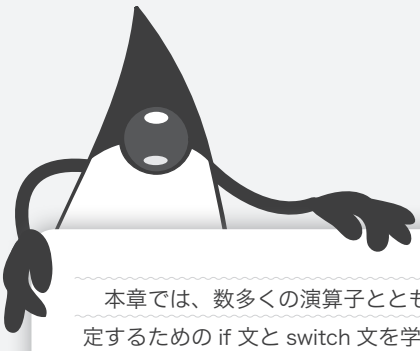
第３章

プログラムの流れの分岐



本章では、数多くの演算子とともに、プログラムの流れを選択的に決定するための if 文と switch 文を学習します。

- □ if 文
- □ switch 文
- □ break 文
- □ 式文と空文
- □ ブロック
- □ アルゴリズム
- □ 演算子の優先順位と結合性
- □ 式と評価
- □ キーワードと識別子

問題3-1

整数値を読み込んで、値が負であれば『その値は負です。』と表示するプログラムを作成せよ。

```
// 読み込んだ整数値は負の値か？

import java.util.Scanner;

class Negative {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      if (n < 0)                                  ← if-then文：if (式) 文
         System.out.println("その値は負です。");   ← n < 0がtrueのときに実行される
   }
}
```

実行例 1

```
整数値：-10↵
その値は負です。
```

実行例 2

```
整数値：35↵
```

if 文（その 1：if-then 文）

キーボードから読み込んだ整数値が 0 より小さければ『その値は負です。』と表示するプログラムです。

読み込んだ値は変数 *n* に格納します。その値を判定する網かけ部は **if 文**（*if statement*）と呼ばれ、その**構文**（文法上の構造）は、次のようになっています。

if（ 式 ）文

これは、**if** 文の一種である **if-then 文**です。先頭の **if**（イフ）は『もしも〜』という意味です。**式**の値を調べて、その値が《真》となったときにのみ**文**を実行します。

なお、本書では、（ ）内に書かれた条件判定のための**式**を**制御式**と呼ぶことにします。

if 文の制御式 *n* **<** 0 で利用している **<** は、左オペランドが右オペランドより小さければ **true**（真）を、そうでなければ **false**（偽）を生成する演算子です。

▶ **true** と **false** は、**論理値リテラル**（*boolean literal*）と呼ばれる論理（**boolean**）型のリテラルです。これらの詳細は、第 5 章で学習します。

if 文のプログラムの流れを表した**フローチャート**（p.131）が **Fig.3-1** です。

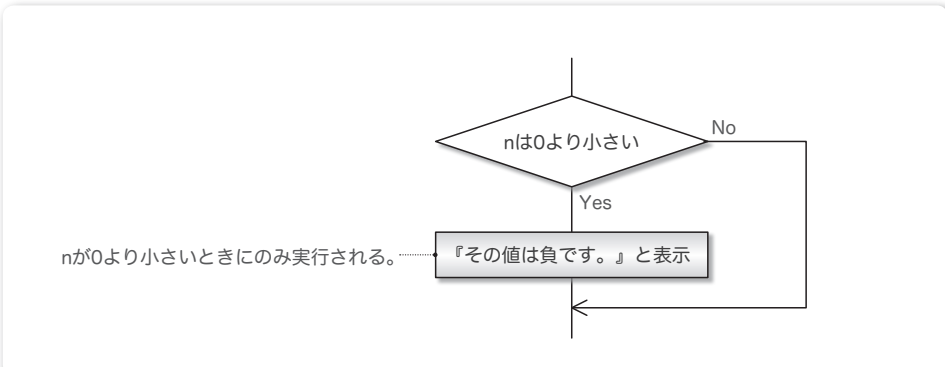


Fig.3-1 プログラムNegativeのif文のフローチャート

実行例①のように、*n*が0より小さければ、制御式の値は**true**となります。そのため、以下の文が実行されて、『その値は負です。』と表示されます。

```
System.out.println("その値は負です。");
```

なお、実行例②に示すように、*n*に入力された値が0以上であれば、この文は実行されません。画面には**何も表示されない**ことになります。

関係演算子

演算子**<**のように、オペランドの大小関係を判定する演算子を、**関係演算子**（*relational operator*）と呼びます。関係演算子には、**Table 3-1** に示す4種類があります。

<=演算子と**>=**演算子の等号を左側にもってきて**=<**あるいは**=>**とすることはできませんし、**<**と**=**の間にスペースを入れて**< =**とすることもできません。間違えないように注意しましょう。

Table 3-1　関係演算子

x **<** *y*	*x*が*y*より小さければ**true**を、そうでなければ**false**を生成。
x **>** *y*	*x*が*y*より大きければ**true**を、そうでなければ**false**を生成。
x **<=** *y*	*x*が*y*より小さいか等しければ**true**を、そうでなければ**false**を生成。
x **>=** *y*	*x*が*y*より大きいか等しければ**true**を、そうでなければ**false**を生成。

if文の構文図

if文には、本問で学習した**if-then**文の他に、次問で学習する**if-then-else**文があります。これらをまとめた**if**文の構文図を**Fig.3-2**に示します。

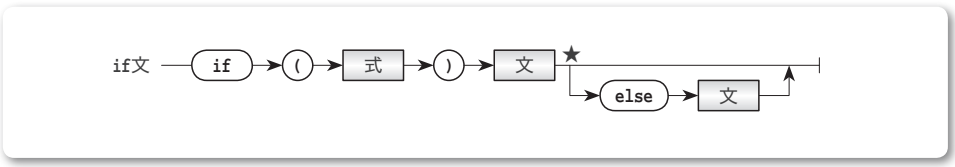


Fig.3-2　if文の構文図

構文図を読むときは、矢印の方向にしたがって進みます。左端からスタートして、ゴールは右端です。分岐点は、どちらに進んでも構いません。

★は分岐点ですから、**if**文の構文図を左端から右端までたどっていくルートは、以下の二つとなります。

```
if ( 式 ) 文              … if-then文
if ( 式 ) 文 else 文      … if-then-else文
```

これが**if**文の形式すなわち構文を表しています。

問題3-2

▶『明解Java入門編』演習3-1 (p.59)

整数値を読み込んで、その絶対値を求めて表示するプログラムを作成せよ。

```
// 読み込んだ整数値の絶対値を表示（その１）

import java.util.Scanner;

class Absolute1 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      if (n >= 0)
         System.out.println("その絶対値は" + n + "です。");
      else
         System.out.println("その絶対値は" + (-n) + "です。");
   }
}
```

if-then-else文：if (式) 文 else 文

実行例 1

```
整数値：62↵
その絶対値は62です。
```

実行例 2

```
整数値：-35↵
その絶対値は35です。
```

```
// 読み込んだ整数値の絶対値を表示（その２）

import java.util.Scanner;

class Absolute2 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      int abs;
      if (n >= 0)
         abs = n;
      else
         abs = -n;
      System.out.println("その絶対値は" + abs + "です。");
   }
}
```

abs = n; ……n >= 0がtrueのときに実行される

abs = -n; ……n >= 0がfalseのときに実行される

if 文（その２：if-then-else 文）

絶対値とは、数値の大きさのことであり、0 からどれだけ離れているかを表す値です。たとえば、5 と -5 の大きさは 5 であり、いずれの絶対値も 5 です。変数 *n* の絶対値は、*n* が 0 以上であれば *n* であり、*n* が負であれば *n* の符号を反転した -*n* です。

n が 0 以上であるかどうかで異なる処理を行うために、プログラム *Absolute1* では **if** 文を利用しています。ただし、その構文は、前問とは異なり、以下のようになっています。

if (式) 文 else 文

この形式の **if** 文は、**if-then-else 文**と呼ばれます。もちろん **else**（エルス）は『～でなければ』という意味です。制御式の値が **true** であれば先頭側の**文**を実行し、**false** であれば末尾側の**文**を実行します。

プログラムの流れは **Fig.3-3** のようになり、*n* が 0 以上であるかどうかで異なる処理が行われます。

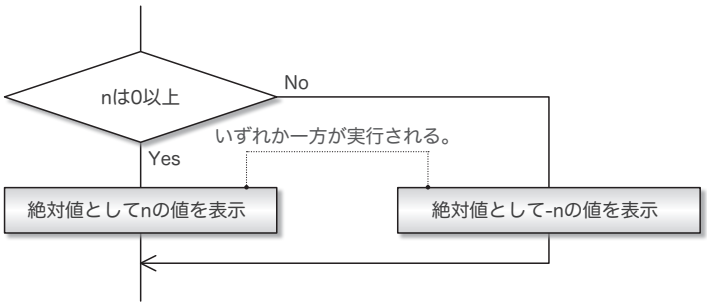


Fig.3-3 プログラムAbsolute1のif文のフローチャート

プログラム *Absolute2* も **if-then-else** 文を利用した解答例です。*n* の絶対値を変数 *abs* にいったん入れておき、それから表示を行います。変数が 1 個増えているものの、表示を行う文が一つにまとめられているという点で優れています。

表示するメッセージの『その絶対値は…』を、『その値の絶対値は…』に変更することを考えましょう。プログラム *Absolute1* では 2 箇所の変更が必要ですが、プログラム *Absolute2* では 1 箇所の変更ですみます。

*

前問のプログラムの **if-then** 文と、本問のプログラムの **if-then-else** 文は、ともにプログラムの流れを**二つに**分岐します。

前問のプログラムでは、変数 *n* の値が 0 以上のときに行うべきことがないため **else** 部がありません。しかし、**Fig.3-1**（p.44）のフローチャートを見ると分かるように、プログラムの流れは、◇部で二つに分岐しています。前問のプログラムの **if-then** 文は、以下に示す **if-then-else** 文で実現することができます。

```
if (n < 0)
   System.out.println("その値は負です。");
else
   ;
```

空文

前章では、文の末尾が原則としてセミコロンであることを学習しました。上に示した **if** 文中の網かけ部のような、単独のセミコロンも文とみなされることになっています。そのような文は、**空文**（*empty statement*）と呼ばれます（構文図は **Fig.3-4**）。

なお、空文を実行しても、何も行われません。

Fig.3-4 空文の構文図

▶『明解Java入門編』演習3-3(p.59)

問題3-3

二つの正の整数値を読み込んで、後者が前者の約数であれば『ＢはＡの約数です。』と表示し、そうでなければ『ＢはＡの約数ではありません。』と表示するプログラムを作成せよ。

```
// 読み込んだ整数値は約数であるかどうか（等価演算子）

import java.util.Scanner;

class Measure1 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("変数Ａ：");  int a = stdIn.nextInt();
      System.out.print("変数Ｂ：");  int b = stdIn.nextInt();

      if (a % b == 0)
         System.out.println("ＢはＡの約数です。");
      else
         System.out.println("ＢはＡの約数ではありません。");
   }
}
```

実行例

```
変数Ａ：12↵
変数Ｂ：4↵
ＢはＡの約数です。
```

等価演算子

変数 b が変数 a の約数であるかどうかは、a を b で割った余りが 0 であるかどうかで判断できます。

たとえば、4 は 12 の約数です（12 を 4 で割った余りは 0 です）が、5 は 12 の約数ではありません（12 を 5 で割った余りは 2 です）。

＊

if 文の制御式で利用している **==** は、左右のオペランドが“等しいかどうか”を判断する演算子です。この演算子 **==** と、“等しくないかどうか”を判断する **!=** 演算子をまとめて**等価演算子**（*equality operator*）と呼びます（**Table 3-2**）。両演算子とも、条件が成立すれば **true** を生成し、成立しなければ **false** を生成します。

Table 3-2　等価演算子

x == *y*	*x* と *y* が等しければ **true** を、そうでなければ **false** を生成。
x != *y*	*x* と *y* が等しくなければ **true** を、そうでなければ **false** を生成。

なお、**!=** 演算子を用いると、本プログラムの **if** 文は以下のように実現できます。二つの文の順序が入れかわることに注意しましょう。

```
if (a % b != 0)
   System.out.println("ＢはＡの約数ではありません。");
else
   System.out.println("ＢはＡの約数です。");
```

問題3-4

前問のプログラムを論理補数演算子！を用いて書きかえよ。

```
// 読み込んだ整数値は約数であるかどうか（論理補数演算子）

import java.util.Scanner;

class Measure2 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("変数Ａ：");  int a = stdIn.nextInt();
      System.out.print("変数Ｂ：");  int b = stdIn.nextInt();

      if (!(a % b == 0))
         System.out.println("ＢはＡの約数ではありません。");
      else
         System.out.println("ＢはＡの約数です。");
   }
}
```

実行例

```
変数Ａ：12↵
変数Ｂ：5↵
ＢはＡの約数ではありません。
```

論理補数演算子

単項演算子！は、**論理補数演算子**（*logical complement operator*）と呼ばれ、オペランドの値が**false**であれば**true**を生成し、**true**であれば**false**を生成します（**Table 3-3**）。

本プログラムの**if**文の判定は、以下のように行われます。

- *a* **%** *b*が0であるとき：*a* **%** *b* **==** 0は**true**。 したがって、**!**(*a* **%** *b* **==** 0)は**false**。
- *a* **%** *b*が0でないとき：*a* **%** *b* **==** 0は**false**。したがって、**!**(*a* **%** *b* **==** 0)は**true**。

Table 3-3 論理補数演算子

!*x*	*x*が**false**であれば**true**を、**true**であれば**false**を生成。

関係演算子と等価演算子は2項演算子であることに注意しましょう。

そのため、たとえば『変数*a*の値が1以上3以下であるか』を

```
1 <= a <= 3          // 駄目！
```

によって判定することはできません。p.58で学習する論理積演算子**&&**を用いた

```
a >= 1 && a <= 3     // ＯＫ！（“aは1以上”かつ“aは3以下”によって判定）
```

によって判定します。

また、たとえば『変数*a*と変数*b*と変数*c*の値が等しいかどうか』を

```
a == b == c          // 駄目！
```

によって判定することはできません。論理積演算子**&&**を用いた

```
a == b && b == c     // ＯＫ！（“aはbと等しい”かつ“bはcと等しい”によって判定）
```

によって判定します。もちろん、

```
a == b && a == c     // ＯＫ！（“aはbと等しい”かつ“aはcと等しい”によって判定）
```

によって判定することもできます。

問題3-5

▶『明解Java入門編』List 3-5(p.58)

キーボードから読み込んだ整数値の符号を判定して表示するプログラムを作成せよ。

```
// 読み込んだ整数値の符号（正／負／０）を判定して表示

import java.util.Scanner;

class Sign {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      if (n > 0)
         System.out.println("その値は正です。");
      else if (n < 0)
         System.out.println("その値は負です。");
      else
         System.out.println("その値は０です。");
   }
}
```

実行例 1

整数値：37↵
その値は正です。

実行例 2

整数値：0↵
その値は０です。

実行例 3

整数値：-35↵
その値は負です。

入れ子となったif文

キーボードから読み込んだ整数値の符号（正であるか／負であるか／0であるか）を判定して表示するプログラムです。

既に学習したように、**if**文には、以下に示す二つの形式があります。

if（ 式 ）文

if（ 式 ）文 **else** 文

このプログラムには**else if** … とありますが、そのような構文が特別に用意されているわけではありません。**if**文は名前のとおり『一種の』文ですから、**else**が制御する文は、当然**if**文であってもよいのです。

プログラム網かけ部の構造を**Fig.3-5**に示します。**if**文の中に**if**文が入った《入れ子》の構造となっていることが分かりますね。

▶ 図では“**System.out.**”を省略しています（p.54の**Fig.3-7**なども同様です）。

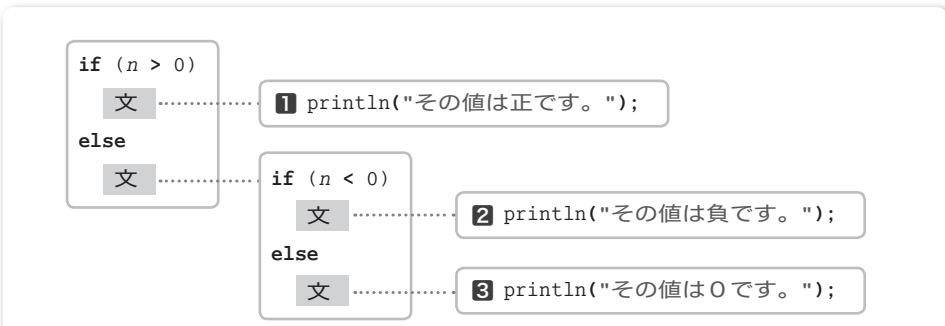


Fig.3-5 プログラムSignのif文（入れ子となったif文）

▶『明解Java入門編』演習3-2(p.59)

問題3-6

前問のプログラムの最後の**else**を、**else if**（*n* == 0）に変更したらどうなるかを検討せよ。

if 文とプログラムの流れの分岐

前問の**if**文を以下のように書きかえるとどうなるかを検討する問題です。

```
if (n > 0)
   System.out.println("その値は正です。");
else if (n < 0)
   System.out.println("その値は負です。");
else if (n == 0)
   System.out.println("その値は０です。");
```

追加した網かけ部にプログラムの流れが到達するのは、*n* > 0 と *n* < 0 の判定の両方が**false**になったとき、すなわち、*n*が0と等しいときのみです。そのため、プログラムを実行すると、変更前と同じ結果が得られます。もっとも、*n*が等しいことが分かっている文脈で、*n*が0と等しいかどうかをわざわざ判定するのは、明らかに無駄です。

ここで、ちょっとした実験をします。この**if**文を以下のように書きかえたプログラムを作ってみましょう。

リスト1

```
if (n == 1)
   System.out.println("それは１です。");  ←1
else if (n == 2)
   System.out.println("それは２です。");  ←2
else if (n == 3)
   System.out.println("それは３です。");  ←3
```

*n*の値が1であれば1が、2であれば2が、3であれば3が実行されます。

ここで、この**if**文から網かけ部を削ったらどうなるかを検討してみましょう。

構文は“**if**（式）文 **else if**（式）文 **else** 文”となります。これは、プログラムの流れを三つに分岐する前問の**if**文と同じ形式です。

ところが、実行の様子は異なります。*n*の値が4でも5でも-10でも、とにかく1と2以外の値であれば3が実行されることになります。

というのも、網かけ部を削る前のリスト1は、以下の**if**文と同じ働きをしているからです。

```
if (n == 1)
   System.out.println("それは１です。");
else if (n == 2)
   System.out.println("それは２です。");
else if (n == 3)
   System.out.println("それは３です。");
else
   ;
```

プログラムの流れは、実質的に四つに分岐しています。前問のプログラムである*Sign*の**if**文とは構造が異なるのですから、網かけ部は削れないのです。

＊

最後の**else**の後ろには、本当に必要なときにのみ**if**（ … ）を置くべきであることが分かりますね。

問題3-7

▶『明解Java入門編』演習3-4(p.61)

二つの変数 *a*, *b* に値を読み込んで、その大小関係を以下のいずれかで表示するプログラムを作成せよ。

『a のほうが大きいです。』『b のほうが大きいです。』『a と b は同じ値です。』

```
// 読み込んだ二つの整数値の大小関係を表示

import java.util.Scanner;

class Balance {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("変数a：");  int a = stdIn.nextInt();
      System.out.print("変数b：");  int b = stdIn.nextInt();

      if (a > b)
         System.out.println("aのほうが大きいです。");
      else if (a < b)
         System.out.println("bのほうが大きいです。");
      else
         System.out.println("aとbは同じ値です。");
   }
}
```

実行例 1

```
変数a：12↵
変数b：3↵
aのほうが大きいです。
```

実行例 2

```
変数a：5↵
変数b：15↵
bのほうが大きいです。
```

二値の大小関係

二つの変数 *a* と *b* の大小関係を判定するプログラムです。前問と同様に、入れ子の **if** 文を利用します。

if 文のフローチャートを示したのが、**Fig.3-6** です。『a のほうが大きいです。』『b のほうが大きいです。』『a と b は同じ値です。』の**いずれか一つ**が表示されます。

▶ すなわち、三つのメッセージのどれも表示されない、あるいは、二つ以上が表示される、ということはありません。

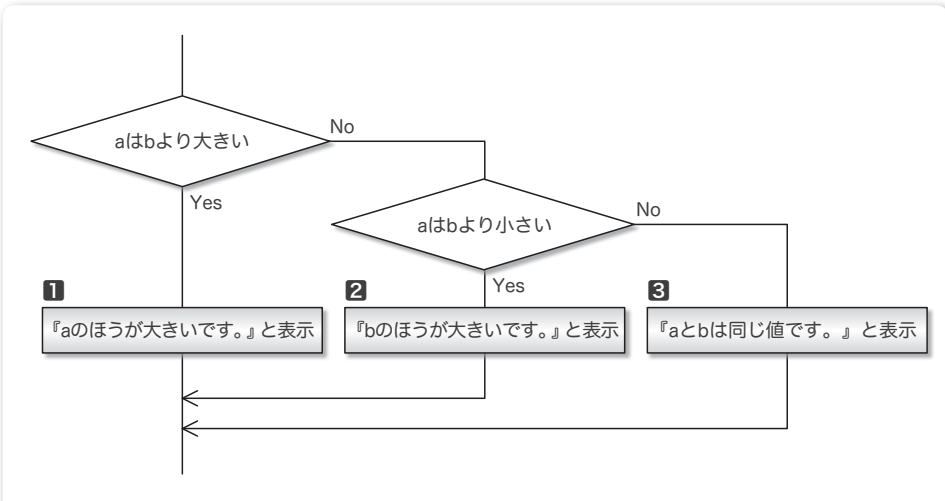


Fig.3-6 プログラムBalanceのif文のフローチャート

```
// 読み込んだ二つの整数値の大小関係を表示（誤り）

import java.util.Scanner;

class BalanceWrong {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("変数a：");  int a = stdIn.nextInt();
      System.out.print("変数b：");  int b = stdIn.nextInt();

      int diff = a - b;
      if (diff > 0)
         System.out.println("aのほうが大きいです。");
      else if (diff < 0)
         System.out.println("bのほうが大きいです。");
      else
         System.out.println("aとbは同じ値です。");
   }
}
```

実行例 1

```
変数a：2147483647↵
変数b：-1↵
bのほうが大きいです。
```

実行例 2

```
変数a：-2147483648↵
変数b：1↵
aのほうが大きいです。
```

プログラム *BalanceWrong* は、変数 *a* から *b* を引いた値を *diff* に入れておき、その値が 0 より大きいか、小さいか、等しいかで、二つの変数の大小関係を判断しようとするものです。

▶ このように書かれたプログラムを、時おり見かけます。

しかし、このプログラムは、変数 *a* と *b* の値によって、期待どおりに動作する場合と動作しない場合とがあります。実際に、ここに示す二つの実行例は期待どおりの結果となっていません。**2147483647 よりも -1 のほうが大きいと判断され、1 よりも -2147483648 のほうが大きいと判断されています。**

このような結果となるのは、無限の数を表せる現実の世界とは異なり、プログラムの世界では**有限の数値しか表せない**ことによります。

前章で学習したように、**int** 型で扱える数値は -2147483648 ～ +2147483647 でしたね。ここに示す実行例のように、2147483647 から -1 を引いた値や、-2147483648 から 1 を引いた値は、**int** 型の表現範囲を超えてしまいます。そのため、正しい判断が行われないのです。

▶『明解Java入門編』演習3-5(p.61)

問題3-8

正の整数値を読み込んで、それが５で割り切れれば『その値は５で割り切れます。』と表示し、そうでなければ『その値は５で割り切れません。』と表示するプログラムを作成せよ。
※正でない値を読み込んだ場合は、『正でない値が入力されました。』と表示すること。

```
// 読み込んだ整数値が正であれば5で割り切れるかどうかを表示

import java.util.Scanner;

class GoInto5 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      if (n > 0)
         if (n % 5 == 0)
            System.out.println("その値は５で割り切れます。");
         else
            System.out.println("その値は５で割り切れません。");
      else
         System.out.println("正でない値が入力されました。");
   }
}
```

実行例 1

整数値：35↵
その値は５で割り切れます。

実行例 2

整数値：36↵
その値は５で割り切れません。

入れ子となった if 文

読み込んだ整数値が正であれば 5 で割り切れるかどうかを表示し、そうでなければ『正でない値が入力されました。』を表示します。

本プログラムの **if** 文の構造を示したのが、**Fig.3-7** です。**if** 文の中に **if** 文が入る構造となっています。ただし、前々問や前問の **if** 文とは構造が違います。

この **if** 文のフローチャートを **Fig.3-8** に示します。『その値は５で割り切れます。』『その値は５で割り切れません。』『正でない値が入力されました。』の**いずれか一つ**が表示されます。

▶ このフローチャートの最初の判定から分岐する“Yes”と“No”は、前問の **Fig.3-6**（p.52）と逆であることに注意しましょう。

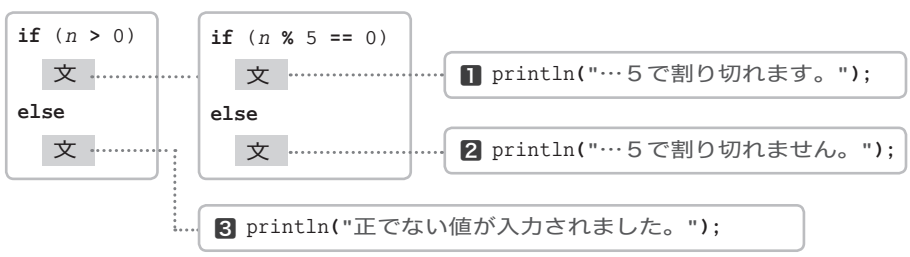


Fig.3-7 プログラムGoInto5のif文（入れ子となったif文）

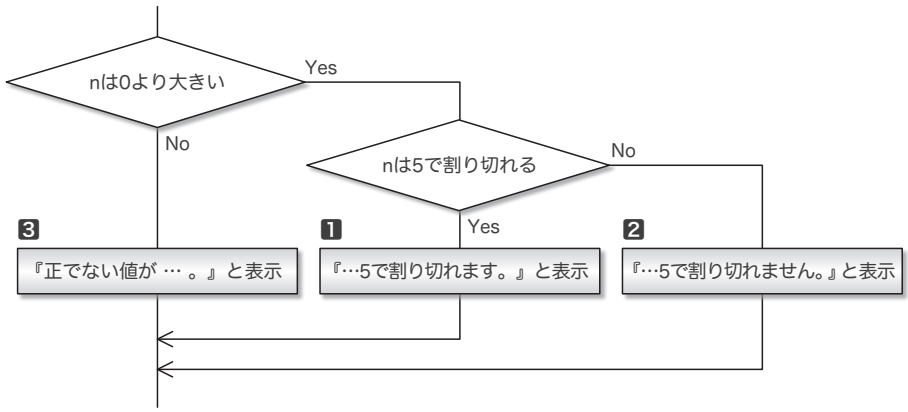


Fig.3-8 プログラムGoInto5のif文のフローチャート

式

これまでに、**式**（*expression*）という用語を何度も使ってきました。式は、以下のものの総称です。

- 変数
- リテラル
- 変数やリテラルを演算子で結合したもの

以下の式を例に考えましょう。

```
abc + 32
```

変数 `abc`、整数リテラル `32`、それらを `+` 演算子で結んだ `abc + 32` のいずれもが式です。

次に、以下の式を考えましょう。

```
xyz = abc + 32
```

ここでは、`xyz`, `abc`, `32`, `abc + 32`, `xyz = abc + 32` のいずれもが式です。

一般に、○○演算子によって結合された式のことを、○○式と呼びます。たとえば、代入演算子によって `xyz` と `abc + 32` が結び付けられた式 `xyz = abc + 32` は、**代入式**（*assignment expression*）です。

式文

第 1 章で学習したように、文の末尾には原則としてセミコロン ; が必要です。たとえば、代入式である `a = c + 32` にセミコロンを付けると文となります。

このように、**式**にセミコロンを付けた**文**が、**Fig.3-9** の構文をもつ**式文**（*expression statement*）です。

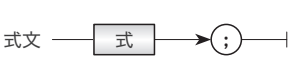


Fig.3-9 式文の構文図

問題3-9

▶『明解Java入門編』演習3-6(p.61)

正の整数値を読み込んで、それが 10 の倍数であれば『その値は 10 の倍数です。』と表示し、そうでなければ『その値は 10 の倍数ではありません。』と表示するプログラムを作成せよ。
※正でない値を読み込んだ場合は、『正でない値が入力されました。』と表示すること。

```
// 読み込んだ整数値が正であれば10の倍数であるかどうかを表示

import java.util.Scanner;

class MultipleOf10 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      if (n > 0)
         if (n % 10 == 0)
            System.out.println("その値は10の倍数です。");
         else
            System.out.println("その値は10の倍数ではありません。");
      else
         System.out.println("正でない値が入力されました。");
   }
}
```

実行例 1

```
整数値：1250
その値は10の倍数です。
```

実行例 2

```
整数値：1254
その値は10の倍数ではありません。
```

式の評価

プログラムの構造は、前問と基本的に同じです。10 の倍数であるかどうかの判定は、10 で割り切れるかどうかによって行います。

＊

式には、基本的に**値**があります。その値は、プログラム実行時に調べられます。式の値を調べることを**評価**（*evaluation*）といいます。

評価のイメージの具体例を示したのが **Fig.3-10** です。この図は、**int** 型の変数 *n* の値が 1254 である場合の例です。もちろん、*n*，10，*n* **%** 10 のいずれもが式です。

変数 *n* の値が 1254 ですから、それぞれの式を評価した値は 1254，10，4 となります。もちろん、三つの値の型はいずれも **int** 型です。

このように、本書では、ディジタル温度計のような図で評価値を示すことにします。左側の小さな文字が《**型**》で、右側の大きな文字が《**値**》です。

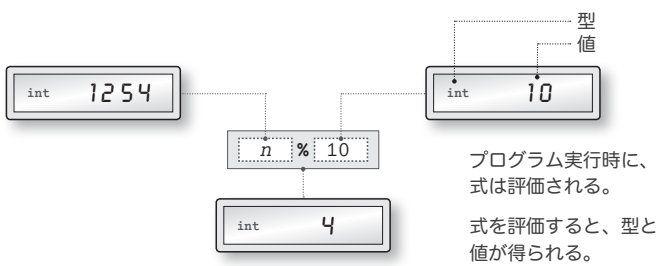


Fig.3-10 式と評価

▶『明解Java入門編』演習3-7(p.61)

問題3-10

正の整数値を読み込んで、それを 3 で割った値に応じて『その値は 3 で割り切れます。』『その値を 3 で割った余りは 1 です。』『その値を 3 で割った余りは 2 です。』のいずれかを表示するプログラムを作成せよ。

※正でない値を読み込んだ場合は、『正でない値が入力されました。』と表示すること。


```
// 読み込んだ整数値が正であれば3で割った剰余を表示

import java.util.Scanner;

class Modulo3 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数値：");
      int n = stdIn.nextInt();

      if (n > 0)
         if (n % 3 == 0)
            System.out.println("その値は3で割り切れます。");
         else if (n % 3 == 1)
            System.out.println("その値を3で割った余りは1です。");
         else
            System.out.println("その値を3で割った余りは2です。");
      else
         System.out.println("正でない値が入力されました。");
   }
}
```

実行例 1

```
整数値：1251
その値は3で割り切れます。
```

実行例 2

```
整数値：1253
その値を3で割った余りは2です。
```

入れ子となった if 文

本プログラムの **if** 文は、前問の **if** 文より少し複雑になっていますが、おおよそ理解できるでしょう。

▶ **if** 文の中に入っている **if** 文の中に **if** 文が入る、という構造となっています。

誤って置かれた空文

以下のプログラムを見てください。*n* の値が正であるときにのみ『その値は正です。』と表示するはずですね。

```
if (n > 0);
   System.out.println("その値は正です。");
```

ところが、このプログラムを実行すると、*n* がどんな値であっても『その値は正です。』と表示されます。そうなる原因は、(*n* > 0) の後ろに置かれた空文 ; (p.47) です。

上のプログラムは、単一の **if** 文ではなく、以下のように解釈されます。

```
if (n > 0) ;                    // if文：n>0であれば空文を実行(何もしない)
System.out.println("その値は正です。"); // if文とは無関係に実行される式文
```

if 文の条件 ( ) の後ろに、誤って空文を置かないように注意しなければなりませんね。

問題3-11

▶『明解Java入門編』演習3-8(p.69)

キーボードから読み込んだ点数に応じて、優／良／可／不可を判定して表示するプログラムを作成せよ。判定は以下のように行うこと。

0～59 → 不可 ／ 60～69 → 可 ／ 70～79 → 良 ／ 80～100 → 優

```
// 点数をもとに成績を判定（その１：論理積演算子&&を利用）

import java.util.Scanner;

class Grade1 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("点数：");
      int point = stdIn.nextInt();

      if (point >= 0 && point <= 59)
         System.out.println("不可");
      else if (point >= 60 && point <= 69)
         System.out.println("可");
      else if (point >= 70 && point <= 79)
         System.out.println("良");
      else if (point >= 80 && point <= 100)
         System.out.println("優");
      else
         System.out.println("不正な点数です。");
   }
}
```

実行例 1

```
点数：68↵
可
```

実行例 2

```
点数：89↵
優
```

実行例 3

```
点数：102↵
不正な点数です。
```

論理積演算子 &&

読み込んだ点数に応じた判定結果（不可／可／良／優）を表示するプログラムです。

点数が《不可》であるかどうかの判断を行うのが、網かけ部の制御式です。

この式で利用している **&&** 演算子は、**Fig.3-11** a に示す《論理積》の演算を行う**論理積演算子**（*logical and operator*）です。この演算子を用いた式 *x* **&&** *y* を評価すると、*x* と *y* がともに **true** であれば **true** が得られ、そうでなければ **false** が得られます。日本語での『*x* かつ *y*』だと考えればよいでしょう（**Table 3-4**：次ページ）。

本プログラム網かけ部の制御式が **true** と評価されるのは、*point* が 0 以上で**かつ** 59 以下であるときです。なお、《可》《良》《優》に関しても、**&&** 演算子によって同様な判定を行っています。

a 論理積　両方とも真であれば真

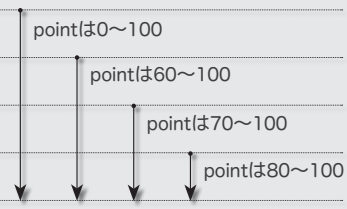
x	*y*	*x* && *y*
true	true	true
true	false	false
false	true	false
false	false	false

b 論理和　一方でも真であれば真

x	*y*	*x* || *y*
true	true	true
true	false	true
false	true	true
false	false	false

Fig.3-11　論理積と論理和の真理値表

```
// 点数をもとに成績を判定（その２：論理和演算子||を利用）

import java.util.Scanner;

class Grade2 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("点数：");
      int point = stdIn.nextInt();

      if (point < 0 || point > 100)
         System.out.println("不正な点数です。");
   ❶ else if (point <= 59)
         System.out.println("不可");
   ❷ else if (point <= 69)
         System.out.println("可");
   ❸ else if (point <= 79)
         System.out.println("良");
   ❹ else
         System.out.println("優");
   }
}
```



論理和演算子 ||

プログラム *Grade2* は、《論理和》を求める**論理和演算子**（*logical or operator*）を利用して実現されています。

表に示すように、式 *x* || *y* を評価すると、*x* と *y* の一方でも **true** であれば **true** が得られ、そうでなければ **false** が得られます。日本語の『*x* または *y*』に近いニュアンスです。

▶ 『僕または彼が行くよ。』といった場合、“僕”か“彼”のいずれか一方のみというニュアンスですが、|| 演算子は、どちらか一方でもという意味であることに注意しましょう。なお、|| は小文字の l（エル）ではなく、縦線記号です。

網かけ部の制御式は、*point* が 0 点未満**または** 100 点を超えるときに **true** と評価され、その結果『不正な点数です。』が表示されます。

そのため、プログラムの流れが❶に到達するのは、*point* が 0 以上かつ 100 以下の場合です。不可の判定では、0 点以上であるかどうかの判定が不要なため、59 点以下であるかどうかのみを調べます。

また、プログラムの流れが❷に到達するのは、*point* が 60 以上かつ 100 以下の場合です。ここでも、60 点以上であるかどうかを判定する必要がないため、69 点以下であるかどうかのみを調べます（❸も同様です）。

プログラム *Grade2* は、論理演算子を利用しているのが最初の 1 回だけであり、それ以降の制御式が単純であるという点で、*Grade1* よりも優れています。

Table 3-4　論理積演算子と論理和演算子

x **&&** *y*	*x* と *y* の両方とも **true** であれば **true** を、そうでなければ **false** を生成。
x **||** *y*	*x* と *y* の一方でも **true** であれば **true** を、そうでなければ **false** を生成。

▶ これらの演算子は短絡評価を行います（p.61）。

▶『明解Java入門編』演習3-9(p.71)

問題3-12

二つの実数値を読み込んで、大きいほうの値を表示するプログラムを作成せよ。

```
// 読み込んだ二つの実数値の大きいほうの値を表示（その１：if文）

import java.util.Scanner;

class Max2A {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("実数a：");  double a = stdIn.nextDouble();
      System.out.print("実数b：");  double b = stdIn.nextDouble();

      double max;     // 大きいほうの値
      if (a > b)
         max = a;
      else
         max = b;

      System.out.println("大きいほうの値は" + max + "です。");
   }
}
```

実行例

```
実数a：25.7↵
実数b：15.3↵
大きいほうの値は25.7です。
```

条件演算子

二つの値を読み込んで大きいほうの値を表示するプログラムです。変数 *a*，*b* に読み込んだ値を比較して、*a* のほうが *b* より大きければ変数 *max* に *a* を代入し、そうでなければ変数 *max* に *b* を代入します。その結果、**if** 文の実行が終了したときには、変数 *max* には大きいほうの値が入っていることになります。

＊

右ページの *Max2B* は、**if** 文を用いずに実現したプログラムです。網かけ部で利用している **? :** は、右ページの **Table 3-5** に示す**条件演算子**（*conditional operator*）です。

この演算子を用いた**条件式**（*conditional expression*）における評価の様子をまとめたのが **Fig.3-12** です。変数 *max* に入れられるのは、*a* が *b* より大きければ *a* の値、そうでなければ *b* の値となります。

条件式は、**if** 文を凝縮したようなものであり、Java のプログラムで好んで使われます。

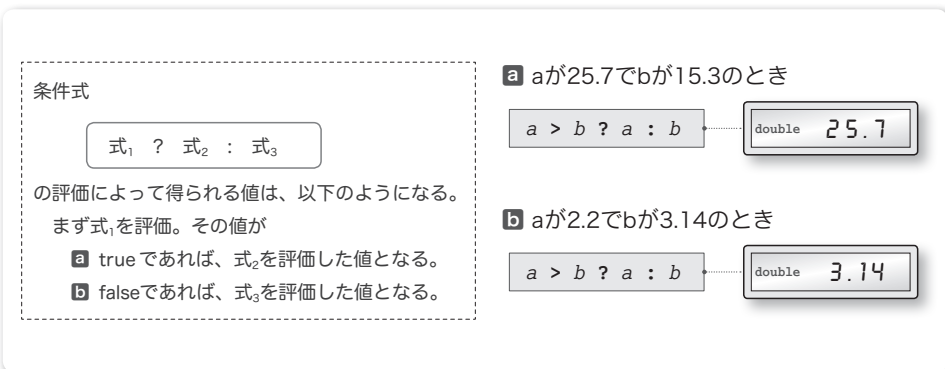


Fig.3-12 条件式の評価

```
// 読み込んだ二つの実数値の大きいほうの値を表示（その２：条件演算子）

import java.util.Scanner;

class Max2B {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("実数a：");  double a = stdIn.nextDouble();
      System.out.print("実数b：");  double b = stdIn.nextDouble();

      double max = a > b ? a : b;   // 大きいほうの値
      System.out.println("大きいほうの値は" + max + "です。");
   }
}
```

Table 3-5 条件演算子

x ? *y* : *z*	*x* が **true** であれば *y* を評価した値を、そうでなければ *z* を評価した値を生成。

▶ *x* を評価した値が **true** であれば *z* は評価されませんし、**false** であれば *y* は評価されません。

なお、以下のように、大きいほうの値を求める条件式を println の中に埋め込めば、変数 *max* を使わずにすみます。

```
System.out.println("大きいほうの値は" + (a > b ? a : b) + "です。");
```

▶ 網かけ部を囲む（ ）を省略することはできません。その理由は p.79 で学習します。

短絡評価

前問のプログラム *Grade1*（p.58）において、変数 *point* が -10 であるとして、最初の **if** 文の制御式 *point* >= 0 && *point* <= 59 で行われる《不可》の判定を考えましょう。

左オペランドの *point* >= 0 は **false** です。そうすると、右オペランドの式 *point* <= 59 を調べなくても式全体が **false** となる（すなわち《不可》でない）ことは自明です。

そのため、**&& 演算子の左オペランドを評価した値が false であれば、右オペランドの評価は行われないことになっています。**

＊

引き続き、プログラム *Grade2*（p.59）の最初の制御式 *point* < 0 || *point* > 100 に着目しましょう。もし *point* が -10 であれば、*point* < 0 は **true** ですから、右オペランドの *point* > 100 を調べるまでもなく、式全体が **true** となる（すなわち《不正な点数》である）ことが分かります。

そのため、**|| 演算子の左オペランドを評価した値が true であれば、右オペランドの評価は行われないことになっています。**

論理演算の式全体の評価結果が、左オペランドの評価の結果のみで明確になる場合に、右オペランドの評価が行われないことを**短絡評価**（*short circuit evaluation*）と呼びます。

▶ 演算子 && と || によく似た演算子として、演算子 & と | があります。演算子 & は論理積を求め、演算子 | は論理和を求めます。ただし、& と | による演算では、**短絡評価が行われません。**そのため、& と | は、論理演算のために使われることは少なく、ビット単位の論理演算を行うために用いられるのが一般的です。詳しくは、第 7 章で学習します。

▶『明解Java入門編』演習3-10(p.71)

問題3-13

二つの整数値を読み込んで、それらの値の差を表示するプログラムを作成せよ。

```
// 読み込んだ二つの整数値の差を表示（その１：if文）

import java.util.Scanner;

class Diff2A {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数a：");  int a = stdIn.nextInt();
      System.out.print("整数b：");  int b = stdIn.nextInt();

      int diff;
      if (a >= b)
         diff = a - b;
      else
         diff = b - a;
      System.out.println("それらの差は" + diff + "です。");
   }
}
```

実行例 1

```
整数a：12↵
整数b：3↵
それらの差は9です。
```

実行例 2

```
整数a：3↵
整数b：12↵
それらの差は9です。
```

```
// 読み込んだ二つの整数値の差を表示（その２：条件演算子）

import java.util.Scanner;

class Diff2B {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数a：");  int a = stdIn.nextInt();
      System.out.print("整数b：");  int b = stdIn.nextInt();

      int diff = a >= b ? a - b : b - a;

      System.out.println("それらの差は" + diff + "です。");
   }
}
```

二値の差

二つの変数の差は、値を減算した結果の絶対値として求められます。したがって、負とはならず、0 または正の値となります。

プログラム *Diff2A* は **if** 文を用いて実現され、*Diff2B* は条件演算子を用いて実現されています。

なお、以下のように、差を求める条件式を println の中に埋め込めば、変数 *diff* を使わずにすみます。

```
System.out.println("それらの差は" + (a >= b ? a - b : b - a) + "です。");
```

▶『明解Java入門編』演習3-11(p.71)

問題3-14

二つの整数値を読み込んで、それらの値の差が 10 以下であれば、『それらの差は 10 以下です。』と表示し、そうでなければ『それらの差は 11 以上です。』と表示するプログラムを作成せよ。

```
// 読み込んだ二つの整数値の差が10以下かどうかを表示（その１）

import java.util.Scanner;

class Diff2digits1 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数Ａ：");  int a = stdIn.nextInt();
      System.out.print("整数Ｂ：");  int b = stdIn.nextInt();

      int diff = a >= b ? a - b : b - a;

      if (diff <= 10)
         System.out.println("それらの差は10以下です。");
      else
         System.out.println("それらの差は11以上です。");
   }
}
```

実行例 1

```
整数Ａ：12
整数Ｂ：3
それらの差は10以下です。
```

実行例 2

```
整数Ａ：35
整数Ｂ：23
それらの差は11以上です。
```

```
// 読み込んだ二つの整数値の差が10以下かどうかを表示（その２）

import java.util.Scanner;

class Diff2digits2 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数Ａ：");  int a = stdIn.nextInt();
      System.out.print("整数Ｂ：");  int b = stdIn.nextInt();

      int diff = a >= b ? a - b : b - a;

      System.out.println("それらの差は" +
                         (diff <= 10 ? "10以下" : "11以上") + "です。");
   }
}
```

条件式を用いた文字列の連結

いずれのプログラムも、二つの変数 a と b の差を条件演算子を用いて求めています（左ページの *Diff2B* と同じです）。異なるのは表示の実現法です。プログラム *Diff2digits1* は **if** 文を用い、プログラム *Diff2digits2* は条件演算子を用いています。

条件演算子に慣れないうちは、プログラム *Diff2digits2* は、理解しにくいかもしれません。しかし、"それらの差は" と "です。" の部分に関して、以下のメリットがあります。

- タイプするのが 1 回ですむ。
- 他の文字列への変更が容易である。

▶『明解Java入門編』演習3-12(p.73)

問題3-15

キーボードから読み込んだ三つの整数値の最小値を求めて表示するプログラムを作成せよ。

```
// 三つの整数値の最小値を求める

import java.util.Scanner;

class Min3 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数a：");  int a = stdIn.nextInt();
      System.out.print("整数b：");  int b = stdIn.nextInt();
      System.out.print("整数c：");  int c = stdIn.nextInt();

      int min = a;
      if (b < min) min = b;
      if (c < min) min = c;

      System.out.println("最小値は" + min + "です。");
   }
}
```

実行例

```
整数a：3↵
整数b：1↵
整数c：2↵
最小値は1です。
```

三値の最小値

三つの変数 *a*，*b*，*c* に整数値を読み込んで、その最小値を求めて表示するプログラムです。

三値の最小値を求める手順は、以下のようになっています。

① *min* を *a* の値で初期化する。
② *b* の値が *min* よりも小さければ、*min* に *b* の値を代入する。
③ *c* の値が *min* よりも小さければ、*min* に *c* の値を代入する。

このように《処理の流れ》を定義した規則を**アルゴリズム**（*algorithm*）と呼びます。三値の最小値を求めるアルゴリズムを表したフローチャートが **Fig.3-13** です。

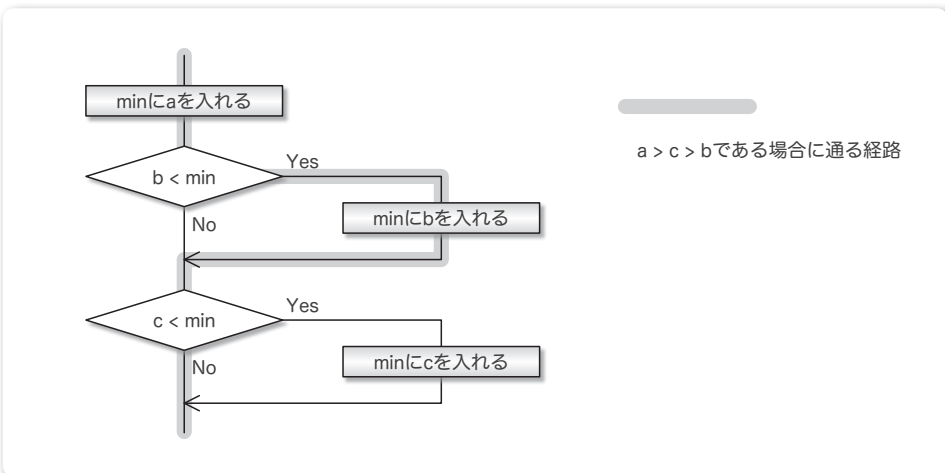


Fig.3-13　三値の最小値を求めるフローチャート

実行例のように、変数 *a*，*b*，*c* に対して 3，1，2 を入力すると、プログラムの流れはフローチャート上の太い線 の経路をたどります。このときの変数 *min* は、**Fig.3-14** a に示すように変化します。

これ以外の値を想定して、フローチャートをなぞってみましょう。たとえば、変数 *a*，*b*，*c* の値が、1，2，3 や 3，2，1 であっても、正しく最小値を求められますね。もちろん、三つの値が 5，5，5 とすべて等しかったり、3，1，3 と二つが等しくても、正しく最小値を求められます。

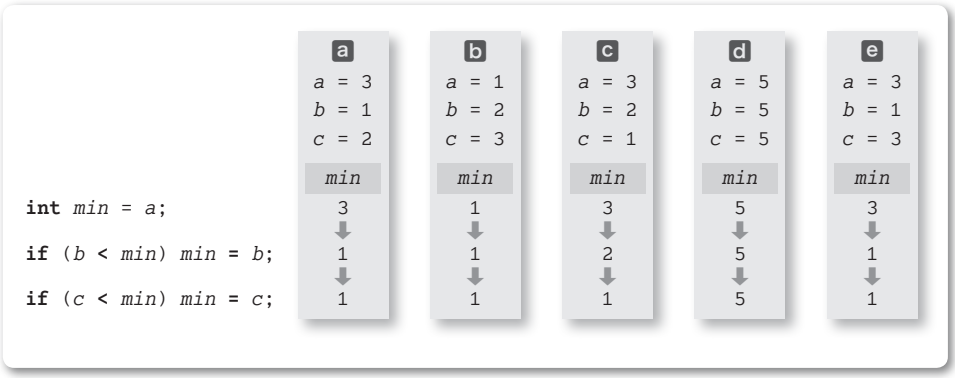


Fig.3-14 三値の最小値を求める過程での変数minの変化

『アルゴリズム』という用語は、JIS X0001 で以下のように定義されています。

> **問題を解くためのものであって、明確に定義され、順序付けられた有限個の規則からなる集合。**

もちろん、いくら曖昧さのないように記述されていても、変数の値によって、問題が解けたり解けなかったりするのでは、正しいアルゴリズムとはいえません。

▶『明解Java入門編』演習3-13(p.73)

問題3-16

キーボードから読み込んだ三つの整数値の中央値を求めて表示するプログラムを作成せよ。

例　1，2，3 の中央値は 2 で、3，2，3 の中央値は 3 で、4，4，4 の中央値は 4 である。

```
// 三つの整数値の中央値を求める

import java.util.Scanner;

class Med3 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数a：");  int a = stdIn.nextInt();
      System.out.print("整数b：");  int b = stdIn.nextInt();
      System.out.print("整数c：");  int c = stdIn.nextInt();

      int med;
      if (a >= b)
         if (b >= c)
            med = b; ←― ⒶⒷⒻⒼ
         else if (a <= c)
            med = a; ←― ⒹⒺⒽ
         else
            med = c; ←― Ⓒ
      else if (a > c)
         med = a;    ←― Ⓘ
      else if (b > c)
         med = c;    ←― ⒿⓀ
      else
         med = b;    ←― ⓁⓂ

      System.out.println("中央値は" + med + "です。");
   }
}
```

実行例

```
整数a：152↵
整数b：324↵
整数c：75↵
中央値は152です。
```

三値の中央値

三値の大小関係の組合せは、**Fig.3-15** に示す図によって列挙できます（このような図を**決定木**と呼びます）。

左端の枠から始めて、◯内の条件が成立すれば上側の実線を、成立しなければ下側の点線をたどっていきます。

右端の□内に示すのが、三つの変数 *a*，*b*，*c* の大小関係です。全部で 13 種類あります。なお、各枠の上に示す数値は、値の組合せの一例です。

最大値・最小値とは異なり、**中央値**（*median*）を求める手続きは、非常に複雑です（そのため、何種類ものアルゴリズムが考えられます）。プログラム *Med3* は一例です。なお、中央値を *med* に代入する文Ⓐ，Ⓑ，…，Ⓜは、**Fig.3-15** と対応しています。

＊

なお、以下に示すのは別解です。

```
if ((b >= a && c <= a) || (b <= a && c >= a))
   med = a;
else if ((a > b && c < b) || (a < b && c > b))
   med = b;
else
   med = c;
```

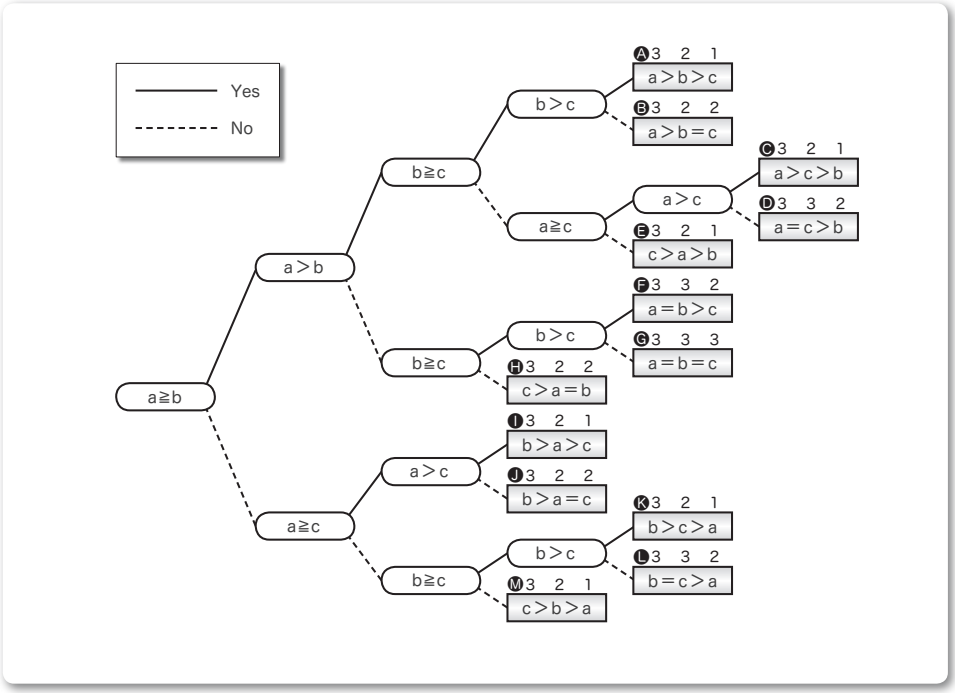


Fig.3-15 三値の大小関係の組合せ

構文図の読み方

構文図に慣れるために、ここでは、いくつかの具体例を理解していくことにしましょう。**Fig.3-16** に示す構文図を見てください。

Ⓐ 先頭から末尾まで行って終了するルートと、分岐点から下におりて《文》を通るルートがあります。

『0 個の文、または 1 個の文』を表します。

Ⓑ 先頭から末尾まで行って終了するルートがあるのはⒶと同じです。分岐点で下におりて《文》を通って先頭に戻ることができます。戻った後は、末尾まで行って終了することもできますし、再び分岐点から《文》を通って、先頭に戻ることもできます。

『0 個以上の、任意の個数の文』を表します。

Ⓒ Ⓐと同じく**『0 個の文、または 1 個の文』**を表します。

Ⓓ 先頭から末尾までのルートの途中に《文》があります。また、分岐点で下におりて先頭に戻ることができます。戻った後は、再び《文》を通過して終了することもできますし、再び分岐点から先頭に戻ることもできます。

『1 個以上の、任意の個数の文』を表します。

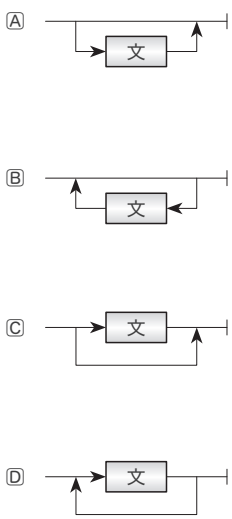


Fig.3-16 構文図

問題3-17

▶『明解Java入門編』演習3-14(p.77)

二つの整数値を読み込んで、小さいほうの値と大きいほうの値の両方を表示するプログラムを作成せよ。二つの整数値が等しい場合は、『二つの値は同じです。』と表示すること。

```
// 二つの整数値の小さいほうの値と大きいほうの値を求めて表示

import java.util.Scanner;

class MinMaxEq {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("整数a：");  int a = stdIn.nextInt();
      System.out.print("整数b：");  int b = stdIn.nextInt();

      if (a == b)
         System.out.println("二つの値は同じです。");
      else {
         int min, max;      // 小さいほうの値／大きいほうの値
         if (a < b) {       // aがbより小さければ
1           min = a;
            max = b;
         } else {
2           min = b;
            max = a;
         }
         System.out.println("小さいほうの値は" + min + "です。");
         System.out.println("大きいほうの値は" + max + "です。");
      }
   }
}
```

実行例 1

```
整数a：12↵
整数b：3↵
小さいほうの値は3です。
大きいほうの値は12です。
```

実行例 2

```
整数a：17↵
整数b：17↵
二つの値は同じです。
```

ブロック

二つの整数値を読み込んで、小さいほうの値と大きいほうの値の両方を求めるプログラムです。値が等しく `a == b` が成立する場合は、『二つの値は同じです。』と表示します。

二つの値が等しくない場合の実行の様子を **Fig.3-17** で考えていきましょう。

```
if (a == b)                                                    if (式)
   System.out.println("二つの値は同じです。");                        文
else {                                                         else {
   int min, max;                                                   宣言文
   if (a < b) {
      min = a;
      max = b;
   } else {                                                        if文
      min = b;
      max = a;
   }
   System.out.println("小さいほうの値は" + min + "です。");            式文
   System.out.println("大きいほうの値は" + max + "です。");            式文
}                                                              }
```

Fig.3-17　プログラムMinMaxEqのif文の構造

この図の網かけ部は、{ } の中に以下の文が入る構造となっています。

- 変数を宣言する宣言文
- 大きいほうの値と小さいほうの値を求める **if** 文
- 表示を行う式文

このように、文の並びを { } で囲んだものを**ブロック**（*block*）と呼びます。

Fig.3-18 が、ブロックの構文図です。ブロックは、構文上は**単一の文とみなされる**ことになっています。

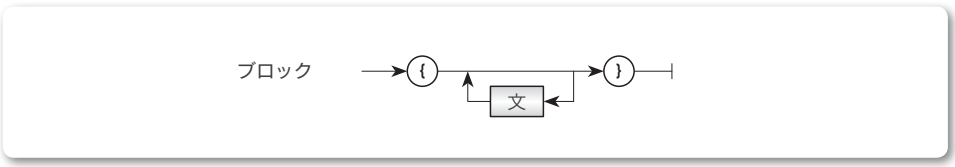


Fig.3-18　ブロックの構文図

ここで、**if** 文の構文を思い出しましょう。以下に示すいずれかの形式でしたね。

```
if ( 式 ) 文
if ( 式 ) 文 else 文
```

すなわち、**if** 文が制御する文は**一つだけです**（**else** 以降も**一つだけです**）。したがって、本プログラムの **if** 文は、この構文にきちんとのっとっていることになります。

単一の文が要求される箇所で、複数の文を実行させねばならないときは、それらをまとめてブロックとして実現しなければなりません。

ブロックの先頭行は、変数 *min* と *max* の宣言です。 このようにブロック中で宣言された変数は、そのブロックのみで使えるものとなります。換言すると、ブロック内でのみ利用する変数は、そのブロック内で宣言することを原則とすべきです。

＊

ブロックの中に置かれた内側の **if** 文は、小さいほうの値と大きいほうの値を求めます。

この **if** 文は、*a* が *b* より小さければ❶のブロック

```
{ min = a; max = b; }
```

を実行し、そうでなければ❷のブロック

```
{ min = b; max = a; }
```

を実行します。

この **if** 文から { } を二つとも削除したらどうなるかを実験してみましょう。

if 文とみなされるのはⒶ部であり、続くⒷ部は式文です。その後ろの **else** は **if** とは対応していません。そのためコンパイル時にエラーとなります。

```
   Ⓐif文              Ⓑ式文     ⬇ 理解不能!!
if (a < b) min = a;  max = b;  else  min = b;  max = a;
if ( 式 )   文          式;
```

▶『明解Java入門編』演習3-15(p.77)

問題3-18

二つの整数値を読み込んで降順（大きい順）にソートするプログラムを作成せよ。

```
// 二つの変数を降順（大きい順）にソート

import java.util.Scanner;

class Sort2Descending {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("変数a：");  int a = stdIn.nextInt();
      System.out.print("変数b：");  int b = stdIn.nextInt();

      if (a < b) {      // aがbより小さければ
         int t = a;     // それらの値を交換
         a = b;
         b = t;
      }
      System.out.println("a≧bとなるようにソートしました。");
      System.out.println("変数aは" + a + "です。");
      System.out.println("変数bは" + b + "です。");
   }
}
```

実行例

```
変数a：13↵
変数b：57↵
a≧bとなるようにソートしました。
変数aは57です。
変数bは13です。
```

二値のソート

二つの変数 *a*，*b* に整数値を読み込んで、降順（大きい順）、すなわち *a* ≧ *b* となるように**ソートする**（*sort*：並べかえる）プログラムです。

▶ 本プログラムが行うのは、二値の『降順ソート』です。大きい順ではなくて、小さい順に並べる手続きは、『昇順ソート』と呼ばれます。

ソートの手順は、以下のようになっています。

- *a* の値が *b* より小さいとき … *a* と *b* の値を交換する。
- そうでないとき … 何もしない（そのままでよい）。

a と *b* の値の交換を行うのが、網かけ部のブロックです。ブロックの先頭行は、変数 *t* の宣言です。これは、二つの変数の値を交換するのに必要となる、作業用の変数です。

■ 二値の交換

さて、ブロックで行っている《二値の交換》の手順は、以下のとおりです。

1. *a* の値を *t* に保存しておく。
2. *b* の値を *a* に代入する。
3. *t* に保存しておいた最初の *a* の値を *b* に代入する。

この三つのステップで、*a* と *b* の値の交換は完了です。

変数 *a* が 13 で、変数 *b* が 57 のときの交換の様子を示したのが **Fig.3-19** です。交換後は、*a* が 57 で *b* が 13 になります。

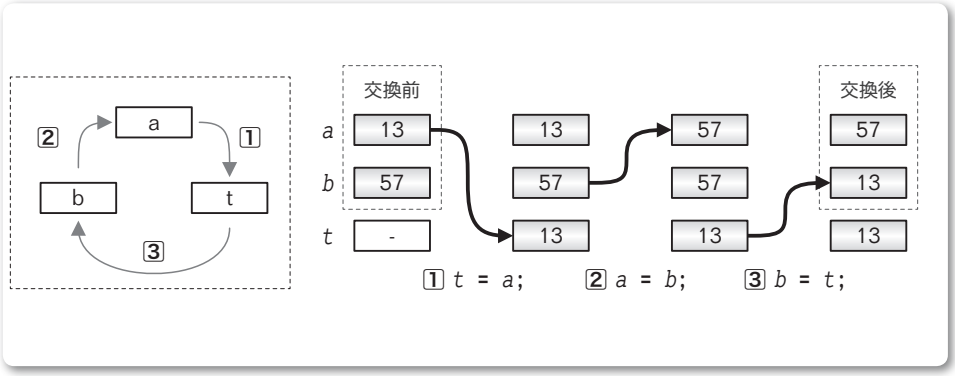


Fig.3-19 二値の交換手順

if 文の構文に関する補足

右に示す **if** 文を見てください。この **if** 文において、**文**$_1$ と **文**$_2$ が実行されるのは、どのような条件のときでしょう。

```
if (a == 1)
   if (b == 1)
      文1
else
   文2
```

二つの文が実行される条件は、下表に示す《条件 1 》のようになっている、と感じられるのではないでしょうか。

しかし、そうではありません。このような **if** 文における **else** は、**最も近い if と対応する**ことになっているからです。すなわち、上の **if** 文の **else** は、**if** (*a* == 1) に対応するものではなく、**if** (*b* == 1) に対応するものなのです。

条件 1

文	実行される条件
文$_1$	*a* が 1 であり *b* も 1 であるとき
文$_2$	*a* が 1 でないとき

条件 2

文	実行される条件
文$_1$	*a* が 1 であり *b* も 1 であるとき
文$_2$	*a* が 1 であり *b* が 1 でないとき

この **if** 文のインデントは『嘘をついている』、といっていいでしょう。以下のように記述すべきですね。こうすると紛らわしさがなくなります。

二つの文が実行される条件は、表の《条件 2 》のようになっていることが明確になります。*a* の値が 1 でなければ、何も実行されないことに注意しましょう。

```
if (a == 1)
   if (b == 1)
      文1          ← aが1のときに実行される文（if文）
   else
      文2
```

もしも《条件 1 》に基づいて二つの文を実行する必要がある場合は、ブロックを導入して、以下のように実現しなければなりません。

```
if (a == 1) {
   if (b == 1)      ← aが1のときに実行される文（ブロック）
      文1
} else
      文2           ← aが1でないときに実行される文
```

問題3-19

▶『明解Java入門編』演習3-16(p.77)

三つの整数値を読み込んで昇順（小さい順）にソートするプログラムを作成せよ。

```
// 三つの変数を昇順（小さい順）にソート

import java.util.Scanner;

class Sort3 {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("変数a：");  int a = stdIn.nextInt();
      System.out.print("変数b：");  int b = stdIn.nextInt();
      System.out.print("変数c：");  int c = stdIn.nextInt();

      if (a > b) {   // aがbより大きければaとbを交換
         int t = a;  a = b;   b = t;                    ←1
      }

      if (b > c) {   // bがcより大きければbとcを交換
         int t = b;  b = c;   c = t;                    ←2
      }

      if (a > b) {   // aがbより大きければaとbを交換
         int t = a;  a = b;   b = t;                    ←3
      }

      System.out.println("a≦b≦cとなるようにソートしました。");
      System.out.println("変数aは" + a + "です。");
      System.out.println("変数bは" + b + "です。");
      System.out.println("変数cは" + c + "です。");
   }
}
```

実行例

```
変数a：53↵
変数b：35↵
変数c：42↵
a≦b≦cとなるようにソートしました。
変数aは35です。
変数bは42です。
変数cは53です。
```

三値のソート

三つの変数 a，b，c を昇順にソートするプログラムです。

三値のソートは、三つの **if** 文で行います。その原理を示したのが、**Fig.3-20** です。a，b，c にそれぞれ 5，3，2 が格納されているとして、ソートの手順を理解していきましょう。

▶ 三つの **if** 文が実行する各ブロック内で変数 t が宣言されています。ブロック内で宣言された変数は、そのブロックに所属する特有のものであって、その名前もブロック内でのみ通用します（p.69）ので、三つの変数 t の名前が衝突することはありません。

1 まず a と b の値を比べます。小さい順に並べかえるのですから、もし左側の a が右側の b よりも大きいのでは話になりません。そこで、それらの値を交換します。

2 2 番目の **if** 文によって、b と c に対して同じ操作を行います。すなわち、左側の b が右側の c よりも大きければ、それらの値を交換します。

ここまでの 2 段階の手続きによって、最も大きい値が c に格納されます。というのも、これは右側の図に示す「トーナメント」だからです。この図と照らし合わせれば、最初の二つの **if** 文は最強（最大）の値を c に格納することが分かるでしょう。

3 最大値が c に格納されたわけですから、次に行うのは、残った二値 a，b の最大値を b に格納することです。これは、第 2 位を決定するための「敗者復活戦」です。**if** 文を

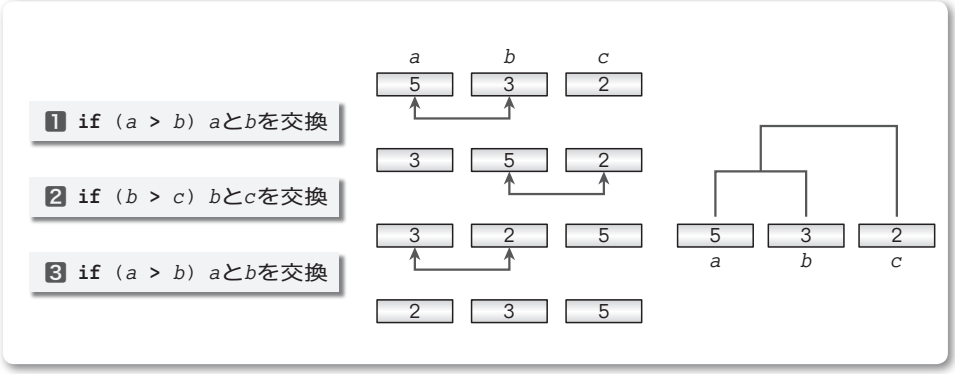


Fig.3-20 三値のソート（その１）

実行すると、*a* と *b* の大きいほうの値が *b* に格納されます。

最大値が *c* に格納され、２番目に大きい値が *b* に格納されたわけですから、*a* には当然最小値が格納されています。これでソートは完了です。

＊

別の数値の組合せでのソートの手順を示したのが、**Fig.3-21** です。ソート前の変数 *a*，*b*，*c* の値は 5，3，4 です。3では、*a* > *b* が成立しないため、交換が行われません。

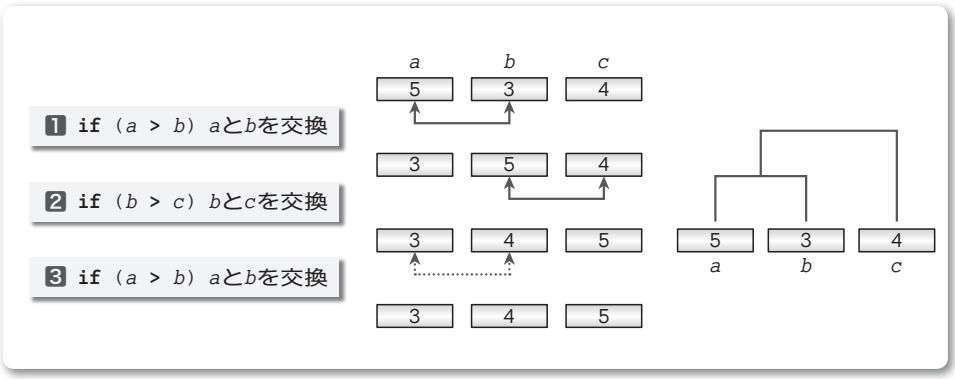


Fig.3-21 三値のソート（その２）

▶ ここに示したソートの手順は、単純交換ソート（バブルソート）と呼ばれるソートアルゴリズムを、ソート対象が３要素である場合に特化したものです。

ソートのアルゴリズムには、単純交換ソート、シェルソート、マージソート、クイックソートなど、数多くのものが考案されています。これらのアルゴリズムについては、『明解 Java によるアルゴリズムとデータ構造』で学習いただけます（巻末の広告をご覧ください）。

▶『明解Java入門編』演習3-17(p.83)

問題3-20

0, 1, 2のいずれかの値の乱数を生成し、0であれば"グー"を、1であれば"チョキ"を、2であれば"パー"を表示するプログラムを作成せよ。

```
// 生成した乱数に応じてジャンケンの手を表示

import java.util.Random;

class FingerFlashing {

   public static void main(String[] args) {
      Random rand = new Random();

      System.out.print("コンピュータが生成した手：");
      int hand = rand.nextInt(3);       // 0〜2の乱数

      switch (hand) {
       case 0: System.out.println("グー");   break;
       case 1: System.out.println("チョキ"); break;
       case 2: System.out.println("パー");   break;
      }
   }
}
```

実行例

コンピュータが生成した手：**チョキ**

switch文

生成した乱数の値に応じてジャンケンの《手》を表示するプログラムです。0であれば"グー"、1であれば"チョキ"、2であれば"パー"と表示します。

本プログラムでは、**switch文**（*switch statement*）によって処理の流れの分岐を実現しています。**switch**文は、ある式を評価した値によってプログラムの流れを複数に分岐させる文であり、その名のとおり、《切替えスイッチ》のようなものです。

■ ラベル

プログラムの流れが**switch**文に差しかかると、まず（ ）内に書かれた制御式の評価が行われます。そして、その結果に基づいて、**switch**文内のどこにプログラムの流れを移すのかが決定されます。もし制御式*hand*の値が1であれば、プログラムの流れは

```
case 1:
```

と書かれた目印へと一気に移ります（**case**と1の間には空白が必要です）。このように、プログラムの飛び先を示す目印が**ラベル**（*label*）です。

▶ 異なるラベルが同じ値をもつことは許されません。また、ラベルの値は《定数》でなければならず、変数は許されません。

プログラムの流れがラベルに飛んだ後は、その後ろに置かれた文が順次実行されます。したがって、*hand*が1であれば、まず、以下の文が実行されます（**Fig.3-22**）。

```
System.out.println("チョキ");
```

■ break文

プログラムの流れが、**break文**（*break statement*）と呼ばれる**break;**に出会うと、**switch**文の実行は終了することになっています。

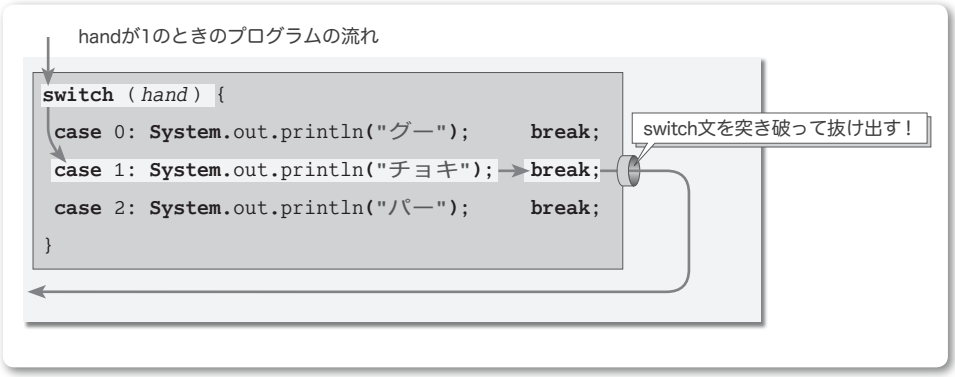


Fig.3-22 switch文におけるプログラムの流れとbreak文の働き

breakとは、『破る』『抜け出る』という意味です。**break**文が実行されると、プログラムの流れは、それを囲んでいる**switch**文を突き破って抜け出るのです。

したがって、*hand*の値が1のときには、画面には"チョキ"とだけ表示されます。その下に置かれている"パー"を表示する文は実行されません。

もちろん、*hand*が0であれば"グー"とだけ表示され、2であれば"パー"とだけ表示されることになります。

▶ **break**文によって抜け出た後は、**switch**文の次に置かれた文が実行されます。本プログラムの場合は、**switch**文の後ろには文がありませんので、プログラムの実行が終了します。

なお、*hand*の値が0，1，2以外の値であれば、一致するラベルがありませんので、**switch**文は実質的に素通りされます（何も表示されません）。

■ 最後のcaseに置かれたbreak文

case 2を見てください。"パー"の表示の後に**break**文が置かれています。これを削除してもプログラムの動作は変わりません（**break**文があってもなくても、**switch**文は終了しますので）。それでは、この**break**文は何のために置いてあるのでしょうか。

もしジャンケンの手が4種類に増えて、値が3の「プー」が追加されることになったとしましょう。**switch**文を以下のように変更することになりますね。

```
switch (hand) {
 case 0: System.out.println("グー");    break;
 case 1: System.out.println("チョキ");  break;
 case 2: System.out.println("パー");    break;
 case 3: System.out.println("プー");    break;
}
```

追加するのは点線部です。今回の**switch**文では、**case** 2:の**break**文を省略できないことは、いうまでもありません。

もし変更前のプログラムの**case** 2:に**break**文がなかったら、『ラベルの追加に伴って必要になる**break**文の追加を忘れてしまう。』というミスを犯すかもしれません。最後の**break**は、ラベルの追加に伴うプログラムの変更を確実かつ容易にするためのものであることが分かりましたね。

▶『明解Java入門編』演習3-18(p.83)

問題3-21

月を 1 ～ 12 の整数値として読み込んで、それに対応する季節を表示するプログラムを作成せよ。

```
// 読み込んだ月の季節を表示

import java.util.Scanner;

class Season {

   public static void main(String[] args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("何月ですか：");
      int month = stdIn.nextInt();

      switch (month) {
       case  3 :
       case  4 :
       case  5 : System.out.println("春");  break;
       case  6 :
       case  7 :
       case  8 : System.out.println("夏");  break;
       case  9 :
       case 10 :
       case 11 : System.out.println("秋");  break;
       case 12 :
       case  1 :
       case  2 : System.out.println("冬");  break;
       default : System.out.println("そんな月はありません。");  break;
      }
   }
}
```

実行例 1

```
何月ですか：6↵
夏
```

実行例 2

```
何月ですか：11↵
秋
```

実行例 3

```
何月ですか：13↵
そんな月はありません。
```

default ラベル

今回の **switch** 文には、前のプログラムにはなかった

```
default:
```

というラベルがあります。分岐のための制御式を評価した値が、どの **case** とも一致しないときは、プログラムの流れは、**default** ラベルへと飛ぶことになっています。

また、**break** 文がない箇所では、プログラムの流れが次の文へと“落ちていく”ことになっています。そのため、*month* が 3 であっても、4 であっても、5 であっても『春』と表示されます。

本プログラムの **switch** 文内のラベルの出現順序を変えると、実行結果が変わります。**switch** 文を使うときは、ラベルの順序に配慮が必要です。

選択文

if 文と **switch** 文は、プログラムの流れを分岐させるという点で共通です。これら二つの文をまとめて**選択文**（*selection statement*）と呼びます。

Fig.3-23 に示すのが **switch** 文の構文図です。（ ）で囲まれた判定のための制御式は、**整数型**でなければなりません。

▶ 具体的には **char**, **byte**, **short**, **int**, **Character**, **Byte**, **Short**, **Integer**, 列挙型のいずれかでなければなりません。実数や文字列は許されません。

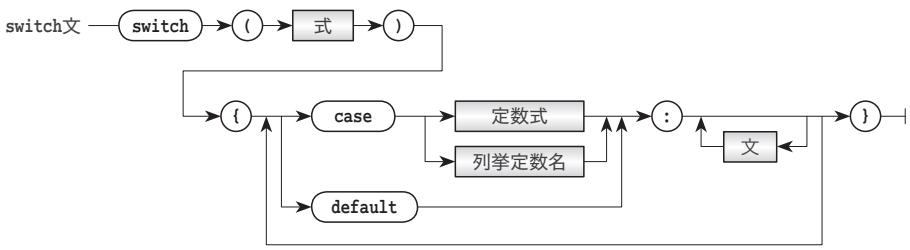


Fig.3-23 switch文の構文図

break 文の構文図を **Fig.3-24** に示します。

▶ **break** の後ろに識別子（p.78）が置かれるプログラム例は、次章で学習します。

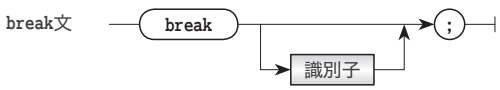


Fig.3-24 break文の構文図

if 文と **switch** 文のどちらを使っても実現できる分岐は、**switch** 文を利用して実現したほうが読みやすくなる傾向があります。そのことを、**Fig.3-25** に示す二つのプログラム部分で考えましょう。

まずは、**if** 文をじっくり読んでみましょう。先頭三つの **if** は *a* の値を調べ、最後の **if** は *b* の値を調べています。変数 *c* に 80 が代入されるのは、*a* が 1，2，3 のいずれでもなく、かつ *b* が 4 であるときです。

連続した **if** 文において、分岐のための比較対象となるのは、必ずしも**単一の式であるとは限りません**。最後の判定は、**if**（*a* == 4）と読み間違えられたり、**if**（*a* == 4）の書き間違いではないかと誤解されたりするかもしれません。

その点、**switch** 文のほうは、全体の見通しがよいため、プログラムを読む人が、そのような疑念を抱くことが少なくなります。

```
if (a == 1)
   c = 10;
else if (a == 2)
   c = 20;
else if (a == 3)
   c = 50;
else if (b == 4)
   c = 80;
```

```
// 左のif文を書き直したswitch文

switch (a) {
 case 1  : c = 10;  break;
 case 2  : c = 20;  break;
 case 3  : c = 50;  break;
 default : if (b == 4) c = 80; break;
}
```

Fig.3-25 等価なif文とswitch文

キーワード

if や **else** といった語句には、特別な意味が与えられています。このような語句は**キーワード**（*keyword*）と呼ばれ、プログラム作成者が変数などの名前として利用することはできません。Java のキーワードは **Table 3-6** に示す 50 個です。

Table 3-6　キーワードの一覧

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	syncronized
this	throw	throws	transient	try	void
volatile	while				

▶ **const** と **goto** はキーワードとして予約されていますが、実際には使うことはできません。正式なキーワードではないものの、それに準ずる存在として、**true**、**false**、**null** があります。

区切り子

キーワードは、一種の『単語』のようなものです。その単語を区切るために使われる記号が**区切り子**（*separator*）です。区切り子は **Table 3-7** に示す 9 個です。

Table 3-7　区切り子の一覧

[	]	(	)	{	}	,	:	.

識別子

識別子（*identifier*）とは、変数（第 2 章）・ラベル（第 4 章）・メソッド（第 7 章）・クラス（第 8 章）などに与えられる名前のことです。名前は自由に与えることができますが、以下の規則にのっとらなければなりません。

- 識別子の 1 文字目は、いわゆる文字（$ と _ を含む）でなければならない。
- 識別子の 2 文字目以降は、いわゆる文字（$ と _ を含む）か数字でなければならない。

数字が使えるのは、2 文字目からであることを覚えておきましょう。また、キーワードに加えて、**true** と **false** と **null** も識別子として利用することはできません。

▶ $ は Java コンパイラがバイトコードを生成する際に内部的に利用する文字です。ソースプログラムでは使わないことが推奨されています。

Java では、Unicode という文字コード体系を用いる（第 15 章）ため、“いわゆる文字”には、アルファベットだけでなく漢字文字なども含まれます。

正しい識別子の例と、誤った識別子の例を示します。

▪ 正しい識別子の例

```
v     v1      va      $x      $1      _1      壱      If      Xデイ   X1000
```

▪ 誤った識別子の例

```
1     9801    1$      !!      if      #911    -x     {x}     #if     0-1-2
```

▶ 変数名などの識別子として、漢字や仮名なども利用できます。ただし、日本語以外の言語圏の人々が理解できないものになってしまいますので、利用すべきではありません。

リテラル

整数リテラル・浮動小数点リテラル・文字列リテラルなども、プログラムを構成する要素の一つです。本章では数多くの**演算子**（*operator*）を学習しました。Java で利用できる全演算子をまとめたのが**Table 3-8**（次ページ）です。

演算子

演算子の一覧表は、先頭側のほうが**優先度**（*precedence*）が高くなるように表記しています。たとえば、乗除算を行う * と / が、加減算を行う + や - より優先度が高いのは、私たちが日常生活で使用する計算での規則と同じです。したがって、式 *a* + *b* * *c* は、(*a* + *b*) * c ではなく *a* + (*b* * *c*) と解釈されます。+ のほうが前（左）にあるにもかかわらず、後ろ（右）にある * の演算が優先されます。

+ 演算子よりも**優先度が低い演算子**を文字列の連結時に使うときには、( ) が必要となることに注意しましょう。以下に例を示します。

```
// *は+より優先度が高いので( )は不要
System.out.println("aとbの積は" + a * b + "です。");

// ?:は+より優先度が低いので( )が必要
System.out.println("大きいほうの値は" + (a > b ? a : b) + "です。");
```

▶ たとえ優先度が + より高い演算子を用いた式であっても、( ) で囲んでおいたほうが読みやすくなる傾向があります。なるべく ( ) で囲むようにしましょう。

同じ優先度の演算子が連続するときに左右どちらの演算を先に行うかを示すのが、**結合規則**（*associativity*）です。すなわち、２項演算子を○と表した場合、式 *a* ○ *b* ○ *c* を

(*a* ○ *b*) ○ *c*　　左結合

とみなすのが左結合の演算子であり、

a ○ (*b* ○ *c*)　　右結合

とみなすのが右結合の演算子です。

たとえば、減算を行う２項 - 演算子は左結合ですから、

```
5 - 3 - 1  ⇨  (5 - 3) - 1          // ２項 - 演算子は左結合
```

です。もし右結合だったら、5 - (3 - 1) と解釈され、演算結果も違ってきます。

Table 3-8 全演算子の一覧

優先順位	演算子	形 式	名 称	結合規則
1	[ ]	*x*[*y*]	インデックス演算子	左
	( )	$x(arg_{opt})$	メソッド呼出し演算子	
	.	*x*.*y*	メンバアクセス演算子	
	++	*x*++	後置増分演算子	
	--	*x*--	後置減分演算子	
2	++	++*x*	前置増分演算子	右
	--	--*x*	前置減分演算子	
	+	+*x*	単項 + 演算子	
	-	-*x*	単項 - 演算子	
	!	!*x*	論理補数演算子	
	~	~*x*	ビット単位の補数演算子	
3	new	new	new 演算子	左
	( )	( )	キャスト演算子	
4	*	*x* * *y*	乗除演算子	左
	/	*x* / y		
	%	*x* % *y*		
5	+	*x* + *y*	加減演算子	左
	-	*x* - *y*		
6	<<	*x* << *y*	シフト演算子	左
	>>	*x* >> *y*		
	>>>	*x* >>> *y*		
7	<	*x* < *y*	関係演算子	左
	>	*x* > *y*		
	<=	*x* <= *y*		
	>=	*x* >= *y*		
	instanceof	*x* instanceof *y*	instanceof 演算子	左
8	==	*x* == *y*	等価演算子	左
	!=	*x* != *y*		
9	&	*x* & *y*	ビット論理積演算子	左
10	^	*x* ^ *y*	ビット排他的論理和演算子	左
11	|	*x* | *y*	ビット論理和演算子	左
12	&&	*x* && *y*	論理積演算子	左

13	`||`	*x* `||` *y*	論理和演算子	左
14	`? :`	*x* `?` *y* `:` *z*	条件演算子	左
15	`=`	*x* `=` *y*	単純代入演算子	右
	`*=`	*x* `*=` *y*	複合代入演算子	
	`/=`	*x* `/=` *y*		
	`%=`	*x* `%=` *y*		
	`+=`	*x* `+=` *y*		
	`-=`	*x* `-=` *y*		
	`<<=`	*x* `<<=` *y*		
	`>>=`	*x* `>>=` *y*		
	`>>>=`	*x* `>>>=` *y*		
	`&=`	*x* `&=` *y*		
	`^=`	*x* `^=` *y*		
	`|=`	*x* `|=` *y*		

代入式の評価

原則として《式》は、その値を評価することができるのでしたね。**代入式を評価すると、代入後の左オペランドの型と値が得られます**。

たとえば、変数 *x* が **int** 型であれば、代入式 *x* `=` 2 を評価して得られるのは、代入後の左オペランド *x* の型と値である『**int** 型の 2』です。

ここで変数 *a* と *b* が **int** 型であるとして、以下の式を考えましょう（**Fig.3-26**）。

```
a = b = 1  ⇨  a = (b = 1)        // 代入演算子 = は右結合
```

まず、代入式 *b* `=` 1 によって *b* に 1 が代入されます（図①）。その後、代入式 *b* `=` 1 を評価した値（代入後の *b* の値）である“**int** 型の 1”が *a* に代入されます（図②）。

その結果、*a* にも *b* にも 1 が代入されることになります。

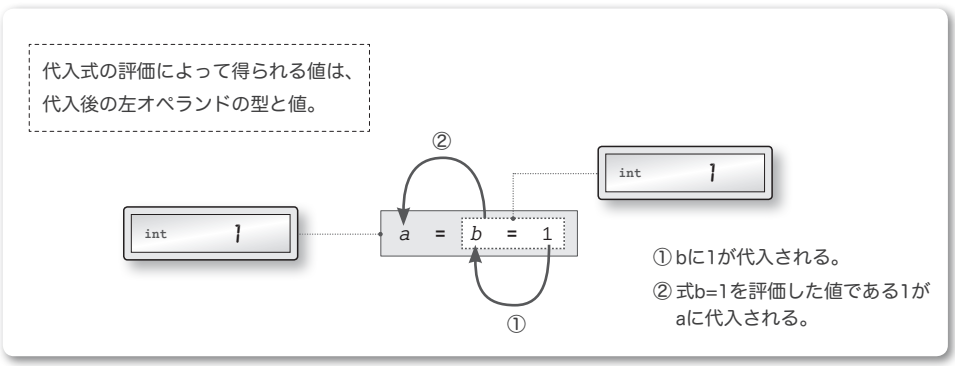


Fig.3-26 代入式の評価

錬成問題

▪ 変数・リテラル・変数やリテラルを演算子で結合したものを［ (1) ］と呼ぶ。
［ (1) ］にセミコロンが付いて文になったものは［ (2) ］であり、セミコロンだけの文は［ (3) ］である。

▪ 任意の個数の文を { } で囲んだ文は［ (4) ］と呼ばれる。

▪ *a* + *b* * *c* では、左側の加算よりも右側の乗算のほうが先に行われる。これは、演算子 * の［ (5) ］が演算子 + のそれよりも高いためである。なお、同じ［ (5) ］の演算子が連続する場合に、左右どちらの演算が先に行われるのかは、演算子の［ (6) ］に基づいて決定される。

▪ オペランドの大小関係を判定する演算子 <，>，<=，>= の総称は［ (7) ］演算子であり、等しいか等しくないかを判定する演算子 ==，!= の総称は［ (8) ］演算子である。

▪ 演算子 &&，|| の総称は［ (9) ］演算子であり、演算子 ! の名称は［ (10) ］演算子であり、演算子 ? : の名称は［ (11) ］演算子である。

▪ 論理演算の式全体の評価結果が、左オペランドの評価のみで明確になる場合に、右オペランドの評価が行われないことを［ (12) ］という。

▪ **if** や **else** などの語句は、特別な意味が与えられており、［ (13) ］と呼ばれる。変数やメソッドなどに与えられる名前は［ (14) ］と呼ばれる。

▪ 変数を値の小さい順や大きい順で並べかえることを［ (15) ］と呼ぶ。

▪ アルゴリズムとは、問題を解くためのものであって、［ (16) ］に定義され、順序付けられた［ (17) ］の規則からなる集合のことである。

▪ 以下に示すプログラム部分の実行結果を示せ。

```
int a = 1, b = 3, c = 5;
System.out.println("a < b     : " + (a < b));
System.out.println("a <= b    : " + (a <= b));
System.out.println("b > c     : " + (b > c));
System.out.println("b >= c    : " + (b >= c));
System.out.println("a == b    : " + (a == b));
System.out.println("a != b    : " + (a != b));
System.out.println("a - b - c : " + (a - b - c));
System.out.println("c - b - a : " + (c - b - a));
System.out.println("a = b = c : " + (a = b = c));
System.out.println("c = b = a : " + (c = b = a));
```

```
a < b     :
a <= b    :
b > c     :
b >= c    :
a == b    :
a != b    :
a - b - c :
c - b - a :
a = b = c :
c = b = a :
```

(18)

▪ 以下に示す真理値表の空欄を埋めよ。

論理積

x	*y*	*x* && *y*
false	false	(19)
false	true	(20)
true	false	(21)
true	true	(22)

論理和

x	*y*	*x* || *y*
false	false	(23)
false	true	(24)
true	false	(25)
true	true	(26)

▪ 以下に示すのは、変数 *c* の最下位桁が 0 であれば（たとえば 50 や 250 であれば）、『変数 c の最下位桁は 0 です。』と表示するプログラムである。

```
(27) (c (28) 10 == 0)
   System.out.println("変数cの最下位桁は0です。");
```

▪ 以下に示すのは、変数 *x* の値が 0 であれば『0 です。』と表示し、そうでなければ『非 0 です。』と表示するプログラムである。

```
(29) (x == 0)
   System.out.println("０です。");
(30)
   System.out.println("非０です。");
```

▪ 以下に示すのは、変数 *a* の絶対値を表示するプログラムである。

```
System.out.println("aの絶対値：" + (31));
```

▪ 以下に示すのは、変数 *a* と変数 *b* の小さいほうの値と大きいほうの値の両方を表示するプログラムである。

```
System.out.println("aとbの小さいほうの値：" + (32));
System.out.println("aとbの大きいほうの値：" + (33));
```

▪ 以下に示すのは、変数 *w* の値が 0 であれば『晴れ』、1 であれば『雨』、2 であれば『曇り』と表示するプログラムである。

```
(34) (w) {
  (35) 0: System.out.println("晴れ");  (36) ;
  (35) 1: System.out.println("雨");    (36) ;
  (35) 2: System.out.println("曇り");  (36) ;
}
```

▪ 真と偽を表す論理値リテラルは、それぞれ (37) と (38) である。

▪ 加算を行う + 演算子は (39) 結合であり、代入を行う = 演算子は (40) 結合である。

▪ 以下に示すのは、変数 m の値が 3，4，5 のいずれかであれば『春です。』と表示し、そうでなければ『春ではありません。』と表示するプログラムである。

```
if ( (41) || (42) || (43) )
   System.out.println("春です。");
else
   System.out.println("春ではありません。");
```

```
if ( (44) && (45) )
   System.out.println("春です。");
else
   System.out.println("春ではありません。");
```

```
if ( (46) ( (47) || (48) ))
   System.out.println("春です。");
else
   System.out.println("春ではありません。");
```

▪ 以下に示すのは、変数 m の値を 3 で割った剰余に応じて、『３で割り切れます。』『３で割った剰余は１です。』『３で割った剰余は２です。』のいずれかを表示するプログラムである。

```
if ( (49) )
   System.out.println("３で割り切れます。");
else (50)
   System.out.println("３で割った剰余は１です。");
else (51)
   System.out.println("３で割った剰余は２です。");
```

```
switch ( (52) ) {
 case (53) System.out.println("３で割り切れます。");       break;
 case (54) System.out.println("３で割った剰余は１です。"); break;
 case (55) System.out.println("３で割った剰余は２です。"); break;
}
```

▪ 以下に示すのは、変数 a の値が b の値の倍数かどうかを判断し、その結果を表示するプログラムである。

```
if ( (56) )
   System.out.println("aはbの倍数です。");
else
   System.out.println("aはbの倍数ではありません。");
```

▪ 以下に示すのは、変数 a と b の値が $a \leqq b$ となるようにソートするプログラムである。

```
if ( (57) ) {
   int t = a;  a = (58);  b = (59);
}
```

▪ 右に示す switch 文は、int 型変数 n の値が 1，2，3 のときに、それぞれ (60)、(61)、(62) と表示する。

```
switch (n) {
 case  1: System.out.print("A");
 case  2: System.out.print("B"); break;
 default: System.out.print("C");
}
```

▪ 右に示すプログラムは、int 型変数 n の値が 0，1 のときに、それぞれ (63)、(64) と表示する。

```
if (n == 0);
   System.out.println("A");
```

▪ 以下に示すのは、変数 a の値が b の値の平方根より小さければ『a は b の平方根より小さい。』と表示するプログラムである。

```
if (a * (65) < (66) )
   System.out.println("aはbの平方根より小さい。");
```

▪ 以下に示すのは、変数 a，b，c の値がすべて等しいときにのみ『★』と表示するプログラムである。

```
if ( (67) && (68) )
   System.out.print("★");
```

▪ 以下に示すのは、変数 a と b に読み込んだ整数値が両方とも奇数か、一方のみ奇数か、両方とも偶数であるかを表示するプログラムである。

```
import (69) .util.Scanner;

(70) Even {
   public (71) void main( (72) args) {
      Scanner stdIn = new Scanner(System.in);

      System.out.print("ａ：");  int a = (73) .nextInt();
      System.out.print("ｂ：");  int b = (73) .nextInt();

      int c = 0;
      if ( (74) ) c = c + 1;
      if ( (75) ) c = c + 1;

      if ( (76) == 0)
         System.out.println("両方とも奇数です。");
      else if ( (77) )
         System.out.println("一方のみ奇数です。");
      else if ( (78) )
         System.out.println("両方とも偶数です。");
   }
}
```