経営論集　第31巻第1号　2021年　59–88頁

ScratchとJavaを並列して進めるプログラミング教育の実践

A Practice of Programming Education that Proceeds While Parallelizing Scratch and JAVA

小　松　香　爾

〈論文要旨〉

小規模で学習目的のプログラムを書くための言語，大規模で実用的なプログラムを書くための言語。異なる方向性を持つ2つのプログラミング言語を同時並行して教育するという実践を行い，授業アンケートを採った。Scratchは，Javaと比較して，学生にとって馴染みやすく理解しやすいプログラミング言語であることを確認できた。

〈Abstract〉

A language for writing small-scale programs for learning purposes, and a language for writing large-scale practical programs. Scratch is found to be more familiar and understandable to the students compared to Java.

1. 経営学部におけるプログラミング教育

選択科目が5つ存在する。2年次配当科目が「プログラミング（基礎）」，「プログラミング（応用）」および「Webプログラミング（応用）」，3年次配当科目が「プログラミング言語」である。

各科目で採用されているプログラミング言語は，Visual Basic，JavaScript，Javaである。現在，新カリキュラム策定中であり，Python，Go，PHP，Ruby等を加える，あるいは入れ替える予定である。

著者は「プログラミング言語」を担当している。他大学での担当を含めると，著者が教育で使用経験がある言語は，C，Prolog，Lisp，JavaScript，Java，ActionScript，Scratchである。他にゼミナールで，Unity用にC#も使用する予定である。

1.1「プログラミング言語」におけるJavaの採用

「プログラミング言語」では，10年以上前の開講時から2019年度まで，Javaのみを採用してき

2021年10月25日受理

た。著者は科目開設時に言語の決定に関与していなかった。しかし，当時から，Javaの採用は妥当と判断していた。教科書として「明解Java入門編」(柴田2007)を指定し，第8章「クラスの基本」までに相当する内容を教えることにした。しかし，授業中は，教科書をほとんど参照せず，授業中に似たようなプログラムをその場で書いて，その場でデバッグしていた（小松2016）。

2021年度は，Javaの教育内容を拡大した。Javaの最大の特徴は汎用性の高さにある。汎用性の高さ以外にも，可読性の高さ，プログラムの組みやすさも無視できない特徴である。可読性の高さは，標準クラスライブラリのクラス名やメソッド名の適切さから，プログラムの組みやすさは，クラスやインターフェースの妥当性の高さから生じる。

標準クラスライブラリの量的・質的な充実は，汎用性の高さにつながる。可読性と汎用性の高さに起因するプログラミング言語としてのシェアの高さから，ライブラリやフレームワークが充実して，汎用性が高くなるという正のフィードバックも生じる。Javaがゲーム以外のシステム開発現場で採用されやすい理由として，例外処理のためのクラスが豊富で，セキュアな業務プログラムを書きやすいという特徴もある。

Javaに対して，Scratchでは，実行時の例外をプログラマが注意する必要がある。0除算では，図1のように「無限大」という値が返ってくる。

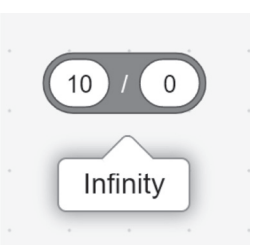

図1 Scratchにおける0除算の戻り値

1.2「プログラミング言語」におけるScratchの採用

2020年度の新型コロナウイルス流行をきっかけに，教育内容を見直し，「新明解Java入門編」第6章「配列」までに相当する内容は，Scratchで簡潔かつビジュアルに記述できると判断した。第7章「メソッド」に関しては，Javaで教えざるを得なかった。Scratchには，ブロック定義の機能しかないからである。

Scratchのブロックの定義は，Pascalにおけるプロシージャの定義やBasicにおけるサブルーチンの定義に相当する。図2のように，値渡しの引数は設定できるが，戻り値が設定できない。戻り値がないのは，古典的なBasicでも同様であり，プログラミング言語学習の初心者にとっては理解しやすさにつながる可能性はある。しかし，他言語を習得済みの教員・学生は，戸惑う可能性が大きい。戻り値が設定できなければ，データのみを呼び出し元に返すことによってモジュール結合度を弱くすることができない。

図2 Scratchにおけるブロック定義

Pascalは1990年ごろに教育用にも流行った言語であり，Scratchで戻り値を設定できないのは，Pascalの影響を受けていると推測できる。しかし，戻り値を呼び出し元に返せないのは，手続き型言語の主流ではない。主流でないことは，教育用言語という観点では大きな問題にはならないが，モジュール間のデータの受渡しにグローバル変数を使わざるを得なくなる。また，ブロック定義の内部に閉じたスコープを持つローカル変数も作れない。したがって，定義されたブロックを再入可能にできない。

1.3「プログラミング言語」にJavaを残す理由

戻り値が存在しないことは，Scratchが大規模なプログラミングに向かない一因である。卒業後，業務でプログラミングする可能性を考慮すると，モジュール間の結合度を下げる方法を理解させておく必要がある。オブジェクト指向言語では，モジュール間の結合度を，主にカプセル化とインターフェースによって下げる。Scratchはオブジェクト指向言語ではないが，図3のようにスプライト内にスコープを限定した変数を宣言することはできる。

図3 Scratchにおけるスプライト内部に限定されたスコープを持つ変数宣言

変数へのアクセス制御は，変数をスプライト内でローカル宣言する方法のみであり，Javaのようなカプセル化は実現できない。当該スプライトに限定されたローカル変数は，そのスプライトのスクリプトエリアからしかアクセスできず，すべてのスプライト用に宣言されたグローバル変数は，ステージとすべてのスプライトからアクセスできてしまう。スプライト間のデータのやりとりは，グローバル変数を使って，メッセージ送受信で同期をとって行うしかない。

Scratchには，クラスやメソッドに相当する概念がない。Scratchでは，1人でプログラムを組むことを前提としているため，抽象化の機能がない。クラスどころか構造体すら存在しない。Scratchには配列もない。リストはあるが，リストのアドレス値を得ることができない。アドレス値を得られないのはリストに限らない。参照型変数が存在せず，サブルーチンへの引数は値渡しのみである。多次元配列は，リストを用いて，多次元配列のデータ構造をプログラマが自作するしかない。

Scratchに参照型が存在しない理由の1つとして，Scratchのプログラミングパラダイムがイベント駆動型であり，複数のスレッドが並行動作することが挙げられる。参照が存在しなければ，複数のイベントから起動されるスレッド間の同期が取れなくても問題は生じにくくなり，マルチスレッドプログラミングにおけるスレッドセーフな実装がしやすくなる。

Scratchには，他の手続き型言語と比較して，少なくとも以下の7つの特異な点がある。「プログラミング言語」では，就職後に業務でプログラミングする可能性を考慮して，Javaも残すべきと判断した。

1. プログラムのモジュール化ができないに等しい
2. ユーザー定義型の変数が存在しない
3. ブロック定義したサブルーチンに戻り値を設定できない
4. 参照型がなく，ブロック定義したサブルーチンの引数を参照渡しにできない
5. 変数のスコープがグローバルとスプライト内の2種類しかない
6. 配列がない。リストはあるが多次元化はプログラマが作成する必要がある
7. ブロックを配置することによりプログラムを作る

「弱点」ではなく「特異な点」と記述したのは，上記7点は，教育用言語としての可読性とトレードオフになっているからである。ただし，可読性といっても，小規模な構造化されたプログラムを読みやすいというだけであり，大規模なプログラムでは可読性が落ちる要因となる。例えば，多数のスプライトにプログラムを分割すれば，スプライト内部の可読性は上がるが，グローバル変数とメッセージブロックを多用せざるを得なくなり，プログラム全体としての可読性は落ちる。

Scratchのプログラムは，スプライトで定義されたキャラクターを動かすために書かれることが多く，スプライト単位での可読性があれば大きな問題はないともいえる。しかし，1つのスプ

ライトに多数のブロックを配置すると，可読性が極端に低下する。

Scratchでは，イベント駆動のために存在するハットブロックと，サブルーチンを作るために存在するブロック定義ブロックに続くブロックの列で，一塊のスクリプトが形成される。形成されたスクリプトの配置は，スプライトのスクリプトエリア内で自由である。このスクリプト配置の自由度は，うまく利用すれば，可読性の高さにもつなげられる。しかし，大規模になると，どのようにスクリプトを配置しても，可読性は著しく落ちる。

図４は著者が3年生のゼミナール中に書いた格闘ゲームのメインキャラクターのスクリプトエリアに配置されたブロック群である。大量のスクリプトを一覧するためには，スクリプトエリアの拡大倍率をマイナスにする必要がある。しかし，拡大倍率をマイナスにすると，ブロック内部に書かれたテキストが読みにくくなる。図4はユーザーが操作するキャラクターのスクリプトのみである。敵キャラクターは簡単なAIで動くが，スクリプトは敵キャラクターのスクリプトエリアに配置してある。

図4 格闘ゲームのキャラクターのスクリプト群の一部

ブロックは色付けされているため，どこにどういう種類のブロックを配置したのかということは，判別しやすい。しかし，特に論理エラーを修正する場合，変数やサブルーチンの位置をピンポイントで特定する必要がある。

Scratchでは，スクリプトエリアに配置されたブロック中のテキストを検索しにくい。ブラウザの機能で検索できるが，テキストにハイライトが付くだけであり，スクリプトエリアがスク

ロールしたり，当該テキストの周辺のブロックが自動的に拡大表示されたりする機能がない。統合環境内にデバッガもなく，プログラムの論理エラー修正は，Scratch VM が統合開発環境内にグラフィックを描く様子を見て，プログラマが当該箇所と推測する部分のブロックの種類や配置を変える以外に方法はない。特に定義ブロックや変数ブロックで名付けた文字列を検索する手段がないのは，デバッグに支障をきたす。

1.4「プログラミング言語」における言語選択

プログラミング言語は多数存在する。UnixOS を書くために，アセンブリ言語を抽象化したのがC言語である。Cは，静的型付けする手続き型言語であり，誕生から50年近く経つ。実用性と可読性を重視したGoogleのGo言語の設計にはCの考案者が関わっており，CはJavaを含めて，多数の言語に影響を与えてきた。

Prolog に代表される論理型言語，Lisp に代表される関数型言語は可読性において多くの手続き型言語に劣る。手続き型言語の中でも，Excel VBA など，可読性の悪いプログラムしか書けない言語は存在する。しかし，VBA の可読性の低さは，Visual Basic ではなく，アプリケーションに由来する。可読性が低いと，プログラムの共同開発や保守に支障が生じる。特に保守性は，BCP の観点から欠かすことができない。ビジネスに継続がないと経営は成り立たない。目的の機能を提供することがプログラムであるが，時間の経過によって，プログラムの動く環境が変化し，機能を修正する必要が生じる。

プログラムの実行速度は速いに越したことはない。実行速度の遅さにも関わらず，業務でもスクリプト言語が多用されるのは，可読性に一因がある。可読性は主観によるものではあるが，多くの主観が集まれば，根拠の弱い定説となる。スクリプト言語には，さまざまな言語があるが，一般的に可読性が高いとされている。

スクリプト言語の中には，Python のようにインデントに意味を持たせるなど，文法上の工夫で，可読性を高めている言語もある。しかし，著者の見解では，スクリプト言語の可読性は，主に手続き型であることから生じる。プログラムは何らかの機能を実現するものである。人間が機能を考えるときには，宣言的でも関数的にでもなく，手続き的に考えるはずである。人間は，生まれたときから時間の経過を意識し，時間の流れの中で生活する。機能を実装するときにも，人間は時間経過を意識せざるを得ないのではないか。関数型のLispや論理型のPrologおよびそれらの亜種は，プログラムの実装時に，時系列に沿ったプログラムの動作をイメージしにくい。他人が書いたプログラムの場合，その難易度は飛躍的に上がる。

Prolog は，述語論理の変数をプログラムの変数となり，型付けシステムは動的である。Prolog のプログラムは，手続きの集合ではない。述語論理式の一種であるホーン節の形式で，宣言の集合として記述される。Prolog は，第5世代コンピュータプロジェクトで人工知能開発用言語として採用された。プロジェクトの成果としてProlog専用マシンも開発された。Prolog マシンでは，ユニフィケーションやバックトラックなどのProlog の機能が機械語レベルで実現

されていた。これらのProlog独自の機能は，定理証明や字句・構文解析ではプログラムを簡潔に書くために有用である。しかし，プログラムの実行スピードを遅くする原因にもなる。1985年にISOにより標準規格が定められたが，現在，亜種も含めて，業務での使用例はほとんどない。述語論理は述語を中心として論理式を組み立てる。したがって，モノや概念を中心とするオブジェクト指向設計をサポートしにくいという欠点もある。

Lispは，業務でよく使われる他のプログラミング言語と同様に，ANSIによって，1994年にCommon Listとして規格化された。ただし，Common Lispは実用性が重んじられている。厳密な関数型言語ではなく，手続き型の命令も言語仕様中に存在するマルチパラダイム言語である。ANSI Common LispやCLOSとして，標準化されている。オブジェクト指向プログラミングが可能であるが，動的オブジェクト指向言語であり，型チェック，型変換，名前束縛がコンパイル時ではなく，実行時に行われる。型チェック，型変換，名前束縛が静的なJavaやC++とは異なる。

手続き型言語であることは，著者が経営学部のプログラミング言語科目で採用するための，必要条件にすぎない。CおよびC++は，Javaと同じ手続き型ではあるが，抽象度がJavaより低い。Cは，1960年頃，UnixOSを書くために設計された言語である。アセンブリ言語に近く，現代の水準からすると標準ライブラリはないに等しい。ただし，メモリ消費量が少なく，VMを使わないため，実行プログラムは高速に動作する。現代でも，ハードウェアの組み込み用のプログラミング言語として多用されている。C++はオブジェクト指向設計に対応できるようにCを拡張した言語であり，1980年代中盤から存在する。WindowsOSは，1995年にリリースされたWindows95から2021年度中にリリースされる予定のWindows11まで，すべてC++で書かれている。

CやC++は，成り立ちからしてシステムプログラミング用言語である。ハードウェアのしくみを同時に学習させる目的ならば最適解になりうる。しかし，経営学部の学生にシステムプログラミングを学ばせるのは，学生のモチベーションを上げにくい。費用対効果，得られる効果に対する学習コストの面からも，経営学部での採用は現実的ではない。

1.5 Scratchによるプログラミング

Scratchは手続き型言語であるが，パラダイムはイベント駆動型といえる。スレッドはイベントブロックのみから開始される。

Scratchの統合開発環境では，各種の基本ブロックが機能ごとに分類されて，ブロックパレットにグラフィカルに配置されている。Scratchでは，基本ブロックをドラッグ＆ドロップすることにより，スクリプトが書かれる。Scratchの型付けは基本的には動的であるが，基本ブロックの形状も型である。

ブロックの形状は，ブロックパレットに表示されていて，変形することはなく，ハットブロック，スタックブロック，真偽ブロック，値ブロック，C型ブロック，キャップブロックの6

種類である。ハットブロックはスクリプトの先頭以外には配置不可，C型ブロックの中にはスタックブロックしか入れられないなどの制限がある。したがって，ブロック形状の型付けは静的といえる。スクリプトを実行するときではなく，スクリプトを組む前に組み方が制限されているからである。

Scratchでは，ブロック形状の違いそのものが型チェックシステムになっている。プログラマは，ブロックをスクリプトの構文的に間違った位置に配置できない。Scratchで構文エラーが生じないのは，基本ブロックの型で構文チェックができるからである。ただし，Javaと同様に，論理エラーは防ぎようがなく，Javaとは異なり，例外処理は存在しない。

Scratchは教育用言語であり，業務用，特に大規模なプログラムを組むことは想定されていない。オブジェクト指向設計はできず，オブジェクト指向プログラミングには対応していない。継承やポリモーフィズムは実現不可能であり，クラスの概念もない。

ただし，ブロック定義されたブロックを呼び出せる範囲やローカル変数のスコープはスプライト単位である。スプライトをオブジェクトとみなせば，JavaScriptのようなプロトタイプベースのオブジェクト指向言語に似た概念を持つともいえる。Scratchには，図5のように，スプライトをコピーしたり，コピーしたスプライトを消去したりするクローン関係のブロック群が存在する。「クローンされたとき」というイベントブロックに続く処理は，スプライトがコピーされたタイミングで実行される。

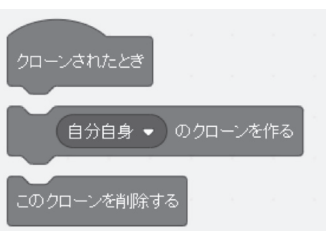

図5 クローン関係のブロック群

Scratchでは，オブジェクト指向設計に基づくプログラミングは不可能である。しかし，スプライトベースでの弱いカプセル化は実装可能である。図6において，スプライト2で宣言されたローカル変数に，他のスプライトからはアクセスできない。アクセスできるのはスプライト2のスクリプトエリアに置かれたブロックからのみである。

図6 スプライト2で宣言されたローカル変数へのアクセス

図7において，スプライト2で宣言されたサブルーチンに，他のスプライトからはアクセスできない。アクセスできるのはスプライト2のスクリプトエリアに置かれたブロックからのみである。

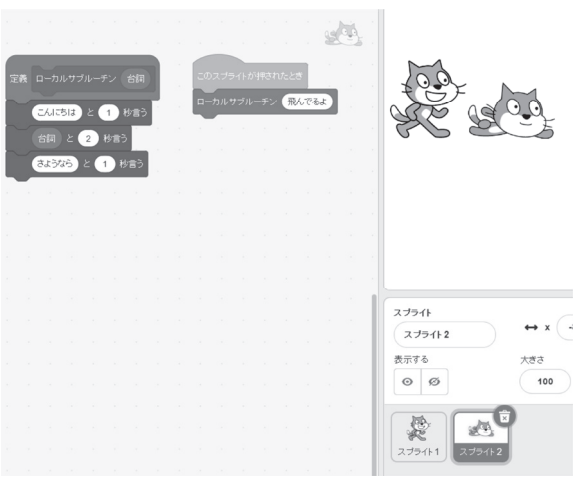

図7 スプライト2で定義されたサブルーチンへのアクセス

Scratchには，スプライトから，別のスプライトのブロックエリアに定義されているサブルーチンを呼び出すしくみはない。Javaには，メソッドにつけるアクセス修飾子としてpublic, protected，パッケージprivate, privateの4種類がある。スプライトのブロックエリアを1つのクラスとみなすと，Scratchのサブルーチンは，privateである。Scratchにはパッケージに相当するプログラムの分類機能もない。

Javaでは，あるクラスから作られたインスタンスが，他のクラスから作られたインスタンスと連携する場合，他のクラスでprivate修飾子が付けられたメソッド以外のメソッドを呼び出す。Scratchのスプライト間で同様なことを実現する場合，図8に挙げられるメッセージの送受信に関するブロックを使う。

図8 メッセージ関係のブロック

Scratchのスプライトをオブジェクトとみなした場合，オブジェクト指向言語の代表的な機能であるメッセージパッシングの機能が実現できる。ただし，Scratchのメッセージは，ステージとすべてのスプライトに必ず届く。Javaでのメッセージパッシングは，他のクラスで定義されたメソッドを呼び出すことに相当する。メソッドには，アクセス修飾子をつけることができ，publicアクセスは，どこからでも呼び出せるが，privateアクセスは同一クラス内，デフォルトはパッケージ内，protectedはパッケージ内と継承されたクラス内から呼び出せる。

2　オブジェクト指向プログラミング

オブジェクト指向は抽象的な概念である。オブジェクト指向設計は，オブジェクト指向の概念を使ってソフトウェアを設計することであり，カプセル化，継承，ポリモーフィズムの3つの手法が意識されるべきとされている。Javaの場合，カプセル化は主にクラス定義とアクセス修飾子で，継承は，主にクラス，抽象クラス，インターフェースの宣言で実現される。ポリモーフィズムは，主に参照型変数と型変換およびメソッドのオーバーライドで実現される。

2.1 オブジェクト指向設計の除外

オブジェクト指向設計は，ロジックの変更に対して安全であり，かつ拡張性に優れたプログラムを書くための設計である。業務ではUMLが使われることが多い。UMLはオブジェクト指向設計用に，ISOによって標準化されたモデリング言語である。

UMLはプログラミング言語ではないので「プログラミング言語」では取り扱わない。他に理由として，なにか一つは言語を学んで，かつチームでプログラミングをした経験がないと，オブジェクト指向設計のメリットを理解しにくいことが挙げられる。教育機関のシラバスにUMLが登場することは稀であり，Javaの書籍に出てくるとしてもクラス図のみである。Java

は，クラスありきの言語であるので，クラス図を設計図とみなせる。

情報系の専門学校でJavaを中心に2年間教育するという前提でも，オブジェクト指向設計でプログラミングできるレベルまで到達させるのは困難である。Javaに限らず，JavaScriptやPythonなどのメジャーなオブジェクト指向言語の機能を，網羅的に教えるのは現実的ではない。経営学部には経営学，経済学，マーケティング，会計学，簿記などの主要科目があり，プログラミング教育に割ける時間は極めて限定される。「プログラミング言語」は，通常の科目と同様90分×15コマであり，合計22.5時間である。

「プログラミング言語」では，オブジェクト指向設計とJavaの新しいバージョンに存在する関数型パラダイムを取り扱わない方針を，科目新設当初から定めていた。ただし，「モノや概念を抽象化してクラスを定義，定義したクラスをインスタンス化することによりオブジェクトとして具体化」というデータ抽象の考え方は取り扱うことにしていた。

2.2 クラスによる抽象化

データ抽象を除外しなかったのは，「データと操作をクラスとしてまとめることにより，構造化されたデータ型をプログラマ自身が定義」できることに意義があるという判断からである。

Cの構造体でも，同様なデータ型の定義は，熟練者であれば実現できる。しかし，入門レベルの学生には困難であり，可読性もJavaのプログラムの方が高いはずである。

Javaは，Simulaの流れを汲むクラスベースかつSmalltalkの流れを汲むメッセージング機構を持つオブジェクト指向言語である。Javaで大規模なプログラムを組む際には，オブジェクト指向設計，特にクラス設計をしてからコーディングに移るのが定石である。

オブジェクト指向設計のSOLID原則に従えば，メンテナンス性と機能拡張性に富むプログラムを書くことができる。しかし，経営学部に限らず，90分授業という形態の中では，大規模なプログラムは題材として扱いにくい。プログラムの規模が小さければ，オブジェクト指向設計のメリットは得られにくい。しかし，データ抽象，Javaではクラスとインスタンスという概念の利便性ならば，小規模なプログラムで理解させることができると判断した。

シングルトンというインスタンスを1つしか生成しないデザインパターンも良く知られているが，Javaでは，通常，クラスから複数のインスタンスがオブジェクトとして生成される。インスタンスのインスタンス変数に対して，アクセス制限されたメソッドが参照や代入を行う。アクセス制限により，Javaのプログラムは論理エラーを出しにくい。

Scratchにはクラスに相当する概念はなく，クラスベースのオブジェクト指向言語で多用されるプログラミングスタイルを使うこともできない。ただし，スプライトのクローンを作るブロックが存在する。コピー元スプライトのローカル変数は，スプライトのクローン作成時に，Scratchの仮想マシンにより，ヒープ領域に領域確保される。コピー先スプライトのローカル変数は，図9のように，クローン作成時のコピー元スプライトのローカル変数の値で初期化される。クローンのローカル変数は，コピー元スプライトのスクリプトエリアからしかアクセスで

きない。コピー元スプライトのローカル変数は，Javaのクラスでprivateアクセス修飾子をつけられたインスタンス変数と同等といえる。

図9 クローンされたスプライトのローカル変数

現在のJavaScriptはクラスベースのオブジェクト指向プログラミングも可能になった。しかし，当初はクラスが存在せず，純粋なプロトタイプベースのオブジェクト指向言語であった。プロトタイプベースのオブジェクト指向言語では，クラスからインスタンスを作るのではなく，オブジェクトのクローンを作ることでインスタンスを作る。オブジェクトのメソッドは，クローンのインスタンスでも使用できる。

図10では，コピー元のスプライトのコードエリアに，「回転拡大」サブルーチンが定義されている。このサブルーチンはクローンとして作られたコピー先のスプライトでも呼び出せる。

Scratchのスプライトをクローンすることは，プロトタイプベースのオブジェクト指向言語で，インスタンスを作ることに相当するといえる。

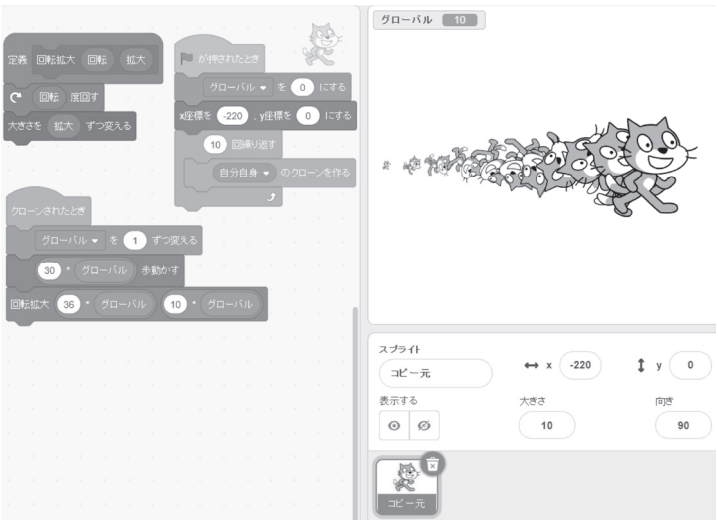

図 10 コピー元のスプライトで定義されたサブルーチンの呼び出し

2.3 クラスライブラリ

Java はユーザー数が多い言語でもある。1995 年に誕生してから，これまで様々な拡張が施されてきた。標準クラスライブラリにも，新しいクラスが追加され，クラス数は増加する一方である。

「プログラミング言語」では，標準クラスライブラリの Java.lang や Java.util パッケージ中のごく一部のクラスを扱うにすぎない。Servlet や JSP，EJB などの Web アプリ向け，業務アプリ向けのパッケージは取り扱わない。また，今後も，授業で説明する可能性はあるが，業務向けのパッケージやフレームワークを使ったプログラムを組む予定はない。

図 11 は，Swing で楕円を描くプログラムである。Swing は，Java で GUI アプリケーションを作成するための業界標準だったフレームワークである。Java の標準クラスライブラリ AWT のクラスを使用することが前提となっている。

```
J CircleFrame.java
 1 import java.awt.BasicStroke;
 2 import java.awt.Color;
 3 import java.awt.Graphics;
 4 import java.awt.Graphics2D;
 5 import java.awt.Rectangle;
 6
 7 import javax.swing.JFrame;
 8 import javax.swing.JPanel;
 9 import javax.swing.SwingUtilities;
10
11 public class CircleFrame extends JPanel {
12         public CircleFrame() {setBackground(new Color(80, 125, 180));}
13             public void paintComponent(final Graphics argG) {
14                 super.paintComponent(argG); //JPanelのメソッドをコール
15                 argG.setColor(new Color(150, 150, 0));
16                 Graphics2D argG2 = (Graphics2D)argG;// ダウンキャスト
17                 BasicStroke bs = new BasicStroke(10);// 線幅を変更
18                 argG2.setStroke(bs);
19                 // コンポーネントの境界オブジェクトをrectで参照
20                 final Rectangle rect = getBounds();
21                 // ウィンドウの大きさに楕円（最初は正円）を描画
22                 argG2.drawOval(0, 0, rect.width, rect.height);
23         }
24         private static void showGUI(int rx, int ry) {
25             // JFrameオブジェクトをframeで参照
26             final JFrame frame = new JFrame("ウィンドウに楕円を描く");
27             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28             // コンテナのオブジェクトにフレームを加える
29             frame.getContentPane().add(new CircleFrame());
30             frame.setLocationRelativeTo(null);
31             // 初期ウィンドウサイズを横rx, 縦ryに設定
32             frame.setSize(rx, ry);
33             frame.setVisible(true);
34         }
35         public static void main(final String[] args) {
36             /* Runnableインタフェースを実装した匿名クラスのオブジェクト
37              * を引数にして、キューにオブジェクトを登録
38              */
39             SwingUtilities.invokeLater(new Runnable() {
40                 // 引数なしのメソッドrun()を定義
41                 public void run() {
42                     showGUI(100, 100);
43                 }
44             });
45         }
46 }
47
```


図11 Swingフレームワークを使った楕円を描くプログラム

Scratchは「楕円を描く」ブロックが存在しない。拡張ブロックであるペンブロックを使って，図12のように三角関数で楕円を描く。

図12 Scratchを使った楕円を描くアニメーションのプログラム

Scratchの標準ライブラリには，図13のように，11種類の拡張機能しか存在しない。そのうち，Scratch3.0にMITが用意した拡張機能は，音楽，ペン，ビデオモーションセンターの3機能のみである。音声合成はAmazon，翻訳はGoogleが拡張機能を提供しており，いずれもインターネット接続が，ブロック実行に際しての必要条件になる。拡張機能のブロック数は少なく，ブロックあたりの機能が多い割に，プログラマが指定するパラメータは少ない。その反面，プログラムの挙動をプログラマが細かく制御できない。

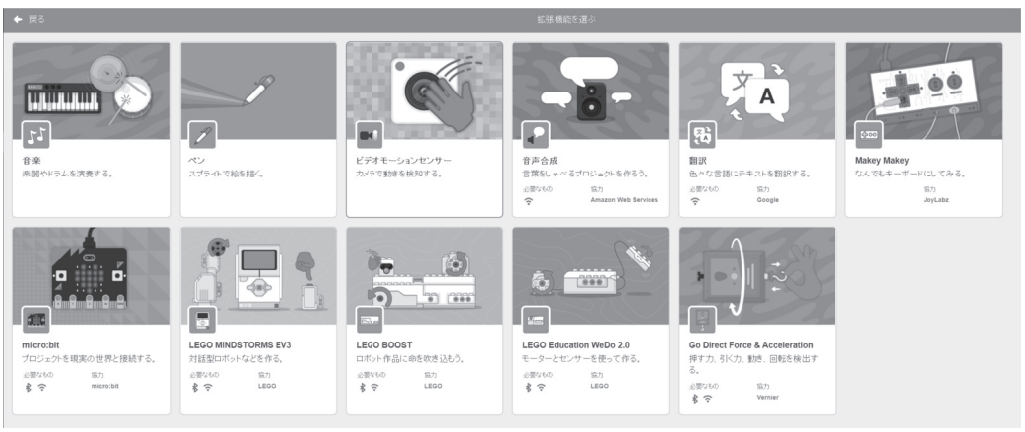

図13 Scratchのすべての拡張機能

ただし，MITはgithubにScratchのソースコードを公開している。したがって，一般ユーザーもScratchを拡張できる。図14は，mod版Scratch「Stretch3」の拡張機能の一部である。

図14 https://stretch3.github.io/ の拡張機能の一部

以下の図15 はビデオモーションセンサーのブロック群を使って作ったMRのプログラムである。先頭にビデオカメラの画像が埋め込まれているブロックがビデオセンサーの拡張ブロックである。

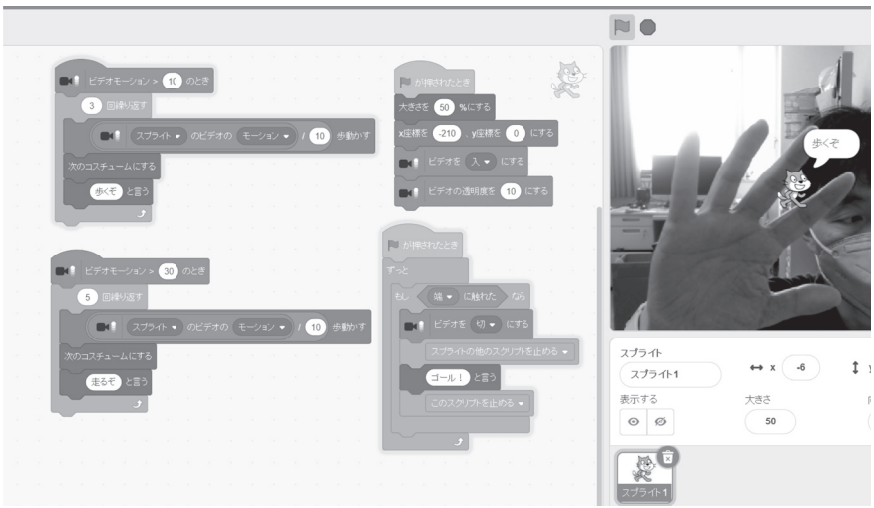

図15 拡張ブロックを使ったMRプログラミング

Scratchの統合開発環境には，Webアプリケーション版とデスクトップアプリ版が存在する。図16のScratchのデスクトップアプリ版を使用した場合には，インターネット接続を必要とする音声合成，翻訳等の拡張ブロックが動作しなくなることに注意が必要である。音声合成は，テキストを音声に自動変換する機能である。著者は授業中でも取り上げた。したがって，授業ではWebアプリケーション版の使用を推奨した。

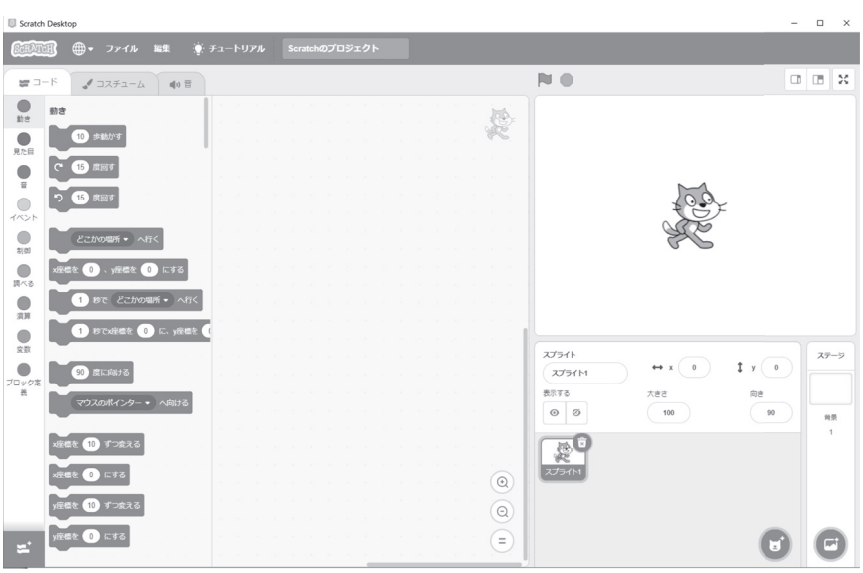

図16 Scratch Desktopの統合開発環境

JavaはCの流れを汲む手続き型であるが，Cのように規格化されてはいない。利害関係者から構成されるJava コミュニティ・プロセスによって標準化されてきた。Java コミュニティ・プロセスは便利な機能を積極的に採用する傾向にある。Javaはマルチパラダイム性を強め，Java8からは関数型プログラミング言語の特徴であるラムダ式やモナドを使えるようになった。これらの使用は，抽象メソッドのポリモーフィズムによって，複雑な機能を簡潔に記述できることに意味がある。Java7以前では，同じ機能を持つプログラムを書きにくい。

関数型のプログラミング言語の最大の特徴は，数値や文字などと同様に，関数を値として扱うことができることである。C言語では，関数へのポインタを引数として渡すことで実現可能である。アセンブリ言語や機械語でも，理論上はプログラミング不可能ではない。実際には，アセンブリ言語や機械語による，それらの機能の実現は人間によるプログラミングは不可能に近い。超人的なプログラマがプログラミングできたとしても，プログラムは可読性がなきに等しい。

JavaのクラスもCの構造体を拡張したものであり，クラス継承も関数へのポインタで実現可能であるが，可読性の点で実用的ではない。アセンブリ言語や機械語でも，理論上は実現可能であるが，バグのない大規模プログラムを作成するのは不可能に近い。

2.4 ActionScript

著者は，通常科目以外にも，経営学部のゼミナール（科目名としては，2年次配当が「演習I」，3・4年次配当が「演習II」）で，プログラミング教育を行ってきた。ゼミナールでは，2003年度から2005年までActionScript1.0を採用した。ActionScriptでは，ゲーム制作が比較的容易で，かつヴィジュアルプログラミングが可能だったからである。JavaScriptを拡張した言語であるため，JavaScriptの演習も兼ねられるという理由もある。

2005年度から2019年度までクラスベースのオブジェクト指向言語ActionScript2.0 に切り替えた。しかし，教員が作成するプログラムが，小規模なゲームのプログラムのみであったため，オブジェクト指向設計を行わなくても支障がなかった。

ActionScriptで組まれたプログラムはFlashPlayerで実行される。ActionScriptのFlashPlayerは，JavaのJVMに相当する。しかし，このFlashPlayerは，セキュリティホールの原因になるというApple社の見解により，2007年登場のiPhoneには，FlashPlayerアプリが存在しなかった。また，2020年にはApple Safariで，2021にはGoogle Chromeで，FlashPlayerプラグインのサポートが打ち切られた。2020年12月31日のAdobe社によるFlashPlayerサポート終了で，ActionScriptは言語としての命運が尽きたといえる。Scratchも2.0はActionScriptで書かれており，FlashPlayer上で動いていた。3.0はActionScriptではなくJavaScript，HTML5，CSSによって書かれており，ブラウザ単体で動作する。

2020年度からは，著者が担当するゼミナールではActionScriptの教育を打ち切り，Scratchを採用した。Scratchでは，市場で取引される商用アプリを作成できない。Scratchは，統合開発環境内で閉じており，汎用性が低く，今後も業務で使う言語にはなり得ない。しかし，ブロックも

統合開発環境も日本語に完全対応しているという利点がある。

ほとんどのプログラミング言語は，命令が英語ベースであるため，英単語の記憶が不正確な学生は，エラーを出しやすいというプログラミング教育上のデメリットがある。ActionScript の統合開発環境は日本語対応していたが，プログラムの記述は英語ベースであった。JavaScript を拡張した言語であるので当然だが，「Random」とクラス名を書くところを「Randam」とスペルを間違えるなど，プログラミングとは本来関係ない部分にも気を配る必要があった。

3. 2020 年度「プログラミング言語」

3 年次配当科目「プログラミング言語」は，2010 年度前後から 2019 年度まで，Java を対面で講義した。教科書は「明解 Java」を指定していたが，年度が進むごとに教員の参照機会が減った。著者の授業スタイルが主な理由であるが，学生が重い教科書を嫌がり，買っても持って来たがらなかったという理由もある。

3.1 新型コロナ対策が言語選択に与えた影響

2020 年度は，新型コロナウイルス感染拡大により，オンデマンド形式で実施した。オンデマンドに移行する際，当初は，例年通り教育内容を Java のみにする予定であった。しかし，以下の 4 つの理由から断念した。

1. PC 上で実行環境を構築できず，学生が授業の最初で躓く可能性がある
2. コンパイル時に構文エラーを頻繁に出すため，学生のモチベーションが崩れやすい
3. OS，エディタの違いにより，実行結果の表示の際に，文字化けが生じる可能性がある
4. オンデマンド形式では，教員が学生のプログラミング中の画面を確認できず，学生が出すエラーを解消しにくい

Java を教育用の言語として使う場合，多かれ少なかれ上記の問題が生じるが，オンデマンド形式では，対面授業より問題が大きくなる。授業効果が得られない可能性が大きいと判断し，急遽，2019 年度ゼミナールで試験的に教えていた Scratch を導入することにした。

経営学部の教員として，業務で使われる可能性がない言語を，ゼミナール以外の授業科目で教えることには抵抗があった。しかし，Scratch には，プログラミング初心者にも直観的な操作ができるシンプルな統合環境が存在する。かつ，構文エラーが原理的に発生しない。他の言語にはないメリットが Scratch には存在する。業務で多用される Java も教えれば良いという最終判断をした。

3.2 新型コロナ禍での2020年度授業実施

教育手法は，2019年度までは学会発表（小松2016）に記述されている方法を採用した。1コマ90分で，2～5個の小規模なプログラムを，解説しながら教員自らが組み，実行，デバッグし，リアルタイムで同様のプログラムを学生にも組ませ，実行，デバッグさせるというものである。

2020年度は，3月のCOVID-19の流行により，Streamでのオンデマンド配信になった。5月の連休明けから授業が開始されるため，4月中に，全15回の教育用コンテンツを作成した。Scratchは，これという無料オンライン講座が見あたらなかった。著者がScratchで解説しながらプログラミングし，著者の音声とデスクトップ画面を，Teamsの1人会議で録画する手法を選択した。

Javaに関しては，近年，Javaのオンラインプログラミング講座が多数あり，ほとんどの講座に無料部分がある。コロナを良い機会と捉え，様々な講座の無料部分をチェックした。その結果，Paizaラーニングの Java講座を採用した。解説が丁寧で自習に向くことと，Webサービスの Paiza.IOでJavaのプログラムを書き，コンパイル・実行ができることが特徴である。Paizaに会員登録する動画を作り，Paizaラーニングの Java講座の無料部分を自習するよう指示した。

3.3 2020年度「プログラミング言語」のレビュー

Scratchの教育に力が入りすぎ，Javaの教育が手薄になった。Paizaラーニングは，配列，クラスの継承あたりまで進むと有料部分の方が多くなる。プログラムは一貫してRPGを題材としたプログラムが提示される。学生にとって身近で良いと判断したが，RPGのプログラムを作るわけではないので実用性はない。

学生のモチベーションは続かなかったようで，授業評価アンケートの結果も例年に比べて悪かった。ただし，大学の標準化されたアンケートでは，Java教育が悪かったのか，Scratch教育が悪かったのか不明であった。ゲームに代表される，作った後も楽しめて改造するモチベーションが湧くプログラムを作る方が，学生の脱落数を抑えられると判断し，Scratchの比重を増やした。しかし，エビデンスはなく，著者の直観にすぎなかった。

ライブラリやフレームワークの類を使わずにゲームを作ることができるプログラミング言語は，現状，Scratchがファーストチョイスである。Scratchでは，ゲーム以外のプログラムも組むことができる。モジュールの単位がスプライトしかなく，かつモジュール結合を弱くすることができないなどの特徴があるため，大規模なプログラムは難しい。しかし，小規模なプログラムならば，同じ機能を持つプログラムをJavaとScratchの両方で作れる。似たようなプログラムを両方の言語でプログラミングすることにより，「プログラミング言語」という科目名に相応しい授業を展開可能と判断し，Web版Scratchでプログラムを組む様子を録画した動画を多数作成した（小松 2020a）。

Javaは代表的なオブジェクト指向言語である。しかし，オブジェクト指向の3 大要素，カプセル化・継承・ポリモーフィズムのうち，カプセル化までしか，授業内容に含めることができ

なかった。クラスの説明が第13回目の授業回に初登場するという状態であり，オブジェクト指向言語を教えているとは言えない授業内容だった。

実務でも，資格試験でも，クラスライブラリのコレクションフレームワーク，List，Map，Set は頻繁に登場する。特にJavaの配列に関しては，あらかじめ要素数が決まっている配列型ではなくArrayList型が使われる。ArrayListはクラスである。リストで実現されている配列であり，Listインターフェースが実装されている。LinkedListもクラスであり，要素の削除が多い場合は，ArrayListより速度的に有利になる。LinkedList型のオブジェクトにもListインターフェースが実装されているため，ArrayList型のオブジェクトと同様にList型の変数に代入するのが定石になっている。List型で定義されたメソッドを呼び出して，データ集合の操作を行うことになる。このようなArrayListやLinkedListの使い方は，継承やポリモーフィズムの理解がなければ困難である。

4. ScratchとJavaの並列進行

2021年度は，2020年度の反省を生かし，15回全授業回でScratchとJavaを講義した。授業コンテンツは，2021年度用に新規作成した。全体としてはScratchのコンテンツを削り，Javaのコンテンツを充実させた。Javaのオブジェクト指向プログラミングをポリモーフィズムまで習得させることを目指したからである。

2021年度の「プログラミング言語」では，全授業回において，Scratchの内容とJavaの内容を両方取り扱った。Scratchでは構造化プログラミングを教え，Javaでは，これらの内容は，Scratchで学習済みとして，ほとんど取り上げなかった。

4.1 資格試験対策

構造化プログラミングの3大要素，逐次・繰り返し・条件分岐を学ぶためのプログラムを，Javaでは最小限にし，主にScratchで教育した。授業コンテンツをJavaのオブジェクト指向を教えることにした理由は，授業を資格試験対策に役立てるようにする目標があったからである。

2021年春の情報処理技術者試験において，著者のゼミナールの4年生が基本情報技術者試験に合格した。基本情報処理技術者試験の午後試験は，アルゴリズムとプログラミング言語の配点が半分である。プログラミングを一通り理解していなければ合格はおぼつかない。基本情報処理技術者試験のアルゴリズム問題は疑似プログラミング言語で記述されている。しかし，プログラミング言語の問題は，アセンブリ言語，C，Java，Python，表計算の具体的なプログラミング言語で記述された5つの問題から1つ選択する。

2020年の秋に，当該学生から，「アセンブラを独学しているが，学習が進んでいる気がしない。勉強したがプログラムを全く解読できない」という相談を受けた。基本情報処理技術者試験のアセンブリ言語はCASLIIである。CASLIIはニーモニック数が30未満のシンプルな設計

である。文法も，1行に，ラベルがあれば最初にラベル，次にオペコード，最後にオペランドを並べるという，オーソドックスな文法である。情報系学科でコンピュータアーキテクチャの単位を取った学生ならば，数時間でマスターできる。しかし，書かれたプログラムを理解するためには，CPUのアーキテクチャと主記憶に関する理解が不可欠である。

CASLIIが前提とするハードウェアはCOMMET IIである。古典的なアーキテクチャの仮想的な16bitマシンである。しかし，本学には，コンピュータのハードウェアを教える科目が存在しない。また，CASLIIとCOMMET IIを独学したところで，いずれも，資格試験用の仮想的な言語とハードウェアである。IT企業への就職を目前にひかえた学習者のモチベーションは保たれにくい。

当該学生は，3年次に「プログラミング言語」を履修済みだった。しかし，当時の授業コンテンツは，Javaのみにも関わらず，構造化プログラミングが中心であり，オブジェクト指向プログラミングに関しては，3大要素のうちカプセル化までしかカバーできていなかった。

90分×15回のプログラミング教育の科目としては，カプセル化までが妥当なラインと判断していたが，内容が構造化プログラミングだけならば，言語はJavaである必然性はない。継承とポリモーフィズムまでカバーするのは，実効性の面で危険と判断していたが，資格試験の学習過程で，初見という理由で理解しようとする気力を失う危険もある。

当該学生は授業の他には，独学を含めてJavaの学習経験がなかった。ゼミナールではActionScriptを，2年強程度，学習したのみであった。基本情報処理技術者試験のプログラミング分野において，Javaの問題が最も難易度が高いというのが定説である。Javaの選択も薦められない状況だったが，業務用プログラムはJavaで組まれることが多い点と，プログラムの可読性の点から，Javaを選択するようにアドバイスした。

本学経営学部で，情報処理技術者試験の対策となる授業は，2年次配当科目の「コンピュータ検定（ITパスポート）」しかない。非情報系学部の学生にとって，基本情報処理技術者試験では，プログラミング言語問題の対策が困難であるというのが定説である。さらに，令和2年度午後試験から，100点満点中25点と，点数の占める割合が25%まで上げられた。著者が担当する「プログラミング言語」の授業におけるJavaの教育内容を，Java問題対策になるように変更し，ITに強い学生が試験に合格する確率を上げることを，シラバスに書かない目標にした。

4.2 オブジェクト指向プログラミング

授業コンテンツを一新したもう一つの理由は，著者自身が構造化プログラミング中心からオブジェクト指向プログラミング中心に教育の軸足を移したかったというものが挙げられる。構造化プログラミングは，Scratchの方が理解しやすく教えられると判断した。図17は，繰り返しの中に条件分岐を入れるという構造化されたプログラムの代表例である。2021年度は，図17のプログラムを第3回という早い授業回で取り扱った。

図17 第3回目の授業で提示したプログラム

現在，商用アプリケーションの作成に使われているプログラミング言語は，C を除けば，ほぼすべてオブジェクト指向の言語である。チームでプログラムを組む場合には，各メンバーが作るプログラムは，モジュール強度が高い必要があり，モジュール強度が高いプログラムを書くためには，オブジェクト指向のプログラミングパラダイムが有利だからである。オブジェクト指向プログラミングパラダイムは3大要素によって実現される。カプセル化と継承とポリモーフィズムである。

データとデータに対する操作であるメソッドを一塊にして，他のクラスからのデータに対する操作は，クラス外部に公開するようにアクセス制御したメソッドを通じてのみ許可する。それがカプセル化である。カプセル化されたクラスを利用するプログラマは，メソッドのシグニチャを認知していれば，クラスの内部構造を知る必要がない。クラスの内部構造を変更が変更されても，メソッドのシグニチャを変更されなければ，メソッドを呼び出すプログラムに変更の必要性は生じない。

各オブジェクトに共通する性質を抽出してスーパークラスを作り，スーパークラスから，継承でサブクラスを作り，スーパークラスにない性質を新たに加えたり，スーパークラスで宣言・定義したメソッドを実装・再定義したりする。それが継承である。継承を使用すれば，似たようなモノや概念を，ひとまとまりとして扱うことができる。具体的には，スーパークラスの宣言・定義によって，スーパークラスの参照型が定義できる。スーパークラスを参照する変数には，当該スーパークラスを継承したサブクラスのインスタンスへの参照を代入することができる。スーパークラスの参照型の配列・リストを宣言し，配列・リストの要素に当該スーパークラスの何種類かのサブクラスへの参照を代入するのが典型的な例である。

図18はカプセル化とクラス継承を理解させるためのプログラムである。Human クラスを宣言して，Student クラスは Human クラスを継承している。Main クラスでは，Human クラス型の

オブジェクトへの参照をaに代入し，Studentクラス型のオブジェクトへの参照をs1とs2に代入している。

Javaにおける参照はCにおけるポインタである。「Javaにはポインタがない」という記述は不適切である。「Javaではポインタが表面には出てこない」や「Javaではアドレス値をプログラマが使用することはない」という表現を使うべきであるが，Cを知らない学生に参照を教えるのは難しい。

```
1  class Human {
2  ^ private String name;//フィールド
3  ^ Human(String name){this.name = name;}//コンストラクタ
4  ^ public void work(){System.out.println(this.name+"は生きる");}//メソッド
5  }
6
7  class Student extends Human {//Humanクラスを継承
8  ^ private int id;//学籍番号フィールド
9  ^ private static int n = 0;//学籍番号をつけるためのクラス変数
10 ^ Student(String name){
11 ^ ^ super(name);//Humanクラスのコンストラクタをコール
12 ^ ^ id = ++n;//nの値に＋１した値をidに代入
13 ^ }
14 ^ public int getId(){return this.id;}//ゲッタ
15 }
16
17 public class Main{
18 ^ public static void main(String[] args){
19 ^ ^ Human a = new Human("アダム");
20 ^ ^ a.work();
21 ^ ^ Student s1 = new Student("文京花子");
22 ^ ^ s1.work();
23 ^ ^ System.out.println("学籍番号は"+s1.getId()+"です。");
24 ^ ^ Student s2 = new Student("文京太郎");
25 ^ ^ s2.work();
26 ^ ^ System.out.println("学籍番号は"+s2.getId()+"です。");
27
28 ^ }
29 }[EOF]
```

図18 カプセル化とクラス継承

ポリモーフィズムは，継承の際のサブクラスにおけるメソッドの再定義とJVMによるメソッドのダイナミックバインディングによって実現される。ダイナミックバインディングでは，呼び出されるメソッドが，コンパイル時に静的に決まるのではなく，実行時のオブジェクトの型によって決まる。スーパークラスの参照型の配列・リストには，サブクラスへの参照が自動的にアップキャストされて代入される。このとき，スーパークラスのサブクラスが多種類かつ，サブクラスでスーパークラスのメソッドが再定義オされていれば，当該スーパークラスのメソッドを呼び出したときに，サブクラスで定義されたメソッドが，オブジェクトの型に応じて呼び出される。

ポリモーフィズムは具象クラス，抽象クラス，インターフェースのいずれでも実現可能である。しかし，時間的にすべての場合のプログラムを取り扱うことはできなかったため，イン

ターフェースを使った場合のポリモーフィズムに絞った。インターフェースは，厳密にはクラスではないが，抽象クラスの抽象度をさらに強めた概念である。インターフェースはメンバーとして，抽象メソッドと public かつ static かつ final 修飾子がつけられたフィールドしか持つことができない。インターフェースは，継承されるのではなく，実装され，複数のインターフェースをクラスに実装することにより，C++ における多重継承に相当する実装も実現可能である。

図 19 は Swing の Jframe クラスが継承したクラスと，実装したインターフェースを表す。クラスライブラリは API としてカプセル化されている。プログラマは，継承とインターフェースを理解していれば，何を実現するためのクラスなのかが，ライブラリ内のコードを見なくても概略として把握できる。

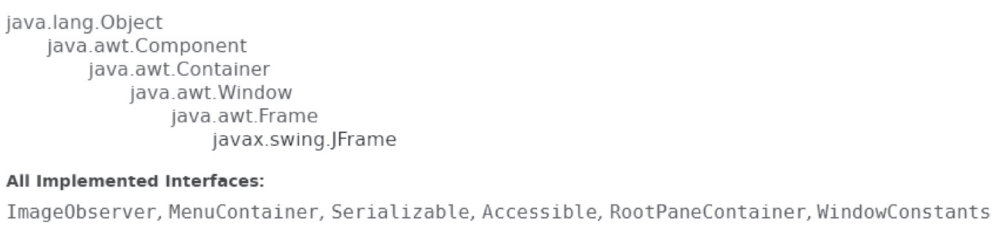

図 19 JFrame が継承したスーパークラスと実装されたインターフェース

チームで大規模な Java プログラムを作成する場合，チームごとにパッケージを作成することが多い。あるパッケージから他のパッケージ内のクラスを利用する場合，インターフェースを使う場合が多い。インターフェースを使わせるのが，オブジェクト指向設計によって作成されたプログラムの特徴ともいえる。また，基本情報技術者試験の Java プログラミングの問題でもインターフェースは登場する。

図 20 はインターフェースを使ったポリモーフィズムを理解させるために，「プログラミング言語」の授業で提示したプログラムである。インターフェースの抽象メソッドを，クラスで実装する際に，「@Override」で，オーバーライドされたメソッドであることを明示している。メソッド名のスペルを間違えると，全く別のメソッドとして，コンパイルできてしまうからである。

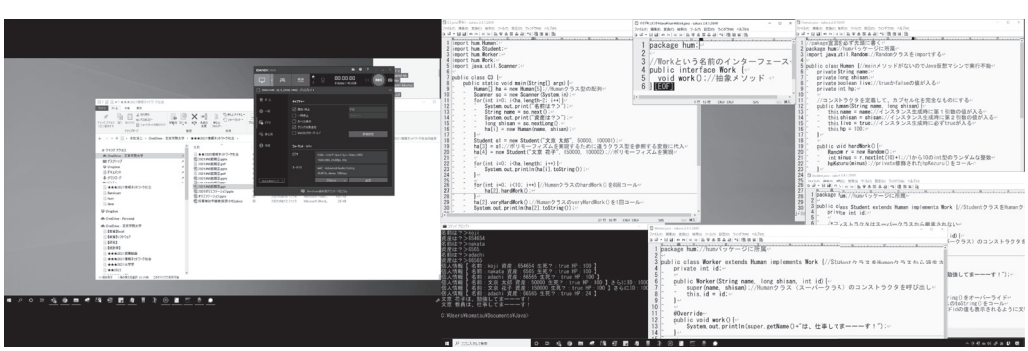

図 20 インターフェースの実装によるポリモーフィズムの実現

5.　2021年度授業実施とアンケート結果

2021年度の経営学部におけるコロナ対策は，履修科目が教室定員の半分を上回った科目では，ハイフレックス形式にするというものだった。Formsで作成したアンケートを，2021年4月9日の第1回授業日の開始前にTeamsの一般チャネルに貼り付けた。図21のように，N = 93で，完全オンデマンド形式を希望する学生が7割強であった。オンデマンド形式には，教員が授業動画を，入念に設計・録画・配置できるというメリットもある。総合的な判断で，2021年度「プログラミング言語」はオンデマンド形式で実施した。

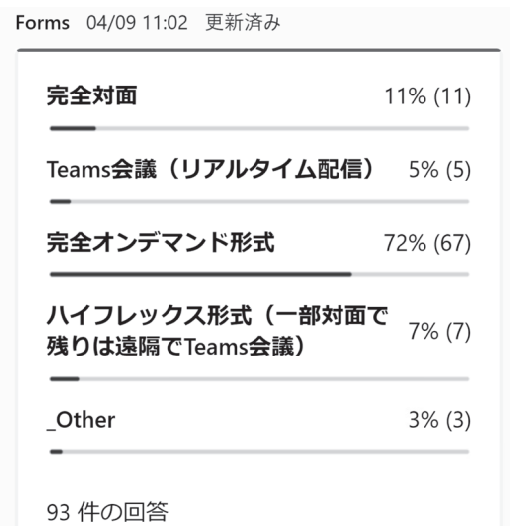

図21 第1回授業前のアンケート

5.1 2021年度「プログラミング言語」授業実施

第1回の授業では，以下の事項をオリエンテーションとしてTeams会議で学生に伝達した。

1. 1本の動画の長さが10分～ 30分になるように，教員がプログラミングしているデスクトップ画面を録画
2. 授業日の一週間前までに授業動画3本～ 7本をStreamに教員がアップロードしておく
3. 授業動画へのリンクはTeamsのチャネル（01回0409，02回0416，…）に教員が張っておく
4. 学生は授業動画をいつどこで見てもよい
5. 毎週金曜4限開始時に各授業回の授業内容を踏まえたFormsクイズ課題を，教員が閲覧・提出可能にする
6. 学生は翌日土曜23時までにFormsクイズ課題を提出

7. クイズ課題の提出をもって，各授業回に出席したものとみなす
8. 毎週のクイズ課題とは別に，第15回に最終レポート課題を課す
9. 最終レポート課題は，「Scratch か Java のどちらか，できれば両方でプログラムを作り，作ったプログラムを解説する」
10. クイズ課題と最終レポート課題の点数で成績をつける。比重は半々

以上のオリエンテーションに続いて，Scratch のアカウント作成と統合開発環境の操作方法を実演し，「動き」ブロックを使って猫スプライトを動かすプログラムを書いて実行した。続いて，サクラエディタのダウンロードとインストールし，エディタで文章を書く実演をした。Java のダウンロードとインストールおよび環境設定を実演した。続いて，エディタで「Hello, World. こんにちは，Java」のプログラムを書き，文字コードを Shift JIS にして保存し，コマンドプロンプトで，javac コマンドでソースファイルをコンパイルして，java でクラスファイルを実行した。Java の環境設定ができない学生向けにオンライン実行環境である ideone.com で「Hello, World. こんにちは，Java」のプログラムを書いて実行し，Java の予習復習用サイトとして，Paiza ラーニングを紹介した。

第2回目以降は，Scratch の動画と Java の動画を1回の授業回に混ぜて配置した。前半は Scratch のプログラミングの動画が多く，後半になればなるほど，Java のプログラミングの動画が多くなった。

5.2 アンケート結果

履修登録が確定する前に，Teams の「プログラミング言語」チームに登録した学生は116名であった。その時点で，経営学部で推奨されていたハイフレックス授業[1]が実施不可能になった。受講学生の1/3の学生でも，COVID-19対策で定員を半分にした教室に収容できなくなったからである。オンデマンド形式での実施を宣言した後に，履修登録をしなかった学生は8名存在した。最終的に，教務システムである Bs Link に登録した学生数は106名であった。

教務システムに登録済みの学生でも，全15回の授業回で出題された Forms のクイズ課題を，ほとんど提出しなかった学生も存在する。提出しなかった学生は2名で，1回，2回，3回だけ提出した学生はそれぞれ1名であった。その他の提出回数の学生も含めて，提出回数9回以下の学生数を図22 に示す。

[1] 受講者を3分割して，1/3の学生に対して対面授業を行い，対面授業を撮影して2/3の学生に原則リアルタイム配信。見逃した学生向けに録画済み動画アップロードしておく

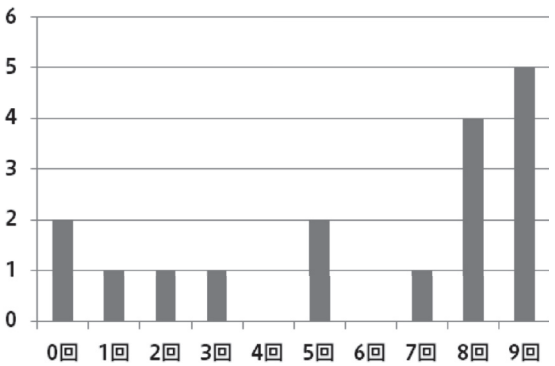

図22 Formsクイズ課題提出9回以下の学生の分布

授業回数は15回であるため，10回の提出で出席要件を満たす。少なくとも17名が出席不良により単位を落とした。第15回目には，大学として採る授業アンケートとは別に，教員独自の授業アンケートをFormsで採った。履修登録者数106名で，回答者数73名である。選択バイアスと生存バイアスがかかるのは免れられない（小松2020b）。

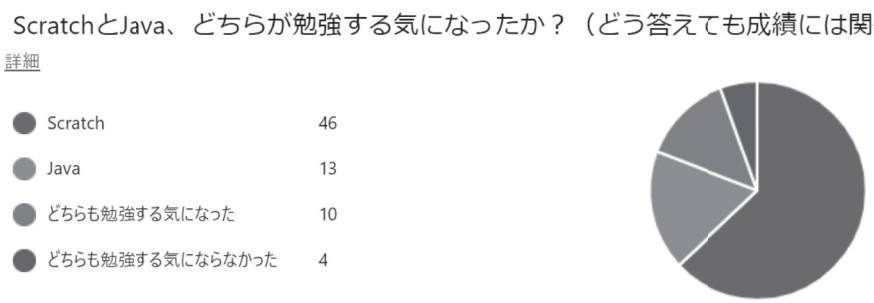

図23 ScratchとJavaに対する勉学意欲

学生の勉学意欲に関しては，図23に示されるとおりScratchの方が高かった。受講学生の中に，著者が担当するゼミナールに所属中でScratchのみを学習している学生が10名含まれていた。そのうち，相当数が，主に目新しいという理由で，Javaの方に強い学習意欲を感じたと推測される。また，授業のチームに，Javaに関して，プログラミング言語としてのシェアの大きさを解説した動画をアップしていた。Scratchに関しては，シェアに関する説明はしなかった。

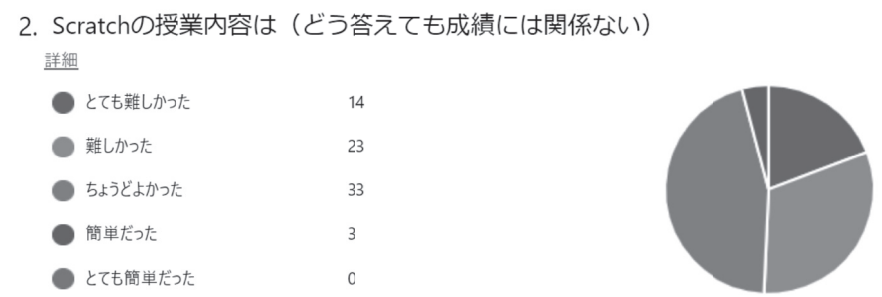

図24 Scratchコンテンツの難易度

Scratchのプログラミングでは，3年次のゼミナールで作成するプログラムと比較すると意図的にレベルを落としたプログラムを取り扱った。しかし，10名のゼミナール所属学生が履修していながらも，図24のように,「とても簡単だった」と答えた学生はいなかった。著者のゼミナールにおけるScratchの教育内容も，ゼミナールの所属学生にとって，消化不良気味になっていた可能性は否めない。

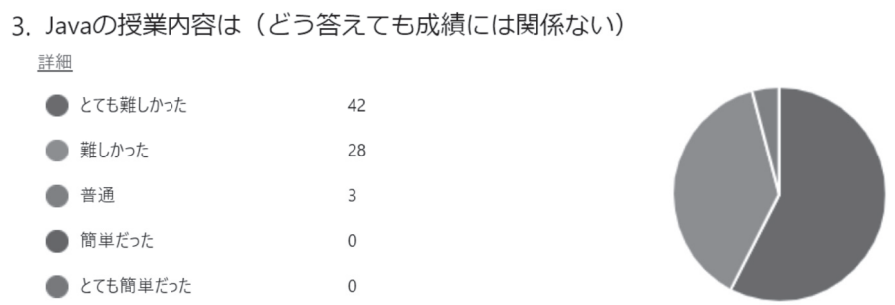

図25 Javaコンテンツの難易度

Javaに関しては，図25のように,「とても難しかった」「難しかった」という回答が圧倒的だった。クラス継承とメソッドのオーバーライド，多重継承を実現できるインターフェースを教えた上で，インターフェースで宣言された抽象メソッドのオーバーライドを使った実装によるポリモーフィズムまで扱うことを目標にコンテンツを構成した。著者がJavaのオリジナルなプログラムをプログラミング・デバッグしている様子を録画した動画がコンテンツの中心であり，授業方式はScratchと同じであるが，Scratchに比べると内容自体が難しすぎたといえる。

Javaも簡単な部分ならば，ほとんどの教員は簡単に教えられるはずである。今回，Javaの難しい部分を簡単に教えようと試みた。特に教えるのが難しかったのはJavaのインターフェースである。クラスをよく理解できていないと，抽象クラスもインターフェースも理解できるはず

がない。アンケート結果が出てくる前までは，来年度以降，ラムダ式まで教えようと考えていたが，ラムダ式の実体は，抽象クラスでありインターフェースである。図25の結果からは，現状のままラムダ式を導入するのは，リスクが高すぎであり，まずはクラスの理解を深めさせるべきであることが推測できる。

6. 将来展望

現在，コンピュータソフトウェアやスマートフォンアプリケーションが使われていない業種はないといえる。しかし，プログラミングの経験がなければ，それらの挙動は魔術に近い。自分でプログラムを組んだことがあれば，あるいは組めるまでいかなくても，成果物であるソースコードを見れば，ソフトウェアやアプリ制作の面白さや苦労，自身のプログラミング適性がわかるはずである。

プログラミングに関して，ノーコードあるいはローコードが主流になるという見解もある。ローコードは程度の問題であり，すでに業務レベルに浸透しているといえる。近年のプログラミング言語，特にPythonには強力な，プログラミング教育に使うには強力すぎるモジュール・パッケージが存在する。PythonがAI向き言語と呼ばれ，人気が上昇しているのは，科学技術計算に関する優れたモジュールが標準で存在するからである。Javaの現在のクラスライブラリは，まさに汎用であり，科学技術計算が重視されているわけではない。Scratchはライブラリがないに等しい。リストの要素の並び替えのコードも，プログラマが作成する必要がある。

ノーコードに関しては，Microsoft PowerAppsがExcel VBAやAccess VBAにとって替わる可能性はある。しかし，ノーコードであれ，業務を厳密に定義し，業務プロセスを自然言語で厳密に記述できないならば，アプリケーションは作れない。プログラムは仕様書に依存し，Javaはいかん仕様にも対応できる。しかし，ノーコードは，定型的な仕様にしか対応できない。

ノーコードやローコードは，プログラミング経験者がスピーディに定型的なアプリケーションを作成する技術であり，プログラミング教育の必要性はなくならないと予想する。

7. まとめ

Scratchは，Javaと比較して，学生にとって馴染みやすく理解しやすいプログラミング言語であることが確認できた。日本語でプログラミングできることも，メリットになっているはずである。教員にとっても，構文エラーが出ないという点で，斬新なプログラミング言語である。その反面，テキストベースの言語とは勝手が違い，コードの一覧性がなく，かつ検索しにくく，ある程度の規模以上のプログラムは扱いにくい。

Scratchは中等教育での使用を想定して設計された教育用言語である。ブロックを組み立てで機能を実現できる。Scratchは統合開発環境を提供するサイトで閉じており，汎用性がない。

また，モジュールを疎結合にしにくいことに起因するプログラムの保守性と再利用性に関する問題がある。

Java は大規模なプログラム開発では，デファクトスタンダードな汎用言語である。汎用的なオブジェクト指向言語であり，言語仕様は複雑である。しかし，オブジェクト指向の考え方は強力で，メソッドを含めてクラスを宣言することにより，モジュールが疎結合で，保守性が高いプログラムを作ることができる。

【参考文献】

⑴ 柴田望洋 2007,“明解 Java 入門編”, SB クリエイティブ
⑵ 小松香爾 2016,“学生からのフィードバックをリアルタイムで利用するプログラミング教育の実践”, 日本教育情報学会第 32 回年会, 2016 年 8 月
⑶ 小松香爾 2020a,“ Teams を用いた 3 科目のオンライン授業の比較”, 日本教育情報学会第 36 回年会, 2020 年 8 月
⑷ 小松香爾 2020b,“エビデンスに基づいた教育”, 文京学院大学総合研究所経営論集第 30 巻第 1 号, 2020 年 12 月