BlockEditor Hinoki:ビジュアル-Java相互変換技術を利用したオブジェクト指向プログラミング教育の提案

大畑 貴史[1,a)]　松澤 芳昭[2,b)]　桐山 伸也[2,c)]　酒井 三四郎[2,d)]

概要：Java を用いたオブジェクト指向プログラミングの授業では，構文エラーによってオブジェクト指向の概念学習が阻害されてしまう学習者が観察される．本研究では，ビジュアルプログラミング言語からシームレスにテキスト記述型言語に移行可能なビジュアル-Java 相互変換システム「BlockEditor」のオブジェクト指向拡張版（コードネーム：Hinoki）を提案する．BlockEditor Hinoki の特徴は，1) カプセル化，継承，ポリモーフィズムの表現能力を持ったブロック型言語の設計，2) 学習者が設計したクラスオブジェクトをブロック言語で利用可能，3) オブジェクト指向構文の Java とブロック型言語の相互変換に対応，である．プログラミング入門教育を修了した大学２年生向けのオブジェクト指向プログラミングコースで BlockEditor Hinoki を利用した授業を実施中である．当該年度の言語選択状況，および課題，提出状況，プログラムの質の年度比較を通じてその本システムの有用性を検証する．

BlockEditor Hinoki: A Proposal of Object-Oriented Programming Education Using Visual to Java Through Mutual Language Translation

Takashi Ohata [1,a)]　Yoshiaki Matsuzawa [2,b)]　Shinya Kiriyama [2,c)]　Sanshiro Sakai [2,d)]

Abstract: In an object-oriented programming course using Java language, it is not infrequent for teachers to observe students who are struggling with object-oriented concepts due to focusing fixing syntax errors. In this research, we have developed Hinoki version of BlockEditor. The original BlockEditor provides the mutual language translation technology between visual (block) language and Java, and we extended it to support object-oriented syntax in Java. BlockEditor Hinoki has following distinctions:(1) block language to support object-oriented programming such as encapsulation, inheritance, and polymorphism, (2) the block definition generator that dynamically produces blocks for user-designed classes, and (3) the mutual language translation between Java and block that covers syntax which is related to object-oriented functions in Java. In an ongoing project, we are conducting an experimental study for this system in our object-oriented programming course.We hypothesized that the seamless migration block to Java would be observed in this study. The usefulness of the design would be verified by comparison for the successful rate of the programming tasks and its quality between the control and the experimental group.

1. はじめに

オブジェクト指向プログラミング（OOP）の授業では，オブジェクト指向の概念理解が重要であるが，オブジェクト指向の概念学習は困難である．オブジェクト指向の概念学習が難しい要因として，1）クラスとインスタンスの違い，2）継承やポリモーフィズムの概念，3）クラス間のメッセージングによる処理の複雑化，4）構文エラー，などが考えられる．特に構文エラーが原因でオブジェクト指向の概念学習が困難である学習者は，オブジェクト指向の概念学習以前に，構文の学習や構文エラーの修正に注力してしまうため，オブジェクト指向の本質的な理解が阻害されてしまうことが考えられる．ビジュアルプログラミング言語（VPL）では，構文エラーに悩まされることなくオブジェ

[1] 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University
[2] 静岡大学情報学部
Faculty of Informatics, Shizuoka University
[a)] ohata@sakailab.info
[b)] matsuzawa@inf.shizuoka.ac.jp
[c)] kiriyama@inf.shizuoka.ac.jp
[d)] sakai@inf.shizuoka.ac.jp

クト指向の概念の学習が可能である．しかしながら，将来的にテキスト記述型言語を利用することを踏まえて，テキスト記述型言語での記述方法を学習したい学習者にとっては，VPL からテキスト記述型言語への移行が難しい．

VPL とテキスト記述型言語の相互変換によるプログラミング入門教育支援システム [1] が松澤らにより提案されている．この相互変換システム（BlockEditor）を利用することで，初学者は構文エラーに悩まされることなく，かつシームレスにテキスト記述型言語へ移行することが可能である．

本研究の目的は，構文エラーによってオブジェクト指向の学習が阻害されている学習者に対しオブジェクト指向の学習に注力させ，かつシームレスな言語移行を提供することである．これらの学習者に対して，BlockEditor によるビジュアルプログラミングと，相互変換が有用であるという仮説を立て，BlockEditor のオブジェクト指向構文へ対応と，BlockEditor を利用した OOP の授業を実施した．当該年度の言語選択状況，および課題，提出状況，プログラムの質の年度比較を通じて，当システムの有用性を検証する．

2. オブジェクト指向プログラミングの授業の問題点

本章では，本学の OOP の授業についての説明と，その問題点について述べる．

2.1 本学のオブジェクト指向プログラミングの授業

本学の OOP の授業は，IS（Information System）を専攻する学科の学生に対して行われている．文化系の学科（情報社会学科）の学生と理科系の学科（情報科学科）の学生を対象にした授業である．受講者は理科系の学生と文化系の学生がどちらも同じ割合である．OOP の授業は以下のような構成になっている．

- 構造化プログラミングの復習（4 週）
- アルゴリズムの設計と効率（4 週）
- OOP の基本（7 週）

授業では，松澤らによるオブジェクト指向プログラミングの教材 [2] を，この授業用に再編した教材を利用している．

構造化プログラミングの復習では，構造化プログラミングの復習を行いながら，HCP チャートを用いてプログラムを抽象的に表現することでプログラムの抽象化能力を身につけるような学習を行う．アルゴリズムの設計と効率では，代表的な探索アルゴリズムやソートアルゴリズムについて，そのアルゴリズムを HCP チャートを記述することで抽象的に理解し，各アルゴリズムの性能に関して学習を行う．オブジェクト指向では，クラスとインスタンス，オブジェクトの状態遷移，継承，ポリモーフィズム，リスト構造に関してシューティングゲーム作成を通じて学習を行う．

図 1: BlockEditor Hinoki の外観

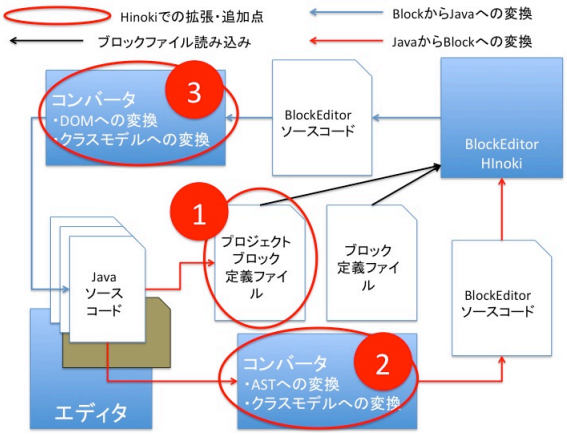

図 2: BlockEditor Hinoki のアーキテクチャ

2.2 授業での問題点

授業担当教員から現在の授業での問題点についてヒアリングを行った結果から得られた問題点について述べる．

2.2.1 躓きによる課題の蓄積

構文的な躓きや，オブジェクト指向概念理解の躓きによって課題が蓄積されてしまう学習者が居ることが問題として挙げられた．授業ではシューティングゲームを題材として扱っており，ゲームを毎週拡張していくことを課題にしている．この課題は授業の回ごとに独立していないため，毎週課題をこなさなければ次回の課題を行うことができない．そのため，一度躓いて課題を溜めてしまった学習者は，次の課題に取り込むことができず，どんどん課題が積み重なってしまう．

2.2.2 他クラスのインスタンス変数の直接参照

カプセル化のされてないクラスの作成が問題として挙げられた．オブジェクト指向の考え方を踏まえると，他のオブジェクトのインスタンス変数を直接参照することは好ましくない．カプセル化しない学習者は，直接他のオブジェクトのインスタンス変数を参照する癖がついてしまい，カプセル化されたクラスの設計や，カプセル化の利点，概念学習が阻害されてしまう．

3. 先行研究

Scratch[3] や Alice[4] は，文法エラーや構文エラーを回避でき，アルゴリズムの構築に注力した学習が可能である．これらの言語では基本的な OOP の概念があるが，言語変換能力が存在しない．従ってこれらを用いた学習者はテキスト記述型言語への移行の難しさが考えられる．

VPL からテキスト記述型言語への移行の難しさに関しては Ian らによって議論されている [10]．Ian らは,Scratch から Greenfoot，Alice から Greenfoot，または Scratch から Alice へ移行し，Alice から Greenfoot へ移行するように，VPL からオブジェクト指向の学習が可能な Greenfoot への移行が望ましいことを述べている．

VPL からテキスト記述型言語への移行に着目した研究としては Erik らの研究がある [5]．この研究では，JubJub と呼ばれるシステムに，VPL で実装したプログラムを Java に変換可能である．ブロックから多言語へ変換可能な VPL としては，Google Blocky[6] がある．Google Blocky はブロックから JavaScript, python, xml などへ変換することが可能である．JubJub や Google Blocky はブロックからテキスト記述型言語への 1 方向のみの変換である．さらにこれらの環境では，オブジェクト指向基本的な概念であるカプセル化，継承，ポリモーフィズムなどを表現することは不可能である．

松澤らによる BlockEditor では，ブロックを組み合わせてプログラムの実装が可能であり，かつブロックと Java の相互変換が可能であった．BlockEditor を用いたプログラミング入門教育 [1] では，学習者がシームレスにブロックから Java へ移行可能であることが示された．しかし相互変換のカバー範囲はあくまで手続き処理の範囲内であり，オブジェクト指向構文の相互変換には対応していない．

主原らは OpenBlocks[7] を利用し，C や Java，LOGO への変換が可能なプログラミング学習環境 oPEN を開発している [8]．oPEN と BlockEditor の違いは相互変換への対応と，他言語変換への対応，ブロックの日本語表現に違いがある．oPEN は相互変換には対応しておらず，ブロックから C や Java,Logo への変換に対応しており，またこの 3 つの言語への変換が可能である．しかしプログラミング入門で利用するには相互変換に対応しなければ，ビジュアルプログラマがテキスト記述型言語になれることが難しい．oPEN と BlockEditor ではブロックの日本語に関しても思想の違いがある．oPEN は PEN に対応した日本語表現であるのに対し，BlockEditor は自然に読むことができる日本語表現をすることで，処理の概念を分かりやすくするように設計している．

Raptor[9] では，VPL でクラスの設計を行うことができる．Raptor の特徴は，フローチャートと UML を組み合わせてプログラムを視覚的に記述することが可能であり，かつデバッグ機能でフローチャートで処理を追える点である．しかしクラスの設計に UML の知識が必要になることを踏まえるとオブジェクト指向入門レベルで利用することは難しいと考えられるほか，こちらは言語変換機能を持たないため，テキスト記述型言語への移行が難しい．

4. BlockEditor Hinoki

4.1 設計目標

本研究の目的は，OOP の授業において，学習者が構文エラーによるオブジェクト指向の概念学習が阻害されてしまうという問題を解決できる環境を開発することである．OOP の授業で利用できる VPL とテキスト記述型の相互変換能力があり，学習者自身が設計したクラスを VPL で利用できることが設計目標である．

4.2 外観

BlockEditor Hinoki の外観を**図 1** に示す．「ファクトリ」からブロックを取り出し，「キャンバス」に設置することでプログラムの実装が可能である．Java とブロックの相互変換も可能である．

4.3 アーキテクチャ

アーキテクチャを**図 2** に示す．BlockEditor Lauan からのアーキテクチャの変更点は，以下の 3 点である．

- Block へ変換する Java ファイルの存在するパッケージ内のクラスを BlockEditor 起動時に毎回解析し，「プロジェクトブロック定義ファイル」を作成する（図 2 の 1）
- 式や文の抽象構文木 (AST) への変換の拡張（図 2 の 2）
- ドキュメントオブジェクトモデル (DOM) への変換のオブジェクト指向構文への拡張（図 2 の 3）

Java のソースコードを Block へ変換は次の手順で行っている．

(1) Block へ変換する Java ファイルの存在する Java プロジェクト内のフォルダを BlockEditor 起動時に毎回解析し，プロジェクトブロック定義ファイルを作成する
(2) ソースコードを解析し，AST へ変換し，DOM へ変換する
(3) BlockEditor のソースコードに変換する
(4) ソースコードを読み込み，ブロックへ変換する

ブロックから Java への変換は，次の手順で行っている，

(1) ブロックを解析し，BlockEditor のソースコードに変換する
(2) BlockEditor のソースコードを AST へ変換し，DOM へ変換する
(3) Java ソースコードに変換する

4.3.1 動的なブロックモデルの作成

図 3 に動的にブロックモデルを作成するアーキテクチャ

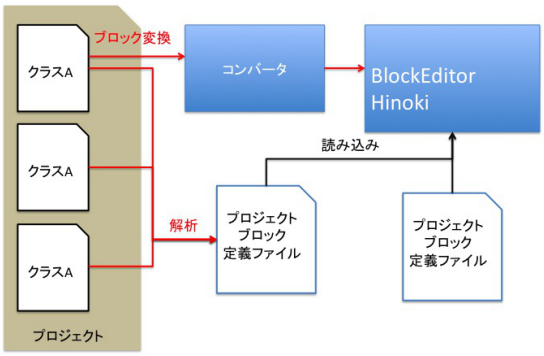

図 3: 動的ブロック作成のアーキテクチャ

の詳細を示す．基本的なブロックモデル（変数や，分岐ブロックなど）は，「ブロック定義ファイル」に記述されている．Hinoki では，Block へ変換する Java ファイルの存在する同一プロジェクト内の Java ファイルに記述されたクラスの継承関係や，メソッドを動的に解析し，学習者の作成したクラスのブロックモデルを定義した「プロジェクトブロック定義ファイル」を作成している．

この 2 つのファイルを読み込むことで，基本的なブロックと学習者自身が設計したクラスオブジェクトのブロックを利用可能にしている．

プロジェクト内の Java ファイルを動的に解析し，作成されるブロックを次に示す．

- クラス型のローカル，インスタンス変数ブロック
- クラス型の引数ブロック
- クラス型への型変換ブロック
- クラスのメソッド実行ブロック
- クラスのインスタンス生成ブロック

これらのブロックは解析後，ファクトリへと追加される．

4.3.2 オブジェクト指向構文への相互変換能力の拡張

オブジェクト指向構文の Block から Java へ変換する際の AST への変換，ブロックから Java へ変換する際の DOM への変換に対応した．これにより，VPL で表現できるプログラム構成要素が増え，OOP に対応したプログラムの表現が可能である．Hinoki バージョンで新たに相互変換に対応した構文要素は，Constructor，SuperIdentifer，ThisIdentifer，ArrayType，NullLiteral，Typeof，Foreach である．

4.4 eclipse プラグイン化

本学の OOP の授業では統合開発環境である eclipse を利用している．授業で BlockEditor を用いるに当たり，本システムの eclipse プラグイン化を行った．Eclipse4.2 以降で利用することが可能である．

5. ブロックの仕様

BlockEditor は，ブロックに記述されている日本語を自然

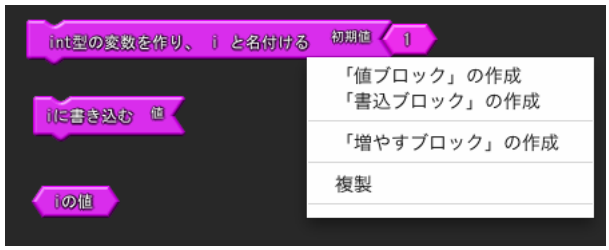

図 4: ローカル変数ブロックの例

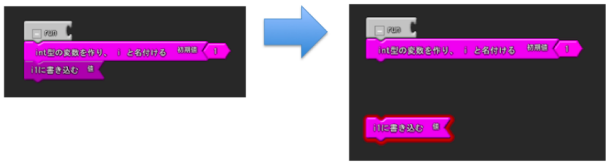

図 5: スコープチェック機能の例

言語として処理の概念を理解しやすいように設計している．本章は，ブロックを示しながら各ブロックの仕様について説明する．例に取り上げるブロックは，以前の BlockEditor からある基本的な処理を行うブロック（変数宣言，分岐，繰り返しなどのブロック）と，BlockEditor Hinoki で新たに対応したブロックである．それらを利用したクラスのキャンバスの全体の例も示す．

5.1 基礎的なブロック

基本的なプログラム構成要素を表すブロックの仕様について本節で述べる．

5.1.1 ローカル変数

図 4 にローカル変数ブロックの例を示す．ローカル変数ブロックは右側の初期値と書かれた凹みに値ブロックを結合することで変数の初期値を設定することができる．図 4 のコンテキストメニューは変数宣言ブロックを右クリックすることで表示される．このメニューから値の参照と書き込みを行うブロックを作成することができる．右クリックで作成可能にすることで，ビジュアルプログラミング特有の煩雑さを削減することができる．

さらに BlockEditor には**図 5** のように，ローカル変数スコープの範囲外ブロックを結合しようとした場合に，そのブロックを弾くアニメーションを行う機能がある．学習者が BlockEditor でのビジュアルプログラミングでエラーのあるソースコードを作成させないためである．

5.1.2 分岐・繰り返しブロック

分岐・繰り返しブロックの例を**図 6** に示す．図 6 の分岐ブロックは，「1 等しい i 値」+「かどうか調べて」+「真のとき/偽のとき」と読むことができる．

5.1.3 メソッド定義ブロックと実行ブロック

メソッド定義ブロックの例を**図 7** に示す．メソッド定義ブロックは，図 7 のようにブロックの右側に引数ブロックを結合することで，引数を定義することが可能になってい

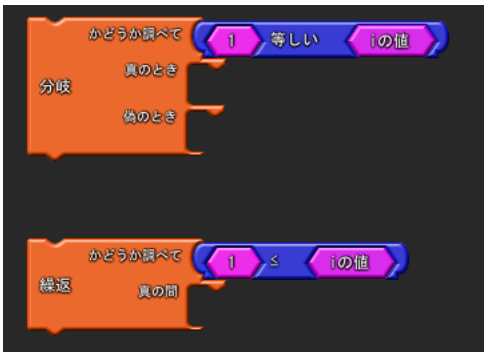

図 6: 分岐・繰り返しブロックの例)

図 7: メソッド定義ブロックとその実行ブロックの例

図 8: インスタンスのメソッド呼び出しブロックの例

る．メソッドのアクセス修飾子は public のみを扱う．これは OOP の入門レベルにおいてはメソッドのアクセス修飾子を学習者が意図的に指定させる必要性はないと考えたためである．

5.1.4 ドット演算子

図 8 は ArrayList クラスのインスタンスの clear メソッドと，getSize メソッドの呼び出しブロックの例である．図 8 の「が」や「の」と書かれたブロックがドット演算子に対応するブロックである．「中身をすべて削除する」と書かれたブロックが clear メソッド，「リストの大きさ」と書かれたブロックが size メソッドに対応している．

返り値が void 型のメソッドは「が」というラベルの貼られたブロックを，それ以外の型の返り値のあるブロックは「の」というラベルの貼られたブロックを利用するように設計した．

5.1.5 インスタンスメソッド実行ブロックの作成

インスタンスメソッドを実行するブロックは，変数の宣言ブロックを右クリックすることで作成することができる．**図 9** に実際のコンテキストメニューと作成されたメソッド実行ブロックの例を示す．図 9 ではタートルクラスで利用可能なメソッドが表示されており，亀が前進するメソッドの実行ブロックが作成されている．

図 9: インスタンスメソッド実行ブロックの作成の例

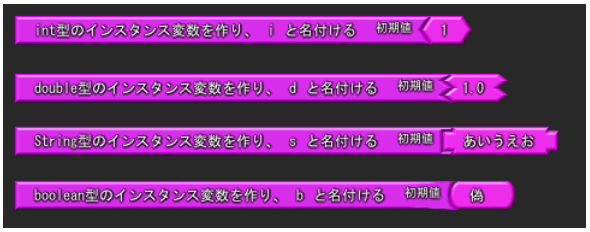

図 10: インスタンス変数ブロックの例

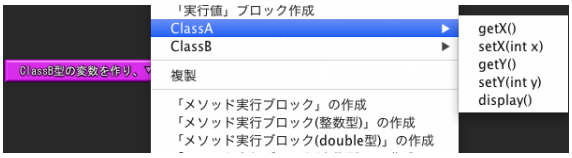

図 11: 継承メソッド一覧の例

5.2 BlockEditor Hinoki で新たに対応したブロック

BlockEditor Hinoki で新たに追加したブロックの仕様を本節で述べる．this,super キーワード，NullLiteral，Foreach に関しては省略する．

5.2.1 インスタンス変数

実際のインスタンス変数を**図 10** に示す．BlockEditor では private 変数のみを扱い，そのアクセスには getter メソッドと setter メソッドによるメッセージングで行うようした．これは，2.2.2 節で問題となっていた他のオブジェクトのインスタンス変数の直接参照をさせないためである．

5.2.2 継承メソッドの利用

図 11 のように，ポップアップメニューでメソッドをクリックすることで継承メソッドブロックの作成が可能である．図 11 では ClassA を継承した ClassB の利用可能なメソッドを表示している．

5.2.3 配列・リスト

配列やリストのブロックを新たに追加した．**図 12** に配列の例を，**図 13** にリストブロックの例を示す．リストは右クリックにより利用可能なメソッドの実行ブロックが作成可能である．配列は配列変数定義ブロックを右クリックすると，要素の代入を行うブロックと，要素の参照を行うブロックの作成が可能である．

図 12 中の「新しく int[] を作る」というブロックには，サイズと書かれた凹みがある．この凹みに配列のサイズと

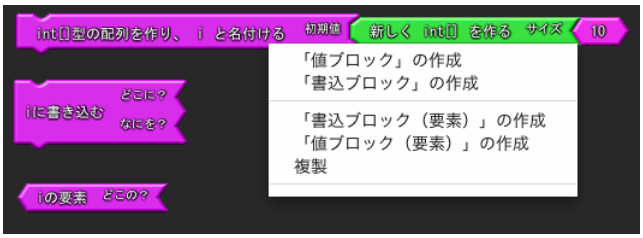

図 12: 配列ブロックの例

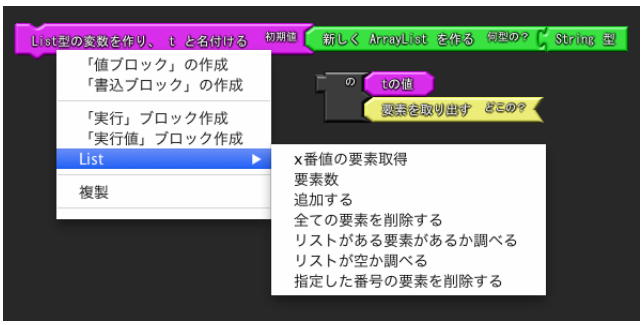

図 13: リストブロックの例

なる値ブロックを結合する．配列の書き込みブロックには凹みが 2 つあり，「どこに？」と書かれた凹みには書き込む対象の配列のインデックスとなるブロックを，「何を？」と書かれた凹みには書き込む値となるブロックを結合する．

5.2.4 コンストラクタ

コンストラクタはメソッド定義ブロックと同様の形と色をしており，引数も利用可能である．メソッド定義ブロックと異なる点は，ラベルが「コンストラクタ」で固定されている点である．

5.3 キャンバス

キャンバスの仕様と，具体的な例を本節に示す．

5.3.1 キャンバス

キャンバス（図 1 の 1）は，ひとつのクラスに対応している．学習者はキャンバス上にインスタンス変数ブロックや，メソッドブロックなどを設置することでクラスの設計が可能である．キャンバスに設置されたブロック位置情報をソースコードに保持し再起動時には位置情報を読み取り同じ位置に復元されるようにもしている．

5.3.2 BlockEditor Hinoki のキャンバス全体の例

BlockEditor のキャンバス全体の例を**図 14** に示す．このキャンバスは，ソースコード 1 をブロックに変換している．図 14 では，銀行口座管理クラスをブロックで表現している．銀行口座管理クラスは，インスタンス変数として銀行口座クラス（BankAccount）のリストを持っており，3 つのメソッドが定義されている．

6. 評価と今後の展望

BlockEditor は本学の OOP コースで利用されている 48 個の構文要素のうち 47 個に対応している．未対応な構文要素は Comment である．Comment は必ずしもプログラムを組む上で必要な構文要素でない．コメントの用途として処理の目的の記述や変数の抽象化，メソッドのアノテーションなどが挙げられる．このうち，処理の目的の記述は代替手段として抽象化ブロック機能が利用可能である．抽象化ブロックとは，ある一連のブロックの処理をまとめ，名前をつけることができるブロックである．

Java と比較して未対応な構文要素は GenericType や try catch,Throw などが挙げられる．すべての構文要素との対応は付録に示す．未対応な構文要素は本学の OOP の授業で利用されているものでなく，OOP の授業において必ずしも必要な構文要素でないと判断できる．

本学の OOP の授業の課題プログラムはすべて変換可能

ソースコード 1: 銀行口座管理クラスを Java に変換したソースコード

```
1  import java.util.ArrayList;
2
3  public class BankAccountManager {
4
5    private ArrayList<BankAccount> accounts = new
         ArrayList<BankAccount>();// @(50,
6                                       // 50)
7
8    public void アカウントを追加する(String 名義人名) {
9      { // アカウントを生成し，追加する
10       accounts.add(new BankAccount(名義人名));
11     }
12   }// @(50, 90) [open]
13
14   public BankAccount 検索する(String 名義人名) {
15     { // ユーザを検索する
16       { // for each
17         int counter = 0;
18         int endIndex = accounts.size();
19         while (counter < endIndex) {
20           BankAccount account = accounts.get(counter);
21           if (名義人名.equals(account.名義人名を参照す
                 る())) {
22             return account;
23           }
24           counter = counter + 1;
25         }
26       }
27       return null;
28     }
29   }// @(709, 90) [open]
30
31   public void アカウントを削除する(String 名義人名) {
32     { // アカウントを削除する
33       BankAccount account = 検索する(名義人名);
34       accounts.remove(account);
35     }
36   }// @(41, 329) [open]
37
38 }
```

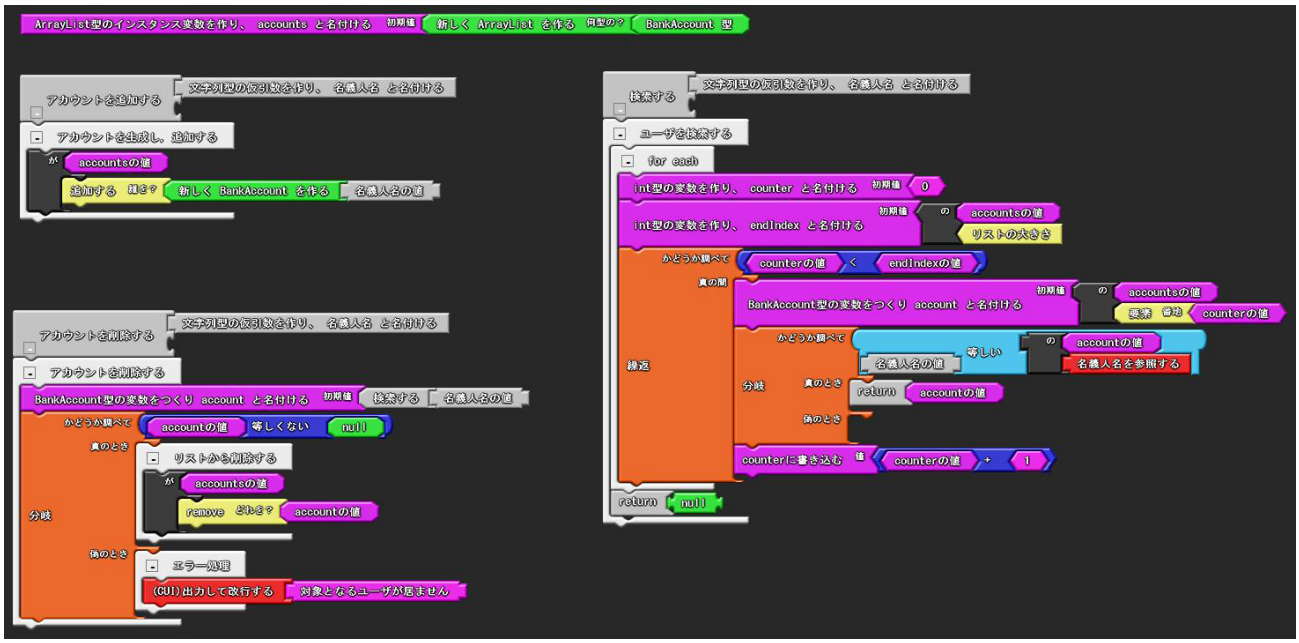

図 14: BlockEditor Hinoki のキャンバス全体の例（銀行講座管理クラス）

である．相互変換の可否は，Java から Block に変換した結果 A と，Block から変換した Java コードを Block に変換した結果 B を比較し，OOP 編のすべての課題プログラムで A と B が同じになることを確認した．これらのことから，Blockeditor は OOP の授業に利用可能な相互変換能力があると評価できる．

現在本学の学部 2 年生を対象とした OOP の授業で BlockEditor を利用している．今後はこの授業で提出されたソースコードの年度間の比較，分析を行う．プログラム作成履歴の分析には Programming Process Visualizer(PPV)[11] を利用する．PPV は，プログラミング過程をアニメーションで視覚化し，観察，分析することが可能である．これを用いて，2.2 節で挙げた問題点を解決出来たかどうかを検証している．

2.2.1 小節で挙げた問題に関しては，課題の提出率を昨年のものと比較し，期限内に課題を提出できる学習者が増えることを検証している．

2.2.2 小節で挙げた問題に関しては，PPV を用いて学習者のソースコードを分析し，public 変数やアクセス修飾子なしのインスタンス変数が減り，private のインスタンス変数が増えることを検証している．

その他，学習者の BlockEditor の利用推移や，ブロックで記述される処理などを分析を行う．本発表ではこれらの検証結果に関して報告を行う．

参考文献

[1] 松澤芳昭，保井元，杉浦学，酒井三四郎：ビジュアル-Java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価, 情報処理学会論文誌, No.55，No.1，pp.57-71(2014).

[2] 松澤芳昭，大岩元：情報系学生を対象としたオブジェクト指向までのプログラミング入門教育の実践と課題，情報教育シンポジウム (SSS2009)，pp199 - 206（2009）.

[3] Scratch Team Lifelong Kingdergarten Group MIT Media Lab/: *Scratch - imegine.program.share-*,http://scratch.mit.edu(2014.04.26).

[4] S Cooper, W Dann, R Pausch, C Mellon.: *ALICE: A 3-D TOOL FOR INTRODUC- TORY PROGRAMMING CONCEPTS.* Journal of Computing Sciences in CollegesVolume 15 Issue 5,pp107-116 (2000).

[5] Erik Pasternak.: *Visual programming pedagogies and integrating current visual programming language features.*, Master’s thesis, Carnegie Mellon University Robotics Institute Master’s Degree (2009).

[6] Google Inc.: *Blockly:a visual programming editor.*, Longman (2000).

[7] Ricarose Vallarta Roque: *Openblocks: An extendable framework for graphical block programming systems.* , Master thesis at MIT(2007).

[8] 主原 佑記，赤井 昭仁，中村 亮太，松浦 敏雄：OpenBlocks を用いたプログラミング学習用ソフトウェアの開発，情報処理学会研究報告． コンピュータと教育研究会報告 2014-CE-124(9)．pp1-7(2014).

[9] Martin C. Carlisle.: *Raptor: a visual programming environment for teaching object-oriented programming.*, Journal of Computing Sciences in Colleges archive Volume 24 Issue 4,pp275-281(2009).

[10] Ian Utting, Stephen Cooper, Michael Kling, John Maloney, Mitchel Resnick: *Alice, Greenfoot, and Scratch - a discussion.* ACM Transactions on Computing Education,Volume 10 Issue 4, November 2010 Article No. 17 (2010).

[11] 松澤芳昭，岡田健，酒井三四郎： Programming Process Visualizer：プログラミングプロセス学習を可能にするプロセス観察ツールの提案，情報教育シンポジウム 2012 論文集 (2012).

付　　録

A.1 BlockEditor Hinoki と対応する構文要素

構文要素と Java，BlockEditor，BlockEditor Hinoki，オブジェクト指向プログラミングコース（OOP コース）の一覧を示す．表中の記号 x は対応する列の要素が行の要素に対応していることを示す．△は他の要素で代替可能であることを示す．OOP コースは，本学情報学部のプログラミング入門を修了した大学 2 年生向けに行われている授業の対応を記述している．表では具象クラスとして実装されている Java の要素のみを掲載している．

表 A·1: 構文要素と使用する言語・BlockEditor・講義一覧（その 1）

要素	Java	BE	BH	OOP コース
Program	x	x	x	x
Block	x	x	x	x
Comment	x	△	△	x
VariableDefinition	x	x	x	x
VariableDefinitionList	x			
ClassDefinition	x	x	x	x
AnnotationDefinition	x			
EnumDefinition	x			
InterfaceDefinition	x			
NamespaceDefinition	x	x	x	x
FunctionDefinition	x	x	x	x
Constructor	x		x	x
StaticInitializer	x			
TypeConstrain	x	x	x	x
SuperConstrain	x	x	x	x
ImplementsConstrain	x			
ExtendConstrain	x	x	x	x
AnnotaionCollection	x			
ArgumentCollection	x	x	x	x
CaseCollection	x			
CatchCollection	x			
ExpressionCollection	x	x	x	x
GenericArgumentCollection	x			
GenericParameterCollection	x			
IdentiferCollection	x	x	x	x
ModifierCollection	x	x	x	x
ParameterCollection	x	x	x	x
TypeCollection	x	x	x	x
TypeConstrainCollection	x	x	x	x
Modifier	x	x	x	x
Annotation	x			
Parameter	x	x	x	x
Argument	x	x	x	x

表 A·2: 構文要素と使用する言語・BlockEditor・講義一覧（その 2）

要素	Java	BE	BH	OOP コース
GenericParameter	x			
GenericArgument	x			
Call	x	x	x	x
New	x	x	x	x
Property	x			
Indexer	x			
Identifier	x	x	x	x
VariableIdentifier	x	x	x	x
SuperIdentifier	x		x	x
ThisIdentifier	x		x	x
Cast	x	x	x	x
Type	x	x	x	x
ArrayType	x		x	x
GenericType	x			
NullLiteral	x		x	x
BooleanLiteral	x	x	x	x
CharLiteral	x			
FractionLiteral	x	x	x	x
StringLiteral	x	x	x	x
IntegerLiteral	x	x	x	x
ArrayLiteral	x			
BinaryExpression	x	x	x	x
TernaryExpression	x	x	x	x
UnaryExpression	x	x	x	x
BinaryOperator	x	x	x	x
UnaryOperator	x	x	x	x
Typeof	x		x	x
If	x	x	x	x
For	x	x	x	x
Foreach	x	x	x	x
While	x	x	x	x
DoWhile	x	x	x	
Switch	x			
Case	x			
Label	x			
Try	x			
Catch	x			
Synchronized	x			
Break	x	x	x	x
Continue	x	x	x	
Return	x	x	x	x
Throw	x			
Assert	x			
Import	x	x	x	x