グラフィックス及び演習・第 1 回

2025 年 6 月 16 日

配布資料の内容

1.1 この科目について 1–2
1.2 「オブジェクト指向及び演習」の復習 1–2
1.3 演習問題 1–9
1.4 付録: Java プログラムのコンパイルと実行 1–11
1.5 付録: クラスパスの指定 1–12
1.6 付録: カードゲーム向けクラスライブラリ 1–13

シラバス抜粋

科目名	グラフィックス及び演習 (2020 年度以降入学生) グラフィックス基礎及び実習 (の一部) (2019 年度以前入学生)
講義概要	Java は標準化された豊富なライブラリ群が用意されていますので、あまり細かいことを意識しなくてもグラフィックス (図形の描画など) や、マウス、キーボードの処理を行ったり、あらかじめ用意されている部品 (ボタンやメニューなど) を組み合わせて比較的容易にグラフィカルユーザインターフェース (GUI) を構築することができます。この科目では、Java プログラミングを学びながら、グラフィックスの基本、グラフィカルユーザーインタフェースの仕組みやその構築法を学びます。
到達目標	画面の描画 (グラフィックス) に関する基本的な手法を学び、グラフィカルユーザーインターフェース (GUI) の構築法を理解できる。
授業方法	配布資料[1]に沿って講義を行います。これと並行して情報実習室での Java プログラミングの演習を行います。
成績評価方法	期末試験 (100 点満点) と提出された課題で評価します。期末試験が x 点、課題の得点率が $y\%$ のとき、総合的な成績は $x+(100-x)y/500$ 点 (端数切り捨て) となります。
講義計画	(1) Java 言語の復習、オブジェクト指向の復習 (2) JFC と Swing、図形や文字の描画 (3) イベント駆動型プログラミング、インタフェース (4) ビットマップ画像の描画、例外のスローとキャッチ (5) レイアウトマネージャ、GUI コンポーネント (6) 抽象クラス、入れ子クラス (7) スレッド、排他制御 (8) まとめ
系統的履修	「オブジェクト指向及び演習」を受講していることを前提として授業を進めます (2020 年度以降入学生)。
参考文献	立木秀樹、有賀妙子『すべての人のための Java プログラミング 第 3 版』(共立出版) (ISBN: 978-4320124233)
Web ページ	`https://www602.math.ryukoku.ac.jp/Graphics/`

[1] 配布資料の多くは `https://www602.math.ryukoku.ac.jp/Graphics/` から入手することができます。

1.1 この科目について

この科目は、Java 言語によるオブジェクト指向プログラミングがどのようなものなのかをそれなりに理解している人を対象に、Java 言語の勉強を進めながら、基本的なグラフィカルユーザインタフェースの考え方と、その構築法を理解して頂くものです。教科書は使用しません。シラバスに参考書として挙げた書籍などを、必要に応じて読みこなすための基礎を構築するのが目標となります。

授業の進め方　この科目では、各回の授業を次のように進めて行きます。

月曜日・4–5 講時・1–608 情報実習室

講義 (60 〜 90 分間)　その回の内容に関する説明を行ないます。席は自由です。資料[2]の配布はこの時間の始めに行ないます。

実習 (残りの時間)　持ち込みノート PC[3]を使ってプログラミングの実習を行います。16:00 頃から TA さんがサポートしてくれます。

メモ

1.2 「オブジェクト指向及び演習」の復習

オブジェクト指向

- **オブジェクト**とは、実行中のプログラムの中で、特定の役割や機能を持って働く仮想的な存在 (仕事人? 妖精さん?) のことをいう。
- オブジェクト指向プログラミングとは

  > 特定の役割や機能を持った多数のオブジェクトが、お互いに仕事を依頼し合ったり
  > 情報を交換し合ったりすることで全体が機能する

  という考え方でプログラミングを行うこと。
- それぞれの オブジェクト は**状態**と**振る舞い**を持つ。

Java 言語

- Java はオブジェクト指向プログラミングを意識したプログラミング言語である。
- Java プログラムは**クラス宣言**の集まりで構成される。
- Java は C 言語と似た文法を持っている。

クラス宣言

- **クラス**とはオブジェクトの種類のこと。
- Java のクラスは、次のような書式[4]のクラス宣言を行うことで定義する。クラス宣言は、主に、そのクラスのオブジェクトの設計図として働く。

```
class クラス名 {
    各種の宣言の並び
}
```

- クラス宣言の 各種の宣言の並び の部分には、

  インスタンス変数 — そのクラスのオブジェクトがそれぞれ保持する変数 (オブジェクトの状態の定義)

  インスタンスメソッド — そのクラスの各オブジェクトができる仕事の具体的な手続き (オブジェクトの振る舞いの定義)

  コンストラクタ — そのクラスのオブジェクトの初期化の手続き (プログラム)

  クラス変数 — そのクラスに関連してプログラム全体で保持する変数

  クラスメソッド — そのクラスに関連した手続き (プログラム)

  などの宣言が記述される[5]。この内、インスタンス変数の宣言、インスタンスメソッドの宣言、コンストラクタの宣言の 3 つがオブジェクトの設計図として働く。

- `static` 修飾子のない変数宣言やメソッド宣言は、インスタンス変数やインスタンスメソッドの宣言となり、`static` 修飾子付きの変数宣言やメソッド宣言は、クラス変数やクラスの宣言となる。

インスタンス

- クラス宣言に基づいて生成されたオブジェクトを、そのクラスのインスタンスと呼ぶ。
- インスタンスの生成は、次の書式のインスタンス生成式で生成する。

```
new クラス名(コンストラクタの引数の列)
```

- それぞれのインスタンスは、そのクラスで宣言された (継承したものを含む) インスタンス変数を内部に保持する。
- インスタンス変数には、次の書式でアクセスできる。

```
オブジェクトを表す式.インスタンス変数名
```

---

[4] `class` の前には、`public`、`abstract`、`final` などの修飾子が、クラス名 と `{` の間には、(後述の) `extends` 節や `implements` 節が置かれることがある。

[5] この他にも、メンバクラスやメンバインタフェース、インスタンス初期化子、クラス初期化子を宣言することができる。

ただし、[オブジェクトを表す式] が this の場合[6]、this. は省略することもできる。

- それぞれのインスタンスは、そのクラスで宣言された (継承したものを含む) インスタンスメソッドを実行することができる。
- コンストラクタやインスタンスメソッドの宣言の中で、初期化されようとしている、あるいはそのインスタンスメソッドを実行しようとしているインスタンスを this で参照することができる。
- インスタンスメソッドは、次の書式のメソッド起動式を使って、ターゲットとなるオブジェクト (インスタンス) を指定した上で起動する。

  [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])

  ただし、[オブジェクトを表す式] が this の場合、this. は省略することもできる。

スーパークラスとサブクラス

- 次のような書式のクラス宣言で、既存のクラスの定義の一部を変更したり追加したりして、新しいクラスを定義することができる。

```
class [クラス名] extends [既存のクラス名] {
    [各種の宣言の並び (変更点あるいは追加点)]
}
```

- 元になったクラスを、新しいクラスの (直接の) スーパークラスと呼び、新しいクラスは、元となったクラスの (直接の) サブクラスと呼ばれる。
- 新しいクラスの宣言では、インスタンス変数やインスタンスメソッドを追加したり、インスタンスメソッドを再定義 (**オーバーライド**) したりすることができる。
- 再定義しない限り、元のクラスの変数やメソッドはサブクラスに**継承**される。
- Java では、複数のクラスを継承したクラスを宣言すること (多重継承) はできない。
- 新しいクラスで再定義される前のメソッド (直接のスーパークラスでの) 定義を、次の書式で起動することができる。

  super. [メソッド名] ([引数の列])

コンストラクタ

- あるクラスのインスタンスが生成される際には、インスタンス生成式で指定された引数を伴って、そのクラスのコンストラクタが起動される。
- コンストラクタは、生成されたばかりのオブジェクトが、そのクラスのインスタンスとして働くための準備を整える役割を持つ。

[6] かつ、仮引数や局所変数と名前が衝突しない場合。

- コンストラクタ宣言の本体の冒頭で、次の書式により直接のスーパクラスのコンストラクタを起動することができる。

  `super(`引数の列`);`

- 同様に、次の書式により同じクラスの他のコンストラクタを起動することができる。

  `this(`引数の列`);`

- コンストラクタ宣言の本体が `super(...);` や `this(...)` で始まっていない場合は、冒頭に次が補われる。

  ```
  super();
  ```

- コンストラクタが全く宣言されていないクラス宣言には、次のようなコンストラクタ宣言が補われる。

  クラス名 `() {`
  `}`

- コンストラクタはサブクラスに継承されないが、必ずスーパークラスのコンストラクタが起動された後に、サブクラスのコンストラクタ本体が実行されるようになっている。

配列

- Java の配列はオブジェクトの一種である。
- 配列は、次の書式の配列生成式で生成する。

  `new` 要素の型名 `[`要素の個数`]`

- 配列オブジェクトは、`length` という `int` 型のインスタンス変数を持つ。
- 一旦生成された配列オブジェクトの大きさ (要素数) を後から変えることはできない。
- Java では、多次元の配列を、配列型を要素とする配列で表現する。
- 配列の各要素を処理するために、

  `for (`型名 変数名`:`配列を表す式`)` 文

  の形の `for` 文を使うことができる。

参照型

- Java ではオブジェクトを、すべて**参照**で扱う。
- 値を参照で扱うデータ型を参照型と呼ぶ。
- どのオブジェクトも指していないことを示す値を null 参照と呼び、プログラム中では `null` で表すことができる。

- クラス C を宣言すると、C は参照型の 1 つとなる。C 型は、そのクラスとそのサブクラスのインスタンスへの参照、および null 参照からなる型となる。
- T 型の要素持つ配列の型を T`[]` で表す。T`[]` は参照型の 1 つとなる。

動的ディスパッチ

- インスタンスメソッドの起動では、通常[7]、そのときターゲットとなったオブジェクト (のクラス) が持つメソッドの定義が用いられる。
- つまり、プログラム中の同じメソッド起動式で起動されるメソッドの定義が、起動の度に異なる場合がある (**多態性**)。
- このように、プログラムの実行時に (メソッドが起動される度に)、そのとき使われるメソッド定義が選ばれることを**動的ディスパッチ**と呼ぶ。

クラス変数とクラスメソッド

- C 言語の大域変数に相当するものとして、Java にはクラス変数がある。
- クラス変数は、プログラム全体に 1 つだけ存在する。
- クラス変数には、次の書式でアクセスできる。

  クラス名 . クラス変数名

  同じクラスのクラス変数へアクセスする場合[8]、クラス名 . を省略することもできる。
- C 言語の関数に相当するものとして、Java にはクラスメソッドがある。
- クラスメソッドは、次の書式のメソッド起動式で起動する。

  クラス名 . クラスメソッド名 (引数の列)

  同じクラスのクラスメソッドを起動する場合、クラス名 . を省略することもできる。
- Java アプリケーションでは、指定されたクラスのクラスメソッド `main` が起動されることでプログラムの実行が開始される。

オーバーロード

- コンストラクタや、インスタンスメソッド、クラスメソッドは、同じ名前[9]でも、引数の数や型が異なるものは区別される。
- 引数の数や型が異なる同名のコンストラクタや、インスタンスメソッド、クラスメソッドを宣言することを**オーバーロード**と呼ぶ。

---

[7] `super.` を使ったインスタンスメソッドの起動は除く。

[8] かつ、仮引数や局所変数と名前が衝突しない場合。

[9] コンストラクタの場合は、同じクラス名

基本的なクラス

- すべてのオブジェクトのスーパークラスとして Object クラスがある。
- 配列型オブジェクトも Object 型である。
- Java では、文字列を String クラスのインスタンスで表す。
- Java の + 演算子は文字列を連結することができる。このとき、+ 演算子の左式と右式のいずれか一方が文字列 (String 型) で、もう一方がそうでないとき、文字列でない方は自動的に文字列に変換された上で連結される。
- Java のすべての値は、文字列 (String のインスタンス) に変換できる。
- 三角関数、指数関数、対数関数など基本的な数学関数が Math クラスのクラスメソッドとして用意されている。
- System クラスを利用すると、標準入力、標準出力、標準エラー出力、現在時刻などにアクセスできる。
- プリミティブ型 (boolean、char、byte、short、int、long、float、double) のそれぞれに対応するラッパクラス (Boolean、Character、Byte、Short、Integer、Long、Float、Double) があって、プリミティブ型の値をラッパクラスのインスタンスとして扱うこともできる。

基本的な構文

- C 言語と同様に、if 文、for 文、while 文、do 文、switch 文、break 文、continue 文などの構文が用意されている。
- いくつかの文の並びを { } で囲って複文 (ブロック) にできるのも C 言語と同様。
- if 文や while 文などの条件式は boolean 型でないといけない。

final

- final という修飾子付きで宣言された変数は (初期化されると) 値を変更する (代入する) ことはできない。
- final という修飾子付きで宣言されたインスタンスメソッドは、サブクラスで再定義 (オーバーライド) できない。
- final という修飾子付きで宣言されたクラスはサブクラスを宣言できない。

プリミティブ型

Java には、プリミティブ型と総称される次の 8 つの型がある。これらの型では (参照ではなく) 値のビット列表現をそのまま受け渡しする。

boolean	真理値 (真か偽か) を表す型。ソースプログラム中では、true で真を、false で偽を表すことができる。== や !=、<、<=、>、>= などの演算子の演算結果は boolean 型となる。また、&& や ||、! は、boolean 型に対する演算子で、結果も boolean 型となる[10]。if 文や while 文、for 文、do 文などの条件式にも boolean 型の式を書く。
byte	8 bit の符号付き整数の型。−128 〜 127 の範囲の整数を表現できる。
short	16 bit の符号付き整数の型。−32768 〜 32767 の範囲の整数を表現できる。
int	32 bit の符号付き整数の型。−2147483648 〜 2147483647 の範囲の整数を表現できる。
long	64 bit の符号付き整数の型。−9223372036854775808 〜 9223372036854775807 の範囲の整数を表現できる。
char	16 bit の符号なし整数の型。0 〜 65535 の範囲の整数を表現できる。UTF-16 という文字コード[11]の 1 符号単位 (16bit) を格納するために用いられる。
float	IEEE 754 標準規格の単精度浮動小数点数 (符号 1bit、仮数部 23bit、指数部 8bit) の型。
double	IEEE 754 標準規格の倍精度浮動小数点数 (符号 1bit、仮数部 52bit、指数部 11bit) の型。

メモ

1.3 演習問題

1. 次の Web ページを参照して、各自の PC に、この科目のための Java プログラミング環境を構築しなさい。

   https://www602.math.ryukoku.ac.jp/OOProg/java.html

   注意:「オブジェクト指向及び演習」で構築済みの受講者はあらためて作り直す必要はありません。

2. 次の Web ページを参照して、各自の PC の適当な場所に、この科目のために使用する Java プロジェクトのフォルダ (ディレクトリ) を作成しなさい。このプロジェクトフォルダの名前は適当につけて構いません。

   https://www602.math.ryukoku.ac.jp/OOProg/project.html

   注意:「オブジェクト指向及び演習」で作成済みの受講者はあらためて作り直す必要はありません。

3. 作成したプロジェクトフォルダの `src` というサブフォルダに、次のような Java プログラム `G101.java` を作成し、コンパイル、実行して、正しく動作すること確認しなさい。

   G101.java
   ```
   class G101 {
       public static void main(String[] args) {
           System.out.println("「グラフィックス及び演習」へようこそ!");
       }
   }
   ```

   注意: このプログラムには日本語の文字が含まれています。通常、Windows 環境の `javac` コマンドは、ソースファイルの文字コードが Shift-JIS であることを期待しますので、Visual Studio Code (のデフォルトの設定) など、UTF-8 コードで作成したソースファイルをコマンドラインでコンパイルする場合は

   Windows (PowerShell)
   ```
   PS C:\Users\myname\Desktop\OOProg> cd src
   PS C:\Users\myname\Desktop\OOProg\src> javac -encoding utf-8 G101.java
   ```

   のように、`-encoding utf-8` というコマンドラインオプションを付けて `javac` コマンドを起動します[12]。

4. 次のページから `cards.jar` をダウンロードし、作成したプロジェクトフォルダの `lib` というサブフォルダにコピーしなさい。

   https://www602.math.ryukoku.ac.jp/OOProg/cards.html

   注意:「オブジェクト指向及び演習」でダウンロード済みの受講者はあらためてダウンロードし直す必要はありません。

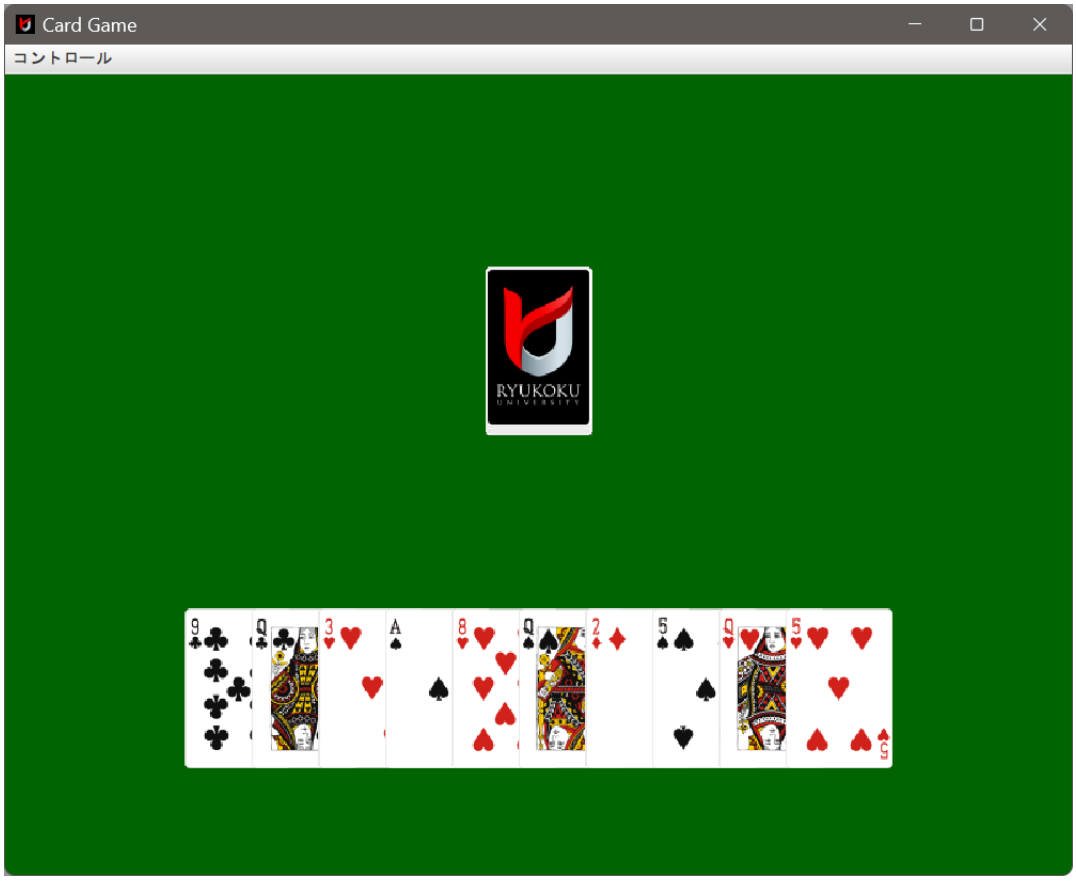
[12] macOS の `javac` コマンドは、デフォルトで UTF-8 を期待しますので、このオプションは必要ありません。

5. 次の Java プログラム G102.java は、この科目のカードゲーム用クラスライブラリを使って、ジョーカーを含まないデッキを (360, 150) の位置に作成し、シャッフルした後、デッキからカードを 1 枚取り出しては移動して表向きにすることで、10 枚のカードを画面の下部に左から順に並べていくものである。ただし、画面下部に並ぶ左端のカードの位置は (135, 400) であり、カードの x 座標は 50 ずつ大きくなる。

G102.java

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

class G102 {
    public static void main(String[] args) {
        GameFrame f = new GameFrame();
        Deck d = [          ]        // デッキ(カード1式の山)を生成
        f.add(d, 360, 150);          // ゲーム盤に追加
        [                   ]        // デッキをシャッフル
        for (int i = 0; i < 10; i++) {
            Card c = d.pickUp();     // デッキから1枚カードを引く
            [               ]        // 引いたカードを移動
            [               ]        // 移動したカードをめくる
        }
    }
}
```



プロジェクトフォルダの src に、G102.java を作成し、上の空欄を埋めて、コンパイル、実行し、正しく動作することを確認しなさい。

グラフィックス及び演習・第 1 回・終わり

1.4 付録: Java プログラムのコンパイルと実行

ソースプログラムのコンパイル　Java のソースプログラムのコンパイルは、Java コンパイラを使って行います。コンパイルの手順は、使用している Java の開発環境によって異なりますが、最も基本的方法は、コンソールウィンドウ[13]を開いて、「`javac`」というコマンドを実行することです。

Windows (PowerShell)
```
PS C:\Users\myname\Desktop\OOProg\src> javac -encoding utf-8 G101.java
```

macOS (zsh)
```
myname@mac src % javac G101.java
```

うまくコンパイルできれば、`javac` コマンドが「`G101.class`」という名前の新しいファイルを作成してくれます。確認してみましょう。

Windows (PowerShell)
```
PS C:\Users\myname\Desktop\OOProg\src> ls


    ディレクトリ: C:\Users\myname\Desktop\OOProg\src


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2025/06/16     16:02            457 G101.class
-a----        2025/06/16     15:58            154 G101.java
```

macOS (zsh)
```
myname@mac src % ls
G101.class  G101.java
```

この (Java のソースファイルをコンパイルすることで生成される) ファイルを**クラスファイル**と呼びます。クラスファイルには (コンパイラによって生成された) Java 仮想機械 (JVM) の機械語プログラム (Java バイトコード) が格納されています。

クラスファイルの実行　Java コンパイラ (`javac` コマンド) によって作成されたクラスファイルを実行するためには `java` コマンドを使用します。`java` コマンドは、クラスファイルに格納されている Java バイトコードを読み取り、その命令を逐次解釈して実行してくれるソフトウェアです。`java` コマンドのコマンドライン引数には、以下の例のように、クラスファイルの名前から `.class` を取り除いたもの[14]を指定します。

Windows (PowerShell)
```
PS C:\Users\myname\Desktop\OOProg\src> java G101
Hello, world!
```

macOS (zsh)
```
myname@mac src % java G101
Hello, world!
```

[13] macOS 環境では「ターミナル」、Windows 環境では「Windows PoerShell」を起動します。

[14] より正確には、ソースプログラムの冒頭の「`class`」の後に書いた名前。

1.5 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 ... \src> javac -cp ".;..\lib\cards.jar" -encoding utf-8 G102.java
PS ... \src> java -cp ".;..\lib\cards.jar" G102
```

のように、また、macOS 環境では、

macOS (zsh)
```
myname@mac src$ javac -cp ".:../lib/cards.jar" G102.java
myname@mac src$ java -cp ".:../lib/cards.jar" G102
```

となります。

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。

[15] -cp の代りに -classpath としても構いません。

1.6 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`[16] `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												

[16] 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	