2005年11月5日
第1回

ソフトウェア設計技術研修コース

兼宗 進 (一橋大学 助教授)

1.講習のねらい

1-1 ソフトウェア設計技術

1.数多くの製品にソフトウェア

2.理解の必要性

- ■全体像の把握
- ■発明者へのインタビュー

3.理解は難しい

- ■進歩が速い(特にIT技術)
- ■つかみ所がない(情報: 実体があるようでない)
- ■多様・複雑

1-2 今回の講習

1. 具体例を交えて概観する
2. 進め方
   - ■ ソフトウェア技術概論
   - ■ オブジェクト指向プログラム
   - ■ 設計書解読、インタビュー手法

1-3 講師

1.兼宗進

- ■一橋大学 助教授
- ■専門はプログラミング言語、情報検索など
- ■企業でソフトウェア開発を経験

2.中谷多哉子(11/12, 11/19)

- ■有限会社エス・ラグーン取締役社長
- ■和歌山大学、九州工業大学客員教授
- ■オブジェクト指向設計を対象としたコンサルティング会社を経営

1-4 講習ページ

1.資料や補足の情報をWebに置きます

2.アクセス方法

- ■http://kanemune.cc.hit-u.ac.jp/
- ■左側の「弁理士05」をクリック
- ■IDに「jpaa」、パスワードに「sw2005」を入力

1-5 講習の流れ

1.ソフトウェア設計技術の概要をつかむ(今回

- ■計算機の構成、プログラミング
- ■Webアプリケーション技術
- ■データベース技術、セキュリティ技術

2.実践的な取り組み(次回以降)

- ■オブジェクト指向設計
- ■モデリング、インタビューなど

2.計算機の仕組み

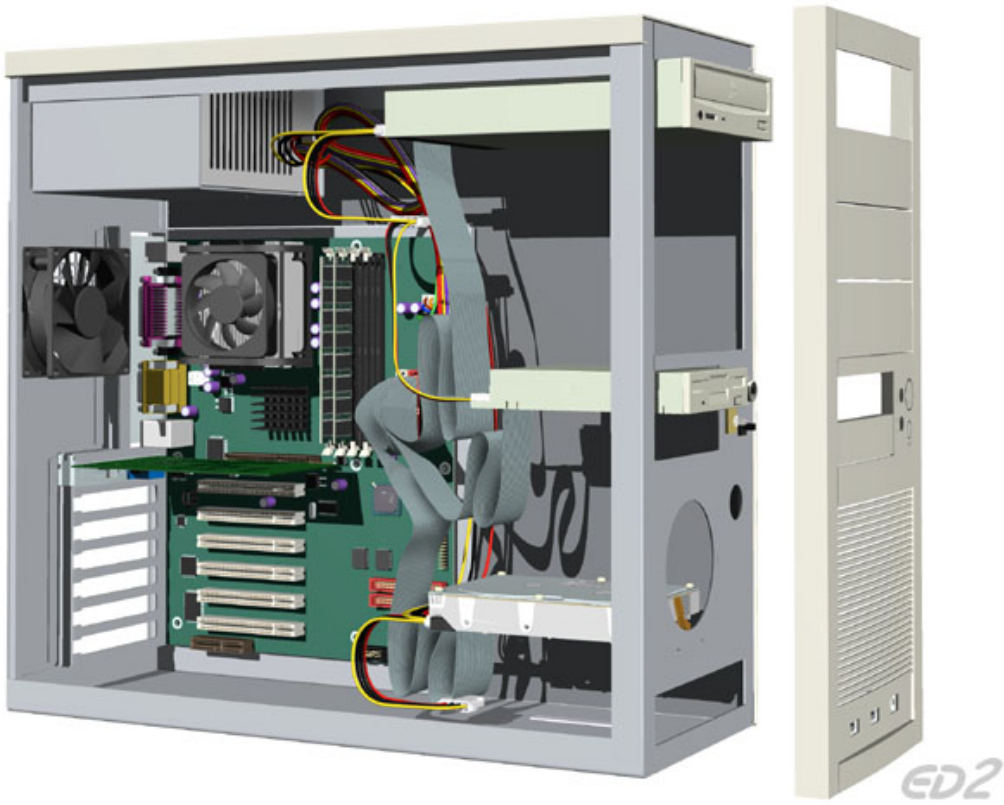
2.1 計算機

1.計算機の５大機能

- ■入力装置
- ■出力装置
- ■演算装置
- ■記憶装置
- ■制御装置

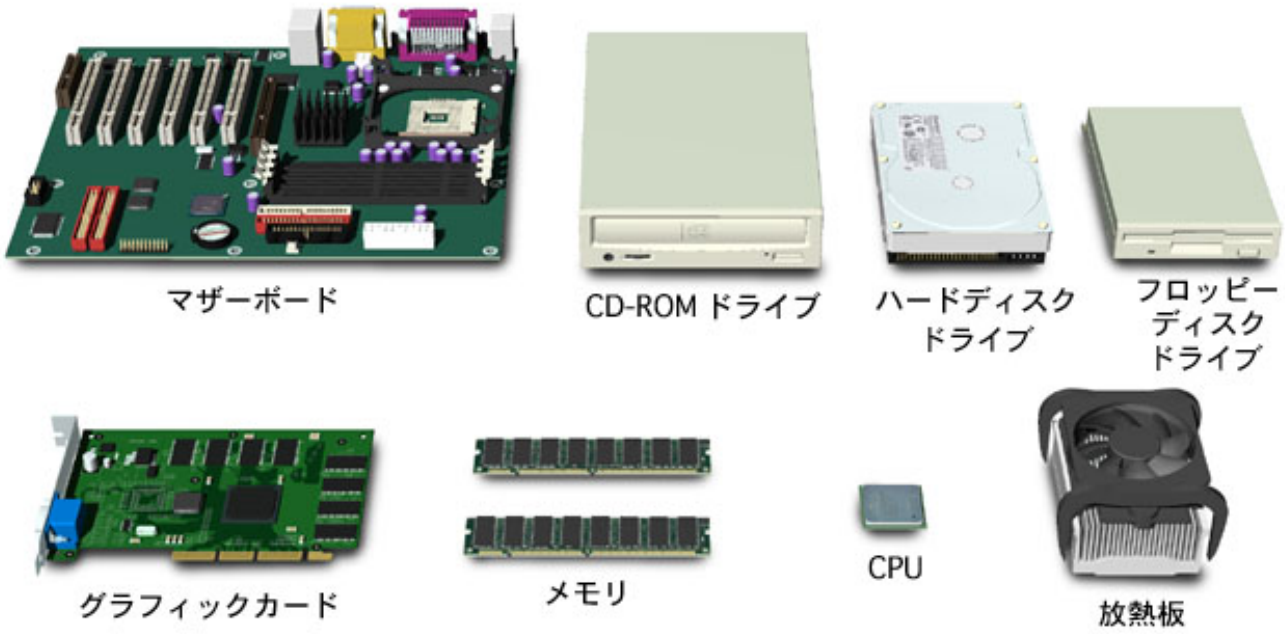
http://www.sugilab.net/jk/joho-kiki/readme.html

2.2 計算機の中身



http://www.sugilab.net/jk/joho-kiki/readme.html

2.3 主な部品



http://www.sugilab.net/jk/joho-kiki/readme.html

2.4 部品の役割

1.CPU
- ■計算。プログラムで操る

2.メモリ
- ■プログラムやデータを一時的に置く
- ■CPUのメモ欄。画面を作る
- ■電源offで消える。速い

3.ハードディスク
- ■ずっと残る。(メモリより)遅い

2.5 OSの役割

1.OS: オペレーティングシステム

2.仮想機械を作る

- ■計算機は部品や接続されている機器を見れば1台ずつ違うが
- ■それを隠して統一した機械に見せかける
- ■アプリケーションソフトからは「Windows」「Macintosh」「Linux」などの計算機に見える

2.6 計算機は汎用機械

1.専用機械(自転車、時計、...)

- ■最初から何かできる
- ■他のことはできない

2.汎用機械(計算機)

- ■最初は何もできない
- ■プログラムを入れると何かできる
  (プログラムを変えると別のことができる)

3.プログラミング入門

3.1 プログラミングを体験する

1.プログラム: 見えない存在

■話を聞いてもわかったような…
→ 動かしてみればわかる！

2.ドリトル

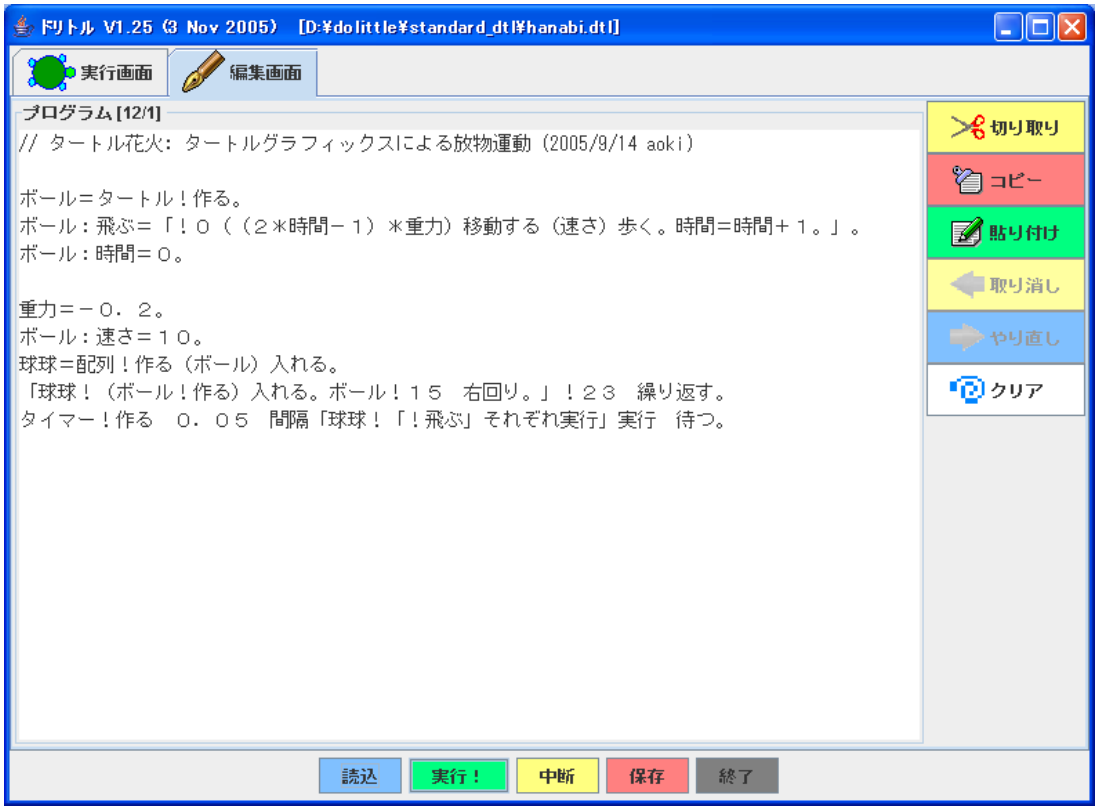
■講師(兼宗)が作った言語

■オブジェクト指向: 大人でも難しい
→ 中学生から学べるようにした

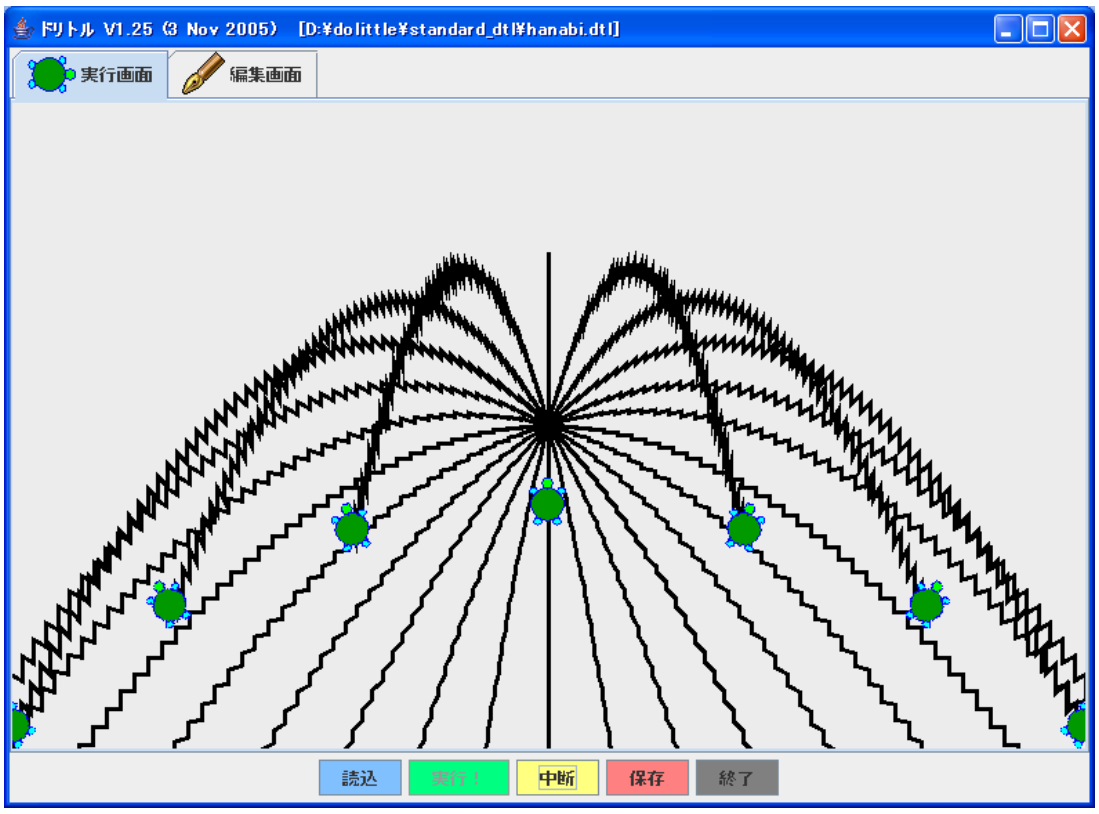
3.今回の講習では

■今回は講師がその場でプログラムを作る
→ 動きを見て体験してください

3-2


ドリトル V1.25 (3 Nov 2005) [D:¥dolittle¥standard_dtl¥hanabi.dtl]
実行画面
編集画面
プログラム[12/1]
// タートル花火: タートルグラフィックスによる放物運動 (2005/9/14 aoki)
ボール＝タートル！作る。
ボール：飛ぶ＝「！０（（２＊時間－１）＊重力）移動する（速さ）歩く。時間＝時間＋１。」。
ボール：時間＝０。
重力＝－０．２。
ボール：速さ＝１０。
球球＝配列！作る（ボール）入れる。
「球球！（ボール！作る）入れる。ボール！１５　右回り。」！２３　繰り返す。
タイマー！作る　０．０５　間隔「球球！「！飛ぶ」それぞれ実行」実行　待つ。
切り取り
コピー
貼り付け
取り消し
やり直し
クリア
読込
実行！
中断
保存
終了

3-3


ドリトル V1.25 (3 Nov 2005) [D:¥dolittle¥standard_dtl¥hanabi.dtl]
実行画面
編集画面
読込
実行！
中断
保存
終了

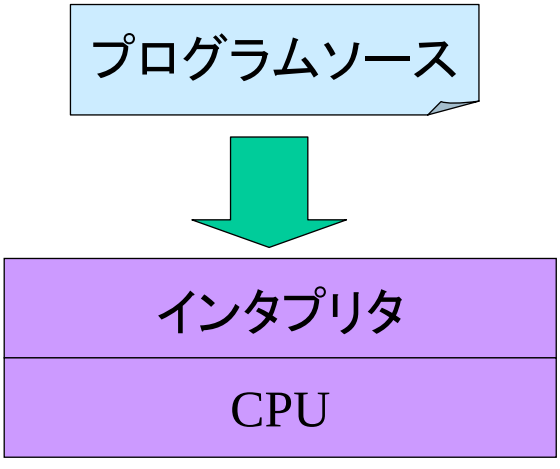
4.ソフトウェア開発

4-1 プログラムの実行方式(1)

1.インタプリタ

■プログラムを解釈しながら実行

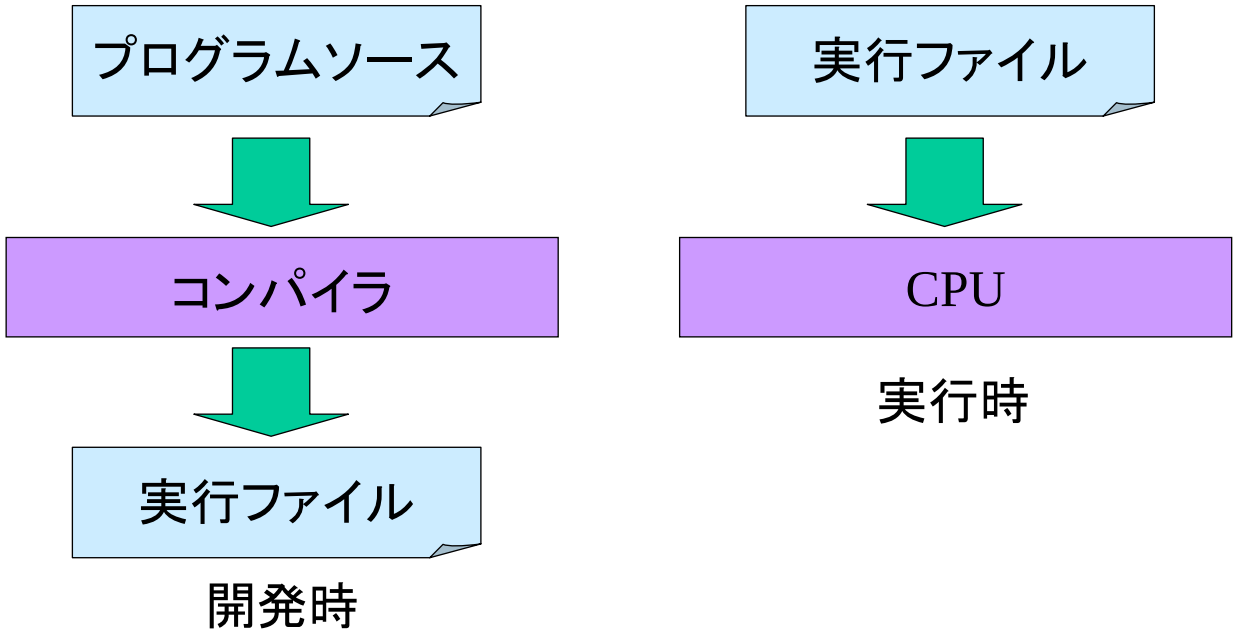
■BASIC、Perl、JavaScript、PHPなど



開発時、実行時

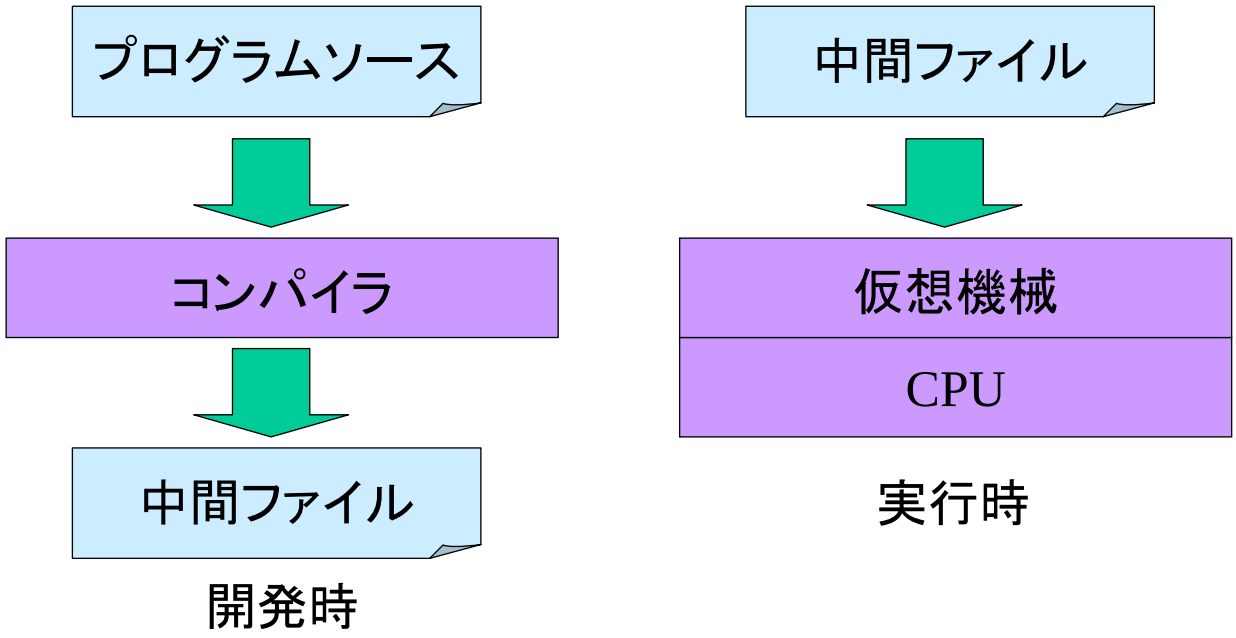
4-2 プログラムの実行方式(2)

1. コンパイラ
   - ■ プログラムを機械語に変換してから実行
   - ■ FORTRAN、COBOL、C++、(Java)など



4-3 プログラムの実行方式(3)

1.仮想機械
   - ■プログラムを仮想機械で実行
   - ■Java (Write once, run anywhere)

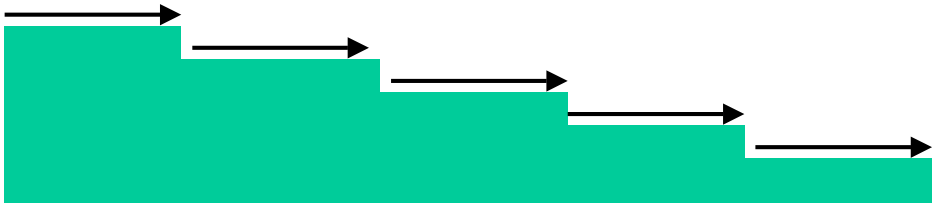


4-4 開発の段階とサイクル(1)

1.ウォーターフォール開発

■水が階段を流れ落ちるように段階的に進める

■企画→要件定義→設計→実装→テスト



■進んだら戻れない(最大の弱点)

4-5 開発の段階とサイクル(2)

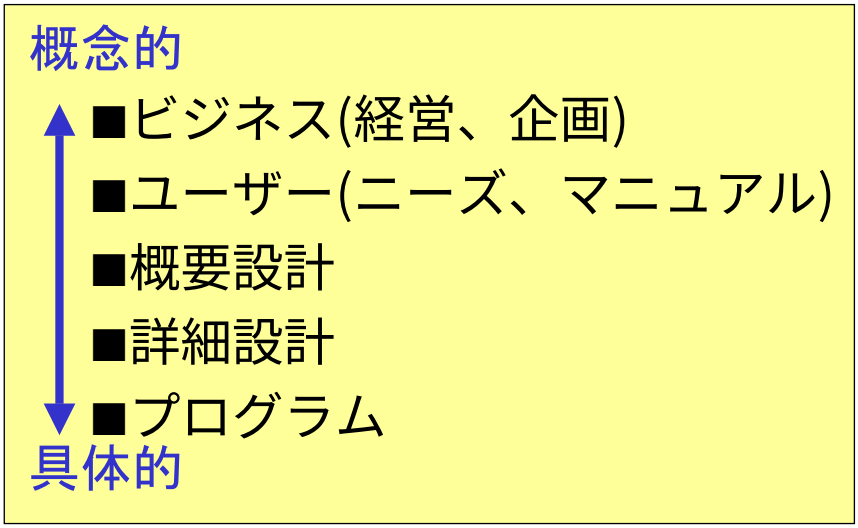
1. アジャイル開発

- 俊敏な開発。繰り返す(イテレーション)
- XP、RUP、スクラムなど
- 仕様は変わるのが前提。顧客参加
- 企画→[要件定義→設計→実装→テスト]×n回

4-6 設計の詳細レベル

1.設計には多くの段階

2.チャートを含め、どのレベルの話かは重要



4-7 小まとめ

1.ねらい: 計算機とプログラムを知る

■ イメージをつかめたでしょうか?

2.計算機

■5大機能、構成部品、汎用機械

3.プログラム

■命令を伝える。文法がある。文字で書く。

■順に実行する。繰り返せる

■オブジェクトに命令を送る。オブジェクトに命令を定義できる

■タイマーで間隔を空けて繰り返せる

■複数の部分を並行して実行できる(スレッド)

5.言語の進化

5-1 言語の進化

1.非構造化言語(1950～)
- ■FORTRAN、COBOL、BASIC、…
- ■フローチャート

2.構造化言語(1970～)
- ■PASCAL、C、…
- ■構造化チャート(PAD、HCP)

3.オブジェクト指向言語(1990～)
- ■C++、Java、…
- ■オブジェクト指向チャート(UML)

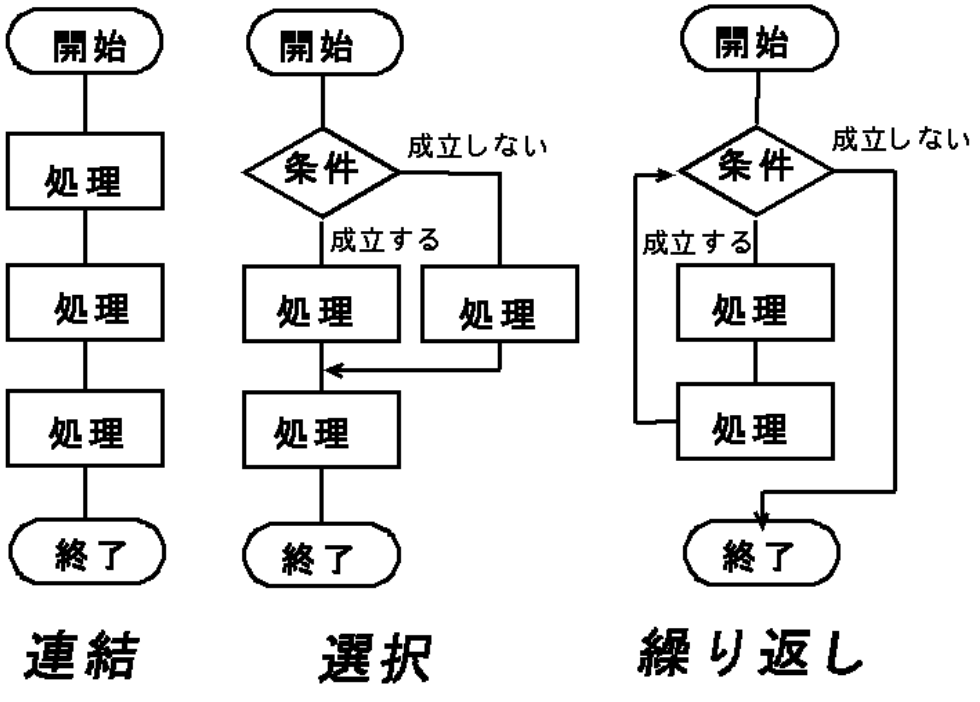
5-2 非構造化言語

1.原始的なプログラム

- ■逐次、分岐、繰り返しなどの基本要素のみ
- ■処理の流れであり、全体の処理をグループ化しようという発想はない
- ■処理をする主体の区別もなく、１人の中の仕事の流れでしかない
- ■数値計算など、処理を記述できればいいもの
- ■数百行は無理なく書けるので教科書の例題
- ■１ページに収まるような全体の説明図

5-3 フローチャート

1.goto(流れの飛躍)→意味が不明瞭になる



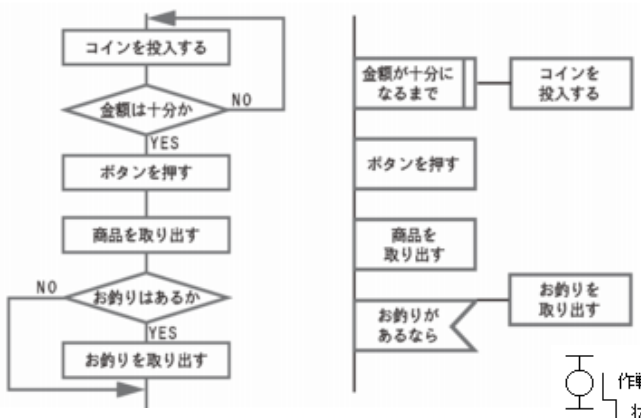
http://mukun_mmg.at.infoseek.co.jp/mmg/cpp.html

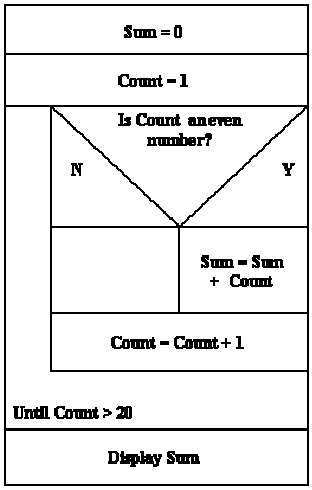
5-4 構造化言語

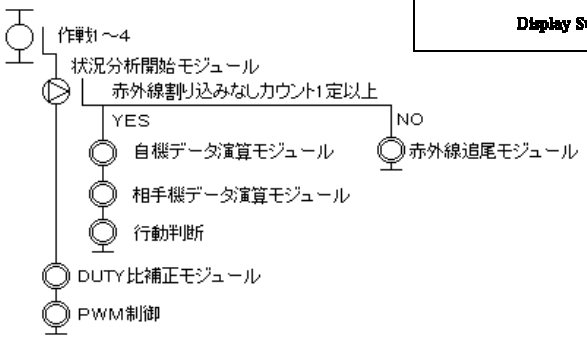
1. 流れの構造を作る
   - ■gotoを繰り返しなどにする
   - ■流れを手続きなどにまとめる
2. (例)
   - ■構造化前
     - ➡a→b→c→d→e→f
   - ■構造化後
     - ➡A→B
     - ➡A: a→b→c
     - ➡B: d→e→f

5-5 構造化チャート

1. PAD、HCP、NSチャートなど







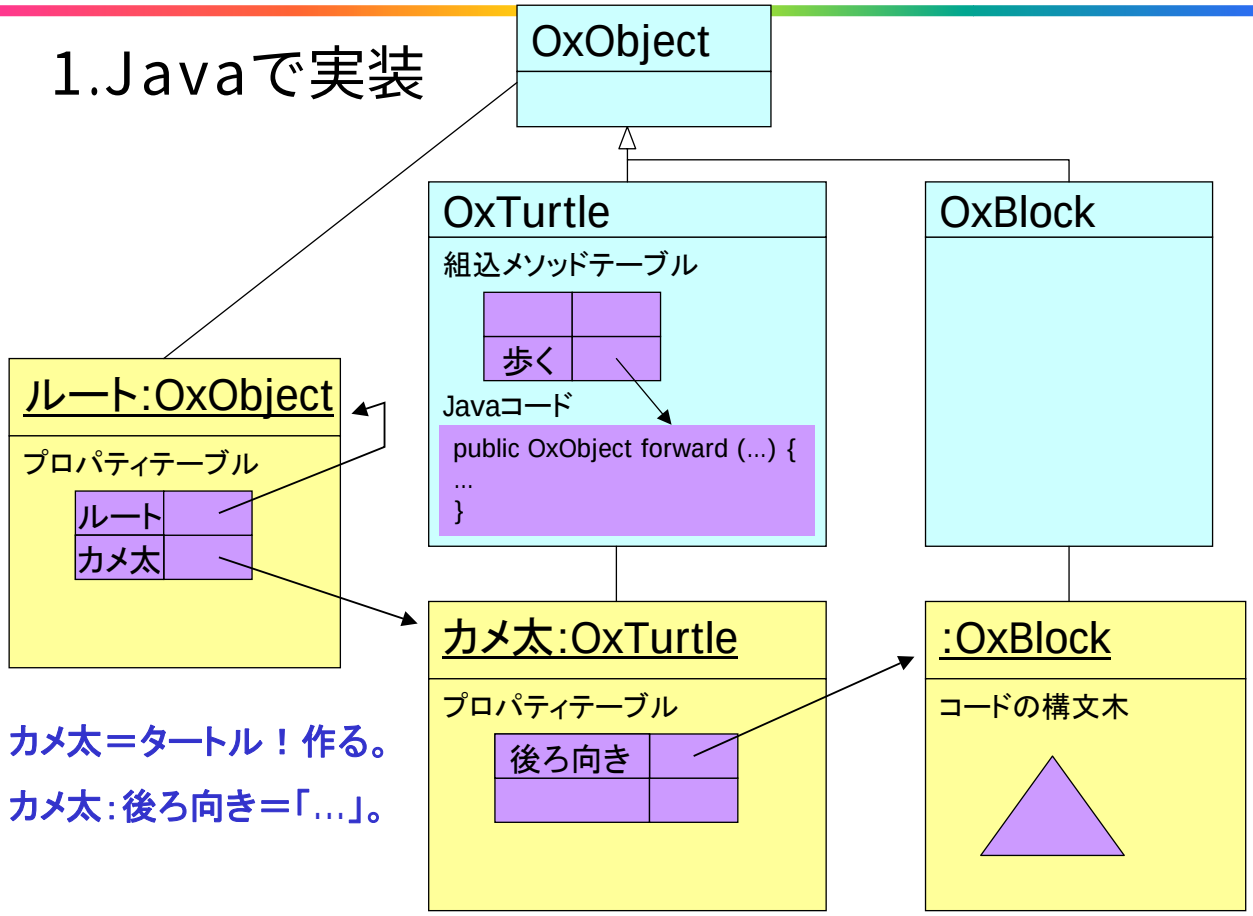
http://users.evtek.fi/~jaanah/IntroC/DBeech/images/3gl_ns_samp1.gif
http://www2.denshi.numazu-ct.ac.jp/mirsdoc/mirs97/03/shousai/soft/hcp_mis.gif
http://www.soi.wide.ad.jp/class/20030007/slides/07/16.html

5-6 オブジェクト指向言語

1. データ＋プログラムで整理
   - ■オブジェクト(データ+プログラム)
     - ➡人間と同じ
     - ➡記憶を持ち、何かをできる
   - ■オブジェクトを単位として考える
     - ➡人を組織化するのと同じ
     - ➡流れでなく、オブジェクトの構造を作る
2. オブジェクト指向設計用のチャート
   - ■UMLなど。次章で説明
   - ■ここでは(ドリトルを覚えているうちに)ドリトルの中身を見ておく

5-7 ドリトルのクラスとオブジェクト例

1. Javaで実装



5-8 なぜ言語は進化するのか(1)

1.ニーズの変化

2.バッチ処理から

- ■文字プログラム
- ■決まった処理を実行する(定型の流れ)

3.対話処理へ

- ■GUIプログラム(ウィンドウ、ボタン、...)
- ■決まった処理はない(定型の流れでは扱えない)

5-9 なぜ言語は進化するのか(2)

1.プログラムの大規模化

- ■村(数十人)→都市(百万人)→国家(数億人)
- ■量が変わると質が変わる
- ■ 分割して統治せよ(古代からの知恵)

2.進化

- ■分割前(ある人の仕事を手順化)
- ■流れの分割(部署内の仕事の流れを整理)
- ■データの分割(「データ+流れ」で分割)
- ■オブジェクト同士の組織化、パターン化

6.現代の表記法(UML)

6-1 UMLとは

1.UML: Unified Modeling Language

- ■オブジェクト指向設計の標準チャート
- ■たくさんの図の寄せ集め。よく使うのは5,6種類
- ■使いこなせている人は少数

2.学ぶ理由

- ■明細書に書くのはフローチャートだが
- ■発明者の文書に出てくるかもしれないので、概要を知っておいて損はない

6-2 例題で考える

1.牛丼屋の注文

■登場人物: 店員(接客係、調理担当)、お客さん

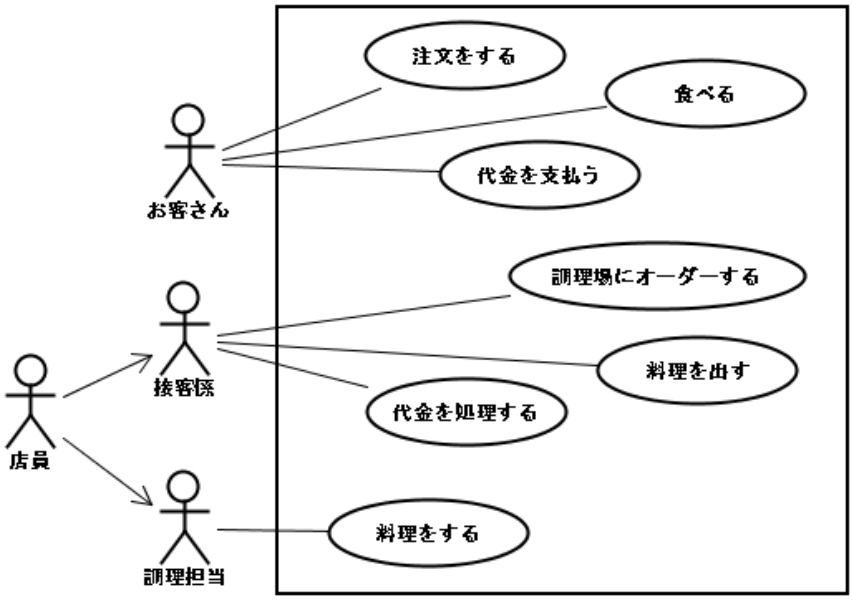
■行動: 注文、調理、食事、支払い

■オブジェクト: 登場人物、料理、注文など

6-3 ユースケース図

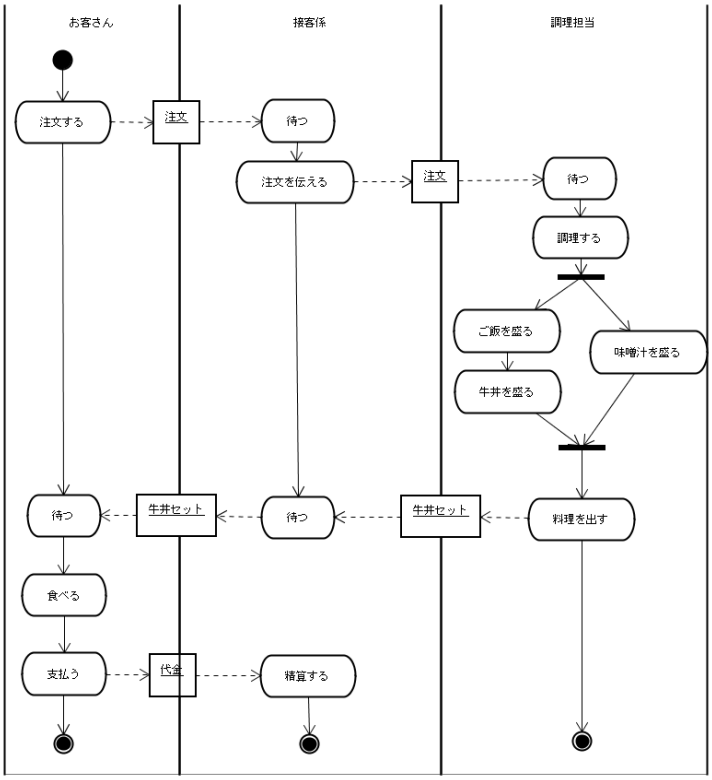
1.アクター(登場人物)と行動

■それぞれの行動を箇条書きで表す
(ユースケース記述: フローチャートに近い?)



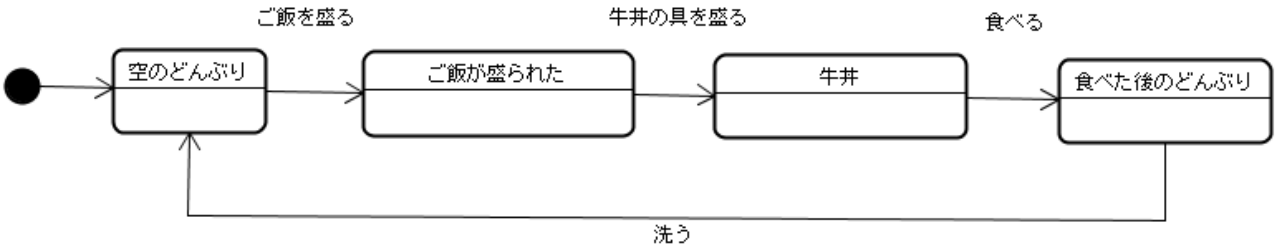
6-4 アクティビティ図

1.アクターの行動を記述。相互作用を書ける



6-5 ステート図

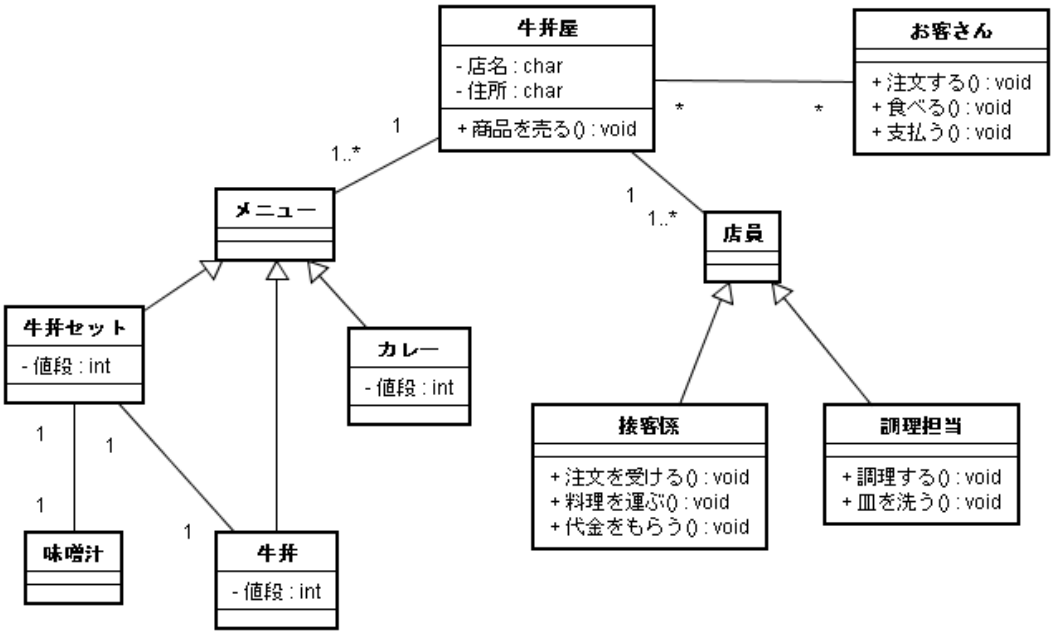
1.オブジェクトの状態がどのように変わるか



6-6 クラス図

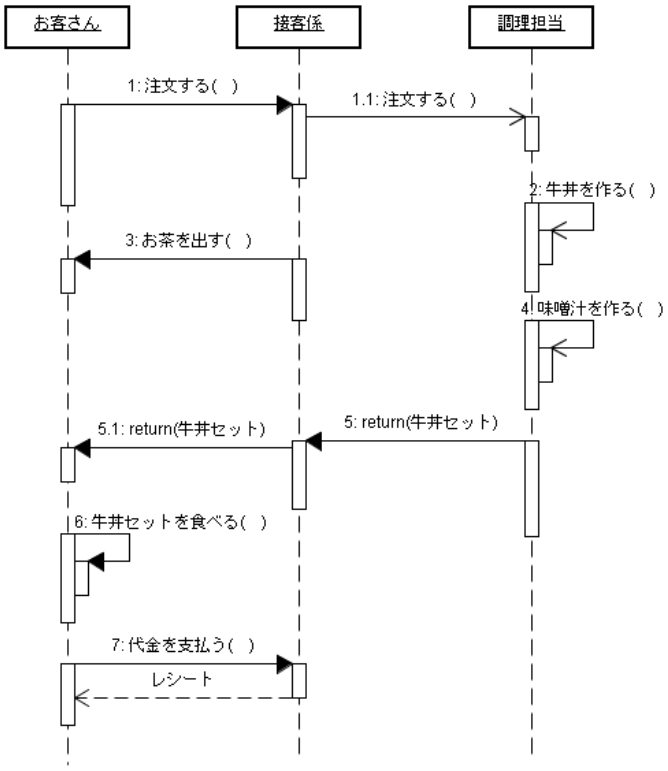
1.オブジェクトの構造を書く

■最も使われる図



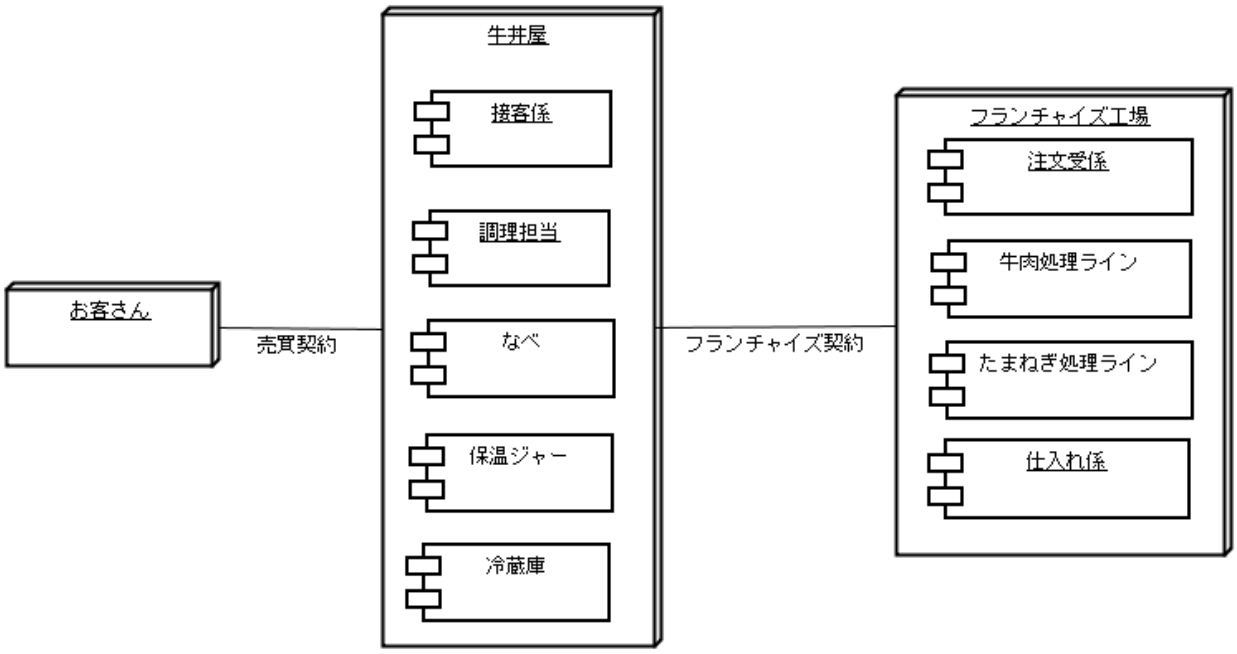
6-7 シーケンス図

1.オブジェクト間のやり取りを書く



6-8 構成図

1.システムとしての配置を書く



まとめ

1.こんなことを学びました

- ■計算機の仕組み
- ■プログラミング
- ■言語の進化
- ■現代の表記法(UML)