

プログラミング基礎 I

第9回 メソッド基礎

九州産業大学 理工学部

pk@is.kyusan-u.ac.jp

プログラムを書く上での注意

- 一区切りごとに実行して、エラーがないか確認する！！
 - エラーを放置して先に進むと、どんどんエラーが増えていく・・・
 - 表示が変わるタイミングでは必ず実行して確認すること！！！！
- インデントに注意！
 - **インデントが崩れたら、Ctrl + A → Ctrl + I**
- エラーが出たら、エラーの内容を確認してみる
 - まずはエラーの内容を見て、自分で解決できないか考えてみよう
 - それでも分からなければ遠慮なく質問しましょう

今回の重要点

- メソッド呼び出し：メソッドを使う
 - 返り値と引数について理解する



メソッド

- ひとまとまりの処理に名前をつけたもの
- よく使う処理をまとめておくと、
同じようなプログラムを何度も書かなくてよくなる
 - ミスが減る、プログラムを修正しやすい

これまでに使ってきたメソッド

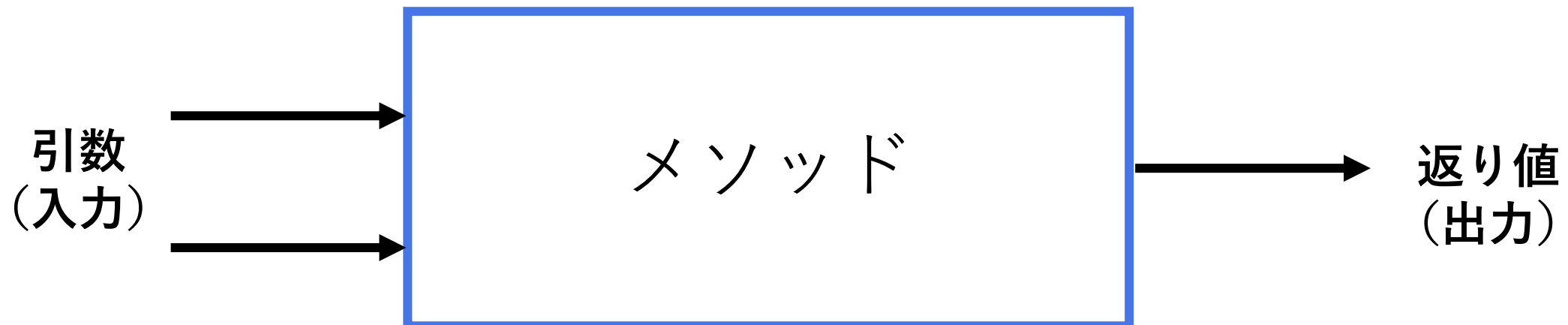
- `System.out.println("文字列");`
 - 文字列を標準出力へ出力するメソッド
 - `System`クラス **の** `out`フィールド **の** `println`メソッド
- `Scanner sc = new Scanner(System.in);`
`int n = sc.nextInt();`
 - 標準入力から入力された整数値を取り込むメソッド
 - `Scanner`クラス **の** `nextInt`メソッド

引数と返り値

- 引数：メソッドに入力する値
- 返り値：メソッドから返される値

メソッドに何を入力すると何が出力されるのか

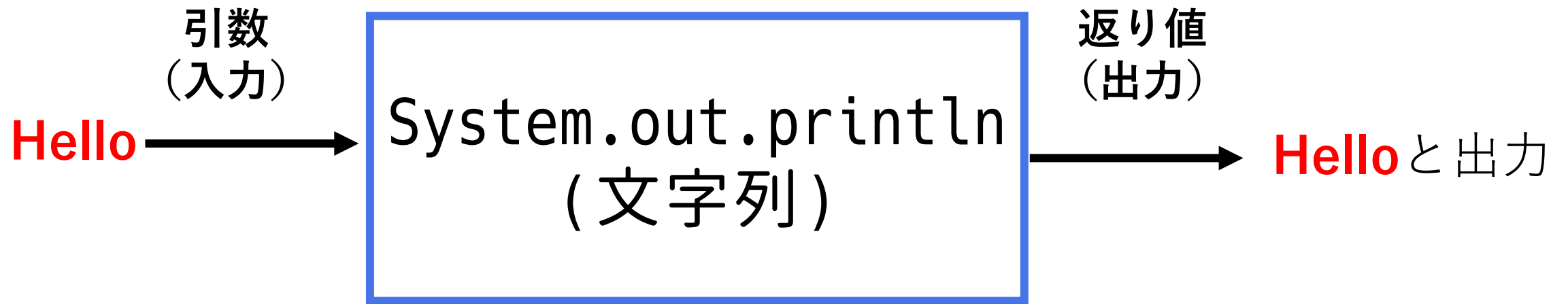
メソッド呼び出しでは、入出力の関係を理解することが重要



引数と返り値

- `System.out.println("Hello");`
 - "Hello" を `println` に渡すとコンソールに出力してくれる
 - 引数(入力)は "Hello" 、 返り値(出力)は無し
 - コンソールに出力はするが、呼び出した側に何かを返す訳では無い
- `int a = Math.abs(-5);`
 - `Math.abs` に -5 を渡すと絶対値を返してくれる
 - 引数(入力)は -5 (int型), 返り値(出力)は 5 (int型)

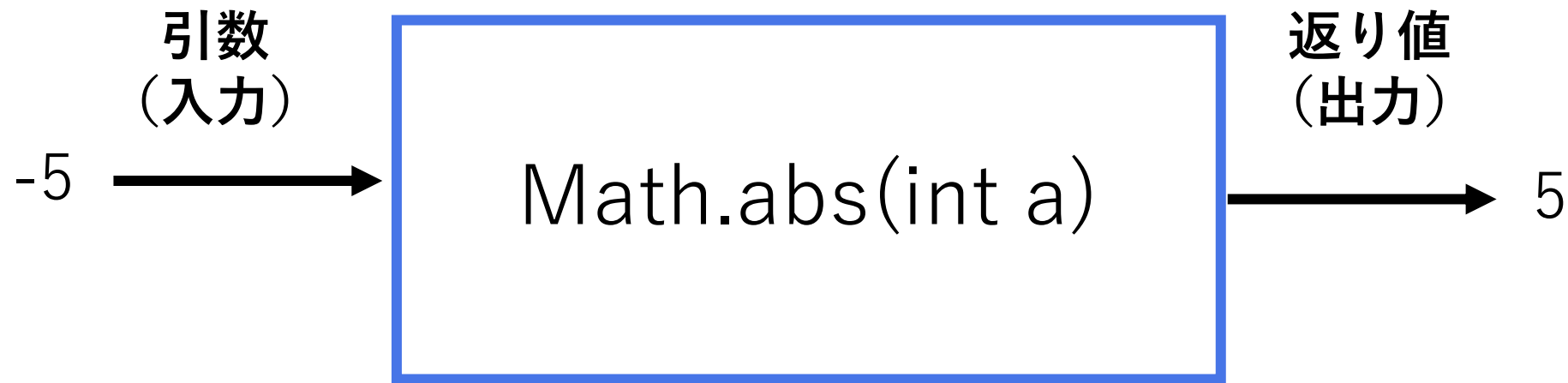
```
System.out.println( "Hello" );
```



- String型の値（文字列）を渡すと、
コンソールに表示してくれる

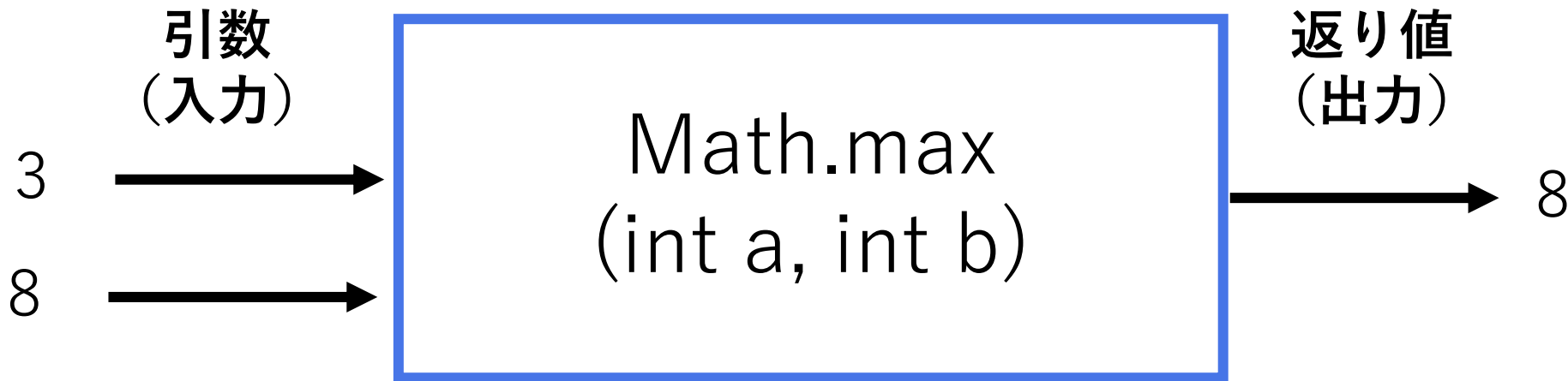
Math.abs(-5);

- -5 を渡すと、絶対値 (5)を返してくれる



- int型の値(整数値)を渡すと、その絶対値を返してくれる

Math.max(3, 8);



- `int`型の値(整数値)を2つ渡すと、最大値を返してくれる

Mathクラス

- Mathクラス
 - 様々な数値処理を実行するためのメソッドが含まれたクラス
 - 絶対値、最大値、最小値、累乗、平方根、三角関数等、様々な計算を実行することが出来る
- Javaに標準で搭載されている特別なクラスで、インスタンス生成無しに利用することが出来る

Mathクラスのメソッド（一部紹介）

- Mathクラス

- 様々な数値処理を実行するためのメソッドを利用できる

メソッド名と呼び出し例	引数 (メソッドへの入力)	返り値 (メソッドからの出力)	処理内容
<code>int a = Math.abs(-5);</code>	int型	int型	整数値を渡すと絶対値を整数値で返す
<code>int b = Math.max(5, 10);</code>	int型, int型	int型	整数値を2つ渡すと、 最大値を整数値で返す
<code>int c = Math.min(5, 10);</code>	int型, int型	int型	整数値を2つ渡すと、 最小値を整数値で返す
<code>double d = Math.pow(2,3);</code>	double型, double型	double型	数値 a と b を渡すと、 a の b 乗をdouble型で返す

Mathクラスのメソッド（一部紹介）

メソッド名と呼び出し例	引数 (メソッドへの入力)	返り値 (メソッドからの出力)	処理内容
<code>double e = Math.sqrt(2);</code>	double型	double型	数値を渡すと、その平方根をdouble型で返す
<code>double f = Math.floor(2.48);</code>	double型	double型	小数値を渡すと、小数点以下を切り捨てた小数値を返す
<code>double g = Math.ceil(2.48);</code>	double型	double型	小数値を渡すと、小数点以下を切り上げた小数値を返す
<code>int h = Math.round(5.6);</code>	double型	int型	小数値を渡すと、小数点第1位で四捨五入した整数値を返す

- Mathクラスのメソッド一覧（公式ドキュメント）

- <https://docs.oracle.com/javase/jp/8/docs/api/java/lang/Math.html>

- 同じ名前のメソッドがあることが気になった人は、教科書P103 を参考に
 - メソッドの多重定義：プログラミング基礎IIの内容

メソッドを呼び出すプログラムの例: Math.max

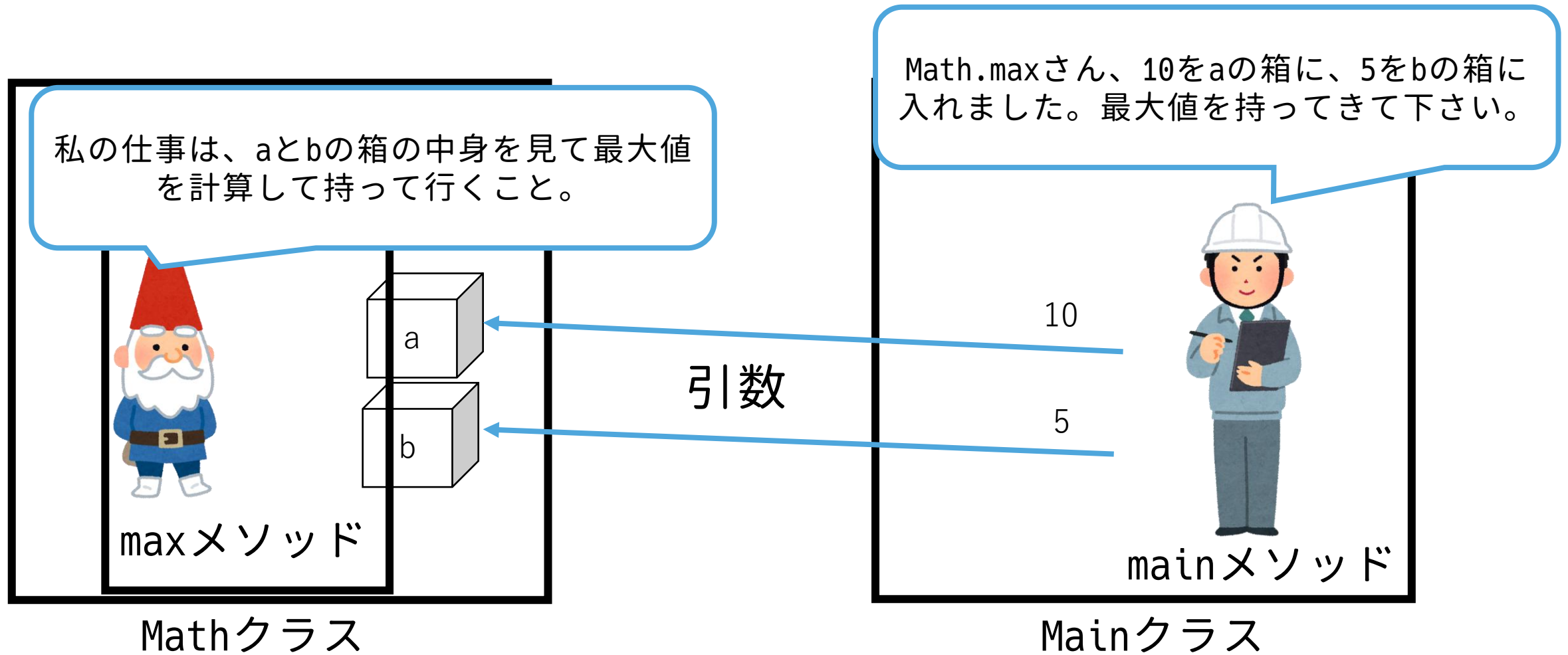
- メソッド呼び出し: メソッド名(引数1, 引数2, ...);

```
int a = -5;  
int b = -3;  
//int型のmax_abにMath.max(a,b)の戻り値を代入  
int max_ab = Math.max(a, b);  
//max_abを表示  
System.out.println(a + "と" + b + "の最大値は" + max_ab);
```

引数(入力): a, b (int型, int型)

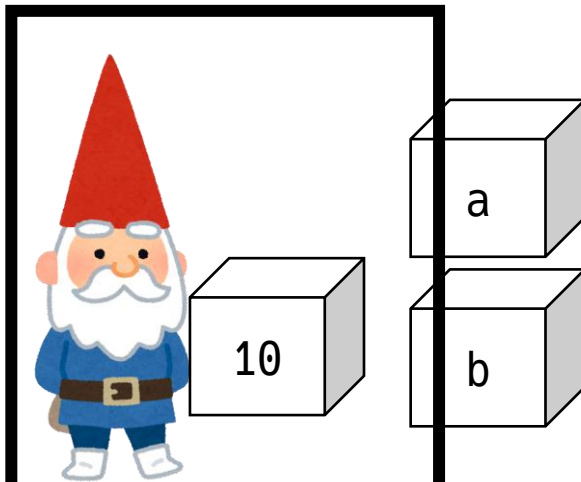
戻り値(出力): int型 を変数に受け取る

メソッドは呼ばれたら仕事をする



返り値があるメソッドは、結果を返す

呼ばれたから働きます...
10(a)の方が5(b)よりも大きいな。
int型の箱に10(aの中身)を入れて...



maxメソッド

Mathクラス

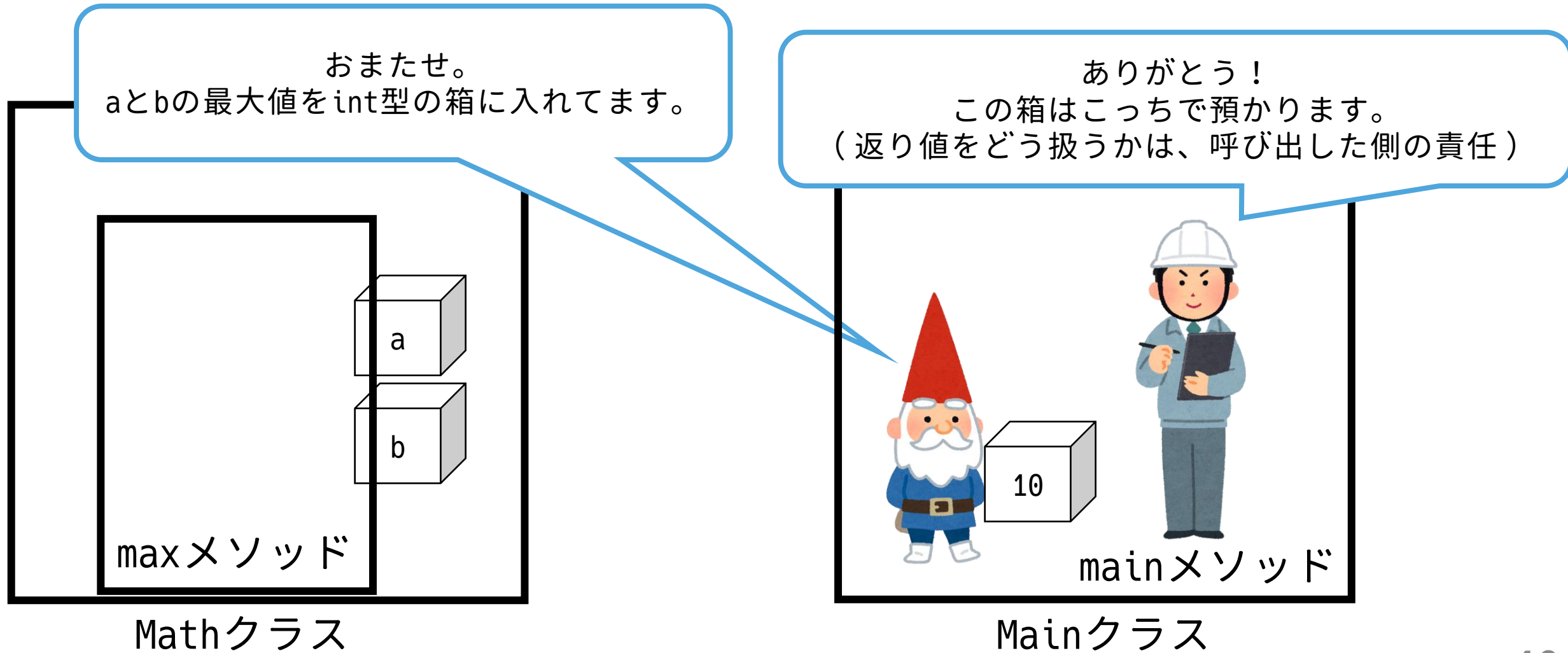
まだかな...
結果が返ってくるまで待機しておかないと



mainメソッド

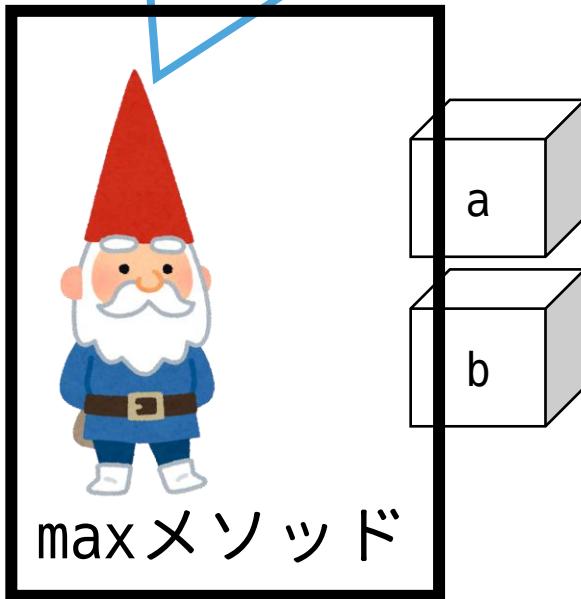
Mainクラス

返り値があるメソッドは、結果を返す



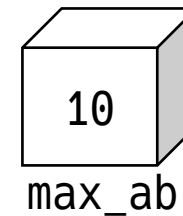
返り値を利用するのは呼び出した側

次に呼ばれるまでは休憩。
(メソッドは呼ばれないと何もしない)



Mathクラス

後で使う（表示したりする）から、
max_abを作っておこう



mainメソッド

Mainクラス

メソッドに渡す引数を間違えると・・・

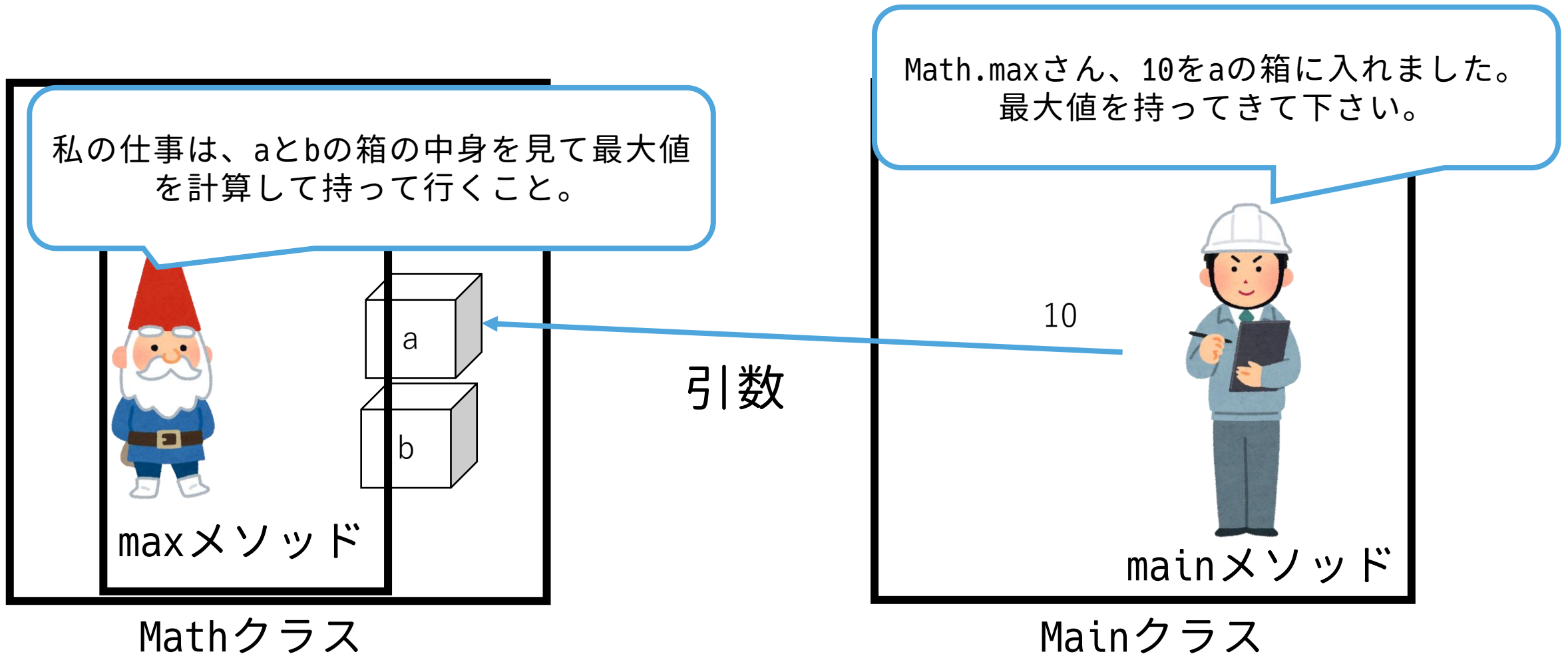
```
int a = -5;  
int b = -3;  
  
int max_ab = Math.max(a);  
  
//max_abを表示  
System.out.println(a + "と" + b + "の最大値は" + max_ab);
```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
 型 Math のメソッド max(int, int) は引数 (int) に適用できません

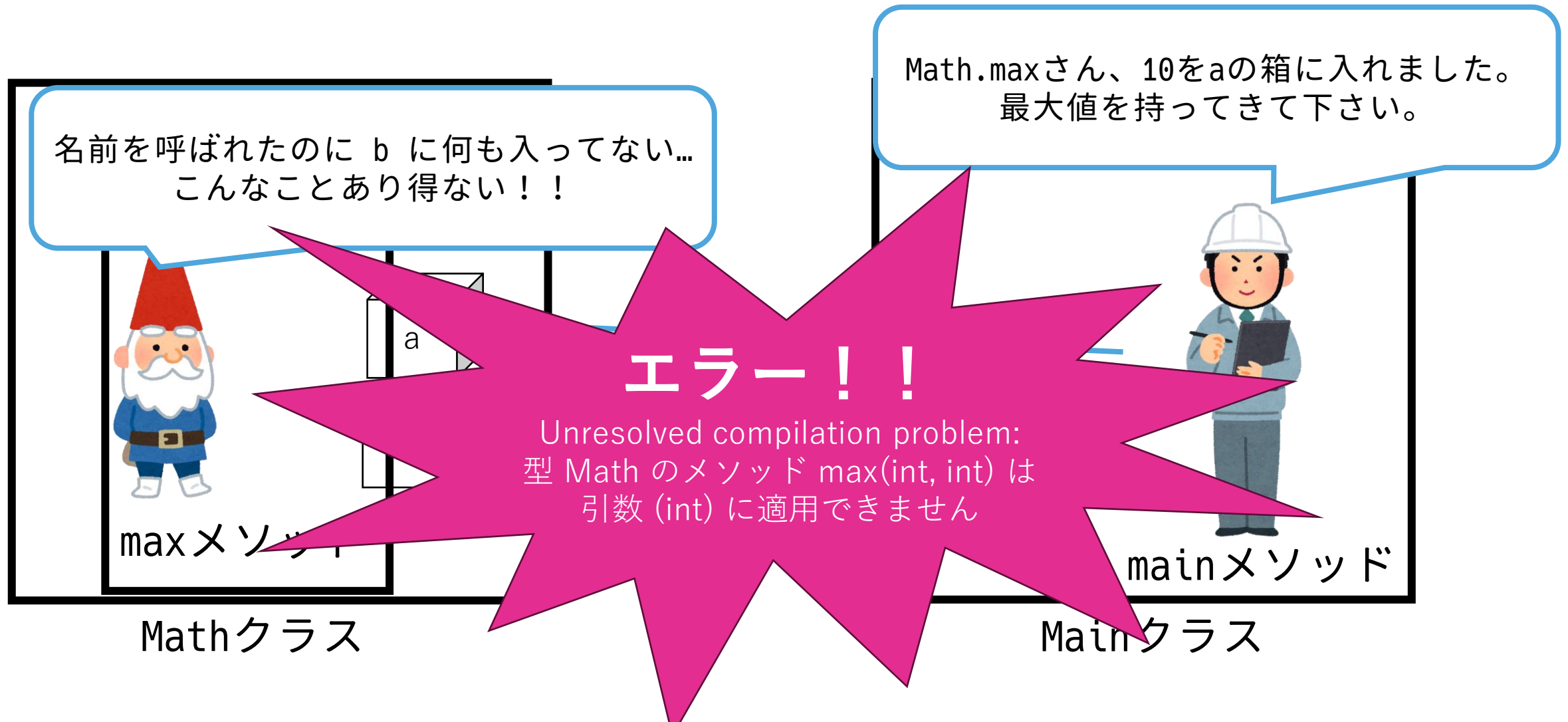
at Foo.main(Foo.java:13)

- max(int, int)はあるが、 max(int) はないためエラーとなる
- メソッド呼び出しでは、**引数を渡す順番と数を厳守**

メソッドは呼ばれたら仕事をする



正しく呼び出さなければエラーになる



Math.abs を利用せずに絶対値を求める場合

- メソッドを使わなくても同様の処理は実現できる

```
int a = -5;
int abs_a = a; //abs_aにaを代入
if(a < 0) { //もしも、aが0未満なら
    abs_a = abs_a * -1; //abs_aの符号を反転(負を正にする)
}
//abs_aを出力
System.out.println(a + "の絶対値は" + abs_a);
```

メソッドを利用する利点

- 複数の変数の値の絶対値を求めことを考える

```
int a = -5;
int abs_a = a; //abs_aにaを代入
if(a < 0) { //もしも、aが0未満なら
    abs_a = abs_a * -1; //abs_aの符号を反転(負を正にする)
}
//abs_aを出力
System.out.println(a + "の絶対値は" + abs_a);

int b = -5;
int abs_b = b;
if(b <= 0) { //もしも、bが0未満なら
    abs_b = abs_b * -1; //abs_bの符号を反転(負を正にする)
}
//abs_bを出力
System.out.println(b + "の絶対値は" + abs_b);
```

メソッド利用だと

- プログラムが短くなって見やすくなる

```
int a = -5;  
int abs_a = Math.abs(a); //abs_aにaを代入  
//abs_aを出力  
System.out.println(a + "の絶対値は" + abs_a);  
  
int b = -5;  
int abs_b = Math.abs(b);  
//abs_bを出力  
System.out.println(b + "の絶対値は" + abs_b);
```

- だけではなく・・・

実はさっきのプログラムには誤りが・・・

- 未満 のところを 以下にしてしまっている

```
int a = -5;  
int abs_a = a; //  
if(a < 0) { //も  
    abs_a = a  
}
```

```
//abs_aを出力  
System.out.println(a + "の絶対値は" + abs_a);
```

```
int b = -5;  
int abs_b = b;  
if(b <= 0) { //もしも、bが  
    abs_b = abs_b * -1  
}
```

```
//abs_bを出力  
System.out.println(b + "の絶対値は" + abs_b);
```

同じプログラムを何度も書くと、
同じミスを何度も書き直さないといけなくなる
ミスも多くなる

プログラムの保守性（メンテナンスのしやすさ）
確保のために、**同じプログラムは極力書かない**

メソッドを利用できる箇所では、
出来るだけメソッドを利用する

メソッドの入れ子

- メソッドの引数に、他のメソッドを入れることができる
- 例) 3 変数の最大値を求める
 - 入れ後にしない場合

```
int a = 8;    int b = 10;   int c = 6;
```

```
int maxAB = Math.max(a, b); // aとbの最大値を求める
```

まず、2つの変数の最大値を求めて保持しておく

```
int maxABC = Math.max(maxAB, c); // maxABとcの最大値
```

2つの変数の最大値と、もう一つの変数の最大値を改めて求める

入れ子にした場合

- 中間結果を保持しておかなくて良くなる

```
int a = 8;    int b = 10;   int c = 6;  
  
// aとbの最大値と,cの最大値  
int maxABC = Math.max(Math.max(a, b), c);
```

1行で書くことが出来る

System.out.println() で入れ子

- 表示する際に使う、

System.out.println() でも入れ子にすることが出来る

```
int a = 8;    int b = 10;  int c = 6;
```

```
// aとbの最大値と,cの最大値 を計算して表示
```

```
System.out.println(Math.max(Math.max(a, b), c));
```



表示結果は下記と同じ

```
int a = 8;    int b = 10;  int c = 6;
```

```
// aとbの最大値と,cの最大値をmaxABCに代入
```

```
int maxABC = Math.max(Math.max(a, b), c);
```

```
System.out.println(maxABC); // maxABCを表示
```

Stringクラスのメソッド（一部紹介）

```
String msg = "Hello";
```

メソッド名と呼び出し例	引数	返り値	処理内容
<pre>int a = msg.length();</pre>	なし	int型	文字列を渡すと、その文字列の長さを整数値で返す
<pre>int b = msg.indexOf("lo");</pre>	String型(文字列)	int型	引数の部分文字列が最初に出現するインデックスを返す
<pre>int c = msg.indexOf("l",3);</pre>	String型, int型	int型	引数で指定されたインデックス以降で、引数の部分文字列が最初に出現するインデックスを返す
<pre>String d = msg.replace("l", "x");</pre>	String型, String型	String型	第1引数の文字列を第2引数の文字列で置換した文字列を返す
<pre>String e = msg.substring(2,5);</pre>	int型, int型	String型	第1引数のインデックスから第2引数のインデックスの手前までの文字列を切り取って返す

Stringクラスのメソッド（一部紹介）

```
String str = "a,b,c,d,e";
```

メソッド名と呼び出し例	引数	返り値	処理内容
<pre>String [] spStr = str.split(",");</pre>	String型	String型 配列	文字列を渡すと、その文字列で分割した文字列型の配列を返す
<pre>String [] spStr = str. split("");</pre>	String型	String型 配列	splitメソッドに、空文字を渡すと一文字ずつ分割した配列を返す
<pre>boolean f = msg.equals("a,b,c,d,e,");</pre>	String型	boolean型	この文字列と引数の文字列を比較する
<pre>String g = msg.replace("l", "x");</pre>	String型, String型	String型	第1引数の文字列を第2引数の文字列で置換した文字列を返す

型変換を行うメソッド

- 文字列を数値に変換するメソッド

メソッド名と呼び出し例	引数	返り値	処理内容
<pre>int h = Integer.parseInt("10");</pre>	String型	int型	文字列を渡すと、その文字列を整数値に変換した値を返す
<pre>double i = Double.parseDouble("10.5");</pre>	String型	double型	文字列を渡すと、その文字列を小数値に変換した値を返す

- 数値に出来ない文字列を変換しようとするとうエラーになる
 - エラーを回避するためには、try-catch文を用いる必要がある
(実用的な内容にはなるが、基礎 I では扱わない)
 - 気になる人は自分で調べてみよう

メソッドの利用例

- 例：メールアドレスのユーザ名を切り出す
 - String.indexOfとString.substringを利用した例

```
String mailAddress = "k25rs999@st.kyusan-u.ac.jp";  
int atMarkIndex = mailAddress.indexOf("@");
```

0番目から @ のある位置 **未満** まで

```
String userName = mailAddress.substring(0, atMarkIndex);  
System.out.println(userName);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
k	2	5	r	s	9	9	9	@	s	t	.	k	y	u	s	a	n	-	u	.	a	c	.	j	p

メソッドの入れ子で処理する場合

- @の位置を変数に保持する必要がなければ、入れ子にして1行で処理できる

```
String mailAddress = "k25rs999@st.kyusan-u.ac.jp";  
  
String userName = mailAddress.substring(0, mailAddress.indexOf("@"));  
System.out.println(userName);
```

- 表示するだけで良いなら、更に入れ子にして処理することも可能

```
System.out.println(mailAddress.substring(0, mailAddress.indexOf("@")));
```

中間結果を変数に格納しておきたい場合

- ユーザ名だけでなく、ドメインも切り抜きたい

```
String mailAddress = "k25rs999@st.kyusan-u.ac.jp";  
int msgLength = mailAddress.length();  
int atMarkIndex = mailAddress.indexOf("@");  
  
String userName = mailAddress.substring(0, atMarkIndex);  
  
String dmainName = mailAddress.substring(atMarkIndex + 1, msgLength);  
  
System.out.println(userName + "    " + dmainName);
```

- @のインデックスを、ドメイン切り抜きでも利用したい
 - 後から利用したい場合には、変数に格納しておくと便利

まとめ

- メソッド
 - 処理をまとめて名前をつけたもの
- メソッド呼び出し
 - メソッド名(引数);
 - メソッド名(引数1, 引数2, 引数3);
 - Mathクラスの場合は、 Math.メソッド名(引数1, 引数2, ...);