1

Javaによるグラフィクス

1.1 Javaによるプログラミング

Java 言語はマルチプラットフォームのオブジェクト指向言語として知られている．マルチプラットフォームとは，複数種類の計算機，より具体的には複数種類のオペレーティングシステム (OS) 上で，実行可能であることを意味している．すなわち，Windows，MacOS，Linux など，様々な OS 上で実行可能であり，作成したプログラムを様々な計算機上で実行できる．Java の開発環境 (JDK) については，付録 1 章を参照して欲しい．

Java はマルチプラットフォームを実現するために，バイトコードと呼ばれる機械語を解釈・実行する**仮想機械** (virtual machine) を利用する．仮想機械自体はプログラムであり，種類の異なる計算機上で同じ動作をするように作られている．オブジェクト指向言語である Java では，**クラス** (class) という単位でプログラムを書く．クラスは，フィールドと呼ばれるデータと，メソッドと呼ばれる小プログラムによって構成されている．

図 1.1 に，Java によるプログラムの作成と実行の流れを示す．Java のプログラミングでは，まず Emacs などのエディタを用いて，Java 言語の**ソースプログラム** (source program) を記述する．ソースプログラムの入っているファイルはソースファイルと呼ばれるが，通常，Java のソースファイルは ”クラス名.java” というファイル名にする．たとえば，以下の例であれば `Renshu.java` というファイル名になる．Java のプログラムは，アルファベットの大文字と小文字を区別するが，基本的に英小文字が用いられる．ただし，慣習としてクラス名の最初の 1 文字は大文字が用いられる．

以下に簡単なプログラム (`Renshu.java`) の例を示す．

```
public class Renshu {
  public static void main(String[] args) {
    System.out.println("Hello, yama!");
  }
}
```

このプログラムについて簡単に説明すると，最初の行は `Renshu` というクラスが定義されることを示している．次の行からは `main` メソッドが定義されているが，Java のプログラムは，この `main` メソッドから起動される．この `main` メソッドは `System.out.println` という文字列を出力する実行文 1 つからなり，引数で与えられた”`Hello, yama!`”という文字列が出力されることになる．

ソースプログラムを仮想機械で実行可能な形式に変換する作業を**コンパイル** (compile) と呼ぶ．コ

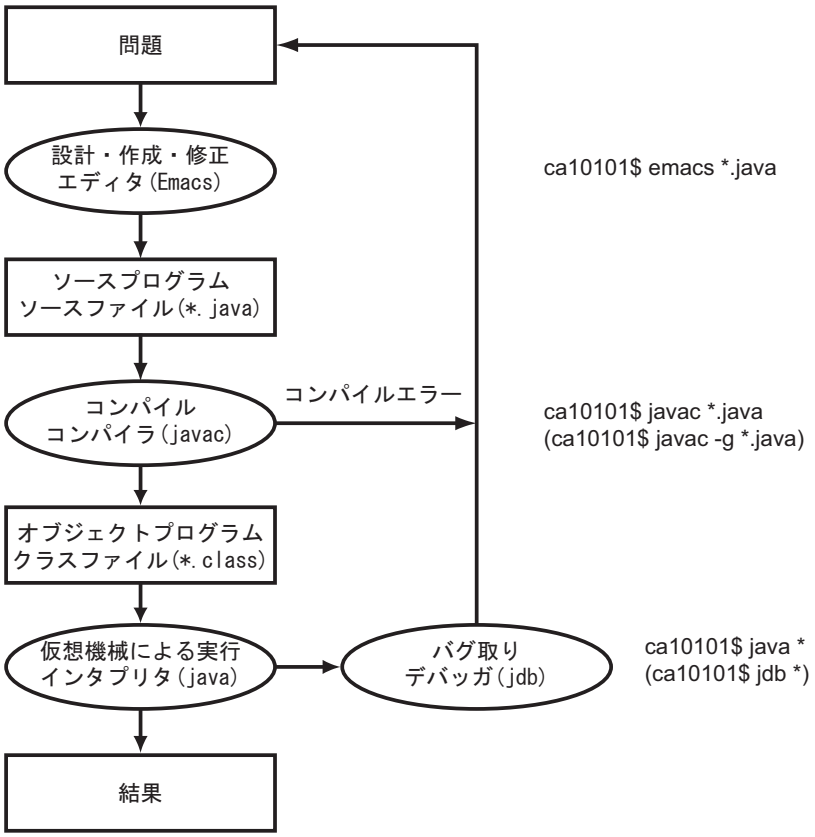

図 1.1 Java プログラミングの流れ

ンパイルによって変換されたプログラムを，一般に**オブジェクトプログラム** (object program) と呼び，オブジェクトプログラムの入っているファイルはオブジェクトファイルと呼ばれる．Java の場合，オブジェクトファイルは ”クラス名.class” というファイル名になり，**クラスファイル** (class file) と呼ばれる．クラスファイルにはファイル名に対応するクラスのプログラムが，バイトコードで記述されている．仮想機械にクラスファイルを与えることで，プログラムが実行される．コンパイルには javac コマンド，仮想機械の実行には java コマンドを用いる．

以下に，MacOS 上でのコンパイルと実行のコマンド例を示す．ここでソースプログラムには，上記のプログラムを用いており，ファイル名は Renshu.java としている．なお，ここで¶ は，リターンキーを意味している．

```
ca10101$ ls ¶                          (ソースファイルは Renshu.java)
Renshu.java
ca10101$ javac Renshu.java ¶           (コンパイルする)
ca10101$ ls -gl ¶                      (クラスファイル Renshu.class ができる)
total 2
-rw-r--r--   1 yama     teacher       418 10 11 19:19 Renshu.class
-rw-r--r--   1 yama     teacher       111 10 11 19:19 Renshu.java
ca10101$ java Renshu ¶                   (プログラムを実行する)
Hello, yama!
```

Java は標準で充実したクラスライブラリ，すなわち有用なクラスファイル群を持っており，それらを

利用することでプログラムを作成する．Java 関連のコマンドや標準クラスライブラリに関する情報は，以下の場所で見つけられる．

`http://docs.oracle.com/javase/`

1.2 グラフィクスプログラミング

図形や文字などを描画するプログラムの作成を，一般にグラフィクスプログラミング (graphics programming) と呼ぶ．グラフィクスの語源であるグラフ (graph) は，本来，線図形を意味するが，計算機用語としてのコンピュータグラフィクスでは，塗りつぶしや画像なども含んだ描画全般を表わす．グラフィクスプログラミングでは，グラフィクス専用のクラスライブラリを利用する．このライブラリは，描画領域の確保などの初期化命令と，線分や多角形などの要素ごとの描画命令などから構成される．

以下に本章のプログラムで利用する AWT(Abstract Window Toolkit) ライブラリのクラスとメソッドを挙げる．読者，特に Java の初学者は，この説明はいったんとばして，1.3 節の「簡単なプログラム」で実際のプログラムの形式を学んでから読んだ方がよいだろう．

`Canvas` **クラス：** 描画領域のクラス

`Canvas` クラスのオブジェクトは, 描画領域を確保するとともに, 描画のためのメソッド (`paint`, `update`) を提供する．以下で紹介するプログラムは，基本的に `Canvas` の機能を利用するために，`Canvas` を拡張 (`extends`) したクラスを定義する．

上位クラスは，`java.awt.Canvas` → `java.awt.Component` → `java.lang.Object`

`paint` 描画メソッド

必要に応じてオーバライドするメソッドで，`Canvas` 内の描画内容を規定する．引数として渡される `Graphics` オブジェクトを用いて，描画命令を実現する．たとえば `Canvas` が新たに表示される場合など画面更新要求が起きると，Java の仮想機械から呼び出されて `Canvas` 内の描画を実行する．

`update` 画像更新メソッド

画面上で `Canvas` が隠された後に，再度描画し直す必要があるときなどに，画面更新要求に伴って自動的に呼び出される．デフォルト (標準) では，背景色で全体を塗りつぶした後に，`paint` メソッドを呼び出して描画を行なう．

`setSize` サイズ指定メソッド

`Component` クラスのメソッドで，引数で与えられた大きさ (ピクセル単位) にする．

`setBackground` 背景色指定メソッド

`Component` クラスのメソッドで，引数で与えられた色を背景色とする．

`setForeground` 描画色指定メソッド

`Component` クラスのメソッドで，引数で与えられた色を描画色とする．

`Graphics` **クラス：** 描画情報および描画のクラス

`Graphics` クラスのオブジェクトは，描画領域，描画のための色やフォントなどの情報を保持

するともに，描画のためのインタンスメソッド提供する．Canvas は Graphics オブジェクトを自動的に生成するので，それを用いて描画を行なう．

上位クラスは，java.awt.Graphics → java.lang.Object

Frame クラス： ウィンドウ生成のクラス

Frame クラスのオブジェクト (インスタンス) は，OS のウィンドウシステムを通じて，スクリーン上にウィンドウを生成する．実際の描画領域は Canvas によって管理されるが，その Canvas を作るためのウィンドウを Frame オブジェクトによって生成し，ウィンドウ内部に Canvas 領域を埋め込んで描画を行なう．

上位クラスは，java.awt.Frame → java.awt.Window → java.awt.Container → java.awt.Component → java.lang.Object

Frame　　コンストラクタ

引数として与えられた文字列を，タイトルバーに持った Frame オブジェクト，すなわちウィンドウを作る．

add　　登録メソッド

Container クラスのメソッドで，Component オブジェクトを Frame に登録する．

pack　　配置確定メソッド

Window クラスのメソッドで，登録されている Component オブジェクトの配置に合わせてウィンドウのサイズを決める．

setVisible　　表示切替メソッド

Window クラスのメソッドで，表示／非表示を切り替える．引数に true を指定すると実際に表示される．

drawLine　　直線描画メソッド

引数で与えられた始点と終点の x, y 座標値を両端とする直線を描画する．

drawString　　文字列描画メソッド

引数で文字列と描画開始位置の x, y 座標値を指定して，文字列を描画する．

drawOval　　楕円描画メソッド

引数で左端の x 座標値，上端の y 座標値，横幅，縦高さを指定して，楕円を描画する（第 2 章以降で利用する）．

1.3 簡単なプログラム

最初に簡単なグラフィクスのプログラム例を示す．

例１： ×線の表示 - CrossLine.java

表示領域において対角線 (2 本) と，実行時にコマンド引数として与えられる文字列とを描画する（図 1.2）．前に書いたように，Canvas クラスを拡張 (extends) して，新たな CrossLine クラスを定義している．なお，addWindowListener については，第 4 章で説明する．

プログラムについて，簡単に説明する．最初の 2 行は abstract window toolkit(awt) ならびにそのイベント (awt.event) パッケージのインポートである．前節で紹介したように，この

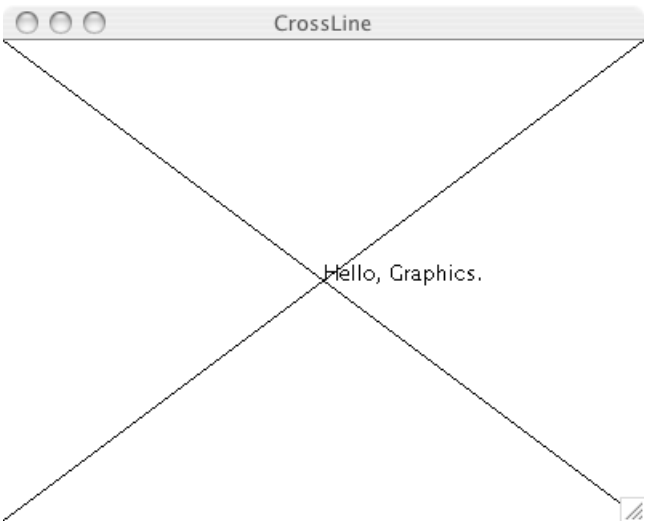

図 1.2 ×線の表示

プログラムで利用している Canvas や Frame などは awt パッケージのクラスであり，awt パッケージをインポートする必要がある．また，イベント関連のパッケージである awt.event も必要であり，これら 2 つのパッケージをインポートしている．4 行目以降が，このプログラムの本体である CrossLine クラスの定義で，Canvas クラスを拡張 (extends) している．

CrossLine クラスでは，まず main メソッドが定義されている．main メソッドでは，実行コマンドで与えられる引数の個数を確認する．実行コマンドの引数は，単語ごとに分割され，String 型の配列 args に格納されているので，配列の長さ args.length で引数の個数を調べられる．引数がなければ，コマンドの実行法に関するメッセージを出力してプログラムを終了する．コマンド引数が与えられていれば，その情報を添えて CrossLine クラス自身のオブジェクトを 1 つ生成する．この際，CrossLine クラスのコンストラクタ (クラスと同名のメソッド，ここでは CrossLine()) が実行される．

CrossLine コンストラクタは，最初に上位クラス Canvas の引数なしのコンストラクタを super(); で呼び出したのち，Canvas の大きさ (400×300 ピクセル) や背景色，描画色，描画文字列などを設定するとともに，オペレーティングシステムが提供するウィンドウ (Java では Frame と呼ぶ) を作成し，Canvas を Frame に埋め込んで，Frame ならびに内部の Canvas を表示している．paint メソッドは，オペレーティングシステムがウィンドウ (Frame) を表示する際に呼び出される．より具体的には，ウィンドウが最初に表示される際や，いったんアイコン化された状態から再表示される際に起動される．このプログラムでは，2 本の直線と文字列を描画している．ちなみに，本書の第 1〜3 章のプログラムは，ほとんどが main メソッド，コンストラクタ，paint メソッドのみで構成されている．

```
import java.awt.*;
import java.awt.event.*;

public class CrossLine extends Canvas {
  public static void main(String[] args) {    // main メソッドの定義
    if (args.length == 0) {                    // コマンド引数の確認
      System.err.println("Usage: java CrossLine <message>");
```

```
    }
    else {
      new CrossLine(args);                      // CrossLine オブジェクトの作成
    }
  }

  private static final int width = 400;         // Canvas の幅
  private static final int height = 300;        // Canvas の高さ
  private static String message = null;         // 描画文字列

  protected CrossLine(String[] words) {         // CrossLine コンストラクタ
    super();                                    // Canvas のコンストラクタ実行
    setSize(width, height);                     // 幅と高さの設定
    setBackground(Color.white);                 // 背景色の設定 (白)
    setForeground(Color.black);                 // 描画色の設定 (黒)
    message = words[0];                         // 描画文字列の設定

    Frame f = new Frame("CrossLine");           // Frame の作成
    f.add(this);                                // Frame に Canvas を登録
    f.pack();                                   // 配置の確定
    f.addWindowListener(new WindowAdapter() { // 無名オブジェクトの作成
      public void windowClosing(WindowEvent e) {
        System.exit(0);                         // ウィンドウ閉鎖で終了
      }
    });
    f.setVisible(true);                         // Frame の表示
  }

  public void paint(Graphics g) {               // paint メソッドの定義
    g.drawLine(0, 0, width-1, height-1);        // 直線: 左上→右下
    g.drawLine(0, height-1, width-1, 0);        // 直線: 左下→右上
    g.drawString(message, width/2, height/2); // 文字列: 中央付近
  }
}
```

実行にあたっては，以下のように表示文字列を指定する．

```
ca10101$ java CrossLine "Hello, Graphics."¶
ca10101$
```

例２： ×線の表示 - CrossLines.java

ここでは CrossLine クラスをさらに拡張して，CrossLines クラスを定義している．以下のように main メソッドで，CrossLines オブジェクトを 3 つ生成しているが，実際には，CrossLines コンストラクタを見ると，super(words); となっていて CrossLine のコンストラクタを呼び出しているだけである（CrossLine のコンストラクタは引数を必要とするので，String 型の配列 words を，そのまま引数として渡している）．つまり，CrossLine オブジェクトを 3 つ生成することになる．

実行すると，例 1 で表示されたウィンドウが 3 つ表示されるはずである．

```
public class CrossLines extends CrossLine {
  public static void main(String[] args) {    // main メソッドの定義
    if (args.length == 0) {                    // コマンド引数の確認
      System.err.println("Usage: java CrossLines <message>");
    }
    else {
      new CrossLines(args);                    // CrossLines オブジェクトの作成
      new CrossLines(args);                    // CrossLines オブジェクトの作成
```

```
      new CrossLines(args);                    // CrossLines オブジェクトの作成
    }
  }

  protected CrossLines(String[] words) {       // CrossLines コンストラクタ
    super(words);                             // CrossLine のコンストラクタ実行
  }
}
```