

今回の内容

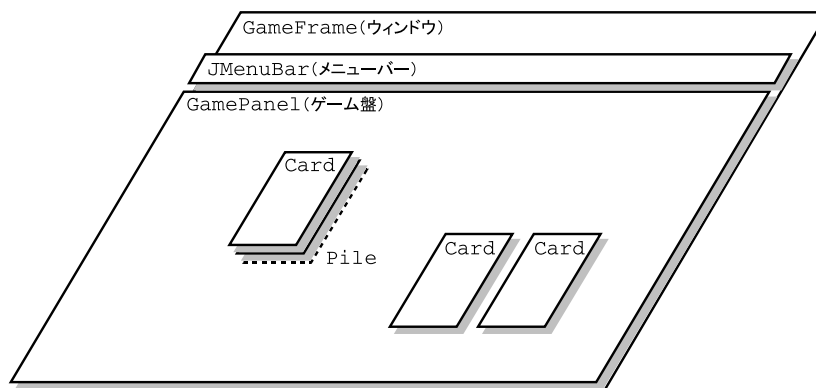
2.1 JFC と Swing	2-1
2.2 グラフィックスコンテキスト (Graphics クラス)	2-3
2.3 図形や文字の描画	2-5
2.4 色の表現 (Color クラス)	2-7
2.5 演習問題	2-9

2.1 JFC と Swing

まず、Java 標準のグラフィックス機能の使い方について勉強します。Java には、グラフィカルユーザーインターフェース (GUI) を構築するための枠組みとして、Java Foundation Classes (JFC) と呼ばれるクラス群が用意されています。JFC は、Swing と呼ばれる GUI の部品群¹や2次元のグラフィックス (文字、図形、画像などの描画) などの機能で構成されています。

この科目で使用しているカードゲーム向けクラスライブラリも、この JFC を用いて実現されています。Card や Pile、GameFrame といったクラスはすべて GUI の部品となっていて、これら以外にも GamePanel (カードなどが配置される濃緑色のゲーム盤) や JMenuBar (ゲーム盤の上のメニューバー) などのクラスが GUI の部品として使われています。

これらの部品は、次の図のように配置されていて、下側にあるものから順に、それぞれが自分自身の描画を行うことで、ウィンドウ全体の様子が描画される仕組みとなっています。



メモ

¹ ボタンやメニューなど

Swing JFC には、Swing と呼ばれる GUI の部品群 (Toolkit) が含まれており、`javax.swing` というパッケージとして、次の表のような多く種類の GUI 部品のクラスが提供されています。

<code>JFrame</code>	タイトルバーなどを持つトップレベルのウィンドウ
<code>JWindow</code>	装飾のないトップレベルのウィンドウ
<code>JDialog</code>	ダイアログボックス用のウィンドウ
<code>JComponent</code>	以下の部品の (直接あるいは間接の) スーパークラス
<code>JPanel</code>	部品を配置するための台紙
<code>JMenuBar</code>	メニューバー
<code>JToolBar</code>	ツールバー
<code>JLabel</code>	アイコン付きの文字列
<code>JButton</code>	ボタン
<code>JCheckBox</code>	on/off を選択できるボタン
<code>JRadioButton</code>	いくつかの選択肢から 1 つだけを選択できるボタン
<code>JMenu</code>	メニュー
<code>JMenuItem</code>	メニュー項目
⋮	⋮

カードゲーム向けクラスライブラリでは、これらの Swing のクラスの内、メニューバー関連のクラス群²は直接、他のクラスについては、そのサブクラスを作成して利用しています。

カードゲーム向けクラスライブラリのクラス	そのスーパークラスとなっている Swing のクラス
<code>GameFrame</code>	<code>JFrame</code>
<code>GamePanel</code>	<code>JPanel</code>
<code>Card</code>	<code>JComponent</code>
<code>Pile</code>	<code>JComponent</code>

また、`Deck` クラスは `Pile` のサブクラスとして実現されています。

メモ

paintComponentメソッド Swing の GUI 部品群は、トップレベルのウィンドウを除けば、すべて `JComponent` というクラスのサブクラスとして実現されています。 `JComponent` はすべての部品の元となるクラスで、すべての部品に共通するインスタンス変数やインスタンスメソッドが宣言されています。その内の 1 つに

```
protected void paintComponent(java.awt.Graphics g)
```

²`JMenuBar` や `JMenu`、`JMenuItem` など

というインスタンスメソッドがあり、その部品を画面に描画しなければならない時に、自動的にこのメソッドが呼ばれるような仕組みとなっています。JComponent のサブクラスとして作成した部品では、このメソッドを再定義 (オーバーライド) することで、その部品独自の表示を行うことが可能となります。

メモ

2.2 グラフィックスコンテキスト (Graphics クラス)

GUI 部品が画面に現れたり、部品の内容が変更されて、その部品を画面に描画する (し直す) 必要が生じると、その部品の `paintComponent` メソッドが自動的に起動されますが、その際、引数として、`java.awt.Graphics` クラスのインスタンスが渡されます³。このクラスのオブジェクトは、画面に対して文字や図形、画像などの描画を行う機能を持っています。

`Graphics` クラスのインスタンスは、出力先 (画面やプリンタなど)、使用する座標系、描画の対象となる領域 (その部品が画面上で占める部分) や、図形を描く際に用いる色、文字を描く際のフォントなど、描画方法に関する情報を内部に記憶しています。このように、描画を行う対象を抽象化したものを、一般に、グラフィックスコンテキスト (graphics contexts) と呼びます。描画を行う際は、グラフィックスコンテキストを適切に設定した上で、そのグラフィックスコンテキストを使って直線、円 (楕円)、矩形、多角形、文字などの描画を行います。

`Graphics` クラスのインスタンスは次の表にあるようなインスタンスメソッドを持っています。グラフィックスコンテキストの設定を変更するメソッドと、実際に描画を行うメソッドの2種類が含まれていることに注意してください。この表に現れている `Color`、`Font` などのクラスは、すべて `java.awt` パッケージに属しています⁴。

Graphics クラス — 画面などへの描画機能を提供するグラフィックスコンテキスト

グラフィックスコンテキストの操作に関する主なインスタンスメソッド	
<code>Graphics create()</code>	自分自身 (グラフィックスコンテキスト) の複製を生成して戻す。
<code>void translate(int x, int y)</code>	新しい座標系の原点が現在の座標系の (x, y) の位置となるように座標系を平行移動する。

³実際には、`Graphics` のサブクラスである `Graphics2D` のさらにサブクラスのインスタンスが引数として渡されません。`Graphics2D` クラスのインスタンスは `Graphics` クラスよりも高度な描画機能を持っています。`paintComponent` の引数の型が `Graphics` と宣言されているのは `Swing` が登場する前に使用されていた `AWT` (Abstract Window Toolkit) との互換性を維持するためです。`paintComponent` に渡された引数は、キャスト演算子を用いて型変換し、`Graphics2D` クラスのオブジェクトとして用いることができます。

⁴これらのクラスや、`Graphics`、`Graphics2D` クラスの詳細については「Java Platform, Standard Edition 8 API仕様」(<https://docs.oracle.com/javase/jp/8/docs/api/>) を参照してください。

<code>void setColor(Color c)</code>	描画に使用する色を <code>c</code> に変更する。
<code>Color getColor()</code>	描画に使用する色を戻す。
<code>void setFont(Font f)</code>	文字の描画に使用するフォントを <code>f</code> に変更する。
<code>Font getFont()</code>	文字の描画に使用するフォントを戻す。
描画処理に関する主なインスタンスメソッド	
<code>void drawString(String s, int x, int y)</code>	(x, y) を起点 ⁵ として文字列 <code>s</code> を描く。
<code>void drawLine(int x1, int y1, int x2, int y2)</code>	$(x1, y1)$ から $(x2, y2)$ まで線分を描く。
<code>void drawRect(int x, int y, int w, int h)</code>	左上角が (x, y) で、幅 <code>w</code> 、高さ <code>h</code> の矩形を描く。
<code>void drawRoundRect(int x, int y, int w, int h, int aw, int ah)</code>	左上角が (x, y) で、幅 <code>w</code> 、高さ <code>h</code> の角の丸い矩形を描く。四隅の弧の横方向の直径が <code>aw</code> 、縦方向の直径が <code>ah</code> となる。
<code>void drawOval(int x, int y, int w, int h)</code>	左上角が (x, y) で、幅 <code>w</code> 、高さ <code>h</code> の矩形に内接する楕円を描く。
<code>void drawArc(int x, int y, int w, int h, int s, int e)</code>	左上角が (x, y) で、幅 <code>w</code> 、高さ <code>h</code> の矩形に内接する楕円弧の角度 <code>s</code> ° から <code>e</code> ° の部分を描く。角度は、 x 軸の正の向きを 0°とした反時計回りで、矩形の各頂点に向かう角度が 45°、135°、225°、315°となるように補正される。
<code>void drawPolyline(int[] x, int[] y, int n)</code>	$(x[0], y[0])$ 、 $(x[1], y[1])$ 、 \dots 、 $(x[n-1], y[n-1])$ を結ぶ折れ線を描く。
<code>void drawPolygon(int[] x, int[] y, int n)</code>	$(x[0], y[0])$ 、 $(x[1], y[1])$ 、 \dots 、 $(x[n-1], y[n-1])$ を頂点とする多角形を描く。
<code>void clearRect(int x, int y, int w, int h)</code>	左上角が (x, y) で、幅 <code>w</code> 、高さ <code>h</code> の矩形の領域をその部品の背景色 ⁶ で塗り潰す。
<code>void fillRect(int x, int y, int w, int h)</code>	<code>drawRect</code> の描く矩形の内部を塗り潰す。
<code>void fillRoundRect(int x, int y, int w, int h, int aw, int ah)</code>	<code>drawRoundRect</code> の描く矩形の内部を塗り潰す。
<code>void fillOval(int x, int y, int w, int h)</code>	<code>drawOval</code> の描く楕円の内部を塗り潰す。
<code>void fillArc(int x, int y, int w, int h, int s, int e)</code>	<code>drawArc</code> の描く楕円弧の内側 (扇形) を塗り潰す。
<code>void fillPolygon(int[] x, int[] y, int n)</code>	<code>drawPolygon</code> の描く多角形の内部を塗り潰す。

メモ

⁵日本語の文字の場合は、先頭文字の左下隅が (x, y) の位置となります。英文字の場合は、ペースラインの左端 (大文字の左下隅の位置) が (x, y) となります。

⁶ 部品の背景色は、`JComponent` のインスタンスメソッド `void setBackground(Color c)` で設定できます。

2.3 図形や文字の描画

次のプログラムでは、Card のサブクラスとして MyCard クラスを宣言し、そこで `paintComponent` を再定義することによって、カードの裏面に龍谷大学の古いロゴマークと「龍谷大学」という文字列を描くようにしています。

```
MyCard.java
1 import java.awt.*;
2 import jp.ac.ryukoku.math.cards.*;
3
4 class MyCard extends Card {
5     static Color[] logoColors = {
6         new Color(150, 0, 150),    // 紫色
7         new Color(0, 150, 0),     // 深緑色
8         new Color(0, 120, 200)    // 空色
9     };
10    static int[][] logoXs = {
11        {13, 13, 26, 26}, {43, 43, 56, 56}, {27, 57, 73, 42}
12    };
13    static int[][] logoYs = {
14        {37, 87, 87, 50}, {17, 50, 50, 17}, {50, 87, 87, 50}
15    };
16
17    MyCard() { super(); }
18    MyCard(int no) { super(no); }
19    MyCard(Suit s, Rank r) { super(s, r); }
20
21    protected void paintComponent(Graphics g) {
22        if (isShowingFace()) {
23            /* 表の面の描画はスーパークラスのまま */
24            super.paintComponent(g);
25            return;
26        }
27        /* 背面の描画を行う */
28        g.setColor(new Color(255, 255, 255)); // 白色
29        g.fillRoundRect(0, 0, 80, 120, 7, 7); // 角の丸い矩形
30        g.setColor(new Color(200, 200, 200)); // 明るい灰色
31        g.fillRoundRect(3, 3, 74, 114, 7, 7); // 角の丸い矩形
32        for (int i = 0; i < logoColors.length; i++) {
33            /* ロゴマークの3つの四角形を塗り潰す */
34            g.setColor(logoColors[i]);
35            g.fillPolygon(logoXs[i], logoYs[i], logoXs[i].length);
36        }
37        g.setColor(Color.BLACK); // 黒色
38        g.drawString("龍谷大学", 17, 108); // 文字列
39    }
40 }
```

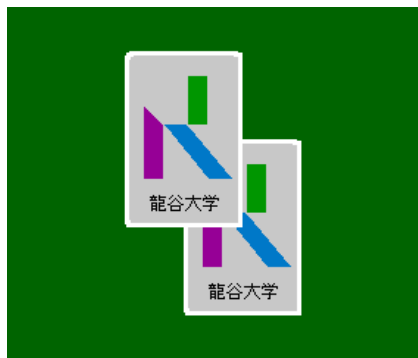
MyCard クラスの `paintComponent` では、カードが裏向きになっている場合のみ、独自の描画を行っています⁷。カードが表向きの場合は、`super.paintComponent(g);` を実行して、スーパークラスである Card の同じメソッドを起動することで、これまで通りにカードの表の面を描画しています。

次のプログラム G201.java では、この MyCard クラスのインスタンスを 2 つ生成し、ゲーム盤

⁷Card クラスのインスタンスメソッド `isShowingFace` は、現在、カードの表が描画されようとしている場合にのみ `true` を返します。

に追加していますが、このとき、MyCard クラスの `paintComponent` が起動されて、下の図のように2枚のカードが表示されます。

```
G201.java
1 import jp.ac.ryukoku.math.cards.*;
2
3 class G201 {
4     public static void main(String[] args) {
5         GameFrame f = new GameFrame();
6         f.add(new MyCard(Suit.HEARTS, Rank.KING), 400, 300);
7         f.add(new MyCard());
8     }
9 }
```



部品を描画する際の座標系 `drawPolygon` や `drawString` などのメソッドでは、描きたい多角形の頂点の座標や、文字列の起点の座標を引数に指定していますが、`paintComponent` に引数として渡されるグラフィックスコンテキスト (`Graphics` のインスタンス) では、その部品の左上角を原点として、右方向に x 軸、下方向に y 軸をとった座標系が設定されています。この座標系の長さの単位は、出力先となっている画面の画素 (ピクセル) です。また、このグラフィックスコンテキストで描画を行うことができるのは、その部品が占めている領域内だけで、その外部に対して描画を行うことはできません。

グラフィックスコンテキストに設定されている座標系は、`translate` メソッドなどを使って変更することもできます。ただし、座標系を変更してしまうと、それ以降の描画は新しい座標系に基づいて行われますので注意が必要です。一部分の描画だけ特定の座標系で行いたい場合は、`create` メソッドを使ってグラフィックスコンテキストの複製を生成し、その複製の座標系を変更して描画に用いることもできます。

メモ

2.4 色の表現 (Color クラス)

JFC では、描画に用いる色を `java.awt.Color` クラスのインスタンスで表現します。MyCard クラスでも、`new Color(150, 0, 150)` というインスタンス生成式や、`Color.BLACK` というクラス変数に代入されている値で、このクラスのオブジェクトを利用していました。Color クラスでは、次のようなコンストラクタやメソッドなどが提供されています。Color クラスのインスタンスは、一旦生成されると、その表現している色を変えることのできない不変な (immutable) オブジェクトとなります。

Color クラス — 色を表現する

主なコンストラクタ <code>Color(int r, int g, int b)</code> <code>Color(int r, int g, int b, int a)</code> <code>Color(float r, float g, float b)</code> <code>Color(float r, float g, float b, float a)</code>	<p>光の3原色 (赤、緑、青) の各成分の値を0から255までの整数値で表したとき、(r, g, b) となる色。</p> <p>光の3原色 (赤、緑、青) の各成分とアルファ成分 (不透明度) の値を0から255までの整数値で表したとき、(r, g, b, a) となる色。</p> <p>光の3原色 (赤、緑、青) の各成分の値を0.0から1.0までの実数値で表したとき、(r, g, b) となる色。</p> <p>光の3原色 (赤、緑、青) の各成分とアルファ成分 (不透明度) の値を光の3原色 (赤、緑、青) の各成分の値を0.0から1.0までの実数値で表したとき、(r, g, b, a) となる色。</p>
主なクラス変数 <code>static final Color BLACK</code> <code>static final Color DARK_GRAY</code> <code>static final Color GRAY</code> <code>static final Color LIGHT_GRAY</code> <code>static final Color WHITE</code> <code>static final Color RED</code> <code>static final Color GREEN</code> <code>static final Color BLUE</code> <code>static final Color YELLOW</code> <code>static final Color CYAN</code> <code>static final Color MAGENTA</code> <code>static final Color PINK</code> <code>static final Color ORANGE</code>	黒 ダークグレー 灰色 ライトグレー 白 赤 緑 青 黄 シアン マゼンタ ピンク オレンジ
主なクラスメソッド <code>static Color getHSBColor(float h, float s, float b)</code>	色相、彩度、明度を、それぞれ0.0から1.0までの実数値で表したとき、(h, s, b) となる色を生成して戻す。
主なインスタンスメソッド <code>Color brighter()</code> <code>Color darker()</code> <code>int getRed()</code> <code>int getGreen()</code> <code>int getBlue()</code> <code>int getAlpha()</code>	<p>この色をより明るくした色を戻す。</p> <p>この色をより暗くした色を戻す。</p> <p>この色の赤成分を0から255までの整数値で戻す。</p> <p>この色の緑成分を0から255までの整数値で戻す。</p> <p>この色の青成分を0から255までの整数値で戻す。</p> <p>この色のアルファ成分 (不透明度) を0から255までの整数値で戻す。</p>

Color クラスのインスタンスで特定の色を表す最も基本的な方法は、光の3原色である RGB (赤、緑、青) の各成分の強さ (r, g, b) を引数として、Color クラスのコンストラクタを呼び出すことです。3つの引数には、光の3原色の各成分の強さを指定しますが、int 型で指定する場合は0から255までの整数で、float 型で指定するときは、0.0から1.0までの実数値で指定します。たとえば、`new Color(0, 0, 0)` や `new Color(0.0f, 0.0f, 0.0f)` で黒色を、`new Color(255, 255, 0)` や `new Color(1.0f, 1.0f, 0.0f)` で黄色を表すオブジェクトを生成できます。また、表にあるように、よく使われる色については、あらかじめクラス変数にその色を表すインスタンスが代入されていますので、その値を使用することもできます。

メモ

アルファ値 Color クラスのインスタンスは、透明度を持った色を表現することもできます。`new Color(255, 255, 0)` のように、RGB (赤、緑、青の各成分) の値だけを指定して生成した色は全く透明度を持たない色となり、描画先の画素は描画色で置き換えられて、下地の色(その画素の元の色)は全く見えなくなってしまいます。これに対して、透明度を持った色で描画を行うと、画素の色は、元の色と描画色との中間の色となります。

描画色が、画素の色を置き換える割合のことを、**アルファ値**と呼びます。通常、アルファ値は、0.0から1.0までの数値をとり、この値を α で表すと、描画先の画素の(各成分の)値は

$$\text{その画素の新しい値} = \alpha \times \text{描画色の値} + (1 - \alpha) \times \text{その画素の元の色}$$

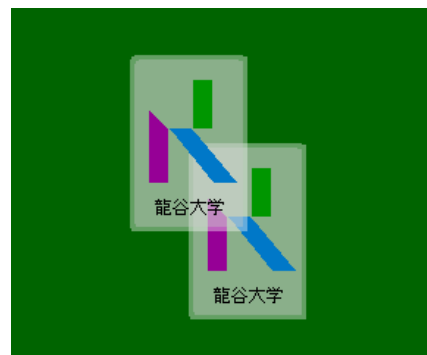
のようになります。アルファ値が1.0であれば(透明度は0で)描画先の画素値は完全に上書きされてしまいます。また、0.0であれば(完全に透明で)その色のRGBの各成分の値にかかわらず、描画先の画素値は変更されません。アルファ値は、0から255までなど、ある範囲の整数値で表すこともあります⁸。

透明度を持ったColorクラスのインスタンスを生成した場合は、1.0未満(int型の場合は255未満)のアルファ値を第4引数に指定してコンストラクタを呼び出します。たとえば、MyCard.javaの28行目と30行目を、それぞれ

```
// 透明度を持った白  
g.setColor(new Color(255, 255, 255, 100));
```

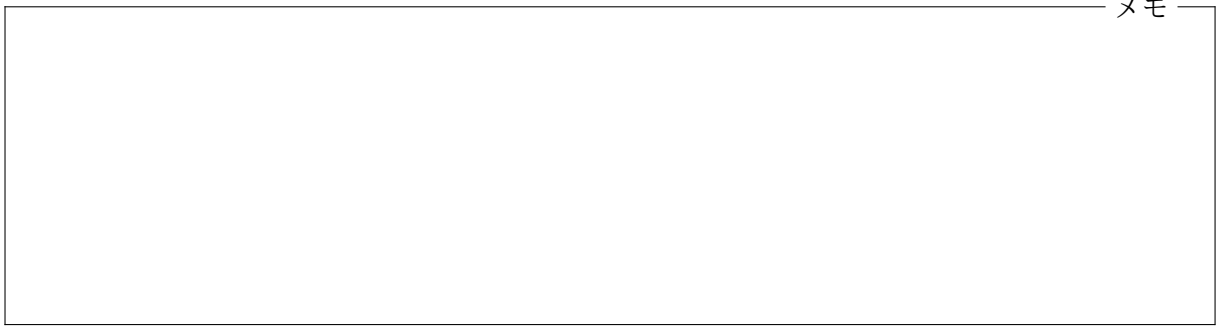
と

```
// 透明度を持った明るい灰色  
g.setColor(new Color(200, 200, 200, 100));
```



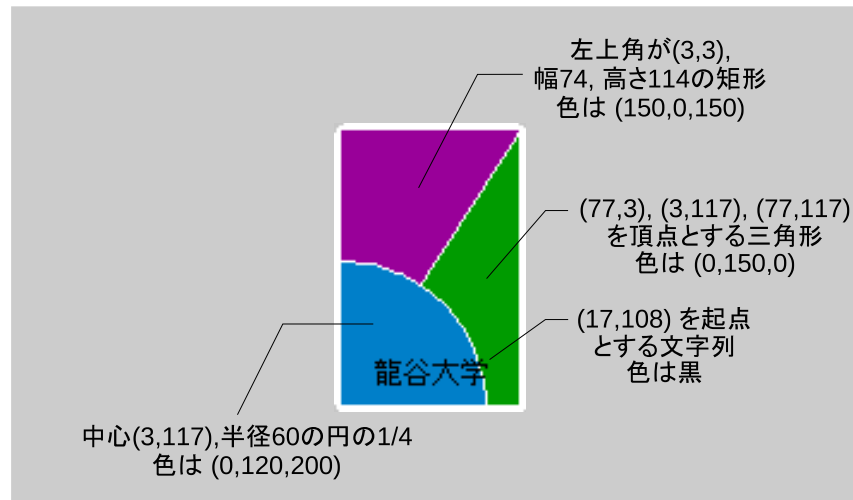
⁸その場合は、とりうる最大の値を1.0に換算します。

のように変更すると、カードの背面は右上の図のよう描かれ、ゲーム盤や下になっているカードが一部透けて見えるようになります。



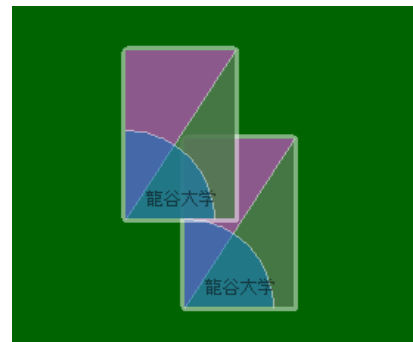
2.5 演習問題

1. Card のサブクラス G204Card を宣言した G204Card.java というソースファイルを作成し、このクラスの paintComponent メソッドを再定義して、カードの背面を次の図のように描くようなメソッドとしなさい。



MyCard.java と同様に、まず、角が7ピクセル分だけ丸くなった、幅80、高さ120の白い矩形が描かれ、これがカードの全体となります。その中に矩形が描かれ、その上に三角形が、さらにその上に円の一部分が描かれています。カードの縁や3つの部分の境界の色は白です。

2. G205Card.java というソースファイルに Card のサブクラス G205Card を宣言し、paintComponent メソッドを再定義して、カードの背面を描く際に使用する色を、すべて半透明(不透明度50%)として、G204Card と同じ図形を描くようにしなさい。例えば、次の G202.java を実行すると、右の図のようにカードの背面が描かれます。



— G202.java —

```

1 import jp.ac.ryukoku.math.cards.*;
2
3 class G202 {
4     public static void main(String[] args) {
5         GameFrame f = new GameFrame();
6         f.add(new G205Card(Suit.HEARTS, Rank.KING), 400, 300);
7         f.add(new G205Card());
8     }
9 }

```

3. Deck クラスには

```

protected Card makeCard(int no) {
    return new Card(no);
}

```

というインスタンスメソッドの宣言が含まれており、デッキを構成するカードは、このメソッドを起動して生成しています。Deck クラスのサブクラスとして、MyDeck クラスを宣言し、そこで makeCard メソッドを再定義することで、MyCard のインスタンスから構成されるデッキを作成できるようにして下さい。MyDeck クラスは MyDeck.java というソースファイルに宣言し、次の2つのコンストラクタを提供するようにして下さい。

MyDeck クラス — MyCard で構成されるデッキ

コンストラクタ	
MyDeck()	ジョーカーを含まないデッキ
MyDeck(int n)	ジョーカーを n 枚含むデッキ (n = 0, 1, 2)

MyDeck.java が完成したら、次のプログラム G203.java で正しく動作するかどうかをテストして下さい。

```

1 import jp.ac.ryukoku.math.cards.*;
2
3 class G203 {
4     public static void main(String[] args) {
5         GameFrame f = new GameFrame();
6         Deck[] decks
7             = { new MyDeck(), new Deck(), new MyDeck(1) };
8         int x = 200;
9         for (Deck d : decks) {
10            f.add(d, x, 120);
11            d.shuffle();
12            x += 160;
13        }
14        x = 80;
15        for (int i = 0; i < 5; i++) {
16            for (Deck d : decks) {
17                Card c = d.pickUp();
18                c.moveTo(x, 400);
19                if (i % 2 == 0) {
20                    c.flip();
21                }
22                x += 40;
23            }

```

```
24     }  
25     }  
26 }
```

G203.java を実行すると、最終的には次の図のような状態になります。

