

enPoly：オブジェクト指向言語における ポリモーフィズムの概念を理解するためのワークベンチ

石川 裕季子¹ 松澤 芳昭¹ 酒井 三四郎¹

概要：オブジェクト指向言語を扱うために重要な概念の一つとして「ポリモーフィズム」が挙げられる。ポリモーフィズムの理解には、オブジェクトの動的振る舞いの理解が必要であるが、既存の学習支援システムは静的構造の可視化にとどまっていることが問題である。そこで本研究では、既存システムを改良し、ポリモーフィズムの学習支援を目的とするワークベンチ「enPoly」を開発した。enPoly の工夫点は、1) 動的振る舞いをアニメーションで表現する、2) インターフェイスの概念習得支援のために、メソッドの定義と実装の違いを視覚的に表現する、の 2 点である。ポリモーフィズムの適用能力を問うプログラミング課題を開発し、本システムの評価を行った。被験者は C 言語でプログラミングを学習し、Java の文法を学習した 12 名の学生であった。その結果、6 名の実験群全ての事例において基本課題を解けなかった被験者が、本システムを利用することでポリモーフィズムの概念を理解し、応用課題まで解けるようになるという結果が得られた。

キーワード：ポリモーフィズム、オブジェクト指向プログラミング、インターフェイス、動的振る舞い、アニメーション、学習支援

1. はじめに

オブジェクト指向プログラミングにおいて、ポリモーフィズムの概念を理解していることは重要である。ポリモーフィズムを利用することのメリットについて Bertrand Meyer は「この技法は頑強で拡張性に富むシステムの構築に広くて起用できる [1]」と述べている。

ポリモーフィズムとは、オブジェクトが実行時に、そのインスタンスによって機能を変えることができる仕組みである。ポリモーフィズムで書かれたプログラムの例を図 1 に示す。図 1 には、同等の結果が得られる if 文のプログラムも示されている。ポリモーフィズムで書かれたプログラムは、person という変数に代入されているインスタンスによって、そのインスタンスが持つ greeting() というメソッドが実行される仕様である。if 文で書かれたプログラムは、person という変数がどのクラスのインスタンスかどうかを判断し、出力命令を実行する仕様である。

筆者らの経験上、ポリモーフィズムの学習は困難である。その理由の一つとして、図 1 にあるように、ポリモーフィズムと同等の結果が得られるプログラムを if 文で書けるこ

とが挙げられる。特に、C 言語などの非オブジェクト指向言語を流暢に使用することができる人ほど、その傾向が強い。Jones[2] はオブジェクト指向プログラミングにおける継承やカプセル化などの概念と比較して、ポリモーフィズムは教材などで取り上げられておらず、学習環境が整っていないとしている。

そこで本研究では、動的な振る舞いをアニメーションで表現し、インターフェイスの概念を強調することで、ポリモーフィズムを学習する支援システムを開発し、評価を行った。

2. 先行研究

本章では、オブジェクト指向プログラミングの教育という観点から先行研究を述べる。

三浦らは、Java などの静的な構造を扱うオブジェクト指向言語を習得するためのワークベンチとして Anchor Garden を提案している [3]。Anchor Garden とは、オブジェクト指向言語において、抽象度が比較的高い「型」「変数」「オブジェクト」の考え方を学ぶことができるワークベンチである。ワークベンチとは、概念の理解を促す為のモデルの視覚化に加えて、モデルを直接操作できる高いインタラクティブ性を備えたシステムのことを指す。Anchor Garden を利用することで、学習者はワークベンチにおけ

¹ 静岡大学情報学部
Faculty of Informatics, Shizuoka University, Hamamatsu,
Shizuoka, 432-8011, Japan

ポリモーフィズム	if文
<pre>Person person = new Japanese(); person.greeting();</pre>	<pre>Person person = new Japanese(); if (person instanceof Japanese) { System.out.println("こんにちは"); } else if (person instanceof American) { System.out.println("Hello"); }</pre>

図 1 ポリモーフィズムと if 文のプログラム例

るモデルの対話を通じて、どんな操作が可能であるかを探究したり、操作によって概念世界のモデルがどう変化するかを模索しながら体験的に学ぶことができる、としている。

しかしながら、このシステムでは静的な構造のみを扱っているため、動的な振る舞いを利用するポリモーフィズムの学習はできない。

Squeak[4]のように、変数の型が存在しない言語を用いてポリモーフィズムを学習することができる。しかしながら本研究では、C 言語でプログラミングを学習し、Java の文法を学習している学生を対象としてポリモーフィズムの学習を支援している。C 言語や Java では変数の型が存在する。Squeak を用いて学習すると、今までの言語との差が大きく、学習者が戸惑ってしまう可能性がある。

Basem らは、BlueJ[5]を使用したポリモーフィズムの学習方法を提案している [6]。BlueJ とは、Kolling らが開発したシステムである。Kolling らは、プログラミング言語の文法を学習する前にクラスやオブジェクトについて学習することが、オブジェクト指向プログラミングを学習するうえで重要だと主張している。BlueJ の特徴は、クラス図やオブジェクトを作成・編集することをとおしてオブジェクト指向プログラミングについて学習することである。しかしながら、BlueJ には動的な振る舞いを表現する機能がないことが、ポリモーフィズムを学習する点で問題である。

Tarsem らは、ゲームプログラミングを題材としたポリモーフィズムの学習環境を提案している [7]。ゲームを完成させることを最終目標とし、学習者が楽しみながらプログラミングを学習できるという特徴がある。しかしながら、ポリモーフィズム学習のための機能がないことが問題である。

Gestwicki らは JIVE という初学者用ビジュアルデバッガを提案している [8]。JIVE は Java で記述されたユーザプログラムを DebugInterface を用いて動作させ、そのビジュアル化を自動で実施するシステムである。しかしながら動的振る舞いについてはシーケンス図の作成にとどまっている。シーケンス図を読解できない初学者に不向きであり、アニメーションによる直接表現は行われていない。

Ben-Ari らも同様の Jeliot というビジュアルデバッガを開発している [9]。Jeliot はアニメーションによってオブジェクトの生成や消滅、ポインタの代入などが示されると

ころが特徴的であるが、メソッド呼出し過程そのものはアニメーションの対象になっておらず、ポリモーフィズムの概念を学習するには不十分と考えられる。これらのツールは、学習者がオブジェクト構造を操作をするわけではないため、記述されたプログラムが必要である。そのため、プログラムを書けない初学者への支援ツールとしては不適切である。

3. 提案システム：enPoly

3.1 enPoly とは

enPoly は、ポリモーフィズムの概念を学習することを目的とした学習支援システムである。

enPoly とは「en-」という「～にする」という意味を持った接頭語と、Polymorphism の「Poly」を繋げることで、ポリモーフィズムがわかる人にする、という意味がある。enPoly は先行研究である Anchor Garden を基盤としており、いくつかの変更点を加えることで開発した。

enPoly が対象とする利用者を以下に提示する。

- C 言語のプログラミング経験がある者
- Java の文法を知っている者
- ポリモーフィズムの概念を未学習、もしくは未習得の者

3.2 システムの特徴 (Anchor Garden からの変更点)

本システムの特徴として、Anchor Garden からの変更点を図 2 に示す。変更点は、動的振る舞いとインターフェイスの概念を追加した点と、アンカーの仕様変更を行った点である。

3.2.1 アニメーション

enPoly では、メソッドの動作をアニメーションで表現できる機能を追加した。メソッドの動作は吹き出しで表現され、動作の種類によって表 1 のとおりに対応している。メソッド呼び出しの場合、吹き出しは変数からアンカーが繋がっているインスタンスに向かって移動する。インスタンスに到達すると、そのインスタンスが持つメソッドを実行する。戻り値がないメソッドの場合は、実行結果がそのインスタンスの右上に吹き出しとして表示される。戻り値があるメソッドの場合は、戻り値の吹き出しが呼び出し元に向かって移動する。インスタンスが変数を持っている場合

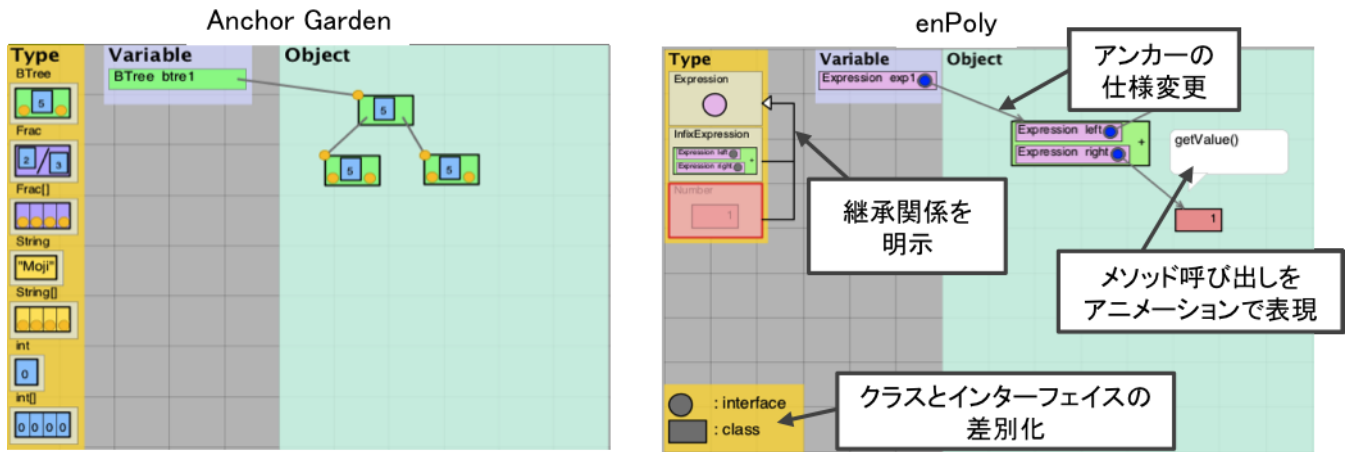


図 2 Anchor Garden からの変更点

表 1 メソッドの種類と吹き出しの対応

種類	呼び出し	実行結果表示	戻り値
吹き出しの色	白色	水色	桃色

は、さらにメソッド呼び出しが行われることもある（再帰構造など）。

3.2.2 インターフェイスの概念

インターフェイスの概念を習得するために、インターフェイスとクラスの違いと継承関係を視覚的に表現するという機能を追加した。インターフェイスを○、クラスを□で表現することで、学習者は見た目でインターフェイスとクラスを見分けることができる。インターフェイスはインスタンスを生成することができないため、学習者がインターフェイスを選択した状態で Object フィールド（後述）にマウスカーソルを重ねると、「cannot generate interface's object」という注釈が表示されるようになっている。継承関係は UML (Unified Modeling Language) のクラス図の表現方法を用いている。

3.2.3 アンカーの仕様変更

Anchor Garden のアンカーの仕様を変更した。Anchor Garden では、アンカータブは変数に付属しており、アンカータブをマウスでドラッグし、オブジェクトにはめることで、学習者は変数から参照するオブジェクトを簡単に指定できるとしている [3]。しかしながら、変数がオブジェクトを参照するということは、オブジェクトが持つポイントを参照するということである。したがって、オブジェクトにアンカータブを付属させ、変数にはめる方が動作として自然であると考えた。

3.3 enPoly の操作方法

実際に enPoly を使用した例を用いて、enPoly の使用方法を示す。

3.3.1 基本操作

enPoly を起動すると、画面には Type, Variable, Object

の 3 つのフィールドが現れる。Type フィールドは、使用できるクラスとインターフェイスが表示^{*1}されている。Variable フィールドは、変数を宣言するフィールドである。Object フィールドは、オブジェクトの生成を行うフィールドである。まず、変数の宣言方法とインスタンスの生成方法について述べる。enPoly の各部の名称を図 3 に示す。

- (1) Type フィールドから任意のクラスまたはインターフェイスをクリックする
- (2) クラスまたはインターフェイスを選択した状態で、Variable フィールドをクリックする（変数宣言）
- (3) 同様の状態で、Object フィールドをクリックする（インスタンス生成）
- (4) インスタンス生成時に現れるアンカーを変数にあるアンカーポイントにはめる（インスタンスの代入）

3.3.2 メソッド呼び出し

メソッド呼び出し中の画面を図 4 に示す。図中の吹き出し上にある矢印はアニメーションの動作の方向を示している。

- (1) 変数上で右クリックする
- (2) 表示されるメニューから、任意のメソッドを選択する
- (3) 引数があるものは、入力画面が表示されるので、任意の引数を入力する
- (4) 変数からオブジェクトに向かって吹き出しが流れる

3.3.3 継承

本実験で使用了課題 1（課題の詳細は付録に示す）を例として、enPoly における継承の使い方について述べる。Person（インターフェイス）型の変数に Japanese（サブクラス）のインスタンスを代入する図を図 5 に示す。図 5 からわかるように、Person（インターフェイス）型の変数に、それぞれサブクラスである Japanese, American のインスタンスを代入することができる。代入できる変数と、そうでない変数はアンカータブの色が青から赤に変化すること

^{*1} Type フィールドに表示されているクラスとインターフェイスは、メニューバーの Extra からモードを選択することで変更できる

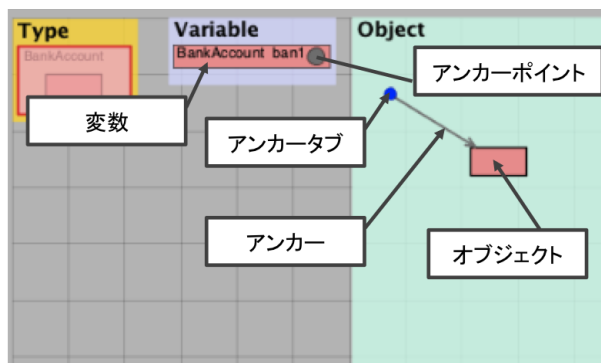


図 3 各部名称

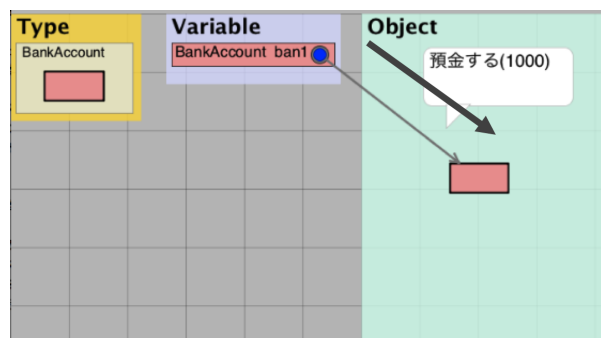


図 4 メソッドを呼び出したとき

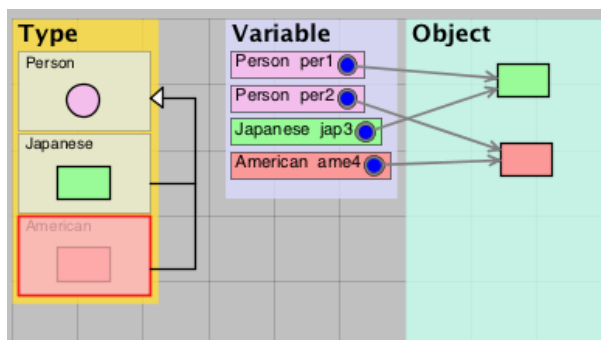


図 5 継承を利用したとき

でわかる。Person 型の変数から Japanese, American のインスタンスへ、それぞれ greeting() というあいさつをさせるメソッドを呼んだ結果を図 6 に示す。

3.3.4 再帰構造の表現

enPoly を利用して再帰構造を表現した図を図 7 に示す。この図は $6/(2+1)$ という中置記法を表現したものである。この構造は InfixExpression というクラスが演算子と、その両側の演算子または数値を持つ構造となっている。ゆえに、Object フィールドにある InfixExpression のインスタンスは Expression (インターフェイス) 型の変数を持っており、その変数にさらに InfixExpression のインスタンスを繋ぐことで再帰構造を表現することが可能となる。

s

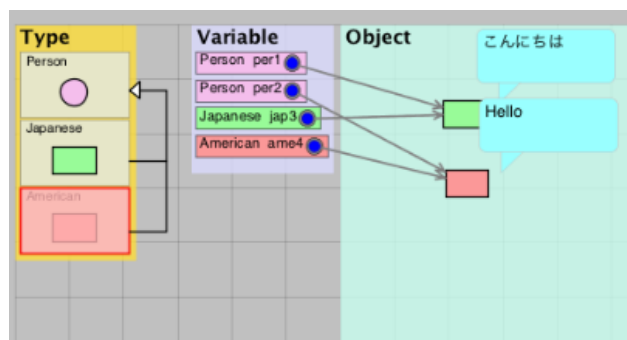


図 6 同じメソッドを呼び出したとき

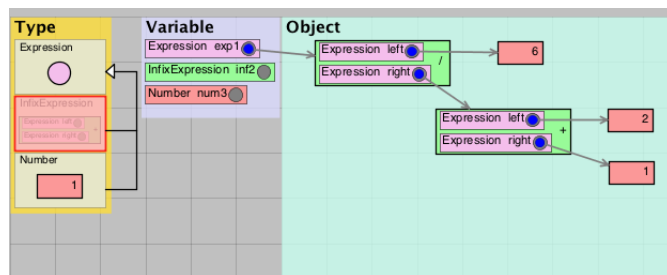


図 7 再帰構造を表現したとき

4. 実験方法

本章では、enPoly の評価実験の方法について述べる。

4.1 仮説

本実験では、学習者は、enPoly を用いて学習することによって、解けなかった課題や、さらに難易度の高い課題を解くことができるかどうかを検証する。

この仮説を検証するポイントを以下に示す。

- 学習者が、より応用的なプログラムのオブジェクト構造図*2を enPoly 上で作成することができるか
- 学習者が、作成したオブジェクト構造図を利用してプログラムを実装することができるか

4.2 実験課題

enPoly の評価のために実施する課題は、Java のプログラム作成課題である。この課題は、被験者が与えられたプログラムで使用している未定義・未実装のクラス・メソッドを作成することでプログラムが完成する。

以下に、実験に使用した例を示す。以下に示すのは課題 1 である (実験に使用した全問題文を付録に示す)。

問題 与えられるプログラム (Greeting.java) はあいさつするプログラムの一部分である。必要なクラスを追加し、コンパイルを通し、プログラムを完成させなさい。

注 1 このプログラムは日本人とアメリカ人がそれぞれの国の言語であいさつするプログラムである。実行結果

*2 与えられたプログラムのオブジェクトの構造を enPoly を用いて表現したもの

```

1  public class Greeting {
2
3      public static void main(String[] args){
4
5          Person taro = new Japanese("たろ
        う");
6          Person tom = new American("Tom");
7
8          taro.greeting();
9          tom.greeting();
10     }
11 }

```

図 8 被験者に与えるプログラム

は次のものになるようにすること。

実行結果 たろう：こんにちは
Tom：Hello

注 2 与えられたクラス (Greeting.java) を変更してはいけない。

被験者に与えるプログラムを図 8 に示す。

課題 1 を用いて、enPoly を利用した場合の被験者の解答プロセスを述べる。被験者は与えられたプログラムから、オブジェクト構造図を作成する。オブジェクト構造図を作成するとき、被験者はまず Person, Japanese, American の変数を宣言する。

同様に Person, Japanese, American のインスタンスも生成しようとするが、Person はインターフェイスなのでインスタンスが生成できないことに気付く。そして Japanese のインスタンスを生成し、Japanese の変数へ代入できることを確認する。同様に American のインスタンスを生成し、American の変数へ代入できることを確認する。そして Japanese のインスタンスが American 型の変数に代入できないことを確認し、Person 型の変数に代入できるかどうかを確認する。ここで、Person 型の変数に Japanese のインスタンスが代入できると気付く。プログラムを確認し、Person 型の変数に Japanese のインスタンスが代入されている状態が、課題プログラムの状態だとわかる。

同様に American も Person 型の変数に代入する。その後、それぞれのインスタンスに対して同じメソッドを呼び出し、結果が異なることに気付く。その結果、被験者はどのクラスにどのメソッドを実装すればいいか理解し、プログラムを作成する。

enPoly を使用しない場合、被験者は与えられたプログラムから必要なインターフェイス・クラス・メソッドを考えなければならない。または、オブジェクト構造図に相当するもの自分で作成する必要がある。このとき、必要な継承を作成することができなかつたり、if 文を用いてしまうとプログラムを作成することができない。

表 2 実験課題の概要

レベル	課題 1	課題 2	課題 3
例	あいさつ	中置記法 1 段	中置記法多段
難易度	戻り値なし	戻り値あり	戻り値あり・再帰あり
解答時間	15 分	20 分	30 分
行数	約 10 行	約 20 行	約 20 行

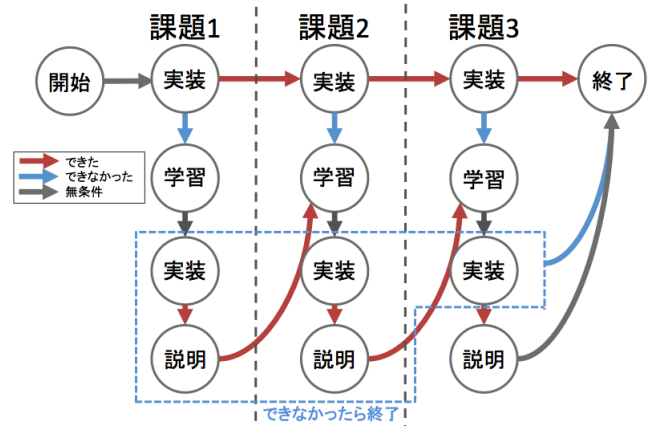


図 9 実験の手順

4.3 実験計画

実験は難易度別の 3 問について、enPoly を利用して課題を解く実験群と、enPoly を利用しない統制群を用いた計画で実施された。

4.3.1 課題の難易度と出題順

実験に用いた課題の概要を表 2 に示す。解答時間は目安の値である。課題は両群で、課題 1 から 3 まで昇順に解答する。課題 1 から順に難易度が上昇する。たとえば、課題 1 から課題 2 では戻り値が追加され、課題 2 から課題 3 では再帰が含まれる。難易度の異なる課題を用いることで、被験者がどのレベルでポリモーフィズムを理解しているかを判断することが目的である。

4.3.2 被験者の選定と振り分け方法

本実験は、大学・大学院でプログラミングを学習している 12 名を対象に行った。

被験者は実験群 6 名、統制群 6 名に振り分けた。ただし、統制群は、作成した課題がポリモーフィズムの理解度を測る指標として有効かどうかの検証も目的としたことから、ポリモーフィズムの理解度が高いと思われる学生が多く含まれている。

4.3.3 実験の手順

本実験における手順を図 9 に示す。実験手順の詳細を以下に述べる。

- (1) 開始：実験の説明 (5 分)
- (2) 実装：課題の実装を行う。課題ができた被験者は次の課題に進む。できなかった被験者は「学習」プロセスに進む。
- (3) 学習：実験担当者が課題の解説を行い、ポリモーフィズムについて学習する。

表 3 実験結果（実験群）

被験者	課題 1		課題 2		課題 3		level
	無	有	無	有	無	有	
A (IS, B3)	○	—	×	○	—	○	1 → 3
B (CS, B3)	×	○	—	○	—	○	0 → 3
C (CS, B3)	×	○	—	○	—	○	0 → 3
D (CS, B3)	×	○	—	○	—	○	0 → 3
E (CS, B4)	×	○	—	○	—	○	0 → 3
F (IS, B4)	×	○	—	△	—	—	0 → 1.5

表 4 実験結果（統制群）

被験者	課題 1		課題 2		課題 3		level
	無	有	無	有	無	有	
G (CS, M2)	○	—	—	—	○	—	3 → 3
H (CS, M2)	○	—	—	—	○	—	3 → 3
I (IS, M2)	○	—	—	—	○	—	3 → 3
J (CS, B4)	×	—	—	—	×	—	0 → 0
K (CS, B4)	×	—	×	—	×	—	0 → 0
L (CS, B4)	×	—	×	—	×	—	0 → 0

(4) 実装：学習を終えた被験者が、同じ課題を実装する。実装できた被験者は「説明」に進む。プロセスに実装できなかった被験者は実験終了となる。

(5) 説明：被験者が実装を終えた課題の構造を説明する。説明ができた被験者は次の課題に進むが、プロセスは学習から行う。説明ができなかった被験者は実験終了となる。

(6) 終了：アンケート [対象：実験群] (5 分)：enPoly に対する使い勝手や印象に関するアンケートを実施した。実験手順の「学習」プロセスは、実験群と統制群とで内容が異なる。実験群の「学習」プロセスは以下のように行う。

- enPoly を用いてオブジェクト構造図を作成する
- 作成したオブジェクト構造図で、メソッド呼び出しを行う

統制群の「学習」プロセスは以下のように行う。

- 実験担当者が講義を行う
- 講義内容は「ポリモーフィズムの定義」を説明し、「課題の構造」を UML のクラス図を用いて行う

4.3.4 実験環境と記録方法

本実験で使用した環境を以下に示す。

- Windows7 Professional 32bit
- Eclipse Java EE IDE for Web Developers. Version:Juno Service Release 1

本実験の記録方法を以下に示す。

- 実験中のコンピュータ操作のスクリーンキャプチャソフトによる録画記録
- 被験者と実験担当者を映したカメラによる録画記録
- 実験担当者による観察記録

5. 実験結果

5.1 課題の正否と解答時間

実験群の各課題における解答の正否、解答時間、被験者のレベルを表 3 に示す。統制群の各課題における解答の正否、解答時間、被験者のレベルを表 4 に示す。解答の正否は以下のような区分がある。

- ○：課題を解くことができた

- △：オブジェクト構造図まで作成できた

- ×：課題を解くことができなかった

- —：未実施

列に実施した課題、行に被験者をとる。列にある「level」は以下の基準で付ける。

- 課題 1 が解けた被験者をレベル 1、課題 2 が解けた被験者をレベル 2、課題 3 が解けた被験者をレベル 3 とする
- 課題 1 が解けなかった被験者はレベル 0 とする
- 矢印の前後で enPoly を使用する前のレベルとして事前レベル、enPoly を使用した後のレベルとして事後レベルと表記する

被験者の属性は学年 (B：学部生, M：院生) と本大学のコースを記載してある (CS・IS)。CS はコンピュータを構成する基本的要素 (ハードウェア、ソフトウェア、ネットワーク、データベース) の基礎原理の理解を目標としており、IS は情報システムを支える理論的な知識を学び、現実社会で役立つ情報システムの計画・設計・開発・運用・評価やそれらの管理を行うことを目標としている。列にある「有無」は enPoly の有無を示している。level の列にある矢印は前後でそれぞれ事前レベル・事後レベルを表している。

表 3 より、実験群の平均事前レベルは約 1.67 (小数第三位四捨五入) であり、平均事後レベルは 2.75 であった。統制群は全員事前レベルからの変化は見られなかった。このことから、実験群は enPoly を用いることで、解くことができなかった課題を解くことができ、さらに難易度の高い課題を解くことができたとわかる。

5.2 個別の解答プロセスの分析結果

実験群の個別の解答プロセスについての質的分析を行い、筆者らが想定したプロセスによる解答が行われているかを検証した。それぞれの事例について、以下に述べる。

5.2.1 継承を理解できたケース (被験者 A, 課題 2)

被験者 A は、オブジェクト構造図を作成するときに、全ての変数を作成し、同様に全てのインスタンスを生成しようとした。ここで、Expression (インターフェイス) はイン

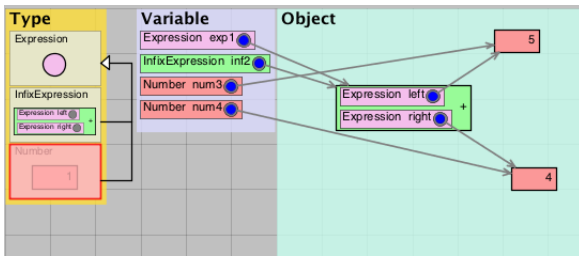


図 10 被験者 A の例

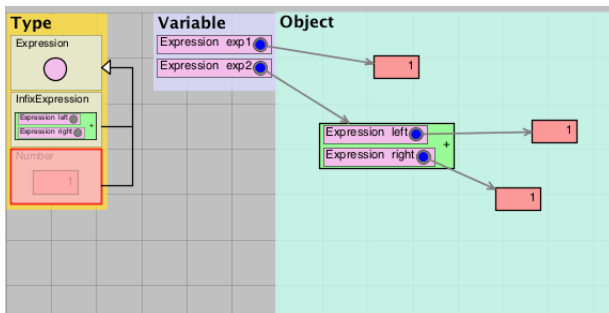


図 11 被験者 C の例

スタンスを作成できないことに気付いたと考えられる。生成したインスタンスは、InfixExpression は InfixExpression 型の変数に、Number は Number 型の変数に代入した。その後、InfixExpression のインスタンスを Number 型の変数に代入しようとしたが、代入することができないことに気付いた。同様に Number のインスタンスを InfixExpression 型の変数に代入しようとした。

被験者 A は、アンカーをドラッグしている最中に Expression 型の変数の上を通過し、アンカータブの色が青色から赤色に変化するのを確認した。ここで、被験者 A は Expression 型の変数に InfixExpression のインスタンスが代入できることに気付いたと考えられる。そして、被験者 A は Expression 型の変数に Number のインスタンスが代入できるかどうか確認した。代入できることを確認した後、しばらく Expression 型の変数にインスタンスを繋いだり外したりする動作を繰り返した。

InfixExpression 型の変数にインスタンスをつなぎ、メソッド (getValue()) を呼び出した。Number も同様に行った。そして、InfixExpression のインスタンスが持っている Expression 型の変数に Number のインスタンスを代入することで、オブジェクト構造図を作成させた。完成したオブジェクト構造図を図 10 に示す。

オブジェクト構造図に対して、2 回 getValue() を呼び出した。そして、InfixExpression が持っている getValue() と交互に呼び出していた。このとき、Expression と InfixExpression が持っているメソッドの違いを確認していたのだと考えられる。その後、数字や演算子を変更して getValue() の呼び出しを繰り返していた。4 通りほど実行した後、プログラムの実装に移っていた。

被験者 A はプログラムを実装するとき、まずインターフェイスである Expression を作成し、そこに必要なメソッド (getValue()) を定義した。そして InfixExpression クラスを作成し、Expression を継承させた。さらに Number クラスを作成し、同様に Expression を継承させた。ここで、一旦 enPoly に戻り、もう一度 getValue() の呼び出しを行っていた。その後、Number クラスのコンストラクタ・getValue() を実装した。同様に InfixExpression クラスのコンストラクタ・getValue() を実装し、プログラムが完成した。

被験者 A のように、オブジェクト構造図を作成する前に、どの変数にどのインスタンスが代入できるかを調べている例は、他に 3 例存在した。この 3 例すべてにおいて、オブジェクト構造図を作成するまえに、変数の代入を吟味していることが確認された。

5.2.2 プログラムの誤りを修正したケース (被験者 C, 課題 2)

被験者 C は被験者 A と違い、変数の代入を吟味せずに、そのまま Expression (インターフェイス) に InfixExpression と Number のインスタンスを代入した。そして、オブジェクト構造図が完成した。完成したオブジェクト構造図を図 11 に示す。その後、メソッド (getValue()) を呼び出した。そして数値を変えて、getValue() を呼び出した後、プログラムの実装に移っていた。

被験者 C はプログラムを実装するとき、まずインターフェイスである Expression を作成した。ここで一旦 enPoly に戻り、getValue() を呼んでプログラムの動作を確認した。enPoly のアニメーションを確認することで、メソッドの動作を把握したと考えられる。そして Expression にメソッドを定義し、クラスの実装に移った。まず被験者 C は、InfixExpression クラスを作成し、Expression を継承させた。enPoly を確認しつつ、コンストラクタを作成した。このとき、被験者 C はコンストラクタの引数を誤っているが、そのまま実装を続けていた。そして同クラスに getValue() を実装した。その後、Number クラスを作成し、Expression を継承させた。InfixExpression クラスと同様に、コンストラクタを作成し、getValue() の実装を行った。

enPoly で getValue() を呼び出し、InfixExpression クラスのコンストラクタの修正を行い、プログラムが完成した。ここで、被験者 C は InfixExpression のインスタンスが持つ Expression という変数を利用することに気付いたと考えられる。

被験者 C のように、プログラム実装中に enPoly を使い、プログラムの誤りを修正した例は、全ての被験者で確認された。

5.3 被験者の意見

被験者の方からいただいた意見を以下に示す。

- メソッド呼び出しを行うと、アニメーションが流れるので、どのメソッドがどんな機能を持っているかわかりやすい
- クラス図のように継承が表現されているため、インターフェイスとクラスの関係がわかりやすい
- どのインスタンスが生成されているかを把握できた
- 再帰構造を enPoly 上で作成することができ、さらに実行結果を見ることができ、自分が抱えているイメージとの差分がわかる

これらの意見から、筆者らが想定したように enPoly を使ってもらうことができたことがわかった。

6. 考察

本章では、実験結果を検証し、enPoly の目標達成度と適用範囲について考察を行う。

6.1 enPoly の有効性

被験者のレベル上昇度^{*3}では、実験群の全ての被験者が統制群を上回っている。解答プロセスの分析結果からもわかるように、筆者らが想定したプロセスによる解答が行われていた。これらを理由として、筆者らはポリモーフィズムの学習における enPoly の有効性を主張する。

この結論の妥当性の脅威として、被験者の質が挙げられる。本実験では、実験群の被験者にプログラミング経験が豊富な学生が多く、プログラミングが苦手、もしくはあまり経験がない被験者が少なかった。この点について、統制群に、実験群とプログラミング経験が同等の被験者がいること、その被験者のレベルが上がらなかったことを根拠として、上記の脅威は棄却され则认为る。

enPoly がポリモーフィズムの概念理解を促すかどうかについて考察する。統制群の実験結果より、ポリモーフィズムの理解度が高い学生は課題を全て解くことができた。つまり、これらの課題はポリモーフィズムの概念を理解していれば解くことができる課題であったと考えられる。実験群の多くは、enPoly を利用することでこれらの課題を解くことができた。したがって、enPoly はポリモーフィズムの概念理解を促す有効性を主張する。

6.2 今後の課題

現在、enPoly は、課題に即したクラスとインターフェイスのみが実装されている。しかしながら、ポリモーフィズムは様々な場面で適用されるため、今回作成した課題以外のクラスとインターフェイスが簡単に実装されるべきである。そこで、今後の課題として、プログラムからクラスとインターフェイスを自動生成する機能の開発を挙げる。この機能を実装することによって、教師が任意の課題を作成

したいとき、すぐに enPoly を活用することができる。学生では、自分が作成したプログラムのデバッグを目的として、enPoly に読み込ませ、自分が考えていた構造と実際のプログラムの構造との差分を確認することができると考えられる。

7. まとめ

オブジェクト指向言語を扱うために重要な概念の一つとして「ポリモーフィズム」が挙げられる。本研究では、オブジェクトの動的振る舞いとインターフェイスの概念を追加することで、ポリモーフィズムの学習を支援するシステム：enPoly を開発した。

enPoly の有効性を検証するために、学生 12 名を対象として、enPoly を学習する実験群 6 名と、講義を聞いて学習する統制群の 6 名による対照実験を行った。その結果、被験者の事前レベルと事後レベルの上昇度において、実験群の全ての事例が統制群の全ての事例を上回っていることが確認された。さらに、実験において想定された解答プロセスを被験者が行い、正答した事例が 5 例存在した。以上をもって enPoly の有効性を示した。

謝辞 三浦元喜さんに、AnchorGarden のソースコードの提供いただきました。感謝します。

参考文献

- [1] Bertrand Meyer[著], 酒匂寛・酒匂順子[訳].: "オブジェクト指向入門", (株) アスキー, 1990
- [2] John Minor Ross.: "Polymorphism in decline?", Journal of Computing Sciences in Colleges, Vol.21, No.2, pp.328-334, 2005
- [3] 三浦元喜, 杉原太郎, 國藤進.: "オブジェクト指向言語における変数とデータの関係を理解する為のワークベンチ", 情報処理学会論文誌, Vol.50, No.10, pp.2396-2408, 2009
- [4] Dan Ingalls, Ted Kaehle, John Maloney, Scott Wallace and Alan Kay.: "Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself", OOPSLA '97, pp.318-326, 1997
- [5] Michael Kolling, John Rosenberg.: "Object first with Java and BlueJ", SIGCSE '00, pp.429, 2000
- [6] Alkazemi.B.Y., & Grami.G.M.: "Utilizing BlueJ to Teach Polymorphism in an Advanced Object-Oriented Programming Course", Journal of Information Technology Education, Vol.11, pp.271-281, 2012
- [7] Tarsem S. Purewal, Jr. and Chris Bennett.: "A framework for teaching polymorphism using game programming", Journal of Computing Sciences in Colleges Vol.22, No.2, pp.154-161, 2006
- [8] Paul Gestwicki, Bharat Jayaraman.: "Methodology and architecture of JIVE", SoftVis '05 Proceedings of the 2005 ACM symposium on Software visualization, pp.95-104, 2005
- [9] Mordechai Ben-Ari, Roman Bednarik, Ronit B. Levy, Gil Ebel, Andres Moreno, Niko Myller, Erkki Sutinen.: "A decade of research and development on program animation: The Jeliot experience", Journal of Visual Languages and Computing, Vol. 22, No. 5, pp. 375-384, 2011

^{*3} 事前レベルと事後レベルの差分

付 録

A.1 課題 1

問題 与えられるプログラム (Greeting.java) はあいさつするプログラムの一部分である。必要なクラスを追加し、コンパイルを通し、プログラムを完成させなさい。
注 1 このプログラムは日本人とアメリカ人がそれぞれの国の言語であいさつするプログラムである。実行結果は次のものになるようにすること。

実行結果 たろう：こんにちは

Tom：Hello

この課題のクラス図を図 A.1 に示す。被験者が実装するのは青いクラスである。このクラス図は被験者には与えない。

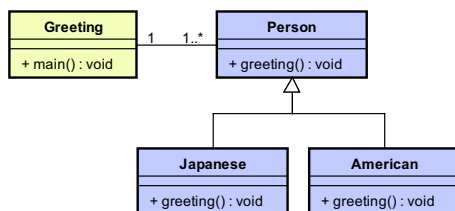


図 A.1 課題 1 のクラス図

A.2 課題 2

問題 与えられるプログラム (Calculator.java, SplitString.java) は計算するプログラムの一部である。必要なクラスを追加し、コンパイルを通し、プログラムを完成させなさい。

注 1 このプログラムは中置記法を解釈して計算 (四則演算) をおこなう計算機である。実行結果は次のものになるようにすること。

入力 5+4

実行結果 5+4 = 9.0

注 2 与えられたクラス (Calculator.java, SplitString.java) を変更してはいけない。

この課題のクラス図を図 A.2 に示す。被験者が実装するのは青いクラスである。このクラス図は被験者には与えない。

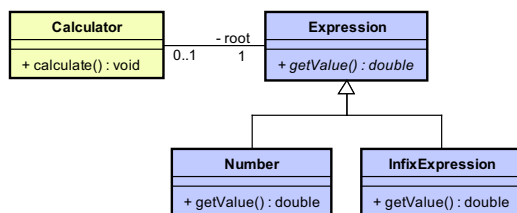


図 A.2 課題 2 のクラス図

A.3 課題 3

問題 与えられるプログラム (Calculator.java, RPN-Maker.java, BinaryTreeMaker.java) は計算するプログラムの一部である。必要なクラスを追加し、コンパイルを通し、プログラムを完成させなさい。

注 1 このプログラムは中置記法を解釈して計算 (四則演算) をおこなう計算機である。実行結果は次のものになるようにすること。

入力 5+4*3-2/1

実行結果 5+4*3-2/1 = 15.0

注 2 与えられたクラス (Calculator.java, RPNMaker.java, BinaryTreeMaker.java) を変更してはいけない。

この課題のクラス図を図 A.3 に示す。被験者が実装するのは青いクラスである。このクラス図は被験者には与えない。

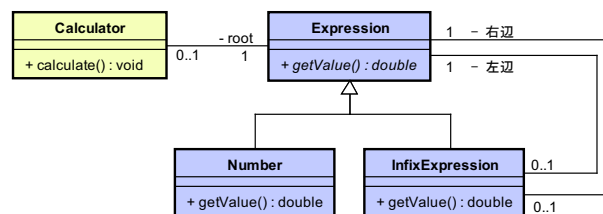


図 A.3 課題 3 のクラス図