グラフィックス及び演習・第4回

2025年7月7日

配布資料の内容

4.1 ビットマップ画像の描画 4–1
4.2 画像ファイルの読み込み 4–3
4.3 クラス初期化子 4–6
4.4 例外のスローとキャッチ 4–7
4.5 演習問題 4–10

4.1 ビットマップ画像の描画

たくさんの画素の集まりとして表現された画像を、一般に**ビットマップ画像**と呼びます。ビットマップ画像は、本来、白黒画像の画素の白と黒を0と1で区別して、このビットを画像全体を構成している画素の数だけ並べたデータのことを意味していました。現在では、様々な色を表現することができるように、1つ画素に関する情報[1]が数 bit から数十 bit の大きさとなっていますが、このような場合でも慣習的にビットマップ画像と呼ばれています[2]。第2回では、`Graphics` クラスを用いて、直線や多角形、楕円などの基本的な図形と文字列の描画について勉強しましたが、今回は、このようなビットマップ画像を扱う方法を紹介します。

メモ

`Image` クラス　Java Foundation Classes (JFC) には、ビットマップ画像を表現するためのクラスとして `java.awt.Image` というクラスが用意されています。このクラスのインスタンスは直接生成することはできませんが、JFC のいくつかのクラスに、`Image` のサブクラスのインスタンスを生成したり取得したりする機能が備わっていますので、これを利用して `Image` 型のオブジェクトを作成することが可能です。たとえば、PNG、GIF、JPEG などの形式のビットマップ画像ファイルがあれば、そのファイルの内容を読み込んで `Image` 型のオブジェクトを作成することができますし、GUI の部品から、その部品が表示される画面の代りに描画先として使用できる (単なる画用紙としての) `Image` 型のオブジェクトを生成したりすることができます。

`Image` クラス — ビットマップ画像

主なクラス変数	
`static final int SCALE_DEFAULT`	標準の拡大縮小アルゴリズムを表す
`static final int SCALE_FAST`	速度優先の拡大縮小アルゴリズムを表す
`static final int SCALE_SMOOTH`	画質優先の拡大縮小アルゴリズムを表す

[1] 第2回で説明したように、描画の際に用いられる不透明度 (アルファ値) を含むこともあります。

[2] この違和感を排除するために、ピクセルマップ画像と呼ぶ場合もあります。

主なインスタンスメソッド	
`Graphics getGraphics()`	この画像を描画先とするグラフィックスコンテクストを戻す
`int getHeight(ImageObserver o)`	この画像の高さを戻す
`int getWidth(ImageObserver o)`	この画像の幅を戻す
`Image getScaledInstance (int w, int h, int hints)`	`hints` で指定したアルゴリズムを用いて、この画像を幅 `w`、高さ `h` に拡大縮小した画像を戻す。

Image 型のオブジェクトは、Graphics クラスの次のようなインスタンスメソッドを使って画面などに描画することができます。

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クスト

画像の描画処理に関する主なインスタンスメソッド[3]	
`boolean drawImage(Image im, int x, int y, ImageObserver o)`	画像 `im` を、その左上隅が (`x`, `y`) の位置となるように描画する
`boolean drawImage(Image im, int x, int y, int w, int h, ImageObserver o)`	画像 `im` を、その左上隅が (`x`, `y`) の位置となるように、また、幅 が `w`、高さが `h` となるように拡大縮小して描画する
`boolean drawImage(Image im, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver o)`	対角線が、点 (`dx1`, `dy1`) と点 (`dx2`, `dy2`) を結ぶ線分となるような矩形内を描画先として、画像 `im` 内の点 (`sx1`, `sy1`) が点 (`dx1`, `dy1`) へ、点 (`sx2`, `sy2`) が点 (`dx2`, `dy2`) へ対応するように (画像を拡大、縮小、左右/上下反転、平行移動して) 描画する

メモ

ImageObserver　これらの drawImage メソッドの最後の引数の型となっている ImageObserver は java.awt.image パッケージで宣言されているインタフェースです。ImageObserver では、ネットワーク上に保存されているものなど、その内容を取得するのに時間が掛ってしまうような画像データに対して、その取得状況に合わせた処理[4]を行うための抽象メソッドが宣言されています。

JFC の GUI 部品 (JComponent[5]のサブクラス) はすべて、ImageObserver を実装していて、画像データの取得状況に合わせてその部品を再描画する処理を行うことが可能ですので、その部品の paintComponent メソッドから drawImage メソッドを起動する場合は、最後の引数を this にし

[3]これら 3 つのメソッドの変種として、最後の引数 o の前に追加された引数で、画像 im 内の透明な画素の部分に透けて見えるの背景色 (java.awt.Color 型) を指定できるメソッドも用意されています。すべての drawImage メソッドは、指定した画像をすべて描画できたかどうかを boolean 型の戻り値として返します。画像データの一部しか取得できていない場合などは、画像の一部だけが描画され、false が返されます。

[4]たとえば、画像全体のデータ内で、取得できた部分が増える度に、画像を描画し直すなどの処理をこのメソッドで行うことができます。

[5]Swing のすべての部品のスーパークラス

ておくと安全です。ただし、描画したい画像データの取得がすでに終わっているのなら、このような配慮は必要ありません。このような場合、最後の引数は null として構いません。

4.2 画像ファイルの読み込み

ビットマップ画像のデータを格納したファイルから Image 型のオブジェクトを生成する方法はいくつかありますが、今回はその内の 2 つの方法を紹介します。

ImageIcon クラスを利用した画像ファイルの読み込み　ビットマップ画像ファイルを読み込む最も簡単は方法は、javax.swing.ImageIcon という、Swing の GUI 部品で用いられるアイコンを表現するためのクラスを利用する方法です。Swing では、アイコンを描画することのできるオブジェクトの資格を javax.swing.Icon インタフェースとして宣言しています。ImageIcon は、このインタフェースを実装したクラスで、画像データを用いてアイコンを描画します[6]。

ImageIcon クラスには、画像ファイル名や、画像ファイルの URL などを引数とするコンストラクタが用意されていますので、これを用いて簡単にインスタンスを生成できます。ImageIcon のインスタンスは、直接 Image として働くことはできませんが、そのアイコンから getImage というインスタンスメソッドを起動して、Image 型のオブジェクトを取得することができますので、これを描画などに使うことができます。

ImageIcon クラス — 画像データを用いたアイコン描画機能

主なコンストラクタ	
`ImageIcon(String file)`	file で指定したファイル名の画像ファイルを読み込んでアイコンを生成
`ImageIcon(java.net.URL url)`	url で指定した URL の画像ファイルを読み込んでアイコンを生成
主なインスタンスメソッド	
`Image getImage()`	アイコン画像を Image 型のオブジェクトとして戻す
`int getImageLoadStatus()`	画像の読み込み状態を戻す
`void paintIcon(Component[7] c, Graphics g, int x, int y)`	グラフィックスコンテクスト g の (x, y) の位置に、このアイコンを描画する。その際、このアイコンに ImageObserver が設定されていない場合は、変わりに c を使用する
`void setImageObserver (ImageObserver o)`	このアイコンの ImageObserver を o に設定する

メモ

[6] Icon インタフェースでは、アイコンを描画できさえすればよいので、画像ファイルを用いないで、このインタフェースを実装することもできます。

[7] JComponent のスーパークラスで、Swing の前身である AWT を含めて、JFC のすべての GUI 部品のスーパークラスとなっています。ImageObserver は Component クラスで実装されています。

ImageIcon クラスのコンストラクタは、PNG、GIF、JPEG 形式の画像ファイルを読み込むことができ、画像データの読み込みが完了して、初めてインスタンスが返されますので、getImage で取得した Image オブジェクトを使って描画を行う際に、ImageObserver を設定する必要はありません。正しく画像データが読み込めたかどうかは、getImageLoadStatus メソッドの戻り値で判定できます。その値は、java.awt.MediaTracker クラスで宣言されている次の 3 つの定数の値のいずれかとなります。

MediaTracker クラス — 画像や動画、音声データなどの読み込み状態を監視するオブジェクト

主なクラス変数[8]	
static final int ABORTED	読み込みが途中で打ち切られた
static final int ERRORED	読み込みが途中で障害が発生した
static final int COMPLETE	読み込みが正常に終了した

メモ

ビットマップ画像の読み込みと描画の例　次のプログラム G401Card.java では、Card クラスのサブクラス G401Card クラスが宣言されています。6 行目では、ImageIcon クラスのコンストラクタで card.png という画像ファイルを読み込んでアイコンを生成し、そこから Image 型のオブジェクトを取得して、クラス変数 img を初期化しています。このクラスの paintComponent メソッドでは、このクラス変数に記憶しておいた画像をカードの背面に描画しています (25 行目)。この行で起動されている getWidth と getHeight は、すべての GUI 部品のクラスに備わっているインスタンスメソッドで、それぞれ部品の幅と高さを int 型の戻り値として返します[9]。

G401Card.java

```
 1 import java.awt.*;
 2 import javax.swing.ImageIcon;
 3 import jp.ac.ryukoku.math.cards.*;
 4
 5 class G401Card extends Card {
 6     static Image img = new ImageIcon("card.png").getImage();
 7
 8     G401Card() {
 9         super();
10     }
11
12     G401Card(int no) {
13         super(no);
14     }
15
```

[8] これら 3 つの他に、読み込みの途中であることを示す static final int LOADING がありますが、ImageIcon クラスの getImageLoadStatus の戻り値となることはありません。

[9] カードの本来の大きさは 80 × 120 ピクセルですが、カードを裏返している途中などでは、画面上の GUI 部品としての高さが 120 よりも小さくなっている場合があります。

```
16      G401Card(Suit suit, Rank rank) {
17          super(suit, rank);
18      }
19
20      protected void paintComponent(Graphics g) {
21          if (isShowingFace()) {
22              super.paintComponent(g);
23              return;
24          }
25          g.drawImage(img, 0, 0, getWidth(), getHeight(), null);
26      }
27
28      public static void main(String[] args) {
29          GameFrame f = new GameFrame();
30          f.add(new G401Card());
31      }
32  }
```

このプログラムを実行するためには、カレントディレクトリに card.png という名前の PNG 形式の画像ファイルが存在する必要があります。このファイルが読み込めない場合は、カードの背面は描画されません。

メモ

ImageIO クラスを利用した画像ファイルの読み込み　画像ファイルは javax.imageio.ImageIO クラスのクラスメソッド read を利用しても読み込むこともできます。read メソッドの戻り値の型 BufferedImage は、Image のサブクラスで、メモリ中に展開された画像データを表すオブジェクトのクラスです。

ImageIO クラス — 画像の入出力を行う

主なクラスメソッド	
static BufferedImage read(java.io.File file)	file で指定した画像ファイルのデータを読み込んで得られた BufferedImage を戻す
static BufferedImage read(java.net.URL url)	url で指定した URL の画像ファイルのデータを読み込んで得られた BufferedImage を戻す

ImageIO クラスの read メソッドを用いると G401Card.java を次のように書き換えることができます。この方法では、PNG、GIF、JPEG 形式に加えて、Windows で使用されている BMP 形式の画像ファイルを読み込むこともできます。

ImageIO を利用した G401Card.java

```
 1 import java.awt.*;
 2 import java.io.*;
 3 import javax.imageio.ImageIO;
 4 import jp.ac.ryukoku.math.cards.*;
```

```
 5
 6 class G401Card extends Card {
 7     static final String IMAGE_FILE_NAME = "card.png";
 8     static Image img;
 9
10     /* クラス初期化子 */
11     static {
12         try {
13             img = ImageIO.read(new File(IMAGE_FILE_NAME));
14         } catch (IOException e) {
15             System.err.println(e.getMessage());
16             System.err.println("画像ファイル "
17                     + IMAGE_FILE_NAME + " を読み込めませんでした");
18             System.exit(1);
19         }
20     }
21     ⋮
```

メモ

4.3 クラス初期化子

ImageIO を利用するように書き換えた G401Card.java の 11 ～ 20 行目は、クラス初期化子と呼ばれるもので、クラス宣言内に次のような書式で宣言される手続きです。

```
static {
    文の並び
}
```

クラス初期化子は、プログラムの起動後、そのクラスが使用される前に 1 度だけ実行されます。

プログラムの 13 行目で、ImageIO のクラスメソッド read を、java.io.File のインスタンスを引数として起動して画像ファイルを読み込み、その戻り値でクラス変数 img を初期化しています。

```
new File(ファイル名)
```

というインスタンス生成式で ファイル名 で指定したファイルを表す File クラスのインスタンスを生成していますが、read メソッドがそのファイルを読み込めない場合は、java.io.IOException という例外が発生します[10]。12 ～ 19 行目では、この例外を処理するために try 文と呼ばれる構文が使われています。このプログラムが card.png という画像ファイルの読み込みに失敗すると、

```
Can't read input file!
画像ファイル card.png を読み込めませんでした
```

[10] ファイルが存在しない場合でも、File クラスのインスタンスは生成されます。

ようなエラーメッセージが表示されて、プログラムの実行が終了します。

4.4 例外のスローとキャッチ

プラグラムの実行中に発生する予期しない出来事を**例外 (exception)** と呼びます。Java では、プログラムの実行中に例外が発生すると、発生した例外に関する情報を `java.lang.Throwable` というクラスのサブクラスのオブジェクトとして表現し、その発生をプログラムに伝えます。このことを、例外が**スロー (throw)** されると言います。

発生した例外の種類によって、その例外を表現するクラス (`Throwable` のサブクラス) は異なり、次のようなクラスの階層 (スーパークラス、サブクラスの関係) として分類されています[11]。

- `Throwable` ···· すべての例外のスーパークラス
  - `Error` ···· JVM が正常に実行できない等、プログラム側では対処できない類の例外
  - `Exception` ···· プログラム側で何らかの対処が可能な例外
    - `RuntimeException` ···· Java プログラムの実行に関する例外
      - `NullPointerException` ···· null 参照を介してインスタンス変数にアクセスしたり、インスタンスメソッドを起動した
      - `ArrayIndexOutOfBoundsException` ···· 許される範囲を越える配列の添字が使われた
      - `NumberFormatException` ···· 文字列から数値への変換に失敗した (`Integer` クラスの `parseInt` メソッドなど)
      - `ClassCastException` ···· オブジェクトを、そのオブジェクトを含まない型にキャスト (型変換) しようとした
      - `ArrayStoreException` ···· 型の合わないオブジェクトを配列の要素に代入しようとした
      - `IllegalArgumentException` ···· 不正な引数でメソッドやコンストラクタが起動された
    - `InterruptedException` ···· このスレッドに対する割り込みが発生した
    - `java.io.IOException` ···· 入出力に関するエラーが発生した

メモ

`throw` 文　`NullPointerException` や `ArrayIndexOutOfBoundsException` のように、Java 仮想機械 (JVM) が例外をスローすることもありますが、Java のアプリケーションプログラム自身や、それが利用しているクラスライブラリから例外がスローされることもあります。Java プログラム自身が予期しない事態に遭遇して、それを例外としてスローしたい場合は、

`throw` `Throwable 型の式` `;`

という書式の **`throw` 文** を用います。たとえば、

[11] パッケージ名が示されていないクラスは、すべて `java.lang` パッケージに属します。

```
throw new IllegalArgumentException("引数が不正です");
```

のように Throwable のサブクラスのインスタンスを生成して、これを例外としてスローすることができます。

この例で使ったように、Java 標準の例外を表すクラス (Throwable のサブクラス) には、その例外の詳細を示す文字列を String 型の引数で受け取るコンストラクタが用意されています。この文字列は、生成された例外オブジェクトに対して、

```
String getMessage()
```

という Throwable クラスのインスタンスメソッドを起動して取得することができます。

例外としてスローするオブジェクトのクラスは、Throwable のサブクラスであればよいので、自分で例外のクラス宣言して、そのインスタンスをスローすることもできます。

メモ

throws 節　メソッドやコンストラクタの起動中に例外がスローされる可能性がある場合は、そのメソッド (やコンストラクタ) を宣言する際に、

```
戻り値の型名 メソッド名 ( 仮引数宣言の列 ) throws 例外のクラス名の列[12] {
    メソッド本体の文の並び
}
```

のように、(仮引数宣言の列) に続いて **throws 節**と呼ばれる書式を書いて、そのメソッド (やコンストラクタ) の起動によってスローされる可能性のある例外 (のクラス名) を宣言しておくことができます[13]。この宣言によって、そのメソッドやコンストラクタを起動したときに起りうる例外をプログラマが知ることができます。

try 文　スローされた例外を検知することを、例外を **キャッチ (catch)** すると言います。また、例外の発生に対処するための処理を、一般に**例外処理**と呼びます。Java プログラムでは、try 文と呼ばれる次のような書式の構文を用いて例外処理を行うことが可能です。

```
try {
    例外の発生を検知したい文の並び
}
catch ( 例外のクラス名 例外を受け取る変数名 ) {
    この種類の例外を処理する文の並び
}
```

[12] 複数ある場合は「,」で区切って並べます。

[13] 後述の検査例外以外の例外を throws 節に書く必要は通常ありません。

try 文では、まず 例外の発生を検知したい文の並び の部分が実行されます。この部分で例外が発生しなかった場合[14]は、これで try 文の実行は終了します。一方、この部分の実行の途中で例外が発生すると、そのとき実行されていたプログラム (メソッド宣言の本体など) は途中で終了し、例外処理が行われます。

メモ

catch 節　try 文の catch (···) { ··· } の部分を catch 節と呼び、例外処理の手続きはこの部分に記述します。発生した例外が、catch 節の 例外のクラス名 の型に含まれている場合は、その例外は、この catch 節でキャッチされます。例外を表すオブジェクトが 例外を受け取る変数名 に代入された上で、この種類の例外を処理する文の並び が実行されます。これが例外処理です。

一方、発生した例外が 例外のクラス名 の型に含まれていない場合は、別の catch 節に任されます。1 つの try 文に、複数の catch 節を書くこともできますので、発生する可能性のある例外が複数ある場合は、それぞれ別の catch 節で、その例外処理を行うことができます。どの catch 節でもキャッチされなかった例外は、さらに外側の try 文の catch 節でキャッチされます。

例外は、例外の発生を検知したい文の並び の部分から起動されたメソッドや、さらにそこから起動されたメソッドの中など、try 文から間接的に呼び出された手続きの中でも発生する可能性がありますが、このような例外も catch 節でキャッチすることができます。catch 節の実行が終わると、try 文 の実行はそこで終了し、例外をスローした部分のプログラムに戻ることはありません。

ImageIO クラスを用いて書き換えた G401Card.java では、ここで、エラーメッセージを出力し、プログラム全体を終了させていました (15 〜 18 行目)。

メモ

finally 節　try 文では、最後の catch 節に続いて、

```
finally {
      try 文 を抜ける際に必ず実行する文の並び
}
```

という finally 節と呼ばれる書式を書くこともできます[15]。finally 節に書かれた文の並びは、

[14] あるいは、この中ですでに例外がキャッチされている場合

[15] catch 節を書かないで、finally 節だけを書くこともできます。

例外の発生の有無に関わらず、try 文 を抜ける際に必ず実行されます[16]。これは、この try 文の catch 節でキャッチされない例外が発生した場合もそうです。try 文の外側へ例外の発生が伝わる前に、finally 節に書かれた文の並びが実行されます[17]。

検査例外　Java では、Error や RuntimeException に含まれない例外を、**検査例外**(あるいはチェック例外) と呼びます。たとえば、java.io.IOException は検査例外です。検査例外は、throw 文や、その例外を throws 節に宣言したメソッドやコンストラクタを起動したときのみに発生する可能性があります。Java では、検査例外をスローする可能性のあるメソッドやコンストラクタを起動する場合は、それらを起動する側で、必ず (try 文を使って) 例外処理を行わなければなりません。ただし、(別の) メソッドやコンストラクタからそのような起動を行っている場合は、その (別の) メソッドやコンストラクタを throws 節付きで宣言する選択肢もあります。その場合、その (別の) メソッドやコンストラクタを起動しているプログラムでの例外処理が必要となります。

メモ

4.5　演習問題

1. 4 ページの (ImageIcon を利用した) G401Card.java というプログラムを G402Card.java にコピーして、カードの背面に、card.png の画像を左右反転した形 (鏡像) で描画するように変更しなさい。

   画像を左右反転させて描画するには、2 ページで紹介した Graphics クラスが提供している 3 つの drawImage メソッドの内、最後の (引数が 10 個の) ものを利用します。画像ファイルに記録されているビットマップ画像の大きさが変わっても、その全体が表示されるようにしてください。Image クラスの getWidth や getHeight メソッドで、読み込んだ画像の幅や高さを取得することができます。

[16] catch 節で例外がキャッチされた場合は、その catch 節の実行が終了した後に、finally 節が実行されます。また、break 文や return 文等が実行されて try 文を途中で抜ける場合でも finally 節が実行されます。

[17] この場合、finally 節の中で新たな例外が発生すると、元の例外は無視されて、新しく発生した例外のみが (try 文の外側へ) スローされます。

2. 4 ページの G401Card.java を、G403Card.java にコピーし、画像ファイル card.png の読み込みに失敗した場合は、カードの背面を Card クラスと同じように描画するプログラムに変更しなさい。画像ファイルの読み込みができたかどうかは、ImageIcon のインスタンスメソッド getImageLoadStatus の戻り値で判定できます。

3. 4 ページの G401Card.java を、G404Card.java にコピーし、ImageIO クラスの read メソッドを使って画像ファイルを読み込む (5 ページのプログラムの) ように変更しなさい。画像ファイルの読み込みに失敗した場合は、「画像ファイル ... を読み込めませんでした」というメッセージを標準エラー出力に出力すること。

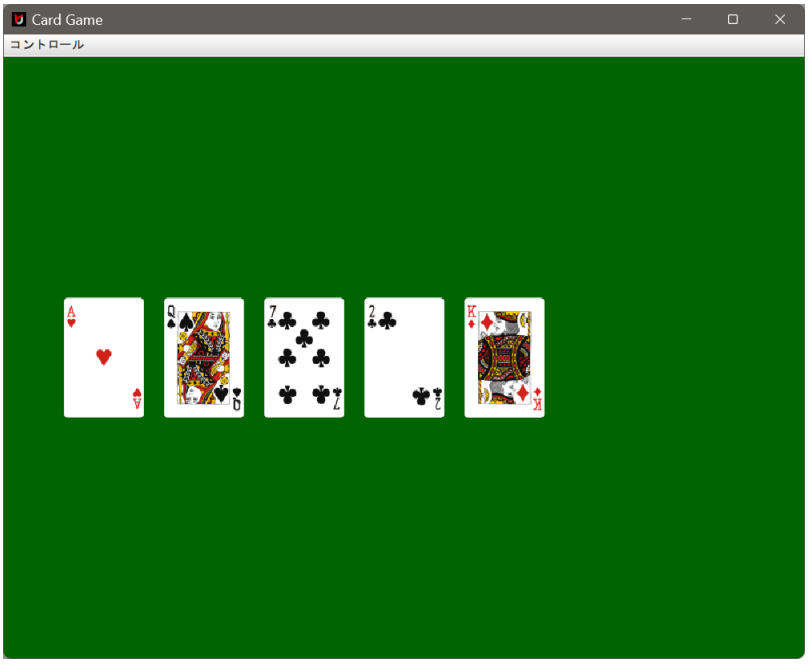
4. 次のプログラム G405.java はコマンドライン引数を使って

Windows (Powershell)
```
... OOProg> java -cp "bin;lib\cards.jar" G405 H1 S12 C7 C2 D13
```

や

macOS (zsh)
```
... OOProg % java -cp "bin:lib/cards.jar" G405 H1 S12 C7 C2 D13
```

のように起動すると[18]、次の図のように、コマンドライン引数で指定したカードを画面に並べて表示するプログラムです。



G405.java
```
import jp.ac.ryukoku.math.cards.*;

public class G405 {
    public static void main(String[] args) {
        GameFrame f = new GameFrame();
        int x = 60;
```

[18] G405.java をコンパイルしてできるクラスファイルは bin ディレクトリにあるものとしています。

```
        for (String s : args) {
            int suit = "SHDC".indexOf(s.substring(0, 1)) + 1;
            int rank = Integer.parseInt(s.substring(1));
            if (suit < 1 || 4 < suit || rank < 1 || 13 < rank) {
                continue;
            }
            Card c = new Card(Suit.suitOf(suit), Rank.rankOf(rank));
            f.add(c, x, 240);
            c.faceUp();
            x += 100;
        }
    }
}
```

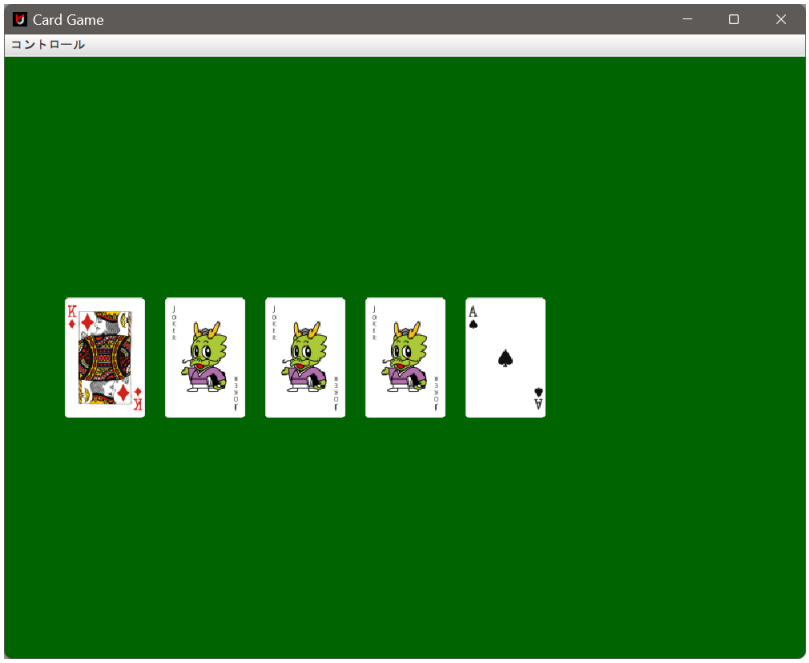
このプログラムを、コマンドラインの引数の書式が正しくなかった場合にはジョーカーが指定されたと解釈するように変更しなさい。変更後のプログラムでは、たとえば

Windows (Powershell)
```
... OOProg> java -cp "bin;lib\cards.jar" G405 D13 A8 HK C100 S1
```

や

macOS (zsh)
```
... OOProg % java -cp "bin:lib/cards.jar" G405 D13 A8 HK C100 S1
```

のようなコマンドラインで起動すると、次のような5枚のカードが表示されます。



カードのランクを表す1～13の書式が正しくない場合は、Integer クラスの parseInt メソッドは NumberFormatException をスローしますので、これを try 文の catch 節でキャッチしましょう。

グラフィックス及び演習・第4回・終わり