showArray(packetPriceArray);
}

static void service(int[] ary){
        ary[0] = 0;
}

static void showArray(int[] ary){
        for(int i = 0; i < ary.length; i++){
                System.out.println(ary[i]);
        }
}
```

さて，初心者さんはついこんなプログラムを書いてしまうんじゃないでしょうか．しかし，先ほど学んだように，メソッドに引数として渡した変数は代入してももとの変数に影響を与えませんでしたよね．我々はすでに初心者を脱していますから，こんなプログラムは動かないことが分かっています．間抜けなプログラムを書いてしまった初心者さんがあわてふためくところをニヤニヤしながら眺めてあげましょうか.

```
0     ←衝撃的な結果
120
13
5
41
32
100
```

あ，あれ！？最初 100 円使っていた初日のパケット代金が 0 円に変更されています．変更されないだろうと思っていた配列の値が変更されています．初日はタダキャンペーン,

見事大成功です.

さて，なぜこんな事が起きたのでしょうか？実は Java には**メソッド内で実体そのものを変更すると，メソッドの外の実体も変更される**という決まりがあるのです．これはまた分かりづらいのですが，配列を例にとって言えば，

- 配列そのものを別の配列に変更したら，メソッドの外の配列は変更されない.
- 配列の中身を入れ替えた場合は，メソッドの外の配列も変更される.

ということになります.

これを理解するために，次の図を見てください.

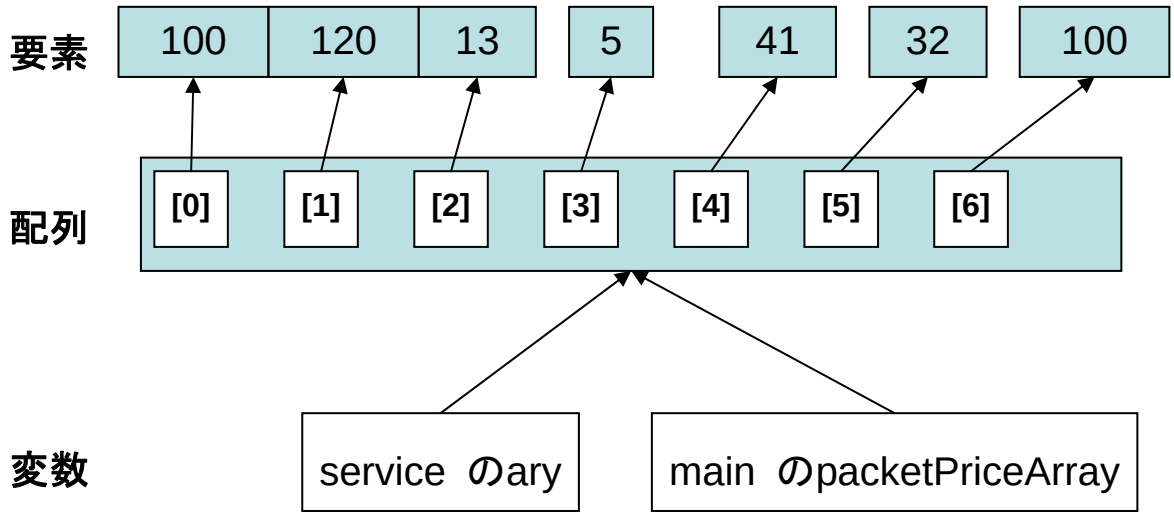

図 配列と要素が指す物

この図で，main の packetPriceArray と service の ary は同じ配列を指します．これは，int 型を渡したときと同じですね．ただし，違うのは，配列は配列自身がいくつかの変数を内部に持っているという点です．そして，それらの変数は，やっぱり別のところに実体を持っています.

次に，ary[0]の値を変化させています．これによって，配列の中身が次の図のように変化します.

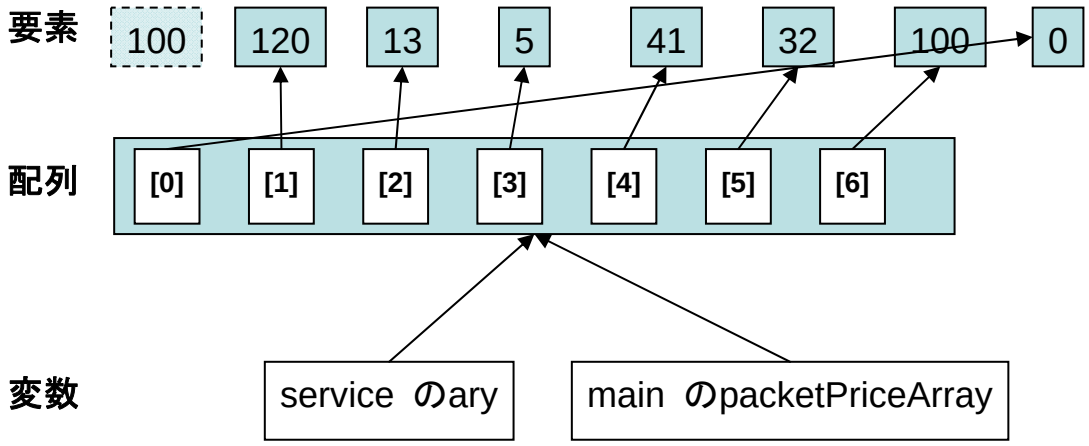

図 ary「0」が指す内容が変化する

これまで，100 を指していた[0]が 0 を指すように変更されました．このように，配列の最初の要素，**[0]が指す内容だけ**が別の値を指すようになります．つまり，配列の中身が変更されたのです．このとき，packetPriceArray も同じ配列を指していますので，**ary[0]と同じ，packetPriceArray[0]も変化する**ことになるのです．

では，配列全体を一気に入れ替えてしまうと，main の packetPriceArray は変化するでしょうか？

ここで，例としてあげるのは一週間分のパケット代すべてのただにしてくれる超素敵なサービスです．先ほどの成功に味を占めたわれわれは，これによってパケット代をまったく払わずにすむ画期的サービスを受けることを決意します．プログラム的には，新しく全部 0 の配列を作って代入しています．

```
        static void superService(int[] ary){
                ary = new int[]{0,0,0,0,0,0,0};
        }
```

さて，この結果はというと・・・

```
100
120
13
5
41
32
100
```

残念，これは認められませんでした．パケット代 0 円計画は大失敗です．

なぜこうなるのかはちょっと考えてみてください．ポイントは，配列そのものを代入してしまっているという点ですね．

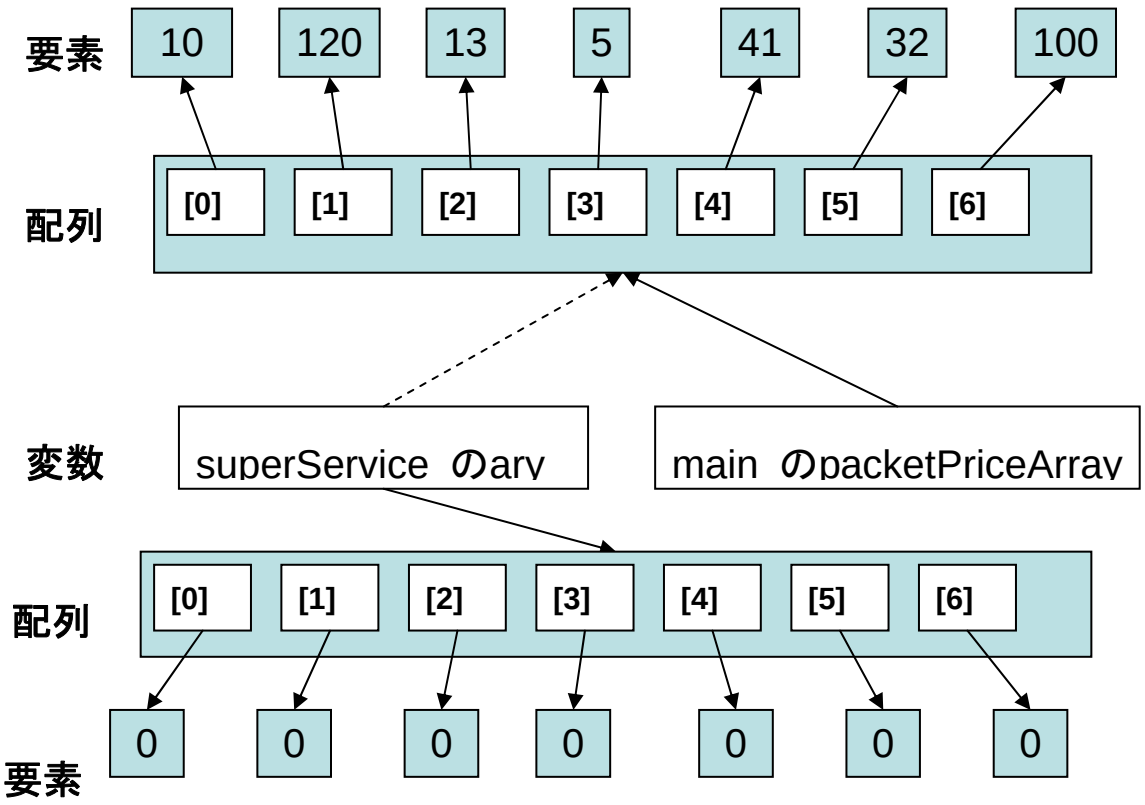

図 superService の ary は何を指すのか？

さて，以上メソッドにおいて引数が元の変数に影響を与えるかどうかを確認して来ました．ちょっと難しいかもしれませんが，覚えておくべきことは以下の 2 点です．

1. **代入を行ってももとの変数には影響を与えない．**
2. **配列で，要素を入れ替えるとと元の変数にも影響を与える．**

ちなみに，次の章から紹介するクラスを考慮に入れると，

3. **クラスのメンバ変数を変更すると元の変数にも影響を与える**

という条件が入りますが，今はあまり気にしないでおきましょう．

プリミティブ型(int, double など)の場合，代入によってしか値を変えることできません．したがって，メソッドのルール 1 によってもとの変数に影響を与えるような変更を行うことが出来ません．つまり，int や double などを引数とした場合は，引数として与えた側の変数が変更されることは決してないということができます．

とりあえず，現時点では，この点だけ覚えておいてください．

- プリミティブ型は，メソッド内で変更しても何も起きない
- 配列は，メソッド内で要素変更すると元の配列も変化する

では，次章からはいよいよ Java の真髄であるクラスについて説明して行きたいと思います.