ソフトウェア工学

2025年9月29日

海谷 治彦

目次

- 学校での問題と現実の問題

- モデリングの視点
- 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
- ユースケースモデル

プログラム開発の入力(問題)

- 学校でのプログラム開発演習の問題文は，いかにも，プログラムを作れという文章.
  - 配列やらポインタやらを使うことを意図した恣意的な文章.
  - 算数の計算問題の延長のようなもの.
  - 処理の目的が意味不明な計算のための計算みたいなもの.
- 現実の開発をおける入力は大きく異なる.
  - 基本的には「業務手順」が書いてある.
  - 長い・・・数ページ～数百ページの文章.
  - どこを機械化するかの指定はあるかもしれないし，無いかもしれない.
  - どうやるかはではなく，何をやってほしいかが書いてある.
  - 別途添付の reqs.docx 参照.

リテラシの問題

演習16 プロジェクト3

- 以下の仕様に基づくプログラムを作成せよ.
- webclass に提出
- プロジェクト名 ex16 の beers
- プログラム名 main.c

あるバーは以下の3種類のビールを販売している. (商品番号 銘柄名 単価 原産国)
1 Guinness 600 Ireland
2 Heineken 500 Netherlands
3 Erdinger 700 Deutschland

プログラムは上記の商品の番号, 銘柄名, 単価(日本円JPY), 原産国を列挙する.
利用者は商品番号もしくは0を入力する.
1から3の商品番号が入力された場合, 当該商品を1杯購入したとみなし, プログラムは購入数を更新し, 上記の商品列挙の処理に戻る.
0が入力された場合, 購入した, それぞれの商品の名前, 購入数と全商品の合計金額(税金は考慮しない)を表示し, プログラムを終了する.
0から3以外の数値が入力された場合, 何もせず, 上記の銘柄列挙の処理に戻る.

業務としての開発では，
問題文（要求と仕様）の理解，
場合によっては作成が必要

モデリングと分析

- 機械化する対象業務で何をやっているかを理解するのにモデルを書くのが有効.
- 書いたモデルを使うと，何をどう機械化するかの検討（分析）もし易い.

- 無論，文章でも業務の理解と分析は不可能では無いが，モデルを使ったほうが効率的.

モデリングの代表的な側面

- 構造的側面
  - 現実世界のどんなモノが当面コンピュータで行いたいことに関係するか，それらの(静的な)関係は何かを明確にする.
  - 通常，クラス図を利用.
- 機能的側面
  - 現実世界の事象(コンピュータへの入力)に対して，コンピュータは何を起こすかを明確にする.
  - 通常，ユースケースモデルを利用.
- 振る舞い的側面
  - 機能の実行順序をモデル化.
  - 通常，ステートマシン図，アクティビティ図，シーケンス図を利用.

ソフトウェアの整理法

- オブジェクト指向
  - 現在，最もポピュラーなもの.
  - 工学的には理由がある.

- 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.

構造化手法とその問題点

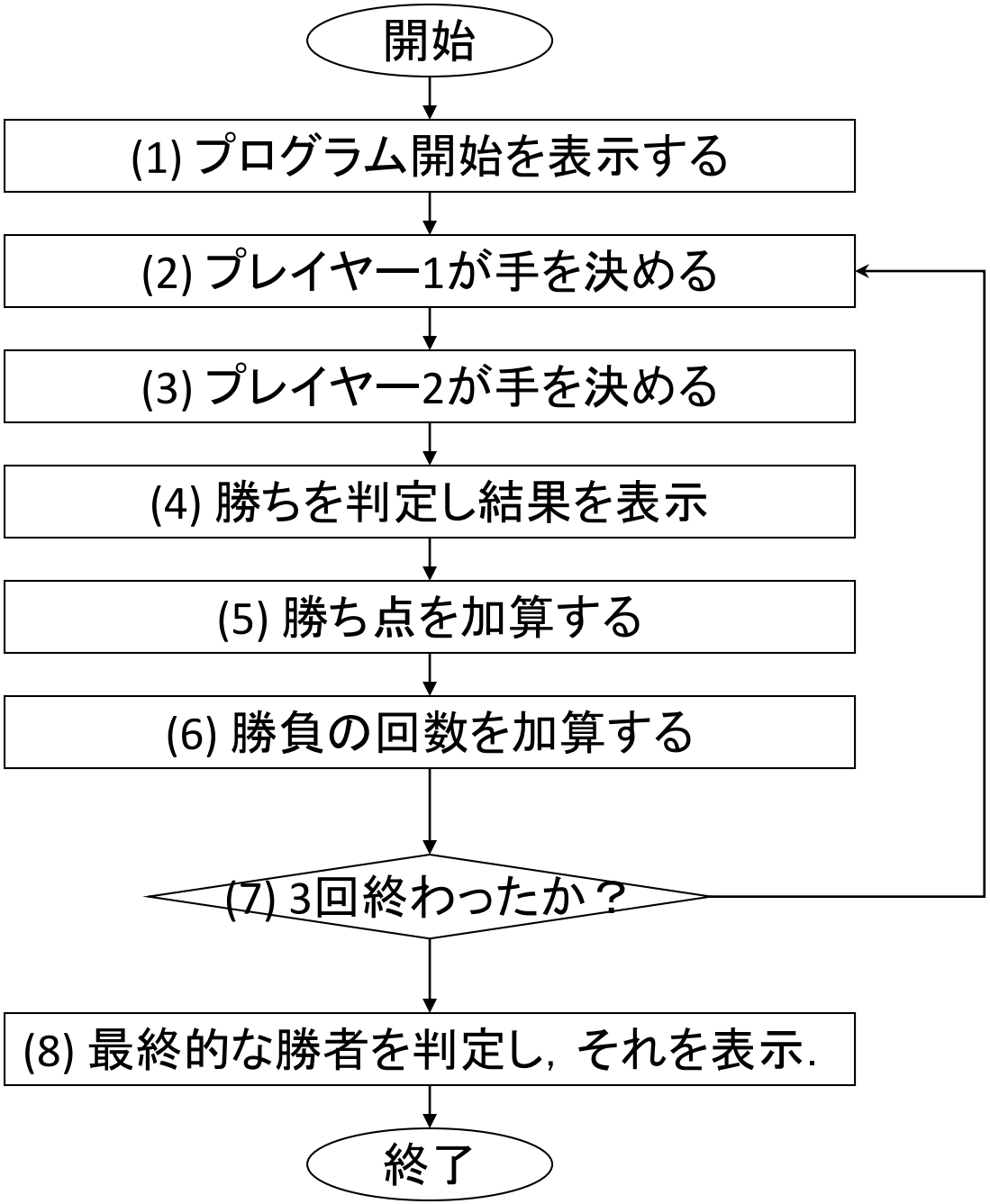
- 基本理念
  - データ構造 ＋ 処理 ＝ ソフトウェア
- 機能(処理)をより簡単な機能群へ階層的に分解することで，ソフトウェアを整理する伝統的な手法．
- C言語やアセンブラ等，歴史の古い言語の基本理念となっている．

- 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．

オブジェクト指向

基本的に以下の二つのアイディアの融合

- 抽象データ型 (Abstract Data Type, ADT)
  - データを直接的なデータ格納表現では無く,
  - データに適用できる関数(群)として定義する考え方.
  - 利点: データ表現という実現方法に縛られずに, データとは何かを定義できる.
- 汎化, 継承に基づく差分記述
  - ADTは関数の集合であるが,
  - その集合に新しい関数を追加したり,
  - 既存の関数の処理内容を更新したりすることで,
  - 新しいデータ型を定義する考え方.
  - 利点: 新しい定義をする際に差分のみ書けばよいので楽.



例
じゃんけんの
手順
構造化手法版

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，

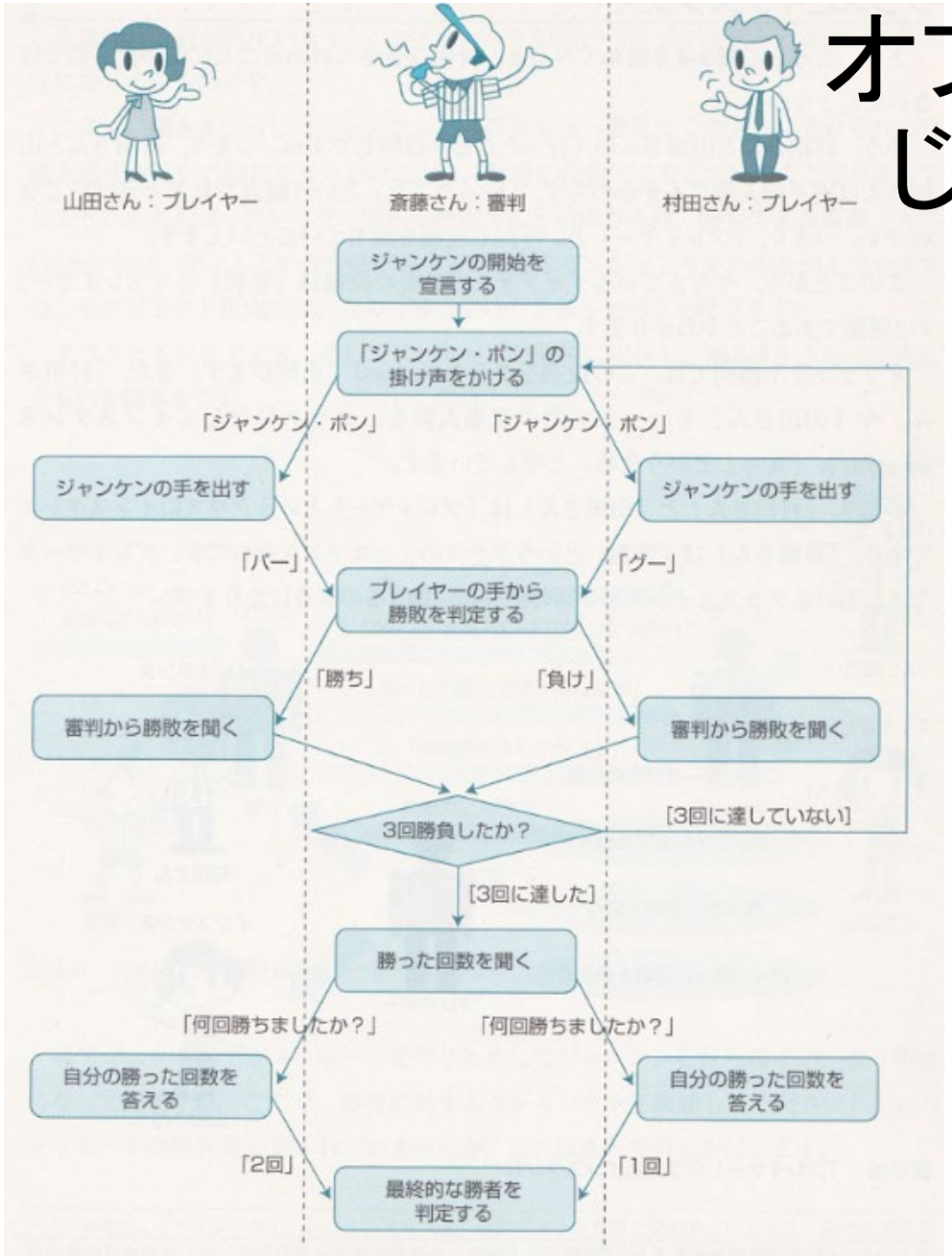
1. 同じソフト(例えば物品販売支援など)でも，使う会社によってカスタマイズする場合がある.
2. 時の変化によって，利用者の要望が変わったり，基盤(OSやネットワークライブラリ)が変化したりする.

- 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

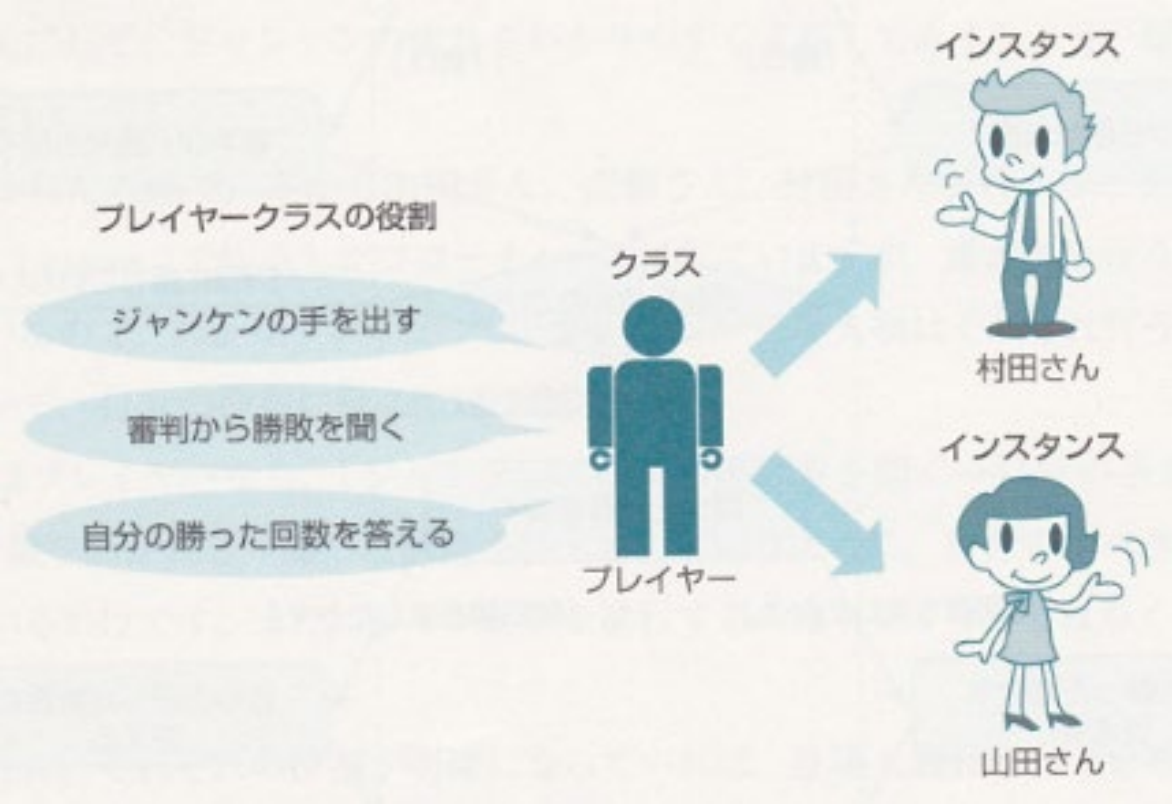
1. システム化対象の中で「役割」に相当するものを見出す.
   - 役割をクラスと呼ぶ
2. 役割毎に実行できる機能群を明らかにする.
   - 機能をメソッドと呼ぶ
3. 役割間の機能の呼び出し関係を明らかにする.
   - ある役割が他の役割に機能遂行を委譲する.
   - この関係をクラス図にまとめる
4. 機能群を実現するために役割が保持すべきデータを明らかにする.
   - このデータを属性(アトリビュート)と呼ぶ

オブジェクト指向でじゃんけんを観る



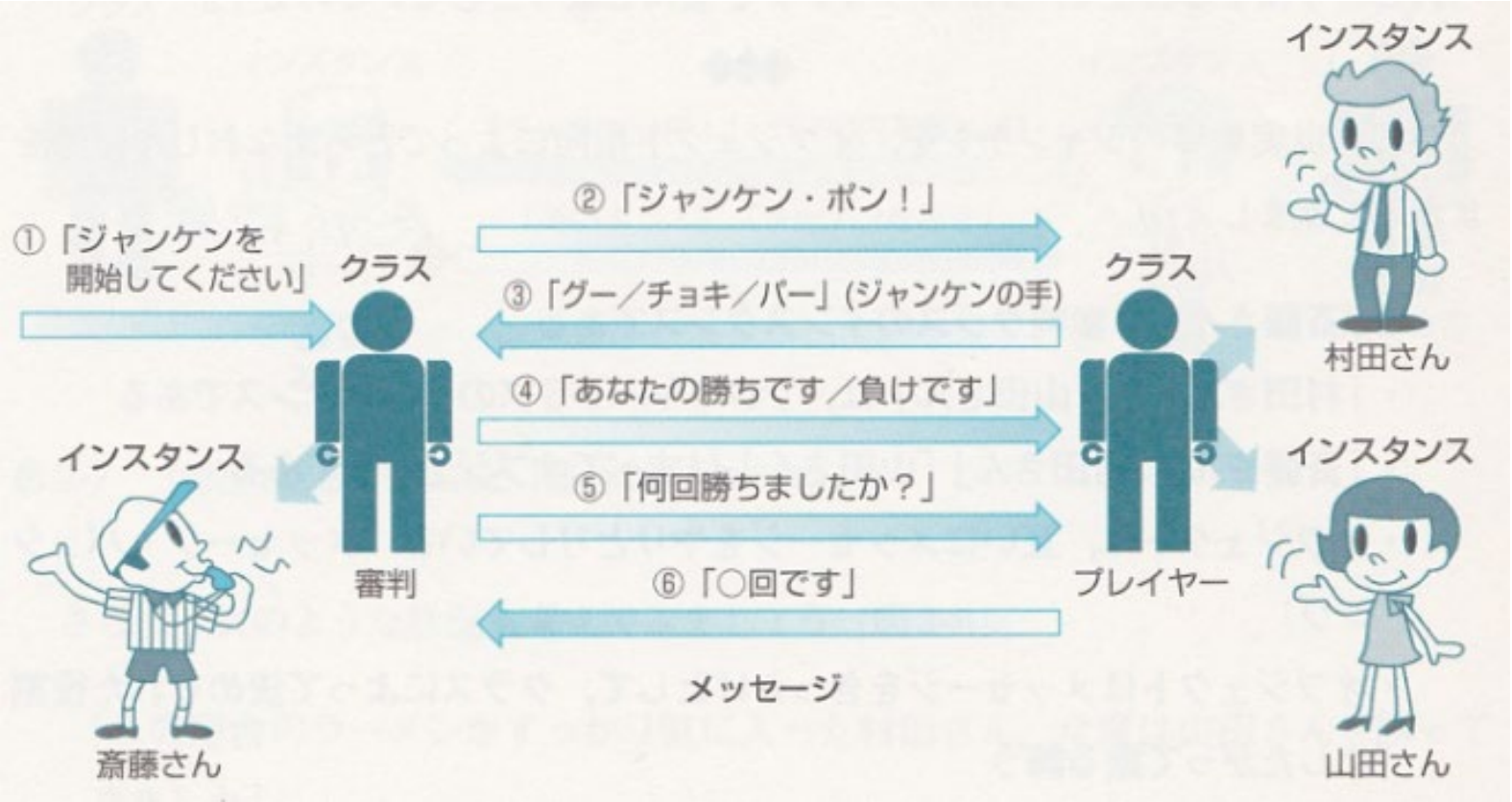
何故Java 図3-4

クラスとインスタンス



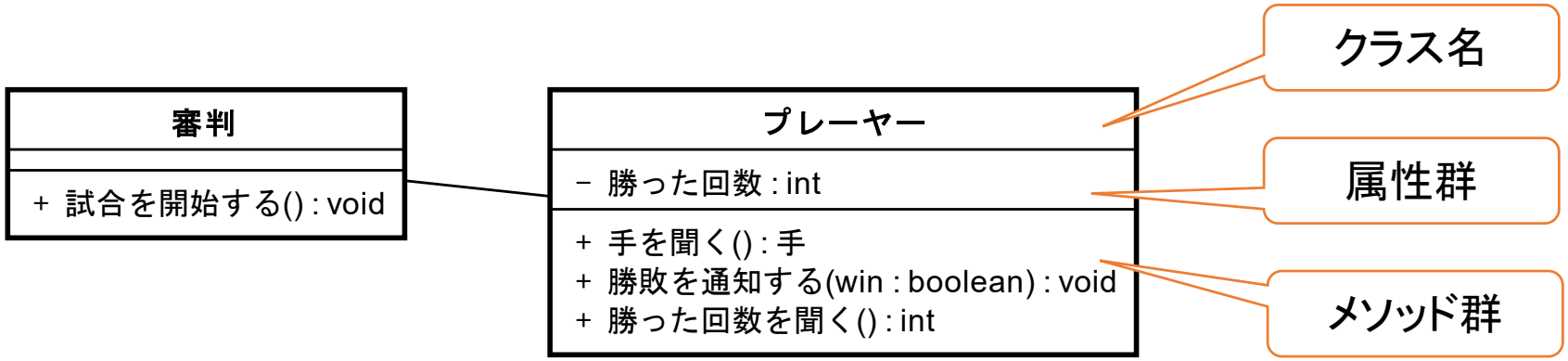
何故Java 図3-5

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．

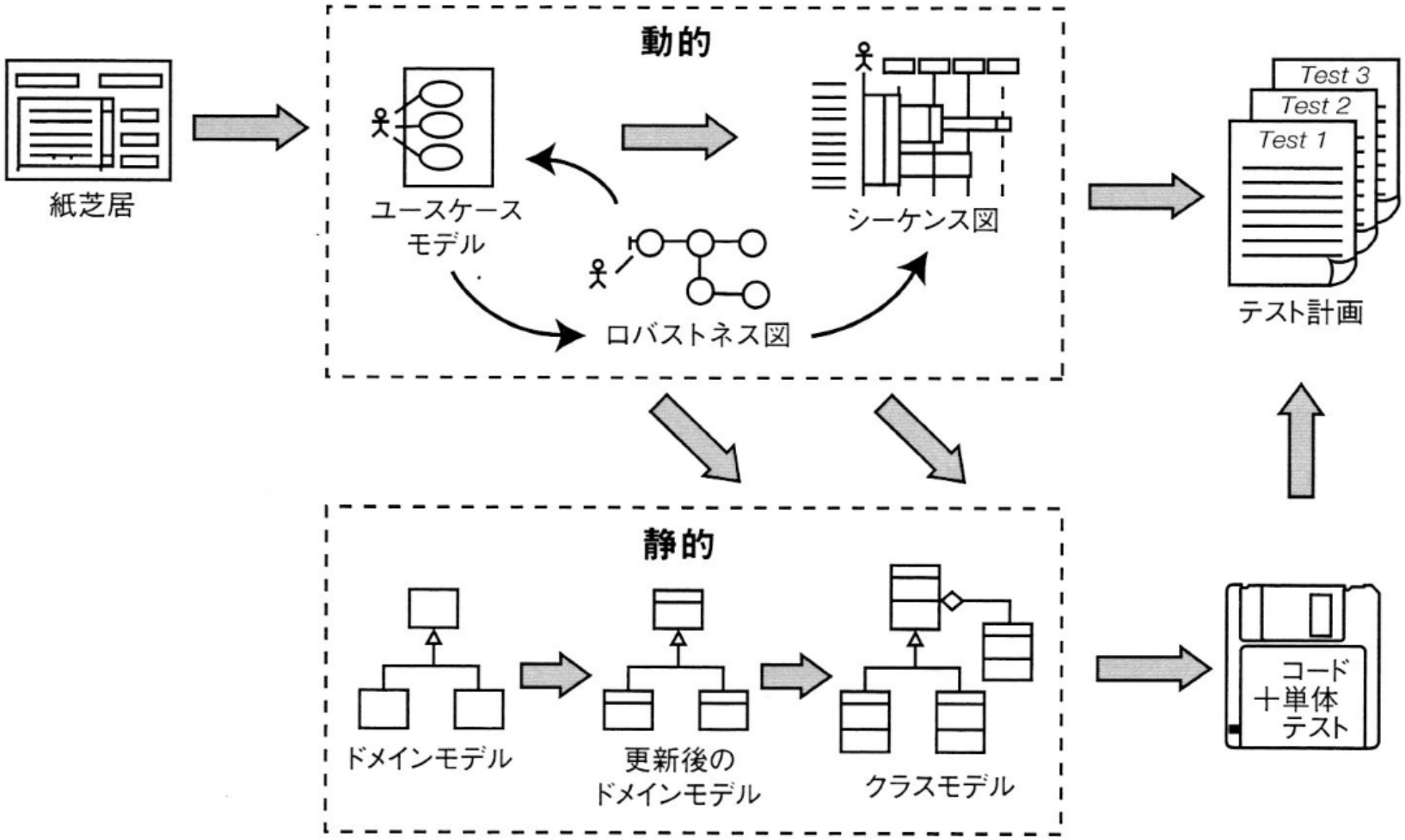
getWinCount 等

- 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webclassから参照.

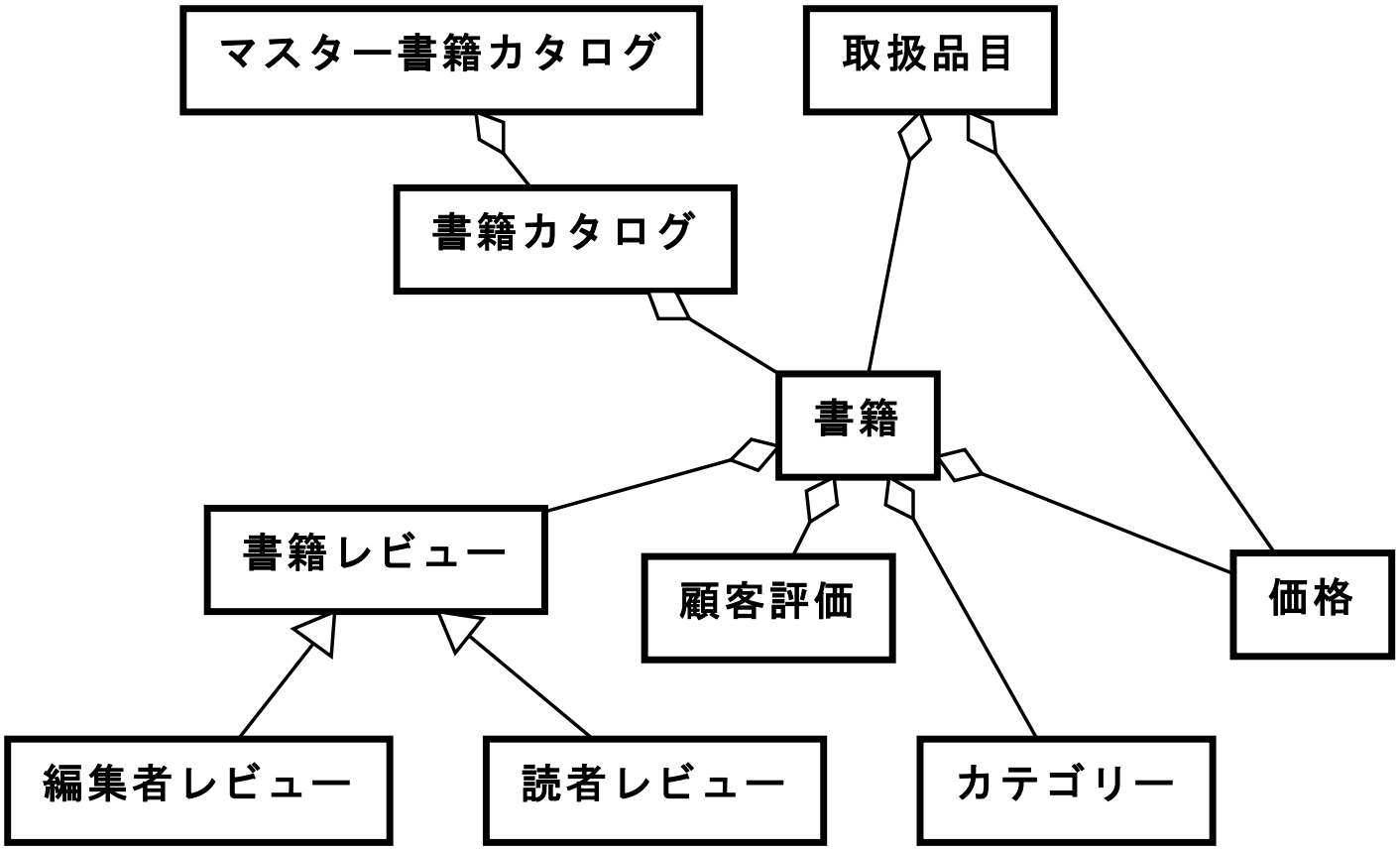
- 要求定義書がちゃんとした文書として整備されていない場合も一般には考えられる.
  - 最近は無いかも.

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」とつぶやいて，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

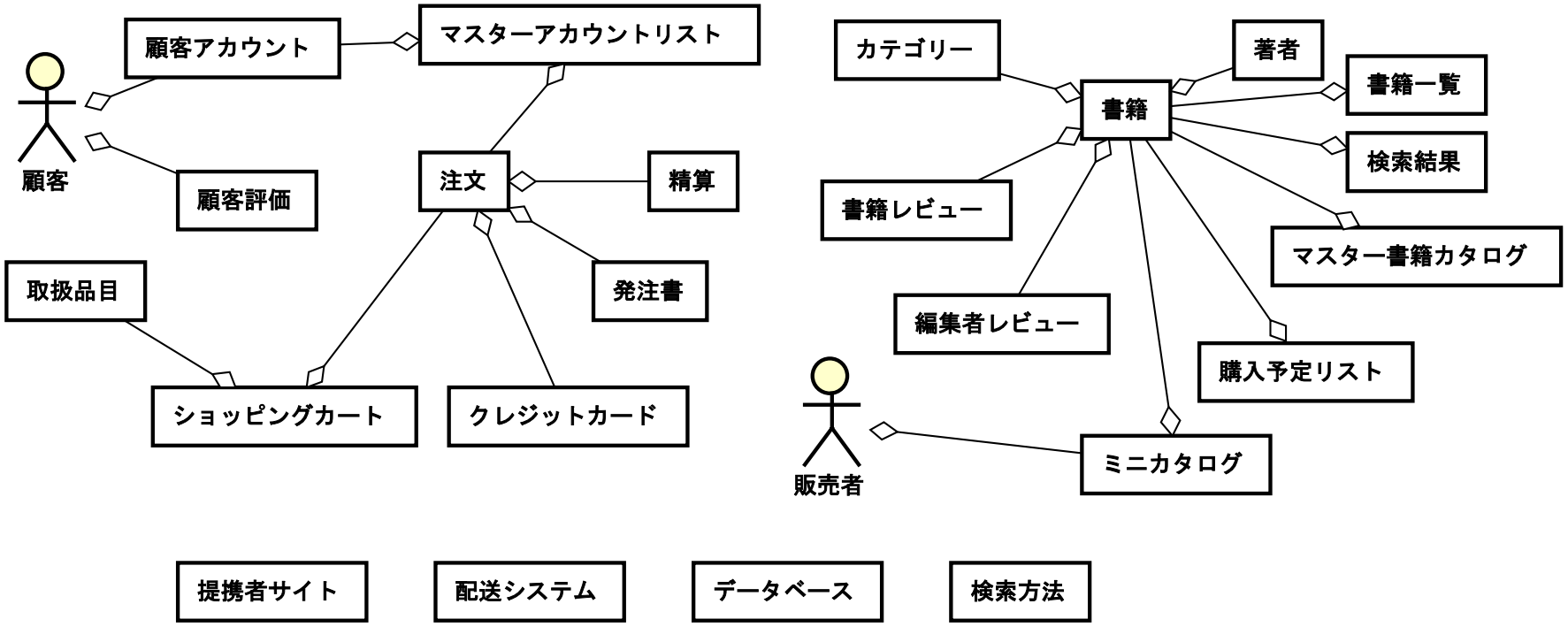
- 「ユーザーアカウント」と「顧客アカウント」は同じなので，例えば後者を残す．
- 「インターネット」は一般的すぎるので除外．
- 「パスワード」，「タイトル」，「キーワード」は概念として小さすぎるので除外．

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

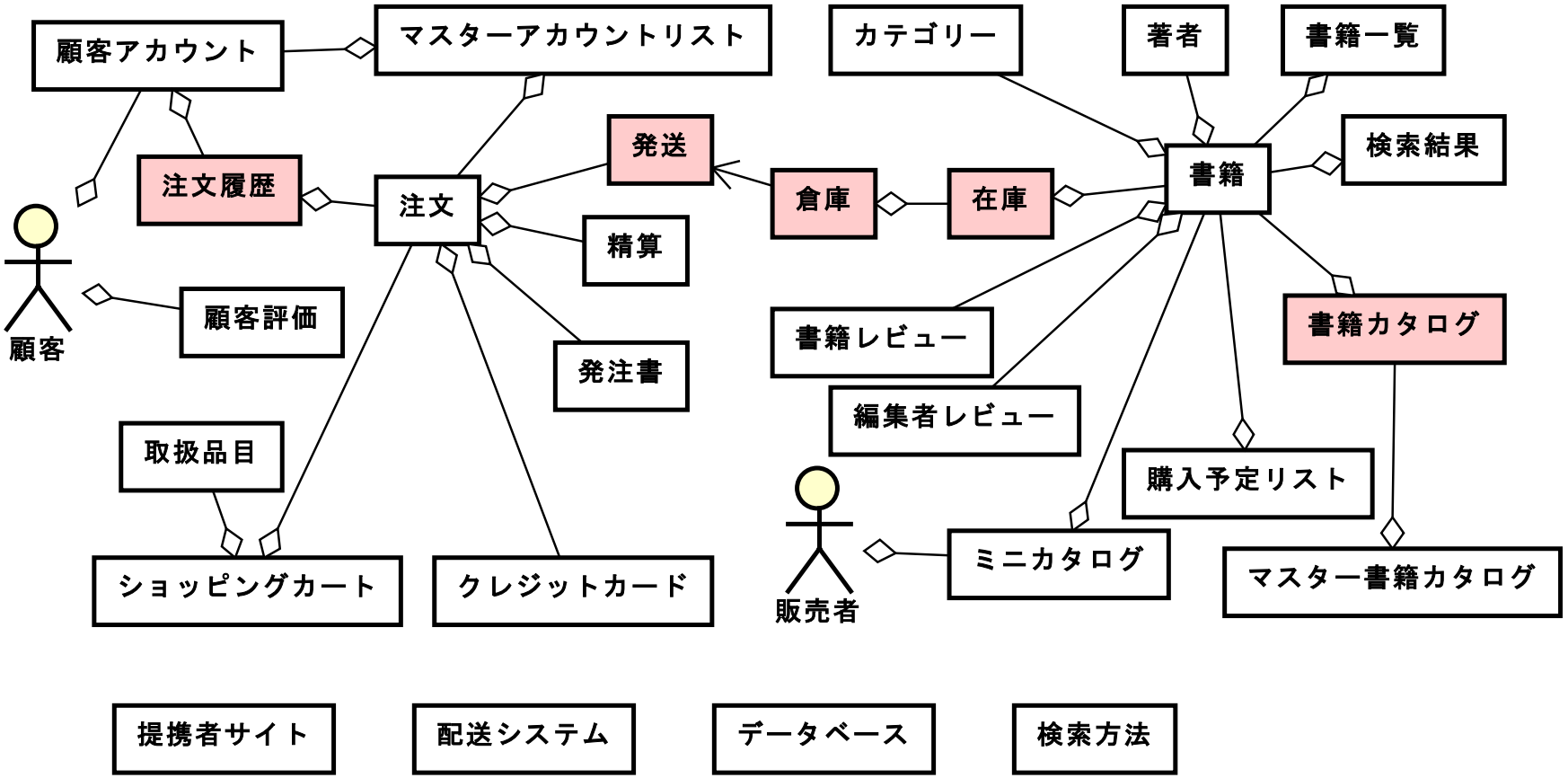


ICONIX 図2.4

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



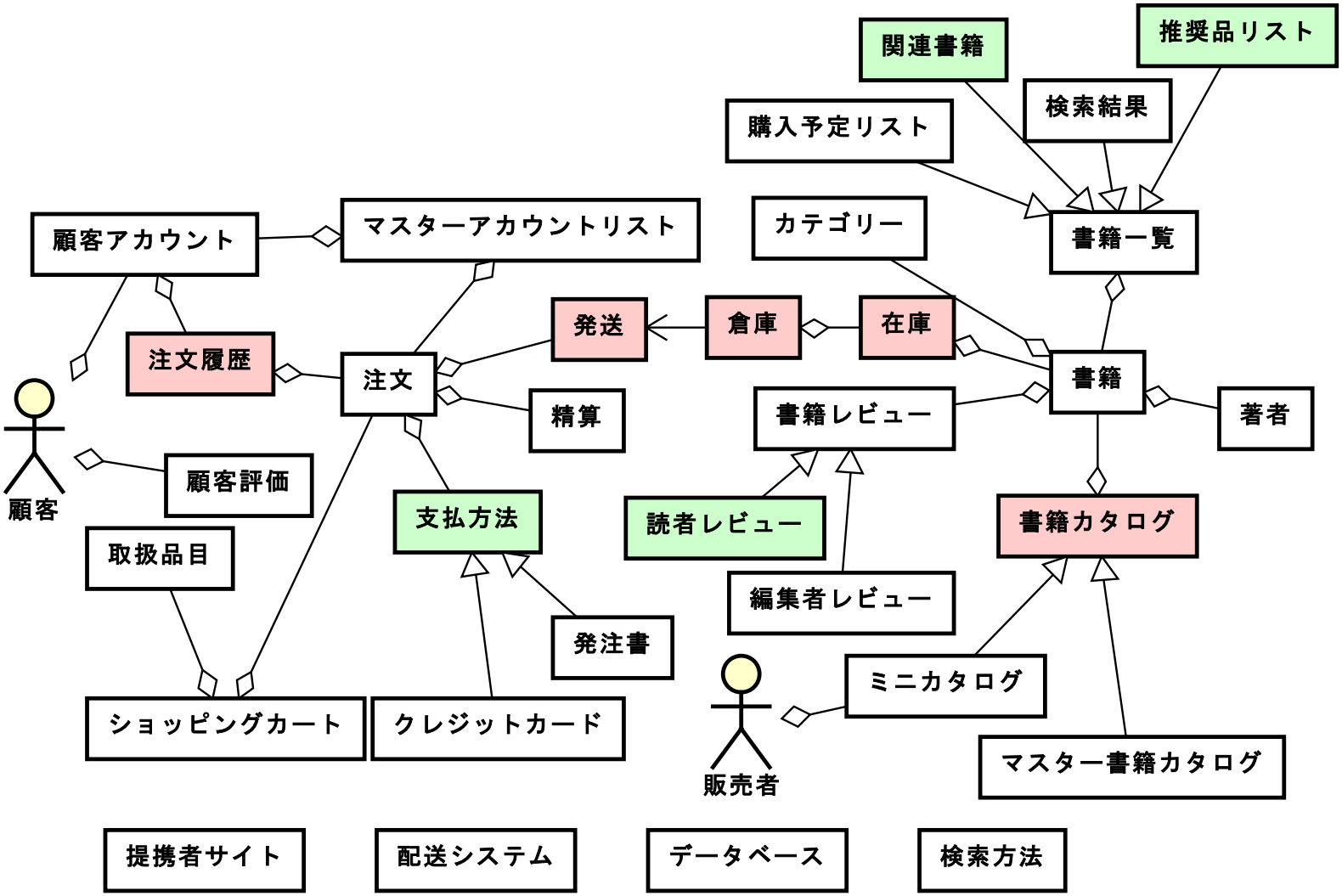
ICONIX 図2.5

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)

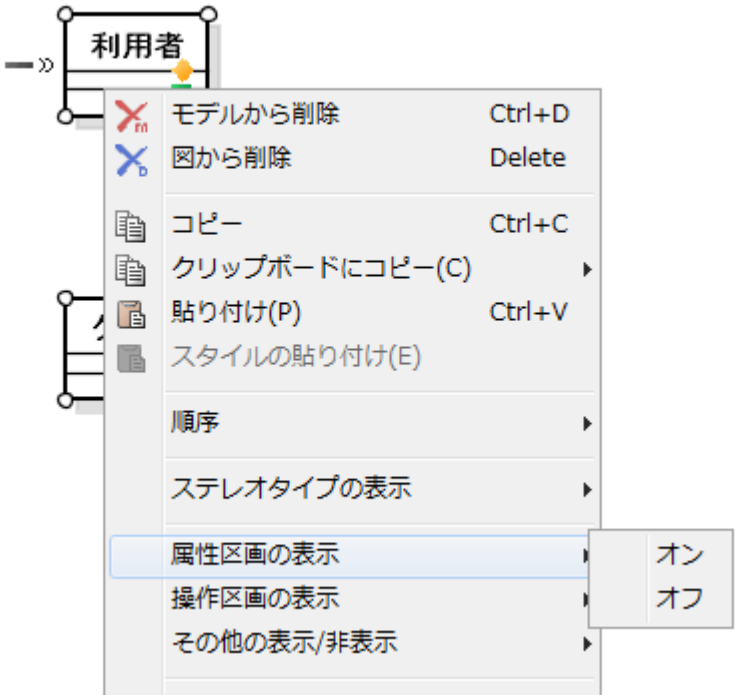
version 3 ドメインモデル



ICONIX 図2.7

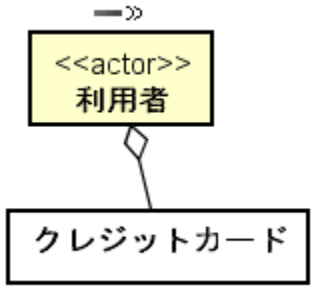
astah TIPS: クラスの簡略表記

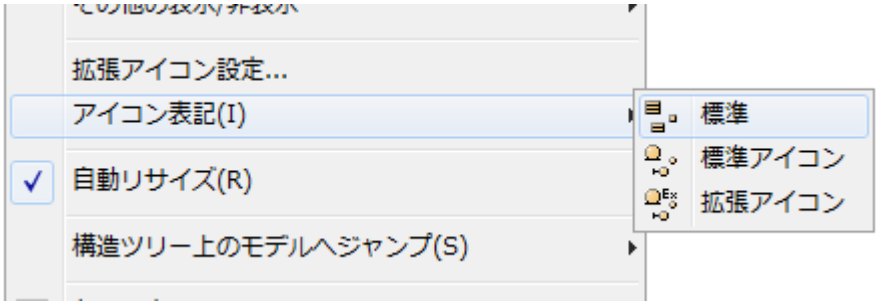
- astahではクラス図を用いてドメインモデルを書く.
- デフォルトのクラス図は三つのパートに分かれている.
- ドメインモデルでは当面，名前だけあればよいので，右のように属性，操作の表示をオフにする.
- とりあえず図を描いて全体選択をしてからだと楽.

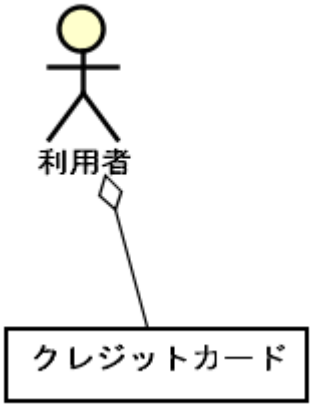


astah TIPS: アクターの人型表示

- 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 card ⇒ 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.

ユースケース

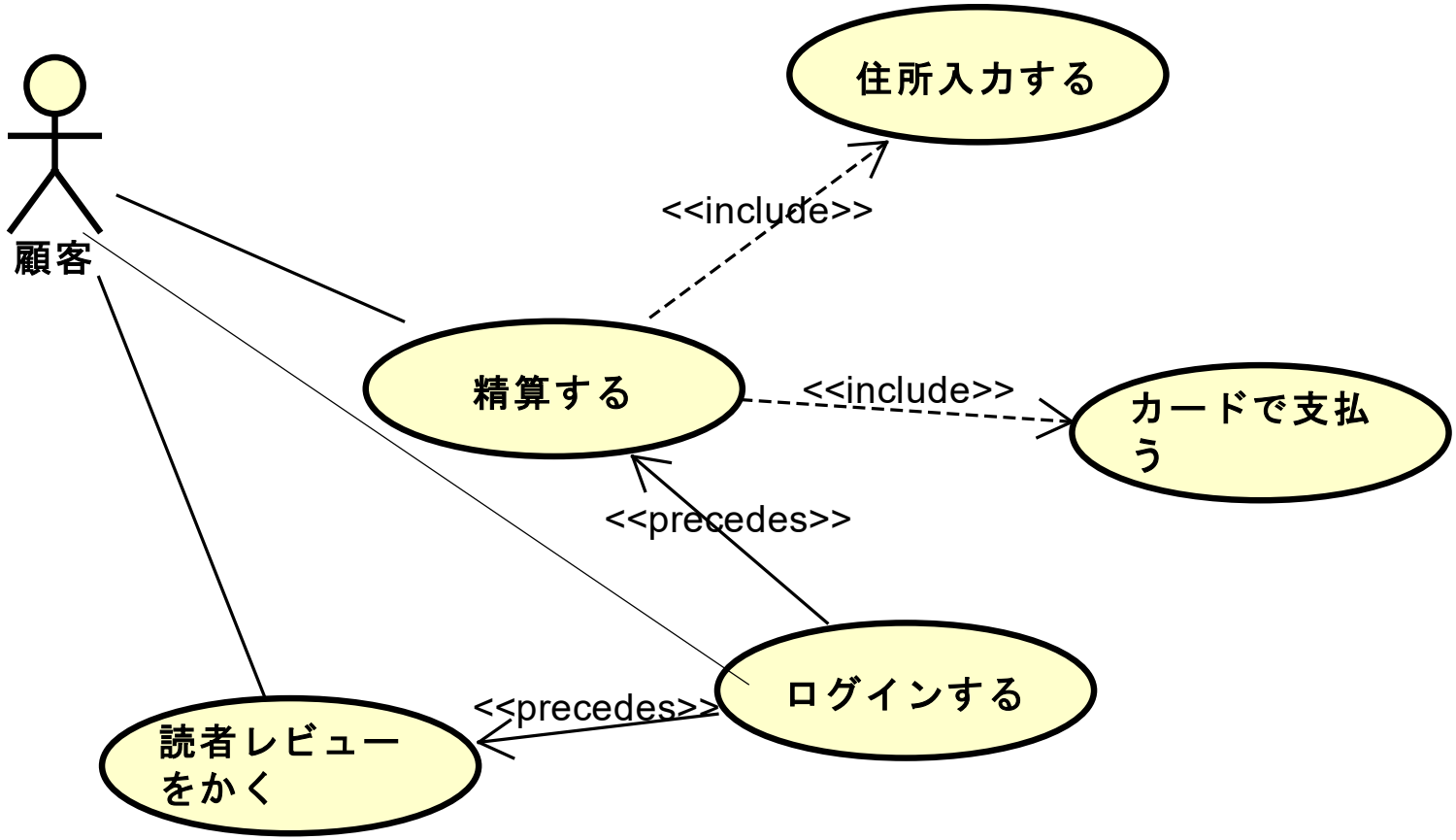
- 端的にいって，システムの機能を示す．
- その機能を遂行するために，システムとシステム外部の人や別システム(アクターと呼ぶ)が相互作用する様として仕様化する．

- ユースケースモデル
  - システム全体がどんな機能群を持ち，それぞれの機能遂行にどんなアクターがかかわるかを記述したもの．
- ユースケース記述
  - 個々のユースケースを遂行するにあたり，システムやアクターが順番に何をするかを記述したもの．

ユースケースモデル

- システムの機能を楕円でかき，
  - この機能ひとつひとつを「ユースケース」と呼ぶ
- その機能遂行に関連するアクターと線でつなぐ簡易な図.
- ユースケース間の関係はわりと沢山あるけど，本講義では，以下の二種類の利用のみ推奨する
  - includes ユースケースAがBを呼び出すこと.
    - サブルーチンや関数コールとほぼ同じ.
  - precedes AがBより先に起こること.
- ユースケースモデルではシステム内部のデータに相当するものは書いてはいけない.

ユースケースモデルの例



標準的な他の関連について

- invokes
  - includesにほぼ同じ.
  - 本来，こっちを使いたいが，ツールの都合，includesを使う.
- extends
  - 本授業では利用しないことを推奨
  - Javaの extends と意味が異なる.
  - Javaでの抽象クラスをexntendsするのにノリが似ていて，拡張される側が事前に拡張されることを意識した作りになっていなければならない.
- generalize
  - 本授業では利用しないことを推奨.
  - こっちは，所謂サブクラス，スーパークラスの関係.
  - どちらかといえば，overrideの側面が強調されている.

その他，例題

- サンプルはwebclassからダウンロードすること

- webclass 等のLMS LMS.asta
  - 意外に複数のアクターがかかわる機能が無い.
- Uber easts ube.asta
  - 実は使ったことない，シンプルになった.

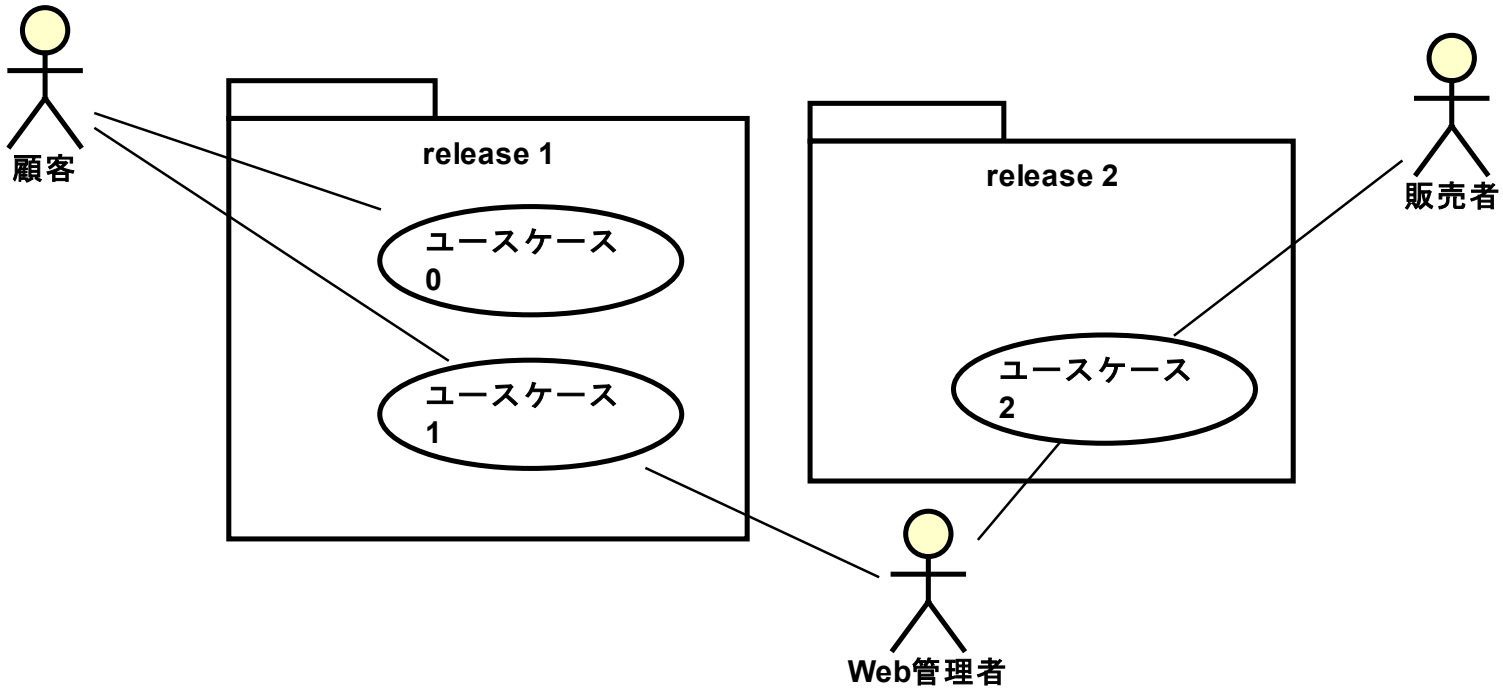
- iTrust 電子カルテシステム
  - とある大学で演習用に作られた，巨大.

ユースケースのパッケージ化

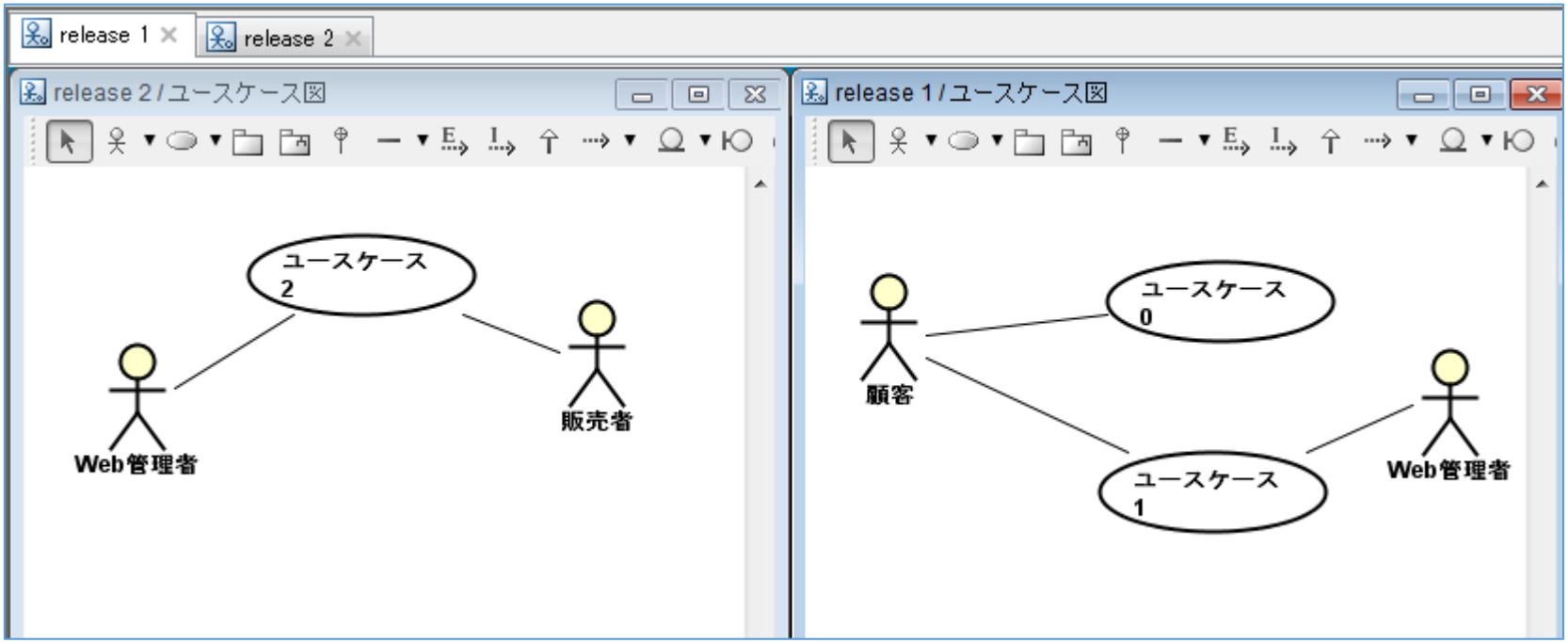
- 大きなシステムの場合，ユースケース(機能)が100を超える場合がある.
- その場合は，ユースケース図を分けてかくか，
- ユースケースをパッケージで分けるかする.
- 分ける指針としては，
  - 機能的に関連した領域
  - リリース

  がある.
- 一気に全てのシステムを作るのではなく，最初は一部機能から作るならリリース毎にパッケージ化するのが良い.

パッケージの例



図を分ける


release 1
release 2
release 2 / ユースケース図
ユースケース
2
Web管理者
販売者
release 1 / ユースケース図
ユースケース
0
顧客
ユースケース
1
Web管理者

ユースケース記述について

- 各ユースケース(楕円で示した機能)が，システム外のアクターと，どんなステップをふんで機能を遂行するかの手順を書く.
- システム内の手順は書かない，書くのはシステム外部とのやり取りのみ.
- 基本，システムもしくはアクターが主語となる文の列となる.
- 文中の目的語等はドメインモデルもしくは境界クラス(後述)から選ぶ. 無ければ，ドメインモデルを更新する.
- 通常の手順に加え，代替の手順，例外の手順も書く.

ユースケース記述のフォーム構成

- 概要 (略可能)
  - 当該機能の概要を一～二行くらいで要約して書く.
- 事前条件 (略可能)
  - この機能を実行する際の前提条件.
- 事後条件 (略可能)
  - この機能が実行された後に成り立つ条件.
  - 要は機能の宣言的な意味.
- 基本系列
  - 機能を遂行する正攻法の手順.
  - 要は機能の手続き的な意味.
- 代替系列 (略可能)
  - 別ルートの手順. もしあれば.
- 例外系列
  - 機能遂行が失敗に終わる場合の手順.
  - システムやアクターが回復不能な状態に陥らないような手順が望ましい.

例

項目	内容
ユースケース	精算する
概要	購入代金の精算を行う.
アクター	顧客
事前条件	顧客はログイン状態にある
事後条件	支払いが完了している
基本系列	顧客はシステムに住所を入力する. システムは入力された住所を表示し確認ボタンとやり直しボタンを提示する. 顧客は表示内容が正しい場合，確認ボタンを押す. システムは支払い方法の選択画面を表示する. 顧客は支払い法を選択する. システムはカード決済が選択された場合，カード番号を要求する. 顧客はカード番号を入力する. システムは決済確認のためのボタンを表示する. 顧客はボタンを押し決済を承認する. システムは承認を受け付けたことを表示する.
代替系列	
例外系列	システムは入力されたカード番号がvalidなものでなければ，その旨を表示し精算を中断する.

記述のポイント

- 文の列として書く.
- 能動態の文で書く，受動態や支持的な文はダメ.
  - 「<アクター>が<何か>をする」の形式がよい.
- 各文の主語はアクターもしくはシステムでなければならない.
- 主語は省略してはならない.

基本，代替，例外と分ける意味

- プログラムのような形式記述なら，これらを分けて書く必要は無いかもしれない．
- しかし，原則，文のリストとしてかくため，制御構造が複雑にならないためにも，分けて書くほうがよいとされている．
- 加えて，本来，意図する動作である「基本」，バックアップとしての「代替」，障害処理的な「例外」は，それぞれ要求分析的に意味合いが違うため，文法的に合成して書けたとしても，合成すべきではない．

画面イメージ

- ユースケースでは，システムとアクターの対話を書く.
- よって，対話の接点となるユーザーインタフェース(UI)の画面イメージをメモ書きしてもよい.

境界クラスについて

- Boundary Class
- システムがアクターとのインタフェースを通して何か行う場合，
  - 「システムはWebページに注文内容を表示する」

と書く場合がある.

- このような場合，インタフェースがより具体的に決まっているならば，その具体的な名称を書いたほうが良い，それが境界クラス，たとえば，
  - 「システムは注文確定ページに注文内容を表示する」

ユースケース vs アルゴリズム

ユースケース	アルゴリズム
ユーザーとシステムの対話	不可分な計算
イベントと応答のシーケンス	ステップの列
基本/代替コース	ユースケースのーステップ
複数のオブジェクトがかかわる	クラス内部での処理
ユーザーとシステム	システムのみ

本日は以上