情報処理学会論文誌　Vol. 50　No. 10　2396–2408 (Oct. 2009)

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ためのワークベンチ

三浦元喜[†1,*1]　杉原太郎[†1]　國藤進[†1]

我々は Java や C#などの静的な型付けを行うオブジェクト指向言語を学習するうえで比較的抽象度が高くつまずきやすい「型・変数・オブジェクトとデータ参照」の理解を促進するためのワークベンチ Anchor Garden を構築した．学習者はマウス操作により視覚的な表現を持つ “変数” や “オブジェクト” を生成したり，変数からオブジェクトにリンクを張る操作体験を通じて，プログラミング言語における独特の考え方に接近することができる．また本ワークベンチはモデルの操作に対応したコードを自動生成する機能を持つ．これにより学習者はプログラムのコードと自身が行ったモデルの操作を対応付けることができる．実験の結果，Anchor Garden に習熟すると変数とオブジェクトの関係の理解能力が向上する傾向が見られた．また変数とオブジェクトの関係理解度とプログラム作成能力との相関が見られ，Anchor Garden がプログラム作成能力を向上する可能性が示された．

A Workbench for Understanding Relationship between Variable and Data in Object Oriented Programming Language

Motoki Miura,[†1,*1] Taro Sugihara[†1] and Susumu Kunifuji[†1]

We propose “Anchor Garden,” an interactive workbench software to learn fundamentals of data structure with concepts of type, variable, object, and its relations in a strongly-typed object oriented programming language such as Java and C#. Learners can approach the concepts by a direct manipulation of graphical models. Anchor Garden (AG) allows the learners to create variables and objects, and to link among them. Since AG automatically generates source-code corresponding to the learner’s manipulations, the learner can relate manipulations and representations of source code. Experimental result showed a tendency of learning effect with AG, and high correlations between the concept understanding and programming ability. Thus AG has a possibility to enhance the programming ability for novice programmers.

1. はじめに

学習者が Java や C#などのオブジェクト指向プログラミング言語を修得する場合，サンプルコードを入力し，コンパイルや実行を繰り返しながら理解を深めていくことが一般的である．近年では統合開発環境の整備が進み，プログラミング上の記述ミスやエラーに気付きやすくなったため，初心者でも試行錯誤をともなう作業が円滑に行えるようになってきた．しかし，ソースコード上の表現と，実行時の意味を正しく関連付けて理解するには，試行錯誤に加えて抽象概念に対する洞察力が求められるため，初心者に対する敷居は依然として高い．

ソースコード上の表現と実行時の意味を関連付けやすくするため，プログラムの動作をアニメーションによって表現する Jeliot3[1] やアルゴリズムアニメーションを簡易に作成できる JAWAA[2]，3 次元キャラクタにメッセージを送り，動作させることができる Alice[3] などのシステムが構築されている．いずれもグラフィカルな表現やアニメーションを利用して，プログラムの振舞いやアルゴリズムを理解させることを目的としている．また Scratch[4] や Squeak eToy[5]，Viscuit[6] など，ソースコードの編集をほとんど行わずに手軽にキャラクタを動かしたりアニメーションを作成したりすることができるプログラミング環境も構築されている．しかしこれらのシステムでは，アルゴリズムや制御構造を視覚的に見せることに重点が置かれており，プログラムにおけるデータ構造や，変数とオブジェクトの参照関係といった，実践的なプログラミングで求められる抽象度の高い内容を対象としていない．

そこで我々は学習者が Java や C#などの静的な型付けを行うオブジェクト指向言語を修得するうえでつまずきやすい概念のうち，抽象度が比較的高く，上記の環境でカバーされていない「型・変数・オブジェクトとデータ参照」に対象を絞り，これらの理解を促進するためのワークベンチ Anchor Garden（以降，AG）を構築した．本稿における「ワークベンチ」とは，概念の理解を促すためのモデルの視覚化に加えて，現実世界のメタファを援用し

†1 北陸先端科学技術大学院大学知識科学研究科
School of Knowledge Science, Japan Advanced Institute of Science and Technology
∗1 現在，九州工業大学大学院工学研究院基礎科学研究系
Presently with Faculty of Engineering, Kyushu Institute of Technology

ながら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ソフトウェアのことを指す．学習者はワークベンチにおけるモデルとの対話を通じて，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たりしながら体験的に学ぶことが可能となる．

2. 関連研究

これまでプログラミング初学者を対象に，オブジェクトの振舞いを視覚化しプログラムの理解を助ける環境として，ドリトル[7] や Alice[3),8)]，The Java Power Tools[9]，PigWorld[10]，Scratch[4]，ロボットを利用した環境[11] などが開発されている．また Nigari[12] は簡素だがドリトルよりも Java に近い構文を利用し，実用言語導入における段階的な概念理解を目的としている．JavaMM[13] も Nigari に類似した簡潔記法を導入している．これらの環境では，コード実行をキャラクタやロボットの振舞いとして学習者に見せることによって，オブジェクト指向のプログラミングという概念を分かりやすく教示している．

ソートや 2 分木探索などのアルゴリズムを動きで示す方法として，アルゴリズムアニメーションが有効である．JAWAA[2] は Web ベースのアニメーションを簡単に作成できるスクリプト言語である．Java アプレットで書かれた実行系が，スクリプトファイルを読み込んで表示する．Jeliot3[1),14)] は，Java プログラムの実行における変数や配列への代入，条件文などをアニメーションで表示するシステムである．配列も扱うことができ，ステップ/連続実行によりソースコードの意味を理解できる点では有効である．しかし抽象度の高いシャローコピーなどのデータ構造を正確に表現することはできない．

初学者用に工夫された開発環境についても構築がすすめられている．PEN[15] は西田らが構築している初学者用プログラミング環境である．入力ミスによる文法エラーを軽減するため，制御構造などをボタンによって入力する機能や，プログラムのステップ実行，変数インスペクタなどを備えている．基本的に日本語を用いたプログラム表記によるアルゴリズム理解に焦点をあてている．BlueJ[16] は，プログラミング初心者の学習を意識して設計された Java 開発環境である．クラスやインタフェースの作成と，継承や関連の定義を UML 図から簡単に行えたり，クラスのインスタンスを生成してメソッドを呼び出すといった操作が行えたりする．UML 図からの操作のみでなく，ソースコードをインポートして UML 図を生成することもできる．NetBeans IDE 用のプラグインにより，大規模開発との連携も考慮されており，UML 図を見ながら複数のクラスを定義していく学習場面では有効であると考えられる．継承やインスタンス化といった抽象度の高い内容を扱っているが，基本的なデータ構造の仕組みについて示唆を与えるものではなく，また内容は AG よりも上級の学習者を対象としている．

双方向リンクや 2 分木などのデータ構造を視覚化する研究については，Myers による INCENSE[17] や，jGRASP のオブジェクトビューア[18] などがある．プログラム実行中の動的なデータ構造をクラス名やフィールド名から推測し，半自動的に表示できる．こちらも対象としている利用者は比較的高度な内容を学ぶ学習者である．またデバッグの途中でデータ構造を参照しながらコードを理解することに主眼を置いているため，AG のようにオブジェクト操作から対応コードを生成することはできない（AG の詳細については 3 章で述べる）．

データ構造を視覚化するツールのなかで，我々の研究に最も関連しているのは Campbell らの LIVE（Languate-Independent Visualization Environment）[19] である．LIVE では視覚化されたオブジェクトをインタラクティブに操作してデータ構造を学習したり，対応するソースコードを生成したりすることができる．また C++と Java のコード表記を切り替えて類似性を発見することができる．グラフィカルな表現を用いているが，オブジェクト生成やリンクを張る操作はポップアップメニューを使用しているため，AG のような直感的な操作を提供していない．

3. Anchor Garden

Anchor Garden（AG）は，静的な型付けを行うオブジェクト指向プログラミング言語における「型」と「変数」および「オブジェクト」の直接操作によって学習者の概念理解を促すことを目的としたソフトウェアである．AG が対象とする利用者は，変数や計算手順の概念は理解しているが，配列をはじめとするデータ構造や上記プログラミング言語における具体的なデータ処理方法を修得していない学習者である．AG ではソースコードを直接編集することなく，型/変数/データやリンクを GUI 上で操作することにより，対応するソースコードをインタラクティブに生成する．モデル操作に対応するソースコードを参照することにより，型/変数/データの関係や，プリミティブとオブジェクトの違い，および参照についての概念形成を支援する．

3.1 基本操作

AG の起動時の画面を**図 1** に示す．画面は左から [Type]，[Variable]，[Object] の 3 つの領域に分かれており，それぞれ「型」「変数」「オブジェクト」を置くスペースとなっている．[Type] 領域には，int や String など「データ型選択ボタン」が配置されている．学習

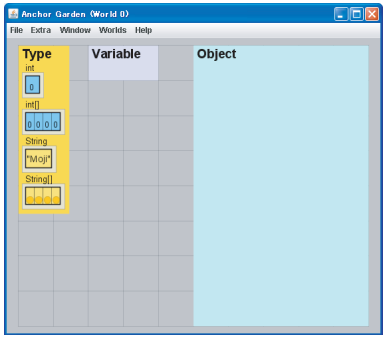


図 1　起動画面
Fig. 1　Initial Screen.

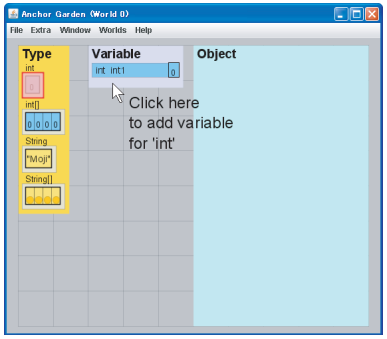


図 2　変数生成
Fig. 2　Create Variables.

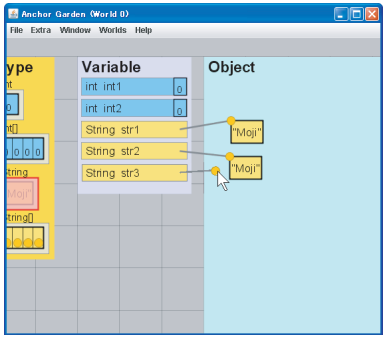


図 3　オブジェクトにリンク
Fig. 3　Link to Objects.

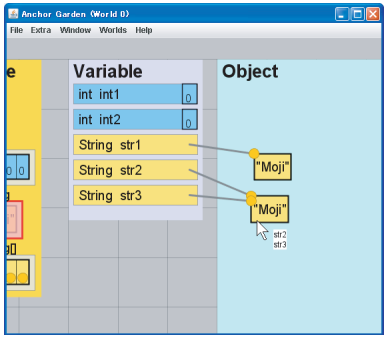


図 4　リンク完了
Fig. 4　Linked.

者はいずれかの「データ型選択ボタン」を押し「データ型」を選択した状態で，[Variable] や [Object] 領域を左クリックすると，選択されたデータ型の変数やオブジェクトが生成され画面に表れる．

Java におけるプリミティブなデータ型である “int” を選択した場合，[Variable] 領域で左クリックすると変数と値が生成される（**図 2**）が，[Object] 領域でクリックしても何も生成されない．String や配列といったコンストラクタによりオブジェクト生成を行うデータ型を選択した場合には [Object] 領域でオブジェクトを生成することができる．学習者は，選択したデータ型に対応したオブジェクトを，紙にスタンプを押すような感覚で生成することができる．

学習者は [Object] 領域に手軽にオブジェクトを生成できるが，生成したオブジェクトを放置しておくと，徐々に表示が薄くなり（透明度が高くなり）約 10 秒で完全に消滅する．この表現により，オブジェクトは変数から参照されないと利用できないことを示している．そこで学習者は消滅を避けるため，「変数」から「オブジェクト」にリンクを張る必要がある．オブジェクトにリンクを張るには，変数左の円形のつまみ（リンクタブ）をマウスでドラッグ（**図 3**）し，接続したいオブジェクト上でボタンを離す．リンクタブは，参照可能なオブジェクトにのみ接続できる．それ以外（変数や型，別の型のオブジェクト）に接続しようとした場合，警告音が鳴り，変数とリンクタブを結ぶ線がゴムのように縮む表現により，リンクタブが元の位置に戻る．このような「つまんでくっつける」という直感的な操作と「つながらないタブは元に戻る」表現によって，学習者は変数から参照するオブジェクトを簡単に指定したり，また変数から接続できるオブジェクトと接続不可のオブジェクトを知ることができたりする．

このような「変数」と「オブジェクト」を独立して生成し，かつリンクを張る操作を行うことにより，学習者は以下の内容を体験的かつ効果的に学ぶことができる．

- 型と変数，オブジェクト（データ）の関係．
- プリミティブ型（int）と，オブジェクト型（String，配列）の違い．
- オブジェクトを継続的に利用可能にするためには，少なくとも 1 つの変数からリンクでたどれる（参照している）必要があること．
- 1 つのオブジェクト型変数から参照可能なオブジェクトは 1 つであること．
- 1 つのオブジェクトが複数の変数から参照されうること（**図 4**）．またその際は，データの実体は共有されており，1 つのみであること（図 4 のマウスカーソル付近にある [str2/str3] という表示は，このオブジェクトを参照する変数やプログラム表記を示している）．
- リンクは張り替えたり，外したりすることが可能であること（リンクをすべて外されたオブジェクトは，急速に表示が薄くなり約 5 秒で消滅する）．
- リンクを外されたオブジェクトは，ガベージコレクション機能によって回収されること．

図 1〜図 4 で示した画面は，初級モードであり，選択できる型が int，String およびその配列に限られている．初級モードで学習可能な内容をすべて理解した学習者は，上級モードに移行できる．上級モード（**図 5**）では，基底クラス（Object）や，その派生クラス（Oval,

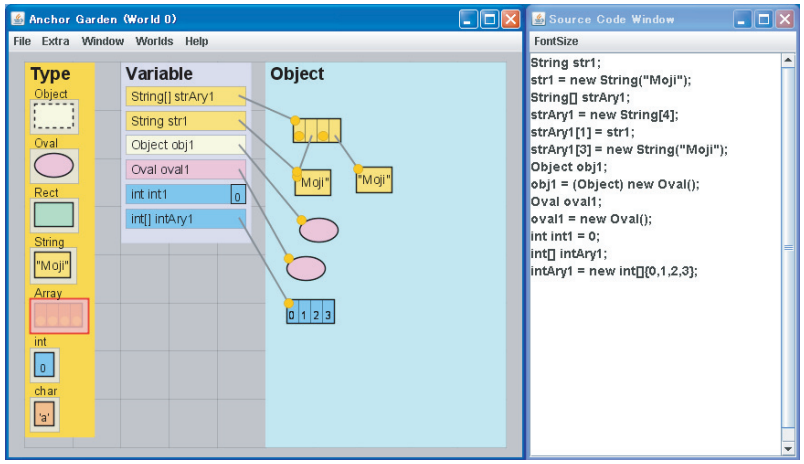


図 5　上級者モードとソースコード
Fig. 5　Advanced mode and source code window.

Rect，String）を [Type] 領域に配置し，それぞれのインスタンスを生成することができる．たとえば Object クラスの変数 obj1 から，Oval クラスのインスタンスには参照可能だが，Oval クラスの変数 oval1 から，Object クラスのインスタンスには参照できないといった，クラスの継承関係と参照ルールも理解できる[*1]．

3.2　ソースコード表示による操作とコードの対応付け

AG では，学習者のモデル操作に対応したソースコードを自動生成し，表示することができる．

学習者が String クラスの変数を生成すると，ソースコードウィンドウ（図 5 右，以下 SrcWin と表記）の 1 行目に `String str1;` が表示される．また String クラスのオブジェクトを生成すると，`new String("Moji");` が 2 行目に表示される．ここで学習者が変数 str1 からオブジェクトにリンクを張ると，2 行目の表示が `str1 = new String("Moji");` という表示に変化する[*2]．もし学習者がリンクを張らずに放置すると，2 行目の `new String("Moji");` はオブジェクト消滅時と同時に SrcWin から削除される．またいったんリンクを張ったあとでオブジェクトからリンクを外すと `str1 = null;` といったコードが表れる．この機能により，学習者はモデルの操作と対応するソースコード表記を同時に観察することができる．また，`null` や `new` キーワードの意味を無理なく学習することができる．通常は変数を準備してからオブジェクトを生成したほうが，オブジェクトへのリンクを張る操作に時間的余裕があるため自然であるが，先にオブジェクト生成後に変数を作成しリンクすることもできる．その場合，2 行目の `String str1;` と 1 行目の `new String("Moji");` が統合され，`String str1 = new String("Moji");` となる．

継承関係にあるクラスを用いた場合は，参照時の「キャスト」についても必要なときのみ自動的にコードに表れる．たとえば Object の変数 obj1 から Oval のインスタンス（Oval の変数 oval1 から参照されている）にリンクする場合，生成されるコードには `obj1 = (Object) oval1;` のように，自動的にキャストが挿入される．これにより，キャストについても試行しながら学ぶことができる．またキャストがインスタンスの型を変更するのではなく，単に参照を受け渡すための記述であることも学習できる．

なお，2 つ以上の変数からリンクされているオブジェクトに対して，新しい変数からのリンクを追加した場合，生成されるソースコードの右辺は，既存リンク元変数のうちのいずれか 1 つが選択される．たとえば，図 4 の状態で，str1 のリンクタブを str2 と str3 が参照しているオブジェクトに接続した場合，生成されるソースコードは `str1 = str2;` または `str1 = str3;` となる．

4.　学習可能な内容

AG を用いると，以下に述べる内容を学習することができる．以下の内容は「参照」概念の理解が必要不可欠であるが，書籍や言葉による説明，サンプルプログラムの実行のみでは理解しにくい内容であるため，システムを用いることを考えた．

4.1　配列の概念

「配列」を初めて学習する場合は，int の配列などプリミティブなデータ型の配列から始めるのが自然である．int の配列型変数を `int[] intAry1;` と定義するまでは既習概念で対応できるが，`intAry1 = new int[4];` のように配列を作成する場面で `new` キーワードが出現し，既習概念のみでは対応できなくなる．そのため学習者は `new` キーワードの意味を理解するか，“おまじない” として覚える必要がある．AG を用いると，`new` の意味や，変数と配列を＝で結ぶことの意味について，文字列を作成する場合の `String str1 = new`

⋆1 現在の実装では，変数を経由して，参照しているインスタンスのメソッドを呼び出す機能を設けていない．そのため，変数の型によって呼び出せるメソッドが規定されることは学習できない．

⋆2 一般的には `str1 = new String("Moji");` を `str1 = "Moji";` と記述するが，ここでは新しいオブジェクト生成と new を対応させるため，あえて new を用いた記法としている．

String("Moji"); との構造上の類似を確認しながら修得できる．

4.2　プリミティブの配列とオブジェクトの配列の違い

int の配列を理解した学習者が，String の配列などオブジェクトの配列を学ぶ際，はじめは int の配列と同様に “箱の中にオブジェクトが入っているイメージ” を持つものと思われる．しかし実際の配列要素は “オブジェクトへの参照” である．これを理解するためには「変数」から「オブジェクト」への “参照” 概念を理解したうえで，「配列要素は変数と同じく，参照である」ことを学ぶほうが効果的である．AG では，String の配列（インスタンス）を生成すると，リンクのタブ（円形のつまみ）が配列要素として表示される．それらのタブは String オブジェクトのみにリンクできる．こうした表示と操作上の制約と一貫性により，オブジェクトの配列の構造を容易に理解することが可能となる．

4.3　シャローコピーとディープコピーの違い

前の 4.2 節と関連して，配列に関して，シャローコピーとディープコピーの違いを意識してプログラミングしていないと，配列をコピーしたうえで，片方の配列要素が指すオブジェクトに変更を加えるといった場面で意図しない現象に遭遇することになる．int の配列をコピーするときは

```
for (int n = 0; n < intAry1.length; n++){
  intAry2[n] = intAry1[n];
}
```

といったコードで要素をコピーできるため問題はないが，String の配列の場合，同様のコード

```
for (int n = 0; n < strAry1.length; n++){
  strAry2[n] = strAry1[n];
}
```

では，**図 6** の strAry1 と strAry2 が示す状態（シャローコピー）である．完全なコピー（ディープコピー）とするためには，

```
for (int n = 0; n < strAry3.length; n++){
  strAry4[n] = new String(strAry3[n]);
}
```

のように，新しいインスタンスを生成しなければならないことを理解する必要がある．不変オブジェクトの配列の場合は問題が発生しにくいが，内部の状態が変化する可能性のあるクラス（Java における StringBuffer や Point など）のオブジェクトを配列で扱う段階でつまずきやすい．こうしたつまずきを防ぐうえでも，プログラムにおける = の意味がプリミティブの場合とオブジェクトの場合で異なる[*1]ことを早い段階で知っておくことが望ましいと我々は考えている．こうした点の理解においても，オブジェクトへの参照操作から自動的にソースコードを生成できる AG が有効に作用すると考えられる．なお AG では上記のコードにおけるループは表現できないため，授業ではループを展開して中身を 1 つずつ実行する必要がある．具体的な教示手順例としては以下のようになる．(1) まずシャローコピーの様子を視覚的に見せるため，元の String 配列を作成する．(2) コピー先の String 配列を作成する．(3) コピー先の配列の参照先を，元の配列の要素に張る（配列のサイズだけ繰り返して，シャロー完成）．(4) 要素が 2 つの配列で共有されていることを示す．(5) 次にディープコピーの様子を視覚的に見せるため，元の String 配列を作成する．(6) コピー先の String 配列を作成する．(7) コピー元の配列の参照先オブジェクトの中身と同じ String オブジェクトを作成[*2]し，コピー先の配列から参照する（配列のサイズだけ繰り返して，ディープ完成）．(8) 要素が共有されていないことを示す．

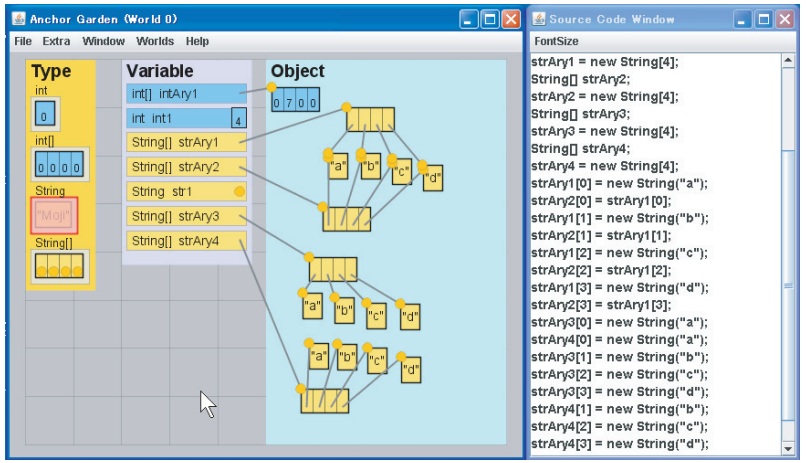


図 6　シャローコピーとディープコピー
Fig. 6　Shallow copy and deep copy.

⋆1 ただしオブジェクトの場合にも「参照情報を代入する」という考え方をすれば，両方とも「代入」と解釈できる．
⋆2 AG では String オブジェクトのコピー（クローン）を作成する操作は用意していない．そのため実際のコーディングにおける手順とは異なることを説明する必要がある．

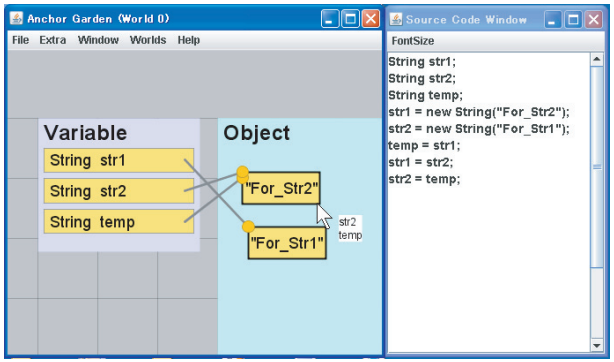


図 7　オブジェクトのスワップ
Fig. 7　Swapping objects.

4.4　オブジェクトのスワップ

2 つの変数が参照しているオブジェクトを交換する際の常套手段として，一時変数 temp を用意しておき，

```
temp = str1;
str1 = str2;
str2 = temp;
```

のようにする方法がある．AG を用いると，このコードが意味する事象を実際のリンク操作によって直感的に理解させることが可能となる．AG で 2 つの String 変数が，2 つの String のインスタンスを参照している状態で，それをリンクを切らないように（消滅しないように）交換するリンク切替え操作を行った例と，その操作によって生成されたコードを**図 7** に示す．

5.　実　　験

AG が初学者の理解にどのような影響を与えるかを調査するため，実験を行った．

5.1　概　　要

AG を用いて学習すると，オブジェクト参照の概念への理解度や，配列を扱いながらデータを処理するプログラムを記述する能力が向上することが予想される．そのため，理解度や記述能力を測るテストの結果について AG 使用群と未使用群を比較することにした．

被験者は，著者が所属する大学院の修士学生（主に 1 年次）を対象として開講している科目「基礎プログラミング（2008 年 4–5 月開講，90 分授業，全 15 回）」の受講者 22 名とした．この科目の目的はプログラミングの基礎を理解してもらうことである．講義では C#を用いて，コンソールプログラムを Visual Studio を用いて作成する演習を含めながら進めた．

講義の構成について述べる．講義の前半では，変数やデータ型（int，String），演算子，if 文，ループ（while，for）などの基本的な事項を導入した．ただし変数への代入概念については「箱の中に値を入れる」というメタファを用いた簡単な説明にとどめた．講義の後半で「配列」の概念を導入する場面で，初めて AG を用いて「参照概念」を説明した．

表 1　実験授業日程
Table 1　Schedule of the experimental lecture.

日 [時限]	A：未使用群	B：使用群
4/9 [3]	概説，環境設定	
4/16 [3]	計算機の構成と入出力	
4/21 [1]	はじめてのプログラミング	
4/23 [3]	変数・データ型・演算子	
4/28 [1]	条件分岐 (if)	
4/30 [3]	ループ (while, for)	
5/7 [3]	中間テスト・演習	
5/12 [1]	配列 (1)：導入（代入概念のみ）60 分	
	小テスト (1) 30 分	—
5/12 [3]	—	AG 使用 (1) 30 分 小テスト (1) 30 分
5/14 [3]	配列 (2)：並びかえ（代入概念のみ）	
5/19 [1]	参照概念を PPT と 白板で説明 60 分 小テスト (2) 30 分	—
5/19 [3]	—	参照概念を PPT と AG で説明&使用 60 分 小テスト (2) 30 分
5/21 [3]	関数 (1)	
5/26 [1]	関数 (2)（小テスト返却）	
5/28 [3]	模擬試験（回収・採点なし）	
6/2 [1]	最終試験	

5.2　手　　順

講義の最初に，パソコンの使用歴やプログラミング経験などについてのアンケート調査を行った．人数とプログラミング経験の有無が均一化するよう，被験者を 2 つのグループ（未使用群：A，使用群：B）に分けた．講義は**表 1** に示す日程で行った．5 月 12 日の 1 限では配列の導入を 60 分で行った後，B 群を退席させ，A 群に小テスト (1) を実施した．同日

の 3 限では B 群のみ，AG を 30 分使用（起動，拡大縮小などの基本操作，int 変数の生成と int 配列の作成について，教員が操作を提示しながら学生も一緒に操作）したのち，A 群と同じ小テスト (1) を行った．小テスト (1) の結果は成績には影響しないが，参考にする旨を伝えて実施した．小テスト (1) は，int 配列の作成と値を格納したのち，別の int 配列に逆順で格納するプログラムを作成するという課題であった（付録 A.1 参照）．被験者は電子提出ではなく，問題用紙内の空欄に回答した．

5 月 19 日は 1 限に A 群，3 限に B 群に対して，参照概念を導入する講義を行った．講義では値型と参照型の違いや，シャローコピーとディープコピーの違いを解説した．A 群と B 群で使用した講義資料（PowerPoint）は同じであったが，概念説明図については A 群では教師が板書で説明し，学生はそれをプリントやノートに写した．B 群では教師が AG を利用してプロジェクタに映示して説明した．また B 群の学生には AG を自分で操作して体験的に学ぶよう働きかけた．B 群に対する説明は，3.1 節で述べた一連の流れ（int の変数を生成 → String の変数を生成 → String のオブジェクトを生成 → リンクを張らないと消える → リンクを張ると消えない → リンクをはずすと消える）のあと，int 配列型変数を生成 → int 配列を生成，String 配列型変数を生成 → String 配列を生成 → String のオブジェクトを生成 → String 配列からリンク，という内容であった．また学生はソースコード表示機能を使用し，操作と対応するコードについても参照したが，時間が限られていたために完全に理解したとはいえない状況であった．シャローコピーとディープコピーの違いについては，教員が 4.3 節で示した流れに沿って操作しながら教示した．60 分の解説後，両グループ共通の小テスト (2) を実施した．小テスト (2) は問 1〜3 で構成されており，（問 1）「値型」「参照型」「ディープコピー」「シャローコピー」の理解を問う○×問題，（問 2）配列の作成，参照，コピーを行うプログラム中の穴埋め記述問題，（問 3）問題に示されたプログラム断片を実行し終えたときの変数とオブジェクトの関係を図で示す問題であった（付録 A.2 参照）．小テストはすべて制限時間 30 分として実施した．途中都合によりグループを変更した学生や，どちらかの講義を欠席した学生を除いた結果，A 群は 10 名，B 群は 8 名のデータを得た．小テスト開始時から 5 月 26 日までの間，両群の学生には自分たちがやっていることを他群の学生に伝えないよう教示した．なお A 群が最終試験に不利になることを避けるため，5 月 26 日に実験内容と AG について説明した．

また，講義終了後にアンケートを配布し，

- 予習の有無（Q1-1），
- 講義 1 時間あたりの予習時間（Q1-2），
- 復習の有無（Q2-1），
- 講義 1 時間あたりの復習時間（Q2-2），
- 他群の学生から何をしているか聞いていたかどうか（Q3-1），
- 聞いていたとしたらその内容はどのようなものか（Q3-2），
- 講義外で AG を使用したかどうか（Q3-3），
- 使用していたらそれは何時間何分か（Q3-4）

について尋ねた．

5.3　結果と考察

システムの効果を考察する前に，AG の使用実態調査の結果について述べる．アンケートは，メールで送付し，A 群 5 名，B 群 7 名から回答を得た（回収率，A 群：50%，B 群：87.5%）．講義の予習を行ったと回答した者は，A 群が 1 名（1 コマあたり 1 時間）であり，B 群は 3 名（同 30 分，1 時間，2 時間）であった．復習したと回答したのは，A 群が 2 名（1 コマあたり 1 時間，1 時間半）で，B 群が 6 名（同 30 分，1 時間 × 2 名，1 時間半，2 時間，2 時間半）であった．この結果からは，B 群の学生たちは予習復習に時間をかけていたことが分かる．

両群間における AG 利用について，具体的な情報交換は行われていなかった．A 群の学生 1 名は，B 群が教育用ソフトを利用していることのみ知っていた．その他の学生は，両群で何をしているのか聞いていなかったと回答した．

AG は講義時間外では利用されていなかったことも明らかとなった．A 群の学生で AG を実験終了後に使用した学生は 0 であった．B 群では 3 名であり，2 時間利用していた 1 名を除いて，20 分，30 分と短時間の利用に限られていた．

表 2 に，小テスト (1)，(2) の得点分布を示す．分散が大きいが参考として平均値も掲げた．小テスト (1)（表 2 左）では，ちょうど上位得点と下位得点の人数が半々となった．5 月 12 日の小テスト (1) では，AG を使用した B 群のほうが若干点数が低い結果となった．B 群は説明直後にテストを受けられなかった点や，被験者の一部が直前に導入した AG に問題解決のヒントが隠れていると誤解していたことが原因と考えられる．実際には AG を使って小テスト (1) の解答プログラムの答えにたどり着けるわけではなく，用途を配列の作り方や添字の意味などに限って提供していた．

小テスト (2)（表 2 右）では，B 群のみ満点の被験者がいることを除き，両群の差はほとんど見られなかった．また小テスト (1)，(2) について帰無仮説「両群の小テスト点数平均値に差がない」として t 検定を行ったが，有意差は見られなかった．原因としては AG を

表 2　小テスト (1)，(2) 結果
Table 2　Result of tests (1), (2).

小テスト (1)（満点 6）			小テスト (2)（満点 15）		
点数	A（名）	B（名）	点数	A（名）	B（名）
6	1	1	14–15	0	1
5	0	1	12–13	1	2
4	4	2	10–11	3	0
3	4	0	8–9	2	2
2	1	1	6–7	4	2
1	0	3	4–5	0	1
平均	3.6	3.0	平均	8.45	9.19
分散	1.16	4.0	分散	5.14	15.00

表 3　最終試験結果
Table 3　Result of final test.

最終試験（全体）（満点 70）			最終試験（問 5 のみ）（満点 20）		
点数	A（名）	B（名）	点数	A（名）	B（名）
60–70	0	4	19–20	0	2
50–59	2	0	16–18	0	1
40–49	4	1	13–15	1	2
30–39	2	3	10–12	4	1
20–29	2	0	7–9	1	0
10–19	0	0	4–6	3	2
0–9	0	0	0–3	1	0
平均	40.2	52.0	平均	8.0	13.4
分散	140.6	302.9	分散	21.1	38.8

使用する時間が足りず，意味を十分に理解していなかったことや，AG の表現と小テストで紙に書く図との対応を理解していなかったことが考えられる．

表 3 に，6 月 2 日に実施した最終試験の得点分布を示す．最終試験は問 1〜5 で構成されており，問 1〜3 が Visual Studio を使って与えられた条件を満たすプログラムを作成する問題（電子提出），問 4 が小テスト (2) の問 1 と同様の○×問題，問 5 が小テスト (2) の問 3 と同様の，変数とオブジェクトの関係を図示する問題であった．配点は問 1，2，4 が 10 点，問 3，5 が 20 点であった（付録 A.3 参照）．

表 3 左側が最終試験全体の得点分布，右側が問 5 のみの得点分布である．小テストと同様，t 検定を行ったところ，とくに問 5 の平均値の差に有意傾向が見られた（最終試験全体 $t(16) = -1.64$，p 両側 $= 0.127$，問 5 のみ $t(16) = -2.04$，p 両側 $= 0.063$）．このことから，AG を使用し，ある程度習熟することによって，変数とオブジェクトの関係の理解度向上につながる可能性が示唆された．今回の実験では被験者数が少なかったため有意差として現れなかったが，被験者数が多ければ有意差として現れる可能性が高い．

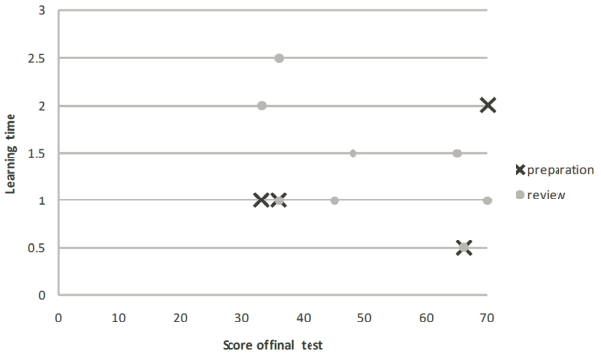


図 8　学習時間と成績
Fig. 8　Relationship between learning times and test scores.

先述のとおり，アンケート結果では，講義外で AG はほとんど利用されていなかった．さらに，AG の使用について，両群ともに情報交換は行われていなかった．したがって，これらは実験には影響を与えなかったと考えられる．

B 群学生のほうが A 群より予習復習に時間をかけていたことについてはすでに述べた．これらに時間をかけると成績向上につながる可能性は高いと考えられるため，この結果が B 群の成績が高かった理由の 1 つにあげられる．そこで，学生の予習復習の時間と試験成績の関係を考察するために散布図を描いた．その結果が**図 8** である．この結果からは，予習時間や復習時間の長さが成績の高低に直結していたわけではなかったことが示された．

講義内でフィールド実験を行う際には，学生の講義外行動を統制することは困難であるため，講義外の行動が実験に与える影響を完全に排除することはできない．今後は，今回の事前調査の項目に加え，講義に対する姿勢なども取り入れ，両群の特性を均さなくてはならない．

最後に AG が促進する「変数とオブジェクトの関係理解」とプログラム作成能力について考察する．**表 4** と**表 5** に，最終試験における各群の問ごとの得点相関を示す．両群とも問 4（○×問題）との相関が低く，問 1〜3（プログラム作成問題）と問 5（変数とオブジェクトの関係を図示する問題）との相関が高い傾向にある．とくに比較的高度な能力が問わ

表 4　最終試験 Pearson の相関係数：A 群（$n = 10$）
（*は 5%水準（両側），**は 1%水準（両側）で有意）
Table 4　Pearson's correlations of final test score (Group A).

	問 1	問 2	問 3	問 4	問 5
問 1	1	0.70*	0.60	0.14	0.59
問 2	0.70*	1	0.69*	−0.23	0.43
問 3	0.60	0.69*	1	0.31	0.83**
問 4	0.14	−0.23	0.31	1	0.50
問 5	0.59	0.43	0.83**	0.50	1

表 5　最終試験 Pearson の相関係数：B 群（$n = 8$）
（*は 5%水準（両側），**は 1%水準（両側）で有意）
Table 5　Pearson's correlations of final test score (Group B).

	問 1	問 2	問 3	問 4	問 5
問 1	1	0.51	0.93**	0.17	0.76*
問 2	0.51	1	0.36	0.60	0.84**
問 3	0.93**	0.36	1	0.23	0.70
問 4	0.17	0.60	0.23	1	0.54
問 5	0.76*	0.84**	0.70	0.54	1

れる問 3 と，問 5 の相関が高い点が共通している（表 5 の [問 3]，[問 5] の相関（0.70）の有意確率は．052）．このことから AG 使用の有無を問わず，一般に変数とオブジェクトの関係を図示する能力と，プログラム作成能力は互いに影響していることが確認された．今回の被験者の多くはプログラミング初学者であり，元々のプログラム能力のみが要因となって高い相関を示すことは考えにくい．よって変数とオブジェクトの概念理解を促す AG は，プログラム作成能力の向上に寄与する可能性が高いといえる．

6. まとめと今後の課題

静的な型付けを持つオブジェクト指向プログラミング言語におけるデータの「型」「変数」「オブジェクト」を視覚化し，さらに視覚化した各要素を操作することによって，抽象的な概念理解を促進させるためのワークベンチを開発した．学習内容が型と変数，オブジェクトの関係や配列の構造に限られているが，操作が「生成」と「リンク」の 2 種類のみであり，学習者にとって簡潔であるという利点がある．またワークベンチ上で表現された概念世界を直接操作できることにより，学習者が頭の中に思い描いているモデルとの相違点を明確にできると考えられる．AG は学習者に概念世界の中で許容された操作のみを実行させるため，学習者の誤解を軽減できる可能性がある．我々はこうしたソフトウェアを用いることにより，プログラミング学習者が実践的なオブジェクト指向プログラミング言語を学習するうえでの障壁を低めることを期待している．また教員にとっても，本ワークベンチを操作しながら解説することで，これまで言葉でなかなか説明しにくかった概念世界について，具体的な表現をもって提示できる利点がある．

実験の結果，AG に習熟すると変数とオブジェクトの関係の理解能力が向上する傾向が見られた．また変数とオブジェクトの関係理解能力とプログラム作成能力との相関が高いことが明らかとなり，AG がプログラム作成能力を向上する可能性が示された．

システム改良としては，ハッシュや 2 分木などのデータ型や，変数側からのメソッド呼び出し機能を追加することが考えられる．AG のデータ型は，視覚化のための親クラスを継承した Java のクラスとして実現しているため，型の追加は比較的容易である．また操作を再生したり，一般のソースコード断片からアニメーションを生成したりできるようにしたい．またシステムによる効果の検証を継続するとともに，システム使用時間と理解度についての知見を得たいと考えている．

AnchorGarden は，以下の URL において公開している．

http://anchorgarden.mydns.jp/

謝辞　本研究の一部は文部科学省科学研究費補助金（課題番号 20680036）の支援によるものです．

参 考 文 献

1) Moreno, A., Myller, N., Sutinen, E. and Ben-Ari, M.: Visualizing Programs with Jeliot3, *Proc. Advanced Visual Interfaces* (*AVI04*), pp.373–376 (2004).
2) Akingbade, A., Finley, T., Jackson, D., Patel, P. and Rodger, S.H.: JAWAA: Easy Web-Based Animation from CS 0 to Advanced CS Courses, *Proc. 34th SIGCSE Technical Symposium on Computer Science Education*, pp.162–166 (2003).
3) Powers, K., Ecott, S. and Hirshfield, L.M.: Through the Looking Glass: Teaching CS0 with Alice, *Proc. 38th SIGCSE Technical Symposium on Computer Science Education*, pp.213–217 (2007).
4) Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M. and Rusk, N.: Programming by Choice: Urban Youth Learning Programming with Scratch, *Proc. 39th SIGCSE Technical Symposium on Computer Science Education*, pp.367–371 (2008).
5) Kay, A.: Squeak Etoys Authoring & Media. http://www.squeakland.org/content/articles/attach/etoys_n_authoring.pdf

6) 原田康徳：Viscuit：柔らかい書き換えによるアニメーション記述言語，情報処理学会インタラクション 2004 予稿集，pp.183–184 (2004). http://www.viscuit.com/
7) 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，情報処理学会誌：プログラミング，Vol.44, No.13, pp.58–71 (2003).
8) Cooper, S., Dann, W. and Pausch, R.: Teaching Objects-first In Introductory Computer Science, *Proc. 34th SIGCSE Technical Symposium on Computer Science Education*, pp.191–195 (2003).
9) Proulx, V.K., Raab, J. and Rasala, R.: Objects from the Beginning – With GUIs, *Proc. 7th Annual Conference on Innovation and Technology in Computer Science Education*, pp.65–69 (2002).
10) Lister, R.: Teaching Java First: Experiments with a Pigs-Early Pedagogy, *Proc. 6th Conference on Australasian Computing Education*, pp.177–183 (2004).
11) Lawhead, P.B., Duncan, M.E., Bland, C.G., Goldweber, M., Schep, M., Barnes, D.J. and Hollingsworth, R.G.: A Road Map for Teaching Introductory Programming Using LEGO©Mindstorms Robots, *ITiCSE-WGR '02: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pp.191–201 (2002).
12) 長　慎也，甲斐宗徳，川合　晶，日野孝昭，前島真一，筧　捷彦：プログラミング環境 Nigari：初学者が Java を習うまでの案内役，情報処理学会論文誌：プログラミング，Vol.45, No.9, pp.25–46 (2004).
13) Cecchi, L., Crescenzi, P. and Innocenti, G.: C : C++ = JavaMM: Java, *Proc. 2nd International Conference on Principles and Practice of Programming in Java* (*PPPJ2003*), pp.75–78 (2003).
14) Moreno, A. and Joy, M.S.: Jeliot 3 in a Demanding Educational Setting, *Electronic Notes in Theoretical Computer Science* (*ENTCS*), Vol.178, pp.51–59 (2007).
15) 西田知博，原田　章，中村亮太，宮本友介，松浦敏雄：初学者用プログラミング学習環境 PEN の実装と評価，情報処理学会論文誌，Vol.48, No.8, pp.2736–2747 (2007).
16) Kölling, M., Quig, B., Patterson, A. and Rosenberg, J.: The BlueJ system and its pedagogy, *Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology*, Vol.13, No.4, pp.249–268 (2003).
17) Myers, B.A.: INCENSE: A system for displaying data structures, *SIGGRAPH Comput. Graph.*, Vol.17, No.3, pp.115–125 (1983).
18) Cross, I.J.H., Hendrix, T.D., Jain, J. and Barowski, L.A.: Dynamic Object Viewers for Data Structures, *Proc. 38th SIGCSE Technical Symposium on Computer Science Education*, pp.4–8 (2007).
19) Campbell, A.E.R., Catto, G.L. and Hansen, E.E.: Language-Independent Interactive Data Visualization, *Proc. 34th SIGCSE Technical Symposium on Computer Science Education*, pp.215–219 (2003).

付　　録

A.1　小テスト (1) 課題

問題：以下の動作をするプログラムを書きなさい．（Main の { } の中のみ）

（1）　ユーザに 6 個の整数を入力してもらい，整数型の配列（ary）に，先頭から順番に，格納する．

（2）　もう 1 つ整数型の配列（revary）を用意し，元の配列に格納された数値を逆順で格納（コピー）する．

（3）　for 文を用い，元の配列（ary）と，逆順に格納した配列（revary）に格納したデータを画面に表示する．

例：ユーザが 1 4 7 9 2 3 の順番で整数を入力したときの（ary）と（revary）

ary	[0]	[1]	[2]	[3]	[4]	[5]
	1	4	7	9	2	3

revary	[0]	[1]	[2]	[3]	[4]	[5]
	3	2	9	7	4	1

A.2　小テスト (2) 課題

問 1：以下の文が正しい場合には○を，間違っている場合×をつけ，間違いを指摘しなさい．

（1）　int[] は「値型」である．

（2）　String[] は「参照型」である．

（3）　StringBuilder は「値型」である．

（4）　配列をコピーする場合，ディープコピーよりもシャローコピーを用いたほうが，実行時のメモリを多く必要とする．

（5）　配列をコピーする場合，ディープコピーはシャローコピーに比べ，データをコピーする処理が余計にかかる．

問 2：以下のプログラム中の下線部に，コメント文 (//) で指示された処理を記入しなさい．

```
Console.WriteLine("英文を入力してください (Input A Sentence in English).");
String  sentence = Console.ReadLine();
String[] words = sentence.Split(' ');
```

```
String[] copywords = _________________ // words と同じ長さの配列を準備
for( int n = 0; n < words.Length ; n++) {
   if (n % 2 == 0) {
        ________________________ // copywords[n] が words[n] を参照する
   } else {
        ________________________ // copywords[n] が words[n] と
                                 // 同じ内容の新しい文字列を参照する
   }
}

int[] orig = new int[words.Length];
for ( int n = 0; n < words.Length ; n++) {
   orig[n] = words[n].Length
}
int[] copy = new int[words.Length];

// for 文をつかって orig の要素をすべて copy に代入する
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
```

問 3：以下のプログラムを最後まで実行したときの，メモリの状態（変数とオブジェクトへの参照状態）を「例 1〜3」と同じように矢印をつかって右の枠内に図示しなさい．

例 1：（複数の変数から 1 つの文字列への参照）

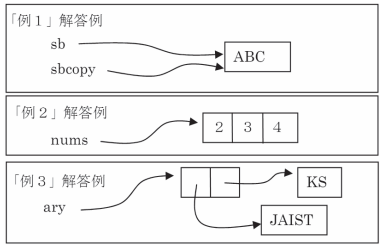
```
StringBuilder sb = new StringBuilder("A");
StringBuilder sbcopy = sb;
sbcopy.Append("BC");
```

例 2：（数値は配列の箱のなかに入る）

```
int[] nums = new int[]{2,3,4};
```

例 3：（文字列は配列の箱のなかに入れない）

```
String[] ary = new String[]{"JAIST","KS"};
```




```
(1)  String[] abc = new String[]{"A","B","C"};
     String[] copy = new String[3];
     for (int n = 0; n < 3; n++) {
         copy[n] = abc[n];
     }
(2)  String[] abc = new String[]{"A","B","C"};
     String[] copy = new String[3];
     for (int n = 0; n < 3; n++) {
         copy[n] = new String(abc[n]);
     }
(3)  StringBuilder[] abc = new StringBuilder[]{new StringBuilder("A"),
                                new StringBuilder("B"),
                                new StringBuilder("C")};
     StringBuilder[] copy = new StringBuilder[3];
     for (int n = 0; n < 3; n++) {
         copy[n] = abc[n];
     }
     copy[0].Append("X");
     copy[1].Append("Y");
     copy[2].Append("Z");
```

A.3　最終試験課題

問 1：以下のプログラム（関数定義のみ）を作成しなさい．（10 点）

static int GetSumAry(int[] ary)

関数 GetSumAry は，引数で得た配列の要素を「すべて足し合わせた値」を返す．

例：引数で得た配列が [2,3,5,4,6] であった場合は，$2+3+5+4+6=20$ を返す．

問 2：以下の動作をするプログラムを「Main 関数内」に作成しなさい．（10 点）

(1) ユーザに，英文（文字列）を入力してもらう．

(2) 英文を半角スペースで区切り，単語（文字列）の配列を得る．

(3) 単語の文字数が「3 以上」の単語だけを，改行なし（1 行）で表示する．

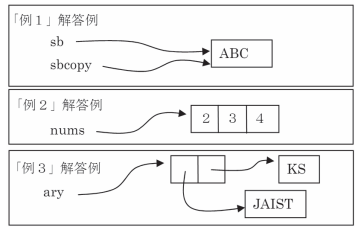
(4) 最後に「改行」する．

動作例：`This is a pen.`　（← ユーザの入力文字列）
`This pen.`　（← プログラムの出力）

問 3：以下の動作をするプログラムを「Main 関数内」に作成しなさい．（20 点）
(1) 整数を 100 個格納する配列（ary）を作成する．
(2) 配列（ary）のすべての要素に，乱数（0 以上 99 以下の整数）を格納（代入）する．
(3) 整数を 10 個格納する配列（divary）を作成する．（ary とは別の配列として作成する）
(4) ary の要素について，3 で割り切れる場合は divary[3] を 1 増やす．4 で割り切れる場合は divary[4] を 1 増やす．というように，3〜9 のどの数字で割り切れるかをチェックし，divary[] にカウントする．結果として，divary[m] には，m で割り切れる ary 要素の「個数」がはいる．（m は 3 以上 9 以下）
(5) divary[3] から [9] までの内容を，数値および棒グラフで表示する．
動作例：
```
divary[3]= 38 : **************************************
divary[4]= 26 : **************************
divary[5]= 25 : *************************
divary[6]= 20 : ********************
divary[7]= 19 : *******************
divary[8]= 14 : **************
divary[9]= 11 : ***********
```
（注 1）プログラムは for 文のループを活用し，なるべく「短く」かつ「簡潔に」する．（注 2）「複数の数」で割り切れる場合は，すべてカウントすること（たとえば，24 は 3，4，6，8 の「4 つの数」で割り切れる）．

問 4：以下の文が正しい場合には○を，間違っている場合×をつけ，間違いを指摘しなさい．（各 2 点/合計 10 点）
（1）　float[] は「値型」である．
（2）　int[] は「参照型」である．
（3）　StringBuilder は「値型」である．
（4）　ディープコピーはシャローコピーよりも，結果としてメモリ内のオブジェクトの数が多くなる．
（5）　1 つのプログラムファイル（ソースコードファイル）内に複数の関数定義を書いて実行した場合，「プログラムファイル中で一番上に（最初に）定義した関数」が最初に呼び出される．

問 5：以下の（1）〜（4）のプログラムを最後まで実行したときの，メモリの状態（定義済み変数とオブジェクト，およびそれらの参照状態）を「例 1〜3」にならって，矢印をつかって図示しなさい．（各 5 点/合計 20 点）



例 1：（複数の変数から 1 つの文字列への参照）
```
StringBuilder sb = new StringBuilder("A");
StringBuilder sbcopy = sb;
sbcopy.Append("BC");
```
例 2：（数値は配列の箱のなかに入る）
```
int[] nums = new int[]{2,3,4};
```
例 3：（文字列は配列の箱のなかに入れない）
```
String[] ary = new String[]{"JAIST","KS"};
```

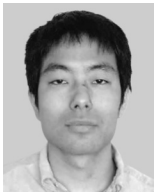
```
(1)   String[] schools = new String[]{"IS","MS","KS"};
      String[] copy = new String[3];
      copy[2] = schools[2];
(2)   String[] schools = new String[3];
      String[] copy = schools;
      schools[0] = "KS";
      schools[1] = schools[0];
      copy[2] = "IS";
(3)   int[] nums = new int[]{1,2,3,4,5};
      int[] copy = new int[4];
      for ( int  n=0 ; n<4 ; n++) {
         copy[n] = nums[n] + nums[n + 1] ;
      }
(4)   int[] orig = new int[]{2,4,6};
      int[] copy = orig;
```

```
int[] backup = new int[3];
for ( int n=0 ; n<3 ; n++) {
   backup[n] = copy[n];
}
copy[1] = 8;
```

(平成 20 年 11 月 25 日受付)
(平成 21 年 7 月 2 日採録)

三浦　元喜（正会員）

1997 年筑波大学第三学群情報学類卒業．2001 年筑波大学大学院博士課程工学研究科修了．博士（工学）．同年筑波大学電子・情報工学系助手．2004 年より北陸先端科学技術大学院大学知識科学研究科助手（2007 年より助教）．2009 年より九州工業大学大学院工学研究院基礎科学研究系准教授．ヒューマンコンピュータインタラクション，モバイルインタフェース，教育支援等に関する研究に従事．ACM，ヒューマンインタフェース学会，電子情報通信学会，IEEE CS，人工知能学会，日本教育工学会，日本ソフトウェア科学会等各会員．



杉原　太郎

1977 年生．2000 年徳山工業高等専門学校専攻科機械制御工学専攻修了．2002 年京都工芸繊維大学大学院工芸科学研究科博士前期課程修了．2005 年同研究科博士後期課程修了．博士（工学）．同年 4 月より北陸先端科学技術大学院大学知識科学研究科助手．2007 年 4 月より助教．現在は主として情報機器に関するユーザ行動分析の研究に従事．ヒューマンインタフェース学会，日本感性工学会，ACM 各会員．

國藤　　進（正会員）

1974 年東京工業大学大学院理工学研究科修士課程修了．同年（株）富士通国際情報社会科学研究所入所．1982〜1986 年 ICOT 出向，1992 年より北陸先端科学技術大学院大学情報科学研究科教授，1998 年より知識科学研究科教授，現在は主として発想支援システム，グループウェア，知識システムの研究に従事．情報処理学会創立 25 周年記念論文賞，情報処理学会 DICOMO2008 シニアリサーチャ賞，ほか受賞多数．博士（工学）．計測自動制御学会，電子情報通信学会，日本創造学会等各会員．