オブジェクト指向及び演習・第 4 回

2023 年 5 月 1 日

今回の内容

4.1 プリミティブ型と参照型

前回、Java では、オブジェクトを「参照」で扱う (変数に記憶したり、引数としてメソッドに渡したり、戻り値としてメソッドから返したりする) ことを説明しました。Java では、int 型や double 型のように、データの実体をそのまま扱うデータ型と、オブジェクトのように、データの実体への参照を扱うデータ型の 2 通りがあり、前者を**プリミティブ型**、後者を**参照型**と呼びます[1]。

プリミティブ型　Java のプリミティブ型には、次の 8 つの型があります。

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[2]。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[3]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。

[1] Java には、この他に null 型と呼ばれるデータ型があります。null 型は、後述の null 参照と呼ばれる値の型ですが、その型名がソースプログラム中に現れることがありませんので、その存在が意識されることはほとんどありません。

[2] boolean 型に関する演算子として、他にも、& (2 つのオペランドを必ず評価する論理積)、| (2 つのオペランドを必ず評価する論理和)、^ (排他的論理和) という 3 つの演算子があります。

[3] 世界で使用されているほとんどの文字を 1 符号単位 (16bit) で表現できる文字コードです。あまり使用されない文字のみ 2 つの符号単位 (16bit×2) で表現します。Unicode と呼ばれる文字コードの割り当て方法に基づいています。

参照型　Java の参照型は、オブジェクトを参照で扱うためのデータ型で、**クラス型**と**配列型**の 2 種類があります[4]。配列型については今回の後半で説明します。クラス型は、クラス宣言によってクラスを定義すると生まれるデータ型で、宣言されたクラスの名前はデータ型の名前の 1 つになります。

たとえば、前回までに出てきた `GameFrame`、`Card`、`Deck`、`Pile`、`Suit`、`Rank` は、すべてクラスの名前でしたが、これらは (クラス型、そして参照型に分類される) データ型の名前でもあります。

メモ

4.2　null 参照

Java では、すべてのオブジェクトを参照で扱いますが、どのオブジェクトも指していないこと示す値を、**null 参照**[5](あるいは単に null) と呼び、ソースプログラム中では、この値を `null`[6]というキーワードで表すことができます。

たとえば、`Card` クラスの各インスタンスは、そのカードのスートを表すオブジェクトを、`Suit` 型のインスタンス変数 `suit` に記憶していますが、ジョーカーにはスートがありませんので、ジョーカーの場合は、このインスタンス変数には null 参照が代入されています。つまり、

```
new Card().suit == null
```

という `boolean` 型の式の値は真 (`true`) となります[7]。

`NullPointerException`　すべての参照型の変数には、

```
Card c = null;
```

のように、null 参照を代入することができ、この変数がどのオブジェクトも指していないことを表現することができます。しかし、プログラム中で null 参照の値に対してできることは、`==` 演算子や `!=` 演算子を使って、`null` や他の参照型の値と等しいかどうかを調べることぐらいです[8]。もし、

[4]データ (値) の型としてはこれだけですが、式や変数の型としては、この他にも、**インタフェース型**や (**型変数**を含む) **総称型**と呼ばれる参照型が現れます。

[5]C 言語におけるポインタ型の 0 に対応します。

[6]C 言語の `stdio.h` や `stdlib.h` などで定義されている `NULL` に相当します。C の `NULL` はマクロ定義ですが、Java の `null` はキーワードとなっています。

[7]`new Card()` のようにコンストラクタの引数なしに、`Card` のインスタンスを生成すると、ジョーカーのカードが生成されることに注意してください。ジョーカーの `suit` は null 参照ですが、`rank` は通常のジョーカーと予備のジョーカーを区別するために、それぞれ `Rank.ACE` と `Rank.DEUCE` の値が代入されています。

[8]もちろん、代入演算子で変数などに代入したり、メソッドやコンストラクタの引数としたり、メソッドの戻り値にしたりすることも可能です。この他に、`instaceof` という演算子のオペランドとなることもあります。

null 参照が代入された変数 c を使って

```
c.moveTo(100, 100);
```

のように、インスタンスメソッドの起動を行ったり、

```
if (c.suit == Suit.SPADES)
    ...
```

のように、インスタンス変数へのアクセスを行ったりすると、moveTo という仕事を行ったり、suit を記憶しているオブジェクトがいない (指定されていない) わけですから、困ったことになってしまいます。Java では、プログラムの実行時に、null 参照に対してメソッドの起動やインスタンス変数へアクセスを行うと、NullPointerException と呼ばれる実行時エラーが発生して、通常、そのプログラムの実行はそこで中断されてしまいます。

P401.java

```
1 import jp.ac.ryukoku.math.cards.*;
2
3 class P401 {
4     public static void main(String[] args) {
5         Card c = null;
6         c.moveTo(100, 100);
7     }
8 }
```

たとえば、この P401.java をコンパイルして実行すると、コンソールに次のようなエラーメッセージが表示されます。

NullPointerException

```
C:\Users\myname\Desktop\OOProg> java P401
Exception in thread "main" java.lang.NullPointerException
        at P401.main(P401.java:6)
```

このエラーメッセージは、P401.java の 6 行目を実行する際に、NullPointerException が発生したことを示しています。Java では、この NullPointerException のように、プログラムの実行時に発生する予期しなかった出来事を一般に**例外 (exceptions)**と呼んでいます。

メモ

4.3 配列

C 言語と同様に、Java でも配列を使用することができます。次のプログラム P402.java は、シャッフルされたデッキから、伏せられたままのカードを 1 枚ずつ引いてゲーム盤の下部に横一列に並べ、5 枚並べたら左から順に表向きにするものです。

P402.java

```
 1 import jp.ac.ryukoku.math.cards.*;
 2
 3 class P402 {
 4     public static void main(String[] args) {
 5         GameFrame f = new GameFrame();
 6         Deck d = new Deck();
 7         f.add(d, 100, 100);
 8         d.shuffle();
 9         Card[] hand = new Card[5];
10         for (int i = 0; i < hand.length; i++) {
11             hand[i] = d.pickUp();
12             hand[i].moveTo(100 * i + 100, 400);
13         }
14         for (int i = 0; i < hand.length; i++) {
15             hand[i].flip();
16         }
17     }
18 }
```

このプログラムでは、デッキから引いた5枚のカードを記憶するために、Card 型の要素5個からなる配列を使っていますが、Java での配列の取り扱いは、C 言語とは若干異なるものとなっています。

C 言語では

```
int a[10];                          // C言語の場合
```

のように配列 a を宣言すると、int 型の要素 10 個からなる配列が用意されますが、これと同様なことを Java で行なうとしたら、

```
int[] a = new int[10];              // Java の場合
```

のような書き方になります。これは、変数 a の宣言と、int 型の要素 10 個からなる配列を生成して、その配列を a に代入するということの2つを一度に行うもので、これら2つを分けて書くと、

```
int[] a;
a = new int[10];
```

のようになります。

メモ

配列型　Java の配列はオブジェクトの一種であり、クラスのインスタンスと同様に、その配列への参照で取り扱います。T 型の要素からなる配列のデータ型を T[] で表します。たとえば、先ほどの int[] は、int 型の要素からなる配列のデータ型を表し、int[] a; は、int 型を要素とする

配列オブジェクト (への参照) を記憶する変数 a の宣言です[9]。同様に、Card[] は、Card 型の要素からなる配列のデータ型となり、P402.java の 9 行目のように、Card[] hand; と宣言された変数 hand は、Card 型を要素とする配列オブジェクト (への参照) を記憶します。

Java では、このような配列のデータ型を**配列型**と呼びます。int[] も Card[] も配列型の一種です。また、配列型は参照型の一種です。

配列の生成　配列型の変数を宣言しただけでは、配列自体は生成されません。配列を生成するには、次のような書式の**配列生成式**を用います。

new 要素の型名 [要素の個数]

配列生成式には、インスタンス生成式に現れていた new というキーワードが再び登場しています。配列生成式の 要素の個数 の部分には、int 型の式を書くことができます。定数はもちろん、変数を含んでいる式など、評価 (実行) してみないと値が決まらない式でも構いません。要素の個数 は 0 でも構いませんが、負であってはいけません。配列生成式を評価すると、指定された個数の要素からなる配列オブジェクトが生成され、その配列への参照が式の表す値となります。生成された配列の要素の個数を後で変更することはできません。

初期値を特に指定しない限り、生成された配列の各要素の値は、次のように初期化されます。

配列要素のデータ型	配列要素の初期値
boolean	偽 (false)
byte、short、int、long、char	(それぞれの型での) 整数値 0
float、double	(それぞれの型での) 浮動小数点数 0.0
すべての参照型	null 参照 (null)

また、配列生成式の 要素の個数 を書かずに [] とし、それに続く { } の中に、その配列の要素の初期値を , (カンマ) で区切って書いて、要素の個数と各要素の初期値を指定することもできます。このような初期値付きの配列生成式の書式は次のようになります。

new 要素の型名 [] { 初期値の列 }

たとえば、new int[] { 2, 3, 5, 7, 11 } という配列生成式は、int 型の要素 5 個からなる配列で、その要素の値が先頭から順に 2、3、5、7、11 であるようなものを生成します[10]。

メモ

[9] int a[]; と書くこともできます。こう書いても、int[] a; と全く同じ意味です。int[] a, b; と書けば、a も b も配列型の変数になりますが、int a[], b; と書くと、a は配列型、b は int 型の変数となります。

[10] 配列型の変数を初期値を指定して宣言するときに限って、たとえば、int[] a = { 2, 3, 5, 7, 11 }; のように、配列生成式の new 要素の型名 [] を省略することもできます。

配列の添字(インデックス)　配列の各要素は、C 言語と同様に

配列への参照を表す式 [添字]

という書式で表し、1 つの要素が 1 つの変数であるかのように、値を代入したり参照したりすることができます。長さ n の配列の要素の添字 (インデックス) は 0 から $n-1$ までです。添字 の部分にはこの範囲の整数を表す int 型の式を書くことができます。

配列への参照を表す式 の値が null 参照だった場合は、NullPointerException[11]という例外が、添字 が負であったり、n 以上であった場合は ArrayIndexOutOfBoundsException という例外が発生します。

配列のインスタンス変数 length　Java の配列はオブジェクトの一種でもあるわけですが、すべての配列オブジェクトは、length という名前の int 型のインスタンス変数を持っています。変数 length の値は、その配列の要素の個数 (配列の長さ) です。先ほどの P401.java では、10 行目と 14 行目で hand.length の形で使われています。

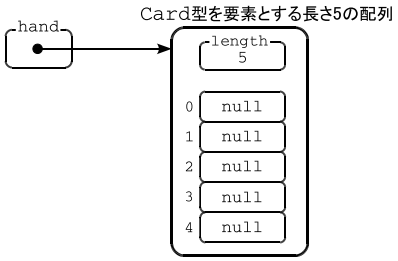
配列のインスタンス変数 length は書き換えることのできない変数となっています。Java では、一度初期化されると、その後は値を変更することができないような変数を使用することができ、このような変数を **final 変数**と呼びます。length は、配列オブジェクトが持つ final なインスタンス変数です。

メモ

P402.java での配列　先ほどの P402.java というプログラムで、配列がどのように使われているかを見てみましょう。9 行目の

```
Card[] hand = new Card[5];
```

が実行されると、Card 型を要素とする配列型 (Card[] 型) の変数 hand が用意されるとともに、5 個の Card 型の値を格納できる配列が生成されて、その配列への参照が hand に代入されます。この時の状態は右の図のようになります。

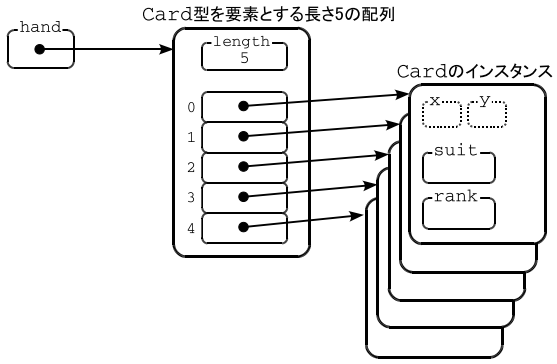


10 行目から始まる

```
for (int i = 0; i < hand.length; i++) {
    hand[i] = d.pickUp();
    hand[i].moveTo(100 * i + 100, 400);
}
```

[11]null 参照に対してインスタンスメソッドを起動したり、インスタンス変数にアクセスした場合にも発生する例外と同じです。

という for 文では、デッキ d から引いたカード (Card クラスのインスタンス) を、1 枚ずつ配列 hand の先頭から順に格納しています。この for 文の繰り返しが終わった時点では、5 枚のカードが、順に hand[0]、hand[1]、···、hand[4] に格納され、次の図のような状態となっているはずです。



メモ

4.4 配列の配列

配列型もデータ型の 1 つですので、配列型を要素とする配列を使用することもできます。たとえば、配列の各要素が、int 型の要素からなる配列型である場合、そのような配列のデータ型は int[][] で表すことができます。また、そのような配列への参照を記憶する変数は、

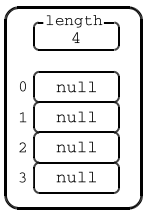
```
int[][] a;
```

のように宣言することができ[12]、このような配列で長さが n のものを生成するためには、

```
new int[n][]
```

という形の配列生成式を用います。この式で生成される配列の各要素のデータ型は int 型の配列型 (つまり int[] 型) ですが、各要素は右の図 ($n = 4$ の例) のように、すべて null 参照で初期化されますから、int 型のデータを記憶する配列は 1 つも生成されません。

new int[4][]で生成される配列

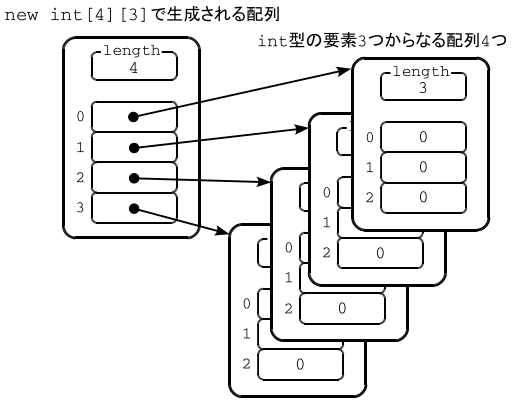


これに対して、

```
new int[n][m]
```

[12] int a[][]; や int[] a[]; と宣言しても同じ意味になります。

のような形の配列生成式を用いると、右下の図 ($n = 4$、$m = 3$ の例) のように、`int[]` 型の要素 n 個からなる配列とは別に、`int` 型の要素 m 個からなる配列が n 個作成されて、先の配列の要素が、これら n 個の配列への参照で初期化されます[13]。



つまり、たとえば、

```
int[][] a = new int[4][3];
```

というプログラムは、

```
int[][] a = new int[4][];
for (int i = 0; i < a.length; i++) {
    a[i] = new int[3];
}
```

というプログラムと等価になります。

C 言語では、2 次元や 3 次元など、多次元の配列を宣言することができましたが、Java では、そのような配列の代りに、配列の配列や、配列の配列の配列、$\cdots$ を使用します。それでも C 言語の場合と変らず、

a`[`n`][`m`]`

のような書式で特定の要素にアクセスすることができます。なぜなら、a が T 型の配列の配列 (T`[][]` 型の値) を表す式であれば、a`[`i`]` は、T 型を要素とする配列への参照 (T`[]` 型の値) を表す式となりますから、それに続いて `[`j`]` を書けば、a`[`i`]` が指す配列の添字 j の要素を表す式となるからです。

メモ

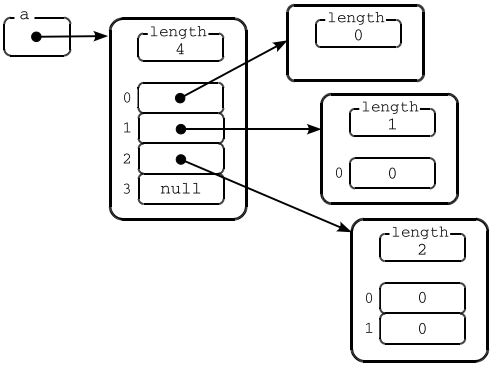
a が配列の配列の場合、a の要素 (配列型) の値は、一部 null 参照でも構いません。また、ちゃんと配列を指している場合でも、その配列の長さはばらばらで構いません。たとえば、

```
int[][] a = new int[4][];
for (int i = 0; i < 3; i++) {
    a[i] = new int[i];
}
```

というプログラムを実行すると次の図のような状態となります[14]。

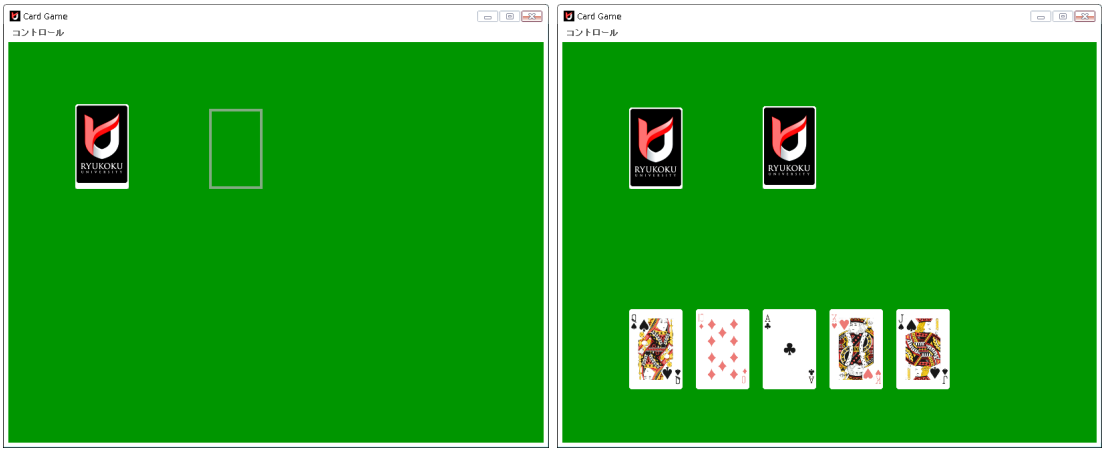
[13] `new int[][] { { 2, 3, 5 }, { 7, 11, 13 }, { 17, 19, 23 }, { 29, 31, 37 } }` のように初期値を書いて、`int` 型の各要素を初期化することもできます。また、変数の宣言時に、`int[][] a = { { 2, 3, 5 }, { 7, 11, 13 }, { 17, 19, 23 }, { 29, 31, 37 } }` のように書くこともできます。

[14] `a[0]` が指す配列は長さが 0 となっています。



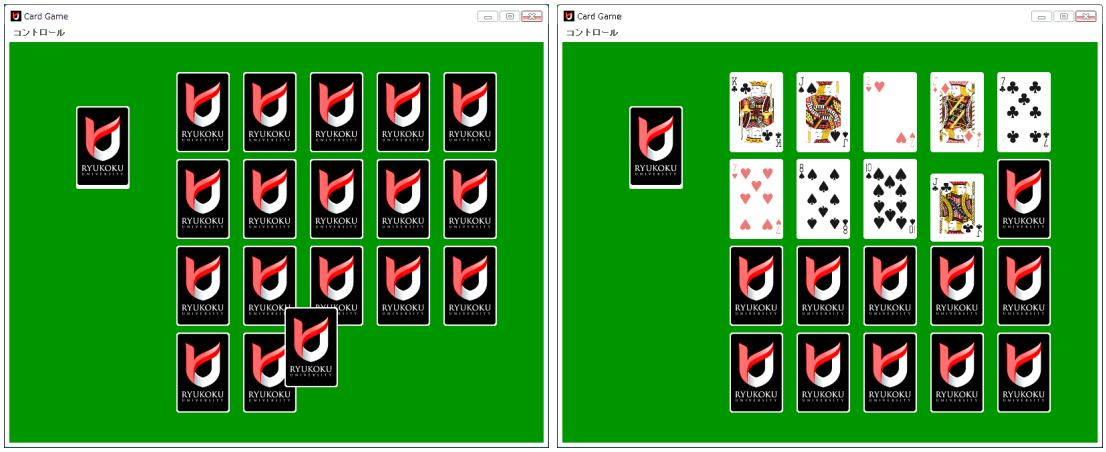
4.5 演習問題

1. 次のようなプログラム `P403.java` を作成しなさい。このプログラムでは、まず、次の図の左のように、ゲーム盤の (100, 100) の位置にジョーカーを含まないデッキを、(300, 100) に空の山 (`Pile` クラスのインスタンス) を置き、デッキはシャッフルします。その後、デッキの 1 番上のカードから順に 1 枚ずつ引いて表向きにし、ランクが 10、J、Q、K、A のいずれかであれば、ゲーム盤の下部に移動し左から右へ並べていきます。これらのいずれのランクでもなければ、右隣の山へ裏向きにして追加します。また、10、J、Q、K、A のいずれかであっても、同じランクのカードをゲーム盤の下部にすでに並べている場合は、やはり右隣の山へ裏向きにして追加します。ゲーム盤の下部に 10、J、Q、K、A の 5 つのランクのカードが 5 枚揃ったら仕事を終わります。



最終的にはゲーム盤は図の右のような状態になります。ゲーム盤の下部に残す 5 枚のカードの x 座標は、デッキから引いた順に 100、200、300、400、500 です。y 座標は 5 枚とも 400 です。

2. 次のようなプログラム `P404.java` を作成しなさい。このプログラムは、シャッフルされたデッキから、伏せられたままのカードを 1 枚ずつ引いて、4 行 5 列の行列状に並べ、20 枚のカードを伏せたまま並び終えたら、左上のカードから順に表向きにするものです。次の図の左はカードを並べている途中、右はカードを表向きにしている途中の状態です。



カードを並べたり、表向きにする際には、左上角のカードから始めて右へ向かい、右端に達したら、1 つ下の行へ行って、また左端から右端へ進むようにしてください。デッキの位置は (100, 100) です。4 行 5 列の行列状に並べるときのカードの x 座標は、左から順に、250、350、450、550、650 です。y 座標は、上から順に 45、175、305、435 です。

行列状に並べたカードを記憶するためには、`Card` クラスの 2 次元の配列 (`Card` 型を要素とする配列の配列) を使ってください。

オブジェクト指向及び演習・第 4 回・終わり

4.6 付録:配列のための for 文

Java では、C 言語と同じ形の for 文の他に、配列の各要素を先頭から順に参照して行くのに便利な、次の書式の for 文を使用することができます。

for (型名 変数名 : 配列を表す式) 文

この書式の 配列を表す式 のデータ型が T[] 型であるとすると、この形の for 文は、

```
{
    T[] a = 配列を表す式;
    for (int i = 0; i < a.length; i++) {
        型名 変数名 = a[i];
        文
    }
}
```

と等価な文となります。ただし、a や i という変数の名前は、他の変数と名前が重ならないように適当に選ばれます。型名 は、型名 変数名 = a[i]; が正しい変数宣言となるようなものにしなければなりません[15]。通常は、配列の要素の型である T と同じになります。

たとえば、P402.java の 14 行目から始まる

```
for (int i = 0; i < hand.length; i++) {
    hand[i].flip();
}
```

という for 文は、この書式を用いると、

```
for (Card c : hand) {
    c.flip();
}
```

のように簡潔に書くことができます。

ただし、この形の for 文では、配列の要素の値を参照することはできても、配列の要素に代入することはできないことに注意が必要です。このため、たとえば

```
for (int i = 0; i < hand.length; i++) {
    hand[i] = d.pickUp();
}
```

という for 文を

```
for (Card c : hand) {
    c = d.pickUp();        // 配列の要素は書き換えられない!
}
```

とすることはできません。なぜなら、c = d.pickUp(); で書き換えられるのは、(配列のある要素の値が代入された) 変数 c の値であって、配列の要素自体ではないからです。つまり、配列 hand の各要素の値は全く変化しないことになります。

---

[15] たとえば、a が int[] 型である場合、int 型の値は double 型の変数に代入することができますので、for (double x : a) 文 のような for 文を書くことができます。