Java の特徴

1991 年頃に米国の Sun Microsystems 社は家電製品用ソフトウェア開発のためのプログラミング言語を作った。その後改良が加えられ、1995 年 5 月に発表されたのが Java である。Java は C 言語や C++に似た構文であり、これらの言語の経験者は比較的容易に Java を学べる。Java はオブジェクト指向プログラミングを実現するための技術がサポートされ、「Write Once, Run Anywhere（プログラムを一度作成したらどこでも動く）」という構想で設計されており、以下のような特徴を持つ。

・一度作成したら、どんなプラットフォーム上でも動作する
・一度作成したら、ネットワーク上のどのマシンからでも利用できる
・一度作成したら、オブジェクト指向によってプログラムを再利用できる

プログラムを一度作成したら、どんなプラットフォーム上でも動作できるようにするために Java では JavaVM（Java 仮想マシン）というシステムを導入している。

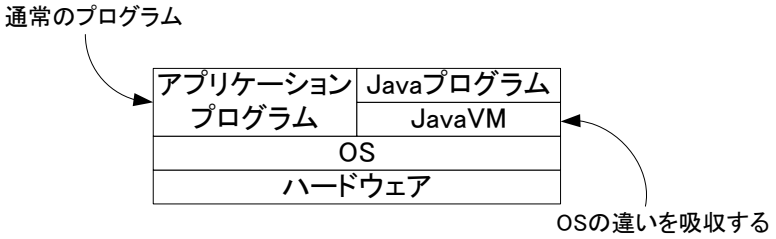

図１ JavaVM

通常、コンピュータでは CPU やメモリなどのハードウェアを管理する OS が動作しており、その OS 上でアプリケーションプログラムが実行される。Java では OS 上で JavaVM を動作させ、その仮想マシン上で OS の違いを吸収して Java プログラムを実行するため、プラットフォームを選ばない。そのため、Java は携帯電話、コンピュータ、ブルーレイディスクプレイヤ、自動車など、どこでも利用されている。JavaVM を通して Java プログラムを実行するため、C 言語や C++といった言語で作成するアプリケーションと比較するならば動作は遅いが、CPU の処理速度が上がっているため、ストレスを感じることは少ないだろう。

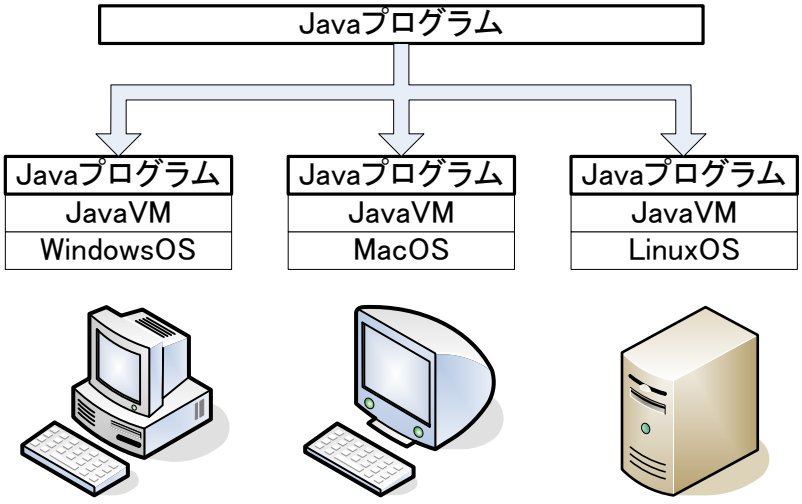

図２ 異なるプラットフォーム上で動作する Java プログラム

「FORTRAN」や「C 言語」などのプログラミング言語は「手続き型言語」と呼ばれるのに対して、C 言語の拡張で

ある「C++」や「Java」などのプログラミング言語は「オブジェクト指向言語」と呼ばれる。

「手続き型言語」では一連の処理の流れを記述していく形でプログラミングを行うが、「オブジェクト指向言語」では、データとそのデータの操作する処理を「オブジェクト」と呼ばれるひとまとまりの部品とみなして一体化し、このオブジェクトの組合せとしてプログラミングを行う。「オブジェクト指向言語」ではプログラムの部品化の作業を徹底的に行うことで、大規模なプログラムや別のプログラムでプログラムの部品を利用する際の再利用性を高めている。

部品化の作業を徹底的に行うということはその分のプログラミングの手間が発生することになるが、Java では標準的に利用する処理やテキスト・Excel・Word のファイル、ネットワーク通信や画像処理といった処理を幅広く扱うための手段が提供されているため、必要最低限のコードを記述することで目的のアプリケーションを作成できる。

Java プログラミング

Java プログラミングの大まかな流れは以下のようになる。

（1） Java プログラミング言語を用いてソースファイル（ファイルの拡張子は[.java] ）を作成する。
（2） コンパイラでソースファイルをコンパイルし、JavaVM で利用できるクラスファイル（ファイルの拡張子は[.class] ）に変換する。
（3） クラスファイルを実行する。クラスファイルは JavaVM で一行ずつ翻訳されて実行される。
（4） （2）、（3）で問題が発生した場合はプログラムの修正作業（デバッグ）を行う。

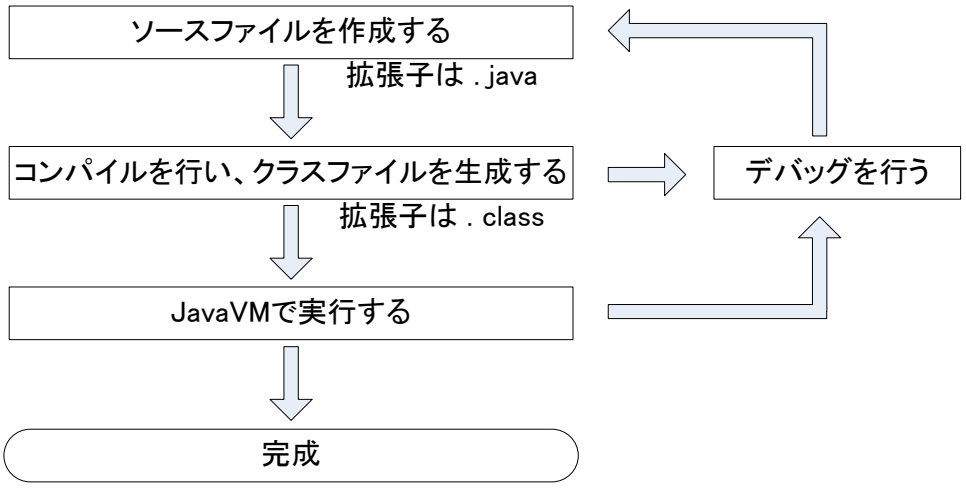

図３ Java プログラミングの流れ

それでは簡単な Java プログラミングを行ってみよう。作成に使うテキストエディタは jcpad を用い、ファイル名は Hello.java とする。

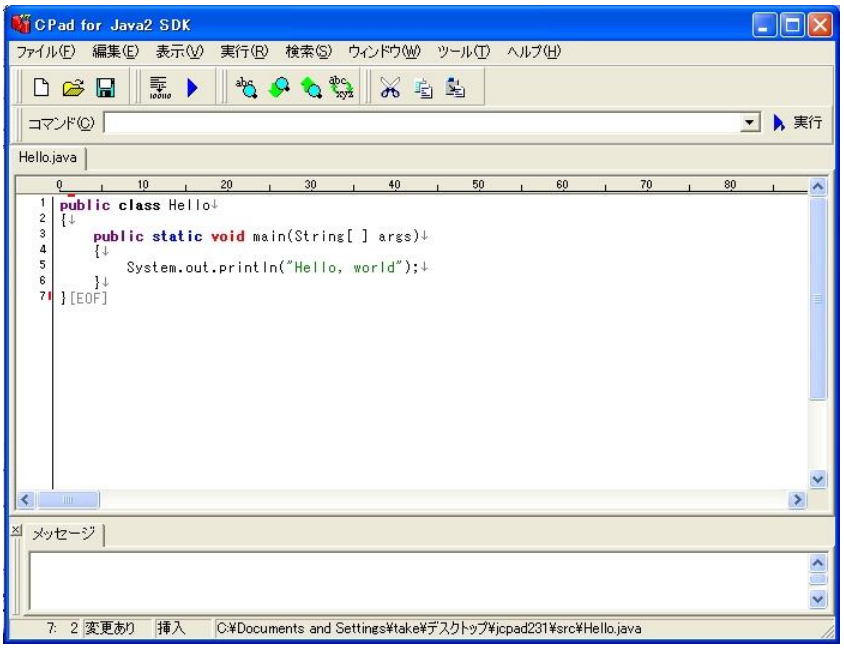

図４ jcpad

以下のプログラムを記述し、名前を付けて保存したら、青い矢印のアイコンをクリックする。コンパイルが自動で行われ、プログラムに問題が無い場合には、コマンドプロンプトの画面が立ち上がり、「Hello, world」と表示される。

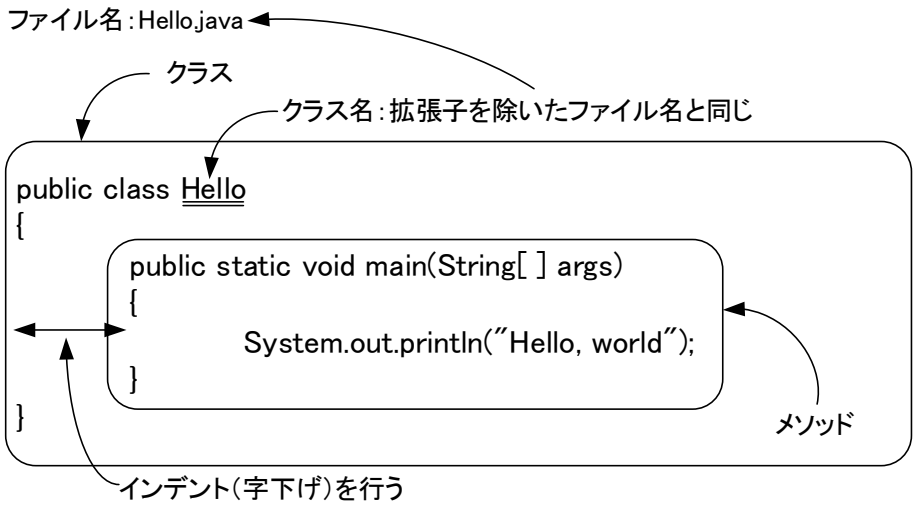

図５ 簡単なJavaプログラム

Java プログラミング言語はデータとそのデータの操作する処理を合わせて記述したクラスというプログラムの集合で成り立っている。クラス内部の処理はメソッド（関数やサブルーチンのようなもの）という単位で記述していく。プログラムの記述に関しては以下のルールがある。

（1） 英数字や記号は全て半角文字で入力する。
（2） 英字の大文字と小文字は区別される。
（3） 単語や括弧の間の空白部分は半角スペースかタブを入力する。
（4） 行末は改行する。
（5） 「:（コロン）」と「;（セミコロン）」、括弧の種類「()」「{}」「[]」を正しく使い分ける。

演習

プログラムの実行が成功した者は以下のようなバグをわざと入れて、どのようなエラーメッセージが表示されるか確認を行いなさい。

（1） スペルミス 「public」を「pubric」などとしてみる。
（2） セミコロンの打ち忘れ 文末のセミコロンをはずしてみる。
（3） 全角のスペースの混在 インデント部分に全角のスペースを入れてみる。
（4） 大文字と小文字の間違い 「System」を「system」などとしてみる。
（5） メソッドの記述間違い 「System.out.println」を「System.println」などとしてみる。
（6） 括弧の対応がくずれている 最後の「 } 」を削除してみる。

標準入出力

コンピュータには通常、入出力装置が用意され、記憶装置との間でプログラムやデータのやり取りを行う。入出力装置の例には以下がある。

入力装置：キーボード、マウス、スキャナ、タッチパネル
出力装置：ディスプレイ、プリンタ

Java プログラムでは標準入力装置として、キーボードが割り当てられ、標準出力装置としてディスプレイが割り当てられており、プログラムを通して、キーボードから値を入力（標準入力）したり、ディスプレイに文字列を表示（標準出力）したりすることができる。今回は Java プログラムで標準出力について学習していく。

文字列の表示

標準出力のプログラムの記述方法はいくつかあるが、図１の例題の５行目で示したものを含めて文字列の表示方法を２つ示す。

System.out.println	文字列を表示した後に改行を行う
System.out.print	文字列を表示するが改行を行わない

これらのプログラムは以下のように用いる。ここで、ダブルクォーテーション（"）で囲まれた文字列は「文字列リテラル」と呼ばれる。

プログラム例１

```
public class Prog01_01
{
        public static void main(String[ ] args)
        {
                System.out.println("Java プログラムの練習");
                System.out.println("文字列を表示する");
        }
}
```

出力例

```
Java プログラムの練習
文字列を表示する
```

プログラム例２

```
public class Prog01_02
{
        public static void main(String[ ] args)
        {
                System.out.print("基本的な");
```

```
                System.out.print("Java プログラム");
        }
}
```

出力例

```
基本的な Java プログラム
```

クラス名とファイル名の管理は各自で行うこと。クラス名はファイル名と一致しなくてはならず、クラス名の最初の文字は大文字を用いるのが通例である。クラスや変数などに付ける名前を「識別子」と呼ぶ。識別子には以下のルールがある。

（1） アルファベット、数字、ひらがな、漢字等のいわゆる文字（$と_を含む）を使用できる。ただし、識別子の１文字目に数字は使用できない。
（2） 大文字と小文字は区別される。
（3） 識別子の長さに制限はない。
（4） 以下に示す Java の「キーワード」は使用できない。

※$は Java コンパイラがバイトコードを生成する際に内部的に利用する文字であり、使わないことが推奨されている。

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

文字列の連結と改行

文字列の連結や改行方法として「+」や「¥n」が用意されている。以下に例を示す。

プログラム例３

```
public class Prog01_03
{
        public static void main(String[ ] args)
        {
                System.out.print("基本的な"+"Java プログラム");
                System.out.print("¥n 文字列を¥n 表示する¥n");
        }
}
```

出力例

```
基本的な Java プログラム
文字列を
表示する
```

例題は出来るだけ短い記述で済むようなものを中心に取り上げる方針である。このプログラムでは短い文字列を連結しているのであまり意味は無いが、文字列が長くなり、複数行に渡って記述しなくてはならない場合やこの後に習う変数を含めた表示の時には「+」が役立ってくる。

拡張表記

文字列の改行には「¥n」を用いた。見た目は２文字であるが、プログラム上では１文字で扱われる。この

ような文字を拡張表記といい、改行やタブといった出力に利用する。拡張表記には以下の例などがある。

¥b	後退	一文字前の位置に移動する
¥f	改ページ	改ページする
¥n	改行	改行する
¥r	復帰	現在の行の先頭に移動する
¥t	水平タブ	現在の行のタブ位置に移動する
¥’	シングルクォーテーション	シングルクォーテーションそのものを表示
¥”	ダブルクォーテーション	ダブルクォーテーションそのものを表示

コメントアウト

処理の説明の記述や開発段階での試しプログラムなど、最終的に実行される内容とは直接関係のない文を無視する方法がある。それをコメントアウトといい、以下のように行う。

「/*」と「*/」で無視したい文を囲む
「//」で記述した以降の行の文を無視する

「/*」と「*/」は複数行にわたってコメントアウトすることが可能である。以下のプログラム例では出力部分を全てコメントアウトしたので実行結果には何も表示されない。

プログラム例４

```
/* コメントアウトの例 */
public class Prog01_04
{
        public static void main(String[ ] args)
        {
                /* System.out.println("Java プログラムの練習"); */
                // System.out.println("文字列を表示する");
        }
}
```

出力例

[出力無し]

演習

・プログラム例１～４を作成すること。
・文字列や拡張表記を変更したりして、いろいろ試すこと。