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 言語の特徴について概観しよう。

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らない。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わないが，もし人にも使ってもらいたいのなら，プログラムを宣伝し，配布し，実行してもらい，その結果のフィードバック（実行結果，バグ情報，感想など）を集め，バージョンアップを行ったならそれをユーザに反映する，そういったことも考えなくてはならない。そのためには，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る。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，それに，たとえ自分のプログラムにバグがあったとしても，ユーザに多大な迷惑をかけないような仕組みの上で動作させることも重要であろう。

また，これだけネットワークが広まった現在，プログラミングの目的も大きく様変わりしているのではないか。昔は，必要なソフトウェアは自分で作らなくてはならず，データ処理などのために一々プログラムを組んでいた。しかし，現在では，ツール類を自作するのは稀だし，簡単なデータ処理なら表計算などの出来合いのソフトウェアのマクロ[4)] を書く程度で済んでしまうことも多い。その一方で，ネットワークを用いてプログラム作品を人にも使ってほしい，プログラミングで得た知見や楽しみを人と共有したいといった気持ちは，多くの人がもっているのではないかと思う。特に，すぐれたグラフィックスとインターラクティブな操作性があれば，分かりやすい形でプログラムの実行結果を見ること，いや，見せることができるし，ネットワーク通信があれば，ユーザの実行結果を集めたり，双方向のやりとりも可能になる。一方で，プログラムを動作させる場所も，携帯電話を始めとするコンピュータ以外のものが増えてきている。それに合った新しいプログラミングの利用も考えられる。

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のこと。プラットフォームはプログラムが動作するための環境である。

2) 分かりやすいマニュアルも必要だが，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だ。

3) プログラムの間違いのこと。

4) ワープロや表計算ソフトで簡単な手順をプログラムとして登録する機能。

Java は，こういったことをすべてコンセプトに入れたプログラミング言語である。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。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する[5)6)]。よって，インターネットで配布することに適している。このことが，Java の最大の特徴である。

特に，主要な Web ブラウザには Java の実行環境が組み込まれており[7)]，**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プログラムが実行される。また，**Java Web Start** という，通常のアプリケーションを Web を通じてダウンロード/実行する仕組みもある。これらを用いると，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らない。また，プログラムからアクセスできるリソースが制限されるので，安心してプログラムを実行することができる。それに加えて，Java 言語は 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。

Java では，インターネット環境でアプレットのようにブラウザ側で動くだけではなく，**サーブレット**，あるいは **JSP**(Java Server Pages) といった，Web のサーバ側で動く仕組みも用意されている。また，携帯電話，PDA，情報家電などにも Java の実行環境が載せられている[8)]。特に，携帯電話にプログラムをダウンロードして実行する環境としては，Java は標準の地位を確立しているといってよいであろう[9)]。

5) Java の開発元の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る。

6) もちろん，プラットフォームに Java の実行環境がインストールされている必要がある。

7) あるいは，組み込むことが可能である。

8) もともと，このような組み込みシステムを制御するための言語として Java は開発された。

9) もちろん，コンピュータ上で普通に起動して利用するプログラムを作成することが，Java 言語の主要な用途である。

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[10)]に目を向けると，それらは大規模化し，複雑化する一方である。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[11)]。

そのときには，プログラムが高速に動くことよりもむしろ，多人数で分担してプログラムを開発しやすいことが重要となる。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るが，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る。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る。**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

10) Java は巨大なシステムで実用に使われている。たとえば，2004 年の NASA の火星探査車の制御にも Java が用いられている。

11) 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る。そこで，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動かせる Java の仕組みが生かされる。また，プラットフォームは更新されていくものだが，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ある。

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ある。そして，オブジェクト指向の考え方を徹底化し，実用的な大規模ソフトウェアの開発にも使えるようにしたのが Java 言語である。

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ない。Java には，グラフィカルなユーザインターフェースの構築 (AWT, Swing)，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**[12] が標準で装備されている。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る。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駆使して作られている[13]。このライブラリを利用したプログラムを作成するためにも，オブジェクト指向についてしっかり理解しておくことが必要である。

[12] 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**と言う。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る。

[13] Java で作成されたプログラムは 標準クラスライブラリとともに動作する。よって，小さなプログラムでも，ライブラリも含めた巨大なソフトウェアの一部を作成しているという見方もできよう。

1.3 仮想マシン

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，少しだけ，Java 言語の実行の仕組みについて説明しておこう。

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，Java に慣れてから読み直してもよい。

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る。このプログラムのことを，**ソースプログラム**と言う。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ない。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ある。その代表的な方法にインタープリタとコンパイラがある。

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ある。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る。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ある。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る。その反面，プログラムの実行が遅いという欠点がある[14]。

[14] 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ろう。

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ある。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**と言う。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る。オブジェクトプログラムは高速に実行可能であるが，機種に依存するため，一

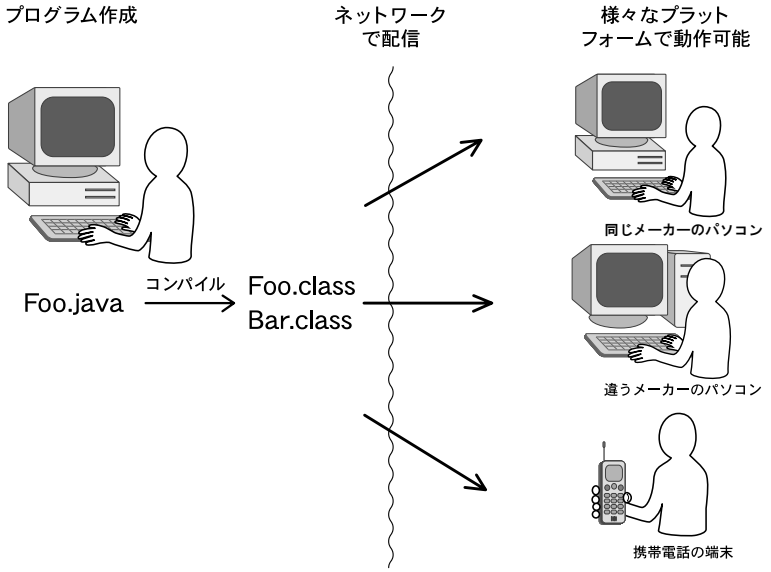


図 1.1　Java のプログラム開発/配布/実行の流れ

つのプラットフォームでしか実行できないのが普通である。

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，十分に高速なプログラムの実行を実現している。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る。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ない。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する。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**と呼ばれている。クラスファイルは機種に依存しないので，これをネットワークなどを用いて利用者のコンピュータに配布すればよい。この概念を図 1.1 に示した[15)]。

[15)] 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るが，実際には，組み込みシステムでは使えるライブラリが違うので，パソコンと同じプログラムが動作するわけではない。

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ソフトウェアが必要である。それは，**JVM**(Java Virtual Machine) と呼ばれている。JVM としては，たとえば，バイトコードのインタープリタが考えられる。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のよう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る。また，より高速に動かすために，**JIT** (Just in time compiler) という，インタープリタと同様にバイトコードを読み込んで，その中でパフォーマンス上重要な部分については，実行中のマシンのマシン語に変換（コンパイル）して作成されたマシン語を実行する技術も存在する。JIT により実現された JVM は，多くのプラットフォームに対して提供されている。また，Web のブラウザに組み込まれて動く JVM も配布されており，アプレットと

いう形式のプログラムを作成すれば，ブラウザ上で実行できる。

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うが，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ある。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 [16)] を勉強し，それに従ってプログラムを書く必要があった。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なかった。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る[17)]。

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る。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**と呼んでいる[18)]。そして，Java プラットフォームでは，標準クラスライブラリを呼び出す手順のことを，**Java プラットフォーム API**, あるいは，**Java API** (Java Application Programming Interface) と呼んでいる。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類ある。本書で扱うのは，**Java SE** (Java Platform Standard Edition) [19)] と呼ばれるものである。

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ある。本書では，前半（1〜10 章）と 17 章の一部で前者，後半（11〜17 章）で後者を学ぶ。

16) 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) と言う。

17) どこででも動くようにするために，どのプラットフォームでも提供されている基本的な機能しか 標準クラスライブラリでは提供されない。よって，ある特定のシステムでしか提供されていないような機能は，通常，Java から利用できない。しかし，どこででも動くことは，それ以上のメリットとなりえる。

18) 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る。

19) 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る。これ以外の Java Platform 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)，携帯電話などへの組み込み向けの **Java ME** (Micro Edition) がある。

1.4 準備 — ソフトウェアなどの入手 —

Java プラットフォームは，1995 年の発表以来，何度かバージョンアップがなされてきた。その中でも，1998 年 12 月に発表された J2SE 1.2(Java2 Platform Standard Edition 1.2) は大きな改訂であり，本書の初版は，それに基づいて書かれていた。その後，J2SE 1.3, J2SE 1.4 での改定は比較的小さなものであったが，2004 年 に発表された J2SE 5.0 (Java2 Platform Standard Edition 5.0) では，言語仕様の変更を伴う大規模な改訂が行われた。現在（2007 年 9 月）の最新バージョンは，Java SE 6 (Java Platform, Standard Edition 6) である[20)]。本書では，混乱を避けるため，Java SE 6 に合わせて J2SE 5.0 のことを Java SE 5 と呼ぶことにする。

20) Java Platform は，バージョン 1.2 から 5.0 までは Java2 Platform と呼ばれており，Java SE の代わりに J2SE という略称が用いられていた。また, 5.0 までのバージョン名は，小数点を含んだ番号であった。さらに, 5.0 より以前は，“1.” が前に付けられていた。

この第 2 版は，Java SE 6 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 5 以降のバージョンの Java でそのまま動作する。また，Java SE 5 で新たに付け加えられた機能を説明するところはそのように明記してあるので，Java 2 以降の処理系なら学習に用いることができる。

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る。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ほしい。そのために，以下の三つのものを用意していただきたい[21)]。

まず，Java SE の処理系である。

```
http://java.sun.com
```

から，リンクをたどれば，Sun Microsystems,Inc. が配布している **JDK**(Java Development Kit)[22)] という Java SE の開発環境が入手可能である。

次に，Java プラットフォームのクラスライブラリの仕様書である『Java Platform Standard Edition API 仕様』というドキュメントを，ブラウザで閲覧できるようにしてほしい[23)]。これは，上記と同じページから閲覧したり，また，ダウンロードすることもできる[24)]。

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ほしい[25)]。本書は，このプログラムが読者のコンピュータのディレクトリ[26)]に置かれていると仮定して書かれている。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るし，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指示して作られている。以下，java-everyone というディレクトリにダウンロードしてきたファイルが展開されて置かれているものとして話を進めることにする。

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[27)]などのコマンドを入力して実行するインターフェースを用いて java を実行するように書かれている。Java の実行に必要な最低限のコマンドプロンプトのコマンドを表にしたので，参考にしてほしい[28)29)]。

cd *d*　　*d* にディレクトリを移動（*d* はディレクトリ名）。
dir　　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の表示。
dir/w　同上。コンパクトに表示。

本書では扱わないが，Eclipse などの統合開発環境を用いると，プログラムの作成を支援する様々な機能が利用できる。本格的なプログラムの開発には，検討に値するであろう。

21) もちろん, それ以外に, 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ある。Windows 環境に付属しているメモ帳でも構わないが, Emacs, Meadow などの高機能エディタは 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ある。

22) JDK には，実行環境とコンパイラなどの開発環境が含まれる。また，Sun では，**JRE**(Java Runtime Environment) という実行環境だけのパッケージも配布している。

23) バージョンによって異なるが，第 3 章の図 3.1 のような画面である。

24) 現時点（2007 年 9 月）では，http://java.sun.com/javase/ja/6/docs/ja/ の『JDK 6 ドキュメント』というページの『API, 言語, 仮想マシンのドキュメント』という項目の中の，『Java Platform API 仕様』というリンクの先にある。

25) ~ はキーボードによっては「¯」と刻印されている。

26) 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 ではフォルダと言う。本書では，ディレクトリで統一する。

27) 通常，プログラムメニューの『アクセサリ』の中にある。

28) 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。

29) Linux や Macintosh のシェルでは, `dir/w` は `ls`, `dir` は `ls -l`, `dir/w` は `ls` コマンドが対応する。

1.5 準備 — 必要なクラスのコンパイル —

本書の前半（1〜10 章）は，タートルグラフィックスの例題を用いて Java 言語の学習を進める。java-everyone の下の turtle というディレクトリで行うので[30]，コマンドプロンプトなどを起動し，cd コマンドでそこに移動しよう[31][32]。

```
> cd java-everyone
> dir/W
....
[.]   [..]  [applet]  [collection]  [event]   [graphics]
[gui] [io]  [net]     README.txt    [turtle]
...
> cd turtle
```

このディレクトリには，“.java” という拡張子の付いたファイルが置かれている[33]。この拡張子は，Java 言語のソースプログラムを意味している。これらのソースプログラムのほとんどはこのテキストに載っている例題であるが，Turtle.java と TurtleFrame.java は例題ではなく，Turtle と TurtleFrame という，本書前半を通して利用する二つのクラスを定義したものである[34]。これらのクラスを利用するには，コンパイルを行い，Java 仮想マシンのバイトコードからなるクラスファイルに変換する必要がある。クラスファイルは，クラスごとに，“クラス名.class” という名前をしている。実際にコンパイルを行い，クラスファイルを作成しよう[35]。

コンパイルは **javac** というコマンドを利用して行う。ソースプログラムをコンパイルすると，そこに定義されている個々のクラスに対応してクラスファイルが作成される。OS のプロンプトの下で

```
> javac Turtle.java
```

を実行することにより Turtle.java がコンパイルされ，クラスファイル Turtle.class が作成される。また，

```
> javac TurtleFrame.java
```

を実行することにより TurtleFrame.java がコンパイルされ，クラスファイル TurtleFrame.class が作成される[36][37][38]。これらのファイルが作成されたことを確認した上で，次章からの学習を始めよう。

[30] 第 11 章以降は，章ごとにディレクトリが存在する。java-everyone の下の README.txt を参照。

[31] > は，OS の出す**プロンプト**（コンピュータが，次のコマンドを受け付ける準備ができたことを示す文字列）だとする。Windows のコマンドプロンプトなら，
`C:¥java-everyone>`
という具合に，現在のディレクトリ名がプロンプト内に表示される。

[32] コマンドと引数（この場合は cd と java-everyone など）の間は，空白を一つ以上あけること。シェルの場合は `dir/W` ではなく `ls` を用いる。`dir/W` では，ディレクトリ名は [ ] で囲んで表示される。

[33] dir などで確認してほしい。

[34] これらのソースプログラムを，本書を読み終わった後に眺めてほしい。

[35] 1.3 節でも述べたように，クラスファイルはマシンに依存しないので，最初からクラスファイルを配布してもよかったが，javac コマンドに慣れてもらう意味で，読者に行ってもらうようにしている。

[36] この他に，内部クラス TurtleFrame$Line.class および匿名クラス TurtleFrame$1.class も作成される（7.7 節参照）。これらも TurtleFrame.java で定義されたクラスだが，ユーザが直接利用するものではない。

[37] 実際には，Turtle.java のコンパイルを行えば，その中で利用されている TurtleFrame.class を作成するために，TurtleFrame.java もコンパイルされる。

[38] コンパイルに失敗する時には，java が正しくインストールされているか，確認されたい。

第2章 オブジェクトの生成とメソッド呼出し

本章では，オブジェクトと呼ばれる “もの” を作成することと，オブジェクトに対して処理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ぶ。

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” を生成[1] することと，メソッド呼出しにより，オブジェクトに処理の依頼を行うことを基本としたプログラミングの方法である。本章では，オブジェクトとはどのようなものか，タートルグラフィックスの例題を用いて説明していこう。

オブジェクトを生成するには，それがどんな “もの” なのか定義が必要である。そのような定義は，**クラス**[2] と呼ばれている。クラスは，システムに最初から用意されているものもあるし，自分で定義するものもある。1.5 節で述べたように，クラスの内容は，“クラス名.class” という名前のクラスファイルに格納されている。つまり，1.5 節で作成した TurtleFrame.class と Turtle.class は，それぞれ TurtleFrame，Turtle というタートルグラフィックスを実現している二つのクラスを定義したクラスファイルである。

[1] オブジェクトは「作成する」ではなくて「生成する」と言うのが一般的だ。本書もそれに従う。

[2] このクラスの説明は本質的ではあるが，単純化しすぎている。詳細は，第 3 章で説明する。

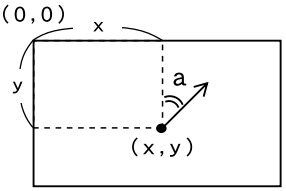
2.2 タートルグラフィックス

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 度回れ」といった命令を送ることによりタートルを動かして，その軌跡を画面に線として残すことにより，プログラムで絵を描く方法である。本書のタートルグラフィックスは，TurtleFrame というタートルが動きまわる画面のウィンドウのクラスと，Turtle というタートルのクラスを用いて実現されている。

ウィンドウ上でタートルの位置を指定するときには，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ることにする[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で表すことにする[4] [5]。図 2.1 は，400 × 400 の大きさのウィンドウを生成し，(200,200) 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

[3] コンピュータの画面は，ピクセルと呼ばれる点が縦と横に並んでできている。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する。

[4]



[5] このタートルグラフィックスでは，座標や角度は整数値のみを考えることにする。

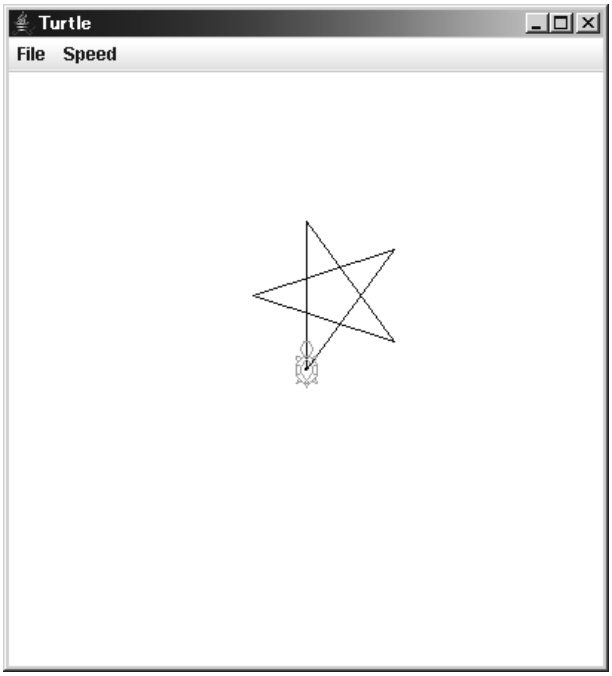


図 2.1　タートルグラフィックで描いた星型

“右に 144 度回れ” という命令を 5 回繰り返すことによって描いた絵である。

2.3　最初の例題 — オブジェクトの生成とメソッド呼出し —

リスト 2.1 のプログラムを見てほしい。これは，turtle ディレクトリの中の，T21.java というファイルに納められている。

(1) コメント

1 行目は，**コメント**である。プログラムの中で，/* と */ で囲まれた部分[6)]や，// の後ろ行末までは，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あり，プログラムの実行に影響を与えない。また，3〜15 行目は，行の先頭に空白文字を入れて，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，**インデント**（字下げ）を行っている[7)]。Java では，プログラム中の改行，空白は区切りであって，どれだけ入っていても意味は変わらない[8)]。インデントやコメントをどう書くかは自由だが，プログラムは動けばよいというだけではない。デバッグ[9)] や保守[10)] のことも考えて，読みやすいプログラムも心掛けてほしい。

[6)] このように，/** で始まるコメントは，7.1.3 項で述べるように，特別な意味をもつ。

[7)] 4〜14 行目は，さらにインデントを行っている。

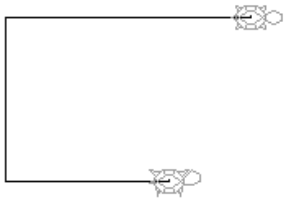
[8)] たとえば，2 行目は，
```
public   class
    T21
```
{
と 4 行に分けられていても同じ意味である。

[9)] バグを修正すること。

[10)] 2.6 節の傍注 46 参照。

リスト 2.1　T21.java

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



(2) クラス名の指定

2, 3 行目の詳細は，第 7 章で説明する。ここでは，2 行目で，行末の { からそれに対応する 16 行目の } までが T21 という名前[11] のクラスの定義だと宣言している[12] ことと，3 行目で，行末の { と 15 行目の } で挟まれた部分に main という名前を付けて，プログラムを起動されたときにこの部分が順に実行されるように指示していることだけを述べておく。T21 という文字列は，このプログラムを作成している人が自由に選んで付けた名前である。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る。このような文字列のことを，**キーワード**と言う[13]。

(3) オブジェクトの生成

4〜14 行目の，プログラムの内容に話を進めよう。5 行目はオブジェクトを生成している。

new クラス名()

は[14] **インスタンスの生成式**と呼ばれ，これを実行すると，クラス名 クラスのオブジェクトが 一つ生成される。あるクラスのオブジェクトのことを，そのクラスの**インスタンス**と呼ぶ。よって，ここで生成されたオブジェクトは，TurtleFrame クラスのインスタンスである。TurtleFrame のインスタンスは，タートルが動く画面ウィンドウという “もの” を意味しており，生成されると，すぐにウィンドウが表示される。

(4) 変数への代入

生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かなくてはならない。オブジェクトや整数値など

[11] クラス名は大文字から始めるのが慣例とされている。

[12] このプログラム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ない。クラス定義の中にはこの main のように，オブジェクト定義以外のプログラムも含めることができ，このクラスは，それだけからなっている（3.4 節）。

[13] 本書では，読みやすさを尊重して，プログラムリスト中の予約語はボールド体で示した。また，多少見にくいが，3 行目の [] は [ ] の 2 文字である。

[14] 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ある。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いる。そのようなものを，文字列の上を動く変数という意味で，メタ変数と言う。本書では，メタ変数は四角で囲んで示すことにする。

の値[15] を記録するための場所のことを**変数**[16]と言う。変数に値を記録することを，変数に**代入**すると言う。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**[17]を書いて，最後にセミコロン (;) を付ける。インスタンス生成式は，生成されたオブジェクトという値を意味する式である。以上をまとめると，5 行目は TurtleFrame クラスのインスタンス，すなわち，タートルを表示する画面を意味するオブジェクトを一つ生成して，それを f という変数に記録することを意味している。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る。このような命令の単位を**文**と言う。5〜14 行目のそれぞれの行は，文となっている。

[15] 値については 2.6 節 (1), 2.8 節。

[16] 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る。ただし，予約語はだめである。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る。

[17] 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ある（6.4 節）。

(5) 変数の型宣言

変数[18]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る。それを，変数の**型宣言**と言う。型宣言は，

型名 変数名;

という形で行われる。**型**については，8.1 節で詳しく扱う。ここではとりあえず，クラスなどの，値の種類を指定するものとだけ述べておく。TurtleFrame などのクラスに対応した型のことを**クラス型**と言う。

この例では，4 行目で 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る。

[18] 2.7, 3.3 節で述べるように，変数にはいろいろな種類がある。ここでいう変数は，正確には，**ローカル変数**と呼ばれるものである。

(6) 変数の初期化

変数の型宣言のときには，

型名 変数名 = 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る[19]。よって，4, 5 行目は，

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

と 1 行で書いてもよい。6 行目は初期値つきの変数宣言で，new Turtle() により Turtle クラスのインスタンスを生成して m という変数に代入している[20]。7 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る。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る。

[19] 変数を**初期化**すると言う。

[20] Turtle クラスのインスタンスはタートルという “もの” を意味しているが，画面には，8 行目で TurtleFrame に貼り付けられるまで表示されない。

(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ない。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る。そして，変数に記録されるのは，そのオブジェクトがヒープ上のどこにあるかという，場所の情報である[21)]。これを，オブジェクトの**参照**と呼ぶ[22)]。傍注[23)] の図を参照されたい。

(8) メソッド呼出し

8 行目からは，**インスタンスメソッド**の呼出しにより，オブジェクトに処理の依頼を行っている。8 行目は，f（に代入されていた TurtleFrame）に対して，add というインスタンスメソッドを，m（に代入されている，最初に作った Turtle オブジェクト）を引数として呼び出すことにより，f に対し，ウィンドウ上に m を載せるように依頼している。オブジェクトに対するインスタンスメソッドの呼出し[24)] の一般形は，次のようになる。

オブジェクト式.インスタンスメソッド名(引数式 1,...,引数式 n)

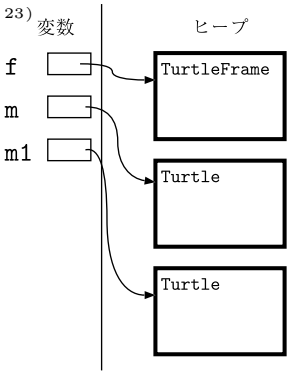
引数[25)] とは，メソッドの処理のために，一緒に渡される値のことである。引数が二つ以上あるときには，括弧の中にコンマ (,) で区切って並べる。インスタンスメソッド呼出しが行われると，呼び出されたオブジェクトは，そのメソッドに応じた処理を行う。よって，この 1 行によって，f は m をウィンドウ上に載せるという処理を行い，ウィンドウ上に亀の絵が現れる。9 行目も同じである。 m と m1 は，標準の初期値である，画面中央の (200,200) という座標と 0 度の角度という状態をもって生成されているので，そのように画面上に現われる。**メソッド**にはインスタンスメソッドと次章で扱うクラスメソッドがある。文脈から明らかなときには，インスタンスメソッドのことを単にメソッドと言う。

10 行目の fd は，引数で与えられた数だけ前に動くという，Turtle クラスのメソッドである。よって，この行を実行することにより，m は前に 100 だけ進み，動いた跡が画面に線として表示される。また，11 行目の rt は，引数で与えられた度数だけ右に向きを変える Turtle クラスのメソッドである。よって，この行の実行により，m のタートルは右に 90 度向きを変える。以下同様である。

21) 人に連絡をとれる状況を保つために，メモ帳に電話番号を記載することはできても，メモ帳の中にその人自身を入れることはできないであろう。

22) 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ところだが，本書ではオブジェクトを変数に代入するということにする。

23)



24) オブジェクトに対してインスタンスメソッドを呼び出すことを，オブジェクトに**メッセージ**を送るという言い方もする。

25) ヒキスウと読む。

2.4 コンパイルと実行

このプログラムをコンパイルして実行してみよう。Java のソースプログラムのファイル名は，クラス名と一致させて，“クラス名.java” という名前にするのが普通である[26)]。よって，上記のプログラムを T21.java というファイル名で java-everyone の下の turtle ディレクトリに作成したとしよう。

T21.java をコンパイルするには，以下のように javac コマンドを用いる。

```
> javac T21.java
```

javac コマンドにより，引数で与えられたソースプログラムの中に書かれているクラスごとに，クラスファイルが作成される。この場合，T21.class が作成される[27)]。このクラスファイルを実行するには，JVM を起動するコマンドである，**java コマンド**を用いる。

```
> java T21
```

java コマンドは，引数で与えられた文字列をクラス名と考え，それを格納してあるクラスファイル（この場合は T21.class）を探し，その main の中身を実行する[28)]。これによって，リスト 2.1 の右に示した絵が描かれる。

TurtleFrame のウィンドウは，上部にメニューバーが付いている。左の File メニューの中から Quit を選ぶことによりプログラムが終了する。また，Speed メニューの中から選択することにより，描画のスピードを変えられる。一番上の no turtle を選べば，亀は画面から消え，プログラムの最後まで一気に描画が行われる。Ctrl-C[29)]を入力すると，java コマンドを強制的に終了させることができる。

練習問題 2.1： 図 2.1 の星の絵を描くプログラム P20.java を作成しよう[30)]。

[26)] 特に，この例のように，public という修飾子（7.3 節参照）を付けて定義されたクラスの場合は，ファイル名とクラス名が必ず一致している必要がある。

[27)] ここでエラーが表示されたら，T21.class は作成されていない。エラーメッセージを手がかりに，どこが間違っているか探し，修正し，再度コンパイルをしよう。たとえば，「T21.java:4: シンボルを見つけられません。シンボル：クラス TurtleFrame」というエラーの場合には，TurtleFrame.class が同じディレクトリにないというエラーである。サポートページのソフトウェアを展開してできた java-everyone の下の turtle というディレクトリで作業しているか，いま一度確認してみよう（1.5，9.3 節参照）。

[28)] その途中で，TurtleFrame と Turtle のクラスを用いる行に出会うが，そのたびに，java コマンドはこれらに対応するクラスファイルを探して読み込む。

[29)] Ctrl キーを押しながら C を入力すること。OS によっては強制終了の手順は異なることがある。

[30)] クラス名は P20 とすること。

2.5 TurtleFrame と Turtle の仕様

このようなプログラムを書くためには，各クラスのオブジェクトがどういうメソッドを受け付けるのかといったことを知る必要がある。1.3 節で説明したように，クラスを用いたプログラムを書く人が用いることのできる機能は，そのクラスの **API** と呼ばれている。以下が，TurtleFrame と Turtle の API の仕様である。

TurtleFrame クラス

コンストラクタ

TurtleFrame() TurtleFrame を，デフォルトの大きさ (400 × 400) で生成する。
TurtleFrame(int width, int height) TurtleFrame を width × height の大きさで生成する。

メソッド

void add(Turtle t) タートル t をこのウィンドウに追加する。
void remove(Turtle t) タートル t をこのウィンドウから削除する。
void clear() いままでに描かれたすべての線を消す。
void addMesh() 方眼紙のような桝目を表示する。

Turtle クラス

コンストラクタ

Turtle() (200,200) という座標に 0 度の角度で Turtle を生成する。
Turtle(int x, int y, int angle) (x, y) という座標に angle 度の角度で Turtle を生成する。

メソッド

void fd(int n) n だけ前に進む。
void bk(int n) n だけ後ろに進む。
void rt(int n) n 度だけ右に向きを変える。
void lt(int n) n 度だけ左に向きを変える。
void setColor(java.awt.Color nc) ペンの色を nc に変更する。
int moveTo(int x,int y) (x, y) という座標の方向を向き，(x, y) まで進む。動いた距離を返す。
int moveTo(Turtle t) タートル t の方向を向き，t と同じ座標まで進む。動いた距離を返す。
int moveTo(int x,int y, int angle) (x, y) という座標の方向を向き，(x, y) まで進んだ後，angle の方向を向く。動いた距離を返す。
int getX() 現在の座標の X 成分を返す。
int getY() 現在の座標の Y 成分を返す。
int getAngle() 現在の角度を返す。
Turtle clone() 自分と同じ状態のタートルを生成して返す。
void up() ペンを上げる。
void down() ペンを下ろす。ペンを下ろした状態で進むと，その軌跡が画面に線として描画される。
boolean isDown() ペンを上げた状態なら false，下げた状態なら true を返す。
void speed(int x) タートルの動きの速さを x に設定する。x = 20 がデフォルトである。数字が小さいほど速い。
static void speedAll(int x) タートル全体の速さを 1〜3 で指定する。1 が高速，3 が低速。

フィールド

java.awt.Color tcolor 亀の絵の色。初期値は緑色。
double tscale 亀の絵の大きさを表す浮動小数点数。初期値は 0.4。
static boolean withTurtleAll もし，false ならすべてのタートルが亀の絵を表示しないで瞬時に描画を行う。true なら通常の描画を行う。初期値は true。

このうちで，T21.java では，TurtleFrame の add メソッド，それに，Turtle の fd と rt メソッドを用いた[31]。メソッドの一覧の中で，先頭に static と書かれているもの（この中では speedAll が該当する）は第 3 章で説明するクラスメソッド，それ以外はインスタンスメソッドである。

このような仕様があれば，それを見ながら，これらのクラスを用いたプロ

[31] 次節で述べるように，実は，TurtleFrame と Turtle の最初のコンストラクタも利用している。

グラムを作ることができる。まだ説明していない部分が多いが，大まかな雰囲気はつかんでもらえると思う。

練習問題 2.2：T21.java の 12 行目の前に，f（に代入された TurtleFrame）に対する addMesh メソッドの呼出しを挿入してみよう。14 行目の後に，f に対する clear メソッドの呼出しを挿入してみよう。また，rt を lt に，fd を bk に置き換えてみよう (P21.java)[32)]。

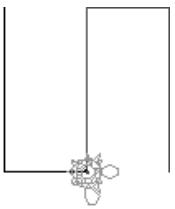
[32)]練習問題のファイル名は，サポートページからダウンロードできる解答と対応させるために付けてある。T21.java を書き換えて，ファイル名を変えずに保存してもよい。

2.6 次の例題 —コンストラクタとメソッドの多重定義（オーバーロード）—

もう少し複雑な例を考えてみよう。

リスト 2.2 T22.java

```
1  public class T22 {
2      public static void main(String[] args){
3          int x = 300, y = 200, d = 100;                  // int 型の変数を用意
4          TurtleFrame f = new TurtleFrame(700,500); //引数のあるコンストラクタ呼出し
5          Turtle m = new Turtle(x,y,180);
6          Turtle m1 = new Turtle(x+ d,y+ d,0);
7          java.awt.Color c = new java.awt.Color(255,0,0); //赤色オブジェクトを生成
8          m1.setColor(c);                                 //m1 の色を赤色に指定
9          f.add(m);
10         f.add(m1);
11         m.fd(d);
12         m1.fd(d);
13         m.lt(90);
14         m1.lt(90);
15         d = d / 2;    //d の値を d/2 に変更
16         m.fd(d);
17         m1.fd(d);
18         m1.moveTo(m);
19     }
20 }
```



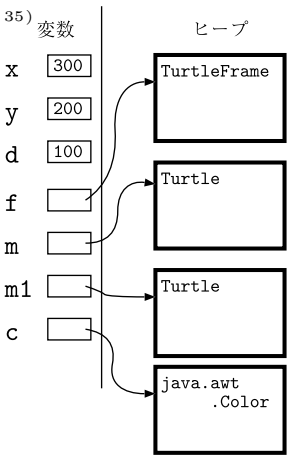
(1) 整数値（プリミティブ値）の変数への代入

3 行目で，三つの変数を型宣言して初期化している。今度の型は，整数を表す型である int である。Java 言語で扱う値には，オブジェクトと**プリミティブ値**の二種類がある（第 6 章）。200 などの整数値はプリミティブ値であり，オブジェクトではない。また，int は**プリミティブ型**と呼ばれるものであり，クラス型ではない[33)]。オブジェクトとプリミティブ値では，変数への記憶の仕方が異なる。2.3 節で，オブジェクトの実体はヒープ上に作られ，変数には

[33)]プリミティブ型，プリミティブ値は，**基本型**，**基本値**ということもある。

その参照が記録されることを述べた[34]。それに対し，プリミティブ値は，変数に値そのものが記録される。傍注[35] の図を参考にしてほしい。オブジェクトとプリミティブ値の違いについては 2.8 節でもう一度考える。

[34] クラス型のように，オブジェクトの参照を意味する型のことを，**参照型**と呼ぶ。Java 言語では，参照型とプリミティブ型の 2 種類の型がある。参照型は，さらに，クラス型，インターフェース型，配列型の 3 種類に分られる (5.1, 8.4 節)。

[35]


(2) コンストラクタ

次に，4 行目で new を用いて TurtleFrame のインスタンスを生成しているが，今回は，700, 500 という整数値が引数として与えられている。このように，new によるインスタンス生成式は，一般に次の形をしている。

new クラス名(引数式 1, . . . , 引数式 n)

この式を実行することにより，**コンストラクタ**と呼ばれる，クラスのインスタンスを生成する手続きが呼び出される。それぞれのクラスは，コンストラクタを複数もつことができる。そして，その中のどれが実際に呼ばれるかは，呼出し時に与えられた引数の数と型によって決められる[36]。前節の TurtleFrame クラスの仕様を見てほしい。コンストラクタの欄に二つの項目が書かれているが，これは，TurtleFrame クラスには 二つのコンストラクタが存在し，一つは引数がないもので，もう一つは二つの int 型の引数をとり，それらが順にウィンドウの幅と高さを指定していることを示している。このように，引数が n 個のコンストラクタの仕様は

クラス名(引数 1 の型 仮引数 1, . . . , 引数 n の型 仮引数 n)

という形で書かれている[37]。よって，4 行目を実行することにより，二つ目のコンストラクタが呼び出され，幅が 700, 高さが 500 の TurtleFrame が生成され，それが変数 f に代入される。

5，6 行目も同様である。5 行目では， x, y, 180 が引数に指定されているが，いま，x, y は int 型の変数なので， int 型の引数を 三つもつコンストラクタが呼び出される。x, y にはそれぞれ 300, 200 が代入されているので，引数の式の値は，300, 200, 180 となる。よって，仕様に従い，(300,200) という座標と 180 度の角度をもった Turtle が生成されて，変数 m に代入される。また，6 行目の引数の x+d, y+d は，計算すると，400, 300 という値になるので，(400,300) という座標と 0 度の角度をもった Turtle が生成されて，変数 m1 に代入される[38]。

(3) クラスライブラリの使用

7 行目は一見ややこしいが，これも同様の，java.awt.Color をクラス名とするインスタンスの生成式である。これは，画面に表示する色を意味する，Java プラットフォームに含まれるクラスである。第 1 章で述べたように，Java には標準で大きなクラスライブラリが付属している。このクラスライブラリの利用方法については 3.1 節で詳しく説明する。ここでは，java.awt.Color クラスには，三つの int 型の引数をもつコンストラクタがあり，これらの引数

[36] このことを，コンストラクタの**多重定義**（**オーバーロード**）と言う。本節 (5) のメソッドの多重定義を参照。

[37] **仮引数**は，一種の変数で，ここでは，右側の説明文の中でその引数に与えられた値を参照するのに用いている。

[38] + などの，演算を指示するものを演算子と言う (4.4 節)。

は，生成される色オブジェクトの r（赤），g（緑），b（青）成分の量を 0〜255 の数として表している。0,0,0 なら黒，255,255,255 なら白であり，255,0,0 は赤ということになる[39]。よって，この行では赤色に対応する Color オブジェクトが生成され，それが変数 c に代入される。

次の行で，c の値は m1 に対する setColor というメソッド呼出しの引数として渡される[40]。ところで，メソッドの引数の所に書けるのは式であり，インスタンス生成式も新たに作られたオブジェクトを意味する式であった。よって，7, 8 行目は，変数に代入せずに，まとめて

```
m1.setColor(new java.awt.Color(255,0,0));
```

と書いてもよいことになる。

(4) 変数への代入

9 行目以降は，T21.java と同様に f，m，m1 に対してメソッド呼出しを行っている。途中，15 行目で，d = d/2; と d に d/2 を代入している[41]。代入は，= の右辺の式の値を求めて，それを変数に代入する。いま，d の値は 100 なので，d/2 という式の値は 50 である。よって，この行で，d に代入されている値は 50 に変わる[42]。これにより，16，17 行目は 11，12 行目と同じことが書かれているが，一方は 100 進み，もう一方は 50 進むという具合に動作が異なる。同じことを書いても，そのときの変数の値によって動きが変化することに注意してほしい。

(5) メソッドの多重定義

最後に，18 行目で，m1 に対して m と同じ場所まで移動するようにメソッド呼出しを行っている。前節の仕様には，moveTo というメソッドは，int 型の引数を二つとるものと，Turtle 型の引数を一つとるものと，int 型の引数を三つとるものの三つが存在している[43]。ここでは，m1 という Turtle 型の引数が一つ与えられているので，起動されるのは二番目のメソッドである。このように，同じ名前で引数の数や型が異なるメソッドを複数もたせることをメソッドの**多重定義**（あるいは**オーバーロード**）と言う[44]。多重定義がないと，これらのメソッドに別々の名前を考えなくてはならない。多重定義を用いると，メソッド名としてはオブジェクトが行う処理（いまの場合は，ある場所に移動するという処理）を意味する名前を付けて，その処理の具体的な指定方法（いまの場合は，移動先の場所を座標で指定するか，他のタートルで指定するか）の違いは引数の数や型で指定することが可能となる。

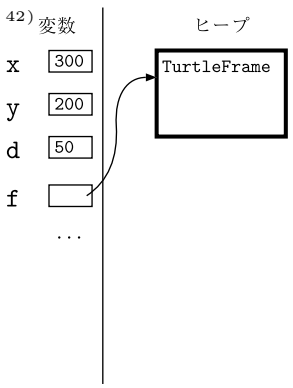
(6) 変数の活用

このプログラムは x, y, d という変数を用いているが，途中の x + d などの式をすべてその値である定数[45] に置き換えても同じ動作をする。それでも変数を用いたのは，このプログラムが，(x, y) を左上の座標，d を 1 辺の長さとしてこの図形を描くという意味をもっており，3 行目の x, y, d の値

[39] 11.5 節を参照。

[40] setColor の意味は，14 ページの Turtle の仕様を参照。

[41] これを，d と d/2 は等しいという意味（ということは，d は 0?）に勘違いしないように。Java の = は，等号ではなく，変数への代入記号である。

[42]


[43] moveTo の前に void ではなく int と書いてあることについては，次節で説明する。

[44] 前述したように，コンストラクタも多重定義が可能である。

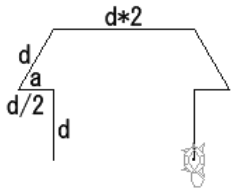
[45] 個々の値の，文字列としての表現を**リテラル**と呼ぶ。ここも，正確にはその値を示すリテラルと言ったほうが正しい。

を変えるだけで，場所や大きさを変えて同じ図形が描けるからである。もし，6 行目が new Turtle(300,300,0) と書かれていたら，なぜ 300 なのか分かりにくいし，1 辺の長さを 150 に変える必要が出ても，どことどこを書き換えたらよいか分かりにくくなる。プログラムの保守[46]を容易にするために，途中に現れる定数はできるだけ減らして，個々の値の間の論理的な依存関係を用いながら，プログラムは書くべきである。

46) プログラムは，一度上から下に書くだけではなく，何度となく修正を行いながら完成させるものである。一度完成したプログラムでも，機能を拡張したり，バグを修正したり，何度も手を入れていかなくてはならないのが普通である。このような一連の作業を**保守**と言う。

練習問題 2.3：T22.java をコンパイルし，実行してみよう。また，このプログラムを変更して，左上の座標が (50,100)，1 辺が 200 で同じ絵が描けるようにしてみよう (P22.java)。

47)

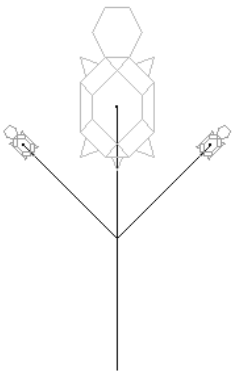


練習問題 2.4：傍注[47] の絵を描くプログラム TurtleHouse.java を作成してみよう。x, y, d, a という四つの変数に対し，左下の座標は (x,y), 家の高さは d, ひさしの長さは d/2, 屋根の角度は a, 屋根の斜面の長さは d, 屋根の横幅は 2d である。これらの変数の値をいろいろ変えて実行してみよう。

2.7　さらにもう一つの例題 — 値を返すメソッドとインスタンス変数 —

リスト 2.3　T23.java

```
1  public class T23 {
2      public static void main(String[] args){
3          int d = 100, x, y, a;
4          TurtleFrame f = new TurtleFrame();
5          Turtle m = new Turtle(200,300,0);
6          f.add(m);
7          m.fd(d);
8          x = m.getX();                    // m の X 座標のとり出し
9          y = m.getY();                    // m の Y 座標のとり出し
10         a = m.getAngle() − 45;           // m の角度のとり出し
11         Turtle m1 = new Turtle(x, y, a); //m1 の生成
12         f.add(m1);
13         m1.fd(d);
14         Turtle m2 = m.clone(); //m2 の生成
15         f.add(m2);
16         m.rt(45);
17         m.fd(d);
18         m2.tcolor = new java.awt.Color(0,255,255);    // m2 の亀の色を水色に変える
19         m2.tscale = m2.tscale * 4;   // m2 の亀を現在の 4 倍の大きさにする
20         m2.fd(d);
21     }
22 }
```



(1) 値を返すメソッド

前節までの例題のメソッド呼出しは処理を実行するだけであったが，メソッドは，処理の結果得られた値を呼出しもとのプログラムに返すこともできる。14 ページの Turtle の仕様を見てほしい。各メソッド名の先頭に，void とか int とか Turtle とか書かれている。この中で，**void** は値を返さないことを示す特別な表記であるが，それ以外は，それぞれのメソッドが返す値の型を意味している。値を示す表現のことを式と言ったが，m.getX() といったメソッド呼出しも式であり，メソッドにより返される値を意味している[48)]。たとえば，仕様には getX メソッドは，タートルの x 座標という int 型の値を返すと書かれている。よって，8 行目の m.getX() は，m(に代入されているタートルオブジェクト) の x 座標を意味する int 型の式となる。このプログラムでは，その値を x という変数に代入している。9, 10 行目も同様である。10 行目の右辺の m.getAngle() − 45 は，m.getAngle() が返す値から 45 を引いた値を示す式である。このプログラムでは，この三つの値を求めて変数に代入してから 11 行目でそれを用いて新たな Turtle オブジェクトを生成しているが，メソッドやコンストラクタの引数には式が書けるのだから，8〜11 行目をまとめて，

Turtle m1 = **new** Turtle(m.getX(), m.getY(), m.getAngle() − 45);

と書いても同じ意味になる。

14 行目では，m に対して，clone() というメソッドを呼び出している。このメソッドは，仕様によると，自分と同じ状態の Turtle を新たに生成して返すことになっている。このように，メソッドは，オブジェクトを生成することもオブジェクトを返すこともできる。

[48)] void 型のメソッドの呼出し式は，例外的に値を意味しない式である。

(2) インスタンス変数

さて，もう一度 Turtle の仕様を見てほしい。コンストラクタ，メソッドと並んで，フィールド という項目があり，java.awt.Color tcolor および double tscale と書かれている。これは，Turtle には tcolor と tscale という名前の**インスタンス変数**（**インスタンスフィールド**とも呼ばれる）があって，それらの型が java.awt.Color および double [49)] であるということを示している[50)]。**フィールド**は，インスタンスフィールド（インスタンス変数）と，次章で扱うクラスフィールド（クラス変数）がある[51)]。本書では，インスタンス変数，クラス変数という用語を主に使い，インスタンス変数とクラス変数を総称するときにフィールドという言葉を使うことにする。

インスタンス変数はそれぞれのオブジェクトがもつ変数である。これまで扱ってきた変数（ローカル変数）と同じように，代入することもできるし，変数の値を式として使うこともできる。違いは，これがオブジェクトごとに存在しており，オブジェクトの状態を表しているということである（図 2.2 参照)。Turtle のインスタンス変数 tcolor と tscale は，このタートル自身の

[49)] 6.1 節に述べるが, double というのは浮動小数点数（実数の近似値）を表すプリミティブ型である。6.1 節のプリミティブ型の一覧表を参照。

[50)] API の仕様は，変数宣言と同じ形で 型名 変数名 と書かれている。

[51)] 先頭に static と書かれている withTurtleAll はクラス変数である。

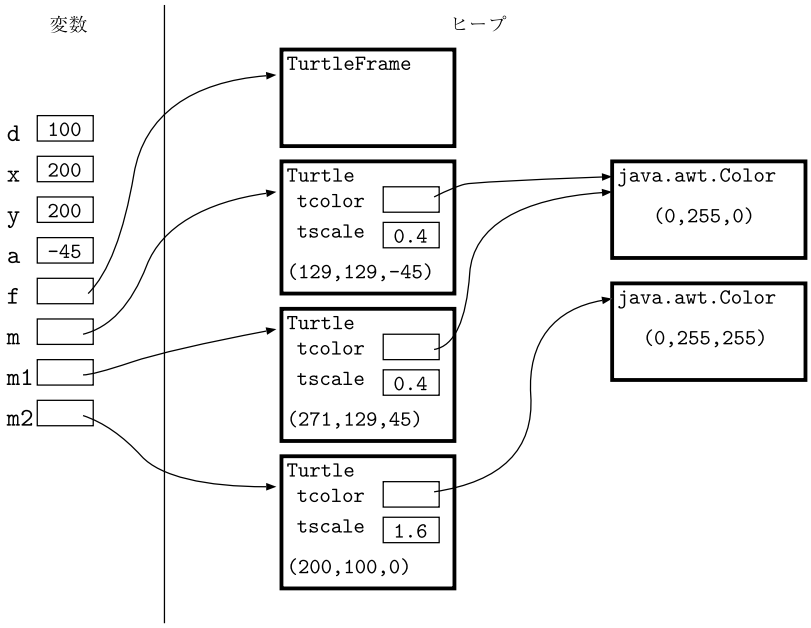


図 2.2　T23 のプログラム実行後の変数とヒープの状態

色と大きさを意味しており，この変数の値を変えることにより，画面の表示も変化する[52)]。

インスタンス変数は，変数名だけではどのオブジェクトのものか分からないので，次のように，オブジェクトと変数名のペアで指定する。

オブジェクト式.インスタンス変数名

18 行目は，新しく色オブジェクト（水色に相当する）を生成し，m2.tcolor, すなわち，m2 のインスタンス変数 tcolor に代入している。また，19 行目は，m2 のインスタンス変数 tscale の値に対してその 4 倍を計算して再び m2 の tscale に代入している。すなわち，現在の値（初期値である 0.4 のはずである） の 4 倍に変更している。これらにより，m2 の亀の表示の色が水色になり，大きさが 4 倍になる。m2 の状態を変えただけで，m や m1 の状態は変化しないことに注意してほしい。インスタンス変数は個々のオブジェクトが個別にもっており，その値を変えても，他のオブジェクトに影響を与えない。このプログラムの 20 行目まで実行した時点での各変数およびオブジェクトの状態を図 2.2 に示す[53)]。Turtle オブジェクトの括弧の中は，そのタートルの座標と向き，Color オブジェクトの括弧の中は，R,G,B 成分の値である。

練習問題 2.5：14 ページの仕様を見ながら，TurtleFrame や Turtle に用意されている，様々なコンストラクタやメソッド，インスタンス変数を用いた絵を描こう。ファイル名は何でもよい。変数の初期値を変えるだけで絵の大きさや場所が変わるように工夫しよう[54)]。

52) 実際には，変数の値を変えるとすぐに表示が変化するのではなく，亀を動かしたときに初めて表示が変化する。このようなインスタンス変数の利用は，決して推奨されていない（7.3 節）。

53) m および m1 の tcolor の値は，Turtle が生成されたときに初期値として代入されている。ここに図示されている以外にも，たくさんのオブジェクトがこのプログラムの実行のために作られていることが，Turtle.java のプログラムを読めば分かるであろう。

54) 複雑な絵を描くのに必要な制御構造などを次章以降で学ぶので，ここでは凝ったものを作る必要はない。

2.8 オブジェクトとは？ — 再び

ここまでの説明で，オブジェクトとはどういう “もの” か，雰囲気をつかんでもらえたと思う。まず，オブジェクトは内部に状態をもつ。仕様に書かれたインスタンス変数の値はもちろんであるが，それ以外にも，様々な状態をもっている。たとえば，Turtle は，現在の x, y 座標と向き，ペンの色，ペンを下ろしているか上げているか，どの TurtleFrame の上に乗っているかといった状態をもっている。また，TurtleFrame は，いまどのタートルが上に乗っているか，どんな線が描かれているかといった状態をもっている。実は，これらの状態も，API の仕様に書くことによって公開されていないだけで，Turtle および TurtleFrame のインスタンス変数として実現されている。実際，Turtle は，x, y という名前のインスタンス変数をもっており，現在の座標を保持している。しかし，これらの変数の値をユーザのプログラムから直接に変更されてはプログラムの動きがおかしくなってしまうので，fd などのメソッドを通してのみアクセスできるように設定されている（7.3 節参照）。そして，API の仕様にもこれらは載せていない。「オブジェクトは，インスタンス変数として状態をもつ」これが第一の特徴である。

次の特徴は，「オブジェクトはインスタンスメソッドの呼出しに応じて処理を実行する」ことである[55)]。メソッドの中でオブジェクトが行う処理は，大まかにいって二つある。一つは，自分の状態を変えることである。たとえば，Turtle は fd メソッドの中で，インスタンス変数 x と y の値を変化させている。もう一つは，（自分自身を含む）他のオブジェクトのメソッドを呼び出すことである。Turtle が fd によって移動した後，その軌跡が線として画面に残るが，これは，Turtle が fd メソッドの中で，自分が乗っている TurtleFrame に対して，一般には公開されていないメソッド（傍注 55 参照）を呼び出して新しい線分を追加するように依頼することにより実現されている。

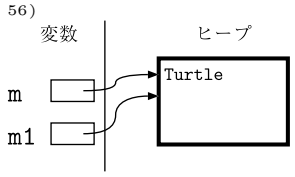
[55)] メソッドについても，API には含まれていないものも存在する。たとえば，TurtleFrame は，線分の両端の座標と色を指定して線分を登録する，addLineElement(int xx, int yy, int x, int y, java.awt.Color c) というメソッドをもつ。

もう一つ意識してほしい特徴は，「オブジェクトは同一性という概念をもっている」ことである。リスト 2.1 で m の Turtle は，メソッドが起動されて状態が変わっても別のオブジェクトになるわけではない。同一のオブジェクトの内部状態が変化しただけである。また，m と m1 は，生成された直後はまったく同じ状態をもっているが，別々のオブジェクトである。一方，

```
Turtle m = new Turtle();
Turtle m1 = m;
```

というプログラムを実行した後の m と m1 は，同一のオブジェクトである[56)]。実際，この後に，m.fd(100); m1.fd(100); と順にメソッド呼出しを行うと，一つのタートルが連続して動く。テキスト上では m と m1 という別のものに命令を送っているようだが，それらの意味している “もの” は同一で

[56)]



ある。このように，同一のものか別のものかという峻別ができることが，オブジェクトの重要な性質である。

これは，プリミティブ値と比べれば，よく分かる。

```
int x = 50;
int y = 20 + 30;
int z = x;
```

というプログラムに対し，x と y, および，x と z は同一のものと見なすべきであろうか。数の世界に同一性の概念は入りにくい。また，オブジェクトとメソッド呼出しによる計算は実行が遅くなるため，足し算などの頻繁に行う計算は，その枠組み以外のものとして提供したほうが効率がよい。このような理由で，Java では，数，文字，論理値などはオブジェクト以外のものとして扱い，このような値を**プリミティブ値**と呼んでいる[57)][58)]。オブジェクトとプリミティブ値では，“変数に代入する” という行為の意味が異なったが，これも，同一性の概念から考えれば納得できよう[59)]。

Turtle などの参照型[60)]の変数にはオブジェクトへの参照が代入されるが，それ以外にもう一つだけ代入可能な値がある。それが **null** であり，意味のあるものが何も代入されていないことを示す特別な値である。null は，どの参照型の変数にも代入可能である。

```
Turtle m = null;
TurtleFrame f = null;
```

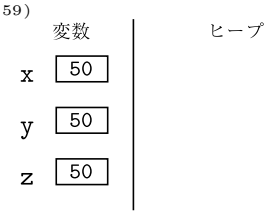
null が代入されている変数に対しメソッド呼出しを行うプログラムは，明らかに間違っている。そのようなエラーは，コンパイル時には検出されない。プログラムを実行したときに，JVM が，java.lang.NullPointerException という例外（エラーなどの例外的な事象，9.1 節）を発生させ，プログラムの実行をそこで終了させる[61)]。

ここまでの例題プログラムを見て，タートルグラフィックスを使ってお絵描きをしただけで，プログラムをした気になれないと思う人もいるかもしれない。それはその通りである。ここでは，最初の例題として，すでに存在するクラスを利用するだけのプログラムを説明した。しかし，通常，Turtle や TurtleFrame のようなクラスを定義すること，つまり，自分の解きたい問題は何をオブジェクトとすれば自然に表現できるか考えて，それにどういうコンストラクタやメソッドやインスタンス変数をもたせるのが自然か考えて（場合によっては，Turtle や TurtleFrame のような API の仕様書を書いて），クラスを作成することが本当のプログラミングの作業となる。

[57)] Smalltalk のように, 数などもオブジェクトとして扱うオブジェクト指向言語もある。

[58)] どうしてもプリミティブ値をオブジェクトと見なしたいときがある。そのときには，ラッパークラスを用いる（6.7 節）。

[59)]



[60)] 2.6 節の (1) を参照。

[61)] Turtle などのウィンドウを開くプログラムでは, 他にスレッドが動作しており，java コマンドを起動した端末に例外が生じたことを表示するだけで，プログラムの実行は終了しない（9.1 節参照）。