Java演習
第4回

2024/5/8

横山大作

講義前・後の質問

- 横山まで
- dyokoyama@meiji.ac.jp

提出課題について

- 実行できないソースは出さないでください
  - 先頭の実行結果はコメントの中に入れる
- 先頭の実行結果は、整形せず、出力結果をそのまま貼ってください
  - 本当に実行できたつもりなのか、想定した結果なのかがよくわからないので
- コンパイルエラーや実行時エラーに注意して
  - 特に、実行時のエラーに気付いていないと思われる回答が多いです
- 返却したコメントに対する質問などあれば横山まで

この資料の内容

- オブジェクト指向高速入門
  - オブジェクト指向の大まかな狙いについて理解する
- コレクション高速入門
- 提出課題3

オブジェクト指向

オブジェクト指向(第II部)

- 第7章
- オブジェクト指向とは？→考え方
  - なるべく簡単に、間違いにくいプログラムを設計する方法
  - プログラミング・パラダイム
  - 非常に成功した(=役立つ)
- p.259
- 文法学習では「これが正解」を学んだ
- これからは「正解にたどり着くための考え方」を学ぶ

- P.260
  - 完全に一歩一歩理解するというより、何度も使っているうちに「肚落ちする」感じが良いでしょう

オブジェクト指向の方向性

- やりたいことを説明する絵を描いてみよう
- そこに出てくる「登場人物」「部品」に着目して整理していくやり方
  - p.267

思考の例: サッカーゲーム

- 対戦型サッカーゲームを作ってみよう
- どんなデータが必要？

- どんな処理が必要？

サッカーゲーム：機能中心設計

- ゲームを開始する
- ボールを前に進める
- ボールを奪う
- オフサイドラインを押し上げる
- ボールの位置を計算する
- 試合時間を進めて終わっていないかチェック

ボールの位置(x,y)

得点

PK中？

前後半・時間

キーパーの位置

キーパーの位置

プレイヤーの位置

プレイヤーの位置

プレイヤーの位置

プレイヤーの位置

プレイヤー
ミナ、
持技…

ごちゃ混ぜで
わかりにくい…

モジュール化

- 人間は複雑なものを一度に扱えない
- 一度に考えやすい程度の大きさがある

- 関数はその1つの現れ
  - 複雑すぎる動きを、部分的に切り出して実装
- 関数のまとまり、さらにそのまとまり…のように、考えるレベルを変えながらプログラムを作っていきたい

- 関数のまとまりの作り方の1つの考え方がオブジェクト指向
  - オブジェクト指向の目的の1つ

サッカーゲーム：オブジェクト指向

審判オブジェクト

PK中？

前後半・時間

得点

ゲームを開始する

試合時間を進めて終わっていないかチェック

だいぶ整理されたかも

ボールオブジェクト

位置

ボールの位置を計算する

プレイヤーオブ

プレイヤーオブ

プレイヤーオブ

プレイヤーオブジェクト

プレイヤーオブジェクト

位置、スタミナ、技能、特技…

ボールを前に進める

ボールを奪う

- すでに現実世界にあるものを作るときは考えやすい
- 必ず現実のものを再現しなければならないわけではない
  - もっと整理できることもある
- 目に見えないものを部品にしていくのは少し難しい
  - 関係とか
  - サッカーの例だと、審判オブジェクトが正しいか？ゲームオブジェクトだったりしないか？

オブジェクトと責務(p.271)

- オブジェクトには何かの「役」がある
- 担当する部分について「責任」を持つ

- 自分が使っている情報は自分が管理責任を持つ
  - 自分にくっついたフィールド・属性を持つ
- 自分の役目は自分でやる
  - 自分にくっついたメソッドで動く

オブジェクト指向のプログラム

- 役割を持ったたくさんのオブジェクトがコンピュータ上にある
- それらが互いに通信して、お互いに指示を出し合いながら動く
- 全体として何かの機能を実現するように動く

のがオブジェクト指向のプログラムの動作

オブジェクト指向言語とは

- オブジェクト指向をサポートする機能が付いている言語
- サポートなしでもオブジェクト指向的に書くことは可能
  - C言語でオブジェクト指向風に書くことも可能
  - 正直、面倒

- プログラミング・パラダイムは言語と密接に関連
- 新しい言語を学ぶことが難しいのではなく、新しいパラダイムを学ぶのが難しい
  - 言語が違うだけなら恐れる必要なし
  - Python, Ruby,…

クラス・インスタンス

クラスとインスタンス

- 違いは言えますか？

クラスは必要？(p.285)

- 1個1個のインスタンスを直接定義しても良いのでは？
- それもあり得る

- クラスはインスタンスの整理の仕方
  - こういう種類のインスタンス、とまとめたい場面が多い
- Java、C++など、多くのオブジェクト指向言語はクラスを使って整理している
- （参考）クラスを使わない考え方に、プロトタイプベースがある
  - JavaScriptが有名(若干混ざってるけど)

クラスとインスタンス

- クラスの基本を思い出しておきましょう(p.291)
- クラスを定義する方法は？
  - クラスの中には何を書けますか？
    - フィールド　（メンバ変数と呼ぶことも）
    - メソッド
- インスタンスを作る方法は？(p.301)
- インスタンスには何が入っていますか？
  - フィールド
- インスタンスを入れた変数は「参照型」
  - 実体への参照が入っていることに注意
  - p.319 変数を代入したときにどうなるか、自信をもって言えるように

インスタンス化

- newしてインスタンスを作ることを「インスタンス化」(instantiation)と呼ぶ

- (余談)「～でPointのオブジェクトをインスタンスする」と書いている文書を見た
- 動詞と名詞がごちゃごちゃ
  - 専門用語を日本語にするときに起きがち？
  - instanceは名詞、インスタンスという「もの」
  - instanceを作るのはinstantiate,「インスタンス化する」
  - 正しく使えないと「わかってない感」強い

オブジェクトは正常でいてほしい

- オブジェクトはなるべく完結した存在でありたい
- 生まれた瞬間から「正常状態」でいてほしい
  - 正しく初期化されていてほしい
- 初期化をするメソッドが必要

コンストラクタ(p.330)(復習)

- 引数なし: デフォルトコンストラクタ
- 引数付きもあり
  - 引数の型でどのコンストラクタが動くかが決まる
    - オーバーロードと呼ぶ

```
class Person {
  String name;
  int age;
  Person() { name = “default”; age = 0; }
  Person(String iname) { name = iname; age = 0; }
}
```

オーバーロード(復習)

- 同じ名前で引数が違うメソッドを定義できる
- メソッドの「オーバーロード」と呼ぶ

- メソッドを特定するもの：シグネチャ
  - メソッド名と引数の型の並び
  - 返り値の型は無視されるのに注意

```
class Person {
  void showInfo() {
    System.out.println(name + " " + age + "才");
  }
  void showInfo(String msg) {
    System.out.println(msg + ":" + name + " " + age + "才");
  }
}
```

フィールドへの代入

- コンストラクタやセッタを書くときに、引数の変数名がフィールドと被ることがよく起きる
  - 引数の名前の方が優先
- フィールド側にthis.を付けて明示する
  - thisは「インスタンスとしての自分」を指す

```
class Person {
  String name;
  int age;
  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

デフォルトコンストラクタ (see p.341)(復習)

- コンストラクタを定義しないときは、「何もしない」コンストラクタが暗黙のうちに作られる
  - フィールドは決まった初期値（0やnull）が入る

- コンストラクタを1個でも定義すると、デフォルトコンストラクタは消える
  - 自分でコンストラクタを定義すると、今までデフォルトコンストラクタで動いていたところがエラーになったりして困惑しがち

コンストラクタを明示的に呼ぶ(p.342)

- コンストラクタの中で別のコンストラクタをthis(引数)で呼べる
- 一番詳しい（引数の多い）コンストラクタを作って、引数が少ないコンストラクタはそれを呼ぶ形で定義することが多い

```
class Person {
  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  Person(String name) {
    this(name, 99);
  }
}
```

引数付きのコンストラクタ

- 「このフィールドは必ず正しい値を入れてほしい」という意図が感じられる
- 「この情報がないとオブジェクトとして意味をなさない」ものはコンストラクタで指定すると良い

フィールドの初期値

- メンバ変数（フィールド）は初期値がある
  - ローカル変数は初期化されていないという違いがある
- コンストラクタが必ず動く、ということ
  - デフォルトコンストラクタが作られる

```
class Person {
  String name;
  int age;
}
...
void func() {
  int i;
  Person p;
  System.out.println(i); // コンパイルできない
  System.out.println(p); // コンパイルできない
  p = new Person();
  System.out.println(p.age); // 0に初期化済み
}
```

配列の要素も初期化済み

- 配列の要素は初期化されている
  - 配列はクラスと近いふるまいをしている
- 配列をnewすることと、配列の要素1つ1つをnewすることは違うことを理解しよう

```
class Person {
  String name;
  int age;
}
...
{
  int[] iarr = new int[5];
  Person[] ps = new Person[5];
  System.out.println(iarr[0]); // 0に初期化されている
  System.out.println(ps[0]); // nullに初期化されている
}
```

イニシャライザ(参考)

- コンストラクタより前に初期化するコードを定義できる
  - 見た目は不思議な感じ。ただのカッコの中
- コンストラクタに書けない処理を書くために利用
  - 後程出てくる無名クラスの場合など
- staticフィールドの初期化はstaticイニシャライザに書く
- そんなのがあったな、と覚えておく程度で大丈夫

```
class Person {
  String name;
  int age;

  {                                  // インスタンスイニシャライザ
    name = “default”; age = 0;
  }
}
```

コレクション(p.574)

- データをまとめて処理するときには配列を利用していた
  - もっと便利に使いたい
    - 例えば、（途中への）追加削除をもっと簡単にするには？
  - 求める機能が色々ある
    - 全機能入りのスーパー配列、みたいなものはできない
- 機能ごとに「袋」オブジェクトを用意する
  - コレクション
  - オブジェクトをまとめるオブジェクト

コレクションの例

- ArrayList
  - 「リスト」
  - ココロは「同じものが並んだ構造」
    - [1, 3, 8, 100]
    - [“abc”, “xyz”, “AAA”]
    - [Pos(3, 5), Pos(8,2), Pos(6,3)]
  - これ自体もオブジェクト
- 追加がいくらでもできる
  - add(Object)メソッド
- 順番にアクセスできる
  - （一応）i番目の要素、とアクセスできる
  - get(int i) メソッド
- ライブラリとして準備されている
  - 使うときには “import Java.util.ArrayList”と先頭に書く必要あり

ArrayList

- まずは単純な（ジェネリクスを使わない）書き方

- 出し入れするのはObject型のインスタンス
  - 様々なオブジェクトの「代表」のような型
  - 入れるとObject型だと思って記憶する
  - 出すときはObject型として出てくるので、使いたい型へのキャストが必要
    - 「この型だと思って扱え」と指定する

```
ArrayList l = new ArrayList();
l.add(“東京”);
l.add(“ロンドン”);

String item = (String)l.get(1);
```

型引数の指定 (p.576)

- 「String型が入るList」「Integer型が入るリスト」のように、要素の型を指定することができる
- これを指定すれば
  - 要素を追加するときに違う型を入れるとエラー
  - 要素を取り出すときに決まった型で出てくる
    - のでキャストの必要なし

```
ArrayList<String> l = new ArrayList<String>();
l.add(“東京”);
l.add(“ロンドン”);

String item = l.get(1);
```

コレクションの中身は参照型

```
String[] src = new String[3];
src[0] = “東京”;
...

ArrayList<String> l = new ArrayList<String>();

for (int i = 0; i < src.length; i++) {
    l.add(src[i]);
}
```

- この時、Stringのインスタンスは何個できている？
  - 参照をコピーしている、ということが理解できるだろうか

Listへ順番にアクセスする方法

```
ArrayList<String> l = new ArrayList<String>();
l.add(“東京”);
...

for (int i = 0; i < l.size(); i++) {
    String item = l.get(i);
    System.out.println(item);
}
```

```
ArrayList<String> l = new ArrayList<String>();
l.add(“東京”);
...

for (String item : l) {
    System.out.println(item);
}
```

拡張for文の方が
楽でわかりやすい

コレクションの種類

- ArrayList
  - 配列っぽいイメージ
  - 追加削除が容易にできる
- HashMap
  - 連想配列と呼ばれるようなもの
    - “国語”の科目番号は100, ”数学”の科目番号は103,…のような対応の記憶

```
HashMap<String, Integer> h = new HashMap();
h.put("国語", 100);
h.put("数学", 103);

System.out.println("math: " + h.get("数学"));
```

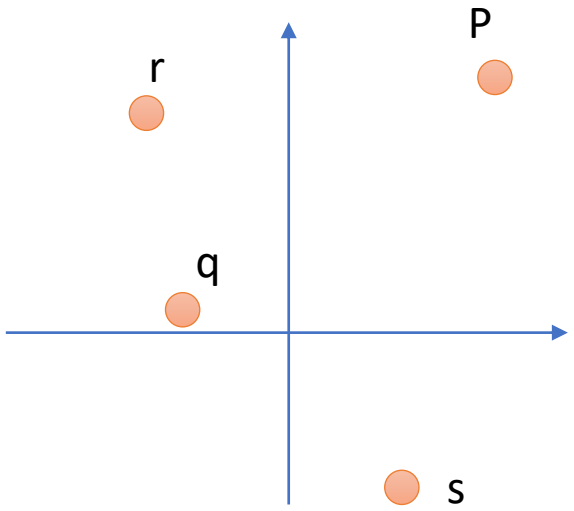
- 他もっとたくさん
  - 系統的な整理が行われている
  - 詳しくは継承やインタフェースを学んでから

クラスを利用する練習

- データをかたまりとして使う訓練をしよう

- 例: 2次元座標

name	x	y
p	20	30
q	-7	2
r	-10	27
s	10	-20



- 4つの点を順番に表示していくとしたら、どういうデータ構造にする？

単純なデータから抜けられない書き方（よく見る）

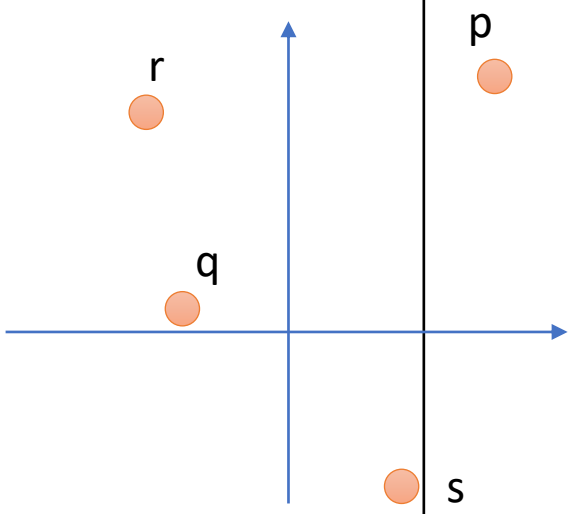
name	x	y
p	20	30
q	-7	2
r	-10	27
s	10	-20

```
String[] names;
int[] xs;
int[] ys;

...

for (int i = 0; i < names.length; i++) {
  point_display(names[i], xs[i], ys[i]);
}

...
```

- オブジェクト指向の良さが発揮できない書き方

オブジェクトとしてまとめる

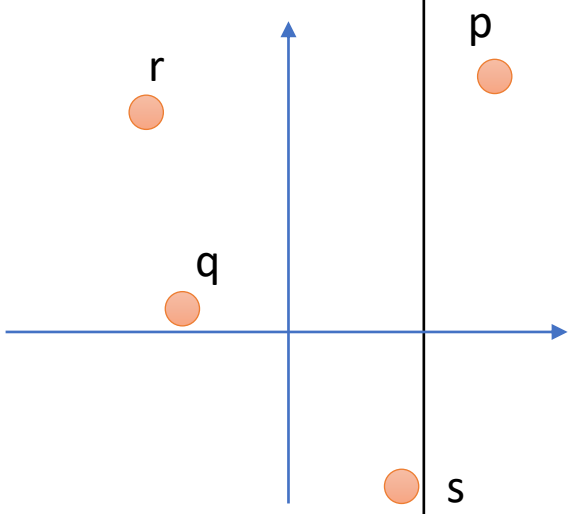
```
class Point {
  String name;
  int x;
  int y;
}
...

Point[] points;
...

for (int i = 0; i < points.length; i++) {
  point_display(points[i]);
}

...
```



- 1つ1つの「点」がオブジェクトなんだ
- 「点」が配列になっているんだ、という意識

メソッド

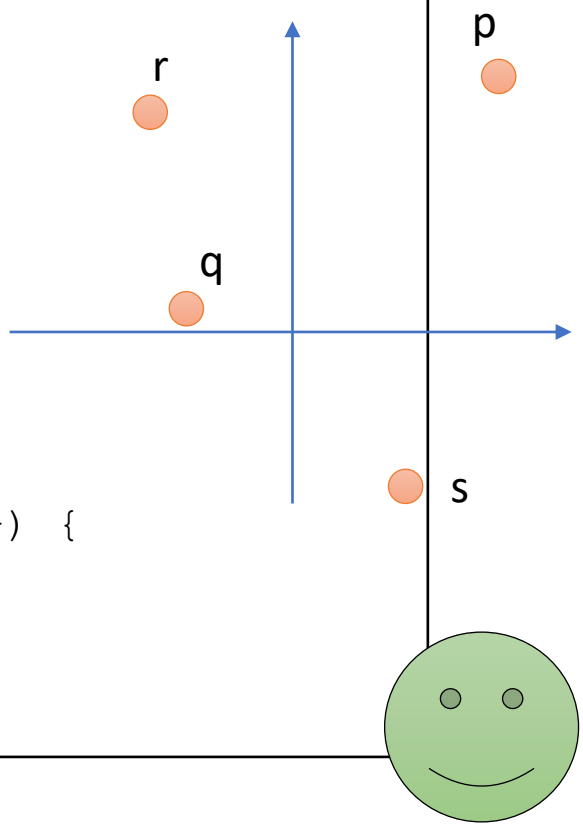
```
class Point {
  String name;
  int x;
  int y;
  void display() { ... }
}
...

Point[] points;
...

for (int i = 0; i < points.length; i++) {
  points[i].display();
}


...
```



- display()は「点」が責任を持っている仕事だよ、という意識がある

メソッドじゃない場合は

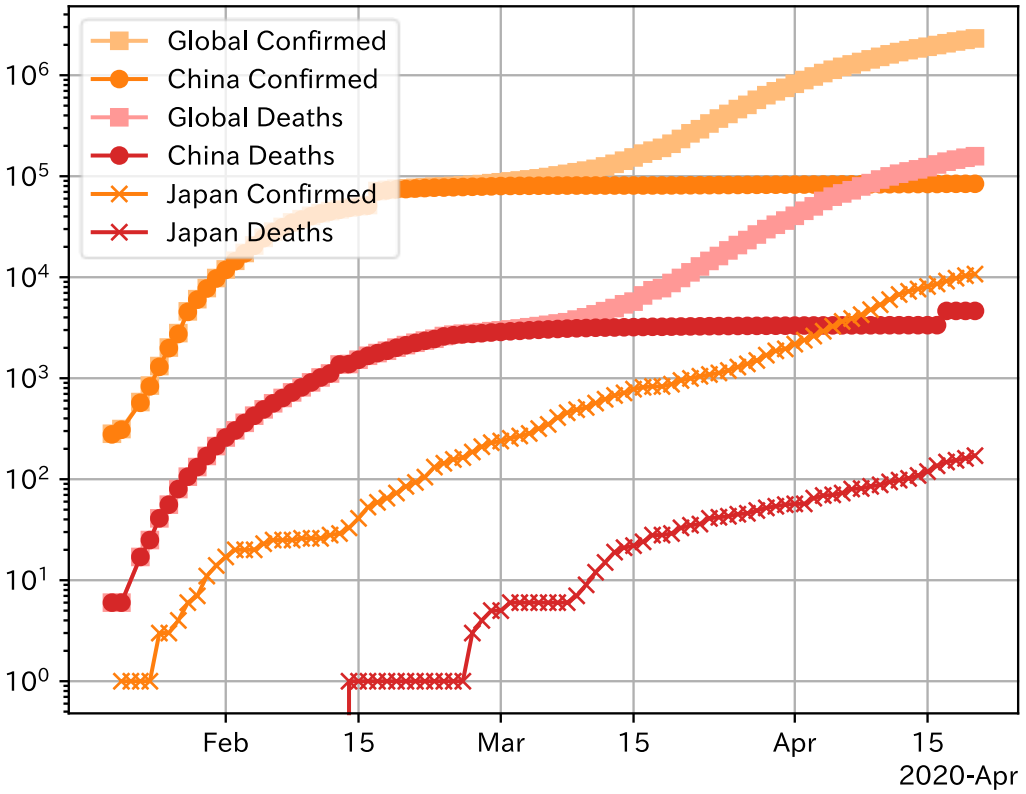
```
class Point {
  String name;
  int x;
  int y;
}
...

Point[] points;
...

for (int i = 0; i < points.length; i++) {
  point_display(points[i]);
}

...
```



- 外に「誰か」がいて、
- 「誰か」がPointの中身を読んで表示している　　感じ
- 動作の責任者が誰か？を考えようとしているのがオブジェクト指向の設計

再び実データ

- どんなデータ構造にする？
- メンバは色々なまとめ方ができそう
  - 正解が決まる感じではない
- staticなものもないかな？



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

提出課題3

- ひな型ソース2つ(FilterTest.java, Pos.java)
- Posクラスのオブジェクトとして表現された座標のリストがある
  - x,yの2次元の要素を保持する
  - ArrayList<Pos>型で作られている
- リストを引数で受け取り、ある基準を満たすPosオブジェクトだけのリストを作って返すような関数filter()を作りたい
  - FilterTest.filter(ArrayList<Pos>list, int th)
  - Posのx,yの両方の要素がともにth以下のものを残す
  - もとのリストはそのまま
    - つまり、filterの中で新たにリストを作って返す
- FilterTest.filter()を完成させよ

仕様

- リストは作り直す必要がある
  - 「袋」のオブジェクトを新しく作る
- リストの要素は作り直す必要はない
  - もともと存在しているPosのオブジェクトを複製する必要はない、オブジェクトの数は変わらない
  - 中身のオブジェクトは「参照」で管理されているので、「参照」をそのまま新しい袋に入れればよい
- よくわからなかったら全部作り直しても大丈夫です

- (参考) mainはフィルターを通した後のPosをすべて表示する
  - どのオブジェクトが表示されるか、考えればわかるはず
  - ついでに元のリストの長さもチェックしてる

提出物

- 提出物はFilterTest.java
  - 先頭に「組番号、名前」と、mainを実行して出力された文字列をコメントで記入
  - package javalec3;
- 答えが正しいことをきちんと確認しよう
  - 元の配列は書き換わっていないかな？
- 〆切は5/14(火) 17:00

参考

- PosクラスのString toString()というメソッドは、Javaでは「わかりやすい表示用の文字列に変換するメソッド」として扱われることが決まっています。main()の中では、これを(暗黙のうちに)使って表示が行われていますね。

今日のまとめ

- オブジェクト指向高速入門
  - オブジェクト指向：責任を持つもの、という意識に従ってやりたいことをまとめる考え方
    - データと機能をもった責任存在がオブジェクト
  - クラスはオブジェクトのまとめ方
    - 同じ種類のオブジェクトを大量に作りたいときに便利
- コレクション高速入門
  - オブジェクトを入れる「袋」のオブジェクト
    - リストやMapなど

- 提出課題3