

Chapter 7. 条件分岐

7.1. 変数宣言と初期値代入

■変数の宣言について

変数について、少し復習を兼ねて、もう一度まとめておきましょう。

★欲しい変数は最初に宣言しておくこと

変数は欲しいと思ったら、いつでも宣言して導入することができます。他のコンパイラ系の言語では最初の方にまとめて宣言しておかなければならなりません、Java 言語では途中で導入することができます。

```
int    y;                // まず変数yは、お約束だろう、、、
y = 10;
int    z;                // そういえば変数zも必要になった、、、
z = 20;
```

導入したら、その名前を使い続けなければいけません。途中で気が変わって他の名前にしてしまうと、コンパイラが、「そんな名前の変数は宣言してないだろう!」と文句をブーブー言ってきます。少なくとも、2回はエラーを出すでしょう（当社調べ）。

```
int    wow;              //   なんとなくwowと宣言してみた
x = 67 % 5 * 2;          //   教科書では、変数名がxになっていたの、
                          //   何も考えず、そのまま書いた
```

上のような場合に、変数 x が出現する以前にどこにもその宣言がないときには、エラーになります。

★いくつ変数は必要なのでしょう？

第4章でも説明しましたが、一つの変数でも式を使えばいくらかでも多種の値を取らせることができます。次の繰返しの記述を見てください。

```
int x;
x = 1;
while ( x < 10 ){
    gc.drawLine( x * 10, 100 / x, 100 - x * 5, 50 + x );
    x++;
}
```

drawLineの最初のパラメータは、10→20→30→40→…と変化していきます。第2番目のパラメータは、100→50→33→25→…と変化していきます。第3番目は、95→90→85→80→…と変化し、第4番目は51→52→53→54→…と変化していきます。一つの変数しか使っていませんが、各パラメータの最初の値や、変化の度合いはまったく異なっています。

これがわかりにくいと思った人は、次のように別々に変数を用意するのも一手でしょう。繰返しの継続条件は、変数 $y1$ の値をもとに記述されています。

```

int x1, y1, x2, y2;
x1 = 10; y1 = 1; x2 = 95; y2 = 51;
while (y1 < 10){
    gc.drawLine(x1, 100 / y1, x2, y2);
    x1 += 10; y1 ++; x2 -= 5; y2 ++;
}

```

どちらを取るかはプログラマの好みに依存しますが、どちらを取るにせよ、後で見たときにわかりやすければ、それに越したことはないでしょう。変数の数が多くなってしまってもわかりにくくなりますし、かと言ってすべて一つの変数にするのもわかりにくいでしょう。アルゴリズムがうまく記述できるのに適した数の変数を使うように考えていってください。

■宣言時における初期値代入

変数を宣言する際には、同時に初期値を代入することができます。これによって、宣言と最初の代入を1つにすることができます。次のような書式で記述します。

▼宣言時の初期値代入の書式：

```

型名 変数名 = 式；

```

例えば整数型の変数の場合は、次のように使います。

```

int    z = 10;           // 整数型の変数zを宣言し、その値を10にする
int    y = z * 67;       // 整数型の変数yを宣言し、その値をz * 67にする

```

上記の2番目の例のように既に存在する変数を使つての初期化することも可能です。しかし、右辺の式中出现する変数は、それまでに値が代入されていなければいけません。

同じ型の変数の宣言は、カンマ（,）で区切っていくつも書いても構いません。

```

int    x = 20, y = 30;    // 整数型の変数xとyを宣言し、その値を設定する
int    w = x * 20, u = y / 20; // 同様

```

なお、次のように記述した場合は、変数 *y* は初期化されますが、変数 *x* は初期化されないので注意してください。

```

int    x, y = 30;        // 整数型の変数xとyを宣言し、yの値だけ設定する

```

7.2. 条件分岐（その1）

■条件分岐とは？

条件分岐は、制御構文の一つです。条件によって、特定の処理を行なうか行なわないかを選択させることができます。例えば、繰返しである回数のときだけ、別の処理をするようなプログラムを記述する必要があると考えてみてください。条件分岐の構文を繰返しの構文と組み合わせれば、そのような処理を簡潔に記述することができます。

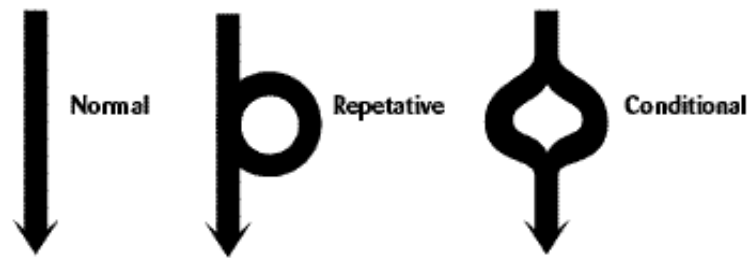


図7-1 制御構文による制御の流れ

条件分岐には、if文とswitch文があります。しかも、if文には書き方が3つあります。ここでは、if文の3つの書式を紹介します。

★if文の書式1と書式2

3つの書式のうちの書式1と書式2は、ある条件が成立するかどうか判断して処理の流れを変えるものです。

▼if文の書式1

```
if ( 条件式 ) {  
    条件式が満足されたときに実行されること  
}
```

この書式の意味は、条件式が満足されるときだけ、波括弧で囲まれた中、すなわちブロック（注1）の中の文を実行するということです。満足されなければ、ブロックの中の部分は何も実行されません。単にスキップされて、それ以降の文に処理が移るだけです。

たとえば、次のような記述があった場合、変数 `money` の値が100よりも大きければ、ブロックの中の `System.out.println` の呼出しを実行します。

```
int    money = 110;                                // moneyに何かの整数値を代入しておく  
  
if ( money > 100 ){  
    System.out.println( "今日は何か食べれる" );  
}
```

書式1だけでもいろいろな記述はできるのですが、条件が満たされなかった場合に実行することも記述できれば便利です。そのために、次の書式2が用意されています。

▼if文の書式2

```
if ( 条件式 ) {  
    条件式が満足されたときに実行されること  
}  
else {  
    条件式が満足されないときに実行されること  
}
```

この書式の意味は、条件式が満足されたら、最初の波括弧で囲まれたブロックを実行し、満足されなければ、else以降の波括弧で囲まれたブロックを実行するということです。どちらかのブロックが必ず実行されることになります。

たとえば、次の記述は変数`money`の値が250以上であれば、カレーを食べることが表示されますし、250未満であれば、我慢すべきことが表示されます。

```
int    money;                                // moneyに何かの整数値を代入しておく

if (money >= 250){
    System.out.println( "今日は昼食でカレーを食べよう" );
} else {
    System.out.println( "今日はやっぱり我慢しよう" );
}
```

★条件式の書き方

条件式は、繰返し（while文）のときの継続条件と同じ書式で記述することができます。単純な比較で使えるのは、次の6つの比較演算子がありました。

<code>==</code>	等しい	<code>!=</code>	等しくない
<code><</code>	小さい	<code>></code>	大きい
<code><=</code>	小さいか等しい	<code>>=</code>	大きいか等しい

条件式が満足されるかどうかは、次のような評価によって決定されます。これも繰返しの継続条件と同じです。

条件式の評価が true のとき	→	条件式が満足される
条件式の評価が false のとき	→	条件式が満足されない

たとえば、変数 `x` に値45が代入されているとき、それぞれ次のように評価されます。

<code>x >= 30</code>	→	true に評価される
<code>x / 20 == 3</code>	→	false に評価される

条件分岐と繰返しの違いは、繰返しでは毎回次の繰返しを行なう前に継続条件が成立しているかどうか評価されるのに対して、条件分岐では1回だけしか条件式が評価されないということです。条件分岐は、1回だけしか繰り返されない、繰返しであるとも考えることができます。

★if文の書式 3

書式の3番目のものは、複数の条件を一括して判定していき、条件に該当した処理を実行させるものです。

▼if文の書式 3

```

if ( 条件式1 ){
    条件式1が満足されたときに実行される文
}
else if ( 条件式2 ){
    条件式1が満足されず、かつ条件式2が満足されたときに
    実行される文
}
.....
else {
    すべての条件式が満足されなかったときに実行される文
}

```

この書式では表しきれないことを、少し説明を補足します。

- ・ の示された部分に、**else if (...) { }**の部分は複数書くことができます。
- ・ 最後の**else**以降のブロックは省略しても構いません。

最後のelse以降のブロックが省略された場合は、すべての条件式が満足されなかったときは何も実行されません。書式1と同じように、ただ単にif文をスキップして次に指定した処理に制御が移るだけです。

この書式の意味は、上から順番に条件式を評価し、条件を満足する if文の後で指定されたブロックを選択し、実行するということです。条件式が上から順番に一つずつ評価されることに注意してください。

たとえば、整数型の変数xに何らかの正の整数値が入っているものとしますと、以下の記述は、上から順番に条件式を評価して、対応するメッセージを端末に出力します。

```

if ( x >= 80 ){                                // 80以上
    System.out.println( "成績はA" );
}
else if ( x >= 60 ){                            // 60以上でかつ80未満
    System.out.println( "成績はB" );
}
else if ( x >= 40 ){                            // 40以上でかつ60未満
    System.out.println( "成績はC" );
}
else {                                          // 40未満
    System.out.println( "落第" );
}

```

(注1) 他の構文と同様に、本書では制御構文では必ずブロックを伴って記述する方式を採用します。

■繰返しと条件分岐との組み合わせ

繰返しの中で、条件分岐を使うと「何回目の繰返しでは、この処理を行ない、それ以外のときは、別の処理を行なう」といった処理を記述することができます。繰返しの途中で、処理の内容を変えたいときに、重宝して用いられます。

次のアプリケーションプログラムは、3回目の繰返しの際は、端末に何も表示がされません。

```

public class ThirdHunter {
    public static void main( String [ ] param ){

```

```

int    i = 1;
while (i < 10){
    if ( i != 3){                // 3回目以外は変数の値を表示する
        System.out.print( " Count up: " + i );
    }
    i++;
}
}
}

```

実行してみると、次のように表示されます。

Count up: 1 Count up: 2 Count up: 4 Count up: 5 Count up: 6 Count up: 7 Count up: 8 Count up: 9

次のアプレットプログラムは、5回目までの繰返しまでは、四角形を描画し、それ以降は正円を描画するようにしている記述です。

```

import java.awt.*;
import java.applet.*;

public class FiveRectsAndElevenCircles extends Applet {
    public void paint( Graphics gc ) {
        int    i = 1;
        while (i < 17){                // 17未満の間、繰り返す
            if (i <= 5){                // 5回目までは、
                gc.drawRect( i * 30, 50, 20, 20 );    // 四角形を描画する
            }
            else {                        // 6回目以降は
                gc.drawOval( i * 30, 50, 20, 20 );    // 正円を描画する
            }
            i++;                        // ループ変数を+1する
        }
    }
}

```

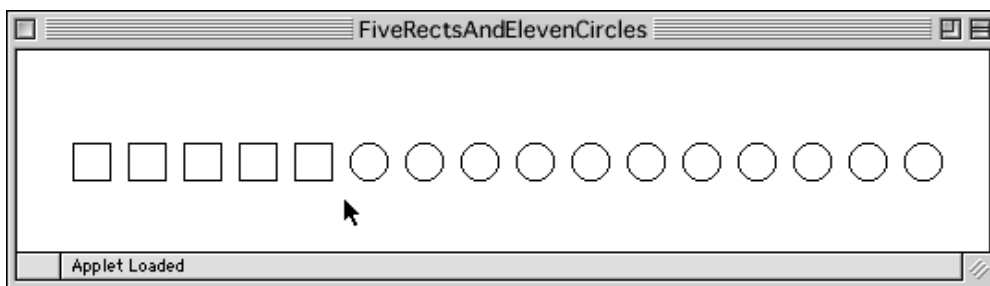


図7-2 実行例

同じように次のpaintメソッドは、繰返しの3回ごとに四角形を描画し、それ以外のときは、正円を描画するようにさせています（クラスの定義などは省略されているので注意してください）。

```

public void paint( Graphics gc ) {
    int    i = 1;
    while (i < 20){                // 20未満の間、繰り返す

```

```

        if ( i % 3 == 0 ) {                                // 3で割り切れるとき
            gc.drawRect( i * 10, 50, 5, 5 );              // 四角形を描画する
        }
        else {                                             // 3で割り切れなかったとき
            gc.drawOval( i * 10, 50, 5, 5 );              // 正円を描画する
        }
        i ++;                                             // ループ変数を+1する
    }
}

```

特にこの例については、変数の値の変遷と条件分岐を注目してみましょう。剰余の式をうまく活用しています。

iの値	条件式	条件式の評価	行なう呼出し
1	1 % 3 == 0	false	drawOval
2	2 % 3 == 0	false	drawOval
3	3 % 3 == 0	true	drawRect
4	4 % 3 == 0	false	drawOval
5	5 % 3 == 0	false	drawOval
6	6 % 3 == 0	true	drawRect
7	7 % 3 == 0	false	drawOval
:	:		

次の繰返しの記述は、ちょっとわかりにくいかも知れません。どのような値が端末にされるのか、何回繰返しが行なわれるのか考えてみてください。

```

int count = 1;
while (count <= 10) {
    if (count == 4) { count += 3; }
    System.out.println( "Loop variable is " + count + ".");
    count ++;
}

```

7.3. 良くある間違い例

if文やwhile文などの制御構文を持つプログラムを記述していると、いろいろな書き間違いをすることがあります。ここでは、典型的なものを集めてみました。

★例1：以下のような場合は、コンパイラがエラーを報告してきます。

```

if ( money >= 100 ) {
    .....
}
else ( money < 100 ) {    // いちいち反対の条件を書かなくても良いのです
    .....
}

```

★例2：if 文やwhile文の条件式の直後にセミコロンを書いてはいけません。コンパイラは、この記述の場合は空文があり、別の新たなブロックが続いているとして解釈してエラーを出してきません。

```

if ( money == 30 ); {      // エラーが報告されませんが、うまくいきません
    ....
}

if ( money == 30 ) {
    ....
} else ; {                  // elseの後もエラーが報告されませんが、うまくいきません
    .....
}

while ( money > 30 ); { // while文でも同様なことが起こります
    ....
}

```

★例3：条件式の中に、代入文を書いてはなりません。等しいことを比較する演算子は==です。

```

if ( money = 300 ) {      // これもコンパイル時にエラーにされます
    .....
}

while ( money = 300 ) { // while文でも同様なことが起こります
    .....
}

```

★例4：if文の書式3では、else ifのelseを書き忘れてしまうと、2つのif文になってしまいます。

```

if ( money > 10 ) { .... }
if ( money > 5 ) { ..... }           // 別のif文になってしまいます
else { ..... }

```

★例5：if文の書式3では、前提条件は書かなくてもいいのです（御丁寧に書く人がいる、そう、あなたです！）。書式3では一番上のifから下のelse ifに向かって順番に条件式が評価されていくことを思い出してください。下のelse ifの中では、もう上のifで条件式が評価されて、「満足されなくて」下に制御が移ってきたのですから、その「満足されなかった」条件をわざわざ指定する必要はないのです。

```

if ( x >= 80 ) {
    System.out.println( "成績はA" );
}
else if ( x >= 60, x < 80 ) {          // こんな記述は間違い、そもそもx < 80は書く必要ありません
    System.out.println( "成績はB" );
}
else if ( x >= 40, x < 60 ) {          // カンマで区切って複数の条件式を書いても機能しません
    System.out.println( "成績はC" );
}
else ( x < 40 ) {                     // 例1と同じ間違い、それでも書きたがる人はいます
    System.out.println( "落第" );
}

```

■まとめ

★変数の初期値代入

変数の宣言を行なうときに、最初の値を代入することができる

★条件分岐

if 文の 3 つの書式

書式 1 : ある条件のときに処理を行なわせる

書式 2 : その条件が満たされないときは別の処理を行なわせる

書式 3 : 複数の条件があり、最初に満たした条件の処理を行なわせる

★繰返しと条件分岐の組合わせ

繰返しのある回数だけ別の処理を行なわせたいときに、繰返しの中に条件分岐の制御構文を記述する

★if文、while文の書き間違い

自分は大丈夫と思って侮っていると、痛い目にあいますので、気をつけましょう

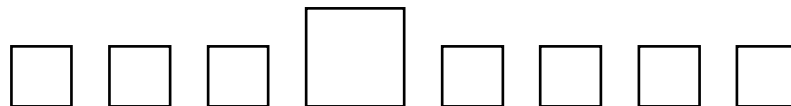
■課題

7-1.

例題として示した、3 回に 1 回は四角形で、残りの 2 回は正円を表示するアプレットを完成させてみなさい。
クラス名は、Circulateを推奨します。

7-2.

次の図のように 8 個中の 1 個を大きな四角形を書くようなアプレットのプログラムを作ってみなさい。大きさは、適当に自分で決めて構いません。色を灰色に設定して、draw3DRectを用いてみなさい。図に示したように四角形は計 8 個表示して、底辺を揃えなさい。プログラムでは while文とif文を用いなさい。クラス名は、FourthRectとしなさい（別の名前でも構いません）。



問題の図

7-3.

4 回に 1 回の割合で正四角形を、3 回に 1 回の割合で正円を、それ以外は三角形を描画するアプレットを作成しなさい。1 2 回目は正四角形の方を描画しましょう。こんな感じでどうぞ、、、クラス名は Rhythmicなどで。



問題の図