Chapter 7. 条件分岐

７－１．条件分岐

７－１－１．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。

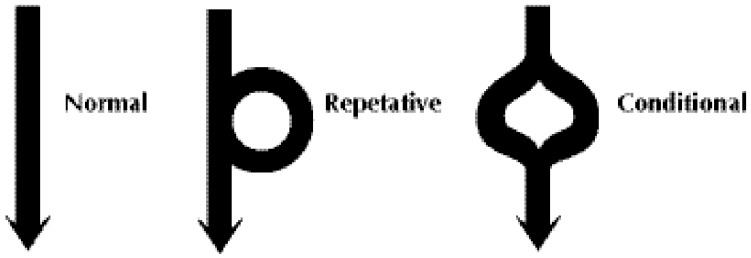


図7-1　制御構文による制御の流れ

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。

７－１－２．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" );
}
```

（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。

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" );
}
```

７－１－３．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。

▼条件式で使える６つの比較演算子

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

x >= 30	→	**true**に評価される
x / 20 == 3	→	**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。

７－１－３．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" );
}
```

７－２．繰返しと条件分岐との組合わせ

７－２－１．単純な組合わせの例

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

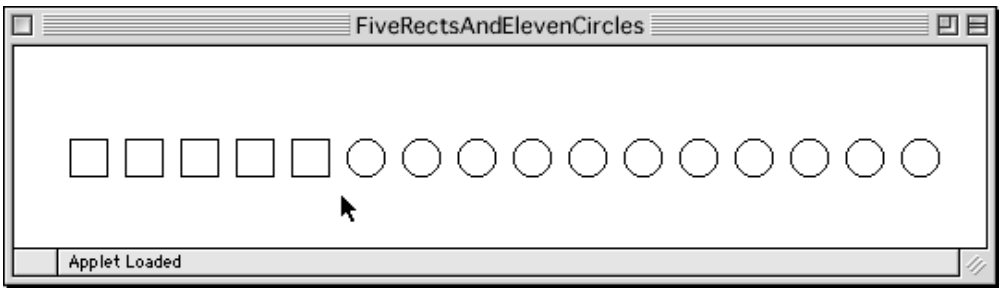


図7-2　実行例

同じように次のpaintメソッドは、繰り返しの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 g ) {
        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
                        g.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
                        g.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 ++;
}
```

７－２－２．整数剰余と整数除算の使用

繰返しと整数剰余と整数除算の組み合わせは、よく用いられます。これらの式を用いると、if文を用いなくとも、かなり都合の良いことができます。一般的に、整数剰余と整数除算は、繰返しの中では、次のように用いることができます。もちろん、繰返しと組み合わせなくとも、このような使い方はできます。

整数剰余	周期的に何かをしたいときに用いる
整数除算	段階的に何かをしたいときに用いる

たとえば、この一番良い例として、アプレットで、カレンダーを描くことを考えてみます。カレンダーの１週間の周期は、７日ですから、ここでは７がキーナンバーとなります。たとえば、変数dayが、ある月の中での日付を持っているとすると、それぞれ次のように使います。

day % 7	7日を周期とさせることができる
day / 7	7日を１つの段階とさせることができる

繰返しを使って、この式を使って表示させてみましょう。下記の記述の中で、day % 7のところを、day / 7に替えても実行してみてください。

```
int     day = 1;
while ( day <= 31 ) {
        System.out.print( (day % 7 ) + "" );
        day++;
}
```

結果は、以下のようになる筈です。27まで追ってみました。

day:	1 2 3 4 5 6	7 8 9 10 11 12 13	14 15 16 17 18 19 20	21 22 23 24 25 26 27
day % 7:	1 2 3 4 5 6	0 1 2 3 4 5 6	0 1 2 3 4 5 6	0 1 2 3 4 5 6
day / 7:	0 0 0 0 0 0	1 1 1 1 1 1 1	2 2 2 2 2 2 2	3 3 3 3 3 3 3

以上のような表をみますと、それぞれ、周期的・段階的になっているのがわかると思います。さらに、この段階的になる整数除算を使って次のように式を書いてみると、どうなるか考えてください。

```
day  / 7   * 7
```

通常では、７で割って７で掛けるのですから、元に戻ると思いますが、段階的になります。0→7→14→21と7を周期として、段階的になっていきます。これもプログラムでは良く使われる手法です。そこで、この手法を利用して実際にカレンダーを表示するアプレットのpaintメソッドの部分だけを記述してみましょう。

```
public void  paint( Graphics  g ) {
        int     day = 1;
        while ( day <= 31 ) {
                if  (  day % 7 == 0 ) { g.setColor( Color.red ); }
                else if ( day % 7 == 6 ) { g.setColor( Color.blue ); }
                else { g.setColor( Color.black ); }
                g.drawString( "" + day,  ( day % 7) *20 +30,  (day  / 7  ) * 20 + 30 );
                day ++ ;
        }
}
```

x座標、y座標を調整するところで、整数剰余と整数除算が使われています。30を掛けているのは、幅や高さを20ピクセルとっているからです。30を足しているのは、それぞれ右方向、下方向に30ピクセル分ずらしているからです。if～ else if 文は、おまけで、色を変えているだけです。

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elseif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elseif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" );
}
```

７－４．複雑な条件分岐

７－４－１．条件分岐のネスト

if文の中に**if**文が入った構文を条件分岐のネスト（多重化：nesting）と呼んでいます。条件分岐のネストは、次のような処理を記述するときに使われます。

・複雑な条件を組み合わせるとき
・大きく処理を分岐した後、より細かな条件分岐を行ないたいとき

例えば、月に関して分類してみましょう。今日の月が変数*month*に求まっているとします。今日の月の求め方は、後の章で説明します。この場合に、小の月（月の日数が31日に満たない月）すなわち2, 4, 6, 9, 11月を求めてみましょう。まず、偶数の月か奇数の月か分けてみます。その後に、偶数の月であれば８よりも小さい月、奇数の月であれば８よりも大きい月に分類しています。

```
if ( month % 2 == 0 ) {              // 偶数の月
        if ( month < 8 ) {       System.out.println( month + "月は、小の月です。" );      }
        else {                   System.out.println( month + "月は、大の月です。" );      }
}
else {                           // 奇数の月
        if ( month > 8 ) {       System.out.println( month + "月は、小の月です。" );      }
        else {                   System.out.println( month + "月は、大の月です。" );      }
}
```

７－４－２．複雑な条件式

条件分岐を組み合わせるときは、条件分岐のネストを使う方法以外に、条件自体を組み合わせる記述の仕方もあります。単純な比較式を組み合わせた条件式を論理式（Logical Expression）と呼んでいます。

▼論理式の書式

条件式　&&　条件式	両方の条件が成立（論理積）	AND（かつ）
条件式　||　条件式	どちらかの条件が成立（論理和）	OR（または）
! (条件式)	否定	NOT（でない）

ちなみに演算子の優先順位としては、否定の!が一番高く、その次は論理積と呼ばれる&&、最後は論理和と呼ばれる||の順になっています。いくつかの例を挙げてみましょう。

```
100 <= x && x <= 200                 // 100 <= x <= 200は間違い
month % 2 == 0 && month < 8          // 偶数の月の小の月の条件
x == 0 || x == 100                   // 0かあるいは100
!( x == 100)                         // 100でない
!( x == 0 || x == 100 )              // 0でも100でもない
```

これらの論理式は、if 文の条件として、あるいはwhile文やfor文の継続条件として用いることができます。論理式を使えば、さきほどのネストを使って記述した大の月、小の月の分類を行なうif文をネストを使わずに記述することができます。&&の方が||よりも優先的に評価されますので、余分な括弧をつけていません。

```
if ( month % 2 == 0 && month < 8 || month % 2 == 1 && month > 8) {
        System.out.println( month + "月は、小の月です。" );      }
else {
        System.out.println( month + "月は、大の月です。" );      }
```

７－４－３．論理式の変換規則

条件式で使われる論理式は、同じ条件を記述するのに複数の記述の仕方ができます。ある条件式と別の条件式が同じ条件を示しているかどうかは、変換法則を適用することによって判別することができます。

★ド・モルガンの定理

ド・モルガン（de Morgan）の定理は論理式の有名な変換法則です。右と左は、等しい記述になっています。

!(論理式１&& 論理式２)	!(論理式１) ｜｜ !(論理式２)
!(論理式１｜｜論理式２)	!(論理式１) && !(論理式２)

たとえば次のような例を見てみましょう。左右は等しい記述です。

!(x == 0 ｜｜ x == 100)	!(x == 0) && !(x==100) あるいは、x != 0 && x != 100
!(100 <= *x* && *x* <= 200)	!(100 <= *x*) ｜｜ !(*x* <= 200) あるいは x < 100 ｜｜ x > 200

★それ以外の変換法則

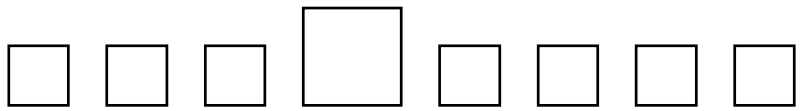
普通の不等号や等号に関して、論理式を使えばいろいろな形で記述することができます。簡単にまとめてみました。左右は等しい記述となっています。AとBは、それぞれ何らかの条件式であると思ってください。

!(A == B)	A != B	等号と不等号の変換
A <= B	A < B ｜｜ A ==B	不等号の分解
!(A >B)	A <= B	逆の不等号への変換

７－５．課題

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。

課題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）。

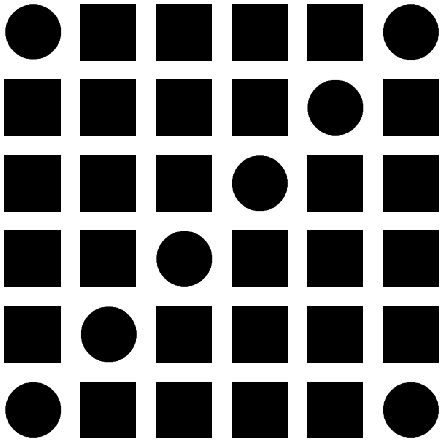


問題の図

課題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。１２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

課題7-4.
多重のwhile文とif文を用いて、次のような図を描くアプレットを作成しなさい。アプレットのクラス名は、OvalinRect。例えば、左上の角の座標は、(20,20)とし、正方形の大きさは一辺15ドットで、上下左右の間隔は5ドットとします。



問題の図