3-3.メソッド

提出する 評定を受ける

■ メソッド

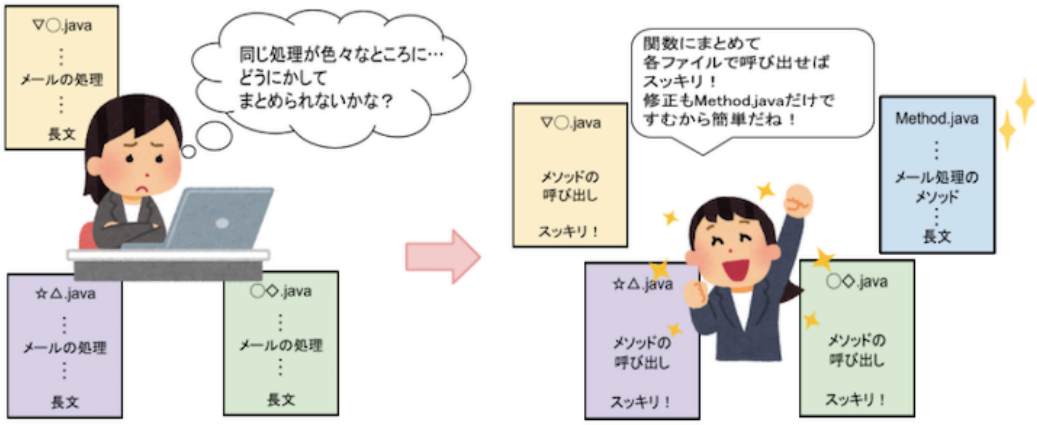
はじめに

コンストラクタ に引き続いて、メソッド、引数、戻り値に関して、さらに理解を深めましょう。
少しづつJavaの解像度をあげて行きましょう！

Step1 : メソッド

メソッド とは、幾つかの **処理をまとめて入れておくもの** です。
たとえば「Aさんにメールを送る」という処理が記述されている **メソッド**があったとします。
それを100ケ所で使う場合は、同じコードを100回記述するのではなく、 **メソッド** を呼び出すだけで済みます。

仮に「Aさんにメールを送る」という処理を「Aさん、Bさん、Cさんにメールを送る」と修正する必要があったとしても
メソッド の処理を変えれば、100箇所に反映されます。コーディングする上でも基本的な考え方になります。



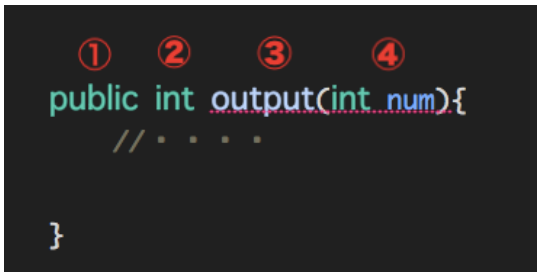
メソッドを使うことで、以下のようなメリットがあります。

- 処理を再利用できるので効率がいい
- エラーコードが少なくなる
- 誰でも同じものとして使える
- コードが読みやすくなる
- 変更が容易になる

メソッドはJavaの基本的な処理となりますので、使い方についてはしっかりと理解しておきましょう。

メソッドは以下のようなものです。

?



解説① アクセス修飾子（省略可）

忘れた方は以前の章を見直してみましょう。

解説② 戻り値の型

これは、このメソッドを実行したらどの型の値を返すのか（「戻り値 や 返り値」といいます。）を指定します。
詳しくはもう少し進んだところにありますが、そんなことを書いてるんだと思っておいてください。

解説③ メソッド名

これが **メソッド** です。
`output();` と記述するだけで、`output` 内の処理が実行されます。
メソッド名は、処理のタイトルのような役割をしていて、同じ処理をする場所に記述していくことで、
なんども同じコードを書かなくて済むということになります。

解説④ 丸括弧内に引数の型

このメソッドを実行する際に一緒に渡される値のことです。
渡される値のことを引数といいます。
（※引数については、Step2で詳しく解説します

以上の①、②、③、④の順で記述し、 ブロック`{ }`で囲むようにして実際の処理を記述します。
メソッド名はオリジナルの名前を記述することが可能ですが、
最初の一文字は半角英小文字で記述するという慣習 があります。

Step2：引数とは

メソッドに何か処理を依頼する際、「このデータを使って処理をしてほしい」ということがあります。
この時メソッドへ受け渡す値を **引数** といいます。
引数は一つだけでなく、 **複数指定することができ** 、複数指定する際は `,（カンマ）` で区切ります。

例題（メソッド＋引数）

例として、底辺（ `base` ）が `10` 、高さ（ `height` ）が `5` の三角形の面積の合計を求め、
コンソールへ出力するメソッドを作成してみましょう。

Study.java

```
public class Study {

    /**
     * 三角形の面積を求めて出力するメソッド
     *
     * @param param 底辺
     * @param height 高さ
     */
    public void printTriangleArea(int param, int height) {
        int area = (param * height) / 2;
        System.out.println(area);
    }
}
```

Main.java

```
public class Main {

    // mainメソッド
    public static void main(String[] args) {

        // 底辺(base)と高さ(height)の変数を指定
        int base = 10;
        int height = 5;

        // Studyクラスのインスタンスを生成して、Studyクラス型の変数studyに保持する
        Study study = new Study();

        // 生成したインスタンスから、StudyクラスのprintTriangleAreaメソッドを呼び出す
        study.printTriangleAreaArea(base, height);

    }
}
```

ポイント①

Study.javaの `printTriangleArea` メソッドは `int param, int height` が引数になってます。
つまり、 `printTriangleArea` メソッドは、 **int型** を **2つ** 受け取れることになります。

Main.javaでは、Study.javaの `printTriangleArea` メソッドに値を渡す必要があります。
その引数は `base` と `height` となります。

※「インスタンス」というワードは次章で解説します。

ポイント②

Study.javaの `printTriangleArea` メソッドでは、変数名が、`param , height` となっていますが、
Main.javaでは、`study.printTriangleArea(base, height)` となってます。
渡す変数と受け取る変数の名前が違っていますが、これは問題ありません。

大切なのは、メソッドの **型** と、その **順番** です。
`printTriangleArea` メソッドは **int型** を **2つ** 受け取れるメソッドなので、
その順番の通りに、変数の中身（今回の場合 **int型** ）が受け渡されるため問題ないです。

ただし、メソッド内と呼び出し側とで著しく異なったり、意味がかけ離れたりするような命名は避けましょう。
（メソッド名と引数はそのメソッドの処理とマッチするよう作成しましょう！

次は、 **引数を使用しないメソッド** のサンプルコードの例題を記述します。

※ただしコンストラクタは引数を使用しています。

例題（メソッド＋引数なし）

```
public class Main {

    // Mainクラスのフィールド変数
    private int base;
    private int height;

    // コンストラクタ(引数あり)
    public Main(int b, int h) {

        // 底辺(b)と高さ(h)の引数をフィールド変数に代入
        base = b;
        height = h;
    }

    // mainメソッド
    public static void main(String[] args) {

        // Mainクラスのインスタンスを生成して、Mainクラス型の変数myselfに保持する
        Main myself = new Main(10, 5);

        // 引数なしのメソッドを呼び出す
        myself.printTriangleArea();
    }

    // 三角形の面積を出力するメソッド
    public void printTriangleArea() {
        int area = 0;
        area = (base * height) / 2;
        System.out.print(area);
    }
}
```

以上のようにメソッドやコンストラクタで指定できる引数の数は0以上となります。
上限は特にありませんが、あまり多くても見づらいため、多くても4つを目途に指定するようにしましょう。

Step3 : 戻り値とは

メソッドには、メソッドの呼び出し元にメソッドからの特定の情報を返すという仕組みもあります。
そのメソッドから返される特定の情報のことを **戻り値 or 返り値（ `return 値` の部分）** といいます。

「何かをしたら、何かしらのレスポンスがある」といったイメージになります。

【引数と戻り値のイメージ】

ここからはより具体的にイメージしていきましょう。

> **例）電卓**

電卓は比較的イメージしやすい例になります！

> **【手順】**
> **1. 引数:** `1 + 1`
> 引数として必要な情報を入力します
> ↓
> **2. 実行「=（イコールボタン）を押下」**
> 計算してもらうために、メソッドの呼び出しを行います。
> ↓
> **3. 結果表示:** `2`
> 電卓が `1+1` を計算して、結果を表示してくれます。

上記の流れの

- `1+1` が引数
- `2` が返り値

のようになります。

> **例）自動販売機**

電卓同様、自動販売機もよい例になります！

【手順】
1. 引数: `150円`
↓
2. 実行「欲しい飲み物のボタンを押下」
↓
3. 結果出力: コーラ

こちらも比較的イメージしやすいのではないかと思います。

> **例）Googleの検索フォーム**

今後みなさんが触れていく作業としては一番近いイメージでしょうか。

メソッドってどう書くんだろう？
と疑問に思いGoogleの検索フォームで調べてみるシチュエーションとしましょう。

【手順】
1. 引数: `Java メソッド 書き方`
↓
2. 実行「検索ボタンを押下」
↓
3. 結果出力: たくさん！

「2件以上の結果が返却された＝記事情報が配列として返却された」 と解釈してみると、
`public 記事情報[] search(検索文字列) { }` といった定義のされ方をしているのでは？
とも想像することができそうですね！

戻り値の型 = return文に指定する処理結果の型

「戻り値」がある場合は、 **処理の結果を返す必要がある** ので、
`return文` を記述して処理結果を呼び出し元へ返すようにします。
その時、 **戻り値の型** と **return文に指定する処理結果の型** が合ってなければなりません。

また、メソッドの処理結果を呼び出し元へ返す必要がない場合は、
戻り値の型を「 `void` 」 と記述して、戻り値なしのメソッドにします。
`void`とは **「このメソッドの戻り値は返しません（ありません）」** という意味のキーワードになります。

戻り値 あり の場合

```
// 引数に渡された x と y の値を加算し、加算結果を呼び出し元へ返すメソッド
public int plus(int x, int y) {
    return x + y;
}
```

戻り値 なし の場合

```
// 引数「result」の内容を画面に表示するメソッド（戻り値なし）
public void print(int result) {
    System.out.println(result);
}
```

以下は戻り値ありと、戻り値なしのメソッドをそれぞれ使用した場合のサンプルコードです。

例題

```
public class Calc {

    public static void main(String[] args) {

        // plusメソッドの足し算に使用する変数と値を設定する
        int x = 10;
        int y = 5;

        // Calcクラスのインスタンスcalcを作成
        Calc calc = new Calc();

        // plusメソッドに上記で設定したxとyを渡して足し算してもらい、変数resultに格納する
        int result = calc.plus(x, y);

        // 足し算の結果を(print)メソッドに渡して表示してもらう
        calc.print(result);

    }

    /** 【戻り値あり】plusメソッド
     *  渡された引数を元に足し算をして結果を返すメソッド
     */
    public int plus(int x, int y) {
        // 渡された引数のxとyを足し算した結果を返す
        return x + y;
    }

    /** 【戻り値なし】printメソッド
     *  渡された引数をそのまま出力（表示）するメソッド
     */
    public void print(int result) {
        System.out.println(result);
    }
}
```

解説

１：まずは足し算に使用する変数とその値をそれぞれ設定してあげます。
２：１で用意した変数を`plus`メソッドに渡して足し算をしてもらい、その結果「`15`」を返してもらって変数「`result`」に格納します。
３：２の足し算の結果「`result`」を`print`メソッドに渡して「`15`」を表示してもらいます。

メソッドには、戻り値があって引数がないパターンや、
戻り値がなくて引数があるパターンなど、自由自在に作成することが可能です。
プログラミングの用途に合わせて、自分自身でどのパターンのメソッドにするか決めて作成します。

補足

> **一般的な修飾子の考え方**

フィールド変数は `private`で記述し、
メソッドは `public` で記述することがほとんどです。

例としては、以下のようなイメージを持つといいでしょう。

ある高校のクラスが10クラスあって、文化祭の出し物を考えているとします。
基本的には **各クラスで考えた出し物はそのクラスのもの** です。
3組で決定した文化祭の出し物（アイデア）は、3組だけで使用するから `private` にする。

でも、1組と3組がやりたい出し物が被ってどうしよう？ってなった場合は、
きっと相談して「 **共同してやる** 」って発想が出て来るはずです。
そうしたら、 共同で使用できるように `public`（or 共通の親クラスを作って `protected` でも可）」にする。
というような感じでしょうか。

なので、フィールド変数は、 **そのクラスだけ** で使用することが多いので、`private` とし、
メソッドは、 **いろんなクラス** で使用することが多いので `public` で記述するのです。

仮に合同の出し物が「お化け屋敷」だったとしましょう。
その場合、1組と3組で作業分担（1組は内装、3組は衣装作成とか）もするでしょうから、
`private` なものとしてそれぞれの組専用の作業（メソッド）があるとも推測できますね。

【public】

- （お化け屋敷の）構成を考える（`public void makeSomeIdea()`）

【private】

- 内装: `private void makeLayout()`
- 衣装作成: `private void makeCostume()`

このようにイメージをふくらませることで、
適用すべき修飾子が理解できてくると思います。

課題

インポートした3-3のフォルダの中にプロジェクトがありますので、
指示通りにコーディングして、ファイルを提出して下さい。

評定概要

学生から秘匿	No
参加者	75
提出	51
要評定	1