Javaプログラミング実習

12. メソッド

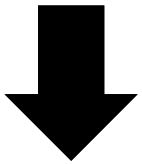
株式会社ジードライブ

今回学ぶこと

- メソッドの概念
- メソッドの定義と利用

メソッドの分割

- プログラムを書き進めていくと、徐々にとmainメソッド内の記述が冗長になってきてしまう



- 幾つかのメソッドに分割することで、mainメソッド内の記述を見やすくすることができる

```
public static void main(String[] args) {
  double h = 172.4; // 身長（cm）
  double w = 54.7;  // 体重（kg）

  h = h / 100; // 身長をメートルに変換
  double bmi = w / (h * h); // BMI値を算出

  double sw = h * h * 22; // 標準体重
  double diff; // 標準体重との差
  String msg; // BMI値に応じたメッセージ

  if (bmi < 18.5) {
    diff = sw - w;
     msg = "体重を" + diff + "kgほど増やしましょう";
  }
  else if(bmi < 25){
    msg = "標準的な体重です";
  }
  else {
    diff = w - sw;
    msg = "体重を" + diff + "kgほど減らしましょう";
  }

  System.out.println(msg);
}
```

メソッドの分割

- 幾つかのメソッドに分割し、mainメソッドから利用することができる

mainメソッド

```
public static void main(String[] args) {
    double h = 172.4; // 身長（cm）
    double w = 54.7;  // 体重（kg）

    h = h / 100; // 身長をメートルに変換
    String msg = getBmiMessage(h, w);

    System.out.println(msg);
}
```

利用

身長・体重を元にメッセージを生成するメソッド

```
public static String getBmiMessage(double h, double w) {
    double bmi = getBmi(h, w);
    double sw = getStandardWeight(h);
    double diff;
    String msg;
    if (bmi < 18.5) {
        diff = sw - w;
        msg = "体重を" + diff + "kgほど増やしましょう";
    }
    else if (bmi < 25){
        msg = "標準的な体重です";
    }
    else {
        diff = w - sw;
        msg = "体重を" + diff + "kgほど減らしましょう";
    }
    return msg;
}
```

利用

BMI値を算出するメソッド

```
public static double getBmi(double h, double w) {
    return w / (h * h);
}
```

標準体重を算出するメソッド

```
public static double getStandardWeight(double h) {
    return h * h * 22;
}
```

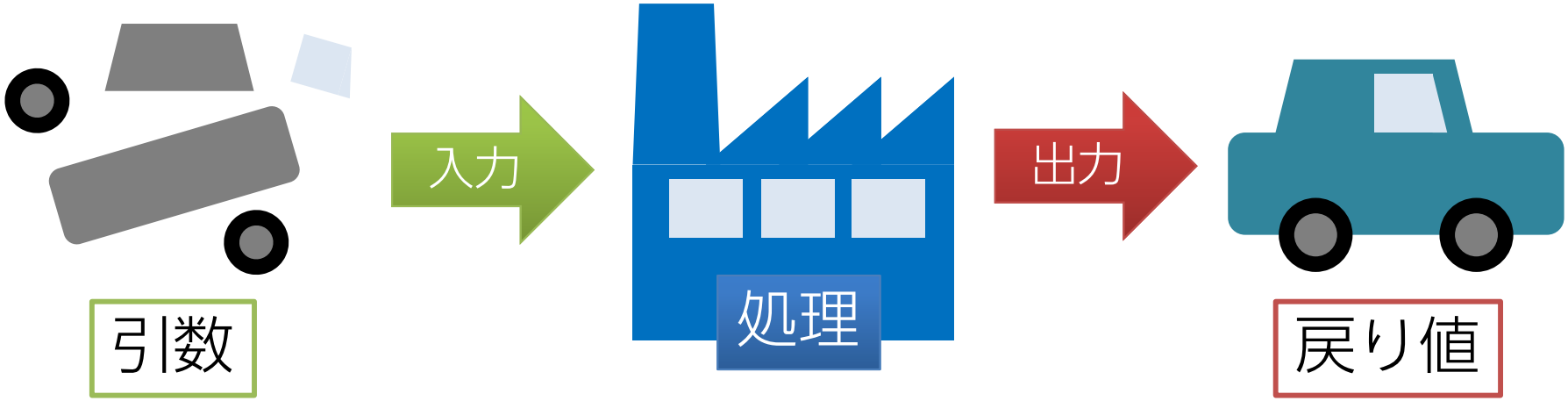
練習

- 練習12-1

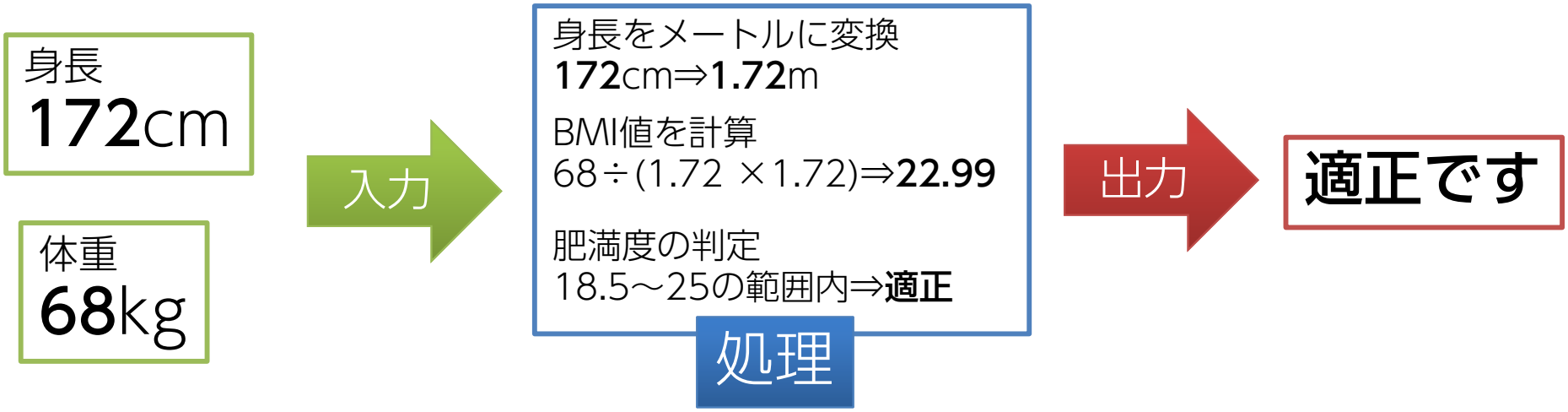
メソッドとは

- いくつかのステートメントで実現される一連の処理に名前を付けて呼び出せるようにしたもの
  - 平均値の計算、データの読み出し、など
- メソッドは**入力機能、処理機能、出力機能**から構成される
  - メソッドに対する入力を引数(ひきすう)(パラメータ)と呼ぶ
  - メソッドからの出力を戻り値(返り値)と呼ぶ

メソッドのイメージ



例：肥満度を調べるメソッド



メソッドの定義と利用

- メソッドは「定義」と「利用」で成立する
  - どういう処理なのか、定義しなければ利用できない
  - 定義をしても、利用しなければ意味がない

```
public class BMI {
  public static void main(String[] args) {
    // メソッドの利用
    double myBMI = calcBMI(1.71, 65.4);
    System.out.println(myBMI);
  }

  // メソッドの定義(メインメソッドの外側に記述)
  private static double calcBMI(double height, double weight) {
    double bmi = weight / (height * height);
    return bmi;
  }
}
```

メソッド定義の記述は、メソッドの利用よりも前に記述されていても、後ろに記述されていても、どちらでも問題ない

メソッドの定義方法

書式

```
private static 戻り値の型 メソッド名 (型 引数名, 型 引数名, … ) {

  // 引数を使った一連の処理

  return 戻り値;
}
```

- 複数の引数がある場合は、カンマで区切る
  - 引数の無いメソッド定義も可能
- 戻り値が無い場合、戻り値の型は `void` と指定する
  - `return`文は省略できる。省略しない場合は `return;` と記述する

※ `private`や`static`については別の機会に学習する

メソッドの命名について

- 命名規則については変数名の規則と同じ
- 一般的な命名方法 (強制ではない)
  - 名前の先頭は小文字にする
  - 複数の単語から成る名前の場合は各単語の先頭を大文字にする (例：`getMemberName()`)
    - この命名法をキャメルケースと呼ぶ
  - 何かを行うメソッドの場合は動詞から始まる名前をつける (例：`printAverage()`)
  - boolean型を返すメソッドの場合「`isValid()`」や「`hasData()`」などの名前を付ける

メソッド定義の例

- 引数／戻り値のあるメソッド

```
public static void main(String[] args) {
  // メソッドの利用
  double myBmi = calcBmi(1.71, 65.4);
  System.out.println(myBmi);
}

// BMI値を算出するメソッドの定義
// 引数：身長と体重　戻り値：BMI値
private static double calcBmi(double height, double weight){
  double bmi = weight / (height * height);
  return bmi;
}
```

表示結果

```
22.365856160870017
```

メソッド定義の例

- 戻り値のないメソッド

```
public static void main(String[] args) {
  // メソッドの利用
  sayHello("山田");
  sayHello("大谷");
}

// 名前に挨拶を付けて表示させるメソッドの定義
// 引数：名前　戻り値なし
private static void sayHello(String name) {
  System.out.println(name + "さん、こんにちは！");
}
```

表示結果

```
山田さん、こんにちは！
大谷さん、こんにちは！
```

メソッド定義の例

- 引数／戻り値のないメソッド

```
public static void main(String[] args) {
  // メソッドの利用
  sayHi();
  sayHi();
}

// ランダムであいさつを表示させるメソッドの定義
private static void sayHi() {
  String[] message = {"やあ", "Hi", "Ciao", "Hola"};
  int randomNumber = (int) (Math.random() * message.length);
  System.out.println(message[randomNumber]);
}
```

表示結果の例

```
Hi
Hola
```

メソッドの利用方法

- 基本の利用方法：引数はリテラルだけでなく、変数・定数・メソッドでもよい

```
メソッド名(第1引数，第2引数，…);
```

- 引数がない場合：引数がなくても( )は必要

```
メソッド名();
```

- 戻り値を変数で受け取る場合

```
変数 = メソッド名(引数，引数，…);
```

- メソッドの引数としてメソッドを使う場合

```
メソッドA(メソッドB(引数));
```

練習

- 練習12-2

戻り値について

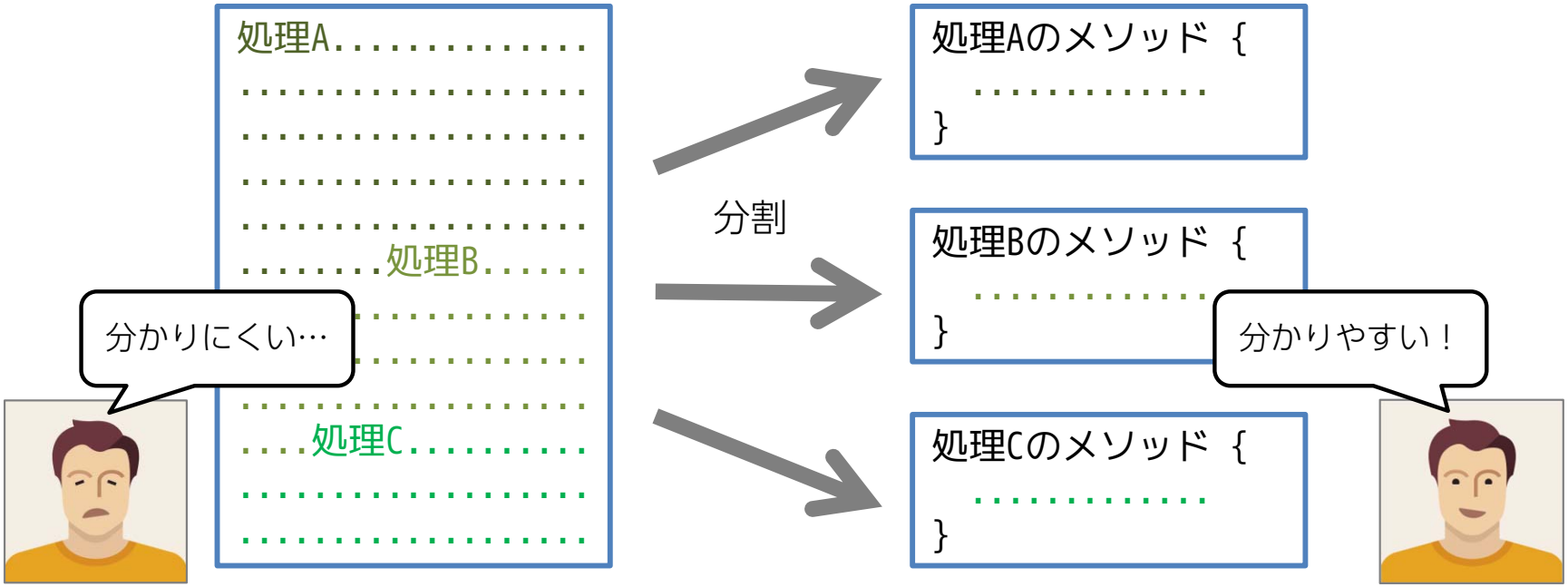
- 複数のオブジェクトを return することはできない
  - 複数のデータを一つの配列としてreturnすることは可能
  - 1つのメソッド内にreturn文が複数存在することは問題ない
  - return後の処理は実行されない

```
// 時間に応じて、日本語と英語のあいさつ文を返すメソッド
private static String[] getGreeting(int time) {
  if(time < 12) return new String[] {"おはよう", "Good Morning"};
  if(time < 18) return new String[] {"こんにちは", "Good Afternoon"};
  return new String[] {"こんばんは", "Good Evening"};
  System.out.println("Hello!");
}
```

return後なので実行されない
→「到達不能なコード」というエラーになる

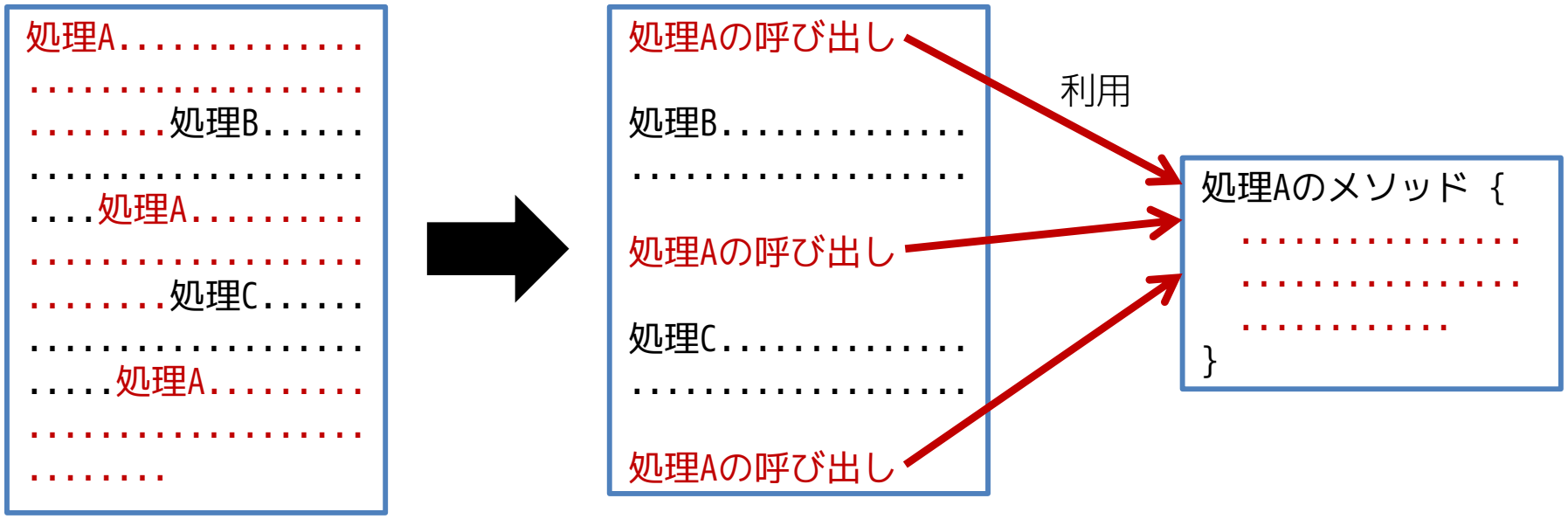
メソッドの使いどころ

- 長い処理を複数のより小さなメソッドに分割することで理解しやすくする
  - 小さな処理に分けることで、一度に把握しなくてはいけない分量が少なくなる



メソッドの使いどころ

- 何度も行う一連の処理をメソッドにすることで、再利用しやすくする
  - 同じ処理を何度も記述するより効率的
  - 処理内容の変更にも対応しやすい



練習

- 練習12-3