オブジェクト指向及び演習・第 2 回

2024 年 4 月 22 日

今回の内容

2.1 クラスとインスタンス

前回の授業では、オブジェクトとはソフトウェア的な部品であるということを説明しましたが、部品と言っても、ソースプログラムの一部分がオブジェクトなのではありません。オブジェクトは、動いているプログラムの中に存在するもので、プログラムが起動した後、プログラム中に書かれた指示によって生成され、プログラムの記述に従って仕事をします。

1 つのプログラムが働くためには、いろいろな種類のオブジェクトが必要になりますが、そのオブジェクトの種類のことを**クラス (class)**と呼びます。オブジェクト指向言語である Java では、ソースプログラムの中に

1. それぞれのクラスのオブジェクトがどのようなものであるかの定義
2. 特定のクラスのオブジェクトを生成する指示
3. 生成したオブジェクトへの仕事の指示

を記述します。Java では、1 を**クラス宣言**と呼ばれる書式[1]で書きます。クラス宣言は、その種類のオブジェクトの設計図に相当するものです。あるクラスのオブジェクトのことを、そのクラスの**インスタンス (instance)** と呼びます。同じクラスのインスタンスは、同じ設計図で作られたものですから、基本的にはどれも同じような働きをすることができます。

2 のオブジェクトの生成や、3 のオブジェクトへの仕事の依頼も、結局は、あるクラスのあるインスタンスが行う仕事の一部になりますので、これらを行う指示の記述も、あるクラスのクラス宣言の一部に現れることになります。このため、1 つの Java のプログラムは、いくつかのクラス宣言の集まりとして構成されます。

[1]前回紹介した例題のプログラムに出てきた `class ... { ... }` の形がクラス宣言です。ただし、これをオブジェクトの設計図として使ったのは `P103TicTacToe.java` だけでした。

Java のプログラミングでは、いろいろなクラスを自分で宣言 (定義) することになりますが、この科目では、まずその準備として、すでに用意 (定義) されたクラスを利用して、そのクラスのオブジェクトを生成し、生成されたオブジェクトに仕事を依頼することから始めます。この中で、オブジェクトとはどのようなものなのかを理解していきたいと思います。

メモ

2.2 Java アプリケーションの実行開始

C 言語では、main 関数が呼ばれることでアプリケーションプログラムの実行が始まりますが、Java では、クラス宣言の一部として定義された main という名前の**クラスメソッド**[2]が起動されることで始まります。クラスメソッドは、C 言語における「関数」に相当するもので、この科目では第 6 回に詳しく勉強することになります。また、C 言語では「関数を呼び出す」と言いますが、Java では「メソッドを起動する」と言います[3]。これらは、単なる用語の違いであって、どちらも同じことを意味していると考えてください。

用語の違い

C 言語	関数	呼び出す
Java 言語	メソッド	起動する (呼び出す)

前回紹介した P101Hello.java では、P101Hello と名付けられたクラスのクラス宣言の中に、main と名付けられたクラスメソッドが定義されていて、その中に、このプログラムが行う仕事が記述されていました。

前回の P101Hello.java

```
/* P101Hello というクラスのクラス宣言 */
class P101Hello {
    /* main というクラスソッドの宣言 */
    public static void main(String[] args) {
        System.out.println("Hello, world!");        // 実行される文
    }
}
```

Java では、C 言語と同様に、/* と */ で囲まれた部分や、// からその行の行末までは**コメント (注釈)**として扱われます。また、

```
System.out.println("Hello, world!");
```

のような、コンピュータに対する指示を、C 言語と同様に「**文**」と呼びます。

[2]Java の言語仕様書では**静的メソッド**と呼んでいます。

[3]C 言語と同様に「メソッドを呼び出す」という言い方もします。

2.3 カードゲーム向けクラスライブラリを使ったプログラム

この科目では、トランプのカードを使った一人遊びのゲームを作成することを想定して、Java プログラミングの勉強をしていきます。そのようなゲームでは、画面にカードを表示したり、表示されているカードを移動したり、裏返したりすることを行うことになりますが、1 枚 1 枚のカードをあるクラスのオブジェクトとして考えることにします。このようなゲームでは、カードの他にも、山札、手札、場、プレーヤー、ゲーム盤などが登場しますので、これらもそれぞれ異なるクラスのオブジェクトとして考えることになるでしょう。

何もないところからこのようなゲームのプログラムを作成するのなら、これらのクラスの定義 (その種類のオブジェクトがどのようなものか) をそれぞれ自分で行う必要がありますが、ここでは、すでにこのようなクラスがクラスライブラリとして用意されている状況から始めたいと思います[4]。

P201.java

```
 1 import jp.ac.ryukoku.math.cards.*;
 2
 3 class P201 {
 4     public static void main(String[] args) {
 5         GameFrame f;            // 変数 f の宣言
 6         Card c1, c2;            // 変数 c1 と c2 の宣言
 7
 8         /* GemeFrame のインスタンス(ゲーム盤)を生成して f へ代入 */
 9         f = new GameFrame();
10         /* Card のインスタンス(ハートのA)を生成して c1 へ代入 */
11         c1 = new Card(Suit.HEARTS, Rank.ACE);
12         /* Card のインスタンス(スペードのJ)を生成して c2 へ代入 */
13         c2 = new Card(Suit.SPADES, Rank.JACK);
14
15         f.add(c1);              // ハートの A をゲーム盤へ追加
16         f.add(c2);              // スペードの J をゲーム盤へ追加
17         c2.moveTo(300, 400);    // スペードの J を (300,400) へ移動
18         c2.flip();              // スペードの J をめくる
19         c1.moveTo(400, 400);    // ハートの A を (400, 400) へ移動
20         c1.flip();              // ハートの A をめくる
21     }
22 }
```

この P201.java というソースファイルをコンパイル[5]すると P201.class というクラスファイルが作られるはずです。コンソール[6]から

```
java P201
```

を実行[7]すると、ソースプログラムの 5 行目以降に書かれた文が順に実行されますが、まず、次の図

---

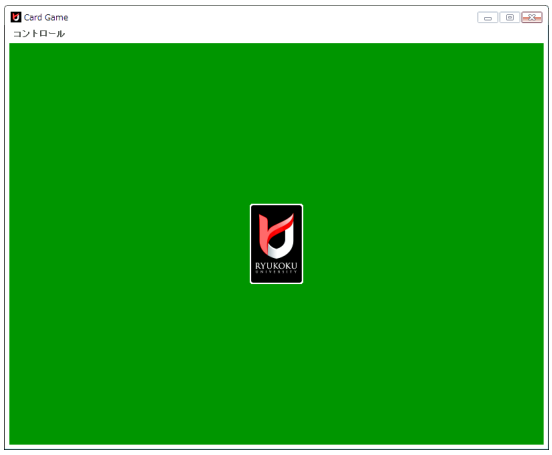
[4] クラスの定義の仕方については、第 6 回以降で解説します。

[5] この科目で使用する (カードゲームのための) クラスライブラリを使用するためには、環境変数や javac コマンドのコマンドライン引数で、そのライブラリファイルを指定しなければなりません。その方法については、**付録:クラスパスの設定**を参照してください。

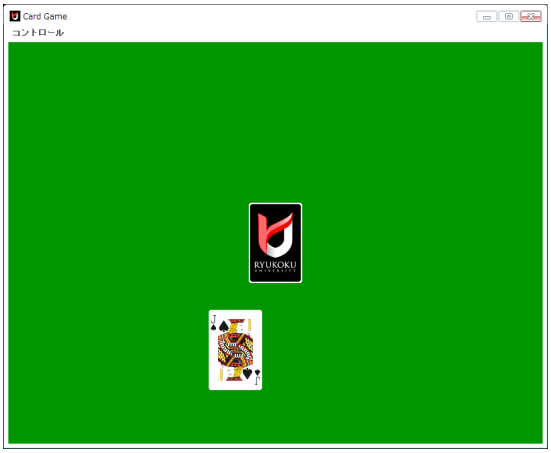
[6] macOS 環境の端末エミュレータや、Windows 環境のコマンドプロンプト (cmd.exe)

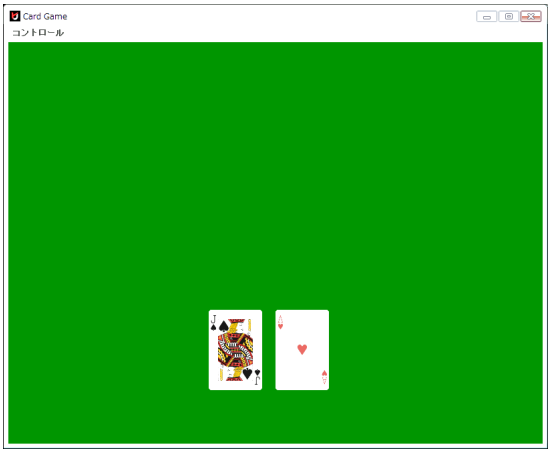
[7] Java バイトコードを実行する場合にも、クラスライブラリが必要となりますので、環境変数や java コマンドのコマンドライン引数で、そのライブラリファイルのパス名を指定しなければなりません。

のようなゲーム盤が表示されて、中央に 2 枚のカードが伏せられた状態で重ねられます。



続いて、2 枚のカードが 1 枚ずつ手前に移動して、カードの表が見える状態に変わります。





このプログラムは、この科目のためのカードゲーム向けクラスライブラリを使用しますが、そこに用意されているクラスを、単純なクラス名で指定して利用するために、プログラムの 1 行目には

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

のように、**インポート宣言**と呼ばれるものが記述されています。インポート宣言については、第 6 回で、**パッケージ**と呼ばれる Java の仕組みについて解説するときに一緒に説明します。

メモ

2.4 オブジェクトの生成

Java プログラムでは、次のような書式の式を書いてオブジェクトの生成を行います。

new クラス名(コンストラクタの引数の列)

この形の式を**インスタンス生成式**と呼び、この式が評価 (計算) されるときに、クラス名 で指定されたクラス宣言に基づいてオブジェクトが生成されます。各クラスのクラス宣言には、生成されたオブジェクト (インスタンス) の初期化を行う**コンストラクタ**と呼ばれる手続きの定義が含まれていますので、このコンストラクタに コンストラクタの引数の列 の部分が引数として渡されて、それに基づき (生成された) オブジェクトが初期化されます。インスタンス生成式は、この初期化されたオブジェクトを表しますので、その値を変数などに記憶しておくことで、生成したオブジェクトに仕事を依頼することができます。

メモ

P201.java では、この科目のクラスライブラリが用意しているクラスの内、GameFrame、Card、Suit、Rank の 4 つを利用しています。それぞれ、次のようなオブジェクトのクラスとして定義されています。

GameFrame	カードゲームのウィンドウに対応するオブジェクトのクラス
Card	トランプの 1 枚のカードに対応するオブジェクトのクラス
Suit	カードのスート (スペード、ハート、ダイヤ、クラブ) を表すオブジェクトのクラス
Rank	カードのランク (2、3、4、…、10、J、Q、K、A) を表すブジェクトのクラス

このプログラムでは、これら 4 つのクラスの内、GameFrame クラスのインスタンスを 1 つ、Card クラスのインスタンスを 2 つ生成しています。他の 2 つのクラス、つまり Suit と Rank に関しては、これらのクラスがあらかじめ生成しているインスタンスを使っているだけで、自分でオブジェクトを明示的に生成することはしていません。

GameFrame クラス　P201.java の 9 行目では、GameFrame クラスのインスタンスを生成しています。これにより、幅 800 ピクセル、高さ 600 ピクセルの大きさの濃緑色のゲーム盤を含むウィンドウが画面に現れます。本来、オブジェクトは目に見えないものですが、GameFrame クラスのインスタンスはゲーム盤の様子を画面に描画する機能を持つように設計されていますので、インスタンスの生成にともない[8]、このようなウィンドウが画面に現れます。

[8] このクラスのコンストラクタがウィンドウの作成を行ってくれます。

Card クラス　P201.java の 11 行目と 13 行目では、それぞれ Card クラスのインスタンスを生成しています。この時、コンストラクタの引数として、生成するカードのスートとランクを指定しています。Suit.HEARTS や Rank.ACE は、Suit クラスや Rank クラスが用意している**クラス変数**[9]と呼ばれる変数[10]で、それぞれ、「ハート」を表す Suit クラスのインスタンスと「A (エース)」を表す Rank クラスのインスタンスがそこに格納されています[11]。

メモ

2.5　オブジェクトを記憶する変数

Java では、生成したオブジェクトを値として変数に記憶することができます。P201.java では、5 行目と 6 行目で、それぞれ GrameFrame クラスと Card クラスのインスタンスを記憶するための変数 f と c1、c2 を宣言しています。クラスのインスタンス (オブジェクト) を記憶する変数は

クラス名 変数名;

のように宣言します。6 行目のように、複数の変数を一度に宣言することもできます。C 言語と同様、宣言されていない変数を使用することはできません。

P201.java では、変数の宣言と初期化 (変数への値の代入) を分けて書いてありますが、

```
GameFrame f = new GameFrame();
Card c1 = new Card(Suit.HEARTS, Rank.ACE);
Card c2 = new Card(Suit.SPADES, Rank.JACK);
```

のように、宣言と初期化を一度に行うこともできます。

メモ

2.6　インスタンスメソッドの起動

オブジェクトが行うことのできる仕事のことを**インスタンスメソッド**と呼びます。1 つのオブジェクトが複数の仕事を行うことができるのが普通ですので、インスタンスメソッドには名前を付けて

[9] Java の言語仕様書では**静的フィールド**と呼んでいます。

[10] 変数といっても、書き換えることはできませんので、実質的には (特定のオブジェクトを表す) 定数です。

[11] その他のスートやランクについては、**付録 : カードゲーム向けクラスライブラリ**を参照してください。

区別し、そのぞれのインスタンスメソッドが行う仕事の手順は、そのクラスのクラス宣言の一部として (ちょうど C 言語の関数を定義するように) 記述しておきます。当然、それぞれのクラスのインスタンスがどのようなインスタンスメソッドを持っているのかは、クラス毎に異なってきます。

インスタンスメソッドの起動は、次のような書式の**メソッド起動式**と呼ばれる式で行うことができます。

オブジェクトを表す式 . インスタンスメソッド名 (引数の列)

オブジェクトを表す式 の部分には、オブジェクトを記憶している変数や、インスタンス生成式など[12]を書くことができます。インスタンスメソッド名 の部分には、その オブジェクトを表す式 が表すオブジェクトが持っている (はずの) インスタンスメソッドの名前を書きます。インスタンスメソッドには、起動する (呼び出す) 際に (C 言語の関数と同様に) 引数として、いくつかの値を手渡すことができますので、これらの式を 引数の列 の部分に , (カンマ) で区切って書きます。

メモ

GrameFrame の add メソッド　P201.java の 15 〜 16 行目では、9 行目で生成した GameFrame クラスのインスタンス[13]の add というインスタンスメソッドを起動して、2 枚のカード (11 行目と 13 行目で生成した Card クラスのインスタンス) をゲーム盤に追加しています。GameFrame クラスの (インスタンスが持つ) add というインスタンスメソッドは、引数として渡されたオブジェクト (Card クラスのインスタンス) をゲーム盤の中央に追加します。

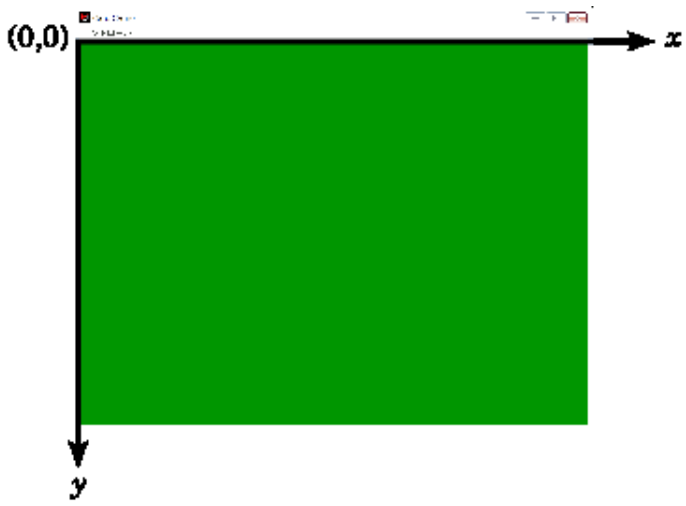
Card クラスのインスタンスは、ゲーム盤 (GrameFrame) に追加されることで、自分の姿を画面に表示することができます。Card クラスのインスタンスは、伏せられた状態で生成されますので、ゲーム盤に追加すると背面しか見えません。また、P201.java では 2 枚のカードを同じ位置 (ゲーム盤の中央) に追加していますので、最初に追加したハートのエースの上に重なるように、続いて追加したスペードのジャックが置かれます。

メモ

[12] 他にも、いろいろな形の式がオブジェクトを表すことがあります。

[13] 変数 f に記憶されています。

Card の moveTo メソッド　Card クラスの moveTo というインスタンスメソッドは、そのインスタンス自身を引数で指定された座標に移動させます。ゲーム盤 (GrameFrame のウィンドウの濃緑色の部分) の標準の大きさは、幅 800、高さ 600 で、その座標系は、左上角を原点 (0, 0) として、右向きに x 軸、下向きに y 軸をとったものとなっています。



moveTo の引数には、そのカードの左上角を位置させたい座標を、x 座標、y 座標の順に整数値で指定します。通常のカードの大きさは、幅 80、高さ 120 ですので、GrameFrame の add メソッドで追加した際のカードの位置 (左上角の座標) は (360, 240) となっています。

Card の flip メソッド　Card クラスの flip というインスタンスメソッドは、そのカードの表裏を反転させます。このメソッドには引数はありません。P201.java の 17 〜 20 行目では、ゲーム盤の中央に伏せられている ハートの A とスペードの J の 2 枚のカードを、それぞれ、(300, 400) と (400, 400) の位置に移動した後、表が見えるようにしています。

メモ

2.7　メソッドの戻り値

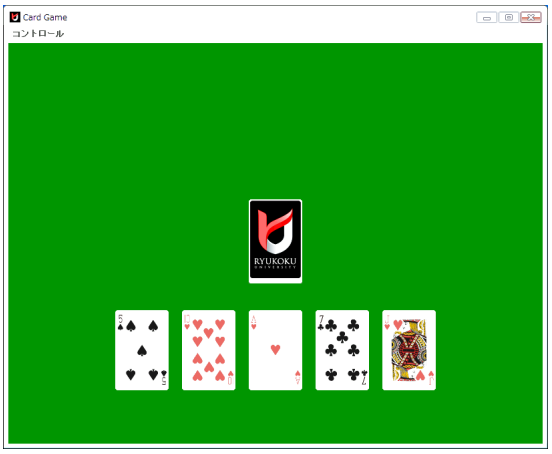
C 言語の関数が戻り値というものを返すことができたのと同じように、Java のメソッド (インスタンスメソッドとクラスメソッド) も戻り値を返すことができます。メソッドの戻り値を利用するプログラムの例を 1 つ紹介します。

P202.java

```
1 import jp.ac.ryukoku.math.cards.*;
2
3 class P202 {
4     public static void main(String[] args) {
5         GameFrame f = new GameFrame();
```

```
 6          Deck d = new Deck();              // デッキ(カード1式の山)を生成
 7          f.add(d);                         // ゲーム盤に追加
 8          d.shuffle();                      // デッキをシャッフル
 9          for (int i = 0; i < 5; i++) {
10              Card c = d.pickUp();          // デッキから1枚カードを引く
11              c.moveTo(i*100 + 160, 400); // 引いたカードを移動
12              c.flip();                     // 移動したカードをめくる
13          }
14      }
15 }
```

このP202.javaというプログラムは、ゲーム盤の中央に置かれたデッキ(トランプのカード1式からなる山)から、1枚ずつカードを引いて手前に移動し、引いたカードを表にします。



P202.javaは、この科目のクラスライブラリに含まれている別のクラスDeckを利用しています。Deckクラスのインスタンスは、トランプのカード1揃いを重ねたものに対応するオブジェクトです。6行目でDeckのインスタンスを生成し、Cardの場合と同様に、GameFrameクラスのaddというインスタンスメソッドを起動して、ゲーム盤に追加しています。GameFrameのaddメソッドは、Cardクラスのインスタンスでも Deckクラスのインスタンスでも引数にすることができます。

8行目では、Deckクラスの shuffleというインスタンスメソッドを起動して、このデッキをシャッフルしています。Javaでは、C言語と同じ書き方の for文が使えますので、9行目から13行目にかけて、デッキから1枚のカードを引いて、手前に移動し、カードをめくる、ということを5回繰り返しています。デッキからカードを引いているのは、10行目の

```
Card c = d.pcikUp();
```

の部分です。変数 dには Deckクラスのインスタンスが記憶されてますので、このオブジェクトの pickUpというインスタンスメソッドを起動しています。この pickUpメソッドはデッキの一番上からカードを1枚取り除いて、その取り除いたカード(Cardクラスのインスタンス)を、メソッドの戻り値として返してくれます。この戻り値を変数 cに記憶して、そのカードを移動、反転させています。

2.8 C 言語との類似点

オブジェクト指向という面では、Java は C 言語と比べて大きく異なりますが、それ以外の部分では、C 言語と似ている部分もたくさんあります。式に続けて ; を書いて文とするところや、メソッド起動の引数を ( ) で囲むところ、{ } でブロック構造を表現するところなど、プログラムの見掛けが似ていることに気づきますが、Java では、次のような部分に関しても、C 言語での書き方をそのまま使うことができます。

- int 型、short 型、long 型、float 型、double 型など、数値を表現するためのデータ型とその定数
- if 文、for 文、while 文、do 文、switch 文、break 文、continue 文などの制御文
- ブロック ({ ... })

P202.java の 9 行目の for 文では、初期設定式が「int i = 0」のように、変数の宣言を含んだ形になっていますが、これは、

```
{
    int i;

    for (i = 0; i < 5; i++) {
        ...
    }
}
```

と書くのと同等です[14]。

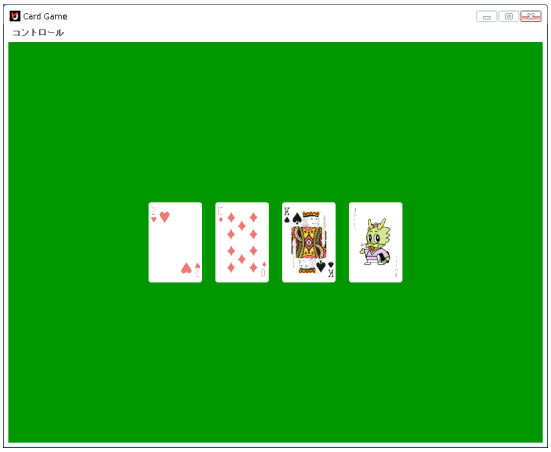
メモ

2.9 演習問題

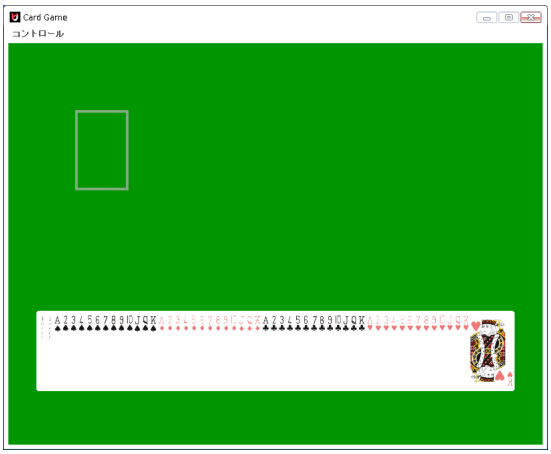
付録を参考にして、以下のようなプログラムを作成しなさい。

1. ゲーム盤の中央に、ハートの 2、ダイヤの 10、スペードの K、ジョーカーを、次の図のように表向きしてに並べるプログラム P203.java を作成しなさい。ハートの 2 のカードの座標は (210, 240) です。隣り合うカードの x 座標の差は 100 です。

[14] C99 と呼ばれている比較的新しい C 言語の規格でも、このような書き方の for 文が許されていますが、コンパイラによっては、この規格に対応していなかったり、特定のオプションをコマンドライン引数に指定する必要があります。

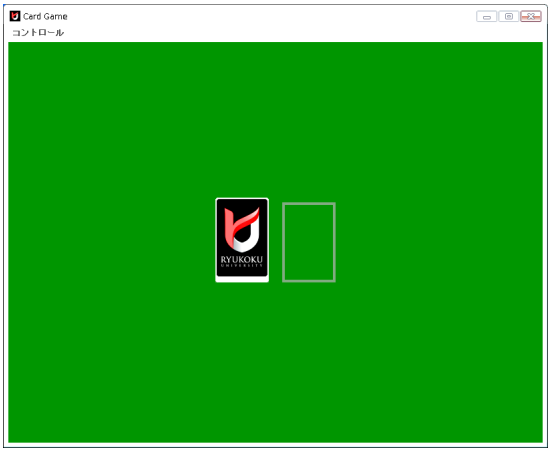


2. ゲーム盤の (100, 100) の位置にジョーカーを 2 枚含むデッキを置き、デッキごと裏返してから、デッキの 1 番上のカードから順に、ゲーム盤の下部に移動して横一列に並べるプログラム `P204.java` を作成しなさい。

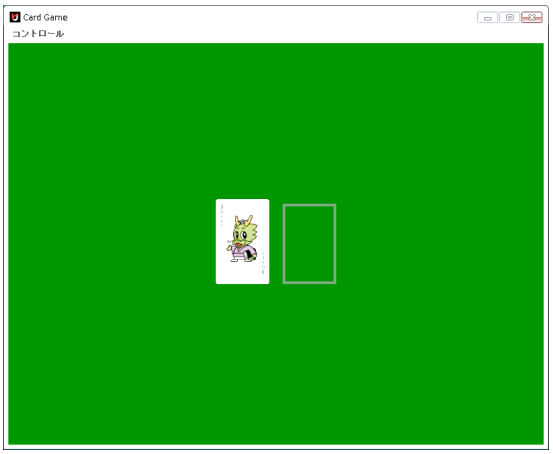


一列に並んだ最も左のカード (ジョーカー) の座標は (42, 400) です。隣り合うカードの x 座標の差は 12 です。

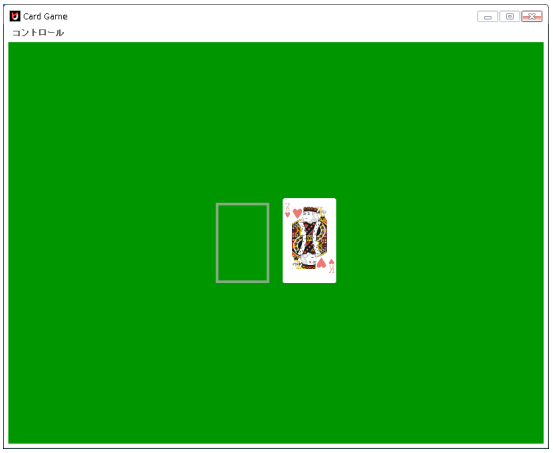
3. 次のようなプログラム `P205.java` を作成しなさい。このプログラムでは、まず、次の図のように、ゲーム盤の (310, 240) の位置にジョーカー 1 枚を含むデッキを、(410, 240) に空の山 (`Pile` クラスのインスタンス) を置きます。



デッキをデッキごと裏返して、次の図にようにする。



デッキの 1 番上のカードから順に、1 枚ずつ右隣の山へ移動する。 この際、単にカードの位置を変えるのではなく、右隣の山 (`Pile` クラスのインスタンス) へ追加するようにしなさい。最終的には、次の図のようになります。



2.10 付録 : クラスパスの指定

javac コマンド や java コマンドが必要なクラスファイルを探す際には、Java の開発環境や実行環境の既定のディレクトリに加えて、**クラスパス (class path)**と呼ばれる設定に含まれるディレクトリを順に探していきます。特にクラスパスを指定しない場合は、カレントディレクトリだけがクラスパスに含まれるものとして扱われます。Java の開発環境や実行環境に含まれている標準的なクラスライブラリだけを使用する場合は特にその必要はありませんが、独自のクラスライブラリを使用する場合には、このクラスパスを指定して、使用するクラスライブラリの場所を javac や java コマンドに教えてあげる必要があります。

この科目では、カードゲームのための独自のクラスライブラリを使用しますが、そのライブラリが提供するクラスファイルは、**jar ファイル**と呼ばれる形式で、cards.jar という名前のファイルにまとめられています。このため、javac や java コマンドに対して、(カレントディレクトリに加えて) この jar ファイルもクラスパスに含めるように指定しなければなりません。

javac や java の -cp オプション

例えば、カレントディレクトリと兄弟関係にある lib というディレクトリに置かれた cards.jar というクラスライブラリを使用したい場合は、javac コマンドや java コマンドを起動する際に、次のように -cp オプション[15]を使用して、bin デイレクトリや cards.jar をクラスパスに含めるように指示します。例えば、Windows 環境では、

Windows PowerShell
```
PS C:\ ... \OOProg\src> javac -cp ".;..\lib\cards.jar" P201.java
PS C:\ ... \OOProg\src> java -cp ".;..\lib\cards.jar" P201
```

のように、また、macOS 環境では、

macOS (zsh)
```
myname@mac src % javac -cp ".:../lib/cards.jar" P201.java
myname@mac src % java -cp ".:../lib/cards.jar" P201
```

のようにします。

コマンド名に続く -cp の後には、カレントディレクトリを表す . と jar ファイルのパス名が、Windows の場合は ; (セミコロン) で、macOS の場合は : (コロン) で区切られて指定されていることに注意してください。

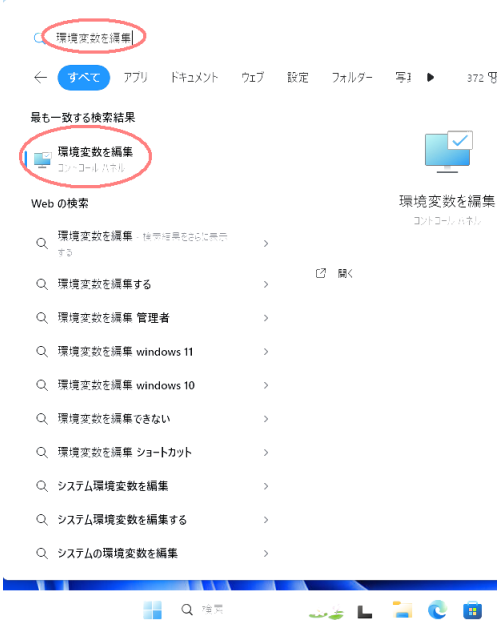
2.11 付録: 環境変数 CLASSPATH の設定

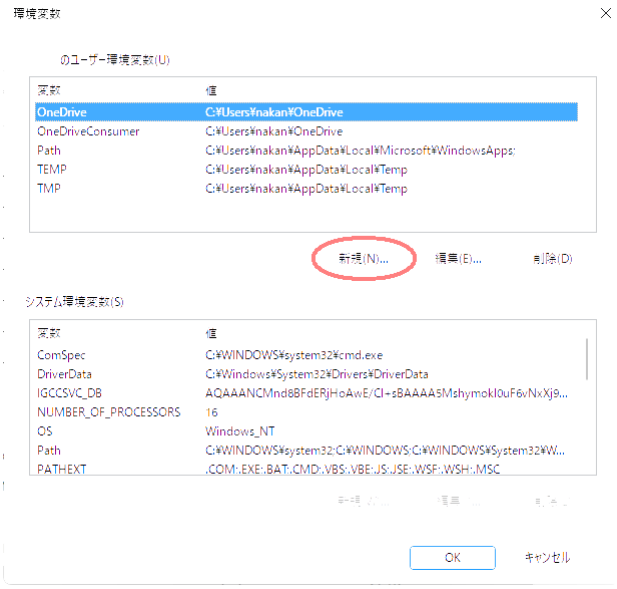
コンパイルや実行の際に、毎回 -cp オプションを指定するのは面倒なので、別の方法でクラスパスを設定することもできます。Windows 環境や macOS 環境には、**環境変数**と呼ばれる各プログラムの動作を変更するための仕組みがあり、クラスパスの場合は、CLASSPATH という名前の環境変数の値を設定しておくことで、クラスパスを指定することができます。こうしておくと、-cp オプションを指定せず、単に「javac P201.java」や「java P201」を実行するだけでこの科目のクラスライブラリが使用できるようになります。

[15] -cp の代りに -classpath としても構いません。

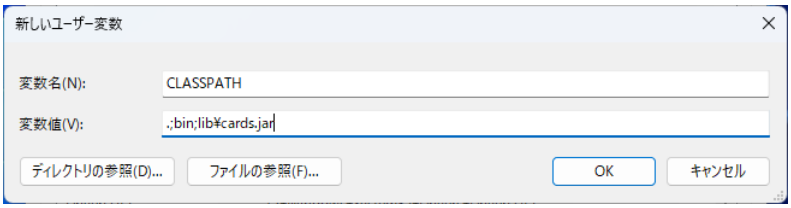
環境変数はコンソールから設定することもできますが、そのコンソールを閉じてしまうと環境変数の設定が失われてしまいますので、サインオン (ログイン) 時に環境変数が自動的に設定されるようにしておくと便利です。これは、以下のような手順で行うことができます。

Windows 環境での手順　Windows 環境の場合、スタートボタン右の検索窓に「環境変数を編集」と入力すると表示される「環境変数を編集」という項目をクリックしてください。





「環境変数」のウィンドウが現れたら、「... のユーザー環境変数」の「新規」をクリックし、「変数名」を「CLASSPATH」に、「変数値」を (指定したい) クラスパスにして「OK」をクリックします。



macOS 環境での手順　macOS 環境の場合、Visual Studio Code などのエディタ[16]を起動して、ホームディレクトリに置かれている `.zshrc` というファイルに、例えば次のような 1 行を追加します[17]。

```
export CLASSPATH=".:bin:lib/cards.jar"
```

2 つの " の間に書かれているのが (指定しようとしている) クラスパスです。macOS の Finder (Visual Studio Code でファイルを開く場合も同様) では `.zshrc` のように「.」で始まる名前のファイルは表示されません。表示するには `command` キー とシフトキーを同時に押しながら「.」キーを押します。

[16]「テキストエディット」を使用する場合は「フォーマット」メニューで「標準テキストにする」を選択 (「リッチテキストにする」が表示されている状態に) してください。

[17] `.zshrc` が存在しない場合は作成します。

2.12 付録 : カードゲーム向けクラスライブラリ

この科目のカードゲーム向けクラスライブラリには、以下のようなクラスが含まれています。

GameFrame クラス — ゲーム盤を含むウィンドウ

コンストラクタ	
`GameFrame()`	幅 800、高さ 600 ピクセルのゲーム盤を含むウィンドウ
`GameFrame(int w, int h)`	幅 `w`、高さ `h` ピクセルのゲーム盤を含むウィンドウ
インスタンスメソッド	
`void add(Elem`[18] `e)`	`e` をゲーム盤の中央に置く
`void add(Elem e, int x, int y)`	`e` をゲーム盤の (`x`, `y`) の位置に置く
`void remove(Elem e)`	`e` をゲーム盤から取り除く

Card クラス — トランプのカード

コンストラクタ	
`Card(Suit s, Rank r)`	スートが `s` でランクが `r` のカード
`Card(int no)`	通し番号が `no` のカード
`Card()`	ジョーカーのカード
インスタンス変数	
`Suit suit`	このカードのスート (ジョーカーは null)
`Rank rank`	このカードのランク (ジョーカーは null)
インスタンスメソッド	
`void faceDown()`	裏向きにする
`void faceUp()`	表向きにする
`void flip()`	表裏を反転する
`int getNumber()`	通し番号を戻り値として返す
`int getX()`	左上角の x 座標を戻り値として返す
`int getY()`	左上角の y 座標を戻り値として返す
`int getWidth()`	カードの幅を戻り値として返す
`int getHeight()`	カードの高さを戻り値として返す
`boolean isFacedDown()`	裏向きかどうかを戻り値として返す
`boolean isFacedUp()`	表向きかどうかを戻り値として返す
`boolean isJoker()`	ジョーカーかどうかを戻り値として返す
`boolean isPictureCard()`	絵札 (J、Q、K、A) かどうかを戻り値として返す
`boolean isBlack()`	スペードかクラブかどうかをを戻り値として返す
`boolean isRed()`	ハートかダイヤかどうかをを戻り値として返す
`void moveTo(int x, int y)`	左上角が (`x`, `y`) となるように移動する
`void moveTo(Pile p)`	`p` の山に移動して、その山の一番上に加える
`void shiftTo(int x, int y)`	カードの重なり順を変えずに、左上角が (`x`, `y`) となるように移動する
`void raise()`	カードの重なり順を最上位にする
`void lower()`	カードの重なり順を最下位にする
`void pause(int ms)`	`ms` ミリ秒だけ時間が経過するのを待つ

カードの通し番号

	A	2	3	4	5	6	7	8	9	10	J	Q	K
スペード	0	1	2	3	4	5	6	7	8	9	10	11	12
ハート	13	14	15	16	17	18	19	20	21	22	23	24	25
ダイヤ	26	27	28	29	30	31	32	33	34	35	36	37	38
クラブ	39	40	41	42	43	44	45	46	47	48	48	50	51
ジョーカー	52												
予備のジョーカー	53												

[18] ゲーム盤に置かれるオブジェクトのクラスで、`Card` や `Deck`、`Pile` などのクラスは、その一種です。

Suit クラス — カードのスート

クラス変数 (静的フィールド)	
`Suit SPADES`	スペード
`Suit HEARTS`	ハート
`Suit DIAMONDS`	ダイヤ
`Suit CLUBS`	クラブ
クラスメソッド (静的メソッド)	
`Suit suitOf(int n)`	`n` 番目のスート (1=スペード、2=ハート、3=ダイヤ、4=クラブ) を戻り値して返す

Rank クラス — カードのランク

クラス変数 (静的フィールド)	
`Rank ACE`	A (エース)
`Rank DEUCE`	2
`Rank THREE`	3
`Rank FOUR`	4
`Rank FIVE`	5
`Rank SIX`	6
`Rank SEVEN`	7
`Rank EIGHT`	8
`Rank NINE`	9
`Rank TEN`	10
`Rank JACK`	J (ジャック)
`Rank QUEEN`	Q (クイーン)
`Rank KING`	K (キング)
クラスメソッド (静的メソッド)	
`Rank rankOf(int n)`	`n` を表すランク (1=A、2=2、···、10=10、11=J、12=Q、13=K) を戻り値して返す
インスタンスメソッド	
`int getNumber()`	ランクの表す数 (ただし、A=1、J=11、Q=12、K=13) を戻り値として返す

Pile クラス — カードの山

コンストラクタ	
`Pile()`	空の山
インスタンスメソッド	
`void add(Card c)`	`c` を山の一番上に加える
`void remove(Card c)`	`c` を山から取り除く (`c` の位置は不変)
`boolean isEmpty()`	山が空かどうかを戻り値として返す
`int count()`	山に含まれるカードの枚数を戻り値として返す
`Card top()`	山の一番上のカードを戻り値として返す
`Card pickUp()`	山の一番上のカードを取り除いて戻り値として返す
`void flip()`	山ごと表裏を反転する
`void shuffle()`	山をシャッフルする
`void moveTo(int x, int y)`	左上角が (`x`, `y`) となるように山を移動する

Deck クラス — 1 セットのカードの山

コンストラクタ	
`Deck()`	ジョーカーを含まない 52 枚のカードの山
`Deck(int n)`	ジョーカー `n` 枚を含む 52+`n` 枚カードの山
インスタンスメソッド	
`Pile` クラスと同じ	

オブジェクト指向及び演習・第 2 回・終わり