グラフィックス基礎・演習・第 8 回

2013 年 6 月 4 日

今回の内容

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。

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。

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	\t	水平タブ
\n	改行	\f	改ページ
\r	復帰	\"	二重引用符 (")
\'	一重引用符 (')	\\	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。

```
"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")) {
```

[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。

```
        ⋮
    }
```

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。

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 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。
`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。

[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。

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。

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。

標準入力、標準出力、標準エラー出力　第 1 回に作成した G101Hello.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、G101Hello.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。

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)`
`static long round(double x)`
`static int round(float x)`
	引数 `x` に最も近い整数値を返す。

`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。

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

[5]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
`Integer(int x)`
`Integer(String s)`
	
	x を表すインスタンス
	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

[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。

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);
```

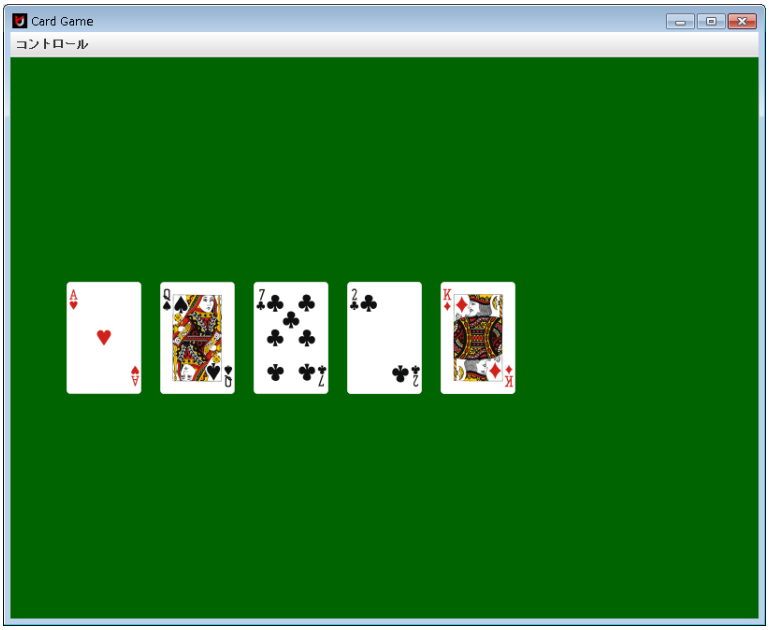
な文を書くことができます。このメソッド起動式では、int 型の 123 という値と、1.0/3.0 の計算結果である double 型の値が、それぞれ自動的にボクシングされて Integer クラスと Double クラスのインスタンスとなり、第 2 引数と第 3 引数として printf メソッドに渡されます[7]。これは、String クラスの format というクラスメソッドの場合も同様です。

8.6　演習問題

1. コマンドライン引数を使って

   ```
   $ java G801 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 G801.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

```
$ java G802 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 G802.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. G802.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 G803.java を作成しなさい。

```
$ java G803 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。

グラフィックス基礎・演習・第 8 回・終り