(7) プログラムの再利用と継承

(7．1)
プログラムの部品化

他人が作ったプログラムの利用について考えます。

(7．2)
クラスをそのまま利用する

他人が作ったクラスをそのまま利用する方法を見ます。

(7．3)
Javaでの継承のしくみ

Javaでの継承について説明します。

(7．4)
継承のモデリング

モデルの観点で継承が何を意味しているのかを考えます。

(7．5)
クラス図での継承

クラス図での継承を、ブラックジャックの例を交えて見て見ます。

飛行機オブジェクト

(7．1．1) 他人のプログラムを再利用したい！

■人のプログラムを利用することは、オブジェクト指向以前にもありました。もっとも単純なプログラムの再利用を考えます。誰かが3角形の体積を求めるプログラムを作ったとします。

■プログラムは、入力パラメータに“底辺”、“高さ”を入力して、戻り値として“体積”を返す関数として作られています。これを再利用することは、簡単です。

昔のプログラム

```
# 他人のプログラム
int sankaku(int 底辺,int 高さ){
  return(底辺*高さ/2);
}
```

再利用

新たに作るプログラム

```
# 自作のプログラム
  taiseki = sankaku(底辺、高さ);
```

(7．1．2) ライブラリ

■前スライド(7.1.1)のようなプログラムの再利用は、“ライブラリ”と呼ばれ、昔から行われていました。

■数学の計算のように、プログラムで何を行うかが一般的に認められる形で明確に定義出来る場合には、今でもライブラリ式のプログラム再利用は有効です。

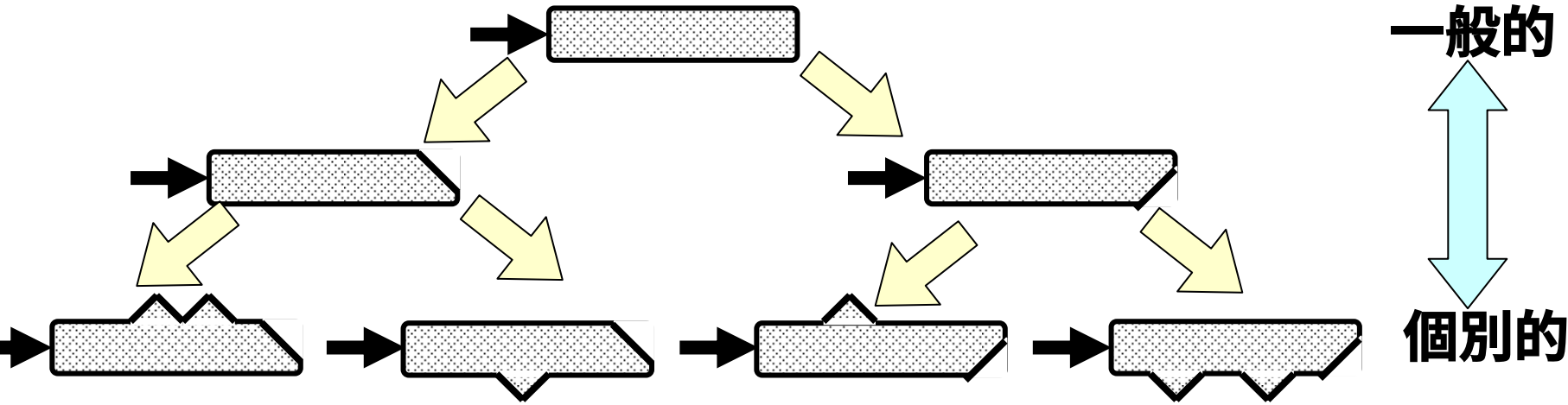
■逆に言うと、ライブラリでは少しでも仕様が異なると再利用が難しくなるという欠点があります。

(7．1．3) 一般化･個別化

■再利用されることを意図したプログラムは、なるべく利用されやすくするために、一般的 (共通的) なものとすることが普通です。

■しかしながら、世の中の実際のシステムの目的は多様であり、一般的なままでは使えないことが普通です。

■ライブラリにも、パラメタ指定やエントリ登録で個別の目的に対応しようとする仕掛けはありますが、そのような仕掛けが用意されたものにしか個別化はできません。



システム (ソフトウェア) の目的

(7．1．4) 個別に書き換えること

■「少しだけ違って再利用出来ないプログラムは、違う部分だけ書き換えて(書き足して)使えば良いのではないか？」と思われる方がいらっしゃると思います。

■実際、そのとおりです。

■しかしながら、書き換えについての枠組みが無いと、何がどのように書き換えられたかが不明瞭な類似のプログラムが氾濫することになり、混乱を招きます。

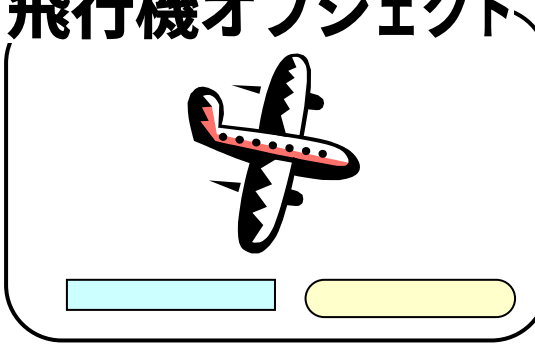
(7．1．5) オブジェクト指向での部品化

■部品を集めて機械を組み立てるように、プログラムを部品のように再利用することが出来ることが、プログラム再利用の目的です (これを“プログラムの部品化”と言います)。

■オブジェクト指向では、次の2つの柱により、プログラムの部品化を図ります。

■ (1) クラスを単位としたプログラムの再利用

- スライド(2.4.3)で見たとおり、設計者にとって自然な対象であるオブジェクトを再利用の単位 (部品) とすることで、利用する側での理解が容易となり、また、同じ考えに基づく部品が流通することになります。



■ (2) 継承

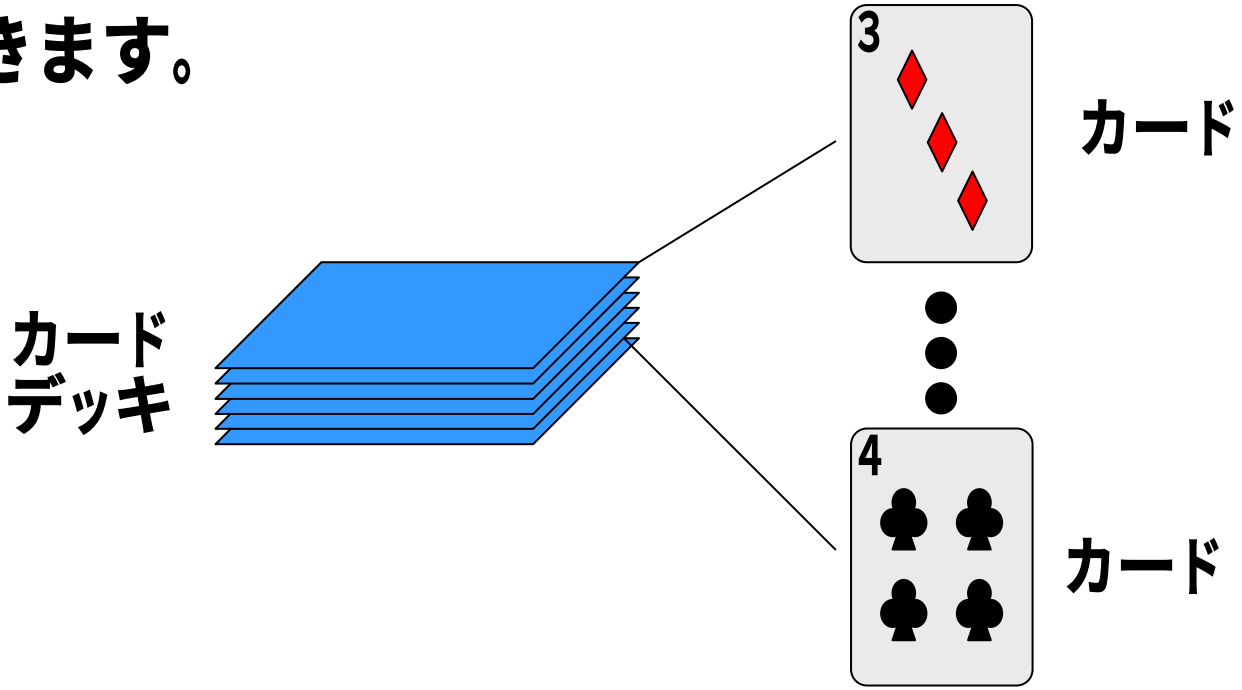
- これがスライド(6.1.4)の問題に答えるものであり、以下のスライドにて説明していきます。

(7．2) クラスを単位としたプログラムの再利用

■本節 (7.2) 節では、まず、他人の作ったプログラムを呼び出す形で**そのまま利用する**場合について考えます。

■このような場合も、クラスを単位として行います。

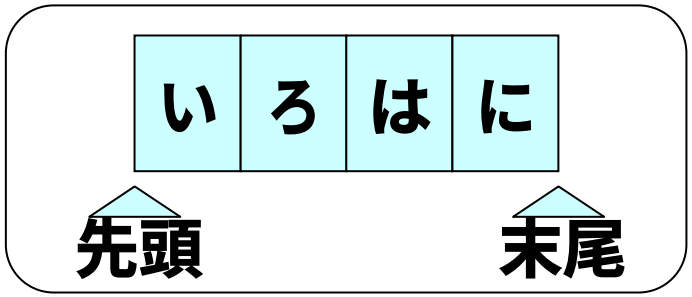
■ブラックジャックの“カードデッキ” について、新しいプログラムをダウンロードして、その内容を例として考えていきます。

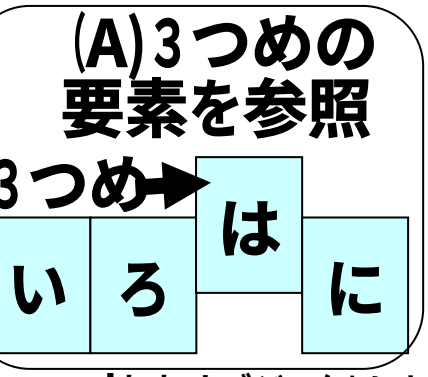


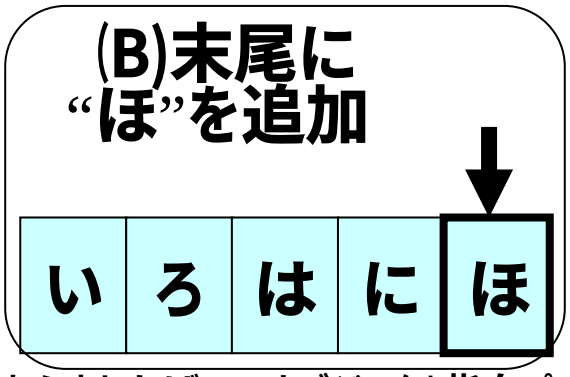
(7．2．1) 必要な機能

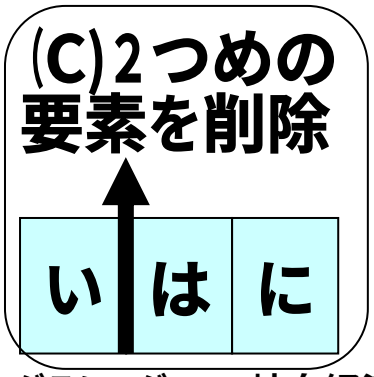
■カードを積んでおくカードデッキについては、次のような機能が必要になります。

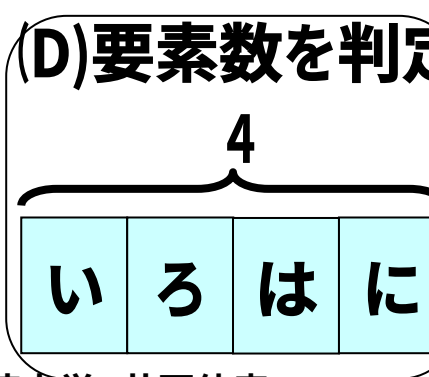
- (A)配列として、任意の位置の要素が参照できる。
- (B)末尾に要素を付け加えることが出来る (＝可変長である)。
- (C)指定した要素を取り除くことが出来る。
- (D)全体の要素数を判定できる。









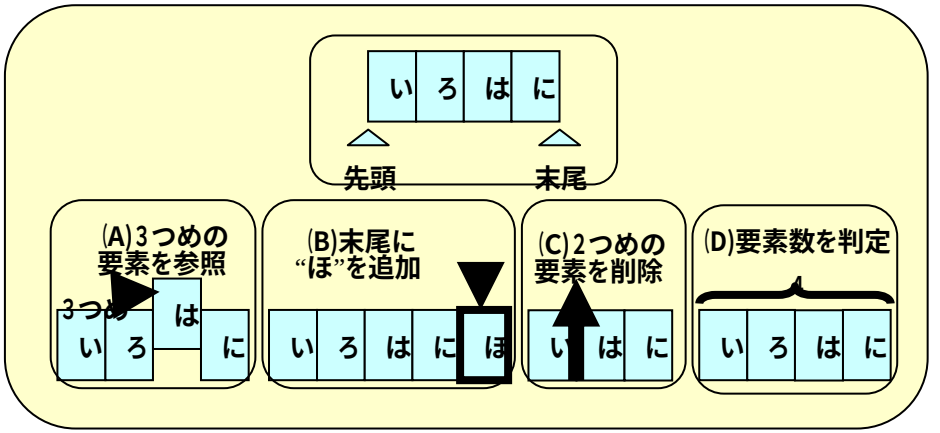


(7．2．2) よく使いそうなもの

■以前配布したプログラムでは、配列を使って、前スライド (7.2.1) に記したような機能を実現していました。

■しかしながら、このような機能はカードデッキに限らずよく使いそうな機能です。

■“よく使いそうなもの”は、自分で作らず、既に誰かが作ったものを利用するというのが、オブジェクト指向・Javaの流儀です。



(7．2．3) ArrayList

■実際、スライド(7.2.1)に記したようなクラスは、作らなくてもすでに用意されています。

■そのクラスは、“ArrayList”と言います。

■これを使う際には、Javaのプログラムファイルの先頭近くで次のように使うことを宣言さえすればOKです。

```
import java.util.ArrayList;
```

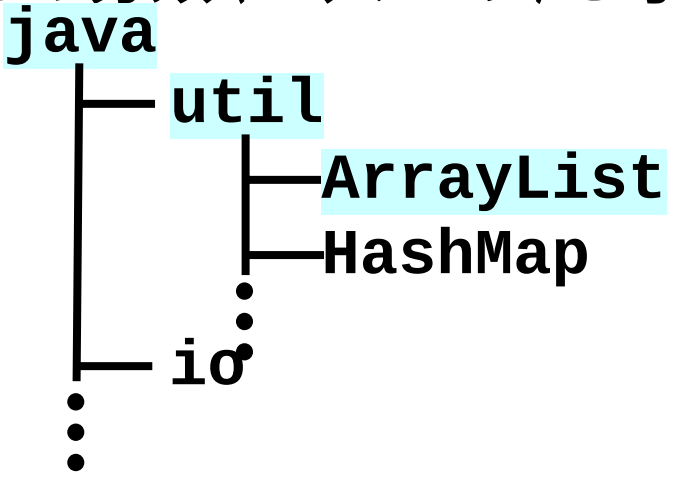
(あるいは、”java.util.ArrayList”と指定します)

■自分で作らずに、まさに“import (輸入) ”する訳です。

(7．2．4) 標準クラスライブラリ (パッケージ)

■Javaが使える環境 (SDK) では、このように輸入して使えるクラスが、系統的に分類・整理され、標準で用意されています。これを標準クラスライブラリ (標準パッケージ) と言います。

■“import”した、“java.util.ArrayList”の“java.util”は、この分類 (パッケージ) と考えてください。



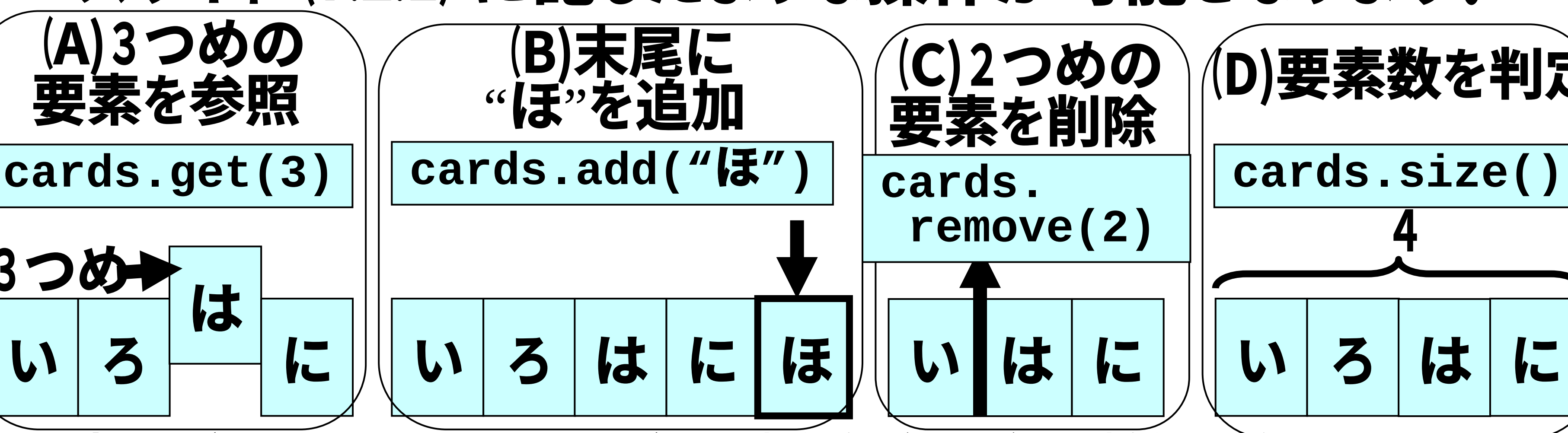
■標準パッケージに収められているのは、Javaでプログラムを作成するのに必要となるクラスです。よく利用されるパッケージには、“java.lang”、“java.applet”、“java.awt”などがあります。

(7.2.5) どうやって使うか？

■インポートしたクラスは、自分で作ったクラスと同様に、インスタンスを生成することが出来ます。

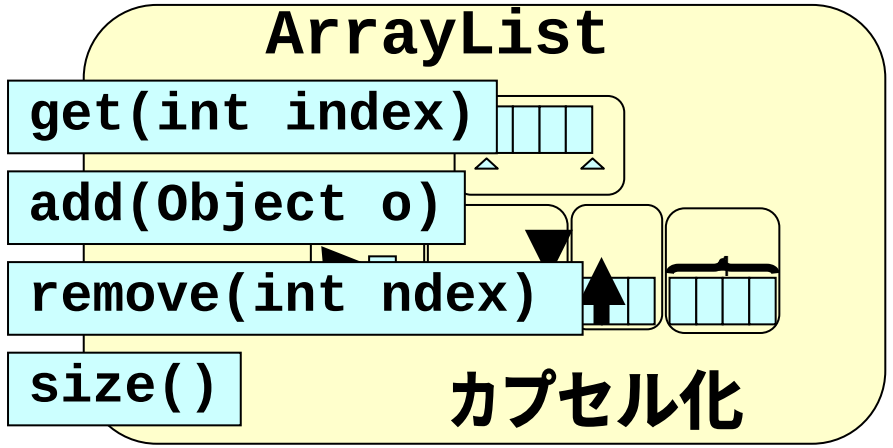
```
import java.util.ArrayList;
    :
 ArrayList cards = new ArrayList();
    :
```

■“ArrayList”として生成された“cards”に対して、スライド(7.2.1)に記したような操作が可能となります。



(7．2．6) 使い方を調べる

■“ArrayList”を使う際、
カプセル化されたその
中身を知る必要は
ありません。
どのように使うかだけを
知れば十分です。



■もちろん多くの参考書に標準クラスの使い方は解説されていますが、Web上で整理されたドキュメントが完備しています（実際はこれが原本となっています）。

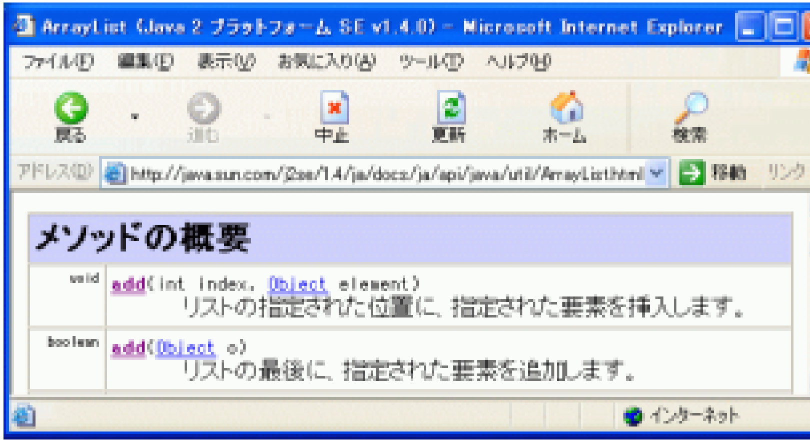
■次のURLにアクセスして、“ArrayList”について調べてみましょう。

- http://java.sun.com/j2se/1.4/ja/docs/ja/api/java/util/ArrayList.html

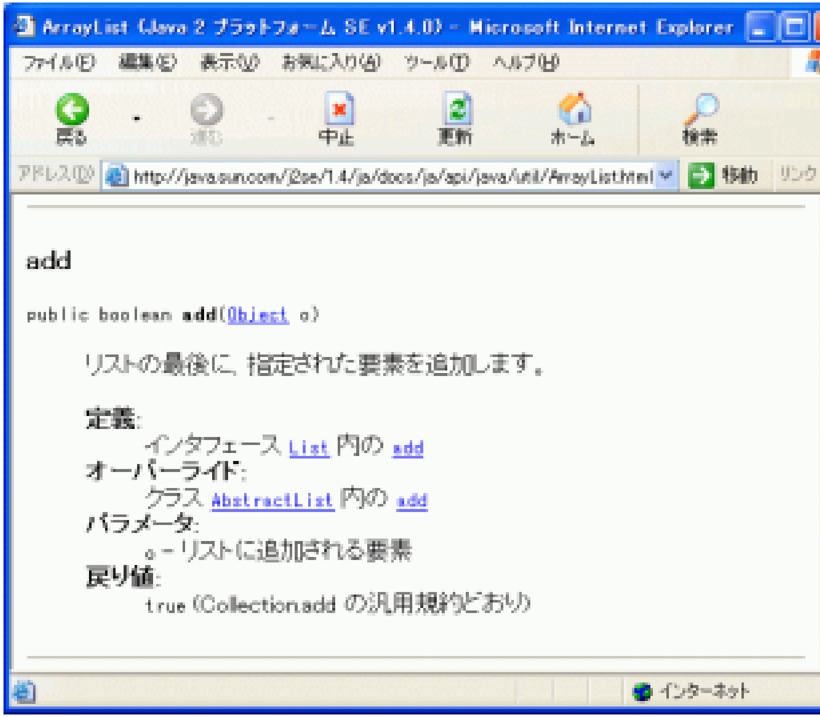
（Google等で、“java ArrayList”と検索すれば、上記のページは先頭で出てきます。）

（7．2．7）Web上のクラスの説明

- ■サイトにアクセスしてみると、中段くらいにメソッドの一覧があります（addやsize以外にも、沢山のメソッドがあります）。



- ■サイトの下段には、メソッド“add”の説明も見えます。



- ■Javaオブジェクト指向スライド（8）（9）参照。

(7．2．8) カードデッキ　ー1ー

■前半部分 (シャッフル) を見てみます。

```
package blackjack;
import java.util.ArrayList;
public class CardDeck {
  // Associations
  protected ArrayList cards = new ArrayList(52);
  public void shuffle() {
      for(int i = 1; i <= 13; i++) {
         Card spade = new Card(Card.SPADE, i);
         Card clover = new Card(Card.CLOVER, i);
         Card dia = new Card(Card.DIA, i);
         Card heart = new Card(Card.HEART, i);
         cards.add(spade);
         cards.add(clover);
         cards.add(dia);
         cards.add(heart);
      }
  }
```

“ArrayList”をインポートする。

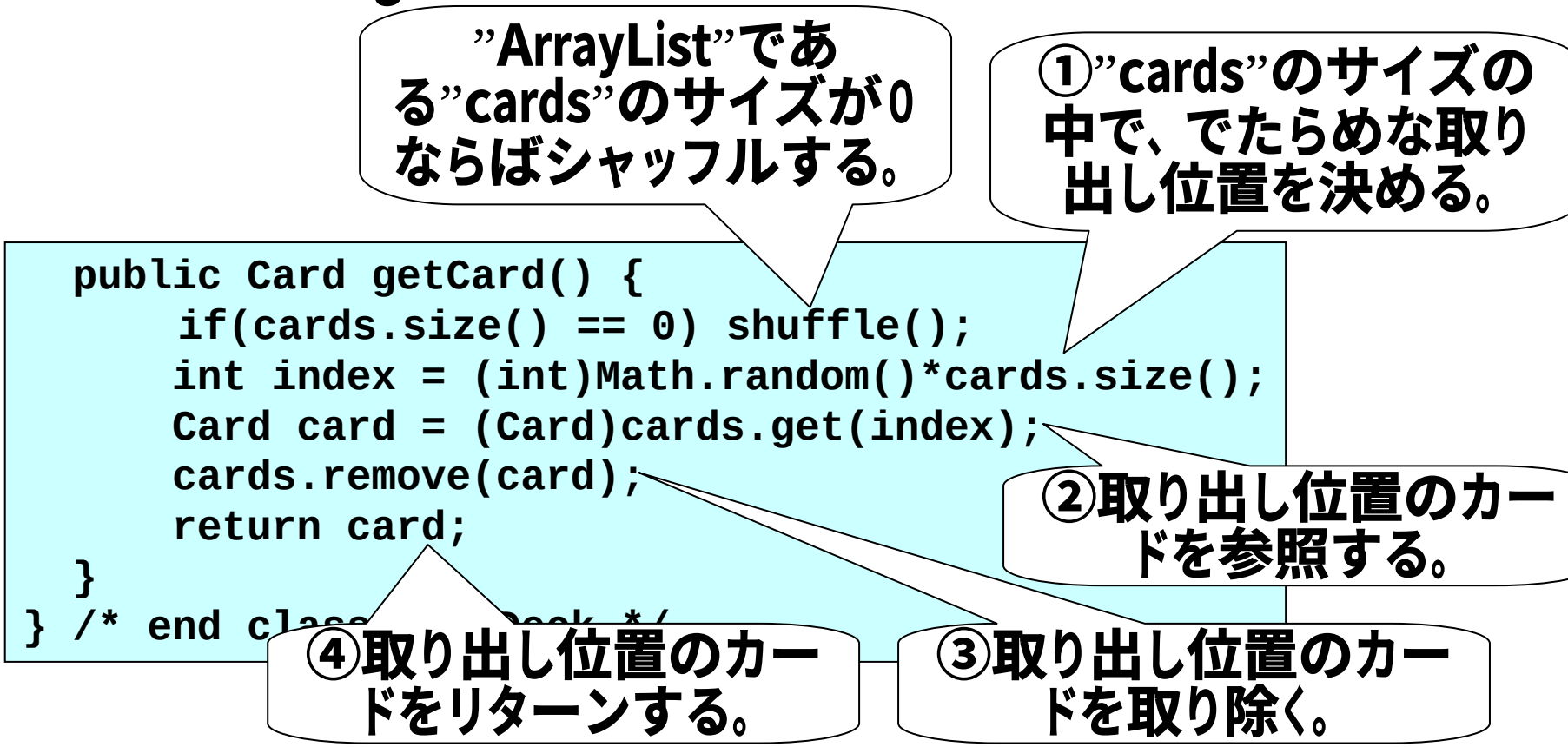
①1から13までのそれぞれの数について、

②スペード、クローバ、ダイア、ハートの4つのカードを生成し、

③”ArrayList”である”cards”の末尾に付け加えている。

(7．2．9) カードデッキ　ー2ー

■後半部分 (getCard) を見てみます。



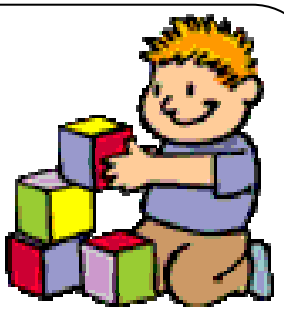

```
  public Card getCard() {
       if(cards.size() == 0) shuffle();
       int index = (int)Math.random()*cards.size();
       Card card = (Card)cards.get(index);
       cards.remove(card);
       return card;
  }
} /* end class ...Deck */
```

■“ArrayList”を使うことで、簡単に実現されていることが解ると思います。

(7．2．10) 小さな整理

■Javaでは、自分で作るよりも、他人が作ったプログラムをクラスとして利用するのが流儀です。

■よく使うクラスを含め、多くのクラスが標準パッケージとして、Javaが使える環境では具備されています。Javaでプログラムを作成するには、これらを使うことが必要です。



■この節 (7.2) では、他人の作ったプログラムを呼び出す形で「**そのまま利用する**」場合を見てきました。次の (7.3) 節から、「**そのままでは利用できない**」場合に使われる方法である、“継承”について見ていきます。

(7．3) Javaでの継承の仕組み

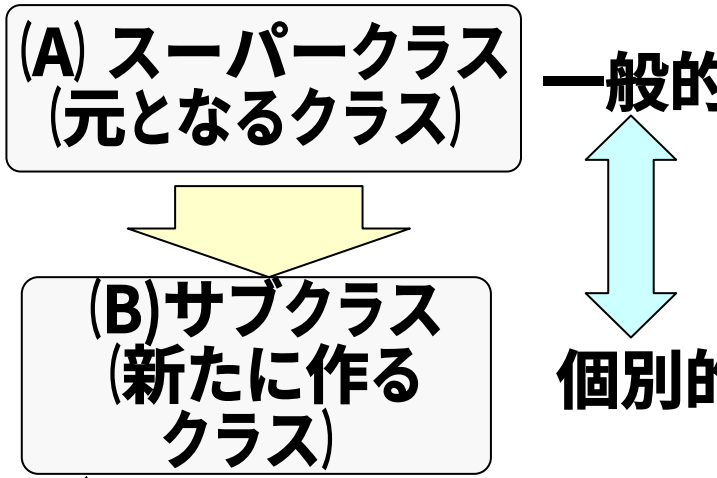
■元となるクラスを利用して、新しいクラスを作る仕組みのことを、継承(inheritance)と言います。

■元になるクラスをスーパークラス、新しく作成されるクラスをサブクラスと言います。スーパークラスを再利用してサブクラスを作り、これを使おうという訳です。

■スーパークラス“A”と、そのサブクラス“B”との関係は、次のような言い方で表されます。

- BはAを拡張している。
- BはAを継承している。
- AはBの汎化である

(これはUMLでの言い方、後述)。



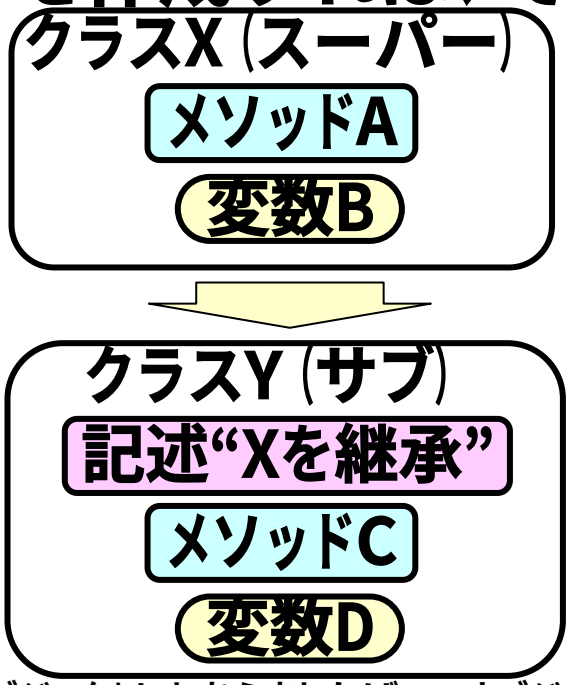
■Javaオブジェクト指向スライド(10)

(7．3．1) 書き足す

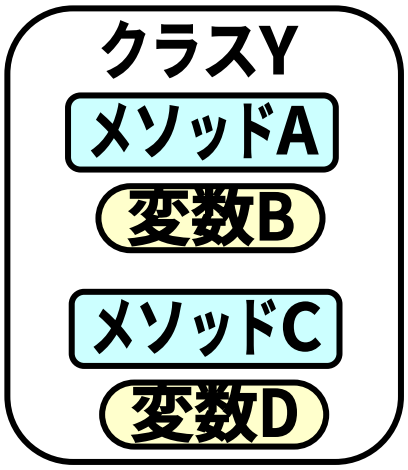
■クラスは、変数とメソッドとから成ります。

■クラスXをクラスYが継承しているとします (すなわち、クラスXがスーパークラス、クラスYがサブクラスです) 。

■このとき、クラスXの変数とメソッドとはクラスYにも記述されていることと同じになります。クラスYに変数とメソッドを作成すれば、それは書き足されたことになります。



クラスYは、



クラスXに
書き足した
ことになる

と同じ働きをする。

(7．3．2) 書き足しのJavaでのイメージ

■Javaのコーディングでは、“extends(拡張する)”の予約語を用いて、継承することの記述を行います。

```
# クラスX(スーパークラス)
class classX{
  int b;
  void methodA(){
        :
  }
}

#クラスY(サブクラス)
class classY extends classX{
  int d;
  void methodC(){
        :
  }
}
```

“classXを継承する”という記述

クラスclassYは、

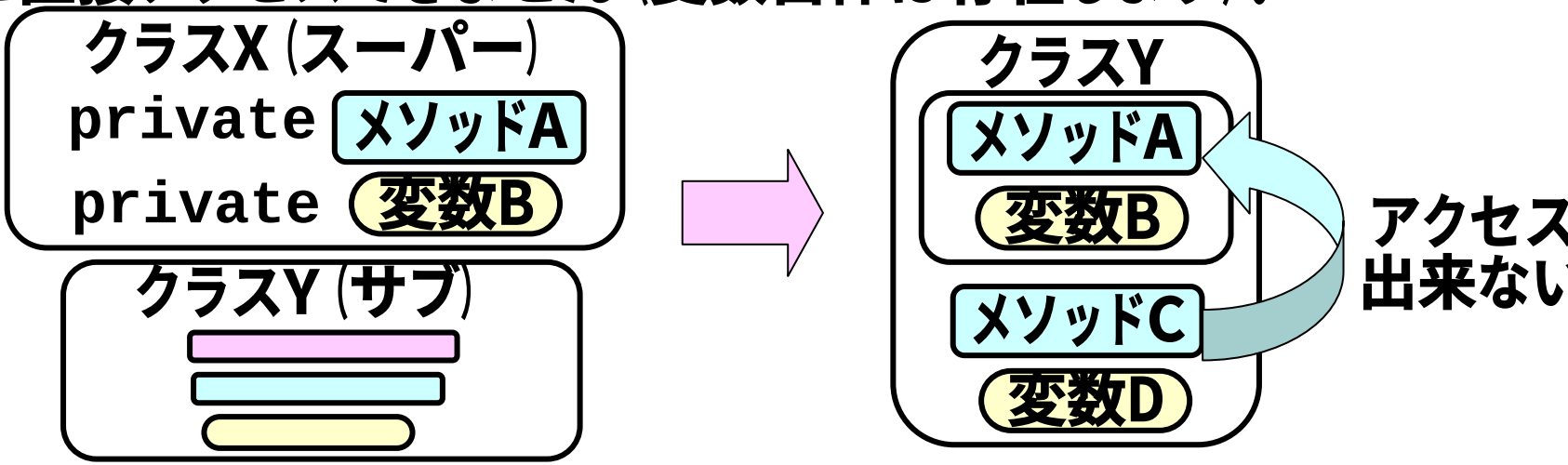
```
#クラスY
class classY{
  int b;
  int d;
  void methodA(){
        :
  }
  void methodC(){
        :
  }
}
```

と同じ働きをする。

(7．3．3)“private”に関する注意

■スライド(7.3.1),(7.3.2)では直感的な理解のために、やや荒っぽい説明をしました。

■実際のコーディングでは、スーパークラスの変数・メソッドに“private”の指定があると、カプセル化によりサブクラスではこれに直接アクセスできません(変数自体は存在します)。

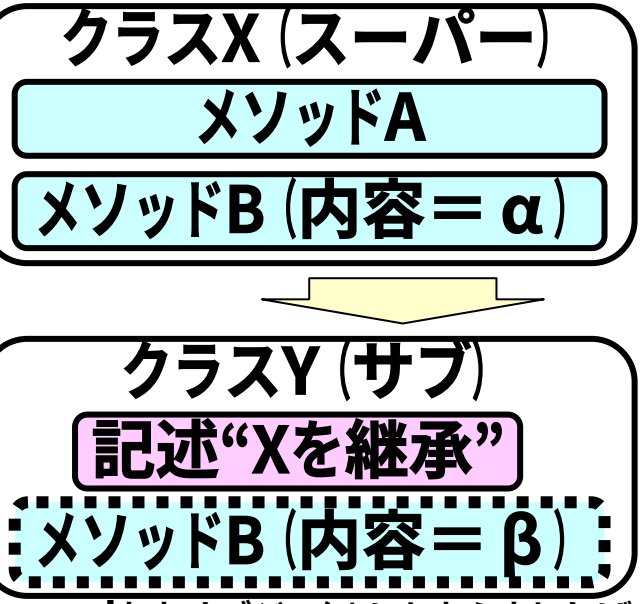


■このため、継承されることを前提とした場合には、“protected”と指定するなどの対応を行います。本スライドでは、このようなJavaの仕様については触れません。

(7．3．4) 書き換える

■元となるクラスのメソッドを書き換えたい場合、オーバーライド (override) という方法を取ります。

■スーパークラスのクラスXのメソッドを書き換えたい場合、同じ名前と引数のメソッドmethodX1をサブクラスのクラスYに記述します。すると、クラスYではクラスXのmethodX1は使われず、クラスYに記述したmethodX1が使われます。



クラスYは、

クラスXの
メソッドBを
書き換えた
ことになる

と同じ働きをする。

(7．3．5) 書き換えのJavaでのイメージ

■サブクラスのclassBでは、スーパークラスのclassAと同じメソッド“methodB”を記述します。

```
# クラスX(スーパークラス)
class classX{
  int c;
  void methodA(){
        :
  }
  void methodB(){
    # 私は、X、X、X、X
  }
}

#クラスY(サブクラス)
class classY extends classX{
  void methodB(){
    # 私は、Y、Y、Y、Y
  }
}
```

オーバーライドされるメソッド

オーバーライドするメソッド

クラスclassYは、

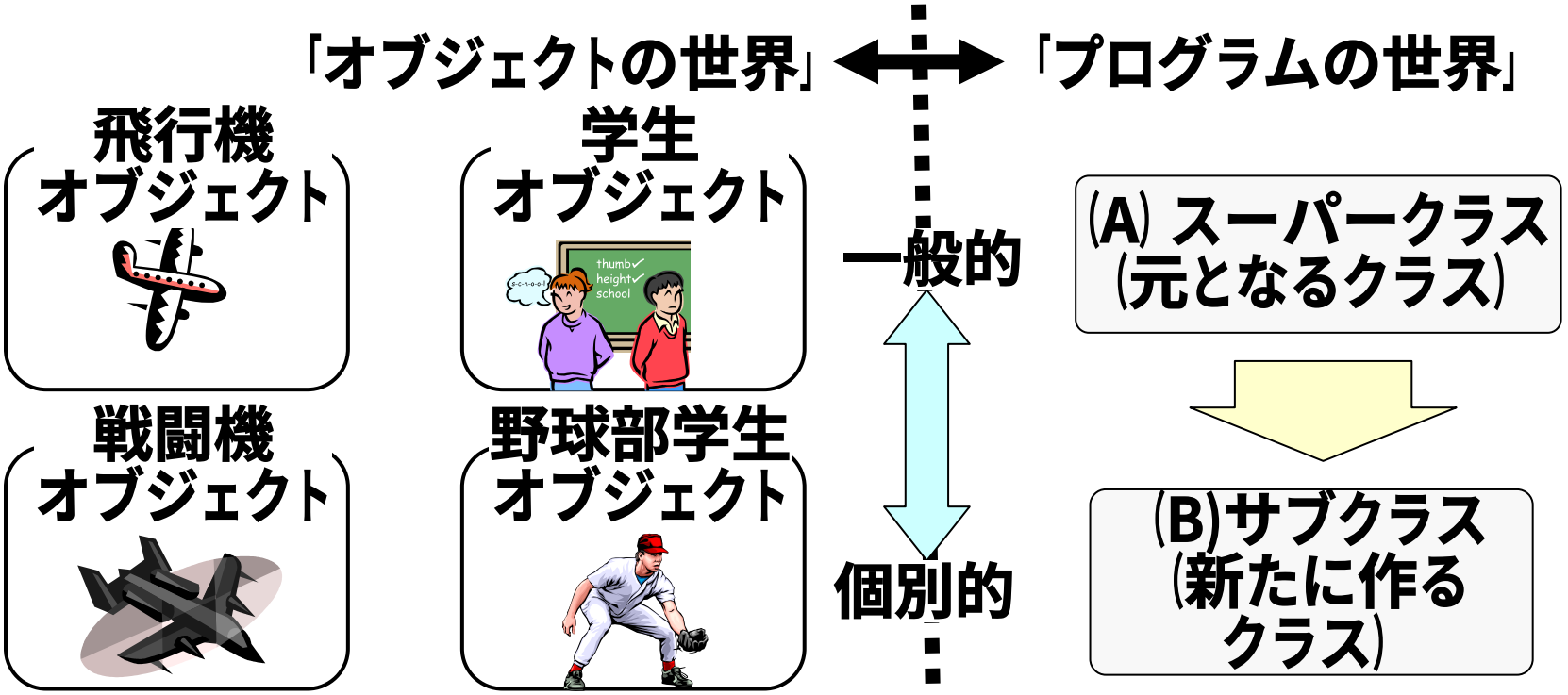
```
#クラスY
class classY{
  int c;
  void methodA(){
        :
  }
  void methodB(){
    #私は、Y、Y、Y、Y
  }
}
```

と同じ働きをする。

(7．4) 継承のモデリング

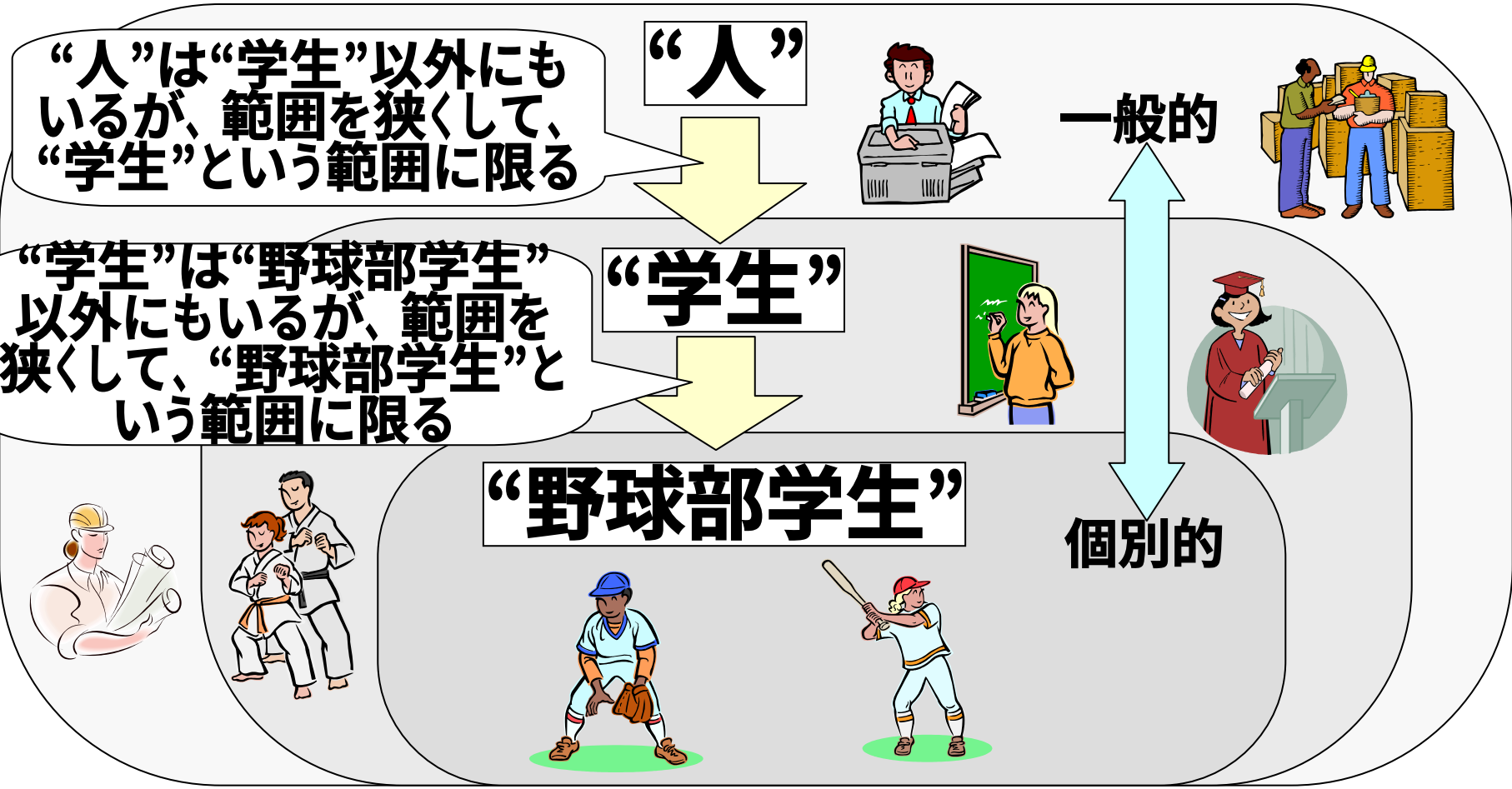
■継承を使ったプログラムの再利用の仕組みは、モデリングの観点からはどのように見えるのでしょうか？

■モデリングの観点、すなわち、“オブジェクトの世界”でも、スーパークラスとサブクラスにおける“一般的／個別的”という視点は、同じように適用できます。



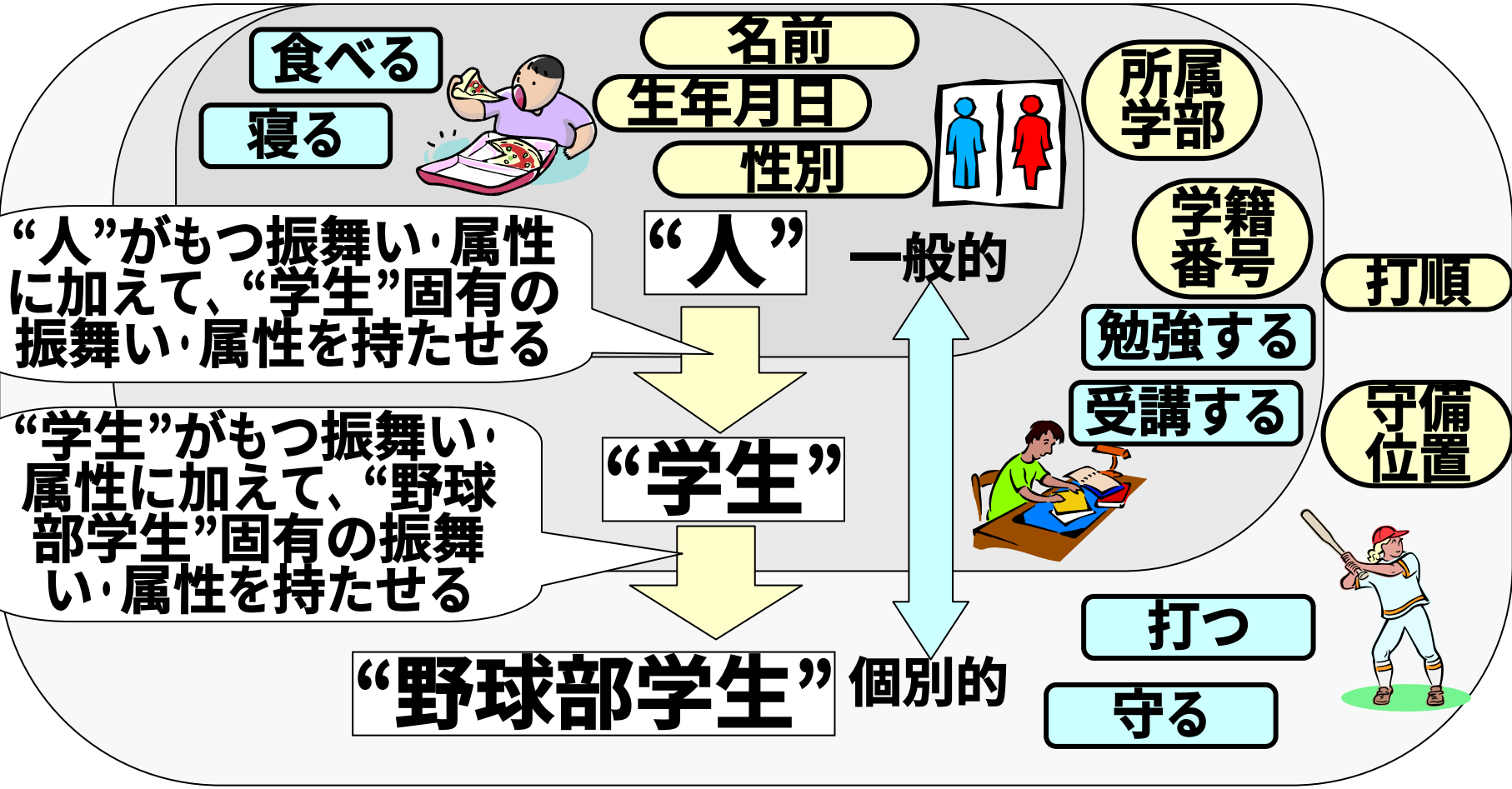
(7．4．1) 範囲から見た一般化･個別化

■ものごとが含まれる範囲の観点から、一般化･個別化を考えます。



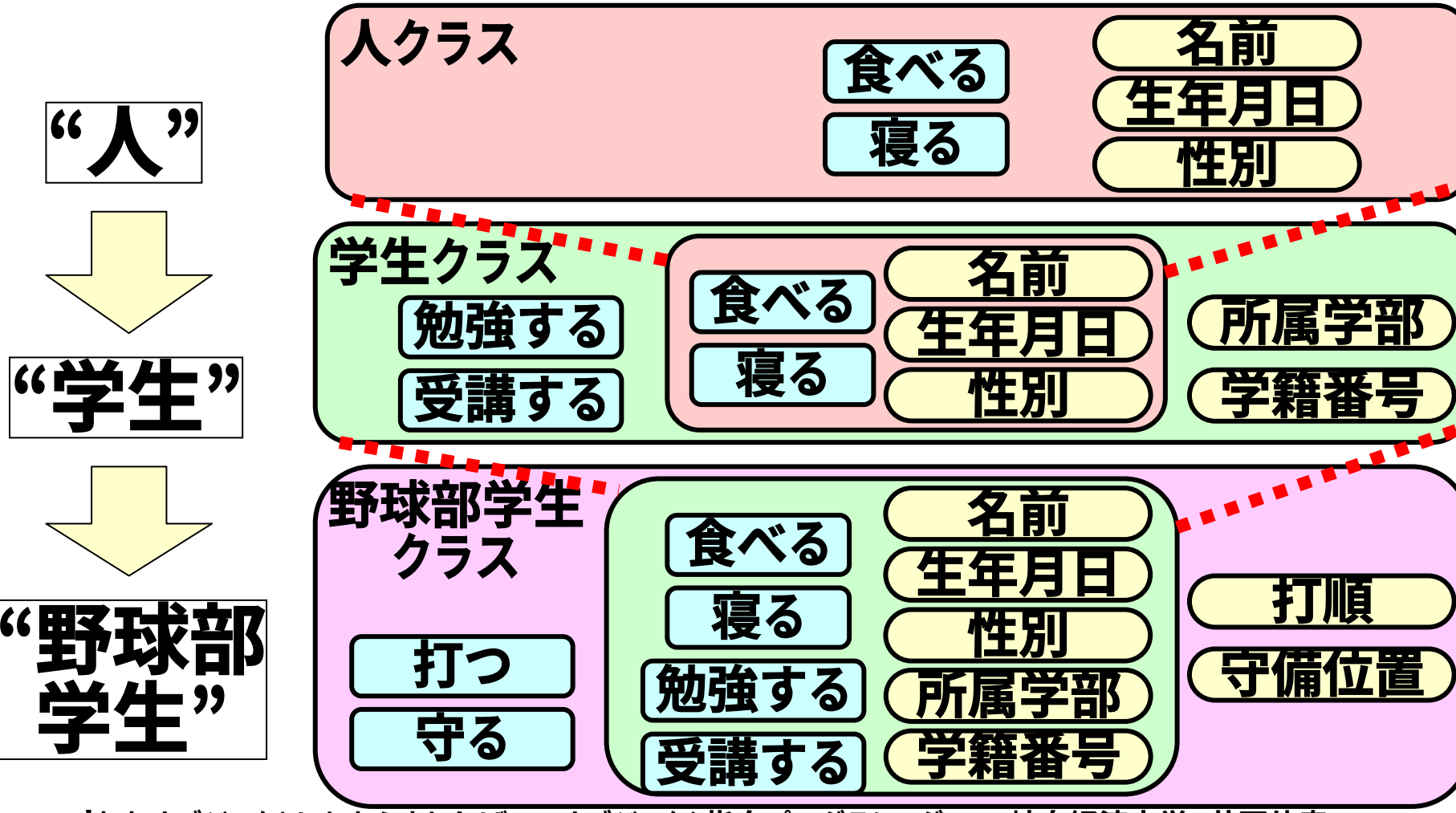
(7．4．2) 振舞い･属性から見た一般化･個別化

■ものごとが持つ振舞い･属性の観点から、一般化･個別化を考えます。



(7．4．3) 継承機能との対応

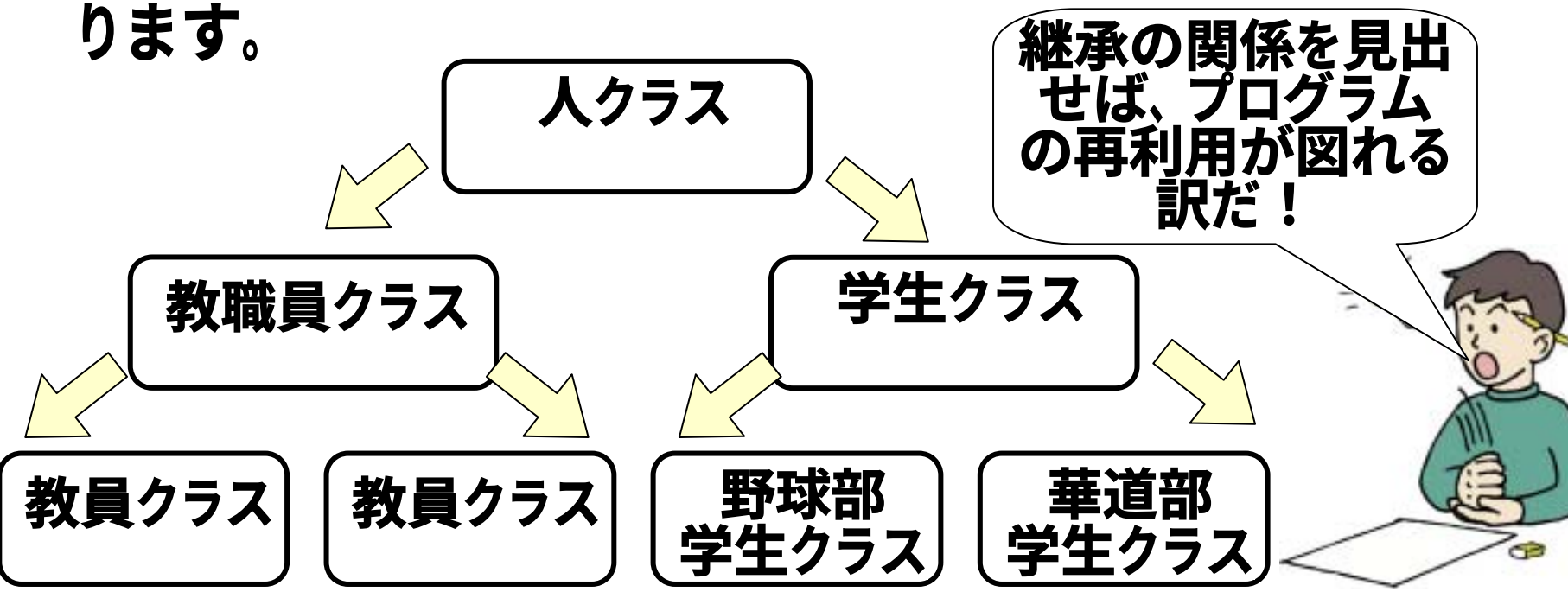
■モデルの観点からの“一般化･個別化”は、Javaプログラムの継承機能と対応が取れています。



(7．4．4) 設計で行うこと

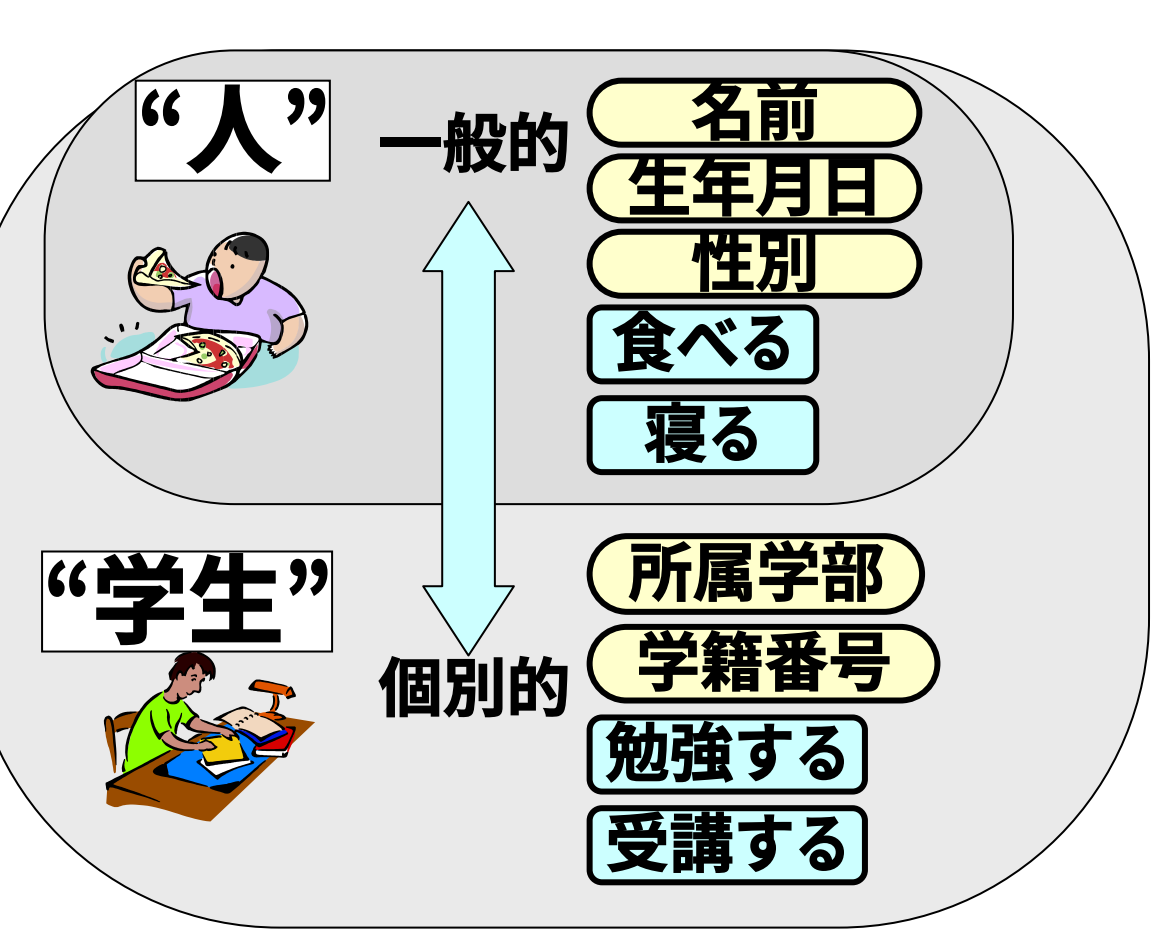
■ソフトウェアを作成するときは、似たようなプログラムを繰り返し書くことを避け、出来るだけ再利用したいものです。

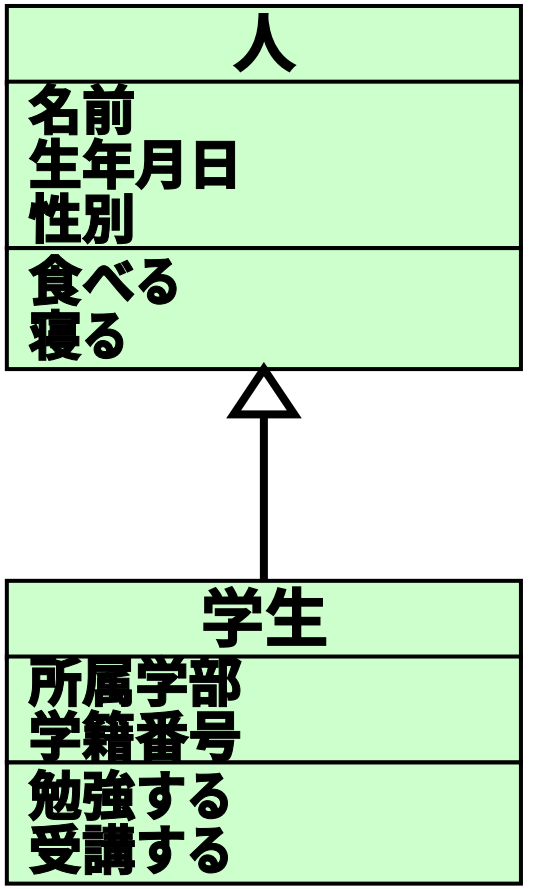
■継承の関係（一般化･個別化の関係）にあるクラスを見出すことは、とりもなおさず、再利用の促進につながります。



(7．5．1) クラス図での継承

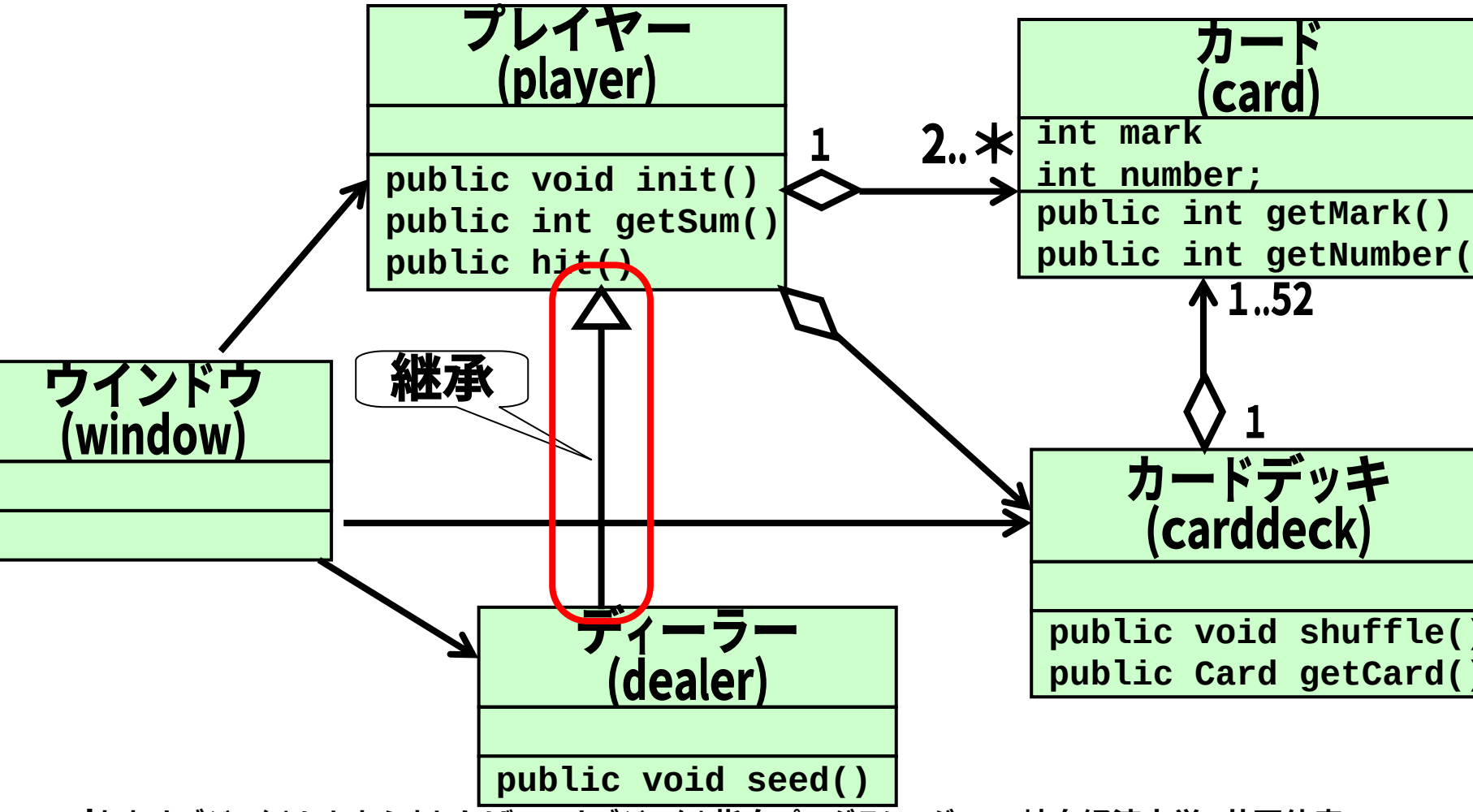
■クラス図では、継承の関係を三角の矢印を持つ線にて表します。





(7．5．2)ブラックジャックでの継承

■ブラックジャックでは、ディーラークラスが、プレイヤークラスを継承しています。



(7．5．3)ディーラークラス

■ディーラークラスのファイルには、“seek”メソッドしか記述されていませんが、プレイヤークラスを継承することで、“init”や“hit”などのメソッドや、“list”などの変数も持っていることになります。

```
# プレイヤークラス
public class Player {
  private CardDeck myCardDeck;
  DefaultListModel list;
   public void init(){...}
  public int getSum(){...}
  public void hit(){...}
  public ... getList(){...}
  protected void addOne(){...}
} /* end class Player */
```

```
# ディーラークラス
public class Dealer
    extends Player {
   public seek(){....}
} /* end class Dealer */
```

```
# ディーラークラスはこのような、
# 継承により、このようなクラスのように
# 振舞う。
public class Dealer {
  private CardDeck myCardDeck;
  DefaultListModel list;
  public void init(){...}
  public int getSum(){...}
  public void hit(){...}
  public ... getList(){...}
  protected void addOne(){...}
  public seek(){....}
} /* end class Dealer */
```

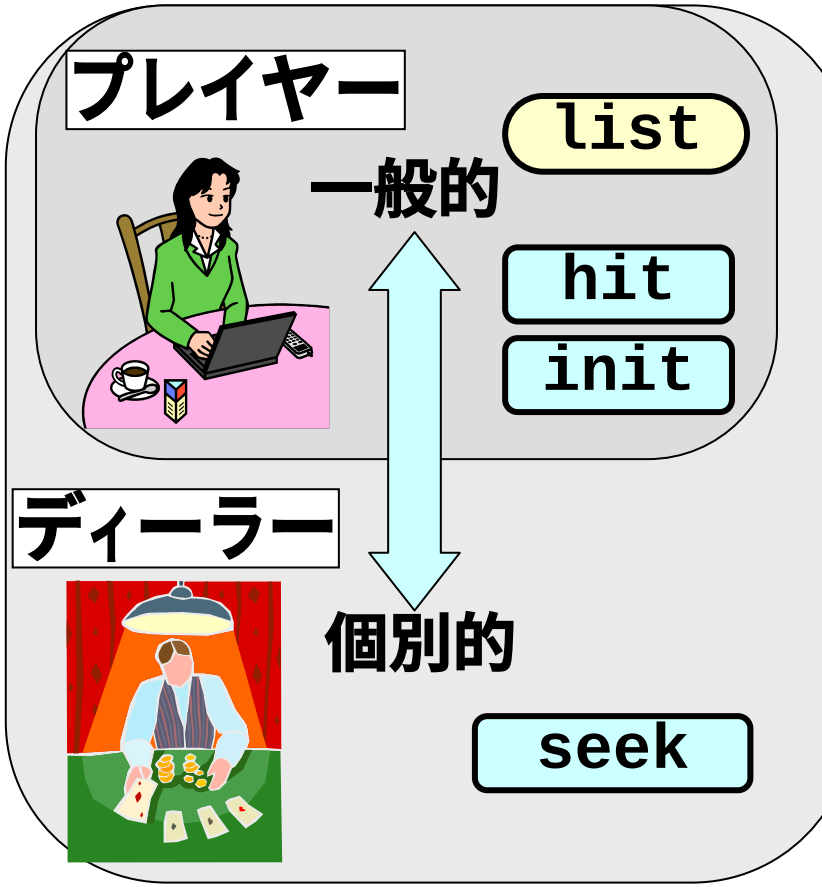
(7．5．4) モデルの観点から

■ブラックジャック･ゲームの中で、次のような点では、ディーラーはプレイヤーとして動作しています。

- カードを所持する
- カードを引く

■ディーラーには、プレイヤーとは別の、より個別の役割りもあります。

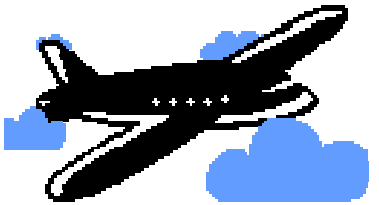
■上記のような関係が、プログラムの中では継承としてそのまま現れている訳です。

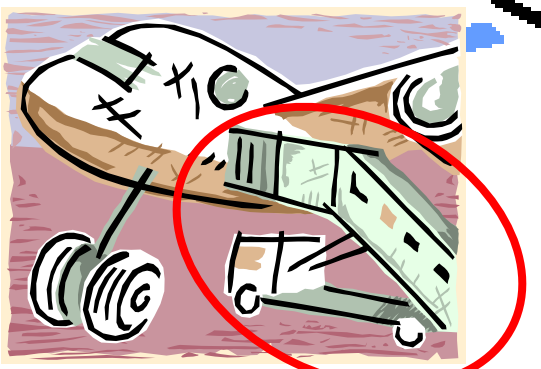


(7．6) 演習：課題7－1

■次に示すクラス、メソッド、属性の関係を整理して、継承関係を表すクラス図を作成してください。

- クラス
  - ◆乗物、飛行機、ダンプカー、旅客機、自動車
- 属性
  - ◆乗客、タイヤ、高度、エンジン、速度、翼、乗客、積載トン数、ハンドル、
- メソッド
  - ◆タラップを付ける
  - ◆離陸する
  - ◆バック (後進) する
  - ◆操縦(運転)する
  - ◆荷台を持ち上げる





タラップ