プログラミング第二 (第 1 週 講義)

増原英彦　　青谷知幸

- 講義概要...
- 第 1 部 データの多様性...
- 第 1 部 データの多様性...
- 第 1 章 原始的 (primitive) データ...
- 第 2 章 クラスの設計...
- 第 3 章 クラス参照とオブジェクトの封じ込め...

講義概要

- 講義の位置付け...
- 講義の目的: プログラム 設計方法 の習得...
- 教科書: “How to Design Classes” ...
- 教科書の構成...
- 注意: 教科書の流儀に従うこと...
- 形式: 講義 / 個人演習 / 全体演習 ...
- 日程と場所 ...

講義の位置付け

- 計算機科学概論 (の演習): プログラミングとはどんなもの?
- プログラミング第一: 与えられた問題を解くための計算手順をプログラムする (+基本的なデータ構造)
- アルゴリズムとデータ構造: 様々な問題に共通するアルゴリズムとデータ構造を知る
- プログラミング第二: 複雑な構造を持つ問題が与えられたときに、それを分解し、データ構造を設計し、処理手順をプログラムする

講義の目的: プログラム 設計方法 の習得

- プログラムに対する要求から次のことが判断できるようになる
  - どんなデータ構造を用意すればよいか
  - データに対する処理をどこに定義すればよいか
- 「オブジェクト指向設計」とも言う
- 例: ロールプレイングゲームを作るときに、どんなデータ構造が必要か?

教科書: “How to Design Classes”

- “How to Design Classes — Data: Structure and Organization”, by Matthias Felleisen, Matthew Flatt, Robert Bruce Findler, Kathryn E. Gray, Shriram Krishnamurthi, and Viera K. Proulx, 2012. (Draft)
  - オブジェクト指向設計の教科書
  - “How to Design Programs” 流のデザイン・レシピに基づく
  - Java 言語を用いる
  - 以下から入手可能
    http://www.ccs.neu.edu/home/matthias/htdc.html
- このスライドは受講者に公開 (後で説明)

教科書の構成

- 第 I 部: 多様なデータ (要求からクラス階層を設計する)
- 第 II 部: 関数的なメソッド (データを用いた計算方法のうち、関数的なもの)
- 第 II 部: クラスによる抽象化 (共通部分のあるクラスを継承によってまとめる)
- 第 IV 部: 循環オブジェクトと命令的メソッド (非関数的なデータの扱い)
- 第 V 部: データ表現の抽象化 (フレームワークとその設計)
——(授業ではこのあたりまで)——
- 第 VI 部: データ走査の抽象化 (デザインパターンによる抽象化)

注意: 教科書の流儀に従うこと

- 教科書の流儀には強い「くせ」がある
  - デザイン・レシピを用いる、継承を使わない、関数的に記述する、標準ライブラリを使わない等々
  - あまり一般的でないので好きになれないかも
- 授業の間だけなので我慢して従ってほしい
- 身につく「設計方法」は一般的なもの

形式: 講義 / 個人演習 / 全体演習

- 1 週 3 × 90 分を次のように構成する
  - (金 34 限) 講義: そのラウンドの内容を解説・課題の出題
  - (金 56 限) 個人演習: 内容理解のための基本的な問題を解く
    - 問題は github にて提供、提出する
  - (火 34 限) 全体演習: 理解を確認するための問題を解く
    - 交替で代表者を指名する
    - 代表者がまとめ役をして問題を解く
      (問題を解く役ではなく「教えてもらう」役)
    - 代表者が黒板または端末を操作する
    - 残りの学生が代表者を助ける
- 評価方法:
  個人演習問題の解答 + 全体演習での代表回数・発言数
  + 期末試験
  - 合計発言数は最低 4 回

日程と場所

April

Su	M	Tu	W	Th	F	Sa
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	26	27	28
29	30					

May

Su	M	Tu	W	Th	F	Sa
		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	26
27	28	29	30	31		

June

Su	M	Tu	W	Th	F	Sa
					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	26	27	28	29	30

- 場所
  - 金曜 34 限 講義　　@ 講義室
  - 金曜 56 限 個人演習 @ 演習室
  - 火曜 34 限 全体演習 @ 講義室
- 注意
  - 4/10 火、4/13 金、5/1 火 は休講
  - 6/5 火は補講期間
  - 6/8 金 34 限に期末試験
- 変更の可能性あり・OCW のカレンダーで最新のものを確認せよ

- ソフトウェア作成の第一歩:
  与えられた問題文を読んで理解する
  - “How to Design Programs” でもやっていた
- 注目点:
  - どんな情報が与えられるのか?
  - どんな情報を答えるべきか?
- 次に考えること: 情報をプログラム中でどう表わすか?
  - Java 言語ではオブジェクトを使って表わす
    - 用語: クラスはデータの集合、オブジェクトは個々のデータ
  - 情報の内部構造、情報どうしの関係をどう表わすか?

第 1 部 データの多様性

- ラウンド 1
  - 第 1 章 原始的 (primitive) なデータ
  - 第 2 章 クラス
  - 第 3 章 クラス参照とオブジェクトの封じ込め
- ラウンド 2
  - 第 4 章 クラスの合併
  - 第 5 章 合併、自己参照、相互参照
  - 第 6 章 クラス階層の設計

第 1 章 原始的 (primitive) データ

- 単純な情報はクラスを使わなくても直接プログラム中で表現できる
- 直接プログラムで表現できるデータを原始的なデータという (内は Java での「型」の名前)
  - 整数 (int)
  - 実数 (double)
  - 真偽値 (boolean) ON/OFF や有無を表わすときに使う
  - 文字列 (String) 記号情報を表わすときに使う
    - 例: 名前、住所、会話など
    - 本当は原始的でない
- 原子的 (atomic) なデータとも言う (それ以上分解できない)

第 2 章 クラスの設計

- 問題記述と情報の例を読む...
- クラス図を描く...
- クラス図の描き方...
- クラス定義に変換する...
- クラス定義の書き方...
- インスタンス生成式を書く...
- プログラムの管理: クラス定義と例クラスの定義...
- 全てのインスタンスに共通するフィールド (1)...
- まとめ: クラスの設計 (1)...

問題記述と情報の例を読む

例: コーヒー店の売上記録 (問題記述の一部)

…コーヒー店のコーヒーの売り上げを記録するプログラムを作れ。伝票はコーヒーの種類、値段 (ポンドあたり) と重さ (ポンド) を記録する。…

例: 情報の例

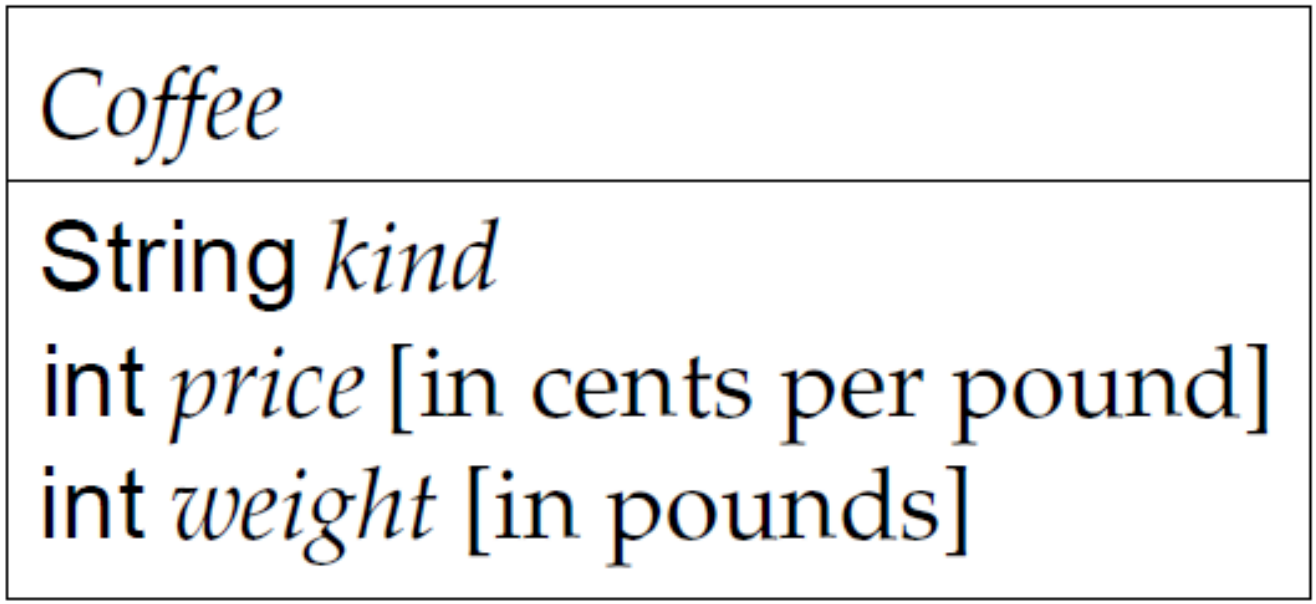
- 100 ポンドのハワイ・コナを 20.95 ドル/ポンド
- 1000 ポンドのエチオピアコーヒーを 8.00 ドル/ポンド
- 1700 ポンドのコロンビア・スペシャルを 9.50 ドル/ポンド

どう表わすかを考える

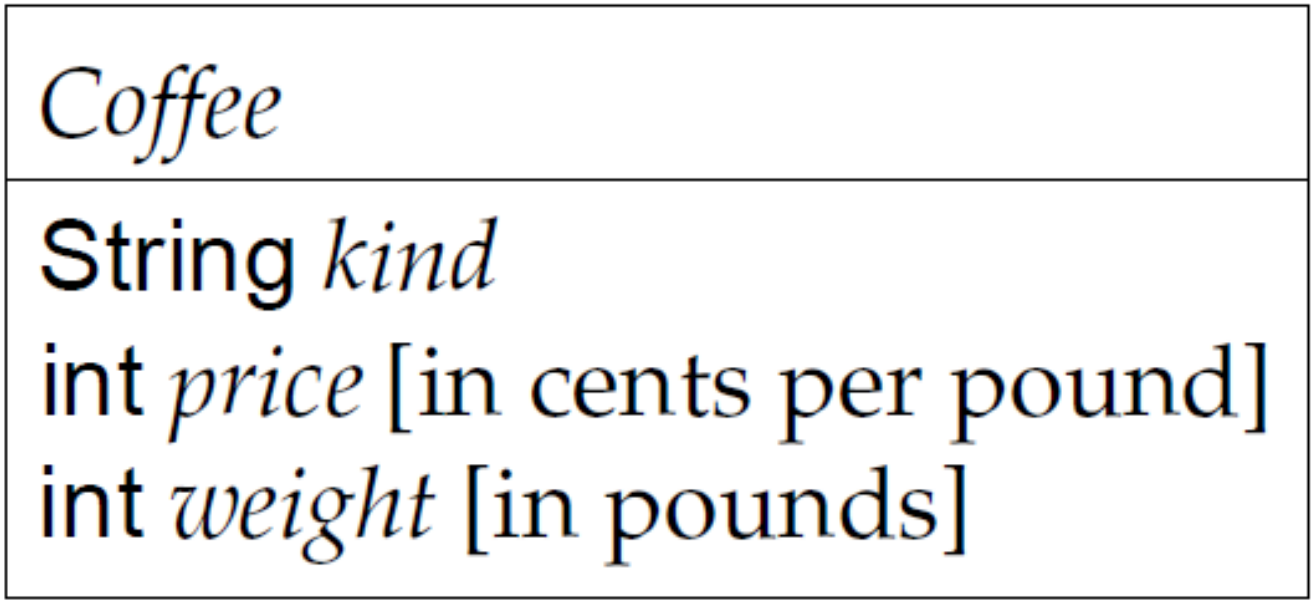
- どんな情報がある? → 1 つの売り上げ
- どんな情報で構成? → コーヒーの種類、値段、重さ
- それぞれはどう表わす? → 文字列、整数 (セント/ポンドで)、整数 (ポンド)

クラス図を描く

- どんな情報がある? → 1 つの売り上げ
- どんな情報で構成? → コーヒーの種類、値段、重さ
- それぞれはどう表わす? → 文字列、整数 (セント/ポンドで)、整数 (ポンド)



クラス図の描き方



- 実際は手で描く
- 箱は１つのクラスを表わす
- 線の上はクラス名
- 線の下フィールド (型 名前 [注釈])
- 注釈は「どうやって表わすか」を説明

クラス定義に変換する

Coffee
String *kind* int *price* [in cents per pound] int *weight* [in pounds]

```
// コーヒー売上伝票                    ★
class Coffee {
  String kind;                         ☆
  int price; // cents per pound
  int weight; // pounds
  (続く)
```

- ★目的文: 何を表わすかの説明
- ☆フィールド定義

クラス定義の書き方

```
// コーヒー売上伝票
class Coffee {
  String kind;
  int price; // cents per pound
  int weight; // pounds
  Coffee(String kind, int price, int weight) { ★
    this.kind = kind;                          ☆
    this.price = price;
    this.weight = weight;
  }
}
```

- 後半: コンストラクタ (constructor) — 自動的に決まる
- ★ [クラス名] ([フィールド 1 の型] [フィールド 1 の名前], ...) {
- ☆ this. [フィールド 1 の名前] = [フィールド 1 の名前];

インスタンス生成式を書く

```
class Coffee {
  ...
  Coffee(String kind, int price, int weight) {
  ...
```

例: 情報の例

- 100 ポンドのハワイ・コナを 20.95 ドル/ポンド
- ...

```
new Coffee("Hawaiian Kona", 2095, 100)
```

- new [クラス名]([フィールド 1 の値],...)
- (どこに書くかは後で)
- 与えられたもの以外も書いてみること
- 注意: 実在しない情報を表わす式も書ける (例:負の重さ)

プログラムの管理: クラス定義と例クラスの定義

- クラスは別々のファイルに定義する
  (Coffee クラスは Coffee.java ファイル)
- 情報の例は元のクラスに「Examples」を追加した
  例クラスに定義する
  (Coffee クラスは CoffeeExamples クラス)
- 例クラスは適当な名前のフィールドを並べ、
  =の右辺にインスタス生成式を書く
- (例クラスをテストする (後のラウンド))

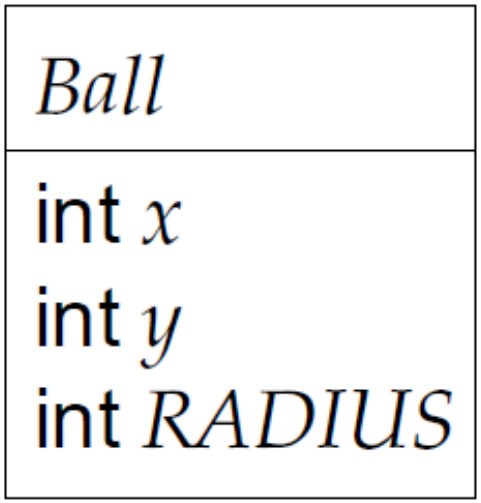
```
//コーヒー売上の例
class CoffeeExamples {
  Coffee kona = new Coffee("Kona", 2095, 100);
  Coffee ethi = new Coffee("Ethiopian", 800, 1000);
  Coffee colo = new Coffee("Colombian", 950, 20);
}
```

全てのインスタンスに共通するフィールド (1)

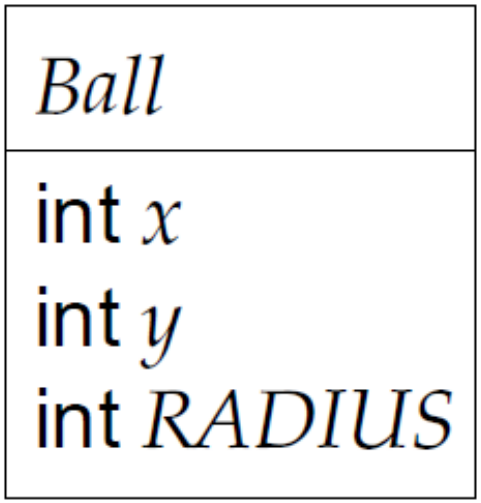
例:

…ビリヤードテーブル上のボールをシミュレートするプログラムを作成せよ。…

- どんな情報? ボールの状態
- どんな情報で構成? X 座標、Y 座標、半径



全てのインスタンスに共通するフィールド (2)




```
// ビリヤードテーブル上を動くボール
class Ball {
  int x;
  int y;
  int RADIUS = 5;          ★
  Ball(int x, int y) {     ※
    this.x = x;
    this.y = y;
  }
}
```

- ★ フィールドの型 フィールドの名前 = 共通する値;
  - 全てのインスタンスに共通する場合
- ※コンストラクタ引数には書かない

まとめ: クラスの設計 (1)

- 問題文を読む。
  - 問題領域のモノの性質を書いた文や一覧を見つける。
  - 何個のフィールドが必要か? どんな種類の情報を表わすか?
  - クラス図に書き込む。(図ならすぐに描ける)
- クラス図をクラス定義に変換する。
  - 目的文 (そのクラスのオブジェクトがどんな情報をどうやって表わすか) を追加。
  - フィールド定義を書く。
  - コンストラクタを書く。引数は全て (※) のフィールド。
  - コンストラクタ本体を書く
    - this.フィールド名 = フィールド名;
    - ※ 各フィールドはクラスの全オブジェクトに共通か?
      共通ならフィールド定義に初期化等式を付ける。

まとめ: クラスの設計 (2)

- 情報の例を例クラスに変換する
  - 情報の例をインスタンス生成式として表わす
  - 適当なクラスのインスタンス生成式を書く
    それを問題世界の情報として解釈するとどうなるか考える
    - 問題文にない例を加えることが重要。
      (コメントにそのことを書いておくこと)

第3章 クラス参照とオブジェクトの封じ込め

- あらまし: 情報が入れ子になっているときにどうするか?...
- 例題: ジョギング記録管理プログラム...
- クラス図でのクラスの参照...
- クラス定義でのクラスの参照 (1)...
- 参照のあるインスタンス生成式...
- まとめ: クラスを参照するクラスの設計...

あらまし: 情報が入れ子になっているときにどうするか?

- これまでは 1 つの情報に入る要素は原始的な情報だった
  (コーヒー売上げに入るのは、重さ:実数など)
- 現実は何重にも入れ子になる

例題: ジョギング記録管理プログラム

例: ジョギング記録管理プログラム

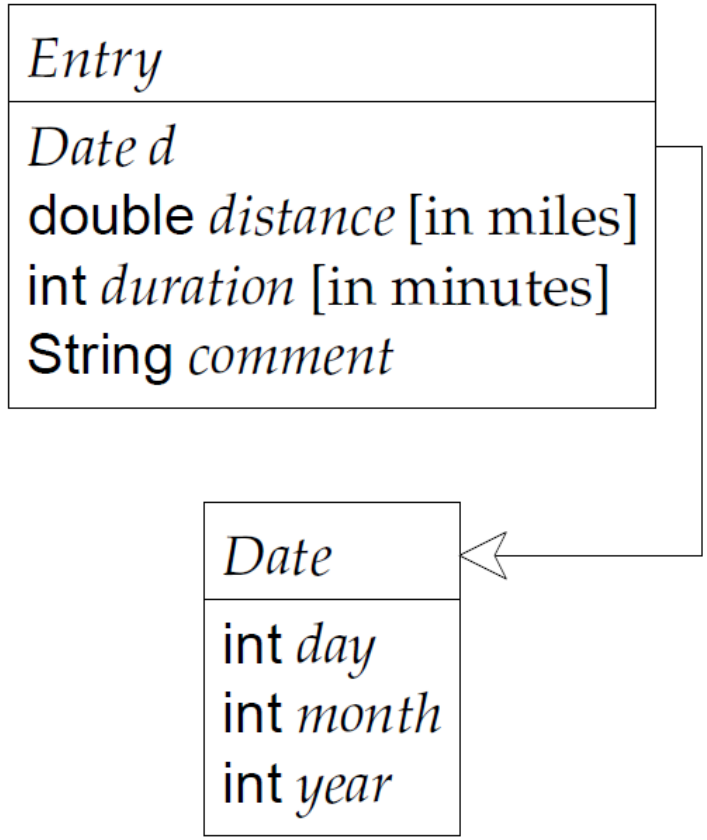
ジョギング記録を管理するプログラムを開発せよ。利用者は毎日、その日のジョギングについての記録を入力する。各記録はその日の日付、走った距離、走った時間、および練習後の体調のメモである。

on June 5, 2003	5.3 miles	27 minutes	feeling good
on June 6, 2003	2.8 miles	24 minutes	feeling tired
on June 23, 2003	26.2 miles	150 minutes	feeling exhausted
...	...	...	...

- 1 つの記録は 4 つの情報から成る: 日付、距離、時間、メモ
  - 距離、時間、メモは原始型: double(マイル), int(分), String
  - 日付は 3 つの情報から成る → 原始型でない

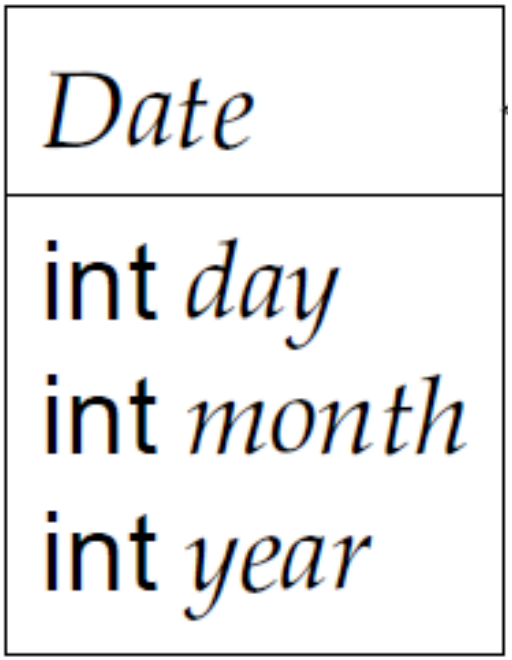
クラス図でのクラスの参照

- 1 つの記録は 4 つの情報から成る: 日付、距離、時間、メモ
  - 距離、時間、メモは原始型: double(マイル), int(分), String
  - 日付は 3 つの情報から成る → 原始型でない
- 日付は記録 (Entry) とは別の箱で表わす
- Entry のフィールド d の横から Date クラスに矢印を引く



クラス定義でのクラスの参照 (1)

まずは末端のクラスから




```
// calendar dates
class Date {
  int day;
  int month;
  int year;
  Date(int day, int month, int year) {
    this.day = day;
    this.month = month;
    this.year = year;
  }
}
```

クラス定義でのクラスの参照 (2)

他のクラスを参照するクラス

Entry
Date d double *distance* [in miles] int *duration* [in minutes] String *comment*

```
// an entry in a runner's log
class Entry {
  Date d;                                    ★
  double distance; // miles
  int duration; // minutes
  String comment;
  Entry(Date d, double distance,
        int duration, String comment) {
    this.d = d;
    this.distance = distance;
    this.duration = duration;
    this.comment = comment;
  }
}
```

★要素の型としてクラス名を書く

参照のあるインスタンス生成式

```
// an entry in a runner's log
class Entry {
... Entry(Date d, double distance, int duration, String comment) { ...
// calendar dates
class Date { ... Date(int day, int month, int year) {...}}
```

- 方法 1: インスタンス生成式を入れ子に書く

```
new Entry(new Date(5, 6, 2003), 5.3, 27, "Good")
```

- 方法 2: 末端のインスタンスを変数にしまい、変数名を使う

```
Date d1 = new Date(5, 6, 2003);
Entry e1 = new Entry(d1, 5.3, 27, "Good");
```

(後で日付だけ検査するときにはこっちが便利)

まとめ: クラスを参照するクラスの設計

- クラス図を描く (クラス間の参照)
- クラス図をクラス定義に変換する
  - 他のクラスを参照していないクラスからやる
    - 「原始的 (primitive) クラス」とも言う
      (vs. 「合成 (compound) クラス」)
    - (いまは) 必ずある
- インスタンス生成式を書く
  - 全てのクラスについて書くこと
  - 原始的クラスからやる
- 情報とインスタンス生成式の対応を理解する