デザインパターンを用いた オブジェクト分析/設計技法［基礎編］

佐原伸

E-Mail≡ sahara@sra.co.jp

URL≡ http://www.sra.co.jp/people/sahara

産業システム第2部第3グループ
オブジェクト指向グループ

E-Mail≡st-info@sran265.sra.co.jp

URL≡ http://www.sra.co.jp/smalltalk

ＳＲＡ

URL≡ http://www.sra.co.jp

セミナーの目的

- オブジェクト指向の概念・基礎を把握する
- UML(Unified Modeling Language)記法の基礎を理解する
- オブジェクト指向分析/設計の基礎を理解する

前提条件

- オブジェクト指向による分析/設計を目指すソフトウェア技術者
- できれば、プログラミング言語の基本知識

基礎1-3

目次

- I. なぜオブジェクト指向か？
- II. オブジェクト指向の基本概念
- III. オブジェクト指向の歴史
- IV. オブジェクト指向分析/設計
- V. オブジェクト指向分析/設計の手順

基礎1-4

I. なぜオブジェクト指向か？

ソフトウェア危機
オブジェクト指向の効果
オブジェクト指向の発想
経済的理由
ソフトウェア工学的理由
オブジェクト指向の使用例

ソフトウェア危機

- ハードウェア・ソフトウェアの進化により、ユーザーの要求が高度化しシステムが複雑になってきている
  - マルティメディア＋ネットワーク＋リアルタイム
- 従来の手法では、開発要員が多すぎて開発コストと開発期間が増大している
  - １０００人以上・１０００万ステップ以上
- 開発規模の拡大にともなってソフトウェアの品質低下が著しい
  - 中華航空機・もんじゅ・第３次オンライン・Y2K
- 保守のコストが増大している
  - 大手ユーザーでは８０%以上のリソースを保守に投入

ソフトウェア危機の背景

- 変化するコンピュータ環境
  - 「恥ずかしながらメモリーは192MBしか...」
  - CPU速度は１～２年で２倍
  - 関数型言語などの速度は、5年前の100倍
- 変化するソフトウェア環境
  - 「早い・安い・うまい」が求められている
    - ユーザーの多様化・適用業務の多様化・情報の洪水

プログラミングはそもそも難しい

- 理由１
  - 人間に仕事を頼むときと異なる頼み方をする
    - 人間の自然な思考方法に一致させることにより、分かりやすいプログラミングが可能となる
- 理由２
  - 特殊な言語（プログラミング）を使ってコンピュータに指示する
- 理由３
  - そもそも「プログラミング」は難しい
    - 数学基礎論・コンピュータ科学などで「難しさ」は証明されている
      - プログラムが正しいことは「実用的な大きさのプログラムでは証明できない」
      - プログラムでは解けない問題がある

従来アプローチが未解決の問題

- 機能中心の弊害
  - 「問題」は必ずしも「機能中心」ではない
  - データ中心であったり振る舞い中心であったり制約中心であったりする
- データと手続の分離の弊害
  - データに自由にアクセスできた
  - データ構造を変えると変更余波が大きかった
  - データ構造を知らないとアクセスできなかった
- 重複開発の弊害
  - 変更がやり易いことと、変更余波を小さくすることの両立ができなかった

基礎1-9

オブジェクト指向技術の登場

- 構造化技法の一種
  - 保守性と再利用性の向上を目指している
- 人間のコミュニケーション方式を模倣
  - 従来の「手続型」指向は自然でない
    - フォン・ノイマン型計算モデルは、偶然の産物
      - オブジェクト指向
      - 関数指向
      - 制約指向...などいろいろな考え方がある
- オブジェクト指向の提案
  - コンポーネント（＝オブジェクト）
  - 抽象化
  - フォン・ノイマン型思考からの脱皮

基礎1-10

オブジェクト指向の効果

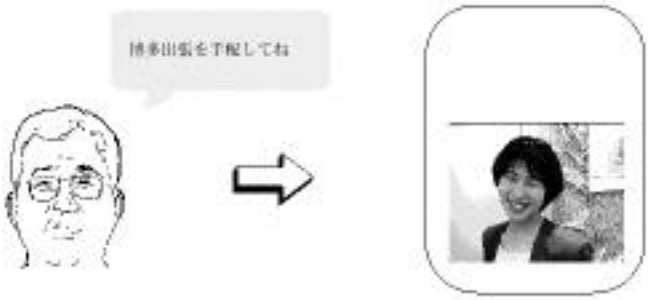
- 品質の向上が期待できる
  - オブジェクト指向技術本来の狙い
- 情報表現力が向上する
  - すべてがオブジェクト
    - 例えば、テキストと図形とプログラムとを区別する必要がない
- 結果として
  - 保守が容易になる
  - 開発費用が削減できる
  - 開発期間の短縮が可能になる

例１：仕事の依頼（旅行の手配）

- 人から人の場合
  - 「５日に博多へ出張するから、手配をしてね」
    - 「何を」が中心で「どうやって」は気にしない
- 人からコンピュータの場合
  - 「旅行代理店のホームページを開いて、５日12:15羽田発福岡着のJASを予約し、５日夜のハイアット・ホテルを予約し、６日12:25福岡発羽田着のJASを予約し、予約ページを印刷せよ。」
    - 「どうやって」が中心で「何を」は抜けていることが多い
    - 手続的に指示しなければならない

例２：仕事の依頼（旅行の手配）

- 目的地までの経路や飛行機やホテルの予約方法は知る必要がない
  - 秘書が全て知っている（はず）

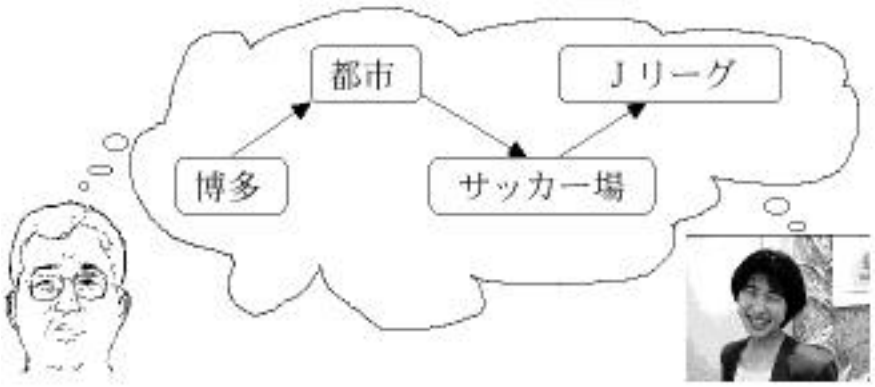


例３：情報の伝達

- 人とコンピュータの違い
  - 人同士の場合は、言葉により簡単に伝えられる
    - 「博多の森競技場は歩いて行けたよ」？
    - 「アビスパ福岡は勝ったんですか？」？
  - 相手がコンピュータの場合は、プログラムで逐一指示しなければならない
    - 「博多の森競技場は歩いて行けたよ」？
    - 「博多とは何か？」「森競技場とは何か？」「歩いて行くにはどう行くのか？」？

情報の伝達
（人同士の場合は何が違うか？）

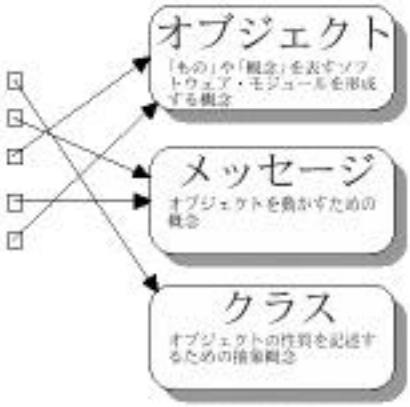
- 人同士は「もの」の抽象イメージを共有している



基礎1-15

オブジェクト指向技術の概念

- オブジェクト指向技術は、単純で強力な３つの概念から構成されてい
- 一般的な言葉で伝達できる
- 秘書に出張手配を依頼する
  - 後は任せる
- 旅行代理店に行き先を言う
  - 細かい指示はしない



基礎1-16

プログラムの主人公が異なる

- 従来
  - フォン・ノイマン型コンピュータのように考える
    - 順序を追って操作を考える
      - 手続的思考
- オブジェクト指向
  - 人間の自然な概念に従って考える
    - 「誰に」「何を」頼むかを考える

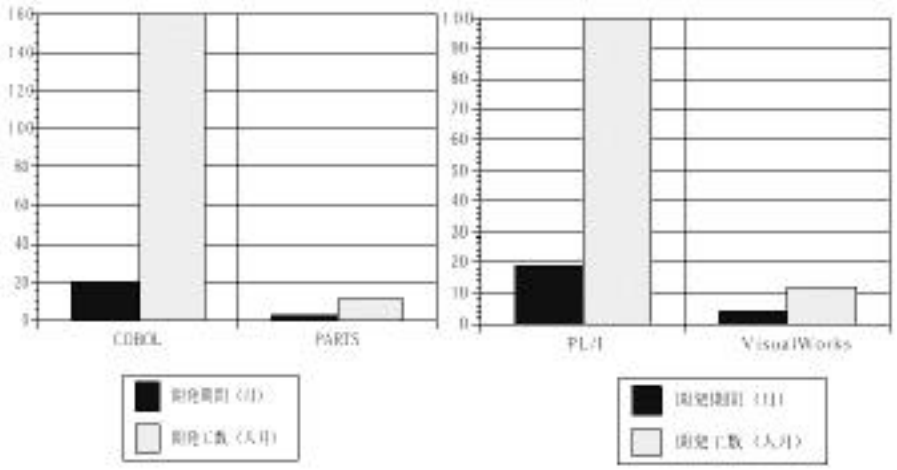
基礎1-17

経済的理由

生産性の比較
実行スピード
使用メモリー

基礎編1-18

生産性の比較（対COBOL,PL/I）



基礎1-19

生産性の比較
（ファンクション・ポイント）

言語	行数/ＦＰ
Assembler	320
C	150
COBOL	106
FORTRAN	106
Pascal	91
PL/I	80
Ada	71
Prolog	64
APL	32
Smalltalk	21
Visual Smalltalk	10
SpreadSheet	6

基礎1-20

実行スピード

- 初期のSmalltalkやLispベースの言語がインタープリタだったため遅かったが...
  - 現在はインクリメンタル・ネイティブ・コンパイラ
  - 動的束縛が遅い原因のひとつ
    - メソッドキャッシュとかハッシュ表の使用により、動的束縛のコストは一定に抑えられるようになった
      - 現在では高々５０%遅いだけ
    - 十分な情報が与えられれば、ほとんどのメソッド呼び出しは動的にならず、静的に行うことができる
- 成熟したクラスライブラリーを持つOO言語では、非OO言語より速いこともある
  - OO言語のオーバーヘッドより、成熟したクラスのデータ構造やアルゴリズムの実現による効率向上の方が大きい

使用メモリー

- OSやNetscapeやC++の方がメモリ喰い
  - 小規模プログラム
    - C <C++<Smalltalk　COBOL　PL/I
  - 中・大規模プログラム
    - C　C++　Smalltalk <COBOL　PL/I

ソフトウェア工学的理由

外的品質要因
内的品質要因
モジュール性の原則
再利用可能なモジュール構造の要件

外的品質要因

- 正確さ
  - 要求された通りに仕事を行う能力
    - 非常に難しい
- 頑丈さ
  - 異常な状態においても機能する能力
    - 仕様によって明示されない状態は必ず存在する
    - 破滅的な状況を回避する
- 拡張性
  - 仕様の変更に容易に適応できる能力
    - 大規模プログラミングにおいて重要
    - 拡張性を向上させるための法則
      - 簡明さ
      - 非集中性

外的品質要因

- 再利用性
  - 新しい応用にどの程度再利用できるかを示すもの
    - 品質の向上に寄与する
- 互換性
  - ソフトウェア相互の組み合わせやすさ
  - 例
    - ファイルフォーマットの標準化
      - UNIX
      - テキストファイルはすべて文字ストリーム
    - データ構造の標準化
      - Lisp
      - 全てのデータとプログラムを２分木で表す
    - ユーザーインタフェースの標準化
      - Smalltalk

内的品質要因

- モジュールの分解しやすさ
  - 例
    - トップダウン設計
  - 悪い例
    - 初期化モジュール
- モジュールの組み合わせやすさ
  - 例
    - プログラム・ライブラリ
    - UNIXのShell規約
      - ファイルの仮想化
      - 入出力の切り替え
      - パイプライン
  - 悪い例
    - プリプロセッサー
- モジュールの分かりやすさ
  - 悪い例
    - 順序に依存するモジュール群

内的品質要因

- モジュールの連続性
  - 変更の影響が小さい場合モジュールは連続性を満たしている
    - 例
      - シンボル定数
    - 悪い例
      - 物理的表記の使用
      - 静的配列
- モジュールの保護性
  - 異常条件の影響が少ないときモジュールは保護性を満たしている
    - 例
      - 入力の妥当性検査
    - 悪い例
      - PL/IやAdaなどの、エラー検出と処理を別々に扱う例外処理

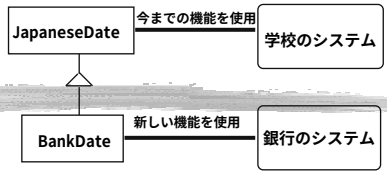
基礎1-27

モジュール性の原則

- 言語としてのモジュール単位
  - 言語の適切な支援なしに高いモジュール性を実現することは不可能
    - 悪い例
      - COBOL
- 少ないインタフェース
  - モジュールはできるだけ少ないモジュールと対話すべき
- 小さいインタフェース
  - ２つのモジュールが交換する情報はできるだけ少なくすべき

基礎1-28

モジュール性の原則



- 明示的なインタフェース
  - モジュール間に対話がある場合、どちらかまたは両方のモジュールにその事実が記載されなければならない
    - 悪い例
      - データの共有
- 情報隠蔽
  - 公にすると宣言されていない限り、モジュールに関する情報はすべて非公開にする
- 開放／閉鎖の両立
  - 以下を同時に満たさなければならない
    - モジュールが拡張可能である（開放）
    - モジュールが他のモジュールから使用できる（閉鎖）

再利用可能なモジュール構造の要件

- 型の変化に対応できる
  - 同じモジュールで、整数の表を検索したり、文字列の表を検索したりできなければならない
- データ構造とアルゴリズムの変化に対応できる
  - （整列済みあるいは整列していない）１次元表、配列、二分木、連結リスト、ハッシュ表などに同じモジュールで対応
- 関連した操作がまとまって定義されている
  - 表検索、表作成、表挿入、表削除ルーチンなどが一緒に定義されること

再利用可能なモジュール構造の要件

- 顧客モジュールが実現方法を知ることなく操作を要求することができる
  - 今までの技術ではうまく解決できない
    - 顧客モジュールがデータの型を知っていると、変更余波が広範囲に及ぶ
    - 供給者モジュールがデータの型を知っていると、複雑怪奇なモジュールになる
  - 後述する動的束縛で解決できる
- 実現方法の間の共通部分がうまくまとめられている
  - サブグループ間の共通性をうまく記述できるか
    - 表検索の場合では、順次形式の表がひとつのサブグループになる
      - 順次配列、連結リスト、順次ファイルの共通点をうまくまとめられるか？

基礎1-31

オブジェクト指向に適した分野

- マルチメディア
  - Adobe Photoshop、CADソフトウェア、Apps MicroTV
  - 地図情報システム
  - 米軍空母搭載基地情報システム、フロリダ電力、ガス会社@加州
- ビジネス
  - ドイツ連邦銀行、野村証券@ロンドン
  - カールスバーグビール、IIJ
- リアルタイム制御システム
  - 航空管制システム、中国の鉄道システム
- CASE＆シミュレーション
  - 動燃高速増殖炉、Rational ROSE、Mei

基礎1-32

大手印刷会社

- 企画部門工程管理システム
  - Smalltalk +GemStone +WWW
- 問題点
  - 実行効率不満
    - WWWサーバーとクライアント側の通信速度
- 効果
  - プログラマー２人で構築できた
  - プログラムの変更が容易

基礎1-33

某病院

- 全システムSmalltalk
  - Smalltalk＋OODB
- 問題点
  - リリース遅れ・工数増大
    - プロジェクト管理が不十分
      - サブシステム分割の失敗
      - アーキテクチャ選択の誤り
    - 要件定義が不十分
      - オブジェクト指向技術修得が不十分
- 効果
  - プロトタイプはプログラマー５人ほどで稼働
  - プロトタイプは１ヶ月半で稼働

基礎1-34

カールスバーグ・ビール @デンマーク

- ２４時間稼働の在庫管理システム
- 効果
  - 3.3人が３ヶ月で構築
  - ３人はオブジェクト指向技術をはじめて体験
  - 在庫の行方不明半減、操作簡単
  - ３６５日・２４時間稼働

基礎1-35

その他の成功例

- すべてSmalltalk
  - パーフェクＴＶ
  - 東京電力
    - 事務処理
  - 高速増殖炉シミュレーション
  - ドイツ連邦銀行
    - 銀行・証券・信託業務の全システム
  - アトランタ・オリンピックのシステム

基礎1-36

オブジェクト指向プログラミングでは苦しい分野

- ハードなリアルタイム制御システム
  - もんじゅ、エアバス、スペースシャトル、衛星制御

基礎1-37

I.のまとめ

- ソフトウェア危機は破局的局面に達している
  - 従来型のアプローチでは、まだ未解決の問題があり、ソフトウェア危機はますます悪化する
- オブジェクト指向は
  - 人間にとって自然である
  - 「経済的」である
  - 「ソフトウェア工学の要求」を満たしている
  - 実際に使われているが、万能ではない

基礎1-38

II. オブジェクト指向の基本概念

- オブジェクト指向アプローチの基本概念
  - オブジェクトの概念
  - メッセージの概念
  - クラスの概念
  - 継承とクラス階層
- オブジェクト指向の概念
  - カプセル化
  - 複合オブジェクト
  - クラスとインスタンス
  - 多相と動的束縛
  - 抽象クラスと多重継承
  - 自動メモリー管理

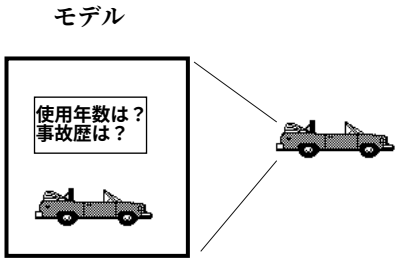
オブジェクトの概念

実世界におけるオブジェクト

- 辞書の定義によるオブジェクト
  - (知覚できる)物, 物体
  - 動作･感情などの対象
  - 目的, 目標
  - 対象, 客観, 客体

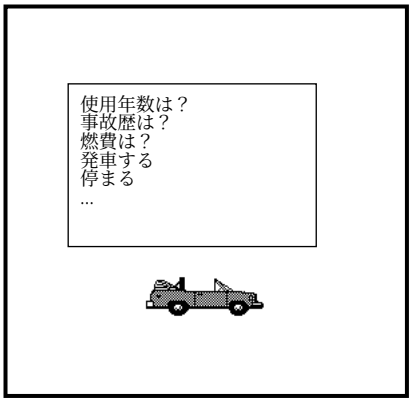
オブジェクト指向における オブジェクト

- 実世界のオブジェクトをコンピュータの世界にモデル化して表現する
  - 辞書の定義によるオブジェクト
    - (知覚できる)物, 物体
    - 動作･感情などの対象
    - 目的, 目標
    - 対象, 客観, 客体



実世界に存在するもののモデル化

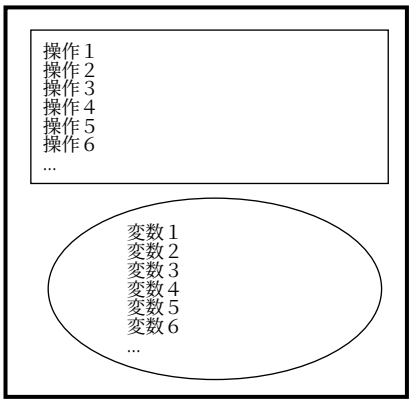
- 車をモデル化する例
  - 車の持つ機能に着目する
    - 最高速度・旋回半径・燃費・使用年数・事故歴...など
    - 発車する・停まる・右へ曲がる・左へ曲がる



車オブジェクト

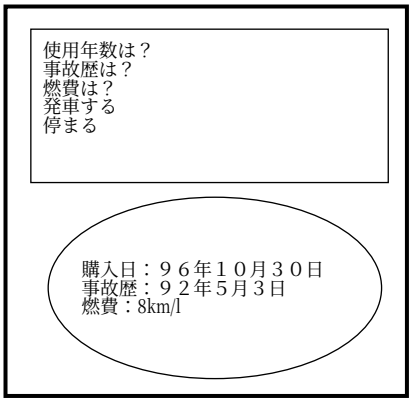
オブジェクトの構成要素

- オブジェクトとは
  - データ（変数）と操作（機能）の両方を備えた情報カプセルである
- 変数とは
  - データをオブジェクト内に保持するための記憶場所である
- 操作とは
  - オブジェクトに定義されている機能（手続）である



オブジェクトの特徴

- 活動結果は変数に保持される
- 最新の状態を維持している
- オブジェクトに関する全情報は変数として表現される
- オブジェクトに関する全ての機能は操作として表現される
- オブジェクトは他と識別されなければならない



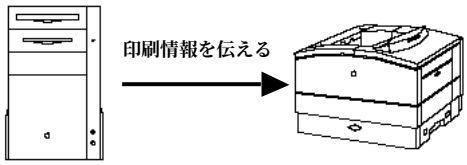
基礎2-7

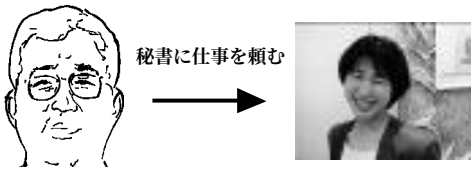
メッセージの概念

基礎編2-8

実世界でのコミュニケーション

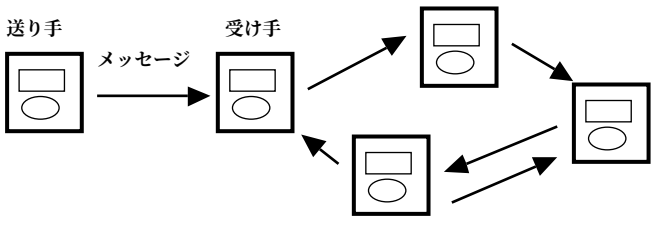
- 実世界のオブジェクトは様々な方法で相互に働きかけている





オブジェクト指向での コミュニケーション

- メッセージとは
  - オブジェクトの持つ手続きを起動するための記号列をいう(岩波情報科学事典より)
  - あるオブジェクトから他のオブジェクトへ操作の実行を依頼する唯一の手段である
  - メッセージによりオブジェクトが活性化される



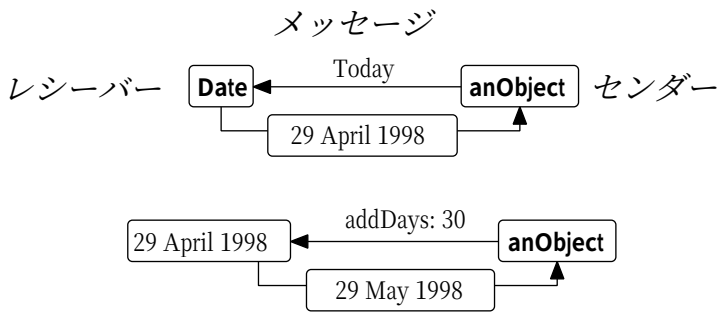
メッセージの構造

- メッセージの構成
  - 様式は言語に依存している
  - 受け手のオブジェクトの名前
  - 操作
  - 実行する際に必要なパラメータ（としてのオブジェクト）
- 例
  - Smalltalkの場合
    - myCar maxSpeed: 220
    - myCar currentSpeed
    - Date leapYear: 1996
    - Date today addDays: 30

Javaの場合
myCar.maxSpeed(220)
myCar.currentSpeed
Date.leapYear(1996)
(Date.today()).addDays(30)

メッセージ送信のイメージ

- 例（Smalltalk）
  - Date today addDays: 30



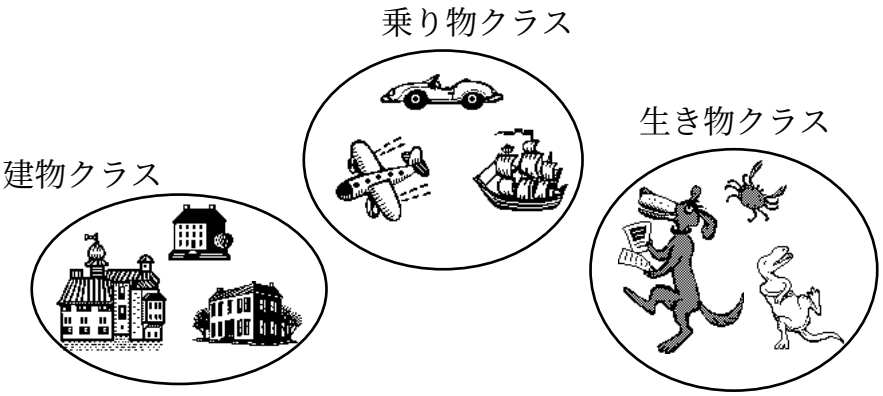
メッセージ

- オブジェクトにメッセージを送る＝
  - データに手続き(メソッド）の名前を渡す
  - 手続きにデータを渡すのではない！
- メッセージ送信の送り先(レシーバー)
  - 「知っている」他のオブジェクト
  - 自分自身
    - self, super(Smalltalk)
    - this, super(Java)
- レシーバーオブジェクトの内容は、メッセージ送信の結果としてしか覗けない

クラスの概念

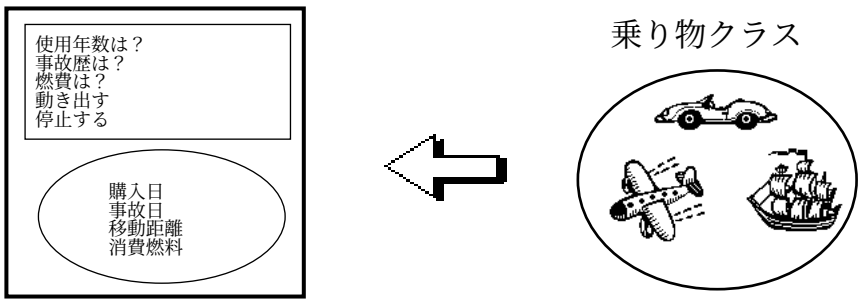
実世界でのクラス

- クラスとは
  - オブジェクトの共通の特性に着目してグループ化したもの



オブジェクト指向でのクラス

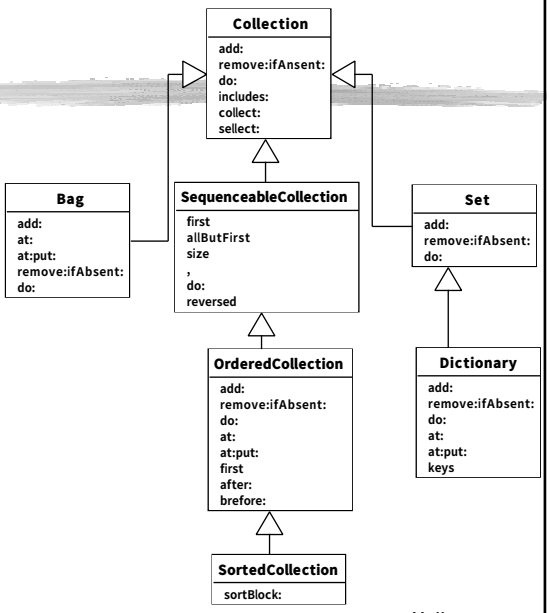
- クラスの役割
  - 共通の概念を抽出するための枠組み
  - 共通する操作／変数を定義するための「場所」



継承とクラス階層

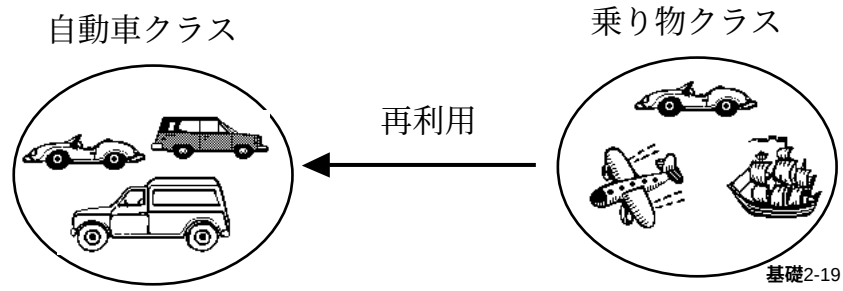
クラス階層

- クラス間の関連を表現した木構造を言う
- クラス階層は継承の仕組みを実現する上で重要である
- スーパークラス
  - あるクラスのより一般的な性質を表す上位クラス
- サブクラス
  - あるクラスのより特殊な性質を表現する下位クラス



継承

- クラス間に階層関係を持たせ、上位のクラスの性質･機能を下位のクラスが受け継ぐことをいう。
- 再利用を実現する仕組み
  - 既存クラスの操作や変数を再利用することができる
- スーパークラスの概念をサブクラスが継承すると共に、新たな性質を追加できる仕組み
  - 差分プログラミングを実現



継承

抽象化 ← → 特殊化

多様性 ↓

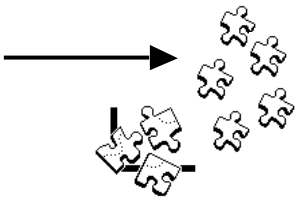
```
Object ()
. . Magnitude ()
. . . . ArithmeticValue ()
. . . . . Number ()
. . . . . Fraction ('numerator' 'denominator')
. . . . . Integer ()
. . . . . LargeNegativeInteger ()
. . . . . LargePositiveInteger ()
. . . . . SmallInteger ()
. . . . . LimitedPrecisionReal ()
. . . . . Double ()
. . . . . Float ()
. . . . . Point ('x' 'y')
. . . . Character ()
. . . . Date ('day' 'year')
. . . . LookupKey ('key')
. . . . . Association ('value')
. . . . Time ('hours' 'minutes' 'seconds')
```

継承の効果

- 類似したモジュール間の本質的な関係を明示できる
- 継承元のクラスに影響を与えることなく、プログラムの拡張を容易にする

部品を体系化できる
　部品を検索のしやすい
　部品を覚えやすい
差分プログラミング
　部品を拡張のしやすい
既存コードと拡張コードの分離
　部品を変更のしやすい

コンポーネントウェア



継承とサブルーチン

- 継承を使った差分プログラミングと、サブルーチンを使った共通化の効果は同じくらいではありませんか？
  - 「再利用可能なモジュール構造の要件」が異なってくる
    - 型の変化に対応できる
    - データ構造とアルゴリズムの変化に対応できる
    - 関連した操作がまとまって定義されている
    - 顧客モジュールが実現方法を知ることなく操作を要求することができる
    - 実現方法の間の共通部分がうまくまとめられている

基本概念のまとめ

- オブジェクト
  - 操作と変数の両方を備えた情報カプセル
- メッセージ
  - 他のオブジェクトに対する操作の実行依頼
- クラス
  - 類似したオブジェクトの共通した性質を記述したオブジェクト
- クラス階層
  - クラス間の関連を表した木構造
- 継承
  - スーパークラスの性質をサブクラスが引き継ぐこと
- スーパークラス
  - あるクラスに対して上位のクラス
- サブ・クラス
  - あるクラスに対して下位のクラス

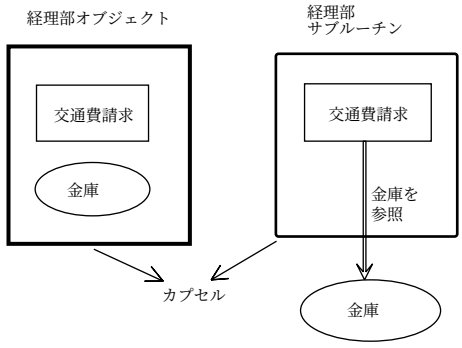
オブジェクト指向の概念

カプセル化
複合オブジェクト
クラスとインスタンス
多相と動的束縛
抽象クラスと多重継承
自動メモリー管理

カプセル化

カプセル化の考え方

- 従来手法
  - 利用者が各自金庫から持ち出すことに相当するプログラミング
    - 仕組み変えると大混乱
- オブジェクト指向
  - 経理部に全てをまかす
    - 仕組み変えても影響が少ない



カプセル化とは

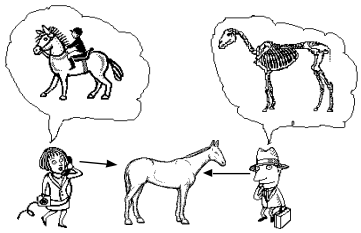
- 操作（手続）と変数（データ）をまとめてモジュール化し、外部との間に厳密なインタフェースを設ける
  - モジュールの独立性が高まり、オブジェクト間の依存性が減る
- 必要なこと以外は外部に見せない（情報隠蔽）

カプセル化の効果

- カプセル内に隠蔽したデータに外部からのアクセスを許さないので、プログラムの信頼性・保守性を向上させる
  - 他のオブジェクトからデータを保護すると同時に、データ構造の変更から他のオブジェクトを保護する
- 利用者はオブジェクトの内部構造を知る必要がない
  - 利用しやすい
- 提供者が内部データ構造を変更しても、利用者に影響を波及させない
  - 変更しやすい

カプセル＝抽象データ型

- 抽象データ型
  - 型定義において、データ型の内部表現としてのデータ構造と、それに対する操作とを一体にして定義したデータ型を言う。操作は、そこで定義したデータ構造を処理する関数または手続きとして実現する。抽象データ型を使用する側では、型名の参照と操作の呼出ししか許されない。(岩波情報科学事典より)
- 抽象化の特性
  - 視点によって異なる抽象化があり得る
  - 一種の偏見でもある



複合オブジェクト

実世界での複合オブジェクト

- 実世界に存在するオブジェクトの多くは、いろいろなオブジェクトの集合体である

引き出しは、ケースと紙の集まり

ケース、1 枚目の紙、2 枚目の紙、...

車は部品の集まり

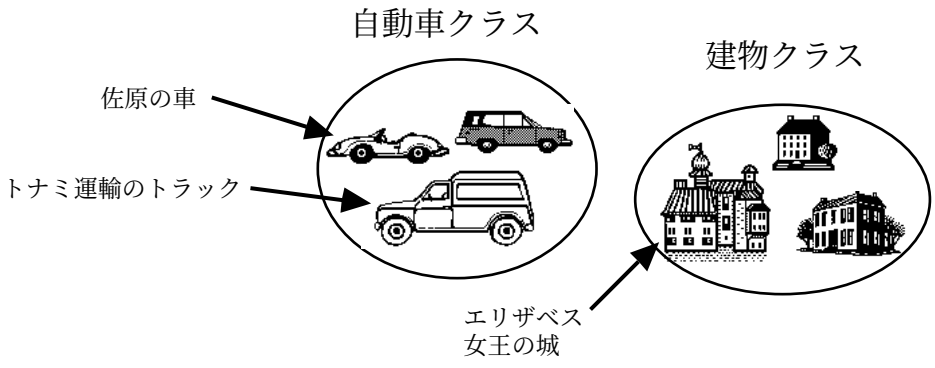
エンジン、ボディ、タイヤ、ブレーキ、...

オブジェクト指向での複合オブジェクト

- 複合オブジェクト
  - 一つ以上の他のオブジェクトを含んだオブジェクトのこと
- 複合オブジェクトの特徴
  - 複合オブジェクトは、他の複合オブジェクトを含んでも良い
    - 自分自身さえも含むことができる
      - 再帰構造
        - 例：ディレクトリー（フォルダー）、木構造
- 複雑な構造を表現できる

クラスとインスタンス

実世界のクラスとインスタンス



クラスとインスタンス
(岩波情報科学事典より)

- クラス
  - 同一の機能や性質を持つが個体として異なるオブジェクトの集まり、あるいはそのようなオブジェクトを生成する言語機構を言う。
- インスタンス
  - あるクラスに属する、あるいはそのクラスによって生成されるオブジェクトを、そのクラスのインスタンスと呼ぶ。
- メタクラス
  - クラスをオブジェクトと見なす場合（Smalltalkなど）、そのクラスを定義･生成するオブジェクトのことをメタクラスと呼ぶことがある。

クラスとインスタンスの具体的役割

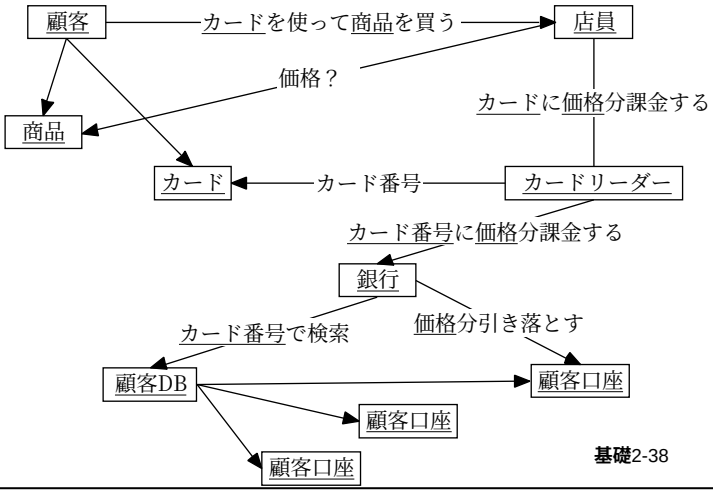
- クラスの役割
  - 共通の概念を抽出するための枠組み
  - 共通する操作／変数を定義するための「場所」
  - インスタンス生成
    - クラスがオブジェクトを生成すること
  - インスタンス管理
    - 自分に所属するインスタンス群を管理する
- インスタンスの役割
  - システム内で使用される個々のオブジェクトで、各種の値や状態を保持している

クラスとインスタンスの関係

- なぜオブジェクトをクラスとインスタンスに分けるのですか？
  - 単なる都合
    - クラスにインスタンス共通の性質を記述するため
      - クラスは一種の型紙
    - クラスのないのオブジェクト指向言語もある
- メタクラスとは何ですか？
  - クラスの性質を記述するためのクラス
    - メタクラスから見ればクラスはインスタンス

クラスとインスタンス

- インスタンスオブジェクトの協調



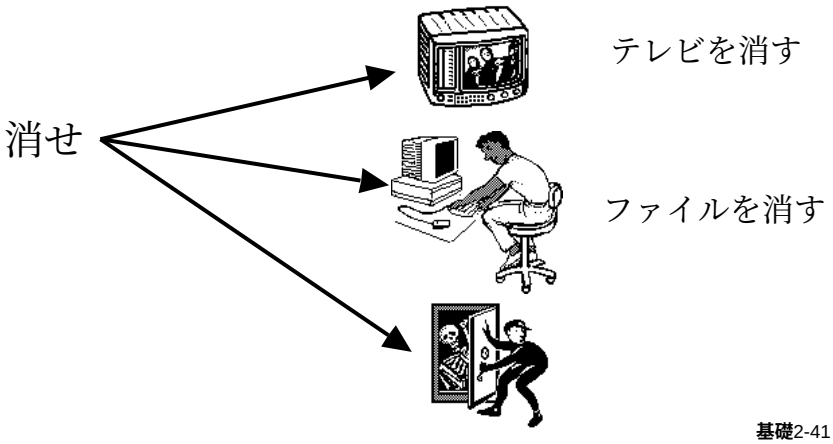
クラスとインスタンス

- 演習問題
  - 交通費精算を行うインスタンスオブジェクトの協調関係を記述せよ

多相と動的束縛

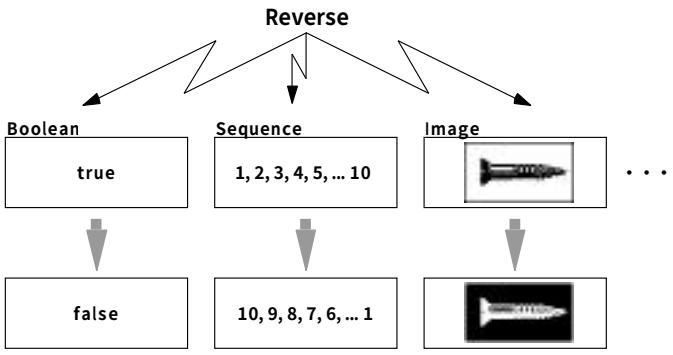
実世界での多相

- 同じメッセージを送っても、相手により動作が異なる



オブジェクト指向における多相

同じメッセージでも受手のオブジェクトによって処方が異なる
様々なオブジェクトを統一的に扱うことを可能にする



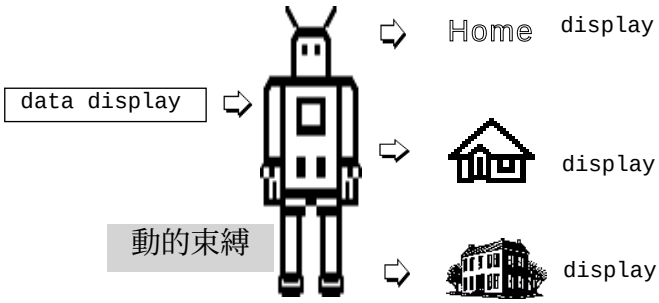
オブジェクト指向における多相

- 同じ名前の操作を、複数のクラスに対して定義できること

```
if data = Text then
    displayText(data)
else if data = Picture then
    displayPicture(data)
end if
```

⇨

```
if data = Text then
    displayText(data)
else if data = Picture then
    displayPicture(data)
else if data = Image then
    displayImage(data)
end if
```



多相の利点

- 同じ意味を持つ操作の名前を共通化することにより、概念を簡素化できる
- プログラムの簡素化と保守性向上を実現できる

従来の方法

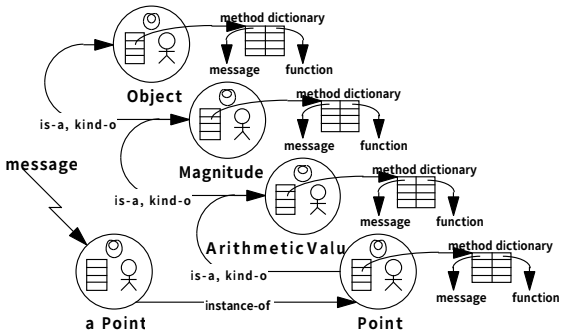
```
if data = Text then
    displayText(data)
else if data = Picture then
    displayPicture(data)
else if data = Image then
    displayImage(data)
end if
```

オブジェクト指向

```
data display
```

動的束縛

- メソッドサーチ



多くのメッセージに同じ名前を付けて混乱しませんか？

- 異なる機能に同じ名前を付けると混乱する
- 同じ機能には同じ名前を付けた方が覚えやすい
  - new, +, printOn:, printString, at:, at:put:
- 多相を使う方が概念が単純化する
  - 分析／設計／実現いずれ工程でも

動的束縛と静的束縛の違いは何ですか？

- コンパイル時にメソッドが確定するのが「静的」
  - C++, Java, Eiffel
- 実行時にメソッドが確定するのが「動的」
  - Smalltalk, CLOS

抽象クラスと多重継承

抽象クラス

- サブクラスで定義される抽象操作がある
  - それ以外の操作は実装されている
- 例：抽象クラスCollectionのサブクラスの場合
  - Collectionのサブクラスで抽象メッセージ(add:, do:, remove:ifAbsent:)を実装すると...
  - addAll:, asArray, asSortedCollection, asString, collect:, detect:, includes:, inject:into:, reject:, removeAll, sizeなどが使えるようになる
- 資産としての再利用性が高い

多重継承

- 同時に複数のクラスを継承できること
- 多重継承の特徴
  - 再利用性の強化
  - 拡張性の増大
- 多重継承の考慮点
  - 重複に対する考慮が必要
    - インスタンスの重複
    - 名前の重複
  - すべてのオブジェクト指向言語でサポートされているわけではない

自動メモリー管理

自動メモリー管理

- プログラマーがプログラミング時にメモリーの取得・解放をするのは、非常に難しい
  - 結果として、実行時エラーの嵐
    - 例
      - Windows, Windows NT
      - Mac OS
      - C, C++で構築されたほとんどすべてのシステム
- 自動メモリ管理（ＧＣ）の仕掛けが必要になる
  - 世代別ごみ集め（generation scavenging）法は、普通のプログラマーがメモリー管理するより速い

オブジェクト指向概念のまとめ

- カプセル化
  - 操作と変数を一緒にパッケージ化すること
- 情報隠蔽
  - 他のオブジェクトから内部情報を隠すこと
- 複合オブジェクト
  - 一つ以上のオブジェクトを含んだオブジェクト
- 多相
  - 同じ名前のメッセージに対して、オブジェクト毎に反応が異なること
- 動的束縛
  - 実行時にメソッドが確定すること
- 自動メモリ管理
  - フォン・ノイマン型コンピュータのアーキテクチャに縛られないこと

III. オブジェクト指向の歴史

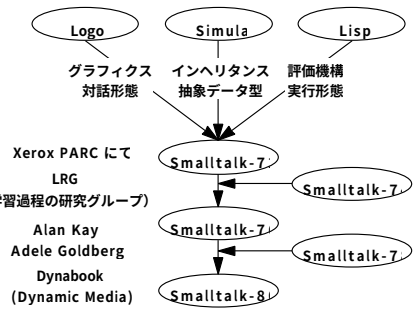
言語
データベース
方法論

言語

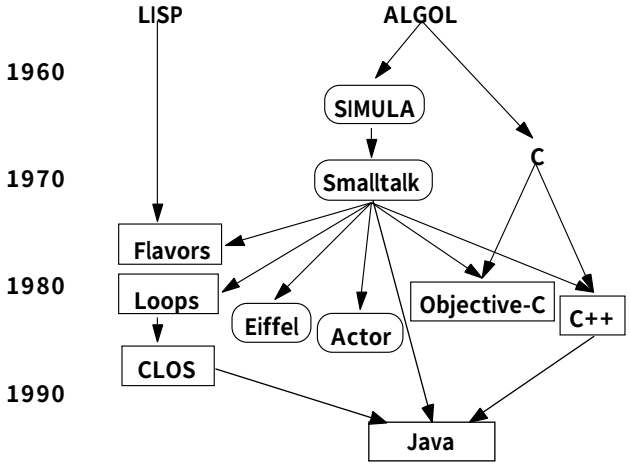
オブジェクト指向誕生（Smalltalkの系譜）

25年以上も前

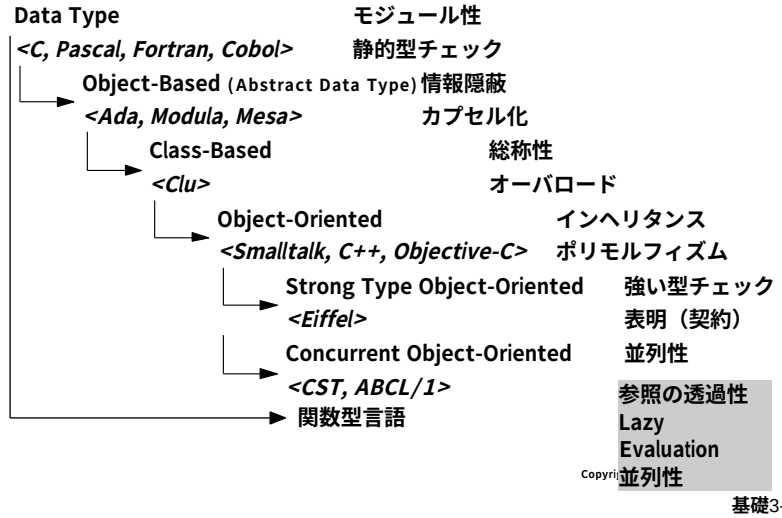
Dahl, Ole-Johan and Nygaard, K,
"SIMULA --- An Algol-Based Simulation Language",
Communications of the ACM, vol. 9, pp. 671-678, 1966.



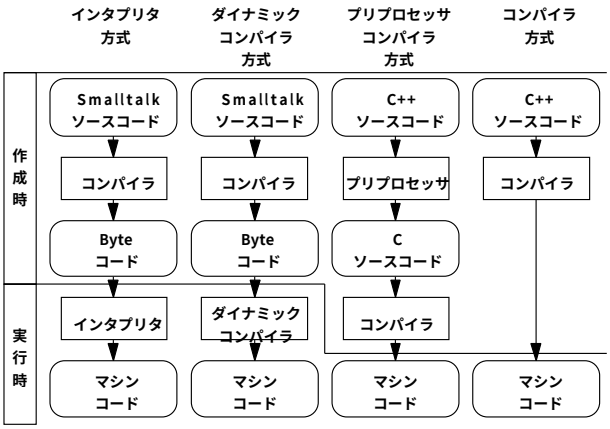
オブジェクト指向言語の歴史



プログラミング言語の傾向



オブジェクト指向言語の方式



主なオブジェクト指向言語の比較

	C++	Smalltalk	CLOS	Eiffel	Objective-C
強い型付けによるチェック	Y	N	N	Y	Y
可視性					
顧客からアクセスの制御	Y	Y	N	Y	Y
サブクラスからのアクセスの制御	Y	N	N	Y	N
パラメータ化クラス	Y	不要	不要	Y	N
表明と制約	N	N	N	Y	N
実行時のメタデータ	N	Y	Y	N	Y
ガーベージコレクション	N	Y	Y	Y	N
効率					
可能なとき静的束縛	Y	常に動的束縛	常に動的束縛	Y	Y
標準クラスライブラリー	N	Y	N	Y	Y
多重継承	Y	N	Y	Y	N

データベース

データベースの傾向

- RDBの問題点
  - 平面的(表)
  - 複合オブジェクトが扱えない
  - 継承が利用できない
  - レコード自身が識別子を持っていない
  - データ定義･データ操作･データ管理の各言語が異なる
  - 遅い
- オブジェクト指向ＤＢへのアプローチ
  - オブジェクト指向言語に永続性を導入
    - GemStone
  - RDBを包含して作り直す
  - 分散オブジェクトに対応させて、OODBのように振る舞う
    - CORBA

オブジェクト指向ＤＢ

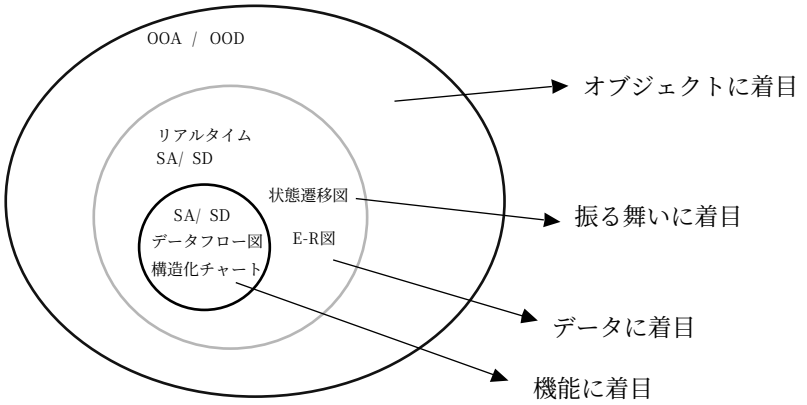
- マルチユーザーを対象とし永続的なオブジェクトを扱える
- 原理的にＲＤＢより速い
- GemStoneの場合
  - データ定義言語・データ照会言語・運用管理コマンドがすべてSmalltalk
  - プログラミング言語とのギャップがほとんどない
    - Cで書いた場合はギャップ有り
  - 分散オブジェクトに対応
  - アプリケーションサーバー機能を持つ
  - ＤＢの構成変更が容易
  - データベース管理機能はＲＤＢとほぼ同等

方法論

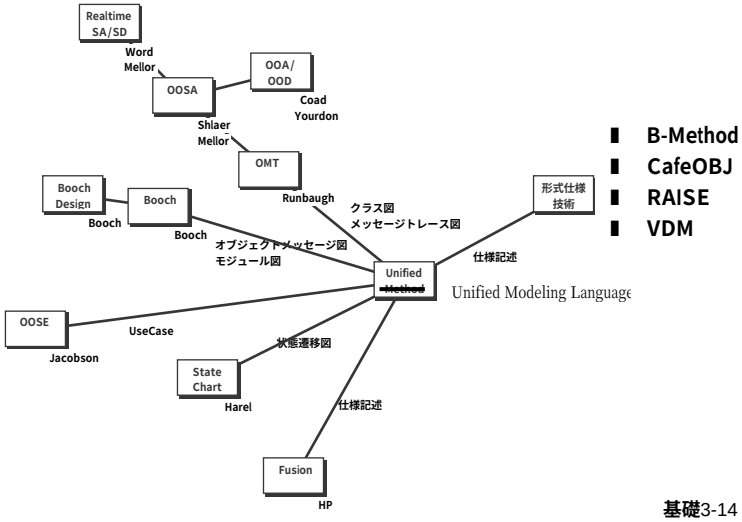
手法開発者の移行

- 構造化技法からオブジェクト指向へ
  - Martin, Yourdon, Jacobson, Wasserman, Wardなど
- 1980年代後半から顕著に

ソフトウェア開発手法の発展



ソフトウェア開発手法動向

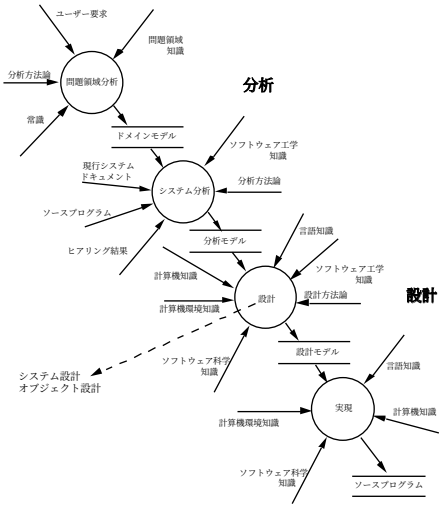


IV. オブジェクト指向分析/設計

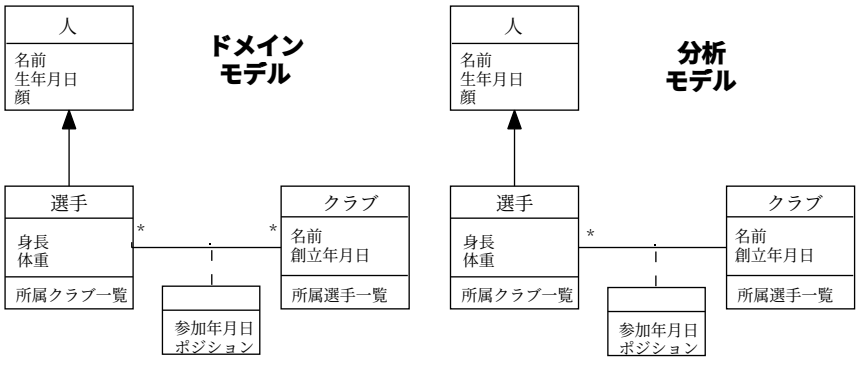
概観
Unified Methodの記法とツール
Use Case図
静的構造図
順序図
状態遷移図（STATECHART）

概観

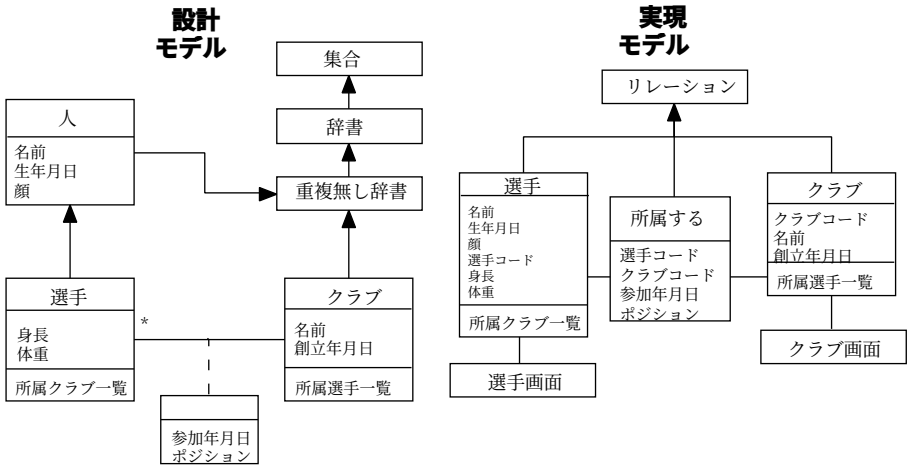
分析／設計／実現の手順（概観）



分析の手順（イメージ）



設計/実現の手順（イメージ）



実現（実行）

Unified Modeling Language (UML)の記法とツール

UseCase
クラス図
状態遷移図
順序図

Unified Methodの記法とツール

- 静的構造図（Static Structure Diagrams）
  - モデルの静的な構造を表すクラス図とオブジェクト図がある
- Use Case図
  - システムの外部インタフェースと機能を表す
- 順序図（Sequence Diagrams）
  - オブジェクトの時間順の対話を表す
- 協調図（Collaboration Diagrams）
  - オブジェクトの協調関係を表す

Unified Methodの記法とツール

- 状態遷移図（STATECHART）
  - オブジェクトの取り得る状態と、刺激への反応と動作を表す
- 活動図（Activity Diagrams）
  - 操作の効率を表すための、状態遷移機械と同値の図
- 実装図（Implementation Diagrams）
  - ソースコードの構造と、実行時の構造を表す図
- 仕様記述言語（OCL）
  - モデルの意味定義を行い、制約を記述するための形式仕様記述言語

他の方法論からの影響

- Meyer⇒事前条件、事後条件
  - Meyer, Bertrand著。二木厚吉, 酒匂寛, 酒匂順子訳。Object-Oriented Software Construction オブジェクト指向入門。アスキー, 1990年
- Harel⇒State Chart
- Booch⇒テンプレート、スコープ（visibility&scope）、オブジェクトの対話、サブシステム
  - G.Booch著。Booch法：オブジェクト指向分析と設計　第２版。アジソン・ウェスレイパブリッシャーズ・ジャパン, 1995
- Jacobson（Objectory, OOSE）⇒Use Case
  - Jacobson著。オブジェクト指向ソフトウェア工学。アジソン・ウェスレイパブリッシャーズ・ジャパン, 1995
- Wirfs-Block⇒責任（responsibilities）
- Fusion⇒メッセージの番号付け
  - D.コールマン、P.アーノルド、他著。YHPカストマ教育センター訳。オブジェクト・オリエンテッド開発設計論：The Fusion Method。プレンティスホール／トッパン, 1994年
- Embley（OSA）⇒singletonクラス、複合オブジェクト
  - Kurtz, Barry D., Embley, David, Woodfield, Scott著。畠中正行 監訳。オブジェクト指向システム分析　３つのモデルに基づくアプローチ。東京電気大学出版局, 1993年
- Ralph Jonson⇒フレームワーク、パターン
  - E. Gamma, R. Helm, R. Johnson, J. Vlissides著。オブジェクト指向における再利用のためのデザインパターン。ソフトバンク, 1995
- IBM Syntropy method⇒仕様記述言語

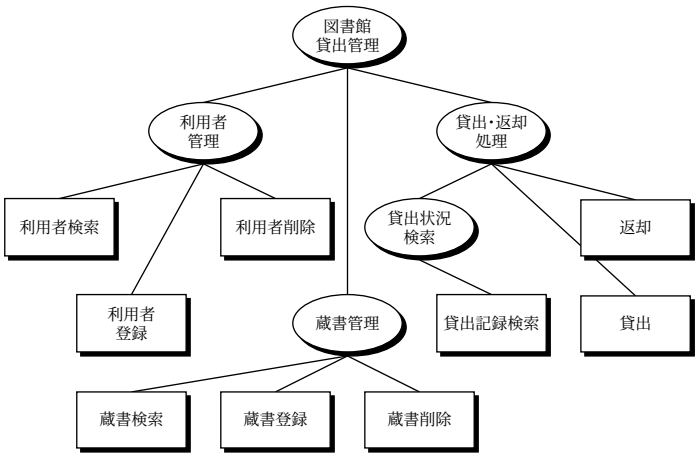
UseCase図

ユーザー要求を整理するためのツール

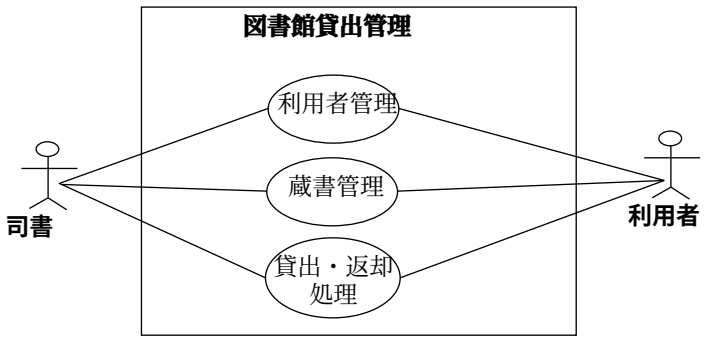
Use Case

- いくつかのオブジェクトに関わるすべての処理・振る舞いの一般化した記述
  - 入子になる
  - 個々のUse Caseの意味は、Use Caseのインスタンスである（通常は複数の）シナリオによってインフォーマルに記述する
- Use Caseとそれに関連するシナリオは、より基本的なモデルから導くことができるので、システムの注目すべき振る舞いだけを表現すればよい
- 最上位レベルのUse Caseは、システム全体およびシステムの振る舞いの境界線を視覚化するのに便利である

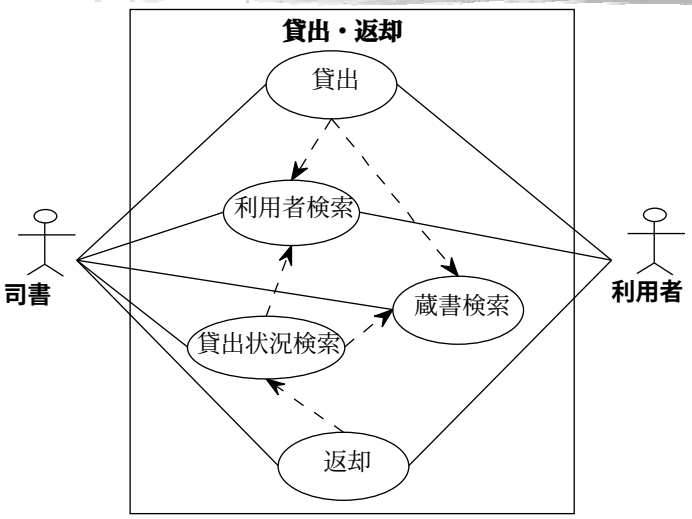
UseCaseとシナリオの関係



UseCase図



UseCase図



シナリオ

- ユーザー要求を洗い出すために作成する
  - UseCaseのインスタンスと見なすことができる
  - UMLでは、記述法がまだ曖昧
- 記述項目例
  - シナリオ名
  - 概要
  - 起動条件
    - このシナリオが起動される条件を記述する
  - 前件
    - このシナリオが動く前の状態を記述する
  - 後件
    - このシナリオ実行後に満たしているべき状態を記述する

シナリオ例

- シナリオ名
  - 現在速度を求める
- 概要
  - シャフトの回転率から現在速度を求める
- 文脈
  - 速度維持、平均速度算出、加速の計算に使う
- 前件
  - 速度変換係数 ≠ 0
- 後件
  - 現在速度 = 回転率 / (速度変換係数 * 3600)

シナリオ演習

- シナリオ名
  - 本の貸出
- 概要
  - 本の貸出を行う
- 文脈
  - 登録された利用者から貸出を請求されたときに、図書館の司書が行う
- 前件
- 後件

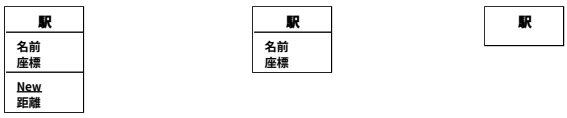
クラス図

分析・設計の核となるツール

クラス図

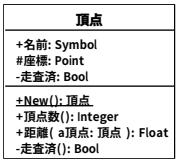
- Unified Modelの核となるモデル
  - システムの静的な論理構造を記述するためのモデル
    - システムの「状態」を構成する「長生きする」要素を示す
  - システムの一般的・包括的な記述を行う
  - オブジェクトモデルは、システムの特定の実例と振る舞いを示す
    - クラス図はクラスを含み、オブジェクト図はオブジェクト（インスタンス）を含むが、この区別は厳密なものではなく、両者を混合してもよい

クラス



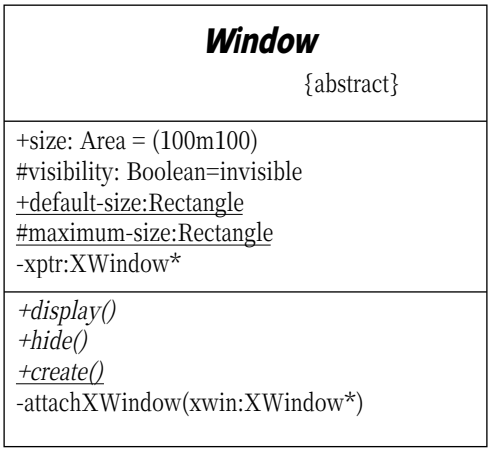
属性と操作

- スコープ
  - クラス(下線)
    - C++ではstatic
    - Smalltalkではクラス変数･クラス操作
  - インスタンス(無印)
- 見え方
  - + public
  - # protected
  - - private
- 制約
  - {自由形式}
    - {abstract},{const},{status=incomplete}
      など

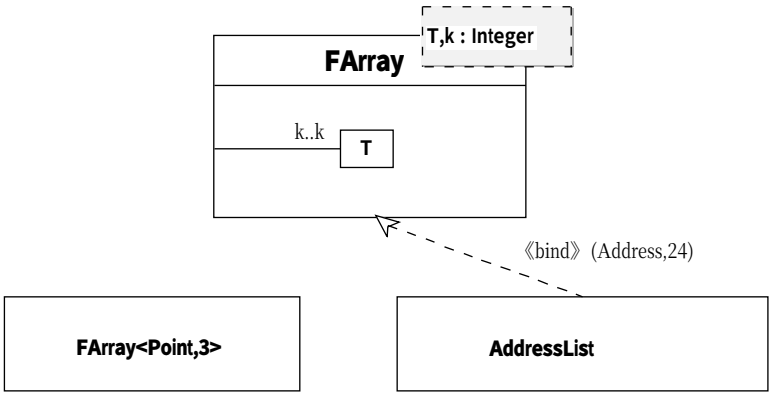


抽象クラスと抽象操作

- イタリック体で表す

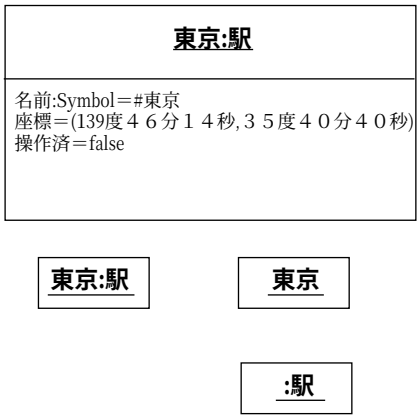


パラメータ化クラス（Template）



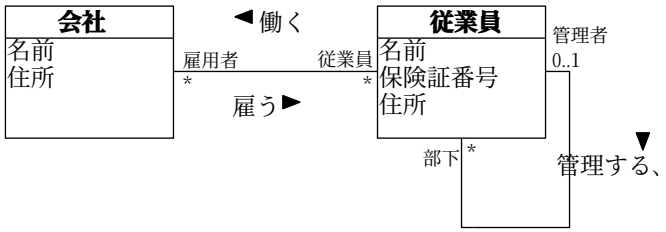
オブジェクト

- インスタンスオブジェクトの定義



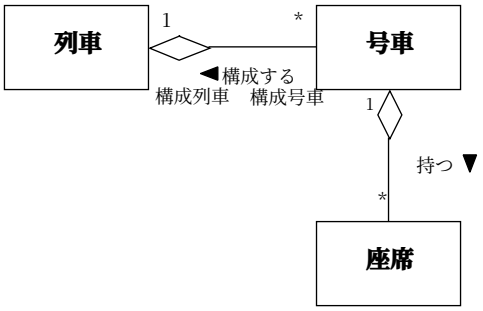
関連

- 異なるクラス間のオブジェクトの構造的関係

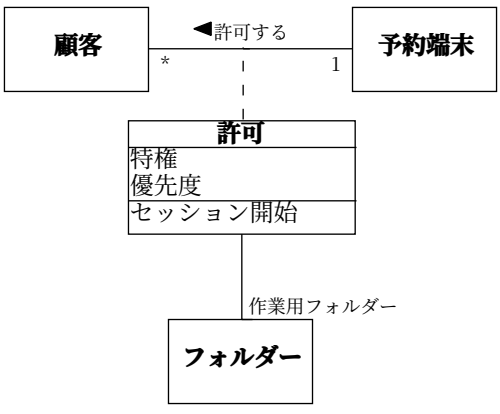


集約

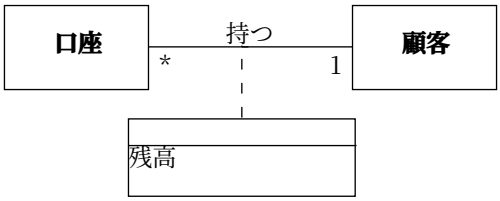
- 全体⇔部分関係



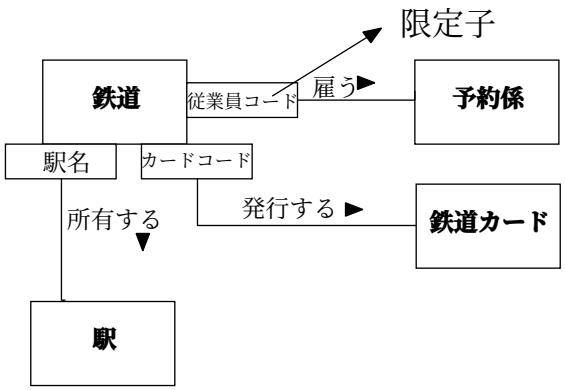
関連クラス



関連属性

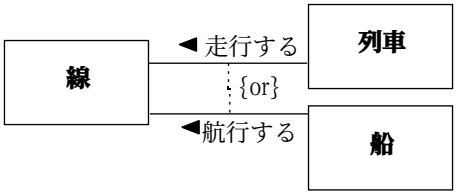


限定付き関連



制約付き関連

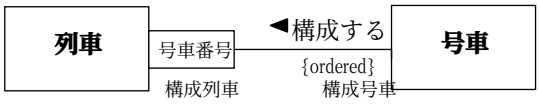
- or



基礎4-31

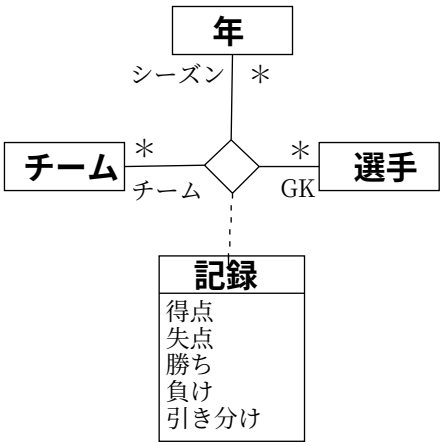
制約付き関連

- ordered

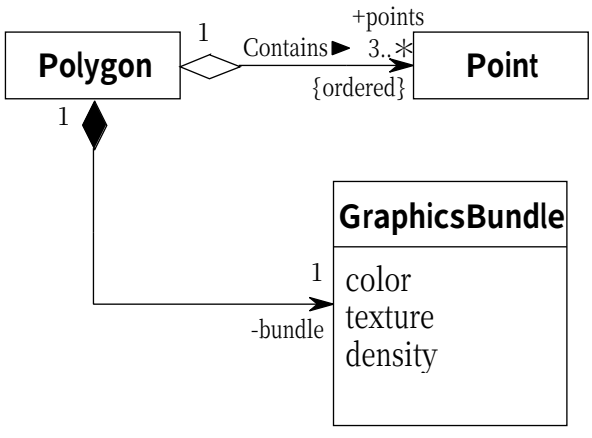


基礎4-32

N項関連

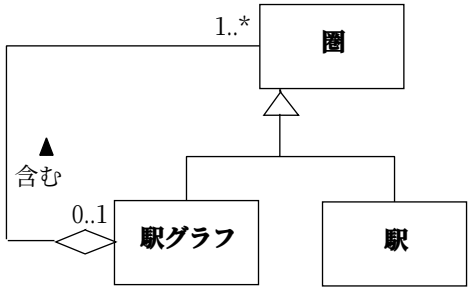

年
シーズン
＊
チーム
＊
チーム
＊
GK
選手
記録
得点
失点
勝ち
負け
引き分け

関連の装飾


Polygon
1
Contains
+points
3..＊
{ordered}
Point
1
GraphicsBundle
1
-bundle
color
texture
density

再帰構造の中の集約

- デザインパターンの一種



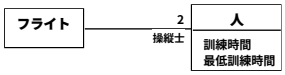
ナビゲーション式

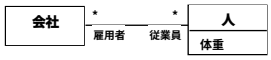
- 制約記述言語OCLの一部

item.selector	selectorはitemの属性名、またはitemに付いているリンクの反対側のロール名。結果は、属性の値か関連するオブジェクトとなる。
item.selector[限定子の値]	限定子の値を満たすオブジェクトを指定する。
set->select(論理式)	集合setの論理式を満たす部分集合を指定する。

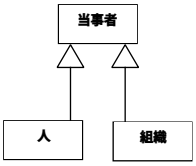
ナビゲーション式

- フライト.操縦士.訓練時間 > フライト.操縦士.最低訓練時間
- 会社.従業員->select(肩書＝"管理職" and self.報告書->size > 10)

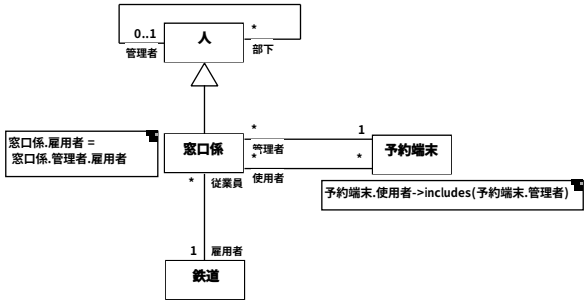




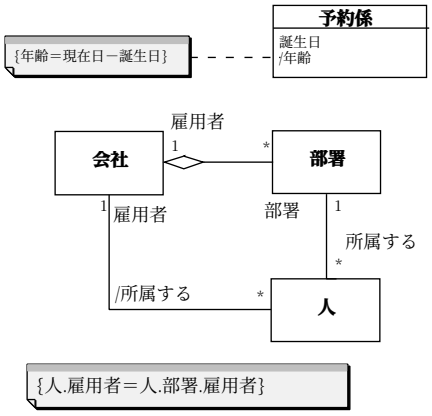
継承(汎化)



制約


人
0..1
管理者
*
部下
窓口係
窓口係.雇用者 =
窓口係.管理者.雇用者
*
管理者
*
1
予約端末
*
使用者
予約端末.使用者->includes(予約端末.管理者)
*
従業員
1
雇用者
鉄道

派生属性


予約係
誕生日
/年齢
{年齢＝現在日－誕生日}
雇用者
1
*
会社
部署
1
雇用者
部署
1
所属する
*
/所属する
*
人
{人.雇用者＝人.部署.雇用者}

クラス図の演習

- 車の所有のクラス図
  - 以下の問題文からクラス図を作成せよ。属性と多重度は示さなくてよい。
    - 人が車を所有し、運転する

クラス図の演習

- 注文のクラス図
  - 以下の問題文からクラス図を作成せよ。属性と多重度は示さなくてよい。
    - 客から電話で注文があり、商品と請求書を発送した

クラス図の演習

- 学校のクラス図
  - 以下のオブジェクトの関係を示すクラス図を作成せよ。属性と多重度は示さなくてよいが、少なくとも１０個の関連または継承関係を示せ。
    - 学校、理事会、運動場、ぶらんこ、校長、先生、生徒、教室、食堂、トイレ、本、コンピュータ、机、椅子、ドア

クラス図の演習

- ファイルシステムのクラス図
  - 以下のオブジェクトの関係を示すクラス図を作成せよ。属性と多重度は示さなくてよいが、少なくとも１０個の関連または継承関係を示せ。
    - ファイルシステム、ファイル、ディレクトリ、ファイル名、ＡＳＣＩＩファイル、実行可能ファイル、ディレクトリファイル、ディスク、ドライブ、トラック、セクター

クラス図の演習

- 住所録のクラス図
  - 以下の問題文からクラス図を作成せよ。属性と多重度は示さなくてよい。
    - 友人・会社関係の人・趣味関係の人その他の連絡先などの、電話番号・ＦＡＸ番号・ネットワークアドレス・住所・自分との関係を管理する、個人の住所録を作る
    - ネットワークアドレス＝sahara@sra.co.jp、sahara、PDF00606、EE9S-SHR、74350,606

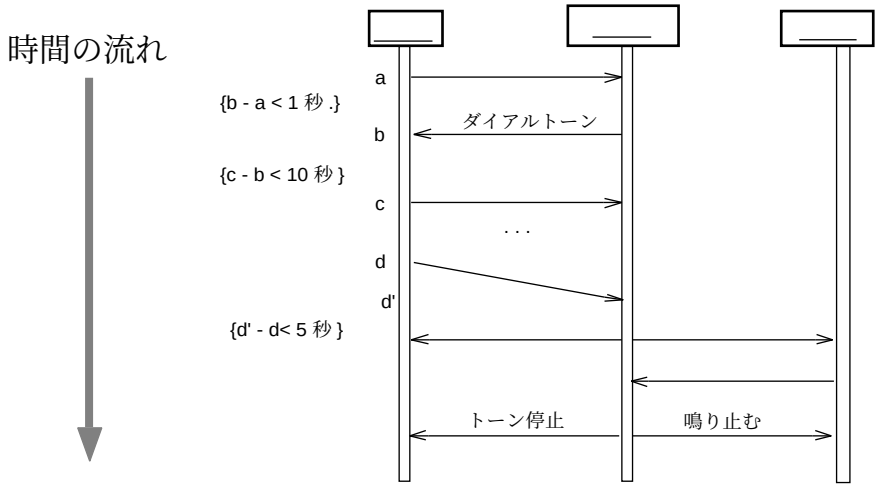
順序図

オブジェクトの協調を表すツール

順序図

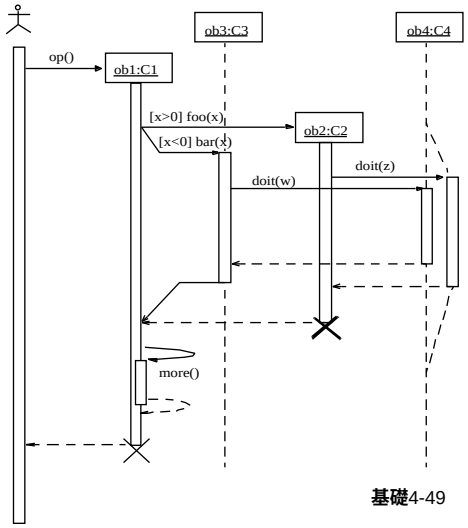
- 時間の流れと、オブジェクト間のメッセージによる対話を表す
- オブジェクトのライフタイムを表すこともできる

順序図

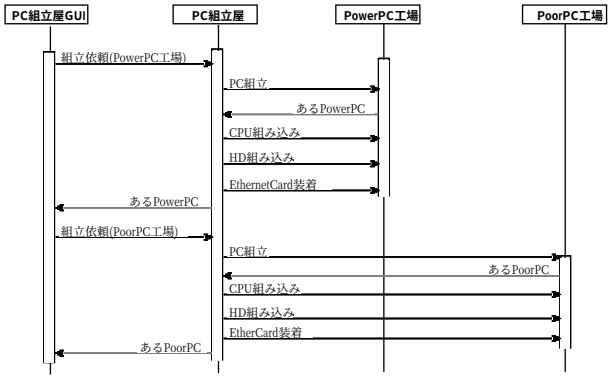


順序図

- 以下を表すことができる
  - 制御
  - 条件
  - 再帰
  - 生成
  - 破滅



順序図の例



順序図の演習

- 本の貸出の順序図
  - どういうオブジェクトがどのようにメッセージを交換すれば本の貸し出しを行えるか?
    - 幾通りもの正解がある

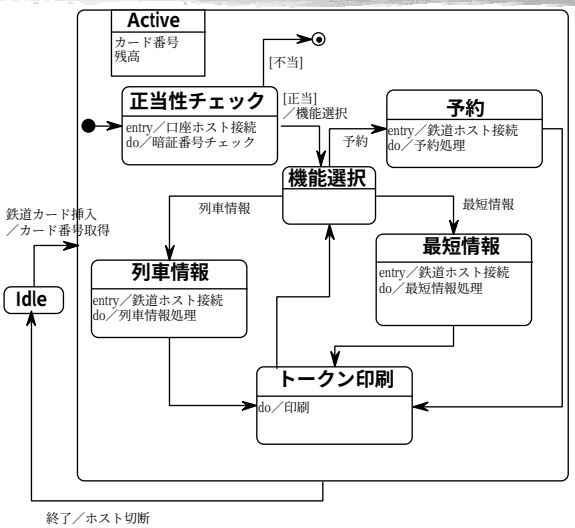
状態遷移図

振る舞いの分析・設計のためのツール

状態遷移図（STATECHART）

- 状態遷移機械
  - コンピュータやプログラムの「動き」を表すモデル
- 状態遷移図
  - 状態遷移機械を表す最も一般的な図
- STATECHART
  - 状態遷移図の最も強力な記述ツールの一つ
  - オブジェクトの振る舞いの形式仕様

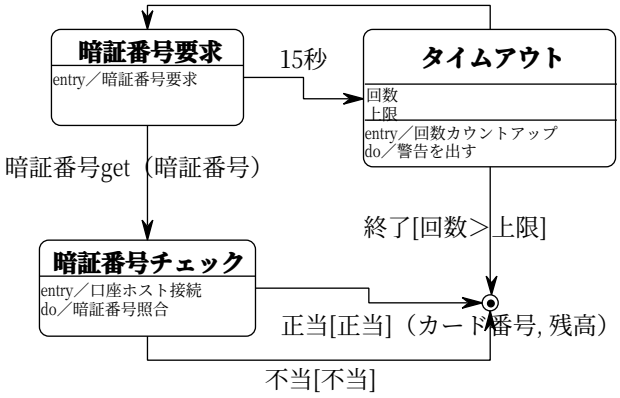
状態遷移図例



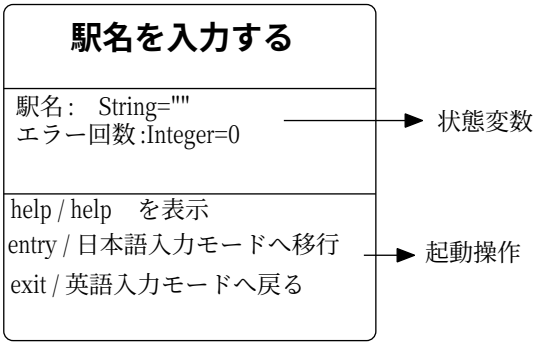
イベント

- 時の流れの中のある時点の出来事
- イベントは原子的で（想定している抽象レベルでは）割り込みできない
- 粒度は抽象化の程度により異なる
- 一般形式
  - イベント名（パラメータ：型, ...）
- イベント名として（１０秒といった）時間指定が可能
- イベントの分類をクラス図で書くことができる

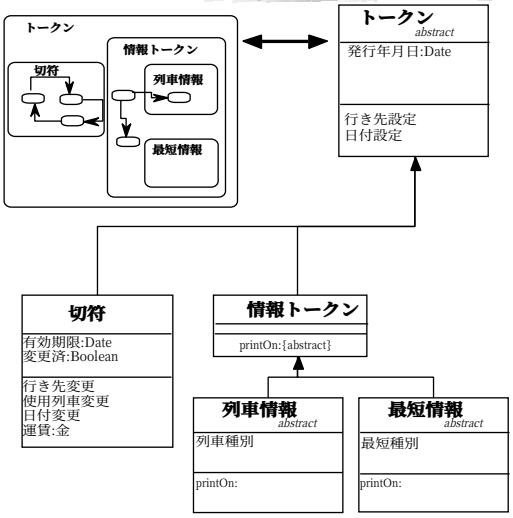
時間指定のイベント



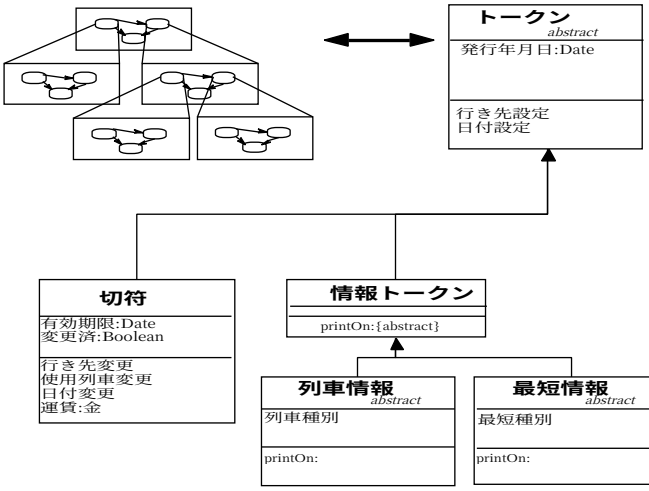
状態


駅名を入力する
駅名: String=""
エラー回数:Integer=0
状態変数
help / help を表示
entry / 日本語入力モードへ移行
exit / 英語入力モードへ戻る
起動操作

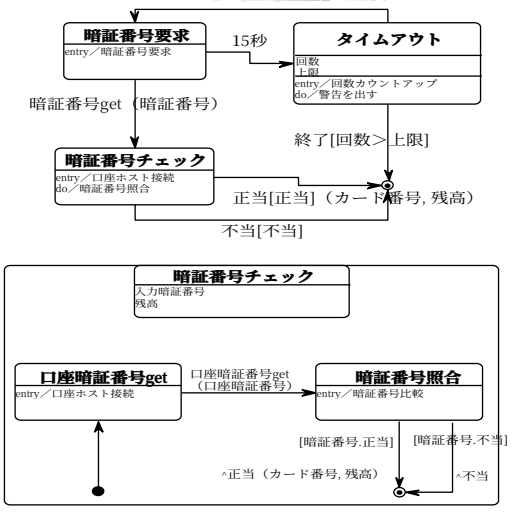
サブ状態


トークン
情報トークン
切符
列車情報
最短情報
トークン
abstract
発行年月日:Date
行き先設定
日付設定
切符
有効期限:Date
変更済:Boolean
行き先変更
使用列車変更
日付変更
運賃:金
情報トークン
printOn:{abstract}
列車情報
abstract
列車種別
printOn:
最短情報
abstract
最短種別
printOn:

入子(composite)状態



入子(composite)状態例

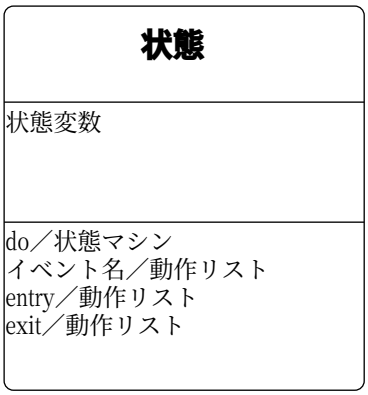


遷移

- オブジェクトが受け取る外部イベントへの反応
- 外部遷移
  - 状態を変え、操作を起動する
- 内部遷移
  - 状態は変えず、操作を起動する
- 一般形式
  - イベント（引数リスト）[条件」
    ^ターゲット.送付イベント（引数リスト）／操作（引数リスト）
    - イベント名は曖昧にならない限り省略可
    - 条件はガードとも呼び、 (OCLの)論理式で表す

動作

- 遷移によって起動されるオブジェクトの操作
  - 動作は時間がかからないと見なす



状態遷移図の演習

エディタの状態遷移図

- 挿入モードとコマンドモードとバッチ処理モードとを持つエディタの状態遷移図を作ってください
- エディターの仕様は以下のとおりです
  - 最初はコマンドモードになっている
  - コマンドモードで:キーを押すとバッチ処理モードになる
  - バッチ処理モードでreturnキーまたはctrl-Cキーを押すとコマンドモードに戻る
  - コマンドモードでaキーまたはiキーを押すと挿入モードに入り、escキーでコマンドモードに入る
  - コマンドモードでescキーを押すとビープ音を発生する

状態遷移図の演習

巡航制御の状態遷移図

- この状態遷移表の例を、状態遷移図として書いてください
- 動作は書かなくてよい

現在状態	イベント	動作	次状態
初期状態	巡航開始	巡航制御動作	スピード維持中
スピード維持中	巡航停止	巡航制御停止動作	初期状態
スピード維持中	加速	加速動作	加速中
スピード維持中	ブレーキ	スピード維持中断動作	巡航中断中
加速中	ブレーキ	加速中断動作	巡航中断中
加速中	加速停止	加速停止動作	スピード維持中
巡航中断中	巡航再開	巡航再開動作	スピード回復中
スピード回復中	設定速度到達	設定速度到達動作	スピード維持中
スピード回復中	ブレーキ	スピード回復中断動作	巡航中断中

状態遷移図の演習

- 巡航制御の状態遷移図
  - 前の問題で作成した状態遷移図の問題点を上げてください
  - 問題点を直した状態遷移図を書いてください

V. オブジェクト指向分析/設計の手順

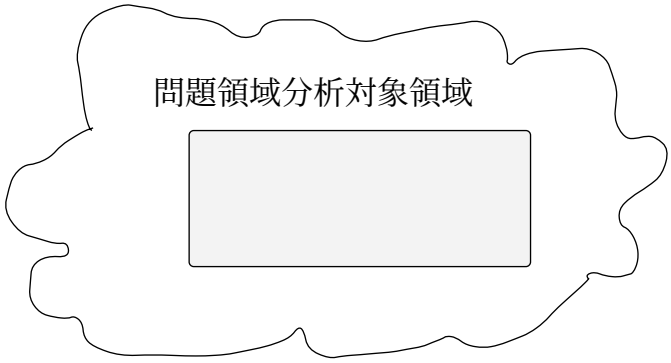
問題領域分析
システム分析
設計
実装との関係

各プロセス共通の主なステップ

- モデルの作成&修正
  - クラスとオブジェクトの認識
  - パターンがあればパターン適用
    - デザインパターン
    - 分析パターン
    - プロセス･パターン
    - アーキテクチャパターン
    - ...
- 勘所(Hotspot)の発見
  - 不変箇所と変更多発箇所(Hotspot)
- 仕様記述
- 場合によってはプロトタイプ作成
- 検証

1.1問題領域(ドメイン)分析

- 対象問題領域では「何をやっているか」をモデル化する
  - システム化する対象よりやや大きい対象領域を考える



問題領域分析の手順

- オブジェクト指向分析でなくとも必要な項目
  - ビジネスゴールの策定
  - 用語集作成
- 振る舞い要求の発見と記述
  - UseCase作成
  - 制約の発見
- ドメインオブジェクトを見つける
  - オブジェクトの名前・責任・役割を見つける
  - オブジェクトを分類し、関連･包含関係を見つける
  - オブジェクトの類型（ステレオタイプ）発見
- オブジェクトの状態変化の記述
  - 状態遷移図の作成
- 要求仕様記述
  - 仕様記述言語による記述
- 要求の検証

用語集

- 分析工程では、主要な用語を定義する
  - 例
    - 「ユーザー」「組織」「～情報」「～種別」
- 分析工程での成功の指標
  - 用語集を見ればそのプロジェクトの状態が分かる
    - 用語集がない＝絶望的
    - 用語集の出来が悪い＝成功は困難
- 特にオブジェクト指向では、「用語」はオブジェクトに直結するので大事

用語集の例

- 利用者
  - 貸出管理システムの利用者。図書館の本を借りることのできる人と、貸出を行う職員からなる。
- 職員
  - 司書の資格を持つ職員。本の貸出を行うことができる。
- 蔵書
  - 図書館で管理しなければならない個々の本。従って、同じ題名の蔵書が複数存在する。
- 本
  - 題名・著者が同じ本を1つと考える場合の、抽象概念としての「本」。
- 本の実体
  - 物理的な個々の本。このうち、図書館で管理しているものが「蔵書」。

ドメインオブジェクトの発見

- ドメインオブジェクト
  - 対象問題領域の主要なオブジェクト
    - 明確に区別できること
    - 変化が少ないこと
    - 役割や一時的構造は変化しやすい

仕様記述言語

- 自然言語による定義は曖昧になる
  - 例
    - 「6時に京都に行くから」
    - 「医師はオーダーを発行することができる」
- 仕様記述言語
  - UMLの場合
    - OCLを推薦
  - RAISEの場合
    - RSL
  - VDMの場合
    - VDM-SL
  - B-Methodの場合
    - AMN

1.2システム分析

- 対象システムで「何をやるか」をモデル化する
  - 「どうやって作るか」は我慢して書かない

システム分析の手順

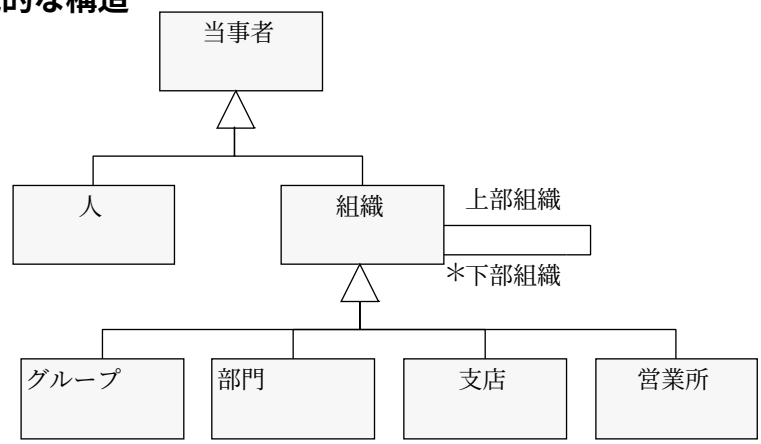
- システム分析
  - 静的構造図を作成する
    - クラス図など
  - 動的構造図を作成する
    - 状態遷移図・順序図など
  - ドメインモデルより詳細な要求仕様を記述する
    - 制約を「操作仕様」中に記述する
  - 上記作業中、常に、以下を考えておく
    - 分析パターンが適用できないか検討する
  - 勘所(Hotspot)の発見
    - 不変箇所と変更多発箇所(Hotspot)
  - その他の要求記述
    - プロジェクトの制約・運用要求などを記述
  - 分析の結果である分析モデルを検証

分析パターン

- まだ「定説」はない
- 問題領域毎に異なる可能性が強い
- 共通的なパターンの例
  - Organizational Structure(組織構造)

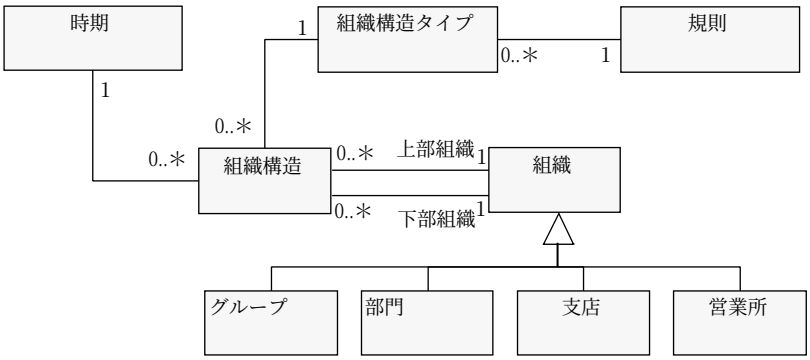
Organizational Structure (組織構造)パターン

- 常識的な構造



Organizational Structure (組織構造)パターン

- 組織は変更されやすい



1.3設計

- 対象システムを「どうやって作るか」をモデル化する
  - 分析モデルを効率・保守性･再利用性などを考えて修正する
  - アーキテクチャを決める

設計の手順

- アーキテクチャを決める
  - アーキテクチャパターンを適用する
- 設計モデルを作成する
  - 分析モデルと同じ構造だが、視点が異なるため構造が大きく変わることがある
  - デザインパターンを適用する
  - 状態遷移の実装方法を決める
  - 関連の実装方法を決める
  - 宣言的仕様の段階的洗練を行う
- 効率を検討する
  - ミクロの効率化でなく、マクロの効率化を図る
- 再利用性･保守性分析を行う
- 設計モデルを検証する

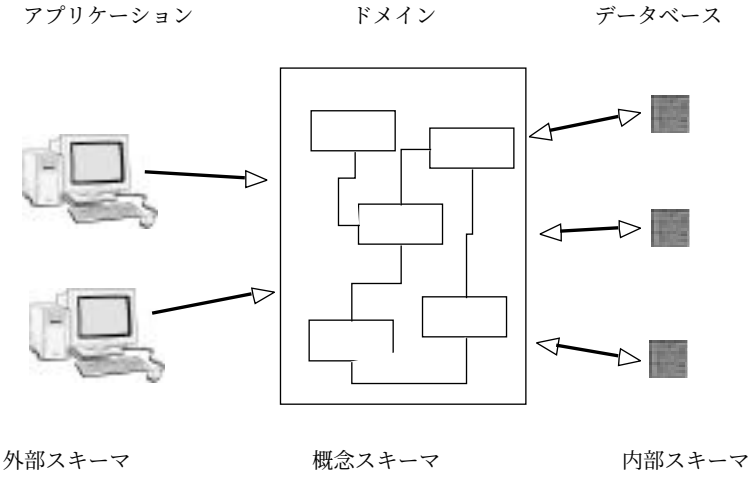
アーキテクチャパターン

- アーキテクチャパターンの例
  - 分散処理パターン
    - 重要な機能を果たすサーバーとその機能を利用するクライアントのマシンを分離する
  - レイヤーパターン
    - システムを何階層もの(一種の)仮想マシンの上に構築する

分散処理パターン

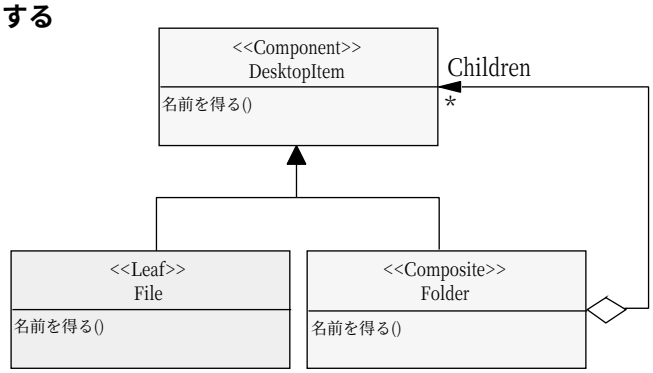
- 分散処理パターン
  - 階層
    - 2 - tier
    - 3 - tier
  - データパス
    - RPC
  - トランザクション処理
    - 2 フェーズコミット
  - 並行制御
    - 2 フェーズロック
    - 楽観的トランザクション、悲観的トランザクション
  - ...

3-tier



デザインパターン

- 例
  - 部分-全体構造を表現するために、オブジェクトを木構造で構成する

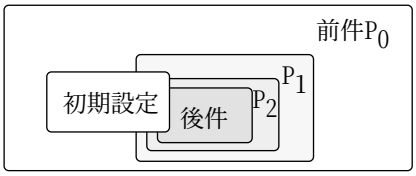


仕様の段階的洗練

- 宣言的仕様から手続き的仕様への変換公式を使って段階的に洗練する
  - グリース 著、筧 訳。プログラミングの科学。培風館、1991
  - ポター 他著、田中 監訳。ソフトウェア仕様記述　先進技法-Z言語。トッパン、1993
  - CRI 著。RAISE Method Manual。1994
  - Jean-Raymond Abrial. The B-Book : Assigning Programs to Meanings. Cambridge University Press, 1996
  - John Fitzgerald, Peter Gorm Larsen, Dines Bjørner, Cliff Jones. Modelling Systems: Practical Tools and Techniques in Software Development, Cambridge University Press, 1998
- アルゴリズムがすでに存在しているときは、それを使った方が早い
  - 島内剛一、有沢誠、野下浩平、他編集。アルゴリズム辞典。共立出版、1994

仕様の段階的洗練手順

- 問題は何であるかを把握する
  - ここでは分析は終わっているはずなので省略
- 前件・後件を見つける
  - プログラムは前件を満たすどんな状態の下で実行しても、有限時間内に実行が終わり、後件を満たす状態をもたらす
- 再帰または繰り返しにより前件を後件に近づけていく
  - 実際には、後件を緩めて、前件P 0から後件へ至る途中の「条件」（P 1 , P 2など）を導出する



後件を弱める方法

- 積項を取り除く
  - 項A, B, Cがあって、A and B and Cのような形をしているとき「積項」と呼ぶ。そのうちのひとつの項を取り除いて条件を緩める。
    - 例
      - post: ある曜日の集合->forAll(d | from <= d and d ≦to and 曜日番号(d)=指定曜日) implies 曜日回数（指定曜日,from,to）= ある曜日の集合->size
      - d ≦toを取り除いてみる
- 定数を変数に置き換える
  - 例えば、1≦d≦10というような条件があったら、1≦d≦I and 1≦i≦10に置き換えてみるという方法である。
- 変数の領域を広げる
  - 例えば、1≦d≦10という条件があったら、0≦d≦100に置き換えてみるという方法である。

不変条件と限度関数の発見

- 反復の上限を決める「蓋」の条件
  - 先ほど取り除いた積項d ≦toの否定d ＞toを使えばよいことが分かっている。
- 反復の最中変わらない不変条件
  - 残りの積項すなわちfrom ≦d **and** dayNumber(d)=dayOfTheWeekとなる
- 反復が終了することを保証する「限度関数」
  - 「限度関数」tをto _ dとすれば、tは常に正であり、かつdは増加するのだから、段々0に近づいていくことが分かり、反復が終了することを保証できる
- プログラムの大筋
  - d := from;
  - do
  - ?
  - d := d +1;
  - until d ＞ to end

一応完成したプログラムの例

- 前のプログラムで？の部分は、dの曜日が指定された曜日と等しいことを保証するコードということになる。すなわち以下のようになる。
- 曜日回数（指定曜日,from,to）≡
  - /*前件：...*/ /*後件：...*/
  - **variable** sum :Int :=0
  - d := from;
  - /*不変条件：∃DW:曜日-set,∀d∈DW・ from ≦d ∧ dayNumber(d) = 指定曜日 */
  - /*限度関数 t :: to - d */
  - **do**
  - **if** dayNumber(d)=指定曜日 **then**
  - sum := sum + 1
  - **end**
  - d := d + 1;
  - **until** d > to **end**;
  - sum

最終的なプログラム

```
曜日回数（指定曜日,from,to）≡
d := from;
while dayNumber(d) ≠ 指定曜日 do
  d := d + 1
end
(to - d) / 7＋1
```

効率の検討

- 実用的なプログラムの効率推測は非常に難しい（Knuth）
- インスタンスの数(最大･最小･平均･分散）を調べる
  - オブジェクトへアクセスするパスの最適化
- 実行・空間など各種の効率に考慮しながら、アルゴリズムを選択する
- 活動図などで実行効率を検討していく
  - シミュレーションツールなどで、モデルを評価する

再利用性･保守性分析

- 勘所（Hotspot）を発見する
  - 変更の少ない場所と、多い場所(Hotspot)を見極める
- 汎用性のある抽象データ型を作る
  - 抽象クラスを見つける
  - 多相を実現する
  - パラメータ化クラスを作る
    - 特定の型に依存しない操作を実現する
- 構造化されたモジュールを作る
  - 小さなモジュールを作る
  - 強結合のモジュールを作る
  - 少なく小さいインタフェースで実現する

1.4 実装との関係

- 実装言語に合わせて、設計モデルを実装モデルへ変換する
- プログラミング作法に従い実装モデルからプログラムへ変換する

実装言語の違いによる注意点

オブジェクトの生成
継承
関連の実装

オブジェクト生成

- 生成
  - クラスオブジェクトにメッセージを送る
    - Smalltalk, DSM（オブジェクトモデリング言語）
  - 特別な操作
    - C++, Eiffel
  - 言語の構文
    - Java
- メモリーの割当
  - オブジェクトのためのメモリー割当とオブジェクトIDの設定は自動的に行われる
    - C++は3種類のメモリー割り当てが行えるが、非常に危険
- メモリー解放
  - 自動ガーベージコレクション
    - Smalltalk, Eiffel, CLOS, Java
  - 明示的な操作
    - C++, Objective-C

継承

- 継承
  - C++はサブクラスで重複定義するには、あらかじめスーパークラスでvirtual関数を使っていなければならない
  - Smalltalkは、スーパークラスの属性がサブクラスに見えてしまうため、Smalltalkプログラミング作法に反するプログラムをするとカプセル化が犯される
- 多重継承
  - C++は多重継承できる
  - Smalltalkは多重継承できないため、実装モデルを多重継承がないように変更しておく必要がある
  - JavaはInterface型を使う
    - クラスの継承関係以外にInterfaceの継承関係もある

関連の実装

- C++の関連の実現
  - 基本的にはポインターで実現する
  - Collectionクラスがあるライブラリーを使うと良い
  - 順序付き関連には、可変長配列クラスを使うことができる
  - 関連自身をオブジェクトとして持つには、２つの辞書オブジェクトを使うのが最も単純明解
    - １つの辞書が１方向の関連を表し、もうひとつがもう１方を表す
- Smalltalkの関連の実現
  - 豊富なライブラリーでさまざまな関連の実現をサポートできる
  - 関連の実現にSetやDictionaryなどのCollectionクラスのサブクラスを使うことができる
- Javaの関連の実現
  - Dictionaryクラスは使えるが...

プログラミング作法

再利用性
保守性
頑丈さ
大規模プログラミングの作法

再利用性のための作法

- メソッドの強度を高め・小さくし・一貫性を保つ
  - 方針（Policy）と実現（Implementation）を分ける
  - 当面必要なものだけでなく、全ての組み合せに対するメソッドを書く
    - 例えば、リストの最後への挿入を書くのだったら、リストの頭への挿入も書くべき
- メソッドをできるだけ汎用にする
- 大域情報を避ける
- 継承の利用
  - ファクタリング
    - 共通コードをくくり出して、親クラスに書く
  - 委譲
    - 外部のコードをカプセル化する

保守性のための作法

- クラスをカプセル化する
  - データ構造を隠す
- 複数のリンクとメソッドをたどるのを避ける
  - 代わりに、隣のオブジェクトの操作を呼び出す
- オブジェクトの型による分岐を避ける
  - 代わりにメソッドを呼ぶ
- 公開操作と私的操作を分ける

頑丈さのための作法

- エラーに備える
  - アプリケーションエラー
    - 分析段階で考慮する
  - 低いレベルのシステムエラー
    - 優雅に死ぬことを考える
      - 必要な情報を保存し、後で回復できるようにする
  - プログラムのバグに対する防御的プログラミングをする
- プログラムを走らせてから最適化する
- 引数を検証する
- あらかじめ定義された限界を作らない
- デバッグと効率測定の仕掛けをプログラムに装備する

大規模プログラミングのための作法

- はやまってプログラミングしない
- メソッドを理解しやすくする
  - メソッドを小さくし、強度を高める
- メソッドを読みやすくする
  - 変数名を意味のあるものにする
- オブジェクトモデルと同じ名前を使う
- 名前を注意深く選ぶ
  - 意味的に異なる操作に同じ名前を付けない
- プログラム標準を使う
- モジュールにまとめる
- クラスとメソッドのドキュメントを書く
- 仕様を公開する

V. オブジェクト指向分析/設計の手順のまとめ

- 問題領域分析・システム分析・設計工程を経て実装する
  - 「何を」を分析し「どうするか」を設計する
- 万能の言語はない
  - オブジェクト指向言語の特性を見極めて実装する
    - 特性によっては、設計工程に影響する
- プログラミング作法を守る

参考文献

- オブジェクト指向
  - Bertrand Meyer. 酒匂寛, 酒匂順子 訳. Object-Oriented Software Construction オブジェクト指向入門. アスキー, 1990
    - ＯＯの必要性をプログラミング技術面から分かりやすく解説
  - Jacobson, Ivar et al. Object-Oriented Software Engineering - A Use Case Driven Approach. Addison-Wesley, 1992.
    - UseCaseの教科書
  - 青木淳. オブジェクト指向システム分析設計入門. ソフト・リサーチ・センター, 1992
    - OOA/OOD/OOPの分かりやすい解説
  - 青木 淳。例題による！！ オブジェクト指向分析設計テクニック。ソフト・リサーチ・センター, 1994
    - OOD/OOPの分かりやすい解説
  - 佐原伸。オブジェクト指向システム分析／設計　Ｑ＆Ａ。ソフト・リサーチ・センター, １９９５年１１月

参考文献

- デザインパターン
  - C.アレグザンダー、パタン・ランゲージ、鹿島出版会、1992年
    Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1994.
    - デザインパターン教科書の定番
  - Sherman R.Alpert, Kyle Brown, Bobby Woolf、The Design Patterns Smalltalk Companion, Addison Wesley, 1998
    - Smalltalkに最適化したデザインパターン
  - 佐原伸。デザインパターン オブジェクト指向システム分析／設計技法。ソフト・リサーチ・センター, 1999年5月

参考文献

- 分析パターン
  - Martin Fowler, "Analysis Patterns: Reusable Object Models"Addison-Wesley, 1997
    David Hay, "Data Model Patterns: Convention of Thought", DorsetHouse, 1996
- イディオム
  - 青木淳、Smalltalkイディオム、SRC、1997年
  - Kent Beck, "Smalltalk Best Practice Patterns", Prentice Hall, 1997
- プロセスパターン
  - Tom DeMarco, Timothy Lister、ピープルウェア、日経BP社、1989
  - J.Rumbaugh, M. Blaha , W.Premerlani, F.Eddy, and W.Lorensen. 羽生田訳. オブジェクト指向方法論：ＯＭＴ. トッパン, 1992
    - 現時点で最も完成されたOOA/OOD技法OMTの教科書
  - J.Rumbaugh, M. Blaha , W.Premerlani, F.Eddy, and W.Lorensen。Solution Manual Object-Oriented Modeling and Design。Prentice Hall, 1991
    - 上の本の解答集

参考文献

- 仕様記述
  - OMG Unified Modeling Language Specification (draft) . Object Management Group, Inc.他, Version 1.3 alpha R5, March 1999
    - UMLの仕様書。制約仕様記述言語OCLの仕様を含んでいる。
  - The RAISE Language Group著. The RAISE Specification Language. Prentice-Hall, 1992
    - RAISE/RSLの仕様書
  - John Fitzgerald, Peter Gorm Larsen, Dines Bjørner, Cliff Jones. Modelling Systems: Practical Tools and Techniques in Software Development, Cambridge University Press, 1998
    - VDM Tool簡易版を使った形式手法のCD-ROM付き入門書。
  - 中川 中著. 代数的仕様記述言語 CafeOBJ. SRA, 1993
    - Cafe OBJの解説
  - Jean-Raymond Abrial. The B-Book : Assigning Programs to Meanings. Cambridge University Press, 1996
    - B Methodの教科書

参考文献

- 記法
  - OMG Unified Modeling Language Specification (draft) . Object Management Group, Inc.他, Version 1.3 alpha R5, March 1999
- パターン･ホームページ(http://hillside.net/patterns/)
  - パターンに関連するすべての情報があるWebページ
- 形式手法ホームページ(http://hello.to/fm/)
  - 仕様記述を含む、形式手法すべてについての情報があるWebページ