グラフィックス基礎・演習・第 13 回

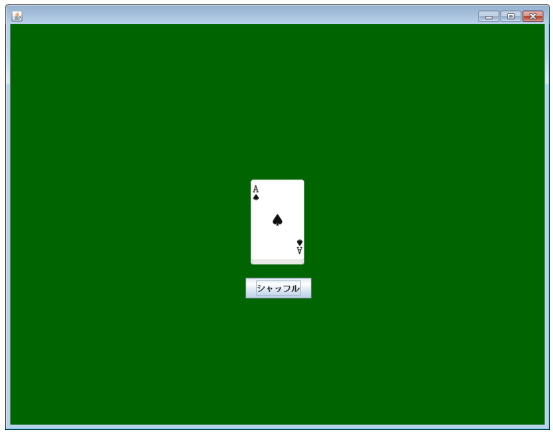
2013 年 7 月 9 日

今回の内容

13.1 入れ子クラス

オブジェクト指向のプログラミングでは、オブジェクトに特定の仕事を行わせるために、その設計図としてクラスを定義します。これまで見てきた Java プログラムでは、基本的にソースファイルのトップレベルでのみクラスを宣言していました。Java では、ソースファイルのトップレベル以外でも、いろいろなところでクラス宣言を行って、それを利用することができるようになっています。ソースファイルのトップベルではなく、他のクラス宣言の内部で宣言されたクラスを**入れ子クラス (nested classes)** と呼びます。今回は、この入れ子クラスと呼ばれるいくつかのクラス宣言の形について勉強します。入れ子クラスには、今回紹介する**メンバクラス**、**ローカルクラス**、**匿名クラス**の 3 種類があります。

例題プログラム　いろいろな入れ子クラスを紹介するための例題として、次の G1301.java というプログラムについて考えます。このプログラムは、右の図のようにゲーム盤の中央にデッキを表向きに置き、そのすぐ下に追加したボタンをクリックすることで、デッキがシャッフルされるようにしたものです。このプログラムは、第 10 回で紹介した G1001.java と同じ仕事を行います。



G1301.java

```
 1 import javax.swing.*;
 2 import jp.ac.ryukoku.math.graphics.*;
 3
 4 class G1301Panel extends GamePanel {
 5     Deck deck = new Deck();
 6     JButton button = new JButton("シャッフル");
 7
 8     G1301Panel() {
 9         deck.flip();
10         add(deck);
11         add(button, 352, 380);
12         button.addActionListener(new ShuffleButtonHandler(deck));
13     }
14 }
15
```

```
16 class G1301 implements Runnable {
17     public void run() {
18         JFrame f = new JFrame();
19         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20         f.add(new G1301Panel());
21         f.pack();
22         f.setVisible(true);
23     }
24
25     public static void main(String[] args) {
26         SwingUtilities.invokeLater(new G1301());
27     }
28 }
```

このプログラムでは、ボタン (JButton) がクリックされたときに発生する ActionEvent のイベントハンドラとして、次のように宣言された ShuffleButtonHandler クラス[1]のインスタンスを生成して使用しています (12 行目)。

ShuffleButtonHandler.java

```
 1 import java.awt.event.*;
 2 import jp.ac.ryukoku.math.graphics.*;
 3
 4 class ShuffleButtonHandler implements ActionListener {
 5     Pile p;
 6
 7     ShuffleButtonHandler(Pile p) {
 8         this.p = p;
 9     }
10
11     public void actionPerformed(ActionEvent e) {
12         p.shuffleAsync();
13     }
14 }
```

13.2 メンバクラス

G1301.java の G1301Panel クラスでは、全く独立に宣言した ShuffleButtonHandler クラスを使用していますが、このクラスを G1301Panel クラスだけが利用するのなら、むしろ G1301Panel のクラス宣言の中で宣言する方が自然です。

Java では、クラス宣言の {} の直下に、つまりそのクラスのコンストラクタ、インスタンス変数、インスタンスメソッド、クラス変数、クラスメソッドなどと同じレベルで、別のクラスを宣言することができます。このように宣言されたクラスは、その宣言を含んでいる (外側の) クラスの**メンバクラス**と呼ばれます。

次のクラス宣言は、ShuffleButtonHandler クラスを、G1301Panel クラスのメンバクラスとして宣言したものです。

ShuffleButtonHandler をメンバクラスとした G1301Panel の宣言

```
1 import java.awt.event.*;
2 import javax.swing.*;
3 import jp.ac.ryukoku.math.graphics.*;
```

[1]第 10 回の 5 ページのプログラムと同じものです。

```
 4
 5 class G1301Panel extends GamePanel {
 6     static class ShuffleButtonHandler implements ActionListener {
 7         Pile p;
 8
 9         ShuffleButtonHandler(Pile p) {
10             this.p = p;
11         }
12
13         public void actionPerformed(ActionEvent e) {
14             p.shuffleAsync();
15         }
16     }
17
18     Deck deck = new Deck();
19     JButton button = new JButton("シャッフル");
20
21     G1301Panel() {
22         deck.flip();
23         add(deck);
24         add(button, 352, 380);
25         button.addActionListener(new ShuffleButtonHandler(deck));
26     }
27 }
```

ShuffleButtonHandler.java で宣言されていた ShuffleButtonHandler クラスの宣言が、そのままの形で、このプログラムの 6 〜 16 行目に取り込まれていることに注意してください。新しいプログラムの 18 行目以降は変っていません。

6 行目の先頭の static は、ここでメンバクラスとして宣言された ShuffleButtonHandler のインスタンスが、G1301Panel クラスのインスタンスとは独立に存在することができることを示しています。このように static という修飾子を伴って宣言されたメンバクラスは、通常のクラス宣言で定義されるクラスと特に違いはありません。たとえば、G1301Panel の static なメンバクラス ShuffleButtonHandler のインスタンスも、ShuffleButtonHandler.java のトップレベルで宣言された ShuffleButtonHandler クラスのインスタンスも同じように働くことができます。

唯一つの違いは、メンバクラスを、それを含んでいるクラス宣言の外から参照する際には、クラス変数やクラスメソッドを使用するときのように、

[そのメンバクラスを含むクラス名].[メンバクラス名]

という形で指し示さなければならない点です[2]。たとえば、25 行目は G1301Panel の宣言の中ですので、単に、ShuffleButtonHandler という単純なクラス名で指し示すことができていますが、もし、この文が G1301Panel のクラス宣言の外側に書かれているのなら、

```
button.addActionListener(new G1301Panel.ShuffleButtonHandler(deck));
```

[2]メンバクラスを完全限定名で指し示す場合は、この前に、さらに [パッケージ名]. が付くことになります。メンバクラスの宣言には、インスタンスメソッドなどの宣言と同様に、public、protected、private などのアクセス修飾子を付けることもできます。そのメンバクラスが、そのメンバクラスを含んでいるクラスの外側からアクセス可能かどうかは、これらのアクセス修飾子の設定によって変わります。

のように書かないといけません[3]。

13.3 内部クラス

ボタンがクリックされた際に、ShuffleButtonHandler クラスのインスタンスがデッキをシャッフルするためには、どのデッキをシャッフルすればよいのか記憶しておく必要があります。このため、ShuffleButtonHandler は、Pile p; と宣言されたインスタンス変数を持っていて、コンストラクタの引数として渡されたデッキを、このインスタンス変数で記憶するようにしています。

G1301Panel のメンバクラス ShuffleButtonHandler のインスタンスの場合、G1301Panel クラスの特定のインスタンスのために働いていますので、シャッフルする対象となるデッキを記憶する代りに、自分が担当している G1301Panel クラスのインスタンスを記憶しておき、次のようなプログラムとすることもできるはずです (変更された行を * 印で示してあります)。

メンバクラス ShuffleButtonHandler の宣言 (その 2)

```
  ⋮
  5 class G1301Panel extends GamePanel {
  6     static class ShuffleButtonHandler implements ActionListener {
* 7         G1301Panel panel;
  8
* 9         ShuffleButtonHandler(G1301Panel panel) {
*10             this.panel = panel;
 11         }
 12
 13         public void actionPerformed(ActionEvent e) {
*14             panel.deck.shuffleAsync();
 15         }
 16     }
 17
 18     Deck deck = new Deck();
 19     JButton button = new JButton("シャッフル");
 20
 21     G1301Panel() {
 22         deck.flip();
 23         add(deck);
 24         add(button, 352, 380);
*25         button.addActionListener(new ShuffleButtonHandler(this));
 26     }
 27 }
```

こちらの ShuffleButtonHandler の宣言では、デッキ (Pile) の代わりに G1301Panel のインスタンスを覚えるようにしています (7 行目)。これに伴い 25 行目では、デッキの代わりに G1301Panel のインスタンス自身をコンストラクタに渡しています。

この例のように、そのクラスのインスタンスのために何らかの仕事をするようなオブジェクトのクラスとして、そのメンバクラスが定義されることがよくあります。その際、メンバクラスのオブジェクトは、自分はどのインスタンスの仕事を担当しているのかを記憶しているのが普通です。

Java では、static という修飾子を付けずにメンバクラスを宣言することで、メンバクラスのインスタンスを、そのメンバクラスを宣言しているクラスのインスタンスに自動的に関係付けることができます。このようなメンバクラス宣言を利用すると、次のプログラムのように、簡潔に

[3] G1301Panel のクラス宣言の内部であっても、こう書いて構いません。

G1301Panel とそのメンバクラス ShuffleButtonHandler を宣言することができます (変更された行を * 印で示してあります)。

内部クラスとしてのメンバクラス ShuffleButtonHandler の宣言

```
  ⋮
  5 class G1301Panel extends GamePanel {
* 6     class ShuffleButtonHandler implements ActionListener {
  7         public void actionPerformed(ActionEvent e) {
* 8             deck.shuffleAsync();
  9         }
 10     }
 11
 12     Deck deck = new Deck();
 13     JButton button = new JButton("シャッフル");
 14
 15     G1301Panel() {
 16         deck.flip();
 17         add(deck);
 18         add(button, 352, 380);
*19         button.addActionListener(new ShuffleButtonHandler());
 20     }
 21 }
```

このプログラムの 6 〜 10 行目のように、static なしに宣言されたメンバクラスのインスタンスは、そのメンバクラスを宣言している (外側の) クラスのインスタンスに従属した形でしか存在できません。特定のクラスのインスタンスに従属した形でしか存在できないようなオブジェクトのクラスを、Java では一般に**内部クラス (inner classes)** と呼びます。

外側のインスタンス　内部クラスのインスタンスが従属しているオブジェクトのことを、そのインスタンスの**外側のインスタンス (enclosing instances)** と呼びます。内部クラスのインスタンスは、言わば、その上司である「外側のインスタンス」の仕事の一部を担当する部下のようなものです。

内部クラスのインスタンスを生成するには、まず、その上司となる「外側のインスタンス」が存在していなければなりません。内部クラスのインスタンスが生成される際には、明示的にあるいは暗黙の内に、その外側のインスタンスが指定されることになります。このとき、生成される (内部クラスの) インスタンスには、自動的に名前のないインスタンス変数が 1 つ追加され、そこに指定された「外側のインスタンス」が記憶されます。

static でないメンバクラスは内部クラスとなりますので、そのインスタンスは、そのメンバクラスを宣言している (外側の) クラスのインスタンスがなければ生成することができません。その外側のクラスのコンストラクタやインスタンスメソッドの中で、このようなメンバクラスのインスタンスを生成すると、このとき this で表されるオブジェクトを「外側のインスタンス」として、メンバクラスのオブジェクトが生成されます。

先のプログラムの 19 行目では、内部クラス ShuffleButtonHandler のインスタンスを生成していますが、ここは G1301Panel のコンストラクタの中ですので、生成されたばかりの G1301Panel クラスのインスタンスを「外側のインスタンス」として、内部クラス ShuffleButtonHandler のインスタンスが生成されます。ShuffleButtonHandler のインスタンスは、自動的に、隠れたインス

タンス変数で、その「外側のインスタンス」である G1301Panel のインスタンスを記憶しますので、メンバクラス ShuffleButtonHandler の宣言からインスタンス変数 panel の宣言がなくなってしまっています。また、そのインスタンス変数を初期化する必要もなくなったため、特にコンストラクタは宣言されていません[4]。

内部クラスでは、そのクラスのインスタンス変数やインスタンスメソッドに加えて、その「外側のインスタンス」のインスタンス変数やインスタンスメソッドに直接アクセスすることが可能です。このため、8 行目では、あたかも deck が ShuffleButtonHandler クラスのインスタンス変数であるかのように、この変数の値を参照しています。

13.4 ローカルクラス

static でないメンバクラスを宣言する方法以外にも、内部クラスを宣言する方法があります。内部クラスは、コンストラクタやメソッドなどの宣言の本体に現れるブロック {} の中で宣言することもでき、このように宣言されたクラスを**ローカルクラス (local classes)** と呼びます[5]。次のプログラムは、ローカルクラスを利用して G1301Panel クラスの宣言を書き換えたものです。

ローカルクラスとしての ShuffleButtonHandler の宣言

```
  ⋮
  5 class G1301Panel extends GamePanel {
  6     Deck deck = new Deck();
  7     JButton button = new JButton("シャッフル");
  8
  9     G1301Panel() {
 10         deck.flip();
 11         add(deck);
 12         add(button, 352, 380);
*13         class ShuffleButtonHandler implements ActionListener {
*14             public void actionPerformed(ActionEvent e) {
*15                 deck.shuffleAsync();
*16             }
*17         }
 18         button.addActionListener(new ShuffleButtonHandler());
 19     }
 20 }
```

13.5 匿名クラス

static でないメンバクラスや、ローカルクラスとして ShuffleButtonHandler を宣言することで、G1301Panel クラスの宣言はかなり簡潔なものとなりましたが、よく注意してみると、このプログラムの中で ShuffleButtonHandler を利用しているのは、

```
button.addActionListener(new ShuffleButtonHandler());
```

の 1 箇所だけであることに気づきます。

このように、たった 1 箇所でのインスタンス生成のためだけに、そのクラスに適当な名前を付けてクラス宣言を行うのは面倒です。Java には、このような場合に便利な、クラス宣言とそのイン

[4] ShuffleButtonHandler クラスはデフォルトコンストラクタのみを持つことになります。

[5] ローカルクラスの宣言では、final でないクラス変数、クラスメソッドの宣言を行うことはできません。

スタンス生成を同時に行う仕組みが用意されています。このようなインスタンスは、次のような書式のインスタンス生成式で生成することができます。

```
new スーパークラス(インタフェース)名( コンストラクタの引数の列 ) {
      インスタンスメソッドなどの宣言[6]
}
```

この書式の意味は、適当なクラス名 X を選び、スーパークラス(インタフェース)名 のサブクラスとして、インスタンスメソッドなどの宣言 を持つ内部クラス X を宣言した上で、

```
new X( コンストラクタの引数の列 )
```

というインスタンス生成式を評価して、内部クラス X のインスタンスを生成する、というものです。ただし、クラス X には コンストラクタの引数の列 を、そのままスーパークラスのコンストラクタに渡すようなコンストラクタが用意されます。この内部クラス X に相当する名前のないクラスのことを**匿名クラス (anonymous classes)** と呼びます。匿名クラスはすべて内部クラスとなることに注意してください。

匿名クラスを利用すると、G1301Panel クラスのプログラムは、次のように書き換えることができます。

匿名クラスを利用した G1301Panel クラス

```
  ⋮
 5 class G1301Panel extends GamePanel {
 6     Deck deck = new Deck();
 7     JButton button = new JButton("シャッフル");
 8
 9     G1301Panel() {
10         deck.flip();
11         add(deck);
12         add(button, 352, 380);
*13         button.addActionListener(new ActionListener() {
*14             public void actionPerformed(ActionEvent e) {
*15                 deck.shuffleAsync();
*16             }
*17         });
 18     }
 19 }
```

これでかなり簡潔なプログラムとなりました。イベントハンドラを別クラスとせず、G1301Panel クラスで直接 ActionListener を実装する方法をとれば、さらにプログラムを短くすることが可能ですが、複数のボタンがあって、それぞれ別のイベントハンドラを登録したいような場合には、その方法を使うことができません。そのような場合には匿名クラスが威力を発揮します。

ローカルクラスや匿名クラスからのローカル変数へのアクセス　コンストラクタやメソッドなどの中などで宣言されたローカルクラスや匿名クラスのインスタンスからは、そのコンストラクタやインスタンスメソッド内の final なローカル変数にもアクセスすることができます。たとえば、上

[6] インスタンスメソッドの宣言の他にも、インスタンス変数、final なクラス変数、メンバクラスの宣言をすることができます。コンストラクタ、final でないクラス変数、クラスメソッドの宣言を行うことはできません。

のプログラムで、匿名クラスの actionPerformed メソッドからアクセスされている変数 deck は G1301Panel クラスのインスタンス変数ですが、これをコンストラクタの final なローカル変数として宣言し、次のようなプログラムに変更することもできます。

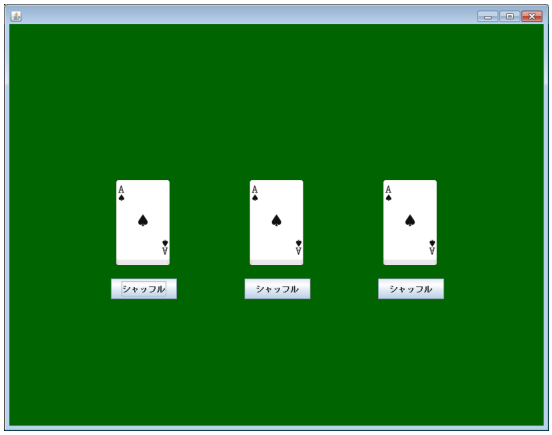
匿名クラスからのローカル変数へのアクセス

```
  ⋮
  5 class G1301Panel extends GamePanel {
  6    JButton button = new JButton("シャッフル");
  7
  8     G1301Panel() {
* 9         final Deck deck = new Deck();
 10         deck.flip();
 11         add(deck);
 12         add(button, 352, 380);
 13         button.addActionListener(new ActionListener() {
 14             public void actionPerformed(ActionEvent e) {
 15                 deck.shuffleAsync();
 16             }
 17         });
 18     }
 19 }
```

ローカルクラスや匿名クラスのからアクセスできるローカル変数は final な変数でないといけないことに注意しましょう。たとえば、上のプログラムで、匿名クラスの actionPerformed メソッドが起動されるのは、その匿名クラスの宣言を含んでいるコンストラクタの起動が終了した後であるはずです。本来なら、この時、コンストラクタのローカル変数 deck は、もう存在していません。このため、一般のローカル変数については、ローカルクラスや匿名クラスの中 (で宣言されたインスタンスメソッド) からその変数にアクセスすることは許されません。一方、final なローカル変数の値は変更されませんので、その値のコピーを (匿名クラスなどの) インスタンス内部に記憶しておくことで、あたかもそのローカル変数が引き続き存在しているかのように振る舞わせることができるようになっています。

13.6 演習問題

1. G1301.java を、匿名クラスをイベントハンドラとして利用するように書き換えたもの (7 ページまたは 8 ページ参照) を、G1302.java にコピーし、ゲーム盤に 3 つのデッキと 3 つのボタンを右の図のように配置し、ボタンをクリックすると、そのボタンの上のデッキをシャッフルできるように改造しなさい。



   3 つのデッキの座標は、左から (160, 240)、(360, 240)、(560, 240)、ボタンの座標は、左から (152, 380)、(352, 380)、(552, 380) となっています。

   ただし、3 つのボタンのイベントハンドラとしては、プログラムの同じ場所で宣言された匿名クラスの 3 つのインスタンスを、それぞれ登録するようにしなさい。また、この匿名ク

ラスの actionPerformed メソッドは、引数 (ActionEvent 型) を参照せずにデッキをシャッフルするようにしてください。

イベントハンドラとなるオブジェクトは 3 つ必要なのに対して、匿名クラスは 1 つだけですので、for 文などで繰り返し実行される文に匿名クラスが現れることになります。匿名クラスの actionPerformed の中から、コンストラクタの final なローカル変数を参照することで、シャッフルする対象となるデッキが区別できるようにしてください。

2. 次のように宣言されたクラス G1303Panel は、最初は中央に置かれた 1 枚のカード (ジョーカー) を、マウスのボタンが押される度に、マウスポインタの位置に移動していくようなゲーム盤のクラス (GamePanel のサブクラス) となっています。

G1303Panel.java

```
import java.awt.event.*;
import jp.ac.ryukoku.math.graphics.*;

class G1303Panel extends GamePanel implements MouseListener {
    Card card = new Card();

    G1303Panel() {
        add(card);
        addMouseListener(this);
    }

    public void mousePressed(MouseEvent e) {
        card.moveAsyncTo(e.getX() - card.getWidth() / 2,
            e.getY() - card.getHeight() / 2);
    }

    public void mouseReleased(MouseEvent e) { }
    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}
```

このゲーム盤のクラス (G1303Panel) では、自分自身のクラスで MouseListener を実装していますが、この G1303Panel クラスに、static でないメンバクラス CardMover の宣言を追加して、その CardMover のインスタンスを MouseListener として使用するように変更しなさい。ソースファイルの名前は G1303Panel.java とし、G1303Panel クラス自身からは mousePressed などのインスタンスメソッドの宣言を削除すること。

MouseListener インタフェースについては、第 10 回の「付録 : いろいろなイベントとイベントハンドラ」(第 10 回 15 ページ) を参照してください。MouseEvent クラスのインスタンスメソッド getX や getY は、イベントが発生したときのマウスポインタの (その部品内での) 位置の x 座標や y 座標を戻り値として返します。

また、このプログラムをテストする際には、次の G1303.java を利用してください。

G1303.java

```
import javax.swing.*;

class G1303 implements Runnable {
    public void run() {
```

```
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new G1303Panel());
        f.pack();
        f.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new G1303());
    }
}
```

3. 演習問題 2 のテストに利用した G1303.java では、G1303 クラスで Runnable インタフェースを実装し、main メソッドでは、この G1303 クラスのインスタンスを SwingUtilities クラスのクラスメソッド invokeLater の引数としています。G1303.java を書き換えて、G1303 クラスでは Runnable インタフェースは実装せず、main メソッド内で宣言した匿名クラスのインスタンスを invokeLater の引数にすることで同じ動作を実現しなさい。

グラフィックス基礎・演習・第 13 回・終り