

Chapter 6. 繰返し

6.1. while文による繰返し

■同じことを何回もしたい！

前章までのプログラムでは、変数の値を変えて何かを描画するということを何行も同じように記述してきました。描画したい回数分だけ、同じ内容を記述するのではプログラムが無闇に非常に大きくなってしまいます。処理する内容が一定になっている場合、それをただ単に繰り返すように記述したいものです。そのためには、上から下へとただ単に順番に実行されるだけのプログラムの実行の流れを制御する必要があります。

プログラミング言語では実行の流れを制御する文のことを制御構文（Control Flow Statement）あるいは単に制御文（Control Statement）や構文（Structural Statement）と呼びます。

★繰返し文の種類

繰返しには3つの制御構文が使えます。最初は、簡単な while文を使って繰返しを記述してみましょう。2つ目のdo while文は実際のプログラムではほとんど使われませんので、本書では省略します。for文は非常に強力で、C / C++ / Java系のプログラミング言語では頻繁に用いられます。while文には、for文ほどの機能はありませんが、古くから用いられている基本的な繰返しの機能を持っています。

▼Javaで用意されている繰返し用の構文

- while文 単純な繰返し（継続条件を先に判定）
- do while文 単純な繰返し（継続条件を後に判定）
- for 文 複雑な繰返し

C / C++ / Java系以外の言語では、これ以外にも、ある回数分だけ繰り返すこと直接記述できるような構文を持っている場合もあります。しかしJavaでは、while文やfor文と変数を用いて繰返しの回数を制御するように記述します。

■while文を使った記述の仕方

while文は、ある条件が満たされている間、処理を繰り返すために用いられます。

▼while 文の書式

```
while ( 継続条件 ) {  
    繰り返したいこと  
}
```

「繰り返したいこと」の中には、文を複数書くことができます。この部分を囲んでいる括弧（ { } ）のこともブロック（Block）と呼ばれています（注1）。それでは、while文の意味をもう少し詳しく、例をみながら考えていきましょう。意味はセマンティクス（Semantics）と呼ばれることがあります。

★while 文のセマンティクスと使い方

「継続条件」が満足されている間、「繰り返したいこと」に書かれている文を繰り返し実行するのが while文の制御の仕方です。「継続条件」は、繰返しが行なわれるたびに、その冒頭で毎回チェックされます。これだ

けのセマンティクスなので、繰返しはいかようにも記述できるのですが、もし3回だけ繰り返したいのであれば、次のように変数を伴って、変数の値が変化するように記述してやるのがよいでしょう。

```
int    count;                // 回数を制御するための変数
count = 1;                  // まずは1回目
while ( count <= 3 ) {      // 3回までやって
    繰り返したいこと
    count = count + 1;      // やった回数を数えるために1回繰り上げる
}
```

このようにすると、繰り返したいことを3回だけ実行してくれます。例えば、「繰り返したいこと」の部分に、`System.out.println( "Hello" );`と書きますと、Helloを3回表示してくれます。変数`count`の値は、1→2→3→4と変化していき、4になった段階で継続条件を満たさなくなり、繰返しが終了します。

実際に、次のアプリケーションプログラムを実行してみても様子を見てみましょう。次のプログラムでは端末画面にどのような表示が現れるでしょうか？

```
public class RepeatTester {
    public static void main( String [] a ){
        int    count ;
        count = 1;
        while ( count <= 7 ){
            System.out.println( "いちめんのなのはな" );
            count = count + 1;
        }
        System.out.println( "かすかなるむぎぶえ" );
        System.out.println( "いちめんのなのはな" );
        System.out.println( "作：山村暮鳥" );
    }
}
```

■変数の値を使う

これまでの繰返しの例では、変数の値を繰返しの回数を数えるために用いました。この変数の値を「繰り返したいこと」の中で参照して利用することもできます。たとえば、単純に変数の値を表示したければ、次のようになります。

```
int    number ;
number = 1;
while ( number <= 10 ){
    System.out.println( number );        // 変数の値を表示する
    number = number + 1;                // 表示した後に、1つ増やす
}
```

このようにすれば、変数の値が1から10まで順番に表示されます。ところで、上の例で繰り返す内容の部分を逆にして、1つ増やした後に、変数の値を表示させるようにしてみましょう。

```
int    number ;
number = 1;
while ( number <= 10 ){
```

```

        number = number + 1;                // 1つ増やす
        System.out.println( number );      // 変数の値を表示する
    }

```

こうした場合、値の表示は2から11までとなります。なぜだか考えてみてください。もう少し、実用的な例を考えてみましょう。フィートとメートルの換算を表示してみましょう。下の例を実行してみてください。

```

int    number ;
number = 1;
while ( number <= 10 ) {
    System.out.println( number + " feetは、" + number * 0.3048 + " meterです。" );
    number = number + 1;
}

```

変数を使うのですから、別に1つずつ値を更新していなくても構いません。次の例では、変数 *y* を使って、値を20ずつ増やしていきながら繰返しを記述しています。しかも、変数の値を *y* 座標の値としても利用しています。

```

int    y;
y = 20;
while ( y < 100 ) {
    gc.drawString( "Hello", 100, y );    // yをパラメータの値としても用いている
    y = y + 20;
}

```

この記述の意味は、*y*の値が100よりも小さい間、*y*の値を20ずつ増やしながら、Helloという文字列を書いていくという処理になっています。この記述を用いて、繰返しの様子をもう少し詳しくみていきましょう。

★変数の値の変化

繰返しでは、注目している変数が保持する値の推移を理解することが一番重要なのです。この変数は、ループ変数（Loop variable）と呼ばれることがあります。この変数は継続条件で毎回かならずチェックされます。

yの値	継続条件	繰返しの中で行なわれること
20	20 < 100	
↓		gc.drawString("Hello", 100, 20); y = y + 20;
40	40 < 100	
↓		gc.drawString("Hello", 100, 40); y = y + 20;
60	60 < 100	
↓		gc.drawString("Hello", 100, 60); y = y + 20;
80	80 < 100	
↓		gc.drawString("Hello", 100, 80); y = y + 20;
100	100 < 100	

この例では、*y*の値が20ずつ増えていきながら、最終的に値が100になったところで、100は100よりも小さくないので、継続条件が満足されなくなってしまい、終了するという形になります。繰返しは4回行なわれました。

■条件式について

★継続条件の書き方

継続条件は、条件式（Conditional expression）を使って書くことができます。単純な条件式としては下のよう
な比較（comparison）をする条件を記述することができます。

▼比較条件の書式

式 比較演算子 式

比較演算子（comparison operator）としては次のようなものがあります。

▼比較演算子の種類

演算子	意味	例
<	左辺の値が小さい	$x < y$
<=	小さいか等しい	$100 \leq y / 4$
>	左辺の値が大きい	$x / 20 > 10$
>=	大きい等しい	$y \geq 100$
==	等しい	$y / 2 == x * 4$
!=	等しくない	$x != 100 - z * 10$

特に==は、代入文の=と異なり、等号を2つ記述する必要がありますので、注意してください。不等号と等号
が組み合わさった演算子は、「不等号を先に書く」と覚えておくと間違えないで済みます。=<や=>といった形
で記述しますと、コンパイラに叱られます。

★条件式と論理値

継続条件で使われる条件式で、条件が成立することを示すのは、論理値（Boolean Value）によって表現されま
す。論理値には、次の2つの値しかありません。それぞれが、成立するかどうか表現しています。

▼論理値の2つの値

true	条件が成立する（真である）
false	条件が成立しない（偽である）

条件式は評価されて、最終的に論理値になります。

例：	$10 < 20$	→	成立する	→	true
	$10 != 10$	→	成立しない	→	false

★1つつ減らしていく例

while文は、継続条件を毎回の繰返しの際に評価し、継続条件が trueである間、繰返しを続けるのです。例え
ば、次の繰返しの記述は、変数positiveの値が正の数である場合は、 $positive > 0$ と記述された条件式がtrueにな
ります。最終的に、positiveの値が0になったときは、条件式がfalseになり、繰返しを終了します。

```

int    positive;
positive = 10;
while ( positive > 0 ) {
    System.out.println( "Count Down :" + positive );
    positive -= 1;
}

```

端末には、10から1までの数が表示されます。ところで、先ほどの例と同じように繰返しのブロックの中にあるSystem.out.printlnメソッドの呼出しとpositive-=1; の文の順番を逆にした場合、端末にはどのような表示がされるのでしょうか？考えてみてください。

★while文はどういうときに使うか？

同じようなことを延々と記述しなければならないときに while文を使います。まずは、繰返して記述できるように、変数を使って式の記述を調整します。

```

wow = 10;
gc.drawRect( wow, wow, 120-wow*2, 120-wow*2 );
wow += 20;
gc.drawRect( wow, wow, 120-wow*2, 120-wow*2 );
wow += 20;
:

```



```

wow = 10;
while ( wow <= 50 ) {
    gc.drawRect( wow, wow, 120-wow*2, 120-wow*2 );
    wow += 20;
}

```

うまく式が記述できたら、その部分を繰返しのブロックの中にまとめます。

★繰返しが終わったら？

継続条件が満足せず、その評価値がfalseになって繰返しが終わったら、どうなるのでしょうか？ただ単に次の文（繰返しのブロックの後に記述されている文）をそのまま継続して実行するだけです。

★繰返しを作るときのコツ

まず大事なのは、繰返しの中でループ変数を変化させて、継続条件がいずれはfalseになるように仕向けなければいけません。ループ変数の値は、足したり引いたりするだけではなく、掛け算などを使っても変化させることもできます。次の例は、掛け算を使っています。そして、いずれは継続条件を満たさなくなります。

```

x = 2;
while ( x < 100 ) {
    x = x * 2;
}
// いずれは100を越えるように仕向ける
// 2 → 4 → 8 → 16 → 32 → 64 → 128と変化していく

```

継続条件が最終的に**false**にならない場合は、繰返しは止まらなくなります。下の例が、その見本です。

```
x = 3;
while ( x > 0 ) {
    y = y + x;                // 別の変数を変化させてもしょうがありません
}
```

次の例は、いつまでも継続条件を満たし続ける例です。繰返しは止まりません。

```
x = 4;
while ( x > 0 ) {            // 継続条件を満たしているのですが
    x = x + 1;               // 数はどんどん大きくなっていきます。
}
```

継続条件は、きつめにつくっておくとよいでしょう。例えば、次の記述では継続条件を $x \neq 0$ としてもいいのですが、熟練したプログラマは何かプログラムが後で拡張されることに備えて、継続条件を少しきつめに記述します。

```
x = 10;
while ( x > 0 ) {
    x -= 1;                  // 次の次の例と比べてみてください
}
```

継続条件と終了条件を間違える人は非常に多いのです！これは、注意してください。こういう場合は、だいたいは、1回も繰返しが実行されないケースとなっています。

```
x = 2;
while ( x > 100 ) {          // xが100を越えたら終了したいつもり
    x = x * 2;               // ところがのっけから何もせずに終了してしまう
}                             // なぜなのだー？
```

反対にいつまでも継続してしまうような条件を書く人もいます。次の例をみてください。なぜ、延々と繰り返されてしまうのでしょうか？

```
x = 9;
while ( x != 0 ) {          // 0になったらやめるつもり
    x -= 2;                  // そのうち0になるでしょう。
}                             // 終わらないぞ！なぜなのだー？
```

プログラムが止まらなくなった場合は、それぞれのオペレーティングシステムやランタイムインタプリタに用意されている「強制的にアプリケーションやアプレットを終了させる」手段がありますので、それを利用して止めてください。

注1 ブロックを使わずに繰返しを記述することも可能なのですが、本書ではわかりやすくするため、および記述の間違いをなくすために、制御構文では必ずブロックを伴う形で記述します。たとえば、ブロックの中の文が一つの文だけの場合でも、ブロックとして記述します。

■while文の多重化（ネスト）

while文で「繰り返したいこと」の中に、さらにwhile文を書くことができます。このような「繰返しの中の繰返し」のことを、繰返しの多重化（Nesting）と呼んでいます。たとえば、次の例は簡単な繰返しの多重化の

例です。

```
int    outer;
outer = 1;
while ( outer <= 4 ) {
    int    inner;
    inner = 1;
    while ( inner <= 5 ) {
        System.out.print( "Thank you! " );
        inner += 1;                // 内側のループ変数の更新
    }
    outer += 1;                    // 外側のループ変数の更新
    System.out.println( "" );      // 改行させている
}
```

変数outerが外側の繰返しのループ変数になっています。変数innerが内側の繰返しのループ変数になっています。外側の繰返しは、4回行なわれます。内側の繰返しは、5回繰り返されます。ですから、端末上に Thank you!の文字列は、4 × 5回で、計20回表示されることになります。外側のループ変数の更新は、内側の繰返しのブロックが終わった後に、記述されていることに注意してください。この例は次のような表示をします。

```
Thank you! Thank you! Thank you! Thank you! Thank you!
Thank you! Thank you! Thank you! Thank you! Thank you!
Thank you! Thank you! Thank you! Thank you! Thank you!
Thank you! Thank you! Thank you! Thank you! Thank you!
```

次のアプレットプログラムの例は、同じように多重化された繰返しによって記述されています。この例について、もう少し詳しくみていきましょう。

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

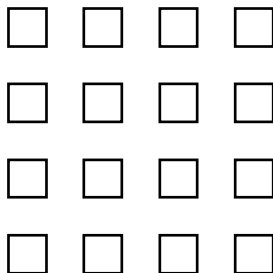
public class LatticeDrawer extends Applet {
    public void paint( Graphics gc ) {
        int    x, y;                // 変数の宣言
        y = 10;
        while ( y <= 70 ) {          // 縦方向の繰返し
            x = 10;                  // 内側のループ変数の初期化
            while ( x <= 70 ) {       // 横方向の繰返し
                gc.drawRect( x, y, 10, 10 ); // 四角形を書く
                x = x + 20;           // 内側のループ変数の更新
            }
            y = y + 20;              // 外側のループ変数の更新
        }
    }
}
```

この例の意味を考えていくのに、まず変数 x と y が繰返しによってどのように変化していくかを探っていきましょう。変数 y の方が外側のwhile文の中で値が変えられているので、変化がゆっくりになっています。変数 x は、内側のwhile文の中で変化させられているために、変化は急速になっています。

いま、(x , y)という記述の仕方、それぞれの変数が、内側のwhile文のdrawRectメソッドを呼び出す時点でどのような値を取っているか追いかけてみましょう。変数 x が頻繁に変化しているのがわかると思います。

```
(10, 10)→( 30, 10)→( 50, 10)→( 70, 10)→  
(10, 30)→( 30, 30)→( 50, 30)→( 70, 30)→  
(10, 50)→( 30, 50)→( 50, 50)→( 70, 50)→  
(10, 70)→( 30, 70)→( 50, 70)→( 70, 70)
```

実際にどう書かれるのでしょうか？変数の値が、それぞれ四角形の開始位置（左上角）の座標になっていますので、こんな感じで描画されます。



6.2. 繰返しと再帰代入文

■代入の更なる省略形

前章で、再帰代入文に関して、 $+=$ や $-=$ といった便利な記述があると説明しました。ところが、変数の持っている値に1を足したり、1を引いたりするのは、更に省略形が用意されている。これらはインクリメント・デクリメント演算子（Increment / Decrement Operator）と呼ばれて重宝されています。それぞれ、 $++$ と $--$ で表現することができます。

```
y = y + 1;      →      y += 1;      →      y++;  
z = z - 1;      →      z -= 1;      →      z--;
```

更に、式中ではこれらの演算子を別の演算子と組み合わせて使うことができます。

```
8 * y++          →      8 * y, y = y + 1  
z = y-- / 10;     →      z = y / 10; y = y - 1;
```

ただし、このようにしますと式を評価しているのか、代入をやっているのか、かなりわかりにくくなります。C言語系でない他の言語のプログラマは、これを副作用（Side Effects）と呼んで、忌み嫌っていました。そこまで嫌う必要はありませんが、あまりに頻繁に使ってしまうと自分でもわからなくなってしまうので、使う際には注意した方がよいでしょう。

インクリメント・デクリメント演算子は、変数の前か後ろのどちらかにつけることができます。前につけると、後ろに付けるのは微妙に副作用が違うので注意が必要です。

▼副作用の違い

前につける	その変数を先に更新してから評価する
後ろにつける	その変数を評価した後更新する

例えば、変数 y が値30を持っていると仮定して、それぞれの場合で次のように変化します。

<code>z = ++y * 10;</code>	<code>→</code>	<code>y = y + 1;</code>	<code>z = y * 10;</code>
<code>z = y++ * 10;</code>	<code>→</code>	<code>z = y * 10;</code>	<code>y = y + 1;</code>

両者ともyの値は31になりますが、zの値は上の方が310、下の方が300となります。

▼インクリメント・デクリメント演算子の副作用のまとめ

++変数名	変数の値を 1 増やす	評価する前に更新
変数名++	変数の値を 1 増やす	評価してから更新
--変数名	変数の値を 1 減らす	評価する前に更新
変数名--	変数の値を 1 減らす	評価してから更新

繰返しで使うと次のような感じになります。わかりやすくするために、括弧をつけてみました。やはり、どこでループ変数の値を更新しているのかわかりにくくなります。充分御理解の上、よろしく御使用ください。

```

int    count;
count = 1;
while ( count < 10 ) {
    System.out.println( "Count is " + (count++) );
}

```

6.3. まとめと課題

■まとめ

★while文の書き方

変数と継続条件を使って、繰返し回数を制御する。

いずれは継続条件が満たされなくなるようにループ変数を変化させる。

★繰返しの多重化

内側の繰返しは頻繁に繰り返され、外側の繰返しは内側の繰返しが終わったら、はじめて繰り返される。

★インクリメント・デクリメント演算子

副作用に注意のこと

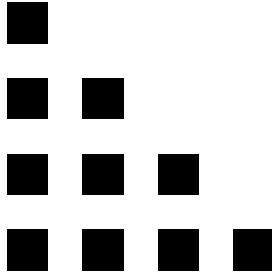
■課題

6-1.

while文を使って、前回の演習の対数グラフを書くアプレット、ピラミッドを書くアプレット、および前回の課題のアプレットの3つのプログラムを書き直してみなさい。

6-2.

問題の格子図を書くようなアプレットのプログラムを作成しなさい。 while文の多重化の書き方を使って記述してみてください。クラス名はStairsにしなさい。

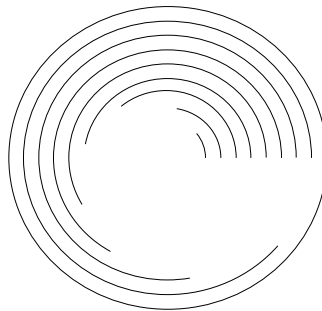


問題の格子図

ヒント：drawRectではなくfillRectを用いなさい。内側のwhile文の継続条件を工夫しなさい。

6-3.

問題の弧形図を書くアプレットのプログラムを作成しなさい。クラス名はArcsにしなさい。



問題の弧形図

ヒント：まずは、drawOvalを使って同心円を書くことから始めてみなさい。それができたら、drawArcを使って、弧を描きましょう。9つの弧があり、一番外側は、円になっていることから、角度差を割り出してみてください。

6-4.

算数を考えてください。 $1 \times 1 = 1$ でした。 $11 \times 11 = 121$ ですね。やや、 $111 \times 111 = 12321$ ですし、 $1111 \times 1111 = 1234321$ です。これは面白そうです。繰返しを使って、この後どうなっていくのか求めてください。クラス名はOnesです。

ヒント：ループ変数として掛ける数を使います。最初は1にしておいて、この変数を10倍して、1を足していきます。毎回、この数とその二乗を表示させるようにしましょう。繰返しは、適当なところで終わるようにしてください。大きい数も表せるように、整数型として、**int**ではなく**long**を使いなさい。