シームレス性と文脈依存性を重視した視覚化機能を持つEclipseプラグインによるプログラミング学習支援システム

大城 正典[1,a)]　永井 保夫[1,b)]

概要：著者らは，プログラミング学習において適切な視覚化を学習者の理解を助ける有効な手段であると考え，Eclipse のプラグインという形で提案・実装を行ってきた．本研究では，初心者向けの新たな学習支援機能として，学習段階で重要になる様々な対応付けや関係・処理の流れなどを分断させずに全体としてシームレスに表示する機能と文脈に依存して関係のある構成要素に意識を集中させるように表示する機能を提案する．

キーワード：プログラミング教育，視覚化，オブジェクト指向プログラミング，Eclipse，Java

Programming learning support system using Eclipse plug-in based on visualization function that emphasizes seamlessness and contextuality

Abstract: The authors have proposed and implemented an Eclipse plug-in, considering appropriate visualization in programming learning as an effective means to help learners understand. In this paper, as a new learning support function for beginners, it has the function to display seamlessly and the context without breaking various associations, relationships and processing flow which are important at the early learning stage We propose a function to display consciousness concentrating on relevant constituent elements.

Keywords: programming education, visualization, object-oriented programming, Eclipse, Java

1. はじめに

オブジェクト指向プログラミングの教育において，学習者を対象としたプログラミング学習環境の提案 [1], [2] や初中等教育におけるオブジェクト指向プログラミング言語を利用した教育に関する研究 [3] が精力的におこなわれている．このようなプログラミング教育においては，プログラムの理解を支援することが不可欠であり，静的な手法で解析したプログラムの振る舞い（プログラムのソースコードからフローチャートや実行パスの生成）やプログラムの実行履歴を取得することによる動的な振る舞いを視覚化情報として提供することで学習者の理解を支援する手法が提案されている [4], [5], [6].

[1] 東京情報大学
Uniersity of Information Sciences, 4–1 Onaridai Wakaba-ku, Chiba, Chiba, Japan
[a)] ohshiro@rsch.tuis.ac.jp
[b)] nagai@rsch.tuis.ac.jp

しかしながら，オブジェクト指向プログラミングの教育では，学生は，プログラミングの初期段階からオブジェクト指向の概念に直面することになる．オブジェクトやクラスなどのモジュール構造に関する概念を，オブジェクト間やクラス間のつながりを可視化することで，プログラム全体もしくは取扱い中のモジュール全体を理解することが必要となるが，オブジェクト間やクラス間の内部構造については可視化が不十分である [7]．したがって，今後の視覚化に関しては，オブジェクトやクラスの内部および外部の構造を同時に把握できることが重要となってくる [7], [8], [9].

また，現状のオブジェクト指向プログラミングの教育では，構造化プログラミングの知識で理解できるプログラムを先に提示するために，オブジェクト指向の基本的な概念の把握が不十分なままに，クラスを用いたプログラムを教える場合が多い．その結果，学習者は自身が記述したプログラムにおけるクラス間やオブジェクト間の関係を十分理解できないまま，プログラミングの教育が進められること

©2017 Information Processing Society of Japan

になってしまう．

この様な問題に対し，若林ら [8] は，クラス構成の問題点を明らかにするために，クラス間の依存関係を初学者のプログラムから解析し，これらの関係の理解を容易にするための手法を提案している．喜多 [9] らは，初心者を対象とした Java プログラム自動可視化ツール Avis に対して，クラス構造の可視化機能を追加することで，プログラムの振る舞いを可視化して理解することを容易にしている．櫻井ら [7] は，プログラムのモジュールの依存関係を可視化するために，静的解析によりプログラムのモジュール性を向上させるための学習支援ツールを提案し，適用実験をおこなっている．また，Java プログラムの視覚化ツールとしては，BlueJ [10] や Jeliot [11], [12] が普及している．

筆者らは，視覚化機能を持つプログラミング学習支援システムを Eclipse プラグインとして開発してきた [13], [14], [15], [16], [17], [18] が，その過程で，学習者の理解の段階によって有効な視覚化の方法も変化すると実感するようになった．一般に，視覚化機能つきのプログラミング学習用ツールは，各ツールの開発現場において教育者が自分達の教育対象となる学習者の持つ修得済みの理解能力に関して一定の水準を仮定しており，それに合わせて作成されているように思われる．つまりこれらのツールは，各教育現場の学習者に関して最適化された設計になっているということである．

著者らは，大学生などを対象とした前掲の視覚化ツール群は，学習者がプログラミング経験者であることを期待している仕様となっていると感じており，プログラミング学習者に対する視覚化の基本的な要件を見直し，プログラミング初学者にどのような視覚化が適切かを再検討した．

2. 視覚化におけるシームレス性

プログラミングを学習するとき，プログラムの実相 (ソースプログラム上に表現された構成単位やプログラムが動作するときの様子) を理解するのは重要である．

通常，プログラミング初心者はプログラミング言語を学習しながら，その過程でプログラムの実相を理解できるようになる．この段階での学習の流れを **図 1**(1) に示す．プログラミングに習熟するにつれ，この流れは双方向となる (図 1(2))．新たな言語を学習したり，新たな言語を設計する場合は，思考の流れの向きは 図 1(3) に示すようになり，プログラムの実相を新たな言語でどのように表現するかを考えて設計することになる．つまり，プログラミング学習の本質はプログラムの実相を理解することと言える．

一般に，言語・記号で表現された実相を学習するとき，言語表現と実相を対応付けることが基礎となる．学習初期で両者の対応付けは弱いが，学習の過程でこの対応付けが維持されると対応付けは強化され，対応付けが十分に強化された状態が習熟状態であると言える．

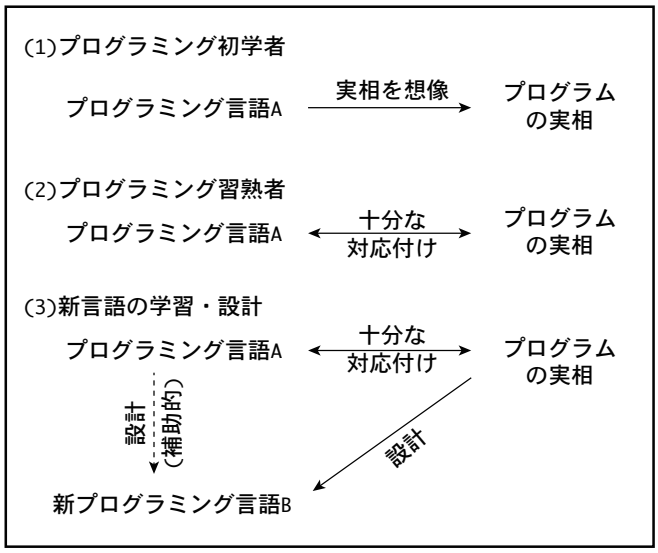


図 1 プログラミングの理解・思考の流れ．
Fig. 1 Flows of understanding / thought of programming.

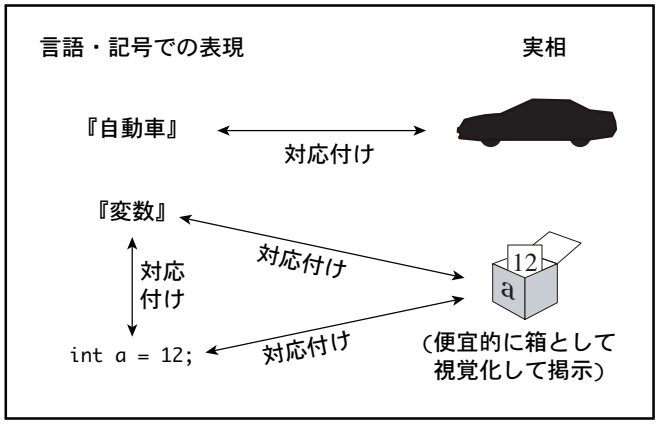


図 2 言語・記号での表現と実相．
Fig. 2 Representation and the real fact.

例えば，**図 2**(1) の『自動車』のように，実相の具体例が実物の場合は理解しやすい．これは，言語表現と実相を対応付ける上で，実相として実物が視界内にあれば，対応付けは維持しやすく結果として強化されやすいためと考えられる．逆に，言語表現と実相のうち，両者もしくは一方が概念上の存在のように想起し続けなければ捉え続けることが難しいものであると，両者の対応付けは維持されにくく，習熟することが難しい．

『変数』を例にプログラミング学習の場合について考えると，変数をはじめとするプログラムの実相 (メモリ上の状態・CPU 命令の実行される様子・バスを介したデータの流れなど) は視覚的に観察するのは難しい上に，学習者は変数の実相を理解した上で，変数の実相，『変数』という用語，プログラミング言語上での変数の表現の三者を十分対応付けしなくてはならない (図 2(2))．

©2017 Information Processing Society of Japan

プログラムの視覚化の最大の目標は，文字通り「プログラムの実相を視覚的に表示して理解しやすくする」ことと言える．そして，習熟状態 (図 1(2)) へ達するためには，学習の過程 (図 1(1)) でプログラミング言語で表現されたものとプログラムの実相の対応付け関係を強く維持することが重要と考えられる．

つまり，プログラムの実相を視覚化する場合にも，学習の過程に存在する様々な対応付けを維持しながら視覚化することで，学習効率の改善が期待できる．また，プログラムの実相の中に存在するメソッドの呼び出し関係，クラス間関係，オブジェクト間関係のような関係も初心者にとっては把握しづらい対応関係であり，分断せずに視覚化することが重要となる．

視覚化において，対応付けを分断する様々な要因が考えられる．著者らは，このような学習上重要になる対応付けを分断しないように提示する視覚化上の性質をシームレス性と呼ぶこととした．シームレス性の要件としては，対象そのものが消えたり長時間隠れたりしない，位置が移動しない，表現が変化しない，関連する要素を近くに配置する，対応付けは可能な限り単純に表現する，などが挙げられる．

既存の視覚化ツールの多くでは，クラスやオブジェクト，処理が異なる領域で表示されたり，メソッド呼び出しの際にいったん画面上の繋がりが途切れたりする場合がある．互いに関係する対象が個別のエリアに分断されていたり，動作の表示中に処理の流れが一時的にでも分断されると，上述したように理解の妨げになると考えられる．そこで，これまで提案・開発を行ってきた視覚化方式について，シームレス性を高めるために再検討を行った．

著者らの視覚化システムでは，視覚化を大きくソースコードを視覚化する静的視覚化と，プログラムの動作を動的に表示する動的視覚化に分けているが，以前より静的視覚化の段階ではソースコードを編集する際に，変数やクラス，メソッドの宣言・定義が書かれた時点でリアルタイムに構造が視覚化されるようになっている．この構造のリアルタイムな視覚化は，ソースコード上の構造と視覚化された対象の対応付けを強化することを意図していた．更に，本研究では静的視覚化の状態から，動的視覚化にシームレスに移行する方式を提案する．この静的視覚化から動的視覚化へのシームレスな移行によって，学習者はソースコード上に書いた内容と視覚化されたプログラムの実相との対応付けを途切れさせることなく，プログラムの挙動を確認する状態へ移行することが可能である．

プログラムの視覚化において大きな問題のひとつが，視覚化した対象の配置・表示方法である．多くの視覚化ツールでは，対象を単純な矩形もしくは矩形に類似した図形を用いている．これは限られた表示領域に効率よく収めることができるためであると考えられる．更に種類が異なる要素を同じ矩形もしくは矩形の類似図形で表示するのは識別しにくく混乱しやすいので，要素を種類ごとに別のエリアにまとめて表示することが多い．このような表示形式の場合，要素の数が多いとスクロールや表示領域のリサイズによってすべての要素をブラウズできるようになっている．

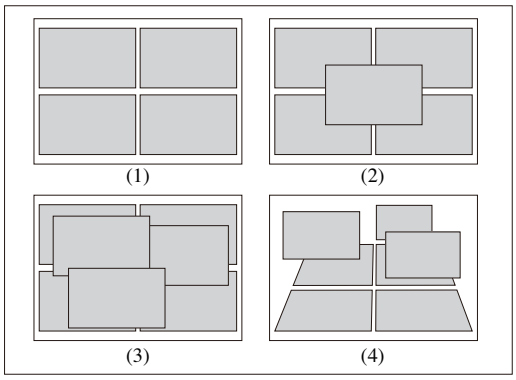


図 3 要素配置の方式.
Fig. 3 Methods of arrangement.

しかし，この様な表示設計がシームレス性を低下させている要因のひとつと考えられる．スクロール操作や表示範囲のリサイズは，表示領域内における要素位置の変化をともない，認識上の対応付けを分断させ，対象の要素の再探索をユーザに強要する．

またこのような表示形式の場合，要素自体は名称順や定義順などに 1 次元的もしくは 2 次元的に綺麗に列べられており (**図 3**(1))，要素間の関連性は考慮されていないことが多い．そのため要素間の関係を示す線などは逆に複雑な形状になりやすく，また関係のある要素が著しく離れて配置される可能性もあると言える．したがって，これらも認識上の対応付けを分断しやすくする原因と考えられる．

著者らは，3D 空間に基本的な要素をすべてまとめて表示することによって，この問題を解決しようと考えた．2D 表現で限られた表示領域に要素図形の合計面積を超えた個数の要素図形を表示する方法としては擬似的な奥行きをともなった層表現が考えられる (図 3(2))．この場合，要素を 1 次元的もしくは 2 次元的に綺麗に配置する必要は無く，互いに関連性の強いものを近くに配置するなどの，シームレス性を高める配置が可能になる．しかし，このような層表現も限界があり，要素の数が増えればいずれは下層の要素が長時間隠される場合が生じたり (図 3(3))，層数が多くなれば層の前後関係が分かりにくくなるなど，対応付けを困難にしたり，分断させたりするものと思われる．

そこで，表示を擬似的な奥行きの層表現から 3D 表現に拡張することによって，可能なかぎり必要な要素を同時に表示し，関連する要素を近くに配置することとした (図 3(4))．3D 表現の奥行きによって，「遠く (奥方向)」に配置されると小さくなるが，消えることは無く，また 3D 画面内に表現された「世界」では位置も変化せずに見ているユーザにとっては自然に維持される．遠方に配置された

©2017 Information Processing Society of Japan

要素も，後述する文脈の変化によって必要があれば「カメラ」自体がその要素に近づいて大きく表示する．

このためシームレス性を保ったまま画面内に多くの要素を配置でき，要素の形も識別のために様々な形をとることが可能になる．

3. 視覚化における文脈依存性

プログラムの各要素を 3D 空間に表示する場合，問題となるのが配置の方法である．そこで，本システムでは実行内容に合わせて表示する対象を視野内にあらかじめ集めることによって，この問題を解決した．正しくコンパイルできるソースコードは JDT を使った解析によって，main メソッドで行われる一連の処理で使用されるクラスとオブジェクトをあらかじめ調べ，それらのみを表示するようにする．このため，継承階層の深いサブクラスを定義しても，実際にソースコードに登場しないスーパークラスは表示されない．たとえば，Java では Object クラスがすべてのクラスのスーパークラスとして定義されているが，main メソッドからの処理の流れの中で利用されていない場合は，継承関係図には表示されない．その他，main メソッドが起動するときには，main メソッドを連続的にクローズアップしてから処理が始まるようにし，main メソッドがプログラムの処理の起点となることを強調する．その後は，main メソッドの処理の流れの中で利用するクラスとその継承階層が表示される範囲まで表示視野を広げて実行の様子を再生するが，非参照となったオブジェクトは積極的に非表示として，学習者が実行中の文脈に集中できるようにする．また，Java 標準パッケージクラスのメンバを利用している場合でも，オブジェクトを生成していない場合は非表示とした．たとえば，System.out.println メソッドを呼び出していても，System クラスは視覚化されない．

これらの表示方法によって，プログラムの文脈に依存した表示を行うことが可能となり，学習者も必要な対象のみに意識を集中させることができると思われる．

4. システムの概要と視覚化の例

本システムは，以前より Eclipse のプラグインとして開発され，プログラミング環境の一部として機能する．**図 4** は，静的視覚化段階で視覚化ビューを開いた状態である．後述するように，この状態からコンパイル・実行を行うとそのまま動的視覚化モードにシームレスに移行する．

例として，**図 5** のプログラムを視覚化した例をあげる．これは，手に持つ道具によって振る舞いの変わる人間を表しており，典型的な Strategy パターンの使用例である．

ソースコードの静的視覚化の様子を **図 6** に示す．実際には，図 4 に示すように，Eclipse に追加した視覚化ビューの内部に表示される．main メソッドで使用されるクラスがメインクラスの周辺に配置される．

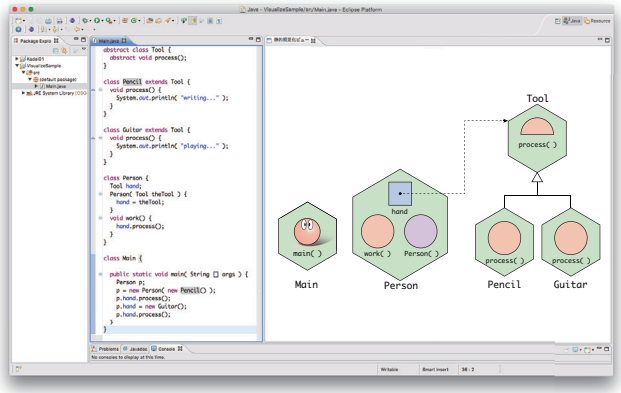


図 4 エディタウィンドウと視覚化ビュー．
Fig. 4 An editor window and visualization view of the source code.

```
abstract class Tool {
  abstract void process( );
}

class Pencil extends Tool {
  void process( ) {
    System.out.println( "writing..." );
  }
}

class Guitar extends Tool {
  void process( ) {
    System.out.println( "playing..." );
  }
}

class Person {
  Tool hand;
  Person( Tool theTool ) {
    hand = theTool;
  }
  void work( ) {
    hand.process( );
  }
}

class Main {
  public static void main( String [] args ) {
    Person p;
    p = new Person( new Pencil( ) );
    p.hand.process( );
    p.hand = new Guitar( );
    p.hand.process( );
  }
}
```

図 5 プログラムの例．
Fig. 5 A sample program.

このままプログラムをコンパイル・実行すると，視線が移動し，動的視覚化の表示状態となる (**図 7**)．画面が切り替わることなくプログラムの挙動を観察できる状態になるので，学習者が同時に表示されているソースコードに書かれたクラス，視覚化されたクラス，そしてソースコードに書かれた処理と視覚化された動的振る舞いの対応関係を断

©2017 Information Processing Society of Japan

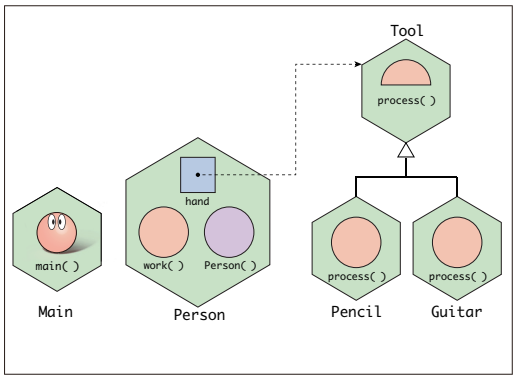


図 6 静的視覚化の表示例.
Fig. 6 An example of static view.

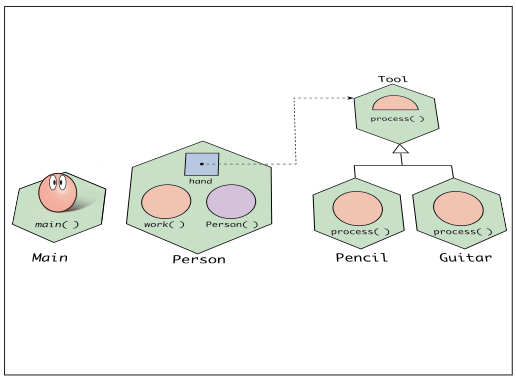


図 7 動的視覚化の最初の状態.
Fig. 7 The first state of dynamic visualization.

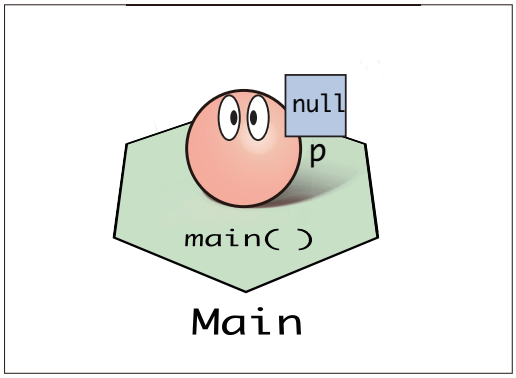


図 8 変数 p の宣言文が実行された状態.
Fig. 8 Declaration of a local variable p.

絶されずに確認し続けることが出来る.

main メソッドが実行されると，main メソッドに描写用カメラが寄り，最初の局所変数 p の宣言文が実行されると p が視覚化される (図 8). このとき，システムの全体の表示は**図 9** のようになっており，ソースコード上で実行されているメソッドと，実行中の式はハイライトされている. 以下同様に，動的視覚化の過程では対応するソースコードがその都度ハイライトされる.

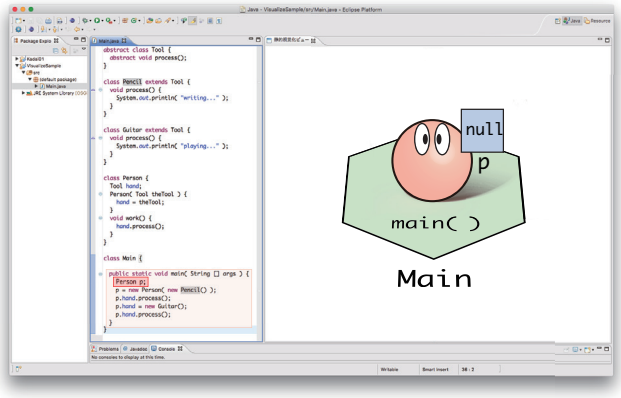


図 9 **図 8** の状態の全体表示.
Fig. 9 The whole view at the status of Fig.8.

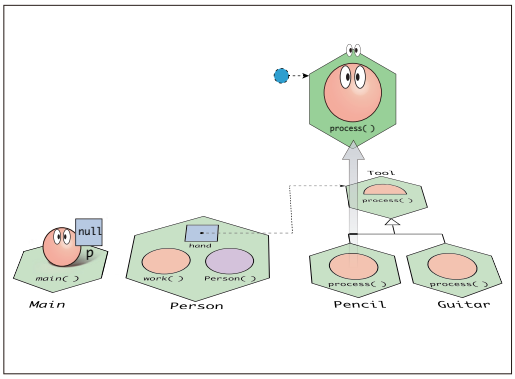


図 10 Pencil 型オブジェクトが生成される様子.
Fig. 10 This figure shows creating an object from class Pencil.

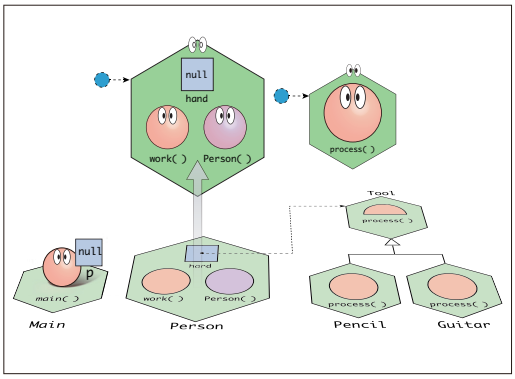


図 11 Pencil 型オブジェクトが生成される様子.
Fig. 11 This figure shows creating an object from class Pencil.

new 演算子で Pencil 型オブジェクトを生成する様子は，Pencil クラスの図より Pencil 型オブジェクトが浮き上がって現れる様にアニメーションが実行される (**図 10**). 次いで，new 演算子で Person 型オブジェクトが生成される様子を **図 11** に示す. Person 型オブジェクトのコンストラクタに，円で表示された Pencil 型オブジェクトへの参照

©2017 Information Processing Society of Japan

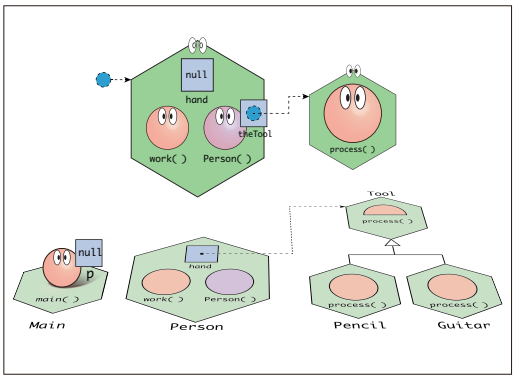


図 12 Person 型オブジェクトのコンストラクタに Pencil 型オブジェクトへの参照値が渡される様子.

Fig. 12 The constructor of Person type object is called by reference to a Pencil type object.

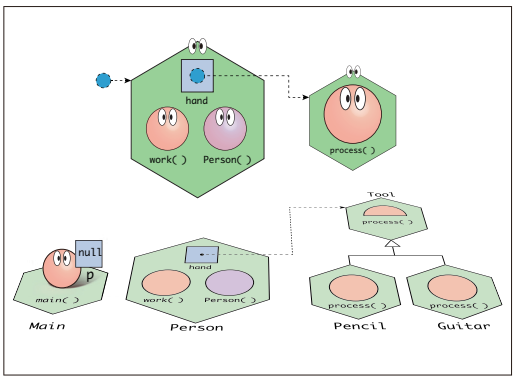


図 13 Person 型オブジェクトが初期化された様子.

Fig. 13 The Person type object has initialized.

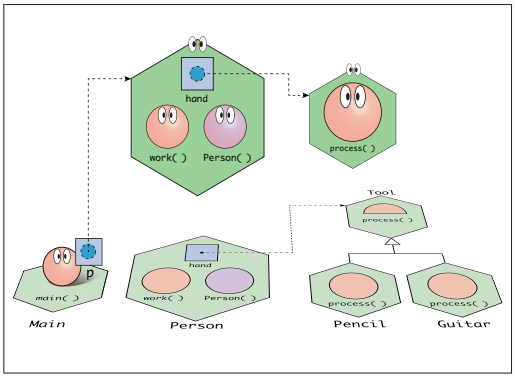


図 14 局所変数 p が初期化された様子.

Fig. 14 The local variable p has initialized.

値が渡された様子が **図 12** である．このとき，参照値を表す円と参照関係を表す破線はなめらかなアニメーションによって移動する．コンストラクタによって初期化された Person 型のオブジェクトの様子が **図 13** である．

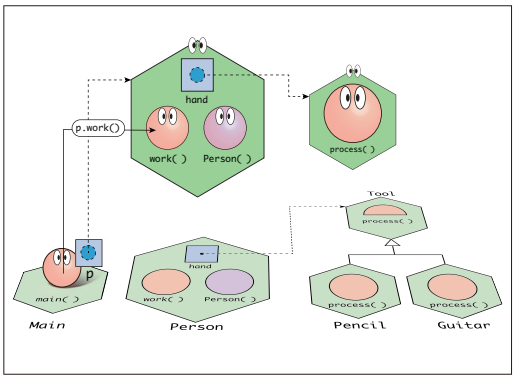


図 15 Prson 型オブジェクトへ work メッセージが送信された様子.

Fig. 15 A message 'work' has sent to the Person type object.

そして，main メソッドの局所変数 p に初期化済みの Person 型オブジェクトへの参照値が代入される (**図 14**). main メソッドは，p で参照している Person 型オブジェクトへ work メッセージを送信する (**図 15**). このメッセージ送信を表す矢印は，メッセージ送信元のメソッドから送信先オブジェクトの該当メソッドまで，アニメーションによってなめらかに伸張していく．更に，Person 型オブジェクトの work メソッドは，フィールド hand によって参照している Tool 型オブジェクト (実際には Pencil 型オブジェクト) へ process メッセージを送信する．これによって，"writing..." という表示が行われる (**図 16**).

main メソッドでは，次の処理として Person 型オブジェクトが参照している Tool 型オブジェクトのつなぎ替えを行う．まず Guitar 型オブジェクトを生成し (**図 17**)，生成した Guitar 型オブジェクトへの参照値を Person 型オブジェクトのフィールド hand に代入する (**図 18**). このとき，Pencil 型オブジェクトはどこからも参照されなくなるのでいずれ Java のガベージコレクタによって削除・解放されるが，視覚化表示上では，ガベージコレクタの削除・解放処理を待たずに徐々に薄くなる効果とともに非表示とする (**図 19**). これは，参照されないオブジェクトは自動的に削除されるという Java のシステムの働きを学習者に知らしめるためでもあるが，基本的には無関係な表示対象を極力無くすという文脈依存表示の方針に基づく．

最後に，main メソッドの局所変数 p で参照している Person 型オブジェクトへ work メッセージが送信され，ついで Person 型オブジェクトの work メソッドは，フィールド hand が参照している Tool 型オブジェクトへ process メッセージを送信し，その結果，"playing..." という表示が行われる (**図 20**).

5. 関連研究について

本節では，既存の視覚化機能を持つ教育手法・ツールと

©2017 Information Processing Society of Japan

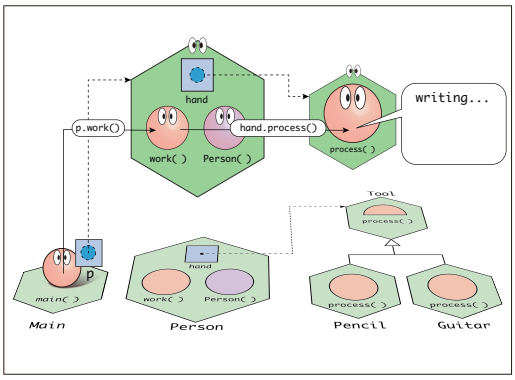


図 16 Pencil 型オブジェクトへ process メッセージが送信された様子.
Fig. 16 A message 'process' has sent to the PEncil type object.

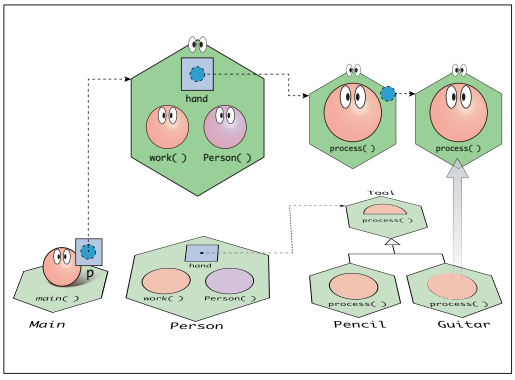


図 17 Guitar 型オブジェクトが生成される様子.
Fig. 17 This figure shows creating an object from class Guitar.

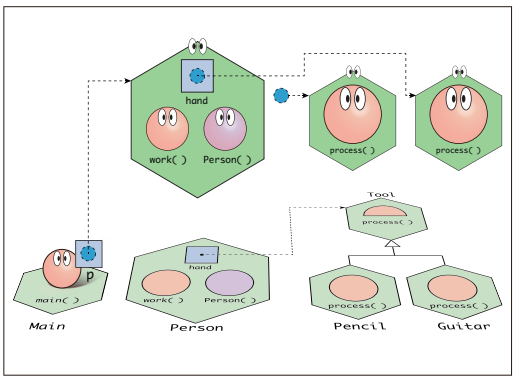


図 18 Guitar 型オブジェクトへの参照値がフィールド hand に代入された様子.
Fig. 18 A reference value to the Guitar type object is assigned to field 'hand'.

本システムとの関連を述べる.

小学生からでも情報科学の基礎を分かりやすく学習できるようにするために工夫された教育法のひとつに，Computer Science Unplugged [19], [20] があり，コンピュータやプログラミング言語を使わずに，実物の教材を用い実世界での体験を通して情報科学の基礎 (例えばアルゴリズムの実相など) を理解できるように工夫されている．実物を使った教育方法は概して初学者が理解し易い．これは，実物は突然消えることもなく継続して視認可能であること，つまり本稿で定義したシームレス性を持った教材・教育方法である点が理解しやすさの根底にあると思われる．アルゴリズムの実相を理解する上では効果的だが，高等教育でのプログラミング言語の学習の側面はあえて省かれている.

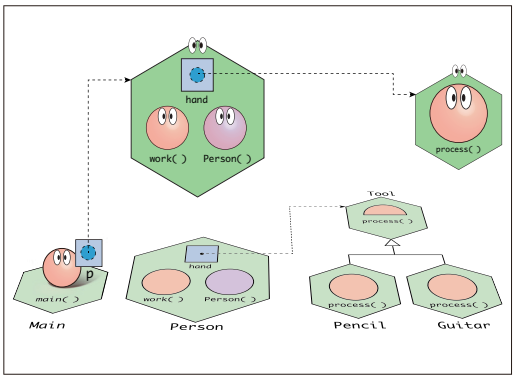


図 19 Pencil 型オブジェクトを非表示にする.
Fig. 19 A Pencil type object is hidden.

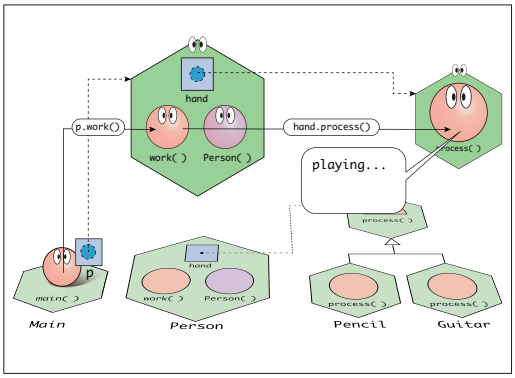


図 20 Guitar 型オブジェクトへ process メッセージが送信された様子.
Fig. 20 A message 'process' has sent to the Guitar type object.

初学者がプログラミング言語の厳密な文法を意識せずに，ブロックとして視覚化されたプログラムパーツを組み合わせることで，手軽にグラフィックや動きをともなった表示を楽しみながらプログラムの実相を理解できるように工夫されたプログラミング環境に Scratch [21] がある．Scratch は，実装としてはスプライトやステージといった対象をオブジェクト指向におけるオブジェクトとして実現しているが，プログラミング言語としては積極的にオブジェクト指向の機能を使えるわけではない．基本的なプログラムの実装を理解する優れた環境ではあるが，高等教育でのオブジェクト指向も含めたプログラムの実相の理解と，実務として使われているオブジェクト指向プログラミング言語の

©2017 Information Processing Society of Japan

学習のための環境ではない．

高等教育でのオブジェクト指向プログラミング学習，特にJava の学習に役立つ視覚化を持ったツールとして，Jeliot や BlueJ がある．Jeliot は，静的視覚化機能は持っておらず，完成されたプログラムの動的視覚化を主眼に置いている．そのため，正しいソースコードを書けることを前提としている．一方，プログラムの挙動からプログラムの動きとオブジェクト指向に慣れさせるのに向いており，特に，変数の値の受け渡しやメソッドの呼び出し関係は分かりやすく視覚化されている．しかし，ほぼすべての構成要素が矩形で表されており，識別は表示位置と色・名称で判別しなければならない．また，メソッドがクラスやオブジェクトと分離表示されていて，クラスメソッドかインスタンスメソッドかは，名称や this 参照メンバの有無などで判別する必要がある．また，BlueJ は，インタプリタ感覚で使用できる環境で，UML に近いクラス・クラス間関係表示やオブジェクトとその挙動を視覚化することで，オブジェクト指向の基本を理解しやすいように工夫されてはいるものの，値の受け渡しやオブジェクト間の関係は直接的には視覚化されておらず，値を見てそれらの関連性を理解しなければならない．前述したように，Jeliot や BlueJ のような視覚化機能を持った高等教育用のツールは，学習者が一定の理解水準まで到達していることを前提に，最適化されて設計されていると言える．対して，著者らの視覚化ツールは，可能な限り初歩の段階から利用できるように意図しており，その重要な要素のひとつとしてシームレス性の維持に注意を払って設計されている．

6. おわりに

本研究では，初心者がオブジェクト指向を学習する際に重要となる様々な対応付け・関連を維持するためのシームレス性を重視した視覚化と表示レイアウトの問題解決を兼ねた文脈依存性をともなう視覚化の方式を提案した．今後，実装を行って実際に授業などで使用して効果を検証する予定である．

参考文献

[1] 萩庭　崇，永田守男：オブジェクト指向言語のための視覚的プログラム支援環境，情報処理学会ソフトウエア工学研究，Vol. 96, No. 4, pp. 25–32 (1996).

[2] 長　慎也，甲斐宗典，川合　晶，日野孝昭，前島真一，筧　捷彦：Nigari-Java 言語へも移行しやすい初学者向けプログラミング言語，情報処理学会研究報告コンピュータと教育，Vol. 103, No. CE-071, pp. 13–20 (2003).

[3] 兼宗　進，中谷多哉子，御手洗理英，福井眞吾，久野　靖：初中等教育におけるオブジェクト指向プログラミングの実践と評価，情報処理学会論文誌，Vol. 44, No. SIG 14 (PRO18), pp. 58–71 (2003).

[4] 喜多義弘，川添貴議，片山徹郎：初心者を対象にした Java プログラム自動可視化ツールの実現に向けて，信学技報 SS，Vol. 104, No. 570, pp. 19–24 (2005).

[5] 谷口孝治，石尾　隆，神谷年洋，楠本真二，井上　克：プログラム実行履歴からの簡潔なシーケンス図の生成手法，コンピュータソフトウェア， Vol. 24, No. 3, pp. 153–169 (2007).

[6] 竹下彰人，片山徹郎：シーケンス図を用いた実行履歴の可視化による Java プログラムの理解支援に関する考察，信学技報 SS，Vol. 106, No. 426, pp. 43–48 (2006).

[7] 櫻井孝平，松浦佐枝子，古宮誠一：モジュール分割を学習するための依存性の解析に基づくツールの開発，信学技報 KBSE2009-63，pp. 25–30 (2010).

[8] 若林智徳，松浦佐江子：Java プログラミング初学者のクラス構成学習のための依存性解析，信学技報 KBSE2010-44，pp. 55–60 (2011).

[9] 喜多義弘，川添貴議，片山徹郎：Java プログラム自動可視化ツール Avis におけるクラス構造可視化のための拡張，信学技報 SS， Vol. 105, No. 490, pp. 7–12 (2005).

[10] Kolling, M., Quig, B., Patterson, A. and Rosenberg, J.: The BlueJ system and its oedagogy, *Journal of Computer Science ducation Special Issue on Learning and Teaching Object Technology*, Vol. 13, No. 4, pp. 249–268 (2003).

[11] Moreno, A., Myller, E. and Ben-Ari, M.: Visualizing Programs with Jeliot 3, *In Procs. of the International Working Conference on Advanced Visual Interfaces*, ACM, pp. 373–376 (2004).

[12] Hongwarittorrn, N. and Krairit, D.: Effects of program visualization (Jeliot3) on students' performance and attitudes towards java programming, *The spring 8th International conference on Computing, Communication and Control Technologies* (2010).

[13] 大城正典，永井保夫：オブジェクト指向プログラムの静的・動的側面を視覚化するプログラミング教育支援システム，情報処理学会第 71 回全国大会講演論文集 (4)，pp. 4–413, 4–414 (2009).

[14] 大城正典，永井保夫：初等プログラミングから設計レベルまでを対象としたオブジェクト指向教育のための支援システムの提案，情報処理学会 情報教育シンポジウム 2011 論文集 SSS2011，pp. 59–66 (2011).

[15] 大城正典，永井保夫：段階的コーディングガイド機能およびモニタ機能を持つオブジェクト指向プログラミング教育のための視覚化支援システムの提案，信学技報教育工学， Vol. 114, No. 260, pp. 53–58 (2014).

[16] 大城正典，永井保夫：Eclipse 視覚化プラグインによる総合的なプログラミング教育支援システム，情報処理学会情報教育シンポジウム 2015 論文集 SSS2015，pp. 23–30 (2015).

[17] 大城正典，永井保夫：視覚化機能を持つ Eclipse プラグインによる段階的コーディングから動作検証までをサポートするオブジェクト指向プログラミング学習システム，信学技報教育工学， Vol. 115, No. 492, pp. 61–66 (2016).

[18] 大城正典，永井保夫：プログラミング初学者を対象としたオブジェクト指向プログラミング教育システムの提案-オブジェクト指向の基本概念の理解に基づいたプログラムの作成・実行支援機能を中心として-，情報処理学会情報教育シンポジウム 2016 論文集 SSS2016，pp. 114–121 (2016).

[19] Bell, T., Witten, I. H. and Fellows, M.: コンピュータを使わない情報教育　アンプラグドコンピュータサイエンス，イーテキスト研究所 (2007).

[20] University of Canterbury: CS Unplugged Computer Science without a computer, , available from ⟨http://csunplugged.org⟩

[21] MIT Media Lab: Scratch, available from ⟨https://scratch.mit.edu⟩

©2017 Information Processing Society of Japan