

オブジェクト指向 1

メソッド・クラスを学ぶ前準備

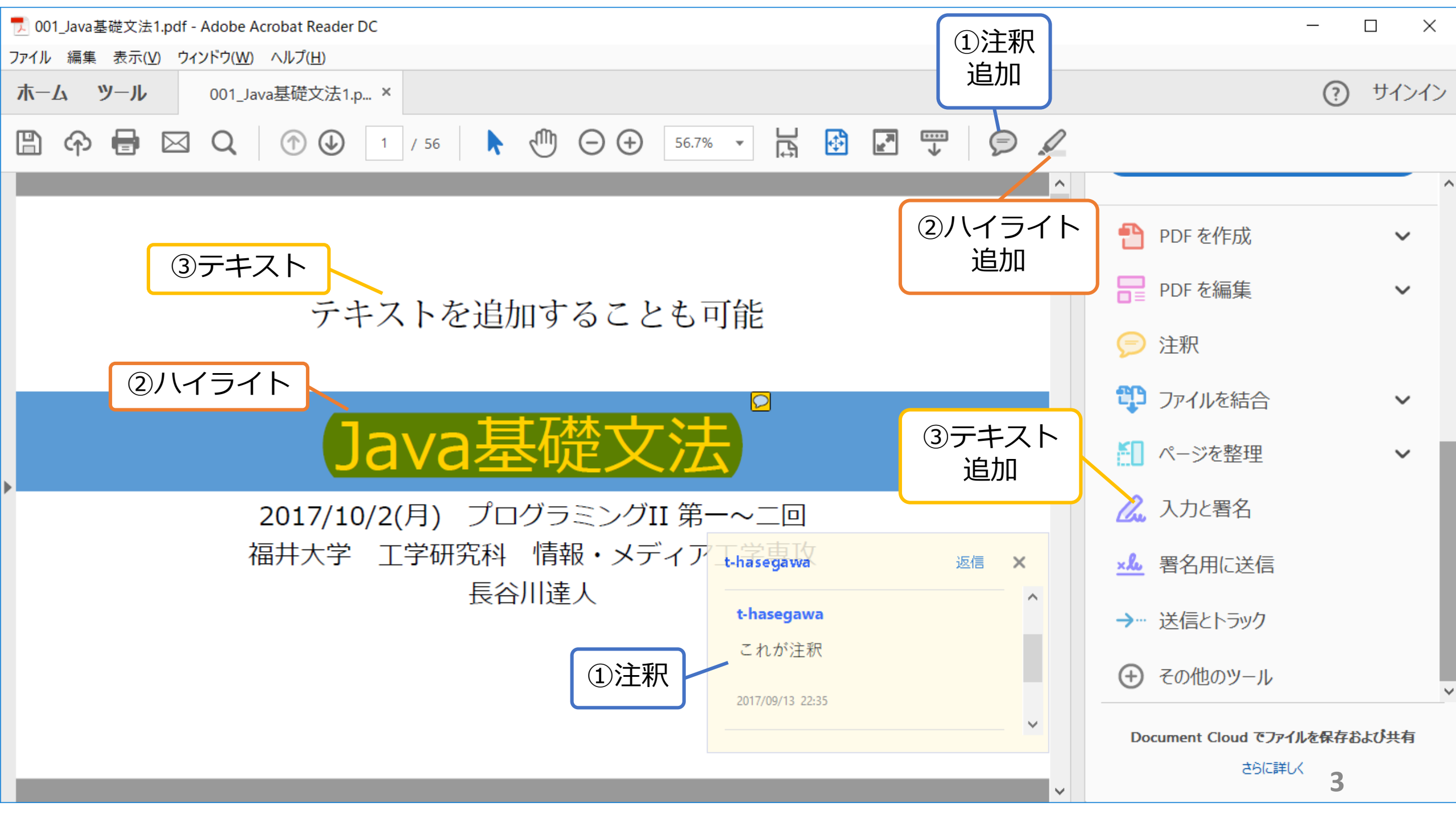
2017/10/23(月) プログラミングII 第三回
福井大学 工学研究科 情報・メディア工学専攻
長谷川達人

※WindowsでPC起動しておいてください.

本講義の概要

前半：Java 担当：長谷川		後半：Scala 担当：石井先生	
第1回	Java基礎文法/開発環境の使い方	第9回	Scala言語の基本
第2回	Java基礎文法/C言語との差異	第10回	関数型プログラミングの基本
第3回	オブジェクト指向	第11回	関数とクロージャ
第4回	クラス/インスタンス/メソッド1	第12回	関数型プログラミングの簡単な例
第5回	クラス/インスタンス/メソッド2	第13回	静的型付けと動的型付け
第6回	継承/カプセル化1	第14回	ケースクラス/パターンマッチング
第7回	継承/カプセル化2	第15回	多相性やパターンマッチングの例
第8回	中間試験/Java言語のまとめ	第16回	期末試験

※来年以降は全てJavaになる予定



本日の目標

概要

Javaで最も重要である**オブジェクト指向**を理解するための準備として、メソッドとクラスの考え方、使い方を学ぶ。

目標

独自のメソッドを実装できる。
メソッドをグループ分けしたクラスを実装できる。



なるほど

本日の目次

- メソッド
 - 変数のスコープ
 - 引数と戻り値
 - 配列引数と配列戻り値
 - 値渡しと参照渡し
 - オーバーロード
 - 可変長引数
- クラスを学ぶ前準備
 - ファイルの分割
- 演習課題

メソッド＝機能の切り分け

```
class Sample1{  
    public static void main(String[] args){  
        処理群A  
        処理群B  
        処理群A  
    }  
}
```

メソッド内処理が長くなるほど
管理がしづらい

- 読むのがつらい
- 影響範囲がわかりづらい
- バグ混在リスクが上がる
- ムダに記述することが多い

メソッド = 機能の切り分け

```
class Sample1{  
    public static void main(String[] args){  
        methodA(); //①  
        処理群B  
        methodA(); //②  
    }  
    public static void methodA(){  
        処理群A  
    }  
}
```

methodA()として
機能を切り分ける。

可読性の向上
methodA()の機能が
わかっていれば、
main()が読みやすい。

手間の削減
同じコードを無駄に
二回書かなくてよい。

影響範囲の限定
1か所のバグを直すと
①も②も直る。

メソッドの使い方

```
public static void main(String[] args){  
    double res = 0.0d;  
    // 呼び出し方法  
    // [メソッド名]([引数], ...);  
    res = methodA(5, 1.2d);  
}  
// 宣言方法  
// public static [戻り値] [メソッド名]([引数], ...){  
public static double methodA(int num, double val){  
    return num*val;  
}
```

重要！

重要！

public static は各々意味はあるが、現段階ではおまじないとしておく。
C言語と異なるのは、プロトタイプ宣言が不要であるという点である。

メソッドの使い方

変数のスコープ（有効範囲）

変数のスコープは宣言されてから
同じブロックが終了するまでの
間に限定される。

ブロック：括弧の開始 "{" から、
括弧の終了 "}" までで囲まれた
領域のこと

int型変数iが二回登場して
いるが、お互いの処理が
影響し合っていない。

```
public class Sample1{  
    public static void main(String[] args){  
        int i = 10;  
        methodA();  
        i *= 2;  
        System.out.print(i+",");  
    }  
    public static void methodA(){  
        int i = 10;  
        i *= -2;  
        System.out.print(i+",");  
    }  
}
```

[出力] -20,20,

メソッドの使い方

練習問題 1 : 変数のスコープ (有効範囲)

このプログラムはコンパイルエラーになる。なぜだろう？

```
public class Sample1{  
    public static void main(String[] args){  
        for(int i=0;i<10;i++){  
            System.out.println("hoge");  
        }  
        System.out.println(""+i);  
    }  
}
```



Sample1.java:6: エラー: シンボルを見つけられません
System.out.println(""+i);

メソッドの使い方

変数のスコープ（有効範囲）

ではどうやってメソッドに変数を送るのか？

→ **グローバル変数もどき**を使う技があるが、**非推奨**である。

要するにほぼ禁止

```
public class Sample1{  
    public static int i = 10;  
    public static void main(String[] args){  
        methodA();  
        i *= 2;  
        System.out.print(i+",");  
    }  
    public static void methodA(){  
        i *= -2;  
        System.out.print(i+",");  
    }  
}
```

スコープ

メソッドの外で
public staticを付けて
宣言する

どのメソッドからでも読み書きが
可能なため、バグの切り分けが
行いづらくなる。

[出力] -20,-40,

メソッドの使い方

引数（ひきすう）と戻り値（もどりち）

ではどうやってメソッドに変数を送るのか？
→ **引数**と**戻り値**を使う。

```
public static void main(String[] args){  
    double res = 0.0d;  
    // 呼び出し  
    res = methodA(5, 1.2d);  
}  
// 宣言  
public static double methodA(int num, double val){  
    return num*val;  
}
```

メソッドに値を渡すことができる

戻り値：メソッドで処理した値を戻すことができる。
一つだけ宣言可能で、return文で指定する。

引数：事前に数と型を宣言しておく。
いくつでも宣言可能だが、呼び出し時には同じ数指定する必要がある。

メソッドの使い方

練習問題 2 : printメソッドの自作

Javaで文字列表示を行う時, 毎回長い文字を打つのが面倒だし,
int型変数を表示するとき「"+i」とするのも面倒である.
int型引数をそのまま表示してくれるメソッドprintInt(int)を実装せよ.

```
public static void main(String[] args){  
    printInt(100);  
}
```

[出力] 100

メソッドの使い方

配列引数と配列戻り値

引数と戻り値には配列を指定することができる。

```
public static void main(String[] args){
    int[] nums = {0,10};
    String[] strs = toStrings(nums);
    for(String str : strs){
        System.out.println(str);
    }
}

public static String[] toStrings(int[] nums){
    String[] array = new String[nums.length];
    for(int i=0;i<nums.length;i++){
        array[i] = "array[" + i + "]=" + nums[i];
    }
    return array;
}
```

引数がint型配列で、
戻り値がString型配列

変数宣言の時のように
要素数を明示しない

[出力] array[0]=0
array[1]=10

メソッドの使い方

練習問題 3 : printメソッドの改良

先ほど自作したメソッドprintInt(int)は1変数しか表示することができなかった。配列の引数を用いて、複数変数をカンマ区切りで表示するように改良せよ。

```
public static void main(String[] args){  
    int[] nums = {0,10,20};  
    printInts(nums);  
}
```

[出力] 10, 20, 30,

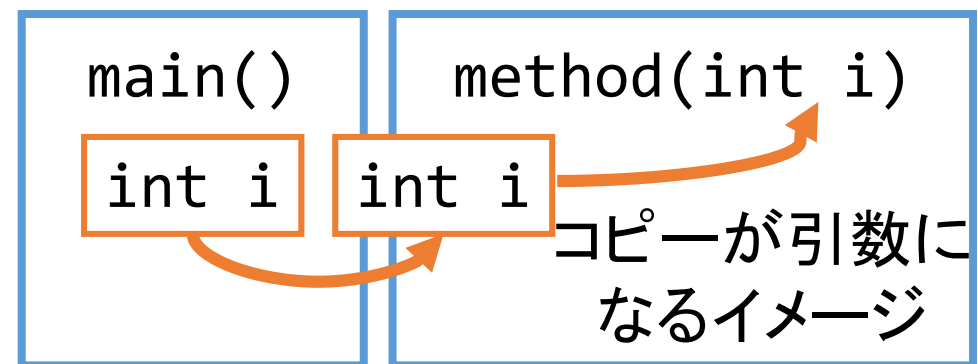
メソッドの使い方

引数として送られる値

int型等，通常型の時

メソッドに引数で値を送るとき、
実物は送らず値だけを送る

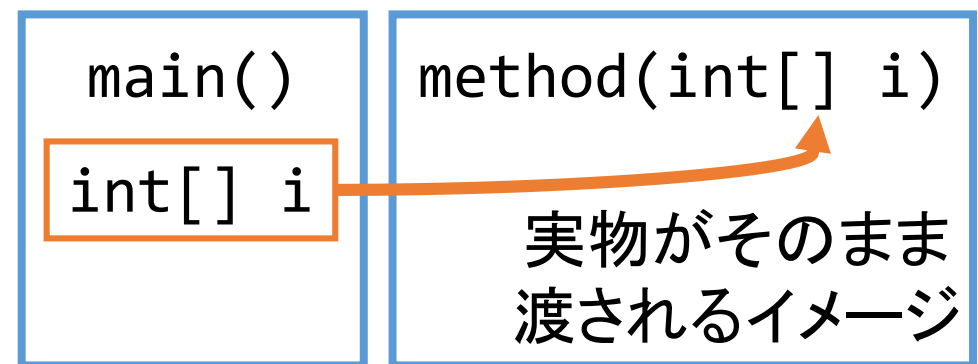
→メソッド内で値を編集しても、
呼び出し元変数は変わらない。



配列等，参照型の時

メソッドに引数で値を送るとき、
実物をそのまま送るイメージ

→メソッド内で値を編集すると、
呼び出し元変数も変更される。



メソッドの使い方

引数として送られる値

練習課題 2 で自作したメソッド内でnumの値を更新してみると, ,

```
public static void main(String[] args){  
    int num = 0;  
    printInt(num);  
    printInt(num);  
    printInt(num);  
}  
public static void printInt(int num){  
    System.out.println(""+num);  
    num += 10; // 更新してみる  
}
```

値を渡すだけで, メソッド内には
実体を送られるわけではない.

0
0
0

[出力]

メソッドの使い方

引数として送られる値

練習課題 3 のnumsを更新してみると

```
public static void main(String[] args){  
    int[] nums = {0,10,20};  
    printInts(nums);  
    printInts(nums);  
    printInts(nums);  
}
```

配列自体が送られる（イメージの）ため、
メソッド内で要素を改変すると、
元の配列の要素も改変されてしまう。

```
public static void printInts(int[] nums){  
    for(int i=0;i<nums.length;i++){  
        System.out.print(nums[i] + ", ");  
        nums[i] += 10; // 値を更新してみる  
    }  
    System.out.println(""); // 最後に改行しておく  
}
```

0, 10, 20,
10, 20, 30,
20, 30, 40,

1週目はそのまま表示されるが、
2回目は10ずつ加算されている。

メソッドの使い方

練習問題 4 : 引数として送られる値

次のプログラムを実行した際の出力を予測せよ.

```
public static void main(String[] args){
    String str = "a";
    String[] sarray = {"b", "c"};
    modify(str, sarray);
    System.out.println(str + ", " + sarray[0] + ", " + sarray[1]);
}
public static void modify(String str, String[] sarray){
    // 引数を改変する
    str += "!!!!!!";
    for(int i=0;i<sarray.length;i++){
        sarray[i] += "!!!!!!";
    }
    // 戻り値なし
}
```

メソッドの使い方

オーバーロード

- 配列引数を用いて複数変数を表示するメソッドを実装したが、毎回配列を宣言して呼び出すのは少々面倒である。普通にint型引数を使って実装はできないものだろうか。

```
public static void main(String[] args){  
    printInts(100);  
    printInts(100,200);  
}
```

1変数でも2変数
でも実行できる。

```
public static void printInts(int num){  
    System.out.println(""+num);  
}
```

オーバーロード：同じ名称のメソッド
を多重定義することができる。その際、
引数の**数か型を変える必要**がある。

```
public static void printInts(int num1, int num2){  
    System.out.println(num1 + ", " + num2);  
}
```

100 100, 200	[出力]
-----------------	------

メソッドの使い方

可変長引数

- オーバーロードでは、1引数の場合、2引数の場合、... をすべて事前に定義する必要があり面倒であった。もっと良い方法はないだろうか。

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

```
    printInts(100);
```

```
    printInts(100,200);
```

```
}
```

```
public static void printInts(int... nums){
```

```
    for(int n : nums){
```

```
        System.out.print(n + ", ");
```

```
    }
```

```
    System.out.println("");
```

```
}
```

可変長引数：いくつでも繰り返してよい引数を「...」で定義することができる。使用時は通常の配列と同じ扱いとなる。

- ・1メソッドにつき1種類のみ定義可能
- ・最後の引数にのみ適応可能

// 最後に改行

100
100, 200

[出力]

メソッドのまとめ

- 使用方法は基本的にはC言語と同じである.

```
[メソッド名]([引数], ...); // 呼び出し方法
public static [戻り値] [メソッド名]([引数], ...){ // 宣言方法
```

- プロトタイプ宣言は不要である.
- 変数のスコープ（有効範囲）はブロック単位（{中括弧}で囲まれた範囲内）である.
- 引数と戻り値には配列を用いることができる.
- 通常型は値渡しに対し配列等参照型は参照渡しである.
- 引数の数か型を変えれば，同じ名称のメソッドを多重定義することができる.

本日の目次

- メソッド
 - 変数のスコープ
 - 引数と戻り値
 - 配列引数と配列戻り値
 - 値渡しと参照渡し
 - オーバーロード
 - 可変長引数
- **クラスを学ぶ前準備**
 - ファイルの分割
- 演習課題

クラスを学ぶ前準備

ファイルの分割

```
class Sample1{
```

メソッドA (計算処理A)

メソッドB (計算処理B)

メソッドC (計算処理C)

メソッドD (入出力処理D)

メソッドE (入出力処理E)

メソッドF (入出力処理F)

```
}
```

クラス内のメソッドが多くなる
ほど管理がしづらい

- 読むのがつらい
- 影響範囲がわかりづらい
- バグ混在リスクが上がる
- ムダに記述することが多い

クラスを学ぶ前準備

ファイルの分割

```
class Sample1{  
    public static void main(String[] args){  
        Calc.methodA();  
        Calc.methodB();  
        InOut.methodE();  
        InOut.methodF();  
    }  
}
```

呼び出し時は
[クラス名].[メソッド名]();

```
class Calc{  
    メソッドA  
    メソッドB  
    メソッドC  
}
```

```
class InOut{  
    メソッドD  
    メソッドE  
    メソッドF  
}
```

可読性の向上

手間の削減

影響範囲の限定

分業の容易さ

クラスファイルごとに
分業がしやすい

役割ごとにクラス
ファイルを分ける.

クラスを学ぶ前準備

練習問題 5 : ファイルの分割

Renshu1_5.javaを作成し, 下記のmain()メソッドを作成する.

```
class Renshu1_5{  
    public static void main(String[] args){  
        add(10, 20);  
        sub(40, 20);  
    }  
}
```

下記のメソッド二つを実装せよ. なお, add()はaとbを加算した結果を出力するメソッド, sub()はaからbを減算した結果を出力するメソッドである.

```
public static void add(int a, int b){}  
public static void sub(int a, int b){}
```

クラスを学ぶ前準備

練習問題 5 : ファイルの分割

続いて, add()とsub()を別のクラスファイルに分割する. 分割先は「Calc.java」とし, メソッドを移動せよ. その際, main()メソッド内の①, ② (下記ソース) を適切に変更しなければ動作しない. どのように変更すればよいか.

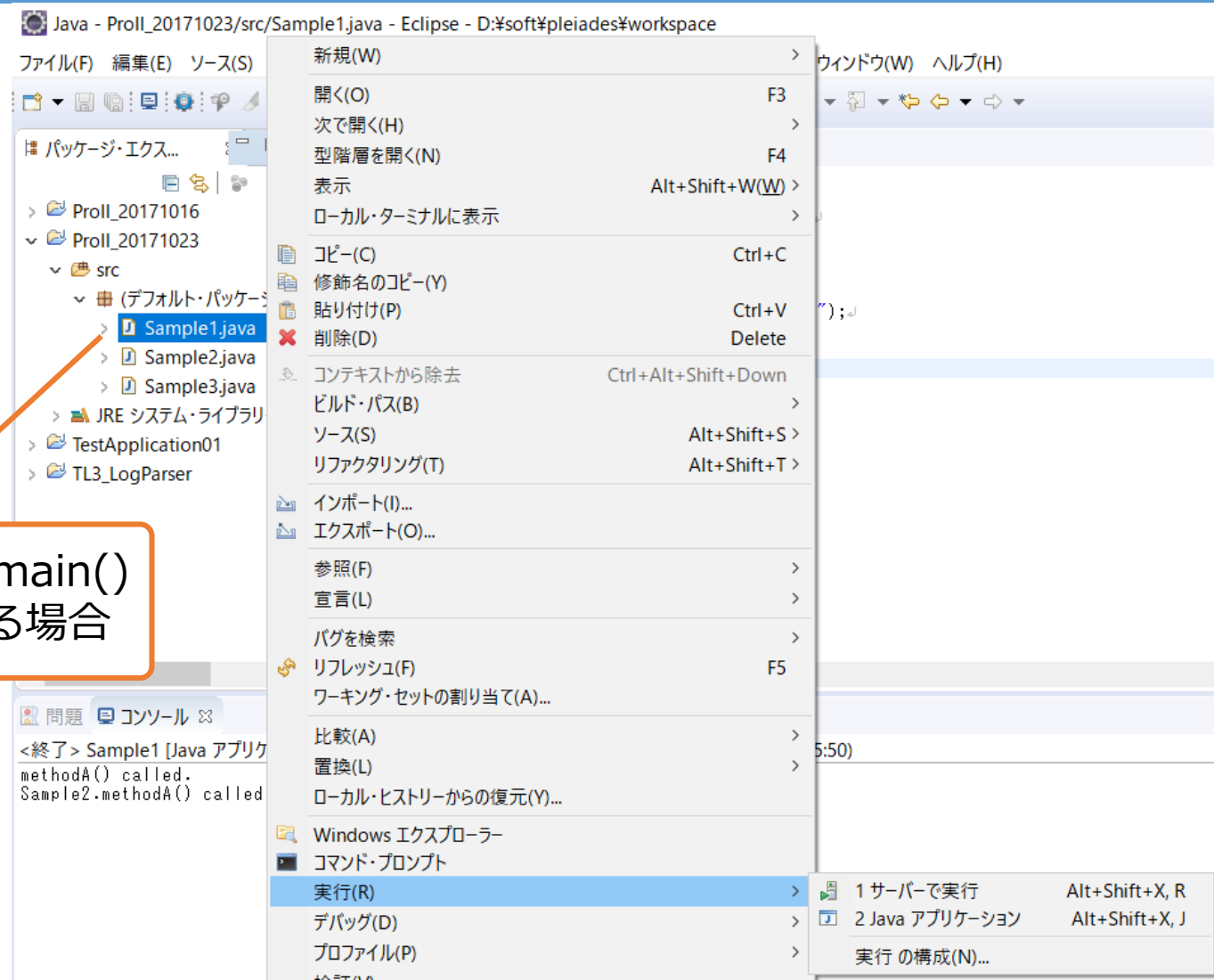
```
class Renshu1_5{  
    public static void main(String[] args){  
        add(10, 20);    // ①  
        sub(40, 20);    // ②  
    }  
}
```

クラスを学ぶ前準備

ファイルの分割：Eclipseで実行するときの注意

ファイルが複数あるため、Eclipseで実行するときには、**main()メソッドの記述されているファイル**上で、右クリック→実行→Javaアプリケーションとすること。

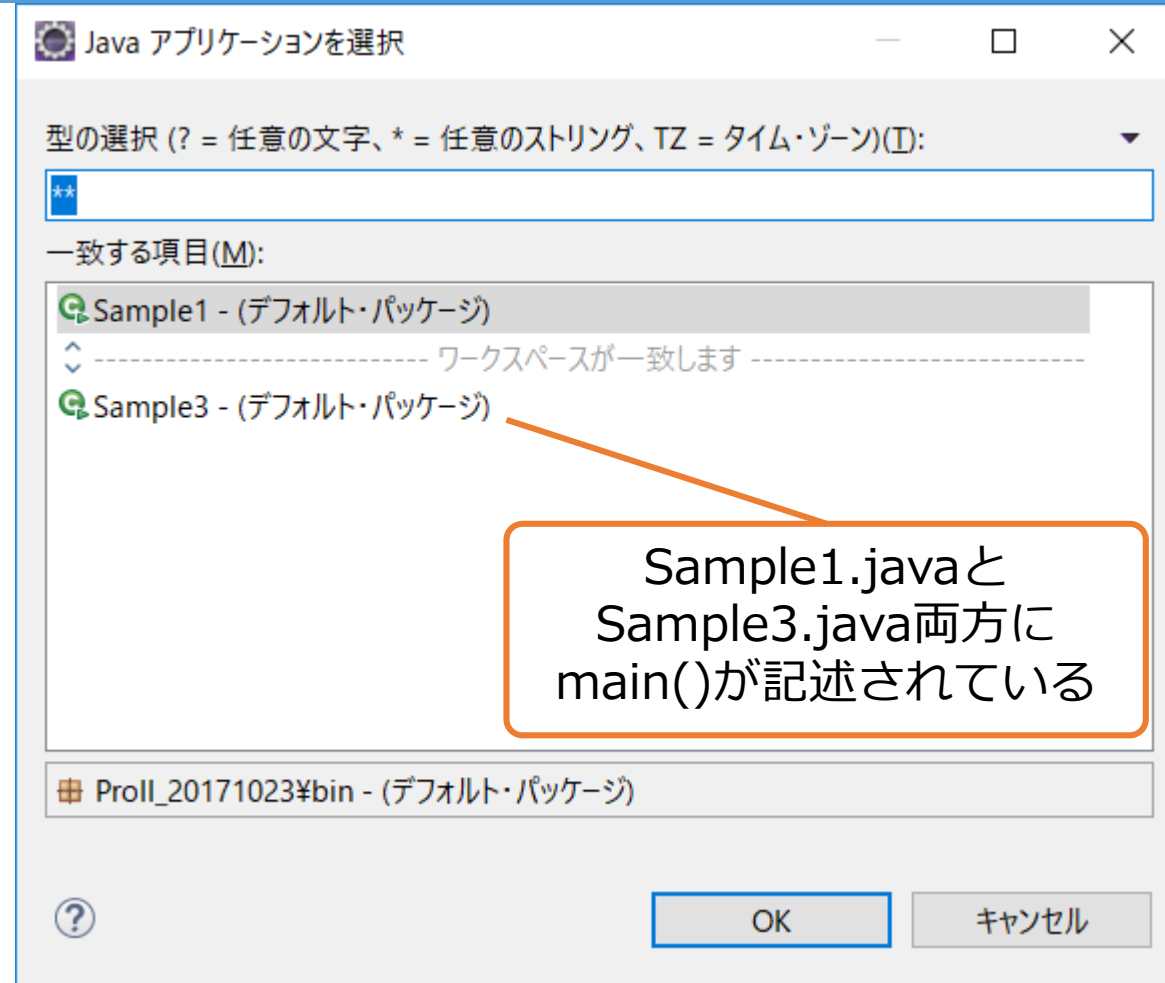
Sample1.javaにmain()が記述されている場合



クラスを学ぶ前準備

ファイルの分割：Eclipseで実行するときの注意

プロジェクトフォルダ上で右クリック→実行→Javaアプリケーションとしてもよいが、main()が複数のファイルに記述されている場合、どれを実行するのか選択する必要がある。



本日の目次

- メソッド
 - 変数のスコープ
 - 引数と戻り値
 - 配列引数と配列戻り値
 - 値渡しと参照渡し
 - オーバーロード
 - 可変長引数
- クラスを学ぶ前準備
 - ファイルの分割
- **演習課題**

本日の提出課題

課題

演習課題をEclipseで開発し,
ソースコードを提出する.

資料を公開しているサイトから, 「課題提出」で提出ページに行ける.

<http://hsgw-nas.fuis.u-fukui.ac.jp/lecture.html>

直リンクはこちら

http://hsgw-nas.fuis.u-fukui.ac.jp/lecture_file.html

演習課題

本日の演習課題では「MyMethods.java」の1ファイルの提出のみでよい。ただし、次の条件に従ってMyMethods.javaを実装すること。

- MyMethods.java内にはmain()は記述しない。
- 次のページ以降で示す演習課題 1 ～ 4 のメソッドが実装されている。
 - 間に合わなかった人は完成したメソッドまでで良い。
- 動作確認を行う際には別途Main.javaを作成し、main()メソッドを実装したうえで、MyMethods.test();のように実行するとよい。

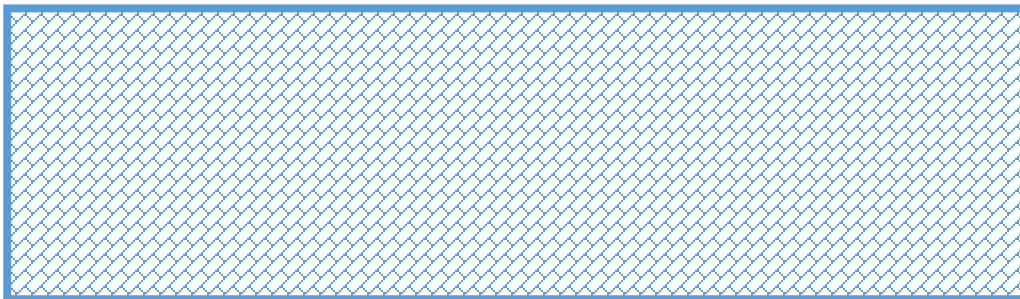
演習課題 1

translate()

動作確認例：

```
public static void main(String[] args){
    String[] strs = {"Boy", "Ant", "C"};
    for(String str : strs){
        MyMethods.translate(str);
    }
}
```

```
public static void translate(String str){
```



```
}
```

課題要件：

- 英→日に翻訳して出力する次のメソッドtranslate()を実装せよ.
- 対象は次の2単語＋その他のみで良い.
 - Ant -> "蟻"
 - Boy -> "男の子"
 - その他 -> "わかりません"
- メソッド内で出力も行うこと.

入出力例：

入力：Boy Ant C

出力：男の子

蟻

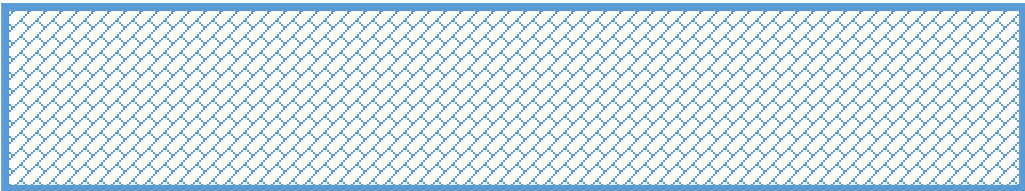
わかりません

演習課題 2

statistics()

動作確認例：

```
public static void main(String[] args){
    double[] values = {10.0, 12.0, 14.0};
    double[] res = MyMethods.statistics(values);
    for(double val : res){
        System.out.println("" + val);
    }
}
```

```
public static double[] statistics(double[] values){
    double[] res = new double[4];
    
    return res;
}
```

課題要件：

- 基本統計量の一部を計算する statistics()を実装せよ.
- 引数と戻り値はどちらも double型配列とし, 戻り値は
res[0] : 合計値
res[1] : 平均値
res[2] : 最大値
res[3] : 最小値
を格納する.

入出力例：

入力：{10.0, 12.0, 14.0}

出力：{36.0, 12.0, 14.0, 10.0}

演習課題 3

append()

動作確認例：

```
public static void main(String[] args){  
    MyMethods.append(10, 20, 30);  
    MyMethods.append(10.0, 20.0, 30.0);  
    MyMethods.append("10", "20", "30");  
}
```

今回はメソッド宣言のヒントなし

課題要件：

- 引数の総和を表示するappend()メソッドを実装せよ.
- オーバーロードを用いて, int型, double型, String型に対応せよ.
- 可変長引数を用いて, 引数の数はいくつでも有効とせよ.
- String型の場合, 文字列として結合した結果を出力すればよい.

入出力例：

入力：左のコードを参照

出力：60

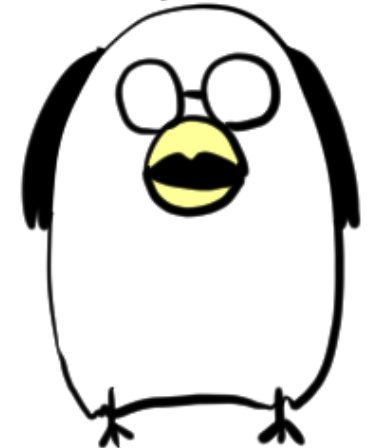
60.0

102030

本日のまとめ

- メソッドの使い方を学習した.
- C言語とは異なる点として以下などが紹介された.
 - プロトタイプ宣言が不要
 - 配列引数, 配列文字列が使用可能
 - オーバーロード (多重定義) が使用可能
 - 可変長引数が使用可能
- クラスを理解する前準備として,
ファイルを分割する方法を学習した.

わかりました



次週予告

※次週以降は計算機室

とうとうJavaの本質であるオブジェクト指向を学習する。
特に、今回学習した「ファイルを分割するため」だけではない
クラスの使い方について学習する。

本日の提出課題

課題

演習課題をEclipseで開発し,
ソースコードを提出する.

資料を公開しているサイトから, 「課題提出」で提出ページに行ける.

<http://hsgw-nas.fuis.u-fukui.ac.jp/lecture.html>

直リンクはこちら

http://hsgw-nas.fuis.u-fukui.ac.jp/lecture_file.html