すべての人のための

Javaプログラミング

第3版 Java for Everyone

立木 秀樹
有賀 妙子 著

Hideki Tsuiki and Taeko Ariga

共立出版

すべての人のための

Javaプログラミング

第3版 Java for Everyone

立木 秀樹
有賀 妙子　著

Hideki Tsuiki and Taeko Ariga

共立出版

第3版 まえがき

本書第 2 版が出版されてちょうど 10 年になります。長年にわたり，多くの学校で Java 言語の教科書として使っていただいていることを嬉しく思います。

この間に，Java 言語をとりまく環境は大きく変化しました。情報機器はめまぐるしく進化しています。Java は 1990 年代後半にインターネットの普及とともに生まれた言語ですが，もはや「新しい言語」ではありません。Java を生み出したサンマイクロシステムズは，オラクルに買収されてしまいました。 その中で，Java 言語も進化してきました。特に，2014 年に発表された Java SE 8 で大きなバージョンアップがなされました。ラムダ式などの関数型プログラミングの概念が導入され，プログラムの書き方が大きく変わりました。また，高機能な GUI を作成できる JavaFX ライブラリが使えるようになりました。並列，並行プログラミングの機能も強化されています。Java 言語は，当初はブラウザの画面上で動く applet を書けることで注目を集めましたが，現在では，オブジェクト指向的なシステム記述言語やスマートフォン上のアプリの開発言語として定評を得ているように思います。

このような Java 言語の変化を取り入れた内容に刷新したのが，この第 3 版の教科書です。関数型プログラミングの機能について丁寧に説明しました。後半で扱う GUI のライブラリは，Swing から JavaFX に切り替えました。また，コレクションや並行プログラミングについても丁寧な説明を追加しました。

第 2 版までと同じように，前半（12 章まで）は，サポートページからダウンロードしたタートルグラフィックスのライブラリを用いて，視覚的にプログラムの動きを理解しながら，段階的に Java 言語の概念を学習していきます。今まで以上に分かりやすいものにしたいと思い，これまでの授業での経験を踏まえて，説明の流れを再構成しました。 タートルグラフィックスのインターラクティブ性を高めたので，最初から楽しみながら学習できることと思います。また，タートルグラフィックスは，並列プログラミングや再帰プログラミングを分かりやすく説明するのに役立っています。後半（13 章以降）は，JavaFX を中心としたライブラリの説明を行っています。JavaFX のアニメーションなどの機能を用いて，より楽しめるアプリケーションが作れるようになっています。

内容は膨大ですが，教科書として授業で使うことを意識して編成してあります。前半は，さらに二つの部分に分かれます。第 1 章の導入の後，2〜7 章はプログラミングの入門から始めてオブジェクト指向の基本的な内容を扱い，8〜12 章で本格的なオブジェクト指向の概念を扱います。また，後半は，13〜15 章で JavaFX ライブラリを使ったグラフィカルユーザインターフェースの作り方について学び，第 16 章で実用に必要となる入出力，第 17 章でさらに進んだ内容としてネットワークプログラミングについて学びます。初学者に向けた半年の授業なら，GUI を持ったプログラムを作ることを目的にするのがいいと思います。その場合，8〜12 章は飛ばして，13〜15 章を授業で扱うことが考えられます。JavaFX のイベント処理はラムダ式で記述しますが，とりあえず書くのは，それほど複雑ではありません。また，Swing と違って，高度なオブジェクト指向の概念をそれほど用いないので，飛ばした部分は必要に応じて参照しながら授業を進

められるのではないかと思います。ラムダ式を用いると JavaFX のイベント処理などが簡単に書けるのは確かですが，それ自身は複雑な概念に基づいて作られており，それに対する理解がないと，本格的なプログラムを書くのは難しいです。ですので，その上で 8 章からの自習を勧めて頂きたいです。また，多少プログラミングの経験のある人に向けた，高度なプログラムを扱う授業なら，逆に，8〜12 章と 16 章以降に力点をおいて授業をするのがいいでしょう。

本書のタートルグラフィックスライブラリは, 慶應義塾大学環境情報学部の萩野達也教授が作成されたものを元にしています。快く使わせて下さったことを感謝します。また, 2017 年春学期に，本書の草稿を用いた授業を京都大学で行いました。有益なフィードバックを下さった学生の皆さんと TA の高山勉君に感謝します。

新しい概念が現れる度に，それに対応する練習問題をつけて，一歩ずつ自分で確認しながら進められるようにしました。また，プログラミングに対する興味を引き出すことを主眼にした発展的な練習問題も多数載せました。後者の練習問題には * をつけてありますので，自分に必要な練習問題を取捨選択して頂ければと思います。例題や演習問題には，数学的なものも幾つかあります。数学が苦手な読者も多いかもしれませんが，プログラミングにおいて，抽象化された概念を扱う能力の重要度は増してきていると思います。是非，チャレンジしてみてください。

これまでの版と同様に，本書のサポートページ

`http://www.i.h.kyoto-u.ac.jp/users/tsuiki/javaEveryone3`

で提供しているライブラリを利用してプログラムを組みながら学習を進めるように作られています。コマンドライン（ターミナルやコマンドプロンプト）でも，Eclipse などの統合環境でも学習できるように，サポートページのプログラムは両方のバージョンを用意しました。例題を実行して，練習問題を解きながら読み進めてください。練習問題の解答も公開しているので，難しい練習問題は解答を動作させるだけでも勉強になると思います。

では，勉強を楽しんでください。

2017 年 7 月

立木秀樹
有賀妙子

目　　次

第1章　はじめに　1

1.1 インターネット環境でのプログラミングと Java 1
1.2 大規模プログラミングとオブジェクト指向 2
1.3 仮想マシン 2
1.4 さらなる進化 5
1.5 準備 — ソフトウェアなどの入手 — 5

第2章　オブジェクトの生成とメソッド呼び出し　7

2.1 オブジェクトとクラス 7
2.2 最初の例題 — オブジェクトの生成とメソッド呼び出し — 8
2.3 コンパイルと実行 11
2.4 TurtleFrame と Turtle の API の仕様 12
2.5 次の例題 — プリミティブ値，コンストラクタ，ライブラリの利用 — 14
2.6 インポート宣言 17
2.7 さらにもう一つの例題 — インスタンス変数 — 18
2.8 オブジェクトとは？ 22

第3章　処理の流れ　24

3.1 for 文による繰り返し 24
3.1.1 同じことを繰り返す 24
3.1.2 繰り返しをカウントする変数値の利用 27
3.1.3 繰り返しのネスト 28
3.2 while 文による繰り返し 30
3.3 論理演算子 30
3.4 if 文 32
3.5 break 文 と continue 文 33
3.6 switch 文 35

第4章　クラス変数とクラスメソッド　36

4.1 クラス 36

4.2 クラス変数 37
4.3 クラスメソッド 37
4.4 Java API に現れるクラス変数，クラスメソッド 39
4.4.1 Java API の利用 39
4.4.2 javafx.scene.paint.Color クラス 40
4.4.3 Math クラス 40
4.4.4 System クラス 42
4.4.5 String クラス 43

第 5 章 クラスの作成 44

5.1 メソッド 44
5.1.1 メソッドの追加 44
5.1.2 メソッドと段階的なプログラムの開発 48
5.1.3 サブクラスと継承 49
5.1.4 メソッドのオーバーロード（多重定義） 49
5.2 インスタンス変数 50
5.3 クラス変数とクラスメソッド 53
5.4 パッケージとアクセス修飾子 54
5.4.1 パッケージ 54
5.4.2 アクセス修飾子 55
5.4.3 カプセル化 56
5.5 コンストラクタ 57
5.6 内部クラス 59
5.7 まとめの問題: グラフの描画 60

第 6 章 配 列 64

6.1 配 列 64
6.2 初期値をもった配列 67
6.3 可変長引数 69
6.4 拡張された for 文 69
6.5 配列の配列 70
6.5.1 多次元配列 70
6.5.2 配列の配列 71
6.6 Arrays クラス 72
6.7 main の引数 73
6.8 まとめの問題: 折れ線の描画 74

第7章　プリミティブ型とラッパークラス　76

7.1　プリミティブ型 . . . 76
7.2　演算子 . . . 77
7.2.1　数の演算とキャスト . . . 77
7.2.2　変数への代入とキャスト . . . 78
7.2.3　式と文 . . . 79
7.2.4　変数の値を変化させる演算子 . . . 80
7.2.5　演算子の優先順位 . . . 81
7.3　ラッパークラス . . . 81
7.4　列挙型 . . . 83
7.5　ガーベッジコレクション . . . 83
7.6　クラスパス . . . 84

第8章　再帰呼び出しと例外処理　85

8.1　メソッド呼び出しとスタック . . . 85
8.2　再帰呼び出し . . . 86
8.2.1　再帰的なメソッド定義 . . . 86
8.2.2　木のフラクタルの描画 . . . 88
8.2.3　自己相似図形の描画 . . . 89
8.3　例外処理 . . . 93
8.3.1　例外 . . . 93
8.3.2　try-catch 文 . . . 94

第9章　メソッドのオーバーライドとインターフェース型　97

9.1　メソッド呼び出しと型検査 . . . 97
9.1.1　型検査 . . . 97
9.1.2　クラスと型の関係 . . . 98
9.1.3　メソッドのオーバーロードの解決 . . . 99
9.1.4　クラス型のキャスト . . . 99
9.2　メソッドのオーバーライド（再定義） . . . 101
9.2.1　メソッドのオーバーライドとメソッド探索 . . . 101
9.2.2　例：Tensen クラスでの fd の再定義 . . . 101
9.2.3　super メソッド呼び出し . . . 102
9.2.4　オーバーライドされたメソッドの呼び出し . . . 103
9.2.5　動的結合 . . . 103
9.2.6　スーパークラスで定義されたメソッドからの呼び出し . . . 105
9.2.7　Object クラスのメソッドのオーバーライド . . . 107
9.3　フィールドの隠蔽 . . . 108

9.4 抽象メソッドと抽象クラス 108
9.5 インターフェース 109
9.5.1 インターフェース型 109
9.5.2 インターフェースを実装したクラス 109
9.5.3 インターフェース型の変数を用いたプログラム 110
9.5.4 インターフェースのその他のメンバー 112

第 10 章 ラムダ式と関数型インターフェース 113

10.1 ラムダ式と関数 113
10.2 ラムダ式を引数や返値にするメソッド 116
10.3 ラムダ式によるローカル変数のキャプチャ 117
10.4 メソッド参照 119
10.5 java.util.function パッケージの関数型インターフェース 119

第 11 章 コレクションフレームワーク 121

11.1 コレクションフレームワーク 121
11.2 List インターフェースと ArrayList クラス 122
11.3 LinkedList クラス 126
11.4 イテレータ 127
11.5 Set インターフェースと SortedSet インターフェース 128
11.6 Map インターフェースと HashMap クラス 129
11.7 Comparator とソート 132
11.8 コレクションと配列の変換 133
11.9 ワイルドカード 134
11.10 まとめの問題: 線画編集プログラム 136

第 12 章 マルチスレッドと並行処理 137

12.1 スレッド 137
12.1.1 Thread クラスと Runnable インターフェース 137
12.1.2 ラムダ式とスレッド 139
12.2 sleep と割り込み 141
12.2.1 join による同期 141
12.2.2 sleep メソッドと割り込み 141
12.2.3 割り込みとスレッドの強制終了 142
12.3 タスクとエグゼキューターサービス 143
12.3.1 エグゼキューターサービス 144
12.3.2 Callable インターフェースと Future オブジェクト 145
12.3.3 invokeAny と invokeAll 146

12.4 スレッド間の同期 . . . 148
12.4.1 synchronized による排他的実行 . . . 148
12.4.2 変数の共有とデッドロック . . . 149
12.4.3 アトミック変数 . . . 150
12.5 スレッドの協調動作 . . . 151
12.5.1 CountDownLatch . . . 151
12.5.2 CyclicBarrier . . . 153
12.5.3 Semaphore（セマフォ） . . . 154
12.5.4 wait と notify . . . 155
12.6 コンカレントコレクション . . . 157
12.6.1 同期化コレクション . . . 157
12.6.2 コンカレントコレクション . . . 157
12.6.3 ConcurrentHashMap . . . 158
12.6.4 CopyOnWriteArrayList . . . 158
12.6.5 BlockingQueue . . . 159
12.7 ストリーム . . . 160
12.7.1 ストリームによる宣言的なデータ変換の記述 . . . 160
12.7.2 並列ストリーム . . . 163

第 13 章 GUI クラス 165

13.1 GUI クラス . . . 165
13.1.1 GUI クラスの役割 . . . 165
13.1.2 GUI ライブラリの種類 . . . 166
13.1.3 JavaFX のクラス . . . 167
13.1.4 UI コンポーネントクラスのプロパティ . . . 170
13.2 JavaFX アプリケーションの基本 . . . 171
13.2.1 枠組み . . . 171
13.2.2 画像の表示 . . . 173
13.3 UI コンポーネントの配置とスタイル . . . 175
13.3.1 概　要 . . . 175
13.3.2 Region クラスのプロパティ . . . 176
13.3.3 レイアウトコンテナ内のノードの大きさと位置合わせ . . . 177
13.3.4 FlowPane クラス . . . 179
13.3.5 BorderPane クラス . . . 180
13.3.6 VBox クラス . . . 181
13.3.7 HBox クラス . . . 183
13.3.8 GridPane クラス . . . 184
13.4 CSS によるスタイルの指定 . . . 186

13.4.1 JavaFX CSS の概要 . . . 186
13.4.2 スタイルの指定場所 . . . 186
13.4.3 スタイル属性の値 . . . 188
13.4.4 CSS ファイルとその指定 . . . 191
13.5 UI コンポーネントの API . . . 192
13.5.1 ラベル：Label . . . 192
13.5.2 ボタン基本抽象クラス：ButtonBase . . . 193
13.5.3 Button . . . 194
13.5.4 CheckBox . . . 195
13.5.5 RadioButton . . . 196
13.5.6 ComboBox . . . 197
13.5.7 TextField . . . 198
13.5.8 メニュー . . . 199
13.5.9 Chart . . . 200

第 14 章 イベント処理 203

14.1 イベント処理の仕組み . . . 203
14.1.1 概　要 . . . 203
14.1.2 イベント処理の流れ . . . 204
14.1.3 EventHandler の設定 . . . 206
14.1.4 イベントの情報を知るメソッド . . . 208
14.2 アクションイベント処理 . . . 209
14.2.1 Button 上のアクションイベント . . . 209
14.2.2 CheckBox 上のアクションイベント . . . 210
14.2.3 TextField 上のアクションイベント . . . 212
14.2.4 MenuItem 上のアクションイベント . . . 214
14.3 マウスイベント処理 . . . 215
14.4 キーイベント処理 . . . 217
14.5 プロパティ値の変化に応じたイベント処理 . . . 218
14.6 プロパティ値へのバインディング . . . 221
コラム　JAR ファイルとアプリケーションの実行 . . . 223

第 15 章 グラフィックス 224

15.1 描画の概要 . . . 224
15.1.1 Shape クラスと Canvas クラス . . . 224
15.1.2 座　標 . . . 225
15.1.3 座標変換 . . . 225
15.2 Shape クラス . . . 226

15.2.1 概　要 . . . 226
15.2.2 Shape オブジェクトへのイベントの設定 . . . 229
15.3 Canvas クラス . . . 235
15.3.1 図形描画メソッド . . . 235
15.3.2 Canvas オブジェクトへのイベントの設定 . . . 239
15.4 アニメーション . . . 241
15.4.1 概要 . . . 241
15.4.2 Transition のサブクラスによるアニメーション . . . 241
15.4.3 Timeline によるアニメーション . . . 243
15.4.4 AnimationTimer によるアニメーション . . . 246
15.4.5 音声の再生 . . . 250
コラム　アプリケーションの Web ブラウザからの実行 . . . 252

第 16 章 データ入出力　253

16.1 データ入出力の概要 . . . 253
16.1.1 io パッケージ . . . 253
16.1.2 ストリームクラス . . . 253
16.1.3 データ入出力の基本 . . . 256
16.2 文字ストリームの使用例 . . . 257
16.2.1 ファイルを 1 文字ずつ読む . . . 257
16.2.2 ファイルを 1 行ずつ読む . . . 259
16.2.3 キーボード入力から文字データを読む . . . 260
16.3 書式付き入力 . . . 262
16.3.1 Scanner クラス . . . 262
16.3.2 区切り文字による入力の解析 . . . 263
16.3.3 パターンを使った入力の解析 . . . 265
16.4 書式付き出力 . . . 266
16.4.1 書式付き出力メソッド . . . 266
16.4.2 書式指示子 . . . 267
16.4.3 標準出力への書式付き出力 . . . 268
コラム　リソースファイルの指定 . . . 269

第 17 章 ネットワークインターフェース　270

17.1 ネットワークプログラミングの基本知識 . . . 270
17.1.1 サーバとクライアント . . . 270
17.1.2 IP アドレスとホスト名 . . . 271
17.1.3 ポート番号 . . . 271
17.1.4 ソケット . . . 271

17.2 クライアントプログラム 272
17.3 サーバプログラム 275
17.3.1 サーバ側のソケット 275
17.3.2 カウンセラーサーバ 276
17.3.3 カウンセラークライアント 279
17.4 マルチクライアント 281
17.5 URL を使ったデータの取得 285
17.5.1 URL クラス 285
17.5.2 URL からのデータ読み込み 285

索　引 **289**

第1章 はじめに

Java 言語によるプログラミングの具体的な勉強を始める前に，Java 言語について概観し，ライブラリのダウンロードなどの準備を行います。

1.1 インターネット環境でのプログラミングと Java

プログラミングという行為は，プログラムを作成するだけでは終わりません。自分で楽しむためだけ，自分で必要な問題を解くためだけならそれでも構いませんが，人にも使ってもらう場合には，プログラムを配布し，実行してもらい，バージョンアップをユーザに反映する，そういったことも考えなくてはなりません。多くの人に使ってもらうためには，できるだけ多くの**プラットフォーム**[1)]で動作可能なようにプログラムを作成することや，利用者がインストールする手間をできるだけ減らす仕組みを組み込むことが必要となります。他人に使ってもらうからには，分かりやすい，使いやすいユーザインターフェースを提供すること[2)]，バグ[3)]をできるだけなくすことも重要です。また，プログラムを動作させる場所も，スマートフォンを始めとするコンピュータ以外のものが増えてきており，そういった新しいデバイスで動作させることも考えられます。

Java は，こういったことをコンセプトに入れて設計されたプログラミング言語です。言語というよりも，ネットワークを想定し，OS（オペレーティングシステム）などの提供する機能を取り込んだプログラムの動作環境と言ったほうがよいでしょう。Java 言語で作成したソフトウェアは，コンパイルし直さなくても同じオブジェクトプログラム（1.3 節）がプラットフォームによらずにどこでも動作します[4)]。よって，インターネットで配布することに適しています。また，**Java Web Start** という，Web を通じてダウンロード/実行する仕組みもあり，これを用いると，ユーザはつねに最新のバージョンにアクセスできるし，インストールの手間もかかりません[5)]。何より，Java 言語は強く型づけされたコンパイル言語なので，コンパイル時にバグを発見しやすくプログラムを高速に実行できますし，C や C++ 言語に比べてバグが入りにくい言語設計になっているので，作成されたソフトウェアの信頼性も高いです。

Java では，**サーブレット**，あるいは **JSP**(Java Server Pages) といった，

[1)] コンピュータのハードウェア（すなわちマシンそのもの），および，OS やウィンドウシステムなどの基本ソフトウェアといったプログラムが動作するための環境のこと。

[2)] 分かりやすいマニュアルも必要ですが，マニュアルを見なくても動かせるくらいの分かりやすいユーザインターフェースのほうが重要でしょう。

[3)] プログラムの間違いのこと。

[4)] Java を開発したサンマイクロシステムズでは，このことを，“Write Once, Run Anywhere” というスローガンで表しています。

[5)] Java 言語で **アプレット**と呼ばれる種類のプログラムを作成し，それを Web ページに組み込んでおけば，ユーザがそのページを訪問するだけでそのプログラムが実行されることが Java の売りでした。しかし，ブラウザのプラグインのセキュリティ上の問題などからあまり使われなくなり，次のバージョンから非推奨の機能になる予定です。

Web のサーバ側で動く仕組みも用意されています。また，携帯電話，情報家電などの組み込みシステムにも Java が使われています。さらに，Android のスマートフォンで動くアプリの開発言語として，Java は広く使われています。

1.2　大規模プログラミングとオブジェクト指向

個人的に作成するプログラムから商用プログラムや企業内業務システムなどの本格的なプログラムに目を向けると，それらは大規模化し，複雑化する一方です。そのようなプログラムは，複数の人数で長い時間をかけて作成され，一度作られたプログラムは長期間保守されて使われ続けていくことが多いです[6)]。

[6)] 巨大なシステムになれば，異なるプラットフォームを組み合わせたネットワーク環境で動かす必然性も高くなります。そこで，一つのオブジェクトプログラムをどのプラットフォームでも動かすことができる Java の仕組みが生かされます。また，プラットフォームは更新されていくものですが，Java さえ搭載されれば，それまで使っていたソフトウェアが動くことも魅力です。

そのときには，プログラムが高速に動くことはもちろん重要ですが，多人数で分担してプログラムを開発しやすいことも重要となってきます。また，プログラムに対する要求の変化に伴い，プログラムは変更を加えていくことが求められますが，全体に思わぬ影響を与えることがないように，部分ごとに変更を加えられるようプログラムが構成されていることが重要となります。そのためには，複雑で巨大なシステムを，**モジュール**と呼ばれる独立した部分に分けることが必要となります。**オブジェクト指向**は，プログラムをクラスの集まりと考えることにより，このような大規模ソフトウェアのモジュール化を可能にするために考えられた機構です。そして，オブジェクト指向の考え方に基づくシステム記述言語として Java 言語は設計されています。

[7)] 他のプログラムに特定の機能を提供するために作られたプログラム部品群を**ライブラリ**と言います。オブジェクト指向言語のライブラリはクラスの集まりなので，クラスライブラリと呼ばれます。

オブジェクト指向のメリットを享受するのは，大規模なソフトウェアだけではありません。Java には，グラフィカル・ユーザインターフェースの構築 (Swing, JavaFX)，データベースへのアクセス (JDBC) など，様々な機能を提供する**クラスライブラリ**[7)] が標準で装備されています。それだけではなく，ファイルへのアクセスやネットワーク通信といった，OS が提供する機能もすべて標準クラスライブラリで提供されています。この巨大なライブラリはそれ自体が大規模ソフトウェアであり，オブジェクト指向の概念を利用して作られています。

1.3　仮想マシン

この節の内容は，次章からの演習に直接必要ではないので，最初は読み飛ばしても構いません。

Java 言語は，プラットフォームに依存しないで動作することと，高速に実行できることを両立させています。次章からの具体的なプログラミングに関する説明に先だって，そのような特徴を実現する Java 言語の仕組みについて説明します。

通常，Java などのプログラミング言語のプログラムは，人間の手によって，エディタなどを用いてテキストファイルとして書かれます。このプログラムのことを，**ソースプログラム**と言います。一方，コンピュータは，マシン語プログラムと呼ばれる，0 と 1 の列からなる命令の列しか実行できません。よって，ソースプログラムとして書かれた内容をコンピュータに実行させるためには何らかの仕組みが必要です。その代表的な方法にインタープリタとコンパイラがあります。

インタープリタは，ソースプログラムを 1 行ずつ読み込んでその通りの動きをするソフトウェアです。コンピュータ上でインタープリタというプログラムを起動し，それにソースプログラムを読み込ませることによって，ソースプログラムをそのまま実行することができます。インタープリタでは，エディタで書いたプログラムがすぐに実行できるため，プログラムの開発が容易です。また，一つのプログラムが機種に依存せずどこでも動作可能であるという利点もあります。その反面，プログラムの実行が遅いという欠点があります[8)]。

コンパイラは，ソースプログラムを，それと同じ内容の処理を行うマシン語プログラムに変換するソフトウェアです。コンパイラの出力は，**オブジェクトプログラム**と言い，ソースプログラムからオブジェクトプログラムへの変換のことを**コンパイル**と言います。コンパイラによるプログラムの実行は，まず，コンパイラを起動してソースプログラムをオブジェクトプログラムに変換し，それからそれを実行するという二つのステップを踏むことになります。マシン語のオブジェクトプログラムは高速に実行できますが，機種に依存するため，一つのプラットフォームでしか実行できないのが普通です。

Java 言語は，両者を組み合わせた**仮想マシン**という方式を採用することにより，機種に依存せず，かつ，高速なプログラムの実行を実現しています。Java 言語のソースプログラムは，まずコンパイラにより処理されます。しかし，このコンパイラは，個々のコンピュータで動くマシン語プログラムを出力するのではありません。その代わりに，**Java バイトコード**と呼ばれる，中間的な言語のプログラムをオブジェクトプログラムとして出力します。このオブジェクトプログラムは，クラスと呼ばれる Java 言語のプログラムの単位ごとに別々のファイルに作られるので，**クラスファイル**と呼ばれています。クラスファイルは機種に依存しないので，一つのオブジェクトプログラムがどのプラットフォームでも実行できます。この概念を図 1.1 に示しました[9)10)]。

クラスファイルを実行するには，利用者のコンピュータに，Java バイトコードを実行する仕組みが必要です。そのような仕組みとしては，バイトコードのインタープリタが考えられます。バイトコードは人間が書くプログラムとは異なりマシン語的な構造をしているので，そのインタープリタは比較的高速に動作できます。Java では，それをさらに高速に動かすために，**JIT**

[8)] プログラムの利用者がソースコードを読めてしまうことも，場合によっては問題となるでしょう。

[9)] この図では，あたかも同じプログラムが様々な機器の上で動作しているように描いていますが，実際には，ライブラリや仮想マシンが違うので，パソコンとスマートフォンで同じプログラムが動作するわけではありません。

[10)] Java 言語以外にも，Scala 言語など，Java バイトコードにコンパイルして実行する言語が存在します。Java 言語とそのような言語を混在させてプログラムを組み，動作させることもできます。

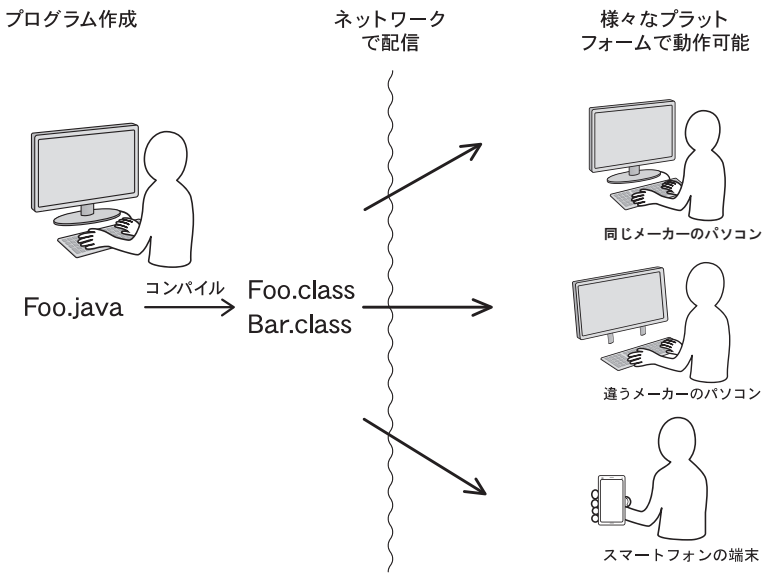


図 1.1　Java のプログラム開発/配布/実行の流れ

(Just In Time Compiler) という，プログラムの実行時にバイトコードを実行中のマシンのマシン語に変換（コンパイル）し，作成されたマシン語を実行する技術[11]を組み込んだ，**JVM**(Java Virtual Machine) と呼ばれるソフトウェアを用いています[12]。

プログラムを動かすときに必要なソフトウェアの集まりを実行環境と言いますが，Java の実行環境には，JVM に加えて，そのプラットフォーム用に作られた前述の標準クラスライブラリも必要です。C 言語などでのプログラム開発では，利用する OS やウィンドウシステム，データベースなどの API[13]を勉強し，それに従ってプログラムを書く必要がありました。当然，そのようにして書かれたプログラムは，異なるプラットフォーム上では動かすことができません。それに対し，Java のプログラムは，プラットフォームの機能を直接呼び出すのではなく標準クラスライブラリを呼び出しているだけなので，同じオブジェクトプログラムが OS などの違いに関わらず動作可能となります。

このように，Java では，仮想マシン方式を採用していることと標準クラスライブラリが存在することにより，同一のプログラムがどのプラットフォーム（ハードウェア＋基本ソフトウェア）でも動作することを実現しています。Java では，この仮想マシンと標準クラスライブラリという実行環境が一つのプラットフォームをなしていると考えて，この組のことを**Java プラットフォーム**と呼んでいます[14]。そして，Java プラットフォームでは，標準クラスライブラリのことを，**Java API** (Java Application Programming Interface) と呼んでいます。Java プラットフォームは，ライブラリの違いなどにより三種

[11] バイトコードの中で，パフォーマンス上重要な部分を捜してそこのみをコンパイルする，Hotspot 技術が使われています。

[12] Android スマートフォン上で動かす時には，Dalvik や ART といった，JVM とは異なる仕組みで実行されます。

[13] アプリケーション・プログラムから OS やライブラリなどの機能を呼び出すインターフェースのことを **API** (Application Programming Interface) と言います。

[14] コンパイラなどのプログラム開発環境も含めて Java プラットフォームと呼ぶこともあります。

類あります。本書で扱うのは，**Java SE** (Java Platform Standard Edition) [15] と呼ばれるものです。

このように，Java でプログラムを作成するには，Java 言語そのものについてオブジェクト指向の概念を中心に理解していることと，標準クラスライブラリの使い方，すなわち，Java API について理解していることが必要です。本書では，前半（1〜12 章）で前者，後半（13〜17 章）で後者を学びます。

[15] Java SE では，通常のプログラミングのために一般的な機能がライブラリで提供されています。これ以外の Java プラットフォームとして，より高度なライブラリを備えた企業サーバー向けの **Java EE** (Enterprise Edition)，組み込み向けの **Java ME** (Micro Edition) があります。

1.4 さらなる進化

Java 言語は，1995 年にサンマイクロシステムズにより開発されました。そして，2010 年にサンマイクロシステムズがオラクルに買収され，それ以降は，オラクルが管理，改良を続けています。Java 言語は，何度かバージョンアップがなされてきました。その中でも，1998 年 12 月に発表された J2SE 1.2(Java2 Platform Standard Edition 1.2)，2004 年 に発表された J2SE 5.0 (Java2 Platform Standard Edition 5.0) は大きな改訂でした[16]。そして，2014 年に発表された Java SE 8 は，ラムダ式や JavaFX が導入され，これまで以上に大きな改訂になっています。

[16] 本書の初版は J2SE 1.2，第 2 版は J2SE 5.0 に基づいて書かれていました。

ラムダ式は主に関数型言語で使われてきた概念で，これを用いると，手続きをメソッドの引数として渡すことが可能となり，今まで手続き的にしか書けなかった処理を，より宣言的に記述することができます。Java SE 8 は，この関数型言語の概念を，オブジェクト指向の枠組みは崩さずに言語に取り込んでいます。また，Java 言語は当初から並行・並列処理の機能を備えていましたが[17]，ラムダ式とストリームを用いた並列実行の記述も可能となりました。

[17] concurrent パッケージが J2SE5.0 で導入され，並行処理が強化されました。本書第 12 章では，この拡張された機能を中心に説明します。

JavaFX はグラフィカルユーザーインターフェースを作るための新しいライブラリです。タッチ入力などの新しい入力デバイスに対応しており，高機能な GUI をもつプログラムを作成できます。また，イベント処理をラムダ式で書けるなど，シンプルな記述が可能になっています。

本書は Java SE 8 に基づいて書かれており，本書に書かれたプログラムは，Java SE 8 以降のバージョンの Java でそのまま動作します[18]。

[18] Java は後方互換性を大切にしているので，今後バージョンアップが行われても，それまでのプログラムが動かなくなる可能性は少ないと思います。

1.5 準備 — ソフトウェアなどの入手 —

本書は，通読するだけで Java 言語の基本的な考え方やプログラミングの方法が身につくように書かれています。しかし，できるだけ Java 言語の処理系を手元に用意してサンプルプログラムを実行し，演習問題を解きながら読み進めてください。そのために，以下のものを用意してください。

まず，Java SE の処理系です。オラクルが配布している **JDK** (Java Development Kit)[19][20]が無料で入手可能です。また，その API のドキュメントをブラウザで閲覧できるようにしてください[21]。

本書は，JDK を利用し，Linux などのシェル，Macintosh のターミナル，あるいは，Windows のコマンドプロンプト[22]などのコマンドラインのインターフェースと，Eclipse などの統合開発環境の，どちらでも Java を学習できるように書かれています。コマンドラインを利用する時には，ソースプログラムを作成するのにテキストエディタも必要です[23]。統合開発環境はプログラムの作成を支援する様々な機能を含んでおり，本格的なプログラムの開発には必要不可欠です。Eclipse も無料で入手可能です。

最後に，本書を学習するために作られたプログラムを，まえがきに述べた本書のサポートページから入手してください。本書は，このプログラムが読者のコンピュータに置かれていると仮定して書かれています。例題はそこに含まれるライブラリを用いて動作するように作られていますし，演習問題も，そこに置かれているサンプルプログラムを書き換えたり，そこに新たにファイルを作成するように作られています。サポートページからダウンロードするファイルは，コマンドライン版と Eclipse 版が用意されています。コマンドライン版を展開すると，chap02，chap03，... といった各章に対応するフォルダ[24]があります。これらは，必要なライブラリなども含めた，独立したプログラムとなっています。Eclipse 版では，全体が一つのプロジェクトになっており，その中に各章に対応するパッケージがあります。また，練習問題の解答もサポートページに置いてあります。同様にダウンロードしてください。

下の表は，コマンドラインで Java の実行に必要な最低限のコマンドプロンプト等のコマンドをまとめたものです。参考にしてください[25]。

Windows コマンドプロンプト	Linux, Mac (ターミナル)	説明
cd *d*	**cd** *d*	*d* に現在のディレクトリを移動（*d* はディレクトリ名）。
dir	**ls -l**	現在のディレクトリにあるファイル名とディレクトリ名を表示。
dir/w	**ls**	同上。コンパクトに表示。

[19] JDK には，実行環境とコンパイラなどの開発環境が含まれます。また，**JRE** (Java Runtime Environment) というパッケージも配布されていますが，こちらは実行環境だけでありコンパイラを含みません。

[20] 現時点（2017 年 6 月）では，Java SE の日本語ドキュメントのページ（ORACLE Java ドキュメント Java Platform, Standard Edition (Java SE) 8），http://docs.oracle.com/javase/jp/8/ に，ダウンロードおよびインストールする方法がまとめられています。

[21] これは，上記のページの『Java SE API ドキュメント』および『JavaFX API ドキュメント』というリンクから閲覧できます。

[22] 通常，プログラムメニューの『アクセサリ』の中にあります。

[23] 高機能なエディタは Java の文法に合わせた括弧の照合や自動インデントの機能が利用できるので便利です。

[24] ファイルを格納する場所のことを，UNIX ではディレクトリ，Windows や Mac ではフォルダと言います。本書では，主にフォルダを用います。

[25] コマンドプロンプトなどには，上向き矢印で前に打ち込んだコマンドを表示するなどの入力の手間を省く便利な機能があることが多いです。

第2章 オブジェクトの生成とメソッド呼び出し

本章では，オブジェクトと呼ばれる “もの” を作成すること，および，オブジェクトに対してメソッドの実行を依頼することという，オブジェクト指向プログラミングの基本について学びます。

2.1 オブジェクトとクラス

オブジェクト指向は，**オブジェクト**と呼ばれる “もの” を作成[1)]することと，メソッド呼び出しにより，オブジェクトに手続きの実行を依頼することを基本としたプログラミングの方法です。本章では，オブジェクトとはどういう “もの” なのか，タートルグラフィックスの例題を用いて説明します。

タートルグラフィックスは，画面上に置かれたタートル（亀）に対して「前に 100 進め」とか「右に 60 度向きを変えろ」といった命令を送ることによりタートルを動かして，その軌跡として，プログラムで絵を描く方法です。ウィンドウ上でタートルの位置は，左上を原点として右向きに x 軸，下向きに y 軸をとった座標系を用いて実数[2)]で指定します[3)]。また，タートルの向きは，上向きを 0 度として右回りに度数を実数で表します[4)]。図 2.1 は，400×400 の大きさのウィンドウを作成し，(200, 200) の座標に 0 度の方向を向いたタートルを配置し，そのタートルに対して「前に 100 進め」と「右に 144 度回れ」という命令を 5 回繰り返すことにより描いた絵です。

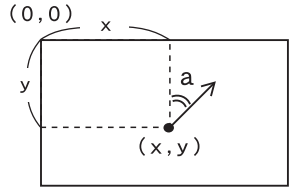
オブジェクトがどういう種類のものか定義したものを**クラス**と言います[5)]。本書のタートルグラフィックスのライブラリは，Turtle という画面上に現れるタートルのクラスと，TurtleFrame というタートルの動きまわるウィンドウのクラスからなっています[6)]。そして，オブジェクトはクラスを指定して作成します。また，あるクラスに属しているオブジェクトのことを，そのクラスの**インスタンス**と言います。よって，この絵は，TurtleFrame クラスのインスタンスを生成し，Turtle クラスのインスタンスを生成し，そのタートルに「前に 100 進め」と「右に 144 度回れ」という処理を繰り返し実行させること，すなわち，そのタートルのオブジェクトに対してそれらに対応するメソッドを繰り返し呼び出すことにより描くことができます。

1) オブジェクトを作ることは，「作成する」と言うこともあるし「生成する」と言うこともあります。

2) プログラム中では，実数の代わりに倍精度浮動小数点数という実数の近似値を用います（7.1 節）。

3) コンピュータの画面は，ピクセルと呼ばれる四角い画素が縦と横に並んでできています。画面上の距離は，1 ピクセルの長さを 1 として指定することにします。

4)



5) このクラスの説明は本質的ですが，単純化しすぎています。詳細は，第 4 章以降で説明します。

6) これ以外に，Point クラスと TurtleGraphics クラスがあります。TurtleGraphics クラスはプログラムから直接扱いません。これらのソースファイルは，コマンドライン版では chap02 フォルダの中の tg フォルダに，Eclipse 版では tg パッケージにあります。

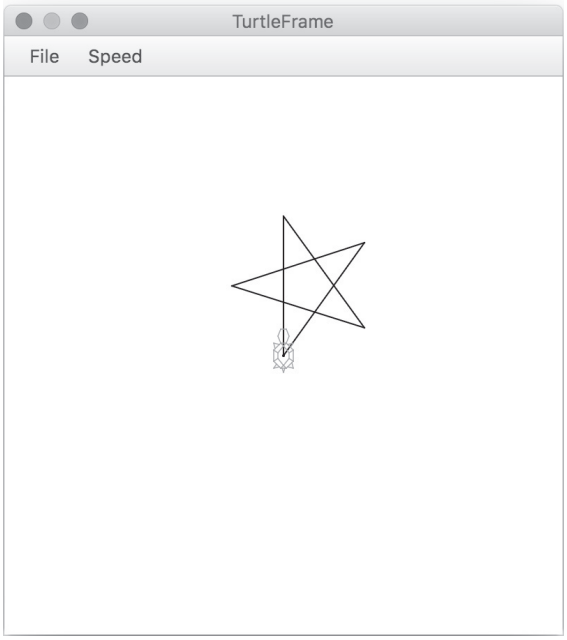


図 2.1　タートルグラフィックで描いた星型

2.2　最初の例題 ― オブジェクトの生成とメソッド呼び出し ―

リスト 2.1 のプログラムの実行を追いながら，オブジェクトの生成とメソッドの呼び出し方法について学びましょう。このプログラムは，T21.java というファイルに納められています[7)]。

(1) コメント

1 行目[8)]は，**コメント**です。プログラムの中で，/* と */ で囲まれた部分[9)]や，// から行末までは，コメントと呼ばれるプログラムを読む人のために書かれた説明文で，プログラムの実行に影響を与えません。また，4〜16 行目は，この範囲が一つの塊りだということが見て分かるように，行の先頭に空白文字を入れて**インデント**（字下げ）を行っています[10)]。Java では，プログラム中の改行，空白は区切りとしての意味だけをもち，どれだけ入っていても意味は変わりません[11)]。インデントやコメントをどう書くかは自由ですが，プログラムは動けばよいというものではありません。デバッグ[12)]や保守[13)]のことも考えて，読みやすくプログラムを書くことを心掛けてください。

(2) クラス名の指定

2 行目については，2.6 節で説明します。3 行目は，行末の { からそれに

[7)] Eclipse 版では，各ファイルの最初に package chap02; といった宣言が書かれており，chap02 などのパッケージに属することになります（5.4 節）。

[8)] 本書では，ページ数を抑えるために，プログラムの対応する部分を文章中に再び掲載することを避けています。面倒ですが，何行目といわれる都度にリストを参照してください。

[9)] この行のように，/** で始まるコメントは，5.1.4 項で述べる，特別な意味を持ちます。

[10)] 5〜15 行目は，さらにインデントを行っています。

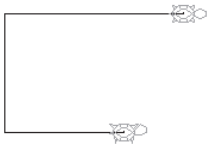
[11)] たとえば，3 行目は，
```
public class
T21
```
{
と 3 行に分けられていても同じ意味です。

[12)] バグ（プログラムの間違い）を修正すること。

[13)] 2.5 節の傍注 56 参照。

リスト 2.1 オブジェクトの作成とメソッド呼び出し（T21.java）

```
1  /** 最初のプログラムの例 */
2  import tg.*;
3  public class T21 {
4        public  static void main(String[] args){
5          TurtleFrame f;              // 変数 f の型宣言
6          f = new TurtleFrame();      // TurtleFrame を作成し f に代入
7          Turtle m = new Turtle();    // Turtle を作成し，m の初期値として代入
8          Turtle m1 = new Turtle();   // もう一つ作成し，m1 の初期値として代入
9          f.add(m);                   // f に m を追加
10         f.add(m1);                  // f に m1 を追加
11         m.fd(100.0);                // m よ前に 100 進め
12         m.rt(90.0);                 // m よ右に 90 度回れ
13         m.fd(150.0);                // m よ前に 150 進め
14         m1.rt(90.0);                // m1 よ右に 90 度回れ
15         m1.fd(100.0);               // m1 よ前に 100 進め
16     }
17 }
```



対応する 17 行目の } までが T21 という名前[14] のクラスの定義だと宣言しています[15]。4 行目は，プログラムが起動されたときに，行末の { と 16 行目の } で挟まれた部分が順に実行されるように指示しています。 これらの詳細は後に回して説明を先にすすめます。

T21 という名前はプログラムを作成している人が自由に選んだ文字列です。それに対し，public や class という文字列は，Java 言語の一部をなしています。このような文字列のことを**キーワード**と言います[16]。

[14] クラス名は大文字から始めるのが慣例です。

[15] これはオブジェクトの定義ではないので，クラスというのは不自然に思われるかもしれません。これについては 4.3 節で説明します。

[16] 本書では，読みやすさのために，プログラムリスト中のキーワードはボールド体で示しています。多少見にくいですが，4 行目の [] は [ ] の 2 文字です。

(3) オブジェクトの生成

6 行目で TurtleFrame オブジェクトを生成しています。

new クラス名 ()

は[17] **インスタンス生成式**と呼ばれ，これを実行するとクラス名 クラスのインスタンスが一つ生成されます。TurtleFrame はタートルが動くウィンドウという “もの” を意味し，作成されるとすぐにウィンドウが表示されます。

[17] この行で，new や () はプログラムの中にそのまま書かれるものです。それに対して クラス名 は，特定の一つの文字列を指しているのではなく，クラス名となる文字列がここにくることを示してます。そのようなものを，文字列の上を動く変数という意味で**メタ変数**と言います。本書では，メタ変数は四角で囲んで示すことにします。

(4) 変数への代入

作成したオブジェクトを後で使うためには，プログラムから参照できるように，どこかに記録しておく必要があります。オブジェクトや整数値などの値[18] を記録するための場所のことを**変数**[19]と言います。変数に値を記録することを，変数に**代入**すると言います。変数に値を代入するには，

変数 = 式 ;

という具合に，= の左辺に変数名を，右辺に代入する値を表す**式**[20]を書い

[18] 値については 2.5 節 (1), 2.8 節を参照。

て，最後にセミコロン (;) を付けます。new TurtleFrame() のようなインスタンス生成式は，作成されたオブジェクトを意味する式です。以上をまとめると，6 行目は，TurtleFrame クラスのインスタンスを一つ作成して f という変数に記録することを意味しています。最後のセミコロン (;) は，ここまででコンピュータに対する一つの命令になっていることを示しています。このような命令の単位を**文**と言います。6〜15 行目のそれぞれの行は文です。

(5) 変数の型宣言

変数[21]は，プログラム中で最初に使われる前に，そこに記録される値の種類を宣言しておく必要があります。値の種類のことを**型**といい，この宣言を変数の**型宣言**と言います。型宣言は，

型名 変数 ;

という形で行われます。型についてはこれから順に学んでいきます。ここでは，クラスはオブジェクトの型になることだけ説明しておきます。5 行目は，f に格納できるのは TurtleFrame 型のオブジェクトだと宣言しています。

(6) 変数の初期化

変数の型宣言のときに，

型名 変数 = 初期値を表す式 ;

という形で，その変数の初期値を同時に代入することもできます[22]。よって，5, 6 行目は，次のように 1 行で書くこともできます。

```
TurtleFrame f = new TurtleFrame();
```

7 行目は初期値つきの変数宣言で，Turtle クラスのインスタンスを作成して変数 m に代入しています[23]。8 行目も，タートルをもう一つ作って m1 という変数に代入しています。同じ型の変数が複数あるときは，

```
Turtle m, m1;
```

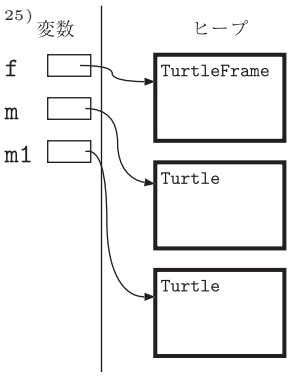
とコンマで区切って並べることにより，一度に型宣言を行うこともできるし，

```
Turtle m = ..., m1 = ...;
```

と，一度に複数の初期値つきの型宣言を行うこともできます。

(7) オブジェクトの変数への代入

変数にオブジェクトを代入するというのは，変数という記憶場所にオブジェクトそのものを格納するのではありません。オブジェクトは，すべて**ヒープ**と呼ばれるコンピュータのメモリ上の領域に置かれています。ヒープのどこにオブジェクトがあるかという情報のことをオブジェクトの**参照**と呼びます。変数に記録されるのはオブジェクトの参照です[24][25]。



19) 変数名には，数字以外の文字から始まる文字列を用いることができます。ただし，キーワードは使えません。また，定数以外の変数名は，小文字から始めるのが慣例とされています。

20) 式は，オブジェクトや数といった値を意味する表現のことです（7.2.3 項）。

21) 後に述べるように，変数にはいくつかの種類があります。この変数は，**ローカル変数**と呼ばれるものです。

22) 変数を**初期化**すると言います。

23) Turtle クラスのインスタンスは画面上を動き回るタートルを意味していますが，9 行目で TurtleFrame に貼り付けられるまで画面に表示されません。

24) 本来ならオブジェクトへの参照を変数に代入するというべきですが，本書ではオブジェクトを変数に代入するということにします。

25)