Java
逆引きハンドブック

株式会社エイベル◆著

Webアプリケーションや企業の大規模なシステム開発、組み込みなど、さまざまな場面で使われているJavaを集大成!

逆引きリファレンスの
決定版!

Java 6/7 対応!

24時間無料でサンプルデータをダウンロードできます。

C&R研究所

Java
逆引きハンドブック

株式会社エイベル◆著

Java
6/7
対応!

C&R研究所

■権利について

- 本書に記述されている社名・製品名などは、一般に各社の商標または登録商標です。
- 本書では™、©、®は割愛しています。

■本書の内容について

- 本書は著者･編集者が実際に操作した結果を慎重に検討し、著述･編集しています。ただし、本書の記述内容に関わる運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- 本書で紹介しているコードの実行結果などは、環境によって異なる場合がございますので、あらかじめ、ご了承ください。
- 本書の内容は、2012年9月現在の情報を基に記述しています。

■サンプルについて

- 本書で紹介しているサンプルは、C&R研究所のホームページ(http://www.c-r.com)からダウンロードすることができます。ダウンロード方法については、5ページを参照してください。
- サンプルデータの動作などについては、著者・編集者が慎重に確認しております。ただし、サンプルデータの運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- サンプルデータの著作権は、著者及びC&R研究所が所有します。許可なく配布・販売することは堅く禁止します。

●本書の内容についてのお問い合わせについて

この度はC&R研究所の書籍をお買いあげいただきましてありがとうございます。本書の内容に関するお問い合わせは、FAXまたは郵送で「書名」「該当するページ番号」「返信先」を必ず明記の上、次の宛先までお送りください。お電話や電子メール、または本書の内容とは直接的に関係のない事柄に関するご質問にはお答えできませんので、あらかじめご了承ください。

〒950-3122 新潟県新潟市北区西名目所4083-6　株式会社 C&R研究所　編集部
FAX 025-258-2801
「Java逆引きハンドブック」サポート係

PROLOGUE

Javaは数ある開発言語の中でも、多くのプログラマから支持されている人気の高い開発言語です。実行するシステム環境に依存しないことや、標準ライブラリが充実していることから、携帯端末からスーパーコンピュータまで、ゲームから大規模システムアプリケーションまでと、幅広くJavaが利用されています。

本書は、Javaでコーディングをしている際に「○○をするにはどうしたらよいか」という問題の解決に役立つ解説とサンプルコードを収録した逆引き辞典です。効率的なコーディングやアルゴリズムといった上級者向けの内容には触れていませんが、とりあえずJavaでどんなことができるかという点に主眼を置いています。

CHAPTER 01では、Javaの基礎知識と統合開発環境であるEclipseのインストール方法と使い方を解説しています。CHAPTER 02ではJavaの言語仕様について解説しているため、過去に少しでもプログラミングの経験があれば、Java初心者の方でも理解できる構成になっています。

CHAPTER 03以降では、「やりたいこと」とそれを達成するためのサンプルコードを、解説を交えて紹介しています。文字列や日付・時刻、ファイル操作、配列、リスト、マップ、数学処理などのJavaの基本的な処理をはじめ、ネットワーク通信、Webアプリケーションの作成、GUI、XML文書の操作、スレッドの作成と実行、データベースへのアクセス、ログ出力など、広い範囲を扱っています。

本書の内容は基礎的なことばかりですが、その分、さまざまな場面で参照することができるはずです。本書をお手元に置いていただき、Javaのプログラミングで困ったときには、本書を開いてみてください。問題解決の手助けとなるヒントが見つかるはずです。

最後に、本書の執筆・制作にあたって、お世話になったスタッフの皆様にに心から感謝申し上げます。そして、本書がJavaでプログラミングを行う読者の皆様に少しでもお役に立てれば幸いです。

2012年10月

株式会社エイベル　第一システムソリューション部
加藤　政洋

本書について

開発環境について

本書のサンプルコードはJava6を対象としています。Java7で追加・変更された部分についてはコラムで解説します。

基本的に標準APIについて解説していますが、メールの送受信に使うJavaMailなどの外部APIについても一部、解説しています。また、Javaはサーバサイドで使われることが多いため、Tomcatを使ったWebアプリケーションで使われるコードも解説しています。

Javaを動かす環境として、Windows 7を想定しています。環境により異なる部分がある場合は、その都度、他の環境での解説を記述します。Windows VistaやWindows XPを合わせてWindows系と記述します。Mac OS XやLinuxを合わせてUnix系と記述します。

ファイルの読み込みなどで、対象となるファイルを「C:¥work¥sample.txt」などと記述しますが、Unix系の環境を使っている場合は適宜「/opt/work/sample.txt」など、自分の環境に合わせて読み替えてください。パスの設定も同様です。

なお、本書のサンプルコードは Windows 7、JDK 1.7.0の環境で実行を確認しています。

コードについて

内容を簡潔にするために、例外処理とパッケージを省略しています。また、コードを解説するためのコメントは記述していますが、Javadocコメントは記述していません。

本書の表記方法

本書の表記についての注意点は、次のようになります。

▶クラス・インタフェース・メソッド・インスタンスの表記について

クラス・インタフェース名は、「」(カギカッコ)で囲んでいます。『「Cookie」クラス～』と記述した場合はクラスを表しますが、単にCookieと記述した場合は一般名詞を表します。

メソッドは「クラス名#メソッド名」、または、「クラス名.メソッド名」と記述します。前者はインスタンスメソッドを表します(例:「Object#toString」メソッドは～)。後者は「static」メソッドを表します(例:「Integer.valueOf」メソッドは～)。

インスタンスはどのクラスのインスタンスかわかるように、『「クラス名」インスタンス～』のように記述します(例:「String」インスタンスを～)。インタフェースを実装したクラスのインスタンスの場合は、『「インタフェース名」インスタンス～』と記述します(例:「List」インスタンスを～)。

一般的に使われるクラス、インタフェースに関しては、日本語で記述する場合もあります。

日本語名	クラス/インタフェース名
文字列	String
リスト	List
マップ	Map

▶ サンプルコードの中の「\」(バックスラッシュ)と「¥」(円記号)について

環境によっては、「\」(バックスラッシュ)は「¥」(円記号)で表示されます。本書の誌面に掲載しているサンプルコードでは「\」で表記していますので、必要に応じて読み替えてください。なお、コマンドプロンプトへの入力などは「¥」で表記しています。詳しくは、45ページを参照してください。

▶ サンプルコードの中の▼について

本書に記載したサンプルコードは、誌面の都合上、1つのサンプルコードがページをまたがって記載されていることがあります。その場合は▼の記号で、1つのコードであることを表しています。

サンプルファイルのダウンロードについて

本書のサンプルデータは、C&R研究所のホームページからダウンロードすることができます。本書のサンプルを入手するには、次のように操作します。

❶「http://www.c-r.com/」にアクセスします。

❷ トップページ左上の「商品検索」欄に「112-2」と入力し、[検索] ボタンをクリックします。

❸ 検索結果が表示されるので、本書の書名のリンクをクリックします。

❹ 書籍詳細ページが表示されるので、[サンプルデータダウンロード] ボタンをクリックします。

❺ 下記の「ユーザー名」と「パスワード」を入力し、ダウンロードページにアクセスします。

❻「サンプルデータ」のリンク先のファイルをダウンロードし、保存します。

サンプルのダウンロードに必要な ユーザー名とパスワード

ユーザー名 **javah**

パスワード **a1sc4**

※ユーザー名・パスワードは、半角英数字で入力してください。また、「J」と「j」や「K」と「k」などの大文字と小文字の違いもありますので、よく確認して入力してください。

サンプルコードの利用方法

サンプルファイルは、CHAPTERごとのフォルダの中に、項目番号のフォルダに分かれています。サンプルはZIP形式で圧縮してありますので、解凍してお使いください。

それぞれのフォルダ内には、Javaファイルなどが保存されています。

なお、サンプルの実行方法については、55ページを参照してください。

CONTENTS

CHAPTER 01 Javaの基礎知識

001 Javaとは …… 34
COLUMN■Javaの用途

002 オブジェクト指向について …… 36
COLUMN■デザインパターン

003 Java開発キットの入手方法について …… 38

004 統合開発環境Eclipseのインストールについて …… 43
COLUMN■「\」と「¥」について

005 はじめてのプログラミング …… 46
COLUMN■ダウンロードしたサンプルファイルをEclipseのプロジェクトに追加するには
COLUMN■ダウンロードしたサンプルファイルを「javac」コマンドでコンパイルするには

CHAPTER 02 Javaの文法

006 ブロックについて …… 58

007 コメントについて …… 60

008 予約語と識別子について …… 61
COLUMN■「const」と「goto」

009 変数について …… 63

010 代入について …… 65

011 キャストについて …… 70
COLUMN■「ClassCastException」例外

012 定数について …… 72
COLUMN■定数とならないケース

013 リテラルについて …… 74
COLUMN■Java7での新しいリテラル表記
COLUMN■補助文字

014 演算子について …… 77
COLUMN■定数の計算式

015 「if」文について …… 84

016 「switch」文について …… 86
COLUMN■Stringによる「switch」文

017 「for」文について …… 90

018 「while」文について …… 94

019 「do」文について …… 96
COLUMN■デバッグ情報の埋め込み
020 「break」文について …… 98
021 「continue」文について …… 101
022 「try-catch-finally」文について …… 104
COLUMN■Java7での複数例外のキャッチ
COLUMN■Java7での例外の再スロー
023 パッケージについて …… 112
024 「import」文について …… 113
025 クラス修飾子について …… 116
026 クラスについて …… 117
027 インスタンスの作成について …… 118
028 内部クラスについて …… 119
029 抽象クラスについて …… 123
030 コンストラクタについて …… 125
031 初期化ブロック、static初期化ブロックについて …… 127
032 フィールド修飾子について …… 132
033 フィールドについて …… 133
034 メソッド修飾子について …… 137
035 メソッドについて …… 139
036 「return」文について …… 145
037 共変戻り値について …… 147
038 可変長パラメータのメソッドについて …… 149
039 抽象メソッドについて …… 152
040 匿名クラスについて …… 153
041 インタフェースについて …… 156
COLUMN■目印としてのインタフェース
042 インタフェース修飾子について …… 158
043 継承と実装について …… 159
044 オーバーライドについて …… 163
COLUMN■staticメソッドを継承したときの注意点
045 オーバーロードについて …… 167
COLUMN■オーバーロードの制約
046 enumについて …… 169

047 例外について……173
048 ジェネリックスについて……177
COLUMN■Java7で使えるダイアモンドオペレータ
049 オートボクシング/オートアンボクシングについて……184
050 アノテーションについて……186
COLUMN■Java7で追加されたアノテーション
051 Javadocについて……190
052 Jarファイルの作成について……200
COLUMN■Jarファイルの形式
053 マニフェストファイルの書き方について……204

CHAPTER 03 クラスとインスタンス

054 クラスとインスタンスについて……208
COLUMN■リフレクション使用時の例外処理
055 インスタンスがどのクラスかを調べる……210
ONEPOINT■インスタンスがどのクラスかを調べるには「Object#getClass」メソッドを使用する
056 インスタンスのクラス名を取得する……212
ONEPOINT■インスタンスのクラス名を取得するには「Class#getName」メソッドを使用する
057 インスタンスが特定のクラスを継承しているか調べる……213
ONEPOINT■インスタンスが特定のクラスを継承しているか調べるには「instanceof」演算子を使用する
058 インスタンスが特定のインタフェースを実装しているか調べる……216
ONEPOINT■インスタンスが特定のインタフェースを実装しているか調べるには「instanceof」演算子を使用する
059 ある変数にインスタンスが代入可能か調べる……218
ONEPOINT■ある変数にインスタンスが代入可能か調べるには「Class#isAssignableFrom」メソッドを使用する
060 スーパークラスが持つメソッドやフィールドにアクセスする……220
ONEPOINT■スーパークラスが持つメソッドやフィールドにアクセスするには「super」キーワードを使用する
061 インスタンス自身のメソッドやフィールドに明示的にアクセスする……222
ONEPOINT■インスタンス自身のメソッドやフィールドに明示的にアクセスするには「this」キーワードを使用する
062 コンストラクタ内で他のコンストラクタを呼び出す……225
ONEPOINT■コンストラクタ内で他のコンストラクタを呼び出すには「super()」「this()」を呼び出す

063 フィールドの一覧を取得する …… 228
ONEPOINT■フィールドの一覧を取得するには「Class#getFields」メソッドや「Class#getDeclaredFields」メソッドを使用する

064 未知のインスタンスのフィールドにアクセスする …… 230
ONEPOINT■未知のインスタンスのフィールドにアクセスするには「Field」クラスを使用する

065 メソッドの一覧を取得する …… 232
ONEPOINT■メソッドの一覧を取得するには「Class#getMethods」メソッドや「Class#getDeclaredMethods」メソッドを使用する

066 未知のインスタンスのメソッドを呼び出す …… 235
ONEPOINT■未知のインスタンスのメソッドを呼び出すには「Method」クラスを使用する

067 コンストラクタの一覧を取得する …… 237
ONEPOINT■コンストラクタの一覧を取得するには「Class#getConstructor」「Class#getDeclaredConstructor」メソッドを使用する

068 Classからインスタンスを作成する …… 239
ONEPOINT■Classからインスタンスを作成するには「Constructor」クラスを使用する

069 クラス名からClassを取得する …… 241
ONEPOINT■クラス名からClassを取得するには「Class.forName」メソッドを使用する

070 インスタンスが破棄されるときに特定の処理を実行する …… 242
ONEPOINT■インスタンスが破棄されるときに特定の処理を実行するには「finalize」メソッドを実装する

071 グローバル変数を宣言する …… 244
ONEPOINT■グローバル変数を宣言するには「public static フィールド」として宣言する

072 インスタンスをコピーする …… 246
ONEPOINT■インスタンスをコピーするには「clone」メソッドを使用する

073 インスタンスをコピー可能にする …… 248
ONEPOINT■インスタンスをコピー可能にするには「Cloneable」インタフェースを実装して「Object#clone」メソッドをオーバーライドする
COLUMN■「シャローコピー」と「ディープコピー」
COLUMN■「clone」メソッドではコンストラクタが呼び出されない

074 クラス、メソッド、フィールドへのアクセスを制限する …… 252
ONEPOINT■クラス、メソッド、フィールドへのアクセスを制限するには「protected」修飾子や「private」修飾子を使用する

075 複数のインスタンスが作成されないように制御する …… 255
ONEPOINT■複数のインスタンスが作成されないように制御するには「インスタンス作成用のstaticメソッド」を実装する

076 クラスやメソッドを継承させないようにする ……257
ONEPOINT ■クラスやメソッドを継承させないようにするには「final」修飾子を使用する

077 メソッドやフィールドの使用時に警告を出す ……259
ONEPOINT ■メソッドやフィールドの使用時に警告を出すには「@Deprecated」アノテーションを指定する

CHAPTER 04 文字列

078 文字列について ……262

079 文字列を作成する ……263
ONEPOINT ■文字列を作成するには「文字列リテラル」を使用する
COLUMN ■「String」クラスのインスタンスを作成することでも文字列を作成可能
COLUMN ■空文字について

080 文字列を連結する ……265
ONEPOINT ■文字列を連結するには「+」演算子を使用するか「StringBuilder」クラスを使用する
COLUMN ■値を変更できないインスタンス
COLUMN ■「StringBuilder」クラスと「StringBuffer」クラスの違い
COLUMN ■「+」演算子はコンパイル時に最適化される

081 文字列の長さを調べる ……271
ONEPOINT ■文字列の長さを調べるには「String#length」メソッドを使用する

082 部分文字列を抜き出す ……272
ONEPOINT ■部分文字列を抜き出すには「String#substring」メソッドを使用する
COLUMN ■Unicodeの補助文字について

083 インスタンスの文字表現を取得する ……275
ONEPOINT ■インスタンスの文字表現を取得するには「Object#toString」メソッドを使用する
COLUMN ■暗黙的に「Object#toString」が呼び出されるケース

084 文字列にフォーマットする ……277
ONEPOINT ■文字列にフォーマットするには「MessageFormat」クラスや「String.format」メソッドを使用する

085 文字列を比較する ……280
ONEPOINT ■文字列を比較するには「String#equals」メソッドを使用する

086 StringとStringBuilderとで比較する ……282
ONEPOINT ■「String」インスタンスと「StringBuilder」インスタンスを比較するには「String#contentEquals」メソッドを使用する
COLUMN ■StringBuilder同士は直接、「equals」メソッドで比較できない

087 大文字・小文字を区別せずに文字列を比較する ……284
ONEPOINT ■大文字・小文字を区別せずに文字列を比較するには「String#equalsIgnoreCase」メソッドを使用する

088 文字列内に特定の文字列が含まれるか調べる ……286
ONEPOINT■文字列内に特定の文字列が含まれるかを調べるには「String#indexOf」メソッドを使用する
COLUMN■文字列内に特定の文字列が含まれるかを末尾から調べるには

089 文字列が正規表現とマッチするか調べる ……288
ONEPOINT■文字列が正規表現とマッチするか調べるには「String#matches」メソッドを使用する
COLUMN■正規表現について

090 文字列を特定の文字または文字列で分割する ……292
ONEPOINT■文字列を特定の文字や文字列で分割するには「String#split」メソッドを使用する
COLUMN■Java APIの非推奨クラスやメソッドについて

091 文字列を分割してプリミティブ型のデータへ変換する ……295
ONEPOINT■文字列を分割してプリミティブ型のデータへ変換するには「Scanner」クラスを使用する

092 文字列から前後の空白を取り除く ……297
ONEPOINT■文字列から前後の空白を取り除くには「String#trim」メソッドを使用する

093 文字列を数値に変換する ……298
ONEPOINT■文字列を数値に変換するには「Long.parseLong」メソッドや「Integer.parseInt」メソッドを使用する
COLUMN■書式を指定して文字列を数値に変換するには
COLUMN■基数を指定して文字列を数値に変換するには
COLUMN■基数とは
COLUMN■Java7での「Integer.parseInt」メソッドの変更点

094 数値を文字列に変換する ……301
ONEPOINT■数値を文字列に変換するには「Long.toString」メソッドや「Integer.toString」メソッドを使用する
COLUMN■書式を指定して数値を文字列に変換するには
COLUMN■基数を指定して数値を文字列に変換するには

095 文字列をバイト配列に変換する ……303
ONEPOINT■文字列をバイト配列に変換するには「String#getBytes」メソッドを使用する

096 バイト配列を文字列に変換する ……309
ONEPOINT■バイト配列を文字列に変換するには「String」のコンストラクタを使用する

097 アルファベットの大文字・小文字を変換する ……313
ONEPOINT■アルファベットの大文字に変換するには「String#toUpperCase」メソッド、小文字に変換するには「String#toLowerCase」メソッドを使用する

098 文字列をURLエンコードする ……315
ONEPOINT■文字列をURLエンコードするには「URLEncoder.encode」メソッドを使用する

099 文字列をURLデコードする ……316
ONEPOINT ■文字列をURLデコードするには「URLDecoder.decode」メソッドを使用する

100 文字列を置換する ……317
ONEPOINT ■文字列を置換するには「String#replace」メソッドを使用する

101 正規表現を使って文字列を置換する ……318
ONEPOINT ■正規表現を使って文字列を置換するには「String#replaceAll」「String#replaceFirst」メソッドを使用する

102 プロパティファイルから文字列を読み込む ……320
ONEPOINT ■プロパティファイルから文字列を読み込むには「Properties」クラスを使用する

103 ロケール別のリソースファイルから文字列を読み込む ……323
ONEPOINT ■ロケール別のリソースファイルから文字列を読み込むには「ResourceBundle」クラスを使用する
COLUMN ■sample_ja.propertiesの実際の内容
COLUMN ■「native2ascii」コマンドのオプション

104 文字の種類を判別する ……330
ONEPOINT ■文字の種類を判別するには「Character.UnicodeBlock.of」メソッドを使用する

105 文字列が特定の文字列で始まるか調べる ……331
ONEPOINT ■文字列が特定の文字列で始まるか調べるには「String#startsWith」メソッドを使用する

106 文字列が特定の文字列で終わるか調べる ……333
ONEPOINT ■文字列が特定の文字列で終わるか調べるには「String#endsWith」メソッドを使用する

CHAPTER 05 日付と時刻

107 日付と時刻について ……336

108 現在日時を取得する ……337
ONEPOINT ■現在日時を取得するには「Date」クラスのインスタンスを作成するか「System.currentTimeMillis」メソッドを呼び出す

109 特定の日付の「Date」インスタンスを取得する ……338
ONEPOINT ■特定の日付の「Date」インスタンスを取得するには「Calendar」クラスを使用する

110 日付の時刻部分を切り捨てる ……340
ONEPOINT ■日付の時刻部分を切り捨てるには「Calendar」クラスを使用する

111 日時の計算を行う ……341
ONEPOINT ■日時の計算を行うには「Calendar#add」メソッドを使用する

１１２　時間の単位を変換する……342
ONEPOINT■時間の単位を変換するには「TimeUnit#convert」メソッドを使用する
COLUMN■Date型とCalendar型、timeミリ秒の相互変換

１１３　日付を文字列に変換する……345
ONEPOINT■日付を文字列に変換するには
「SimpleDateFormat#format」メソッドを使用する

１１４　日付文字列を「Date」インスタンスに変換する……347
ONEPOINT■日付文字列を「Date」インスタンスに変換するには
「SimpleDateFormat#parse」メソッドを使用する

１１５　タイムゾーンを指定して日時を文字列に変換する……349
ONEPOINT■タイムゾーンを指定して日時を文字列に変換するには
「SimpleDateFormat#setTimeZone」メソッドを使用する

１１６　タイムゾーンの一覧を取得する……350
ONEPOINT■タイムゾーンの一覧を取得するには
「TimeZone.getAvailableIDs」メソッドを使用する

１１７　日付を比較する……351
ONEPOINT■日付を比較するには「Date#before」「Date#after」
「Date#compareTo」「Date#equals」メソッドを使用する
COLUMN■「Calendar」クラスでの日付の比較について

１１８　月の最終日を取得する……355
ONEPOINT■月の最終日を取得するには
「Calendar#getActualMaximum」メソッドを使用する

１１９　ナノ秒単位で時間を計測する……356
ONEPOINT■ナノ秒単位で時間を計測するには
「System.nanoTime」メソッドを使用する

CHAPTER 06　入出力

１２０　入出力について……358
COLUMN■「try-with-resources」文について

１２１　標準出力に文字列を出力する……361
ONEPOINT■標準出力に文字列を出力するには「System.out」を使用する

１２２　標準エラー出力に文字列を出力する……363
ONEPOINT■標準エラー出力に文字列を出力するには「System.err」を使用する
COLUMN■標準出力と標準エラー出力の違い

１２３　標準入力から文字列を読み込む……364
ONEPOINT■標準入力から文字列を読み込むには「System.in」を使用する

１２４　パスワードなどをエコーなしで入力させる……365
ONEPOINT■パスワードなどをエコーなしで入力させるには
「Console#readPassword」メソッドを使用する

１２５　ファイルからデータを読み込む ……366
ONEPOINT ■ファイルからデータを読み込むには「FileInputStream」クラスを使用する
COLUMN ■「FileChannel」クラスを使用してファイルからデータを読み込む方法

１２６　テキストファイルを読み込む ……371
ONEPOINT ■テキストファイルを読み込むには「FileReader」クラスを使用する
COLUMN ■NIO.2を使用してテキストファイルを読み込むには

１２７　エンコードを指定してテキストファイルを読み込む ……374
ONEPOINT ■エンコードを指定してテキストファイルを読み込むには「FileInputStream」クラスと「InputStreamReader」クラスを使用する

１２８　ファイルにデータを書き込む ……376
ONEPOINT ■ファイルにデータを書き込むには「FileOutputStream」クラスを使用する
COLUMN ■「FileChannel」クラスを使用してファイルにデータを書き込む方法

１２９　テキストファイルに書き込む ……379
ONEPOINT ■テキストファイルに書き込むには「FileWriter」クラスを使用する
COLUMN ■NIO.2を使用してテキストファイルに書き込むには

１３０　エンコードを指定してテキストファイルに書き込む ……383
ONEPOINT ■エンコードを指定してテキストファイルに書き込むには「FileOutputStream」クラスと「OutputStreamWriter」クラスを使用する

１３１　特定の位置からファイルを読み書きする ……384
ONEPOINT ■特定の位置からファイルを読み書きするには「RandomAccessFile」クラスを使用する

１３２　クラスパスからファイルを読み込む ……387
ONEPOINT ■クラスパスからファイルを読み込むには「Class#getResourceAsStream」メソッドを使用する
COLUMN ■クラスローダ

１３３　インスタンスをファイルに保存する ……389
ONEPOINT ■インスタンスをファイルに保存するには「ObjectOutputStream」クラスを使用する
COLUMN ■「直列化」について

１３４　ファイルからインスタンスを復元する……392
ONEPOINT ■ファイルからインスタンスを復元するには「ObjectInputStream」クラスを使用する

１３５　バイト配列をストリームとして扱う ……394
ONEPOINT ■バイト配列をストリームとして扱うには「ByteArrayOutputStream」クラスか「ByteArrayInputStream」クラスを使用する

１３６　ファイルにロックをかける ……396
ONEPOINT ■ファイルにロックをかけるには「FileChannel#lock」メソッドや「FileChannel#tryLock」メソッドを使用する

CHAPTER 07 ファイル操作

137 ファイル操作について …… 400
COLUMN■Java7でのファイル操作について

138 ファイル/ディレクトリを削除する …… 402
ONEPOINT■ファイル/ディレクトリを削除するには「File#delete」メソッドを使用する
COLUMN■NIO.2を使用してファイル/ディレクトリを削除するには

139 ファイル/ディレクトリの名前を変更する …… 405
ONEPOINT■ファイル/ディレクトリの名前を変更するには「File#renameTo」メソッドを使用する
COLUMN■NIO.2を使用してファイル/ディレクトリの名前を変更するには

140 ファイルをコピーする …… 408
ONEPOINT■ファイルをコピーするには「FileChannel#transferFrom」メソッドを使用する
COLUMN■NIO.2を使用してファイルをコピーするには

141 ファイル/ディレクトリを移動する …… 411
ONEPOINT■ファイル/ディレクトリを移動するには「File#renameTo」メソッドを使用する
COLUMN■NIO.2を使用してファイル/ディレクトリを移動するには

142 ファイル/ディレクトリのアクセス権を確認する …… 415
ONEPOINT■ファイル/ディレクトリのアクセス権を確認するには「File」クラスの「canExecute」「canRead」「canWrite」メソッドを使用する
COLUMN■NIO.2を使用してファイル/ディレクトリのアクセス権を確認するには

143 ファイル/ディレクトリのアクセス権を変更する …… 419
ONEPOINT■ファイル/ディレクトリのアクセス権を変更するには「File」クラスの「setExecutable」「setReadable」「setWritable」メソッドを使用する
COLUMN■NIO.2を使用してファイル/ディレクトリのアクセス権を変更するには

144 ファイルのサイズを取得する …… 423
ONEPOINT■ファイルのサイズを取得するには「File#length」メソッドを使用する
COLUMN■NIO.2を使用してファイルのサイズを取得するには

145 ファイルの最終更新日時を取得する …… 425
ONEPOINT■ファイルの最終更新日時を取得するには「File#lastModified」メソッドを使用する
COLUMN■NIO.2を使用してファイルの最終更新日時を取得するには

146 ファイルの最終更新日時を変更する …… 427
ONEPOINT■ファイルの最終更新日時を変更するには「File#setLastModified」メソッドを使用する
COLUMN■NIO.2を使用してファイルの最終更新日時を変更するには

147 パスからファイル名を取得する …… 430
ONEPOINT■パスからファイル名を取得するには「File#getName」メソッドを使用する
COLUMN■NIO.2を使用してパスからファイル名を取得するには

１４８　ファイル名から拡張子を取得する ……432
ONEPOINT ■ファイル名から拡張子を取得するには「String#lastIndexOf」メソッドと「String#substring」メソッドを使用する

１４９　パス区切り文字を意識せずにディレクトリ名とファイル名を結合する ……434
ONEPOINT ■パス区切り文字を意識せずにディレクトリ名とファイル名を結合するには「File」クラスを使用する
COLUMN ■NIO.2を使用してパス区切り文字を意識せずにディレクトリ名とファイル名を結合するには

１５０　指定したパスがディレクトリかファイルかを調べる ……436
ONEPOINT ■指定したパスがディレクトリかファイルかを調べるには「File#isDirectory」「File#isFile」メソッドを使用する
COLUMN ■NIO.2を使用して指定したパスがディレクトリかファイルかを調べるには

１５１　指定したパスが存在するか調べる ……438
ONEPOINT ■指定したパスが存在するか調べるには「File#exists」メソッドを使用する
COLUMN ■NIO.2を使用して指定したパスが存在するか調べるには

１５２　ディレクトリを作成する ……440
ONEPOINT ■ディレクトリを作成するには「File#mkdir」メソッドを使用する
COLUMN ■NIO.2を使用してディレクトリを作成するには

１５３　ディレクトリ内のファイル一覧を取得する ……442
ONEPOINT ■ディレクトリ内のファイル一覧を取得するには「File#list」「File#listFiles」メソッドを使用する
COLUMN ■NIO.2を使用してディレクトリ内のファイル一覧を取得するには

１５４　ファイル一覧にフィルタをかける ……445
ONEPOINT ■ファイル一覧にフィルタをかけるには「FilenameFilter」「FileFilter」インタフェースを実装したクラスを使用する
COLUMN ■NIO.2を使用してファイル一覧にフィルタをかけるには
COLUMN ■日本語ファイル名でフィルタをかける際の注意点

１５５　相対パスを絶対パスに変換する ……450
ONEPOINT ■相対パスを絶対パスに変換するには「File#getAbsolutePath」メソッドを使用する
COLUMN ■NIO.2を使用して相対パスを絶対パスに変換するには

１５６　テンポラリファイルを作成する ……452
ONEPOINT ■テンポラリファイルを作成するには「File.createTempFile」メソッドを使用する
COLUMN ■NIO.2を使用してテンポラリファイルを作成するには

１５７　ファイルをzip圧縮する ……455
ONEPOINT ■ファイルをzip圧縮するには「ZipOutputStream」クラスを使用する

１５８　zipファイルを解凍する ……458
ONEPOINT ■zipファイルを解凍するには「ZipInputStream」クラスを使用する
COLUMN ■日本語のファイル名を使う場合の注意点

CHAPTER 08 配列

159 配列について……462
COLUMN■多次元配列を宣言する方法
COLUMN■ジェネリックスの配列は作成できない

160 配列の中身を表示する……466
ONEPOINT■配列の中身を表示するには「Arrays.toString」メソッドや「Arrays.deepToString」メソッドを使用する

161 配列の長さを調べる……468
ONEPOINT■配列の長さを調べるには「length」を使用する

162 配列の要素を順番に取得する……469
ONEPOINT■配列の要素を順番に取得するには「拡張for」文を使用する

163 配列をコピーして新しい配列を作成する……470
ONEPOINT■配列をコピーして新しい配列を作成するには「Arrays.copyOf」メソッドを使用する

164 配列の要素をソートする……471
ONEPOINT■配列の要素をソートするには「Arrays.sort」メソッドを使用する

165 配列の要素を一度に初期化する……474
ONEPOINT■配列の要素を一度に初期化するには「Arrays.fill」メソッドを使用する

166 配列の要素から値を検索する……475
ONEPOINT■配列の要素から値を検索するには「Arrays.binarySearch」メソッドを使用する

167 配列を比較する……477
ONEPOINT■配列を比較するには「Arrays.equals」メソッドや「Arrays.deepEquals」メソッドを使用する

168 ランダムなバイト配列を取得する……480
ONEPOINT■ランダムなバイト配列を取得するには「SecureRandom#nextBytes」メソッドを使用する

CHAPTER 09 リスト

169 リストについて……482

170 リストに要素を追加する……485
ONEPOINT■リストに要素を追加するには「List#add」メソッドを使用する

171 リストの要素を上書きする……486
ONEPOINT■リストの要素を上書きするには「List#set」メソッドを使用する

172 2つのリストをマージする……487
ONEPOINT■2つのリストをマージするには「List#addAll」メソッドを使用する

１７３　リストの要素数を調べる ……489
ONEPOINT ■リストの要素数を調べるには「List#size」メソッドを使用する

１７４　リストが空かどうかを調べる ……490
ONEPOINT ■リストが空かどうかを調べるには「List#isEmpty」メソッドを使用する

１７５　リストから要素を取り出す ……491
ONEPOINT ■リストから要素を取り出すには「List#get」メソッドを使用する

１７６　リストの要素を順番に取得する ……492
ONEPOINT ■リストの要素を順番に取得するには「拡張for」文を使用する

１７７　リストから部分リストを作成する ……494
ONEPOINT ■リストから部分リストを作成するには
「List#subList」メソッドを使用する

１７８　リストから要素を削除する ……496
ONEPOINT ■リストから要素を削除するには「List#remove」メソッドを使用する

１７９　リストの全要素を削除する ……498
ONEPOINT ■リストの全要素を削除するには「List#clear」メソッドを使用する

１８０　2つのリストで重複する要素を削除する……500
ONEPOINT ■2つのリストで重複する要素を削除するには
「List#removeAll」メソッドを使用する

１８１　2つのリストで重複しない要素を削除する……502
ONEPOINT ■2つのリストで重複しない要素を削除するには
「List#retainAll」メソッドを使用する

１８２　リストの要素をソートする……504
ONEPOINT ■リストの要素をソートするには「Collections.sort」メソッドを使用する

１８３　リスト内に特定の要素が存在するか調べる ……507
ONEPOINT ■リスト内に特定の要素が存在するか調べるには
「List#contains」メソッドを使用する

１８４　リストを配列に変換する ……509
ONEPOINT ■リストを配列に変換するには「List#toArray」メソッドを使用する

１８５　リストをスレッドセーフにする ……511
ONEPOINT ■リストをスレッドセーフにするには
「Collections.synchronizedList」メソッドを使用する

１８６　リストを読み取り専用にする ……514
ONEPOINT ■リストを読み取り専用にするには
「Collections.unmodifiableList」メソッドを使用する

１８７　リストの要素の順序を逆にする ……516
ONEPOINT ■リストの要素の順序を逆にするには
「Collections.reverse」メソッドを使用する

１８８　リストの要素の順序をランダムに入れ替える……518
ONEPOINT ■リストの要素の順序をランダムに入れ替えるには
「Collections.shuffle」メソッドを使用する

189 配列からリストを作成する ……520
ONEPOINT■配列からリストを作成するには「Arrays.asList」メソッドを使用する
COLUMN■「Arrays.asList」メソッドで作成したリストには要素を追加できない

CHAPTER 10 マップ

190 マップについて ……524
191 マップに値を格納する ……525
ONEPOINT■マップに値を格納するには「Map#put」メソッドを使用する
192 2つのマップをマージする ……526
ONEPOINT■2つのマップをマージするには「Map#putAll」メソッドを使用する
193 マップから値を取り出す ……528
ONEPOINT■マップから値を取り出すには「Map#get」メソッドを使用する
194 マップからマッピングを削除する ……529
ONEPOINT■マップからマッピングを削除するには「Map#remove」メソッドを使用する
195 マップからすべてのマッピングを削除する ……531
ONEPOINT■マップからすべてのマッピングを削除するには「Map#clear」メソッドを使用する
196 マップ内のマッピング数を調べる ……532
ONEPOINT■マップ内のマッピング数を調べるには「Map#size」メソッドを使用する
197 マップのキーを順番に取得する ……533
ONEPOINT■マップのキーを順番に取得するには「Map#keySet」メソッドを使用する
198 マップを読み取り専用にする ……535
ONEPOINT■マップを読み取り専用にするには「Collections.unmodifiableMap」メソッドを使用する
199 自作クラスをHashMapのキーとして使用する ……537
ONEPOINT■自作クラスをHashMapのキーとして使用するには「equals」メソッドと「hashCode」メソッドを実装する
200 自作クラスをTreeMapのキーとして使用する ……541
ONEPOINT■自作クラスをTreeMapのキーとして使用するには「Comparable」インタフェースを実装する

CHAPTER 11 数学処理

201 数学処理について ……546
COLUMN■「Math」クラスと「StrictMath」クラスの違い

202 絶対値を取得する……549
ONEPOINT■絶対値を取得するには「Math.abs」メソッドを使用する

203 三角関数を使用する……550
ONEPOINT■三角関数を使用するには「Math」クラスに用意されているメソッドを使用する
COLUMN■度単位をラジアンに変換するには

204 平方根を計算する……552
ONEPOINT■平方根を計算するには「Math.sqrt」メソッドを使用する

205 累乗を計算する……553
ONEPOINT■累乗を計算するには「Math.pow」メソッドを使用する

206 小数点以下の切り捨てを行う……554
ONEPOINT■小数点以下の切り捨てを行うには「Math.floor」メソッドを使用する

207 小数点以下の切り上げを行う……555
ONEPOINT■小数点以下の切り上げを行うには「Math.ceil」メソッドを使用する

208 小数点以下の四捨五入を計算する……556
ONEPOINT■小数点以下の四捨五入を計算するには「Math.round」メソッドを使用する

209 2つの数値の大きい方を求める……557
ONEPOINT■2つの数値の大きい方を求めるには「Math.max」メソッドを使用する

210 2つの数値の小さい方を求める……558
ONEPOINT■2つの数値の小さい方を求めるには「Math.min」メソッドを使用する

211 小数点以下を正確に扱う……559
ONEPOINT■小数点以下を正確に扱うには「BigDecimal」クラスを使用する
COLUMN■「BigDecimal」クラスの計算用メソッドについて

212 指定した桁で数値を丸める……561
ONEPOINT■指定した桁で数値を丸めるには「BigDecimal#round」メソッドを使用する

213 BigDecimalで値の比較を行う……563
ONEPOINT■BigDecimalで値の比較を行うには「BigDecimal#compareTo」メソッドを使用する

214 乱数を取得する……565
ONEPOINT■乱数を取得するには「Math.random」メソッドを使用する

CHAPTER 12 ネットワーク

215 ネットワークについて……568

216 TCP/IP通信を行う……569
ONEPOINT■TCP/IP通信を行うには「Socket」クラスを使用する

217 接続相手のIPアドレスを取得する ……573
ONEPOINT■接続相手のIPアドレスを取得するには「Socket#getInetAddress」メソッドを使用する

218 ホスト名をIPアドレスに変換する……575
ONEPOINT■ホスト名をIPアドレスに変換するには「InetAddress.getByName」メソッドを使用する

219 IPアドレスをホスト名に変換する……576
ONEPOINT■IPアドレスをホスト名に変換するには「InetAddress#getHostName」メソッドを使用する

220 通信が到達可能か確認する……577
ONEPOINT■通信が到達可能か確認するには「InetAddress#isReachable」メソッドを使用する

221 WebサーバへHTTP接続する ……578
ONEPOINT■WebサーバへHTTP接続するには「URL#openConnection」メソッドを使用する

222 HTTP通信のレスポンスコードを取得する ……581
ONEPOINT■HTTP通信のレスポンスコードを取得するには「HttpURLConnection#getResponseCode」メソッドを使用する

223 HTTP通信で自動的にリダイレクトに従うか否かを設定する ……583
ONEPOINT■HTTP通信で自動的にリダイレクトに従うか否かを設定するには「HttpURLConnection#getResponseCode」メソッドを使用する

224 メールを送信する ……586
ONEPOINT■メールを送信するには「MimeMessage」クラスを使用する
COLUMN■Windows系でメールを送信する際の注意点
COLUMN■JavaMailのインストール方法

225 メールにファイルを添付して送信する ……591
ONEPOINT■メールにファイルを添付して送信するには「MimeBodyPart」クラスを使用する

226 メールを受信する ……595
ONEPOINT■メールを受信するには「Store」クラスを使用する

CHAPTER 13 Webアプリケーション

227 Webアプリケーションについて ……602

228 Tomcatのインストールと環境の設定 ……605

229 クライアントの要求を処理する ……617
ONEPOINT■クライアントの要求を処理するには「HttpServlet」クラスの「doPost」「doGet」メソッドを使用する
COLUMN■Tomcat7.0でのサーブレットの利用

230 コンテンツのエンコードを指定する ……622
ONEPOINT ■コンテンツのエンコードを指定するには「HttpServletResponse#setContentType」メソッドを使用する

231 リクエストパラメータの値を取り出す ……624
ONEPOINT ■リクエストパラメータの値を取り出すには「HttpServletRequest#getParameter」メソッドを使用する

232 リクエストパラメータの日本語の文字化けを防ぐ ……626
ONEPOINT ■リクエストパラメータの日本語の文字化けを防ぐには「HttpServletRequest#setCharacterEncoding」メソッドを使用する
COLUMN ■Filterを使った文字化け防止
COLUMN ■GETパラメータの文字化けを防ぐ

233 セッション情報にアクセスする ……631
ONEPOINT ■セッション情報にアクセスするには「HttpServletRequest#getSession」メソッドを使用する

234 セッション情報を破棄する ……634
ONEPOINT ■セッション情報を破棄するには「HttpSession#invalidate」メソッドを使用する

235 Cookieに値を設定する ……638
ONEPOINT ■Cookieに値を設定するには「HttpServletResponse#addCookie」メソッドを使用する
COLUMN ■Cookieの確認方法

236 Cookieから値を取得する ……640
ONEPOINT ■Cookieから値を取得するには「HttpServletRequest#getCookies」メソッドを使用する

237 レスポンスヘッダを追加する ……642
ONEPOINT ■レスポンスヘッダを追加するには「HttpServletResponse#addHeader」メソッドを使用する
COLUMN ■レスポンスヘッダの例

238 レスポンスコードを変更する ……644
ONEPOINT ■レスポンスコードを変更するには「HttpServletResponse#setStatus」メソッドを使用する
COLUMN ■レスポンスコードの番号体系

239 リクエストされたURLを取得する ……646
ONEPOINT ■リクエストされたURLを取得するには「HttpServletRequest#getRequestURL」メソッドを使用する

240 別のページにリダイレクトさせる ……648
ONEPOINT ■別のページにリダイレクトさせるには「HttpServletResponse#sendRedirect」メソッドを使用する

241 JSPに値を出力する ……650
ONEPOINT ■JSPに値を出力するには「<c:out>」タグを使用する
COLUMN ■JSPで使える暗黙インスタンス
COLUMN ■EL式で使える暗黙インスタンス

242 <c:out>タグでHTMLエスケープせずに値をそのまま出力する …654
ONEPOINT■<c:out>タグでHTMLエスケープせずに値をそのまま出力するには「escapeXml」属性を設定する

243 JSPで繰り返し処理を行う ……656
ONEPOINT■JSPで繰り返し処理を行うには「<c:forEach>」タグを使用する

244 JSPで条件分岐を行う ……659
ONEPOINT■JSPで条件分岐を行うには「<c:if>」タグや「<c:choose>」タグを使用する

245 カスタムタグを作成する ……662
ONEPOINT■カスタムタグを作成するには「TagSupport」インタフェースか「BodyTagSupport」インタフェースを実装する

246 Webアプリケーション起動時に特定のサーブレットを実行させる …667
ONEPOINT■Webアプリケーション起動時に特定のサーブレットを実行させるには「@WebServlet」アノテーションで「loadOnStartup」を設定する

247 「@WebInitParam」アノテーションで指定した初期化パラメータの値を取得する …669
ONEPOINT■「@WebInitParam」アノテーションで指定した初期化パラメータの値を取得するには「HttpServlet#getInitParameter」メソッドを使用する

CHAPTER 14 GUI

248 GUIについて ……672

249 ウィンドウを表示する ……674
ONEPOINT■ウィンドウを表示するには「JFrame」クラスか「JDialog」クラスを使用する
COLUMN■ウィンドウの閉じるボタンをクリックしたときにアプリケーションも終了させる
COLUMN■ウィンドウの位置やサイズを指定するには

250 ウィンドウにボタンを配置する ……677
ONEPOINT■ウィンドウにボタンを配置するには「JButton」クラスを使用する
COLUMN■コンポーネントの自動配置

251 イベント処理を行う ……685
ONEPOINT■イベント処理を行うには「EventListener」インタフェースを使用する

252 ウィンドウにチェックボックスを配置する ……690
ONEPOINT■ウィンドウにチェックボックスを配置するには「JCheckBox」クラスを使用する

253 ウィンドウにコンボボックスを配置する ……692
ONEPOINT■ウィンドウにコンボボックスを配置するには「JComboBox」クラスを使用する

254 ウィンドウにラベルを配置する ……696
ONEPOINT■ウィンドウにラベルを配置するには「JLabel」クラスを使用する

255 ウィンドウに画像を表示する ……699
ONEPOINT ■ウィンドウに画像を表示するには「JLabel」クラスを使用する

256 表示領域を分割する ……703
ONEPOINT ■ウィンドウの表示領域を分割するには「JPanel」クラスを使用する

257 ウィンドウにリストを配置する ……706
ONEPOINT ■ウィンドウにリストを配置するには「JList」クラスを使用する

258 表示領域をスクロール可能にする ……711
ONEPOINT ■表示領域をスクロール可能にするには「JScrollPane」クラスを使用する

259 ウィンドウにメニューバーを配置する ……713
ONEPOINT ■ウィンドウにメニューバーを配置するには「JMenuBar」クラスを使用する

260 ウィンドウにプログレスバーを配置する ……717
ONEPOINT ■ウィンドウにプログレスバーを配置するには「JProgressBar」クラスを使用する

261 ウィンドウにラジオボタンを配置する ……720
ONEPOINT ■ウィンドウにラジオボタンを配置するには「JRadioButton」クラスを使用する

262 ウィンドウにスピナーを配置する ……724
ONEPOINT ■ウィンドウにスピナーを配置するには「JSpinner」クラスを使用する

263 ウィンドウにテキスト入力エリアを配置する ……728
ONEPOINT ■ウィンドウにテキスト入力エリアを配置するには「JTextArea」クラスを使用する

264 ウィンドウにテキスト入力フィールドを配置する ……731
ONEPOINT ■ウィンドウにテキスト入力フィールドを配置するには「JTextField」クラスを使用する

265 ウィンドウにツリーを配置する ……734
ONEPOINT ■ウィンドウにツリーを配置するには「JTree」クラスを使用する

266 ポップアップメニューを表示する ……738
ONEPOINT ■ポップアップメニューを表示するには「JPopupMenu」クラスを使用する

267 メッセージボックスを表示する ……742
ONEPOINT ■メッセージボックスを表示するには「JOptionPane」クラスを使用する

268 ユーザにファイルを選択させる ……748
ONEPOINT ■ユーザにファイルを選択させるには「JFileChooser」クラスを使用する

269 コンポーネントにフォーカスを与える ……753
ONEPOINT ■コンポーネントにフォーカスを与えるには「Component#requestFocus」メソッドを使用する

270 ウィンドウの外見を変更する ……756
ONEPOINT■ウィンドウの外見を変更するには「JFrame.setDefaultWindowDecorationDecorated」メソッドを使用する

271 画面のサイズを取得する ……758
ONEPOINT■画面のサイズを取得するには「Toolkit#getScreenSize」メソッドを使用する

272 クリップボードにテキストをコピーする ……759
ONEPOINT■クリップボードにテキストをコピーするには「Clipboard#setContents」メソッドを使用する

273 クリップボードからテキストを取得する ……762
ONEPOINT■クリップボードからテキストを取得するには「Clipboard#getData」メソッドを使用する

CHAPTER 15 XML処理

274 XML処理について ……768
COLUMN■XMLの歴史

275 XMLファイルを読み込みながら解析をする ……771
ONEPOINT■XMLファイルを読み込みながら解析をするには「SAXParser#parse」メソッドを使用する

276 ツリー構造としてXMLファイルを読み込む ……775
ONEPOINT■ツリー構造としてXMLファイルを読み込むには「DocumentBuilder#parse」メソッドを使用する

277 ルートノードを取得する ……777
ONEPOINT■ルートノードを取得するには「Document#getDocumentElement」メソッドを使用する

278 ノードの種類を判別する ……779
ONEPOINT■ノードの種類を判別するには「Node#getNodeType」メソッドを使用する

279 子ノードの一覧を取得する ……782
ONEPOINT■子ノードの一覧を取得するには「Node#getChildNodes」メソッドを使用する

280 要素名を指定して要素を取得する ……784
ONEPOINT■要素名を指定して要素を取得するには「Element#getElementsByTagName」メソッドを使用する

281 ノードのテキストコンテンツを取得する ……786
ONEPOINT■ノードのテキストコンテンツを取得するには「Node#getTextContent」メソッドを使用する

282 親ノードを取得する ……789
ONEPOINT■親ノードを取得するには「Node#getParentNode」メソッドを使用する

283 要素の名前を取得する……791
ONEPOINT■要素の名前を取得するには
「Element#getTagName」メソッドを使用する

284 要素の属性を取得する……793
ONEPOINT■要素の属性を取得するには
「Element#getAttribute」メソッドを使用する

285 XPathを使ってノードを取得する……796
ONEPOINT■XPathを使ってノードを取得するには
「XPath#evaluate」メソッドを使用する

286 名前空間があるXMLでXPathを使ってノードを取得する……799
ONEPOINT■名前空間があるXMLでXPathを使ってノードを取得するには
「NamespaceContext」インタフェースを実装したクラスを使用する

287 新しくXMLツリーを作成する……803
ONEPOINT■新しくXMLツリーを作成するには
「DocumentBuilder#newDocument」メソッドを使用する

288 要素を作成する……804
ONEPOINT■要素を作成するには
「Document#createElement」メソッドを使用する

289 要素に属性を設定する……807
ONEPOINT■要素に属性を設定するには
「Document#createAttribute」メソッドを使用する

290 テキストノードを作成する……810
ONEPOINT■テキストノードを作成するには
「Document#createTextNode」メソッドを使用する

291 XMLツリーをファイルに出力する……813
ONEPOINT■XMLツリーをファイルに出力するには
「Transformer#transform」メソッドを使用する

292 XSL変換を行う……816
ONEPOINT■XSL変換を行うには
「TransformerFactory#newTransformer」メソッドを使用する

CHAPTER 16 スレッドと並列処理

293 スレッドと並列処理について……820

294 スレッドを作成する……821
ONEPOINT■スレッドを作成するには「Thread#start」メソッドを使用する

295 スレッドを終了させる……824
ONEPOINT■スレッドを終了させるには「Thread#run」メソッドや
「Runnable#run」メソッドから抜ける

296 外部からスレッドを終了させる……826
ONEPOINT■外部からスレッドを終了させるには「volatile」変数で制御する

297 現在実行しているスレッドを取得する……828
ONEPOINT■現在実行しているスレッドを取得するには「Thread.currentThread」メソッドを使用する

298 一定時間スレッドを停止させる……829
ONEPOINT■一定時間スレッドを停止させるには「Thread.sleep」メソッドやTimeUnit列挙型を使用する

299 スレッド内のメソッド呼び出し履歴を取得する……831
ONEPOINT■スレッド内のメソッド呼び出し履歴を取得するには「Thread#getStackTrace」メソッドを使用する

300 スレッドの終了待ちを行う……834
ONEPOINT■スレッドの終了待ちを行うには「Thread#join」メソッドを使用する

301 スレッドをデーモンスレッドにする……836
ONEPOINT■スレッドをデーモンスレッドにするには「Thread#setDaemon」メソッドを使用する

302 タイマーで処理を実行する……838
ONEPOINT■タイマーで処理を実行するには「Timer#schedule」メソッドを使用する

303 タイマーを使って一定間隔で処理を実行する……841
ONEPOINT■タイマーを使って一定間隔で処理を実行するには「Timer#scheduleAtFixedRate」メソッドを使用する

304 タイマーを使って処理終了から一定時間後に処理を再実行する……843
ONEPOINT■タイマーを使って処理終了から一定時間後に処理を再実行するには「ScheduledExecutorService#scheduleWithFixedDelay」メソッドを使用する

305 タイマーをキャンセルする……846
ONEPOINT■タイマーをキャンセルするには「Timer#cancel」メソッドを使用する

306 処理をスレッドセーフにする……848
ONEPOINT■処理をスレッドセーフにするには「synchronized」修飾子を使用する

307 待ち状態または停止状態のスレッドに割り込みをかける……852
ONEPOINT■待ち状態または停止状態のスレッドに割り込みをかけるには「Thread#interrupt」メソッドを使用する

308 メインスレッド以外のスレッドでのランタイム例外をキャッチする……854
ONEPOINT■メインスレッド以外のスレッドでのランタイム例外をキャッチするには「ThreadGroup#uncaughtException」メソッドを使用する

309 スレッドグループを使わずにメインスレッド以外のランタイム例外をキャッチする……856
ONEPOINT■スレッドグループを使わずにメインスレッド以外のランタイム例外をキャッチするには「Thread#setUncaughtExceptionHandler」メソッドを使用する

310 スレッド間でタイミングを取る……858
ONEPOINT■スレッド間でタイミングを取るには「Object#notify」メソッドと「Object#wait」メソッドを使用する

311 スレッド単位で値を管理する……863
ONEPOINT■スレッド単位で値を管理するには「ThreadLocal」クラスを使用する

312 スレッドのプーリングを行う……866
ONEPOINT■スレッドのプーリングを行うには「ExecutorService」クラスを使用する
COLUMN■Fork/Joinフレームワークについて

313 プーリングされたスレッドから戻り値を取得する……873
ONEPOINT■プーリングされたスレッドから戻り値を取得するには「ExecutorService#submit」メソッドを使用する

314 プーリングされたスレッドでのランタイム例外を取得する……876
ONEPOINT■プーリングされたスレッドでのランタイム例外を取得するには「ThreadFactory」インタフェースを実装したクラスを作成する

CHAPTER 17 データベース

315 データベースについて……880

316 データベースへ接続する……884
ONEPOINT■データベースへ接続するには「DriverManager.getConnection」メソッドを使用する
COLUMN■DataSourceを使った接続の作成
COLUMN■JDBCドライバのタイプ

317 データベースから検索を行う……887
ONEPOINT■データベースから検索を行うには「Statement#executeQuery」メソッドを使用する

318 データベースへ登録・更新・削除を行う……889
ONEPOINT■データベースへ登録・更新・削除を行うには「Statement#executeUpdate」メソッドを使用する

319 結果セットを使ってデータベースを更新する……892
ONEPOINT■結果セットを使ってデータベースを更新するには「ResultSet#updateRow」メソッドを使用する
COLUMN■JDBCドライバのバージョンに注意

320 SQLインジェクションを防止する……895
ONEPOINT■SQLインジェクションを防止するには「PreparedStatement」インタフェースを使用する

321 データベースからBLOB/CLOBの検索を行う……897
ONEPOINT■データベースからBLOBの検索を行うには「Blob#getBinaryStream」メソッド、CLOBの検索を行うには「Clob#getCharacterStream」メソッドを使用する

322 データベースへBLOB/CLOBの更新を行う ……901
ONEPOINT■データベースへBLOB/CLOBの更新を行うには
「PreparedStatement#setBinaryStream」メソッドや
「PreparedStatement#setCharacterStream」メソッドを使用する

323 データベースのトランザクションを開始する ……905
ONEPOINT■データベースのトランザクションを開始するには
「Connection#setAutoCommit」メソッドを使用する

324 SQLエラー発生時にDB固有のエラーコードを取得する ……909
ONEPOINT■SQLエラー発生時にDB固有のエラーコードを取得するには
「SQLException#getErrorCode」メソッドを使用する

325 複数のSQLを一度に実行する……910
ONEPOINT■複数のSQLを一度に実行するには「Statement」クラスや
「PreparedStatement」クラスの「addBatch」メソッドと
「executeBatch」メソッドを使用する

326 ストアドプロシージャを実行する ……914
ONEPOINT■ストアドプロシージャを実行するには
「CallableStatement#execute」メソッドを使用する

CHAPTER 18 システム

327 システムについて ……920

328 システム情報を取得する ……921
ONEPOINT■システム情報を取得するには
「System.getProperty」メソッドを使用する

329 カレントディレクトリを取得する ……922
ONEPOINT■カレントディレクトリを取得するには
「System.getProperty("user.dir")」メソッドを使用する

330 ホームディレクトリを取得する……923
ONEPOINT■ホームディレクトリを取得するには
「System.getProperty("user.home")」メソッドを使用する

331 改行文字を取得する……924
ONEPOINT■改行文字を取得するには
「System.getProperty("line.separator")」メソッドを使用する

332 システムプロパティを設定する ……925
ONEPOINT■システムプロパティを設定するにはJavaの起動オプションに
「-DXX=XXX」を指定する

333 プログラムを終了する ……926
ONEPOINT■プログラムを終了するには「System.exit」メソッドを使用する
COLUMN■「main」メソッド内で「return」文を使って終了したときとの違い

334 プログラム終了時に特定の処理を行う ……928
ONEPOINT■プログラム終了時に特定の処理を行うには「Runtime#addShutdownHook」メソッドを使用する

335 外部プログラムを起動する ……930
ONEPOINT■外部プログラムを起動するには「ProcessBuilder#start」メソッドを使用する

336 起動した外部プログラムの標準入出力を制御する ……932
ONEPOINT■起動した外部プログラムの標準入出力を制御するには「Process」クラスを使用する

337 環境変数の値を取得する ……934
ONEPOINT■環境変数の値を取得するには「System.getenv」メソッドを使用する

338 JavaVMのメモリ空き容量を取得する ……935
ONEPOINT■JavaVMのメモリ空き容量を取得するには「Runtime#freeMemory」メソッドを使用する
COLUMN■JavaVMのヒープメモリを増やす

339 ガベージコレクタを明示的に実行する ……937
ONEPOINT■ガベージコレクタを明示的に実行するには「System.gc」メソッドを使用する

340 ガベージコレクタのログを出力する ……939
ONEPOINT■ガベージコレクタのログを出力するにはJavaの起動オプションに「-verbose:gc」を指定する
COLUMN■ガベージコレクタのログをファイルに記録する

341 実行中のマシンのエンディアンを判別する ……942
ONEPOINT■実行中のマシンのエンディアンを判別するには「ByteOrder.nativeOrder」メソッドを使用する

CHAPTER 19 ログ出力

342 ログ出力について ……944
COLUMN■Apache log4j

343 ログを出力する ……946
ONEPOINT■ログを出力するには「Logger」クラスを使用する

344 ログ出力の設定を変更する ……948
ONEPOINT■ログ出力の設定を変更するには「logging.properties」を編集する
COLUMN■「loggiong.properties」の場所

345 アプリごとにログ出力の設定を変更する ……951
ONEPOINT■アプリごとにログ出力の設定を変更するには独自に用意した設定ファイルを「LogManager#readConfiguration」メソッドで読み込む
COLUMN■ログ出力時の注意

346 ログをファイルに出力する ……953
ONEPOINT■ログをファイルに出力するには「FileHandler」クラスを使用する
COLUMN■ログ出力時にキャッシュを使う

347 ログの出力フォーマットを変更する……959
ONEPOINT■ログの出力フォーマットを変更するには
「Formatter」クラスを継承したクラスを作成する

CHAPTER 20 その他

348 例外が発生した箇所についての情報を取得する ……962
ONEPOINT■例外が発生した箇所についての情報を取得するには
「Throwable#getStackTrace」メソッドを使用する
COLUMN■スタックトレースの読み方

349 スタックトレースをストリームに出力する ……966
ONEPOINT■スタックトレースをストリームに出力するには
「Throwable#printStackTrace」メソッドを使用する

350 アプリケーション固有の設定情報を保存する ……968
ONEPOINT■アプリケーション固有の設定情報を保存するには
「Preferences」クラスを使用する

351 バイト配列を16進文字列に変換する……970
ONEPOINT■バイト配列を16進文字列に変換するには
「DatatypeConverter.printHexBinary」メソッドを使用する

352 16進文字列をバイト配列に変換する……971
ONEPOINT■16進文字列をバイト配列に変換するには
「DatatypeConverter.parseHexBinary」メソッドを使用する
COLUMN■BASE64について

353 MD5やSHA-1のハッシュ値を作成する ……974
ONEPOINT■MD5やSHA-1のハッシュ値を作成するには
「MessageDigest」クラスを使用する
COLUMN■ハッシュ値とは

●索引……976

CHAPTER 01

Javaの基礎知識

SECTION-001

Javaとは

Javaの特徴

Javaは、オブジェクト指向プログラミング言語の1つです。プログラムを直接、OS上で実行するのではなく、Java Virtual Machine(JavaVM)を起動してその上でプログラムを実行するというのが大きな特徴です。そのため、C言語などと比べるとメモリの使用量が大きく、実行速度も遅くなるという短所はありますが、コンパイルしたモジュールはJavaVMが動けばコンパイルし直すことなく、そのまま動かせるという長所があります。そのパフォーマンスがよくないという欠点も、Java1.2でJITという技術を採用したことによって大きく改善されています。最初からスレッドやネットワークが使えたり、セキュリティに対する考慮がなされたりしているのも特徴の1つです。

Javaでは、変数の型については強い静的型付けが使われています。スクリプト言語では動的型付けが使われていて同じ変数に文字列を代入したり数値を代入したりできますが、Javaではあらかじめ決められた型の値しか代入することができません。少し面倒にも見えますが、意図しない挙動が発生しそうな箇所をコンパイルの段階で発見することができます。

メモリ管理

Javaでは、後述する「インスタンス」を必要に応じて作成し、処理を行っていきます。インスタンスを作成すると、その分、メモリを消費します。インスタンスをいつまでも残しておくとメモリをどんどん消費していくため、役割を終えたインスタンスはメモリから解放する必要があります。このインスタンスの解放はJavaVMが自動的に行うため、普段は意識する必要はありません。この仕組みを「ガベージコレクタ」(garbage collector)といいます。

作成したインスタンスを保持する領域を「ヒープ領域」といいますが、Javaのガベージコレクタではヒープ領域をNEW領域とOLD領域に分けて管理します。インスタンスを作成すると、NEW領域に格納されます。NEW領域がいっぱいになると、ガベージコレクタを実行します。このとき、使われなくなったインスタンスは解放されます。一定の回数、ガベージコレクタを実行してもまだ残っているインスタンスは、OLD領域に移動します。OLD領域がいっぱいになったときも、ガベージコレクタを実行します。このときのガベージコレクタは「フルガベージコレクタ」(full garbage collector)といい、その間の他の処理は停止します。

Javaの歴史

1996年にJDK 1.0が公開されて以来バージョンアップを続けてきて、2012年9月時点ではJava7(Java SE 7)が最新です。

- 1996年　JDK 1.0
- 1997年　JDK 1.1
- 1998年　J2SE 1.2

- 2000年　J2SE 1.3
- 2002年　J2SE 1.4
- 2004年　J2SE 5.0
- 2006年　Java SE 6
- 2011年　Java SE 7

バージョン1.2になったときに、名称が「Java2」に変わり、JDKも「Java2 Platform Standard Edition」(J2SE)に変わりました。また、バージョン1.5になったときには「J2SE 1.5」ではなく、「J2SE 5.0」とバージョンの表記が変わりました(ただし、内部的にはまだ「1.x.x」の形でバージョンを管理している)。次のバージョン名称では「J2SE」から「Java SE」に変え、「Java SE 6」となりました。なお、本書では、「Java1.4」「Java5」「Java6」のように「Java+バージョン番号」の形で表します。

バージョンアップは、基本的に旧バージョンとの互換性(後方互換)を保つようになっています。そのため、言語仕様の変化というよりはクラスの追加・修正が中心です。しかし、「J2SE 1.2」と「J2SE 5.0」ではJavaVMの仕様や言語仕様が大きく変わったため、それ以前のバージョンとの互換性は低くなっています。そして、次のJava SE 8でも大きな変更が加えられる予定です。

COLUMN　Javaの用途

Javaが生まれるまでのコンピュータプログラムはCPUの種類やオペレーティング・システムといった動作環境の影響を受け、Windows用に作成されたプログラムはそのままではMac OS上では動作しないといった制約がありました。Javaはこのような環境による制約をできるだけ減らすため、プログラムを一度書けば、どの環境でも動作することを目的の1つとして設計されました。

当初はこの特徴を生かしたパソコン上で動作するアプリケーション開発が主なJavaの用途と思われていましたが、初期のJavaではユーザインタフェース(画面の入出力)が見劣りしたことなどから、一般的なアプリケーションとして利用されることは少なく、代わりに裏方であるサーバ側のプログラミングに利用されるようになりました。また、携帯電話や家庭電化製品の中でも広くJavaが利用されています。

SECTION-002

オブジェクト指向について

オブジェクト指向

「オブジェクト」は「物」という意味の英単語ですが、ここでは部品と考えた方がわかりやすいかもしれません。「車輪」「車軸」「板」という部品を組み合わせることで、荷車になります。車軸が木製でよく壊れるようなものだった場合、鉄製の車軸に変えれば耐久性は向上します。また、これらの部品に「屋根」を付ければ雨の日でも荷物を運べるようになります。

このように部品を作り直していくことで性能を向上させたり、新しい部品を作ることで機能を追加していくのがオブジェクト指向の特徴です。

Javaの場合、その部品にあたるオブジェクトは、データとそれに対してどのような処理を行うかという操作を持っています。それらを定義したものがクラスといいます。このクラスもまた「定義した“もの”」なので、オブジェクトの1つです。そして、そのクラスを基に作成したオブジェクトをインスタンスといいます。

クラスを定義して、そのクラスを基にインスタンスを作成し、インスタンスを組み合わせてプログラムを作成していくのがオブジェクト指向言語です。

カプセル化

データとそれに対する操作を1つにまとめることを「カプセル化」(encapsulation)といいます。カプセル化することによって、誤った操作でデータを破壊することを防ぐことができます。Javaではこのデータを「フィールド」、操作を「メソッド」といいます。特にフィールドを設定・取得するメソッドのことを「アクセサメソッド」(アクセスメソッド)といいます。

継承

既存の部品を修正して機能を拡張することを「継承」(inheritance)といいます。Javaでは、あるクラスを継承した別のクラスを作ることができます。たとえば、クラスAを継承したクラスAAを作成したとすると、このクラスAAはクラスAの定義内容をすべて引き継ぎ、さらにクラスAAで定義した内容も使うことができます。同じオブジェクト指向言語のC++では複数のクラスを継承したクラスを定義することができますが、Javaでは複数のクラスを継承することはできません。その代わり、インタフェースというものを使うことができます。

多態性

同じ操作を行わせても、その操作を行うインスタンスや処理対象によって処理内容が異なることを「多態性」(polymorphism)といいます。同じ操作といっても実際に同じメソッドを呼び出すのではなく、メソッドの呼び出し元のクラスや対象によって処理内容の異なる複数のメソッドを同じメソッド名で定義して、場合によって使い分けています。

たとえば、数値を2倍にするメソッドを作るとします。もし、多態性が使えなければ、整数を2倍にする「doubleValueInt」と小数を2倍にする「doubleValueFloat」の2つを定義して、処

理対象によって使い分ける必要があります。多態性を使うと、整数を2倍にするメソッドと小数を2倍にするメソッドの2つを定義するのは同じですが、どちらの場合も「doubleValue」という同じメソッド名で定義できるため、処理対象によってメソッドを使い分ける必要がなくなります。

「カプセル化」「継承」「多態性」の3つを使わずに、Javaでコーディングして、アプリケーションを作ることはできます。しかし、アプリケーションの規模が大きくなればなるほど、これらを意識した設計を行い、コーディングしていくことで保守性が高まります。

実際にこれらをどのようにJavaで実現するかについては、CHAPTER 02で解説します。

COLUMN　デザインパターン

オブジェクト指向によって部品がブラックボックス化されると、プログラムの再利用率が高まります。その反面、部品を作成するためには、高度なプログラミング技術が必要とされるようになりました。そこで先人たちによって考えだされたのが「デザインパターン」と呼ばれる、プログラムの設計における問題を解決する手法をまとめたもので書籍も出版されています。

設計をパターン化して呼び名を付けることにより、誰でもその手法を利用できるようになり、同じようなことで多くの人が頭を悩ませる必要がなくなります。本書では「デザインパターン」についての解説はファクトリパターンについて少し触れるだけですが、知識として身に付けておくとソフトウエア開発の役に立つはずです。

関連項目 ▶▶▶

- クラスについて……p.117
- フィールドについて……p.133
- メソッドについて……p.139
- インタフェースについて……p.156
- 継承と実装について……p.159
- オーバーライドについて……p.163
- オーバーロードについて……p.167

SECTION-003

Java開発キットの入手方法について

Java開発キットのダウンロード

Javaの開発環境を構築するには、まず、Java開発キットをダウンロードします。Oracle社のサイトからJava SE 7のSoftware Development Toolkit(JDK 1.7)をダウンロードするには、次のように操作します。

❶「http://www.oracle.com/jp/index.html」を表示し、ページ上部にある「ダウンロード」メニューをクリックします。

❷「オラクルのダウンロード」のページが表示されるので、ページ内にある「Java SE」のリンクをクリックしてます。

❸「Java SE ダウンロード」ページが表示されます。最新版はUSサイトにあるので、「こちら(USサイト)」のリンクをクリックします。

❹「Java SE 7uX」(Xはバージョン)の欄の「JDK」の下にある「DOWNLOAD」ボタンをクリックします。2012年9月時点での最新バージョンは「Java SE 7u7」です。もう1つのJREというのはJavaの実行環境のみのパッケージです。JDKの中にJREも含まれているので、ここでは不要です。

❺ 次にプラットフォーム選択画面が表示されるので、[Accept License Agreement](ライセンスに同意)をONにし、自分の環境にあったプラットフォームのインストーラのリンクをクリックしてダウンロードします。

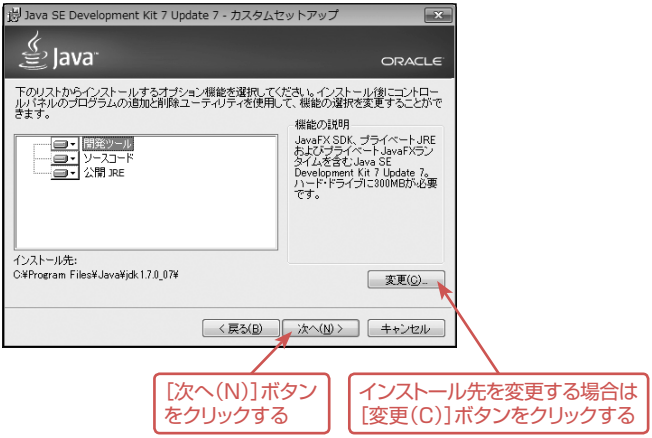
Java開発キットのインストール

インストーラのダウンロードが終わったら、インストールを行います。ここでは、Windows 7にWindows(x86)版をインストールする場合について説明します。次のように操作します。

❶ ダウンロードしたファイル(jdk-7u7-windows-i586.exe)をダブルクリックし、インストーラを実行します。このとき、管理者権限を持たないユーザの場合、管理者権限のパスワードの入力が求められます。管理者権限を持つユーザの場合は、ユーザーアカウント制御による警告ダイアログが表示されるので[はい(Y)]ボタンをクリックします。

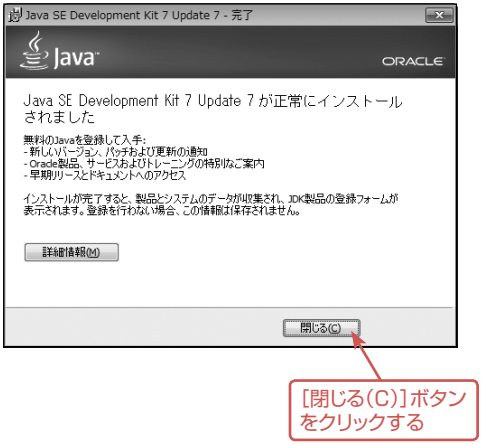
❷ インストールウィザードが起動するので、[次へ(N)]ボタンをクリックします。

❸ JDKをインストールします。必要に応じてインストール先を変更し、[次へ(Y)]ボタンをクリックします。その後、ファイルのコピーが終わるまで待ちます。



❹ 次に、JREをインストールします。必要に応じてインストール先を変更し、[次へ(N)]ボタンをクリックします。なお、インストール先を変更する場合は、JDKとは異なるディレクトリにしてください。その後、ファイルのコピーが終わるまで待ちます。

❺ [閉じる(C)]ボタンをクリックします。



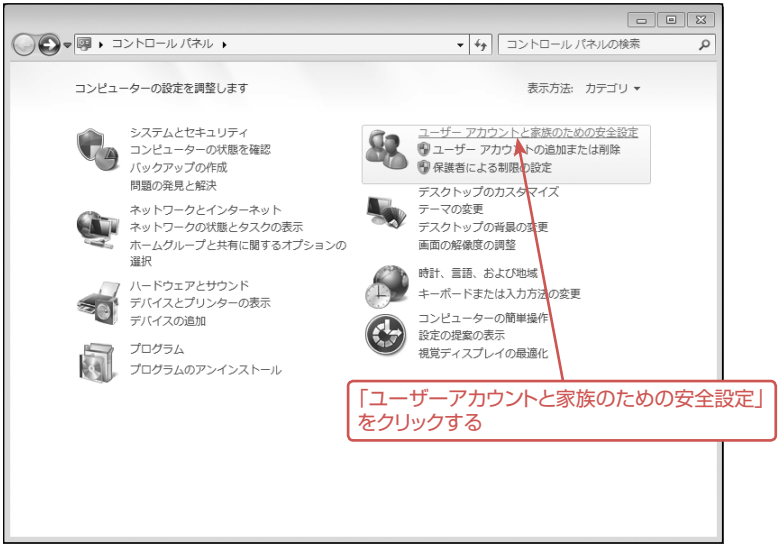
環境変数の設定

次に環境変数の設定を行います。

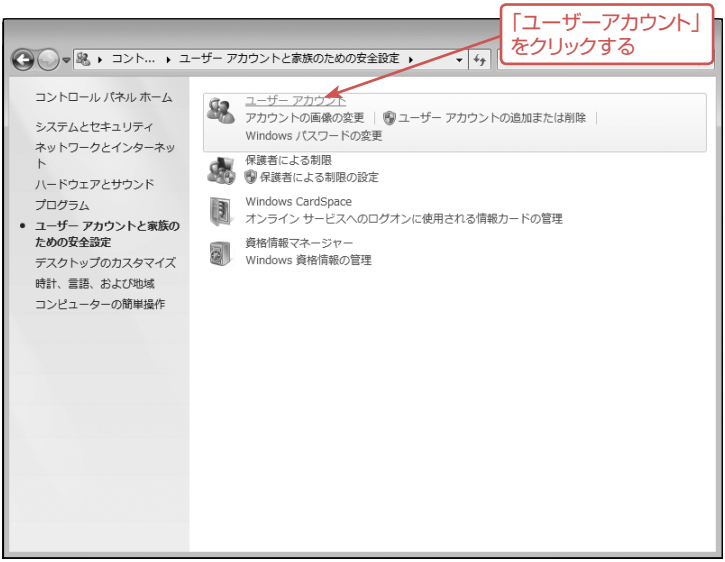
▶ Windowsでの環境変数の設定

Windows系の場合は、次のように操作します。

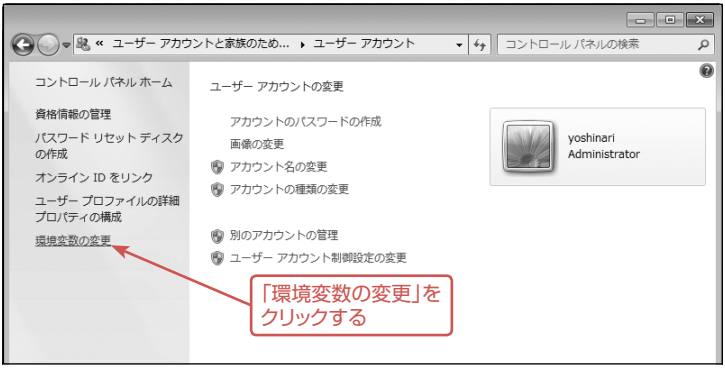
❶ コントロールパネル内の「ユーザーアカウントと家族のための安全設定」をクリックします。



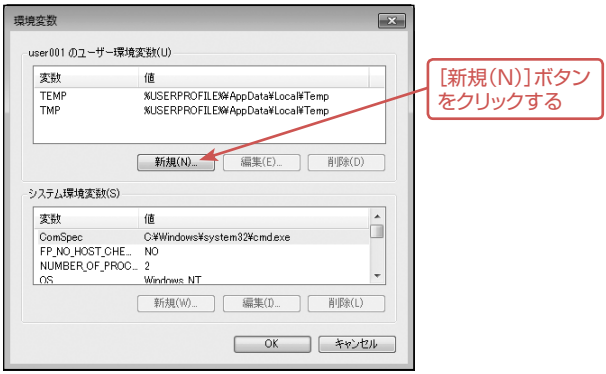
❷ 「ユーザーアカウント」をクリックします。



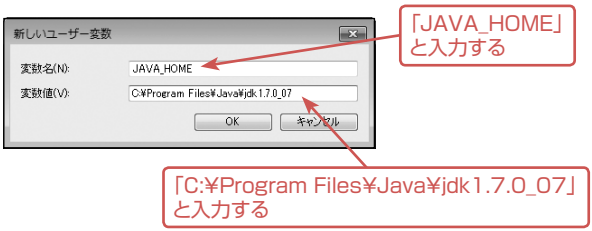
❸ 左側の「環境変数の変更」をクリックします。



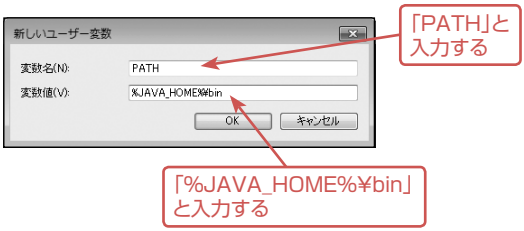
❹ [xxxのユーザー環境変数(U)]にある[新規(N)]ボタンをクリックします。



❺ [変数名(N)]に「JAVA_HOME」、[変数値(V)]に「C:¥Program Files¥Java¥jdk1.7.0_07」(JDKをインストールしたディレクトリ)と入力して、[OK]ボタンをクリックします。



❻ 同様に[新規(N)]ボタンをクリックした後、[変数名(N)]に「PATH」、[変数値(V)]に「%JAVA_HOME%¥bin」と入力して[OK]ボタンをクリックします。その後、「環境変数」ダイアログボックスの[OK]ボタンをクリックして、設定を完了します。



▶ Linuxでの環境変数の設定

Linuxの場合は、「${HOME}/.bashrc」に次の設定を追加します(Bashの場合)。「JAVA_HOME」には、JDKをインストールしたディレクトリを入力してください。

```
export  JAVA_HOME=/opt/jdk1.7.0_07
export  PATH=${PATH}:${JAVA_HOME}/bin
```

▶ Mac OS Xでの環境変数の設定

Mac OS X Lion(10.7)では、標準だと「${HOME}/.bashrc」は読み込まれないので、「${HOME}/.bash_profile」に、次の設定を追加してください。

```
if [ -f ${HOME}/.bashrc ] ; then
  . ${HOME}/.bashrc
fi
```

「${HOME}/.bashrc」には、次のように記述してください。

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home
```

統合開発環境Eclipseのインストールについて

Eclipseについて

Javaのプログラムを作成する際は、統合開発環境を利用すると便利です。Javaの開発でよく利用させる統合開発環境にEclipseがあります。なお、本書では、Eclipseを使った開発を前提としています。

Eclipseのダウロード

公式サイトからEclipseをダウンロードします。次のように操作します。

❶「http://www.eclipse.org/」を表示し、ページ上部のメニューから「Downloads」をクリックします。

❷ 各プラットフォーム、開発言語用のEclipseが表示されたダウンロード画面が表示されるので、自分のプラットフォーム用の「Eclipse IDE for Java EE Developers」のリンクをクリックします。Webアプリケーションを扱わないのであれば、「Eclipse IDE for Java Developers」を選んでも構いません(本書ではCHAPTER 13でWebアプリケーションを扱っているため、「Eclipse IDE for Java EE Developers」を前提としている)。

❸ 次に、ミラー選択画面が表示されるので、任意のミラーサイトを選択してダウンロードします。

Eclipseのインストール

ファイルがダウンロードできたら、Eclipseをインストールします。ここでは、Windows(x86)版について説明します。ダウンロードしたzipファイルを解凍します。ここでは、「C:¥apps」ディレクトリに解凍します。インストールはこれで終わりです。「C:¥apps¥eclipse」というディレクトリができて、その中に「eclipse.exe」という実行ファイルがあるはずです。

Eclipseの日本語化

Eclipseで使われる言葉は、すべて英語です。そこで、Eclipseの日本語化を行います。

PleiadesというプロジェクトでEclipseの日本語化が行われているので、そこからPleiadesパッケージをダウンロードしてEclipseにインストールします。次のように操作します。

❶ まず、EclipseにPleiadesをインストールするディレクトリを作成します。Eclipseをインストールしたディレクトリにある「dropins」ディレクトリの下に「pleiades」というディレクトリを作成し、さらにその下に「eclipse」というディレクトリを作成します。

❷ 次に「http://mergedoc.sourceforge.jp/」のページを表示し、ページ内に「Pleiadesダウンロード」と書かれている場所があるので、そこから最新版、または、安定版をクリックしてダウンロードします。

❸ ダウンロードが終わったら、「pleiades.zip」(または「pleiades_1_x_x.zip」(1_x_xはバージョン番号))を先ほど作成した「pleiades¥eclipse」ディレクトリに解凍します。ディレクトリ構成は、次のようになります。

```
C:.
└─apps
    └─eclipse
        ├─configuration
        ├─dropins
        │  └─pleiades
        │     └─eclipse
        │        ├─features
        │        ├─plugins
        │        └─readme
        ├─features
        ├─p2
        ├─plugins
        └─readme
```

❹ Eclipseをインストールしたディレクトリにある「eclipse.ini」をテキストエディタで編集します。次の1行を最後に追加してください。誌面の都合上、折り返していますが、実際には1行で入力してください。なお、eclipse.iniの改行コードはLFなので、メモ帳ではなく、改行コードLFに対応したエディタかワードパッドで編集してください。

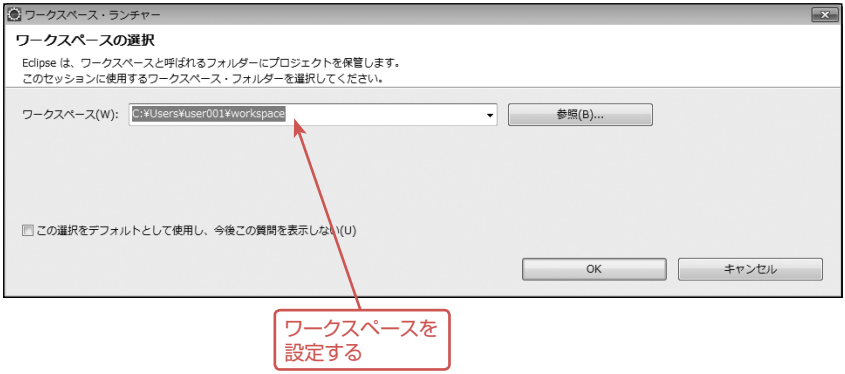
```
-javaagent:C:/apps/eclipse/dropins/pleiades/eclipse/plugins/jp.sourceforge.mergedoc.pleiades/
pleiades.jar
```

これで日本語化は完了です。

Eclipseの起動

次に「ecilpse.exe」を実行して、Eclipseを起動します。起動時にセキュリティの警告が表示されます。[このファイル開く前に常に警告する(W)]をOFFにして実行ボタンをクリックします。

スプラッシュが表示された後、ワークスペースをどこにするか聞かれます。ワークスペースとは、これから自分が作成するプログラムを置く場所です。任意のディレクトリを設定してください。Eclipseを起動するたびに聞かるので、常に同じワークスペースを使う場合はダイアログボックスの左下にある[この選択をデフォルトとして使用し、今後この質問を表示しない(U)]をONにしてください。



JREの設定

Eclipseで使うJREを設定します。ここでJDKを設定しておくと、標準APIのソースファイル情報を見ることができます。

❶ [ウィンドウ(W)]メニューから[設定(P)]を選択します。

❷ 「Java」の中にある「インストール済のJRE」を選択します。このとき、ロケーションに表示されているパスがJDKをインストールしたパスになっていれば、以降の操作は不要です。

❸ [追加(A)]ボタンをクリックします。

❹ 「標準VM」を選択して[次へ(N)]ボタンをクリックします。

❺ [JREホーム(J)]にJDKをインストールしたパス(C:¥Program Files¥Java¥jdk1.7.0_07)を入力します。[JRE名(N)]に任意の名前を入力します。[完了(F)]ボタンをクリックします。

❻ ❺で追加したJREにチェックをONにして[OK]ボタンをクリックします。

COLUMN 「\」と「¥」について

日本語のWindowsではエスケープ文字として使われるバックスラッシュ「\」と半角の円マーク「¥」に同じ文字コードが割り当てられています。そのため、通常は一般的な109キーボードの「Backspace」キーの左のキーを押しても、右SHIFTキーの左のキーを押しても「¥」が表示されます。しかし、フォントを変えると同じ文字でも「\」と表示されます。

たとえば、メモ帳を起動して、「¥」を入力してからメニューの[書式(O)]メニューから[フォント(F)]をクリックしてください。通常、使われる「MS ゴシック」や「メイリオ」などを選択すると「¥」と表示されますが、「Courier」を選択すると「\」と表示されます。

Eclipseの初期設定では「\」が表示されるようになっているため、本書では基本的にこの文字を「\」と記載します。ただし、コマンドラインでは通常「¥」と表示されるので、コマンドラインについて解説する場合は「¥」と記載します。

SECTION-005

はじめてのプログラミング

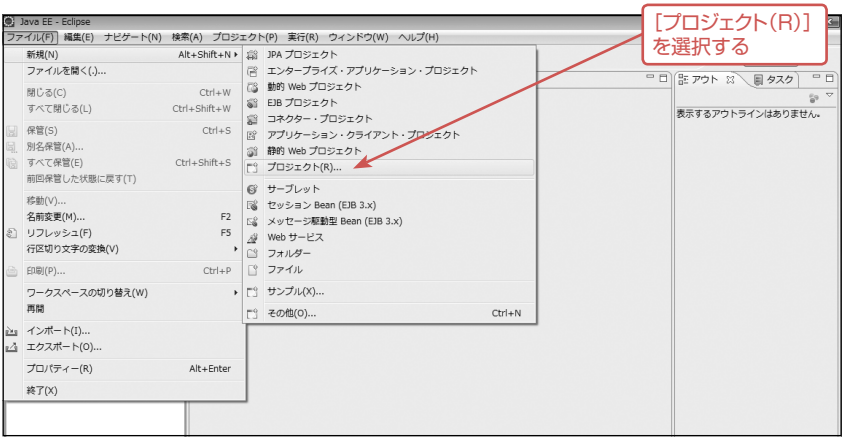
プロジェクトの作成

それではさっそくJavaのプログラムを作成して、動かしてみましょう。ここではEclipseを使ったJavaプログラムの作成、実行方法について解説します。ここで作成するプログラムは「Hello,World!」というものです。画面に「Hello,World!」と表示するだけのプログラムで、開発環境が正しく構築されたかを確認するのによく使われます。

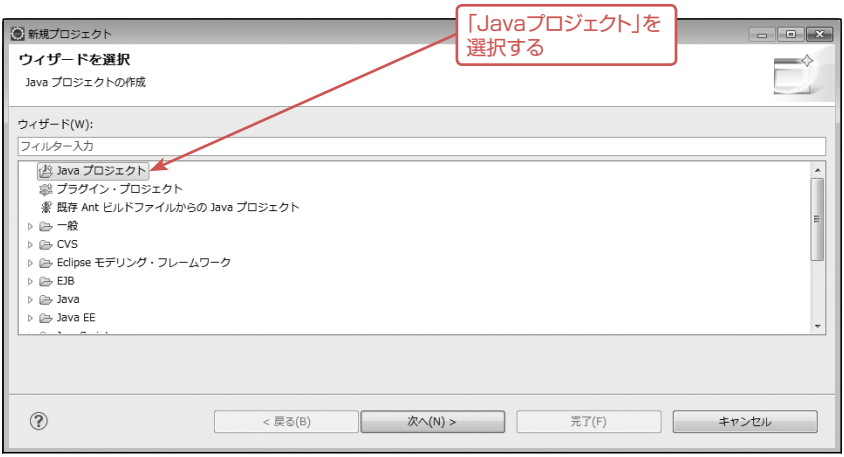
なお、ここではEclipseについて最低限のことのみ解説します。

まず、次のように操作して、プロジェクトを作成します。

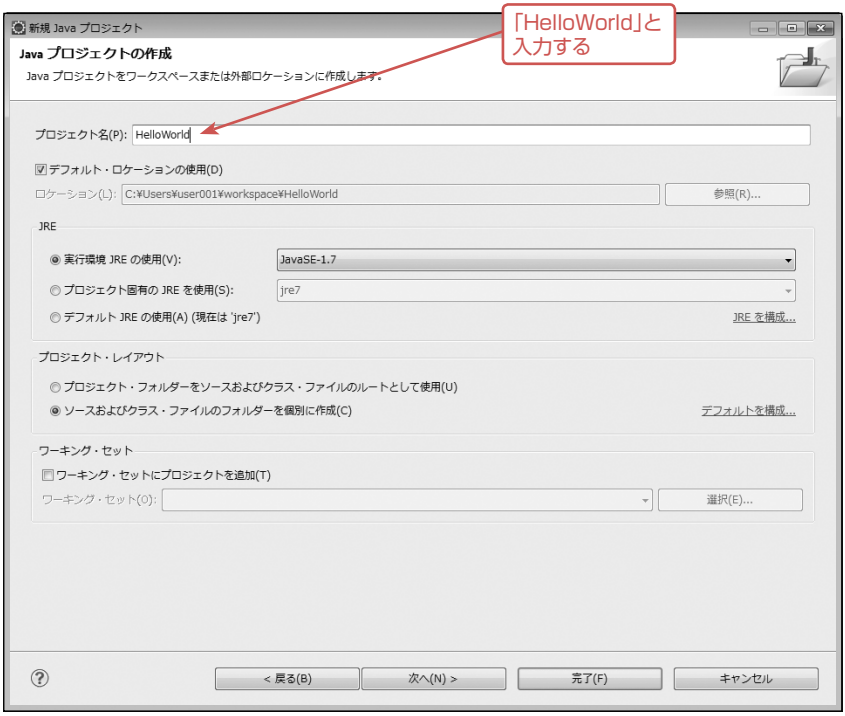
❶ [ファイル(F)]メニューから[新規(N)]→[プロジェクト(R)]を選択します。



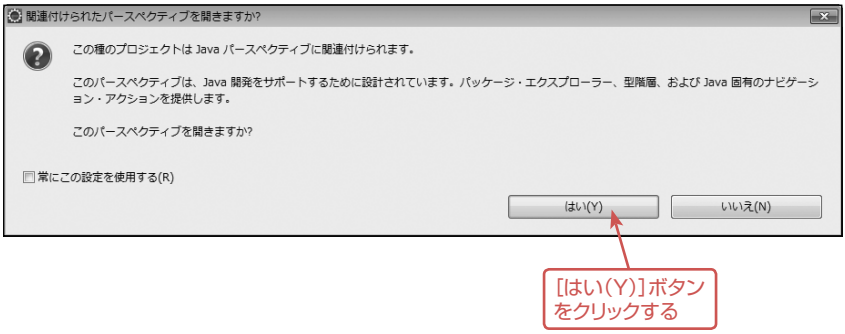
❷「新規プロジェクト」画面が表示されるので、「Javaプロジェクト」を選択して[次へ(N)]ボタンをクリックします。



❸「Javaプロジェクトの作成」画面が表示されるので、[プロジェクト名(P)]に「HelloWorld」と入力し、[完了(F)]ボタンをクリックします。



❹ Javaプロジェクトを開くときにJavaパースペクティブに切り替えてよいかを聞かれます。パースペクティブとはEclipseの画面構成のようなもので、JavaパースペクティブにするとJavaのコーディングに適したレイアウトに、デバッグパースペクティブにするとデバッグに適したレイアウトになります。ここではJavaパースペクティブを使うので、[はい(Y)]ボタンをクリックします。これでプロジェクトの作成は完了です。



プロジェクトの設定

プロジェクトで扱うテキストファイルの文字コードを設定します。本書のコードを手入力する場合はデフォルトの文字コード(Windowsの場合はMS932)のままでかまいませんが、サンプルコードをダウンロードして使う場合は扱う文字コードを「UTF-8」にする必要があります。また、改行コードも「LF」にします。

❶ [プロジェクト(P)]メニューから[プロパティー(P)]を選択します。

❷ プロジェクトのプロパティー画面が表示されるので、左側の「リソース」を選択します。

❸ [テキスト・ファイルのエンコード(T)]で[その他(O)]をONにし、「UTF-8」を選択します。

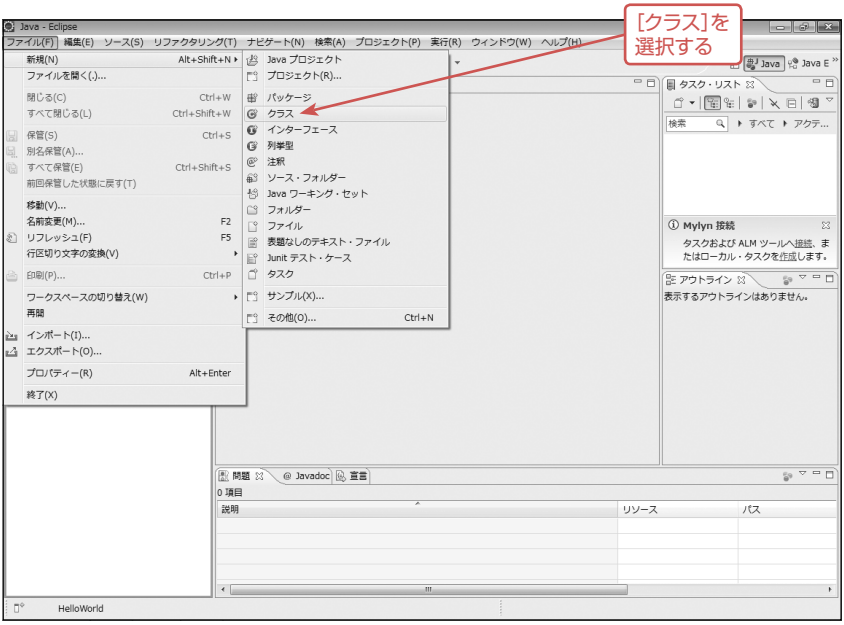
❹ [新規テキスト・ファイルの行区切り文字(F)]で[その他(H)]をONにし、「Unix」を選択します。

❺ [OK]ボタンをクリックします。

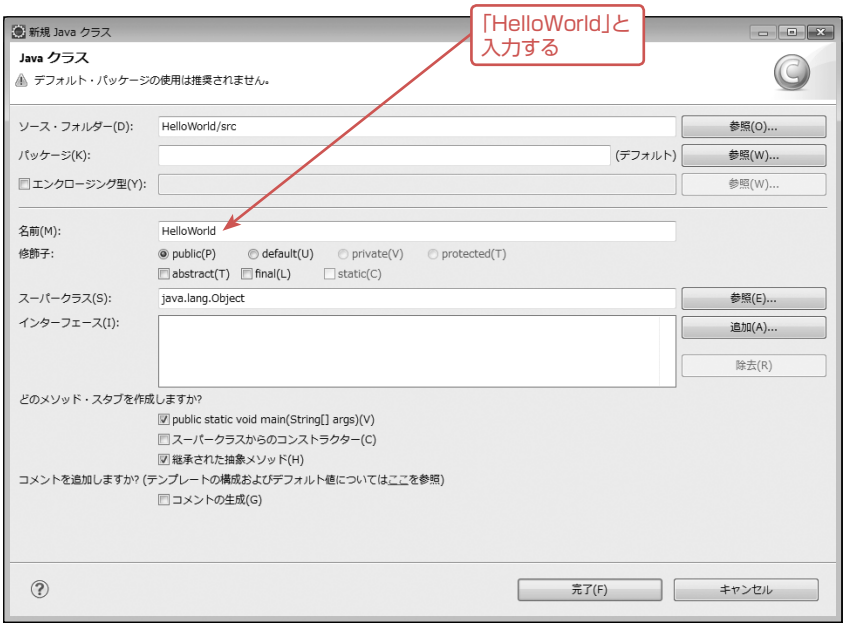
コードの入力

次にクラスを作成します。次のように操作します。

❶ 左側のパッケージエクスプローラ内の「HelloWorld」というプロジェクト名が選択された状態で、[ファイル(F)]メニューから[新規(N)]→[クラス]を選択します。



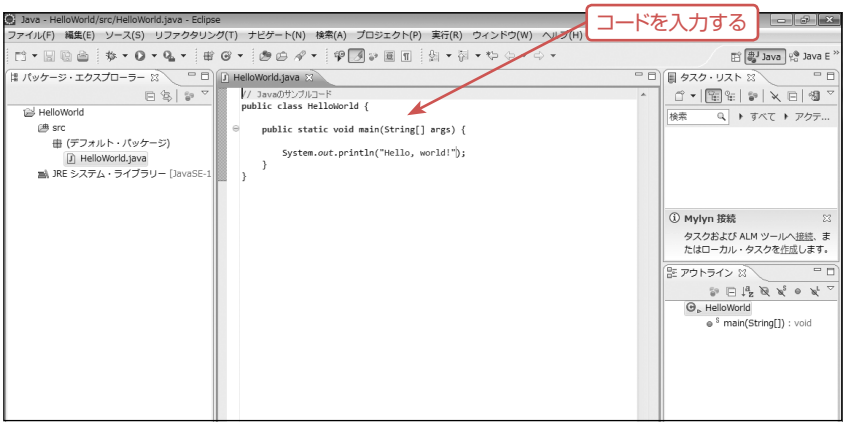
❷ 「Javaクラス」作成画面が表示されるので、[名前(M)]に「HelloWorld」と入力します。また、[public static void main(String[] args)(V)]をONにします。これをONにすると、メインメソッドのスタブが作成されます。Javaのプログラムは必ずメインメソッドから始まるので、プロジェクト内に最低1つはメインメソッドが必要です。



❸ [完了(F)]ボタンをクリックします。

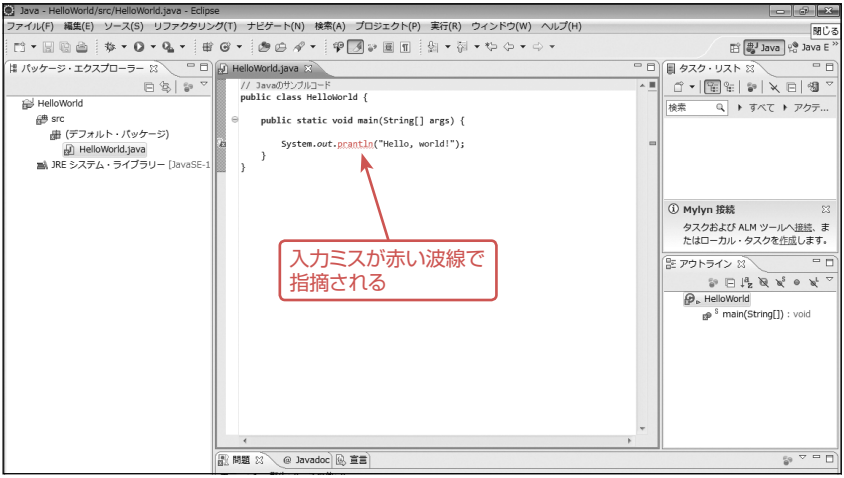
次に、HelloWorldクラスのスケルトンがエディタに表示されるので、次のコードを入力します。

```
// Javaのサンプルコード
public class HelloWorld {

    public static void main(String[]  args) {

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



「System.out.println」メソッドは、パラメータを表示するメソッドです。

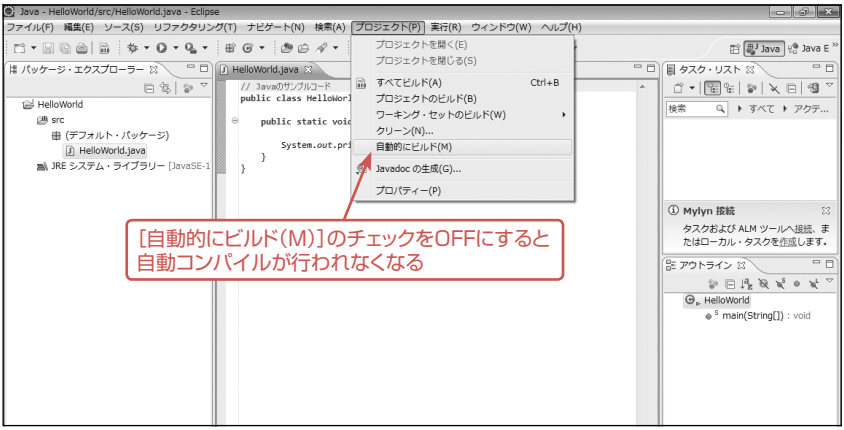
また、入力内容にミスがあると赤い波線で指摘されます。たとえば、「println」というメソッド名を「prantln」と入力していた場合は、次のような画面になります。



コンパイル

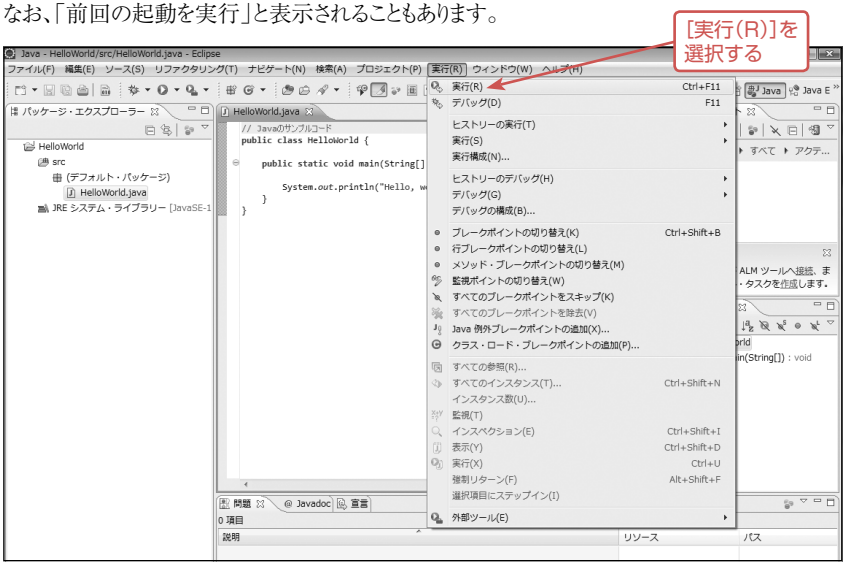
Javaのプログラムを実行するには、コンパイルしてソースファイルからクラスファイルを作成する必要があります。Eclipseではエディタでソースファイルを編集後、その内容を保存するだけでコンパイルは自動で行われます。

マシンパワーの低いPCを使っている場合、ファイルを保存するたびにコンパイルが行われると、そのたびに時間がかかってしまいます。そのような場合は[プロジェクト(P)]メニューから[自動的にビルド(M)]を選択し、チェックをOFFにしてください。[自動的にビルド(M)]のチェックをOFFにした場合、コンパイルするときには、[プロジェクト(P)]メニューから[プロジェクトをビルド(B)](または、[すべてビルド(A)])を選択します。

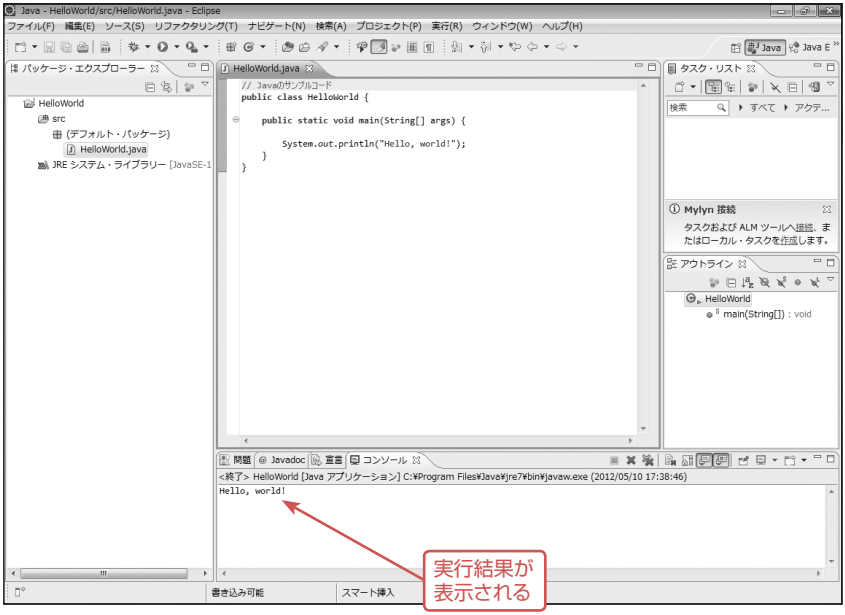


プログラムの実行

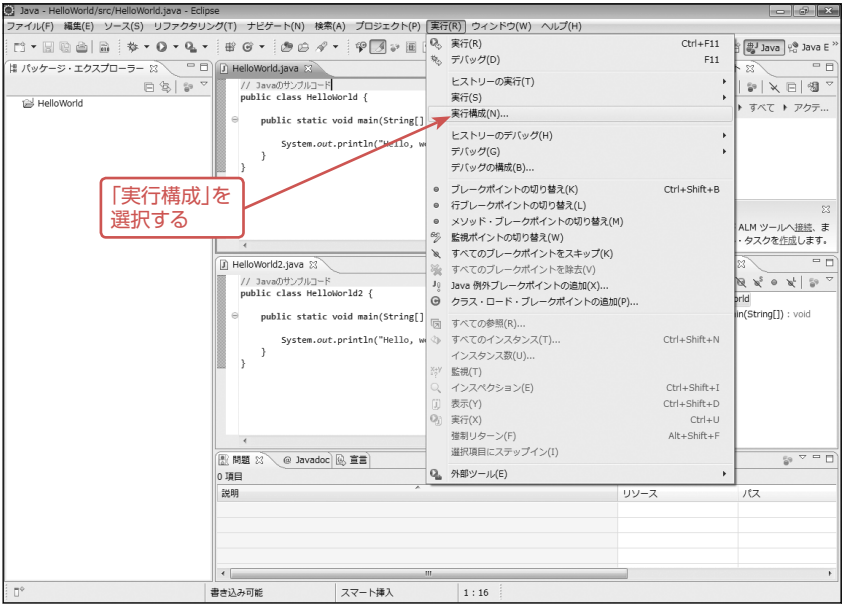
コードの入力が終わったら、実行します。[実行(R)]メニューから[実行(R)]を選択します。なお、「前回の起動を実行」と表示されることもあります。

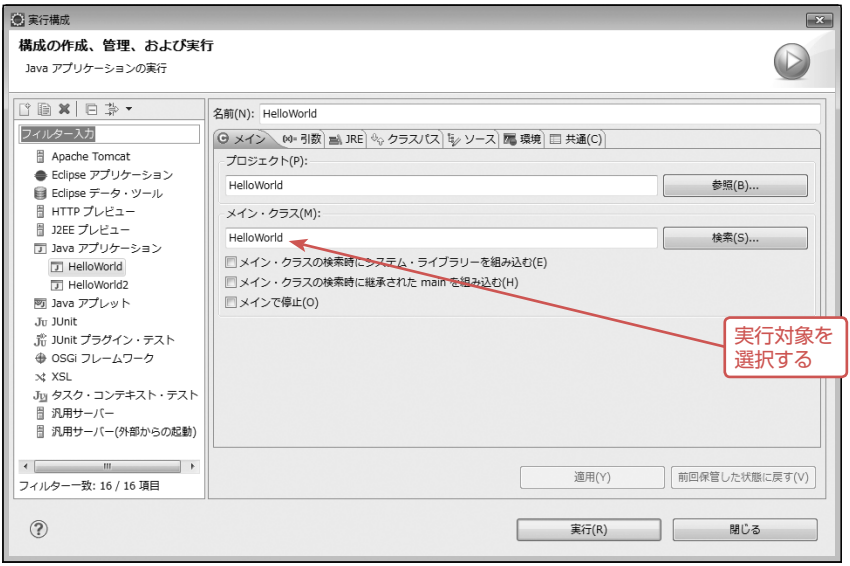


実行すると、画面下部のコンソールに実行結果が表示されます。



対象プロジェクト内に複数のメインメソッドがある場合は、[実行(R)]メニューから[実行構成(N)]を選択します。「実行構成」ダイアログボックスが表示されるので、「メイン」タブ内の[メイン・クラス(M)]で任意のクラスを選択して[実行(R)]ボタンをクリックします。





コマンドラインでのコンパイル

Eclipseを使えない場合、コマンドラインからコンパイルすることができます。メモ帳などのテキストエディタで先ほどと同じソースコード(下記参照)を作成し、「HelloWorld.java」というファイル名で保存します。文字コードは「ANSI」を選択してください。そうすると文字コード「MS932」で保存されます

```
// Javaのサンプルコード
public class HelloWorld {

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

次に、Javaコードのコンパイルを行います。コンパイルを行うには、「javac」コマンドを使います。コマンドプロンプトを起動し、「HelloWorld.java」の保存先にカレントディレクトリを移動した後、次のコマンドを入力します。

```
> javac HelloWorld.java
```

エラーがなければそのまま終了して、クラスファイル(HelloWorld.class)が作成されているはずです。次のように、「dir」コマンドで確認することができます。

```
> dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は XXXX-XXXX です

 c:\apps\java のディレクトリ

2011/12/26  16:16    <DIR>          .
2011/12/26  16:16    <DIR>          ..
2011/12/26  16:17               427 HelloWorld.class
2011/12/26  16:16               132 HelloWorld.java
               1 個のファイル                 132 バイト
               2 個のディレクトリ  1,227,015,270,400 バイトの空き領域
```

コンパイル時には、次のオプションを指定することができます。

オプション	説明
-encoding	ソースファイルの文字コード
-d	クラスファイルの出力先ディレクトリを指定する
-source	特定のバージョン用のソースファイルとしてコンパイルを行う

Javaはバージョンによって、使える識別子名が異なります。たとえば、Java1.4までは「enum」という名前を識別子として使えていましたが、Java5以降では予約語となってしまったために識別子として使うことができません。そのため、Java1.4以前に作られたソースファイルをコンパ

イルするには、次のようにする必要があります。

```
> javac -source 1.4 HelloWorld.java
```

コマンドラインでの実行

実行には「java」コマンドを使います。「java」コマンドを使って実行対象のクラス名を指定します(ファイル名ではない)。パッケージに属するクラスの場合は、パッケージ名も含めた完全クラス名である必要があります。パッケージに関しては、**《パッケージについて》**(p.112)を参照してください。

```
> java HelloWorld
Hello, world!
```

クラスパス

「クラスパス」とは、Javaコードをコンパイルしてできたクラスファイルの場所を指します。クラスパスは複数、指定できるので、プロジェクトが巨大になって数千のクラスファイルがあったとしても、それらすべてを同じディレクトリに格納する必要はありません。

クラスパスは、Javaのコンパイル時と実行時に参照されます。環境変数「CLASSPATH」にパスを設定するか、javaコマンド、または、javacコマンド実行時に、オプションでクラスパスを指定します。例外としてJavaのコアAPIが含まれるrt.jar(JREインストール先にある)とカレントディレクトリは、指定しなくてもクラスパスに含まれます。

クラスパスはクラスファイルのある場所、または、jarファイルを指すようにします(jarファイルの場合はファイル名まで指定する)。Windows系のコマンドプロンプトで一時的に設定する場合は、次のようにします。

```
> set CLASSPATH=c:¥java¥test
> set CLASSPATH=%CLASSPATH%;c:¥work¥sample.jar
```

なお、javaコマンドのオプションで指定する場合は「-cp」オプション、javacコマンドのオプションで指定する場合は「-classpath」オプションを使ってください。

COLUMN ダウンロードしたサンプルファイルをEclipseのプロジェクトに追加するには

ダウンロードしたサンプルファイルをEclipseのプロジェクトに追加するには、標準のエクスプローラでファイルを選択して、Eclipseの「パッケージ・エクスプローラー」ビュー上のソース部分(デフォルトの場合は「src」)にドラッグ&ドロップします。ドラッグ&ドロップすると、「ファイル操作」ダイアログボックスが表示されるので、[ファイルをコピー(C)]をONにして[OK]ボタンをクリックしてください。

ソースファイルがいずれかのパッケージに属するクラスを定義するものであった場合、先にパッケージを作成してから、そのパッケージにドラッグ&ドロップしてください。

エクスプローラやコンソールを使ってソースファイルをEclipseのプロジェクト上のソースフォルダにコピーした場合は、Eclipseの[ファイル(F)]メニューから[リフレッシュ(F)]を選択してください。

COLUMN ダウンロードしたサンプルファイルを「javac」コマンドでコンパイルするには

「javac」コマンドでコンパイルを行う場合、ソースファイルはその環境でのデフォルトの文字コード(MS932)で記述されている必要があります。サンプルファイルは文字コード「UTF-8」で記述されているため、そのままではコンパイルできません。次のように、「javac」コマンドに「-encoding」オプションを付けて文字コードを指定する必要があります。なお、ソースファイルの改行コードは指定する必要がありません。

```
> javac -encoding utf-8 HelloWorld.java
```

CHAPTER 02

Javaの文法

ブロックについて

文

Javaのプログラムの処理単位を「文」といいます。Javaには、「変数宣言文」「式文」「if文」「switch文」「while文」などの文があります。

文の後ろには原則として「;」(セミコロン)を記述します。文は複数行にわたっても構いません。次の2つの文は、「a」という変数に「3 + 2」の計算結果を代入するという同じ処理を表します。

```
a
   =
       3
     + 2;

a = 3 + 2;
```

また、1行内に複数の文を記述することも可能です。

```
int  a = 2;   a = 2 + 3;   a = 5 + 20;
```

どのように記述しても問題はありませんが、ソースコードを見やすくするために、基本的には1行に1文を記述するようにしてください。そして、1行が長くなってしまう場合には、適当なところで改行を入れるようにした方が見やすくなります。

なお、「;」(セミコロン)だけの文を「空文」といいます。空文は何も処理を行いません。

```
;    // 何も処理を行わない
```

ブロック

「{}」(波カッコ)で囲まれた部分を「ブロック」といいます。ブロック内には、複数の文を記述することができます。変数の有効範囲(「スコープ」という)は、そのブロック内のみとなります。

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

        {
            // ブロック内で変数vを宣言
            int  v = 31;

            // ブロック内では参照可能
            System.out.println("v = " + v);
        }

//      ブロックを抜けたため、変数vはここでは参照できない
//      System.out.println("v = " + v);
```

```
        // メソッド内のみで有効な変数xを宣言
        int x = 0;

        // for文のブロック内でのみ有効な変数yを宣言
        for (int y = 0; y < 2; y++) {

            // for文のブロック内でのみ有効な変数zを宣言
            int z = 0;

            System.out.println(
                "x = " + x++ + ", y = " + y + ", z = " + z++);
        }

        System.out.println("x = " + x); // 参照可
//      for文のブロックを抜けたため、変数y、変数zはここでは参照できない
//      System.out.println("y = " + y);
//      System.out.println("z = " + z);
    }
```

ブロック内に記述できるのは、「文」と「局所変数宣言文」と「クラス宣言」だけです。

処理上、ブロックは通常の文とみなされます。そのため、「for」文や「if」文などでよく使われます。

メソッド内に「for」文や「if」文などと関係のないブロックを記述することも可能です。クラス宣言部の中、メソッドの外にブロックを記述すると初期化ブロックとなります。

関連項目 ▶▶▶

● 初期化ブロック、static初期化ブロックについて……p.127

SECTION-007

コメントについて

コメントの種類

コード中に記述する簡単な処理の説明を「コメント」といいます。そこに何が記述されていてもコンパイラは無視するため、処理に影響を与えません。Javaでは3種類のコメントがあります。

「//」で始まるコメントを「行コメント」といいます。「//」からその行の行末までがコメントとなります。

```
// ファイル名を定義
String  name = "sample.txt";
```

「/*」と「*/」で囲まれたコメントをブロックコメントといいます。複数行にわたるコメントを記述するときに使います。

```
/*
 * ここで●●処理を行う。
 * 最大でcntの回数分だけ繰り返す。
 */
for (int i = 0; i < cnt; i++) {
}
```

「/**」から始めると、Javadocコメントとして扱われます。Javadocコメントの記述方法については、《Javadocについて》(p.190)を参照してください。

```
/**
 * これは●●するメソッドです。
 */
public void sample() {
}
```

関連項目 ▶▶▶

● Javadocについて ……p.190

予約語と識別子について

予約語

Javaの言語仕様でキーワードとして使われる特別な言葉を「予約語」といいます。次の言葉が予約語です。

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

なお、「const」「goto」のように、実際には使われていない予約語もあります。

識別子

開発者は、コード中にクラス、メソッド、変数、パッケージ、ラベルなどを定義していきます。これらの名前を「識別子」といいます。文字列をコンソールに表示するのに「System.out.println("Hello")」と記述しますが、「System」はクラス名、「out」はフィールド名(変数名)、「println」はメソッド名で、それぞれ識別子です。識別子には、次の文字が使えます。

- 半角アルファベットの大文字・小文字
- 半角数字
- 「$」「_」
- 読点などの一部例外を除くUnicode文字(全角文字)

半角数字は、先頭文字としては使えません(2文字目以降でしか使えない)。非ASCIIのUnicode文字(いわゆる全角文字)は、慣習的に使われることはほとんどありません。また、次の言葉は識別子として使うことができません。

- 予約語
- 「true」「false」「null」のリテラル

なお、予約語そのものではなく、「ifa」「dos」のように予約語を含む言葉は、問題ありません。

一般的にメソッド名、変数名は小文字+数字で単語の境目の部分だけを大文字にします（例「getCode」「familyName」など）。クラス名は、それに加えて先頭の文字を大文字にします（例「TranslateText」など）。ラベルの場合は、すべて大文字で定義します（例「OUTER」など）。パッケージ名は、すべて小文字で定義します（例「jp.sample.test」など）。

COLUMN 「const」と「goto」

予約語を識別子として使用すると、コンパイルエラーが起きます。これは、予約語が言語として特別な意味を持つ言葉であるためです。

先に述べたように「const」「goto」はJavaでは使用されておらず、本来は予約語である必要はないのですが、これらの言葉が使用されないように予約語として扱っているようです。たとえば、多くの言語では予約語「goto」は指定した位置へ実行を移すという意味を持つので、もし、Javaで「goto」という名前のメソッドが作成できたとしたら混乱のもとになります。

変数について

変数

Javaで値やインスタンスを入れる箱のことを「変数」といいます。値を入れる変数のことをプリミティブ型変数（基本型変数）、インスタンスを入れる変数のことを参照型変数といいます。

どちらの場合も「型名 変数名」の形で宣言します。

```
int    val;   // プリミティブ型
String str;   // 参照型
```

プリミティブ型変数

プリミティブ型とはデータだけを持つ型で、インスタンスではありません。次の8つがJavaのプリミティブ型のすべてです。

データ型	ビット数	初期値	値域
boolean	1	false	trueまたはfalse
char	16	\u0000	\u0000から\uffff （2バイトUNICODE文字）
byte	8	0	−128から127（-2^7から2^7-1）
short	16	0	−32768から32767（-2^{15}から$2^{15}-1$）
int	32	0	−2147483648から2147483647 （-2^{31}から$2^{31}-1$）
long	64	0	−9223372036854775808から 9223372036854775807 （-2^{63}から$2^{63}-1$）
float	32	0	IEEE754規格に従う浮動小数点数
double	64	0	IEEE754規格に従う浮動小数点数

参照型変数

参照型変数について、先ほど「インスタンスを入れる変数」と書きましたが、この表現は正確ではありません。実際には変数内にインスタンスが入っているわけではなく、メモリのどこかにあるインスタンスの場所（インスタンスの参照）を格納している変数です。

参照型変数の型は、Javaで最初から用意されているクラス、Webで公開されているクラス、これから自分で作るクラスなど膨大な数になります。

変数修飾子

変数の前には、次の修飾子を付けることができます。

- アクセス指定子
- final
- volatile
- transient
- static

「final」以外の修飾子は、フィールドでのみ使うことができます。詳しくは、《**フィールド修飾子について**》(p.132)を参照してください。

「final」を付けて宣言した変数には、最初に値を代入した後は他の値を代入することはできません。

```
final StringBuilder  buf = new StringBuilder();

// インスタンス内のデータの変更は可能
buf.append("ABC");

// 他のインスタンスを代入することはできない
// コンパイルエラーになる
buf = new StringBuilder();
```

関連項目 ▶▶▶

- 代入について …… p.65
- フィールド修飾子について …… p.132

代入について

代入

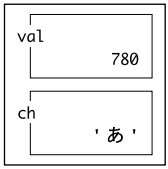
宣言した変数には、値を代入することができます。宣言と同時に代入することも可能です。一度、宣言したら、基本的にその変数にはその型の値、または、インスタンスしか代入することはできません。値の代入には、「=」演算子を使います。「=」の左辺にこれから値を設定する変数、右辺に代入する値もしくは変数を記述します。左辺にリテラルデータを記述することはできません。

```
// int型に整数はそのまま代入できる
// 宣言と同時に代入も可能
int  val = 123;

// int型に真偽型を代入することはできない
// コンパイルエラーになる
val = false;
```

プリミティブ型の場合、変数の中に値を直接、格納します。

```
int   val = 780;
char  ch  = 'あ';
```

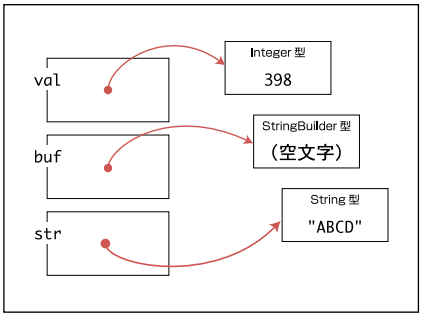
上のようなコードを実行した場合、次のように値が格納されます。



参照型変数への代入も基本的には同じです。代入するときには、リテラルではなく、インスタンスを作成する必要があります。ただし、String型やClass型のように、リテラルを使って代入することができるクラスもあります。リテラルに関しては、《リテラルについて》(p.74)を参照してください。

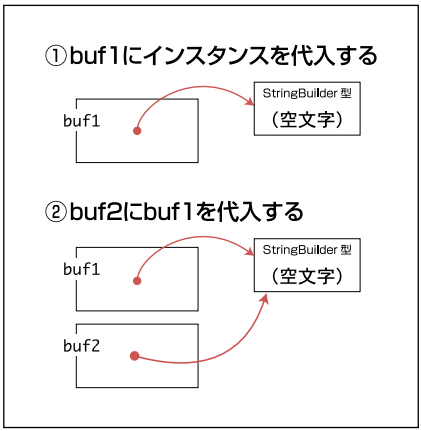
```
// staticメソッドを使ってインスタンスを作成
Integer        val = Integer.valueOf(398);

// new演算子を使ってインスタンスを作成
StringBuilder  buf = new StringBuilder();

// 文字列リテラルによる初期化
String  str = "ABCD";
```

前ページのようなコードを実行した場合、次のように値が格納されます。変数に格納されるのはインスタンスそのものではなく、インスタンスの場所です。



もし、次のようなコードを実行した場合、変数「buf1」と「buf2」は同じインスタンスを指すことになります。

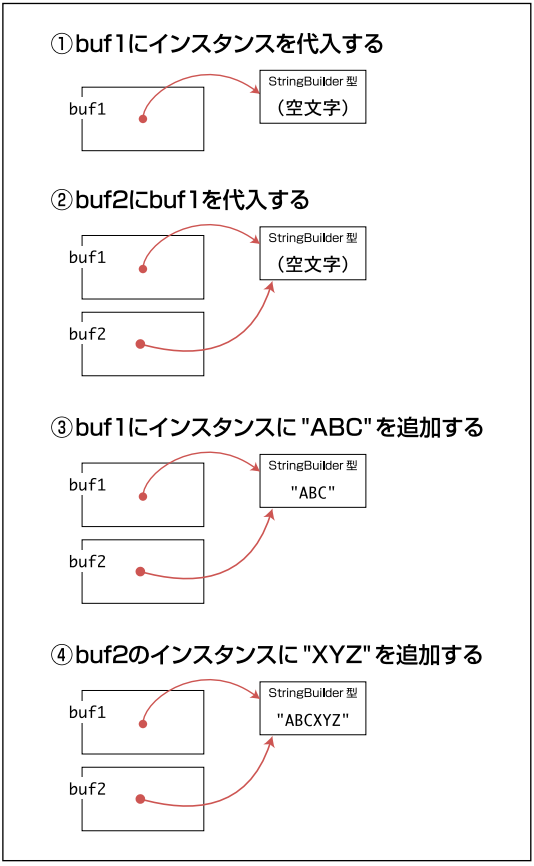
```
StringBuilder  buf1 = new StringBuilder();
StringBuilder  buf2 = buf1;
```



そのため、一方の変数にだけ変更を加えたつもりでも、他方の変数にも影響を与えてしまうので気を付けてください。次のコードを実行すると、「ABCXYZ」と表示されてしまいます。これは、参照型の変数の中身は他の変数への参照値が入っていて、参照型の変数を代入することはその参照値をコピーするためです。「StringBuilder#append」メソッドに関しては、**《文字列を連結する》**(p.265)を参照してください。

```
StringBuilder  buf1 = new StringBuilder();
StringBuilder  buf2 = buf1;

buf1.append("ABC");
buf2.append("XYZ");
```

```
System.out.println(buf1.toString());
```



null

参照型の変数は、インスタンスのある場所を示す変数です。変数を宣言しただけでインスタンスを代入しなかった場合は、何もないことを示す特別な値「null」が入っています。宣言しただけで初期化（何らかのインスタンスの代入）を行っていない変数を使おうとすると、コンパイルエラーが発生します。

```
String  str;

// 初期化していない変数strを使おうとしているため
// コンパイルエラーになる
System.out.println(str.substring(1, 3));
```

「null」は、すべての参照型変数に代入可能です。

```
String  str = null;
Integer val = null;
Object  obj = null;
```

「null」は何もないことを表すので、「null」が代入された変数を使うと「NullPointerException」という例外が発生します。

```
String  str = null;

// NullPointerExceptionが発生する
System.out.println(str.substring(1, 3));
```

ただし、staticフィールドとstaticメソッドはクラスに紐付くので、例外は発生しません。

```
// 以下は正常に動作する
Integer  var = null;
System.out.println(var.valueOf(100));
System.out.println(var.MAX_VALUE);
```

異なる型への代入

プリミティブ型変数の場合は、宣言した変数の型が代入する値の型より広い型であれば代入することができます。

広い方から順に、double型、float型、long型、int型、short型、byte型となります。ただし、int型の値をfloat型に、long型の値をfloat型やdouble型に代入すると桁落ちする可能性があります。

また、char型は文字型ですが、実際は、その文字のUnicodeの文字コードの数値（32bit）を保持しているため、int型と互換があります。

```
// 広い型への代入
byte  b = 3;
short s = b;
int   i = s;
long  l = i;

float  f = i;
double d = f;

// char型をint型に代入
char ch = 'A';
i = ch;
```

参照型変数の場合は、その型のクラスのサブクラスや、型のインタフェースの実装クラスのインスタンスを代入することができます。詳しくは、《**継承と実装について**》（p.159）を参照してください。

◉SuperClass.java

```
// スーパークラス
public class SuperClass {
}
```

◉Interface.java

```
// インタフェース
public interface Interface {
}
```

◉SubClass.java

```
// SuperClassのサブクラス
// Interfaceの実装クラス
public class SubClass extends SuperClass implements Interface {

    public static void main(String[] args) {

        // スーパークラス型の変数にサブクラスのインスタンスを代入
        SuperClass  obj1 = new SubClass();

        // インタフェース型の変数に実装クラスのインスタンスを代入
        Interface   obj2 = new SubClass();
    }
}
```

関連項目 ▶▶▶

- リテラルについて …… p.74
- フィールドについて …… p.133
- メソッドについて …… p.139
- 継承と実装について …… p.159
- オートボクシング/オートアンボクシングについて …… p.184
- 文字列を連結する …… p.265

SECTION-011

キャストについて

プリミティブ型のキャスト

《代入について》(p.65)でプリミティブ型では、狭い型から広い型に代入できることを解説しました。

```
byte  b = 100;
int   i = b;        // 狭い型の変数を広い型に代入するのはOK
```

それとは逆に、広い型から狭い型への代入を行いたい場合は、キャストを使います。キャストは、値の前に変換したい型をカッコで囲んで記述します。

```
int  i = 100;
byte b = (byte) i;  // int型をbyte型に変換(キャスト)する
```

狭い型から広い型への変換は問題ないのですが、その逆は型の範囲を超える値が代入される可能性があります。その場合は桁あふれを起こしてしまい、エラーにはなりません。次のコードでは、byte型の範囲を超える値を代入しているため、桁あふれが起きます。

```
int  i = 200;
byte b = (byte) i;      // int型をbyte型に変換(キャスト)する

System.out.println(b);  // 「-56」と表示される
```

小数型を整数型に変換すると、端数が切り捨てられます。

```
int  i = (int) 4.99;
System.out.println(i);    // 「4」と表示される

i = (int) -4.99;
System.out.println(i);    // 「-4」と表示される
```

また、boolean型とint型のように互換のない型の間でのキャストはできません。

```
int  i = (int) true;      // コンパイルエラーになる
```

オートボクシングを使った広い型から狭い型へのキャストもできません。

```
byte  b = (byte) Integer.valueOf(9);  // コンパイルエラーになる
int   i = Byte.valueOf((byte) 9);     // こちらはOK
```

参照型のキャスト

参照型のキャストも基本的には同じです。スーパークラスからサブクラスへのキャストが可能です。まったく関係のないクラスへのキャストはできません。

```
// SuperClassのサブクラス
public class SubClass extends SuperClass {

    public static void main(String[] args) {

        // スーパークラスからサブクラスへのキャストはOK
        SubClass  obj1 = (SubClass) new SuperClass();

        // 全く関係の無いクラスへのキャストはコンパイルエラーになる
        SubClass  obj2 = (SubClass) Integer.valueOf(9);
    }
}
```

COLUMN 「ClassCastException」例外

キャストが明らかに不正な場合はコンパイルエラーとなりますが、コンパイル時に判断が付かないようなケースではプログラムの実行時に「ClassCastException」例外が発生します。どうしてもキャストを使用しなければならないケースもありますが、できるだけジェネリックスを使うなど、キャストを使わないほうが思わぬミスの防止になります。

```
Object x = new Integer(0);
System.out.println((String)x); // ClassCastExceptionが発生する
```

関連項目 ▶▶▶

- 継承と実装について……p.159
- ジェネリックスについて……p.177
- オートボクシング/オートアンボクシングについて……p.184

定数について

定数の定義

Javaで定数を定義するには、フィールドを宣言するときに「static final」を付けて宣言します。「final」を付けるとフィールドは値を変更できなくなるため、定数として扱うことができます。「static」はなくてもかまいませんが、インスタンスごとに同じ値のフィールドが作成される無駄を省きます。

◉Constant.java

```
// 定数を定義するクラス
public class Constant {

    // 定数フィールド
    public static final String  APP_NAME = "特製アプリケーション";
}
```

◉ConstantSample.java

```
// 定数を使うクラス
public class ConstantSample {

    public static void main(String[]  args) {

        System.out.println(Constant.APP_NAME + "を開始します。");
    }
}
```

ConstantSample.javaを実行すると、次のように表示されます。

```
特製アプリケーションを開始します。
```

厳密には、「Constant.APP_NAME」は定数ではなく、定数フィールドです。Javaでは定数を定義することはできません。ただし、慣習的に定数フィールドのことを「定数」といいます。

定数を使う場合の注意点

コンパイラによりますが、final宣言されたメソッドやフィールドは、それを使っている場所を定数の値に置換してからコンパイルします。これをインライン展開といいます。ConstantSample.javaの「main」メソッドは、次の形で書き換えられてからコンパイルしています。

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

        System.out.println("特製アプリケーションを開始します。");
    }
```

そのため、Constant.javaの定数フィールドの値を変えて、Constant.javaだけをコンパイルしても、ConstantSample.javaの実行結果は変わりません。定数を変更した場合は全クラスをコンパイルし直すようにしてください。

COLUMN　定数とならないケース

static finalのフィールドがすべて定数になるわけではなく、コンパイル時に値が決定するものだけが定数となります。たとえば、次のような例を考えてみます。

```
public static final int X = 1 + 1;                    // 定数
public static final int Y = new Integer(2).intValue();  // 定数ではない
```

この例では、フィールド「X」「Y」ともに値は「2」となります。「X」はコンパイル時に「1 + 1」の算出が行われて定数値となります。しかし、「Y」は「Integer」クラスのインスタンスが作成されてからでないと値が取得できないため、コンパイル時ではなく、プログラムの実行時に値が代入されるので定数ではありません。

両者の違いとしては、前者は先に述べたようにインライン展開が行われますが、後者は実行時にしか値がわからないため、インライン展開されません。そのため、後者は「switch」文の「case」の値として使用できません。「switch」文内の「case」の値は重複してはならないルールがあり、この重複チェックがコンパイル時に行われるためです。

関連項目 ▶▶▶

- フィールド修飾子について ……p.132
- フィールドについて ……p.133

リテラルについて

リテラル

コード中に記述する値そのものを「リテラル」といいます。「a = 100」の「100」、「str = "ABC"」の「"ABC"」がリテラルです。

型によって次のような種類があります。

▶ ブーリアンリテラル

boolean型のリテラル表記です。「true」と「false」の2つだけです。

▶ 文字リテラル

文字1字を表します。char型になります。「'A'」「'あ'」などのように、文字を「'」(シングルクォーテーション)で囲んで表します。

▶ 文字列リテラル

文字列を表します。String型になります。「"ABC"」「"あいうえお"」などのように、文字列を「"」(ダブルクォーテーション)で囲んで表します。

なお、リテラルもインスタンスなので、メソッドを呼び出すことも可能です。

```
// 「BC」と表示される
System.out.println("ABC".substring(1));
```

▶ 整数リテラル

整数を表します。int型になります。「123L」のように、数値の最後に「L」を付けると、long型になります。「0x12ab」のように、数値の先頭に「0x」を付けると16進数表記になります。「0123」のように、数値の先頭に「0」を付けると8進数表記になります。8進数表記時に「8」「9」の数字を使うと、コンパイルエラーになります。

int型は符号付き32bit整数なので、表せる範囲は-2147483648～2147483647です。これを16進数で表すと、0x80000000～0x7fffffffになります。「-0x80000000」と記述するとint型の範囲を超えてしまうため、「0x80000000」にされてしまいます。そのため、「0x80000000 == -0x80000000」の判定結果は「true」となります。

▶ 浮動小数点数リテラル

小数を表します。double型になります。「12.3F」のように、数値の最後に「F」を付けるとfloat型になります。整数部が「0」の場合は「.123」のように省略することができます。

仮数と指数を使った表記もできます。「e」の前に仮数部を、後ろに指数部を記述します。「1.234e2」と「123.4」は同じ値になります。

▶ クラスリテラル

Class型のインスタンスを表します。「String.class」のように、クラス名の後ろに「.class」を付けると、そのクラスのClass型のインスタンスになります。「Class」クラスは、そのクラスのメタ情報(どのようなフィールド、メソッドを持っているかなどの情報)を保持するクラスです。

▶参照リテラル

参照リテラルは、「null」だけです。

エスケープシーケンス

エスケープ記号「\」を使って「改行」などの特殊な制御文字を表したものを、「エスケープシーケンス」といいます。エスケープシーケンスは、次の表のようになります。

エスケープシーケンス	意味
\b	バックスペース
\f	書式送り。印刷時の改ページに使われる
\n	改行。Line Feed
\r	復帰。Carriage Return
\t	水平タブ
\"	ダブルクォーテーション。文字リテラルではエスケープする必要はない
\'	シングルクォーテーション。文字列リテラルではエスケープする必要はない
\\	バックスラッシュ
\uXXXX	文字コードXXXX(16進数)の文字1字を表す
\XXX	文字コードXXX(8進数)の文字1字を表す

文字をUnicodeの文字コードを使って表すこともできます。その場合、「\u」の後ろに16進数の文字コードを記述します。文字リテラルでも、文字列リテラルでも使うことができます。「'\u3042'」は、「あ」を表す文字リテラルです。「"\u3042\u3044\u3046\u3048\u304a"」は、「あいうえお」を表す文字列リテラルです。

「\」の後ろに、8進数の文字コードを記述することも可能です。しかし、「\u0000」～「\u00ff」の範囲しか表すことができないため、ほとんど使われません。「'\101'」は、「A」を表す文字リテラルです。

COLUMN　Java7での新しいリテラル表記

Java7では新しい数値のリテラル表記が追加されました。

「0b」を先頭に付けることで、2進数をリテラル表記できるようになりました。「0b010101」のように記述します。

もう1つ、数値の中にアンダースコアがあっても無視して、普通に数値として扱えるようになりました。桁の大きい数値を見やすく記述することが可能になります。ただし、次の場所には記述することができません。

- 「0x」「0b」の直後
- 「L」「F」などの型を示す文字の直前
- 小数点の直前、直後

```
int     a = 1_000;        // 1000 と同じ
int     b = 0x12__abc;    // 0x12abc と同じ
```

```
int     c = 0_1234;       // 01234 (8進数)と同じ
double  d = 1.234_56;     // 1.23456 と同じ
int     e = 0b_1101;      // エラー
long    f = 1234_L;       // エラー
```

COLUMN 補助文字

Java5で、Unicode4.0に含まれる補助文字がサポートされるようになりました。この補助文字は、16進数5桁で表されます。2byteのchar型では表現できなくなったため、これを「コードポイント」と呼び、int型で表すようになりました。

文字列の中の任意の文字の文字コードを取得するために「String#charAt」メソッドを使っていましたが、補助文字を含む場合は「String#codePointAt」メソッドを使う必要があります。

補助文字の文字コードは、特別な変換式で4byteに変換して上位2byteと下位2byteをUnicodeの未使用領域に割り当てるようにしています。上位2byteを「上位サロゲート」、下位2byteを「下位サロゲート」と呼びます。これは「String#toCharArray」メソッドで取得できます。

補助文字の例として「吉」(上の部分が「士」ではなく「土」になっている)がありますが、これをchar型の変数に代入しようとするとコンパイルエラーになります。

```
char  ch = '吉';    // 4byteデータなのでコンパイルエラーになる
```

演算子について

演算子

「1 + 2」のような計算処理を「演算」といいます。演算内容を意味する記号(「1 + 2」の場合は加算を意味する「+」)のことを演算子(オペレータ)といい、演算の対象となる値や変数(「1 + 2」の場合は「1」と「2」)のことを被演算数(オペランド)といいます。四則演算のような算術演算だけでなく、比較演算や代入演算などの演算子が用意されています。

▶算術演算子

四則演算を行う演算子に剰余を加えたものを算術演算子といいます。算術演算子は次のようになります。

演算子	意味	記述例	記述例の意味
+	加算	a + b	aとbを足す
−	減算	a − b	aからbを引く
*	乗算	a * b	aとbを掛ける
/	除算	a / b	aをbで割る
%	剰余算	a % b	aをbで割ったときの余り
++	インクリメント	a++、++a	aの値を1増やす
−−	デクリメント	a−−、−−a	aの値を1減らす

「+」「-」「*」「/」「%」は、数値型変数と数値リテラルに対して使うことができます。このように、被演算数が2つの演算子を二項演算子といいます。

整数型の場合、除算で割り切れなかった場合、端数は切り捨てられます。

整数型で「0」による除算を行うと「ArithmeticException」例外が発生しますが、浮動小数点数型の「0」による除算では「Infinity」(無限大)が戻ります。

被演算数の型が異なるもの同士で演算を行った場合、演算結果は広い型になります。「広い型」については、《代入について》(p.65)を参照してください。なお、同じ型同士の演算でも、byte型、short型での演算結果は、int型になります。

```
long  a = 12 + 234L; // intとlongの演算結果はlong型になる

byte  b = 100;
short c = 200;
int   d = b + c;     // byte, shortのみでの演算結果はint型になる
```

「++」「--」は、数値型変数のみで使うことができます。変数の前についている場合は値を1増やしてから(または1減らしてから)、その変数の中身を参照します。変数の後ろについている場合は、その変数の中身を参照してから値を変えます。被演算数が1つなので単項演算子といいます。

```
int  a = 3;
```

```
int  b = a++;  // bには3が代入され、aの値は4になる

int  c = 3;
int  d = ++c;  // cの値は4になり、dには4が代入される
```

▶ビット演算子・シフト演算子

ビット演算子・シフト演算子は、次のようになります。

演算子	意味	記述例	記述例の意味
&	論理積	a & b	aとbの論理積
|	論理和	a | b	aとbの論理和
^	排他的論理和	a ^ b	aとbの排他的論理和
~	否定	~a	aの否定
<<	符号付き左シフト	a << b	aをbだけ左に算術シフト
>>	符号付き右シフト	a >> b	aをbだけ右に算術シフト
>>>	符号なし右シフト	a >>> b	aをbだけ右に論理シフト

Javaでは、数値をすべて2進数で処理しています。普通の四則演算以外にも、「0」と「1」を使って数値の組み合わせによる演算も行えます。それを「ビット演算」といいます。ビット演算の対象は、整数のみです。

論理積は両方が「1」だった場合のみ「1」になり、それ以外の場合は「0」になります。論理和はいずれか一方が「1」だった場合に「1」になり、両方が「0」だった場合は「0」になります。排他的論理和は両方の値が異なれば「1」になり、両方の値が同じならば「0」になります。否定は「0」を「1」に、「1」を「0」にします。

2つの値とそのビット演算結果は、次のようになります。なお、数値を2進数に変換して、すべてのビットに対して演算します。

aの値	bの値	&の結果	|の結果	^の結果
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

シフト演算子は2進数で表した値を右、または、左にずらします。右にずらした場合は最も左に、左にずらした場合は最も右に「0」を追加し、はみ出した部分は切り捨てます。「>>」は一番左のビット(最上位ビット)が「1」だった場合、最も左に「1」を追加します。

```
// 「41」は2進数で「101001」
int  a = 41;

// 「12」(1010)が表示される
System.out.println(a >> 2);

// 「164」(10100100)が表示される
System.out.println(a << 2);
```

```
// 「-41」は2進数で「11111111111111111111111111010111」
a = -41;

// 「-11」(11111111111111111111111111110101)が表示される
System.out.println(a >> 2);

// 「1073741813」(11111111111111111111111111110101)が表示される
System.out.println(a >>> 2);
```

▶比較演算子

比較演算子は、2つの値を比較します。演算結果は、必ずboolean型です。判定が成り立つ場合は「true」、成り立たない場合は「false」を戻します。比較演算子は、次のようになります。

演算子	意味	記述例	記述例の意味
>	左が大きい	a > b	aがbより大きいかを判定
>=	左が等しいか大きい	a >= b	aがb以上かを判定
<	左が小さい	a < b	aがbより小さいかを判定
<=	左が等しいか小さい	a <= b	aがb以下かを判定
==	式が等しい	a == b	aとbが等しいかを判定
!=	式が等しくない	a != b	aとbが等しくないかを判定

「<」「>」を含む演算子は、数値のみに使えます。「==」「!=」は、すべての型で使えます。**《代入について》**(p.65)で説明したように、参照型変数にはインスタンスの場所が格納されているだけです。「null」かどうかの判定以外でインスタンスの比較する場合は、「Object#equals」メソッドや、そのクラスで実装されている比較用のメソッド(「compareTo」メソッドなど)を使ってください。次のコードを実行すると、「等しくない」と表示されます。

```
// aとbは同じ「ABC」という文字列だが異なるインスタンス
String  a = "ABC";
String  b = new String("ABC");

if (a == b) {
    System.out.println("等しい");

} else {
    System.out.println("等しくない");
}
```

▶論理演算子

論理演算は、boolean型のみ使います。論理演算子は、次のようになります。

演算子	意味	記述例	記述例の意味
&	論理積	a & b	aとbの論理積
&&	条件積	a && b	aとbの論理積
|	論理和	a | b	aとbの論理和
||	条件和	a || b	aとbの論理和
^	排他的論理和	a ^ b	aとbの排他的論理和

2つの値とその論理演算結果は次のようになります。

aの値	bの値	&の結果	&&の結果	|の結果	||の結果	^の結果
false	false	false	false	false	false	false
false	true	false	false	true	true	true
true	false	false	false	true	true	true
true	true	true	true	true	true	false

「&」と「&&」、「|」と「||」は、結果は同じです。これらの違いは、「&&」と「||」は左辺の評価で「true」か「false」かが決まった場合、右辺の評価を行わないということです。文字列が「null」かどうかを判定して、「null」でなければ「ABC」という文字列かを判定する場合、次のような違いがあります。

```
String  str = null;

// 「null != str」の判定時点でfalseになるので
// 「str.equals("ABC")」は実行されない
if (null != str && str.equals("ABC")) {
    System.out.println("OK");

} else {
    System.out.println("NG");
}


// 「null != str」と「str.equals("ABC")」を判定してから
// 論理積をとるのでNullPointerExceptionが発生する
if (null != str & str.equals("ABC")) {
    System.out.println("OK");

} else {
    System.out.println("NG");
}
```

▶代入演算子

代入演算子は、変数に値を代入するのに使います。単に値を代入するだけでなく、演算結果を代入することもできます。

演算子	記述例	意味	「=」を使った記述
=	a = b	aにbを代入	なし
+=	a += b	aにa + bの結果を代入	a = a + b
−=	a −= b	aにa − bの結果を代入	a = a − b
*=	a *= b	aにa * bの結果を代入	a = a * b
/=	a /= b	aにa / bの結果を代入	a = a / b
%=	a %= b	aにa % bの結果を代入	a = a % b
<<=	a <<= b	aにa << bの結果を代入	a = a << b
>>=	a >>= b	aにa >> bの結果を代入	a = a >> b
>>>=	a >>>= b	aにa >>> bの結果を代入	a = a >>> b

演算子	記述例	意味	「=」を使った記述
&=	a &= b	aにa & bの結果を代入	a = a & b
|=	a |= b	aにa | bの結果を代入	a = a | b
^=	a ^= b	aにa ^ bの結果を代入	a = a ^ b

▶条件演算子

条件演算子は、少し特殊です。「a ? b : c」の形で記述します。「a」が「true」の場合は「b」を、「false」の場合は「c」を戻します。「if」文を使うと、次のように書き換えることができます。

```
int  x;

// 条件演算子を使った場合
x = a ? 100 : 200;

// if文を使った場合
if (a) {
    x = 100;

} else {
    x = 200;
}
```

被演算数が3つあるのはこの演算子だけなので、三項演算子ともいいます。

▶文字列の演算子

String型は「+」演算子を使うと、文字列を結合することができます。

```
String  str1 = "ABC";
String  str2 = str1 + "XYZ";  // "ABCXYZ"がstr2に代入される
```

String型とString型以外を「+」で演算すると、String型以外の被演算数は文字列表現に置き換えられて、文字列同士の結合となります。プリミティブ型であれば、値がそのまま文字列になります。参照型変数であれば、「toString」メソッドの結果が結合されます。「null」だった場合は、「null」という文字列になります(「NullPointerException」例外は発生しない)。

また、「+=」演算子も同様に使うことができます。

```
String  str = "ABC" + 123; // "ABC123"という文字列になる
Integer val = null;
str += val;                // "ABC123null"という文字列になる
```

▶その他

カッコは、優先順位の低い演算子による演算を優先して行う場合に使います。

```
int  a = 3 + 5 * 2;      // aには13が代入される
int  b = (3 + 5) * 2;    // bには16が代入される
```

また、メソッドやコンストラクタを呼び出すときにも使います。

```
Date  d = new Date();
long  l = d.getTime();
```

「.」演算子は、メソッドの呼び出し元やフィールドを保持するクラス、インスタンスを指定するために使います。

```
String  str = "ABC";
System.out.println(str.substring(1));
System.out.println(Integer.MAX_VALUE);
```

「[]」は、配列の何番目のデータを使うかを指定したり、配列のサイズを指定したりするために使います。

```
int[]  values = new int[3];
values[0] = 100;
```

「instanceof」演算子は、インスタンスの型を判定するのに使います。左オペランドのインスタンスが右オペランドのクラスにキャスト可能な場合、「true」を返します。詳しくは、**《インスタンスが特定のクラスを継承しているか調べる》**(p.213)を参照してください。

```
void  castMethod(Object  obj) {
    if (obj instanceof Integer) {
        Integer  val = (Integer) obj;
        System.out.println(val);
    }
}
```

演算子の優先順位

Javaでは、複数の演算子を一度に使うことができます。基本的に左側から順に処理していきますが、演算子には優先順位があり順序が入れ替わることがあります。

演算子の優先順位は、次のようになっています(記載順で優先順位が高い)。

- ()、[]、.
- ++、--、~、!
- *、/、%
- +、-
- <<、>>、>>>
- >、>=、<、<=、instanceof
- ==、!=
- &
- ^
- |
- &&

- ||
- ?:（条件演算子）
- =（代入演算子）

なお、代入演算子だけは、右側から処理していきます。

```
int  a = b = c = 100;  // a, b, c すべてに100が代入される
```

COLUMN　定数の計算式

定数の計算式は実行時ではなく、コンパイル時に計算されます。文字列を「+」演算子で結合したときも同様です。

```
int i = 1 + 2 + 3 * 4;
String s = "Hello" + " " + "World";
```

上記の例は、次の記述と同じことです。

```
int i = 15;
String s = "Hello World";
```

関連項目 ▶▶▶

- 代入について …… p.65
- 「if」文について …… p.84
- オートボクシング/オートアンボクシングについて …… p.184
- インスタンスが特定のクラスを継承しているか調べる …… p.213
- インスタンスの文字表現を取得する …… p.275

「if」文について

「if」文の記述方法

処理を分岐させたいときには、「if」文を使います。「if」文は、次のように記述します。

```
if (評価式)
    (評価式の値がtrueだった場合に行う文);
```

```
if (評価式)
    (評価式の値がtrueだった場合に行う文);
else
    (評価式の値がfalseだった場合に行う文);
```

評価式の値は、必ず、boolean型でなければいけません。boolean型の変数、戻り値がboolean型のメソッド呼び出し式、論理演算式などを記述します。

評価結果が「true」の場合は評価式の次の文を実行し、「false」の場合はその文をスキップして次の処理に移ります。「else」を使うと評価結果が「false」だった場合の文も記述できます。

複数の式の評価を行いたい場合は、「かつ」を表す「&&」演算子や、「または」を表す「||」演算子を使います。

複数の処理を記述したい場合は、文をブロックにします。

```
if (評価式) {
    (評価式の値がtrueだった場合に行う文1);
    (評価式の値がtrueだった場合に行う文2);

} else {
    (評価式の値がfalseだった場合に行う文1);
    (評価式の値がfalseだった場合に行う文2);
}
```

処理が1つの場合にブロックを使わずに記述することを、「ぶら下がり文」といいます。ただし、ぶら下がり文を使うと、少し複雑な「if」文で勘違いを起こしやすくなるので注意してください。

たとえば、次のコードで「c = a + b」の処理が行われるのは「a >= 0」の場合ではなく、「a < 0」かつ「b >= 0」の場合です。

```
if (a < 0)
    if (b < 0)
        c = 0;
else
    c = a + b;
```

else if

「ぶら下がり文」を使ってelseの後ろに、さらに「if」文を記述することもできます。次の2つの「if」文は、同じ処理を行います。

```
// elseの中にif文を記述
if (a >= 100) {
    System.out.println("100以上");

} else {

    if (a >= 50) {
        System.out.println("50以上100未満");

    } else {
        System.out.println("50未満");
    }
}


// 「else if」を使って記述
if (a >= 100) {
    System.out.println("100以上");

} else if (a >= 50) {
    System.out.println("50以上100未満");

} else {
    System.out.println("50未満");
}
```

関連項目 ▶▶▶

● 「switch」文について …… p.86

「switch」文について

「switch」文の記述方法

「switch」文も、「if」文同様に、条件分岐に使います。主に分岐が多い場合に使われます。「switch」文は、次のように記述します。

```
switch (評価式) {
case  定数値1:
    // 評価式の値が定数値1だった場合に行う処理
case  定数値2:
    // 評価式の値が定数値2だった場合に行う処理
default:
    // 評価式の値がいずれの定数値でもなかった場合に行う処理
}
```

上の例では定数値を2つ記述しましたが、いくつでも記述可能です。評価式の値が「case」の後ろの定数値と一致した場合、その「case」の後の文を実行します。そのため、評価式の値の型と、各定数値の値の型は一致する必要があります。

いずれの値とも一致しなかった場合は、「default」の後の文を実行します。「default」は、省略可能です。「default」は、「switch」文の途中に記述しても問題ありません。

評価式の部分には、通常、プリミティブ型、または、Enum型を記述する必要があります。オートアンボクシングによって自動変換が行われるため、プリミティブ型のラッパークラスも記述することができます。Enum型もプリミティブ型のラッパークラスも、どちらも「null」を判定しようとすると「NullPointerException」例外が発生するので、「null」チェックを行ってから判定するようにしてください。

定数値には、リテラル表記、または、コンパイル時に値の決まっているfinal宣言された変数を記述する必要があります。

```
final int  a = get();    // コンパイル時にaの値が決まらない

switch (x) {
case  a:                 // aの値が不定なのでエラーになる
}
```

「if」文とは異なり、enum型を使う場合には比較対象の型がすでに決まっているため、「case」に型名を記述する必要はありません。

```
enum  Type {
    A, B, C
};
```

```
void method1(Type  t) {

    // switch文の場合、caseに型名の記述は不要
    switch (t) {
    case  A:
        System.out.println("Aを受け取った");
        break;
    default:
        System.out.println("Aを受け取らなかった");
        break;
    }

    // if文の場合、型名を記述しないとコンパイルエラーになる
    if (t == Type.A) {
        System.out.println("Aを受け取った");

    } else {
        System.out.println("Aを受け取らなかった");
    }
}
```

「break」文

「case」「default」は処理の分岐先を示すラベルです。そのため、分岐先に複数の文を記述する場合でも「if」文のようにブロックで囲む必要はありません。また、次の「case」「default」ラベルのある行に処理が遷移した場合は、そのまま処理を続行します。次のラベルに処理が遷移する前に「switch」文を抜けるためには、「break」文を使います。

```
switch (x) {
case  0:
    System.out.println("x = 0 です");
    break;

case  1:
    System.out.println("x = 1 です");

case  2:
    System.out.println("x = 2 です");

default:
    System.out.println("xの値は不定です");
}
```

上記コードで「x=0」の場合は次のように表示されます。

```
x = 0 です
```

「x=1」の場合は、途中に「break」がないため、「case 1:」から「switch」文の最後まで処理が実行され、次のように表示されます。

```
x = 1 です
x = 2 です
xの値は不定です
```

COLUMN Stringによる「switch」文

Java7からは、String型変数を「switch」文で判定できるようになりました。

```
String  str = "ABC";

switch (str) {
case "ABC":
    System.out.println("ABCです");
    break;

case  "XYZ":
    System.out.println("XYZです");
    break;

default:
    System.out.println("ヒットしませんでした");
    break;
}
```

Javaの言語仕様が変わったわけではなく、オートボクシング/オートアンボクシングと同様に、コンパイラがコードの変換を行っています。上記コードは、次の形に変換されます。

```
byte byte0 = -1;
switch (str.hashCode()) {
case 64578:
    if(str.equals("ABC"))
        byte0 = 0;
    break;

case 87417:
    if(str.equals("XYZ"))
        byte0 = 1;
    break;
}

switch(byte0) {
case 0:
    System.out.println("ABCです");
```

```
        break;

    case 1:
        System.out.println("XYZです");
        break;

    default:
        System.out.println("ヒットしませんでした");
        break;
    }
```

「64578」は、文字列「ABC」のハッシュコードです。「87417」は、「XYZ」のハッシュコードです。一度、文字列のハッシュコードを取得して、それを使って「switch」文で分岐しています。ハッシュコードによる識別は100%確実ではないため、「case」の中でさらに「if」文で確認を行い、結果を「byte0」に格納します。そして、もう1つ「switch」文を作成し、「byte0」の値で処理を分岐します。

文字列のハッシュコードを取得する必要があるため、判定対象の文字列は「null」であってはいけません。

関連項目 ▶▶▶

- 「if」文について …… p.84
- 「break」文について …… p.98
- オートボクシング/オートアンボクシングについて …… p.184

「for」文について

「for」文

繰り返し処理(「ループ」ともいう)を行うには、「for」文を使います。「for」文は、次のように記述します。

```
for (初期化式; 条件式; 更新式)
    (繰り返し処理);
```

「if」文と同様に、繰り返し処理が1つの文の場合はこのままで構いませんが、複数の文からなる場合はブロックを使って記述します。

初期化式では、繰り返し処理のカウンタとして使われる変数(ループカウンタ)の初期化を行います。ここで変数宣言を行うと「for」文のブロック内でしか参照できないので、「for」文の外で参照したい場合は変数宣言だけを先に行っておく必要があります。

条件式には、「for」文を続ける条件を記述します。条件判定は最初の繰り返し処理を行う前と、繰り返し処理を終えて更新式を実行した後に行います。判定結果が「true」の場合は繰り返し処理を実行し、「false」の場合は「for」文を終了します。

更新式には、「for」文の処理を終えた後に実行する式を記述します。具体的には、ループカウンタ変数を1増やす処理などを記述します。

「1」〜「10」の数字を表示する場合、「for」文は次のように記述します。

```
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

初期化式、条件式、更新式は、それぞれ省略可能です。上記コードは、次のように記述することも可能です。

```
int i = 1;
for (; i <= 10;) {
    System.out.println(i++);
}
```

後述するリストやジェネリックスを使った繰り返し処理を行う際は、次のような「for」文を記述することもできます。

```
void  sample(List<String>  list) {

    for (Iterator<String>  ite = list.iterator();
         ite.hasNext(); ite.hasNext()) {
```

```
            System.out.println(ite.next());
        }
    }
```

初期化式、更新式には、複数の式を記述することが可能です。

◉ForSample.java

```
// for文 複数の初期化、更新を行うサンプル
public class ForSample {

    public static void main(String[] args) {

        // iかjのいずれかが3以上になったら繰り返し処理完了
        for (int  i = 0, j = 0; i < 3 && j < 3; i++, j++) {
            System.out.println("i=" + i + ", j=" + j);
        }
    }
}
```

ForSample.javaを実行すると、次のように表示されます。

```
i=0, j=0
i=1, j=1
i=2, j=2
```

「拡張for」文

他の言語で「foreach」文といわれる繰り返し処理も、Javaでは「for」文で行います。従来の「for」文と区別するため、「拡張for」文といわれます。「拡張for」文は、次のように記述します。

```
    for (ループ変数 : 集合) {
        // 繰り返す処理
    }
```

ここでいう「集合」というのは、配列、または、「Iterable」インタフェースを実装したクラスのインスタンスです。後述するコレクションフレームワークという集合を扱うクラス群がありますが、そのほとんどが「Iterable」インタフェースを実装、継承しています。

◉EnhancedForSample.java

```
// 拡張for文のサンプルクラス
import java.util.ArrayList;

public class EnhancedForSample {

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

```
        int[]  values = {1, 3, 5, 7};

        // 配列を使った拡張for文
        for (int  val : values) {
            System.out.println("val = " + val);
        }

        System.out.println("--------------------");

        // 匿名クラスを使ってArrayListを初期化
        ArrayList<String>  list = new ArrayList<String>() {
            {
                this.add("ABC");
                this.add("DEF");
                this.add("GHI");
            }
        };

        // Iterableを使った拡張for文
        for (String  str : list) {
            System.out.println("str = " + str);
        }
    }
}
```

EnhancedForSample.javaを実行すると、次のように表示されます。

```
val = 1
val = 3
val = 5
val = 7
--------------------
str = ABC
str = DEF
str = GHI
```

多重ループ

繰り返し処理の中に、さらに繰り返し処理を記述することが可能です。

◉MultiplicationTable.java

```
// 二重ループを使って九九を表示するクラス
public class MultiplicationTable {

    public static void main(String[] args) {

        for (int i = 1; i <= 9; i++) {
```

```
            for (int j = 1; j <= 9; j++) {
                System.out.printf("|%2d", i * j);
            }

            System.out.println("|");
        }
    }
}
```

MultiplicationTable.javaでは、「j」を使った内側の繰り返し処理で横方向に9回、計算結果を表示していきます。内側の繰り返し処理が終了したら、最後の「|」を表示した後に改行しています。この処理(「i」を使った外側の繰り返し処理)を9回繰り返しています。MultiplicationTable.javaを実行すると、次のように表示されます。

```
| 1| 2| 3| 4| 5| 6| 7| 8| 9|
| 2| 4| 6| 8|10|12|14|16|18|
| 3| 6| 9|12|15|18|21|24|27|
| 4| 8|12|16|20|24|28|32|36|
| 5|10|15|20|25|30|35|40|45|
| 6|12|18|24|30|36|42|48|54|
| 7|14|21|28|35|42|49|56|63|
| 8|16|24|32|40|48|56|64|72|
| 9|18|27|36|45|54|63|72|81|
```

三重、四重ループにすることも可能ですが、繰り返し処理の中に繰り返し処理を入れると制御が難しくなるので注意してください。

ここでは「for」文を使って解説しましたが、「while」文や「do」文でも同様に多重ループが可能です。

関連項目 ▶▶▶

- 「while」文について …… p.94
- 「do」文について …… p.96
- 「break」文について …… p.98
- 「continue」文について …… p.101
- 匿名クラスについて …… p.153
- ジェネリックスについて …… p.177
- 配列について …… p.462
- リストについて …… p.482

「while」文について

「while」文

「while」文も、「for」文と同様に、繰り返し処理を行う文です。「for」文は繰り返し処理に使う変数の初期化、繰り返し処理の継続条件、繰り返し処理を行うたびに実行される更新式をまとめて記述するのに対し、「while」文では繰り返し処理の継続条件のみを記述します。「while」文は次のように記述します。

```
while (条件式)
    (繰り返し処理);
```

「if」文と同様に、繰り返し処理が1つの文の場合はこのままで構いませんが、複数の文からなる場合はブロックを使って記述します。

条件式には、「while」文を続ける条件を記述します。条件判定は最初の繰り返し処理を行う前と、繰り返し処理を終えて更新式を実行した後に行います。判定結果が「true」の場合は繰り返し処理を実行し、「false」の場合は「while」文を終了します。

◉WhileSample.java

```
// while文のサンプルクラス
public class WhileSample {

    public static void main(String[] args) {

        int  cnt = 0;

        while (cnt < 5) {
            System.out.println("cnt = " + ++cnt);
        }
    }
}
```

WhileSample.javaでは、「for」文と異なり、ループカウンタ変数の初期化、更新は「while」文のカッコの外で行っています。WhileSample.javaを実行すると、次のように表示されます。

```
cnt = 1
cnt = 2
cnt = 3
cnt = 4
cnt = 5
```

条件式しか記述しないため、無限ループ処理（特定の条件を満たすまでずっと同じ処理を繰り返すこと）を行う際によく使われます。

◉InfiniteLoopSample.java

```
// while文を使った無限ループのサンプル
public class InfiniteLoopSample {

    public static void main(String[] args) {

        int  cnt = 0;

        while (true) {
            // ループカウンタを増やす
            cnt++;

            // サイコロを振る
            int  dice = (int) (Math.random() * 6.0 + 1);

            // 1が出たらループを抜ける
            if (1 == dice) {
                break;
            }
        }

        System.out.println(
            "1が出るまで" + cnt + "回サイコロを振りました");
    }
}
```

InfiniteLoopSample.javaを実行すると、次のように表示されます。なお、乱数を使っているため、表示される回数は、毎回、変わります。

```
1が出るまで6回サイコロを振りました
```

関連項目 ▶▶▶

- 「for」文について …… p.90
- 「do」文について …… p.96
- 「break」文について …… p.98
- 「continue」文について …… p.101
- 乱数を取得する …… p.565

「do」文について

「do」文

「do」文は、「while」文とほとんど同じ繰り返し処理を行います。「do」文は、次のように記述します。

```
do
    (繰り返し処理);
while (条件式);
```

「while」文と同様に、繰り返し処理が1つの文の場合はこのままで構いませんが、複数の文からなる場合はブロックを使って記述します。条件式の後ろに「;」(セミコロン)が必要になります。

「while」文と異なる点は、繰り返し処理継続の判定のタイミングです。「while」文は繰り返し処理の前に行うのに対し、「do」文は後で行います。そのため、最低でも1回は繰り返し処理を行います。

◉DoSample.java

```
// do文のサンプルクラス
public class DoSample {

    public static void main(String[] args) {

        int  val = 11;

        StringBuilder  buf = new StringBuilder();

        do {
            int  flg = val & 1;
            buf.insert(0, flg);
            val >>>= 1;
        } while (val != 0);

        System.out.println("2進数への変換結果:" + buf.toString());
    }
}
```

DoSample.javaで、は数値の2進数変換を行っています。DoSample.javaを実行すると、次のように表示されます。

```
2進数への変換結果:1011
```

ほとんどの場合、繰り返し処理の部分を「while」に変えても同じように動作します。

```
while (val != 0) {
    int  flg = val & 1;
    buf.insert(0, flg);
    val >>>= 1;
}
```

ただし、「val = 0」の場合、「do」文を使うと「2進数への変換結果:0」と表示されるのに対し、「while」文を使うと「2進数への変換結果:」と表示されます。これは、「while」文を使うと最初に繰り返し処理の継続判定を行ってしまい、繰り返し処理を一度も実行しないからです。

COLUMN　デバッグ情報の埋め込み

Javaをコンパイルするときのオプション指定で、クラスファイルへのデバッグ情報の埋め込みの制御ができます。デバッグ情報を埋め込むとスタックトレースを出力したときにソースファイル名や行番号が出力されますが、デバッグ情報を埋め込まないと代わりに「Unknown Source」と出力されます。

すべてのデバッグ情報を埋め込むときは、コンパイル時の「javac」コマンドのオプションに「-g」を指定します。埋め込まないときは、「-g:none」を指定します。「-g:」の後ろに「source」(ソースファイル)、「lines」(行番号)、「vars」(局所変数)のいずれかを指定すると、その情報だけ埋め込まれます。「,」(カンマ)で区切ることで複数指定可能です。デフォルトは「-g:source,lines」です。

なお、デバッグ情報を埋め込むと、クラスファイルのサイズが若干、大きくなります。

関連項目 ▶▶▶

- 「for」文について …… p.90
- 「while」文について …… p.94
- 「break」文について …… p.98
- 「continue」文について …… p.101

「break」文について

ループ内での「break」文

ブロック内で処理を中断してブロックから抜けたい場合は、「break」文を使います。「break」文は、繰り返し処理と「switch」文で使います。

◉BreakSample.java

```
// break文のサンプルクラス
public class BreakSample {

    public static void main(String[] args) {

        for (int i = 0; i < 3; i++) {
            System.out.println(i + "回目の処理開始");

            if (i > 0) {
                // iが0より大きくなったら繰り返し処理終了
                break;
            }

            System.out.println(i + "回目の処理終了");
        }
    }
}
```

BreakSample.javaを実行すると、次のように表示されます。

```
0回目の処理開始
0回目の処理終了
1回目の処理開始
```

多重ループ内での「break」文

多重ループの内側の繰り返し処理中に、「break」文を使うと、直近の繰り返し処理に対してのみ作用します。

◉MultipleForBreakSample1.java

```
// 二重ループ内でbreak文を使うサンプルクラス
public class MultipleForBreakSample1 {

    public static void main(String[] args) {

        for (int i = 1; i <= 9; i++) {
```