

今回の内容

7.1 Object クラス	7-1
7.2 String クラス	7-2
7.3 System クラス	7-5
7.4 Math クラス	7-6
7.5 ラッパークラス	7-7
7.6 演習問題	7-9

7.1 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.*;
```

があると思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。

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、クラス型は参照型的一种で、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。

## 7.2 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<sup>1</sup>。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<sup>2</sup>、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。

<code>\b</code>	バックスペース	<code>\t</code>	水平タブ
<code>\n</code>	改行	<code>\f</code>	改ページ
<code>\r</code>	復帰	<code>\"</code>	二重引用符 (")
<code>\'</code>	一重引用符 (')	<code>\\</code>	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。

```
"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、

---

<sup>1</sup>`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。

<sup>2</sup>`final class String ...` のように宣言されています。

```

    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。

**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) となります。

Java の + は、数値の加算を行う演算子でもあります。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 真	"false" "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 $-\infty$ $+\infty$ $-0.0$ $+0.0$ $ x  < 10^{-3}$ $10^{-3} \leq  x  < 10^7$ $10^7 \leq  x $	"NaN" "-Infinity" "Infinity" "-0.0" "0.0" $x$ の浮動小数点 10 進表記 (整数部 1 桁で E 付き) $x$ の固定小数点 10 進表記 (E なし) $x$ 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 r も 1 桁)となります。

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となります。

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<sup>3</sup>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 <code>String(char[] a)</code> <code>String(char[] a, int i, int n)</code>	a の要素からなる文字列 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 <code>static String format(String format, Object... args)</code>	C 言語の <code>sprintf</code> と同様に、書式文字列 <code>format</code> に従って、 <code>args</code> 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 <code>boolean contains(CharSequence<sup>4</sup> s)</code> <code>boolean endsWith(String s)</code> <code>int indexOf(String s)</code> <code>int length()</code> <code>boolean startsWith(String s)</code> <code>String substring(int beg)</code> <code>String substring(int beg, int end)</code>	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。 この文字列が s で終わっているかどうかを判定する。 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。 この文字列の長さを返す。 この文字列が s で始まっているかどうかを判定する。 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。 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。

<sup>3</sup>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。

<sup>4</sup>String 型の引数を渡すこともできます。CharSequence は String のスーパーインターフェースです。インターフェースやスーパーインターフェースについては、次回以降に解説します。

<code>String trim()</code>	この文字列の先頭と末尾の空白をすべて取り除いてできる文字列を返す。
----------------------------	-----------------------------------

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

標準入力、標準出力、標準エラー出力 第 1 回に作成した `S0101.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!` という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、`S0101.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 <code>PrintStream(String name)</code>	<code>name</code> 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、 <code>FileNotFoundException</code> 例外が発生する。
--	--

<pre> 主なインスタンスメソッド void close() void flush()  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)  void print(String s) void printf(String format,     Object... args) void println() 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)  void println(String s) </pre>	<p>出力先を閉じる。</p> <p>出力バッファをフラッシュする。</p> <p>引数 <math>x</math> を文字列変換して得られる文字列を出力する。</p> <p>文字列 <math>s</math> を出力する。</p> <p>C 言語の <code>printf</code> と同様に、書式文字列 <code>format</code> に従って、<code>args</code> を文字列に直した結果を出力する。</p> <p>改行文字を出力する。</p> <p>引数 <math>x</math> 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。</p> <p>文字列 <math>s</math> と改行文字を出力する。</p>
--	--

## 7.4 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成す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

<pre> 主なクラス変数 static final double E static final double PI </pre>	<p>自然対数の底 <math>e</math> の近似値</p> <p>円周率 <math>\pi</math> の近似値</p>
<pre> 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)  static double atan2(double y, double x) static double ceil(double x) static double cos(double x) static double exp(double x) static double floor(double x) static double log(double x) static double log10(double x) 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) </pre>	<p>引数 <math>x</math> の絶対値を返す。</p> <p><math>\tan^{-1} \frac{y}{x}</math> の値を返す。</p> <p>引数 <math>x</math> 以上の最小の整数を返す。</p> <p><math>\cos x</math> の値を返す。</p> <p><math>e^x</math> の値を返す。</p> <p>引数 <math>x</math> 以下の最大の整数を返す。</p> <p><math>\log x</math> の値 (<math>x</math> の自然対数) を返す。</p> <p><math>\log_{10} x</math> の値 (<math>x</math> の常用対数) を返す。</p> <p>引数 <math>a</math> と <math>b</math> の最大値を返す。</p>

<code>static double min(double a, double b)</code> <code>static float min(float a, float b)</code> <code>static int min(int a, int b)</code> <code>static long min(long a, long b)</code>	引数 a と b の最小値を返す。
<code>static double pow(double x, double y)</code>	$x^y$ の値を返す。
<code>static double random()</code>	0.0 以上 1.0 未満の乱数を返す。
<code>static double rint(double x)</code> <code>static long round(double x)</code> <code>static int round(float x)</code>	引数 x に最も近い整数値を返す。
<code>static double signum(double x)</code> <code>static float signum(float x)</code>	引数 x が負なら $-1.0$ を、0 なら $0.0$ を、正なら $1.0$ を返す。
<code>static double sin(double x)</code>	$\sin x$ の値を返す。
<code>static double sqrt(double x)</code>	$\sqrt{x}$ の値を返す。
<code>static double toDegrees(double rad)</code>	角度 rad ラジアンを度に変換した値を返す。
<code>static double toRadians(double deg)</code>	角度 deg 度をラジアンに変換した値を返す。
<code>static double tan(double x)</code>	$\tan x$ の値を返す。

## 7.5 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<sup>5</sup>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となりますので、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できるデータでも、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場合があります。Java には、このような時に便利なラッパークラス (wrapper class) 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	
<code>static final int MAX_VALUE</code>	<code>int</code> 型で表現可能な最大の整数値
<code>static final int MIN_VALUE</code>	<code>int</code> 型で表現可能な最小の整数値

<sup>5</sup>厳密に言うと、さらに `null` 型があります。

主なコンストラクタ <code>Integer(int x)</code> <code>Integer(String s)</code>	<code>x</code> を表すインスタンス 10進表記が文字列 <code>s</code> となる整数値を表すインスタンス
主なクラスメソッド <code>static int parseInt(String s)</code>	文字列 <code>s</code> が表す <code>int</code> 型の値を返す。 <code>s</code> が10進表記の整数値を表していない場合は、 <code>NumberFormatException</code> 例外が発生する。
主なインスタンスメソッド <code>byte byteValue()</code> <code>double doubleValue()</code> <code>float floatValue()</code> <code>long longValue()</code> <code>int intValue()</code> <code>short shortValue()</code>	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。

**ボックスとアンボックス** 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ックス (boxing) と呼び、その逆をアンボックス (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ックス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ックス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。

ラッパー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 (アンボックスを行う) には、各ラッパー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<sup>6</sup>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。ボックスの場合と同様に、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

---

<sup>6</sup>`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。



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。

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);
```

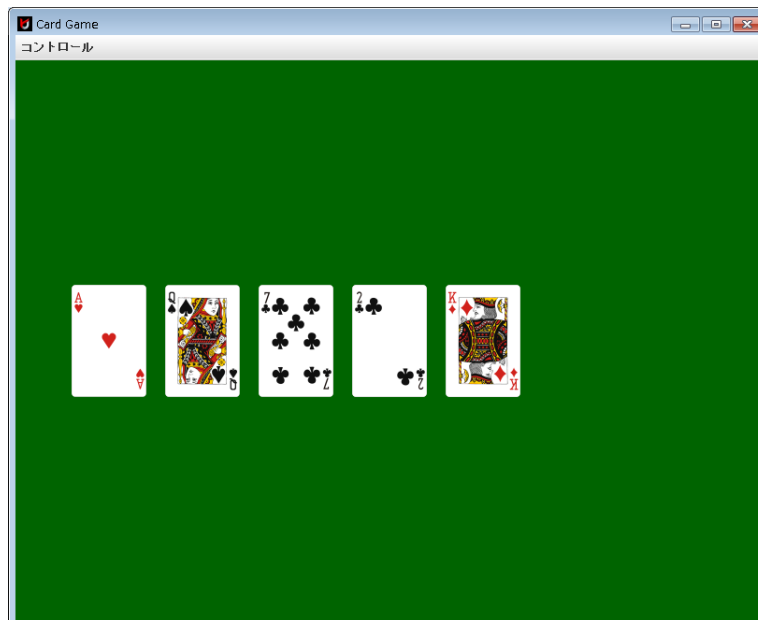
な文を書くことができます。このメソッド起動式では、int 型の 123 という値と、1.0/3.0 の計算結果である double 型の値が、それぞれ自動的にボクシングされて Integer クラスと Double クラスのインスタンスとなり、第 2 引数と第 3 引数として printf メソッドに渡されます<sup>7</sup>。これは、String クラスの format というクラスメソッドの場合も同様です。

## 7.6 演習問題

### 1. コマンドライン引数を使って

```
$ java S0701 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 S0701.java を作成しなさい。



<sup>7</sup>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。

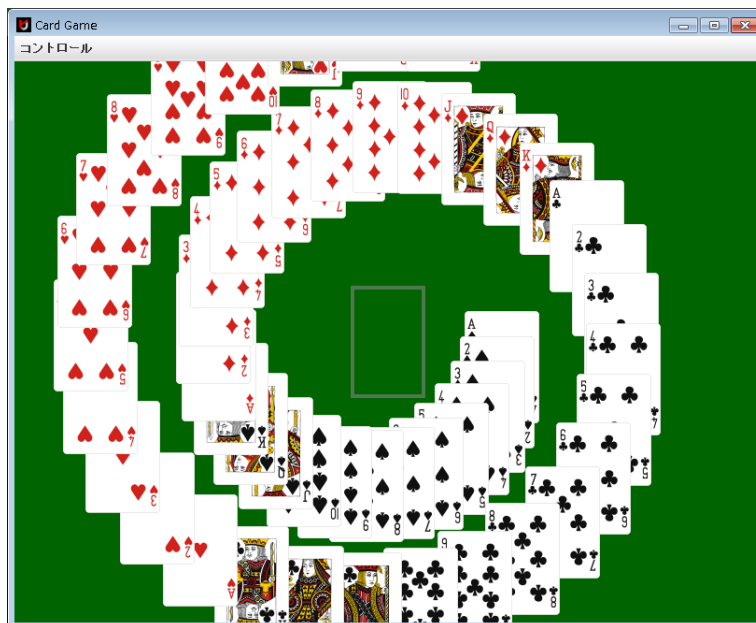
```
public static void main(String[] args) {
    for (String s : args) {
        System.out.println(s);
    }
}
```

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

```
$ java S0702 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 `S0702.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. `S0702.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 `S0703.java` を作成しなさい。

```
$ java S0703 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。