プログラミング入門1

第7回

メソッド（1）

授業開始前に自己点検

前回までの必須課題はすべてできていますか

前回までの学習項目であいまいな所はありませんか

他人による評価でなく、自身による評価ができるということが自立するということです

自立なしには大学での勉学は成り立ちません

前回のテーマ

- switch 文
  - 主にbreak文とともに
  - 条件分岐のもうひとつのやり方
- for文，while文におけるbreak 文
  - ループ抜け出しの非常手段
- プロジェクトの持ち運び
  - Eclipseの機能から
  - export
  - import

多くの選択肢からひとつを選んで実行する
こんな形でよく使う

switch文を使うと

```
if (n==2)
  B1
else if (n==3)
  B2
else if (n==5)
  B3
else if (n==7)
  B4
else
  B5
```

```
switch(n){
  case 2:
    B1
    break;
  case 3:
    B2
    break;
  case 5:
    B3
    break;
  case 7:
    B4
    break;
  default:
    B5
}
```

ループ抜け出しのbreak文の例
valueが素数か判定する

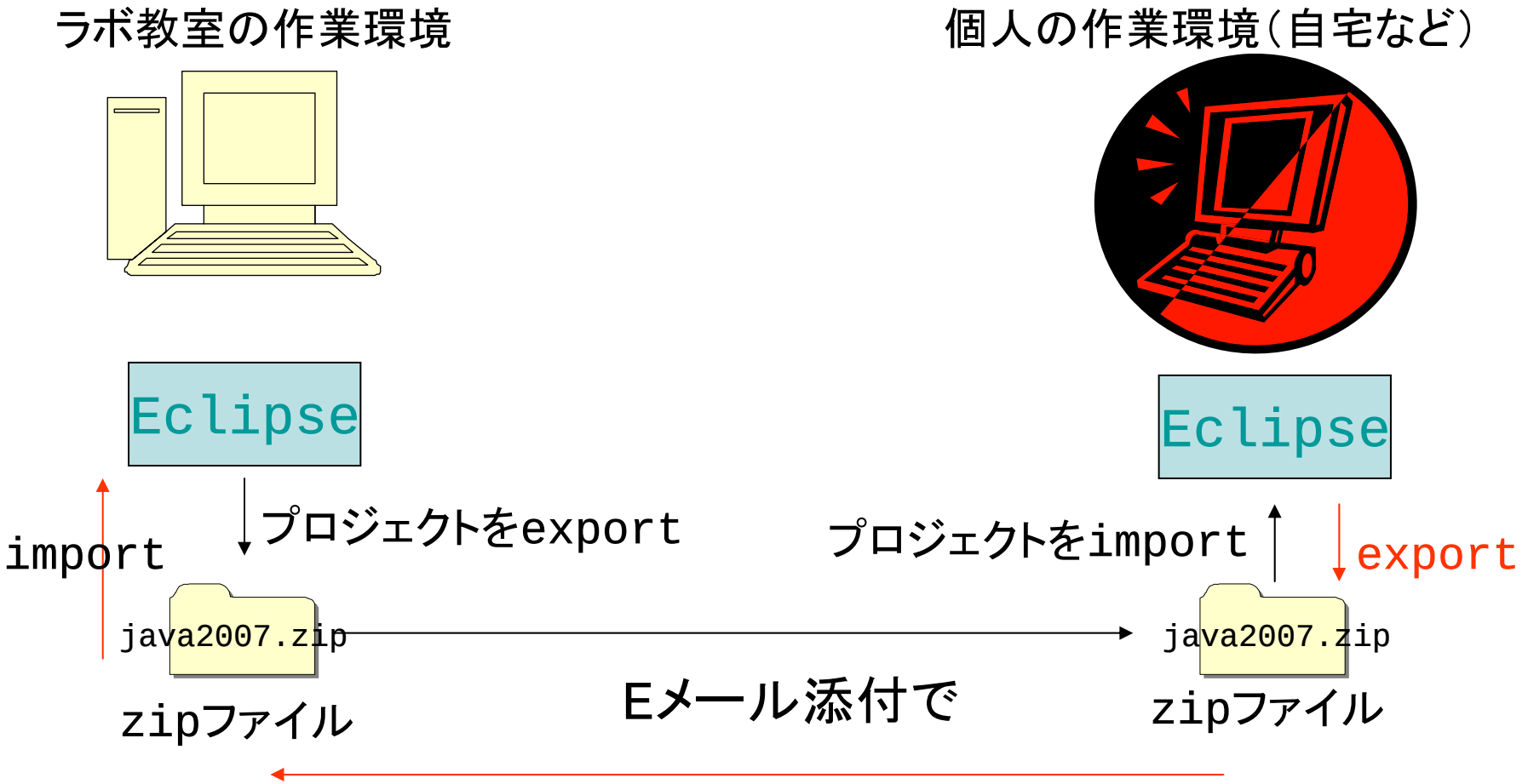
```
int divisor = 0;
for (int i = 2; i < value; i++) {
  if (value % i == 0) {
    divisor = i;
    break;
  }
}

if (divisor==0)
  System.out.println(value + " is prime");
else
  System.out.println(value + " is divisible by " + divisor);
```

素朴な素数判定法である。

break文は途中でのループ抜け出しに使える。

プロジェクトの持ち運び

export/importの詳細ついては第1回講義資料を参照



今回のテーマ

- メソッドとは
  - いくつかの命令の列を束ねて、一つの命令として扱えるようにしたもの
  - 今回学ぶメソッドの役割は、その他のプログラミング言語では関数またはサブルーチンと呼ばれることがある
- メソッドを書く
  - 宣言あるいは定義
- メソッドを使う
  - 起動あるいは呼び出し(call)

メソッドの例

```
public class SayHello {
  public static void main(String[] args) {
    sayHello();
    sayHello();
    sayHello();
  }

  public static void sayHello() {
    System.out.print("Hello!");
    System.out.println("This is sayHello.");
  }
}
```

これを実行すると

```
Hello!This is sayHello.
Hello!This is sayHello.
Hello!This is sayHello.
```

メソッドを定義する

```
public class SayHello {
  public static void main(String[] args) {
    sayHello();
    sayHello();
    sayHello();
  }

  public static void sayHello() {
    System.out.print("Hello!");
    System.out.println("This is sayHello.");
  }
}
```

これを実行すると

```
Hello!This is sayHello.
Hello!This is sayHello.
Hello!This is sayHello.
```

sayHelloメソッドを起動する(呼び出す)-初めて

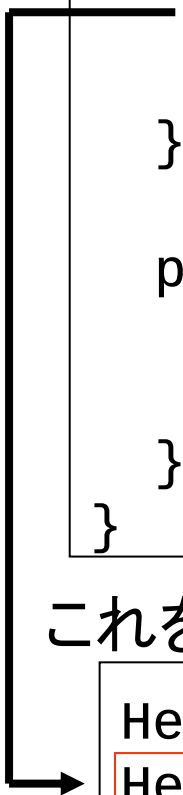
```
public class SayHello {
  public static void main(String[] args) {
    sayHello();
    sayHello();
    sayHello();
  }

  public static void sayHello() {
    System.out.print("Hello!");
    System.out.println("This is sayHello.");
  }
}
```

これを実行すると

```
Hello!This is sayHello.
Hello!This is sayHello.
Hello!This is sayHello.
```

sayHelloメソッドを起動する（呼び出す）-2回目

```
public class SayHello {
  public static void main(String[] args) {
    sayHello();
    sayHello();
    sayHello();
  }

  public static void sayHello() {
    System.out.print("Hello!");
    System.out.println("This is sayHello.");
  }
}
```



これを実行すると

```
Hello!This is sayHello.
Hello!This is sayHello.
Hello!This is sayHello.
```

sayHelloメソッドを起動する（呼び出す）-3回目

```
public class SayHello {
  public static void main(String[] args) {
    sayHello();
    sayHello();
    sayHello();
  }

  public static void sayHello() {
    System.out.print("Hello!");
    System.out.println("This is sayHello.");
  }
}
```

これを実行すると

```
Hello!This is sayHello.
Hello!This is sayHello.
Hello!This is sayHello.
```

sayHelloメソッドを使わないと

```
public static void main(String[] args) {
  System.out.print("Hello!");
  System.out.println("This is sayHello.");

  System.out.println("Hello!");
  System.out.println("This is sayHello.");

  System.out.println("Hello!");
  System.out.println("This is sayHello.");
}
```

メソッドを利用することの利点

- 命令を束ねて使うことができる
  → プログラムを短くできる
- 一連の命令に名前をつけられる (sayHello)
  → プログラムが見やすくなる
- 一連の処理を一つにまとめることができる
  → プログラムに間違いがあっても修正しやすい

メソッドを利用するには

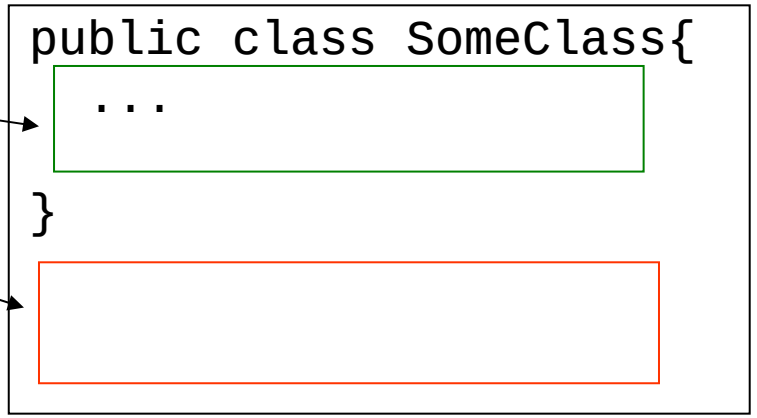
- メソッドの宣言（定義）が必要
  - メソッドの名前
  - メソッドがどのような命令を束ねているのかの記述
- メソッドの起動（呼び出し）
  - メソッドを1つの命令として実行すること
- メソッドの宣言と起動をしっかり区別すること

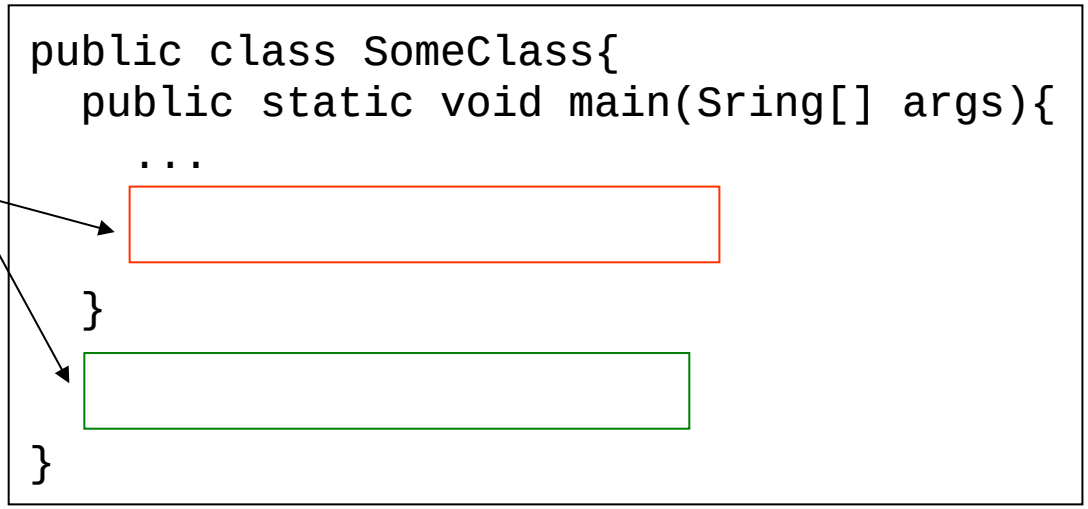
メソッドの宣言　最も単純な形

```
public static void <メソッドの名前> () {

   <メソッド本体>
   ここに命令文を並べる


}
```

メソッドの宣言を置く位置

- クラスブロックの中
  - クラスブロックの外にメソッドを宣言してはいけない
- 各メソッドの外側
  - メソッドの中に他のメソッドを宣言してはいけない





クラスブロックの外にあるとエラー

```
public class SayHello2 {
  public static void main(String[] args) {
    sayHello();
  }
}

// クラスブロックの外側にあるのでエラー
public static void sayHello() {
  System.out.print("Hello!");
  System.out.println("This is sayHello.");
}
```

mainメソッドの内側にあるのでエラー

```
public class SayHello3 {
  public static void main(String[] args) {
    sayHello();
    // main メソッドの内側にあるのでエラー
    public static void sayHello() {
      System.out.print("Hello!");
      System.out.println("This is sayHello.");
    }
  }
}
```

メソッドの起動

起動したい位置で

<メソッドの名前>();

```
public class SayHello {
  public static void main(String[] args) {
    sayHello();
    sayHello();
    sayHello();
  }

  public static void sayHello() {
    System.out.print("Hello!");
    System.out.println("This is sayHello.");
  }
}
```

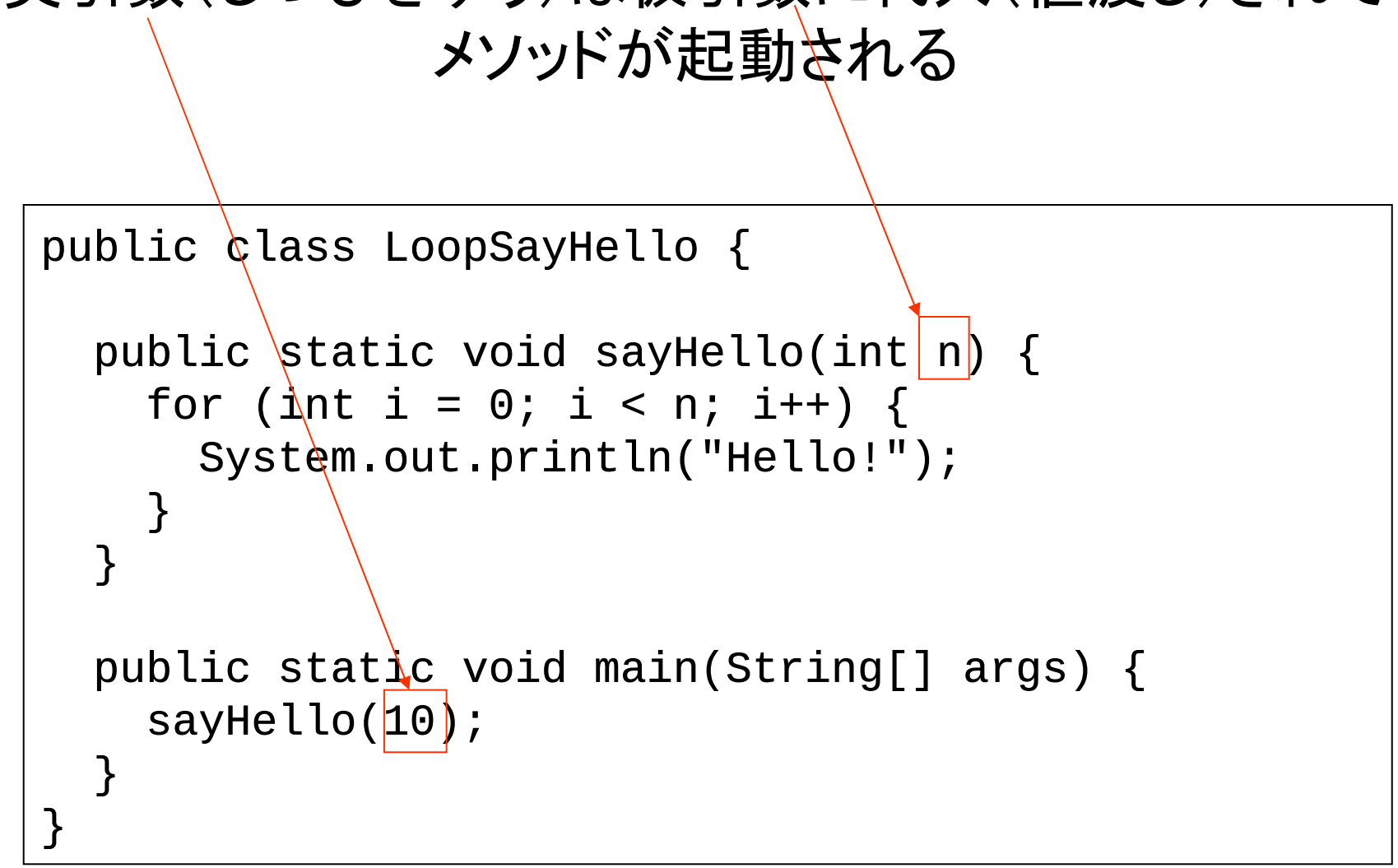
以下の質問に答えられますか？

- メソッドの宣言とは、起動とは何ですか
- メソッドの宣言はどのように書きますか
- メソッドの宣言はどこに置きますか
- メソッドの起動はどのようにしますか

引数（ひきすう、パラメータ）のあるメソッド

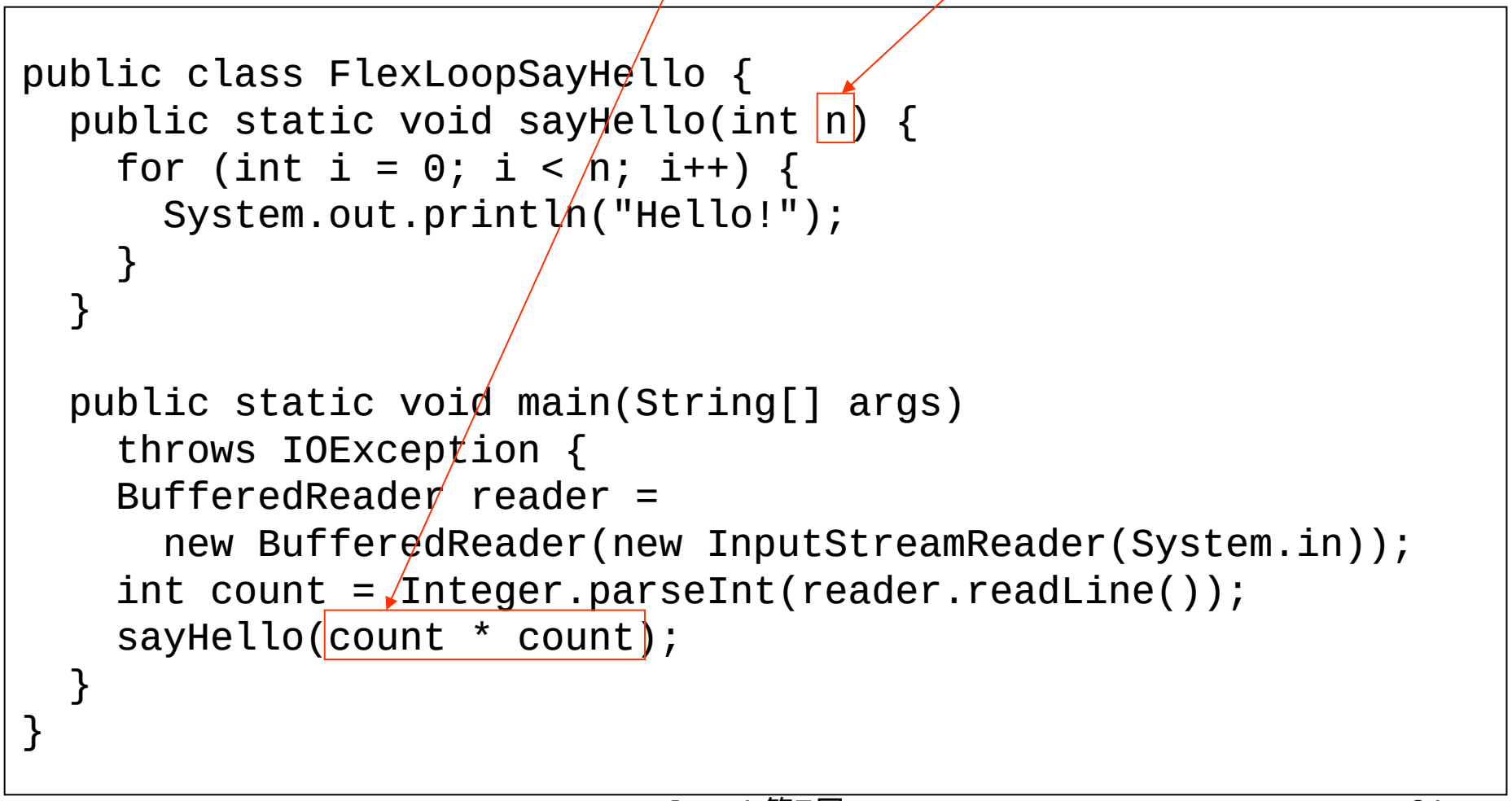
```
public class LoopSayHello {

  public static void sayHello(int n) {
    for (int i = 0; i < n; i++) {
      System.out.println("Hello!");
    }
  }

  public static void main(String[] args) {
    sayHello(10);
  }
}
```

実引数（じつひきすう）は仮引数に代入（値渡し）されてメソッドが起動される



```
public class LoopSayHello {

  public static void sayHello(int n) {
    for (int i = 0; i < n; i++) {
      System.out.println("Hello!");
    }
  }

  public static void main(String[] args) {
    sayHello(10);
  }
}
```

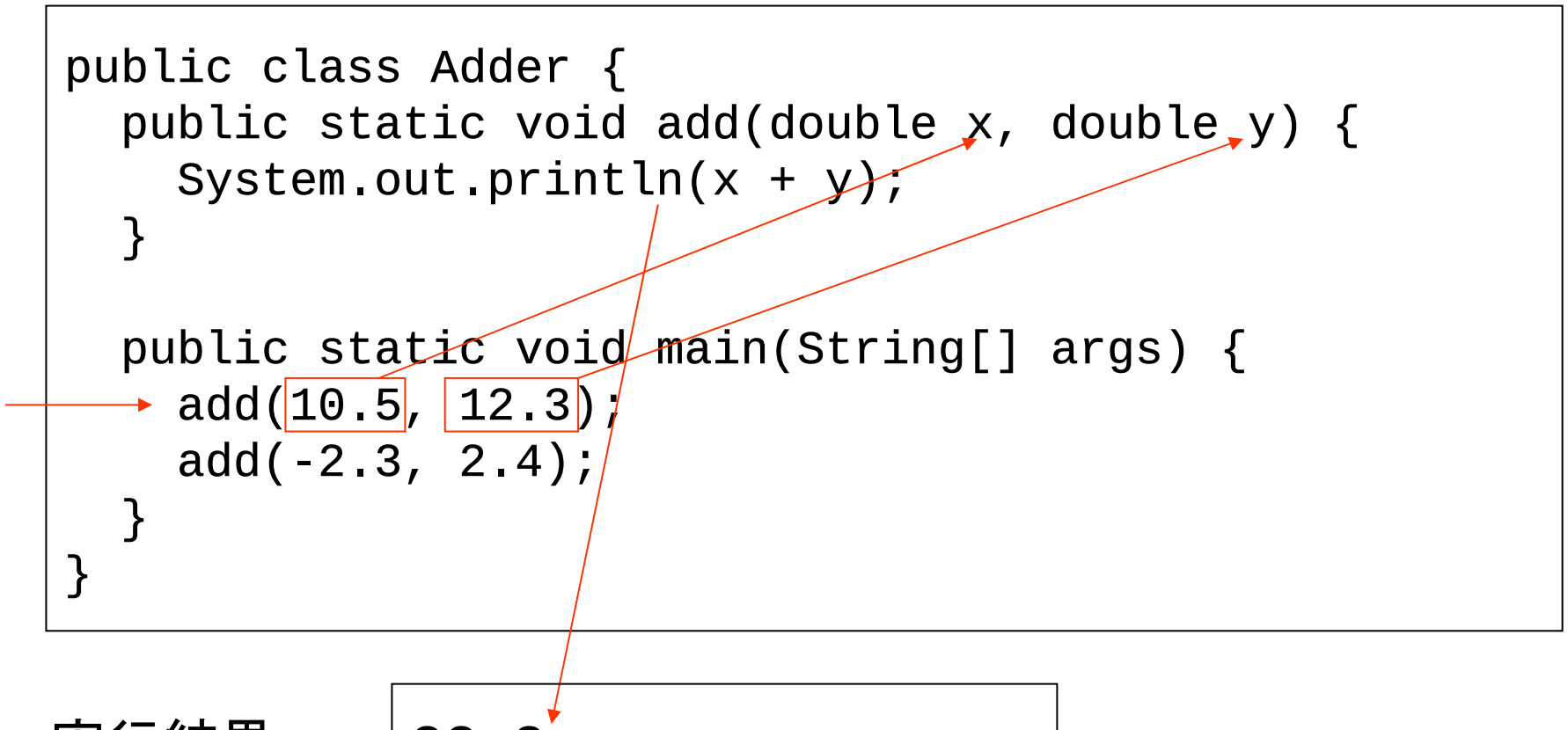
実引数には変数や式を書いてもよい
実際には変数や式の値が仮引数に渡される



```
public class FlexLoopSayHello {
  public static void sayHello(int n) {
    for (int i = 0; i < n; i++) {
      System.out.println("Hello!");
    }
  }

  public static void main(String[] args)
    throws IOException {
    BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
    int count = Integer.parseInt(reader.readLine());
    sayHello(count * count);
  }
}
```

引数は2つ以上でもOK
実引数は順序通りに仮引数に渡される

```
public class Adder {
  public static void add(double x, double y) {
    System.out.println(x + y);
  }

  public static void main(String[] args) {
    add(10.5, 12.3);
    add(-2.3, 2.4);
  }
}
```



実行結果

```
22.8
```

実引数の個数と仮引数の個数は一致しなければならない

```
public class Adder {
  public static void add(double x, double y) {
    System.out.println(x + y);
  }

  public static void main(String[] args) {
    add(10.5, 12.3);
    add(-2.3, 2.4);
  }
}
```

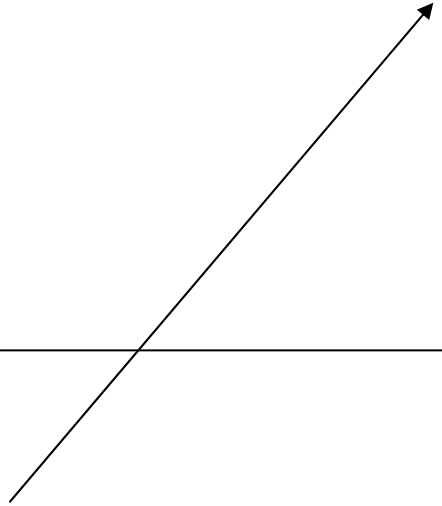
実行結果

```
22.8
0.10000000000000009
```

余計な桁はなぜでしょう？

引数をもったメソッドの宣言

```
public static void <メソッドの名前> (<仮引数>) {

   <メソッド本体>

}
```



あるいは「,」で区切られた複数の仮引数

以下の質問に答えられますか？

- メソッドの仮引数、実引数とは何ですか
- メソッドの起動にあたって実引数はどのようにして仮引数に渡されますか
- 引数つきのメソッドの宣言はどのようにしますか
- 引数つきのメソッドの起動はどのようにしますか

変数のスコープ（有効範囲）

```
for (int i = 0; i < 10; i++) {
 // ...
}
// ERROR: ここで変数 i は使用できない
System.out.println(i);
```

```
// i を for の外で宣言することによって、
// for の外側でも利用できるようにする
int i;
for (i = 0; i < 10; i++) {
 // ...
}
// これなら変数 i を使用できる
System.out.println(i);
```

ローカル変数のスコープ

- これまで使用してきた変数はすべてローカル変数
  - ローカルでない変数も近い将来勉強する
- スコープは宣言した場所から、その宣言が置かれている最も内側のブロックの末尾まで

パラメータ変数

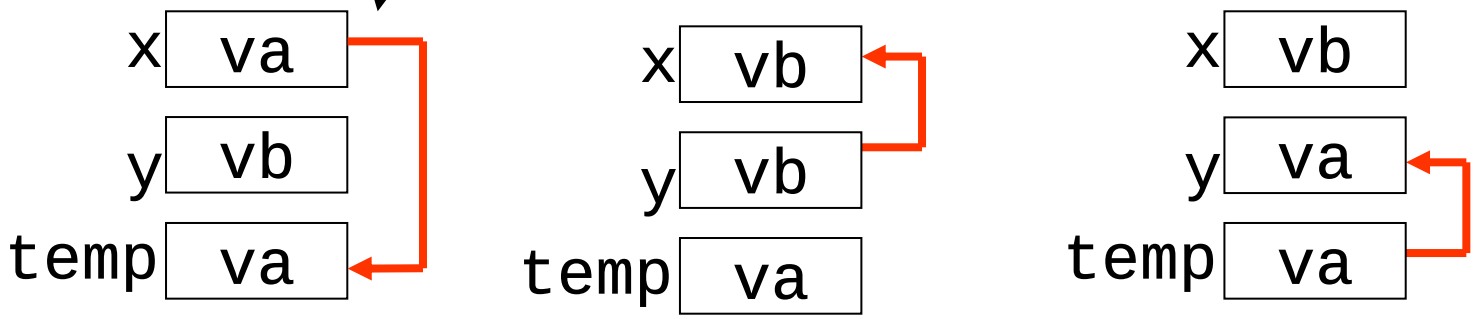
```
public class ParameterScope {
  public static void hoge(int foo) {
  // foo の有効範囲を開始
    ...
  } // foo の有効範囲を終了

  public static void main(String[] args) {
  // ここでは foo を使用できない (foo は有効範囲外)
    ...
  }
}
```

値渡し(call by value)

```
public class CallByValue {
  public static void main(String[] args) {
    int a = 10;
    int b = 20;
    swap(a, b);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
  }

  // 二つの変数の中身を交換する (値渡しをするので失敗)
  public static void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
  }
}
```

swapメソッドのやっていること
変数xとyの値を入れ替えている

```
// 二つの変数の中身を交換する（値渡しをするので失敗）
  public static void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
  }
}
```



実際にはswap(10,20)としてswapメソッドを起動

```
public class CallByValue {
  public static void main(String[] args) {
    int a = 10;
    int b = 20;
    swap(a, b);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
  }

  // 二つの変数の中身を交換する (値渡しをするので失敗)
  public static void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
  }
}
```

swapメソッドはコピーされた値を自分のローカル変数の間でやり取りするだけ

```
public class CallByValue {
  public static void main(String[] args) {
    int a = 10;
    int b = 20;
    swap(a, b);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
  }

  // 二つの変数の中身を交換する（値渡しをするので失敗）
  public static void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
  }
}
```

注意:ローカル変数のスコープ

名前は同じでも無関係な変数なので状況は変わらない

```
public class CallByValue {
  public static void main(String[] args) {
    int a = 10;
    int b = 20;
    swap(a, b);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
  }

  // 二つの変数の中身を交換する (値渡しをするので失敗)
  public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
  }
}
```

仮引数の名前をa,b
にしてみたが

以下の質問に答えられますか？

- ローカル変数のスコープとは何ですか
- ローカル変数の宣言とその宣言の置かれているブロックはスコープにどのような影響を与えますか
- メソッドの仮引数のスコープはどうなっていますか
- 値渡しとは何ですか

演習に入る前に

- 自動テストは大変な恩恵を与えてくれるが、自動テストをパスしたら、それで満足、というのではあなたは伸びません！
- 講義資料にのっているテストの詳細を熟読し、自分でテストを設計できるようになって欲しい
- そのための準備として、課題でプログラムを書いたら、自動テストだけに頼らずに、自分で考えたテストを実行すること

演習に入る前に
今週より「骨格テスト」の内容が変わる

- 今までは
  - クラス
    - クラスは存在するか
  - mainメソッド
    - mainメソッドは存在するか
    - public static void ... で宣言されているか
    - throws IOException が必要なら付いているか
- 今回から追加
  - main以外のメソッドについても正当性が検証される
- 骨格テストのタイミング
  - すべてのメソッドの宣言がなされた後

演習に入る前に
テストは骨格、単体、機能テストの３本立てに

- 今までは
  - 骨格テスト
  - 機能テスト
- 今回から
  - 骨格テスト
  - 単体テスト
    - 各メソッドを単独で起動したときの動作を確認する
  - 機能テスト

演習に入る前に

- Print3MethodとPrintMultという2つのプログラムを各自作成し実行、テストまでを一斉にやる
- 頃合をみて2つのプログラムの詳細な解説をするが
- 講義資料の解説を見ながらその意味を自分でよく考えること
- この2つのプログラムの意味をきちんと理解できた人だけが課題に進んでよい

一緒にやってみよう

- 今回の演習で使うテストドライバをいつものようにインストールする
  - ライブラリのアップデートがあるので手順を正確に実行すること
  - テストドライバの導入に成功すると
    - プロジェクト「java2007」の中の「test」というフォルダに「j1.lesson07.xml」という名前のファイルが作成される。
    - このファイルには今週使用するテスト一式が記述されている。
- j1.lesson07 というパッケージを作成する
- 講義資料にあるPrint3Method, PrintMultというプログラムを、このパッケージに作成する
  - 講義資料にある手順でテスト、実行までやること

Print3Methodの解説

```
public class Print3Method {
  public static void main(String[] args) {
    printHello();
    printHello();
    printHello();
    printBye();
  }

  public static void printHello() {
    System.out.println("Hello!");
  }

  public static void printBye() {
    System.out.println("Bye.");
  }
}
```

メソッドprintHello, printByeを宣言
宣言しただけでは実行されない

```
public class Print3Method {
  public static void main(String[] args) {
    printHello();
    printHello();
    printHello();
    printBye();
  }

  public static void printHello() {
    System.out.println("Hello!");
  }

  public static void printBye() {
    System.out.println("Bye.");
  }
}
```

printHelloメソッドを3回起動する

```
public class Print3Method {
  public static void main(String[] args) {
    printHello();
    printHello();
    printHello();
    printBye();
  }

  public static void printHello() {
    System.out.println("Hello!");
  }

  public static void printBye() {
    System.out.println("Bye.");
  }
}
```

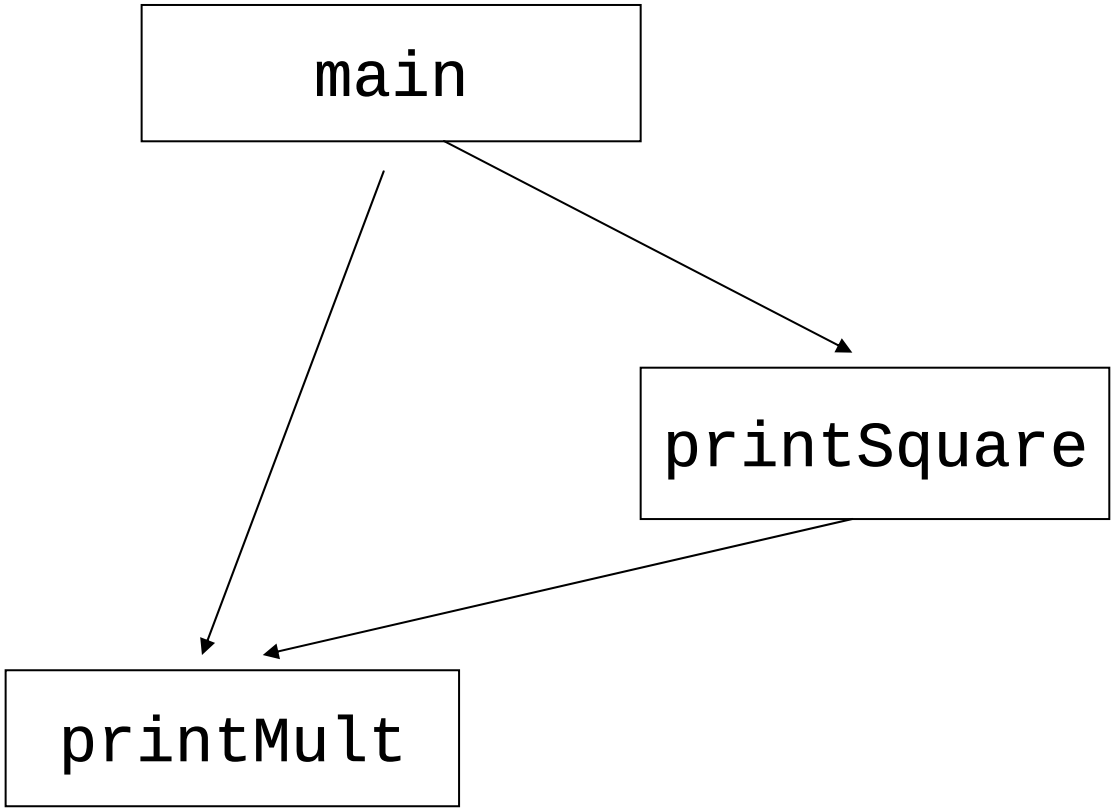
printByeメソッドを1回起動する

```
public class Print3Method {
  public static void main(String[] args) {
    printHello();
    printHello();
    printHello();
    printBye();
  }

  public static void printHello() {
    System.out.println("Hello!");
  }

  public static void printBye() {
    System.out.println("Bye.");
  }
}
```

PrintMultの解説

```
public class PrintMult {
  public static void main(String[] args) {
    for (int i = 1; i <= 9; i++) {
      printMult(10, i);
    }
    for (int i = 1; i <= 9; i++) {
      printSquare(i);
    }
  }

  public static void printMult(int a, int b) {
    int c = a * b;
    System.out.println(a + " * " + b + " = " + c);
  }

  public static void printSquare(int a) {
    printMult(a, a);
  }
}
```

メソッド間の呼び出し関係



printMultメソッドを9回起動

```
public class PrintMult {
  public static void main(String[] args) {
    for (int i = 1; i <= 9; i++) {
      printMult(10, i);
    }
    for (int i = 1; i <= 9; i++) {
      printSquare(i);
    }
  }

  public static void printMult(int a, int b) {
    int c = a * b;
    System.out.println(a + " * " + b + " = " + c);
  }

  public static void printSquare(int a) {
    printMult(a, a);
  }
}
```

起動

実引数から仮引数への受け渡し

printSquareメソッドを9回起動

```
public class PrintMult {
  public static void main(String[] args) {
    for (int i = 1; i <= 9; i++) {
      printMult(10, i);
    }
    for (int i = 1; i <= 9; i++) {
      printSquare(i);
    }
  }

  public static void printMult(int a, int b) {
    int c = a * b;
    System.out.println(a + " * " + b + " = " + c);
  }

  public static void printSquare(int a) {
    printMult(a, a);
  }
}
```

起動

実引数から仮引数への受け渡し

printSquareからprintMultを起動

```
public class PrintMult {
  public static void main(String[] args) {
    for (int i = 1; i <= 9; i++) {
      printMult(10, i);
    }
    for (int i = 1; i <= 9; i++) {
      printSquare(i);
    }
  }

  public static void printMult(int a, int b) {
    int c = a * b;
    System.out.println(a + " * " + b + " = " + c);
  }

  public static void printSquare(int a) {
    printMult(a, a);
  }
}
```

実引数から仮引数への受け渡し

起動

課題

各自のペースで
「第07週目の課題」をやってみよう

課題0701のヒント

```
main
      入力のための準備
      a,b,c を入力する
      heron を a,b,cを実引数として起動する

heron 引数を3つとる
      面積を計算して結果をプリントする
```

課題0702のヒント

```
main
    入力のための準備
    入力する(入力の適切性をみる)
    入力が適切なら
        入力された整数を実引数として printDivisors を起動する

printDivisors(n)  // n の約数を小さい順にプリントする
    1 をプリント
    for(i を 2 から n まで走らせる)
        n を i で割って割り切れたら
            スペースと i をプリント
    改行する
```