「見ひらきで学べる Java プログラミング」
正誤表

古井陽之助, 神屋郁子, 下川俊彦, 合志和晃

2023/05/10

本資料について

本資料は「見ひらきで学べる Java プログラミング」（近代科学社, 2019. 以下「本書」）の正誤表です。本書中の誤りにつき、読者に皆様にご迷惑をおかけしたことをお詫びいたしますとともに、本資料の通り訂正いたします。

改版履歴

- 2019/09/10 公開。
- 2019/10/25 2.4 節 (p.14) Ricty Diminished Discord の URL 変更を反映。
- 2020/05/26 初版第 2 刷での修正を反映。
- 2020/07/22 2.4 節 (p.14) Ricty Diminished Discord の URL 変更を反映。
- 2021/01/21 2.2 節 (p.11) 図 2.4 の誤りとその訂正について追記。
- 2021/06/08 5.2 節 (p.60) 図 5.1 の誤りとその訂正について追記。
- 2022/03/24 JDK 18 から既定の文字コードが変更されたことを反映。
- 2023/03/27 初版第 3 刷での修正を反映。
- 2023/05/10 4.3 節（p.48）表 4.2 の誤りとその訂正について追記。

全体について

コマンドプロンプトにおいて javac コマンドを実行する方法を、2.6 節 (p.19) では

javac によるコンパイル処理

```
javac ファイル名.java
```

としていますが、もしこのとき javac や java の実行結果に文字化けが見られたら、簡単な対処法として次の 1., 2. のいずれかうまくいくほうをおこなうようにしてください。なお、この現象および文字コードの詳細については本資料の付録 A を参照してください。

1. javac の文字コード指定を UTF-8 に変更する。

   javac によるコンパイル（文字コードとして UTF-8 を指定）

   ```
   javac -encoding utf8 ファイル名.java
   ```

2. javac の文字コード指定をシフト JIS に変更する。

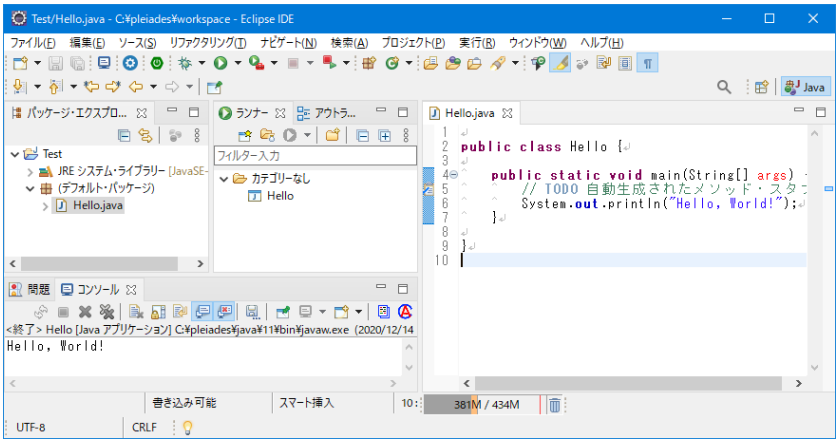
   javac によるコンパイル（文字コードとしてシフト JIS を指定）

   ```
   javac -encoding ms932 ファイル名.java
   ```

第 2 章 はじめてのプログラミング

（以下の項目は初版第 3 刷で修正済み）

2.2 節 (p.11) 図 2.4 図中左側のパッケージ・エクスプローラーに src というフォルダのアイコンが表示されていますが、この節の手順にしたがって操作すればこのアイコンは現れません。正しい図は次のとおりです。



2.4 節 (p.14) 側注 Ricty Diminished Discord の URL が変更されました。最新の URL には本書のサポートサイトからリンクしていますので、そちらでご確認ください。

（以下の項目は初版第２刷で修正済み）

2.3 節 (p.13) 上から 3 行目「今度はたくさん表示されていますが」の「今度は」を削除。

2.5 節 (p.16) 最後の行「(元と同じ行数に収めるため、段落は分けなくて OK)」は削除。

2.6 節 (p.19) コマンドプロンプトにおける javac の実行方法については、本資料の最初の項目「全体について」を参照してください。また、文字コードの詳しい説明については本資料の付録 A を参照してください。

第４章 条件分岐

（以下の項目は初版第３刷でも未修正）

4.3 節 (p.48) 表 4.2 において記号 ∨ と ∧ が入れ替わって記載されています。正しくは下表のように ∧ が「かつ」で、∨ が「または」です。

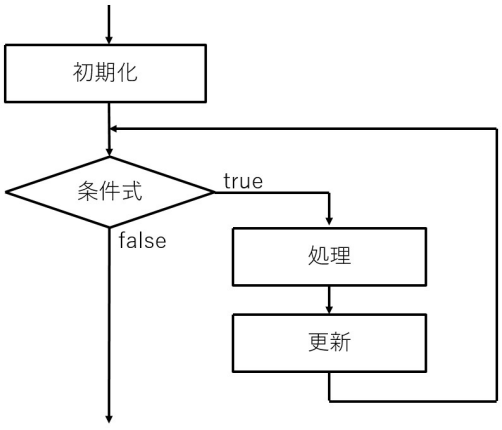
表 4.2 条件積演算子と条件和演算子

演算子	数学記号	意味	式の例	式の例の説明
&&	∧	かつ	条件１ && 条件２	条件１と条件２が両方成り立つ
||	∨	または	条件１ || 条件２	条件１と条件２の少なくとも一方が成り立つ

第５章 繰返し

（以下の項目は初版第３刷で修正済み）

5.2 節 (p.60) 図 5.1 「更新」から出た矢印が「初期化」の前に接続するのは誤りです。正しい接続先は「初期化」と「条件式」の間です。正しい図は次のとおりです。



第9章 インスタンスメソッドと修飾子

（以下の項目は初版第2刷で修正済み）

9.2 節 (p.126) 最後の行「リスト 8.4 節（8.3 節）」を「リスト 8.4（8.3 節）」に修正。

9.10 節 (p.142) 章末問題 問 2 のリスト 9.15 の 1 行目にあるクラス名 Pr09_2 を Pr09_02 に修正。

```
1 public class Pr09_02 {
```

第10章 コンストラクタと多重定義

（以下の項目は初版第2刷で修正済み）

10.8 節 (p.158) 章末問題 問 3 の中央付近「フィルード fuel」を「フィールド fuel」に修正。

付録 A. JDK とテキストエディタによる開発における文字コード指定

人間の書いた Java プログラムそのものは文字データです。そのデータ形式（**文字コード**）の種類について、本書 2.1〜2.4 節で紹介した Eclipse のような統合開発環境を使う限りはあまり注意を払う必要がありません。

しかし、本書 2.6 節のようにテキストエディタでプログラムを書くときには注意が必要です。もしプログラムのデータ形式と JDK の想定するデータ形式が一致していないと、次の A.1 のような現象が発生することがあります。

原因を確認する方法については A.2 を参照してください。解決方法については A.2〜A.4 を参照してください。また、その原因の詳細については A.5 を参照してください。

A.1 文字コードが一致しないときの現象

人間の書いた Java プログラムと JDK とで文字コードが一致していないときに起こるエラーの例を次の図 1, 2 に示します。これらの例では javac を実行したときにエラーが起きています。

```
C:¥Users¥maya¥Desktop¥java>javac Sample03_04_Input.java
Sample03_04_Input.java:6: エラー: この文字(0x85)は、エンコーディングwindows-31jにマップ
    System.out.print("謨-蛟、繧貞?･蜉帙@縺ｮ縺上□縺輔>:");

エラー1個
```

図 1: 文字コード不一致によるエラー例 1

```
C:¥Users¥taro¥Desktop¥java>javac Sample03_04_Input.java
Sample03_04_Input.java:7: エラー: この文字(0x90)は、エンコーディングUTF-8にマップできません
    System.out.print("???1??????????????:");

Sample03_04_Input.java:7: エラー: この文字(0x94)は、エンコーディングUTF-8にマップできません
    System.out.print("???1??????????????:");

Sample03_04_Input.java:7: エラー: この文字(0x92)は、エンコーディングUTF-8にマップできません
    System.out.print("???1??????????????:");

Sample03_04_Input.java:7: エラー: この文字(0x82)は、エンコーディングUTF-8にマップできません
    System.out.print("???1??????????????:");
```

図 2: 文字コード不一致によるエラー例 2

また、一見 javac が問題無く完了しても、そのあと java を実行したときに、次の図 3 のように奇妙な文字列が表示されることがあります。このような現象を**文字化け**といいます。

```
C:¥work>javac Hello.java

C:¥work>java Hello
縺薙ｓ縺ｫ縺｡縺ｯ
```

図 3: 文字コード不一致による文字化けの例

A.2 javac のオプションで文字コードを指定する方法

A.1 のような現象が起きた場合、javac によるコンパイル時に次のようにして文字コードを指定すれば解決するはずです。原因を確認するためには、あるいはとりあえず対処するだけなら、この

方法が簡単です。

javac によるコンパイル時の文字コード指定

```
javac -encoding 文字コード ファイル名.java
```

文字コードとしては UTF-8（あるいは utf8）や MS932（あるいは ms932, windows-31j）などが指定できます[1]。これらで A.1 のような現象が無くなるかどうかを試してみるとよいでしょう。たとえば、本書 p.19 の図 2.11 の場合には次のようにしてください。

```
C:\Users\taro> javac -encoding utf8 Hello.java
```

または

```
C:\Users\taro> javac -encoding ms932 Hello.java
```

A.3 Java プログラムの文字コードを変更して保存しなおす方法

別の解決方法です。テキストエディタで Java プログラムのソースファイル（ファイル名.java）を開き、文字コードを JDK に合うものに変更して保存しなおすと、以後は javac に -encoding 文字コード という指定を付ける必要が無くなります。

Windows 付属のメモ帳を用いる場合には、次の手順でソースファイルの文字コードを変更することができます。

1. メモ帳でソースファイルを開く。
2. メニューから [ファイル]-[名前を付けて保存] を選択する。
3. 文字コードを選択する。
   - UTF-8 で保存するなら「UTF-8」を選ぶ（「UTF-8 (BOM 付き)」ではない）。
   - シフト JIS で保存するなら「ANSI」を選ぶ。
4. 保存する。

いったん文字コードを指定してソースファイルに保存すると、次にそのソースファイルを開くときには、メモ帳は文字コードを自動的に判別します。

ただし、メモ帳で新たに書いた Java プログラムを保存するときの文字コードはあらかじめ決まっているので、それを変更するには新規保存のたびに文字コードを指定するしかありません。メモ帳以外のテキストエディタでは新規保存時の文字コードの設定もあらかじめ変更できることが多いので、必要に応じて検討するとよいでしょう。

A.4 JDK の環境変数で文字コードを設定する方法

また別の、Windows のコマンドプロンプトに慣れた人向けの解決方法です。コマンドプロンプトで JDK を使用する場合、その文字コードの設定を次のコマンドによって変更することができます。以後は javac に -encoding 文字コード という指定を付ける必要が無くなります。

[1] MS932 は日本語 Windows 環境で使用されるシフト JIS（ジス）という文字コードを指します。これとは別に Shift_JIS もありますが、Java において MS932 と Shift_JIS は同一ではありませんので注意してください。

JDK の文字コード設定

```
set JAVA_TOOL_OPTIONS=-Dfile.encoding=文字コード
```

JDK 17 あるいはそれ以前を使用するときに、あらかじめ文字コードを UTF-8 に設定しておくには次のようにしてください。

```
C:\Users\taro> set JAVA_TOOL_OPTIONS=-Dfile.encoding=utf8
```

JDK 18 あるいはそれ以降を使用するときに、あらかじめ文字コードをシフト JIS に設定しておくには次のようにしてください。

```
C:\Users\taro> set JAVA_TOOL_OPTIONS=-Dfile.encoding=ms932
```

なお、このコマンド自体を本書 p.17 のリスト 2.4 のバッチファイルに加えておくと便利でしょう。次のリスト 1 のように cmd よりも前の行に挿入してください。必要なら文字コード別に複数のバッチファイルを用意することもできるでしょう。

リスト 1: 文字コード設定を本書リスト 2.4 に追加

```
1 set PATH=JDKのパス;%PATH%
2 set JAVA_TOOL_OPTIONS=-Dfile.encoding=文字コード
3 cmd
```

A.5 解説

本書 2.6 節のように Windows 付属のテキストエディタであるメモ帳と JDK を組み合わせて使ったときに、A.1 のような現象が起きる原因について説明します。

文字コード

人間が書いた Java プログラムのような文字データは、それぞれの文字を決まった番号（符号）に変換することでデータ化されます。たとえば「A」は 65、「B」は 66 というような具合です。このような文字データの形式が**文字コード**です[2]。

日本語の文字を使用できる文字コードには **UTF-8**、**シフト JIS**、EUC など複数の種類があります。日本語 Windows 環境では伝統的にシフト JIS（厳密にいうとその一種である MS932）が使われてきましたが、他言語と共通に使用できる UTF-8 が広く普及するにしたがって、日本語 Windows 環境でも UTF-8 を使う場面が多くなりました。

メモ帳と JDK

日本語 Windows に付属するメモ帳は複数の文字コードを扱うことができます。ただし、使用者が特に指定しないときに使われる文字コード、すなわち**既定（デフォルト）**の文字コードは、かつてはシフト JIS でしたが、2019 年にリリースされた Windows 10 バージョン 1903 (May 2019 Update) からは UTF-8 に変更されました（図 4）。

[2]取り扱い対象の文字の集まりが**文字セット**、その符号が**文字コード**、文字を符号に変換するときの規則が**文字エンコーディング**です。この 3 つの用語は厳密にはそれぞれ別の概念を指しますが、互いに密接に関係するため、区別されず同じような意味で使われることがあります。本資料でも原則的に文字コードという用語のみを使用します。

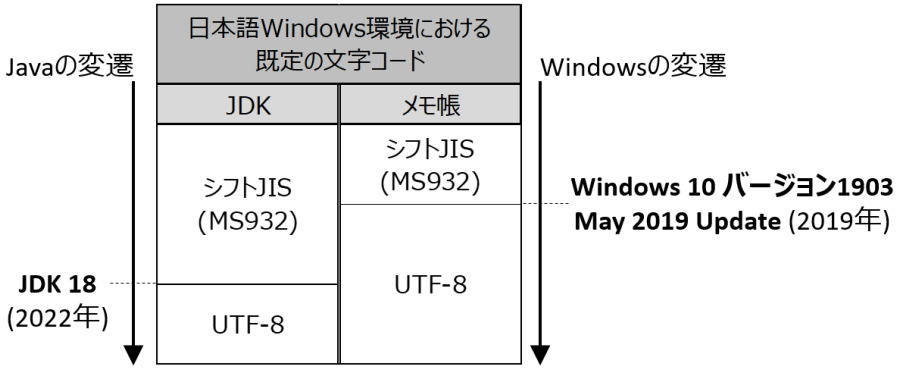


図 4: 既定の文字コードの変遷

一方、JDK が Java プログラムのソースファイルを読み取るときに使う既定の文字コードも、JDK 17 まではシフト JIS でしたが、2022 年にリリースされた JDK 18 からは UTF-8 に変更されました。

したがって、最新の Windows のメモ帳と最新の JDK を組み合わせて、文字コードを特に指定せずに使うならば、自然と文字コードは一致することになります。

しかし、メモ帳と JDK のどちらかが最新版でなかったり、過去に作った Java プログラムを最新の JDK でコンパイルしたりしたときには、文字コードが一致しているかどうかに注意を払う必要があります。もし文字コードが食い違っていると、JDK によるコンパイル処理や実行過程のどこかで A.1 のような現象が起こりえます。

A.6 補足: 統合開発環境の場合

本書 2.1〜2.4 節で紹介した Eclipse のような統合開発環境は、原則的にエディタとコンパイラの文字コードを合わせてくれます。したがって、統合開発環境を普通に使っているぶんには、文字コードの不一致によるエラーや文字化けは起こりません。

もし統合開発環境でプログラミングをおこなっていて問題がなければ、その文字コードの設定を変更したりソースファイルを保存しなおしたりする必要はありません。