ソフトウェア紹介

プロトタイプ階層を持つ教育用オブジェクト指向言語「ドリトル」

兼宗 進　久野 靖

1　はじめに

ドリトル[1][2][3]は筆者らが設計した言語環境である．初心者が楽しさを感じて「プログラムを作っていきたい」と思える言語を目指している．

言語は Smalltalk 風のブロックと JavaScript 風のオブジェクトのプロトタイプ階層を持つ．英語のほかに，利用される現地の言葉で命令語や変数名を用いてプログラムを記述できる．

処理系を公開し，ダウンロードして利用する形に加え，Web ブラウザ上でも動作する．主に，中学校，高校，大学の授業で利用されている．

2　ドリトルの言語設計

ドリトル言語の設計に際しては，次の方針を立てた．

- 中学校，高等学校で教育目的に有効に使えるような言語および環境とする．このため，中学生にとって学習が難しいと思える概念の理解が必須でないような言語設計とする．
- テキストによる記述という，伝統的なプログラミング言語の枠組みを維持する．これは，学習者に一般的なプログラミング言語とはどのようなものかを体験して欲しいという意図から選択した．ただし，記述に際しては日本語の利用や構文の工夫などで，学習の敷居をできるだけ低くする．
- 教育用ではあっても，現代のプログラミング言語が持つ柔軟性や拡張性をきちんと持った言語とする．具体的には，オブジェクト指向を取り入れることで，必要に応じてオブジェクト群を追加して多様な題材の教育に適用可能とする．
- 言語仕様を小さくし，コンパクトな構文と基本メカニズムに基づいたものとする．これにより実装を小さくでき，言語に関して学ぶべきことも少なくできる．これには，学習者にも言語の少数のメカニズムを組み合わせることで複雑なものが記述できることを学んで欲しいという意図がある．

以下では上記の方針に基づいたドリトルの設計について，構文，意味，実行環境に分けて説明する．

2.1　構文の設計

オブジェクト指向言語の構文は，基本的に「オブジェクトに対して，引数とメソッド名を指定して，メソッド呼び出しを行う」という形になる．たとえば Java であれば，このための構文は次のようになる．

```
オブジェクト.メソッド名(引数，引数，...);
```

しかし，中学生に対してこのような多数の記号を決まった規則に従って並べるように求めるのは困難である．このため，できる限り記号類を少なくして，次のようなメッセージ送信式を基本とするようにした．

```
オブジェクト！引数 引数 … メソッド名。
```

メソッド名は後に置き[†1]，引数やメソッド名どうしは

Dolittle: An Object-Oriented Language for Education.
Susumu Kanemune, 大阪電気通信大学, Osaka Electro-Communication University.
Yasushi Kuno, 筑波大学, University of Tsukuba.
コンピュータソフトウェア, Vol.28, No.1 (2011), pp.43–48.
2010 年 2 月 12 日受付.

†1　親しみやすいように日本語や韓国語の語順に合わせたという意味もある．

空白文字で区切る．引数列の範囲を示す括弧を省略できたため，必要な記号はオブジェクトと引数を区切る「！」と文末の「。」だけとなった．さらに，記号類についてはいわゆる半角と全角の対応する文字，および「。」と「.」を同一文字とみなすようにすることで，文字種別の使い分けを強いないようにした．

たとえば，変数「かめた」に入っているタートルオブジェクトに対して，線を引きながら 100 ピクセル前進させるメソッド呼出は，次のようになる．

```
かめた！100 歩く。
```

また，引数とメソッド名の区別は「トップレベルの識別子はメソッド名」という原則を採用した．そして，メソッド名の後にさらに続けて引数を書くことで，連続したメソッド呼び出し (カスケード) を記述する．

```
かめた！100 歩く 90 右回り 100 歩く。
```

引数部分に変数参照を書きたい場合は，() で囲む．個々の式は,「！」が含まれない場合は通常の中置記法の式，含まれる場合はメッセージ送信式となる．

```
かめた！(x + 10) 歩く (入力欄！読む) 右回り。
```

制御構造については，Smalltalk-80 にならい，ブロックを引数として渡すことで通常のメソッドに制御構造の働きをさせるようにした．たとえば指定回数の実行は次のようになる．

```
[かめた！100 歩く 90 右回り]！4 繰り返し。
```

ブロックが持つメソッド「繰り返し」は指定回数の反復を行う．分岐はもう少し複雑で次のようになる．

```
[x < 10]!なら [x = x + 1。y = y - 1] 実行。
```

ここで「なら」はブロックオブジェクトが持つメソッドの呼び出しであり，枝分かれを制御するオブジェクトを返す．これに対して then 部に相当するブロックを引数として「実行」を呼び出すことで，普通の言語の if 文と同様のことが記述できる．if-else や else if の連鎖，while なども同様に制御オブジェクトにブロックを渡すことをカスケードすることで実現されている．これにより，ドリトルが持つ構文は「メッセージ式」「中置記法」「ブロック」の 3 種だけで済み，図 1 のように簡潔なものとなった．

このような構文にしたことで，プログラムは「動作をある程度続けて横に長く書く」ことが自然なスタイルとなった．これにより，コードが 1 画面内に収まりやすくなり，一覧性が増すという効果があった．

```
プログラム ::= (文 ′。′)…
文 ::= [ 変数 ′ =′] 式
変数 ::= [ 項 ′ :′] 名前
式 ::= 単純式 | 送信
送信 ::= [ 項 ]′!′ 電文
電文 ::= 単純式… 名前 ( 単純式… 名前 )…
括弧 ::=′ (′ 中置式 ′)′|′(′ 送信 ′)′
単純式 ::= 数値定数 | 文字列 | 括弧 | ブロック
ブロック ::= ′[′ [′|′ 名前… ′|′] 文 (′。′ 文 )… ′]′
中置式 ::= 中置式 演算子 中置式 | 項
項 ::= 単純式 | 名前
```

図 1　ドリトルの構文 (一部)

2.2　意味の設計

オブジェクト指向という点から考えると，オブジェクトを定義し，インスタンス変数を持たせ，メソッドを定義することが前提となる．Java などのクラス方式の言語であれば，クラス定義の構文があり，その中で特定の書き方でインスタンス変数やメソッドを定義することになるが，中学生にこのような複雑な構造を文法に従って書くことを求めるのは難しい．

そこで，Self や JavaScript を参考に，プロトタイプ方式のオブジェクト指向を採用した．まず，すべてのオブジェクトは「作る」というメソッドを持ち，これを呼び出すことでそのオブジェクトが「親」(プロトタイプ) となるような子オブジェクトを作り出す．

```
カウンタ = オブジェクト！作る。
```

次に，オブジェクトは任意個数のフィールドを持つことができる．フィールドにアクセスするためには,「オブジェクト:フィールド名」と書き，このフィールドをオブジェクトのインスタンス変数として用いる．さらに，フィールドにブロックを入れることで，そのブロックがメソッドのコードとして働く (| | で囲んでパラメタを任意個数指定できる)．ブロック実行の最後で評価した値がメソッドの返値となる．

```
カウンタ:値 = 1。
カウンタ:足す = [|n| 値 = 値 + n。値]。
x = カウンタ！3 足す。
```

複数のカウンタを使う場合は,「c1 = カウンタ！作る」などによりカウンタをプロトタイプとするオブジェクトを複数作る．これにより，前節の構文の範囲

内で，オブジェクト群を定義できるようになった．

変数はグローバル変数，インスタンス変数，ローカル変数の 3 種類であるが，グローバル変数は「ルート」という特別なオブジェクトのプロパティであるので，前二者は基本的に同じものである．

ブロックは基本的にローカルスコープの変数を参照するクロージャであるが，オブジェクトのフィールドに代入されてメソッドとして起動された場合はフリーな変数はインスタンス変数を参照するものとして解釈される．プロトタイプ方式なので，参照時に手前のオブジェクトで見つからない名前はプロトタイプの連鎖をたどって探されるが，代入時には常に手前のオブジェクトに値が格納される．これらの正確な仕組みは説明するにはやや複雑であるが，中学・高校の授業の場面でこの詳細が問題になることは無かった．

2.3 実行環境の設計

図 2 にドリトルの実行画面を示す．上部のタブで編集画面と実行画面を切り替える形で作成した．プログラムを作成したり直す場面と，実行する場面をはっきり区別させたいと考えたためである．

また，Lisp のトップレベルや irb などの環境とは異なり，プログラムは常に「初期状態の環境で」「先頭から」実行される．このため「1 行入れてすぐその動作を見る」ことはできないが，常に同じ状態から実行されることで，状態による動作の違い (混乱の原因となる) が起きないようにしたものである．

言語の中核部分は，オブジェクト指向の実行機能のみを提供していて，実際の教育場面で使用するさまざまな題材は組み込みのオブジェクトとして用意した．主要なオブジェクト群として，次のものがある．

タートルグラフィクス — LOGO などと同様のタートルグラフィクスで図形が描画できるほか，描かれた図形をオブジェクトとして，複製したり動かすことができる．これにより，タートルグラフィクスで描いた図形を「部品」として，それを組み合わせて自分が思うような絵を制作するなどの活動が行える．

アニメーション — 一定間隔でブロックを実行する「タイマー」オブジェクトにより，絵を動かしたりゲームを実装できる．図形やタートルに「衝突」メソッドを定義することで，衝突に反応させることもできる．

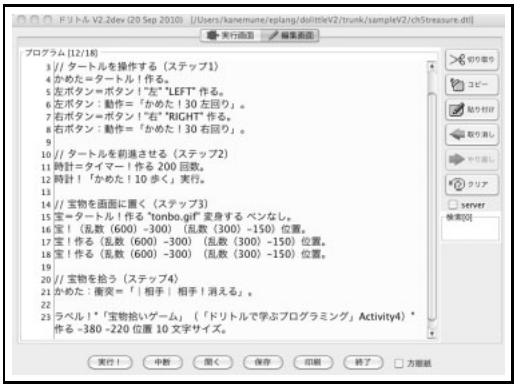


図 2 ドリトルの画面 (編集状態)

音楽 — General MIDI に対応する各種のオブジェクトが用意されており，音名やリズムを文字列で指定してオブジェクトを生成することで，任意の MIDI シーケンスを生成できる．これにより，メロディを作成したりゲームに音をつけるなどの活動ができる．

計測・制御 — 車輪やセンサーを持つ自律型ロボットカー等の教材に対し，バイトコードから成るプログラムを USB などを経由して送信することで計測・制御の学習が行える．

ネットワーク — ドリトル実行系にはシリアライズされたドリトルオブジェクトを文字列キーと対にして蓄積するサーバになる機能が組み込まれており，サーバに対して複数の実行系からオブジェクトを読み書きすることでネットワーク通信などが行える．

これらのオブジェクト群はコア部分は Java で記述され，ドリトル実行系内部に組み込まれているが，直接 Java で記述しなくても済む部分はプレリュードファイルにドリトルの構文で書かれていて，実行開始時にこれを読み取ることで初期化が完了する．

ドリトルには予約語が一切なく，出現する名前はすべて変数やメソッド名などの識別子であるため，プレリュードファイルでこれらを適宜置き換えることで，日本語以外の言語 (韓国語版，英語版，中国語版がある) にも容易に対応できた．

3 実装

ドリトルの実行環境は Java によって記述されてお

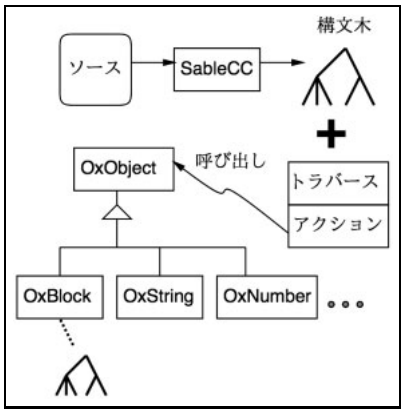


図 3　実行エンジン部分の構造

り，JRE が動く環境であれば動作する．

実行エンジン部分の構造を図 3 に示す．字句解析と構文解析には，Java により記述され Java のコードを生成するコンパイラコンパイラである SableCC[†2]を使用している．SableCC は，字句解析部が UTF-16 を前提に作られていて日本語の文字を使用する処理系の構築に好都合であることと，構文木が自動的に組み立てられ，Visitor パターンのアダプタクラス自動生成による柔軟なトラバースが行えることから採用した．字句定義と構文定義の行数は約 200 行である．

言語の実行エンジン部分はツリーインタプリタであり，構文木を上記のアダプタクラスを継承して作った実行用アダプタでトラバースしながら各動作を実行する．ドリトルの基本となる実行メカニズムはオブジェクトのフィールド参照/代入，およびオブジェクトに対するメソッド呼び出しだけである．

ドリトルのオブジェクトは Java 上ではクラス OxObject とそのサブクラス群のインスタンスに対応しており，このクラスがフィールドの参照/代入機能とメソッド呼び出しのためのメソッドを提供する．

OxObject はフィールドの値を格納するためのハッシュ表を持っており，サブクラスでもこれを継承して値の格納に用いる．プロトタイプ方式のオブジェクト指向を採用しているため，各オブジェクトは親オブジェクトへの参照の連鎖をたどり値を取り出す．

OxObject およびその各サブクラスはクラス変数としてメソッド名とメソッド実体 (Java のリフレクション機能が提供する Method オブジェクト) の対応表を持っており，この部分をコード上で記述することでオブジェクトの種類ごとにどのような Java メソッドをドリトル側から呼べるようにするかを指定できる．

実際にメソッド呼び出しが行われた場合は，まずオブジェクトに対してメソッド名と同名のフィールド値の取り出しを試み，ブロックが取れた場合は，そのブロックオブジェクトが内部に保持している SableCC 構文木のトラバースを行う．それ以外の場合は，対応表を検索して Java のメソッド呼び出しを行う．

実行環境全体は，グラフィクス機能のためのオブジェクト群に対応した実行画面とコード編集エディタを備えた GUI から「実行」ボタンに対応して SableCC の解析機能とツリーインタプリタを呼び出すようにできている．そのほか，音楽など前述の各種オブジェクトも搭載されている．コード量の内訳を次に示す．

実行エンジン部分	約 8,000 行
GUI オブジェクト	約 2,000 行
音楽オブジェクト	約 2,000 行
グラフィクスオブジェクト	約 1,500 行
計測・制御オブジェクト	約 500 行
ネットワークオブジェクト	約 500 行

4　サンプルプログラムと利用例

4.1　1 時間のプログラム体験

図 4 は，授業で使われる典型的なプログラム例である．説明のために行頭に番号を置いた．

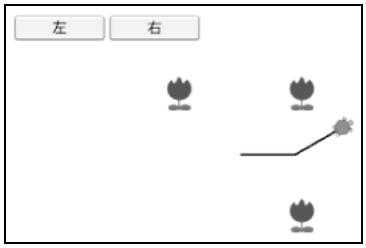
授業では，最初にゲームを作ることを伝える．資料は配らず，教員の画面を見せて，1 行ずつ入力させる．

1 行目では，タートルオブジェクトを作り，「かめた」という名前を付けている．実行すると画面にカメの姿をしたオブジェクトが現れる．

2 行目では，ボタンオブジェクトを作り，「左ボタン」という名前を付けている．実行すると画面に「左」と書かれたボタンが現れる．授業ではボタンを押しても反応がないことを確認させ，理由を考えさせる．

3 行目では，押されたときに実行される「動作」というメソッドを定義している．実行すると，ボタンを

†2　http://sablecc.org/




```
1 かめた＝タートル！作る。
2 左ボタン＝ボタン！"左" 作る。
3 左ボタン：動作＝「かめた！30　左回り」。
4 右ボタン＝ボタン！"右" 作る。
5 右ボタン：動作＝「かめた！30　右回り」。
6 時計＝タイマー！作る。
7 時計！「かめた！10　歩く」実行。
8 タートル!作る"tulip.png" 変身する ペンなし
  100 100 位置。
9 かめた：衝突＝「｜相手｜　相手！消える」。
```

図 4　プログラム例と実行画面 (宝拾いゲーム)

押したときにかめたが 30 度ずつ左に回転する．

4, 5 行目は，2, 3 行目を参考に記述させる．

6, 7 行目は，少しずつ移動を繰り返すことで，アニメーション表示になることを説明してから入力させる．タイマーオブジェクトは標準では 0.1 秒間隔で，パラメータで与えられたブロックを繰り返し実行する．実行後，ボタンで操作できることに気づかせる．

8 行目では，カメが拾う宝物を置いている．ここでは無名のタートルオブジェクトを生成し，姿を画像ファイルで変更した後，線を引かずに移動するように設定し，画面上の特定の座標に配置する．

9 行目では，他のオブジェクトと接触したときに実行されるメソッドを定義している．パラメータとして，衝突した相手のオブジェクトが渡される．

授業では，ここまでを入力させると，およそ 30 分程度になる．時間に余裕がある場合は，8 行目を複製して，複数の宝物を画面に配置させる．このとき，意欲のある生徒には，乱数を説明して，実行するたびに異なる位置に現れるようにさせる．たとえば，X 座標として −300 から 300 までの範囲で乱数を生成する場合は「(乱数 (600)-300)」と記述することを伝える．

4.2　高校での利用状況

教科「情報」は，普通高校において必履修科目であるが，教員に経験がないこともあり，プログラミングはほとんど扱われていない．そこで，前節の内容を「1 時間で学ぶソフトウェアの仕組み」としてまとめ，ソフトウェアの理解を目的とした 1 時間のプログラム体験授業を実施できるようにした[†3]．

いくつかの高校教員の Blog では，その効果が驚きの声とともに，報告されている[†4]．これらの報告と，実施した教員からのヒアリングによると，高校の授業での利用には，次の利点があることがわかった．

- 1 時間 (50 分授業) で実施できる．
- 1 行ずつ入力して実行させることにより，エラーが起きた場合には最後の行を教員の例と比較すればよく，デバッグの問題が発生しない．
- プログラミング経験のない教員でも実施可能．
- 生徒は積極的に取り組み，プログラミングに対する学習意欲が維持される．
- 生徒はソフトウェアがプログラムによって作成されているというモデルを理解できる．

行ったアンケートから，この授業を実施することで，生徒は次のことを学習していることがわかった．特に，プログラムを自分たちが作成できることの理解や，OS の理解につながるシステムソフトウェアの理解については，学習の意義が大きいと感じている．

- ゲームなどソフトはプログラムで作られている．
- プログラムは人間が特別な「言語」で書く．
- 文法が違うとエラーになる．間違って書くと間違って動く．書かれていないことは実行されない．
- 上から順に実行されるが，ある状態になったときに実行される命令もある．
- ソフトウェアは自分たちでも作れる．
- OS のキー入力やマウスカーソルもプログラムが表示している．

†3　http://kanemune.eplang.jp/diary/2008-11-06-1.html

†4　http://blog.goo.ne.jp/yoshi-sato_2004/e/dcdf6da2b6da91051fe98241ce8f5590
http://jyohoka.exblog.jp/10222741
http://htanaka.exblog.jp/10256122
http://www.ariori.com/diary/2009/01/09/

表 1　プログラミング入門授業の構成例

	内容
1	授業解説，1 時間のプログラム体験
2	反復，タートルグラフィックスの基礎
3	図形オブジェクト，メソッド定義
4	ボタンオブジェクトによるペイントソフト作成
5	タイマーオブジェクトによるアニメーション
6	アニメーション作品製作 (1)
7	アニメーション作品製作 (2)
8	衝突判定とスクロールゲーム
9	乱数の利用
10	壁での跳ね返り
11	条件判定とブロック崩し
12	得点計算，作品構想
13	自由作品制作 (1)
14	自由作品制作 (2)
15	自由作品制作 (3)，作品発表

4.3　大学での利用状況

大学では，情報系以外の学科における，プログラミング入門の授業で利用されることが多い．筆者が行っている複数大学の授業において，半期を通して，プログラミングに対する高い学習意欲が維持された．

表 1 に，半期のカリキュラム例を示す．プログラミングの初心者を対象としているが，受講生の全員がオリジナルの作品プログラムを作成することができた．作品には，季節に応じたアニメーション (雪が降る中で雪だるまが大きくなるなど)，乱数を利用した占い，迷路，シューティングゲームなどがあった．

最終回に実施した，半期を通したアンケートを見ると，4 段階の選択結果は次のようになった．適度な難易度を保ちつつ，高い学習意欲を維持できたことがわかる．自由記述では,「楽しかった」という感想が多く，次の学期に行われる C 言語の授業や制御プログラミングの授業への期待も多く見られた．

- 「よく/だいたい理解できた」が計 86.5%
- 「とても/まあまあ楽しかった」が計 100%
- 「とても/少し難しかった」が計 83%

この授業は，プログラミングに対するポジティブな意識を持たせることを目的として行った．繰り返しと条件分岐という，プログラミングの基本概念を学習しつつ，C 言語等の汎用的な開発言語の学習への期待と意欲を持たせられたことは，初年度の入門教育としては意義があったと考えている．

5　配布とサポート

ドリトルは，Web サイト†5で配布を行っている．個人や教育利用における利用は制限がない．動作は，Windows, Macintosh, Linux, FreeBSD で確認を行っている．インストールは任意のディレクトリにファイルを展開する形式であり，管理者権限は不要である．

Web サイトでオンライン版にアクセスした場合は，Java Applet として利用できる．ただし，ローカルファイルの保存や外部機器の制御など，一部の機能に制限が存在する (プログラムの保存は画面上でコピーペーストにより行う)．

6　まとめ

ドリトルという言語環境を紹介し，教え方や言語環境を工夫することで，プログラミングの楽しさを初心者に伝えることが可能であることを報告した．

現在は組込デバイスへの対応 [4] などを進めている．今後は，高校で広く利用してもらうための情報提供を進めつつ，Squeak Etoys，Scratch，Viscuit など，より低年齢層を意識した画面上の図形的な操作によりプログラミングを行う言語 [3] との連携，情報科学教育手法 [5] との連携，大学で C や Java などの汎用言語につなげていくための入門授業のカリキュラムなどを整備していきたいと考えている．

参 考 文 献

[1] 兼宗進, 久野靖 : ドリトルで学ぶプログラミング, イーテキスト研究所, 2008.
[2] 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖 : 学校教育用オブジェクト指向言語「ドリトル」の設計と実装, 情報処理学会論文誌, Vol. 42, No. SIG11 (2001), pp. 78–90.
[3] 兼宗進, 阿部和広, 原田康徳 : プログラミングが好きになる言語環境, 情報処理, Vol. 50, No. 10 (2009).
[4] 並木美太郎, 久野靖, 兼宗進, 早川栄一 : ユビキタス・組込みシステムにおけるオブジェクト指向言語 Dolittle の実装, 情報処理学会, 第 50 回プログラミングシンポジウム, 2009.
[5] 兼宗進監訳 : コンピュータを使わない情報教育, イーテキスト研究所, 2007.

†5　http://dolittle.eplang.jp