オブジェクト指向及び演習・第 8 回

2025 年 6 月 9 日

配布資料の内容

8.1 Object クラス

今回は、Java に備え付けられているいくつかの基本的なクラスを紹介します。まず最初は Object クラスです。Object クラスは java.lang というパッケージに含まれるクラスで、その完全限定名は java.lang.Object となります。java.lang は、Java のもっとも基本的なクラスなどを集めたパッケージです。Java のソースファイルの冒頭には (package 宣言の後に)、

```
import java.lang.*;
```

があると見なされることになっていますので、Object クラスを含めた java.lang のクラスは、ソースファイル中で、その単純なクラス名だけを書いて指定することができます。

Object クラスは、Java のすべてのオブジェクトに共通する基本的な状態と振る舞いについて記述したクラスです。Object クラスには、後述の toString メソッドなど、いくつかのインスタンスメソッドが定義されています。これらは、(サブクラスで再定義されない限りは) 他のすべてのクラスとすべての配列型に継承されます。Object クラスには、外部からアクセス可能なインスタンス変数、クラスメソッドやクラス変数は 1 つも宣言されていません。

Object クラスは、他のすべてのクラスの (直接あるいは間接の) スーパークラスとなります。

```
class クラス名 {
    ⋮
}
```

のように、(直接の) スーパークラスを指定せずにクラスを宣言すると、クラス名 の後に extends java.lang.Object があるものと解釈されます。たとえば、第 5 回で定義した Hand クラスは、この Object クラスの直接のサブクラスであり、Object クラスが Hand の直接のスーパークラスとなります。

メモ

Object クラスは、引数のない public なコンストラクタをただ 1 つ持っています。このコンストラクタは何の仕事もしません。Object クラス自身には (直接の) スーパークラスはありませんので、このコンストラクタは super(); の実行もしません。

Object 型　第 4 回で説明したように、Java では、クラスの名前はデータ型の名前でもあり、クラスを宣言することで生まれる型をクラス型と呼びます。一般に、クラス型は参照型の一種で、そのクラスおよびそのサブクラスのインスタンス (への参照) と null 参照を含む型となりますが、その中でも Object 型は特別な型で、すべてのオブジェクト (への参照) と null 参照を含む型となります。Java では配列もオブジェクトの一種ですから、すべての参照型の値 (すべてのクラスのインスタンスとすべての配列オブジェクト) と null 参照が Object 型に含まれることになります。このため、すべての参照型の値 (と null 参照) は Object 型の変数で扱うことができます。

メモ

8.2 String クラス

java.lang パッケージに属する String クラスは、Java で文字列を表現するために用意されているクラスです。String クラスのインスタンスは、それぞれ 1 つの文字列を表します。ある文字列を表す String クラスのインスタンスを一旦生成すると、そのインスタンスが表す文字列を後で変更することはできません[1]。また、String クラスは、クラス自身が final という修飾子を伴って宣言されており[2]、それを元にしてそのサブクラスを宣言することができないクラスとなっています。このようなクラスを final なクラスと呼びます。

文字列リテラル　C 言語と同様に、文字列を "" (二重引用符の対) で囲ったものを**文字列リテラル**と呼びます。Java の文字列リテラルは、その文字列を表すような String クラスのインスタンスを表します。C 言語と同様に、\ を使って、文字列リテラル中にいくつかの特殊な文字を含めることができます。

\b	バックスペース
\n	改行
\r	復帰
\'	一重引用符 (')

\t	水平タブ
\f	改ページ
\"	二重引用符 (")
\\	バックスラッシュ (\)

たとえば、次の 3 つはそれぞれ文字列リテラルです。

[1] String クラスのインスタンスのように、状態が変化することのないオブジェクトは**不変な** (immutable) オブジェクトと呼ばれます。

[2] final class String ··· のように宣言されています。

```
"Hello!"
"Hello!\nBye.\n"
"こんにちは。"
```

文字列の比較　文字列の比較は、String クラスのインスタンスメソッド

```
boolean equals(Object s)
```

を用いて行います。String クラスのインスタンスに対してこのメソッドを起動すると、引数 s に指定したオブジェクトが、自分と同じ文字列を表しているかどうかを判定して、その結果を boolean 型の戻り値として返してくれます。引数 s が String クラスのインスタンスでない場合は false が返されます。

たとえば、String 型の変数 str が "ABC" という文字列である時のみ行ないたい仕事がある場合、

```
if (str.equals("ABC")) {
    ⋮
}
```

のように書きます。== 演算子を使って 2 つの文字列を比較することはできないことに注意してください。String 型の 2 つの値に対して、== 演算子を適用すると、2 つのオペランドが同じ 1 つのオブジェクトを指しているかどうかが判定されます。異なる 2 つの String クラスのインスタンスが同じ文字列を表しているかどうかを == 演算子で調べることはできません。

メモ

文字列の連結と文字列変換　Java の文字列は + 演算子を使って連結することができます。たとえば、"ABC" + "DEF" という式を評価して得られる値は、"ABCDEF" という文字列 (String クラスのインスタンス) となりります。

Java の + は、数値の加算を行う演算子でもありますが、その 2 つのオペランドの両方またはいずれか一方が String 型の場合は、文字列を連結する演算子として働きます。オペランドの一方が String 型で、一方がそうでない場合、String 型でない方のオペランドが自動的に文字列に変換されてから、文字列の連結がされます。この型変換を**文字列変換**と呼びます。

たとえば、1.2 + 3 + "ABC" という式を評価すると、1.2 + 3 の部分がまず評価されて double 型の 4.2 という値になります。次に、この 4.2 という数値が文字列変換されて "4.2" という文字列となり、"ABC" と連結されて、最終的には "4.2ABC" という文字列となります。一方、"ABC" + 1.2 + 3 という式を評価すると、まず、1.2 という数値が文字列変換されてできた "1.2" が "ABC" に連結されて "ABC1.2" という文字列となり、さらに 3 を文字列変換した "3" が連結されて、最終

的には "ABC1.23" という文字列となります。もし "ABC4.2" という文字列にしたい場合は、"ABC" + (1.2 + 3) という式を書かなければなりません。

プリミティブ型の文字列変換　プリミティブ型の値は、次の表のように文字列変換されます。

型	値	文字列変換の結果
boolean	偽	"false"
	真	"true"
char	x	UTF-16 の文字コードが x であるような 1 文字
byte short int long	x	x の 10 進整数表記
float double	非数	"NaN"
	$-\infty$	"-Infinity"
	$+\infty$	"Infinity"
	-0.0	"-0.0"
	$+0.0$	"0.0"
	$\lvert x\rvert < 10^{-3}$	x の浮動小数点 10 進表記 (整数部 1 桁で E 付き)
	$10^{-3} \leqq \lvert x\rvert < 10^{7}$	x の固定小数点 10 進表記 (E なし)
	$10^{7} \leqq \lvert x\rvert$	x の浮動小数点 10 進表記 (整数部 1 桁で E 付き)

float 型や double 型の場合、小数部は (その型の隣接する値を区別するための) 必要最小限の長さ (ただし、少なくとも 1 桁) となります。

メモ

toString メソッド　null 参照を文字列変換すると、"null" という文字列となります。参照型の値が文字列変換される際には、そのオブジェクトに対して

```
String toString()
```

というインスタンスメソッドが起動され、その戻り値が文字列変換の結果となります。toString メソッドは Object クラスで宣言されていますので、すべてのオブジェクトがこのメソッドを持っていますが、クラスによっては、このメソッドを再定義している場合がありますので、その場合は、その新しい定義が使われることになります。

Object クラスの toString メソッドの戻り値は、"java.lang.Object@1befab0" のような文字列となります。この文字列は、そのオブジェクトのクラスの完全限定名と "@"、そのオブジェク

トのハッシュコードと呼ばれる整数値[3]の 16 進表記を連結したものです。

String クラスのその他のメソッド　String クラスには文字列処理のためのたくさんのインスタンスメソッドが用意されています。次の表はよく使われるもののみをまとめたものです。

String クラス — Java の文字列

主なコンストラクタ	
`String(char[] a)`	a の要素からなる文字列
`String(char[] a, int i, int n)`	a の添字 i 以降の n 個の要素からなる文字列
主なクラスメソッド	
`static String format(String format, Object... args)`	C 言語の sprintf と同様に、書式文字列 format に従って、args を文字列に直した結果を返す。
主なインスタンスメソッド	
`boolean contains(CharSequence[4] s)`	この文字列が s を部分文字列として含むかどうかを判定する。
`boolean endsWith(String s)`	この文字列が s で終わっているかどうかを判定する。
`boolean equals(Object s)`	s が、この文字列と同じ文字列を表す String クラスのインスタンスであるかどうかを判定する。
`int indexOf(String s)`	この文字列内で、初めて部分文字列 s が現れる位置 (文字列の先頭が 0) を返す。s が現れていない場合は -1 を返す。
`int length()`	この文字列の長さを返す。
`boolean startsWith(String s)`	この文字列が s で始まっているかどうかを判定する。
`String substring(int beg)`	この文字列の、beg の位置 (先頭が 0) 以降の文字からなる部分文字列を返す。
`String substring(int beg, int end)`	この文字列の、beg から end の手前までの位置 (先頭が 0) の文字からなる部分文字列を返す。
`String trim()`	この文字列の先頭と末尾の空白をすべてとり除いてできる文字列を返す。

メモ

[3]オブジェクトを区別しやすいように付けられたオブジェクトの番号のようなものですが、異なるオブジェクトが異なるハッシュコードを持つことは保証されていません。

[4]String 型の引数を渡すこともできます。CharSequence は String のスーパーインタフェースです。インタフェースやスーパーインタフェースについては、第10回で解説します。

8.3 System クラス

System クラスは、Java の実行環境に関するいくつかの有用なクラス変数やクラスメソッドを提供するクラスです。System クラスはコンストラクタを公開していませんので、このクラスのインスタンスを作ることはできませんし、インスタンスメソッドも存在しません。System クラスも java.lang パッケージに属する final なクラスです。

System クラス — Java の実行環境に関するクラス変数やクラスメソッドを提供

主なクラス変数	
`static final java.io.PrintStream err`	標準エラー出力ストリーム
`static final java.io.InputStream in`	標準入力ストリーム
`static final java.io.PrintStream out`	標準出力ストリーム
主なクラスメソッド	
`static void arraycopy(Object src, int srcPos, Object dst, int destPos, int num)`	配列 src の添字 srcPos 以降の num 個の要素を、配列 dest の添字 destPos 以降にコピーする。
`static void exit(int status)`	引数 status の値を終了コードとして、Java 仮想機械を終了する。
`static long currentTimeMillis()`	協定世界時 1970 年 1 月 1 日午前 0 時からのミリ秒単位の経過時間を返す。

メモ

標準入力、標準出力、標準エラー出力　第 1 回に作成した P101Hello.java では、main メソッドの中で、

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

という文を実行していました。この文が実行されることにより、プログラムの標準出力に Hello, world! とい文字列が表示されますが、ここで使われているのが、System クラスです。out は、この System クラスのクラス変数で、そこには java.io.PrintStream というクラスのインスタンスが格納されています。この java.io パッケージの PrintStream クラスのインスタンスは、次の表のようなインスタンスメソッドを持っており、P101Hello.java で起動している println もその 1 つです。このメソッドは、引数で指定した文字列と改行文字を出力します。PrintStream には C 言語の printf 関数と同様に、書式を指定して 数値などを出力できる同名のインスタンスメソッ

ドもあります。浮動小数点数の小数点以下の桁数などを指定したい場合は、この printf メソッドを利用すると便利です。

java.io.PrintStream クラス — いろいろな出力機能を持つオブジェクト

主なコンストラクタ	
PrintStream(String name)	name という名前のファイルに出力を行うインスタンスを作成する。ファイルが存在しなかったり、書き込みが許されていない場合は、FileNotFoundException 例外が発生する。
主なインスタンスメソッド	
void close()	出力先を閉じる。
void flush()	出力バッファをフラッシュする。
void print(boolean x) void print(char x) void print(int x) void print(long x) void print(float x) void print(double x) void print(Object x)	引数 x を文字列変換して得られる文字列を出力する。
void print(String s)	文字列 s を出力する。
void printf(String format, Object... args)	C 言語の printf と同様に、書式文字列 format に従って、args を文字列に直した結果を出力する。
void println()	改行文字を出力する。
void println(boolean x) void println(char x) void println(int x) void println(long x) void println(float x) void println(double x) void println(Object x)	引数 x を文字列変換して得られる文字列と改行文字を出力する。
void println(String s)	文字列 s と改行文字を出力する。

メモ

8.4 Math クラス

Math クラスは、指数関数、対数関数、三角関数などの基本的な数学関数をクラスメソッドとして提供するクラスです。Math クラスもコンストラクタを公開していませんので、このクラスのオブジェクトを作ることはできませんし、インスタンスメソッドを使用することもありません。Math クラスも java.lang パッケージに属する final なクラスです。

Math クラス — 数学関数や定数を提供するクラス

主なクラス変数	
`static final double E`	自然対数の底 e の近似値
`static final double PI`	円周率 π の近似値
主なクラスメソッド	

`static double abs(double x)`
`static float abs(float x)`
`static int abs(int x)`
`static long abs(long x)`
	引数 x の絶対値を返す。

`static double atan2(double y, double x)`	$\tan^{-1}\frac{y}{x}$ の値を返す。
`static double ceil(double x)`	引数 x 以上の最小の整数を返す。
`static double cos(double x)`	$\cos x$ の値を返す。
`static double exp(double x)`	e^x の値を返す。
`static double floor(double x)`	引数 x 以下の最大の整数を返す。
`static double log(double x)`	$\log x$ の値 (x の自然対数) を返す。
`static double log10(double x)`	$\log_{10} x$ の値 (x の常用対数) を返す。

`static double max(double a, double b)`
`static float max(float a, float b)`
`static int max(int a, int b)`
`static long max(long a, long b)`
	引数 a と b の最大値を返す。

`static double min(double a, double b)`
`static float min(float a, float b)`
`static int min(int a, int b)`
`static long min(long a, long b)`
	引数 a と b の最小値を返す。

`static double pow(double x, double y)`	x^y の値を返す。
`static double random()`	0.0 以上 1.0 未満の乱数を返す。
`static double rint(double x)`	引数 x に最も近い整数値 (偶数優先) を返す。
`static long round(double x)` `static int round(float x)`	引数 x に最も近い整数値 (大きい方の値優先) を返す。
`static double signum(double x)` `static float signum(float x)`	引数 x が負なら -1.0 を、0 なら 0.0 を、正なら 1.0 を返す。
`static double sin(double x)`	$\sin x$ の値を返す。
`static double sqrt(double x)`	$\sqrt{x}$ の値を返す。
`static double toDegrees(double rad)`	角度 rad ラジアンを度に変換した値を返す。
`static double toRadians(double deg)`	角度 deg 度をラジアンに変換した値を返す。
`static double tan(double x)`	$\tan x$ の値を返す。

メモ

8.5 ラッパークラス

Java で扱うことのできるデータ型には、プリミティブ型と参照型の 2 種類があります[5]が、この 2 つでは値の扱われ方が異なっています。プリミティブ型では、その値の表現を直接扱いますので、データを記憶する際に必要となるメモリを最小限に押さえることができます。また、その値の表現は計算機のハードウェアが直接扱うことができる形式であることが多いので、データを高速に処理することが可能です。

一方、すべての参照型では、データ自身はメモリのどこかに置いておいて、そのデータへの参照を扱うことになりますので、必要となるメモリ領域が大きくなったり、処理を行う際の手間が余計に掛かってしまいます。しかし、一方では、すべてを参照で扱うため、サブクラスのインスタンスをスーパークラスのインスタンスとして扱ったり、すべてのオブジェクトを `Object` 型として統一的に扱うことができるといった便利さがあります。

真理値や数値など、通常はプリミティブ型で扱うことの多いデータでも、場合によっては一般のオブジェクトと同じように扱いたくなることがあります。Java には、このような時に便利な**ラッパークラス** (wrapper classes) と呼ばれるクラス群が用意されています。ラッパークラスには、`Boolean`、`Character`、`Byte`、`Short`、`Integer`、`Long`、`Float`、`Double` の 8 つがあり、それぞれ、プリミティブ型の `boolean`、`char`、`byte`、`short`、`int`、`long`、`float`、`double` の値を表現するようなオブジェクトのクラスとなっています。各ラッパークラスは、`java.lang` パッケージに属するクラスですので、単純なクラス名だけを書いてそのクラスを指定できます。

たとえば、`Integer` クラスは次の表のようなクラスとなっています。

`Integer` クラス — `int` 型に対応するラッパークラス

主なクラス変数	
`static final int MAX_VALUE`	`int` 型で表現可能な最大の整数値
`static final int MIN_VALUE`	`int` 型で表現可能な最小の整数値
主なコンストラクタ	
`Integer(int x)`	`x` を表すインスタンス
`Integer(String s)`	10 進表記が文字列 `s` となる整数値を表すインスタンス
主なクラスメソッド	
`static int parseInt(String s)`	文字列 `s` が表す `int` 型の値を返す。`s` が 10 進表記の整数値を表していない場合は、`NumberFormatException` 例外が発生する。

主なインスタンスメソッド
`byte byteValue()`
`double doubleValue()`
`float floatValue()`
`long longValue()`
`int intValue()`
`short shortValue()`
	このインスタンスが表す数値を返す。

`Integer` 以外のラッパークラスも同様のコンストラクタやメソッドを持っています。ラッパークラスのインスタンスは、`String` クラスと同様に、ある値を表すオブジェクトとして生成される

[5]厳密に言うと、さらに null 型があります。

と、後からその表している値を変更することはできません。また、各ラッパークラスは final なクラスとなっています。

メモ

ボクシングとアンボクシング　あるプリミティブ型の値を、同じ値を表すそのラッパークラスのインスタンスに変換することを**ボクシング** (Boxing) と呼び、その逆を**アンボクシング** (Unboxing) と呼びます。各ラッパークラスには、そのラッパークラスに対応するプリミティブ型を引数とするコンストラクタが用意されていますので、これを利用して `new Integer(123)` のようなインスタンス生成式でボクシングを行うことができますが、`(Integer)123` のように**キャスト演算子**を使って行うこともできます。また、Java では、代入時やメソッドに引数として渡される際に、必要に応じて自動的にボクシングを行う仕組みが用意されていますので、たとえば

```
Integer p = 123;
```

と書くだけで、自動的に `int` 型の 123 という値を、同じ値を表す `Integer` クラスのインスタンスに変換してくれます。

ラッパクラスのインスタンスから、そのオブジェクトが表すプリミティブ型の値を取り出す (アンボクシングを行う) には、各ラッパクラスに用意されている

```
boolean booleanValue()
char charValue()
byte byteValue()
short shortValue()
int intValue()
long longValue()
float floatValue()
double doubleValue()
```

というインスタンスメソッド[6]を使うことができますが、キャスト演算子を使って行うこともできます。ボクシングの場合と同様に、代入時やメソッドに引数として渡したり、数値を対象とする演算 (四則演算など) を適用する際は、必要に応じて自動的なアンボクシングも行われます。

メモ

[6] `Byte`、`Short`、`Integer`、`Long`、`Float`、`Double` の 6 つのクラスは、すべて `byteValue`、`shortValue`、`intValue`、`longValue`、`floatValue`、`doubleValue` の 6 つのインスタンスメソッドを持っています。

PrintStream クラスの printf メソッド　java.io.PrintStream クラスには、printf というインスタンスメソッドがあり、C 言語の同名の関数と同じように書式を指定して数値などを出力することができます。この printf は

```
public void printf(String format, Object... args)
```

のように宣言されたメソッドです。仮引数の宣言の最後の部分が、Object... args のようになっていますが、この書式は、この部分の引数の数が可変であることを意味しています。printf の場合、ここに Object 型の引数が 0 個以上並ぶことを示しています。

printf を使うと、たとえば、

```
System.out.printf("%d, %.3f\n", 123, 1.0/3.0);
```

な文を書くことができます。このメソッド起動式では、int 型の 123 という値と、1.0/3.0 の計算結果である double 型の値が、それぞれ自動的にボクシングされて Integer クラスと Double クラスのインスタンスとなり、第 2 引数と第 3 引数として printf メソッドに渡されます[7]。これは、String クラスの format というクラスメソッドの場合も同様です。

メモ

8.6 演習問題

1. コマンドライン引数を使って

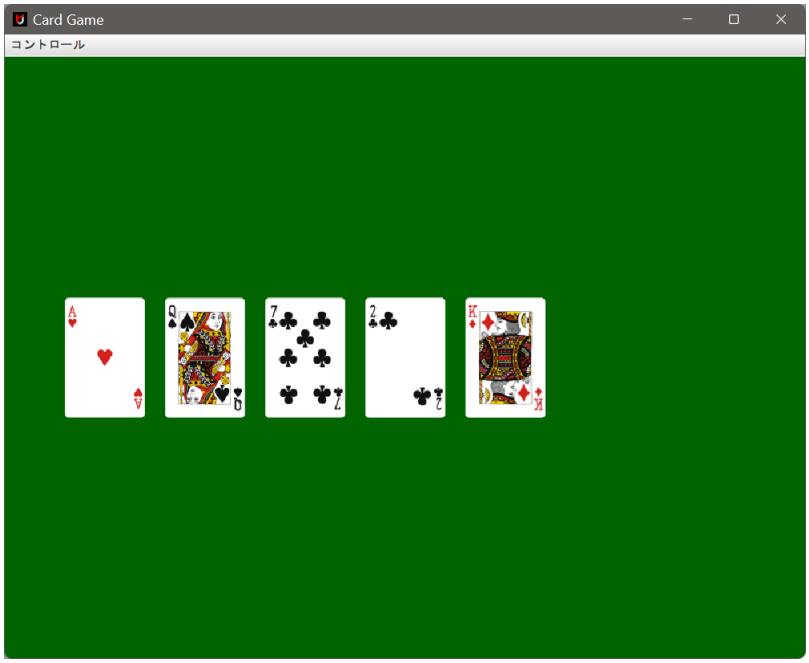
   ```
   C:\ ... \OOProg> java -cp "bin;lib\cards.jar" P801 H1 S12 C7 C2 D13
   ```

   や

   ```
   myname@mac OOProg % java -cp "bin:lib/cards.jar" P801 H1 S12 C7 C2 D13
   ```

   のように起動すると、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ようなプログラム P801.java を作成しなさい。

[7]実際には、この 2 つのオブジェクトを要素とする長さ 2 の配列が生成されて、それが仮引数 args の値として printf メソッドに渡されます。



カードのスートについては、スペード、ハート、ダイヤ、クラブをそれぞれ S、H、D、C の 1 文字で、ランクについては 1 から 13 までの整数 (10 進表記) で表し、スートとランクをつなげた文字列で 1 枚のカードを表すことにします。たとえば、H1 はハートのエース、S12 はスペードのクイーン、C7 はクラブの 7 を表すといった具合です。画面にカードを並べる際には、最初のカードを (60, 240) の位置に表向きに置き、そこから x 座標を 100 ずつずらすようにして、左から右へカードを表向きに並べてください。

Java プログラムが起動される際には、コマンドラインで指定した引数の文字列をを要素とする (String[] 型の) 配列オブジェクト生成され、これを引数として main メソッドが起動されます。たとえば、次のような main メソッドを持つプログラムを起動すると、コマンドライン引数を 1 行に 1 つずつコンソール (標準出力ストリーム) に出力します。

```
public static void main(String[] args) {
    for (String s : args) {
        System.out.println(s);
    }
}
```

引数の文字列の部分文字列は、String クラスのインスタンスメソッド substring を使って取り出すことができます。

2. コマンドライン引数で 2 つの数値を指定して

```
C:\ ... \OOProg> java -cp "bin;lib\cards.jar" P802 4.5 12.3
```

や

```
myname@mac OOProg % java -cp "bin:lib/cards.jar" P802 4.5 12.3
```

のように起動すると、画面中央に表向きに置いた (ジョーカーを含まない) デッキから、1 枚ずつカードを引いて、次の図のように螺旋状にカードを並べるプログラム P802.java を作

成しなさい。

コマンドライン引数で指定した 2 つの数値が、順に a と b であった場合、デッキから n 枚目に引いたカードの位置は $((an+120)\cos bn^\circ+360,\ (an+120)\sin bn^\circ+240)$ となるようにします。各カードの x 座標と y 座標は、それぞれ最も近い整数値に丸めるようにしてください。引数の文字列を実数値へ変換する際には、Double クラスのクラスメソッド

```
static double parseDouble(String s)
```

を使うことができます。

3. P802.java を改造して、main メソッドが起動されてからカードの配置が完了するまでの経過時間を、次の実行例のように小数点以下 1 桁の精度で表示するプログラム P803.java を作成しなさい。

```
C:\ ... \OOProg> java -cp "bin;lib\cards.jar" P803 4.5 12.3
カードの配置が完了するまでに 18.3 秒かかりました。
```

経過時間の測定には、System クラスのクラスメソッド currentTimeMillis を利用しましょう。

オブジェクト指向及び演習・第 8 回・終わり

8.7 付録: この科目のまとめ

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[8]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)

---

[8] `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。

などの宣言が記述される[9]。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]

  ただし、[オブジェクトを表す式] が this の場合[10]、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)]
}
```

[9] 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。

[10] かつ、仮引数や局所変数と名前が衝突しない場合。

- 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。
- 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[11]、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。

[11] `super.` を使ったインスタンスメソッドの起動は除く。

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[12]、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[13]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。

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。

---

[12] かつ、仮引数や局所変数と名前が衝突しない場合。

[13] コンストラクタの場合は、同じクラス名

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[14]。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[15]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。