Java演習
第2回 (2024/4/17)

横山大作

講義前・後の質問

- 横山まで
- dyokoyama@meiji.ac.jp

今回の内容

- 課題1の解説

- 配列の使い方
  - new
- クラスの基礎
  - クラスとインスタンス
  - フィールドとメソッド
  - インスタンスの作り方、使い方
  - 参照型の変数

前回の提出課題1: ループと配列（再掲）

- javalec1というパッケージを作り、Looptestクラスを作ってください
- main()メソッドの中に、numsという大きさ10のintの配列を準備してください
- numsに、3でも5でも割り切れない正整数を小さい数から（1から）順に10個入れてください
- numsの数を「0番目+9番目」「1番目+8番目」…のように、両端からペアにして足して、その値を1行に1個ずつ表示してください

まず、配列準備まで

```
package javalec1;

public class Looptest {
    public static void main(String[] args) {
        int[] nums = new int[10];



    }
}
```

- 配列は宣言して、newして作る

配列の中身（解答例）

- 3でも5でも割り切れない数を10個
  - whileループが書きやすいかな？

```
int n = 0;
int x = 1;
while (n < 10) {
    if (x % 3 != 0 && x % 5 != 0) {
        nums[n] = x;
        n++;
    }
    x++;
}
```

  - 別の書き方でも書けるかな？練習してみよう
    - ちょっと面倒かもね

出力（解答例）

```
for (int i = 0; i < 5; i++) {
    System.out.println(nums[i] + nums[9-i]);
}
```

- これは色々書き方がありそう

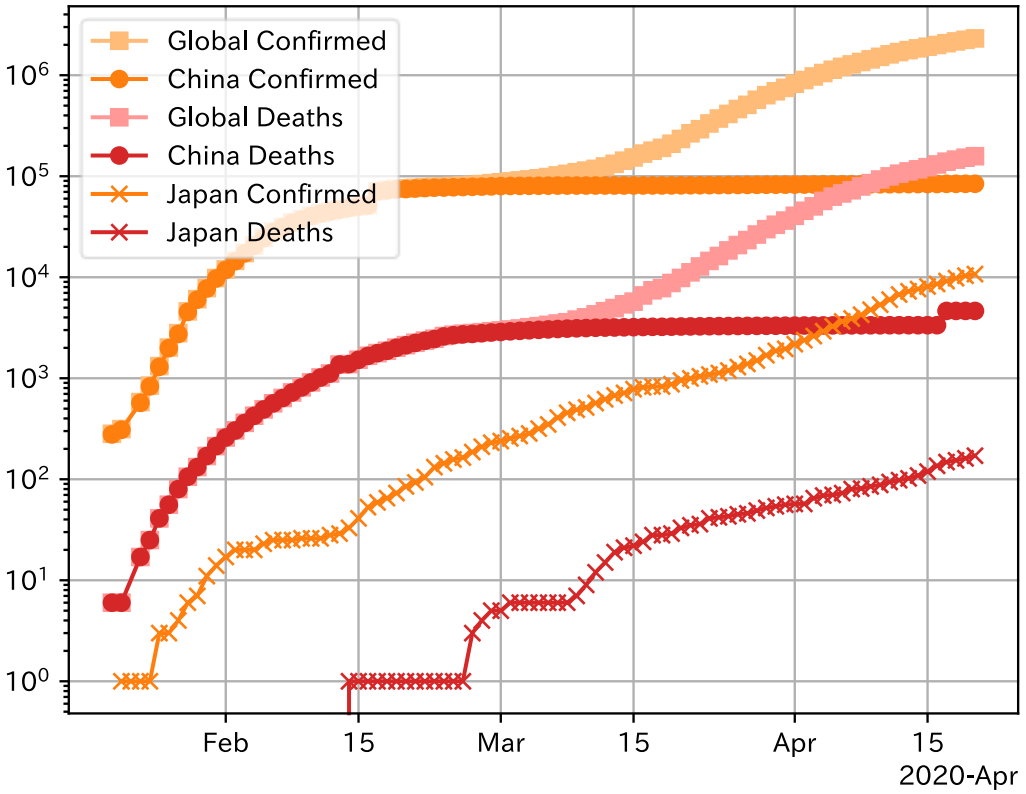
配列

同じものを固まりにする

- プログラムの基本は「同じものを見つける」こと
  - 設計のキモ
  - オブジェクト指向のキモ

同じものはどれ？

- どんなデータがある？
- データにはどんなまとまりがある？
- 一番細かいまとまりは？



三重大奥村先生: WHO日報のCOVID-19感染状況グラフ
https://oku.edu.mie-u.ac.jp/~okumura/python/COVID-19.html

変数名

- まとまったものは普通、まとまった名前にする
- 4月1日のデータ、4月2日のデータ…について
  - int data_20200401 = 3;
  - int data_20200402 = 5;

  としていたら普通は辛い

配列 (See p.142)

- まとめて処理する変数はまとめておきたい
  - 毎月支出したお金の一覧とか
- まとめたものを「配列」として、まとまったものに名前を付ける
  - []を付けると「～の配列」という型の変数になる
  - int[] a; はまとまったもの全体にaという名前がつく
  - 1個1個にはa[0], a[1],…という名前がつく

```
int[] a;
a = new int[3];

a[0] = 8;
a[2] = 10;

System.out.println(a[2]);
```

配列 (See p.142)

```
int[] a;
a = new int[3];

a[0] = 8;
a[2] = 10;

System.out.println(a[1] + a[2]);
```

- 配列も「型」なので、変数宣言して使う
- 配列は作る時に “new” が必要
  - 今までのintとはちょっと違う。
  - 「オブジェクト」の仲間
- 1個1個の要素は元の型
  - intとして代入や演算、利用ができる

new

- データをメモリ上に「作る」
- 変数宣言の時点では「この型のデータをこういう名前で呼ぶ」と宣言しているだけ
- データを実際に保存する場所を作る必要がある
- 今までのintやdoubleは実はJavaでは特別扱いされていた
  - 宣言と同時にデータの場所が確保されていた感じ
- 配列は特別扱いされないので、newが必要

初期化

```
int[] a;
a = new int[3];

a[0] = 8;
a[3] = 10;

System.out.println(a[1] + a[2]);
```

- 配列はnewするときに大きさを決める
- 配列は自分の大きさを覚えている

- 範囲を超えたアクセスがあると実行時にエラーが起きる
  - 例外: Exception というものが起きる
  - 動かして見てみよう。エラーメッセージはわかりやすい。

初期化(2)

```
int[] a = new int[] {3,6,9};
```

```
int[] a = {3,6,9};
```

- 初期値を与えるための簡単な書き方がある
- 上の書き方は宣言とnew、初期値の代入を一緒にしている
  - 配列の長さは自動的に決められる
- この場合、さらにnewを省略できる（下の書き方）
  - やっていることは同じ、コンパイラが気を利かせている

初期値

- Javaは初期値が必ず入る
  - intならば0
  - booleanならばfalse
- （参考: C言語では初期化されず、不定なゴミの値が入っていた）

- 右だとa[1]には
  必ず0が入っている

```
int[] a;
a = new int[3];

a[0] = 8;
a[2] = 10;

System.out.println(a[1] + a[2]);
```

配列の長さ

- 配列の長さは配列自身が知っている
- 配列名.length という変数に入っている
- これを使って繰り返しなどが書ける

```
int[] a;
a = new int[3];

for (int i = 0; i < a.length; i++) {
  System.out.println(a[i]);
}
```

- .lengthのように、固まりの「属性」が . で使えるようになっているのがJavaのオブジェクト指向の機能

様々な配列

- どんなデータ型も配列にできる
  - double[] ddata;
  - String[] sdata;
  - boolean[] bdata;

  - sdata = new String[3];
  - sdata[0] = “zero”;

- 今後、自分の作ったデータ型（クラス）も配列にできる

クラスの基本

クラスとは

- 関連するデータの組をひとまとめにして
- 関係する処理もひとまとめに扱うと
- プログラミングが楽になるよね、という考え

- ひとまとめにする固まりを「クラス」と呼ぼう

- （今日のところはこんな素朴な理解で）

Cにもあった：構造体

- データの組をひとまとめにして扱う機能

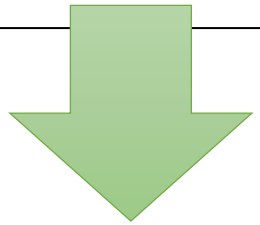
- 例: 座標 (int x, int y)
- 座標プラス名前 (int x, int y, String name)
- プラス評価 (int x, int y, String name, double score)

- データを扱おうとするとき、複数の属性をまとめて扱いたい場面は多い
  - 学生：学籍番号、メールアドレス、クラス…
  - 商品：価格、重さ…
  - 店：場所、名前、平均価格…

Cの構造体とJavaのクラスはよく似てる

- {}の中がメンバー
- メンバーは型と名前がある
- Storeという構造は
  - xという名前のint
  - yという名前のint
  - scoreという名前のdouble
- 使うときは普通の型と同様
  - intとかdoubleとかと同じように使う
- メンバーは”.”でアクセスできる

```
struct Store {
  int x;
  int y;
  double score;
};
```

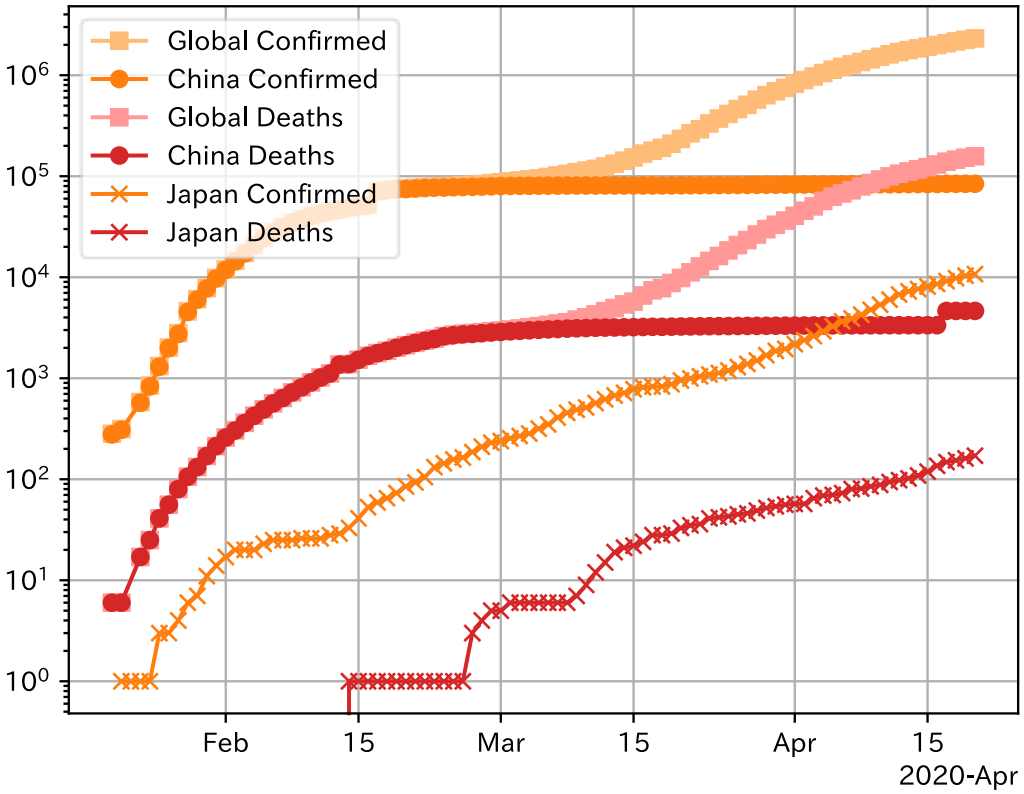


```
class Store {
  int x;
  int y;
  double score;
}
```

```
int func() {
  Store s1, s2;

  s1 = …
  int a = s1.x;
}
```

再掲：同じものはどれ？

- まとめるべきデータはある？



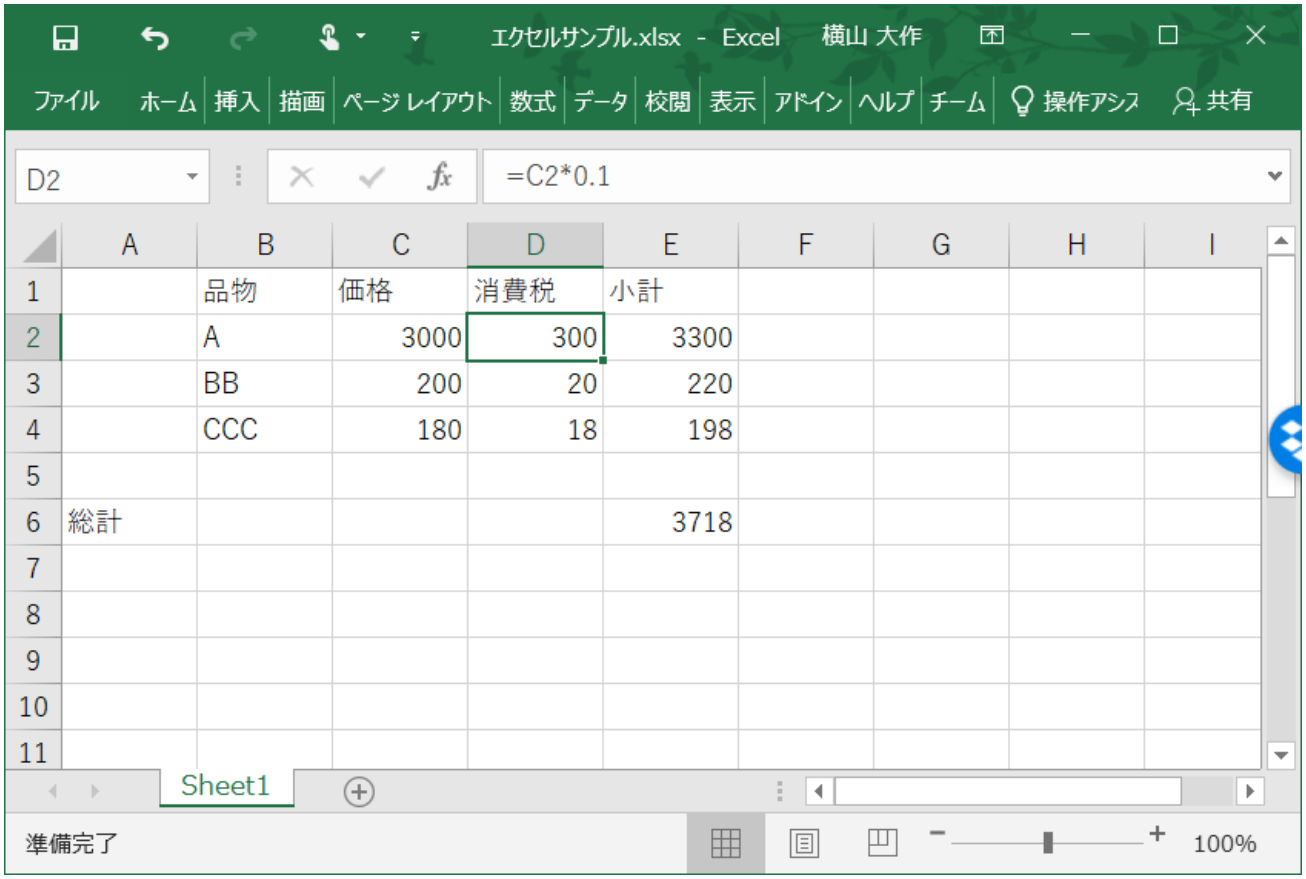
三重大奥村先生: WHO日報のCOVID-19感染状況グラフ
https://oku.edu.mie-u.ac.jp/~okumura/python/COVID-19.html

ところで...2つの世界がある

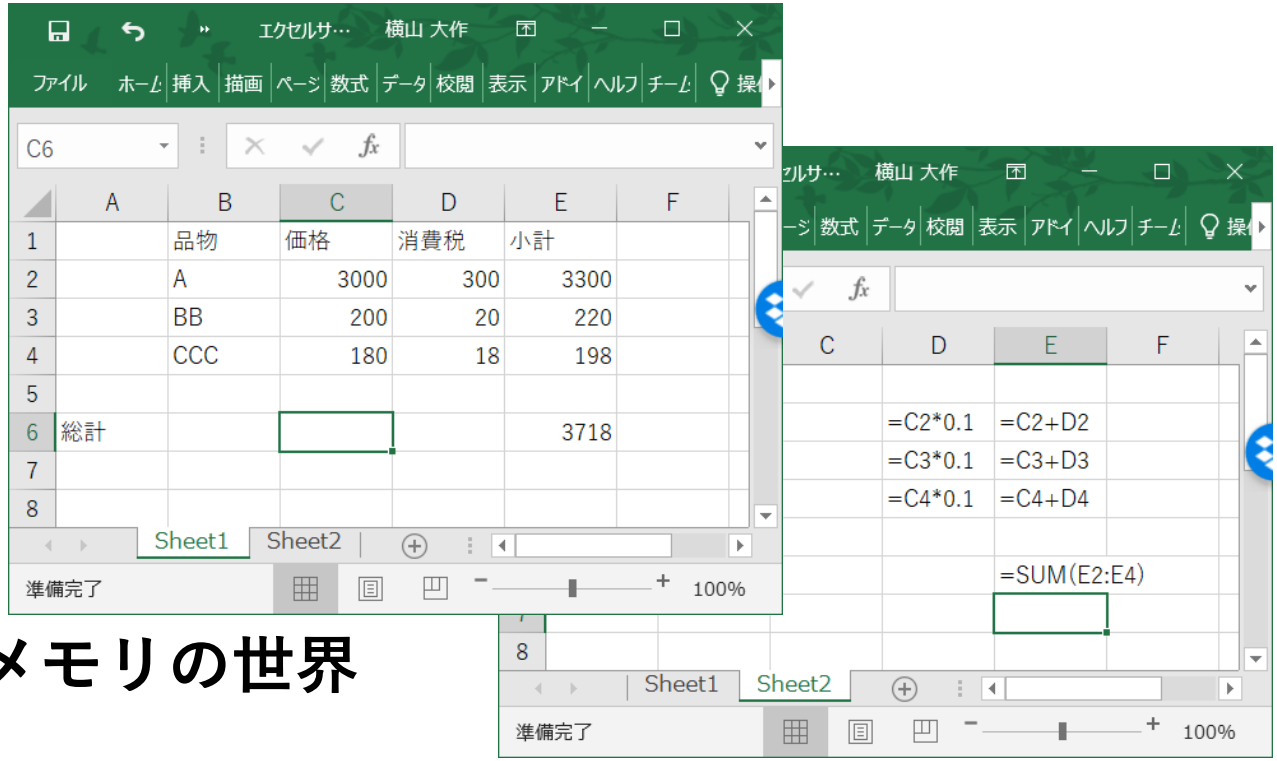
- プログラムの「定義」を書いてある世界
  - ソースコードの世界
- プログラムの「実体」「データ」が置いてある世界
  - メモリの世界

エクセルなら

- セルはメモリ
- セルをクリックすると計算式入れられる



2つの世界が重なっている



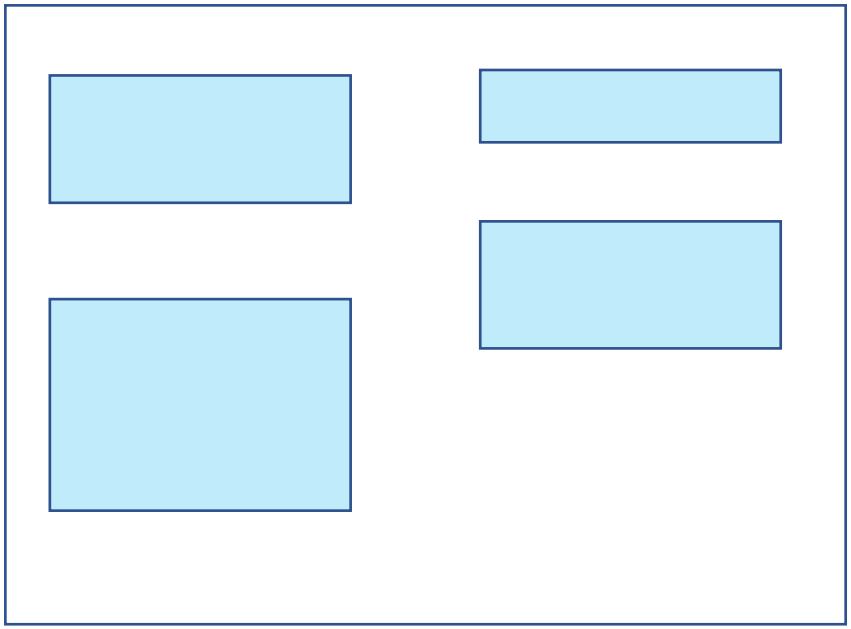
メモリの世界

手続きの世界

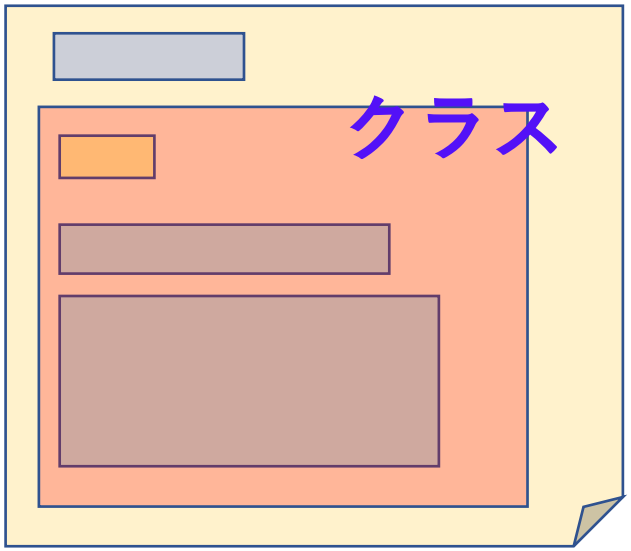
- メモリの世界にデータを置く
- 手続きの世界の計算結果もメモリに置かれる
- 普段はメモリが見えている
- クリックしたセルだけ穴が開いて手続きの世界が見える

Javaも同じ

- メモリの世界にデータがある
- それを定義したり、使ったりするのがソースコードの世界



メモリの世界（＝実体）



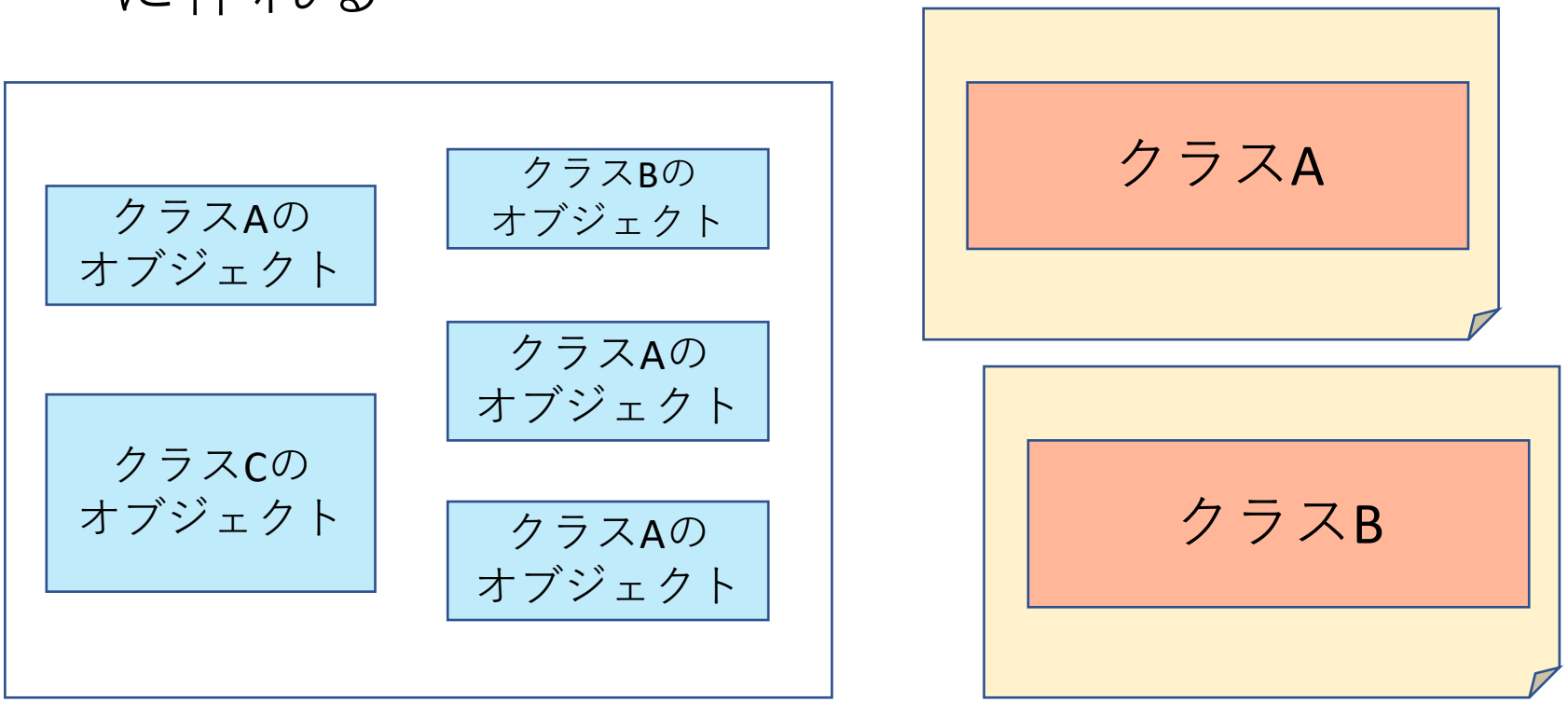
ソースコードの世界

クラスとオブジェクト

- データをまとめて扱うための機能がクラス
- class Store {…} で書いているのは「定義」
  - エクセルでいうと「手順」の世界のもの
- 「定義」に従った「実体」がメモリ上にあるはず
- 実体のことを「**オブジェクト**」あるいは「**インスタンス**」と呼ぶ
  - オブジェクトは比較的広い意味で使う
  - インスタンスは「実体」の意味でしか使わない

オブジェクトはたくさん

- クラスは定義、1つだけ
- それに従ったオブジェクトはたくさんメモリ上に作れる



メモリの世界（＝実体）

ソースコードの世界

メソッド

- まとめたデータに強く関連する手続きがある
  - 座標: 原点からの距離を計算する、x軸からの角度を計算する…
  - 店: 点数を3段階に変換して表示する…
- データをまとめるのと同時に、関連する手続きもまとめてしまうと良いのでは？
  - 手続き: Cでは「関数」と呼んでいた
  - クラスにくっついた関数を「メソッド」と呼ぶ

メソッドの基礎 (See p.176)

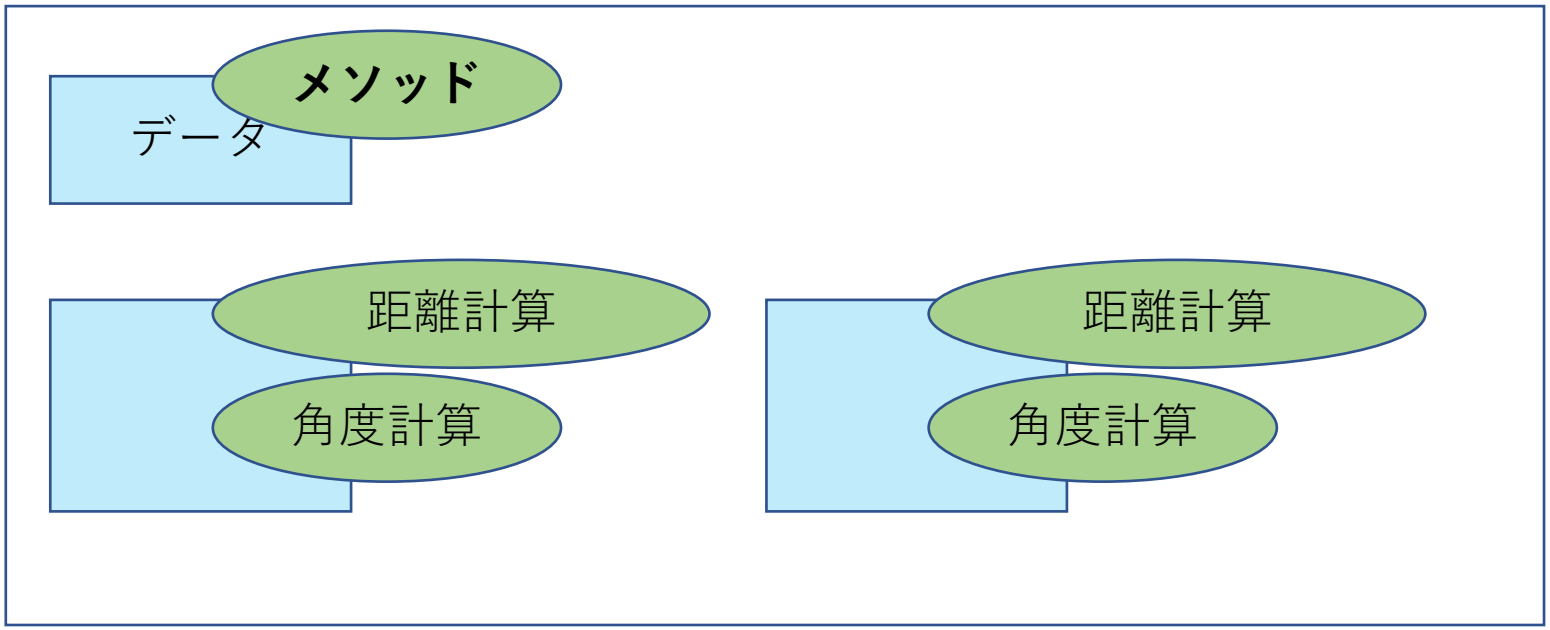
- 関数なので
  - 返り値の型
  - 関数の名前
  - 引数（型と名前）

  で決まる
- 関数でやりたい処理を{}の中に書く
  - 修飾子についてはまた後日

```
public class Sample {
  int f(int x) { return x + 3; }
}
```

メソッドはデータにくっつくことが可能

- データに関連した関数がまとまっている
- データに関数がくっついていればよい
- オブジェクトはデータと処理がくっついた、完結した存在（を目指したもの）



クラスの定義(See p.291)

- クラスの中には
  - フィールド: まとめて扱うデータの組
  - メソッド: まとめて扱う関数の組

  を書く

- フィールドがメモリ上のデータを定義する
- メソッドはデータにくっつく
  - くっついているデータを処理する関数、という性質が強い
  - くっついているデータを読み書き可能

フィールドとメソッド

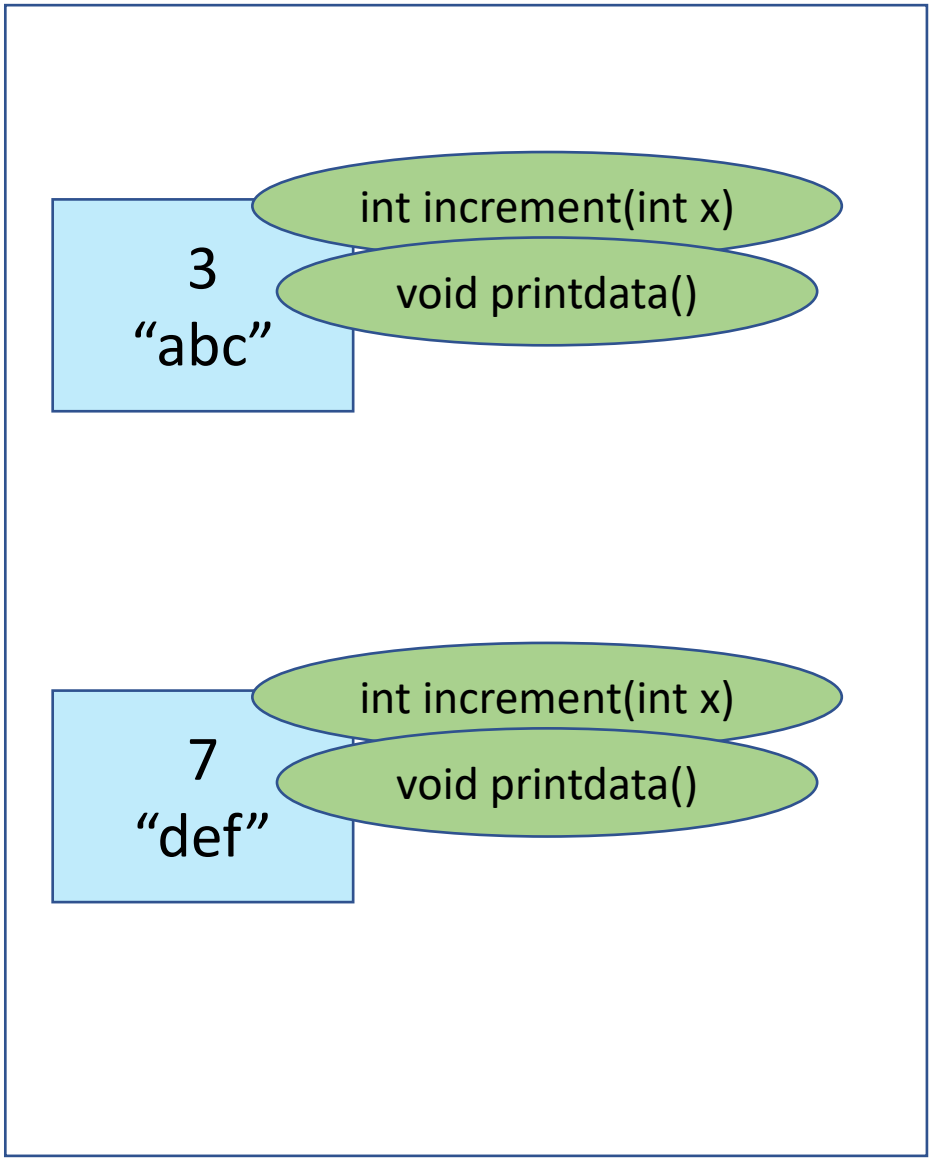
```
public class PairData {

  int count;
  String name;

  int increment(int x) {
    count += x; return count;
  }

  void printdata() {
    System.out.println(name + count);
  }
}
```

クラス

フィールド

メソッド

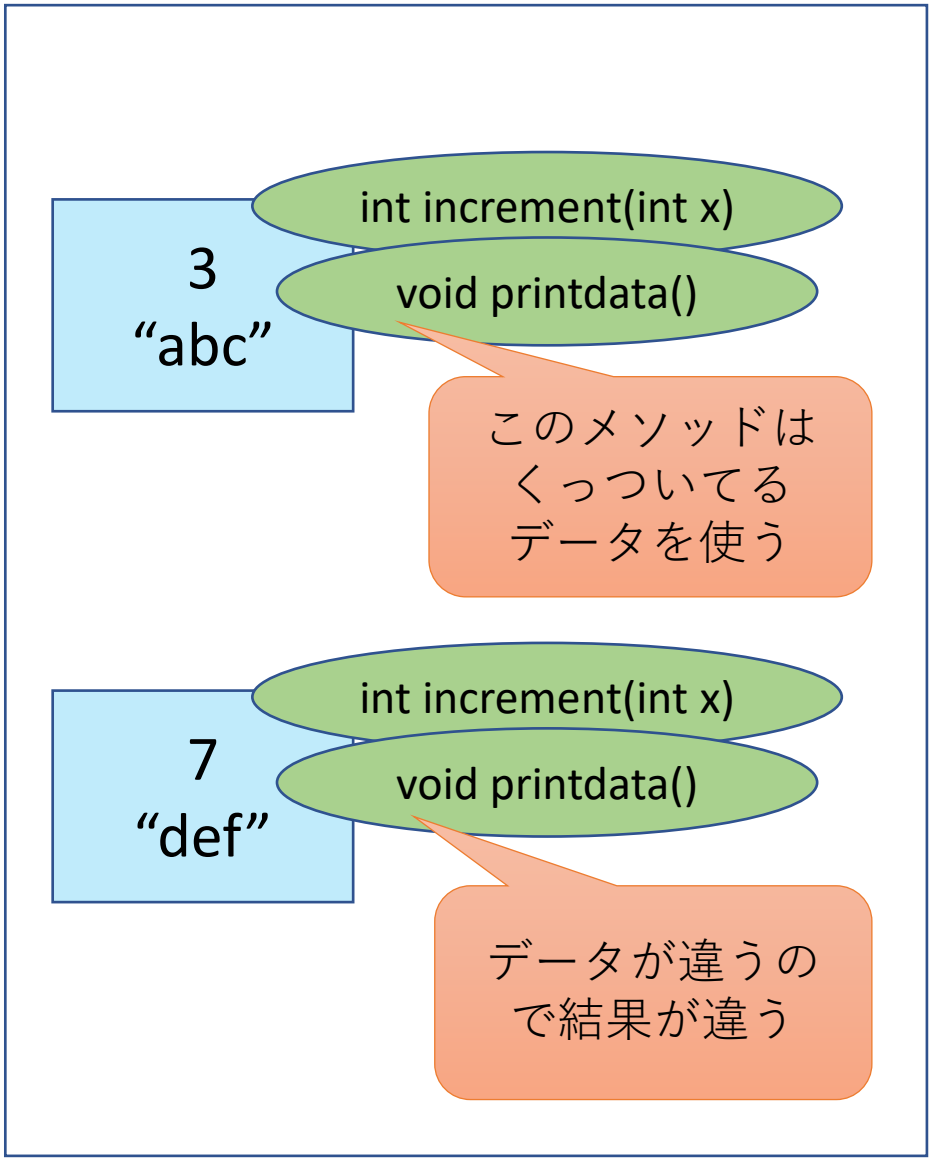
イメージしよう



クラスは1つ、
インスタンスは
たくさん

```
public class PairData {

  int count;
  String name;

  int increment(int x) {
    count += x; return count;
  }

  void printdata() {
    System.out.println(
      name + count);
  }
}
```

メソッドはインスタンスにくっつく




```
public class PairData {

  int count;
  String name;

  int increment(int x) {
    count += x; return count;
  }

  void printdata() {
    System.out.println(
      name + count);
  }
}
```

このname, count
はくっついてる
メモリ上のもの

クラスの使い方 (See p.315)

- クラスは型と同じもの
- 宣言し、実体を作って利用する
- 実体の作り方は “new”
  - new は前回、配列の時に使いましたね
  - int, double, booleanなど「以外」はすべてnewする、とJavaでは決めた

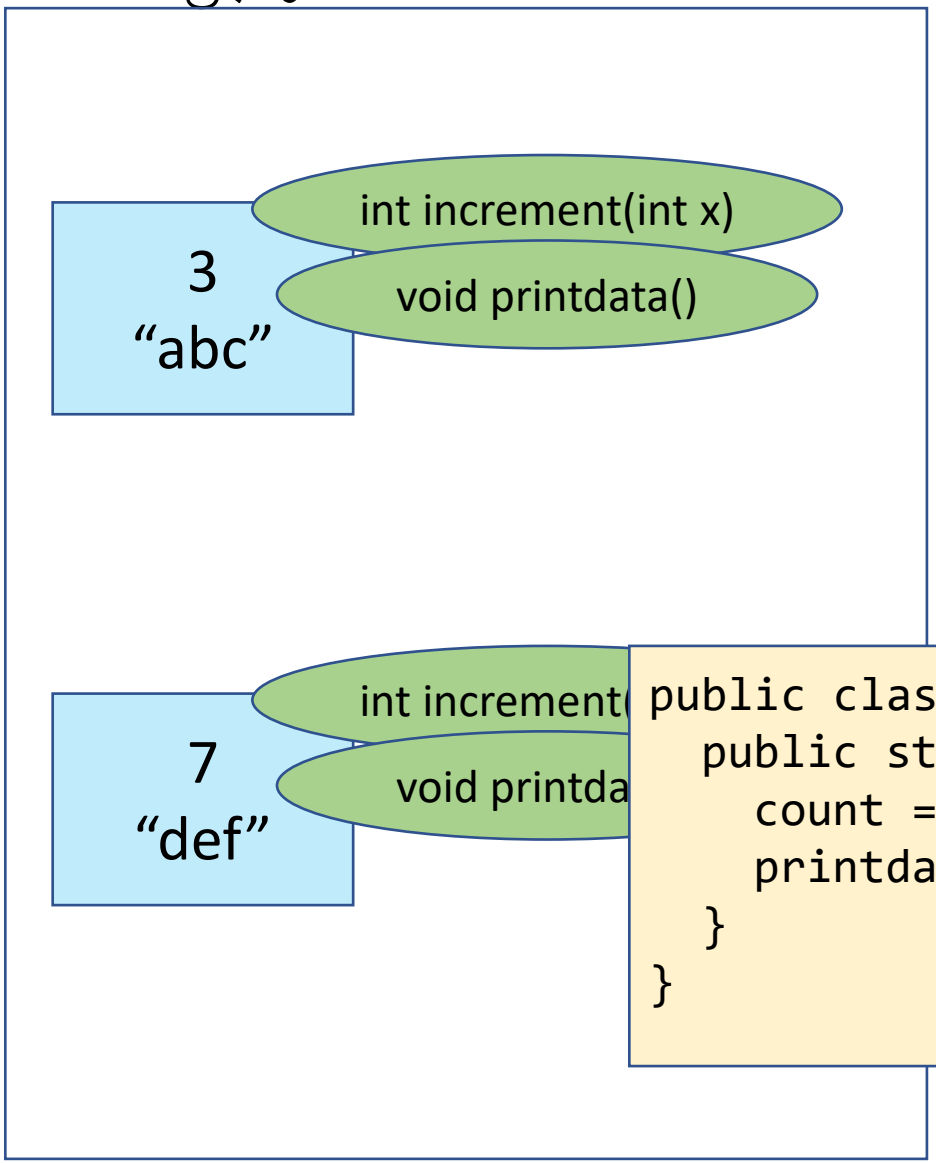
```
public class Sample {
  public static void main(String[] args) {
    PairData p;
    p = new PairData();
    p.name = ”abc”;
    p.count = 8;
    p.printdata();
  }
}
```

クラスの使い方 (See p.319)

- クラスの中に入っているデータへのアクセスは“.”
- クラスの中にまとめられたメソッドも“.”で呼べる

```
public class Sample {
  public static void main(String[] args) {
    PairData p;
    p = new PairData();
    p.name = ”abc”;
    p.count = 8;
    p.printdata();
  }
}
```

オブジェクトなしではメソッドは呼べない

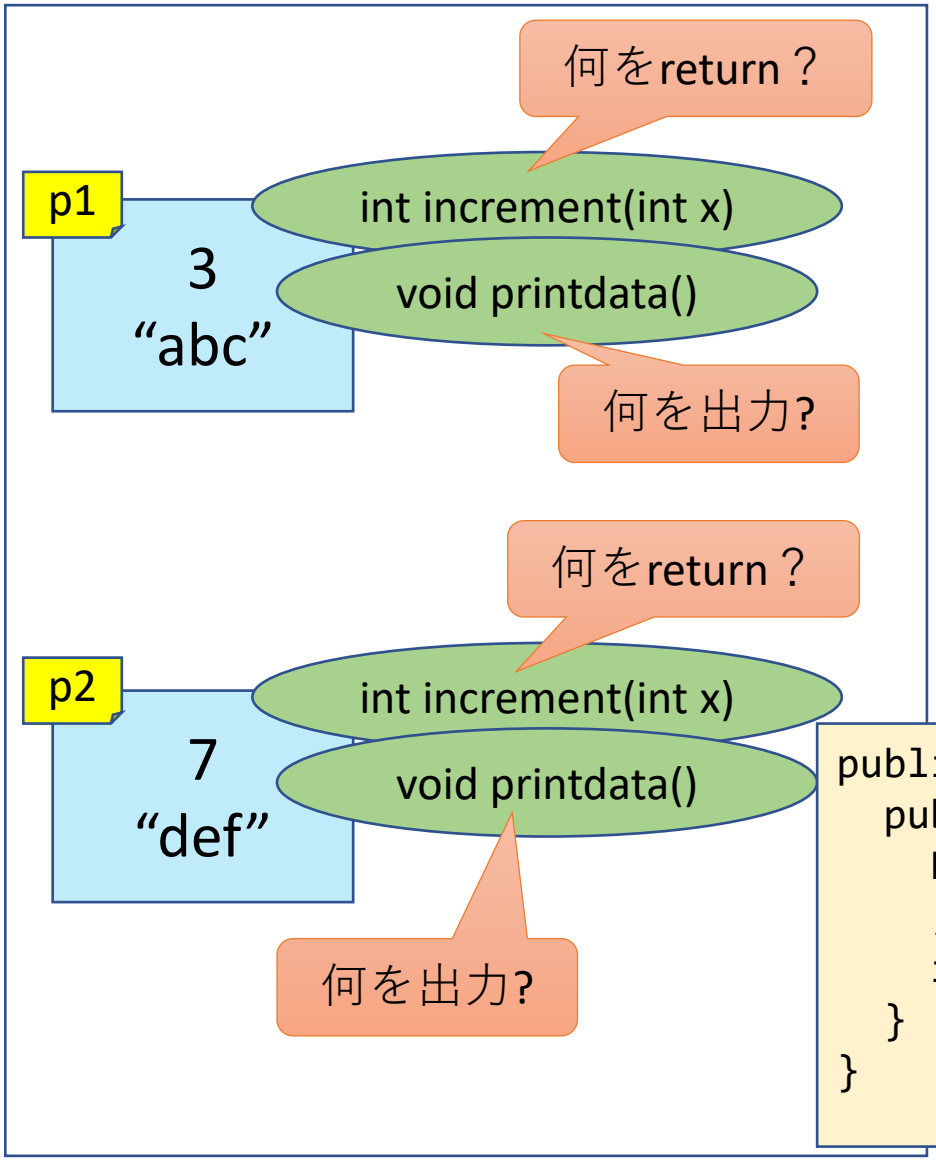



```
public class PairData {

  int count;
  String name;

  int increment(int x) {
    count += x; return count;
  }

  void printdata() {
```

```
public class Sample {
  public static void main(String[] args) {
    count = 10;
    printdata();
  }
}
```

どこのcount?
どこのprintdata()を呼ぼうとしている？

練習: 考えてみよう




```
public class PairData {

  int count;
  String name;

  int increment(int x) {
    count += x; return count;
  }

  void printdata() {
    System.out.println(
      name + count);
  }
}
```

```
public class Sample {
  public static void main(String[] args) {
    PairData p1 = new PairData();
    // 何か3,”abc”の初期化があって…
    int i = p1.increment(5);
  }
}
```

変数

- インスタンスを学んでの疑問：変数には何が入っている？
  - あるインスタンスのコピー？
  - ではない

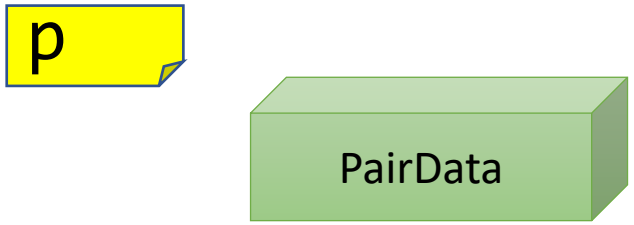
```
public class Sample {
  public static void main(String[] args) {
    PairData p = new PairData();
    p.count = 10;
    PairData q = p;
    q.count = 8;
  }
}
```

PairDataのインスタンスは何個できた？
pやqはメモリ上では何？

参照型(See p.317)

- Javaの変数は「原則」「参照型」と呼ばれるもの
  - 何かを「指し示す」もの
  - ここでは、付箋のようなものだと思ってみましょう
- 変数宣言は「この名前を使うよ」という付箋が作られる
- newではインスタンスが作られる
- 代入では、インスタンスに付箋が貼られる
- p.countみたいに使うときには、pの付箋が貼られたデータを探して、そのcountを見る

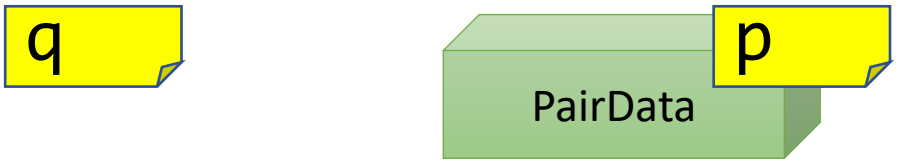
```
PairData p;
p = new PairData();
p.count = 10;
```



参照型(See p.317)

- 参照型の代入は「付箋を貼る」
  - q = p; は「同じインスタンスにqを貼る」
  - q.countは何か？

```
PairData p;
p = new PairData();
p.count = 10;
PairData q;
q = p;
q.count = 8;
```



参照型(See p.317)

- インスタンスができるのは　new　されるとき
  - クラスのオブジェクトや配列はnewで作っていた
- ちょっとだけ例外！intやbooleanなど
  - newしなくて使っていた変数たち
  - 基本データ型と呼ばれるものだけ、参照型ではない変数になっている
  - 変数の中に「実体」がそのまま入る
    - そのままコピーされる、とか、自分が分かりやすい理解をしてみてください
    - 変数を定義した瞬間に「実体」ができている

違い

```
PairData p;
p = new PairData();
p.count = 10;
PairData q;
q = p;
q.count = 8;
```

p.countはいくつ？

```
int p;
p = 10;
int q;
q = p;
q = 8;
```

pはいくつ？

- よく理解しよう

参照型 (See p.317)

- 参照型は、変数は「実体を指している」だけ
  - 使うときに毎回メモリを見に行っている、ようなもの
  - newしないと「実体」はない
- 実体を指すまでは null という特別な値が入る
  - 何も指していない、使えないもの

- C言語が得意な人は、ポインターだという説明でもわかるかな？

提出課題2: 配列とクラス

- javalec2というパッケージを作り、以下の2つのクラスを作ってください
  - Sequence : 課題でやりたいことをmainメソッドに持つクラスです
  - Rectangle : 四角形を表すクラスです
- それぞれの中身を、配布するソースのものにしてください
  - Sequence.java: mainの中が作りかけのひな形です。
  - Rectangle.java: 配布ファイルは完成形です
- Sequence.main()のメソッドの中に、pointsという配列があるのを確認してください
  - 数列$\{p_i\}$が与えられたと思ってください

課題（続き）

- 与えられた数列 $\{p_i\}$ の隣り合う2つの要素について、以下の処理を行ってください
  - 幅が$p_{i+1} - p_i$、高さが幅の2倍になるような四角形のインスタンスを作りなさい
  - そのインスタンスの面積をarea()というメソッドで計算して、改行付きで出力しなさい
- 結果は、数字が何行か出てくるものになるはずです
- 出力された結果と、組番号名前、をソースの頭にコメントとして付けてください
- Sequence.javaのみをoh-meijiで提出してください
- 〆切: 4/23(火) 17:00

ヒント

- 配列の要素を順番にfor文などで繰り返し処理すると良いでしょう
- 配列の長さは.lengthというフィールドに入っていました。使うと良いですね。
- 四角形のインスタンスの作り方は「クラスの使い方」の資料を参照してください
- javalec2というパッケージにあるすべてのソースコードの頭には

  package javalec2;

という行を入れる必要があります。

まとめ

- 配列の使い方
  - newして使えるようになる
- クラスの基礎
  - クラスとインスタンス
  - フィールドとメソッド
  - インスタンスの作り方、使い方
  - 参照型の変数
- このあたりのキーワードがごっちゃにならないように