初学者向けデバッガDENOの設計と アルゴリズム構築能力育成授業への適用効果

袴田 大貴[1,a)]　松澤 芳昭[1,b)]　太田 剛[1,c)]

概要：本研究の仮説は，初学者がプログラムの実行動作を理解するのにデバッガが有用だということである．既存のデバッガはプロフェッショナルのために開発されたものであるため，初学者にとって利用法習得における負荷が大きいという問題がある．本研究では，プログラムの実行動作理解を支援する初学者向けデバッガ「DENO」を開発した．DENO の特徴は（1）省略可能なブレークポイント，（2）ワンボタンステップ実行である．文科系大学生向けプログラミング入門授業で学生 108 名に対して適用実験を行った．実験では，挿入ソートのフローチャートを設計した学生達が Java で実装を行い，その動作確認のために DENO を使用することが期待された．学習者のエディタ操作記録を用いて，作業時間の量的分析と作業プロセスの質的分析を行った．結果，DENO で動作確認をした学生が 11 %，sleep メソッドの適宜挿入による速度調整で動作確認をした学生が 34 %，両方併用した学生が 35 %であった．DENO のみを利用した学生が DENO を使うことで正しく動作確認できていることを確認できた．

1. はじめに

プログラミング教育において重要なことの 1 つはプログラムの実行動作を理解することである．プログラミング初学者はプログラムの実行動作理解が不十分であるため，アルゴリズムが理解できているのにも関わらず，それを実現するコードを実装できないという事態が発生する．今泉らは，初学者は制御構造，関数を含んだプログラムの適切な実行動作イメージが作成できないと述べている [1].

初学者がプログラムの実行動作を理解するためには，デバッガが有用であるというのが本研究の仮説である．山本らはプログラムの実行動作の理解が難しい原因をプログラムの動きが目に見えないことにあると述べている [2]．西田らはプログラムの実行状況を観察できるようにすることでプログラムの動作を理解しやすくなると述べている [3].

ところが既存のデバッガには，初学者にとって利用法習得における負荷が大きいという問題がある．なぜなら既存のデバッガはプロフェッショナルのために開発されているからである．例えば，制御フローを可視化する機能の 1 つである「ブレークポイント」は，初学者が適切に設定するのは難しいと言われている [4].

本研究では，プログラムの実行動作理解を支援する初学者向けデバッガ「DENO」を提案する．我々が行なっている文科系大学生向けプログラミング入門授業で適用実験を行った．対象回の目的は「アルゴリズム構築能力育成」[4] である．そこで得られた知見について本稿で述べる．

2. 先行研究

一般的なデバッガの例として Eclipse JDT が挙げられる．Java のデファクトスタンダート IDE「Eclipse」に搭載されたデバッガであり，ブレークポイントを基にステップ実行や変数値参照などの機能が GUI で提供されている．しかし，1 章で述べたように初学者が利用するには負担が大きい．例として，Debug ビュー・BreakPoint ビュー・Outline ビューなど情報量過多，ブレークポイントの適切な設定，ステップ実行の適切な使い分けが挙げられる．

初学者用プログラミング学習環境として西田らの PEN[3] がある．プログラムの実行状態表示機能を持つが，当該機能単体の評価がなされていないことが問題点である．

デバッグ支援システムとして江木らの DESUS[5] がある．システムによってデバッグプリントによるトレース指導が行われるものである．しかし，デバッガを用いることでデバッグプリント無しでトレースを行うことができる．

[1] 静岡大学情報学研究科
Graduate School of Informatics, Shizuoka University
[a)] gs12029@s.inf.shizuoka.ac.jp
[b)] matsuzawa@inf.shizuoka.ac.jp
[c)] ohta@inf.shizuoka.ac.jp

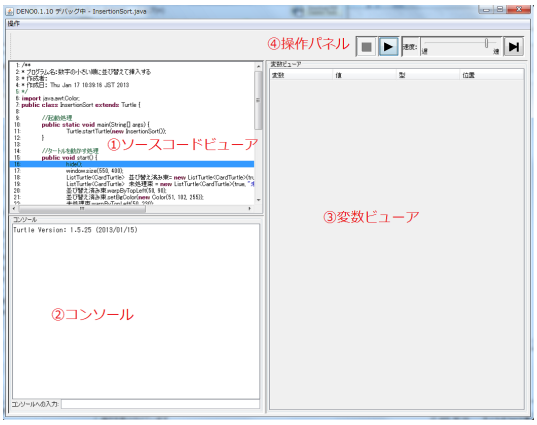

図 1　デバッガの動作画面

3.　DENO

3.1　DENO とは

本研究では，プログラミング初学者に対し，プログラムの記述とその実行動作理解を支援するデバッガ「DENO」を開発した．対象言語は Java である．DENO は JDK に同梱されている JDI(Java Debug Interface) のデモである簡易 GUI デバッガ「javadt」を基盤として開発し，我々が Java 教育に用いている学習環境に組み込んだ．

DENO の動作画面を**図 1** に示す．DENO は 4 つのフィールドから構成されている．左上の 1 は「ソースコードビューア」である．対象プログラムのソースコードが表示され，次に実行される行は青色で表示される．左下の 2 は「コンソール」である．プログラムの標準出力はこのフィールドに出力される．プログラムへの標準入力はこのフィールド下部にある入力欄から行う．右下の 3 は「変数ビューア」である．その時点で生存している全ての変数が表示される．実行中のスコープ内の変数はハイライト表示される．右上の 4 は「操作パネル」である．ここからステップ実行などの操作を行う．

3.2　DENO の特徴

3.2.1　省略可能なブレークポイント

DENO はブレークポイントを使用せずともデバッグが可能なように設計した．吉村らの研究 [6] によると，ブレークポイントはプログラムの実行動作を理解していなければ適切に設定することができず，初学者にとっては困難である．

そこで，従来のブレークポイントを設定する方式ではなく，デバッガ起動時にプログラム先頭で停止し，以降はステップ実行のみでプログラム終了まで動作する方式にした．これによって，利用者はブレークポイントの設定位置に悩むことなく，デバッグを行うことができる．

この方式のデメリットは，ステップ数が多い場合，ステップ実行ボタンを押す回数が増えることである．しかしながら，初学者が作成するプログラムは，高々 30〜50 ステップであるため，問題にはならないと考えた．

3.2.2　ワンボタンステップ実行

DENO はステップ実行の使い分けを意識せずともデバッグが可能なように設計した．一般的なデバッガにはステップイン，ステップオーバー，ステップアウトという 3 種のステップ実行機能がある．これを状況に応じて使い分ける必要があり，初学者にとっては難しい．

DENO では，ステップ実行 3 種を統合し自動で使い分けられるようにした．これによって，利用者が使い分けを意識する必要はなくなった．

自動使い分けの基準は，対象のメソッドがユーザの自作メソッドであるか否かである．これによって，ユーザ自身が作成したコードにのみ集中することができる．

ステップアウトは，学生が行う課題では高々 5 ステップ程度のメソッドしか実装しないため，不要だと考えた．

4.　実験

4.1　目的と仮説

実験目的は，提案システムについて，初学者が利用可能であること，プログラム実行動作の理解が深まることを評価することである．

我々が立てた仮説を以下に示す．

仮説 1　全ての学生が DENO を利用してプログラムの動作確認を行う．

仮説 2　DENO を使うことで課題正答率が向上する．

仮説 3　DENO を使うことで作業時間が短くなる．

4.2　実施環境

著者所属学部で 2012 年度後期に行われた，文科系の学生を対象にした授業「プログラミング」の第 12-13 回において使用実験を行った．この授業は 108 名の学生が受講しており，教員 2 名とティーチングアシスタント 6 名が配置されている．

実験は第 12-13 回の授業で行った．この回の目的は「アルゴリズム構築能力の育成」である．第 1 回から第 4 回の授業では，タートルグラフィックスを利用した図形描画，第 5 回から第 6 回の授業でアニメーション作品の制作，第 7 回から第 8 回ではコンソールアプリケーションについて学ぶ．後半は，メソッド，集合データ構造，アルゴリズムについて学習する．中間課題では個人で自由に作品を制作し，最終課題ではグループで作品を制作する．

習得目標の言語は Java であるが，ビジュアル言語とテキスト型言語（Java）の相互変換ができる「BlockEditor[7]」も併用できる環境が与えられている．

4.3　演習課題

実験を行った第 12-13 回の授業は，杉浦ら [4] が「ことだま on Squeak」で行った実験を「Java」で行うものであ

表 1　演習一覧

目的	名称	概要
集合データ構造の理解	アニメーションの作成	ListTurtle で複数の画像を扱いアニメーションを作成する
手作業によるアルゴリズムの理解と実行	手作業による最小値選択法	フローチャートを参考に選択法のアルゴリズムを手作業で実行し理解する
DENO の使い方を学習	TenCards	ListTurtle に 1〜10 の数字を入れてシャッフルするプログラムを作成しデバッガの使い方を学習する.
アルゴリズム構築能力の育成	挿入法の構築	動画を基に挿入法のフローチャートを作成しプログラミングする

る．演習一覧を表 1 に示す．

4.3.1　演習１：アニメーションの作成

演習内容は，第 5-6 回の授業で作成したアニメーション作品を，集合データ構造 (ListTurtle) を用いて改良することである．杉浦らの研究 [4] でも，「ことだま on Squeak」を用いて，データ構造の理解のためにアニメーションの作成を行なっている．

4.3.2　演習２：手作業による最小値選択法

演習内容は数字が書かれたカードを用いて手作業による選択ソートを体験することである．杉浦らの研究 [4] では，選択ソートのコーディングも行なっているが，我々の授業では時間の関係で，選択ソートのソースコードを示すのみで，学生はコーディングを行なっていない．

4.3.3　演習３：TenCards

リスト構造に 1〜10 までの数字を入れる，アニメーションで 1 コマごとに各数字をカーソルが指し示すようにする，1 秒に 1 回リストの中身をシャッフルし先頭の値を出力するという 3 つの小課題で構成されている．実装したプログラムの動作確認を DENO で行わせることで，DENO を使う機会を強制的に作り使い方を体験させた．課題提出者は 99 人で，DENO 使用者は 87 人であった．

4.3.4　演習４：挿入法の構築

演習内容は，動画を元に挿入ソートのフローチャートを書き，書き終えた者からその実装を行うことである．実装自体は授業時間内に終える必要のない形式で，授業時間内に終えた学生は 5 人であった．授業終了時にフローチャートの模範解答を配布した．

2011 年度のプログラムの動作例を**図 A·1**, 2012 年度の動作例を**図 A·2** に示す．図 A·1 の「ステップ」ボタンが，「押されるまで一時停止させるボタン」である．DENO を利用することで，一時停止させるボタン無しで，プログラムの動作確認ができるようになった．

2011 年度のソースコードを**図 A·3**，2012 年度のソースコードを**図 A·4** に示す．ListTurtle クラスは集合データ構造を扱うクラスで，図 A·2 の未処理束，並替済束がそれに該当する．赤い枠はカーソルである．sleep メソッドは任意の時間一時停止するメソッド，update メソッドは画面の更新をするメソッドである．

このプログラムでは，リスト構造内のデータ，カーソル位置が全てグラフィカルに表示されるので，DENO の変数確認機能は不要である．

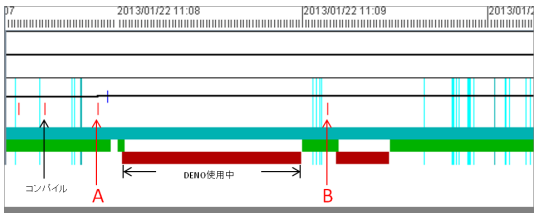

図 2　操作ログを視覚化したタイムラインの例

4.4　使用するデータと採取方法

開発環境付属の操作記録保存機能を用いてプログラミングの過程の記録を行い，以下のデータを採取した．コーディング時間には，フローチャート作成にかかった時間は含まない．

- Tc：コーディング時間
- Td：DENO 使用時間
- Tw：作業時間 （Tc + Td）
- Rd：DENO 利用率 （Td / Tw）

課題の正否は，提出された課題のソースコードを実際に実行することで，○（正答）×（誤答）で評価した．

筆頭筆者が Programming Process Visualizer[8] を用いて作業プロセスの質的分析を行った．Programming Process Visualizer[8] によって操作ログを視覚化したタイムラインの例を**図 2** に示す．中央下部の焦げ茶色の長方形が DENO を使用していることを表している．矢印で示した赤線はコンパイルしたことを表している．

質的分析では，DENO 使用前後直近のコンパイル時 (A, B) のソースコードを比較した．A の時点のソースコードに含まれる幾つかの修正が必要な箇所を B の時点のソースコードで 1 つ以上修正できていれば，正しくデバッグを行えているとした．

5.　実験結果

5.1　量的分析

5.1.1　概要

課題の正否と動作確認手段別の人数，平均作業時間，平均 DENO 利用率を表 2 に示す．DENO のみは DENO を使って動作確認した者を示し，sleep のみは sleep メソッド

表 2 課題の正否と動作確認手段別の人数，平均作業時間，平均 DENO 利用率

	人数	DENO 使用		DENO 不使用	
		DENO のみ	両方使用	sleep のみ	両方不使用
○	80 人 (74 %)	10 人 (9 %) Tw=90 分 Rd=9.28	34 人 (33 %) Tw=94 分 Rd=5.57	31 人 (28 %) Tw=49 分 Rd=0	4 人 (4 %) Tw=53 分 Rd=0
×	16 人 (15 %)	2 人 (2 %) Tw=139 分 Rd=3.78	2 人 (2 %) Tw=168 分 Rd=1.79	6 人 (6 %) Tw=30 分 Rd=0	7 人 (6 %) Tw=160 分 Rd=0
未提出	12 人 (11 %)	-	-	-	-

の適宜挿入による速度調整で動作確認した者を示す．両方使用は DENO と sleep メソッドを併用した者，両方不使用は DENO も sleep メソッドも不使用だった者である．

sleep メソッドによる動作確認法は提出されたソースコード，および作業プロセスの分析によって発見された．図 A·4 の 26 行目・31 行目に sleep メソッドを挿入することで，カーソルを移動したときと並替済束に値を移したときに 1 秒間プログラムが停止する．これによって DENO の 1 行単位の停止とまではいかなくとも，ソートの過程を確認するのに最低限の一時停止をすることができる．

DENO で動作確認した人数は，課題提出者の 50 %であった．sleep メソッドを使用した者が多かった理由としては，画面を更新する update メソッドとセットとして教えられていたこと，ソートの前段階であるカードをリストに挿入する部分は約 40 ステップあり，DENO で毎回ステップボタンを押すことが手間であったと考えられる．

5.1.2 課題の正否

DENO の課題の正否への影響を分析する．

正答者は 108 人中 80 人で，正答率は 74 %であった．杉浦らの研究 [4] では，実験を行った学生全員が挿入ソートのプログラムを作ることができた．正答率低下の原因は，実装言語の違いによるものだと考えられる．これを支持する結果として，2011 年度の正答率が 55 %であったことが挙げられる．2012 年度の正答率が 2011 年度に比べて向上した要因としては BlockEditor や DENO が考えられる．

正答者と誤答者の割合は DENO 使用者が 44:4，DENO 不使用者が 35:13 である．この結果から，DENO 使用者の正答率が高いことが分かる．

5.1.3 作業時間と DENO 利用率 Rd

本稿の実験と杉浦らの実験 [4] の作業時間の差異について分析する．正答者の平均作業時間は 73 分であった．杉浦らの研究 [4] では，平均 23 分でコーディングを終えている．このようになった要因としては，前述のプログラミング言語の違いの他に，課題の実施環境が考えられる．我々の実験では，ほぼ全ての学生が講義中にはフローチャート作成までしか終えることができなかった．それ故に，挿入ソートのコーディング課題は宿題という形式になっていた．その結果，作業意欲の低下を招き，正答率低下，作業時間の増大に繋がったと考える．

DENO の作業時間への影響を分析する．DENO 利用率は最も多い DENO で動作確認をした者でも 9.28 %，約 8 分である．この結果から，DENO を使用することは作業時間が長くなった原因ではないと考える．

正答者の作業時間について分析する．平均作業時間は DENO 使用者は 92 分，DENO 不使用者は 51 分で 41 分の差がある．平均 DENO 利用率は 7.42 %なので，DENO 使用時間は約 7 分である．つまり DENO 使用者はコーディング時間が約 33 分長い．この理由は「プログラミングが得意な人は DENO を使わず，苦手な人が DENO を使った」からだと考える．得意な人はプログラムの実行動作の理解度が高く，デバッグも自分なりの手法を確立しているので，あえて DENO を使う理由がなく，自分なりの手法（sleep メソッド）を使用したと考えられる．苦手な人はプログラムの実行動作の理解度が低く，デバッグも不得手としているので，DENO を頼ったと考えられる．したがって，コーディング時間の約 33 分の差は，苦手な人は得意な人に比べて，アルゴリズムをコードで表現するのに時間がかかることを示していると考える．

DENO のみを利用した者は，DENO と sleep メソッドを併用した者に比べて，DENO 利用率が約 2 倍という結果になった．これは，DENO と sleep メソッドを同程度使用したことを示すと考える．

誤答者の DENO 利用率に対して，正答者の DENO 利用率は，DENO のみの者は約 2.5 倍，DENO と sleep メソッド併用の者は約 3 倍である．これはそれだけ DENO を使って理解しようとしたことを示していると考える．正答者は誤答者に比べて DENO によって動作を理解した結果として，正答に至ったと考えられる．

5.2 質的分析

5.2.1 正答者は正しくデバッグできているか

DENO のみを使用した 10 人の操作ログを分析し，DENO を使用することで適切な箇所を修正できたかを調べた．「適切な箇所を修正」とは，DENO 使用時点のソースコードに含まれるいくつかの問題箇所を，DENO 使用後に 1 つ以上修正できていることを指す．その結果，全員が適切な箇所

を修正することができていた．この結果は，DENO による効果だと考えられる．

5.2.2 誤答者は正しくデバッグできているか

誤答者のうち，DENO を利用した 4 名の DENO 使用後の編集箇所を確認し，適切な箇所を修正できているかを検証した．その結果，4 人中 3 人が適切な箇所を修正することができていた．これは DENO による効果だと考えられる．残りの 1 人は模範解答フローチャートと明らかに異なるソースコードを構築しているので，フローチャートについて詳細に教えることで正答に導くことができると考える．

6. 考察

6.1 仮説 1：全ての学生が DENO を利用してプログラムの動作確認を行う

初学者のデバッガの自発的使用率は 50 %という結果であった．これは仮説 1 を棄却する結果である．

第 12-13 回という講義終盤になって導入したにも関わらず，50 %もの学生が使用した．これはブレークポイントの設定，ステップ実行の使い分けという初学者には難しい作業無しでデバッグを行うことができる DENO の特徴が影響していると考えられる．プログラミングが苦手な学生が DENO を使用していた結果はこの考察を補強する．

DENO の使用人数が少なかった理由として，第 1 に sleep メソッドの存在が挙げられる．sleep メソッドは一度記述してしまえば，以降は普段通りプログラムを実行することで動作確認を行うことができる．DENO を使うためには，普段と異なる操作をする必要がある．

第 2 に DENO の不便さが挙げられる．DENO はプログラムの最初から最後までステップ実行を行う必要がある．プログラムのステップ数が大きくなるほど，ステップ実行ボタンを押す手間が増える．本稿の実験においても，ソートの前段階を何回もステップ実行するのは手間であったと考えられる．

6.2 仮説 2：DENO を使うことで課題正答率が向上する

正答者と誤答者の割合は，DENO 使用者は 44:4，DENO 不使用者は 35:13 であった．この結果は仮説 2 を支持するものである．この支持を揺るがすものとして，DENO 使用者には sleep メソッドを併用した者が含まれており，DENO と sleep メソッドのどちらの影響が大きいかは評価できていないことがある．

2011 年度の正答率 55 %に対して，2012 年度の正答率 84 が 74 %であることも仮説 2 を支持する．しかし，2011 年度と 2012 年度の差異は DENO 以外に BlockEditor があるので，全てが DENO の効果によるものとは言えない．

6.3 仮説 3：DENO を使うことで作業時間が短くなる

正答者の内，DENO 使用者の平均作業時間が 92 分，DENO 不使用者の平均作業時間が 51 分であった．この結果は仮説 3 を棄却するものである．DENO 使用時間が約 7 分であることから，作業時間の差はコーディング時間によるものが大きい．

7. まとめ

プログラムの実行動作理解支援を目的として初学者向けデバッガ DENO を開発し，実際の演習で利用した結果を分析した．得られた知見は以下の 3 つである．

(1) 講義終盤の導入にも関わらず 50 %の学生が DENO を使用したのは，省略可能なブレークポイント，ワンボタンステップ実行という DENO の特徴が寄与した．

(2) DENO によって課題正答率は向上する．DENO と sleep メソッドのどちらの影響が大きいかは評価できていない．

(3) 作業時間には DENO 使用時間よりもコーディング時間のほうが影響が大きい．

DENO を使用することで課題正答率が向上するという知見は，デバッガがプログラミング実行動作理解に有用であるという本研究の仮説を支持する結果である．

参考文献

[1] 今泉俊幸，橋浦弘明，松浦佐江子，古宮誠一：ブロック構造の可視化によるプログラミング学習支援環境 azur ～関数の動作の可視化～，電子情報通信学会技術研究報告. ET, 教育工学，Vol. 109, No. 193, pp. 45–50 (2009).

[2] 山本義一，辻野嘉宏，都倉信樹：プログラミング教育を目的としたチャート型言語システム，電子情報通信学会技術研究報告．ET，教育工学，Vol. 94, No. 425, pp. 49–56 (1994).

[3] 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. 44, No. 8, pp. 2736–2747 (2007).

[4] 杉浦　学，松澤芳昭，岡田　健，大岩　元：アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果，情報処理学会論文誌，Vol. 49, No. 10, pp. 3409–3427 (2008).

[5] 江木鶴子，竹内　章：プログラミング初心者にトレースを指導するデバッグ支援システムの開発と評価，日本教育工学会論文誌，Vol. 32, No. 4, pp. 369–381 (2009).

[6] 吉村巧朗，亀井靖高，上野秀剛，門田暁人，松本健一：ブレークポイント使用履歴に基づくデバッグ行動の分析，電子情報通信学会技術研究報告：信学技報，Vol. 109, No. 307, pp. 85–90 (2009).

[7] 松澤芳昭，酒井三四郎：ビジュアル型言語とテキスト記述型言語の併用によるプログラミング入門教育の試みと成果，情報処理学会研究報告．コンピュータと教育（CE），Vol. 119, No. 2, pp. 1–11 (2013).

[8] 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 (SSS2012), pp. 257–264 (2012).

付　　録

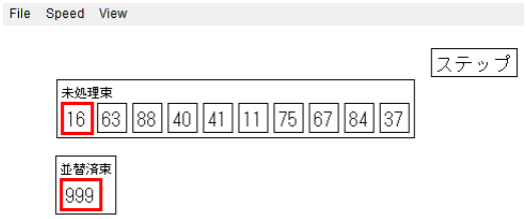

図 A·1 課題プログラムの動作例（2011）

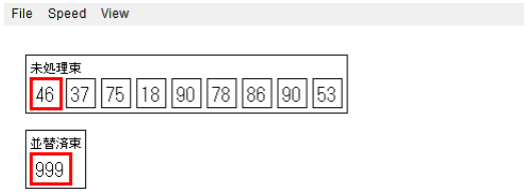

図 A·2 課題プログラムの動作例（2012）

```
1  public class InsertionSort extends Turtle {
2    public void start() {
3      // カードの束を作る
4      ListTurtle<CardTurtle> 未処理束 =
5        new ListTurtle<CardTurtle>(true, "未処理束");
6      ListTurtle<CardTurtle> 並替済束 =
7        new ListTurtle<CardTurtle>(true, "並替済束");
8      // ステップボタンの設置
9      ButtonTurtle stepButton
10      = new ButtonTurtle("ステップ");
11     stepButton.warpByTopLeft(400, 20);
12     // カードを 10 枚作り、未処理束に入れる
13     for (int i = 0; i < 10; i++) {
14       未処理束.addLast(new CardTurtle(random(100)));
15       waitForPushButton(stepButton);
16     }
17     並替済束.addLast(new CardTurtle(999)); // 番兵
18     waitForPushButton(stepButton);
19     // 以下，挿入法
20     while (未処理束.getSize() > 0) {
21       並替済束.setCursor(0);
22       while (並替済束.getObjectAtCursor().getNumber()
23         < 未処理束.getObjectAtCursor().getNumber()) {
24         並替済束.moveCursorToNext();
25         waitForPushButton(stepButton);
26
27       }
28       並替済束.addToCursor(
29         未処理束.getObjectAtCursor());
30       waitForPushButton(stepButton);
31
32     }
33     waitForPushButton(stepButton);
34   }
35   // ステップボタンが押されるまで待つ
36   void waitForPushButton(ButtonTurtle button) {
37     do {
38       update();
39       sleep(0.025);
40     } while (!(button.isClicked()));
41   }
42 }
```

図 A·3 課題プログラムの解答例（2011）

```
1  public class InsertionSort extends Turtle {
2    public void start() {
3      // カードの束を作る
4      ListTurtle<CardTurtle> 未処理束 =
5        new ListTurtle<CardTurtle>(true, "未処理束");
6      ListTurtle<CardTurtle> 並替済束 =
7        new ListTurtle<CardTurtle>(true, "並替済束");
8
9
10
11
12     // カードを 10 枚作り、未処理束に入れる
13     for (int i = 0; i < 10; i++) {
14       未処理束.addLast(new CardTurtle(random(100)));
15       update();
16     }
17     並替済束.addLast(new CardTurtle(999)); // 番兵
18     update();
19     // 以下，挿入法
20     while (未処理束.getSize() > 0) {
21       並替済束.setCursor(0);
22       while (並替済束.getObjectAtCursor().getNumber()
23         < 未処理束.getObjectAtCursor().getNumber()) {
24         並替済束.moveCursorToNext();
25         update();
26         // sleep(1);
27       }
28       並替済束.addToCursor(
29         未処理束.getObjectAtCursor());
30       update();
31       // sleep(1);
32     }
33     update();
34   }
35
36
37
38
39
40
41
42 }
```

図 A·4 課題プログラムの解答例（2012）