オブジェクト指向開発論

2020年5月28日

海谷 治彦

目次

- ちょいと研究室配属の宣伝
  - 水 PM8-10 に Zoom でオフィスアワー
- 構造化手法 VS オブジェクト指向
- ICONIXの概要
- ドメインモデル

復習

ソフトウェアの整理法

- オブジェクト指向
  - この授業のメインテーマ

- 構造化手法
  - 機能分解，手続き型とも呼ばれる.
  - C言語等の従来型ソフトウェアの基本概念.
  - オブジェクト指向の中でも利用されている.

復習

構造化手法とその問題点

- 基本理念
  - データ構造 ＋ 処理 ＝ ソフトウェア
- 機能(関数)をより簡単な機能群へ構造的に分解することで，ソフトウェアを整理する伝統的な手法.
- C言語やアセンブラ等，歴史の古い言語の基本理念となっている.

- データと処理の関係付けが希薄.
- 上記の理由から，ソフトウェア中の部品(関数やデータ構造)をグループ化しても，グループの結束が弱い.

復習

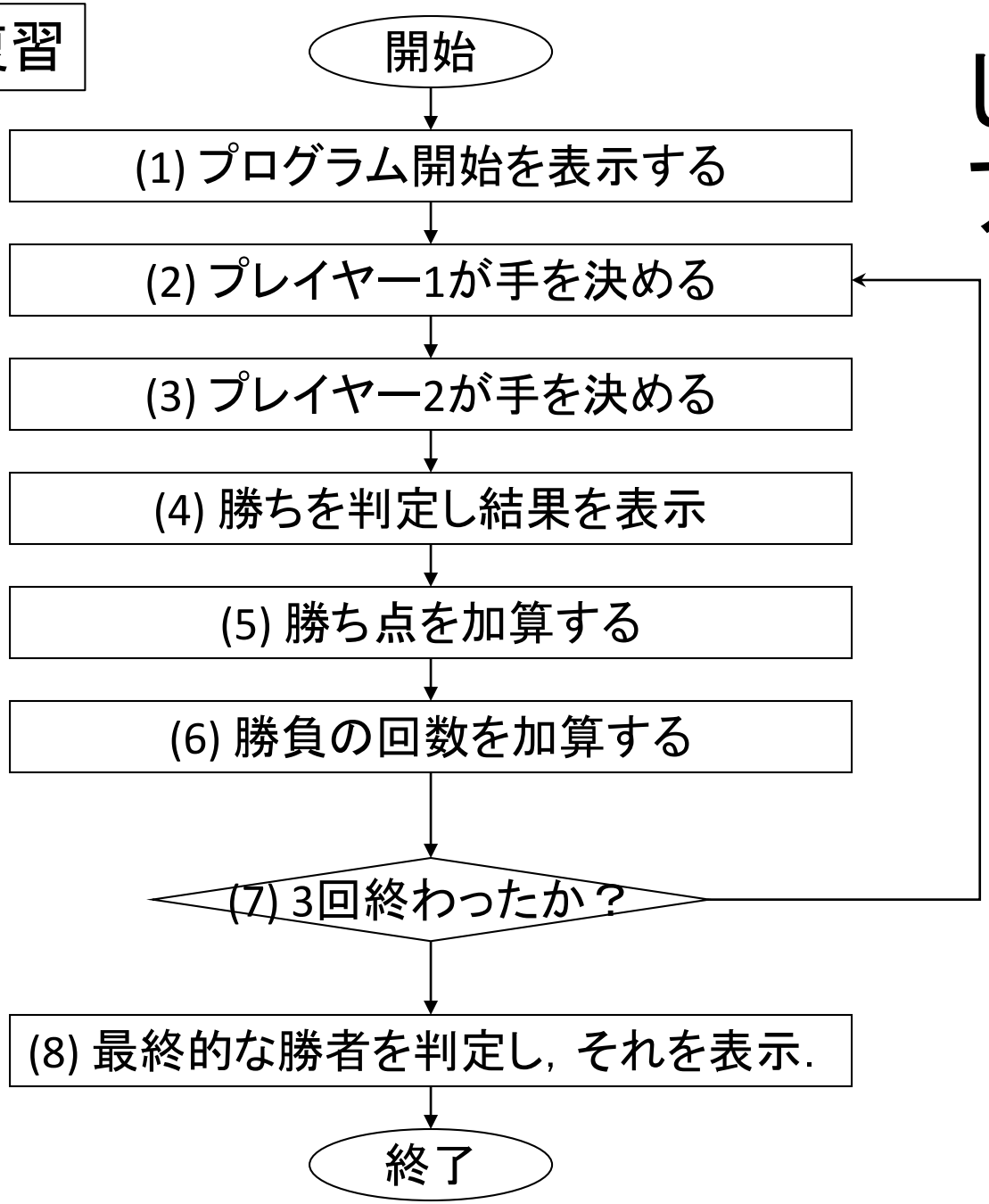
Java即オブジェクト指向では無い

- dotcampusのほうにJavaで書いた，オブジェクト指向ではないプログラムを挙げる.
- これを用いて，Java即オブジェクト指向では無いことを説明する.
- SimpleJanken.java
  - じゃんけんの振る舞いを模倣するプログラム
- SimpleMath.java
  - 簡単な数値計算，整数の平方根と最大公約数.

- 上記はそれぞれ構造化手法に基づきかかれている.

復習

じゃんけんのアルゴリズム



構造化じゃんけんの問題点

- 以下のような修正をするのが困難 (大幅修正か作り直しが必要)

1. 「プレーヤー1」等の味気ない名前ではなく，「村田さん」等の名前で表示したい．
2. 二人じゃんけんだけでなく，三人，四人・・・じゃんけんにも対応したい．
3. 手の出し方をプレーヤー毎に変えたい (現状では皆ランダム)

ソフトウェアと修正

- ほとんどのソフトウェアは最初の開発後に機能修正を要求されることが多い，理由は，

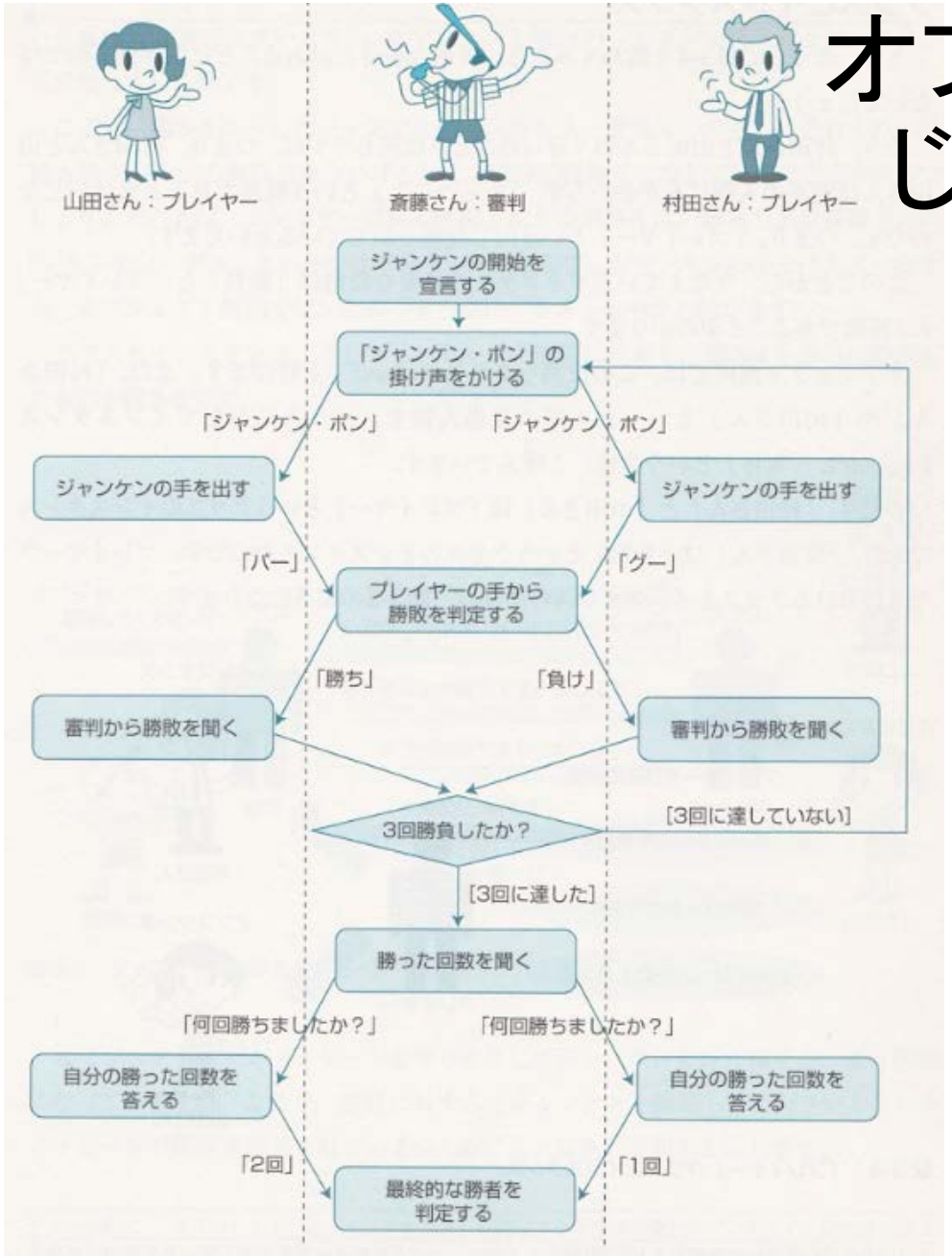
1. 同じソフト(例えば物品販売支援など)でも，使う会社によってカスタマイズする場合がある.
2. 時の変化によって，利用者の要望が変わったり，基盤(OSやネットワークライブラリ)が変化したりする.

- 修正しやすいようにソフトウェアを開発するのは，必須であり，オブジェクト指向や(設計)モデルを用いると，構造化手法よりは，修正しやすい作りになる.

オブジェクト指向開発の指針

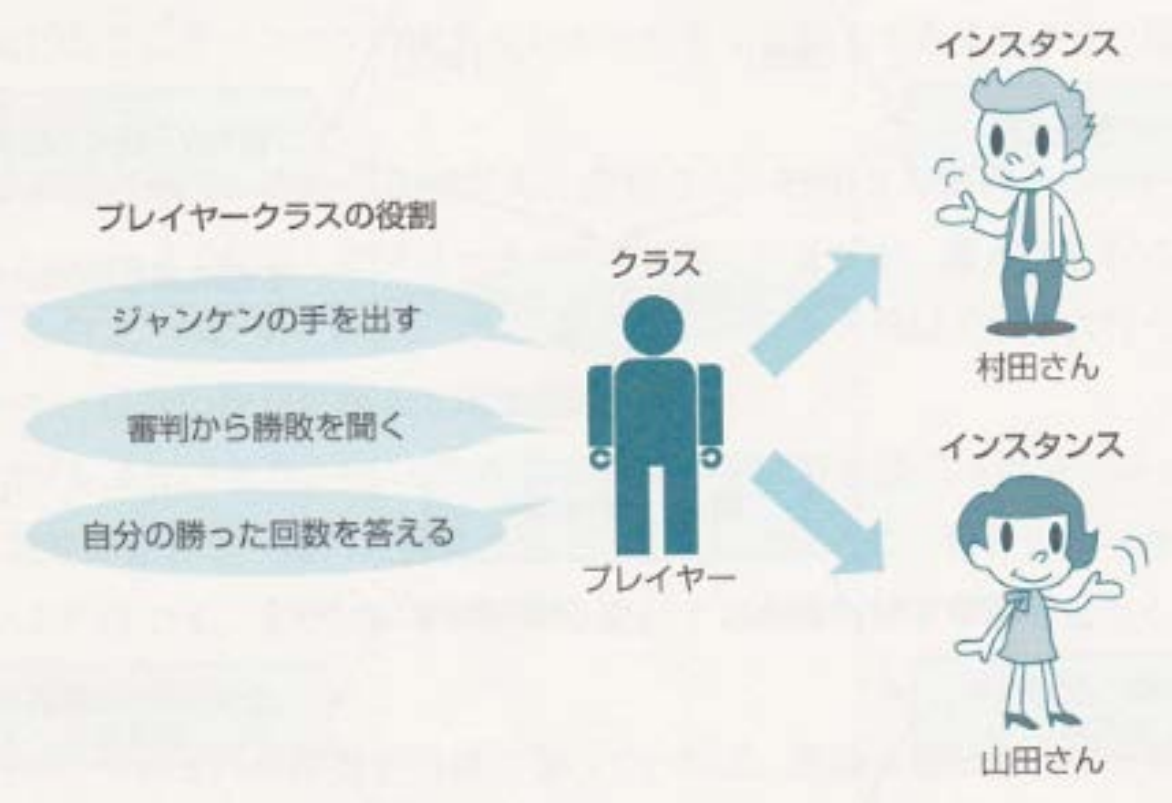
1. システム化対象の中で「役割」に相当するものを見出す.
   - 役割をクラスと呼ぶ
2. 役割毎に実行できる機能群を明らかにする.
   - 機能をメソッドと呼ぶ
3. 役割間の機能の呼び出し関係を明らかにする.
   - ある役割が他の役割に機能遂行を委譲する.
   - この関係をクラス図にまとめる
4. 機能群を実現するために役割が保持すべきデータを明らかにする.
   - このデータを属性(アトリビュート)と呼ぶ

オブジェクト指向でじゃんけんを観る



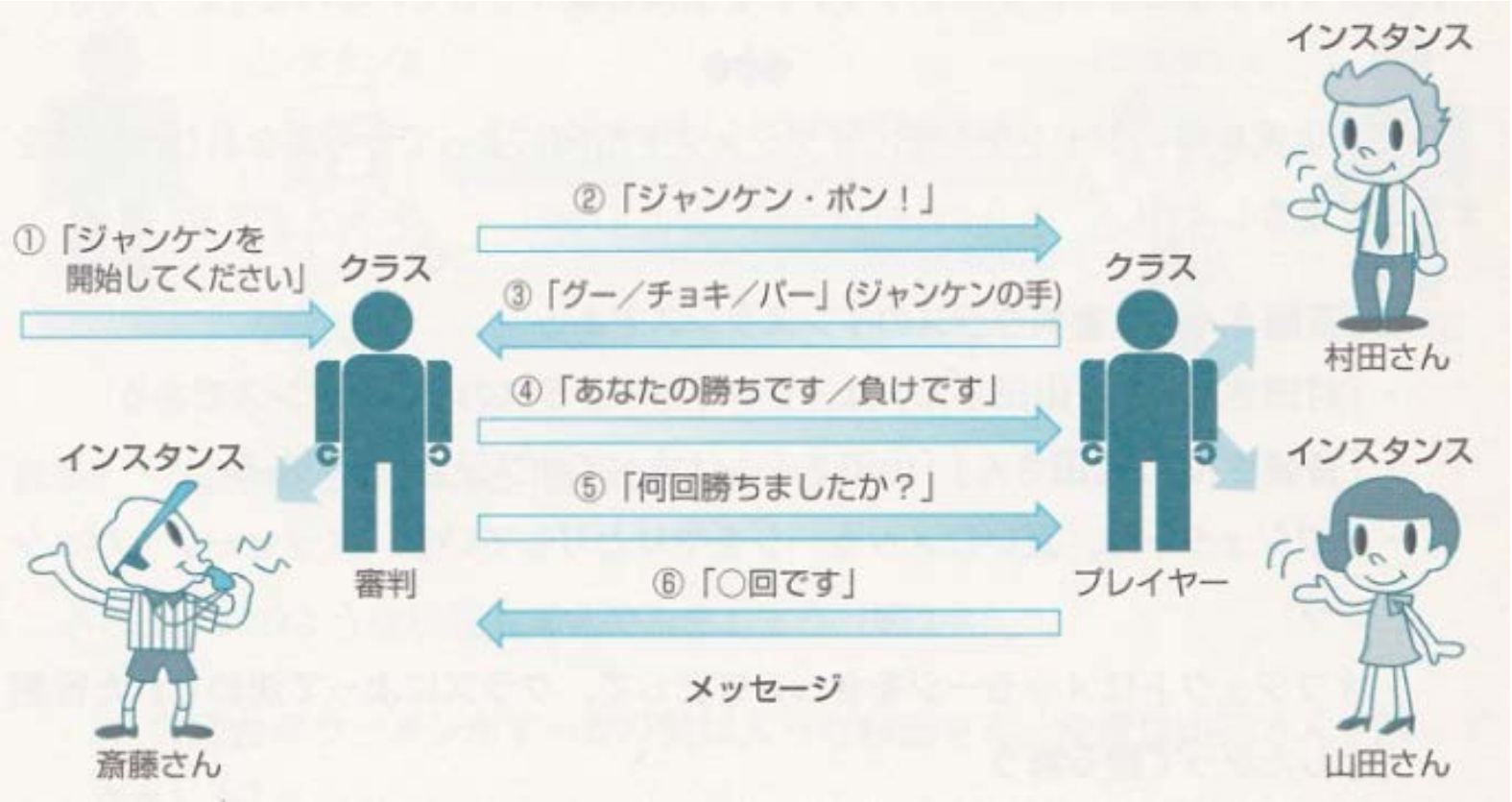
何故Java 図3-4

クラスとインスタンス



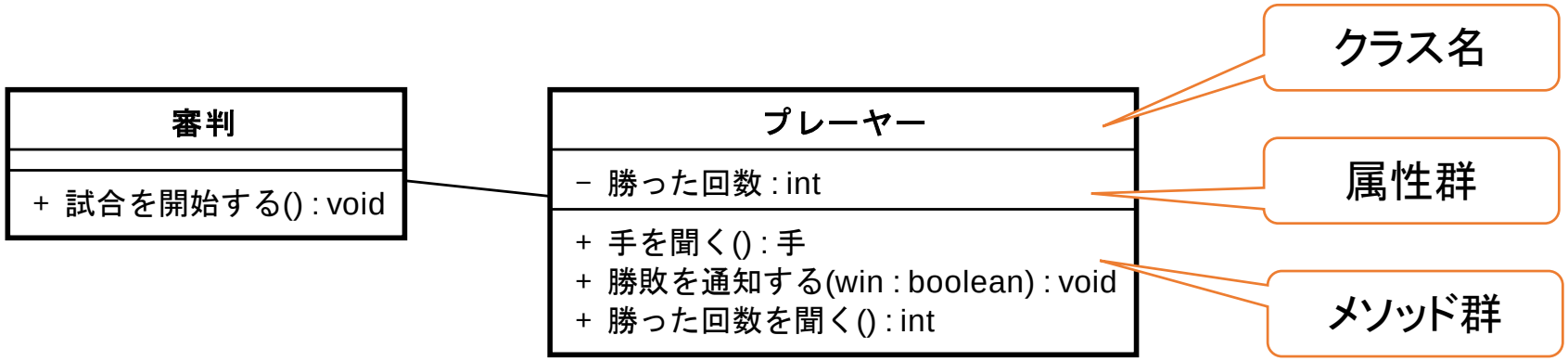
何故Java 図3-5

クラス間の呼び出し関係



何故Java 図3-6

クラス図



TIPS

- メソッドは，それを包含するクラス以外が呼ぶことが多い．
  - 結果として，メソッドの命名は，主語が自身以外の動詞(句)となる．
  - 例「審判はプレーヤーから勝った回数を聞く」
    - 「勝った回数を答える」としない．

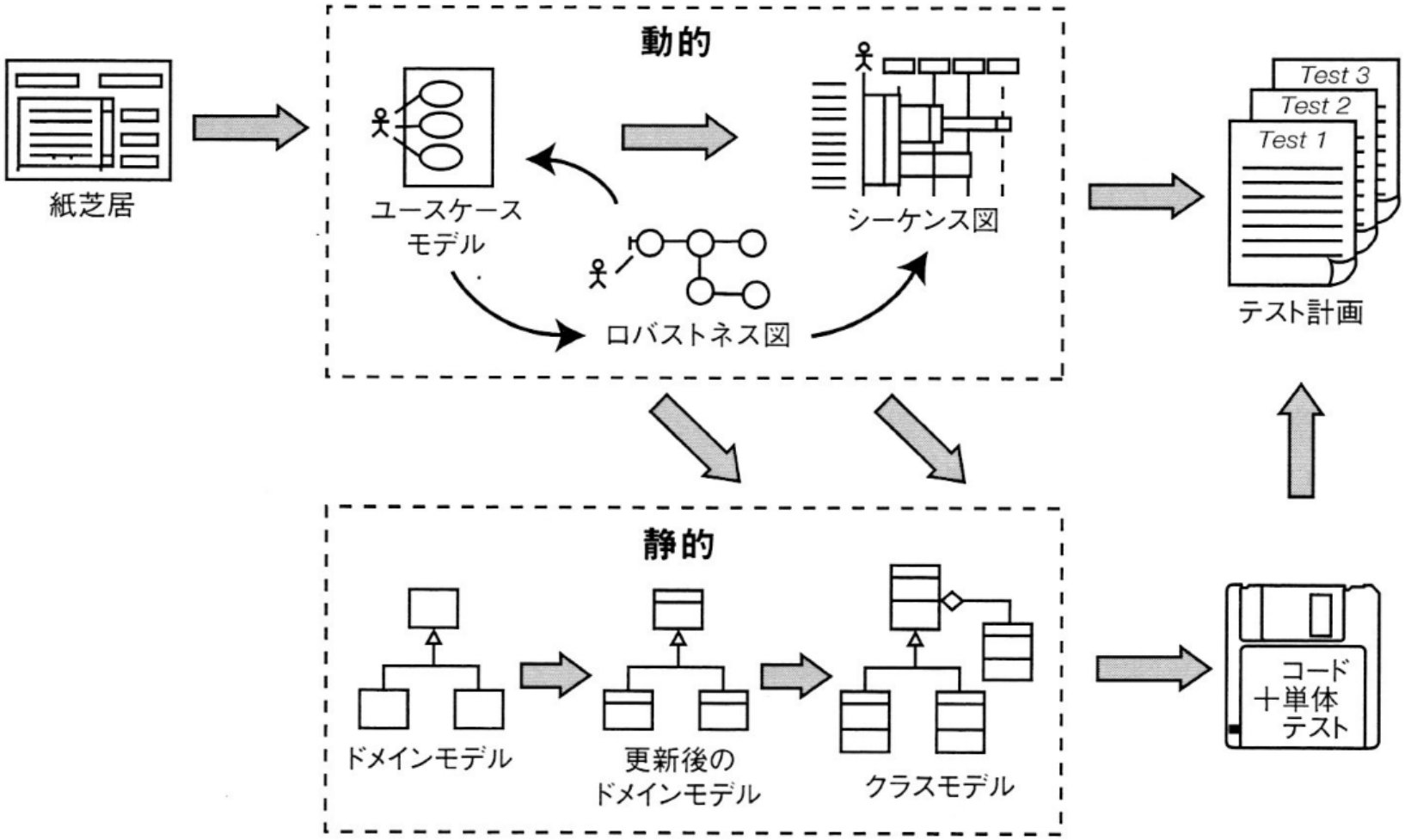
getWinCount 等

- クラス図のクラスはJavaやC++, C#のクラスと対応する．

開発手法について

- 料理でも完成品の成分や構造，見た目が分かっていて，それが作れるわけではない.
  - クックブックやレシピが通常，必要.
- ソフトウェアも同様で，完成品の文法がわかっていても，ソフトウェアが作れるわけではない.
  - Cの文法がわかっているだけでは，複雑なプログラムは通常組めない.
- ICONIXはオブジェクト指向開発を行う代表的な開発手法(手順書，クックブック)として知られている.
  - 単にクラス図やなんとか図の文法を知ってるだけでは，開発はできないため.

ICONIXの全体手順



ICONIX説明のための例題

- インターネット書店システム
  - アマ〇ンみたいなもの
- それぞれのステップの図を説明するのに使う.
- 最初の要求定義書(reqs.docx)はteamsを参照.

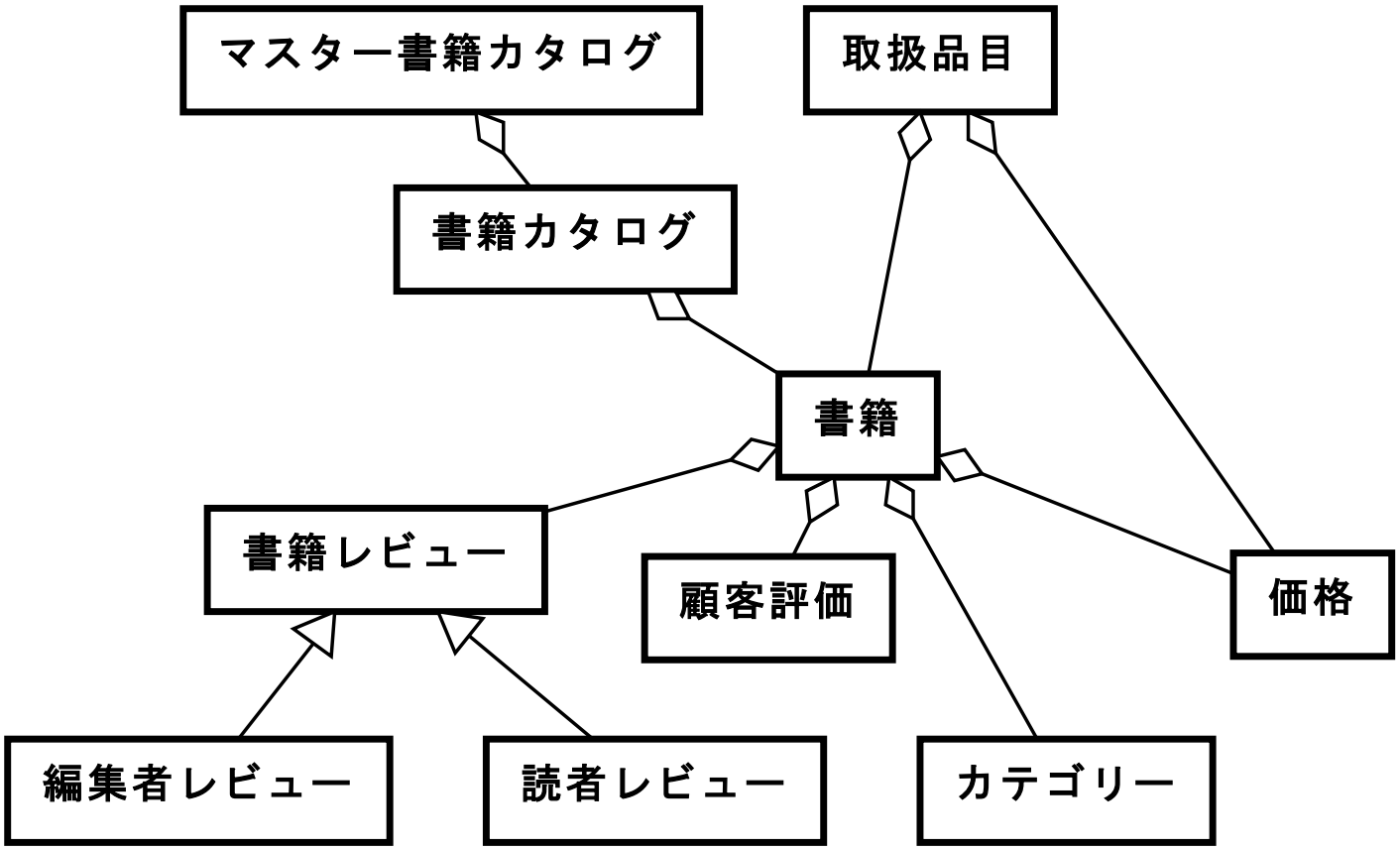
- 要求定義書がちゃんとした文書として整備されていない場合も一般には考えられる.
  - 特に日本の場合.

ドメインモデル

- プロジェクトの用語集，辞書
- 用語間の関係をグラフィカルに定義する.
  - is-a, has-a 関係を用いる.
- クラス図の元になる.

- 次ステップのユースケースモデルを書く際の，記述要素は，このドメインモデルからピックアップする.
- ユースケースおよびその後の手順において，ドメインモデルを修正する.
  - 最初から完璧なドメインモデルはかけない.

ドメインモデルの例


マスター書籍カタログ
取扱品目
書籍カタログ
書籍
書籍レビュー
顧客評価
価格
編集者レビュー
読者レビュー
カテゴリー

ドメインモデルのガイドライン 1/2

1. 現実世界(問題領域)のモノ(オブジェクト)に焦点をあてなさい.
   - システムではなく，対象業務を注視する.
2. オブジェクト同士の関係をis-aとhas-aで整理しなさい.
3. 最初のドメインモデリングは2時間程度で打ち切りなさい.
   - 初期のバージョンは不完全で誤りがあってもOK
4. 問題領域の主要な概念を中心にクラスを構成しなさい.
5. ドメインモデルはデータモデルではありません.
   - 一般にドメインモデルの要素であるクラスのほうが，粒度が細か目.

ドメインモデルのガイドライン 2/2

6. オブジェクトとデータベースのテーブルは別物です.
7. ドメインモデルをプロジェクトの用語集として使いなさい.
8. 名称を統一して使うため，ユースケースを書くより先にドメインモデルを作りなさい.
9. 最終的なクラス図がドメインモデルとかなり変わってもOKです.
10. ドメインモデルには画面やGUI等の実装に依存したクラスは配置しないでください.

最初に何をするか？

1. 要求定義書等から名詞(句)を取り出す.
2. 同じ概念を示す語を探し，代表以外は省く.
3. システムの外部にある者や物はアクターとして記述する. (後述の人型の絵)
4. has-a関係を識別する. (◇の線)
   - 部分-全体関係のこと.
   - 「AがBを持っている」,「BはAの一部である」と呟いて，違和感なければOK

- 判断に困る部分はシステム構築を依頼した顧客やユーザーに聞く.

実際にピックアップした語句群

- 顧客
- 顧客アカウント
- 販売者
- ユーザーアカウント
- アカウントリスト
- マスターアカウントリスト
- マスター書籍カタログ
- 書籍レビュー
- レビューコメント
- 書籍カタログ
- 書籍一覧
- ミニカタログ
- マスターカタログ
- インターネット
- パスワード
- タイトル
- キーワード
- ・・・・・以下略

除外の例

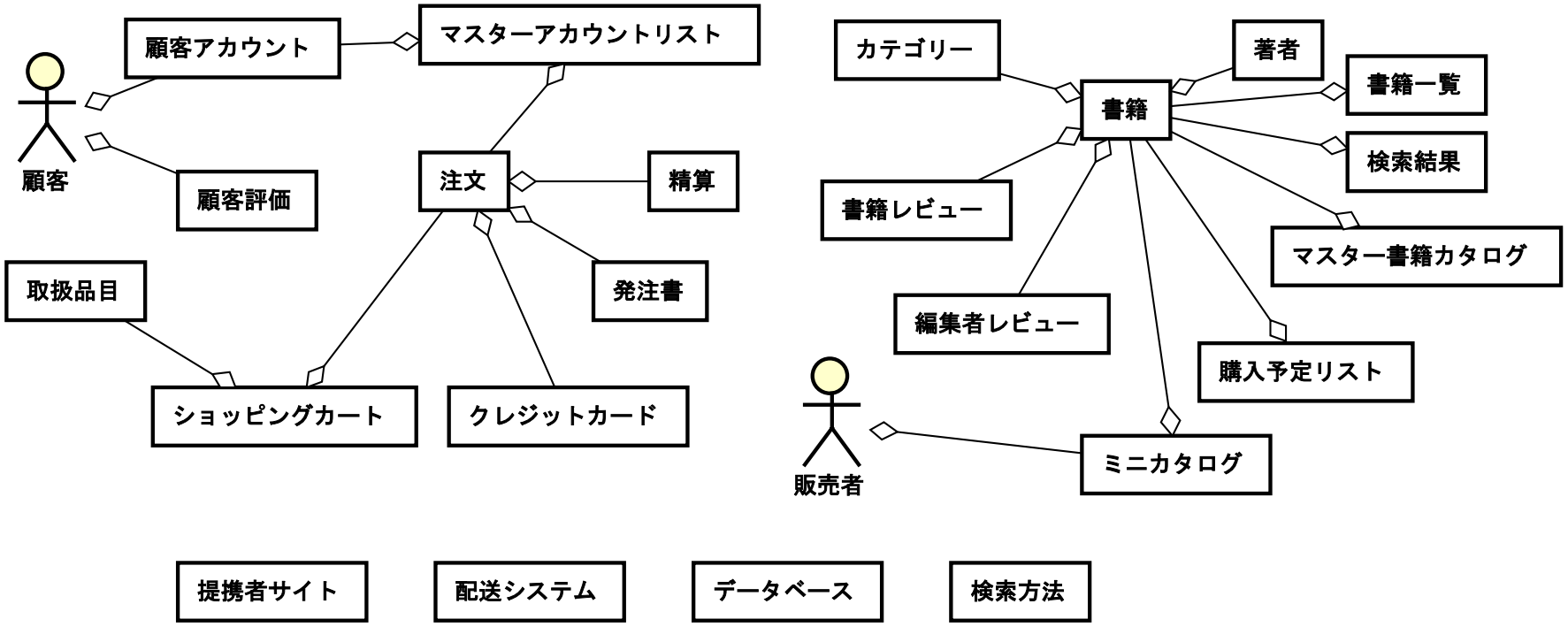
- 「ユーザーアカウント」と「顧客アカウント」は同じなので，例えば後者を残す．
- 「インターネット」は一般的すぎるので除外．
- 「パスワード」，「タイトル」，「キーワード」は概念として小さすぎるので除外．

has-aの例

- 「顧客が顧客アカウントを持っている」は違和感ない.
- 「書籍は書籍一覧の一部である」もおかしくない.

等

version 1 ドメインモデル

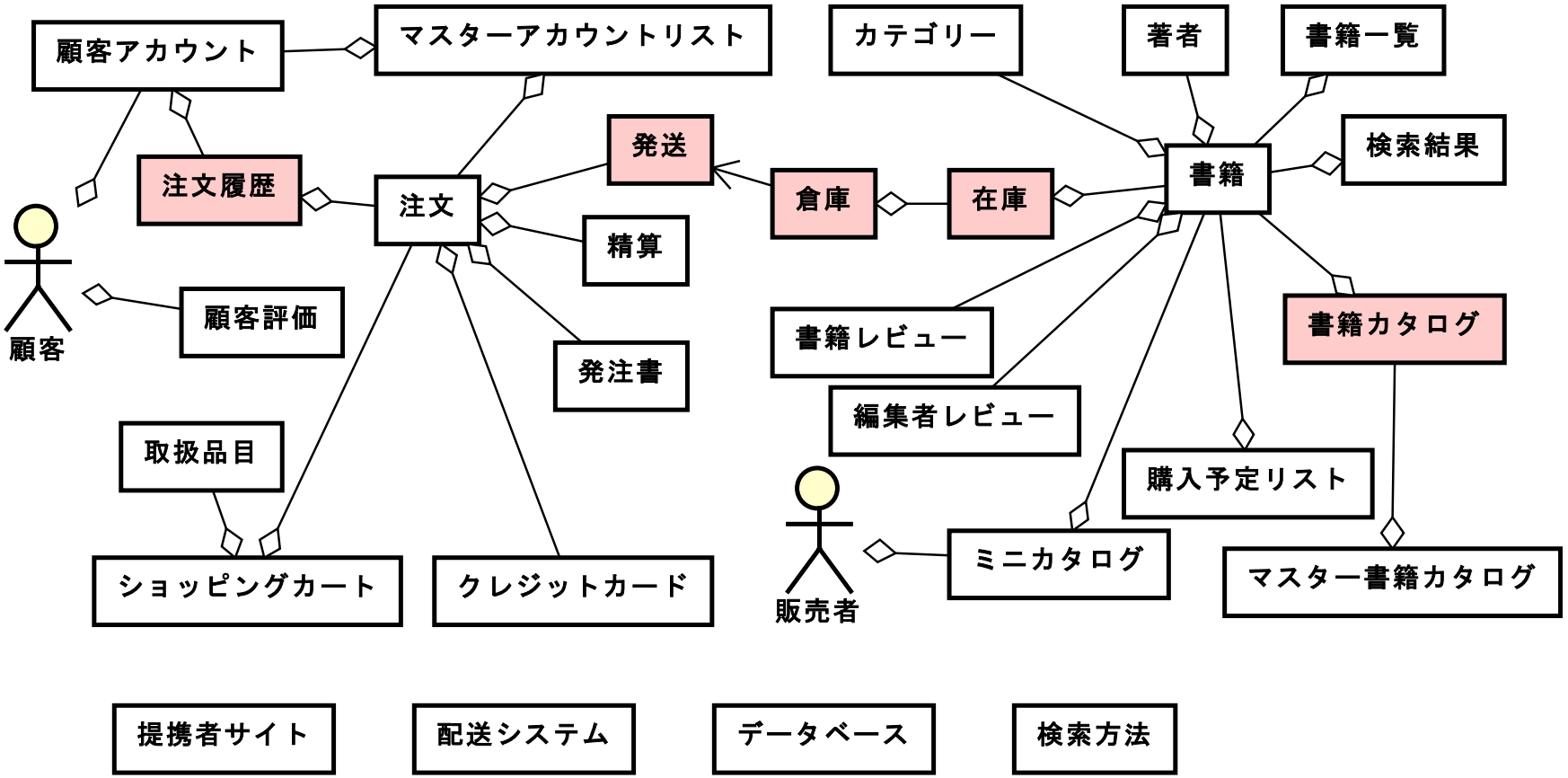


ICONIX 図2.4

不足概念の補完

- 現時点のドメインモデルを見て，不足している概念を補う.
- また，中間概念的なものが必要なら追加する.

version 2 ドメインモデル



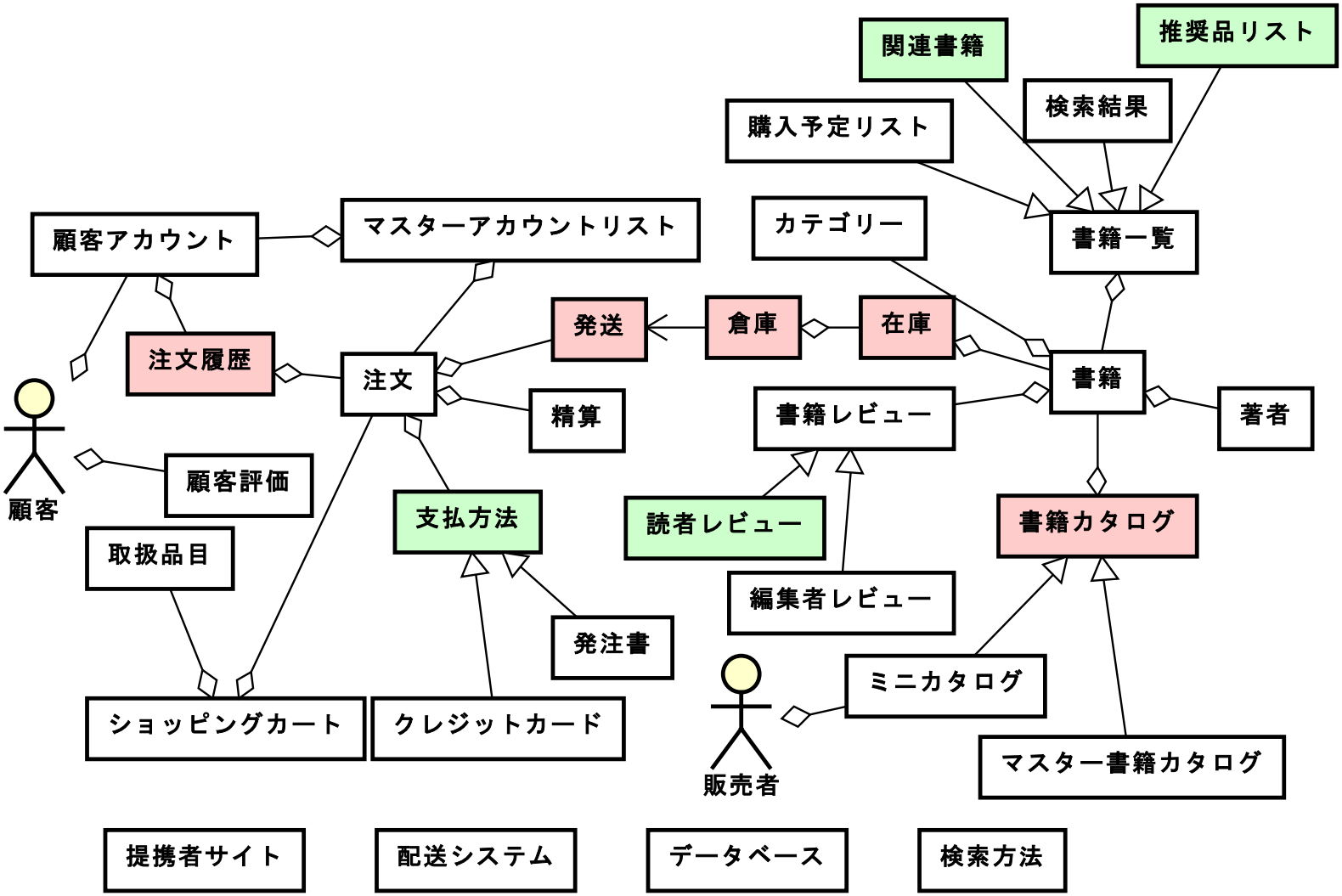
ICONIX 図2.5

is-a関係の導入

- 一般-具体関係の整理.
- 「AはBである」と呟いて違和感が無ければ is-a 関係.
- ドメインモデル(クラス図)では△で書く，△がついてるほうが，一般概念.

- 一般化して束ねたほうがよい概念があればやる程度. そんなに気合をいれてやらなくてもよい.
  - メインはあくまで has-a関係 (部分全体関係)

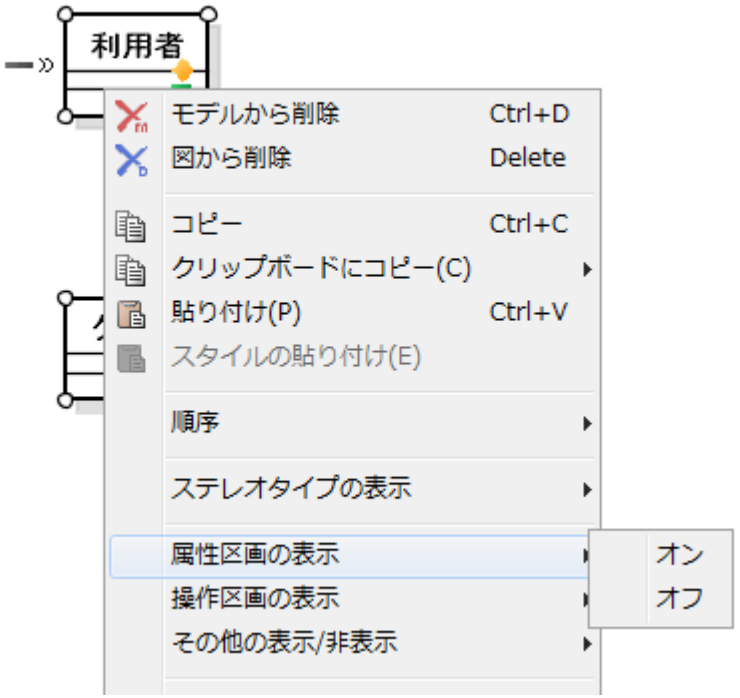
version 3 ドメインモデル



ICONIX 図2.7

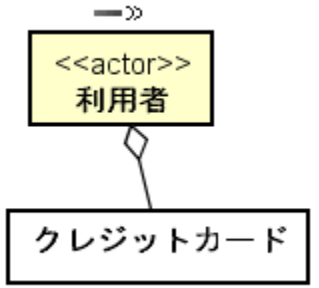
astah TIPS: クラスの簡略表記

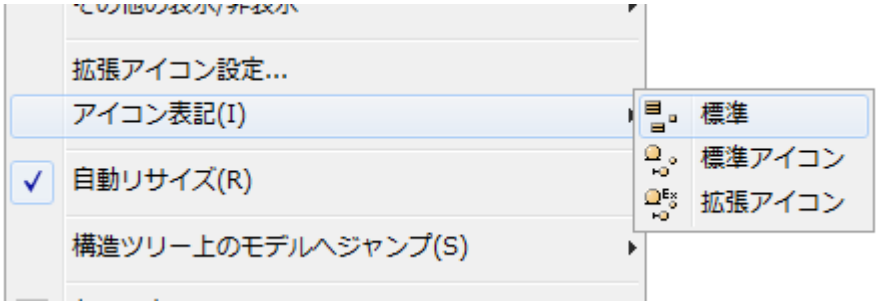
- astahではクラス図を用いてドメインモデルを書く.
- デフォルトのクラス図は三つのパートに分かれている.
- ドメインモデルでは当面，名前だけあればよいので，右のように属性，操作の表示をオフにする.
- とりあえず図を描いて全体選択をしてからだと楽.

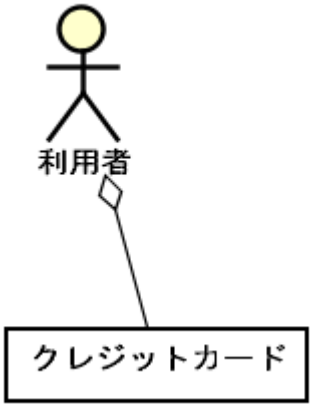


astah TIPS: アクターの人型表示

- クラスを選ぶ.
- ステレオタイプのタブを選ぶ.
- 追加で actor を追加する, これで上図にはなる.
- クラスを再度選択して, メニューを出して, アイコン表記を標準アイコンにする.







名詞抽出法の拡張

- 名詞抽出法(何故Java p.137)によるクラス図の記述手順はある意味妥当.
- しかし，もうちょっと，支援が欲しいところ.
- 金田先生(同志社大)の提案する手法を紹介.

クラスと関連 (従来法)

- クラス
  - 名詞
- 属性
  - 名詞
- 関連
  - 動詞
- メソッド
  - 動詞

クラスと関連 (金田法)

- クラス
  - 可算名詞 (複数形になれる)
- 属性
  - 非可算名詞
- 関連
  - 状態動詞
- メソッド
  - 動作動詞

動作動詞と状態動詞

- 動作動詞 (メソッド)
  - 時間上の始まりと終わりがある.
  - さまざまな動きのまとまりがひとつの動作になっている.
  - 動作を繰り返すことができる.
  - 例: カードを配る，数字を見る.
- 状態動詞 (関連)
  - 時間の始まりと終わりがプログラム動作中には無い. 永続的.
  - 区切りが無いので，動作の繰返しはできない.
  - 例: 持っている，構成する.

英語の5つの基本文型とクラス図

1. S + V
   - 動詞は自動詞. 仕様書ではあまり使われない.
2. S + V + C
   - 同上.
3. S + V + O
   - クラス-関連-クラス を表す場合が濃厚.
   - 加算/非加算, 動作/状態のチェックは必要.
4. S + V + O1 + O2
   - 基本, SVOに同じ.
   - O1とO2がHas-a関係(部分-全体関係)の場合があり.
5. S + V + O + C
   - O, C がHas-a関係の可能性あり.
   - もしくは, CがOの属性である可能性あり.

SVOの例 1

- 「進行役はトランプをシャッフルする. 」
  - A moderator shuffles the cards.
  - 加算名詞 moderator cards ⇒ クラス
  - 動作動詞 shuffle ⇒ メソッド
  - moderator と card の関係は不明.
- プレイヤーは自分の手札を持つ.
  - Each player owns his/her hand.
  - 状態動詞 own ⇒ 関連
  - 加算名詞 player hand ⇒ クラス
  - 典型的な クラス-関連-クラスの表現.

SVOOの例

- 「進行役は全てのプレイヤーにカードを配る」
  - The moderator deals each player cards.
  - deal 動作動詞 ⇒ メソッド
  - moderator, player (O1), card (O2) 加算名詞 ⇒ クラス
  - player と card の関係 ⇒ has-a 関係の可能性
    - 実際には途中に「手札」が挟まることになる.

クラス図記述の指針 (金田法+)

- 現実世界の事物について英語で記述する.
  - もしくは英語的に理解する.
- 英語の5つの基本文型のどれかを認識する.
- 文の上の構造に基づき, (1) 加算名詞, (2) 非加算名詞, (3) 状態動詞, (4) 動作動詞を識別する.
- 上記それぞれを(1)クラス (2)属性 (3)関連 (4)メソッドとする.
- もとの文の共起関係も考慮する.
- 例外は適宜調整する.

参考文献

- 金田 重郎, 世良 龍郎. **認知文法に基づくオブジェクト指向の理解**. 電子情報通信学会技術研究報告, Vol. 111, No. 396, pp. 61-66, Jan. 2012. ISSN 0913-5685, 知能ソフトウェア工学 KBSE2011-63.
  - スライド後半は金田先生のご講演のベースに作成しました.