グラフィックス基礎・演習・第1回

2013年4月9日

今回の内容

1.1 この科目について

この科目は、C 言語のプログラミングがそれなりにできるようになった人を対象に、Java という新しい言語の入門を行いながら、グラフィックスプログラミングの基本とオブジェクト指向と呼ばれるプログラミングの考え方について理解して頂くものです。教科書は使用しません。シラバスに参考書として挙げた書籍などを、必要に応じて読みこなすための基礎を構築するのが目標となります。

授業の進め方　この科目では、各回の授業を次のように進めて行きます。

火曜日・1−2講時・1−542実習室

講義 (60 〜 90 分間)　その回の内容に関する説明を行ないます。席は自由です。資料[1]の配布はこの時間の始めに行ないます。

実習 (残りの時間)　Windows 環境や Linux の環境を使って、プログラミングを行います。2 講時目は TA さんがサポートしてくれます。

シラバス抜粋

講義概要　Java は比較的新しいプログラミング言語で、C 言語に似た文法を持っていますが、オブジェクト指向と呼ばれる考え方に基づいたプログラミングを行うことを意識して設計されています。Java は普通のパソコンから携帯電話までいろいろな機器の上で動作し、標準化された豊富なライブラリ群が用意されていますので、あまり細かいことを意識しなくてもグラフィックス (図形の描画など) や、マウス、キーボードの処理を行ったり、あらかじめ用意されている部品 (ボタンやメニューなど) を組み合わせて比較的容易にグラフィカルユーザインターフェース (GUI) を構築することができます。この科目では、Java プログラミングを学びながら、オブジェクト指向プログラミングの考え方に触れ、グラフィックスの基本、グラフィカルユーザーインタフェースの仕組みやその構築法を学びます。

到達目標　オブジェクト指向のプログラミングの考え方を知る。オブジェクト、クラス、インターフェイス、継承など、オブジェクト指向プログラミングに付随するいろいろな概念を理解する。画面の描画 (グラフィックス) に関する基本的な手法を学ぶ。グラフィカルユーザーインターフェース (GUI) の構築法を知る。

授業方法　配布資料に沿って講義を行います。これと並行して情報処理実習室での Java プログラミングの演習を行います。

[1] 配布資料の多くは `http://www602.math.ryukoku.ac.jp/~nakano/GKiso/` から入手することもできます。

成績評価方法 期末試験 (100 点満点)、平常点 (プログラミング課題と適宜行う小テスト) で評価します。期末試験が x 点、平常点の得点率が $y\%$ のとき、総合的な成績は $x + (100 - x)y/400$ 点 (端数切り捨て) となります。

講義計画
(1) Java とは
(2) オブジェクトの生成とメソッドの起動
(3) クラス定義
(4) メソッドの継承と再定義
(5) コンストラクタ
(6) 配列
(7) 変数とメソッドのまとめ
(8) クラス定義のまとめ
(9) GUI 入門
(10) 図形の描画
(11) インタフェースとイベント処理
(12) 画像の処理
(13) スレッドと例外処理
(14) Swing コンポーネント
(15) まとめ

系統的履修 「計算機基礎実習 I」と「計算機基礎実習 II」、「プログラミング・演習」で学んだ内容が前提となります。

参考文献 立木秀樹、有賀妙子『すべての人のための Java プログラミング 第 2 版』(共立出版) (ISBN: 9784320121942)

1.2 Java とは

Java はプログラミング言語の 1 つです。C 言語に似た文法を持つ言語ですが、大きな違いとして次の 2 点を挙げることができます。

オブジェクト指向言語である。 1 つのソフトウェア (1 つのプログラム、あるいは複数のプログラム群からなる情報システム) を考えるとき、

> 特定の役割や機能を持った多数の (ソフトウェア的な) 部品が、お互いに仕事を依頼し合ったり情報を交換し合ったりすることで全体が機能する

と捉える考え方を**オブジェクト指向 (object-oriented)** と呼びます。「オブジェクト指向」の「**オブジェクト (object[2])**」とは、その 1 つ 1 つの (ソフトウェア的な) 部品のことです。Java は、この「オブジェクト指向」と呼ばれる考え方に基づいたプログラミングを行うことを意識して設計されたプログラミング言語です。

C 言語が、情報を表現するための「データ構造」やそれを処理する「手続き」に着目し、この 2 つを容易に表現することを目指した**手続き型言語**であるのに対して、Java 言語では、それぞれの部品 (オブジェクト) がどのような役割を持っていて、どのような仕事をどのようこなすのか、といった視点でプログラムを記述することを目指した**オブジェクト指向言語**となっています。オブジェク

[2] object という英単語にはいくつかの意味がありますが、ここでは「物体」を意味しています。

ト指向の考え方やそれに関連する特徴的な概念については、この科目の中で少しずつ勉強していくことになります。

作成したプログラムがいろいろなプラットフォームで動作する。　C 言語では、プログラマが書いたソースプログラムをコンパイルすることで、特定の実行環境 (CPU や OS 等) 向けの機械語プログラム (オブジェクトプログラム[3]) に変換し、その機械語プログラムを実行しますが、出来上がった機械語プログラムは、その特定の実行環境でしか動作しません。プログラムが実行される土台となる環境 (どのような CPU か、どのような OS か等) のことを**プラットフォーム (Platform)** と呼びますが、C 言語では、

1. プラットフォーム毎にソースプログラム (の一部) を変えなければならない。
2. プラットフォーム毎にソースプログラムをコンパイルし直さなければならない。

といった制限があります。

たとえば、みなさんが Linux で作成した数値シミュレーションのプログラムは、コンパイルし直さないと (たとえ同一の PC であっても) Windows では実行できませんし、もし、そのプログラムが計算結果をグラフィックスで表現するようなものであるとすると、そもそも Windows ではコンパイルできないかも知れません。なぜなら、グラフィックスのために用意されているライブラリ群が、Linux 環境と Windows 環境では異なっていることが多いからです。

これに対して Java は、1 つのソースプログラムをコンパイルしてできた 1 つの機械語プログラムを、多くのプラットフォームの上で同じように動作させることができます[4]。実際に Java プログラムは、一般のパソコンから携帯電話、家電製品等の組み込みプログラムの実行環境まで、多様なプラットフォーム上で動作します。

Java は、次のような仕組みで、プラットフォーム毎の CPU や OS 等の違いを吸収しています。

1. **Java 仮想機械 (Java Virtual Machine — JVM)** と呼ばれる仮想的な CPU を定義して、この JVM の機械語プログラムを Java コンパイラが生成するようにしました。これにより、JVM の機械語プログラムを実行するためのソフトウエアを、それぞれのプラットフォームに用意することさえできれば、同じ (JVM の) 機械語プログラムがどの CPU でも動作するようになりました。JVM の機械語プログラムを **Java バイトコード**と呼びます[5]。
2. 基本的なデータ構造やアルゴリズムを実現したり、グラフィックスやネットワーク、データベース等を利用するための**豊富な機能を持つ標準ライブラリ群**を定義して、OS の違いをこれらのライブラリ群で吸収しました。もちろん、パソコンと携帯電話では装備されているハードウェアがかなり異なりますから、基本的な部分を共通化し、どうしても変わってしまう部

---

[3] オブジェクトプログラム (object program) の object は「目標」や「目的」を意味していて、オブジェクト指向の object とは別の意味です。

[4] このことを “Write once, run everywhere” と言って、Java を開発した Sun Microsystems 社 (現在は Oracle 社に吸収) が、Java 言語を売り込む際のスローガンにしていました。

[5] Java バイトコードのように、高レベルの (人間が書きやすい) プログラミング言語を低レベル (CPU が直接理解できる機械語などの) 言語へ変換する際に、それらの中間に位置して橋渡しの役割を果たす言語を**中間言語**と呼びます。

分のみを、用途毎にそれぞれ標準化することで、細かなプラットフォーム (OS やデバイス等) の違いを吸収しています。

このため、Java では、細かいプラットフォームの違いを意識せずに、グラフィックス (図形の描画など) や、マウス、キーボードの処理を行ったり、あらかじめライブラリの中に用意されている部品 (ボタンやメニューなど) を組み合わせて比較的容易にグラフィカルユーザインターフェース (GUI) を構築することができます。

1.3 Java プログラムのコンパイルと実行例

簡単な Java プログラムを作成し、コンパイルし、実行してみましょう。

ソースプログラムの作成

まず、エディタを使って、次のようなソースプログラムを作成します[6]。

G101Hello.java

```
1 class G101Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello, world!");
4     }
5 }
```

この G101Hello.java という名前のソースファイルは、C 言語で書かれた次のプログラムと同じことを行う Java アプリケーションのプログラムとなっています。

hello.c

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello, world!\n");
5 }
```

C 言語のソースファイルの名前は「.c」で終わります[7]が、Java 言語の場合は「.java」となります。

ここではプログラムの内容については深く立ち入りませんが、おおよそ、hello.c の 4 行目の

```
printf("Hello, world!\n");
```

が、G101Hello.java の 3 行目の

```
System.out.println("Hello, world!");
```

に対応していることが推察できると思います。また、これを囲むように C では、

```
int main() {
    ...
}
```

があり、Java では

```
public static void main(String[] args) {
    ...
}
```

[6] 行頭の数字は、説明のために付した行番号であって、ソースプログラムの一部ではありません。

[7] このことを「拡張子が .c」であると言うことがあります。

があって、どちらも main という英単語が現れていることが分かります。また、C には対応するものがありませんが、Java では、さらにこれを囲んで

```
class G101Hello {
        ...
}
```

があって、先頭の「class G101Hello」の部分に、ソースファイルの名前の一部 (.java を取り除いたもの) が現れていることに気づきます[8]。これらの意味は、これから勉強して行きますので、ここではあまり気にしないでください。

ソースプログラムのコンパイル

ソースプログラムが完成したら、Java コンパイラを使って、コンパイルを行います。コンパイルの手順は、使用している Java の開発環境によって異なりますが、最も基本的方法は、コンソールウィンドウ[9]を開いて、「javac」というコマンドを実行することです。

Windows 環境

```
Q:\GKiso>javac G101Hello.java
```

Linux 環境

```
s01542f161:~/GKiso$ javac G101Hello.java
```

うまくコンパイルできれば、javac コマンドが「G101Hello.class」という名前の新しいファイルを作成してくれます。確認してみましょう。

Windows 環境

```
Q:\GKiso>dir /b
G101Hello.java
G101Hello.class
```

Linux 環境

```
s01542f161:~/GKiso$ ls
G101Hello.class  G101Hello.java
```

この (Java のソースファイルをコンパイルすることで生成される) ファイルを**クラスファイル**と呼びます。クラスファイルには (コンパイラによって生成された) Java 仮想機械 (JVM) の機械語プログラム (Java バイトコード) が格納されています。

プログラムの実行

Java コンパイラ (javac コマンド) によって作成されたクラスファイルを実行するためには java コマンドを使用します。java コマンドは、クラスファイルに格納されている Java バイトコードを読み取り、その命令を逐次解釈して実行してくれるソフトウェアです。java コマンドのコマンドライン引数には、以下の例のように、クラスファイルの名前から .class を取り除いたもの[10]を指定します。

---

[8] 常に同じにしないといけない訳ではありませんが、同じになる (する) のが普通です。

[9] Linux 環境では「端末」、Windows 環境では「コマンドプロンプト」を起動します。

[10] より正確には、ソースプログラムの冒頭の「class」の後に書いた名前。

Windows 環境

```
Q:\GKiso>java G101Hello
Hello, world!
```

Linux 環境

```
s01542f161:~/GKiso$ java G101Hello
Hello, world!
```

この java コマンドのように、あるプログラミング言語で記述されたプログラムを読み取り、そこに記述されている内容を解釈しながら対応する仕事を逐次行っていくソフトウェアのことを**インタプリタ**と呼びます。java コマンドは Java バイトコードのインタプリタですが、バイトコードを一部分ずつ (あるいはまとめて全部) 使用している CPU の機械語プログラムにコンパイルして[11]、その結果を CPU に直接実行させていくことができるようなっています。これにより、Java バイトコードのより高速な実行が可能になっています。

1.4 演習問題

1. Java プログラムの例として挙げた G101Hello.java を、Windows 環境で、作成、コンパイルし、実行してみなさい。

2. 同様に、次の Java プログラム G102Mean.java を、Windows 環境で、作成、コンパイルし、実行してみなさい。

G102Mean.java

```
import java.util.Scanner;

class G102Mean {
    public static void main(String[] args) {
        int sum = 0, num = 0;
        Scanner sc = new Scanner(System.in);
        System.out.print("整数をいくつか入力した後、");
        System.out.println("最後に ; を入力してください。");
        while (sc.hasNextInt()) {
            sum += sc.nextInt();
            num++;
        }
        if (num > 0) {
            System.out.printf("個数 = %d, 平均 = %.2f\n",
                num, (double)sum/num);
        }
    }
}
```

このプログラムは、次の C プログラム mean.c と同等の仕事を行います。

mean.c

```
#include <stdio.h>

int main() {
    int val, sum = 0, num = 0;
```

[11] このように、プログラムが (インタプリタによって) 実行される際に、必要に応じて、実際の CPU の機械語にコンパイルするコンパイラを「実行時コンパイラ」あるいは「Just-in-time コンパイラ」と呼びます。

```
    printf("整数をいくつか入力した後、");
    printf("最後に ; を入力してください。\n");
    while (scanf("%d", &val) == 1) {
        sum += val;
        num++;
    }
    if (num > 0) {
        printf("個数 = %d, 平均 = %.2f\n", num, (double)sum/num);
    }
}
```

3. PC を Linux 環境で起動して、演習問題 1 と 2 で作成した、ソースファイルとクラスファイルを Windows 環境の `Q:` ドライブから Linux 環境へコピーしなさい。

4. 演習問題 3 でコピーしたクラスファイルを Linux 環境で実行しなさい。

5. 演習問題 3 でコピーしたソースファイルの文字コーを UTF-8 へ、また、行末が LF (改行文字) のみとなるように変換しなさい[12]。さらに、変換後のソースファイルを Linux 環境でコンパイルし、実行してみなさい。

6. 次の Java プログラム `G103TicTacToe.java` を、Linux 環境で、作成、コンパイルし、実行してみなさい。

G103TicTacToe.java

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

class G103TicTacToe extends JPanel {

    int turn = 0;

    G103TicTacToe() {
        setLayout(new GridLayout(3, 3));
        for (int i = 0; i < 9; i++) {
            add(new Cell());
        }
    }

    class Cell extends JButton implements ActionListener {

        Cell() {
            setPreferredSize(new Dimension(100, 100));
            setFont(new Font(Font.MONOSPACED, Font.PLAIN, 64));
            setFocusable(false);
            addActionListener(this);
        }

        public void actionPerformed(ActionEvent e) {
            if (turn++ % 2 == 0) {
                setText("○");
```

[12] テキストファイルの標準の文字コードは、Windows 環境では Shift-JIS に、(瀬田学舎の情報処理実習室の) Linux 環境では UTF-8 となっています。また、瀬田学舎の情報処理実習室に限らず、テキストファイルの行末は、Windows 環境では CR (復帰文字) と LF (改行文字) の 2 つの連続で、Linux/Unix 環境では LF (改行文字) 1 つだけで表すことになっています。

```
            } else {
                setText("×");
            }
            removeActionListener(this);
        }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Tic Tac Toe");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new G103TicTacToe());
        frame.pack();
        frame.setVisible(true);
    }
}
```

このプログラムをコンパイルすると、G103TicTacToe.class というクラスファイルの他に、G103TicTacToe$Cell.class というクラスファイルが作られます。このように、1 つのソースファイルをコンパイルした結果として複数のクラスファイルが生成されることがあります。プログラムの実行の際には、

```
java G103TicTacToe
```

のように、java コマンドの引数には G103TicTacToe を指定しますが、java コマンドの実行の途中で、もう 1 つのクラスファイル G103TicTacToe$Cell.class が読み込まれますので、こちらのクラスファイルも必要となります。

グラフィックス基礎・演習・第 1 回・終り

1.5 付録 : 記号の読み方

記号	一般的な読み方	JIS X 0201 規格での名称
!	びっくりマーク、エクスクラメーションマーク	感嘆符
"	ダブルクオート、二重引用符	引用符
#	シャープ、いげた	番号記号
$	ドルマーク、ダラー	ドル記号
%	パーセント	パーセント
&	アンパサンド、アンド	アンパサンド
’	シングルクオート、一重引用符	アポストロフィー、アクサンテギュ
(	左 (丸) かっこ、開き (丸) かっこ	左小かっこ
)	右 (丸) かっこ、閉じ (丸) かっこ、こっか	右小かっこ
*	アスタリスク、星、スター	アステリスク
+	プラス (記号)、たす、プラ	正符号
,	コンマ、カンマ	コンマ、セディユ
-	マイナス (記号)、ハイフン、ひく	ハイフン、負符号
.	ドット、ピリオド、点、ぽち	ピリオド
/	スラッシュ、スラ、斜線	斜線
:	コロン	コロン
;	セミコロン	セミコロン
<	小なり (記号)	不等号 (より小)
=	イコール、等号	等号
>	大なり (記号)	不等号 (より大)
?	はてなマーク、クエスチョンマーク	疑問符
@	アットマーク	単価記号
[	左ブラケット、左鍵かっこ、左大かっこ	左大かっこ
\	バックスラッシュ、バックスラ、逆スラッシュ	(¥ 円記号)
]	右ブラケット、右鍵かっこ、右大かっこ	右大かっこ
^	ハット、カレット、やま	アクサンシルコンフレックス
_	アンダースコア、下線、アンダーライン	アンダライン
‘	バッククオート、逆クオート、逆引用符	アクサングラーブ
{	左中かっこ、左ブレース、左カーリーブラケット	左中かっこ
|	縦棒、縦線	縦線
}	右中かっこ、右ブレース、右カーリーブラケット	右中かっこ
~	チルダ、波線、にょろ	(￣ オーバライン)