

## 第3章 Java の基本制御構造

Java のその他の制御構造の構文 (**if** 文、**for** 文、**while** 文) は基本的には C 言語と全く同じである。制御構造の復習を兼ねて、これらの制御構造を使ったグラフィックスの例題を取り上げる。

### 3.1 if 文

Java の if 文は C 言語と同じ書き方である。

**if (条件式) 文<sub>1</sub>**

**if (条件式) 文<sub>1</sub> else 文<sub>2</sub>**

条件式が成り立てば文<sub>1</sub>を実行する。1 番めの形式は条件式が成り立たなければ何もしない。2 番めの形式は文<sub>2</sub>を実行する。文<sub>1</sub>,文<sub>2</sub>は、当然ブロック (“{” と “}” で括った文の並び) でも良い。

**Q 3.1.1** 次のプログラムの断片の出力を書け。

```
1.  1      int n = 2;
    2      if (n <= 1) {
    3          System.out.printf("A");
    4      }
    5      if (n <= 2) {
    6          System.out.printf("B");
    7      }
    8      if (n <= 3) {
    9          System.out.printf("C");
   10     }
```

答:           

```
2.  1      int n = 2;
    2      if (n <= 1) {
    3          System.out.printf("A");
    4      } else if (n <= 2) {
    5          System.out.printf("B");
    6      } else if (n <= 3) {
    7          System.out.printf("C");
    8      }
```

答:           

ここで、条件式の型は **boolean** 型である。既に紹介した Graphics クラスの draw3DRect や fill3DRect の引数としても用いられていた。C 言語と異なり整数型 (int 型) とは区別されている。このため (C 言語では OK だった) while (1) ... のような文はエラーとなる。

**問 3.1.2** int 型と boolean 型を区別することの長短をまとめよ。

条件判断文としてはこの他に**switch～case** 文 もあるが、構文は C 言語と同じなので、ここでは説明を割愛する。

**例題 3.1.3** Calendar クラス を使って挨拶を行なう。

java.util.Calendar クラスの使用方法については、API ドキュメントを参照すること。このプログラムでは、日曜日だけ色を赤色に変更し、時間に応じて挨拶文を変えている。

ファイル CalendarTest.java

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.util.*;
4
5 public class CalendarTest extends JPanel {
6     public CalendarTest() {
7         setPreferredSize(new Dimension(250, 100));
8     }
9
10    @Override
11    public void paintComponent(Graphics g) {
12        super.paintComponent(g);
13        Calendar now = Calendar.getInstance();
14        int day = now.get(Calendar.DAY_OF_WEEK);
15        int hour = now.get(Calendar.HOUR_OF_DAY);
16        int min = now.get(Calendar.MINUTE);
17        if (day == Calendar.SUNDAY) {
18            g.setColor(Color.RED);
19            g.drawString("今日は日曜日です。", 30, 25);
20        }
21        if (hour < 12) {
22            g.drawString("おはようございます。", 30, 75);
23        } else if (hour < 18) {
24            g.drawString("こんにちは。", 30, 75);
25        } else {
26            g.drawString("こんばんは。", 30, 75);
27        }
28        g.drawString("ただいま " + hour + "時 " + min + "
29        // g.drawString(String.format("ただいま %d時 %d分
30    }
31
32    public static void main(String[] args) { /* 省略 */
33 }
```

## 3.2 文字列 (String) に関する演算子とメソッド

Java では、\_\_\_\_\_ を用いて String 型と String 型のオブジェクトを **接続する**（あるいは、String 型と String 以外の型のオブジェクトを String 型に変換したものを接続する）ことができる。

例:

```
System.out.println("2 + 2は" + (2 + 2));  
System.out.println(2 + " * " + 3 + "は" + 2 * 3 + "です。");
```

一方、C 言語のような書式指定を行う printf や sprintf メソッドに相当するメソッドも使用できる。上の drawString の場合、String.format というクラスメソッドを使って、次のように書くこともできる。

```
g.drawString(String.format("ただいま %d時 %d分です。", hour, n  
30, 50);
```

書式指定の詳しい仕様は String.format のドキュメントを参照すること。

**詳細:** この printf のようなメソッドは利用するのは簡単だが、総称クラス (Generics) ・ オートボクシング (Autoboxing) ・ 可変個の引数 (Varargs) など、いろいろな考え方が組合せられている。このうち総称クラスについては後述する。

可変個の引数を持つメソッドは API のドキュメントでは、

```
public static String format(String format, Object... args)
```

のように ... を使って表されている。（この format メソッドは java.lang.String クラスのクラスメソッドである。）

**Q 3.2.1** 次の中で "1 + 1 は 2 です。" と出力して改行する式に○を、エラーとなる式に△を、それ以外に×を付けよ。

- ☐ System.out.println("1 + 1 は + (1 + 1) + です。")
- ☐ System.out.println("1 + 1 は " + (1 + 1) + " です。")
- ☐ System.out.println("1 + 1 は " + 1 + 1 + " です。")
- ☐ System.out.println("1 + 1 は "(1 + 1)" です。")
- ☐ System.out.printf("1 + 1 は %d です。%n", 1 + 1)
- ☐ System.out.println(String.format("1 + 1 は %d です。", 1 + 1))

### 3.3 for 文, while 文, do ~ while 文

```
while (条件式1) 文1  
for (式1; 式2; 式3) 文1  
for (型1 変数1: 式1) 文1  
do 文1 while (条件式1);
```

while 文は条件式<sub>1</sub>が成り立つ間、文<sub>1</sub>の実行を繰り返す。

1 つめの形式の **for** 文はループに入る前に、まず式<sub>1</sub>を評価する。式<sub>2</sub>が成り立つ間、文<sub>1</sub>、式<sub>3</sub>の実行を繰り返す。

2 つめの形式の **for** 文は JDK5.0 で導入されたものである。**for-each** 文 と呼ばれることもある。（ただし、each というキーワードを使うわけではないので注意する。）この場合、式<sub>1</sub>は直感的には何かの集まりを表すデータ型（配列など—正確には配列またはインタフェース `Iterable` を実装するクラス）でなければならない。コロン (:) の前で宣言された変数<sub>1</sub>に、この列の要素が順に代入され、文の実行が繰り返される。この形式の **for** 文の使用例はもう少し後で紹介する。

繰り返し文としてはこの他に **do ~ while** 文 もあるが、C 言語と同じなのでここでは説明を割愛する。

変数への代入はCと同様 \_\_\_\_\_ を使う。

**Q 3.3.1** 次のプログラムの断片の出力を書け。

```
1      int i;
2      for (i = 0; i < 4; i++) {
3          System.out.printf("%d", i);
4      }
5      System.out.printf("|%d", i);
```

答: \_\_\_\_\_

**例題 3.3.2** 正多角形の描画正  $n$  角形を描画する。

ファイル `N_gon.java`

```
1  import javax.swing.*;
2  import java.awt.*;
3  import static java.lang.Math.*;
4
5  public class N_gon extends JPanel {
6      public N_gon() {
7          setPreferredSize(new Dimension(220, 220));
8      }
9
10     @Override
11     public void paintComponent(Graphics g) {
12         super.paintComponent(g);
13
14         int np = 7;
15         int sc = 100;
16
17         int i;
18         double theta1, theta2;
19         for (i = 0; i < np; i++) {
20             // 単位 ラジアン
21             theta1 = PI * 2 * i / np; // 360 * i / np
22             theta2 = PI * 2 * (i + 1) / np; // 360 * (i + 1) / np
23             g.drawLine((int)(sc * (1.1 + cos(theta1))),
24                       (int)(sc * (1.1 + sin(theta1))),
25                       (int)(sc * (1.1 + cos(theta2))),
26                       (int)(sc * (1.1 + sin(theta2))));
27         }
28     }
29
30     public static void main(String[] args) { /* 省略 */
```

```
31 }
```

`Math.PI` は円周率  $\pi$  (=3.1415...)、`Math.sin`、`Math.cos` は正弦、余弦関数である。これらはクラスフィールド、クラスメソッドである。上のプログラムでは `static import` しているので、プログラム中では単に `PI`、`sin`、`cos` で使用している。

二次関数のグラフの描画 数学関数のグラフを描くには、定義域を細かい区間に区切り、短い線分をつなぎ合わせれば良い。

ファイル `Parabola.java`

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class Parabola extends JPanel {
5     public Parabola() {
6         setPreferredSize(new Dimension(200, 200));
7     }
8
9     @Override
10    public void paintComponent(Graphics g) {
11        super.paintComponent(g);
12        double a = -0.0025, b = 1, c = 0;
13        for (int x0 = 0; x0 < 200; x0 += 10) {
14            double y0 = a * x0 * x0 + b * x0 + c;
15            int x1 = x0 + 10;
16            double y1 = a * x1 * x1 + b * x1 + c;
17            g.drawLine(x0, (int)y0, x1, (int)y1);
18            System.out.printf("(%d, %.1f) -- (%d, %.1f)",
19                x0, y0, x1, y1);
20        }
21
22        public static void main(String[] args) { /* 省略 */
23    }
```

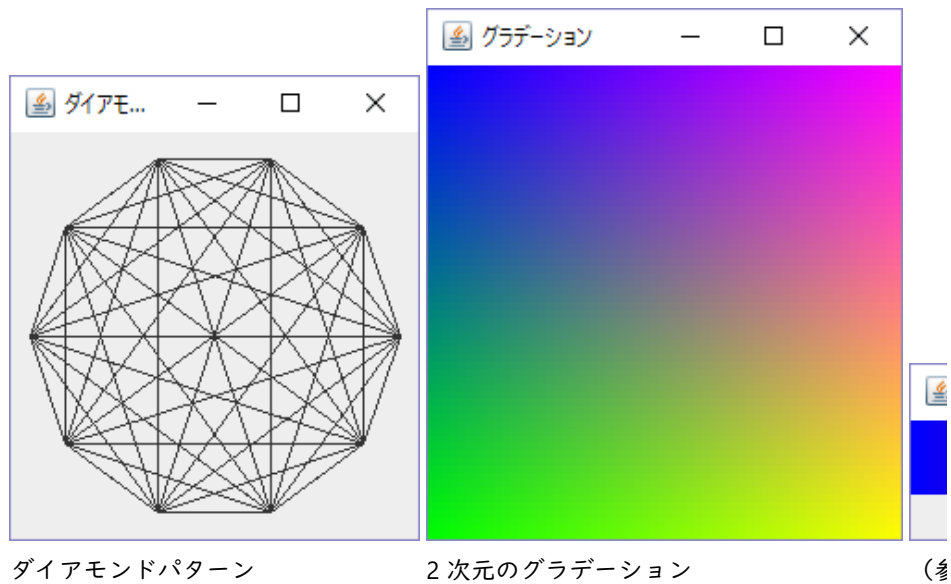
**問 3.3.4**  $y = ax^2$ ,  $y = \sin(x)$ ,  $y = \cos(x)$  などの数学関数のグラフを描く GUI アプリケーションを書け。

参考:

<https://docs.oracle.com/javase/jp/15/docs/api/java.base/java/la>

**問 3.3.5** 正  $n$  角形のすべての頂点を結んでできる図形（ダイヤモンドパターン）を描画する GUI アプリケーションを書け。

**問 3.3.6** 色のグラデーション（2次元 — 縦方向と横方向が別の色に変わる）を作成する GUI アプリケーションを書け。



### (参考)

#### 1次元のグラデーション

ファイル `Gradation1.java`

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class Gradation1 extends JPanel {
5     public Gradation1() {
6         setPreferredSize(new Dimension(256, 40));
7     }
8
9     @Override
10    public void paintComponent(Graphics g) {
11        super.paintComponent(g);
12        int scale = 4;
13        int i;
14
15        for (i = 0; i < 64; i++) {
16            g.setColor(new Color(i * 4, 0, 255 - i * 4));
17            g.fillRect(i * scale, 0, scale, scale * 10);
18        }
19    }
20
21    public static void main(String[] args) { /* 省略 */
22    }

```

#### 例題 3.3.7 棒グラフの描画



整数のデータを与え、そのデータの棒グラフを描く。

ファイル Graph.java

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class Graph extends JPanel {
5     public Graph() {
6         setPreferredSize(new Dimension(200, 105));
7     }
8
9     @Override
10    public void paintComponent(Graphics g) {
11        super.paintComponent(g);
12        int[] is = {10, 4, 6, 2, 9, 1};
13        Color[] cs = {Color.RED, Color.BLUE};
14        int scale = 15;
15        int i, n = is.length;
16
17        for (i = 0; i < n; i++) {
18            g.setColor(cs[i % cs.length]);
19            g.fillRect(0, i * scale, is[i] * scale, scale);
20        }
21    }
22
23    public static void main(String[] args) { /* 省略 */
24 }
```

配列オブジェクトの \_\_\_\_\_ というフィールド(?) によって配列の大きさ（要素数）を知ることができる。これも C 言語と異なる点である。**for** 文の中のブロックは変数 *i* が 0 ～ *n* - 1 まで変化する間、繰り返される。

**Q 3.3.8** *ds* という `double` 型の配列があったとき、*ds* の要素の平均値を求めるメソッドを定義する。2 箇所の空欄を同じ内容で埋めて定義を完成せよ。

```
1     double average(double[] ds) {
2         double n = 0
3         int i;
4         for (i = 0; i < _____; i++) {
5             n += ds[i];
6         }
7         return n / _____;
8     }
```

答: \_\_\_\_\_

C 言語では、配列の範囲外をアクセスしても通常エラーにならないが、Java では `ArrayIndexOutOfBoundsException` という例外が発生する。例外については次章で詳しく説明する。

**Q 3.3.9** 次のプログラムを実行して、エラーメッセージを確認せよ。

ファイル ArrayIndexOutOfBoundsExceptionTest.java

```
1 public class ArrayIndexOutOfBoundsExceptionTest {
2     public static void main(String args[]) {
3         int[] a = {1, 2, 3};
4         for (int i = 0; i <= a.length; i++) {
```

```

5         System.out.println(a[i]);
6     }
7 }
8 }

```

また、次の C 版も実行してみよ。

ファイル ArrayIndexOutOfBoundsExceptionTest.c

```

1 #include <stdio.h>
2
3 int main(void) {
4     int a[] = {1, 2, 3};
5     int i;
6     for (i = 0; i <= sizeof(a) / sizeof(a[0]); i++) {
7         printf("%d\n", a[i]);
8     }
9
10    return 0;
11 }

```

## 3.4 多次元配列

**例題 3.4.1** int 型の  $8 \times 8$  の大きさの配列の配列を調べて、1 なら白丸、2 ならば黒丸を画面上の対応する位置に描画する。

ファイル Othello.java

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class Othello extends JPanel {
5     private final int scale = 40;
6     private final int space = 3;
7
8     public Othello() {
9         setPreferredSize(new Dimension(scale * 8 + 1, scale
10     })
11
12     @Override
13     public void paintComponent(Graphics g) {
14         super.paintComponent(g);
15
16         int[][] state = {{0,1,0,0,1,2,2,2}, {2,2,1,1,1,1,2,0},
17                         {0,0,0,2,2,2,1,1}, {1,1,1,0,2,2,0,2},
18                         {1,2,1,0,0,0,0,0}, {0,0,0,1,1,2,2,2},
19                         {2,1,0,0,0,2,2,2}, {2,2,2,2,0,0,0,0}};
20
21         int i, j;

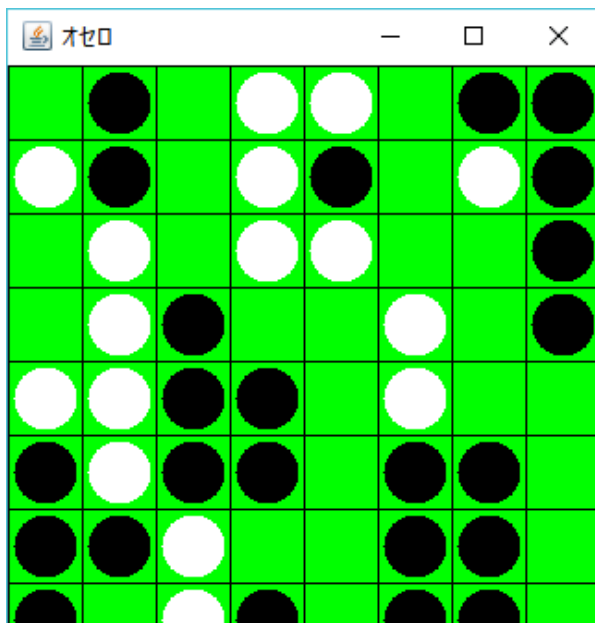
```



```

22     for (i = 0; i < 8; i++) {
23         for (j = 0; j < 8; j++) {
24             g.setColor(Color.GREEN);
25             g.fillRect(i * scale, j * scale, scale, scale);
26             g.setColor(Color.BLACK);
27             g.drawRect(i * scale, j * scale, scale, scale);
28             if (state[i][j] == 1) {
29                 g.setColor(Color.WHITE);
30                 g.fillOval(i * scale + space, j * scale + space,
31                     scale - space * 2, scale - space * 2);
32             } else if (state[i][j] == 2) {
33                 g.setColor(Color.BLACK);
34                 g.fillOval(i * scale + space, j * scale + space,
35                     scale - space * 2, scale - space * 2);
36             }
37         }
38     }
39 }
40
41 public static void main(String[] args) { /* 省略 */ }
42 }

```





2次元配列（配列の配列）を宣言するには、上のように [] を2つ重ねる（3次元以上も同様）。C言語と異なり、要素数を宣言する必要はない。（ただし、C言語でも最初の次元の要素数は省略することができる。）state は配列の配列で、例えば、state[0][1] は、0番めの配列 {0,1,0,0,1,2,2,2} の1番めの数だから   が入っている部分である。つまりこの位置（0列めの1行め）には白丸が描画される。

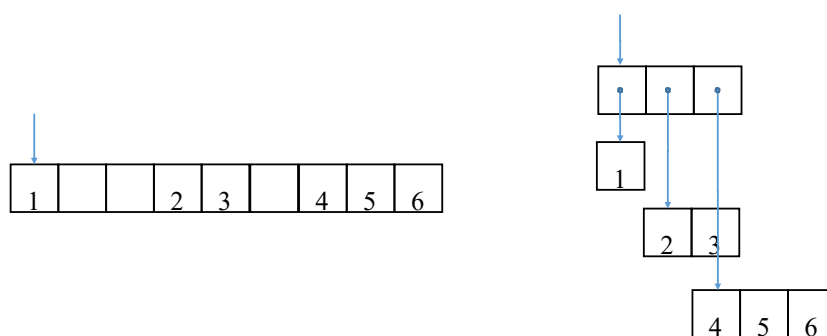
**注意:** なお、Javaの2次元配列とCの2次元配列はメモリー上の配置の仕方が異なる。（もっともJavaでメモリー上の配置を意識する必要はほとんどない。）このためJavaではCではメモリーの効率が悪い次のような2次元配列（異なるサイズの配列が混在している）

```
int[][] xss = {{1}, {2, 3}, {4, 5, 6}};
```

も使用できる。（Cだと

```
int xss[][3] = {{1}, {2, 3}, {4, 5, 6}};
```

と宣言する必要がある。）



C言語の場合

Javaの場合

**Q 3.4.2** int[][] xss = {{1}, {2, 3}, {4, 5, 6}}; のとき次の式の値を答えよ。範囲外でエラーになるときは×を書け。

1. xss[1][1]      答:
2. xss[0][1]      答:
3. xss[2][1]      答:
4. xss.length    答:
5. xss[1].length 答:

**キーワード**

if 文, if ~ else 文, while 文, for 文, for-each 文, 配列, length フィールド, ArrayIndexOutOfBoundsException 例外, static, Math クラス, 多次元配列,