目次

第1章 序論

1.1. はじめに

日本における IT 技術者不足は深刻である。2002 年には 30 万人～40 万人の技術者が不足するといわれている。平成 13 年度労働経済白書[1]によると、採用予定人数全部を採用できた企業は 45%で、半数以上の企業では予定数を採用できない状況にある。

もっと深刻なのは数ではなく質の問題である。IT を駆使してサービスを提供するためには、高度な専門知識と能力が要求される。しかし、絶対数が足りないために、一般的な企業では専門ではない人材、例えば文系出身者に技術を詰め込んで現場に投入し、あとは OJT（On Job Training）によって、仕事をしながら能力を身に付けていく方法がとられている。そのような方法で技術者は一人前になるまで４～５年かかると言われている。それらの期間に半人前の技術者が現場で作ったソフトウエアを製品として完成させようとする結果、その弊害はソフトウエア納期の遅延、バグの混入、効率の欠如といった形として現れる。

そのような中、筆者はソフトウエア開発を行っている NK-EXA 社（2002 年 1 月 1 日より、EXA と社名が変更された。ただし、実際に本研究が行われたのはそれ以前であるので、本論文では NK-EXA という社名を使っている）から新入社員研修の改善の依頼を受けた。新入社員研修として 3 ヶ月、プログラミング/オブジェクト指向技術の入門教育を集合教育により実施した後、現場に投入し OJT によって技術者を育成している。しかし、新入社員研修の達成率が悪く、結局 OJT によって、長い歳月をかけて技術者を育てている。達成率が悪いのは何故か。ヒアリングすると、さらに悪いことに、

- 誤ったクラスを作ってしまい、余計にプログラムを複雑にする
- デザインパターンを誤って適用してしまい、設計を複雑にする

などの悪い癖がついてしまい、結局現場で教えているという問題があることが分かった。

筆者は、この点に注目した。

何故、誤ってオブジェクト指向技術を適用してしまうのか。

オブジェクト指向技術は非常に難しい技術である。筆者は、学部時代から 4 年間かけてオブジェクト指向技術を学んできた。その方法は、入門書や専門書を読み、実際の開発プロジェクトの経験を何度も積み重ねるというものである。その過程で何度も失敗し、試行錯誤を重ねることで、ようやく本質がつかめてきた。つまり、筆者も誤ってオブジェクト指向技術を適用しまうことを体験しているのである。では、何故誤ってオブジェクト指向技術を適用しまうのか。

筆者の達した結論は、「良い教育方法論がないからである」というものである。なぜなら、

まず、一般的な入門書にオブジェクト指向技術の本質から書かれている入門書がない。また、20 冊ほど読んだ専門書は、オブジェクト指向技術の様々な側面を用いて、本質を表現してあるものであるために、オブジェクト指向の本質を理解していないとその内容を理解するのは非常に難しいためである。

その主な例が、デザインパターンである。デザインパターンはオブジェクト指向技術の本質を基につくられている公式である。しかし、適用するのは現実問題であって、公式どおりには通用しない。高校生で使った公式のようにはいかないのである。デザインパターンを読み解くためには、オブジェクト指向の本質を理解している必要がある。そのため、入門として背景にある考え方から教育する方法論があれば、オブジェクト指向技術を効率よく学ぶことができると考えたのである。またそのような方法論があれば、オブジェクト指向技術者を効率よく育てることができる。

本論文は、オブジェクト指向技術の入門教育として、オブジェクト指向を考え方から教育するための方法論を論ずるものである。

1.2. Instructional Design

本研究は、前項で述べた目的、つまりオブジェクト指向技術を考え方から教育するための方法論を作るということを達成するために、Instructional Design の手法を適用する。Instructional Design とは教育を短期間で効率よく効果的に行う手法のことであり、歴史的には、アメリカの政府機関が、米軍における新兵教育を短期間で効率よく効果的に行う手法を求めた研究の成果である。教育コース、教材、授業を開発、実施するために情報を分析し評価していく過程である。

つまり、本論文は、オブジェクト指向技術の入門教育に関して、Instructional Design の過程を記したものだともいえる。

1.3. 論文の構成

本論文の構成は次のようになっている。

まず、第 2 章では、何故現在のオブジェクト指向技術の入門教育がうまくいかないのか、問題の原因を探る。

第 3 章では、第 2 章で挙げた問題を解決するためにどのような教授法を考えるべきか、その基本的な考え方を述べる。

第 4 章では、研究の概要として、教授法の試行錯誤の過程を具体例を用いながら述べる。

第 5 章では、本研究において試作したカリキュラムである「オブジェクト指向哲学」の概要を述べる。

第 6 章では、「オブジェクト指向哲学」の有用性について、評価と考察を行う。

最後に、第 7 章で本研究の成果をまとめる。

第2章 既存の教育の問題点

本章では、新しい教育方法論を考える最初の段階として現状のオブジェクト指向技術入門教育の問題点を分析する。

2.1. 教え方の問題

まず、一番大きな原因として教え方そのものの問題を取り上げる。一般的な入門書（詳しくは 2.2 項参照）にも見られる典型的な教え方の問題の例として、NK-EXA社でのクラス/インスタンスの概念を教える説明を取り上げる。(図 2-1)

クラスとはデータとそのデータを操作するメソッド(手続き，関数)をまとめたものである。クラスは型・枠組み(C の構造体のような物)であり，実体をインスタンスと呼ぶ。

例えば java.lang.String は クラスで，"abc" はインスタンスである。オブジェクトは“もの”で，インスタンスはオブジェクト である。Java ではクラスをオブジェクトとして扱う事ができる。

Ｊａｖａでプログラムを作るという事は class を定義するという事である。 class は C の構造体に似ている。構造体に関数を持たせ(C++では構造体に関数をメンバーとして持たせる事ができる)，アクセス制限機能があれば class に近いものになる。 class 定義は変数(データ)と関数(メソッド)を持つ型を定義することといえる。

Java のプログラムはクラス単位に行う。クラスはデータとそれを扱う手続き(メソッド)をまとめたものである。これをカプセル化という。(広義) クラス(型)からインスタンス(実体)を作り，インスタンス間で相互作用させる。これが Java(オブジェクト指向)のプログラムである。 クラスは高度なプログラムのモジュール単位ともいえる。

オブジェクト指向プログラミングで難しいものに「何をクラスにするか」という点がある。クラスをうまく作るとわかりやすく，再利用も容易なプログラムになる。クラスとする基準に次のようなものがある。

- ひとつの物として考えると都合のいいもの，わかりやすいもの
- データとそのデータ固有の手続きを一まとめにしたもの

しかし，この基準ではクラスにするしないを明確に分けられない。結局，他のプログラミング言語と同様にいいプログラムを参考にし，実際に自分でいろいろ作っていくしかないのかもしれない。ただ，一つのクラス・一つのメソッドを巨大化し，すべての処理を記述するというような事はしてはならない。

図 2-1：NK-EXA 社でのクラス/インスタンスを教える例（抜粋）

例題（図 2-1）における、問題点を列挙していく。

- クラスの概念を教えるのに、抽象的な概念を説明してしまっていること。これは具体例をあげたとしても、抽象的な概念を説明してしまった場合は同義である。何故、String はクラスで、"abc"はインスタンスなのかが分からないためである。
- クラスの概念を Java のプログラムを書くための公式として教えてしまっていること。例えば、「Ｊａｖａでプログラムを作るという事は class を定義するという事」という説明により、学習者は「Java プログラム=class」という公式として解釈してしまう。
- 「何をクラスとするか」という基準を、意味の説明なしに教えていること。意味の説明なしで教えることは、公式を教えることと同じことである。上記の例でいうと、1 つのクラスを巨大化してはならない、という基準のために、むやみにクラスを増やすという失敗をすることになる。

などが挙げられる。上記の問題は、既存の教え方が、考え方を教えずに、知識や方法論としての技術の説明に終始してしまっていることに集約されると考えられる。

この問題の弊害を明らかにするために、ガニェの学習目標の分類[2]を利用して、詳しくこの問題を掘り下げる。

ガニェによれば、学習目標は次の 5 つに分類される。

言語情報（Verbal Information）	認知領域のうちの述べることができる知識、knowing what
知的技能（Intellectual Skills）	認知領域のうちのやってみせられる知識、knowing how
認知的方略（Cognitive Strategies）	学習の方法に関する内的な知識・技能、メタ認知
態度（Attitudes）	個人の選択行動を支える内的な状態
運動技能（Motor Skills）	筋肉の運動を伴う技能

表 2-1：ガニェによる学習目標の分類

ここで、言語情報、知的技能、認知的方略の三つの分類に着目する。言語情報とは、すでに学習したことを思い出すことができる情報、いわゆる「知識」のことである。知的技能とは、学習したことを具体例へ適用する能力のことで、概念、法則、などの理解を通して、最終的に問題解決ができる能力のことである。認知的方略とは「学び方を学ぶ」ものであり、学習者内の情報処理能力のことを指す。

オブジェクト指向技術には様々な概念があり、それらは知的技能に分類される。重要なことは、この分類の種類によって教え方の種類も変える必要があることである。知的技能

の学習は、その性質上、概念や法則の定義を「知識」として知っているだけでは、能力を獲得したことにならず、その概念を具体例へ適用できる能力を教える必要がある。

ガニェの分類を利用して、問題点を説明しなおす。前に取り上げた例（図 2-1）における例えば「クラスとはデータとそのデータを操作するメソッド(手続き，関数)をまとめたものである。」という説明は、言語情報として教える方法である、と考えられる。また、公式は言語情報である。知的技能は、未知の具体例へ適用することができなければならないからである。つまり、従来の教え方における問題の本質は、知的技能として教えなければならない学習目標を言語情報として教えてしまっていることにあるといえる。

また、プログラミング言語(Java 等)や、モデリング言語(UML 等)は、その性質上、オブジェクト指向の概念と密接な関係があるため、同時に教えられていることが多い。オブジェクト指向の概念は、プログラミング言語やモデリング言語で表現されるべきであるから、同時に教える、つまり具体的に教えること自体は非常に有効であるといえる。しかし、プログラミング言語やモデリング言語は言語情報である。オブジェクト指向の概念は知的技能である。そのため、同時に教える際には、この 2 種類の性質の異なる学習目標を教えているということを意識して、教え方を変える必要がある。

さらに注意しなければならないのは、プログラミング言語の「文法」は言語情報であるが、プログラムを実際に書くこと、つまりプログラミング自体は、知的技能であることである。

また、筆者は知的技能として教えるために、認知的方略を育てる必要があると考える。知的技能は、未知の具体例へ応用できなければならないため、学習者は概念や法則について、議論することや実際にやってみることで、その問題領域について考えをめぐらせる、といった熟考する過程を踏まなければならないからである。そのような環境を作るために、実体験をさせる、議論に参加させるといった、教授法での工夫が考えられる。しかし、学習者が言語情報として認知する方略をとっている間はそのような環境を作ったとしても、学習効果が期待できないのである。特に、日本人に対してこの問題は非常に重要な問題と考える。日本では、従来から知識伝達型の授業が行われてきた。そのため、言語情報として認知する方略、つまり知識をいち早く吸収する戦略をとっている学習者は多いのである。そのため、オブジェクト指向技術のより良い教授法を考えるには、学習者の認知的方略を熟考する方略にする教育をすることが非常に重要といえる。

ここまでの議論によっていままでのオブジェクト指向教育における問題点が、学習目標の分類という面から明らかになった。既存の教え方から推測される分類と正しい分類を（表 2-2）にまとめる。

	既存の教え方から推測される分類	正しい分類
言語情報	● プログラミング言語の文法 ● モデリング言語の文法 ● オブジェクト指向の概念 ● プログラミング ● モデリング	● プログラミング言語の文法 ● モデリング言語の文法
知的技能		● オブジェクト指向の概念 ● プログラミング ● モデリング
認知的方略		● 熟考しながら、知的技能を獲得していく方略

表 2-2：オブジェクト指向技術教育における学習目標の分類

2.2. 教える内容の問題

オブジェクト指向技術を教えるのが難しい原因の一つは、オブジェクト指向技術の学習カリキュラムとして様々な学習内容が混在していることである。これらの中には、入門教育としてふさわしくないものや、オブジェクト指向と関係があるだけで含まれているものがある。では、何を学習内容とするべきなのか。その学習内容を整理するために、一般的な入門書（何を一般的とするかは非常に難しい。おおよそ「オブジェクト指向」、「UML」、「Java」、「入門」というキーワードが入っているもののことを指す。参考文献参照）から、代表的な学習内容を以下の４つの分類によって整理する。

１．オブジェクト指向の概念に関する学習内容
２．オブジェクト指向を応用した技術に関する学習内容
３．Java によるオブジェクト指向プログラミング（文法を含む）に関する学習内容
４．UML によるオブジェクト指向モデリング（文法を含む）に関する学習内容

１．オブジェクト指向の概念に関する学習内容
- オブジェクト
- クラス/インスタンス
- 抽象化
- 継承
- 多態性
- カプセル化
- メッセージ送信

２．オブジェクト指向を応用した技術に関する学習内容
- フレームワーク
- デザインパターン
- オブジェクト指向開発プロセス

３．Java によるオブジェクト指向プログラミング（文法を含む）に関する学習内容
- クラス
- メソッド
- 継承
- 配列
- インターフェイス
- マルチスレッド

- Java クラスライブラリ
- 入出力
- ネットワーキング
- イベント処理
- AWT

４．UML によるオブジェクト指向モデリング（文法を含む）に関する学習内容

- ユースケース図
- 状態遷移図
- シーケンス図
- クラス図
- コラボレーション図
- アクティビティー図
- コンポーネント図
- 配置図

このように、一言でオブジェクト指向といっても様々な内容が混在し、そして互いに密接に関係している。

また、NK-EXA社の既存のカリキュラム（表 2-3）は、これらの内容を網羅的に教える典型的なカリキュラムである。

しかし、入門教育において、これらの学習内容を網羅的に教える必要があるのだろうか。例えば、Java の文法はすべて網羅的に教える必要があるだろうか、UML の文法はどうだろうか。Java でプログラミングをしたら必ずオブジェクト指向なのだろうか。

つまり、問題は「オブジェクト指向」と呼ばれる技術の教育に関して学習内容が非常にあいまいであることである。

オブジェクト指向技術の入門教育を考えるためには、オブジェクト指向技術者にはどのような能力が必要なのか、学習目標として定められなければならない。次章でオブジェクト指向技術者の能力について、議論する。

＜１日目＞

- オブジェクト指向の概念と表現
- Java 概要
- Java 言語仕様Ⅰ（コメント、識別子、予約語、定数、変数、配列、変数の有効範囲、文字列 、演算子、制御文　など）

＜２日目＞

- Java 言語仕様Ⅱ（クラス,インスタンス、メソッド、コンストラクタ、パッケージ、スーパークラス/サブクラス、キャスト　インターフェイス、オーバーライド　など）
- Java 言語仕様Ⅲ（アクセス制御、インナークラス、例外処理　など）
- Java 言語仕様Ⅳ（並行プログラミング、マルチスレッド　など）

＜３日目＞

- I/O
- ファイル操作
- 国際化
- URL
- デザインパターン
- フレームワーク
- JDK 1.1.1 パッケージ概要

＜４日目＞

- 典型的な Usage 演習

＜５日目＞

- AWT(GUI プログラミング)

表 2-3：NK-EXA 社の既存カリキュラム

第3章 本研究における基本的考え方

本章では、第 2 章で述べられたような問題点を解決するためにはどのようにしたらよいかを議論し、本研究における基本的考え方をまとめる。

3.1. オブジェクト指向技術者に必要な能力

効果的な教授法を考えるためには、学習した結果として獲得する能力を、明確な学習目標として定義すること、つまり何を学んでほしいのかを明らかに示すことが重要である。理由は 2 つある。1 つ目は目標を明確にすることで、その目標が達成できたかどうかを判断する指針となることである。2 つ目はその目標が妥当なものであるかどうかを吟味することが可能になることである。吟味した結果、目標に到達するために有効な授業の条件を知る手がかりとなるのである。本節では、オブジェクト指向技術者に必要な能力を、明確な学習目標として定義する。

3.1.1. 知的技能としての能力

オブジェクト指向技術者に必要な能力として、まずオブジェクト指向の概念（第 2 章で概念とまとめられたものを指す）を理解していることが挙げられる。しかし、「理解する」というのは、明確な学習目標ではない。学習者の「行動で」目標を表す必要がある。なぜなら、「理解する」ということは学習者の内的な状態が何らかの形で変わったということであり、外からその変化が観察できないからである。

では、どうしたら明確になるか。第 2 章で述べたように、本研究では、オブジェクト指向の概念の理解を知的技能に分類する。ガニェによれば、「知的技能の学習とは、知的な事柄を行う方法を学習することで、その前提となる技能をもつ場合が多く、ある特定の技能を行えるようになるには、その基礎となる概念や法則をあらかじめ理解しておく必要がある。」[3]とし、知的技能を精神活動の「複雑さ」によって分類している。[3] (①が下位技能、数字が上がるにつれ上位技能になる)

① 弁別

一つ以上の物理的次元で、異なる刺激に対して学習者が区別できる能力

② 具体的概念

対象の特性や属性を確認できる能力

③ 法則（定義された概念を含む）

ある規則性をさまざまな特定の状況において適用できる能力

④ 問題解決（高次の法則）

問題に出会ったときに、それまでに学習した様々な法則をその場面に適用したり、新たに創造したりすることで問題を解決できる能力

この分類から考えた場合、オブジェクト指向の概念を理解しているということは、問題

を解決できる能力（④）にあたると考える。ここで注意するべきことは、オブジェクト指向の基礎的な概念（クラス、継承など）は定義された概念であるが、オブジェクト指向そのものの概念は定義された概念ではないということである。なぜなら、オブジェクト指向技術を使ってソフトウエアを開発できるためには、クラスや継承の概念を理解しているだけではなく、それらを応用して問題解決をしていかなければならないからである。本研究では、オブジェクト指向そのものの概念を「基本概念」、クラス、継承などの各概念を「基礎概念」、それらの総称を「概念」と区別する。このように区別した場合、オブジェクト指向技術者にとって、最終的に必要な能力はオブジェクト指向の概念を適用することによって問題解決ができる能力である。

そのための段階として、定義されている基礎概念を理解していることを考える。法則（③）における「ある規則性をさまざまな特定の状況において適用できる能力」を「概念を適切に適用できる能力」と解釈して学習目標とする。

しかし、基礎概念を適切に適用できる能力を最初のステップとして教えるのは非常に難しい。オブジェクト指向の基礎概念はそれ自体が非常に抽象的で難しいからである。学習効果を高めるために、ガニェの分類における具体的概念（②）の能力を利用して教えることを考える。つまり、まずオブジェクト指向の基礎概念を認識することから段階を踏んで教えるのである。その能力を「基礎概念の利点を説明できる能力」として学習目標とする。（詳しくは 3.3.1 項を参照）

まとめると、オブジェクト指向の概念を理解するという能力は、以下の 3 つのレベルとして定義されることになる。

レベル１（具体的概念）：オブジェクト指向の基礎概念の利点を説明できる能力

レベル２（法則）：オブジェクト指向の基礎概念を適切に適用できる能力

レベル３（問題解決）：オブジェクト指向の概念を適用して、問題解決ができる能力

3.1.2. 言語情報としての能力

本研究における焦点は、知的技能としてオブジェクト指向の概念を教えることであるが、知的技能を習得するための前提として言語情報としての能力も必要である。オブジェクト指向の概念はソフトウエアシステムに適用されるべきであるから、学習の成果は、ソフトウエアをプログラムとして記述するためのプログラミング言語、システムをモデルとして記述するためのモデリング言語などを用いて表現されなければならない。

言語情報は知的技能と密接に関係しているため、比重が偏らないようにバランスよく教えていく必要がある。ただし、本研究においては、知的技能としてのオブジェクト指向の概念、つまり前項でのべたオブジェクト指向の概念を適用して、問題解決ができる能力を獲得することを最終的な目標とするため、言語情報は、オブジェクト指向を表現できる最

低限の能力が習得できるように教えればよいと考える。技術者として活躍するためには、十分な言語情報としての知識、例えば、細かいプログラミング言語の文法の知識などを知っている必要がある。しかし、本研究の対象は新人教育である。習得するのが難しいのは知的技術であり、知的技術さえ習得すれば、言語情報としての知識はその後の自習によって十分に獲得可能であると考える。

また、本研究においては、オブジェクト指向を表現するための言語情報として、世界標準となっている、

1)プログラミング言語として「Java」（Ver1.3）

2)モデリング言語として「UML」（Ver1.3）

を選ぶ。

したがって、言語情報としての能力を次のように定める。

- プログラミング言語(Java)でオブジェクト指向の概念を表現できる最低限の能力
- モデリング言語(UML)でオブジェクト指向の概念を表現できる最低限の能力

「最低限」と言う表現は明確な学習目標としては不十分なものであるが、この段階では、「言語情報は最低限でよい」という本研究の指針を明らかにするためにこのように定義してある。（具体的に何が「最低限」なのかは第 6 章を参照）

3.1.3. 認知的方略としての能力

オブジェクト指向技術者、広くは IT 技術者にとって、技術の習得する方法そのものを学ぶことが非常に重要なことである。IT 技術は進歩が早いため、常に新しい技術を学んでいく必要があるからである。オブジェクト指向技術もそのものが現在も進歩しており、徐々に新しい技術に移り変わっていくと考えられる。最新の技術は誰かに教えてもらえるとは限らない。技術者は最終的に技術を自分で学ぶ能力が要求されるのである。つまり、技術が進歩しても対応できる技術者を育てるためには、学び方を教える必要がある。

この能力は、ガニェのいう認知的方略にあたる。ガニェによると、認知的方略は、「学び方を学ぶ」ことであり、個人の学習,記憶,思考という行動を管理する能力であるとし、具体的な例として相手の話をよく聞く，注意を集中する，質問をする，仮説をたてる，批判をする，評価をするなど挙げている。[2][3]

では、オブジェクト指向を始めとする IT 技術の学び方とは何であるか。まず、第 2 章で述べたように、言語情報ではなく、知的技能として学ぶことである。さらに、3.1.1 項で示したように、知的技能を、3 つの段階を通して最終的に問題解決ができるようになるよう学ぶことである。この学び方の特徴は、最初に利点を説明できるようになるという概念の認識から学ぶことであり、根本原理から学ぶ、つまり技術の背景にある「考え方」から学ぶ

という意味が込められている。考え方から学ぶことで、新しい技術を学ぶ際にもその考え方と既存の技術の考え方を対比して、技術を捉えることによって理解がしやすくなる。また、技術の根底にある「考え方」は人間が考えるものである限り劇的に変わるものではないからである。（これは 3.3 節で詳しく述べる）

そのため、本研究では知的技術を教える基本姿勢として、この「考え方」から教えるということを非常に重要と考える。

これまでの議論から、認知的方略として教えるべきことを次のように定める。

- 技術を考え方から学ぶことができる能力

3.2. オブジェクト指向の考え方とは

前節において、認知的方略を育てるために、技術を「考え方」から教える必要があることを述べた。本節では、本研究の対象となるオブジェクト指向という技術に関して、その考え方とは何か議論する。

「オブジェクト指向の考え方とは何か」という問いに対して、一般的な入門書では次のような基礎概念の説明に終わることが多い。

- クラス/インスタンス
- 継承
- ポリモーフィズム
- カプセル化
- メッセージ送信

しかし、これらの記述はオブジェクト指向の基礎概念を記述しているだけに過ぎないと考える。3.1.1 項で述べたように、本研究では、これらの基礎概念とオブジェクト指向そのものの概念を区別する。ここで考えるのは基本概念、つまりオブジェクト指向のそのものの根底にある考え方とは何かということである。

ここでは、「オブジェクト指向」という言葉のそのものの意味から考える。原語は「Object Oriented」である。Object とは「もの・対象」の意で、Oriented とは「指向（優先、本位、中心）の」という意味がある。対象はソフトウエアシステム（以下システムとする）であるので、本来の意味は、システムをものの集まりとして捉える考え方ということになる。オブジェクト指向の創始者の一人であるヤコブソンは「システムは、相互にやり取りする多くのオブジェクトの集まりとしてモデル化される」[4]と表現している。しかし、ただ、やみくもにシステムをオブジェクトに分解すればよいというものではなく、その作業は容易ではない。よい設計だという指針が「あるものがオブジェクトとしてモデル化されるかは、モデル作成者がそれをオブジェクトモデルとして表現したいかどうかによって決まる。」[4]という非常にあいまいなものであるからである。現状では熟練した技術者が経験と勘をたよりに、神業的な能力を発揮して、この作業を行っているが、果たしてその能力の根底にある考え方は何であろうか。何を指針としてものの集まりとして分解しているのだろうか。

指針を探るために、システムをものの集まりとして表現する必要性をソフトウエア工学、プログラミングスタイルとしての歴史からの視点から議論する。

オブジェクト指向は前身である構造化アプローチが全盛の時代であった 80 年代の後半に、構造化アプローチの弊害を解決するために生まれた手法である。ランボーは、「表面的には、"オブジェクト指向"とは、データ構造と振る舞いが一体となったオブジェクトの集まりとしてソフトウエアを組織化することである。このことはデータ構造とその振る舞いの結合関係が非常に弱い従来のプログラム言語とは対照的である。」[5]と表現している。ランボーが

表面的と表現している理由は、データ構造と振る舞いを一体とした理由に、いくつかの構造化手法の重要な弊害が隠されているからである。構造化手法の弊害を「オブジェクト指向システム開発」[6]から抜粋する。

(1)機能中心の弊害

機能を中心に分析/設計するため、作られたシステムとユーザの要求するシステムとの間にズレが生じてしまう。

(2)データと手続きの分離の弊害

データに対するアクセスの制限がないため、プログラマはデータをアクセスする際に、その構造をすべて理解する必要に迫られた。

(3)重複開発の弊害

拡張が大変であった。これは既存のコードと、拡張後のコードに論理的な関係が存在しないためである。

(4)暗黙の利用手順の弊害

同じデータを操作する複数の手続き同士が、それらが関連していることを示す手段がなく、提供者が想定する利用手順を推し量りながら、手続きを利用しなければならなかった。

これらの弊害は、大規模ソフトウエアを多数のプログラマが協調しながら、正しいソフトウエアを作る際、または機能の拡張を行う際、他人の作ったプログラムの可読性が低く、間違った使い方を防止する策がなかったことに起因するもので、オブジェクト指向はプログラムの論理性を向上させることによって、これらの問題を解決しようとする考え方であることが言える。また、ヤコブソンはオブジェクト指向の利点の中で、卓越しているものに「システムの理解：システムと現実世界との間の意味的乖離が小さいため、システムが容易に理解できること。」[4]を挙げている。現実世界、つまり人間のいる世界とシステムの意味の乖離が少なくなるということ、そこが非常に重要な点である。日本人にとって、「オブジェクト」と読んでしまうとシステムの一部というように聞こえる。しかし、「もの」と読むと、確かに人間に近いように感じられる。オブジェクト指向とは、現実世界の「もの」を中心に考えることを意味しているのである。つまり、本研究においては、**システムの「人間にとっての意味」を考えることが、オブジェクト指向の本質であると捉える。**

良いオブジェクト分割の指針を「人間にとって意味があること」とするならば、**オブジェクトとは人間にとっての「意味のカタマリ」と捉えることができる。**

まとめると、オブジェクト指向の考え方とは、システムを人間にとって見通しを良くするために、「意味のカタマリ」に分割して考える考え方であり、「意味のカタマリ」のことをオブジェクトと呼ぶ。となり、本研究におけるオブジェクト指向の考え方の定義とする。この定義に従えば、前節で最終的な学習目標として定義した「オブジェクト指向の概念を適用して、問題解決ができる能力」は、「システムを意味のカタマリに分解することによって、人間にとって見通しを良くできる能力」と言い換えることができる。

3.3. 考え方から教えるには

前節において、オブジェクト指向の考え方は、システムを「意味のカタマリ」に分解して考えることであると述べた。では、それをどのように教えていったらよいか。本節では、考え方から教えるための基本的な考え方を議論する。

考え方から教えるのは難しい。システムを「意味のカタマリ」に分解すること、これは一つの考え方でしかない。もちろん世の中には様々な考え方がある。へたに説明をすると、一部の学習者は「押し付けられている」という感覚を受けてしまう。では考え方から教えるにはどうしたらよいか。何故「意味のカタマリ」を作るのか、その必然性を納得させる必要があると考える。そのため、ソフトウエア工学の歴史的背景から、何故必要なのかを実体験する必要がある。つまり、構造化プログラミングの問題点を考えていった結果として、この「意味のカタマリ」に分解するという考え方の必然性が分かるような環境を作らなくてはならないのである。そのため、本研究ではオブジェクト指向を考え方から教えるためには、**構造化プログラミング以前のプログラミングスタイルから、歴史の流れに沿って教えるべき**であると考える。

この教え方にはもう一つの重要な視点がある。「意味のカタマリ」を考えるというアプローチは、構造化プログラミングを学ぶ上でも非常に重要な考え方であるということである。構造化プログラミングは、GOTO 文が氾濫することなどの理由から、プログラムの意味が分散されてしまい、可読性が落ちてしまうという問題点から、「意味のカタマリ」として構造化するべきだという結果として生まれた考え方と考えることができる。配列などのデータ構造も「意味のカタマリ」を作るための考え方と考えられる。つまり、**「意味のカタマリ」を作るという考え方は、技術が変化しても変わらない根本的な考え方なのである。**従来の教え方の問題として、本質的にオブジェクト指向技術は従来の技術の延長線であるにもかかわらず、新しい技術として教えてしまっていることに問題があるともいえる。そのため本研究では、オブジェクト指向を新しい技術としてではなく、従来の技術の延長線として考えるという視点から、教える方法を考える。

第4章 研究概要　―教授法の模索―

本研究の目的は、第 2 章で述べられたような既存のオブジェクト指向技術教育の問題点を解決するために、第 3 章で述べられた考え方をもとにした新しい教授法を提案することである。

その方法は、まず基本的な考え方に従って教授法の試作をし、実験を試みたうえでその成果を考察し、改良していくというものである。本章では、その試行錯誤の概要を具体例を交えながら述べる。

4.1. 考え方を教える教育

4.1.1.　考えさせる授業を行うには

第 2 章、第 3 章で述べたように、考え方を学ぶ授業を構成するときに重要なことは、学習者がその問題領域について熟考した結果として、その考え方の必要性が導き出されるという仕組みを作ることである。ここでは、特に学習者が熟考しなければならないというところに注目する。学習者が熟考する環境を作るのは非常に難しい課題である。

本教授法では、「議論する」いう方法で、熟慮できる環境を作ることを考える。まず学習者に具体的な失敗の体験をさせる。そしてその失敗が何故起こったか、成功例と比較しながら議論させることで問題点を明らかにする。具体的な問題に失敗することで、学習者の動機付けがされ、活発な議論がしやすくなる。問題点を明らかにしていく段階で、「意味のカタマリ」という考え方のヒントを出しながら、この考え方を導入することによって解決できるという方向に議論を進める。

しかし、カリキュラムの始めから「意味のカタマリ」という考え方の議論をさせようとすると、学習者は困惑してしまい、いくらヒントを出しても「意味のカタマリ」で解決するという方向に議論が進まない。それは何故か。原因は、初心者プログラマはおろか、実際に業務を行っている中級者プログラマでも、まず「プログラムの意味を考える」という習慣がないことであった。第 3 章で述べたように、ここでいう「意味」とは、人間から見たプログラムの意味のことである。初心者プログラマは、プログラムを動かすということに集中してしまう結果、人間から見たプログラムの意味を考える習慣がなくなってしまうのである。したがって、「意味のカタマリ」の議論をさせる際には、前段階として学習者がプログラムの意味について考えることができる必要がある。

4.1.2. 具体例①:コメントの授業でプログラムの意味を考えさせる

前項で議論したように、学習者に「意味のカタマリ」の議論をさせるためには、プログラムの意味を考えられること、つまり、プログラムには人間から見た意味があり、その意味を考える必要があるのだということ教える必要がある。そのために、まずプログラムの意味について考えさせる授業を試みた。

題材として、プログラムにコメントをつける（リスト 4-1）というものを選んだ。この授業は、プログラムにコメントをつけることによって、プログラムの意味を明らかにする、という作業を通して、プログラムの意味について教える授業である。

授業の仕組みを説明する。まず、プログラムの意味が明らかになるためには、プログラムにつけるコメントには、プログラムの意味が書かれなければならない。（リスト 4-2）しかし、プログラムの意味を考えることができない学習者は、典型的な間違いとして、プログラムの解説を書いてしまう。（リスト 4-3）

```
public class Main1_prac2{
  public static void main( String args[] ){
    int weight = 55;
    System.out.println( “体重は” + weight + “ｋｇです。”);
    int height = 168;
    System.out.println( “身長は” + height + “ｃｍです。”);
    int rohrer = weight * 10000000 / height / height / height;
    System.out.println( “ローレル指数は” + rohrer + “です。”);
    if( rohrer < 100 ){
      System.out.println(“やせすぎですね。”);
    }
    if( rohrer >= 100 && rohrer < 115 ){
      System.out.println(“やせていますね。”);
    }
    if( rohrer >= 115 && rohrer < 145 ){
      System.out.println(“普通ですね。”);
    }
    if( rohrer >= 145 && rohrer < 160 ){
      System.out.println(“太っていますね。”);
    }
    if( rohrer >= 160 ){
      System.out.println(“太りすぎですね。”);
    }
  }
}
```

リスト 4-1：次のプログラムに適切なコメントをつけよ

ここから、このプログラムにどのような意味があるのか、失敗したコメントを基に成功したコメントと比較して、議論していく。例えば、「System.out.println(“体重は” +weight + “ｋｇです。”);」という行の「体重の表示」というコメントについて、「体重」は意味で

あるが、「表示」はプログラムの解説である。そこで、何故「体重の表示をしているのか」を議論する。もう一つの例として、後半部分「ローレル指数により出力するメッセージを変更する」というコメントについて、このメッセージの意味とは何か議論する。

最後に、このプログラム全体としてどのような意味があるのか、何故「ローレル指数を求める」必要があるのか議論する。このような授業をすることで、学習者にプログラムの意味を考えさせることができる。

```
/**
*  健康を管理するプログラム
*/
public class Main1_prac2{

  /**
  * ローレル指数を求めるプログラム
  */
  public static void main( String args[] ){

    //体重を保存し、確認する
    int weight = 55;
    System.out.println( “体重は” + weight + “ｋｇです。”);

    //身長を保存し、確認する
    int height = 168;
    System.out.println( “身長は” + height + “ｃｍです。”);

    //ローレル指数を求める
    int rohrer = weight * 10000000 / height / height / height;
    System.out.println( “ローレル指数は” + rohrer + “です。”);

    //ローレル指数から肥満に対するコメントを提示する
    if( rohrer < 100 ){
      System.out.println(“やせすぎですね。”);
    }
    if( rohrer >= 100 && rohrer < 115 ){
      System.out.println(“やせていますね。”);
    }
    if( rohrer >= 115 && rohrer < 145 ){
      System.out.println(“普通ですね。”);
    }
    if( rohrer >= 145 && rohrer < 160 ){
      System.out.println(“太っていますね。”);
    }
    if( rohrer >= 160 ){
      System.out.println(“太りすぎですね。”);
    }
  }
}
```

リスト 4-2：コメントにプログラムの意味がつけられたプログラム

```
/**
*  ローレル指数を求めるプログラム
*/
public class Main1_prac2{

  /**
  * メイン
  */
  public static void main( String args[] ){

    int weight = 55;//体重変数セット
    //体重の表示
    System.out.println( “体重は” + weight + “ｋｇです。”);

    int height = 168;//身長変数セット
    //身長の表示
    System.out.println( “身長は” + height + “ｃｍです。”);

    //ローレル指数のセット
    int rohrer = weight * 10000000 / height / height / height;
    //ローレル指数の表示
    System.out.println( “ローレル指数は” + rohrer + “です。”);

    //ローレル指数により出力するメッセージを変更する
    if( rohrer < 100 ){
      System.out.println(“やせすぎですね。”);//ローレル指数 100 未満
    }
    if( rohrer >= 100 && rohrer < 115 ){
      System.out.println(“やせていますね。”);//ローレル指数 100 以上 115 未満
    }
    if( rohrer >= 115 && rohrer < 145 ){
      System.out.println(“普通ですね。”);//ローレル指数 115 以上 145 未満
    }
    if( rohrer >= 145 && rohrer < 160 ){
      System.out.println(“太っていますね。”);//ローレル指数 145 以上 160 未満
    }
    if( rohrer >= 160 ){
      System.out.println(“太りすぎですね。”);//ローレル指数 160 以上
    }
  }
}
```

リスト 4-3：コメントにプログラムの解説が書かれたプログラム

しかし、この授業を進めるときに、講師が解答例を解説してしまうと、学習者は、「プログラムにはプログラムの意味を書かなければならない」という言語情報として認知してしまう学習者が出てくる。その場合、学習者はプログラムの意味を考える能力がつかない。それはたとえ失敗と成功の例を比較して解説してもうまくいかない。学習者がプログラムの意味を熟考した結果、学習者自らがプログラムの意味を考える必要があるという結論に

達しなければならないのである。また、実際に失敗の体験をしなくてもうまくいかない。プログラムの意味について熟考することの動機付けをする必要があるのである。普段学習者がプログラムを書くときには、そのプログラムは自分で書いたものであるから、その意味はコメントとして書くまでもなく、暗黙知として分かっている。意味を考えなければならない動機がないのである。そのためには人の書いたプログラムの意味が分からないという実体験をする必要がある。さらなる動機付けのために、簡単な拡張を行わせるような問題を考えるとさらに効果的であると思われる。

4.1.3.　具体例②：

メソッドの授業で「意味のカタマリ」について考えさせる

次に、「意味のカタマリ」を考えさせる授業の例として、メソッドの授業から、「次の重複行をメソッド化すべきか」という例題（リスト 4-4）を取り上げる。

```
int x;

…省略…
System.out.println( “体重は” + weight + “ｋｇです。”);

…省略…
System.out.println( “体重は” + weight + “ｋｇです。”);

…省略…
System.out.println( “体重は” + weight + “ｋｇです。”);
```

リスト 4-4：次の重複行をメソッド化すべきか

この 3 行の重複行をメソッド化するべきか。この問題とりあげて、学習者に議論させたところ、「意味のカタマリ」を考えさせる上で興味深い議論が展開された。ここでは、議論の経過を具体的に述べる。

議論の中で、意見が 2 つに分かれた。

＜メソッド化しないという意見＞

- （メソッド化の対象が）1 行ずつだから、メソッド化しない
- メソッド化すると、余計コードが長く、複雑になるのでメソッド化しない
- もう少し重複行が多くなったらメソッド化する。3 行ではメソッド化しない

＜メソッド化するという意見＞

- （メソッド化の対象が）1 行ずつでも、重複行はまとめるべきなのでメソッド化する
- 変更するときに、(3 箇所を変えるのが）大変だからメソッド化する

メソッド化するという理由が「変更するときに大変だ」という意見から、「何故、3 箇所を 1 度に変える必要があるのか」という質問をしたところ、「もし、出力を英語に変えたくなったとき、3 箇所を変えるのが大変だ」という具体例が出た。その時に、メソッド化しないという意見から、かなりの人数がメソッド化するという意見に変更した。この意見は、メソッド化することの利点を具体的に説明した意見であり、この意見によって、プログラムをカタマリにすることの利点を授業で教えることができた。

ただし、ここで本当に教えたいのは「プログラムのカタマリ」ではなく、「意味のカタマ

リ」にすることである。この例題（リスト 4-4）では、「メソッド化するかどうかはプログラムの意味による」というのが「意味のカタマリ」という考え方から正答である。そこで、ヒントとして次の例題（リスト 4-5）を用意した。

```
int x;

…省略…
System.out.println( “体重は” + weight + “ｋｇです。”);//ユーザへ入力確認をする

…省略…
System.out.println( “体重は” + weight + “ｋｇです。” );//ユーザへ入力確認をする

…省略…
System.out.println( “体重は” + weight + “ｋｇです。”);//デバッグ出力
```

リスト 4-5：次の重複行をメソッド化すべきか（コメントつき）

この例題を見せると、「ユーザへ入力確認をするコードだけ英語化したい場合がある」という理由から、ほとんどの学習者が「意味ごとにまとめるべきだ」という意見になり、その後「プログラムの意味を考えることは重要だ」という議論をすることができた。

この授業では、ほとんどの学習者にプログラムの意味に関する重要性を熟考させることができたと考える。しかし、本頁の例題（リスト 4-5）から議論を始めると、意味ごとにまとめるという意見が学習者たちの間であたりまえのこととして議論されてしまうため、熟考させることができない。その結果、テストでは、「メソッドをどのように分割するか」という問いに対して、「意味ごとに分割する」と答えられるものの、前頁の例題（リスト 4-4）のような問題は解けないという結果になる。言語情報として習得されてしまっているのである。このことから、前頁の例題（リスト 4-4）を議論すること大きな意味をもつといえる。

ではなぜ、前頁の例題（リスト 4-4）を議論することで授業はうまくいくのか。本頁の例題（リスト 4-5）から議論した場合との違いは、学習者が熟考できるかできないかという点である。このことを、前項で述べたコメントの授業と照らし合わせて考える。コメントの授業における、学習者が熟考するために必要な要素は、具体的な失敗の経験であった。つまり、前頁の例題（リスト 4-4）の議論をすることは、具体的な失敗の経験に近いものだと考えられる。前頁の例題（リスト 4-4）の正答は、「メソッド化するかどうかはプログラムの意味による」という、初心者にとっては不親切な問題である。しかし、不親切な問題にあえて取り組ませることで、具体的な失敗の経験に近い経験をさせることができる。その経験をさせることが、考えさせる授業を行う際に非常に有効なのである。

また、このような授業を効果的に行うために次のような課題があることが分かった。

(1)1 クラスの人数

このような授業は、プログラムの意味、カタマリについての深い議論が行われる必要があるため、授業に参加する学習者の適切な人数について考える必要がある。実験の経験から適切な人数を考察すると。10 人程度が一番議論しやすく、全員を議論に参加させるための限界は 20 人程度である。

(2)学習者の認知的方略

このような授業を成功させるためには、学習者を積極的に議論に参加させ、熟考させることが必要条件である。しかし、実際には学習者を積極的に議論に参加させるのは難しく、議論に消極的な学習者が多い。消極的になってしまう学習者が多い理由として、学習者が熟考し、議論をする方略になれていないことが考えられる。第 2 章で述べたように、日本では初等教育から、知識伝達型の教育方法がとられることが多い。そのため、多くの学習者は、知識を受動的に受け取る学習方略をとっている。言語情報として知識を吸収する能力は高いのだが、熟考して自分なりに考え方を捉え、吸収するということが苦手である。そのため、このような授業を成功させるためには、学習者の認知的方略も徐々に学習させる必要がある。また、カリキュラムの最初の段階から、このような授業を行い徐々にこのスタイルに慣れてもらうことでこの問題は改善していく傾向が見られた。

(3)教師の役割

このスタイルの授業での教師の役割は、学習者の意見を引き出し、議論をまとめていくことである。授業がうまくいくかどうかは、議論の質に大きく影響されるため、教師の教授能力が非常に大きな意味をもつ。では教師をどのように育てたらよいか。大きな方針として次の 2 つの方法が考えられる。

1) 教授能力を持っている、いわば教育のプロにオブジェクト指向の考え方を教えて、教師とする方法

2) オブジェクト指向を考え方から熟知している、いわば現場のプロに教授法を教えて、教師とする方法

この授業では、オブジェクト指向の考え方に関する非常に深い議論をする必要がある。そのため、最低でも 3.1.1 項で定義したレベル 3 の能力がないと、学習者の意見の意図を汲み取り、議論を発展させるのは難しい。つまり、オブジェクト指向の考え方を熟知しているのは必須の能力だといえる。だから、1)の方法で、いわば「オブジェクト指向の考え方マニュアル」を作ったとしても授業を成功させるのは難しい。逆に、議論のさせ方の問題は、重要なポイントを教えることで、いわば「議論のさせ方マニュアル」を作ることによって解決できる。そのため、2)の方法のほうが、教師を育てる方法として適していると考えられる。

4.2. 基礎概念を教える教育

4.2.1. 知的技能をどのように教えるか

オブジェクト指向の基礎概念を知的技能として獲得できる環境とはどのようなものだろうか。学習者にとって、概念が一番良く分かるときはどんなときだろうか。例として、「継承」という概念を教える過程から分析する。

講師がまず、「継承とは概念を抽象化すること」だと説明したとする。そのような教え方では、学習者の頭で「抽象化」という言葉のイメージしかなく、言語情報として「継承＝抽象化」と公式化してしまうため効果がない。次に講師が人間と哺乳類の関係を例に説明したとする。具体的に教えることにより、その段階では学習者の頭の中に人間と哺乳類のイメージができ、概念の輪郭が見えてくる。具体的な例を使うのは概念教育の基本である。

そして、講師が Java で継承したクラスの書き方を説明して、学習者に演習を行わせたとする。この作業により、頭の中の具体的なイメージが、現実に動くプログラムと結合してさらに概念の理解が深まる。特にソフトウエア工学の概念の場合、頭の中では具体的なイメージがつかみにくいため、実際にやってみることが非常に重要である。しかし、実際にやってみることには弊害もある。学習者が理解しているという錯覚を起こしてしまうことである。「継承」は使えるようになる。しかしそのような段階では、間違った使い方をしてしまう例が多く見られる。

何が問題なのか。3.1.1 項で定義した学習目標の 3 つのレベルから考察すると、レベル２「基礎概念を適切に適用できる能力」がない状態であるといえる。ここで、知的技能は前提条件となる知的技能ができて次の段階に進むことができるというところに注目する。

では、レベル１「基礎概念の利点を説明できる能力」はあるだろうか。学習者の間違い方の特徴を考察すると、ほとんどが継承を適用すべきでないところに適用してしまう例で、適用すべきところに適用していないという例は少ない。つまり、習ったからとりあえず適用するという学習者が続出すると考えられる。このことから、利点や欠点を説明する能力がないといえる。またこのことはオブジェクト指向の基礎概念を教える際にほとんど共通の現象である。そのため、実際にやってみるという教え方は効果的であるが、まずレベル１ができるようになること学習目標とすることが重要である。まず「継承」という概念を認識させることが必要なのである。本教授法において、レベル１を定義した理由はそこにある。

そのため、「継承」を何故使うのかが分かるように講義、演習を行う必要がある。必要な

のは過去の失敗した経験、この場合は「継承」を使わなかった場合の経験である。比較対象がないと、利点が分からないからである。失敗した経験と成功した経験を照らし合わせた結果として、何故「継承」という概念が必要なのかが分かるのである。この経験から、レベル１、つまり概念を使う必要性を教え、利点欠点を説明できるようになることが、非常に重要といえる。ここまでの考察から、知的技能を教えるときに重要なことを次のようにまとめる。

- 具体的なイメージを頭の中で作らせること
- 実際に体験をさせることにより、頭のイメージと結合させること
- 失敗の経験と成功の経験から、概念を使う必要性を教えること

4.2.2. 具体例③：「クラス/インスタンス」の概念を教える

まず、オブジェクト指向の主要概念であるクラス/インスタンスの概念をどのように教えるかを考察する。本教授法では、クラス/インスタンスの概念を教えるために、具体的な例を挙げて議論する。そのため、クラス/インスタンスの概念の導入の前に、クラス/インスタンスの概念を使わなければ大変だが、クラス/インスタンスの概念を使えば、プログラムの意味がわかりやすくなるというプログラムを、その概念を教えない段階で一度書かせる。この一度失敗したという経験が概念の導入での非常に重要となる。

逆に、失敗のさせ方が弱いと、概念を教えるのは非常に難しくなる。最初に考えたカリキュラムの構成（表 4-1；左）においては、メソッドと配列を利用して、それを「意味のカタマリ」として結合できるという教え方を実験した。クラス/インスタンスの概念を利用することの本質は、データ構造とアルゴリズムを「意味のカタマリ」として結合することだからである。しかし、初期の小さいプログラムは、配列とそれを操作するアルゴリズムのみであり、同じファイルに書かれているため、クラスとして結合する利点を実体験させることができなかった。

またその段階で、学習者が、アルゴリズムとデータ構造を 1 種類しか知らないことも問題であった。1 種類しか知らないのであれば、結合する利点を実感するのは非常に難しいのである。だから、アルゴリズムとデータ構造の結合という利点を教えるためには、少なくとも、データ構造とアルゴリズムを 2 種類以上知っている必要があるのではないかと考えた。そのため、改良したカリキュラム（表 4-1；右）では、連結リストを先に教え、アルゴリズムとデータ構造を 2 種類知っている段階で、その組を「意味のカタマリ」にするべきだという議論を行ったところ、概念の理解が比較的スムーズに行われた。

	最初に考えたカリキュラム	改良したカリキュラム
第1回	コメントをつける	コメントをつける
第2回	配列	配列の利用
第3回	メソッド	メソッド
第4回	クラス/インスタンス	構造体としてのクラス
第5回	継承	連結
第6回	連結リスト	クラス/インスタンス

表 4-1：カリキュラムの改良

また、知的技能を教えるためには、基礎的な技能（下位目標となる前提技能）をひとつずつ習得させていくのが有効であるとされている。そのため、改良したカリキュラム（表

4-1；右）では、クラス/インスタンスの概念の本質を教える前に、まず、歴史的な経緯である構造体としてクラスを教える方針をとった。これは、「意味のカタマリ」としてのクラスの導入が分かりやすい例であるために、非常にスムーズに理解が進む結果となった。

また、クラスに関する重要な概念としてカプセル化がある。今回の教授法では、カプセル化を教えるのが非常に難しい事が分かった。カプセル化の利点は、大規模システムになったときに、または人が書いたプログラムを使ってプログラムを書くときに、既存のプログラムの意味を誤って解釈してしまうことを防ぐことにある。そのため、入門教育における小規模のプログラムにおいては、カプセル化をしなかったことによって失敗させる、という経験をさせるのが非常に難しいのである。

小規模なプログラムでもカプセル化しないことによって失敗する例を考えたが、もっともな例を作るのが非常に難しかった。学習者にも故意だと分かってしまうような例題を使ってしまうと、カプセル化することの利点を考えさせることができなくなってしまう。そのように教えてしまった結果として、誤ったカプセル化をする学習者が続出するというものになる。誤った理解をしてしまうくらいなら、入門教育では教えないほうが良いと思われる。

4.2.3. 具体例④：「継承」の概念を教える

オブジェクト指向の概念の中で、教えるのが難しいと考えられる継承の概念をどのように教えるべきか考察する。何故教えるのが難しいのか。それは、初心者に継承の概念を中途半端に教えた場合、誤解して適用してしまう例が多いからである。典型的な誤解の例として、次のような例を挙げる。(図 4-1)

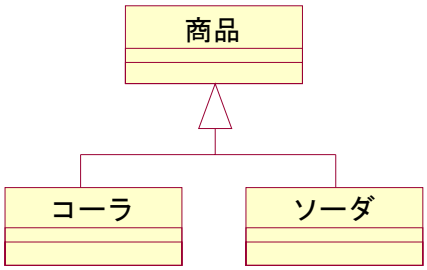

図 4-1：適切でない継承の適用例

上記の例は、自動販売機システムを構築するという問題で、その自動販売機が取り扱う商品のプログラムを書くときに、継承を教えた学習者がまず始めに考えるプログラムのモデル図である。この例は、一見すると継承の適用例として自然に見える。しかし、実際にプログラムを書くときには、このような設計はしない。なぜなら、このような設計では、新しい商品を追加するたびに新しいクラスを追加しなければならない、などの拡張性がないばかりか、プログラムが複雑化する原因となる。適切な設計は商品の種類をクラスにする方法である。(図 4-2)

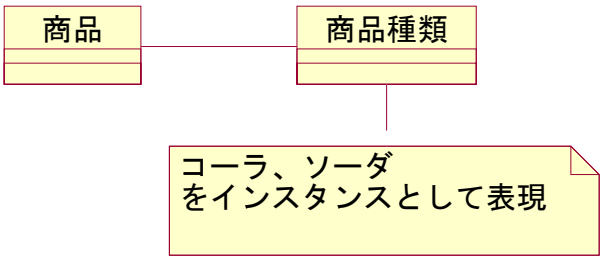

図 4-2：適切な設計

もちろん、この方法は継承の前の段階として教えている。つまり、基本的な方法で十分解決できる問題に、無理矢理継承を適用してしまった結果、余計にプログラムが複雑になってしまうのである。この現象は継承を習いたての学習者よく見られる。そして、このよ

うな適切でない適用によって、余計にプログラムが複雑化してしまった結果、学習者に、継承を適用すると余計にプログラムが難しくなる、という誤解を抱かせることになる。このような問題の原因は、教え方が適切でないからである。

例えば、一般的な参考書の説明に使われる次のような例を考える。

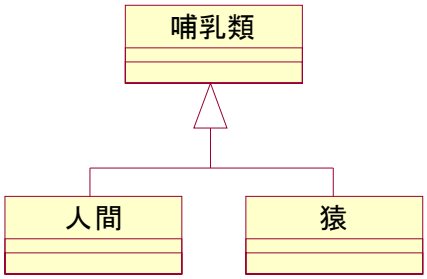

図 4-3：一般的な説明に使われる題材

確かに、このような題材を使えば、抽象クラスと具象クラスの関係は良く分かる。しかし、この題材は実際のプログラムに適用できない場合がある。できるかできないかはコンテクストで決まる。オブジェクト指向における継承という概念は、抽象化するという概念を応用した概念なのである。そのため、抽象化の例として正しい例を説明する教え方をした場合、学習者は先に述べた自動販売機における商品の例のような適用をしてしまうことになる。そのような誤りをしないように、継承を使う利点を具体的なプログラムによって明らかにする必要がある。

また別の教え方として、アプレットの原理などから、継承を使う利点を「簡単に拡張できる」として教える方法もある。しかし、簡単に拡張できるというのは、継承の利点の一側面に過ぎない。そのような教え方をすると、学習者は次のような間違いをする。

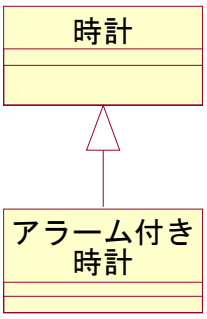

図 4-4：適切でない拡張例

コンテクストにもよるので一概に適切ではないとはいいきれないが、アラーム付きのデジタル時計に拡張したいときなど、この例が適切でない可能性は高い。初心者が安易に継承を使ってしまうのは、何故継承することによって拡張できるのかが分からないからである。これらのことを学ぶためには、専門書を読む必要がある。そのため、入門教育として、拡張性を主な利点として教えるのはふさわしくないと考える。

そのため、そこで、本教授法では図のような例を利用し、継承の概念を利用しなかった場合を体験させた上で、このプログラムをクライアント（Main）から利用したときに得られる利点を、議論するようにした。

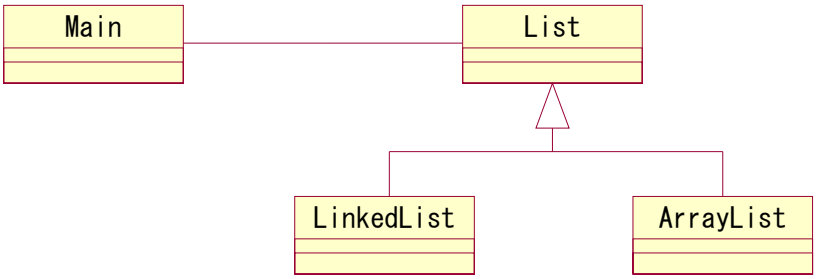

図 4-5：本教授法で利用する題材

この例を用いると、Main でかかれていた if 文と重複コードがなくなるため、（詳しくは付録を参照）具体的に継承の利点を教えることができる。また、間違った適用例を取り上げて議論することも効果があると考える。ただし、何故間違っているのかを議論するのは非常に難しいと考える。

入門教育として、継承をどこまで教えるべきかというのは難しい問題である。継承を使ったプログラム設計ができるようになるには、相当の経験が必要だと考える。入門教育としては、考え方を理解して、継承を使って作られたプログラムを理解できるようになることがまず必要だと考えられる。その点において、このような例は、Java の API における構造と同一なため、適用しやすいと考えられる。

第5章 「オブジェクト指向哲学」カリキュラム

本研究では、第 4 章で述べられた試行錯誤の過程を通して導かれた教授法を「オブジェクト指向哲学」という試作カリキュラムの形にまとめ、提案する。

本章では、「オブジェクト指向哲学」カリキュラムについて、その概要を述べる。

5.1. 「オブジェクト指向哲学」とは

本カリキュラムは、オブジェクト指向技術者の入門教育を効果的、効率的に行うことを目的としている。

主な特徴として、オブジェクト指向の基礎概念を言語情報としてではなく、知的技能として教えること、また、技術者にとって必要な認知的方略を育てるために、オブジェクト指向技術をその考え方から教えるということが挙げられる。

本カリキュラムのタイトル「哲学」とは、オブジェクト指向技術を考え方から教える、つまり、オブジェクト指向技術の根本原理を追求するカリキュラムであるという意味が込められている。

5.1.1. 到達目標

第 3 章で定義したオブジェクト指向技術者に必要な能力を獲得することを本カリキュラムの到達目標とする。つまり、

知的技能として

- レベル１：オブジェクト指向の基礎概念の利点を説明できる
- レベル２：オブジェクト指向の基礎概念を適切に適用できる
- レベル３：オブジェクト指向の概念を適用して、問題解決ができる

言語情報として

- プログラミング言語(Java)でオブジェクト指向の概念を表現できる
- モデリング言語(UML)でオブジェクト指向の概念を表現できる

認知的方略として

- 技術を考え方から学ぶことができる

このうち知的技能としての能力について、本カリキュラムでは、入門教育としてレベル１～２の能力を獲得させることを主眼とし、特にレベル１の能力を獲得することを重視する。

5.1.2. 対象学習者

本カリキュラムは、企業における新入社員教育の一環として使われることを想定して作られたものである。つまり、対象として「将来オブジェクト指向技術者になる者」を想定している。

ただし、本カリキュラムは、オブジェクト指向技術を、プログラミングを通して教えるものであるため、基本的なプログラミングができることが前提条件となる。
ただし「基本的なプログラミングができること」とは次のようなものとする。

- 50 行～100 行程度のプログラムを書き、動かした経験があること
- 制御構造（連接、繰り返し、条件分岐）を理解していること、制御構造を使ったプログラムを書けること
- 配列の仕組みを理解していること、配列を使ったプログラムを書けること
- 関数の仕組みを理解していること、関数を使ったプログラムを書けること

なお、上記の経験においてプログラミング言語は問わない。

5.1.3. カリキュラムの構成

本カリキュラムは、２部構成となっている。概要を示す。

(1) プログラミング編（全 10 回）

オブジェクト指向の考え方、基礎概念の初歩を、プログラミングをしながら学ぶカリキュラムである。オブジェクト指向を表現する、つまりオブジェクト指向プログラミングをするための最低限必要な Java の文法が含まれる。

(2) UML編（全 5 回）

UML を使って、簡単なモデリングを行うことで、オブジェクト指向の考え方、基礎概念の理解を深めるとともに、最低限の必要な UML の知識を獲得するためのカリキュラムである。このカリキュラムは、プログラミング編で得る能力が前提条件となるため、プログラミング編を先に学習している必要がある。

5.1.4. テーマ

本カリキュラムは「自動販売機」システムを作ることを全体のテーマとし、システムを一から作り上げていく内容となっている。

テーマに自動販売機を選んだ理由を挙げる。

- 単純なシステムであること。
- 一般的であり、誰でも一度は見たことがあること。
- 具体的であり、「もの」のイメージがしやすいこと。
- 様々なデータ構造や、状態遷移といった学習するための基本要素が含まれていること。

自動販売機の完成イメージ（図 5-1）と、本カリキュラムで設計されるモデル（図 5-2）を下に示す。

図 5-1：自動販売機の完成プログラム

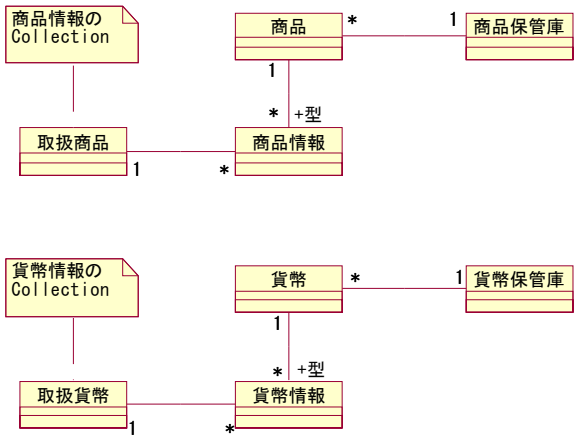

図 5-2：本カリキュラムで設計される自動販売機のモデル

5.1.5. 教育環境

本カリキュラムは、教師が教える対面教育の形態をとり、講義－演習形式で進められる。

１教室の学習者の人数は、学習者の質によるが、講義では、議論に参加することが前提となるため、20 名以上での実施は難しい。

演習では、プログラミング初心者が多い場合、本質的な要素以外での様々なトラブルを回避するために学習者 5 名～10 名に一人程度の TA が必要である。

一人一台の開発環境は必須となる。本カリキュラムにおいて、「実体験」させることが大きな意味をもつからである。

5.1.6. 教材

本カリキュラムは以下の教材から構成される。

- 詳細到達目標

本カリキュラムで学ぶべき内容を、具体的な到達目標として列挙したものである。

- 教師用プレゼンテーションスライド

教師が教えるために使うスライドを、Microsoft Power Point 形式のファイルとして記述したものである。教える内容がすべて記述されている。

- 学習者用テキスト

学習者が授業を受ける際に利用するテキストである。教科書としての位置付けではなく、スライドではサポートできないソースコードの全リストを参照したり、考えた結果を書き込んでいくために利用する。

なお、UML 編は学習者用テキストがまだ実装されていない。

- 練習問題（解答を含む）

学習者が、各回の内容の理解を深めるために使う練習問題である。

5.2. カリキュラムの特徴

5.2.1. 歴史に沿って教える

本カリキュラムの大きな特徴は、ソフトウエア工学の歴史に沿って、まず、徹底した構造化プログラミングの考え方を教え、その延長線としてオブジェクト指向の概念を教えることである。最初はプログラムの意味を考えることを、コメントの書法を通して教える。その後、配列、メソッド分割を議論することにより、構造化プログラミングをする能力を養う。この学習を通して、「意味のカタマリ」についての考え方が徐々に深まっていく。そしてその後に、「意味のカタマリ」の議論を発展させていった結果、自然にオブジェクト指向の考え方になるように誘導する。この教授法によって、オブジェクト指向で考える意義を教えることができ、その基本はプログラムの意味を考え、それをカタマリにしていくことだという考え方を育てることができる。

5.2.2. 考え方から教える

本カリキュラムでは、考え方から教えるために、プログラムの意味についての徹底した議論をおこなう。プログラムの意味について議論するべき課題を具体的に提示し、議論することで、学習者を熟考させ、考え方から学ぶという認知的方略を育てる。

5.2.3. 一貫したテーマで教える

一貫したテーマで最後に一つの作品を作り上げることにより、達成感を味わうことができ、学習の意欲を高めることができる。

5.2.4. 利点を体感させる

本カリキュラムでは、学習の対象となる概念の利点を体感してもらうことによって、その概念の必要性を学び、概念の理解を深めるという方針が貫かれている。

概念の利点を体感してもらうためには、新しい概念を導入する際に、それを使わなかった場合との比較を議論して、利点を明らかにしていくことが必要である。そのため、本カリキュラムでは、「商品を管理する」プログラム（「取扱商品」と「商品情報」のコレクション部分のこと。モデルを（図 5-3）に示す。）の機能を変えずに、新しい概念を使って書き直すという課題が与えられ、書き直したプログラムと元のプログラムの比較をする。機能を変えずに、プログラムだけを書き換えることによって、比較が容易にでき、利点が浮きぼりになる。

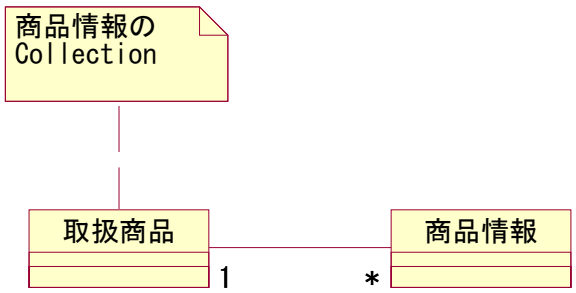

図 5-3：「商品を管理する」プログラムのモデル

つまり、過去の「失敗の経験」と、新しい概念を結びつけることによって、概念の理解を深める教授法であるといえる。そのため、故意に「一度失敗させる」ということが必要である。そのために、最初はオブジェクト指向でないプログラムから、徐々に概念を導入する。このことにより、その概念の利点を明らかにすることができる。

5.3. プログラミング編カリキュラム

5.3.1. 概要

オブジェクト指向の考え方、基礎概念の初歩を、プログラミングをしながら学ぶカリキュラムである。オブジェクト指向プログラミングをするための最低限必要な Java の文法が含まれている。

本カリキュラムは全 10 回の単元に分けられている。1 回ごとに講義―演習形式で進められる。1 回分の授業にかかる時間は学習者の質にも影響されるが、おおよそ講義 1 時間～1. 5 時間、演習 1 時間～1. 5 時間の合計 2 時間～3 時間程度と想定している。

第 1 回～第 9 回までは、自動販売機における取扱商品の追加、削除、検索だけをするプログラムを書くという内容になっている。概念を学びながら、機能を変えずに徐々にプログラムを書き直していくことで、概念の利点を実感させるためである。

第 1 回～第 3 回までは、プログラミングの基本から、構造化プログラミングまでを通して、プログラムの意味について議論を重ね、プログラムを「意味のカタマリ」にする、という考え方を教える内容となっている。

第 4 回～第 7 回の内容は、第 3 回まで議論した、プログラムを「意味のカタマリ」にする、という考え方を用いて、オブジェクト指向の概念そのものについて、議論するものとなっている。

第 8 回～第 10 回までは、データ構造とアルゴリズムについての理解を深めながら、オブジェクト指向の概念を適用することの練習をするという内容となっている。

5.3.2. 各回の概要

各回ごとの概要を述べる。

第1回 プログラムの基本

コメントの書法を通して（リスト 5-1)、または人の書いたプログラムにコメントをつけるという議論（第 4 章を参照）を通して、プログラムの意味について考える。

```
/**
* オブジェクト指向哲学-プログラミング編-
* 第１回　一番簡単な自動販売機の取扱商品情報管理システム
*/
public class Main1_2 {

  /**
  * 取扱商品を表示するプログラム
  */
  public static void main(String[] args) {

    //自動販売機プログラムの開始を表示する
    System.out.println("自動販売機が開始しました。");

    //商品情報を保存する
    int itemInfo;

    //商品情報（コーラ）を代入する
    itemInfo = 1000;

    //現在取り扱っている商品情報を表示する
    System.out.println(itemInfo+"は販売中です");

  }

}
```

リスト 5-1 ：書法を教えるためのプログラム

第2回　配列の利用

第 1 回で書いた、配列を使わずに書かれたプログラム（リスト 5-2）を読み、その問題点を議論する。そして、問題点を解決するためにプログラムに配列を利用して書き直す、（リスト 5-3）その過程を通して、配列の利点を学び、配列はどのようなときに使えるのか、つまり、「意味のカタマリ」の議論をする。

```
int itemInfo01;
int itemInfo02;
int itemInfo03;
int itemInfo04;

if(itemInfo01 != -1){
  System.out.println(itemInfo01+"は販売中です");
}
if(itemInfo02 != -1){
  System.out.println(itemInfo02+"は販売中です");
}
if(itemInfo03 != -1){
  System.out.println(itemInfo03+"は販売中です");
}
if(itemInfo04 != -1){
  System.out.println(itemInfo04+"は販売中です");
}
```

リスト 5-2：配列を使わずに書かれたプログラム（抜粋）

```
//商品情報を保存するための配列を定義する
int[] itemInfoArray = new int[10];

//商品情報を保存するための変数を初期化する
//何も入っていないことを-1 として扱う
for(int i=0;i<10;i++){
  itemInfoArray[i] = -1;
}
```

リスト 5-3：配列を使って書き直したプログラム（抜粋）

第3回　手続き指向のプログラム

第 2 回で書いた、メソッドを使わずに書かれたプログラム（リスト 5-4）を読み、その問題点を議論する。そして、問題点を解決するために、プログラムにメソッドを使って書き直す。（リスト 5-5）その過程を通して、メソッドの利点を学び、どのようにメソッドを分割すればよいか、つまり、「意味のカタマリ」の議論をする。

```
//検索の準備として商品情報を登録する
int registerNum = 1001;//コーラを登録する
for(int i=0;i<10;i++){//商品情報が入っていない箱を探す
  if(itemInfoArray[i] == -1){//入っていない
    itemInfoArray[i] = registerNum;//書き込む
    break;
  }
}
registerNum = 1022;//ソーダを登録する
for(int i=0;i<10;i++){//商品情報が入っていない箱を探す
  if(itemInfoArray[i] == -1){//入っていない
    itemInfoArray[i] = registerNum;//書き込む
    break;
  }
}
```

リスト 5-4：メソッドを使わずに書かれたプログラム（抜粋）

```
/**
* 商品情報を登録するメソッド
*/
public static void register(int[] targetArray,int registerNum){
  //商品情報が入っていない箱を探す
  for(int i=0;i<10;i++){
    if(targetArray[i] == -1){//入っていない
      targetArray[i] = registerNum;//書き込む
      break;
    }
  }
}
```

リスト 5-5：メソッドを使って書き直したプログラム（抜粋）

第4回　クラスとオブジェクト

第３回で書いた、クラス/インスタンスの概念を適用していないプログラム（リスト 5-6）を読み、その問題点を議論する。そしてその問題点を、クラス/インスタンスの概念を適用することで解決していく。クラス/インスタンスの概念を段階的に学ぶために、第４回では、データ構造とアルゴリズムを結合するという問題については扱わず、単純にデータをまとめる、（リスト 5-7）という内容になっている。

```
//商品番号を保存するための配列を定義する
int[] itemNumberArray = new int[10];

//商品名を保存するための配列を定義する
String[] itemNameArray = new String[10];
```

リスト 5-6：クラス/インスタンスを使っていないプログラム（抜粋）

```
/**
* 商品情報クラス(Ver.1)
*/
public class ItemInfo {
  int no;//商品番号
  String name;//商品名
}
```

リスト 5-7：クラスとして「意味のカタマリ」を作ったプログラム

第5回　連結リストの利用

第 4 回までのプログラムで利用していた配列を使ったプログラムの問題点を議論した上で、連結リストの概念を学ぶ。第 6 回でデータ構造とアルゴリズムを結合するための複線となっている。

第6回　データ構造とアルゴリズムの結合

第 5 回で書いた連結リストを使って書かれたプログラムと、第 4 回で書いた配列を使って書かれたプログラムを比較しながら、その問題点を議論する。(図 5-4) 問題点はデータ構造とアルゴリズムを「意味のカタマリ」にすること解決できる。(図 5-5) 問題点を解決する過程を通して、クラス/インスタンスの概念を学ぶ。

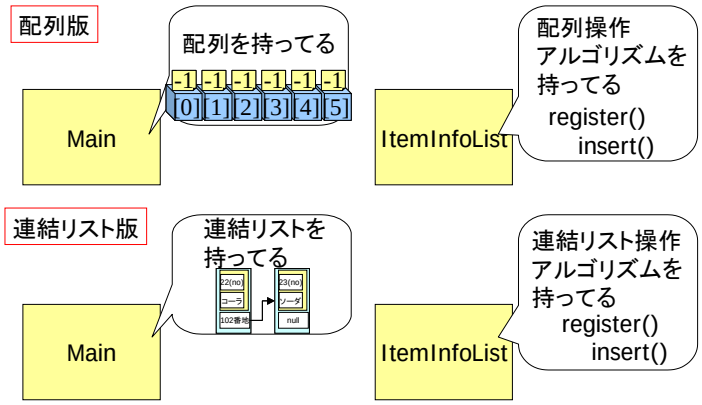

図 5-4：配列で書かれたプログラムと連結リストで書かれたプログラムを比較する

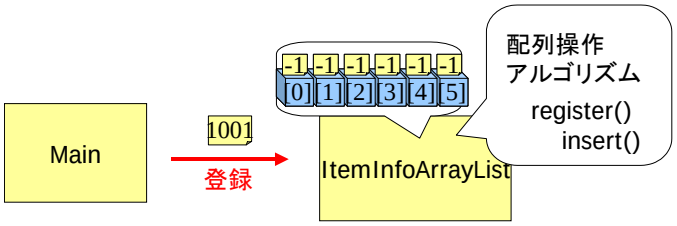

図 5-5：データ構造とアルゴリズムを「意味のカタマリ」にする

第7回　継承とインターフェイス

第 6 回で結合された 2 種類のクラス（配列で実装されたリスト、連結リストで実装されたリスト）を比較し、問題点を議論する。（図 5-6）問題点は「意味のカタマリのカタマリ」として、スーパークラスを作ることで解決できる。問題点を解決する過程を通して、継承の概念を学ぶ。（図 5-7）

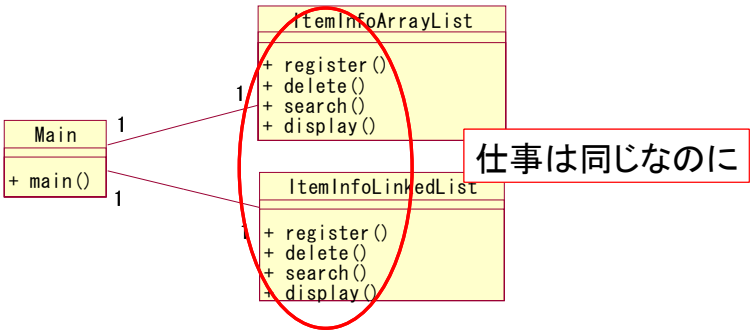

図 5-6：配列で実装されたクラスと連結リストで実装されたクラスを比較する

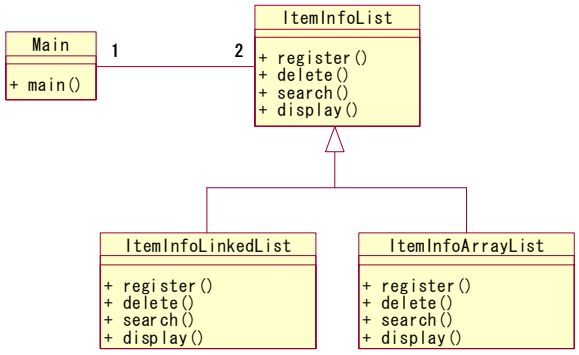

図 5-7：「意味のカタマリのカタマリ」を作る

第8回　効率の良いプログラム

検索の効率を向上させるためにはどうしたらよいか、バイナリサーチとリニアサーチの違いを実際にプログラムを動かしてみることによって実感する。また、アルゴリズムを「意味のカタマリ」にするという議論を通じて、クラス/インスタンスや継承の概念の復習をする。（図 5-8）その際に、デザインパターン（Strategyパターン）としてではなく、まず、メソッドとして実装し、「意味のカタマリ」にした場合と比較して議論を行う。

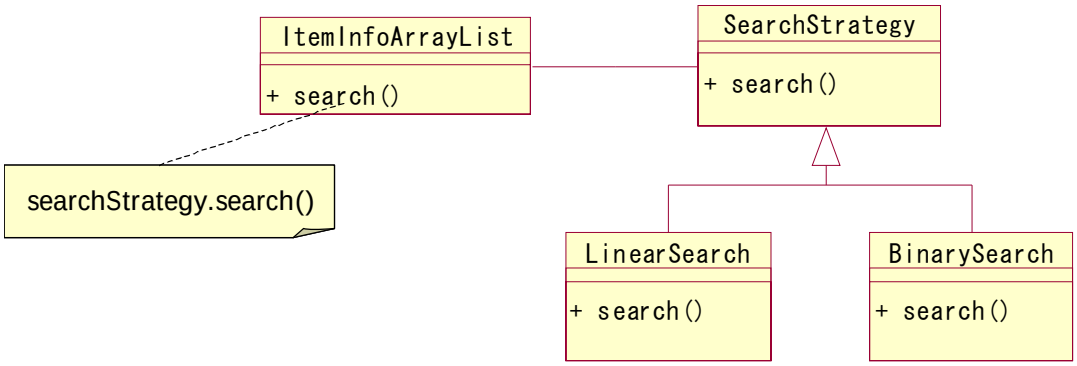

図 5-8：サーチアルゴリズムを「意味のカタマリ」にする

第9回　さまざまなアルゴリズム

第 8 回の続編で、並び替えの効率を向上させるためにはどうしたらよいか、簡単な「バブルソート」、「選択ソート」、「挿入ソート」のアルゴリズムの比較検討をする。アルゴリズムを「意味のカタマリ」にすることの復習も兼ねている。（図 5-9）

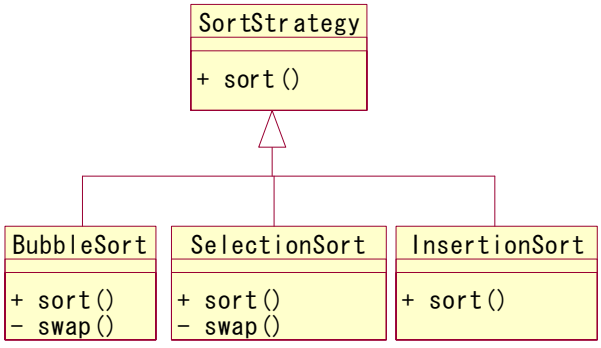

図 5-9：ソートアルゴリズムを「意味のカタマリにする」

第10回　スタックとキュー

自動販売機を完成させるために、商品の補充と販売機能（図 5-10）と、貨幣の投入とやり直し(Undo)機能（図 5-11）を追加する。商品の補充と販売では、古い商品から販売されなければならないのでキューの概念を扱う必要があり、投入された貨幣のやり直しでは、最後に投入された貨幣を取り出さなければならないので、スタックの概念を扱う必要がある。スタックとキューの違いから、どのように「意味のカタマリ」を作ったらよいか議論する。

図 5-10：「商品の補充と販売」機能（キュー）

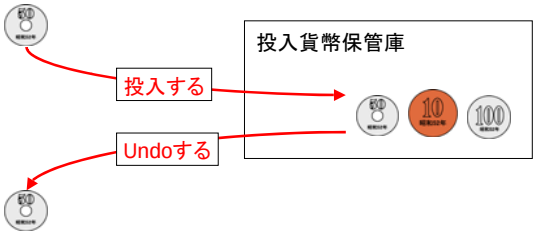

図 5-11：「貨幣の投入とやり直し」機能（スタック）

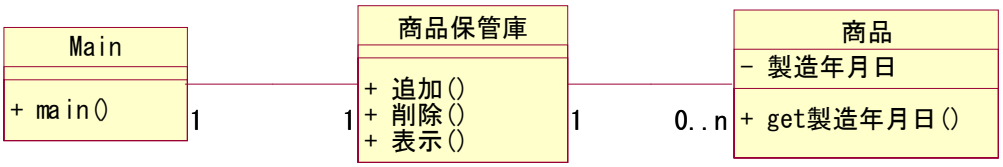

図 5-12：商品の補充と販売をするための設計モデル

5.3.3. 到達目標

プログラミング編カリキュラムの具体的な到達目標を各回ごとに示す。

第1回 プログラムの基本

- Java プログラムを書き、動かせる
  - クラス名とファイル名を同じにできる
  - プログラムは main()から始まることを説明できる
  - コンパイルし、正しくできたことを確認できる
  - 実行でき、正しくできたことを確認できる
- （見た目に）美しいプログラムが書ける
  - 適切なインデントを付けることができる
  - 変数やクラス名に適切な名前を付けることができる
  - プログラムの解説ではなく、目的を書いたコメントを付けることができる
- 仕事をプログラム手順に分解できる
  - プログラムでする仕事を目的に分解できる
  - 目的の階層構造を作ることができる
- Java の仕組みを説明できる
  - 変数と代入を説明できる
  - 出力する方法（System.out.println()の使い方）を説明できる
  - if 文を説明できる
  - 条件式を説明できる

第2回 配列の利用

- 配列を使ったプログラムが書ける
  - 配列を使う利点を説明できる
  - 配列と for 文を使って、コードの重複を減らすことができる
- 初期化の重要性を説明できる
- 配列を使って、1 対他の実装をするアルゴリズムを説明できる
  - 要素の追加アルゴリズムを説明できる
  - 要素の削除アルゴリズムを説明できる
  - 要素の検索アルゴリズムを説明できる
- Java の仕組みを説明できる
  - for 文の仕組みを説明できる
  - 配列を作る方法を説明できる

第3回　手続き指向のプログラム

- メソッドによって構造化されたプログラムが書ける
  - メソッドを使う利点を説明できる
  - メソッドを使って、重複コードを取り除くことができる
  - メソッドの定義、呼び出しの仕組みを説明できる
  - なぜ引数と返り値を使わなければならないのか説明できる
  - 引数と返り値を使うことにより、より汎用的なメソッドを作ることができる
  - 実引数と仮引数の結合の仕組みが説明できる
  - 適切なレベルで仕事をメソッドに分割することができる
- Java の仕組みを説明できる
  - メソッドの書式を説明できる
  - 変数のスコープ（有効範囲）について説明できる

第4回　クラスとオブジェクト

- クラスとオブジェクトを利用した簡単なプログラムが書ける
  - クラスとオブジェクトを使う利点を説明できる
  - クラスとオブジェクトの違いを説明できる
  - クラスを利用して、変数をまとめたプログラムが書ける
    （構造体的使い方ができる）
  - クラスを利用して、メソッドを目的別にまとめたプログラムが書ける
    （メソッド集的使い方ができる）
  - クラスは変数の型になることを説明できる
  - クラスの配列を使ったプログラムが書ける
  - クラスの配列と int の配列の違いを説明できる
  - 何も入っていない状態（null）を説明できる
- Java の仕組みを説明できる
  - クラスの書式を説明できる
  - クラスごとに別ファイルにできる
  - クラスをインスタンス化できる
  - コンストラクタの仕組みを説明できる
  - コンストラクタを使う利点を説明できる
  - NullPointerException はどのようなプログラムミスか説明できる
  - オブジェクトのメソッドには static を付けなくてよいことを説明できる

第5回　連結リストの利用

- 配列の問題点を説明できる
- 連結リストを使ったプログラムが書ける
  - 連結リストを使う利点を説明できる
  - 連結リストの概念を説明できる
  - 連結リストの問題点を説明できる
  - 連結リストの挿入アルゴリズムを説明できる
  - 連結リストの削除アルゴリズムを説明できる
  - 連結リストの検索アルゴリズムを説明できる
- Java における参照の仕組みを説明できる
  - 変数とメモリの関係を説明できる
  - primitive 型の変数がメモリに書き込まれる仕組みを説明できる
  - object 型の変数がメモリに書き込まれる仕組みを説明できる
  - 配列を new しなければならない訳を説明できる
  - primitive 型と object 型の違いを説明できる
  - primitive 型をすべて列挙できる
- Java の仕組みを説明できる
  - ガベージコレクションの利点を説明できる

第6回　データ構造とアルゴリズムの結合

- データ構造とアルゴリズムを結合させたクラスを作ることができる
  - データ構造とアルゴリズムを結合させることの利点を説明できる
  - 役割を考えたクラス設計ができる
- クラスをカプセル化することができる
  - カプセル化することの利点を説明できる
  - カプセル化しないと不正な操作ができることを説明できる
  - カプセル化されたクラスをアクセサを利用して適切に操作することができる
- クラス図を読むことができる
  - クラス図を使う利点を説明できる
  - 関係がプログラムでの参照になることを説明できる
  - クラス図での可視性を Java のアクセス修飾子と対応させることができる
  - 多重度を見て、オブジェクトのイメージ図が書ける
- Java の仕組みを説明できる
  - Java のアクセス修飾子を列挙でき、それぞれの意味を説明できる

第7回　継承とインターフェイス

- 継承とインターフェイスを利用したプログラムが書ける
  - 継承を使う利点を説明できる
  - 継承の概念（抽象化）を説明できる
  - 継承を使って、重複コードを取り除くことができる
  - スーパークラス、サブクラスを説明できる
  - スーパークラスの変数にサブクラスを代入することの意味を説明できる
- ポリモーフィズムを利用したプログラムが書ける
  - ポリモーフィズムを使う利点を説明できる
  - ポリモーフィズムの仕組みを説明できる
  - if 文を使ったプログラムを、ポリモーフィズムを使って書き直せる
  - if 文とポリモーフィズムを適切に使い分けられる
- 継承を使ったクラス図が読める
- Java の仕組みを説明できる
  - 継承の書式を説明できる
  - オーバーライドするメソッド名を間違えるとどうなるか説明できる
  - メソッドの定義と実装の違いを説明できる
  - 抽象クラスを使う利点を説明できる
  - インターフェイスを説明できる
  - 継承とインターフェイスの違いを説明できる
  - instanceof 演算子を説明できる

第8回　効率の良いプログラム

- 効率が考慮されたプログラムが書ける
  - 効率の悪いプログラムを使うとどうなるか説明できる
  - アルゴリズムを理解した上で、比較、入れ替えの実行回数を計算できる
  - 無駄な比較を排除し、性能を上げることができる
  - リニアサーチのアルゴリズムを説明できる
  - バイナリサーチのアルゴリズムを説明できる
  - 効率を BigO 記法を使って比較できる
- アルゴリズムをオブジェクトにしたプログラムが書ける（Strategy パターン）
- 定数は定数変数を使って、一まとめにできる
- Java の仕組みを説明できる
  - int 型と long 型の違いを説明できる
  - 現在時刻を知る方法(System.currentTimeMillis())を説明できる

第9回　さまざまなアルゴリズム

- 適切なアルゴリズムを用いたプログラムが書ける
  - 並び替え（ソート）アルゴリズムの性能について比較検討し、説明できる
  - バブルソートのアルゴリズムを説明できる
  - 選択ソートのアルゴリズムを説明できる
  - 挿入ソートのアルゴリズムを説明できる
  - 番兵つき挿入ソートのアルゴリズムを説明できる
- コンピュータ上での「入れ替え」(Swap)の仕組みを説明できる
- 番兵の概念を説明できる
  - 番兵を利用して問題解決ができる
- スーパークラスへメソッドを移動することにより、重複コードを取り除くことができる
- Java の仕組みを説明できる
  - ランダムな数字を発生させることができる（Math.random()）
  - 簡単なキャストができる（double 型→int 型）

第10回　スタックとキュー

- スタック、キューの概念を説明できる
  - スタックとキューの利点を説明できる
  - スタックとキューの違いを説明できる
- スタックとキューを利用して問題解決ができる
- 配列を用いたキューのアルゴリズムを説明できる
- 配列を用いた効率的なキュー（円環キュー）のアルゴリズムを説明できる
- 配列を用いたスタックのアルゴリズムを説明できる
- 新しいクラスを作ることにより、重複コードを取り除くことができる
- Java の仕組みを説明できる
  - コンソールから入力ができるプログラムを書ける
  - import 文の意味を説明できる
  - static 修飾子の意味を説明できる
  - 適切にクラスメソッドを利用できる

5.4. UML編カリキュラム

5.4.1. 概要

プログラミング編を終えた学習者が、UML を使ってオブジェクト指向の基礎概念を記述できるようになるためのカリキュラムである。UML を使って、簡単なモデリングを行うことで、オブジェクト指向の考え方、基礎概念の理解を深めるとともに、最低限の必要な UML の知識を獲得することを目的としている。

本カリキュラムは全５回の単元に分けられている。1 回ごとに講義―演習形式で進められる。1 回分の授業にかかる時間は学習者の質にも影響されるが、おおよそ講義 1 時間～1．5 時間、演習 1 時間～1．5 時間の合計 2 時間～3 時間程度と想定している。

全 5 回によって、ユースケース図、アクティビティー図、クラス図、オブジェクト図、シーケンス図、コラボレーション図、状態遷移図の各図を学ぶことができるようになっている。

UML の知識は言語情報であるが、モデリングは知的技術である。UML 各図を教えるときに、何故その図を書くのかを明らかにする必要がある。そのため、モデル化することの意味を教えるカリキュラムを作るために、すべての図式化することの意味を「システムを明らかにすること」と定め、システムを明らかにするためにはどうしたらよいかということを議論の焦点としている。従って、本カリキュラムはモデリングを通して、自動販売機というシステムを明らかにしていくという内容になっている。

5.4.2.　各回の概要

各回ごとの概要を述べる。

第1回　ユースケースを考える

人を幸せにするサービスを提供するシステムを作るために、人を幸せにするサービスを明らかに必要がある、という視点から議論が始まる。自動販売機に必要なサービスを考え、ユースケース図として図式化する。（図 5-13）

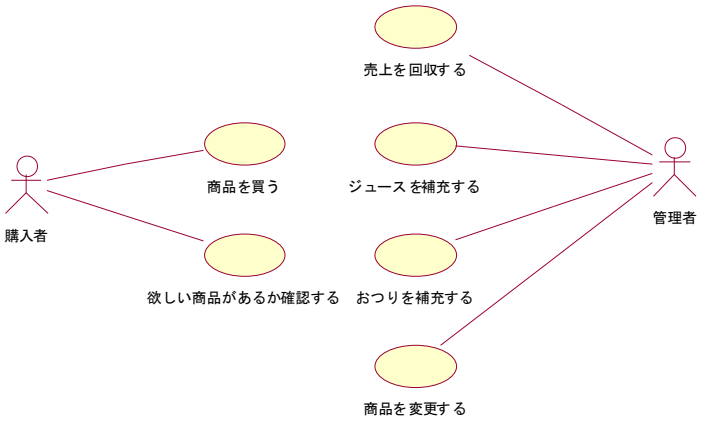

図 5-13：自動販売機のユースケース

第2回　ユースケースの図式化

ユースケース図はシステムをユースケースの集まりとして記述する記法であり、ユースケース内部のことは明らかではない、という議論から、ユースケースの内部をアクティビティー図によって図式化する。(図 5-14)

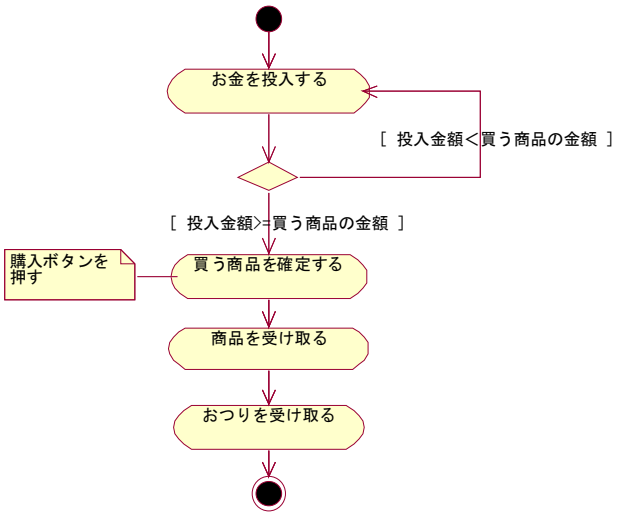

図 5-14：「商品を買う」ユースケースのアクティビティー図

第3回　オブジェクトの構造

ユースケースは、システムを外部から見た記述であるために、システムの内部が明らかでない、という議論から、システム内部の構造をオブジェクト図、クラス図を用いて図式化する。(図 5-15) 何をオブジェクトにするか、異なるモデルを比較しながら議論する。(図 5-16)

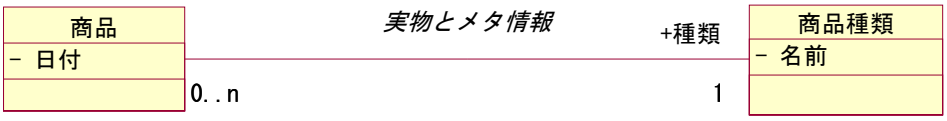

図 5-15：自動販売機のクラス図（抜粋）

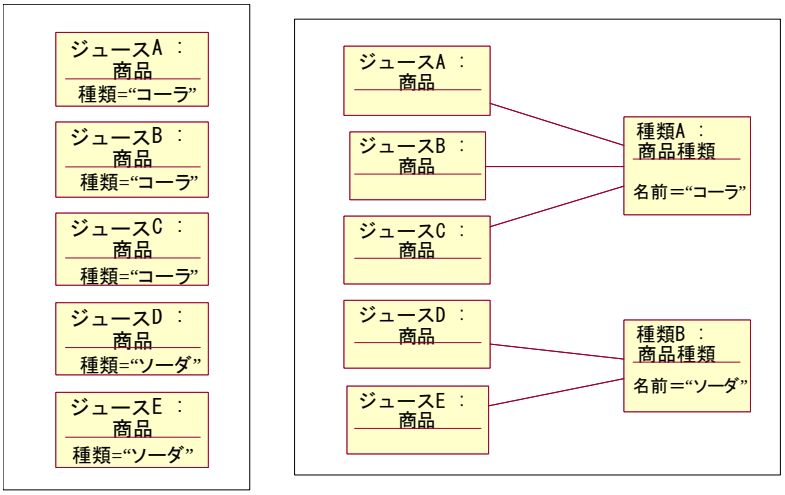

図 5-16：商品の種類をオブジェクトにするか属性にするか議論する

第4回　オブジェクトのメッセージ交換

オブジェクト図によってシステム内部が明らかになったが、システムの動的な変化が明らかでない。という議論から、システムの動的な変化を、オブジェクトのメッセージ交換の仕組みを議論しながら、相互作用図（コラボレーション図、シーケンス図）を用いて図式化する。(図 5-17)

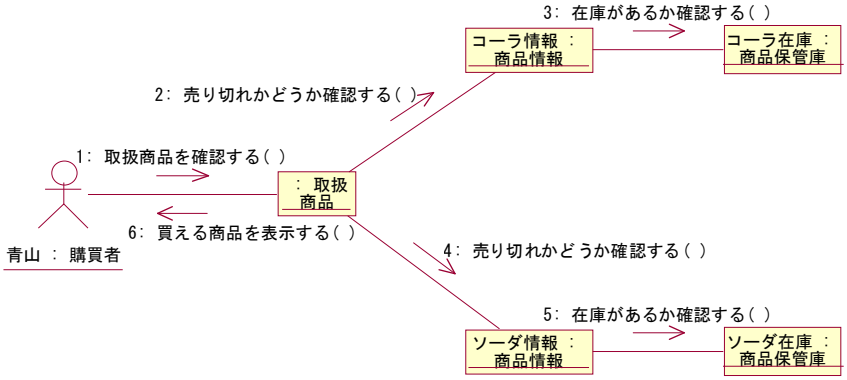

図 5-17：自動販売機のコラボレーション図（抜粋）

第5回　オブジェクトの動的変化

相互作用図によって、システムの動的な変化は明らかになったが、オブジェクト内部の動的な変化は明らかでない、という議論から、オブジェクト内部の動的変化を状態遷移図として図式化する。オブジェクトの状態について、2 つの例（図 5-18、図 5-19）を比較しながら議論する。

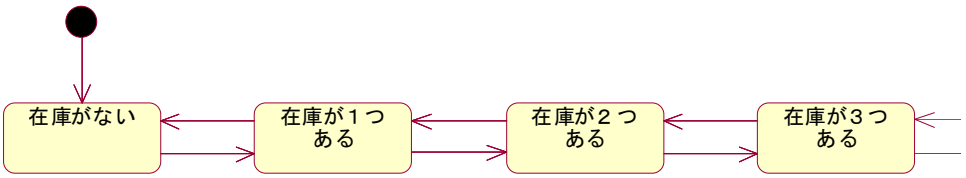

図 5-18：起こりうるすべての場合を考えたときの「商品保管庫」の状態遷移図

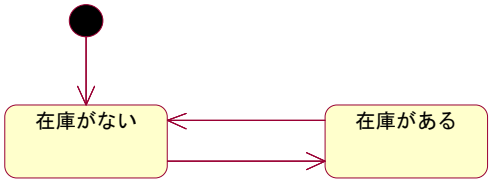

図 5-19：「意味のカタマリ」を考えた「商品保管庫」の状態遷移図

5.4.3. 到達目標

UML 編カリキュラムの具体的な到達目標を各回ごとに示す。

第1回 ユースケースを考える

- ユースケースモデルを使って、対象システムについて人とコミュニケーションできる
  - ユースケースモデルを使う利点を説明できる
  - ユースケースモデルを適切に記述できる
  - シナリオを作れる
  - ユースケースとシナリオの違いを説明できる
  - シナリオが与えられたとき、適切にユースケースを抽出できる
  - シナリオが与えられたとき、適切にアクタを抽出できる
- 人を幸せにするサービスとは何か議論できる
  - 人と完成イメージの共有化ができる
- ユースケース図が書ける
  - 適切に粒度を調整できる
  - システムと外部の境界を定められる

第2回 ユースケースの図式化

- 複雑なイベントフローが与えられたとき、アクティビティー図を使って明らかにすることができる
  - アクティビティー図を書くことの利点を説明できる
  - ユースケースとアクティビティー図の関係を説明できる
  - アクティビティー図を使って分岐を明らかにできる
  - レーンを使ったアクティビティー図を使って、アクタとシステムの仕事を明らかにできる

第3回 オブジェクトの構造

- クラス図とオブジェクト図を利用して、システムの構造を人とコミュニケーションできる
  - 分析モデルを作る利点を説明できる
  - 分析モデルと設計モデルの違いを説明できる
- クラス図が読める
  - クラスとオブジェクトの関係を説明できる
  - クラス図とオブジェクト図の変換ができる
- 適切にクラスをモデル化できる

- オブジェクトの種類（バウンダリ･コントロール･エンティティ）を分類できる
- 適切にクラスを抽出できる
- クラスにする粒度を決められる
- 属性にするか、クラスにするか決められる

第4回　オブジェクトのメッセージ交換

- 相互作用図を使ってシステムの動的モデルについて人とコミュニケーションできる
  - 相互作用図を書くことの利点を説明できる
- 相互作用図が読める
  - クラスではなく、オブジェクトで考えなければならない理由を説明できる
  - ユースケースではなく、シナリオで考えなければならない理由を説明できる
  - メッセージとは何か説明できる
  - シーケンス図とコラボレーション図のうち、適切な表現方法を選択できる
- 動的モデルを書くことで、概念モデルがユースケースに適合するかを検証できる
  - 相互作用図とユースケースの関係を説明できる
  - 相互作用図を用いて、システムの動的モデルを明らかにできる
- オブジェクトが受信できるメッセージをクラス図に書く理由を説明できる

第5回　オブジェクトの動的変化

- 状態遷移図を用いてオブジェクト内部の動的変化について人とコミュニケーションできる
  - オブジェクト内部の状態について考える利点を説明できる
- 状態遷移図が読める
  - イベントとメッセージの違いを説明できる
  - ガード条件とは何か説明できる
- 状態遷移図が書ける
  - 「振る舞いの変化」を基準に、何を状態とするか決められる
  - 状態遷移図とアクティビティー図の違いを説明できる
  - どのオブジェクトについて状態図を書くべきかをオブジェクトの責任に留意して決められる

第6章 評価と考察

本研究で提案する教育カリキュラムである「オブジェクト指向哲学」の有用性を評価するために、本研究を行うきっかけとなった、新入社員教育に問題を抱えている NK-EXA 社において、20 名の社員に「オブジェクト指向哲学」を受講してもらった。本章では、モニター授業のアンケート調査として受けた評価から、提案カリキュラム「オブジェクト指向哲学」の有用性について考察する。

6.1. 評価方法

今回モニター授業に参加していただいたのは、NK-EXA 社に勤務している社員 20 名である。この中には、実際に技術者として仕事をされている方や、新入社員教育担当の方など、様々な方が含まれている。

本来は教育カリキュラムとして、対象となる学習者を集めてモニター授業を行い、実際の学習効果を測定した結果を評価とするのが望ましい。しかし、今回は時期の問題もあって、今回のカリキュラムの対象である新入社員を集めるのは困難であった。

そのため、今回のモニター調査では、集まっていただいた社員の方に「オブジェクト指向哲学」の授業を新入社員として参加していただき、

- 自分が過去に受けた新入社員教育と比較して今回提案するカリキュラムは有用であるか
- 自分が今技術者として仕事をしているという観点から、将来技術者になるための入門教育として今回提案するカリキュラムが適当かどうか

という視点からのアンケート調査を行うという方法で評価を受けることにした。

なお、今回のモニター授業は、時間が限られているので、「オブジェクト指向哲学」の主たる特徴である、1)オブジェクト指向の基本概念を知的技能として教える、2)オブジェクト指向を考え方から教える、という特徴が有用であるかを特に評価しやすい個所を抜粋して授業を行った。

具体的には、1)の特徴が評価しやすい個所として、クラス/インスタンスの概念を教える単元である「オブジェクト指向哲学」プログラミング編の第 4 回～第 6 回、2)の特徴が評価しやすい個所として、プログラムの意味について考えさえる単元である「オブジェクト指向哲学」プログラミング編の第 1 回～第 3 回、つまり「オブジェクト指向哲学」プログラミング編の第 1 回～第 6 回をとりあげた。また、単元の順番どおり第 1 回から始めることにした。

6.2. アンケート結果

本節では、モニター授業でのアンケート結果を示す。

6.2.1. カリキュラム全体、基本方針について

「オブジェクト指向哲学」のカリキュラム全体としての有用性を問うものとして、「新人社員にとってこの講座は有益だと思いますか？」という質問をした。4 段階評価（非常に役立つ、役立つ、普通、あまり役立たない）とコメントによる質問の結果を示す。

まず、4 段階評価の結果を（図 6-1）に示す。

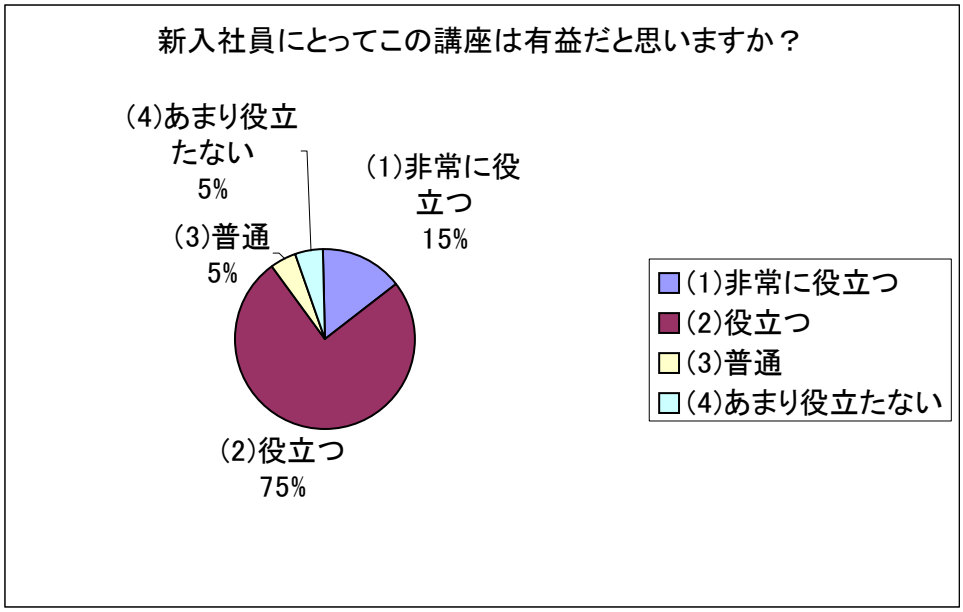

図 6-1：新人社員にとってこの講座は有益だと思いますか？

次に、主なコメントを示す。

A) プログラミングの本質を考えることができるのでよい。言語教育は文法中心になりがちだが、それを打破した講座でよい。

B) いきなりクラスやオブジェクト指向を出さずに、そういったものを使用するに至る理由や利点・欠点をまず洗い出してから、その結果オブジェクト指向なんだよという展開なので無理がないと感じた。

C) JAVA を習得する流れでオブジェクト指向の考え方が理解できる。但し、前段で Java のコーディングを経験させる必要がある。また、教えるとき言語教育と指向の考え方の調和した教え方をしないと中途半端になる恐れあり。

D) 早期にオブジェクト指向の考え方を教えることはその後のプログラミングの理解にとって非常に役立つと思います。

E) 初めのころは文法よりも考え方がわからないはずですから、そういった意味でもとても有用な内容であったと思います。

F) 私が受けた新人教育は意味がないことが多かった。それは、知識をつけるための教育だったからだ。今回のように、考えさせられるような講座は、業務においても、利用できると思った。つまり、Java の講座だとしても、考え方を Java 以外でも使えるということだ。

G) プログラムの目的というものを明確にしながらコーディングしていく習慣が身につく。

H) プログラムを作成するときただ作成するのではなく、機能別にまとめて考えることを重視している教材なので、受講後のプログラム作成の考え方にいい影響が出る。

I) 決して「オブジェクト指向哲学」の題名どおりの内容ではなかった。とても、中途半端な印象だった。Java 言語の教育なのか、オブジェクト指向の教育なのか、プログラミング入門なのか、目的が弱い印象を受けます。

J) やっておいて損はないと思います。でもやるなら Java のみとかにした方が負担もかるくてよいと思います。あれもこれもやってもどれも中途半端になると思うので。

6.2.2. 教授法について

「オブジェクト指向哲学」の教授法、つまり議論を中心として、「考えさせる」スタイルの有用性を問うものとして、「この講座の教育スタイルはいかがでしたか？」という質問をした。４段階評価（非常に良かった、良かった、普通、あまり良くなかった）とコメントによる質問の結果を示す。

まず、4 段階評価の結果を（図 6-2）に示す。なお、「あまり良くなかった」という答えはなかった。

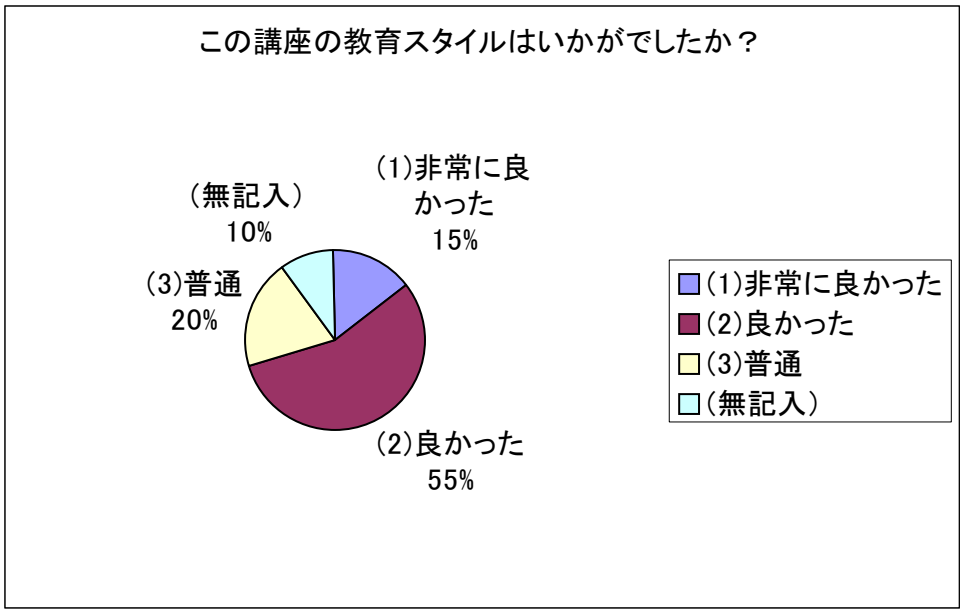

図 6-2：この講座の教育スタイルはいかがでしたか？

次に、主なコメントを示す。

K) 質問を受講者に多く投げることにとって常に考えていく姿勢が自然と身につく内容でした。皆に問いかけを行う方法はよいと思います。もう少し演習を増やすべきでは。

L) プログラミングの文法や慣習の述べる前に、議論の時間をもち、考えさせようとするのは理解しやすい。

M) みんなでディスカッションというのはよかった。案外、当たり前の事を知らないので、実際に説明しようとしてもできないということがわかった。

N) 一方通行の教え方でなく考えるようにしむけていたのはよいと思いました。ただあまりにも深く考える必要はあるのかな？と思いました。やっていくうちに本当の意味、役割が見えてくることもあるので…

6.2.3.　教師の役割について

「オブジェクト指向哲学」の教授法における、教師の役割を問うものとして、「この講座の講師･アシスタントの教え方はいかがでしたか？」という質問をした。４段階評価（非常に良かった、良かった、普通、あまり良くなかった）とコメントによる質問の結果を示す。

まず、4 段階評価の結果を（図 6-3）に示す。

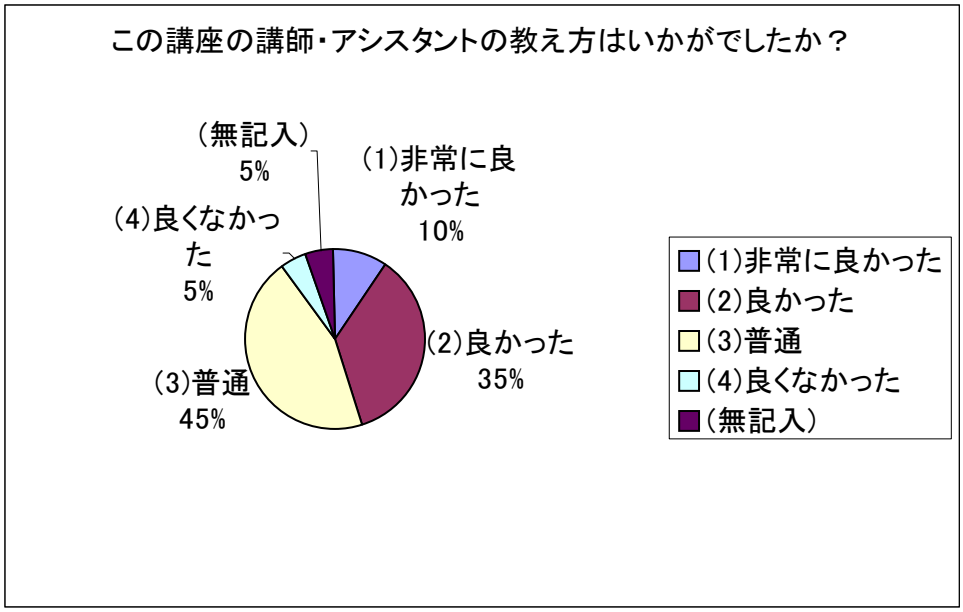

図 6-3：この講座の講師･アシスタントの教え方はいかがでしたか？

次に、主なコメントを示す。

O) 声も聞き取りやすく、教え方も新入社員の目線になって教えていたので、とてもわかりやすかったです。

P) テンポもよく、説明の仕方もよかった。テキストとパワーポイントもうまく使っていた。

Q) 質問があまりに抽象的過ぎて、どの観点から答えれば良いのかわかりませんでした。しかも、求めている解答ではなかった時の対応には気を付けたほうがよいでしょう。言葉の節々に劣等感を感じさせられることがありました。

R) "議論をする"という観点はよいと思うが、議論というよりも追求もしくは尋問のように感じた人も少なくないのでは？講師の解答への導き方が強引すぎる点を改善すべきだと思う。

S) ただ全員が深く理解できるようになるにはかなりの時間を要すると思うので時間配分等困難を極めるのでは。

6.2.4. 難易度について

「オブジェクト指向哲学」の新入社員教育として難易度、また想定した前提知識が適切かどうかを問うものとして、「新入社員にとってこの講座の難易度は適切だと思いますか？」という質問をした。３段階評価（難しすぎる、ちょうど良い、簡単すぎる）とコメントによる質問の結果を示す。

まず、３段階評価の結果を（図 6-4）に示す。なお、「簡単すぎる」という答えはなかった。

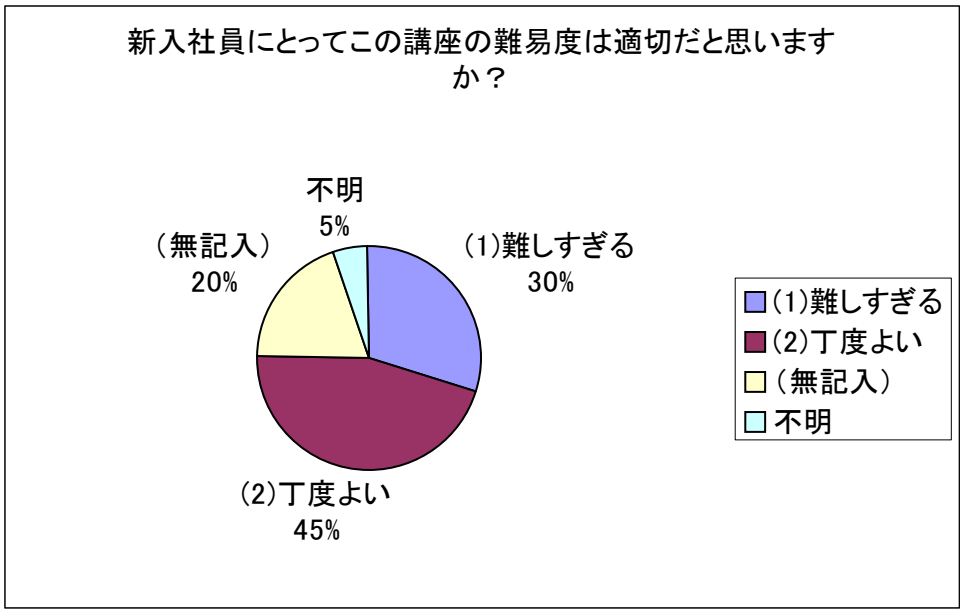

図 6-4：新入社員にとってこの講座の難易度は適切だと思いますか？

T) オブジェクト指向はけっこう奥が深いものであると思います。そのため研修のみでの理解はけっこう苦しいものがあると思うので、方法論とかはあとまわしでとりあえずはプログラムになれる方がとっつきやすいと思います。

U) ある程度プログラミングの勉強をしてからではないとプログラミング演習はムリ。オブジェクト指向とは何かの理解ができない。

V) 文法を理解しているのなら、オブジェクト指向プログラミング入門として丁度よいと思います。

W) プログラムの基礎部分の学習が短期間だと少し難しいと思います。配列を使ったプログラムを作成したことがない人も多数いると思うので、その人達に「何が問題か？」と問いかけても比較対象がなければ答えられません。何度かメソッドなどを作成してプログラミングのイメージがつかめている人には適当だと思います。

X) コーディングのレベルの演習は、新入社員のスキルのバラツキを考えると考慮が必要かなと思います。

6.3. 考察

本節では、前節で示したアンケート結果より、「オブジェクト指向哲学」の有用性を考察する。

6.3.1. 基礎概念を知的技能として教えることについて

まず、「オブジェクト指向哲学」カリキュラムの大きな特徴である、「オブジェクト指向の基礎概念を言語情報としてではなく知的技能として教える」ということについて考察する。

本研究では、第 3 章において、知的技能として教えるための学習目標として「レベル１：オブジェクト指向の基礎概念の利点を説明できる能力」を定め、「オブジェクト指向哲学」では、レベル１から教えていくこと、つまり「利点を説明できるようになる」ように教えることを重視した。A),B)のコメントから、この学習目標は適切であり、知的技能として教えるということの重要性を示唆するとともに、「オブジェクト指向哲学」は、この学習目標を教えるために有用であると評価されたと考えられる。

また、コメント C)は、知的技能として教えるのは重要であるが、言語情報としての教育とのバランスが重要であることを指摘している。つまり、「オブジェクト指向哲学」で定義している「最低限」の言語情報では、不十分である可能性がある。

6.3.2. 考え方から教えるということについて

次に、「オブジェクト指向哲学」のもう一つの大きな特徴である、「考え方から教える」という方針についてのコメントを取り上げて考察する。

D),E)など多数の受講者から、「考え方から教える」ことは重要であり、そのためのカリキュラムとして、「オブジェクト指向哲学」は有用性が高い、というコメントをいただいた。このことから、「考え方から教える」という本研究の基本姿勢は適切であるといえる。

また、コメント F)は、認知的方略としての能力、すなわち第 3 章で定義した、IT 技術者として必要な「技術を考え方から学ぶことができる能力」が重要であることを示唆しているとともに、その能力を育てるカリキュラムとして「オブジェクト指向哲学」が有用であるという評価を受けたといえる。

また、本研究においては、第 3 章でオブジェクト指向の考え方を、システムを「意味のカタマリ」に分解していくことと定義しており、「オブジェクト指向哲学」はその考え方から教えるものである。コメント G),H)から、人間からみたプログラムの意味（目的）を考え

ることが重要であることと、「意味のカタマリ」を作ることが重要であることを示唆しているとともに、その教育法として「オブジェクト指向哲学」は有用であるという評価を受けたといえる。

6.3.3. 「考えさせる」教授法について

「オブジェクト指向哲学」の教授法、つまり議論を中心として、「考えさせる」スタイルの有用性を問うものとして、「この講座の教育スタイルはいかがでしたか？」という質問をした。（図 6-2）結果、70%の受講者から、「良かった」または「非常に良かった」という評価を受けた。コメントからも、「議論」中心の教授法が有用であるという指摘があった。

しかし、この教授法にとって教師の役割が重要な問題となることが、「この講座の講師アシスタントの教え方はいかがでしたか？」という質問（図 6-3）から分かる。この質問の結果は「良かった」または「非常に良かった」が 45%であり、コメントQ)の「教師の質問があまりに抽象的過ぎる」、「言葉の節々に劣等感を感じさせられる」や、コメントR)の「講師の解答への導き方が強引すぎる」というような指摘があり、教師の役割が非常に重要であり、教師の役割を明確に定める必要があるといえる。

6.3.4. カリキュラム全体として

カリキュラム全体としての有用性を問うものとして、「新人社員にとってこの講座は有益だと思いますか？」という質問をした。（図 6-1）結果、90%の受講者から「役立つ」または「非常に役立つ」という評価を得た。

また、本節で考察してきたように、本カリキュラムの特徴である、

- オブジェクト指向の基礎概念を知的技能として教えること
- オブジェクト指向の考え方から教えること

は総じて有用であるという評価を受けた。

このことから、「オブジェクト指向哲学」カリキュラムは有用性があると考える。つまり、本研究の方向性、基本的考え方は間違っていないと言える。

しかし、問題点も数多く指摘された。I),J)のコメントは、「オブジェクト指向哲学」の問題点についてのコメントである。言語情報としてのJava 言語の教育か、知的技能としての、オブジェクト指向の基礎概念の教育か、それともプログラミング教育かがつかめないために、このような評価を受けたと考えられる。このことから、この「オブジェクト指向哲学」と周囲の教育環境の関係を記述することが非常に重要であることが言える。そのために第 5 章で述べた「オブジェクト指向哲学」の事前条件について、もう少し具体的に記述すべきであることを示していると考えられる。これは、難易度について聞いた質問のコメント T)～X)からも指摘されている。「基礎的なプログラミング能力」というものについて、これから明らかにしていく必要があると考えられる。

第7章 総括

本章では、本研究の研究成果をまとめる。

まず、第 2 章では、既存の教育方法を分析することによって、学習目標の整理と分類が必要であるということを述べた。

第 3 章では、知的技能を教える最初の段階として、「オブジェクト指向の基礎概念の利点を説明できる能力」が必要であり、認知的方略としての「考え方」が重要であるということを述べた。また、オブジェクト指向の考え方を「意味のカタマリ」と定義し、それによって、歴史的に構造化プログラミングから教える方法を述べた。

第 4 章では、具体的な例を通して、学習者に熟考させるための議論のさせ方、オブジェクト指向の基礎概念を教えるために、どのようにカリキュラムを組めばよいかを述べた。

第 5 章では、提案するカリキュラム「オブジェクト指向哲学」の概要を述べた。

第 6 章では、評価実験より、若干の問題点はあるものの、総じて有用性が高いという評価を得た。

結論として本研究の基本的な考え方の方向性は概ね正しいと言える。

今後「オブジェクト指向哲学」が、改良を重ねることによってさらによいカリキュラムになることが望まれる。

また、本研究は Instruction Design の手法を適用した。論文中に Instructional Design の用語は出てこないが、本論文の過程そのものが Instructional Design の過程である。

Instructional Design の過程の中で学習目標を整理し分類していくことが非常に重要であったといえる。学習目標を整理することで、教育するべきことが明らかになり、それらの分類によって効果的な教授法への方針が導かれた。その後教授法の仮説を立て、改良を重ねることによって、「オブジェクト指向哲学」が完成した。

今後様々な教育の場で Instructional Design の手法が適用されることが望まれる。

謝辞

主査であり、私の研究を温かく見守っていただいた環境情報学部の大岩元教授に深く感謝します。学部時代から懇切丁寧な指導をいただき、数々の有益なアドバイスをいただきました。私が様々な研究をするきっかけを作ってくださり、その研究の集大成として論文の形にすることができました。本当に有難うございます。

また、この研究を進めるにあたり、NK-EXA（株）の児玉公信さんを始めとする技術部の皆さんには大変お世話になりました。特に、日程的に苦しかったにもかかわらず、モニター授業の手配をしていただいたことを感謝します。また、モニター授業として参加してくださった皆さんに感謝いたします。

博士課程の中鉢欣秀さんには、4 年間大変お世話になりました。プログラミングの基礎から大変丁寧に指導していただきました。中鉢さんに教わったオブジェクト指向技術がなかったら、この研究はできませんでした。アシスタントまでやっていただき本当に有難うございました。

そして、4 年間一緒にオブジェクト指向技術の試行錯誤の過程を共にしてきた、浅加浩太郎君、海保研さんに感謝します。あなたたちとの議論は学生生活の一番の思い出として残ると思います。

一緒に教材作りを進めてくださり、またアシスタントもやっていただいた岡田健君にこの場を借りて感謝の意を表します。

最後に、CreW Project のみんなには、様々な面からバックアップをしていただいて本当に助かりました。本当にどうも有難う。

参考文献

[1] "平成１３年版労働経済の分析",厚生労働省,2001
http://www.mhlw.go.jp/wp/hakusyo/roudou/01/

[2]Leslie J.Briggs, Kent L.Gustafson, Murray H.Tillman, "Instructional Design Principles and Applications",1991

[3]Patricia L.Smith, Tilman J.Ragan, "Instructional Design Second Edition", John Wiley & Sons, inc., 1999

[4]I.ヤコブソン, M.クリスターソン, P.ジョンソン, G.ウーバガード, "オブジェクト指向ソフトウェア工学 OOSE -use-case によるアプローチ-", 株式会社トッパン, 1995

[5]ジェームズ・ランボー, マイケル・ブラハ, ウィリアム・プレメラニ, フレデリック・エディ, ウィリアム・ローレンセン, "オブジェクト指向方法論 OMT", 株式会社トッパン, 1992

[6]本位田真一, 山城明宏, "オブジェクト指向システム開発", 日経 BP 社, 1993

[7]ピーター・コード, マーク・メイフィールド, "UMLによるJavaオブジェクト指向設計 第２版", 株式会社ピアソンエデュケーション, 1997

[8]ジョセフ・シュムラー, "独習 UML", 翔泳社,2000

[9]MISCO オブジェクト指向研究会, "オブジェクトモデリング表記法ガイド", 株式会社プレンティスホール出版, 1998

[10]クレーグ・ラーマン, "実践 UML -パターンによるオブジェクト指向開発ガイド-", 株式会社プレンティスホール出版, 1998

[11]Ivar Jacobson, Grady Booch, James Rumbaugh, "UML による統一ソフトウェア開発プロセス", 翔泳社, 2000

[12]テリー・クアトロニー, "UML 活用型開発入門 Rational -Rose を使った実践的ケーススタディ-", 株式会社トッパン, 1998

[13]H.E. エリクソン, M. ペンカー, "UML ガイドブック", 株式会社トッパン, 1998

[14]グラディ・ブーチ, "UML ユーザガイド", 株式会社ピアソンエデュケーション, 1999

[15]ペルディタ・スティーブンス, ロブ・プーリー, "オブジェクト指向とコンポーネントによるソフトウェア工学 -UML を使って-", 株式会社ピアソンエデュケーション, 2000

[16]中村紀幸, "即戦 UML モデリング", 株式会社リックテレコム, 2001

[17]ゲリ・シュナイダー, ジェイソン・ウィンタース, "ユースケースの適用:実践ガイド", (株)オージス総研, 2000

[18]Martin Fowler, Kendall Scot, "UML モデリングのエッセンス -標準オブジェクトモデリング言語の適用-", アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社, 1998

[19]Martin Fowler, "UML モデリングのエッセンス 第２版", 翔泳社, 2000

[20]Martin Fowler, "アナリシスパターン", アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社, 1998

[21]Mark Grand, "UML を使った Java デザインパターン -再利用可能なプログラミング設計集-", 株式会社カットシステム, 2000

[22]W.プリー, "デザインパターンプログラミング", 株式会社トッパン, 1996

[23]Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, "-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", ソフトバンク株式会社, 1995

[24]Matthias Felleisen, Daniel P.Friedman, "Java とピザのデザインパターン", ソフトバンク株式会社, 1998

[25]F.ブッシュマン, R.ムニエ, H.ローネルト, P.ゾンメルラード, M.スタル, "ソフトウェアークテクチャ -ソフトウェア開発のためのパターン体系-", 近代科学社, 2000

[26]Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, "-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 改訂版", ソフトバンクパブリッシング株式会社, 1999

[27]PLoPD Editors, "-Pattern Languages of Program Design 選集- プログラムデザインのためのパターン言語", ソフトバンクパブリッシング株式会社, 2001

[28]佐藤英人, "オブジェクト指向がわかる本", オーム社, 1998

[29]Yoo Hong Jun, "[図解] Java 流オブジェクト指向入門", 株式会社技術評論社, 1998

[30]岩田裕道, 手島歩三, "ゼロからわかるオブジェクト指向の世界 -ビジネス活用のために-", 日刊工業新聞社, 1996

[31]P.Wegner, "はやわかりオブジェクト指向", 共立出版株式会社, 1992

[32]Timothy Budd, "オブジェクト指向プログラミング入門 第２版", アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社, 1997

[33]マーチン・ファウラー, "リファクタリング: プログラミングの体質改善テクニック", 株式会社ピアソン・エデュケーション, 2000

[34]B.F.ウェブスター, "オブジェクト指向開発の落とし穴", 株式会社プレンティスホール出版, 1995

[35]斎直人, "構造化分析設計技法入門 -これで分かった CASE ツールの基礎技法-", オーム社, 1996

[36]Robert Lafore, "Java で学ぶアルゴリズムとデータ構造", ソフトバンク株式会社, 1999

[37]戸松豊和, "増補改訂 Java プログラムデザイン", ソフトバンク株式会社, 1998

[38]ローラ・リメイ, チャールズ・パーキンス, "Java 言語入門 -アプレット・AWT・先進的機能-", 株式会社プレンティスホール出版, 1996

[39]Micheal C.Daconta, Eric Monk, "Java の落とし穴", ソフトバンクパブリッシング株式会社, 2000

[40]ナイジェル・ウォーレン, フィリップ・ビショップ, "Java の格言 -より良いオブジェクト設計のためのパターンと定石-", 株式会社ピアソン・エデュケーション, 2000

[41]ジョセフ・オニール, "独習 Java", 翔泳社, 1999

[42]佐伯胖, 大村彰道, 藤岡信勝, 汐見稔幸, "すぐれた教育とは何か -授業の認知科学-", 東京大学出版会, 1989

[43]"教育工学事典", 廣済堂, 2000

[44]Goerge M.Piskurich, "Rapid Instructional Design -Learning ID Fast and Right-", Jossey-Bass Pfeiffer, 2000

[45]William J.Rothwell, H.C.Kazanas, "Mastering The Instructional Design Process -A Systematic Approach- Second Edition", Jossey-Bass Publishers, 1998

[46]八幡紕芦史, "教育プログラムの戦略的構築法", 日経連出版部, 2000

[47] (株)オージス総研, "かんたん UML", 翔泳社,1999

付録

- 「オブジェクト指向哲学」-プログラミング編-　学習者用テキスト
- 「オブジェクト指向哲学」-プログラミング編-　教師用プレゼンテーション
- 「オブジェクト指向哲学」-プログラミング編-　確認問題
- 「オブジェクト指向哲学」-プログラミング編-　確認問題解答
- 「オブジェクト指向哲学」-UML 編-　教師用プレゼンテーション
- 「オブジェクト指向哲学」-UML 編-　確認問題