22 Javaオブジェクト指向プログラミ

22–1 クラス名，変数名，メソッド名 の付け方

Sun Microsystems (現Oracle)が推奨しているプログラミング作法
(http://java.sun.com/docs/codeconv/html/Code ConvTOC.doc.h
http://www.oracle.com/technetwork/java/codeconvtoc-136057.
が一般的な慣習・しきたりとして広まっている。

これ(と一部別の所の規約)によれば、

- 一般的な指針：

  ◇ 英単語など、意味のある単語を並べて名前を付ける。

  ◇ 「意味のある単語」として、漢字等の特殊な文字は使わない。

  ◇ 「意味のある単語」としては、不可解な短縮形は使わず、
  可能な限り省略なしの単語を用いる。

  ◇ ドル記号（$）は基本的には使わない。

  ◇ アンダースコア（_，下線記号）は特別な場合以外は使わない。

- クラス名，フィールド名，メソッド名，変数名の付け方（定数値を保持する領域は除く）：
  - ◇ 主として英小文字を用いる。
  - ◇ 2つ以上の「意味のある単語」から構成する場合は、2つ目以降の単語の頭文字を大文字にする。 Sun
  - ◇ クラス名の場合 は、先頭文字を大文字にし、名詞（句）を表す単語列にする。 Sun
  - ◇ フィールド名，変数名の場合 は、先頭文字を小文字にする。 Sun
  - ◇ メソッド名の場合 は、先頭文字を小文字にし、動詞（句）を表す単語列にする。 Sun
  - ◇ フィールドの値を調べるメソッドの名前 は「‘‘get’’+フィールド名」（e.g. getNum） 電通国際情報サービス
  - ◇ フィールドの値を設定するメソッドの名前 は「‘‘set’’+フィールド名」（e.g. setValue）

電通国際情報サービス

- 定数値を保持するフィールド名, 変数名の付け方：

  ◇ 「意味のある単語」を構成する全ての英文字を大文字にする。 Sun

  ◇ 2つ以上の「意味のある単語」から構成する場合は、
  単語間にアンダースコア（_, 下線記号）を入れる。 Sun

22–2 マニュアルの自動作成

クラス定義の直前, クラス定義中のフィールド宣言やコンストラクタ宣言, メソッド宣言の直前に

`/** ~ */` という形式の注釈 .（ドキュメンテーションコメント）

を入れておくと、...

⇨ `javadoc` コマンドを使ってクラスの概要を説明するマニュアルを `html` ファイルの形で構築することができる。

例22．1（javadocコマンドによるマニュアル自動作成）　例題13.3では int型データが最大100個入るスタックモジュールと、このスタックモジュールを利用して、①文字列を1個読み込み ② それを反転した後 ③出力するプログラムを、C言語で記述した。
そして、 例題19.2では これとほぼ同等のJavaプログラムを示した。

これに対して、 ここでは

- 例題19.6で作成したStackOfAnyObjectsクラスのインスタンスを利用して、同様の処理を行うJavaプログラムを考えた。 また、
- 後でjavadocコマンドによるマニュアル自動作成を行うことを念頭に入れ、例題19.6で示したStackOfAnyObjects.javaも書き直した。

得られたJavaプログラムを表示し、更に続けて、
①コンパイル、②実行し、
③マニュアルを入れるディレクトリを作成、
④javadocコマンドを適用してマニュアルをhtmlファイルの形で生成、
⑤生成されたマニュアルをfirefoxで表示、
している会話の様子を次に示す。

```
[motoki@x205a]$  ls
ExampleJavadoc.java  StackOfAnyObjects.java
[motoki@x205a]$  cat -n ExampleJavadoc.java
     1 import java.util.*;
     2
     3 /**
     4  * javadocコマンドの実行例を示すためのプログラム例
     5  * @author 元木達也
     6  */  標準タグ
     7 public class ExampleJavadoc {
     8     /** 処理内容：
     9      * (1)文字列を1個読み込み,
    10      * (2)それを(前後)反転した文字列をStackOfAnyObjects
                                                を利用して構成し,
    11      * (3)出力
    12      */
    13     public static void main(String args[]) {
```

```
14          String inputString, reversedString;
15
16          // 文字列1個を inputString に読み込む
17          System.out.print("Input a string:  ");
18          Scanner inputScanner = new Scanner(System.in);
19          inputString = inputScanner.next();
20
21          // 入力文字列を反転して reversedString に格納
22          int len = inputString.length();
23          StackOfAnyObjects stack =
                                    new StackOfAnyObjects(len);
24          for (int i=0; i<len; i++)
25              stack.pushdown(
                    new Character(inputString.charAt(i)));
26          char[] reversedSequenceOfChar = new char[len];
27          for (int i=0; i<len; i++)
28              reversedSequenceOfChar[i]
```

```
29                  = ((Character)stack.popup())
                                                .charValue();
30          reversedString =
                        new String(reversedSequenceOfChar);
31
32          // 反転文字列を出力
33          System.out.println("Reversed string: "
                                        + reversedString);
34      }
35 }
```

[motoki@x205a]$ cat -n StackOfAnyObjects.java

```
 1 import java.util.*;
 2
 3 /**
 4  * Objectインスタンスを格納するpushdownスタックのクラ...
 5  * @author 元木達也
 6  * @version 0.0
```

```
 7  */
 8 public class StackOfAnyObjects {
 9     /** 初期容量のデフォルト値 */
10     private static final int
                                DEFAULT_INITIAL_CAPACITY = 100;
11
12     /** 容量不足の際に増やす容量のデフォルト値 */
13     private static final int
                              DEFAULT_CAPACITY_INCREMENT = 100;
14
15     /** Objectインスタンス(への参照)を格納するため... */
16     private Object[] stack;
17
18     /** スタックの最も上部の要素が格納されている位置... */
19     private int indexOfTopEle;
20
21     /**
```

```
22      * 空のスタックを構成する
23      */
24     public StackOfAnyObjects() {
25         this(DEFAULT_INITIAL_CAPACITY);
26     }
27
28     /**
29      * 空のスタックを構成する
30      * @param initialCapacity スタックの初期容量
31      */
32     public StackOfAnyObjects(int initialCapacity) {
33         stack = new Object[initialCapacity];
34         indexOfTopEle = -1;
35     }
36
37     /**
38      * スタックオブジェクトの標準的な文字列表現を求める
```

```
39      * @return 標準的な文字列表現
40      */
41     @Override
42     public String toString() {
43         return "pushdownStack (type=Object, capacity="
                                                + stack.length
44                 + ", currentNumOfEle="
                                    + (indexOfTopEle+1) + ")";
45     }
46
47     /**
48      * スタックに格納されている要素の情報を得る
49      * @return スタックの内容を表す文字列
50      */
51     public String getDetailedConfig() {
52         String result = "stack contains "
                    + (indexOfTopEle+1) + " elements: {";
```

```
53          for (int i=0; i<indexOfTopEle; ++i)
54              result += "\n    " + stack[i] + ",";
55          if (indexOfTopEle >= 0)
56              result += "\n    " + stack[indexOfTopEle];
57          result += " }";
58          return result;
59      }
60
61      /**
62       * スタックが空かどうかを調べる
63       * @return スタックが空かどうか
64       */
65      public boolean isEmpty() {
66          return indexOfTopEle == -1;
67      }
68
69      /**
```

```
70      * 新しいObject要素をスタックにpush-downする
71      * @param element スタックにpush-downする新要素
72      */
73     public void pushdown(Object element) {
74         if (indexOfTopEle+1 == stack.length) {
75             stack = Arrays.copyOf(stack, stack.length
76                         + DEFAULT_CAPACITY_INCREMENT);
77             System.out.printf("###Stack capacity is inc
78                                 "#<New> %s%n", this);
79         }
80         stack[++indexOfTopEle] = element;
81     }
82
83     /**
84      * スタックから最も上部の要素を取り出す
85      * @return スタックの最も上部の要素
86      */
```

```
87      public Object popup() {
88          if (indexOfTopEle < 0)
89              throw new EmptyStackException();
90          Object element = stack[indexOfTopEle];
91          stack[indexOfTopEle--] = null;
                                    //取り出した要素への参照を解除
92          return element;
93      }
94
95      /**
96       * スタックに格納された要素の個数を調べる
97       * @return スタックに格納された要素の個数
98       */
99      public int getNumOfEle() {
100         return indexOfTopEle+1;
101     }
102
```

```
103     /**
104      * スタックのtop要素をのぞき見
105      * @return スタックのtop要素(への参照)
106      */
107     public Object peepTop() {
108         if (isEmpty())
109             return null;
110         else
111             return stack[indexOfTopEle];
112     }
113
114     /**
115      * スタックの指定要素をのぞき見
116      * @param index のぞき見したいスタック要素の番号
117      * @return 番号indexのスタック要素(への参照)
118      */
119     public Object peepEleOfIndex(int index) {
```

```
120         return stack[index];
121     }
122 }
[motoki@x205a]$  javac ExampleJavadoc.java
[motoki@x205a]$  ls
ExampleJavadoc.class  StackOfAnyObjects.class
ExampleJavadoc.java   StackOfAnyObjects.java
[motoki@x205a]$  java ExampleJavadoc
Input a string:  abc123
Reversed string: 321cba
[motoki@x205a]$  mkdir html_by_javadoc
[motoki@x205a]$  javadoc -private -d html_by_javadoc *.java
ソースファイル ExampleJavadoc.java を読み込んでいます...
ソースファイル StackOfAnyObjects.java を読み込んでいます...
Javadoc 情報を構築しています...
標準 Doclet バージョン 1.6.0_24
全パッケージとクラスの階層ツリーを作成しています...
```

```
html_by_javadoc/ExampleJavadoc.html の生成
html_by_javadoc/StackOfAnyObjects.html の生成
html_by_javadoc/package-frame.html の生成
html_by_javadoc/package-summary.html の生成
html_by_javadoc/package-tree.html の生成
html_by_javadoc/constant-values.html の生成
全パッケージとクラスのインデックスを作成しています...
html_by_javadoc/overview-tree.html の生成
html_by_javadoc/index-all.html の生成
html_by_javadoc/deprecated-list.html の生成
全クラスのインデックスを作成しています...
html_by_javadoc/allclasses-frame.html の生成
html_by_javadoc/allclasses-noframe.html の生成
html_by_javadoc/index.html の生成
html_by_javadoc/help-doc.html の生成
html_by_javadoc/stylesheet.css の生成
[motoki@x205a]$  ls
```

```
ExampleJavadoc.class   StackOfAnyObjects.class   html_by_javadoc
ExampleJavadoc.java    StackOfAnyObjects.java
[motoki@x205a]$  ls html_by_javadoc
ExampleJavadoc.html        help-doc.html        package-summary.h
StackOfAnyObjects.html     index-all.html       package-tree.html
allclasses-frame.html      index.html           resources/
allclasses-noframe.html    overview-tree.html   stylesheet.css
constant-values.html       package-frame.html
deprecated-list.html       package-list
[motoki@x205a]$  firefox html_by_javadoc/index.html&
[2] 5065
[motoki@x205a]$
```

⇨ 次の様なウィンドウが画面上に表示される。

StackOfAnyObjects - Fx ウェブブラウザ

ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)

StackOfAnyObjects

file:///home/motoki/C-Java2012/Programs-Java/objectoriented/Javadoc/html_by_javadoc/index.html

よく見るページ オンラインマニュアル バグトラッキング MLアーカイブ Yahoo! JAPAN 読売新聞 NHK

すべてのクラス
ExampleJavadoc
StackOfAnyObjects

パッケージ クラス 階層ツリー 非推奨 API 索引 ヘルプ

前のクラス 次のクラス フレームあり フレームなし
概要: 入れ子 | フィールド | コンストラクタ | メソッド 詳細: フィールド | コンストラクタ | メソッド

クラス StackOfAnyObjects

```
java.lang.Object
  └StackOfAnyObjects
```

```
public class StackOfAnyObjects
extends java.lang.Object
```

Objectインスタンスを格納するpushdownスタックのクラス

フィールドの概要

private static int	DEFAULT_CAPACITY_INCREMENT 容量不足の際に増やす容量のデフォルト値
private static int	DEFAULT_INITIAL_CAPACITY 初期容量のデフォルト値
private int	indexOfTopEle スタックの最も上部の要素が格納されている位置(配列要素の添字番号)
private java.lang.Object[]	stack Objectインスタンス(への参照)を格納するための配列領域

コンストラクタの概要

StackOfAnyObjects() 空のスタックを構成する
StackOfAnyObjects(int initialCapacity) 空のスタックを構成する

22–3 入れ子クラス

入れ子クラス，ネストしたクラス： Javaでは、クラス定義の中に別のクラスの定義を書ける。

```
アクセス修飾子 class 外側のクラス名 ..... {
        .....
    アクセス修飾子 (場合によっては)static class 入れ子クラス名 {
            .....
    }
        .....
}
```

ここで、

- 入れ子クラスから外側のクラスの他のメンバーへのアクセスは（状況次第で）可能。修正

  但し、同じ名前のフィールドが外側のクラスの中にも...

- 入れ子クラス名の前に付ける アクセス修飾子 としては、private,（修飾子なし）, protected, public の4つが可能

```
アクセス修飾子 class 外側のクラス名 ...... {
        ......
    アクセス修飾子 static class 入れ子クラス名 {
            ......
    }
        ......
}
```

- static宣言された入れ子クラスの場合 、
  入れ子クラスのクラス定義も外側のクラス全体を管理するクラスオブジェクトに属し、言わば「インスタンス工場の中に小さなインスタンス工場がある」という状態になる。

⇨ （アクセス制限がなければ）入れ子クラスへのアクセスは...

外側のクラス名 . 入れ子クラス名

入れ子クラスのインスタンスを生成したい場合は例えば...

外側のクラス名 . 入れ子クラス名

変数名 = new 外側のクラス名 . 入れ子クラス名（引数列）;

```
アクセス修飾子 class 外側のクラス名 ..... {
        ......
    アクセス修飾子 class 入れ子クラス名 {
            ......
    }
```

staticな入れ子クラスから外側のクラスのインスタンス変数へのアクセスは 、適切なインスタンス参照を通してのみ可能

- static宣言されていない入れ子クラス は、特に内部クラスと呼ばれる。
  この場合、入れ子クラスの定義も外側のクラスの各々のインスタンスに配備される。
  また、内部クラスのインスタンスから外側のクラスのインスタンス変数を暗黙に参照可。

⇨ 内部クラスのインスタンスの生成は、通常、(外側のクラスの)インスタンスメソッドやコンストラクタの中で行う。

内部クラス名 変数名 = new 内部クラス名(引数列);

外側のクラスのインスタンス外から内部クラスのインスタンス生成も可能で、その場合は、
まず外側のクラスのインスタンスを生成した上で、次の様に書く。

外側のクラス名 . 内部クラス名

変数名 = 外側のクラスのインスタンス参照.new 内部クラス名(引数列);

例22．2（staticな入れ子クラス）　必要そうな場所ではこれまでにも使用してきた。

- 例題19.7の TowerOfHanoiConfig.java で定義されたクラス Disk,
- 例題19.8の NumberWith1000DecimalPlaces.java で定義されたクラス Digit,
- 例題19.9の BinaryTreeOfStringInt.java で定義されたクラス Node

例22．3（内部クラス,K.Arnold他「プログラミング言語Java第4版」p.

```
public class BankAccount {
    private long number;     //口座番号
    private long balance;    //現在の残高
    private Action lastAct; //最後に行われた処理

    public class Action {
        private String act;
        private long amount;
        Action(String act, long amount) {
            this.act = act;
            this.amount = amount;
```

```
        }
        public String toString() { // identify our enclosing a
            return number + ": " + act + " " + amount;
        }           // ↑ Actionが内部クラスなので、こんな風に、
    }               //    外側のクラスのインスタンス変数を参照可

    public void deposit(long amount) {
        balance += amount;
        lastAct = new Action("deposit", amount);
    }

    public void withdraw(long amount) {
        balance -= amount;
        lastAct = new Action("withdraw", amount);
    }
    // .........
}
```

実際には、次の例題で示される様に、入れ子クラスはメソッド定義の中に置くこともできるし、また、一時的にしか使わないクラスについては名前を付けずにクラス定義して即インスタンス生成ということも可能である。

例題22．4（無名内部クラス；java.util.Arrays.sortを用いた並び替え

① 入力ストリームに現れる 識別子 $\underbrace{\quad\quad}_{\text{1個以上の空白}}$ 整数 という形のデータを読み込んでは、

② それを配列に登録する、

という作業を繰り返し、得られた配列内のデータをjava.util.Arrays クラスに備わったクラスメソッド sort を利用して識別子に関して辞書順になる様に並べ替え、その結果を出力するJava プログラムを作成せよ。

（考え方）　例題19.9では2分木を用いたが、ここでは並び替えに java.util.Arrays.sort というユーティリティメソッドを使え、ということである。

その前に、まず

不定個のデータ群を配列に格納する際、配列に空きがなかった場合

java.util.Arraysクラス内に用意された

```
copyOf(Type[] original, int newLength)
```

というクラスメソッドを利用できる。

ジェネリック型(パラメータ付きの型)

メソッド sort(Type[] a, Comparator<? super Type> c) の利用

データ群の格納された配列と

配列要素間の順序関係を判定するオブジェクト(コンパレータという)

を引数として指定するだけ。

(第２引数である順序関係を判定するオブジェクトは
mainメソッド内のこの場所でしか使用しない
⇨ このインスタンスを生成する時に
対応するクラスの定義も行うというので十分)

（プログラミング）

```
[motoki@x205a]$  cat -n SortIdIntPairsByArraysSortMain.java
     1 import java.util.Scanner;
     2 import java.util.Arrays;
     3 import java.util.Comparator;
     4
     5 /**
     6  * (1)識別子と整数データの組を読み込んでは
     7  * (2)配列に登録する
     8  * という作業を繰り返し、得られた配列内のデータを
                                                    Arraysクラスに
     9  * 備わったクラスメソッド sort を利用して識別子に関して
                                                    辞書順に
    10  * なる様に並べ替え、その結果を出力するJavaプログラム
    11  * @author 元木達也
    12  */
    13 class SortIdIntPairsByArraysSortMain {
```

インタフェース

```
14      //識別子と整数データの組を表すオブジェクトのクラス
15      private static class IdIntPair {
16          private String stringData;
17          private int    intData;
18
19          //コンストラクタ
20          public IdIntPair(String stringData,
                                          int intData) {
21              this.stringData = stringData;
22              this.intData    = intData;
23          }
24
25          //IdIntPairインスタンスの標準的な文字列表現を...
26          @Override
27          public String toString() {
28              return "String-int pair (" + stringData + '
29          }
30      }
31      //-----------------------------------------------
```

```
32
33      private static final int INCREMENT_SIZE = 100;
34
35      public static void main(String[] args) {
36          Scanner inputScanner = new Scanner(System.in);
37          IdIntPair[] data =
                                new IdIntPair[INCREMENT_SIZE];
38          int numOfData     = 0;
39
40          //データ入力と配列への登録
41          while (inputScanner.hasNext()) {
42              String newString = inputScanner.next();
43              if (inputScanner.hasNextInt()) {
44                  int newInt = inputScanner.nextInt();
45                  if (numOfData >= data.length)
46                      data = Arrays.copyOf(data,
                            data.length + INCREMENT_SIZE);
47                  data[numOfData++] =
                        new IdIntPair(newString, newInt);
```

```
48                  } else {
49                      System.out.printf("ERROR: invalid data
50                                        "  ==>inut is aborted
51                  }
52              }
53              if (numOfData < data.length)
54                  data = Arrays.copyOf(data, numOfData);
55
56              //識別子に関して辞書順になる様に並べ替え
57              Arrays.sort(data,                    無名クラス定義
58                  new Comparator<IdIntPair>() {
59                      public int compare(IdIntPair d1,
                                           IdIntPair d2) {
60                          return d1.stringData.compareTo(
                                           d2.stringData);
61                      }
62                  });
63
64              //配列に蓄えられた内容を表示
```

```
65          System.out.printf("       intData    stringData
66                            "----  ----------  ----------
67          for (int i=0; i<data.length; ++i)
68              System.out.printf("%4d %11d  %s%n",
69                                  i+1, data[i].intData,
                                       data[i].stringData);
70      }
71 }
[motoki@x205a]$  cat dynamic-llist.data
asdfghjk 55555
qwertyui 222222
abc      123
ppp_qqq  2345
zxcv     987
kkk      456
efghi    77
[motoki@x205a]$  javac SortIdIntPairsByArraysSortMain.java
[motoki@x205a]$  java SortIdIntPairsByArraysSortMain < dynamic
```

```
       intData    stringData
----  ----------  ----------
   1         123  abc
   2       55555  asdfghjk
   3          77  efghi
   4         456  kkk
   5        2345  ppp_qqq
   6      222222  qwertyui
   7         987  zxcv
[motoki@x205a]$
```

入れ子クラスを使うことの利点：

...{S.Zakhour「Javaチュートリアル第4版」p.115}

- クラス間の包含関係、主従関係を反映した形でクラスを合理的にグループ化できる。

- 情報隠蔽、カプセル化の促進。

- 関連したクラスがソースコードレベルで近くに置かれることになるので、コードの理解とメンテナンスも容易になる。

22–4 初期化ブロック

……

- クラス定義の中で、
  フィールド宣言やコンストラクタ定義, メソッド定義の一部になっていないブロック
  (i.e."{" と "}" で囲まれた部分)
- インスタンスやクラス変数の初期化に利用

この内、

- static宣言のない初期化ブロック は
  インスタンスを初期化するためのもので、
  ブロック中に書かれた初期化がコンストラクタ呼出しの直前に行われる。

- 例えば無名内部クラスのインスタンスを初期化したい場合に有用

- static宣言された初期化ブロック 、すなわち

```
static {
    ......
}
```

は、クラス変数を初期化するためのもので、
クラスが実際に使用される前に実行される。

22–5 既定義クラスの拡張

既定義クラスの拡張と継承：

既に定義されたクラスを拡張(extend) して新しいクラスを定義することができる。

この場合、元になったクラスをスーパークラス新しく定義したクラスを(元のクラスの)サブクラスという。

- 具体的には、Java では次の様に書く。

```
修飾子 class クラス名 extends スーパークラス
{
    新しい変数の宣言、
    新しいメソッドの定義、など
}
```

- スーパークラスで定義されたインスタンスフィールドやインスタンスメソッドはサブクラスに暗黙に継承(inherit)される。

  すなわち、スーパークラスで定義された非staticフィールドについては、サブクラスのインスタンスにおいても暗黙に領域確保され、また、スーパークラスで定義された非static メソッドは、サブクラスの中で再定義／上書き(オーバーライド,override,という)されてなければ、サブクラスのインスタンスメソッドとして暗黙に使用することができる。

  ⇨ 継承をうまく利用すれば、同じ定義をあちこちのクラス定義の中で繰り返す手間はかなり省ける様になる。

---

is-a関係，has-a関係：

オブジェクト指向では、クラス間の次の2つの関係に注目する。

- is-a関係 ··· クラス A のインスタンスがクラス B のインスタンスの一種になる時、クラス A,B の間にis-a関係があるという。

  ⇨ クラス B の定義を拡張してクラス A の定義

- has-a関係 ··· クラス A のインスタンスがクラス B のインスタンスを一部分として持つ時、クラス A,B の間にhas-a関係があるという。

  ⇨ クラス A のインスタンスフィールドとして
  クラス B のインスタンス

サブクラスにおけるコンストラクタの記述：

- スーパークラスのコンストラクタを呼び出してスーパークラスに関連するフィールド等を初期設定したい場合は、

  super(引数列);

- サブクラス内の別のコンストラクタを呼び出してフィールド等を初期設定したい場合は、

  this(引数列);

- コンストラクタの最初に super(引数列); も this(引数列); も現れない場合は、デフォルトの作業、すなわち、Javaコンパイラはコンストラクタの最初の場所に

  super();

  という行を挿入してコンパイル作業を行う。

@Overrideアノテーション： .....J2SE5.0(JDK1.5)以降

オーバーライド(上書き)するメソッド定義の直前に
‘‘@Override’’ という文字列を入れることにより、
コンパイラにオーバーライドの意思を知らせる
ことができる。

⇨ メソッド名のつづりミス等をコンパイラがチェック

補足(アノテーション)： 一般に、@ で始まる(処理系/コンピュータ向けの)注釈をアノテーションと呼んでいる。@Override以外にも次の様なものがある。

- @Deprecated ··· 非推奨であることを明示する。(コンパイラ向け)
- @SuppressWarning ··· 警告の抑制を指示する。(引数を付ける)
- .....

補足(アノテーションの実体)：
各々のアノテーションは標準ライブラリ java.lang の中で定義されている。例えば、@Override は java.lang.Override という名前で宣言されている。

例22．5（色付き長方形のクラス,Rectangleクラスの拡張）

例題19.4... 長方形を表すオブジェクトのクラス Rectangle を示した。

ここでは、このクラスを拡張して色付き長方形を表すオブジェクトのクラス ColoredRectangle を定義

[motoki@x205a]$ cat -n Rectangle.java ...再掲,コメント部加筆

```
 1 /**
 2  * 長方形を表すオブジェクトのクラス
 3  */
 4 public class Rectangle {
 5     protected final int id;    //長方形インスタンスに...
 6     protected double width;    //長方形の幅
 7     protected double height;   //長方形の高さ
 8
 9     private static int numberOfRectangles = 0;
                                  //生成した長方形インスタンスの個数
10
```

```
11      /** 幅1.0,高さ1.0の長方形オブジェクトを構成する */
12      public Rectangle() {
13          this(1.0, 1.0);
14      }
15
16      /** 指定された幅,高さの長方形オブジェクトを構成する */
17      public Rectangle(double width, double height) {
18          this.id     = ++numberOfRectangles;
19          this.width  = width;
20          this.height = height;
21      }
22
23      /** 長方形インスタンスの標準的な文字列表現を返す */
24      @Override
25      public String toString() {
26          return "rectangle(id=" + id + ", width=" + widt
27                 + ", height=" + height + ")";
```

```
28      }
29
30      /** [ゲッター]長方形インスタンスのid番号を調べて返す */
31      public int getId() {
32          return id;
33      }
34
35      /** [ゲッター]長方形インスタンスの幅を調べて返す */
36      public double getWidth() {
37          return width;
38      }
39
40      /** [ゲッター]長方形インスタンスの高さを調べて返す */
41      public double getHeight() {
42          return height;
43      }
44
```

```
45      /** [ゲッター]生成した長方形インスタンスの個数を... */
46      public static int getNumberOfRectangles() {
47          return numberOfRectangles;
48      }
49
50      /** [セッター]長方形インスタンスの幅を設定 */
51      public void setWidth(double width) {
52          this.width = width;
53      }
54
55      /** [セッター]長方形インスタンスの高さを設定 */
56      public void setHeight(double height) {
57          this.height = height;
58      }
59
60      /** 長方形インスタンスの面積を計算して返す */
61      public double getArea() {
```

```
 62         return width*height;
 63     }
 64 }
[motoki@x205a]$  cat -n ColoredRectangle.java
  1 /**
  2  * 色付き長方形を表すオブジェクトのクラス
  3  * (既定義のRectangleクラスを拡張して定義)
  4  */
  5 public class ColoredRectangle extends Rectangle {
  6     protected String color;  //長方形の色
  7
  8     /** 黒色の長方形オブジェクトを構成する */
  9     public ColoredRectangle() {
 10         super();
 11         color = "black";
 12     }
 13
```

```
14      /** 指定された幅,高さ,色の長方形オブジェクトを構成... */
15      public ColoredRectangle(double width,
                                double height, String color) {
16          super(width, height);
17          this.color = color;
18      }
19
20      /** 色付き長方形インスタンスの標準的な文字列表現を... */
21      @Override
22      public String toString() {
23          return "rectangle(id=" + id + ", width="  + wi
24                 + ", height=" + height + ", color=" + c
25      }
26
27      /** [ゲッター(追加)]色付き長方形インスタンスの色を... */
28      public String getColor() {
29          return color;
```

```
30      }
31
32      /** [セッター(追加)]長方形インスタンスの色を設定 */
33      public void setColor(String color) {
34          this.color = color;
35      }
36
37      //--------------------単体での動作テスト用------------
38      public static void main(String[] args) {
39          ColoredRectangle rect = new ColoredRectangle();
40          System.out.println("(1)" + rect);
41          System.out.printf(
                "    |rect.getId()=%d, rect.getWidth()=%f,%n
42              "    |rect.getHeight()=%f, rect.getColor()=%
43              "    |rect.getArea()=%f%n",
44              rect.getId(), rect.getWidth(),
45              rect.getHeight(), rect.getColor(),
```

```
46                  rect.getArea());
47              rect.setWidth(2.0);
48              rect.setHeight(3.0);
49              rect.setColor("blue");
50              System.out.println("(2)" + rect);
51              System.out.printf(
                    "    |rect.getId()=%d, rect.getWidth()=%f,%r
52                  "    |rect.getHeight()=%f, rect.getColor()=%
53                  "    |rect.getArea()=%f%n",
54                  rect.getId(), rect.getWidth(),
55                  rect.getHeight(), rect.getColor(),
56                  rect.getArea());
57              System.out.println("(3)"
                    + new ColoredRectangle(4.0, 5.0, "red"));
58          }
59 }
[motoki@x205a]$  javac ColoredRectangle.java
```

```
[motoki@x205a]$  java ColoredRectangle
(1)rectangle(id=1, width=1.0, height=1.0, color=black)
   |rect.getId()=1, rect.getWidth()=1.000000,
   |rect.getHeight()=1.000000, rect.getColor()=black
   |rect.getArea()=1.000000
(2)rectangle(id=1, width=2.0, height=3.0, color=blue)
   |rect.getId()=1, rect.getWidth()=2.000000,
   |rect.getHeight()=3.000000, rect.getColor()=blue
   |rect.getArea()=6.000000
(3)rectangle(id=2, width=4.0, height=5.0, color=red)
[motoki@x205a]$
```

22–6 Objectクラス

- クラス定義時に ‘‘extends スーパークラス’’ の指定をしないと...

  ⟶ コンパイラによって
  自動的に‘‘extends Object’’ という指定が挿入され、...

  ⇨ Objectはクラス階層の頂点に位置

- java.lang.Objectクラスの中で定義されているメソッドを次に示す。

メソッド名 ··· 説明
protected Object clone() throws CloneNotSupportedException ··· 自オブジェクトのコピーを生成して返す。
public boolean equals(Object obj) ··· 自オブジェクトが引数で与えられたオブジェクトobjと等しいかどうかを判定し、その結果を返す。

メソッド名 … 説明
protected void finalize() throws Throwable … オブジェクトに対する参照がもう無いと判断された場合に、ガベージコレクタによって呼び出される。
public final Class getClass() … オブジェクトの基になったクラスのClassオブジェクト(i.e.クラスについての情報を保持しているオブジェクト)への参照を返す。
public int hashCode() … オブジェクトのハッシュコード値を返す。
public String toString() … オブジェクトの文字列表現を返す。
.....(スレッドをサポートするメソッドについては省略) .....

⇨ これらのメソッドは、全てのクラスに(暗黙に)継承される
