Chapter 8.主要構文

８－１．for文を使った繰返し

８－１－１．for文の書き方

while文の場合、繰返しの中でしか使わないループ変数の宣言やその最初の値を代入する場合など、while文が始まる前に独立して書かなければなりませんでした。これらの不便を解消してくれる強力な繰返しのための構文が存在します。それが以下に述べるfor文です。Java言語のfor文は、内部で変数の宣言もできます。

▼for文の書式

```
for ( 最初に行なうこと ; 継続条件 ; 繰り返すときに行なうこと ) {
        繰り返したい内容
}
```

書式では、for文の４つの部分がすべて記述されいますが、どの部分も、省略することが可能です。ただし、継続条件が省略されたときは、そこに**true**が書いてあるものとみなします。この場合、永遠に繰返すこととなるでしょう。

★for文の意味

それぞれの部分について、もう少し詳しく実行の過程を追ってみましょう。

1. 最初に、繰返しを行なう前に「最初に行なうこと」の式が評価されます。これは、繰返しを１つもしない場合でも実行されます。
2. 次に、「継続条件」を満足しているかチェックする。条件式を評価し、真（true）であれば繰返しを継続しますし、偽（false）であれば繰返しを止めて次の処理に制御を移します。
3. 繰返しを行なうときは、まず「繰り返したい内容」の方を先に実行します。そして、次の繰返しに移る前に「繰り返すときに行なうこと」の部分を評価します。
4. 以降の繰返しは、2.のところから行なわれます。

以上をfor文の各部分をABCDという形で分け、制御の流れを図示しますと、次のようになります。

```
for ( A; B; C ) {
        D;                    A→B→D→C→B→D→C→ ....  B→D→C→B→次へ
}                                                     ↑
                                                B→D→C→が何回か繰り返される
```

８－１－２．初期値設定とループ変数

for文の「最初に行なうこと」の部分には、変数の宣言ができます。変数の宣言をする場合は、初期値を代入する必要があります。この変数を使って、１０回の繰返しをする記述の仕方をみてみましょう。

```
for ( int count=0 ; count < 10; count ++ ) {
        System.out.println( "Count up: " + count );
}
```

上の繰返しは、変数*count*を0から9まで変化させて、計10回の繰返しを行なわせています。毎回の繰返しの中で、変数の値を端末画面に出力しています。なお、for文の中で宣言された変数は、for文の繰返しのブロック内でしか用いることができません。

```
for ( int  x=1 ; x<10 ; x++ ) {
        sum = x + 5;        // このブロックの中では、変数xを用いることができる
}
int y = x + 10;             // エラー、繰返しが終わった後は、変数xを用いることができない
```

８－１－３．それぞれの値を設定する目安

for文のそれぞれの部分に関して、さまざまな記述ができますが、たとえばループ変数の値を繰返しする中で増やしていく場合は、次のように考えるといいでしょう。

```
for ( int  i = 初期値; i < 上限値 ; i += 間隔 ) {
        ……
}
```

このようにすると繰返しをする回数は、（上限 - 初期値）／間隔で求められます。ループ変数と上限値との比較が不等号で行なわれていることに注意してください。たとえば、次の繰返しは、7回繰返しをします。

```
for ( int w = 10;  w < 150;  w += 20 ) {
        System.out.println( "The value of w  is  " + w + "." );
}
```

８－１－４．for文を使った様々な記述例

以下のプログラムの断片は、それぞれfor文を使った例を示しています。初期値や上限値、不等号や等号の使い方、あるいはループ変数をどのように辺かさせているか注意してみてください。

1から10まで値の二乗を表示する

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

50から–50までの値について、10間隔で表示する。

```
for ( int  w = 50;  w >= –50 ; w –= 10 ) {
        System.out.println( "Value of w:  " + w );
}
```

1から100000以下の数について、２のべき乗を表示する。

```
for ( int  power = 1;  power <= 100000 ; power *= 2  ) {
        System.out.println(   "power of 2:  " + power );
}
```

何もしないで永遠に繰り返す（A〜Dまでの部分はいずれも省略可能です）。

```
for ( ; ; ) { }                 // プログラムが止まらなくなるので要注意、forever
```

８－１－５．ループ変数のトレース

1から10までの値の総和を求めるプログラムの断片を記述してみましょう。

```
int        sum = 0;
for ( int i = 1;  i<= 10 ; i++ ) {
        sum = sum + i;
}
System.out.println( "total: " + sum );
```

変数の値がどのように移り変わっていくのか注意してください。**for**文の中で宣言された変数は、繰返しを制御するためのループ変数になっています。特に、ループ変数とループ変数に関係のある変数の変遷を追っていく（トレースする）ことは重要です。上記の総和のプログラムにおいては、変数*sum*の値とループ変数 *i* の値の移り変わりに注目してみましょう。

sum: 0 → 1 → 3 → 6 → 10 → 15 → 21 → ....
i: 1 → 2 → 3 → 4 → 5 → 6 → 7 → ....

２つの代入　*sum* = *sum* + *i*; と *i*++ によって、このように、値が相互に変わっていきます。*sum*は、最初が0でどんどん値が足し込まれています。これは、例えば風呂桶などに、小さな桶を何回も使って水を足していくのに似ています。

図10-1　水を足し込んでいく

８－１－６．止まる繰返しを設計しよう

while文のときと同様に、いずれは停止する繰返しを記述するべきです。継続条件は、適切なものを選択しよう。また、継続条件と停止条件の区別をしましょう。**while**文も**for**文も条件として書くのは、繰返しを継続するときの条件です。

```
for ( int i=1;  i >= 10; i++ ) { .... }      // 一回も実行されない
for ( int  x=10; x>0; x++ ) { .... }         // 止まらない
```

★**for**文で間違いやすい例

for文で、初期設定、継続条件、再設定をカンマ（,）では区切ってはいけません。また、閉じる丸括弧　）の直後にセミコロン（；）を入れると意図していない動作になります。

```
for ( int i=1, i<=10, i ++ ) { .... }        // 正解はセミコロン ;

for ( int i=1; i<10; i++ ) ; { .... }        // ここにセミコロンを書いてもコンパイラはここで
                                             // エラーを出さない（if文、while文でも起こり得る）
```

８－２．while文とfor文との互換性

両方を書き直してみましょう。for文がwhile 文に比べて強力な記述ができることがわかります。

★while文をfor文で書き直してみる

```
while ( B  ) {                          for ( ; B ; ) {
      D              =                        D
}                                       }
```

★for文をwhile文で書き直してみる（注１）

```
for ( A ; B;  C ) {                                  A;
        D                    =                       while ( B  ) {
}                                                            D
                                                             C;
                                                     }
```

★書き直してみた例

第６章の格子枠を書くpaintメソッドをfor文で書き直してみました。かなり短い記述で済んでいます。

```
public void paint( Graphics gc ) {
        for( int y = 10; y <= 70; y += 20 ) {
                for ( int x = 10; x <= 70; x += 20 ) {
                        gc.drawRect( x, y, 10, 10 );
                }
        }
}
```

（注１）変数の有効範囲を考えると、左側のfor文と同じにするためには、右側のwhile文ではその外側にもう一つブロックが必要になります。

８－３．繰返しの途中脱出

繰返しにおいて、繰返しの途中まで処理を行ない、それ以降の処理を行なわずに繰返しを終了させたり、次の繰返しを始めたりしたいことがあります（かも知れません）。そのときに、用意されているのが、それぞれ**break**文と**continue**文です。

８－３－１．break文

break文は、**for**や**while**文などの繰返しから強制的に脱出するという効果を持ちます。**break**文は単体で現れることはなく、**if**文とともに脱出条件を指定して、途中脱出するのに用いられます。

▼**break**文の典型的な使用例：

```
while ( true ) {            //  for ( ; ; ) {   と書いてもよい
        ........
          if  (  終了条件  ) {  break; }
        ........
}
```

たとえば、総和が初めて100を越えるときの総和と数を表示してみましょう。次の繰返しは**while**文を使ってみましたが、**if**文で100を越えたときのループ変数*i*と変数*sum*の値を表示して、繰返しを脱出しています。

```
int sum = 0;
int  i = 0 ;
while ( true ) {
        sum +=  i;
        if (  sum >= 100 ) {  System.out.println( "sum: " + sum + "  where i: " + i ); break; }
        i ++ ;
}
```

break文を好んで用いるプログラマもいます。何故でしょうか？**while**文や**for**文などでは、継続条件を指定する必要があります。それは、脱出条件の反対の指定になっています。それに対して、**if**文と**break**文を用いた場

合は、脱出条件を直接記述することができます。脱出条件の方がわかりやすいと考えているプログラマは、後者の方法を好むのです。次の２つの繰返しは、同じ内容を表しているのですが、どちらの方がわかりやすいでしょうか？わかりやすい書き方を用いてください。

```
for ( int power = 1; power <= 100000 ; power *= 2 ) {          // 100000以下ならば続ける
        System.out.println( "power of 2: " + power );
}

for ( int power = 1; ; power *= 2 ) {
        System.out.println( "power of 2: " + power );
        if ( power > 100000 ) { break; }                        // 100000より大きくなったら脱出
}
```

break文は、繰返しからネストしているときは、一番内側繰返し（ブロック）から脱出するという効果を持ちます。もしブロックを何重にもネストしているときに、その外側に脱出したいときは、ラベルを使った脱出法があります。たとえば、次の例ではラベルを予め指定してあります。ラベルは、コロン（：）を伴って、記述しておきます。break文の後に、そのラベル名を指定したら、そのラベル書かれているレベル（ブロック）まで脱出してくれます。

```
escapeLevel:                                        // ラベルで指定します
    for ( int x = 1 ; x < 10 ; x ++ ) {
        for ( int y = 1 ; y < 10 ; y ++ ) {
            System.out.print( x + " multiplies with " + y + " is " + x * y );
            if ( x * y > 50 ) {                     // 掛け合わせた結果が50より大きければ
                break escapeLevel;                  // ラベルで指定したレベルに脱出
            }
        }
    }
// 脱出先はここになります
```

８－３－２．continue文

continue文は、それ以降の処理を行なわずに次の繰返しを行なわせるために用いられる制御文です。

▼continue文の典型的な使用例

```
while ( 継続条件 ) {
        A
        if ( 条件式 ) { continue; }
        B
}
```

上の使用例では、**if**文の中の条件式が満足されるとBの部分を実行せずに、次の繰返しに進むことになります。条件式が満足されない場合は、Aの部分の後に、Bの部分も続けて実行されます。

たとえば、東京エリアのTVチャンネルに関して、表示することを考えてみます。2チャンネルはビデオとして、それ以外の 1, 3, 4, 6, 8, 10, 12チャンネルが通常のTVチャンネルである仮定します。以下は、各チャンネルに何が割り当てられているのか表示するようなプログラムの断片です。

```
for ( int i = 1 ; i <= 12 ; i++ ) {
        System.out.print( "Channel: " + i );
        if ( i == 2 ) {     System.out.println( "  Video Channel" ); continue ; }
        if ( 12 % i != 0 && i % 2 != 0 ) { System.out.println( "" ); continue ; }
        System.out.println( "  TV Channel in Tokyo Area" );
}
```

上の例では、12をその数で割ったときに割り切れなくて、かつ奇数の場合には、次の繰返しに行くように記述されています。すなわち変数 iの値が、5, 7, 9,11のときは最後のprintlnメソッドの呼出しはスキップされます。

８－３－３．**return**文

paintやinitメソッド、あるいは既出のactionPerformedなどのメソッドにおいて、メソッドの実行を途中で終わらせたいときは、**return**文を用いることができます。**return**文は、メソッドの定義の章でもう一度説明しますが、ここではメソッドを強制終了させるものとだけ覚えておいてください。

たとえば、次の例はdrawLineで線を描いているのですが、始点と終点の座標値が同じになったら、paintメソッドを終了させるようにしています。

```
public void paint( Graphics gc ) {
        int x1 = 10, y1= 100, x2= 300, y2 = 200;
        while ( true ) {
                if ( x1 == x2 && y1 == y2 ) { return ; }
                gc.drawLine( x1, y1, x2, y2 );
                x1 += 2; x2 –= 2;
                y1 += 1; y2 –= 1;
        }
}
```

８－４．その他の主要な構文

８－４－１．**do～while**文

do～while文は繰返しを行なう制御構文の一つですが、あまり使う機会は多くないようです。どちらかと言えば、この文を使うよりも**break**文で脱出させる方がわかりやすいでしょう。**do～while**文では、継続条件を最後にチェックします。ですから、繰返しをさせたいことは、必ず１回は実行されてしまいます。**do～while**文は、次のような書式で書きます。特に、継続条件を書き終わった後、セミコロン（）を書かなければいけないので、注意してください。このセミコロン（通常の**while**文や**for**文の後にセミコロンを書くと誤動作をする）のこともあって、あまり使われないようです。

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

次のプログラムの断片は、計算結果が10000000以内であれば、繰返す

```
int x = 100;
do {
        System.out.println( x* x );
        x = x*x;
} while ( x < 10000000 ) ;          // ここにセミコロンをつけることに注意
```

注意しなければならないのは、最後の**while**文の条件式で用いられる変数はブロックの外側で定義しておかなければならないことです。変数*second*は、ブロックの手前で宣言されています。また、*second*に新たな秒が代入されているのも、次の繰返しにいく直前であるということに注意してください。

８－４－２．**switch**文

定数と比較する条件式は、**switch**文を使うとわかりやすい記述できる場合があります。プログラマによっては、**if** ～ **else if**文を使うよりも、こちらの方を使う場合があります。

▼switch文の書式

```
switch ( 式 ) {
case  定数1:
        式の評価結果が定数1のときに実行すること
        break;

case  定数2:
        式の評価結果が定数2のときに実行すること
        break;
                        :
default:
        それ以外のときに実行すること
        break;
}
```

これは、以下の**if～else if** 文と等しくなっています。

```
if (  式  ==  定数1 ) {
        式の評価結果が定数1のときに実行すること }
else if (  式  ==  定数2 ) {
        式の評価結果が定数2のときに実行すること }
                        :
else {
        それ以外のときに実行すること                }
```

たとえば、現在の時間が整数型の変数*hour*に求まっているとしましょう。このときに、８時と１２時と、午後７時（１９時）ならば、食事に行くような条件分岐を書いてみましょう。

```
if (  hour == 8 )  {          System.out.println( "朝食を食べましょう" );  }
else  if (  hour == 12 )  {   System.out.println( "ランチを食べましょう" );  }
else  if (  hour == 15 )  {   System.out.println( "午後の紅茶を飲みましょう" );  }
else  if (  hour == 19 )  {   System.out.println( "ディナーを食べましょう" );  }
else  {                       System.out.println( "食べ過ぎにご用心" );  }
```

これをswitch文を使って書き直してみますと、次のようになります。それぞれの場合の最後に、break文を入れなければならないことに注意しましょう。break文がないと、次のcaseまで実行してしまいます。

```
switch (  hour ) {
case  8:          System.out.println( "朝食を食べましょう" );  break;
case 12:  System.out.println( "ランチを食べましょう" );  break;
case 15 :  System.out.println( "午後の紅茶を飲みましょう" );  break;
case 19 :         System.out.println( "ディナーを食べましょう" );  break;
default :         System.out.println( "食べ過ぎにご用心" );  break;
}
```

８－４－３．if 式

条件文を使いますと、いろいろな場合分けができるのですが、代入文などではいちいち条件分岐を使うのが面倒な場合があります。たとえば、次のプログラムの断片を考えてみましょう。

```
if  ( month < 9 )  {  season = 1; } else { season = 2; }
```

これをもっと簡単に記述するための式が用意されています。これを一般にif式と呼んでいます。

▼**if** 式の書式

```
( 条件式 ) ? 条件式が真のときの値 : 条件式が偽のときの値
```

このif式を使いますと、さきほどの条件分岐は一つの代入文で記述することができます。

```
season = ( month < 9 ) ? 1 : 2;
```

このように、**if**文がなくなり、簡潔に記述できるので、これを愛用するプログラマはかなり多いのですが、あまり多用しすぎると、かえってプログラムを読みにくくします。使うときには読みにくくなるかどうか、いつも注意しながら使いましょう。なお、**if**式もネストさせることができます。例えば、季節毎に変数の値を１～４まで変えたいときには、if式をネストさせて、次のように記述します。これは、**if～else** if文に相当します。

```
season = ( month <= 3 ) ? 1 :
                ( month <= 6 ) ? 2 :
                        ( month <= 9 ) ? 3 : 4;
```

８－４－４．繰返しのときのローカル変数の有効領域とブロック

ブロックは、波括弧（brace）すなわち｛と｝で囲まれた領域のことを指しています。プログラムの実行時において、**while**文、**for**文や、**if**文のブロックの中で宣言した変数というのは、それ以降使えなくなりますので、注意しなければいけません。これらの制御文のブロックの中で宣言した変数は、そのブロックの中でしか有効ではないのです。このことについて、幾つかの例で考えてみましょう。...の部分は、何がしかの記述がされていると考えてください。

```
while ( ..... ) {
        int  x = 10;
        .....
}
```

この記述では、変数xは繰返しのたびに毎回宣言され、初期値が10として代入されます。**while**文が終了した段階で、このブロックで記述された変数x は消滅します。次の例は、ブロックだけを単体で記述したものです。内側のブロックでは、変数*x*も*y*も有効ですが、外側のブロックでは変数xのみが有効です。

```
{       int  x = 10;
        .....                           // ここでは変数xのみが有効
        {       int     y = 20;
                .....                   // ここでは変数xとyの両方が有効
        }
        .....                           // ここでは変数xのみが有効
}
```

なお、次のような記述はどうなるのでしょうか？

```
for ( int x = 10; x < 20; x ++ ) {
        while  ( y  < 10 )  {
                int  x = 20;
                .....
        }
}
```

for文で宣言した変数xには、最初の値が10になっています。しかし、内側の**while**文の中のブロックでは、変数*x*は毎回20として初期化されています。この場合、一番内側のブロックの同じ名前の変数が外側のブロックの変数を見えなくしています。これを変数の隠蔽（Shadowing）と呼んでいます。Javaでは、ローカル変数どうしの隠蔽は、コンパイラによってエラーにしています。ただし、後で出てくるインスタンス変数はローカル変数で隠蔽できますから注意してください。

８－５．論理値を保持する変数と繰返し

while文でもfor文でも、あるいは、while文の中に、if文があり、そこでbreakする繰返しでも可能なのですが、論理値を変数で保持しておいて、その値を使って繰返しを制御するということもできます。また、代入についても、論理式を記述しておいて、その評価結果を変数に代入させるということでもできます。

８－５－１．論理値を保持する変数の宣言と代入

次の書式のように、変数を宣言するときに、変数の型名の部分に、booleanを記述します。

▼論理値を保持する変数の宣言の書式

```
boolean　　変数名;
```

例：

```
boolean　flag;
```

宣言した後は、条件式を代入の＝演算子の右辺に記述することができます。また、宣言時に初期値代入することも可能です。

▼論理値を保持する変数への代入

```
変数名 ＝ 論理式;
```

例：

```
flag = ( x > 100 ) && ( x < 200 );
boolean cheker = false;
```

８－５－２．論理値を持つ変数を使った繰返しの制御

たとえば、次のプログラムの断片のように論理値を持つ変数を使って、繰返しを制御することもできます。この場合は、while文で行なっていますが、継続条件の部分を変数で記述しています。継続条件に使う場合は、最初にtrueの値を代入しておき、繰返しの中で、falseに評価されるように記述します。

```
boolean processing = true;
int  w = 10,  total = 1;
while ( processing ) {
        total = total * w- - ;                    // total に計算結果を保持
        processing = w > 0 && total < 10000000)   // processing に条件式の結果を保持
}
System.out.println( total );
```

同じように、終了条件にも用いることができます。上の断片を、終了条件という形で書き直してみました。また、それぞれの条件に条件式の変数を割り当ててみました。

```
boolean  satisfied, overflowed;
int  w = 10, total = 1;
while ( true ) {
        total *= w - - ;
        satisfied = w <= 0;
        overflowed = total >= 10000000;
        if ( satisfied || overflowed ) { break; }
}
System.out.println( total );
```

８－６．課題

課題8-1.　アプリケーションのプログラムを作成し、１つのfor文を使って、以下の総和を求めなさい。

１．１から100までのすべての数の総和

２．1から100までの偶数の総和

３．1から100までの、３の倍数の総和

これらの総和が、等差数列（Arithmetical Progression）の総和の公式と等しくなっていることを確かめなさい。等差数列の公式は、初項をaとし、交差をdとすると、第n項までの総和は、以下のようになる。この公式もプログラム中で記述し、for文を使って求めた総和と等しくなっているかどうか表示しなさい。クラス名は、APSumにて。

総和 = n *(2*a+(n-1)*d)/ 2

課題8-2.　19世紀のイギリスにおいて、チャールズ・バベッジは、歯車を使って階差機関を試作しました。方程式の結果を記した数列は、階差を取っていくと、次数の高い方程式も、すべて足し算で計算できます。たとえば、ｘの２乗について考えてみましょう。１から順に初めて、計算結果を求めていき、その階差を取ると次のようになります。

x：	1	2	3	4	5	6	7	8	9
x^2：	1	4	9	16	25	36	49	64	81
第１階差：		3	5	7	9	11	13	15	17
第２階差：			2	2	2	2	2	2	2

これを利用し、任意の$px^2 + qx + r$という二次方程式について、掛け算を使わないで、計算するアプリケーション・プログラムを作りなさい。整数で行ないなさい。ｐとｑ、およびｒには、適当な整数の定数を代入しておきます。ｘを１から１０ぐらいまで変化させ、そのときに、その二次方程式に適用したときの計算結果を出すようにfor文で作ります。もちろん、第１階差の初項と第２階差を求めるとき以外は、掛け算を使わないで、足し算だけで求めていきます。クラス名は、Differentialにします。なお、チャールズ・バベッジは、階差機関を発展させていろいろな方程式を求めることができる解析機関を歯車で試作しようとしました。これは、現在のコンピュータの元祖の一つになっています。

ヒント：$px^2 + qx + r$の第１階差は、$p(2x + 1)+q$になります。第２階差は、$2p$になります。