

基本の修得から  
ゲームプログラム作成まで

**本格学習**  
[改訂3版]  
**Java**  
**入門**

**佐々木 整**  
著

技術評論社

# 例題 解答・解説

## 第1章 プログラミング言語Java

### 1-1 Javaとは

例題

1-1 (1)

Q

本文 ▶▶ P.014

解答

- ① オブジェクト指向
- ② once
- ③ anywhere
- ④ バグ

解説

- ① オブジェクト指向という手法の詳細については、第7章で解説します。
- ①～③ 「1度書けばどこでも動作させられる」という意味です。
- ④ バグの語源は、コンピュータ創成期、1匹の蛾（バグ）が配線の間に挟まったことで、マシンが誤動作する原因となったことに由来します。

### 1-2 プログラム動作の仕組み

例題

1-2 (1)

Q

本文 ▶▶ P.016

解答

- ① マシン語（機械語）
- ② Java仮想マシン（Java VM）

解説

- ① コンピュータは、コンパイラやインタプリタが機械語に翻訳した言語に沿って処理を行います。
- ② Javaの場合、コンパイラによって翻訳されたプログラム（これをクラスファイルといいます）を、インタプリタによってJava VMが順次実行します。

## 第2章 JShellによるJavaプログラミング体験

### 2-3 プログラムの基本

例題

2-3 (1)

Q

本文 ▶▶ P.026

解答

空文

解説

「;」（セミコロン）だけでも、文として成立します。逆に言うと、「;」がないと文として成立しないので注意しましょう。



## 2-3 (2)

本文 >>> P.028

解答

```

01 {
02   {
03     A
04     B
05     {
06       C
07       {
08         {
09           D
10         }
11         E
12         F
13       }
14       G
15     }
16   }
17 }

```

解説

ブロックが入れ子になっている場合、内側のブロックに段落をつけ、階層構造を見やすくすると、他の人にも理解しやすいプログラムにすることができます。

## 2-5 データの表示



### 2-5 (1)

本文 >>> P.036

解答

- ① `print`
- ② `'G'`
- ③ `7`

解説

- ① 実行結果をみると、最初の4単語は改行されていないので、ここで使われているのは`print()`メソッドだということがわかります。
- ② ここで表示しているのは「G」1文字です。文字を1文字だけ扱うときは「`'`」で囲みます。
- ③ 数字は、クォートで囲まなくても数値として表示することができます。クォートで囲むと文字として扱われます。

## 第3章 型と変数

### 3-1 データの表示



#### 3-1 (1)

本文 >>> P.038

解答

`float`

解説

浮動小数点型は`float`と`double`の2つですが、このリストには`float`しかありません。

**例題** 3-1 (2)  
**Q** 本文 >>> P.039

Q1

**解答** byte short int long

**解説** 整数型の範囲は、byteが $2^8$ 、shortが $2^{16}$ 、intが $2^{32}$ 、longが $2^{64}$ となります。

Q2

**解答** C

**解説** Javaでは、整数型の範囲は±の符号つきになります。正の値には0も考慮に入れるので、範囲に加わる値の数は、「-128 ~ -1」「0」「1 ~ 127」の、合計256 ( $2^8$ 、byte型の範囲) になります。

**例題** 3-1 (3)  
**Q** 本文 >>> P.041

**解答** ① double

② double

③ float

**解説** ①② とともに接尾辞を省略しているので、double型となります。

③ 接尾辞にFがついているので、float型です。

**例題** 3-1 (4)  
**Q** 本文 >>> P.042

**解答** ① println

② \n

**解説** ① 実行結果の画面で「Fine!」の前で改行されているので、println メソッドになります。

② 出力する文字列中で改行を指定する場合は、エスケープ・シーケンスの「\n」を使用します。

**例題** 3-1 (5)  
**Q** 本文 >>> P.043

**解答** ① boolean

② true

③ false

**解説** プログラム中でtrueやfalseを値に持つ場合は、boolean 型で型を定義しておく必要があります。

## 3-2 変数

**例題** 3-2 (1)  
**Q** 本文 >>> P.047

**解答** ① 値

② 変数

**解説** 変数と定数は利用の仕方は同じですが、定数には値を代入しなおすことができません。また、定数はすべて大文字にするのが慣例となっています。





### 3-2 (2)

本文 ▶▶▶ P.049

解答

**C (Aは予約語であり、Bは記号を含んでおり、Dは1文字目が文字でないから)**

解説

superは予約語です(使用法については8-3で解説します)。Bは記号が混じっており、Dは先頭が数字であるため、これが修正されれば変数名として使用できます。よって、残ったCが答えとなります。



### 3-2 (3)

本文 ▶▶▶ P.051

Q1

解答

**final float PI=3.14;**

解説

定数宣言と初期値設定は、このように同時に行うことができます。これは変数の場合も同様です。

Q2

解答

**int a=6, b=-3, c=2;**

解説

変数を一度に列挙して宣言している場合、このように式を書くことで初期値設定を同時に行うことができます。



### 3-2 (4)

本文 ▶▶▶ P.053

解答

**変数c**

解説

このプログラムをJShellで実行すると、次のエラーメッセージが表示されます。

```

エラー:
シンボルを見つけられません
  シンボル:   変数 c
    場所: クラス
              c = 6;
              ^

エラー:
シンボルを見つけられません
  シンボル:   変数 c
    場所: クラス
              c = 3;
              ^

jshell>

```

## 3-3 キャスト



### 3-3 (1)

本文 ▶▶▶ P.055

解答

**int、long**

解説

ワイドニング変換は、型範囲の小さい方から大きい方、整数型から浮動小数型へと型を変換することなので、基本型はこの2つが挙げられます。



### 3-3 (2)

[本文 >>> P.058](#)



① (short)    ② (int)



このプログラムの実行結果は次の通りです。

```

890
12345
890.1234

jshell>

```

## 3-4 配列



### 3-4 (1)

[本文 >>> P.060](#)



① `short sh[]`; または `short[] sh`;  
 ② `int[] a`;



配列の宣言形式には、[] (添え字演算子) を型に付ける書式と変数に付ける書式の2種類があります。



### 3-4 (2)

[本文 >>> P.061](#)



`int array[] = new int[56];`  
 または  
`int[] array = new int[56];`



添え字は必ず0から始まるので、配列の要素数は添え字の最大値+1以上でなければなりません。



### 3-4 (3)

[本文 >>> P.063](#)



① `new`  
 ② `0`  
 ③ `380`  
 ④ `2`



① 配列要素の作成には「new」を使います。  
 ②～④ `println` メソッドの表示順を添え字順に整理すると、`price[0]` が120、`price[1]` が380、`price[2]` が80となります。



### 3-4 (4)

[本文 >>> P.065](#)

#### Q1



① `double` または `float`  
 ② `{0.0, 1.0, 1.414}`



① 実行結果が実数なので、型の部分にはdoubleかfloatが入ります。  
 ② 3つの要素の初期値設定が1箇所にまとめられているため、「{ }」(中カッコ) を用いた表記をとります。

**解答** 6**解説** 配列messageの要素には、順番にHello. という6文字が格納されます。**例題** 3-4 (5)**Q** 本文 ▶▶ P.067**解答** ① new

② 4

③ a

**解説** 配列を代入するためには、すでにその配列の実体が生成されていなければなりません。**例題** 3-4 (6)**Q** 本文 ▶▶ P.069**解答**

```
01 {
02     int[][] mTables;
03
04     mTables = new int[9][9];
05
06     for (int i=0; i<9; i++) {
07         for (int j=0; j<9; j++) {
08             mTables[i][j] = (i+1) * (j+1);
09         }
10     }
11     System.out.print(mTables[8][0]);
12     System.out.print('\t');
13     System.out.print(mTables[8][1]);
14     System.out.print('\t');
15     System.out.print(mTables[8][2]);
16     System.out.print('\t');
17     System.out.print(mTables[8][3]);
18     System.out.print('\t');
19     System.out.print(mTables[8][4]);
20     System.out.print('\t');
21     System.out.print(mTables[8][5]);
22     System.out.print('\t');
23     System.out.print(mTables[8][6]);
24     System.out.print('\t');
25     System.out.print(mTables[8][7]);
26     System.out.print('\t');
27     System.out.println(mTables[8][8]);
28 }
```

**解説** 最初に、九九を示すよう、要素に値を代入しておきます。表示する際は「mTables[8][0]」のように、添え字が「実際の順番－1」であることに注意してください。

## 第4章 演算子

### 4-1 演算子の種類



4-1 (1)

本文 ▶▶▶ P.078

解答 ▶

- ① 論理演算子
- ② 代入演算子
- ③ 比較演算子

解説 ▶

論理演算子と比較演算子は、boolean型の結果を返します。代入演算子の「=」と比較演算子の「==」は、全く別物です。



4-1 (2)

本文 ▶▶▶ P.080

解答 ▶

() / + !=

解説 ▶

「=」や「+=」など代入演算子系は、優先順位のランクが最低です。逆に演算子と気づきにくい「()」は、最も高いランクになります。

### 4-2 代入演算子



4-2 (1)

本文 ▶▶▶ P.081

解答 ▶

代入演算子は右辺の値を左辺に代入するが、左辺が変数でないため

解説 ▶

左辺は定数なので、変数piが保持している値3.0を代入することができません。



4-2 (2)

本文 ▶▶▶ P.084

解答 ▶

```
01 {  
02     int ans;  
03     double x, y;  
04  
05     x = 1.8;  
06     y = 0.9;  
07  
08     ans = (int)(x - y);  
09     System.out.println(ans);  
10 }
```

解説 ▶

修正前のプログラムでは、ansに入る値は1になってしまいます。そこで演算子「()」によって、double型の演算を先に行い、その結果をint型にキャストします。

## 4-3 算術演算子

例題  
Q

### 4-3 (1)

本文 >>> P.086

解答

```
01 {  
02     int a, b, c, d, e;  
03  
04     a = -1 + -1;  
05     b = -1 - -1;  
06     c = -1 * -1;  
07     d = -1 / -1;  
08     e = -1 % -1;  
09  
10     System.out.println(a);  
11     System.out.println(b);  
12     System.out.println(c);  
13     System.out.println(d);  
14     System.out.println(e);  
15 }
```

解説

「-」(負符号)は、どの四則演算の演算子よりも優先順位が高いため、Calculation.javaの「1」を「-1」に置き換えるだけで十分です。演算子が並んで見にくくなる場合は、次のように明示的に演算子()を使用します。

```
a = -1 + -1;  
b = -1 - -1;  
c = -1 * -1;  
d = -1 / -1;  
e = -1 % -1;
```

例題  
Q

### 4-3 (2)

本文 >>> P.089

解答

- ① G
- ② a
- ③ A

解説

変数 upperCase から文字 'A' を引くことで、'A' から数えて何番目の文字なのかを求めることができます。それを文字 'a' に加えることで小文字に変換しています。

## 4-4 比較演算子

例題  
Q

### 4-4 (1)

本文 >>> P.092

解答

- ① C
- ② A
- ③ B

解説

- ① 「-1.0」と「-1」は同じ値なので、falseを返すようにするには、ノットイコールで結びます。ABを挿入するとtrueを返します。
- ② 小なりイコール、大なりイコール、ノットイコールの「=」は、いずれも右側につきます。
- ③ 左辺が定数なので代入演算子は使えません。

## 4-5 論理演算子



4-5 (1)

本文 ▶▶▶ P.095

解答 ▶

- ① false
- ② false
- ③ true

解説 ▶

- ① 「!」は偶数回かけると元の値に戻ります ( $\text{false} \rightarrow \text{true} \rightarrow \text{false}$ )。ここでは10回かけているので、値はfalseとなります。
- ② 最初のaはfalseのままですが、「&&」では、両方trueでない場合はfalseが返ります。
- ③ 右辺のaに「!」をかけるとtrueになります。「||」では、いずれかがtrueの場合はtrueが返ります。

## 4-6 内部表現に関わる演算子



4-6 (1)

本文 ▶▶▶ P.097

Q1

解答 ▶

- ① 10
- ② 14

解説 ▶

- ① 公式どおり、「 $2^3 \times 1 + 2^1 \times 1$ 」の結果を求めます。
- ② 公式どおり、「 $2^3 \times 1 + 2^2 \times 1 + 2^1 \times 1$ 」の結果を求めます。

Q2

解答 ▶

- ① 0101
- ② 1100

解説 ▶

- ① 2で割ると、余りは「101」の順に出てきます。4ビットで表現しなくてはならないため、頭に0を付けて表記します。
- ② 2で割ると、余りは「0011」の順に出てきます。



4-6 (2)

本文 ▶▶▶ P.098

解答 ▶

- ① 11110110
- ② 11110010

解説 ▶

- ① 10を8ビットの2進数で表現すると「00001010」になります。これをビット反転して「11110101」にし、1を追加すると「11110110」になります。
- ② 14を8ビットの2進数で表現すると「00001110」になります。これをビット反転して「11110001」にし、1を追加すると「11110010」になります。



#### 4-6 (3)

本文 ▶▶▶ P.101

解答

- ① `10 >> 1`
- ② `3 << 3`

解説

- ① 10と5を2進数で表すと、それぞれ「1010」「0101」になります。これは、「2で割る」ということは「1桁右シフトした」ということと同義だからです。
- ② 「8をかける」ということは「2の3乗をかける」ということです。そのため、2進表示の場合、桁は3桁繰り上がります。



#### 4-6 (4)

本文 ▶▶▶ P.103

解答

- ① `&`
- ② `^`

解説

- ① 比較する桁が同じ値の場合に1を返すのは「&」です。
- ② 比較する桁が違う値の場合に1を返すのは「^」です。

### 4-7 二項演算子以外の演算子



#### 4-7 (1)

本文 ▶▶▶ P.106

解答

- ① `x /= y`
- ② `x++`      または    `++x` ( または `x += 1` )
- ③ `x *= -y`   または   `x *= (-y)`

解説

四則演算を単項演算子にまとめる際は、代入演算子「=」を右側に記します。また、加減する値が1であれば、インクリメント演算子とデクリメント演算子を使用することができます。



#### 4-7 (2)

本文 ▶▶▶ P.108

解答

`price = age >= 20 ? 10000 : 2000;`  
 または  
`age >= 20 ? price = 10000 : price = 2000;`

解説

最初の解の場合、10000か2000を、条件によっていずれかが選択された後で変数priceに代入しています。次の解の場合、条件分岐後、変数priceに値を代入する処理を直接設定しています。

## 第5章 条件判断

### 5-1 単純な条件分岐 (if)



#### 5-1 (1)

本文 ▶▶▶ P.113

Q1

解答

「How are you?」と表示される

解説

ifによる処理が{ }で括られていないため、ifによる条件分岐は直後の「System.out.println("Hello.");」にしか適用されません。インデント(字下げ)に惑わされないように！

Q2

**解答** if分の条件式にある「c = 'A'」の代入演算子「=」が誤り

**解説** ifの条件式は結果がboolean 型のものでなければなりません。変数cの値と文字'A' が等しいかどうかを判断するのであれば、比較演算子「==」を使わなければなりません。

**例題**  
Q

5-1 (2)

本文 ▶▶▶ P.117

Q1

**解答** 「Hello.」と表示される

**解説** {} によるブロックが形成されていなくても、「System.out.println("How are you?");」はelseの制御を受けます。

Q2

**解答** true

**解説** 条件式に直接boolean型の値trueを記述すれば、どんなときでもそのif文の処理が実行されることになります。反対にfalseを記述すれば処理は実行されません。

**例題**  
Q

5-1 (3)

本文 ▶▶▶ P.119

Q1

- 解答**
- ① false
  - ② false
  - ③ false
  - ④ true
  - ⑤ true

**解説** 問題の論理演算の結果は、具体的には次のようになります。

A	B	C	結果
false	—	false	false
—	false	false	false
true	false	false	false
—	—	true	true
true	true	—	true

Q2

**解答** (A || B) && !C

**解説** 関係演算子「||」は「&&」よりも優先順位が低いので、「||」を「()」で括って「&&」よりも先に演算を行う必要があります。

Q3

**解答** !((fat <= -10) || (fat >= 20)) など

**解説** 「A && B (A であり B である)」は、「A ではないか B ではない」を全て否定したものと同じ事を意味しています。この問題の場合、(fat > -10) ではないは (fat <= -10) であり、同様に (fat >= 20) と「または (||)」で結合して、それを否定(!) することで同じ意味になります。



## 5-2 複数の条件分岐 (switch)



### 5-1 (1)

本文 ▶▶ P.125

解答 ▶

```
01 switch (ans) {  
02     case 1: a = 1;  
03         break;  
04     case 2: a = 2;  
05         break;  
06     default: a = 3;  
07 }
```

解説 ▶ if文の条件は、ans の値が1の場合、1より大きく3より小さい場合(つまり2)、それ以外の場合で分けられます。したがって、switch文にする際はcase1と2とdefaultで対処します。

## 第6章 繰り返し

### 6-2 条件指定の繰り返し



### 6-2 (1)

本文 ▶▶ P.134

Q1

解答 ▶

- ① 5
- ② >

解説 ▶

①はfor文の初期化式、②は条件式に対応します。

Q2

解答 ▶

- ① 6
- ② 2

解説 ▶

while文の中でも論理演算子を使用することができます。また、インクリメント演算子の使い方に注意しましょう。「--i」となっているので、減算が行われてから変数iの値が出力されることになります。



### 6-2 (2)

本文 ▶▶ P.135

解答 ▶

```
01 {  
02     int i = 5;  
03  
04     do {  
05         System.out.print("No.");  
06         System.out.println(--i);  
07     } while(i>0);  
08 }
```

解説 ▶

do while は後判定なので単純に書き直したただと結果が異なってしまう場合もあるので注意しましょう。

## 6-3 繰り返しの制御

例題



6-3 (1)

本文 ▶▶ P.137

解答

```
01 {
02     for (int i=9; i>0; i--) {
03         for (int j=9; j>0; j--) {
04             System.out.print(i * j);
05             System.out.print(" ");
06         }
07         System.out.println();
08     }
09 }
```

解説 ▶ ループ変数 i, j の初期値に注意しましょう。

例題



6-3 (2)

本文 ▶▶ P.140

解答

```
01 {
02     for (int i=1; i<10; i++) {
03         for (int j=1; j<10; j++) {
04             if (i < j) {
05                 break;
06             }
07             System.out.print(i * j);
08             System.out.print(" ");
09         }
10         System.out.println();
11     }
12 }
```

解説 ▶ break 文を使わなければ、次のように記述することもできます。

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

## 第 7 章 メソッド

### 7-1 Javaプログラムの入力と実行

例題



7-1 (1)

本文 ▶▶ P.149

Q1

解答

```
01 class MyName {
02     public static void main(String[] args) {
03         System.out.println("My Name is Hitoshi.");
04     }
05 }
```

#### ▼ 実行結果

```
C:\$SRC>java MyName
My Name is Hitoshi.

C:\$SRC>
```

**解説** 動作結果だけでなく、プログラムのコンパイルや実行方法についても確認しておきましょう

#### Q2

#### **解答** メッセージ「Hello」が「」でくくられていないから

**解説** この場合、Helloは第3章で解説した変数として認識されてしまうため、次のエラーメッセージが表示されます。必ずダブルクォートでくくりましょう。「」（シングルクォート）を2回タイプしても「」（ダブルクォート）とはみなされないので、注意してください。

#### ▼ 実行結果

```
C:\$SRC>javac Error5.java
Error5.java:3: シンボルを解決できません。
シンボル: 変数 Hello
易所    : Error5 の クラス
          System.out.println(Hello);
                              ^
エラー 1 個

C:\$SRC>
```

## 7-2 メソッドの基本

### 例題



#### 7-2 (1)

本文 ▶▶ P.153

#### **解答**

```
01 class MessageTest1 {
02
03     static void message() {
04         System.out.println("Hello.");
05     }
06
07     static void message2() {
08         System.out.println("Good bye.");
09     }
10
11     public static void main(String[] args) {
12         message();
13         message2();
14     }
15 }
```

**解説** message() メソッドを参考にして、System.out.println()の中身を書き換えます。このとき、メソッド名をmessage2()にすることを忘れないようにしましょう。

### 例題



#### 7-2 (2)

本文 ▶▶ P.153

#### **解答**

```
01 class MessageTest2 {
02
03     static void message() {
04         System.out.println("Hello.");
05         message2();
06     }
07 }
```

```

07
08     static void message2() {
09         System.out.println("Good bye.");
10     }
11
12     public static void main(String[] args) {
13         message();
14     }
15 }

```

**解説** 題文にあるように、main() メソッドから message2() メソッドは呼び出されていませんから、message() メソッドで message2() メソッドを呼び出す必要があります。このようにメソッドからメソッドを呼び出すことができますが、次の様に message2() メソッドから message() メソッドを呼び出すと、お互いに呼び出し合い続けるのでプログラムは終了しなくなってしまいます(その場合は、強制終了を行って終了させてください)。

※ランタイムエラーが発生して異常終了する場合があります。

## 7-3 引数と戻り値

### 例題 7-3 (1)

**Q** 本文 ▶▶ P.155

**解答**

```

01 class MessageTest3 {
02
03     static void message(String message) {
04         System.out.println(message);
05     }
06
07     public static void main(String[] args) {
08         message("Good morning.");
09         message("Good afternoon.");
10         message("Good evening.");
11     }
12 }

```

**解説** 文字列を引数として受け取るときには String を使います。これは main() メソッドでも記述されているもので、main() メソッドでは String の配列として引数を受け取っているのです。

### 例題 7-3 (2)

**Q** 本文 ▶▶ P.155

**解答**

```

01 class SquareTest1 {
02
03     static void square(int n) {
04         System.out.println(n + " の二乗は" + n * n + " です。");
05     }
06
07     public static void main(String[] args) {
08         for (int i = 0; i < 10; i++) {
09             square(i);
10         }
11     }
12 }

```

**解説** main() メソッドから渡された変数 i を引数として square() メソッドで受け取ります。このとき、変数 n として受け取るので、メソッド内では i ではなく、n として扱われることになります。



### 7-3 (3)

本文 ▶▶ P.155

解答

```
01 class EvenOddTest1 {
02
03     static void check(int n) {
04         System.out.print(n + " は");
05         if (n % 2 == 0) {
06             System.out.println(" 偶数です。");
07         } else {
08             System.out.println(" 奇数です。");
09         }
10     }
11
12     public static void main(String[] args) {
13         for (int i = 0; i < 9; i++) {
14             check(i);
15         }
16     }
17 }
```

解説

奇数か偶数かの判断は2で割ったときの余りが0かどうかで判定しています。メソッド内でもこれまで通りif文やfor文などの命令を利用できます。



### 7-3 (4)

本文 ▶▶ P.157

解答

```
01 class PowerTest1 {
02     static void power(double m, int n) {
03         double ans = 1.0;
04
05         for(int i=0; i < n; i++) {
06             ans *= m;
07         }
08         System.out.println(m + " の" + n + " 乗は" + ans + " です。");
09     }
10
11     public static void main(String args[]) {
12         power(2.3,8);
13         power(16.9,4);
14     }
15 }
```

解説

1つめの引数がdouble型なので、メソッドpower()の定義もそれにあわせる必要があります。



### 7-3 (5)

本文 ▶▶ P.158

解答

```
01 class SquareTest2 {
02     static long square(int n) {
03         return n * n;
04     }
05
06     public static void main(String[] args) {
07         for (int i = 0; i < 10; i++) {
08             System.out.println(i + " の二乗は" + square(i) + " です。");
09         }
10     }
11 }
```

**解説** square() メソッドでは int 型同士の計算を行っていますが、かけ算なので計算結果は int 型の範囲を超える可能性があります。そのため、戻り値の型は long 型として指定しています。

**例題** 7-3 (6)

**Q** 本文 ▶▶ P.159

**解答**

```
01 class EvenOddTest2 {
02     static void check(int n) {
03         System.out.print(n + " は");
04         if (n % 2 == 0) {
05             System.out.println(" 偶数です.");
06             return;
07         }
08         System.out.println(" 奇数です.");
09     }
10
11     public static void main(String[] args) {
12         for (int i = 0; i < 9; i++) {
13             check(i);
14         }
15     }
16 }
```

**解説** if文の条件判断がtrueの時には「偶数です。」というメッセージを表示してreturnしますから、check()メソッド内のそれ以降の処理は実行されません。そのため、elseを使うことなく、「奇数です。」の表示を行うことができます。

## 7-4 オーバーロード

**例題** 7-4 (1)

**Q** 本文 ▶▶ P.159

**解答**

```
01 class SquareTest3 {
02     static long square(int n) {
03         return n * n;
04     }
05
06     static double square(double n) {
07         return n * n;
08     }
09
10     public static void main(String[] args) {
11         for (int i = 0; i < 10; i++) {
12             System.out.println(i + " の二乗は" + square(i) + " です.");
13             System.out.println(i + 0.5 + "の二乗は" + square(i+0.5) + "です.");
14         }
15     }
16 }
```

## 7-5 メソッド呼び出し



### 7-5 (1)

本文 ▶▶ P.165

解答

```
01 static int max(int a, int b, int c, int d) {  
02     return max(max(a, b, c), d);  
03 }
```

解説

変数 a, b, c の順序や 2 つの値から最大値を求める順序は計算結果と関係ないので、max(a, max(b, c, d)) のような記述もできます。



### 7-5 (2)

本文 ▶▶ P.165

解答

```
01 class CubeTest {  
02     static long square(int n) {  
03         return n * n;  
04     }  
05  
06     static long cube(int n) {  
07         return n * square(n);  
08     }  
09  
10     public static void main(String[] args) {  
11         for (int i = 0; i < 10; i++) {  
12             System.out.println(i + " の三乗は" + cube(i) + " です。");  
13         }  
14     }  
15 }
```

解説

square() メソッドの戻り値が long 型ですから、cube() メソッドの計算結果も long 型になるので、戻り値も long 型で宣言します。



### 7-5 (3)

本文 ▶▶ P.167

解答

```
01 class GCDTest {  
02     static int gcd(int a, int b) {  
03         if (b == 0) {  
04             return a;  
05         }  
06         if (a % b == 0) {  
07             return b;  
08         }  
09         return gcd(b, a % b);  
10     }  
11  
12     public static void main(String[] args) {  
13         System.out.println("1071と1029の最大公約数は" + gcd(1071, 1029) + "です。");  
14     }  
15 }
```

解説

gcd() メソッド内での再帰呼び出しでは、2 つの引数を扱っていることに注意してください。特に、2 つ目のパラメータが再帰呼び出しでは 1 つ目のパラメータとして用いられています。

## 第8章 クラス

### 8-2 クラスの作成



#### 8-2 (1)

本文 ▶▶ P.176

解答

▼ Fraction.java の改良

```
01 class Fraction {
02     int numerator;
03     int denominator;
04
05     void add(Fraction f) {
06         numerator = numerator * f.denominator + denominator * f.numerator;
07         denominator = denominator * f.denominator;
08     }
09
10     void add(int n) {
11         numerator = numerator + denominator * n;
12     }
13
14     void sub(Fraction f) {
15         numerator = numerator * f.denominator - denominator * f.numerator;
16         denominator = denominator * f.denominator;
17     }
18
19 }
```

▼ FractionTestQ1.java

```
01 public class FractionTestQ1{
02     public static void main(String arg[]) {
03         Fraction f1 = new Fraction();
04         Fraction f2 = new Fraction();
05
06         f1.numerator = 1;
07         f1.denominator = 2;
08         f2.numerator = 3;
09         f2.denominator = 4;
10
11         System.out.println("f1=" + f1.numerator + "/" + f1.denominator);
12         System.out.println("f2=" + f2.numerator + "/" + f2.denominator);
13
14         f1.sub(f2);
15         System.out.println("f1=" + f1.numerator + "/" + f1.denominator);
16     }
17 }
```

解説

引き算を行うメソッドをsub()と名付けています。add()メソッドの働きを理解していれば、sub()の作成は難しくはなかったはずです。なお、計算結果によっては、約分が必要なものがあります。約分は第7章の例題で作成したgcd()メソッドを使って行ってみましょう。





## 8-2 (2)

本文 >>> P.176

解答

### ▼ Fraction.java の改良

```
01 class Fraction {
02     int numerator;
03     int denominator;
04
05     void add(Fraction f) {
06         numerator = numerator * f.denominator + denominator * f.numerator;
07         denominator = denominator * f.denominator;
08     }
09
10     void add(int n) {
11         numerator = numerator + denominator * n;
12     }
13
14     void sub(Fraction f) {
15         numerator = numerator * f.denominator - denominator * f.numerator;
16         denominator = denominator * f.denominator;
17     }
18
19     void sub(int n) {
20         numerator = numerator - denominator * n;
21     }
22 }
```

### ▼ FractionTestQ2.java

```
01 public class FractionTestQ2 {
02     public static void main(String arg[]) {
03         Fraction f1 = new Fraction();
04         Fraction f2 = new Fraction();
05
06         f1.numerator = 1;
07         f1.denominator = 2;
08
09         System.out.println("f1=" + f1.numerator + "/" + f1.denominator);
10
11         f1.sub(10);
12         System.out.println("f1=" + f1.numerator + "/" + f1.denominator);
13     }
14 }
```

解説

これも add() メソッドの演算子を修正するだけで実現できますね。



## 8-2 (5)

本文 >>> P.180

解答

### ▼ Fraction.java

```
01 class Fraction {
02     int numerator;    // 分子
03     int denominator; // 分母
04
05     Fraction() {
06         numerator = 0;
07         denominator = 1;
08     }
09 }
```

```

08 }
09
10 Fraction(int n, int d) {
11     numerator = n;
12     denominator = d;
13 }
14
15 Fraction(int n) {
16     numerator = n;
17     denominator = 1;
18 }
19
20 void add(Fraction f) {
21     numerator = numerator * f.denominator + denominator * f.numerator;
22     denominator = denominator * f.denominator;
23 }
24
25 void add(int n) {
26     numerator = numerator + denominator * n;
27 }
28
29 public String toString() {
30     return numerator + "/" + denominator;
31 }
32 }

```

#### ▼ FractionTestQ3.java

```

01 public class FractionTestQ3 {
02     public static void main(String[] args) {
03         Fraction f1 = new Fraction(2);
04
05         System.out.println("f1 = " + f1.numerator + "/" + f1.denominator);
06     }
07 }

```

**解説** コンストラクタ `Fraction(int n)` は、次のように記述することもできます。thisを使うことで、同じコンストラクタ名で引数の違う（この場合はint型を2つ）コンストラクタを呼び出しているのです。

```

Fraction(int n) {
    this(n, 1);
}

```



## 8-2 (6)

本文 ▶▶ P.182



```

01 public String toString() {
02     if (denominator == 1) return numerator + "";
03     return numerator + "/" + denominator;
04 }

```

**解説** 空の文字列""を加算することで、整数をStringに変換していますが、IntegerクラスのtoString()メソッドを使って、

```

if (denominator == 1) return Integer.toString(numerator);

```

と記述することもできます。

## 8-3 クラスの継承



### 8-3 (1)

本文 ▶▶ P.185

解答

```
01 class Fraction3 extends Fraction2 {
02     int gcd(int a, int b) {
03         int tmp;
04
05         if (a < b) {
06             tmp = a;
07             a = b;
08             b = tmp;
09         }
10         if (b == 0) return a;
11         if (a % b == 0) return b;
12         return gcd(b, a % b);
13     }
14
15     public String toString() {
16         return numerator / gcd(numerator, denominator) + "/" +
17             denominator / gcd(numerator, denominator);
18     }
19 }
```

解説

メソッドgcd()は計算途中で使っても構いませんが、toString()メソッドの中で利用して分数を出力させる直前で使うと効果的です。



### 8-3 (2)

本文 ▶▶ P.188

解答

```
01 Fraction2() {
02     super();
03 }
04
05 Fraction2(int n, int d) {
06     super(n, d);
07 }
```

解説

この例では、super()しか呼び出していませんが、他に処理を行う必要がある場合は、必ずsuper()の後に記述する必要があることに気をつけてください。



### 8-3 (3)

本文 ▶▶ P.189

解答

```
01 void add(Fraction f) {
02     numerator = numerator * f.denominator + denominator * f.numerator;
03     denominator = denominator * f.denominator;
04     int g = gcd(numerator, denominator);
05     numerator /= g;
06     denominator /= g;
07 }
```

解説

この問題では足し算の計算後に約分を行っています。通分をして足し算ができるように、最小公倍数を求めるメソッドを作って、通分もあわせて行えるように改良してみましょう！

## 第9章 例外処理

### 9-1 例外とは

#### 例題 9-1 (1) Q 本文 >>> P.205

Q1

解答 Error、Exception

解説 Errorはメモリ不足など、主にシステムの内部が原因で生じる例外です。一方Exceptionは、プログラム実行中に生じる人為的なミスが原因で生じる例外です。

Q2

解答 Throwableクラス

解説 例外オブジェクトはすべて、このThrowable から派生するError やException、およびそのサブクラスのインスタンスとなります。

### 9-2 例外処理の記述

#### 例題 9-2 (1) Q 本文 >>> P.211

Q1

- 解答
- ① throw
  - ② FileNotFoundException()
  - ③ FileNotFoundException

解説 例外発生を知らせるthrow と、例外処理を親メソッドに任せるthrowsを混同しないようにしましょう。

Q2

解答

```
01 class AnswerException extends Exception {
02     public AnswerException() { }
03     public AnswerException(String s) { }
04 }
```

解説 Stringクラスのデータをパラメータとするコンストラクタの作成も必要です。  
Exception クラスもStringデータを扱えるので、次のようにExceptionクラスのコンストラクタを利用することもできます。

```
class AnswerException extends Exception {
    public AnswerException() { }
    public AnswerException(String s) {
        super(s);
    }
}
```

Q3

解答

```
01 double sampleMethod(int a, int b) throws ArithmeticException, IOException {
02     処理A;
03 }
```

解説 例外が親メソッドに引き渡されるので、処理Bと処理Cに該当する処理を親メソッドで行う必要があります。

## 第10章 データの入出力

### 10-1 キーボードからの入力



#### 10-1 (1)

本文 ▶▶ P.214

解答

▼ HelloNameQ1.java

```
01 class HelloNameQ1 {
02     public static void main(String[] args) {
03         if (args.length > 0) {
04             System.out.println("Hello, " + args[0] + '!');
05         } else {
06             System.out.println("Hello!");
07         }
08     }
09 }
```

解説

コマンドライン引数の数は配列argsのlengthで取得することができるので、args.lengthが0より大きいかどうか(1つ以上引数が存在したか)の判断を行っています。このlength は配列の大きさを示すもので、Stringクラスのlength()メソッドとは異なっていることに注意しましょう。



#### 10-1 (2)

本文 ▶▶ P.215

解答

▼ BMIQ1.java

```
01 class BMIQ1 {
02     public static void main(String[] args) {
03         double height, weight, weightAve, fat;
04
05         height = Double.parseDouble(args[0]);
06         weight = Double.parseDouble(args[1]);
07
08         weightAve = 22 * Math.pow(height, 2);
09         fat = (weight - weightAve) / weightAve * 100;
10
11         System.out.print(" あなたの肥満率は ");
12         System.out.printf("%2.1f", fat);
13         System.out.println("% です。");
14
15         if (fat >= 20) {
16             System.out.println(" あなたは太りすぎです。");
17         }
18     }
19 }
```

解説

プログラムを実行する時には、java BMIQ1 1.75 85 のように、コマンドライン引数として身長と体重を入力することを忘れないようにしましょう。



#### 10-1 (3)

本文 ▶▶ P.216

解答

▼ BMIQ2.java

```
01 import java.util.Scanner;
02
```

```

03 class BMIQ2 {
04     public static void main(String[] args) {
05         double height, weight, weightAve, fat;
06
07         Scanner sin = new Scanner(System.in);
08
09         height = sin.nextDouble();
10         weight = sin.nextDouble();
11
12         weightAve = 22 * Math.pow(height, 2);
13         fat = (weight - weightAve) / weightAve * 100;
14
15         System.out.print(" あなたの肥満率は ");
16         System.out.printf("%2.1f", fat);
17         System.out.println("% です。");
18
19         if (fat >= 20) {
20             System.out.println(" あなたは太りすぎです。");
21         }
22     }
23 }

```

**解説** 身長と体重のデータは別々に入力しますから、それぞれのデータを入力した後にリターンキーを押すのを忘れないようにしましょう。

## 10-2 ファイルからのスキャン

### 例題 10-2 (1) 本文 ▶▶ P.218

#### 解答 ▼ PrintList.java

```

01 import java.util.Scanner;
02 import java.io.FileReader;
03
04 class PrintListQ1 {
05     public static void main(String[] args) {
06         FileReader fr = null;
07
08         try {
09             fr = new FileReader(args[0]);
10         } catch (Exception e) {
11             System.out.println("ファイルが見つかりません。");
12             System.exit(0);
13         }
14
15         Scanner sin = new Scanner(fr);
16         int i = 1;
17         while (sin.hasNext()) {
18             String s = sin.nextLine();
19             System.out.println(i + ": " + s);
20             i++;
21         }
22     }
23 }

```

**解説** 行番号をそろえるには、3-1-7で学習したprintf()メソッドで桁数を指定して表示させると良いでしょう。

## 第11章 マルチスレッド

### 11-1 マルチスレッドの体験



#### 11-1 (1)

本文 ▶▶ P.226

解答

- ① `time`
- ② `start()`
- ③ `time`
- ④ `InterruptedException`

解説

MultiplicationTest.javaの3行目などから、コンストラクタの引数がMultiplication(5, 500)のように2つあることが判ります。Multiplicationクラスのコンストラクタでは、1つ目で指定された値dをdan(九九の段)に使用して、2つ目で指定された値tをsleepする時間に使用しています。MultiplicationTest.javaの7行目以降では、それぞれのMultiplicationクラスのインスタンスでstart()メソッドを実行していますので、Multiplicationクラスはstart()メソッドを持っていないけません。



#### 11-1 (2)

本文 ▶▶ P.228

解答

- ① `Thread`
- ② `t`
- ③ `public`
- ④ `run()`

解説

run()メソッドの前に修飾子publicを付けるのを忘れないでください。スーパークラスであるThreadクラスではrun()はpublicになっていますので、publicよりも強いアクセス制限を指定することができません。つまり、publicを必ず付けることになります。

### 11-2 マルチスレッドプログラムの作成



#### 11-2 (1)

本文 ▶▶ P.232

解答

- ① `Thread` クラスの継承
- ② `Runnable` インタフェースの実装

解説

Threadクラスを継承する場合、多重継承できないというデメリットはありますが、スレッドを作成する手続きが簡単です。逆にあるクラスのサブクラスとして作成する場合などは、Runnableインタフェースの実装が適切です。



#### 11-2 (2)

本文 ▶▶ P.234

解答

- ① `synchronized`
- ② `wait()` メソッド
- ③ `notifyAll()` メソッド

解説

マルチスレッドを利用する場合、synchronizedによる同期処理は必須となります。wait()メソッドで待機させたスレッドの動作を開始させるのはnotify()メソッドですが、待機中の全メソッドを開始するには、notify()メソッドを使用します。

## 第12章 ネットワークプログラミング

### 12-1 通信の仕組み



12-1 (1)

本文 ▶▶ P.237

解答 ▶

- ① 48
- ② MAC
- ③ IP
- ④ DNS
- ⑤ ポート
- ⑥ 1

解説 ▶

MACアドレスは、合計で248(=281兆4749億7671万656)通りあるので、足りなくなる心配はないといっていでしょう。ただしIPアドレスは、情報家電の台頭により限界に近づく恐れがあるので、IPv6という規格が制定されました。



12-1 (2)

本文 ▶▶ P.238

解答 ▶

- ① クライアント・サーバモデル
- ② クライアント
- ③ サーバ
- ④ localhost

解説 ▶

サーバは要求を受け取り、クライアントは要求を送る側です。

### 12-2 クライアントの作成



12-2 (1)

本文 ▶▶ P.239

解答 ▶

- ① `Socket soc1 = new Socket("java.gihyo.co.jp", 1000);`
- ② `Socket soc2 = new Socket("123.123.123.123", 1234);`

解説 ▶

サーバへのソケットは、第1引数にサーバのIPアドレスかホスト名を、第2引数にポート番号を渡して生成します。



12-2 (2)

本文 ▶▶ P.241

解答 ▶

**UnknownHostException、  
IOException**

解説 ▶

UnknownHostExceptionは、サーバのIP アドレスが判定できなかった場合に発生します。  
IOExceptionは、ソケット生成時に何らかの入出力エラーが生じた場合に発生します。



## 12-3 サーバの作成



### 12-3 (1)

本文 ▶▶ P.242



**ServerSocket sSocket = new ServerSocket(1234);**



クライアントへのソケットは、引数にポート番号だけを渡して生成します。



### 12-3 (2)

本文 ▶▶ P.243



- ① **close()** メソッド
- ② **getOutputStream()** メソッド
- ③ **accept()** メソッド



接続を切断する際は、必ず **close()** メソッドでソケットを閉じます。



### 12-3 (3)

本文 ▶▶ P.246



**ソケットはクライアントが生成したものを利用するから**



クライアントから要求のあるソケットの実体は、サーバではなくクライアントが持っています。**accept()** メソッドによって、クライアントからの要求を受理し、その要求に基づいてクライアントが用意したソケットをサーバ側で **clinet** と呼んでいるのです。そのため、サーバ側ではクライアントで生成されたソケットのインスタンスにどんな名前が付けられているかなどを知らなくても、**accept()** メソッドで割り当てられたソケットを利用すればよいことになります。

## 12-4 プログラムの改良



### 12-4 (1)

本文 ▶▶ P.250



- ① **ServerSocket**
- ② **accept()**
- ③ **switch**
- ④ **s**
- ⑤ **close()**
- ⑥ **IOException**



サーバ側のソケットを作成するには、**ServerSocket** クラスを使用します。サーバソケットを作成したら、接続の **accept()**・切断の **close()** は必須です。ここで問題になっている語句のほとんどは、サーバプログラムを作成する上での基本です。忘れないようにしましょう。

## 12-5 複数のクライアントへの対応



### 12-5 (1)

本文 ▶▶ P.254



```
01 import java.net.ServerSocket;
02 import java.net.Socket;
03 import java.io.InputStreamReader;
04 import java.io.BufferedReader;
05 import java.io.PrintWriter;
```

```

06 import java.io.IOException;
07
08 class EncodeHandler extends Thread {
09     private Socket incoming;
10     private int number;
11
12     public EncodeHandler(Socket i, int n) {
13         incoming = i;
14         number = n;
15     }
16     public void run() {
17         try {
18             InputStreamReader isr = new InputStreamReader(incoming.getInputStream());
19             BufferedReader in = new BufferedReader(isr);
20             PrintWriter out = new PrintWriter(incoming.getOutputStream(), true);
21
22             out.println("Hello, client No." + number + "! Enter 'Bye' to exit.");
23
24             while (true) {
25                 String str = in.readLine();
26                 System.out.println(str);
27                 if (str != null) {
28                     if (str.toUpperCase().equals("BYE")) {
29                         out.println("Good bye!");
30                         break;
31                     }
32                     String s = "";
33                     for (int i=0; i<str.length(); i++) {
34                         char c = str.charAt(i);
35                         if (((c >= 'A') && (c <= 'Z')) || ((c >= 'a') && (c <= 'z'))) {
36                             switch (c) {
37                                 case 'z': c = 'a';
38                                     break;
39                                 case 'Z': c = 'A';
40                                     break;
41                                 default: c++;
42                             }
43                         }
44                         s += c;
45                     }
46                     out.println("(client No." + number + "):" + s);
47                 }
48             }
49             incoming.close();
50         } catch (Exception e) {
51             System.out.println(e);
52         }
53     }
54 }
55 public class EncodeThreadServer {
56     public static void main(String[] args) {
57         int n = 1;
58
59         try {
60             ServerSocket server = new ServerSocket(1000);
61             while (true) {
62                 Socket incoming = server.accept();
63                 System.out.println("accept client No." + n);
64                 new EncodeHandler(incoming, n).start();
65                 n ++;

```

```

66     }
67     } catch (Exception e) {
68         System.err.println(" エラーが発生しました: " + e);
69     }
70 }
71 }

```

**解説** Threadクラスを継承し、run()メソッドにEncodeServerの処理の一部を移行しています。

## 第13章 GUIとイベント処理

### 13-1 GUIの作成



#### 13-1 (1)

本文 ▶▶ P.258



- ① swing
- ② JFrame
- ③ setSize(400, 300)



このプログラムリストでは、フレームの作成・サイズ設定・表示という、最も簡単な処理を行っています。1行目のimportでとりこんでいるのが、javax.swing.JFrameクラスであることに注意しましょう。

```

01 import javax.swing.JFrame;
02
03 class JFrameTest extends JFrame {
04     public static void main(String[] args) {
05         JFrame f = new JFrame();
06         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
07         f.setSize(400, 300);
08         f.show();
09     }
10 }

```

ウィンドウを閉じたときの処理は、setDefaultCloseOperation()メソッドで指定します。指定できる処理内容は表13-01を確認しましょう。



#### 13-1 (2)

本文 ▶▶ P.264

Q1



- ① f.getContentPane()
- ② FlowLayout.RIGHT
- ③ add



getContentPane()は、フレームからコンテンツ領域を抽出するメソッドです。切り取ったコンテンツ領域をもとに、コンテナを作成します。

Q2



- ① GridLayout
- ② FlowLayout



- ① その名のとおり、格子(grid)状に区切ったコンテナにコンポーネントを配置します。
- ② 左詰の場合は左上からコンポーネントを流して(flow)いき、その行に収まりきらない場合は、次の行の左からコンポーネントを流します。

## 13-2 イベント処理



### 13-2 (1)

本文 ▶▶ P.266

解答

- ① イベントソース
- ② イベント処理
- ③ ActionListener
- ④ actionPerformed()

解説

イベントソースからイベントリスナーにイベントが起きたことが伝わり、その上でイベント処理が実行されます。ActionListener と actionPerformed() は、イベント処理で最も頻繁に使用されるインタフェース・メソッドの組み合わせです。



### 13-2 (2)

本文 ▶▶ P.271

解答

- ① MouseListener
- ② addMouseListener(this)
- ③ mouseClicked
- ④ Color.white

解説

マウス操作に伴うイベント処理では、実装するメソッドが複数あるので注意が必要です。使用しないメソッドであっても、空のまま必ず記述しておきます。

## 13-4 ゲームプログラミング



### 13-4 (1)

本文 ▶▶ P.282

解答

```
01 public void actionPerformed(ActionEvent e) {
02     padY += pd;
03     if (padY < 0) {
04         padY = 0;
05     } else if (padY > hight - 30) {
06         padY = hight - 30;
07     }
08
09     x += 10 * dx;
10     y += 5 * dy;
11
12     if (x > width - 15) {
13         dx *= -1;
14         x = width - 15;
15     }
16     if (y < 5) {
17         dy *= -1;
18         y = 5;
19     } else if (y > hight - 15) {
20         dy *= -1;
21         y = hight - 15;
22     }
23
24     if (x < 30) {
25         if ((y >= padY) && (y <= padY+30)) {
```

```

26         dx *= -1;
27         x = 30;
28     } else {
29         x = width / 2;
30         dx = 1;
31         y = (int)(Math.random()*(height-10)) + 5;
32     }
33     }
34     repaint();
35 }

```

**解説** ボールとパッドの当たり判定は if (x < 30) 以降で行っています。ボールの y 座標がパッドの y 座標の範囲内に入っていなければ、else 以降が実行されることになるので、ここに乱数で y 座標を決める式を書きます。Math.random() は 0 からの乱数を生成するので、壁の厚さを加えて乱数を 5～height-5 の範囲にシフトしています。