Javaからはじめよう

Android プログラミング

AndroidStudio対応版

大津 真 [著] Otsu Makoto

Javaを知らない人のための入門書！

インプレス

はじめに

ご存じAndroidは、スマートフォンやタブレットなどモバイルデバイス向けに、Googleが無償で提供し、かつ自由に利用可能なオープンソースのOSです。現在ではAppleのiOSと双璧をなす存在といえるでしょう。

Androidアプリの開発には、かつては、フリーの統合開発環境「Eclipse」と「ADT（Android Developer Tools）」の組み合わせが標準的でしたが、2014年にGoogleがオリジナルの統合開発環境「Android Studio」の提供を開始しました。Eclipse向けのADTは2015年末に公式サポートを終了することがアナウンスされており、今後はAndroid Studioに一本化されるでしょう。Android StudioはJetBrains社が開発したIntelliJ IDEAをAndroid用に特化させたもので、Windows、Linux、OS X（Mac）上で使用可能です。もちろん、無償で提供されるので、好奇心と情熱さえあれば、誰もがAndroidアプリの開発に挑戦できます。

ただし、実際にAndroidアプリを作成できるようになるための道のりはそう平坦ではありません。まず、オブジェクト指向プログラミング言語であるJavaの知識が不可欠です。もちろん、AndroidのAPIに関する理解も不可欠です。そのため、初心者がいきなり、見栄えのよいサンプル主体のAndroidプログラミング解説書に挑戦しても、途中で挫折してしまうケースが少なくありません。

本書は、プログラミングの初心者を対象に、段階的にステップアップしながら、Androidプログラミングのための基礎知識をしっかりと学んでいくための学習書です。

最初の2つの章では、Androidの概要と、Javaプログラミングの基礎についてわかりやすく解説しています。Androidアプリのプログラミングをはじめるにあたって、Java言語のすべてを理解しておく必要はありません。たとえば、Androidでは独自のGUIが使用されるため、Javaの標準的GUIであるAWTやSWINGなどの知識は不用です。そのため、本書では、変数や制御構造、オブジェクトの取り扱いといったJava言語の基本事項に絞って説明しています。

Java言語にある程度慣れたところで、3章ではAndroid Studioのインストールと基本設定を行います。4章からは、Android Studioを使用したAndroidアプリの作成方法について基礎から解説します。サンプルもできるだけシンプルなものを用意し、ポイントを把握しやすいように配慮しています。

Androidには膨大な量のAPIが用意されていますので、本書だけでそのすべてを解説することは残念ながら不可能です。ただし、本書で得た基礎知識をベースにすれば、Web上に用意されているドキュメントや、他書の複雑なサンプルなども容易に理解できるようになるでしょう。

最後に、本書によって、読者のみなさんがAndroidプログラミングの奥深さ、おもしろさを実感していただき、本格的なAndroidアプリ作成の世界へ足を踏み入れるきっかけとなれば幸いです。

2015年8月　大津真

Javaからはじめよう
Androidプログラミング
[目次]

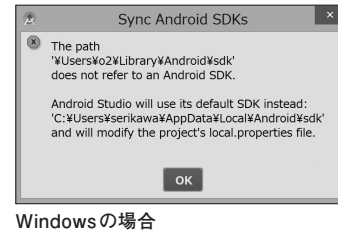
●サンプルプログラムについて

本書で解説したサンプルプログラムは、以下のURLからすべてダウンロードできます。
エミュレーターや実機での動作を確認しながら読み進めていってください。

http://book.impress.co.jp/books/1114101137

なお、Android Studioで本書のプロジェクトファイルを最初に開いたとき、図のようなアラートが表示されることがあります。

Macの場合



Windowsの場合

これはAndroid SDKのロケーションが、プロジェクト作成時と、現在Android Studioを実行中のシステムで異なるというメッセージです。「OK」ボタンをクリックするとSDKのロケーションが修正されます。

●キー表記

MacとWindowsのキー表記が異なる場合は、Mac/Windowsの順で併記しています。

【例】 return/Enter ← Macではreturnキー、WindowsではEnterキー
control/Ctrl + F11 ← Macではcontrol + F11キー、WindowsではCtrl + F11キー

●コマンドプロンプト

本書ではコマンドラインでの操作も行っています。コマンド入力行のプロンプト記号はMacの「$」で表記しています。Windowsの場合は、プロンプト記号を「>」に読み替えてください。

【例】

```
$ java RobotTest [return]
```

Macのターミナルのプロンプト記号

```
> java RobotTest [Enter]
```

Windowsのターミナルのプロンプト記号

Learning Java Programming For Android Development

CHAPTER 1

Androidプログラミングを始めるために

Androidは、検索最大手のGoogle社が2007年に満を持して発表した、スマートフォンやタブレットPCなどのモバイル端末向けプラットフォームです。本書最初のChapterでは、まずAndroidの概要と、開発に必要なソフトウエアなど、Androidプログラミングを行うための予備知識について解説します。そのあとで、Javaプログラムのコンパイルと実行方法について説明します。

Learning Java Programming
For Android Development

1-1 Androidの概要

Androidの世界へようこそ! 本書ではプログラミングの初心者を対象に、Androidアプリケーション開発の基礎について段階を追って説明していきます。はじめに、Androidの概要と、その基本的なアーキテクチャについて見てみましょう。

1-1-1 | Androidの特徴

Androidは、モバイル機器をターゲットとしたソフトウエア・プラットフォームです。2007年11月5日、Googleを中心とし、複数の通信キャリアや携帯メーカーなどによって発足した「Open Handset Alliance」（OHA）によって発表されました。OHAには、世界中のそうそうたる企業が名前を連ねています。日本ではNTTドコモ、KDDI、ソフトバンク（旧ソフトバンクモバイル）などがそのメンバーです。ご存じの通り、発表から数年で魅力的なAndroid搭載デバイスが相次いでリリースされたことにより、すさまじいスピードで普及しました。特に、スマートフォンの世界では、AndroidとiPhoneという二大勢力が、その代表として確固たる地位を築いています。

まずは、Androidの特徴について簡単にまとめておきましょう。

▶Androidはオープンソース

Androidが急成長した理由の1つに、Androidが誰もが無償で利用可能なオープンソースのモバイル端末向けOSであることが挙げられます。それはOSの利用にロイヤリティがかからないことを意味します。加えて、主要なソフトウエアが「Apache License 2.0※」というライセンスに基づいて公開されている点も重要です。Apache Licenseでは、ソフトウエアを改変した場合に、ソースコードを公開する必要がありません。従って、企業のノウハウが外部に漏れることを防げるわけです。

ただし、OSの基幹部分であるカーネルやそのライブラリなど一部ソフトウエアは、GNUプロジェクトによるGPL（GNU General Public License）というLinuxなどでおなじみのライセンスです。GPLではソフトウエアを改変した場合にはソースコードの公開が義務づけられています。

※ Webサーバソフトウエアとして有名な「Apache」を開発／保守するApacheソフトウエア財団 (ASF) によるソフトウエア向けライセンス規定。

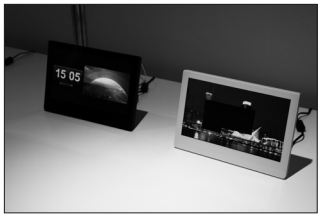
▶キャリアや端末に依存しないアプリケーションの開発が可能

Android以前は、キャリアやメーカー、さらには端末ごとに規格やAPIが異なり、アプリケーションの対応が大変でした。それがAndroidの登場により、基本的に世界中のAndroid端末上で同じように動作するアプリケーションの開発が可能になったわけです。

▶スマートフォンだけでない活用分野

Androidを搭載したデバイスとしては、現時点ではスマートフォンが主流ですが、タブレットPCやネットブックといったPCでの採用も増えています。さらにカーナビやテレビなどの家電製品などの組み込みOSとしても注目を集めています。

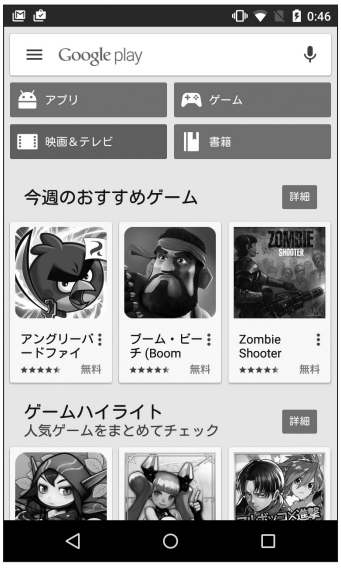
Androidを搭載したスマートフォンやタブレットPC


▶コンテンツの入手はGoogle Playから

アプリケーションや音楽、ビデオといったAndroid向けデジタルコンテンツ配信は、Googleが運営する**Google Play**（旧Android Market）というサイトを介して行われ、Android端末の「Google Play Store」アプリにより、有償もしくは無償でコンテンツをダウンロードできます。

Google Play

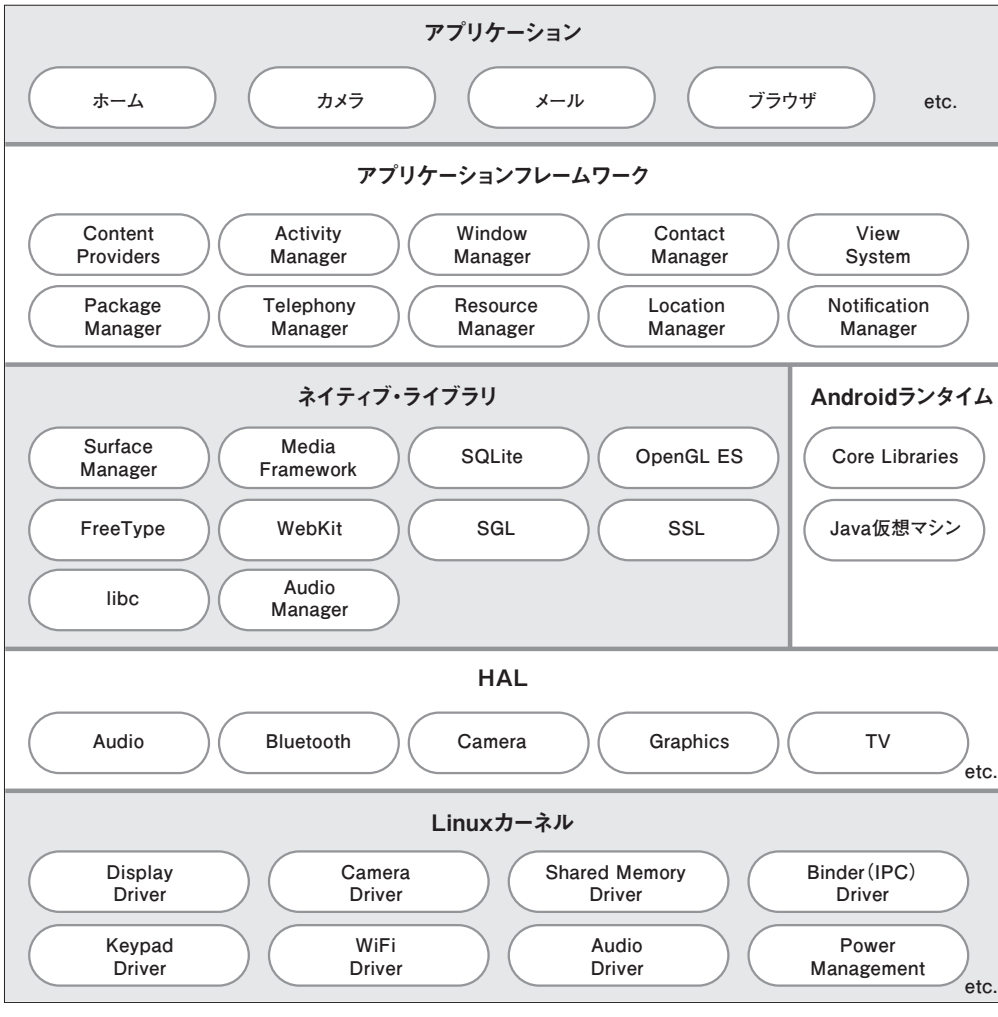


Google Playは、企業だけでなく一般ユーザーもアプリケーションを販売することが可能です。Androidのライバルであるiphone、iPadなどのiOSデバイス用ソフトウエアの配布サイトApp Storeは、Appleの審査が厳しいのに対して、Google Playの場合には、開発者登録を行えば誰でも短期間でアプリケーションを公開できるというメリットがあります。ただし、その分アプリケーションの安全性が問題となってきているのも確かです。ユーザーからのクレームにより、公序良俗に反する悪質なものはGoogleが削除することになっています。

1-1-2 | Androidのアーキテクチャ

Androidは、実際には完全にゼロから作り上げられたプラットフォームというわけではありません。オープンソースのUNIX互換OSの代表であるLinuxを基盤にした、さまざまなソフトウエア技術の集合体です。次に、Androidのアーキテクチャの階層構造を示します。

Androidのアーキテクチャ



各階層の概要について説明しておきましょう。

▶アプリケーション

最上位の階層には、Webブラウザやカレンダーといったユーザーが実際に使用する**Androidアプリケーション**が位置します。アプリケーションは基本的にJava言語で記述します。なお、個々のAndroidのアプリケーショ

ンには、**アクティビティ**と**サービス**という2つの要素があります。アクティビティとは簡単にいえば"Androidアプリケーションの1つの画面"です。パソコン用ソフトのウィンドウに相当するイメージです。それに対して、サービスはバックグラウンドで動作可能なプログラムです。たとえばアプリケーションのダウンロードなどは、サービスとして実行されるため、ダウンロード中に別の処理を行えるわけです。1つのアプリケーションは、アクティビティとサービスのどちらか一方、もしくは両方を含むことができます。

▶アプリケーションフレームワーク

近年のソフトウエア開発には**アプリケーションフレームワーク**（あるいは単に「**フレームワーク**」）が欠かせません。「フレームワーク」は日本語では「枠組み」といった意味ですが、アプリケーション構築のための、さまざまな機能をまとめたものです。実際にAndroidアプリケーションを開発する際には、これらのフレームワークに用意されている**API**※を利用することになります。たとえば、ユーザーインターフェースの基本となるのが、いわゆるGUIツールキットであるView Systemです。View Systemには、ボタン、テキストボックス、ラジオボタンといった基本的なGUI部品だけでなく、Webページを表示する「WebView」などの部品も用意されています。

※「Application Program Interface」の略。フレームワークやライブラリをアプリケーションから利用する際の入り口部分のこと。通常、関数やメソッドとして提供されています。

▶ネイティブ・ライブラリ

その下の階層には、CおよびC++言語で開発された、よりプリミティブな**ライブラリ**が位置します。C言語ライブラリである「libc」を基本に、軽量のリレーショナルデータベース「SQLite」や、2Dグラフィックエンジン「SGL」、3Dグラフィックエンジン「OpenGL ES」などが用意されています。さらには、Webブラウザのレンダリングエンジン「Web Kit」などもこの階層です。

▶Androidランタイム

Androidランタイムは、Javaで記述されたアプリケーションのための実行環境部分です。次節で説明するJava仮想マシン、および基本的なAPIを提供するコア・ライブラリで構成されています。

▶HAL

HAL（Hardware Abstraction Layer)は、ハードウエアを抽象化し容易にアクセスできるようにするためのレイヤーです。

▶Linuxカーネル

基本部分には、今やオープンソースの代表となったUNIX互換OS「**Linux**」のカーネルをモバイル向けにカスタマイズしたものを採用しています。

Learning Java Programming
For Android Development

1-2 Androidアプリケーションの開発について

Androidアプリケーションの開発言語には、オブジェクト指向言語として有名なJava言語が使用されます。ここでは、まずそのJava言語の概要について説明します。続いて、Androidアプリケーション開発に必要なソフトウエアをまとめておきましょう。

1-2-1 ｜開発言語はJava

Androidアプリケーションの開発言語には、現在もっとも普及しているオブジェクト指向言語の1つである**Java言語**が採用されています。

AndroidのライバルといえるiPhoneやiPadなどのiOSデバイスのアプリケーション開発では、一般的にはマイナーな**Swift**もしくは**Objective-C**という言語が使用されるため、その分、敷居が高くなります。その点、Java言語は、言語仕様も素直で、学校などでプログラミングの授業に採用されるケースも多く、またWeb上での情報や解説書も数多く存在しているため、初心者にも取り組みやすい言語といえるでしょう。

▶Java言語について

Javaは、サン・マイクロシステムズ社（当時）※によって開発が開始されたプログラミング言語で、最初のバージョンが登場したのは1995年です。当初は、Webブラウザ内で「Javaアプレット」と呼ばれるJavaで作成したプログラムを実行できる点がもっとも注目を集めました。ただし、現在では、Javaアプレットの得意分野であったインタラクティブなWebページは、Flashムービーさらには HTML5アプリケーションが主流となりました。そのため、今ではWebブラウザ上でのJavaアプリケーションの利用はあまり見かけません。その代わりに、Java Servletなどサーバサイド Javaと呼ばれる、インターネットサーバ側でのJavaを利用したWebアプリケーション構築のための技術が広く使用されるようになっています。

もちろん、Javaの利用分野はインターネットサーバ分野だけではありません。一般的なデスクトップ・アプリケーション、携帯アプリ、さらには家電製品などの組み込みシステムにもJavaが広く採用されています。

※2010年1月27日、Oracle社による買収が完了。

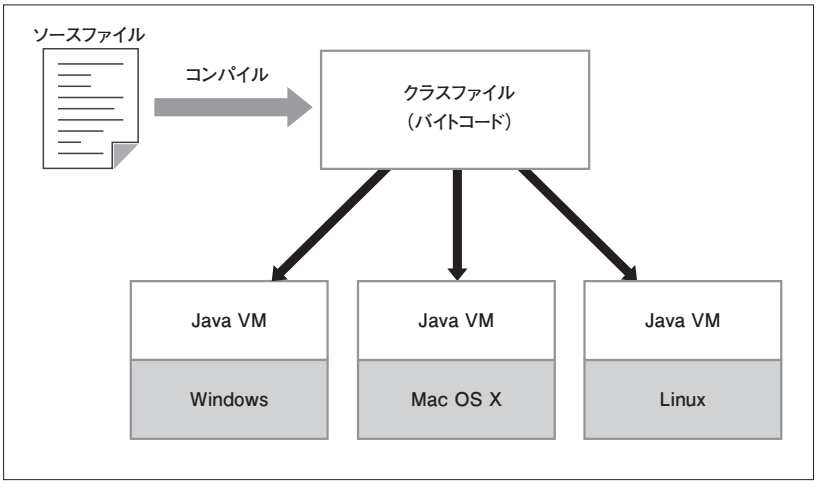
▶Javaプログラムを実行するJava仮想マシン

Java登場時のキャッチコピーの1つに、「Write Once Run Anywhere」（一度作ったらどこでも動く）というものがありました。そのことは、作成したJavaプログラムが、OSに依存せず、たとえばWindowsでもMac（OS

X)、さらにはLinuxでも同じように動くことを意味します。その秘密は、「**Java仮想マシン**（**Java VM**）」と呼ばれる仮想的なコンピューター環境を実現するソフトウエアにあります。

ここで、人間が読めるテキスト形式のプログラムのことを「**ソースファイル**」、CPUが実行可能な形式のファイルのことを「**オブジェクトファイル**」と呼びます。また、ソースファイルをオブジェクトファイルに変換する処理を「**コンパイル**」と呼びます。Javaのソースファイル（拡張子は「**.java**」）をコンパイルすると、「**バイトコード**（中間コード）」と呼ばれるオブジェクトファイルが生成されます。このバイトコードは、Java仮想マシンが実行可能なオブジェクトファイルです。Java仮想マシンは、バイトコードを読み込むと、それを、逐一現実のCPUのマシン語に変換しながら実行します。つまり、Java仮想マシンをOSごとに用意することによって、同じプログラムがWindows上でもMac上でも同じように動作するのです。バイトコードのファイルの拡張子は「**.class**」が使用されるため、「**クラスファイル**」とも呼ばれます。

Java仮想マシン



▶Android専用の仮想マシン

Androidアプリケーションの場合も、作成したプログラムは仮想マシン上で実行されます。ただし、Androidで使用される仮想マシンは標準的な「Java VM」ではありません。携帯電話のような低メモリの環境に最適化され、複数の仮想マシンが効率的に動作することを可能にした、Googleオリジナルの仮想マシンが使用されます。Android 4.4以前は「**Dalvik VM**（ダルビックVM）」という仮想マシンが使用されていました。Android 4.4以降では、あらかじめネイティブコードにコンパイルしておくことでより動作効率が向上した**ART（Android Runtime）VM**が試験的に利用可能になり、Android 5.0でそれが標準の仮想マシンとなりました。

なお、Dalvik VMおよびART VMともに、OracleのJava VMと互換性はありません。標準のJava VMで動作するクラスファイルをそのまま持ってきても動作しません。

NOTE 実際のAndroidアプリケーションは、apkファイル（拡張子「**.apk**」）の圧縮ファイルです。これはオブジェクトに加えて、アイコンやイメージ、レイアウトファイルといったリソースファイルをパッケージングして、ZIP形式で圧縮したものです。

▶Javaの3種類のエディション

現在、Javaは、Oracle社から次の3種類のエディションとして提供されています。

Javaの3つのエディション

エディション	説明
Java SE（Java Platform, Standard Edition）	デスクトップ・アプリケーションなど汎用的な用途に使用されるエディション
Java EE（Java Platform, Enterprise Edition）	Java SEに、クライアントサーバ型の大規模システムを開発するためのAPIを追加したエディション
Java ME（Java Platform, Micro Edition）	リソースの限られた、主に組み込みシステムでの使用を想定したエディション

Androidアプリケーションの開発で使用するエディションはJava SEです。

1-2-2 ｜ Javaはオブジェクト指向言語

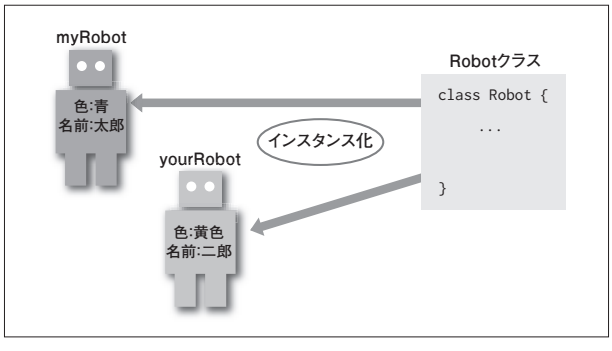
Javaは、**オブジェクト指向言語**に分類されるプログラミング言語です。オブジェクト指向言語では、プログラムの対象をオブジェクト（もの）としてとらえてプログラミングを行います。ソフトウエアを部品として再利用しやすいという利点によって、最近主流のプログラミング技法です。オブジェクト指向の概念やメリットを正確に理解するには、いろいろな予備知識が必要になりますので、ここではそのほんのさわりについて説明しましょう。

▶クラスとインスタンス

実際のJavaプログラムにおいて、オブジェクトの設計図となるのが「**クラス**」です。クラスから生成されたオブジェクトのことを「**インスタンス**」、またインスタンスを生成することを「**インスタンス化**」と呼びます。

たとえば、ロボットをモデル化したRobotクラスがあるとします。このRobotクラスをもとにインスタンス化することによって、myRobotやyourRobotといったインスタンスが生成できるわけです。個々のインスタンスは、個別のデータを変数として持つことができます。そのような、インスタンス固有の変数のことを「**インスタンス変数**」といいます。たとえば、Robotクラスを設計しようとする場合、まずは色や名前などのデータがインスタンス変数として思いつくでしょう。

クラスとインスタンス

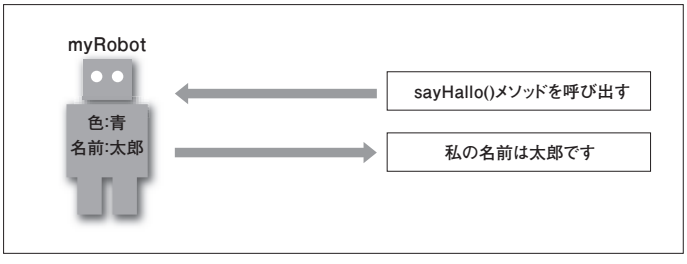


▶メソッドについて

クラスに用意されている、何らかの処理のことを「**メソッド**」と呼びます。C言語などをご存じの方は関数と同じようなものと考えてもよいでしょう。たとえばRobotクラスにsayHallo()※という挨拶をするメソッドが用意されているとしましょう。それぞれのインスタンスに対してsayHallo()メソッドを呼び出すと、「私の名前は～です」と表示するといったイメージです。

※ 本書では、それがメソッドであることを明確にするためにメソッド名のあとに「()」を記述することにします。

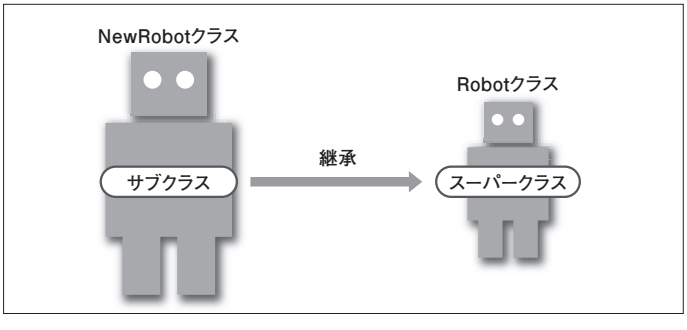
メソッド



▶クラスの資源を引き継ぐ継承

現在のようにプログラミング言語の分野においてJavaのようなオブジェクト指向言語が主流になってきた背景には、ソフトウエアを再利用しやすいという点があります。たとえばRobotクラスのデータやメソッドを引き継いで、機能を追加したNewRobotクラスを作成することができます。このことをクラスの「**継承**」といいます。あるクラスをもとにした別のクラスを記述する場合に、すべての変数やメソッドを記述し直すよりも、使える部分はそのまま使ってプログラムの生産性を高めようというのが継承の考え方です。このとき、継承元のクラスを「**スーパークラス**（親クラス）」、それを継承して作成したクラスを「**サブクラス**（子クラス）」と呼びます。

クラスの継承



1-2-3 | Androidアプリケーションの開発に必要なソフトウエア

Javaについて一般的な説明はこれくらいにしておいて、Androidアプリケーション開発の話題に戻りましょう。残念ながらAndroidアプリケーション作成のための環境を構築する作業はそれほど簡単ではありません。複数のソフトウエアを組み合わせて、少し面倒な設定を行う必要があります。具体的には、**JDK**、統合開発環境**Android Studio**、**Android SDK**といったソフトウエアをインストールして、開発環境を構築していく必要があります。これらのツールはすべて無料で、誰もが自由に利用できるものばかりです。

なお、開発プラットフォームとなるOSには、Windows 8/7/Vista/2003 (32 もしくは 64-bit)、Mac（v10.8.5以降のIntel Mac)、Linux（GNOMEもしくはKDEデスクトップ）が利用可能です。

Androidの開発環境

NOTE Android Studioの動作環境の詳細については、http://developer.android.com/sdk/index.htmlの「System Requirements」を参照してください。

▶JDK（Java開発キット）

Oracle社が提供するJava SEは、実行環境である**JRE**（Java Runtime Environment：Java実行環境）と、開発キットである**JDK**（Java Development Kit:Java開発キット）の2種類のパッケージとして配布されています。Androidアプリケーションの開発に必要なのは、JDKのほうです。JDKには実行環境であるJREが含まれ、さらにコマンドラインで動作するjavac（コンパイラ）、jar（アーカイバ）、javadoc（ドキュメント生成ツール）などの開発ツール群が同梱されています。

▶Android SDK

Android SDK（Android Software Development Kit）は、Googleによって提供されているAndroidアプリケーション作成のための開発キットです。Android SDKにはソフトウエア開発に必要なコマンドラインツールやライブラリといったツール群、さらにドキュメント、サンプルプログラムなどが含まれます。

なお、作成したアプリケーションの動作を実機なしでも確認できるように、「**AVD**（Android Virtual Device:Android仮想デバイス）」というエミュレータが用意されています。

エミュレータの実行画面

▶Android SDKのバージョンについて

Androidの進化のスピードはすさまじく、Android SDKのバージョンアップもかなり頻繁に行われています。バージョン1.5以降ではコードネームにアルファベットの「C」からのアルファベット順で始まるお菓子の名前が付けられています。また、バージョン番号とは別に、利用可能なAPIを「**APIレベル**」という数値で設定しています。数値が高いほど利用可能なAPIが増えていきます。

次に、主なAndroid SDKのバージョンと、APIレベルの対応を示します。

主なAndroid SDKのバージョン

バージョン	リリース	コードネーム	APIレベル
1.0	2008年9月	非公開	1
1.1	2009年2月	非公開	2
1.5	2009年4月	Cupcake	3
1.6	2009年9月	Donut	4
2.0	2009年10月	Eclair	5
2.0.1	2009年12月	Eclair	6
2.1	2010年1月	Eclair	7
2.2	2010年5月	Froyo	8
2.3	2010年12月	Gingerbread	9
2.3.3	2011年2月	Gingerbread	10
3.0	2011年2月	Honeycomb	11
3.1	2011年5月	Honeycomb	12
3.2	2011年7月	Honeycomb	13
4.0	2011年12月	Ice Cream Sandwich	15
4.1	2012年7月	Jelly Bean	16
4.2	2012年11月	Jelly Bean	17

主なAndroid SDKのバージョン（前ページの続き）

バージョン	リリース	コードネーム	APIレベル
4.3	2013年7月	Jelly Bean	18
4.4	2013年10月	KitKat	19
5.0	2014年11月	Lollipop	21
5.1	2015年3月	Lollipop	22

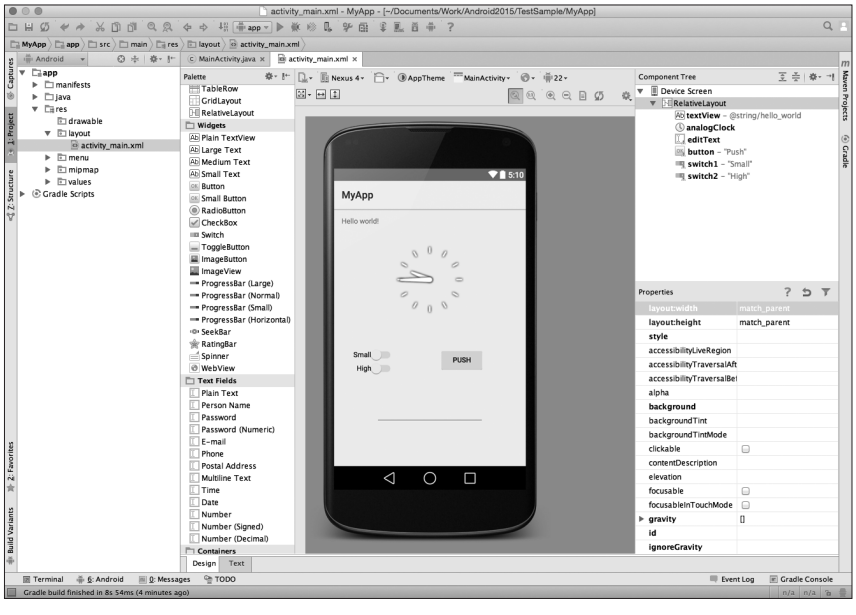
NOTE 3.0/3.1/3.2は主にタブレットPC向けに特化するために分岐したSDKです。2011年末に登場予定した4.0以降（Ice Cream Sandwich）でスマートフォン向けとタブレット向けの両方で使用できるように統合されました。

▶統合開発環境Android Studio

最低限JDKとAndroid SDKがあれば、Androidアプリケーションの開発が行えます。ただし、操作はすべてコマンドラインで行う必要があり、ある程度のアプリケーション規模になってくるとあまり現実的ではありません。そこで、登場するのが統合開発環境です。統合開発環境とは、従来はエディタやコンパイラ、デバッガなどに分割されていた開発ツールを統合し、さらに、プロジェクト管理機能やユーザーインターフェースの作成機能、ソースコードの補完機能などさまざまな便利機能を追加したアプリケーション開発用ソフトウエアです。

長らく、IBMによって開発されたオープンソースの統合開発環境Eclipse + ADT（Android Developer Tools）がAndroidの標準開発環境でした。現在では、Googleが提供する**Android Studio**が標準開発環境として使用されるようになってきています。Android StudioはGoogleがゼロから開発したものではありません。JetBrains社のIntelliJ IDEAという多言語対応の統合開発環境をベースに、Android開発に特化させたものです。本書でもAndroid Studioを基本に解説します。

Android Studio



Learning Java Programming
For Android Development

1-3 JDKのインストールと設定

この節では、Java開発キットであるJDK（Java Development Kit）のインストールと設定方法について説明しましょう。JDKは、現在Oracle社から、WindowsやMac、Solaris、Linuxに対応したものが、無償で提供されています。なお、開発環境を含まない実行環境のみのJRE（Java Runtime Environment）も用意されていますが、必ずJDKのほうをダウンロードするようにしてください。JDKにはJREが含まれています。

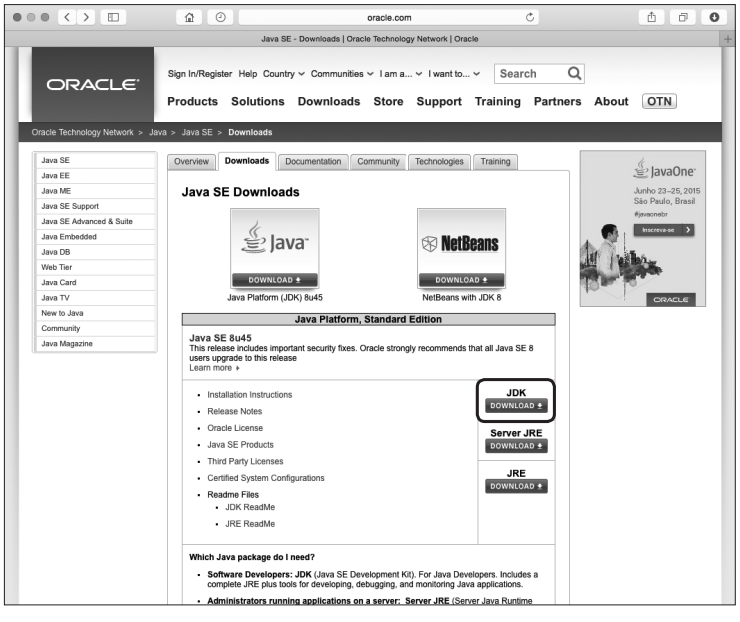
1-3-1 ｜ JDKをインストールする（Mac編）

Mac（OS X）を例にJava SEのJDKのダウンロード／インストール方法について説明します。本稿執筆時点でのJDKの最新バージョンは、「Java SE 8 8u45」になります（「u45」はアップデート番号が45であることを表します）。

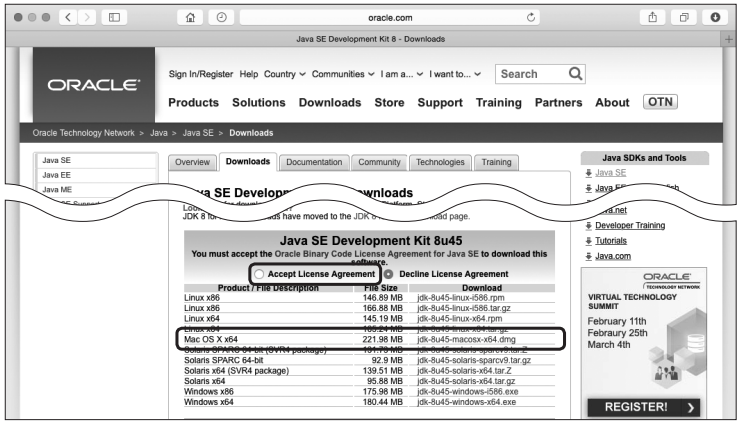
▶JDKをダウンロードする

❶ WebブラウザでOracle社のJava SEのダウンロードページにアクセスします。WebブラウザでOracle社のJava SEのダウンロードページにアクセスします。

http://www.oracle.com/technetwork/java/javase/downloads/index.html



❷「Java Platform, Standard Edition」の「JDK」→「DOWNLOAD」ボタンをクリックします。最新版のJDKの一覧が表示されるので、「Accept License Agreement」をチェックし、OSに応じたバージョンをダウンロードします。Macの場合には「Mac OS X x64」の右の「jdk-8u ～ -macosx-x64.dmg」を選択します（「～」部分にはアップデート番号を指定）。



▶JDKをインストールする

イメージファイルのダウンロードが完了したら、次のようにしてインストールします。

❶ ダウンロードしたイメージファイルをダブルクリックしてマウントします。インストーラのパッケージファイル「JDK 8 Update ～ .pkg」が表示されます。

❷ パッケージファイルをダブルクリックしてインストーラを起動し、指示に従ってインストールを行います。

NOTE インストール先は、「/Library/Java/JavaVirtualMachines/jdk1.8.0_～ .jdk」になります。

▶Javaコントロールパネル

インストールが完了すると「システム環境設定」に「Java」が追加され、クリックすると「**Javaコントロールパネル**」が表示されます。「更新」パネルで「更新を自動的にチェック」を選択しておくとよいでしょう。

「Javaコントロールパネル」

▶ターミナルでJavaのコマンドを確認する

次に「**ターミナル**」（「アプリケーション」→「ユーティリティ」フォルダ）を起動し、インストールが正しく行われたことを確認しましょう。

プロンプトに続いて「**java -version** return」とタイプします。するとJDKのバージョン情報が表示されます。

```
$ java -version return
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

また、Javaコンパイラである**javac**コマンドを引数なしで実行すると使用方法が表示されます。

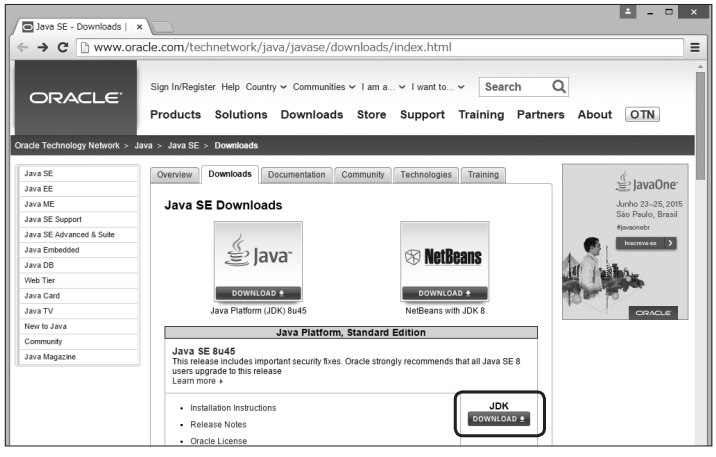
```
$ javac return
使用方法: javac <options> <source files>
使用可能なオプションには次のものがあります。
  -g                         すべてのデバッグ情報を生成する
  -g:none                    デバッグ情報を生成しない
    ～略～
```

1-3-2 ｜ JDKをインストールする（Windows編）

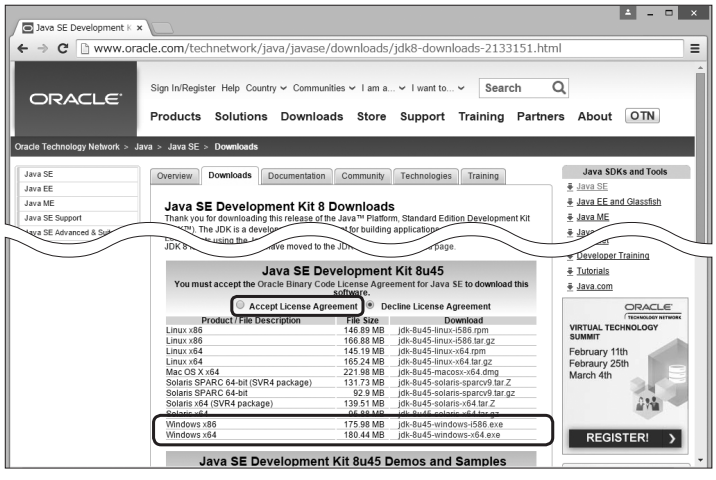
次に、Windows 8.1を例にJava SEのJDKのダウンロード／インストール方法について説明します。本稿執筆時点でのJava SEの最新バージョンは、Java SE 8 8u45になります（u45はアップデート番号が45であることを表します）。

▶JDKをダウンロードする

❶ WebブラウザでOracle社のJava SEのダウンロードページにアクセスします。
http://www.oracle.com/technetwork/java/javase/downloads/index.html



❷「Java Platform, Standard Edition」の「JDK」→「DOWNLOAD」ボタンをクリックします。最新版のJDKの一覧が表示されるので、「Accept License Agreement」をチェックし、OSに応じたバージョンをダウンロードします。32ビット環境のWindowsの場合には「Windows x86」→「jdk-8 ～ -windows-i586.exe」、64bit環境の場合には「Windows x64」→「jdk-8 ～ -windows-x64.exe」をクリックしてダウンロードします。



▶JDKをインストールする

ダウンロードしたexeファイルをダブルクリックするとインストーラが起動するので、その指示に従ってインストールを行います。インストール先などの設定はデフォルトのままでかまわないでしょう。

JDKをインストール

NOTE デフォルトのインストール先は「C:¥Program Files¥Java¥jdk1.8.0_～」になります。

▶環境変数Pathを設定する

Windowsの動作環境を設定する変数に「**環境変数**」があります。JDKのコマンド群を使用するためには、環境変数**Path**にコマンドの保存場所「C:¥Program Files¥Java¥jdk1.8.0_～¥bin」を設定する必要があります。

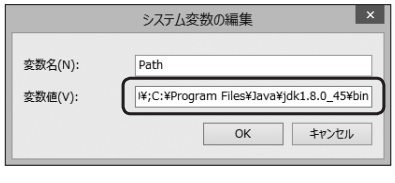
❶「コントロールパネル」を表示します。Windows 8.1の場合、「コントロールパネル」を表示するにはスタートメニューを右クリックしてメニューから「コントロールパネル」を選択します。ここで「システムとセキュリティ」→「システム」と進みます。

❷「システム」の画面が表示されたら、左のリストから「システムの詳細設定」をクリックします。

❸「システムのプロパティ」ダイアログの「詳細設定」パネルが表示されます。「環境変数」ボタンをクリックします。

❹「環境変数」ダイアログが表示されます。上部の「～のユーザー環境変数」は現在ログイン中のユーザーのための環境変数、「システム環境変数」はシステム全体に有効な環境変数です。通常は「システム環境変数」を設定すればよいでしょう。「システム環境変数」→「Path」を選択し、「編集」ボタンをクリックします。

❺「システム変数の編集」ダイアログで、「変数値」の最後に「;C:¥Program Files¥Java¥jdk1.8.0_ ～¥bin」（「～」部分にはアップデート番号を指定）を追加し、「OK」ボタンをクリックします。1つの環境変数に変数値が複数ある場合にはセミコロン「;」で区切って指定する点に注意してください。



NOTE Pathが存在しない場合には手順❹のダイアログで「新規」ボタンをクリックし、表示されるダイアログで「変数名」に「Path」を、「変数値」に「C:¥Program Files¥Java¥jdk1.8.0_～¥bin」を設定し、「OK」ボタンをクリックします。

▶コマンドプロンプトでJavaコマンド確認する

「**コマンドプロンプト**」（「スタート」ボタンを右クリックして「コマンドプロンプト」を選択）を起動して、JDKのインストールが正しく行われているかを確認しましょう。

まず、「echo %PATH% Enter」とタイプすると環境変数Pathの内容がセミコロン「;」で区切られて表示されます。

```
C:¥Users¥o2>echo %PATH% Enter
C:¥ProgramData¥Oracle¥Java¥javapath;C:¥WINDOWS¥system32;C:¥WINDOWS;C:¥WINDOWS¥System32¥Wbem
;C:¥WINDOWS¥System32¥WindowsPowerShell¥v1.0¥;C:¥Program Files (x86)¥QuickTime¥QTSystem¥;C:¥
Program Files¥Java¥jdk1.8.0_45¥bin
```

続いて、「**java -version** Enter」とタイプします。するとJDKのバージョン情報が表示されます。

```
C:¥Users¥o2> java -version Enter
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

また、Javaコンパイラである**javac**コマンドを引数なしで実行して使用方法が表示されることを確認します。

```
C:¥Users¥o2>javac  Enter
使用方法: javac <options> <source files>
使用可能なオプションには次のものがあります。
  -g                         すべてのデバッグ情報を生成する
  -g:none                    デバッグ情報を生成しない
  -g:{lines,vars,source}     いくつかのデバッグ情報のみを生成する
  -nowarn                    警告を発生させない
  -verbose                   コンパイラの動作についてメッセージを出力する
  -deprecation               非推奨のAPIが使用されているソースの場所を出力する
    ～略～
```

Learning Java Programming
For Android Development

1-4 はじめてのJavaプログラム

さて、JDKの準備ができたところで、実際に簡単なJavaプログラムを作成して実行してみることにしましょう。ソースファイルを作成し、javacコマンドでコンパイル、javaコマンドで実行という流れになります。

1-4-1 ｜Javaクラスとソースファイルの基礎知識

「1-2-2 Javaはオブジェクト指向言語」（P.014）では、クラスはオブジェクトの設計図と説明しましたが、クラスはJavaプログラムの最小単位でもあります。オブジェクトとしての機能を使用しないプログラムの場合でも、少なくとも1つのクラスを作成する必要があります。Javaクラスとそのソースファイルには次のような決まりがあります。

- クラス名の先頭は大文字で始める
- クラス名とファイル名は同じにする
- ソースファイルの拡張子は「.java」にする

たとえば「Hello」というクラスのためのソースファイルは「Hello.java」という名前で保存します。基本構造は次のようになります。

```
public class Hello {
    クラスの中身
}
```

クラスを指定しているのが1行目の「**class**」というキーワードで、この例では「Hello」がクラス名です。そのうしろの「{」から、最後の行の「}」までがクラスの中身になります。先頭の「**public**」というのがこのクラスが外部に公開されたパブリックなクラスであることを示しています。

なお、「{」と「}」で囲まれた範囲を「**ブロック**」と呼びます。ブロックはクラスの中身、メソッドの中身などいろいろなまとまりを記述する際に使われます。

1-4-2 | ソースファイル「Hello.java」を用意する

次に、画面に「Javaの世界へようこそ」という文字列を出力する命令を記述した、Javaのソースファイル「Hello.java」の中身を示します。みなさんもテキストエディタを使用して実際にファイルを作成してみるとよいでしょう。

Hello.java

```
public class Hello {
    /*
        はじめてのJavaプログラム                    ←1
    */

    public static void main(String[] args) {
        System.out.println("Javaの世界へようこそ");  ←2 mainメソッド
    }
}
```

▶コメントについて

ソースファイルに記述した注釈のことを「**コメント**」といいます。1の部分がコメントです。プログラムの内容をあとから自分で見直したり、ほかの人が理解しやすくしたりするためには、わかりやすいコメントは欠かせません。Javaのコメントには次の3種類があります。

```
// コメント
```

//以降行末までがコメントと見なされます。

```
/* コメント */
```

「/*」から「*/」までの範囲がコメントと見なされます。複数行に渡るコメントを記述できます。

```
/** コメント */
```

「/* コメント */」のバリエーションで、ドキュメンテーションコメントと呼ばれる種類のコメントです。javadocというドキュメント自動生成ツールにより抽出されるコメントです。

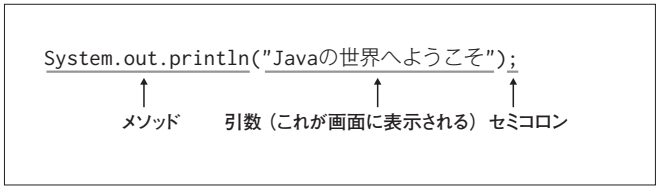
▶main()メソッド

Javaのクラスにおける処理は「**メソッド**」という単位で記述します。もっともシンプルなJavaプログラムでは、「**main**」という名前のメソッドのみが存在します（前ページ「Hello.java」の**2**の部分）。

```
public static void main(String[] args) {
    System.out.println("Javaの世界へようこそ");
}
```

mainの前に「public」「static」「void」という単語がスペースで区切られて記述されています。また、mainのあとには「(String[] args)」が記述されています。これらの意味についてはのちほど説明しますので、この段階では意味を気にせずにmain()メソッドには必ずそれらが必要だと覚えてください。

この例ではmain()メソッドの中身は次の1行だけです。



これが画面に「Javaの世界へようこそ」と表示するためのJavaの命令です。日本語や英語の文章と同じようにプログラミング言語では、1つの命令のことを「文」あるいは「**ステートメント**」といいます。Javaではステートメントの終わりにセミコロン「;」を必ず記述する必要があります。

この**System.out.println()**は、「( )」内に記述した文字列を画面に表示するメソッドです。つまり、main()メソッドから別のメソッドを呼び出していたわけです。

メソッドのあとの「( )」にはメソッドに渡す値を指定します。その値のことをメソッドの「**引数**」と呼びます。System.out.println()メソッドの場合、引数で渡された文字列を画面に表示するという処理を行うわけです。

1-4-3 ｜ Javaプログラムをコンパイルし実行する

続いて、Javaのソースプログラムをコンパイルして実行してみましょう。コンパイルを行うには**javac**コマンドを次の書式で実行します。

```
javac -encoding 文字コード ファイルのパス
```

「**-encoding**」はソースファイルの文字コードを指定するオプションです。たとえば、UTF-8の場合には「-encoding UTF-8 」、ShiftJISの場合には「-encoding sjis」となります。文字コードがUTF-8のソースファイル「Hello.java」をコンパイルするには次のようにします。

```
javac -encoding UTF-8 Hello.java return/Enter
```

NOTE Macでは、文字コードが「UTF-8」の場合に「-encoding 文字コード」を省略できます。Windowsでは、文字コードがShiftJISの場合に「-encoding 文字コード」を省略できます。

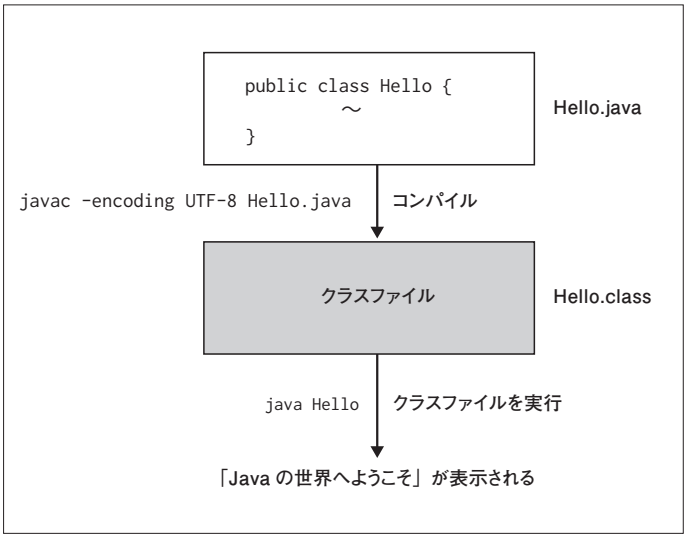
コンパイルが正常に終了すると「**クラスファイル**」と呼ばれるオブジェクトが作成されます。クラスファイルの拡張子は「**.class**」となります。これを実行するには次のように**java**コマンドを実行します。

```
java クラス名
```

引数にはクラスファイルの拡張子「.class」は付けません。たとえばカレントディレクトリの「Hello.class」を実行するには次のようにします。

```
java Hello return/Enter
```

ソースプログラムをコンパイルして実行



▶Macでコンパイル／実行する

次にMacのターミナル上で、Javaのソースファイルをコンパイルし、実行する場合の手順を示します。

① cdコマンドでソースファイルのあるフォルダに移動する
② lsコマンドでファイルの一覧を確認する
③ javacコマンドでコンパイルを行う

ここでは、「ドキュメント」→「JavaProgs」フォルダに「Hello.java」が文字コードUTF-8で保存されているものとして実行例を示します。

```
$ cd Documents/JavaProgs/ return      ← ディレクトリを移動する
$ ls return       ← ファイルの一覧を確認
Hello.java            ← ソースファイル
$ javac Hello.java return     ← javacコマンドでコンパイルを実行する
$ ls return       ← ファイルの一覧を確認
Hello.class Hello.java        ← クラスファイルが生成された
```

なお、コンパイルが正常に終了した場合には何もメッセージは表示されません。生成されたクラスファイル「Hello.class」を実行するには次のようにします。

```
$ java Hello return  ← javaコマンドで実行する
Javaの世界へようこそ  ← 実行結果
```

NOTE ホームディレクトリ以下の「ドキュメント」や「ダウンロード」などの日本語のフォルダ名は、「Documents」「Download」など英語のディレクトリ名がローカライズされたものです。ターミナルでは英語で扱います。

▶Windowsでコンパイル／実行する

次にWindowsのコマンドプロンプトで、Javaのソースファイルをコンパイルする場合の手順を示します。

① cdコマンドでソースファイルのあるフォルダに移動する
② dirコマンドでファイルの一覧を確認する
③ javacコマンドでコンパイルを行う

ここでは「ドキュメント」→「JavaProgs」フォルダに「Hello.java」が文字コード「UTF-8」で保存されているものとして、実行例を示します。

```
C:¥Users¥o2>cd Documents¥JavaProgs Enter     ← ディレクトリを移動する

C:¥Users¥o2¥Documents¥JavaProgs>dir Enter    ← ファイルの一覧を確認
～略～
 C:¥Users¥o2¥Documents¥JavaProgs のディレクトリ

2015/04/25  00:04    <DIR>          .
2015/04/25  00:04    <DIR>          ..
2015/04/24  23:09               179 Hello.java  ← ソースファイル
               1 個のファイル                 179 バイト
               2 個のディレクトリ  360,415,232,000 バイトの空き領域
C:¥Users¥o2¥Documents¥JavaProgs>javac -encoding UTF-8 Hello.java Enter  ← javacコマンドでコンパイルを実行する

C:¥Users¥o2¥Documents¥JavaProgs>dir Enter        ← ファイルの一覧を確認
～略～
 C:¥Users¥o2¥Documents¥JavaProgs のディレクトリ

2015/04/25  00:54    <DIR>          .
2015/04/25  00:54    <DIR>          ..
2015/04/25  00:54               432 Hello.class      ← 生成されたクラスファイル
2015/04/24  23:09               179 Hello.java
               2 個のファイル                 611 バイト
               2 個のディレクトリ  360,419,364,864 バイトの空き領域
```

なお、コンパイルが正常に終了した場合には何もメッセージは表示されません。生成されたクラスファイル「Hello.class」を実行するには次のようにします。

```
C:¥Users¥o2¥Documents¥JavaProgs>java Hello Enter     ← javaコマンドで実行する
Javaの世界へようこそ  ← 実行結果
```

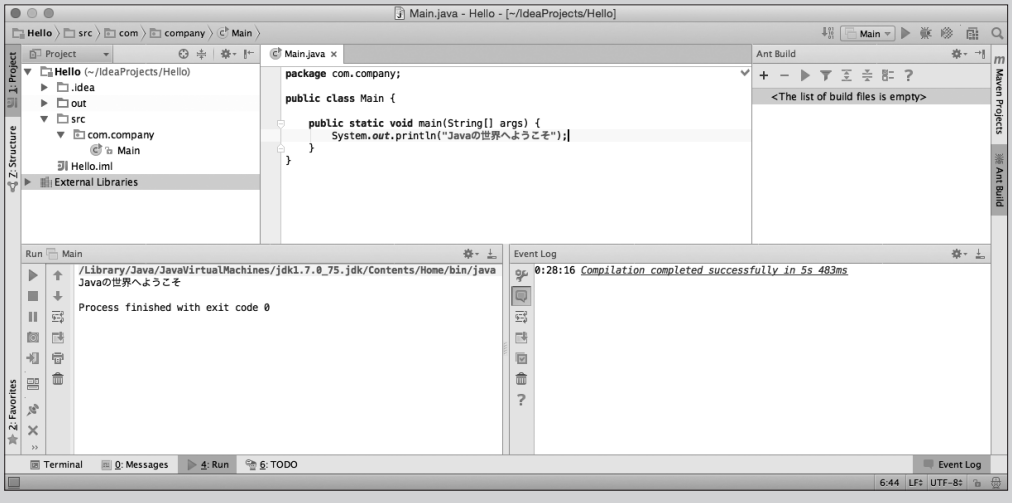
統合開発環境IntelliJ IDEA

コマンドラインの操作は苦手という方は、Javaプログラムの学習にも統合開発環境を使用するとよいでしょう。Javaアプリケーションの開発が可能なフリー統合開発環境としてはEclipseがポピュラーです。ただし、今後Android StudioでAndroidアプリケーションを開発するにはIntelliJ IDEAがオススメです。IntelliJ IDEAはAndroid Studioのベースとなったで、基本的なウィンドウの構成や使い勝手がAndroid Studioと同じだからです。

IntelliJ IDEAには、有料のUltimate Editionと、無料のCommunity Editionがありますが、Java言語の学習にはCommunity Editionで十分です。次のアドレスからダウンロードできます。

https://www.jetbrains.com/idea/

IntelliJ IDEAの実行画面



Learning Java Programming For Android Development

CHAPTER 2

Javaプログラミングの基礎を確認する

このChapterでは、Java言語の概要について駆け足で見ていきます。もちろん、ここだけでJavaのすべてを説明することはできませんので、Androidアプリケーションを開発するために必要な基本事項に絞って解説します。ざっと、目を通したあとは、あとから必要に応じて参照するといった読み方でもかまいません。

Learning Java Programming
For Android Development

2-1 変数と演算について

この節では、数や文字といったJava言語に用意されている基本データ型の値の取り扱いと、基本的な演算子を使用した数値の計算方法について説明します。さらに、さまざまな算術演算を扱うクラスであるMathクラスについても説明します。

2-1-1 ｜基本データ型について

Javaはオブジェクト指向言語の代表のような存在ですが、すべてのデータがオブジェクトというわけではありません。数や文字などのシンプルなデータは「**基本データ型**（プリミティブ型）」として扱います。足し算や引き算などの単純な計算を行うには、単純な数値のほうが使い勝手がよいからです。それ以外のデータは**オブジェクト型**となります。たとえば文字列はStringクラスのインスタンス、つまりオブジェクト型です。

▶変数の宣言と値の代入

プログラミング言語では、値を入れておく箱である「**変数**」の取り扱いが重要です。変数を使うためには、次のような書式で、あらかじめ変数を宣言しておく必要があります。

```
データ型 変数名;
```

たとえば、整数を表す基本データ型に**int**型があります。int型の変数として「age」を宣言するには次のようにします。

```
int age;
```

以上で変数「age」が使用できるようになります。変数「age」に値として「30」を代入するには次のようにします。

```
age = 30;
```

なお、次のようにすると宣言と同時に値を代入することもできます。

```
int age = 30;
```

▶整数型

続いて、Javaの基本データ型の種類を説明していきましょう。まずは、**整数を扱うデータ型**からです。扱えるデータのサイズに応じて、**byte型**、**short型**、**int型**、**long型**の4種類が用意されています。

整数型

型	サイズ	範囲
byte	1バイト（8ビット）	-128～127
short	2バイト（16ビット）	-32,768～32,767
int	4バイト（32ビット）	-2,147,483,648～2,147,483,647
long	8バイト（64ビット）	-9,223,372,036,854,775,808～9,223,372,036,854,775,807

▶浮動小数点型

「3.14」や「0.998」などの小数を管理するには「**浮動小数点形式**」というデータ型を使用します。Javaでは浮動小数点形式の値のためのデータ型としては**float型**と**double型**の2種類が用意されています。整数型のデータ型と同様に、float型とdouble型の違いはデータサイズです。float型のデータは4バイト、double型のデータは8バイト必要になります。

浮動小数点型

型	サイズ
float	4バイト（32ビット）
double	8バイト（64ビット）

デフォルトはdouble型です。「3.91」のように小数をそのまま記述した場合にはdouble型と見なされます。明示的にdouble型にするには最後に「**d**」または「**D**」を記述します。「**f**」または「**F**」を付けるとfloat型と見なされます。

```
double d1 = 3.91;   ← double型
double d2 = 1.45d;  ← double型
float f1 = 1.54f;   ← float型
```

NOTE 「=」の右辺の「3.91」や「1.45d」のようにソースコード内に直接記述した値そのもののことを「**リテラル**」といいます。