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。

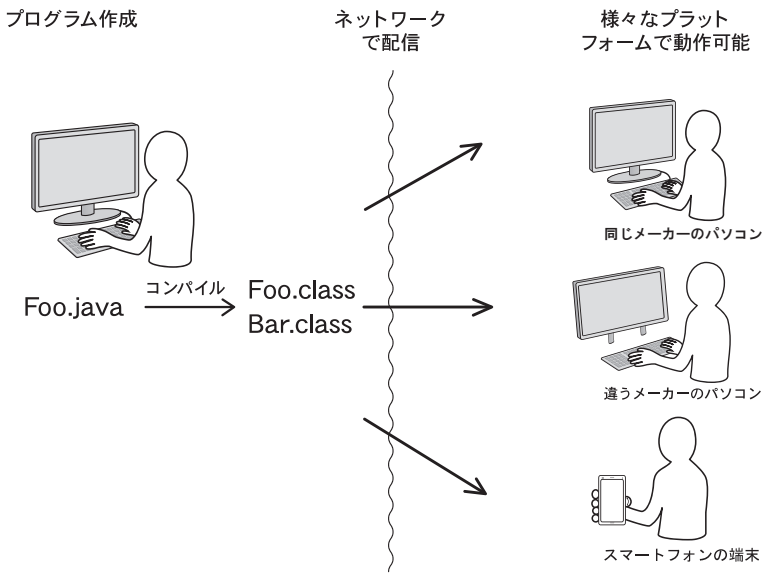


図 1.1　Java のプログラム開発/配布/実行の流れ

(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] cuncurrent 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。

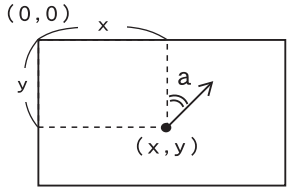
オブジェクトがどういう種類のものか定義したものを**クラス**と言います[5)]。本書のタートルグラフィックスのライブラリは，Turtle という画面上に現れるタートルのクラスと，TurtleFrame というタートルの動きまわるウィンドウのクラスからなっています[6)]。そして，オブジェクトはクラスを指定して作成します。また，あるクラスに属しているオブジェクトのことを，そのクラスの**インスタンス**と言います。よって，この絵は，TurtleFrame クラスのインスタンスを生成し，Turtle クラスのインスタンスを生成し，そのタートルに「前に 100 進め」と「右に 144 度回れ」とい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。

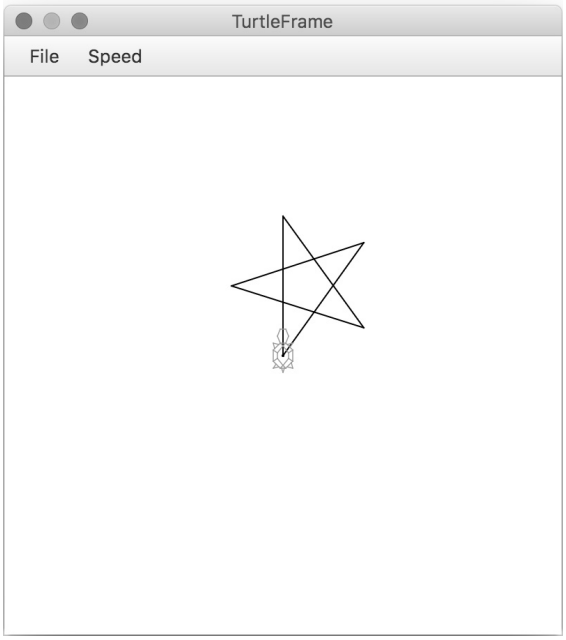


図 2.1　タートルグラフィックで描いた星型

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。

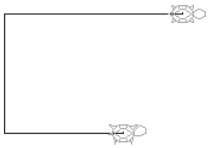
[11)] たとえば，3 行目は，
```
public class
T21
```
{
と 3 行に分けられていても同じ意味です。

[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;
```

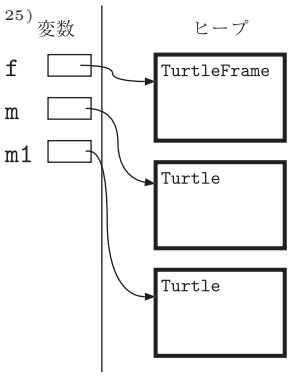
とコンマで区切って並べることにより，一度に型宣言を行うこともできるし，

```
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)

(8) メソッド呼び出し

オブジェクトが行う手続きを**インスタンスメソッド**，あるいは単に**メソッド**と言います[26]。9 行目からは，インスタンスメソッドを呼び出すことによりオブジェクトに手続きを実行させています。9 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，ウィンドウ上に m を載せるように依頼しています。オブジェクトに対するインスタンスメソッドの呼び出し[27]の一般形は次の形です。

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);

ここで**引数**[28]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す。引数が二つ以上あるときには，括弧の中にコンマ (,) で区切って並べます。

インスタンスメソッド呼び出しが行われると，呼び出されたオブジェクトはメソッドに応じた処理を行います。よって，この 1 行により 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ます。10 行目も同じです。 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で現れます。

11 行目の fd は，引数で与えられた数だけ前に動くという，Turtle クラスで定義されたメソッドです。引数は 100.0 と指定されています。これは，100 という数を，実数の近似値のコンピュータ上での表現形式である倍精度の浮動小数点数という形式で表現したものです[29]。通常，コンピュータは実数を近似値として扱います。そして，近似の精度の違いにより，その近似値の表現方法に倍精度と単精度の二種類があります（7.1 節）。この行を実行することにより，m は前に 100 [30]だけ進み，動いた跡が画面に線として表示されます。また，12 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 クラスで定義されたメソッドです。よって，この行の実行により，m は右に 90 度向きを変えます。以下同様です。

[26] メソッドにはインスタンスメソッドと 4 章で扱うクラスメソッドがあります。メソッドという言葉は両者を総称する時に用います。

[27] 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します。

[28] ヒキスウと読みます。

[29] コンピュータ内では整数と実数の近似値は区別されており，100 は整数，100.0 は倍精度浮動小数点数を表しています。

[30] プログラム中の定数としては 100.0 ですが，数学的に 100.0 と 100 という異なる数がある訳ではないので，本文中でたまたま整数となる実数値は 100 などと記すことにします。

2.3 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ましょう。ここでは，コマンドプロンプトでの実行方法を説明します[31][32]。この説明に現れるコンパイルなどの用語については，1.3 節を参照してください。

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一般的です[33]。上記のプログラムを T21.java というファイル名で chap02 フォルダに作成したとします。

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

```
> javac T21.java
```

[31] Eclipse では「実行」を意味するボタンを押すことによりコンパイルと実行が一度になされます。

[32] コマンドプロンプトで，パッケージに置かれたソースファイルのコンパイルや実行する方法については，5.4 節で扱います。

[33] この例のように，public という修飾子（5.4.2 項参照）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一致している必要があります。

javac コマンドにより，このファイルに書かれているクラスごとにクラスファイルが作られます。この場合，T21.class が作成されます[34)35)]。

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**を用います。

```
> 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 の中身を実行します。これによって，リスト 2.1 の右に示した絵が描かれます。

TurtleFrame のウィンドウは，上部にメニューバーが付いています。左の File メニューの中から Quit を選ぶとプログラムが終了します。また，Speed メニューの項目を選択して，描画のスピードを変えられます。一番上の no turtle を選べば，亀が画面から消え，プログラムの最後まで一気に描画が行われます。プログラムが暴走して反応がなくなった時には，Ctrl-c[36)]を入力することにより，java コマンドを強制的に終了させることができます。

練習問題 2.1： 図 2.1 の星の絵を描くプログラム P20.java を作成して実行しよう。T21.java と同じフォルダに作ること[37)]。

34) エラーが表示されたら，エラーメッセージを手がかりに間違っている場所を探し，修正し，再度コンパイルをします。

35) コンパイルの途中で，TurtleFrame と Turtle を用いる行に出会いますが，これらに対応するクラスファイルが存在しなかったり，対応するソースファイルの方が新しければ，これらのコンパイルも行われます。

36) 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が異なることがあります。Eclipse には終了ボタンがあります。

37) Eclipse では，「新規 Java クラス」ボタンで作成します。Eclipse 版では，各ファイルの最初に package chap02; といった行が必要です（5.4 節）。

2.4 TurtleFrame と Turtle の API の仕様

T21 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かといったことを知る必要があります。ライブラリやクラスを利用したプログラムを書く人が呼び出すことのできる機能のことを，それらの **API** と呼びます[38)]。以下は，タートルグラフィックスのライブラリの API の仕様です[39)]。

38) プログラムを書く人に重要なのは，どういう手順でどんな機能を利用できるかという API の情報であり，それを実現しているライブラリのプログラムの詳細ではありません。

39) Point クラスは TurtleFrame で用いられます。

TurtleFrame クラス　　**tg パッケージ**

コンストラクタ

TurtleFrame()　TurtleFrame をデフォルトの大きさ (400 × 400) で作成する。
TurtleFrame(double width, double height)　TurtleFrame を width×height の大きさで作成する。

メソッド

void **add**(Turtle t)　Turtle t をこのウィンドウに追加する。
void **remove**(Turtle t)　Turtle 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。
void **addControlArea**()　左上に 10 × 10 の大きさの赤と青の領域を表示する。
Point **getMousePosition**()　マウスがクリックされるのを待ち，その位置を示す Point オブジェクトを作成して返す。

Turtle クラス **tg パッケージ**

コンストラクタ

Turtle() 座標 (200, 200) に 0 度の角度で Turtle を作成する。
Turtle(double x, double y, double a) 座標 (x, y) に a 度の角度で Turtle を作成する。

メソッド （57 ページに追加のメソッドがある。）

void **fd**(double n) n だけ前に進む。
void **bk**(double n) n だけ後ろに進む。
void **rt**(double n) n 度だけ右に向きを変える。
void **lt**(double n) n 度だけ左に向きを変える。
void **setColor**(javafx.scene.paint.Color nc) ペンの色を nc に変更する。
void **setWidth**(double width) 線の太さを width に変更する。デフォルトは 1.0。
double **moveTo**(double x, double 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。
double **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。
double **moveTo**(double x, double y, double 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。
double **getX**() 現在の座標の x 成分を返す。
double **getY**() 現在の座標の y 成分を返す。
double **getAngle**() 現在の角度を返す。
Turtle **clone**() 自分と同じ状態のタートルを作成して返す。
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の線が画面に描画される。
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) タートルの速さを x にする。数が小さいほど速い。x = 20 がデフォルト。
static void **speedAll**(int x) タートル全体の速さを 1（高速）〜4 （低速）にする。3 がデフォルト。

フィールド

javafx.scene.paint.Color **tColor** 亀の絵の色。初期値は緑色 (Color.LIME)。
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** 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。

Point クラス **tg パッケージ**

フィールド

double **x** x 成分。
double **y** y 成分。

TurtleFrame と Turtle のメソッドの項を見てください。T21.java では，TurtleFrame の add メソッド，それに，Turtle の fd と rt メソッドを用いました[40]。先頭に static と書かれているのは第 4 章で説明するクラスメソッド（この中では speedAll が該当します），それ以外はインスタンスメソッドです。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グラムを作ることができます。まだ説明していない部分が多いですが，大まかな雰囲気はつかめると思います。

[40] 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ます。

練習問題 2.2：T21.java の 13 行目の前に f （に代入された TurtleFrame）に対する addMesh メソッドの呼び出しを，15 行目の後に f に対する clear メソッドの呼び出しを挿入してみよう。また，lt，bk を使ったり，タートルが動く距離を画面からはずれるくらい大きくしたりしよう (P21.java)[41)42)]。

[41)] 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のものです。T21.java を書き換えても構いません。

[42)] 画面からタートルが出ると，スクロールバーが画面につきます。

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 —

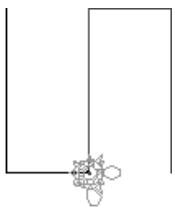
もう少し複雑な例（リスト 2.2）を考えましょう。

(1) 実数値（プリミティブ値）の変数への代入

4 行目で三つの変数を型宣言して初期化しています。今度の型は，倍精度の浮動小数点数という形式で実数の近似値を表す double です。Java 言語で扱う値には，オブジェクトなどの**参照値**と整数値や実数値などの**プリミティブ値**の二種類があります。300 などの整数値や 300.0 などの実数の近似値はプリミティブ値でありオブジェクトではありません。参照値とプリミティブ値では変数への記憶の仕方が異なります。2.2 節で，オブジェクトの実体はヒープ上に作られ変数にはその参照が記録されることを述べました。それに

リスト 2.2　プリミティブ値，コンストラクタ，ライブラリの利用（T22.java）

```
1  import tg.*;
2  public class T22 {
3      public static void main(String[] args){
4          double x = 300.0, y = 200.0, d = 100.0;          // double 型の変数を用意
5          TurtleFrame f = new TurtleFrame(700.0, 500.0); //引数のあるコンストラクタ呼び出し
6          Turtle m = new Turtle(x, y, 180.0);
7          Turtle m1 = new Turtle(x+d, y+d, 0.0);
8          javafx.scene.paint.Color c = new javafx.scene.paint.Color(0.8, 0.0, 0.0, 1.0);     //赤色
9          m1.setColor(c);       //m1 の色を c（8 行目で作成した色オブジェクト）に指定
10         f.add(m);
11         f.add(m1);
12         m.fd(d);
13         m1.fd(d);
14         m.lt(90.0);
15         m1.lt(90.0);
16         d = d / 2;            //d の値を d/2 に変更
17         m.fd(d);
18         m1.fd(d);
19         m1.moveTo(m);
20     }
21 }
```



対し，プリミティブ値は変数に値そのものが記録されます[43]。参照値とプリミティブ値の違いについては 2.8 節でもう一度説明します。

Turtle などのクラスに対応した型のことを**クラス型**と言います。より一般に，オブジェクトの参照を意味する型のことを，**参照型**と言います[44]。それに対し，整数を表す型である int や実数の近似値を表す型である double などの，プリミティブ値の型を**プリミティブ型**と呼びます[45]。クラスはプログラマが自由に定義できますが，プリミティブ型は，整数を表す byte, short, int, long [46]，単精度浮動小数点を表す float，倍精度浮動小数点を表す double，論理値[47]を表す boolean，文字を表す char の 8 つしかありません（7.1 節）。

(2) コンストラクタ

5 行目も new を用いた TurtleFrame インスタンスの作成ですが，今度は 700.0 と 500.0 が引数として与えられています。new によるインスタンス生成式は，一般に次の形をしています。

new クラス名(引数式 1, . . . , 引数式 n)

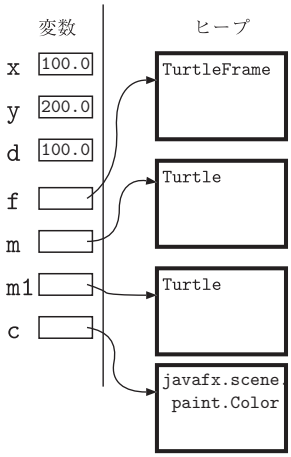
この式を実行することにより，**コンストラクタ**と呼ばれる，クラスのインスタンスを作成する手続きが呼び出されます。それぞれのクラスは，複数のコンストラクタを持つことができます。そして，その中のどれが実際に呼ばれるかは，呼び出し時に与えられた引数の数と型によって決められます[48]。前節の TurtleFrame クラスの仕様を見てください。コンストラクタの欄に二つの項目がありますが，これは，TurtleFrame クラスには 二つのコンストラクタが存在し，一つは引数がないもので，もう一つは二つの double 型の引数をとり，それらが順にウィンドウの幅と高さを指定していることを示しています。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ます[49]。よって，5 行目を実行することにより，引数の数と型が合っている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 が作成され，それが変数 f に代入されます。

6，7 行目も同様です。6 行目では x, y, 180.0 が引数に指定されていますが，x, y は double 型の変数なので， double 型の引数を三つもつコンストラクタが呼び出されます。x, y にはそれぞれ 300.0, 200.0 が代入されているので，引数式の値は 300.0, 200.0, 180.0 です。よって，13 ページの仕様に従い，(300, 200) という座標と 180 度の角度をもった Turtle が作成され，変数 m に代入されます。また，7 行目の引数の x+d, y+d を計算すると 400.0, 300.0 になるので，(400, 300) という座標と 0 度の角度をもった Turtle が作成されて変数 m1 に代入されます。

[43] 下図は，参照型とプリミティブ型の変数への記録方法の違いを表しています。



[44] 型は，参照型とプリミティブ型の 2 種類があります。参照型は，さらに，クラス型，インターフェース型，配列型などに分けられます（6.1, 9.5 節）。

[45] プリミティブ型，プリミティブ値は，**基本型**，**基本値**ということもあります。

[46] 型により表現できる数の範囲が違います。

[47] 3.1 節で説明します。

[48] このことを，コンストラクタの**オーバーロード**（**多重定義**）と言います。

[49] **仮引数**は一種の変数で，この API の仕様では，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ます。

(3) Java API の使用

8 行目はややこしく見えますが，これも javafx.scene.paint.Color をクラス名とするインスタンス生成式とそれを用いた変数 c の初期化です。第 1 章で述べたように，Java には Java API と呼ばれるクラスライブラリが付属していますが，このクラスはそれに含まれるものです。そして，画面に表示する色を意味しています。このクラスの説明は 4.4 節および 15.2 節で行いますが，この クラスには double 型の引数を四つもつコンストラクタがあり，作成される色オブジェクトの red，green，blue 成分の量，および，不透明度（下に描かれたものの色をどれだけ透過させないか）を 0 以上 1 以下 の double 型の数として表しています。よって，0.0, 0.0, 0.0, 1.0 なら黒，1.0, 1.0, 1.0, 1.0 なら白であり，0.8, 0.0, 0.0, 1.0 は（少し暗い）赤です。よって，この行では不透明な赤色を意味する Color オブジェクトが作成され，それが変数 c に初期値として代入されます。

次の行で，c の値は m1 に対する setColor というメソッド呼び出しの引数として渡されます[50]。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した。よって，8，9 行目は，変数に代入せずに，まとめて

```
m1.setColor(new javafx.scene.paint.Color(0.8, 0.0,  0.0,  1.0));
```

と書いても同じ動作をします。

[50] setColor の意味は，13 ページの Turtle の仕様を参照して下さい。

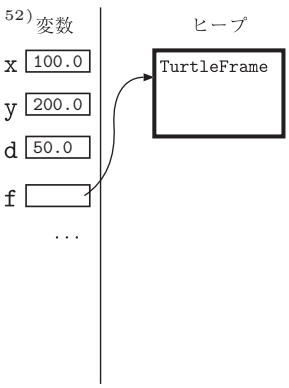
(4) 変数への代入

10 行目以降は，T21.java で行ったのと同様なメソッド呼び出しが f, m, m1 に行われています。途中，16 行目で，d = d/2; と d に d/2 を代入しています[51]。代入文は

変数 = 式;

という形で，実行すると，式 の値を求めて，変数に代入するのでした。いま，d の値は 100.0 なので，d/2 の値は 50.0 です。よって，この行で，d に代入されている値は 100.0 から 50.0 に変わります[52]。これにより，17, 18 行目は 12，13 行目と同じことが書かれていますが，一方は 100 進み，もう一方は 50 進むという具合に異なる動作になります。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ください。

[51] 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してください。Java の = は数学的な等号ではなく，変数への代入記号です。



[52] 変数

(5) メソッドのオーバーロード（多重定義）

最後に，19 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び出しを行っています。前節の仕様によると，moveTo というメソッドは，double 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，double 型の引数を三つとるものの三つが存在しています[53]。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す。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

[53] moveTo の前に void ではなく double と書いてあることについては，2.7 節で説明します。

数もつことをメソッドの**オーバーロード**（あるいは**多重定義**）と言います[54]。オーバーロードがないと全てのメソッドに異なる名前を考える必要があります。オーバーロードを用いると，メソッド名にはオブジェクトが行う処理（この場合は，ある場所に移動すること）を意味する名前を付けて，その処理の具体的な指定（この場合は，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）を様々な方法で行えるようになります。

[54] すでに述べたように，コンストラクタもオーバーロードが可能です。

(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ますが，途中の x + d などの式をすべてその値である定数[55] に置き換えても同じ動作をします。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，4 行目の x, y, d の値を変えるだけで，場所や大きさを変えて同じ図形が描けるからです。もし，7 行目が new Turtle(300.0, 300.0, 0.0) と書かれていたら，なぜ 300.0 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ります。プログラムの保守[56]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す。

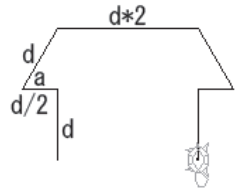
[55] 個々の値の文字列としての表現を**リテラル**と呼びます。ここも，正確にはその値を示すリテラルと言ったほうが正確です。

[56] 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す。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，何度も手を入れる必要がでてきます。このような一連の作業を**保守**と言います。

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)。

練習問題* 2.4：傍注 57 の絵を描くプログラム TurtleHouse.java を作成しよう。x, y, d, a という四つの変数に対し，左下の始まりの点の座標は (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 です。これらの変数の値をいろいろ変えて実行しよう。

[57]



2.6 インポート宣言

リスト 2.2 では，Java API で提供されている javafx.scene.paint.Color クラスを用いました。Java API はグラフィカルユーザインターフェース，アプレット，ネットワーク，入出力など，多彩な機能を提供する膨大な数のクラスから構成されています。それを管理するには，ファイルシステムのフォルダに相当するようなクラスを分類整理する機能が必要です。それを提供するのが**パッケージ**（5.4 節）です。

javafx.scene.paint.Color というクラス名は，javafx.scene.paint というパッケージに属する Color という名前のクラスを意味しています[58]。パッケージ名も，javafx.scene は javafx パッケージのサブパッケージという具合に，階層的に名前を付けられています。このように，パッケージに属するクラス[59]は，

[58] javafx.scene.paint は javafx というグラフィカルユーザインターフェースを作成するためのライブラリにおいて，scene の色などを表現するためのクラスをまとめたパッケージで，第 11 章以降で詳しく扱います。

[59] より正確にいうと，自分がいま書いているプログラムの属するパッケージと異なるパッケージに属するクラス (5.4 節)。

パッケージ名.クラス名 という形で指定します。この記法を，そのクラスの**完全限定名**と言います。しかし，毎回パッケージ名を付けてクラス名などを指定するのは不便なので，それを省略する方法があります。

import パッケージ名.クラス名;

というインポート宣言をファイルの先頭[60]で行っておくと，そのクラスをクラス名 だけで指定できるようになります。あるいは，

import パッケージ名.*;

と宣言すると，そのパッケージに属するすべてのクラスがパッケージ名なしで参照できるようになります。よって，2.5 節のリスト 2.2 の先頭で，

import javafx.scene.paint.Color;

と宣言すれば[61]，8 行目は次のように書くだけでよくなります。

Color c = **new** Color(0.8, 0.0, 0.0, 1.0);

練習問題 2.5：上記のように T22.java を書き換えよう (P23.java)。

T21.java や T22.java には，最初に

import tg.*;

という宣言が入っていました。これは，tg というパッケージに入っているクラスに名前だけでアクセスできるということを意味しています。実際，tg はタートルグラフィックスのパッケージであり，Turtle と TurtleFrame は，このパッケージに属したクラスです[62][63]。

[60] すなわち，ファイル中の最初のクラスなどの宣言より前。

[61] あるいは，
import
javafx.scene.paint.*;

[62] よって，new Turtle() は，new tg.Turtle() と書いてもよいです。

[63] 5.4 節参照。

2.7　さらにもう一つの例題 — インスタンス変数 —

オブジェクトは，インスタンス変数として状態をもつことができます。そのことを，次の例題（リスト 2.3）で学びましょう。

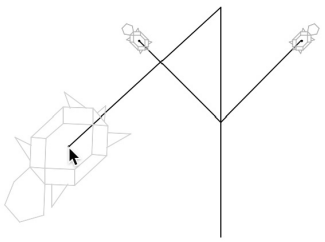
(1) 値を返すメソッド

5 行目において d, x, y, a は全て double 型の変数と宣言されていますが，d の初期値は double ではなく，int の 100 という値が与えられているように見えます。このように double 型の式を書かないといけない所に書かれた int 型の式は，値が自動的に double に変換されてから用いられます。よって，この d = 100 により， d = 100.0 と同じく d には 100.0 が代入されます。7 行目のコンストラクタの引数や 18 行目のメソッドの引数も同様です[64]。double であることを意識するためには 100.0 などと書いた方がいい

[64] ただし，もし int の引数をとるコンストラクタも存在すると，そちらが選択されます。

リスト 2.3　インスタンス変数（T23.java）

```
 1  import tg.∗;
 2  import javafx.scene.paint.∗;
 3  public class T23 {
 4      public static void main(String[] args){
 5          double d = 100, x, y, a;
 6          TurtleFrame f = new TurtleFrame();
 7          Turtle m = new Turtle(200, 300, 0);
 8          f.add(m);
 9          m.fd(d);
10          x = m.getX();                    // m の x 座標のとり出し
11          y = m.getY();                    // m の y 座標のとり出し
12          a = m.getAngle() − 45;           // m の角度のとり出し
13          Turtle m1 = new Turtle(x, y, a); //m1 の作成
14          f.add(m1);
15          m1.fd(d);
16          Turtle m2 = m.clone();           //m2 の作成
17          f.add(m2);
18          m.rt(45);
19          m.fd(d);
20          double newscale = m2.tScale ∗ 4; // m2 の tScale の 4 倍の数
21          m2.tScale = newscale;            // m2 の tScale に代入
22          m2.tColor = new Color(0.0, 1.0, 1.0, 1.0);   // m2 の亀の色を水色に変える
23          m2.fd(d);
24          Point p = f.getMousePosition();
25          m2.moveTo(p.x, p.y);
26      }
27  }
```



場合もありますが，記述が煩雑になるので，今後，座標や角度などで，たまたま整数となる値をプログラム中で表す時には，特に必要がなければ int のリテラル[65]を用いることにします。

[65]定数を表記したもの。傍注 55 参照。

前節までの例題のメソッド呼び出しは処理を依頼するだけでしたが，メソッド呼び出しでは，処理の結果得られた値を，呼び出し元のプログラムに返してもらうこともできます。13 ページの Turtle の仕様を見てください。各メソッド名の先頭に，void とか double とか Turtle とか書かれています。この中で，**void** は値を返さないことを示す特別な表記です。それ以外は，それぞれのメソッドが返す値の型を示しています。たとえば，仕様には， getX メソッドはメソッドを呼び出されたタートルが自分の x 座標という double 型の値を返すと書かれています。よって，10 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 座標の値を返します。値を示す表現のことを式と言いましたが，m.getX() といったメソッド呼び出しも式であり，メソッドにより返される値を意味しています[66]。よって，この代入文

[66]void 型のメソッドの呼び出し式は，例外的に値を意味しない式です。

で，m の x 座標が変数 x に代入されます。11, 12 行目も同様です。12 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す[67)68)]。

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 13 行目でそれを用いて新たな Turtle オブジェクトを作成していますが，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，10〜13 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ります。

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

(2) インスタンス変数

もう一度 Turtle の仕様に戻りましょう。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，javafx.scene.paint.Color tColor および double tScale と書かれています。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**とも呼ばれます）があり，それらの型が javafx.scene.paint.Color および double であるということを示しています[69)70)]。

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す。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，変数の値を式として使うこともできます。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す（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色と大きさを意味しており，変数の値を変えると画面表示も変化します [71)]。

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します。

オブジェクト式 . インスタンス変数名

20 行目は，m2 のインスタンス変数 tScale の値の 4 倍を計算して，double 型の変数 newscale に初期値として代入しています。そして，21 行目で m2 の tScale に newscale の値を代入しています。これにより，m2.tScale の値を現在の値（初期値である 0.4 のはずです）の 4 倍に変更しています[72)]。22 行目は，新しく色オブジェクト（水色に相当します）を作成し，m2.tColor, すなわち，m2 のインスタンス変数 tColor に代入しています。これらにより，m2 の亀の色が水色になり，大きさが 4 倍になります。m2 の状態を変えても m や m1 の状態は変化しないことに注意してください。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いるので，その値を変えても他の

67) m.getAngle() の値は double 型，45 は int 型ですが，第 7 章で述べるように，double と int の間の計算式は，int の値を double に変換してから double の演算として行われ，double の値が得られます。

68) API の仕様に void 以外の返値の型が書かれており，値が返される場合でも，その値を使わないなら今までと同じようなメソッド呼び出し式が可能です。実際，moveTo はタートルを移動させると同時にその移動距離が返されますが，リスト 2.2 の 19 行目やリスト 2.3 の 25 行目では，その値を使っていません。

69) API の仕様には，変数宣言と同じ形で 型名 変数 と書かれています。

70) **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，第 4 章で扱うクラスフィールド（クラス変数）の二種類があります。API の仕様で先頭に static と書かれている withTurtleAll はクラスフィールドで，それ以外はインスタンスフィールドです。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します。

71) 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，その後に亀を動かしたときに初めて表示が変化します。インスタンス変数の概念の説明のためにこれらをインスタンス変数にしていますが，このような，アプリケーションプログラムから自由にアクセスできるインスタンス変数の利用方法は決して好ましいものではありません。そのため，tColor, tScale の値を変化させるためのメソッド setTColor と setTScale も Turtle クラスに用意されています。それらについては 5.4.2 項で述べます。

72) この 2 行は，変数 newscale を用いずに
m2.tScale=m2.tScale*4;
と書いても同じです。

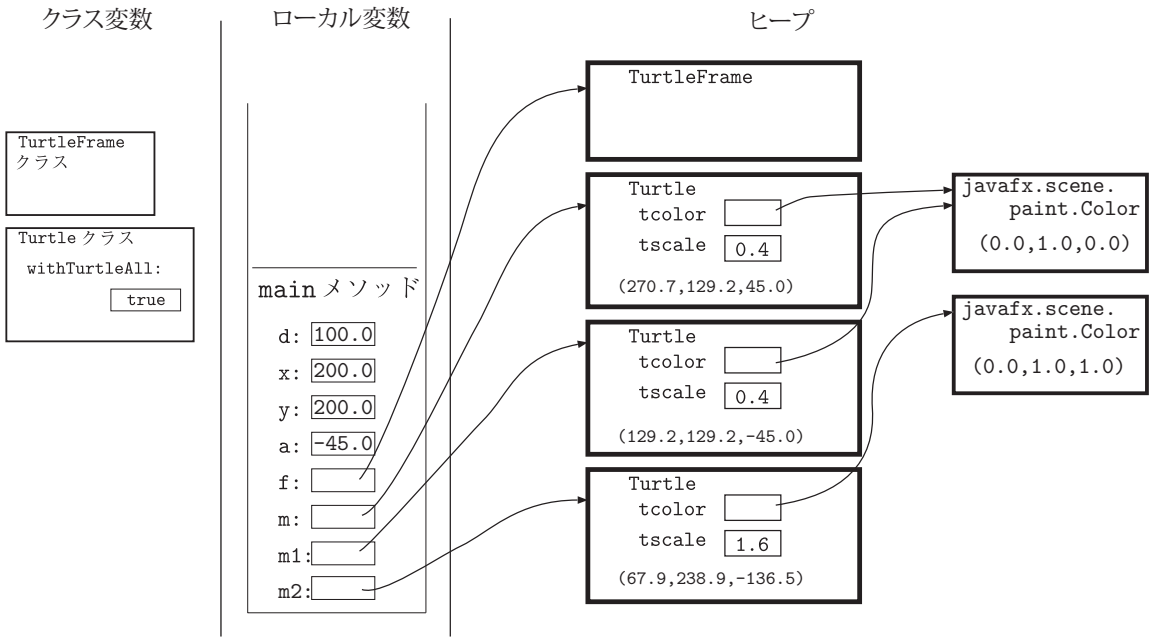


図 2.2 T23 のプログラム実行後の変数とヒープの状態

オブジェクトに影響を与えません。

このプログラムの 23 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 に示します[73]。この図で，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，red, green ,blue 成分の値です。図左のクラス変数は第 4 章で説明します。

24 行目で，TurtleFrame の getMousePosition メソッドを呼び出しています。この呼び出しは，すぐに終了するのではなく，マウスがクリックされるまで待って，その時のマウスの x, y 座標を Point クラスのオブジェクトとして返します。Point は，x, y というインスタンス変数だけを持ちメソッドを持たないクラスです。このように，構造を持ったデータを表現するのにオブジェクトを用いることができます[74]。この行では，返された Point オブジェクトを変数 p に初期値として代入しています。そして，次の行で，p のインスタンス変数 x, y の値を取り出して，moveTo メソッドで m2 をそこまで移動しています[75]。

練習問題 2.6： デフォルトの位置に表示されたタートルが，マウスがクリックされるのを待って，そこに移動して星型を描くプログラムを作成しよう（Mouse21.java） [76][77]。

練習問題 2.7： 最初にマウスがクリックされた点にタートルを作成し，そこからマウスで指定された 10 個の点をつないだ線画を描き，描画が終わったらタートルを削除するプログラムを作成しよう（Mouse22.java）。

[73] m および m1 の tColor に代入されている Color オブジェクトは，Turtle が作成されたときに初期値として与えられています。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，これから勉強する中で分かります。

[74] 構造を持ったというのは，一つのプリミティブ値ではなく，それからペアなどの方法で組み合わせて作られたという意味です。

[75] マウスの位置を得るのに，ここではメソッド呼び出しの中でマウスが押されるのを待っていますが，通常は，これとは異なるイベント駆動という方法を用います。詳細は，14 章で説明します。

[76] up, moveTo, down を順に呼び出すことで，線を描かずに移動ができます。

[77] rt(162) してから描画を始めると，きれいな星型になります。

練習問題* 2.8：13 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[78]。

[78] 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ありません。

2.8　オブジェクトとは？

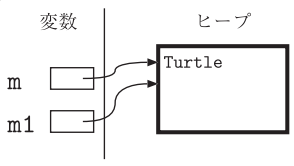
ここまでの説明で，オブジェクトとはどういう “もの” か，雰囲気がつかめたと思います。オブジェクトは内部に状態をもちます。仕様に書かれたインスタンス変数の値はもちろんですが，それ以外にも，様々な状態をもっています。たとえば，Turtle は，現在の x, y 座標と向き，ペンの色，ペンを下ろしているか上げているか，どの TurtleFrame 上に乗っているかといった状態があります。これらの状態も，API の仕様に書くことによって公開されていないだけで，Turtle のインスタンス変数として実現されています。実際，Turtle は，x, y という名前のインスタンス変数をもっており，現在の座標を保持しています。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なっており，API の仕様にもこれらは載せていません（5.4.2 項参照）。「オブジェクトは，インスタンス変数として状態をもつ」これがオブジェクトの第一の特徴です。

次の特徴は，「オブジェクトはインスタンスメソッドの呼び出しに応じて処理を実行する」ことです[79]。メソッドの中でオブジェクトが行う処理は，大まかに二つあります。一つは，自分の状態を変えることです。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ます。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す。Turtle が fd によって移動した後，その軌跡が線として TurtleFrame の画面に残るのは，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ます。

[79] メソッドについても，API には含まれていないものも存在します。

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つ」ことです。リスト 2.1 で m に代入された Turtle は，メソッドが呼び出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ありません。同一のオブジェクトの内部状態が変化しただけです。また，m と m1 は，作成された直後はまったく同じ状態をもっていますが別々のオブジェクトです。一方，

[80]




```
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す[80]。実際，この後に，m.fd(100); m1.fd(100); と順にメソッド呼び出しを行うと，

一つのタートルが連続して動きます。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に見えますが，それらの意味している “もの” は同一です。このように，同一のものか別のものか峻別できることが，オブジェクトの重要な性質です。

これは，プリミティブ値と比べれば，よく分かります[81]。

```
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しょうか[82]。数の世界に同一性の概念は入りにくいです。また，オブジェクトとメソッド呼び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くなります。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**と呼んでいます[83][84]。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りましたが，これも，同一性の概念から考えれば納得できるでしょう。

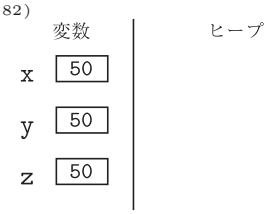
Turtle などの参照型[85]の変数にはオブジェクトへの参照が代入されますが，それ以外にもう一つだけ代入可能な値があります。それは **null** という，意味のあるものが何も代入されていないことを示す特別な値です。null は，どの参照型の変数にも代入可能です。

```
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び出しを行うプログラムは，明らかに間違っています。しかし，そのようなエラーは，コンパイル時には検出されません。プログラムを実行したときに，java.lang.NullPointerException という例外（エラーなどの例外的な事象，8.3 節）が発生し，プログラムの実行はそこで終了します[86]。

ここまでの例題プログラムでは，タートルグラフィックスでお絵描きをしただけで，プログラミングをした気になれないかもしれません。それはその通りです。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ました。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が必要かを考えて（場合によっては API の仕様を書いて），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ります。

[81] **int** は整数を表すプリミティブ型です（7.1 節）。

[82]


[83] 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ります。

[84] プリミティブ値をオブジェクトと見なしたいときがあります。そのときには，ラッパークラスを用います（7.3 節）。

[85] 2.5 節の (1) を参照。

[86] 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ません（8.3 節参照）。