プログラミング第二 (第 2 週 講義)

増原英彦　　青谷知幸

- お知らせ...
- 第 4 章 クラスの合併 (union)...
- 第 5 章 合併、自己参照、相互参照...
- 第 6 章 クラス階層の設計...

お知らせ

前週の個人演習の提出内容を確認した

- 第 2 週の個人演習資料中の
  ファイル evaluation.pdf を参照せよ
- 疑問等は email:p2-2017@prg.is.titech.ac.jp へ

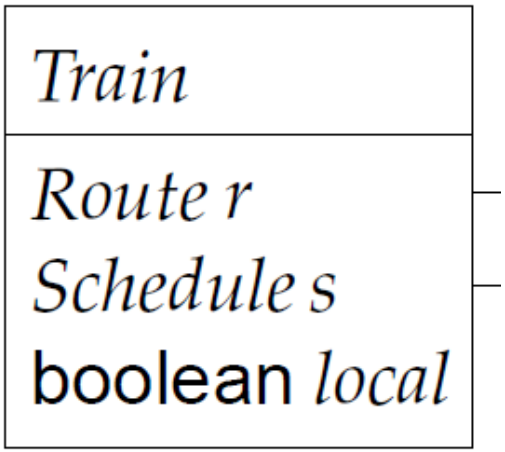
第 4 章 クラスの合併 (union)

- 異なる種類の情報を同様に扱う...
- 例題: 幾何図形...
- 4.1 型とクラス (合併とインスタンス生成式)...
- 4.1 型とクラス (規則)...
- まとめ: クラスの合併を設計する...

異なる種類の情報を同様に扱う

例: 鉄道旅行支援プログラム

鉄道の旅を支援するプログラムを開発せよ。列車に関する情報としてはスケジュール、経路、各駅停車であるか(そうでない場合は急行)。経路は発駅と着駅からなる。スケジュールは出発時刻と到着時刻から成る。



- 2 種類の列車 (各駅・急行)
  どちらも同じ「列車情報」
- とりあえずブール値で区別したが、
  もっと種類が増えたらどうする?

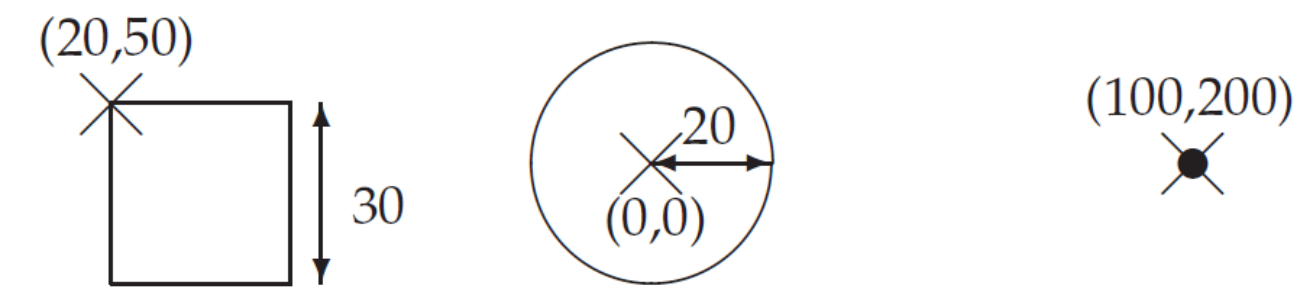
例題: 幾何図形

- 問題文: 幾何図形を扱うプログラム...
- それぞれの図形種類の設計をしよう...
- クラスの合併 (union) を表わすクラス図...
- インタフェースとクラスの定義...

問題文: 幾何図形を扱うプログラム

例: 幾何図形

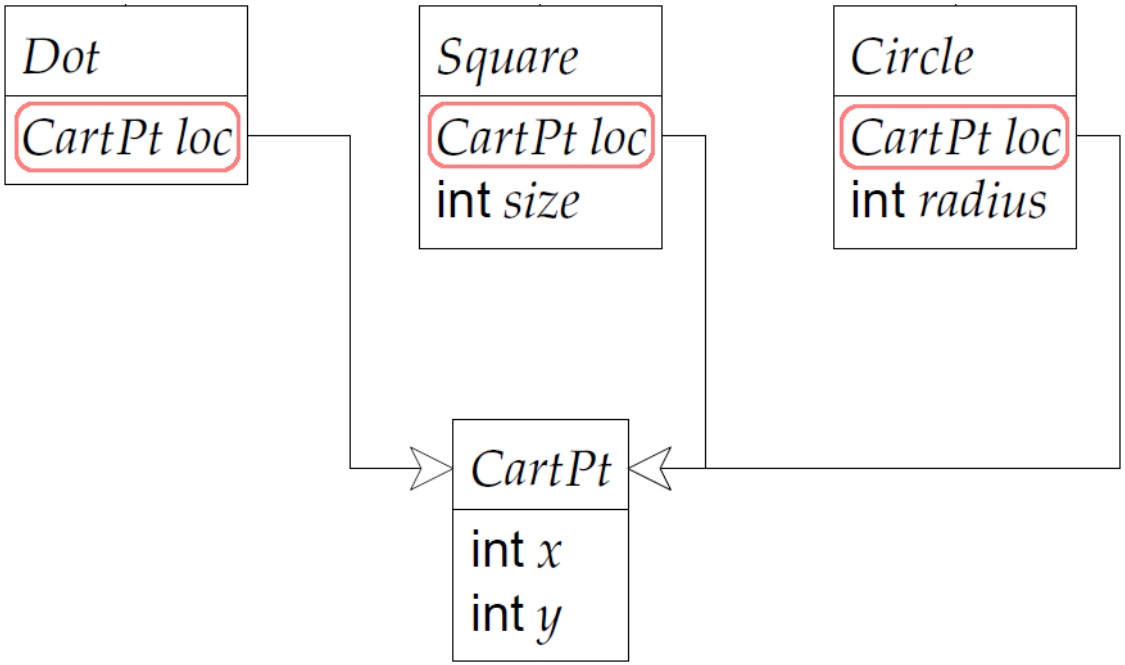
…デカルト座標系上にある 3 種類の図形—正方形、円、点—を扱う描画プログラムを作れ。



正方形は左上角の位置と大きさで与えられる。円は中心の位置と半径、点はその位置 (半径 3 の円盤を描くことにする)

- 3 種類の図形がある。内容は異なる
- (書いてはいないが) プログラムは 3 種類の図形の どれか 1 つを扱う。どの種類かは実行するまで分からない

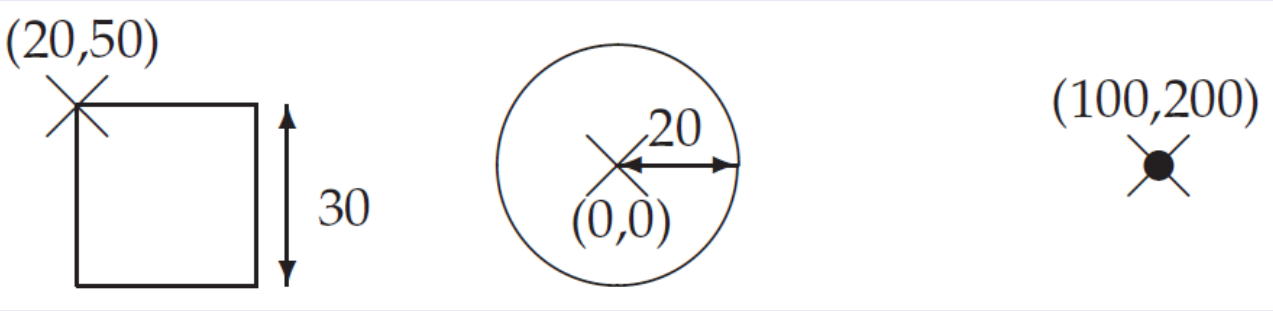
それぞれの図形種類の設計をしよう



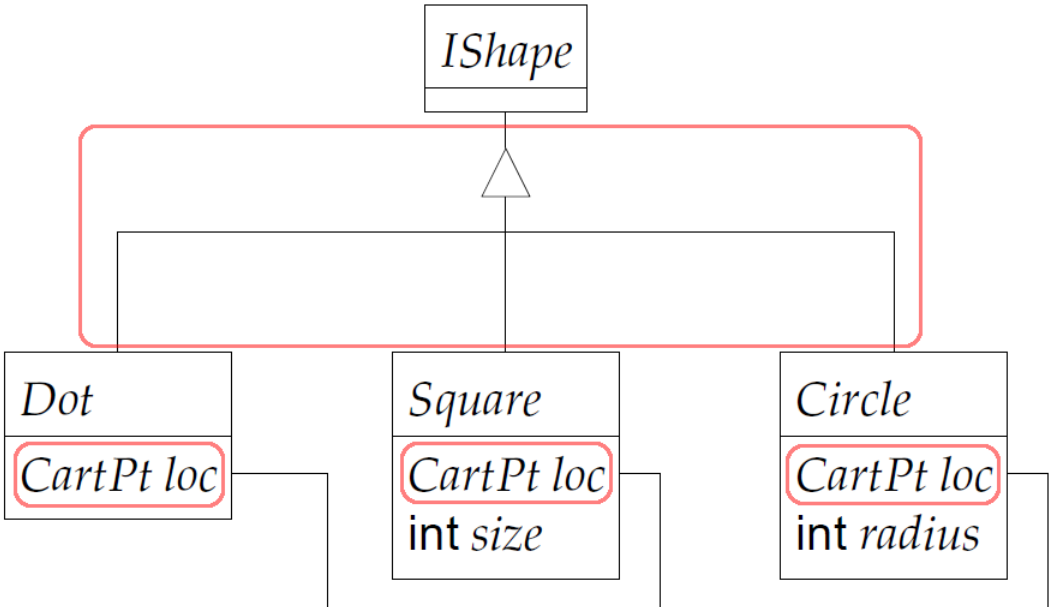
例: 幾何図形

…正方形は左上角の位置と大きさで与えられる。円は中心の位置と半径、点はその位置…

(ここは前章でやった通り)



クラスの合併 (union) を表わすクラス図



例: 幾何図形

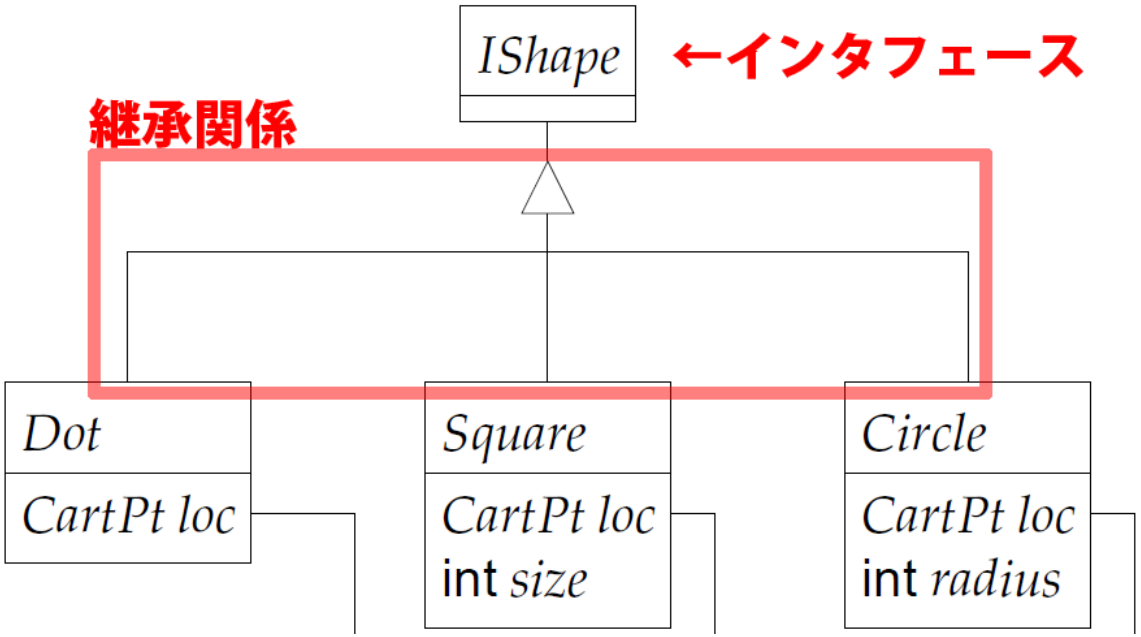
…デカルト座標系上にある 3 種類の図形—正方形、円、点—を扱う描画プログラムを作れ。…

3 種類の図形のどれかを扱う (実行するまで分からない)

方法: 複数クラスの合併を使う — 合併とは?

- Dot クラス = 色々な Dot インスタンスの集合
- IShape = Dot, Square, Circle クラスの合併集合
  = 色々な Dot, Square, Circle インスタンスの集合

用語: インタフェースと継承関係

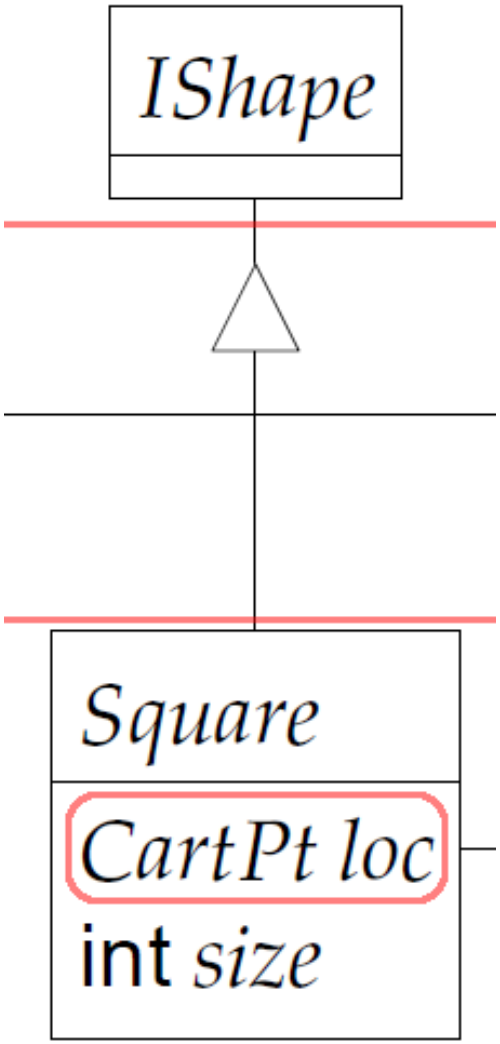


合併を表わす箱を
「インタフェース (interface)」
描き方はフィールドのないクラスと同じ
名前は「I」で始める

- 合併の「元」からインタフェースへ矢印を引く
  大きく白い三角 (△) を使う
  「継承関係」と呼ぶ

「Dot は IShape を継承 (inherit) している」
「Dot は IShape を実装 (implement) している」という

インタフェースとクラスの定義



インタフェースは interface 定義に変換する
(文法: interface IF 名 {})

```
// geometric shapes
interface IShape {}
```

継承関係→クラスに implements 宣言を付ける
(文法: class クラス名 implements IF 名 {...)

```
// a square shape
class Square implements IShape {
  CartPt loc;
  int size;
  Square(CartPt loc, int size) {
    this.loc = loc;
    this.size = size;
  }
}
```

合併を表わすインタフェースとクラスの定義 (2)

```
// a dot shape
class Dot implements IShape {
  CartPt loc;
  Dot(CartPt loc) {
    this.loc = loc;
  }
}
```

```
// a circle shape
class Circle implements IShape {
  CartPt loc;
  int radius;
  Circle(CartPt loc, int radius) {
    this.loc = loc;
    this.radius = radius;
  }
}
```

残りのクラス定義も同様に
implements IShape を付ける

```
// Cartesian points on a computer
// monitor
class CartPt {
  int x;
  int y;
  CartPt(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

4.1 型とクラス (合併とインスタンス生成式)

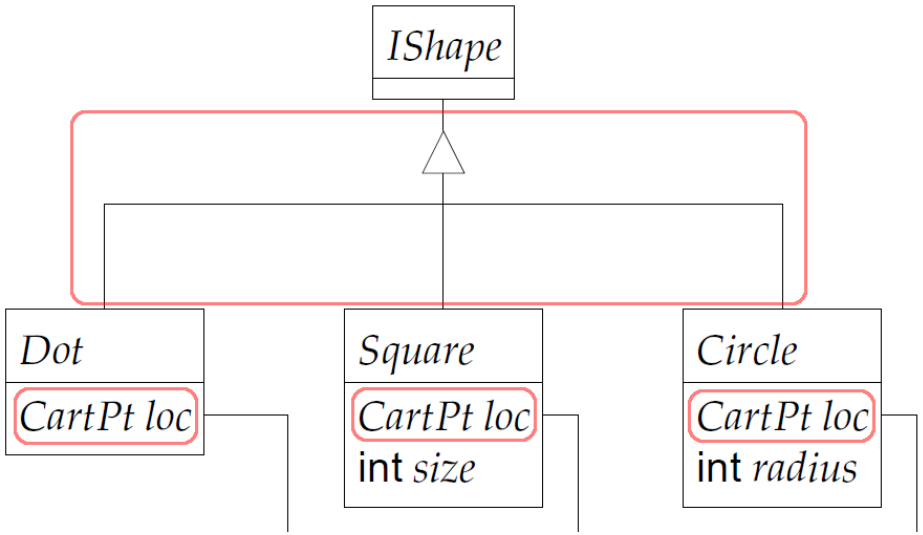
合併があるときのインスタンス生成式は次のように書ける

- 「図形を表わす変数 s が、(いまは) 正方形を表わしている」

```
IShape s = new Square(...);
```

- 「正方形を表わす s がある」

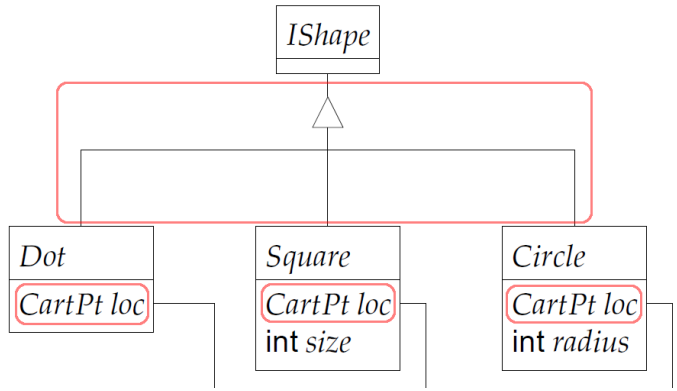
```
Square s = new Square(...);
```



(今後 s に対してどんな操作が許されるかが違ってくる)

4.1 型とクラス (規則)

IShape s = new Square(...);



左辺と右辺の違いは?

- 左辺: フィールド s が何を
  表わしているつもりか?
- 右辺: フィールド s に実際に
  格納するインスタンス

フィールドの型に関する規則(Java)

- 型は、インタフェース、クラス、基本型のどれか
- T f = new C(...) の意味
  - 変数 f は型 T を持つ
  - 変数 f は C のインスタンスを表わす
  - ただし (1)T と C が同じまたは (2)C implements T でなければいけない

まとめ: クラスの合併を設計する

- いつ: 1 つの情報の集まりが n 個の性質の異なる集まりから成るとき
  - 性質の異なる = 内部に持つ情報の種類が違う
- どのように: 1 つのインタフェースと
  それを実装している n 個のクラスを作る
- クラス図を描く
  - インタフェース: 名前を I で始める (慣習),
    フィールドのないクラスと同じ見た目
  - 実装しているクラスからインタフェースに
    継承関係矢印を引く
- クラスとインタフェース定義に変換する
  - インタフェースは interface を使って定義
    - 目的文を書く (クラスと同じ)
- データ例を作る
  - 変数の型にはインタフェースが使える
  - インタフェースのインスタンスは作れない

第 5 章 合併、自己参照、相互参照

- 不定個の情報...
- 5.1 合併の中での封じ込め (1)...
- Cons リストのクラス図...
- クラス定義への変換...
- インスタンス生成式...
- 5.2 合併の中での封じ込め (2)...

不定個の情報

- ここまで: 1 つの情報の中身は固定個の情報
  - 例: 時刻は「時」と「分」
- 1 つのモノが不定個の情報から成るときはどうする?
  - 例: ジョギング記録管理プログラムは不定個の「毎日の記録」から成る
  - 例: 鉄道旅行支援プログラムは不定個の列車のスケジュール情報から成る

可変個ともいう

「配列があるじゃないか!?」

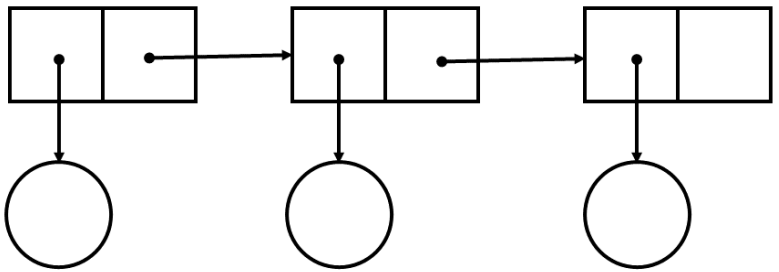
- M: 「いままでは固定した個数の情報を表わす方法しかなかったわけで。そこで不定個の情報を表わすことを考えましょう。」
- A: 「それって配列のこと? Java にもあるじゃん。」
- M: 「いえ、配列のことは知らないふりをして下さい。第一、配列は作るときに個数を決めておかなければいけません。ここでは個数が後から増えたり減ったりするような場合も考えます。」
- B: 「でも、わたしが愛して止まない Perl / Python / Ruby / Javascript / ... の配列は後から個数を増やせるます!」
- M: 「そうですね。Java にも Vector クラスとかありますね。でも、そういう『配列』は、実際には内部で何らかのデータ構造として設計されているわけです。ここではそういうデータ構造の設計方法を学ぶので、やっぱり今は知らないふりをして下さい。」

5.1 合併の中での封じ込め (1)

例: ジョギング記録管理プログラム (再掲)

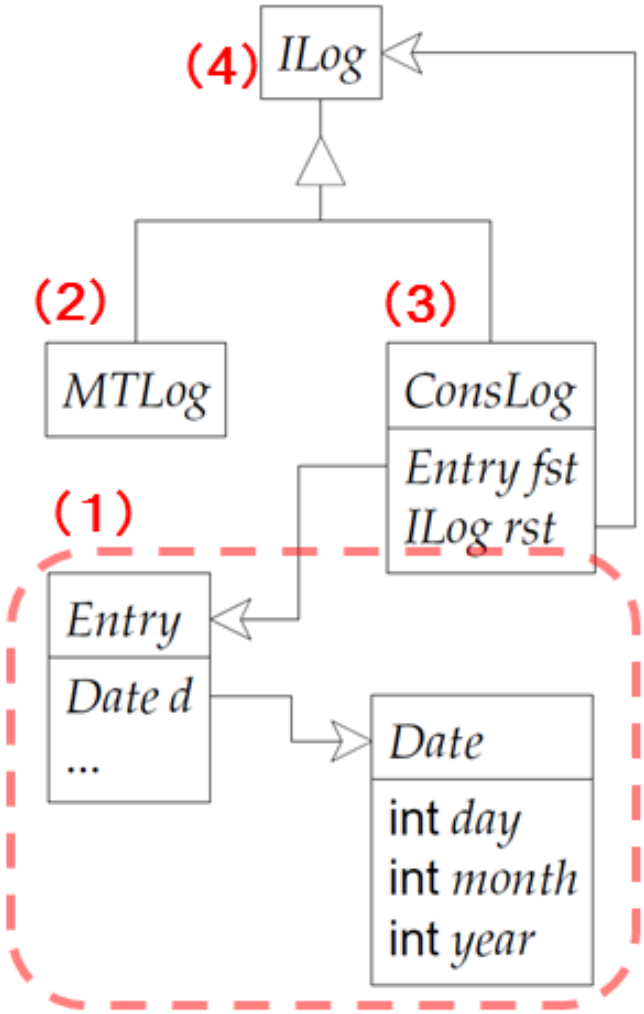
ジョギング記録を管理するプログラムを開発せよ。利用者は毎日、その日のジョギングについての記録を入力する。各記録はその日の日付、走った距離、走った時間、および練習後の体調のメモである。

- プログラムは記録の列を 1 つの情報として扱う
  - 不定個数の記録がある
  - 例: 月ごとの平均距離を求める
- 表わし方: consリスト



Cons リストのクラス図

- (1) リストの要素のクラス図を作る
- (2)空リストのクラスを作る (MTLog)
- (3)リストに1つ要素を追加したリストのクラスを作る (ConsLog)
  - 追加される要素 (fst)
  - 追加対象のリスト (rst)
- (4)あらゆる長さのリストを表わすインタフェースを作る (ILog)



ConsLog が ILog 経由で自分を参照している!

クラス定義への変換

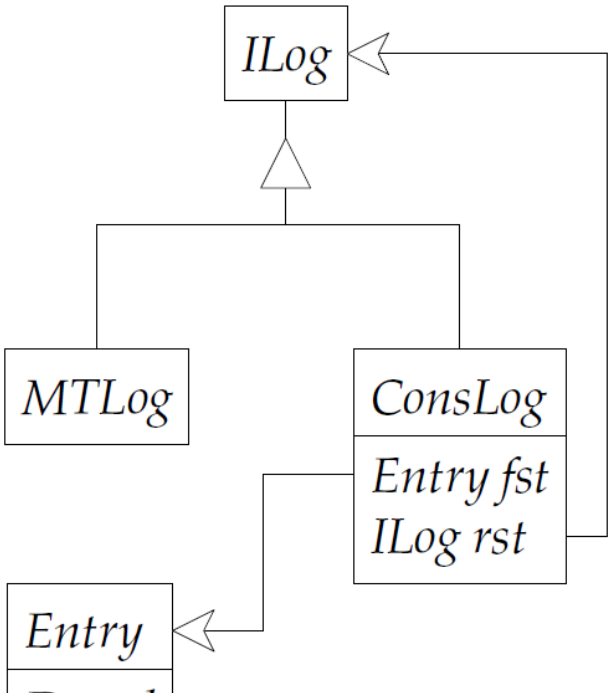
特別なことはない。

```
// a runner's log
interface ILog {}
```

```
// the empty log
class MTLog implements ILog {
  MTLog() {}
}
```

```
// adding an entry to a log
class ConsLog implements ILog {
  Entry fst;
  ILog rst;

  ConsLog(Entry fst, ILog rst) {
    this.fst = fst;
    this.rst = rst;
  }
}
```




```
// an individual entry
class Entry { ... }
```

インスタンス生成式

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
...	...	...	...

各要素は定義済み

```
Date d1 = new Date(5, 6, 2003);
Date d2 = new Date(6, 6, 2003);
Date d3 = new Date(23, 6, 2003);
Entry e1 = new Entry(d1, 5.3, 27, "Good");
Entry e2 = new Entry(d2, 2.8, 24, "Tired");
Entry e3 = new Entry(d3, 26.2, 150, "Exhausted");
```

リストを作る (NEW!)

```
ILog l1 = new MTLog();
ILog l2 = new ConsLog(e1,l1);
ILog l3 = new ConsLog(e2,l2);
ILog l4 = new ConsLog(e3,l3);
```

空のリストから作ればよい

5.2 合併の中での封じ込め (2)

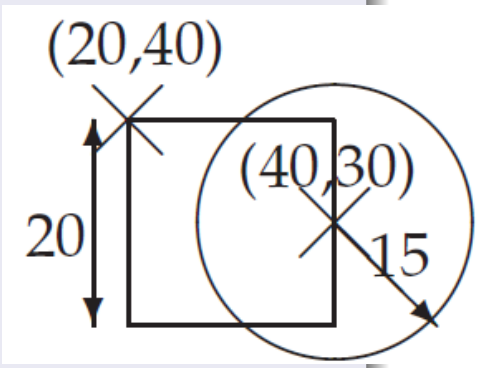
- 問題文: 図形の重ね合わせ...
- 図形の重ね合わせ: 考え方...
- 図形の重ね合わせ: クラス図...
- 図形の重ね合わせ: クラス定義とインスタンス生成式...
- 図形の重ね合わせ: インスタンス生成式...

問題文: 図形の重ね合わせ

循環が複数ある場合 (リストは循環が 1 つ)

例: 幾何図形

……少なくとも次の三種類の図形: 点、正方形、円を扱う描画プログラムを作れ。…さらにプログラムは「重ねた図形」を扱える。例えば以下は円を正方形の右に重ねた図形である。
この重ねた図形をさらに別の図形に重ねることもできる…

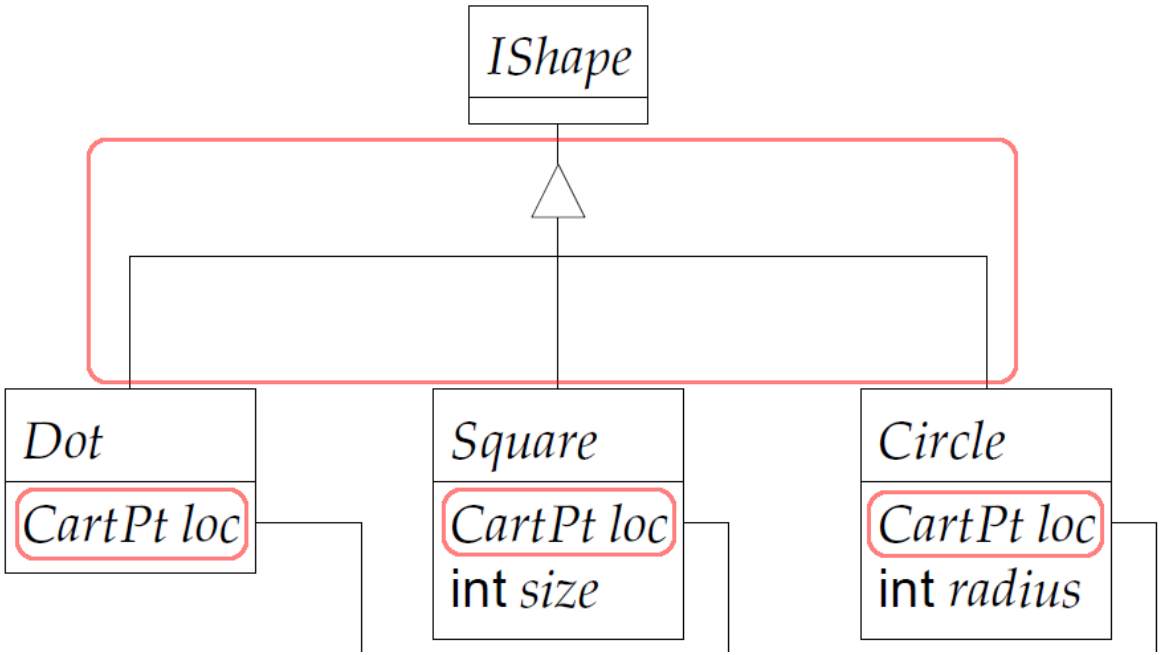


※重ねた図形も 1 つの図形としたい!

図形の重ね合わせ: 考え方

例: 幾何図形

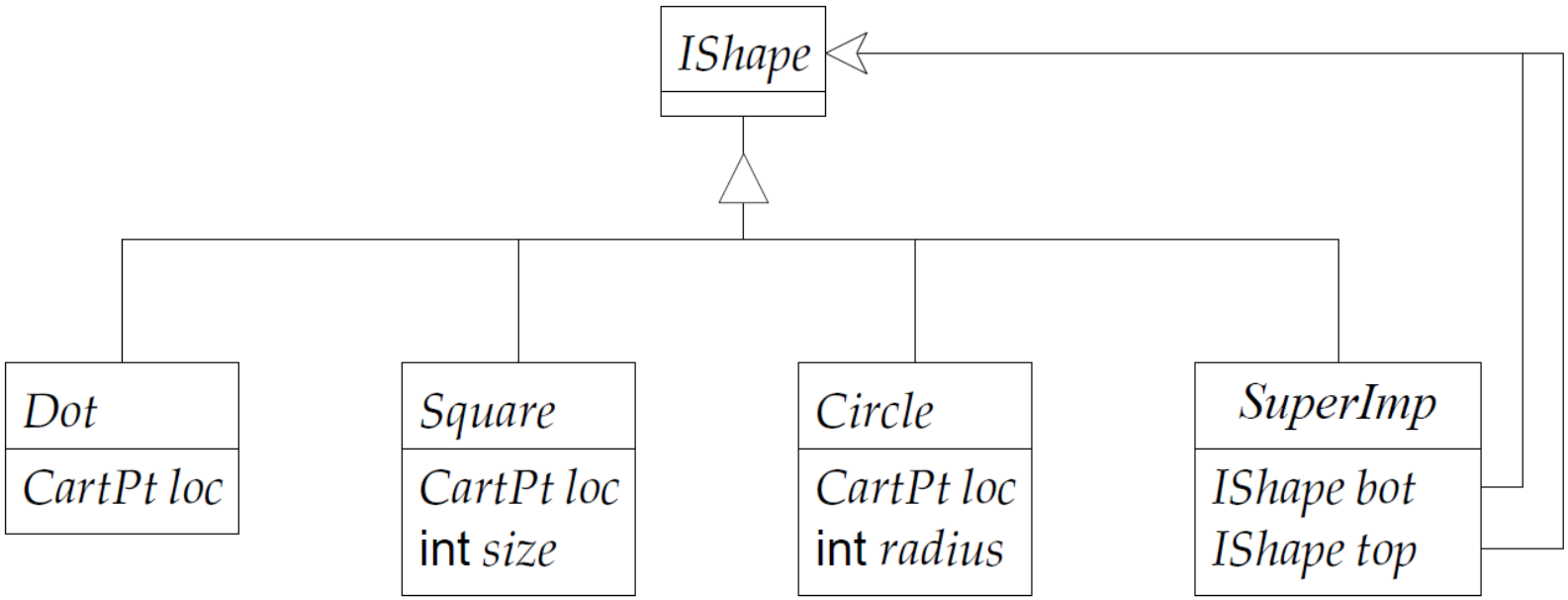
……少なくとも次の三種類の図形: 点、正方形、円を扱う描画プログラムを作れ。…さらにプログラムは「重ねた図形」を扱える。例えば以下は円を正方形の右に重ねた図形である。
この重ねた図形をさらに別の図形に重ねることもできる…



ここに
「重ねた図形」を
表わすクラスを
追加

- SuperImp
  クラスとする

図形の重ね合わせ: クラス図



2 つの循環参照がある

図形の重ね合わせ: クラス定義とインスタンス生成式

クラス定義の変換はいままでと同じ。(他のクラスは省略)

```
class SuperImp implements IShape {
  IShape bot;
  IShape top;
  SuperImp(IShape bot, IShape top) {
    this.bot = bot;
    this.top = top;
  }
}
```

インスタンス生成式:

```
new SuperImp(new Square(new CartPt(20,40),20),
             new Circle(new CartPt(40,30),15))
```

図形の重ね合わせ: インスタンス生成式

以下における sh1, sh2, sh3 はどのような図形を表わしているか?

```
CartPt cp1 = new CartPt(100, 200);
CartPt cp2 = new CartPt(20, 50);
CartPt cp3 = new CartPt(0, 0);

IShape s1 = new Square(cp1, 40);
IShape s2 = new Square(cp2, 30);
IShape c1 = new Circle(cp3,  20);

IShape sh1 = new SuperImp(c1, s1);
IShape sh2 = new SuperImp(s2, new Square(cp1, 300));
IShape sh3 = new SuperImp(s1, sh2);
```

第 6 章 クラス階層の設計

- 第 1 部のまとめ...
- 6.2 事例: 戦う UFO...
- 河川系...

第１部のまとめ

- クラス階層の設計...
- 1. 問題文析...
- 2. クラス図 (データ定義)...
- 3. クラス定義と目的文...
- 4. インスタンス生成式を書く...

クラス階層の設計

- ここまでやったことを「クラス階層の設計」という
  良い設計だと後から拡張・変更するときに楽になる
  良い設計を得る方法を身につけよう!
- クラスの目的: 問題文の世界の情報の集まりを表わす
  情報を表わすことができると計算や解釈ができる
  問題(入力)情報とプログラムが返す解(出力)情報に注目せよ

1. 問題文析

- 問題文をよく読む
- どんな種類の情報を扱うのかを決定する
- 結果: クラス名の一覧、各クラスの短い説明
- 情報の例を書き出す・無ければ作る

2. クラス図 (データ定義)

分析結果をもとにクラス図を描く

- 1. 原始型との対応が明白な場合は、原始型を使う
- 2. 複数の情報から成る情報には新しいクラスを作る
- 3. ある情報 (クラス A) の要素 1 つが複数の情報から成るときは、クラス B を作りクラス A から参照する
- 4. 異なる種類の情報 (クラス) を 1 つのまとまりとして扱いたいときは合併を使う
- 5. 不定個の情報を表わしたいときは自己参照(循環参照)を使う

3. クラス定義と目的文

クラス定義は機械的に得られる

- クラスを表わす箱はクラス定義になる
- 参照を表わす矢印はフィールドの型になる
- 継承を表わす矢印は implements になる
- 目的文を考えて書く

4. インスタンス生成式を書く

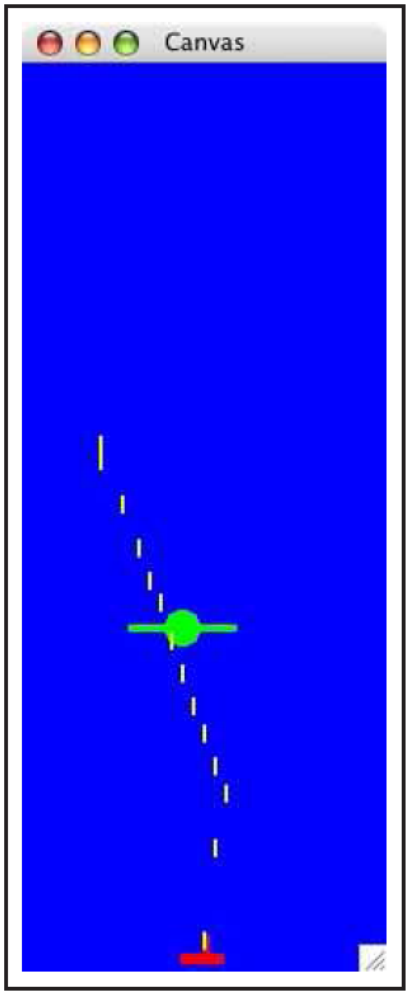
文章や表で書いた情報の例をインスタンス生成式に変換する

- ～Examples クラスを作り、フィールドの値として生成する
- 独自のインスタンス生成式を書き、それを解釈する
  (問題文の世界で何を表わすかを考える)
  ※一般にはクラス表現が許すデータ表現は、
  問題世界に存在するモノよりも広い
- (本授業では)JUnit Test Case としてクラスを作る
  (次ラウンド以降で計算結果を自動チェックするため)

6.2 事例: 戦う UFO

- 問題文...
- 手順...
- それぞれの物体の中身と例を考える
  - ゲーム世界と UFO...
  - AUP、弾リスト、弾...
  - 弾...
- クラス図...
- ライブラリ...
- クラス定義 UFOWorld...
- クラス定義 AUP と UFO...
- クラス定義 Shot と Shot のリスト...
- インスタンス生成式...

問題文

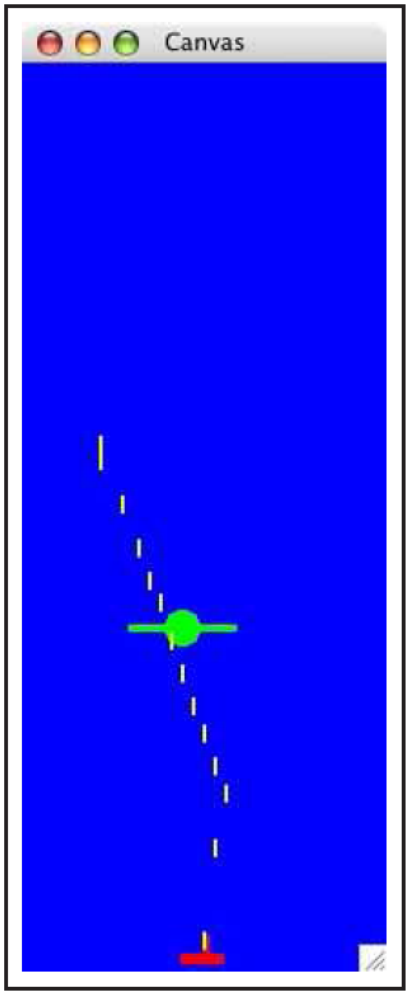


あなたはゲーム開発者です。上司がゲームの初期バージョンを作れと命じました。

例: War of Worlds

… "War of Worlds" ゲームを開発せよ。UFO が 1 つ空から降りてくる。プレーヤは地上を左右に動く対 UFO 砲 (anti-UFO platform; AUP) を使う。AUP は弾を複数を撃つことができ、弾は砲台の中央から垂直に上昇する。弾が 1 つでも UFO に当たればプレーヤの勝ちである。UFO が地上まで降りたら地球が破壊される。
おまけに上司はデザイナにゲーム世界のスケッチを描いてもらった (左図を見よ)。…

手順



問題文を読み、表わすべき情報を洗い出す

- 画面上にあるモノ: UFO, AUP, 弾
- UFO と AUP は 1 つ
- 弾は複数
  - 弾の個数は任意→リストを使う

それぞれの物体の中身と例を考える

ゲーム世界と UFO

- ゲーム世界
  - 中身: UFO, AUP, 弾のリスト
  - 大きさは 200x500、背景色は青
  - 例 1: UFO が上端に、AUP が下端にあるだけ
  - 例 2: これに弾を 2 つ加えたもの
- UFO
  - 上端のランダムな位置に出現
  - 地面まで降りてくる
  - 緑の長方形の中央に緑の円盤を重ねた絵
  - 中身: x,y 座標、 下降速度、水平速度、色 (緑)
  - 例: UFO が (100,10) に出現し、1 単位時刻につき 2 ピクセルの速度で下降、水平方向には動かないとする。つまり、続く時刻には (100,12)、(100,14) のように移動

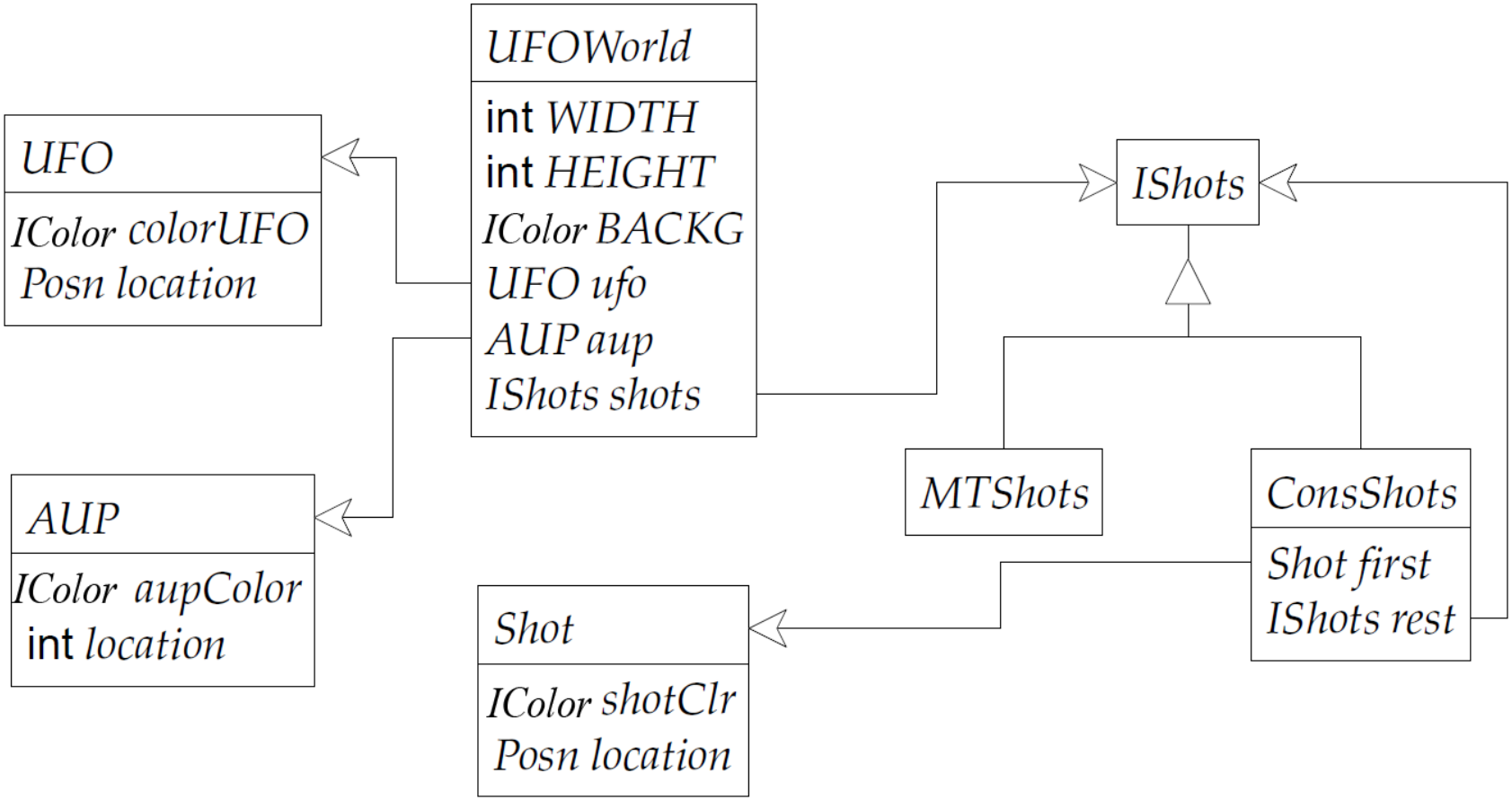
AUP、弾リスト、弾

- AUP
  - キーボードに反応して自走する
  - 左矢印キーで左に数ピクセル移動する (右も同じ)
  - 赤く平らな四角形と短い上向きの四角形の絵
  - 中身: x 座標 (Y 座標は下端に固定)
  - 例: AUP が X 座標 100(で下端付近) にある。左矢印キーを押す 3 ピクセル移動して座標 97 に移る
- 弾リスト
  - (a) 空であるか (b)1 つの弾を他の弾リストに追加したもの
  - 例 1: 空の弾リスト
  - 例 2: 2 つの弾があり、1 つ目が 2 つ目の後に発射されたとする。その間 AUP は動いていない。このときゲーム世界には黄色の四角が 2 つ上下に並んでいる

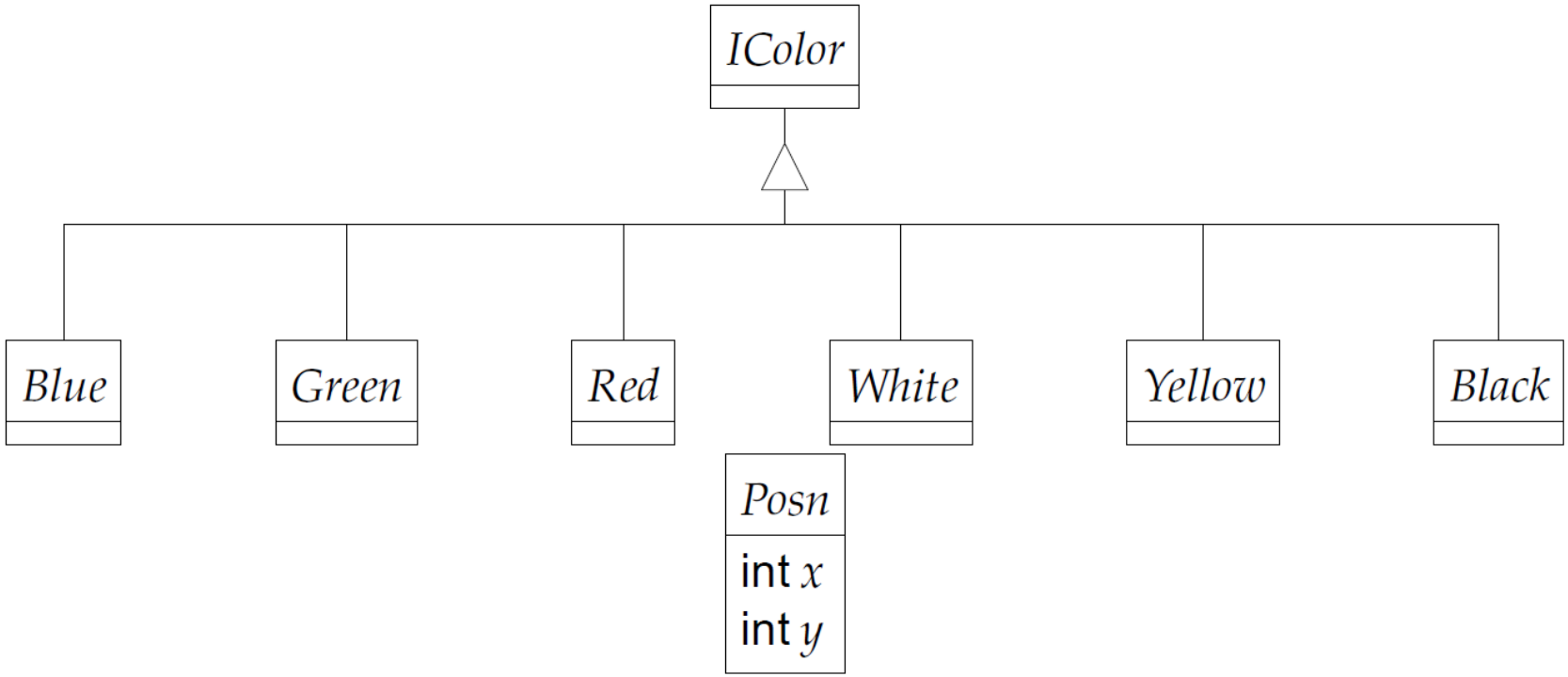
弾

- 弾
  - 上へ移動する
  - 黄色い縦長の四角
  - 中身: 位置を表わす座標、上昇速度 (定数)

クラス図


UFOWorld
int WIDTH
int HEIGHT
IColor BACKG
UFO ufo
AUP aup
IShots shots
UFO
IColor colorUFO
Posn location
AUP
IColor aupColor
int location
IShots
MTShots
ConsShots
Shot first
IShots rest
Shot
IColor shotClr
Posn location

ライブラリ



色や位置を表わすクラス (IColor, Posn) は色々な場面で使われる

- ライブラリとして提供 (Java では package という)
- import geometry.*; import colors.*; とクラス定義ファイルの先頭に書くと使える

クラス定義 UFOWorld

```
import colors.*;     //以降では略すが実際には必要
import geometry.*; //同上

//the world of UFOs, AUPs, and Shots
class UFOWorld {
    UFO ufo;
    AUP aup;
    IShots shots;
    IColor BACKG = new Blue();
    int HEIGHT = 500;
    int WIDTH = 200;

    UFOWorld(UFO ufo, AUP aup, IShots shots) {
        this.ufo = ufo;
        this.aup = aup;
        this.shots = shots;
    }
}
```

クラス定義 AUP と UFO

```
//an AUP: a rectangle, whose upper left corner is
//located at (location, bottom of the world)
class AUP {
    int location;
    IColor aupColor = new Red();
    AUP(int location) {
        this.location = location;
    }
}
```

```
//a UFO, whose center is located at location
class UFO {

    Posn location;
    IColor colorUFO = new Green();

    UFO(Posn location) {
        this.location = location;
    }
}
```

クラス定義 Shot と Shot のリスト

```
//managing a number of shots
interface IShots {}
```

```
//the empty list of shots
class MTShots implements IShots {
    MTShots() {}
}
```

```
//a shot in flight , whose upper left
//corner is located at location
class Shot {
    Posn location;
    IColor shotColor = new Yellow();

    Shot(Posn location) {
        this.location = location;
    }
}
```

```
//a list with at least one shot
class ConsShots implements IShots {
    Shot fst;
    IShots rst;

    ConsShots(Shot fst, IShots rst) {
        this.fst = fst;
        this.rst = rst;
    }
}
```

インスタンス生成式

```
class UFOWorldExamples {
    // an anti−UFO platform placed in the center:
    AUP a = new AUP(100);
    // a UFO placed in the center, near the top of the world
    UFO u = new UFO(new Posn(100,5));
    // a UFO placed in the center, somewhat below u
    UFO u2 = new UFO(new Posn(100,8));
    // a Shot, right after being fired from a
    Shot s = new Shot(new Posn(110,490));
    // another Shot, above s
    Shot s2 = new Shot(new Posn(110,485));
    // an empty list of shots
    IShots le = new MTShots();
    // a list of one shot
    IShots ls = new ConsShots(s,new MTShots());
    // a list of two shots, one above the other
    IShots ls2 = new ConsShots(s2,new ConsShots(s,new MTShots()));
    // a complete world, with an empty list of shots
    UFOWorld w = new UFOWorld(u,a,le);
    // a complete world, with two shots
    UFOWorld w2 = new UFOWorld(u,a,ls2);
}
```

河川系

- 問題文...
- 小さな例を作る...
- 何を１つの「川」にするか?...
- クラス図を描く...
- クラス定義への変換 (1/2)...
- クラス定義への変換 (2/2)...
- インスタンス生成式を書く...

問題文

例:

…環境省は河川系の水質を監視している。河川系は川、その支流たち (tributaries)、支流の支流たち、等々から成る。支流が川に流れ込む場所を合流点 (confluence) という。川の終点 (海か他の川につながる) は河口 (mouth) という。川の始点は水源 (source) という。…

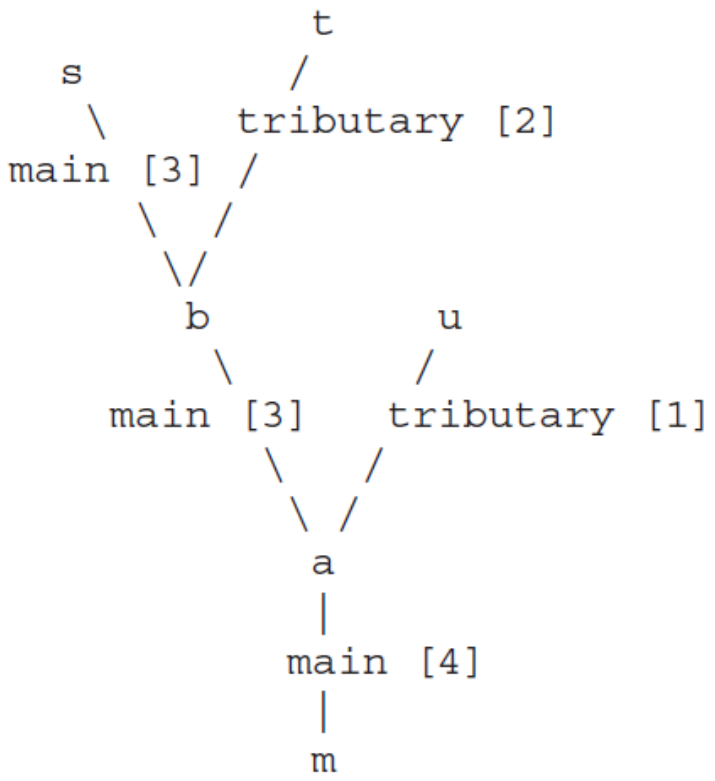
複雑な説明をどう扱えばよいか?
→ 小さな例を作る!

- 問題文をカバーできる要素が全部入っている

小さな例を作る

例:

…環境省は河川系の水質を監視している。河川系は川、その支流たち、支流の支流たち、等々から成る。支流が川に流れ込む場所を合流点という。川の終点 (海か他の川につながる) は河口という。川の始点は水源という。…

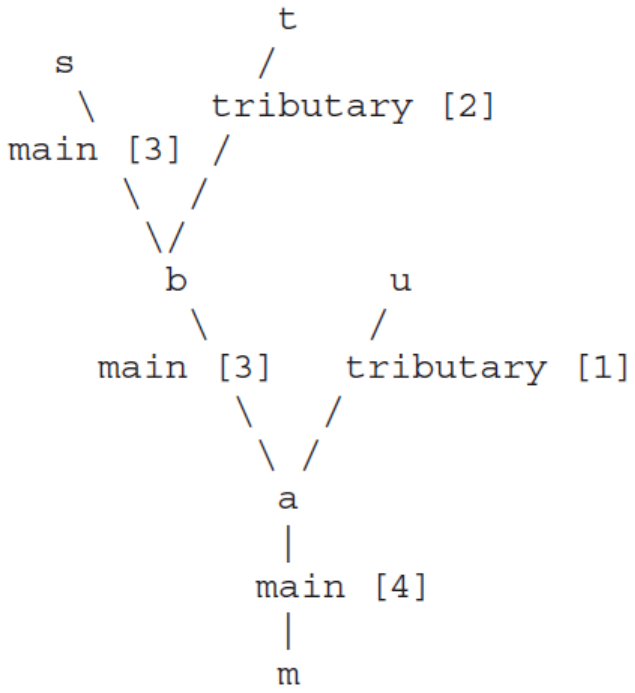


一通りの要素があること

- 海で終わるのが 1 つ
- 合流点から始まるもの
- 水源から始まるもの
- 合流点から始まり合流点で終わるもの

何を 1 つの「川」にするか?

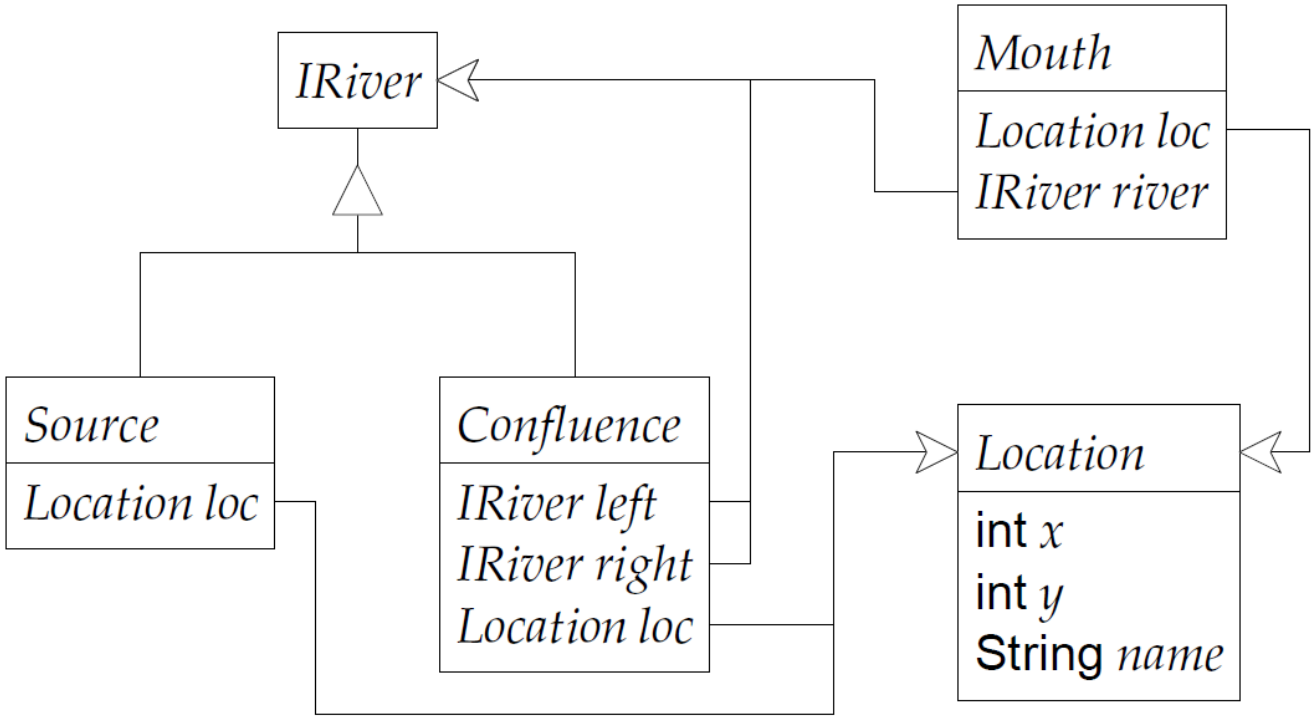
- 1 つの川 =
  (水源 または 上流の川) +
  (下流の川 または 河口)
- 必要がない限り重複した情報を持たないようにする
- 「下流が上流を知っている」
  または
  「上流が下流を知っている」
  のどちらかで充分
- 今回は前者にする
  (どちらが良いかの議論は略)



クラス図を描く

分析

- 川 (の 1 区間) には水源から始まるものと合流点から始まるものの 2 種類 (合併!)
- 合流点には流入元となる 2 つの川 (の区間) がある
- 1 つの河川系は河口で代表される



クラス定義への変換 (1/2)

```
// the end of a river
class Mouth{
  Location loc;
  IRiver river;

  Mouth(Location loc, IRiver river){
    this.loc = loc;
    this.river = river;
  }
}
```

```
// a location on a river
class Location{
  int x;
  int y;
  String name;

  Location(int x, int y, String name){
    this.x = x;
    this.y = y;
    this.name = name;
  }
}
```

```
// a river system
interface IRiver{ }
```

クラス定義への変換 (2/2)

```
// the source of a river
class Source implements IRiver {
  Location loc;

  Source(Location loc){
    this.loc = loc;
  }
}
```

```
// a confluence of two rivers
class Confluence implements IRiver{
  Location loc;
  IRiver left;
  IRiver right;

  Confluence(Location loc,
             IRiver left,
             IRiver right){
    this.loc = loc;
    this.left = left;
    this.right = right;
  }
}
```

インスタンス生成式を書く

```
class RiverSystemExample {
  Location lm = new Location(7, 5, "m");
  Location la = new Location(5, 5, "a");
  Location lb = new Location(3, 3, "b");
  Location ls = new Location(1, 1, "s");
  Location lt = new Location(1, 5, "t");
  Location lu = new Location(3, 7, "u");
   IRiver s = new Source(ls);
   IRiver t = new Source(lt);
   IRiver u = new Source(lu);
   IRiver b = new Confluence(lb,s,t);
   IRiver a = new Confluence(la,b,u);
  Mouth mth = new Mouth(lm,a);
  RiverSystemExample() { }
}
```