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
```

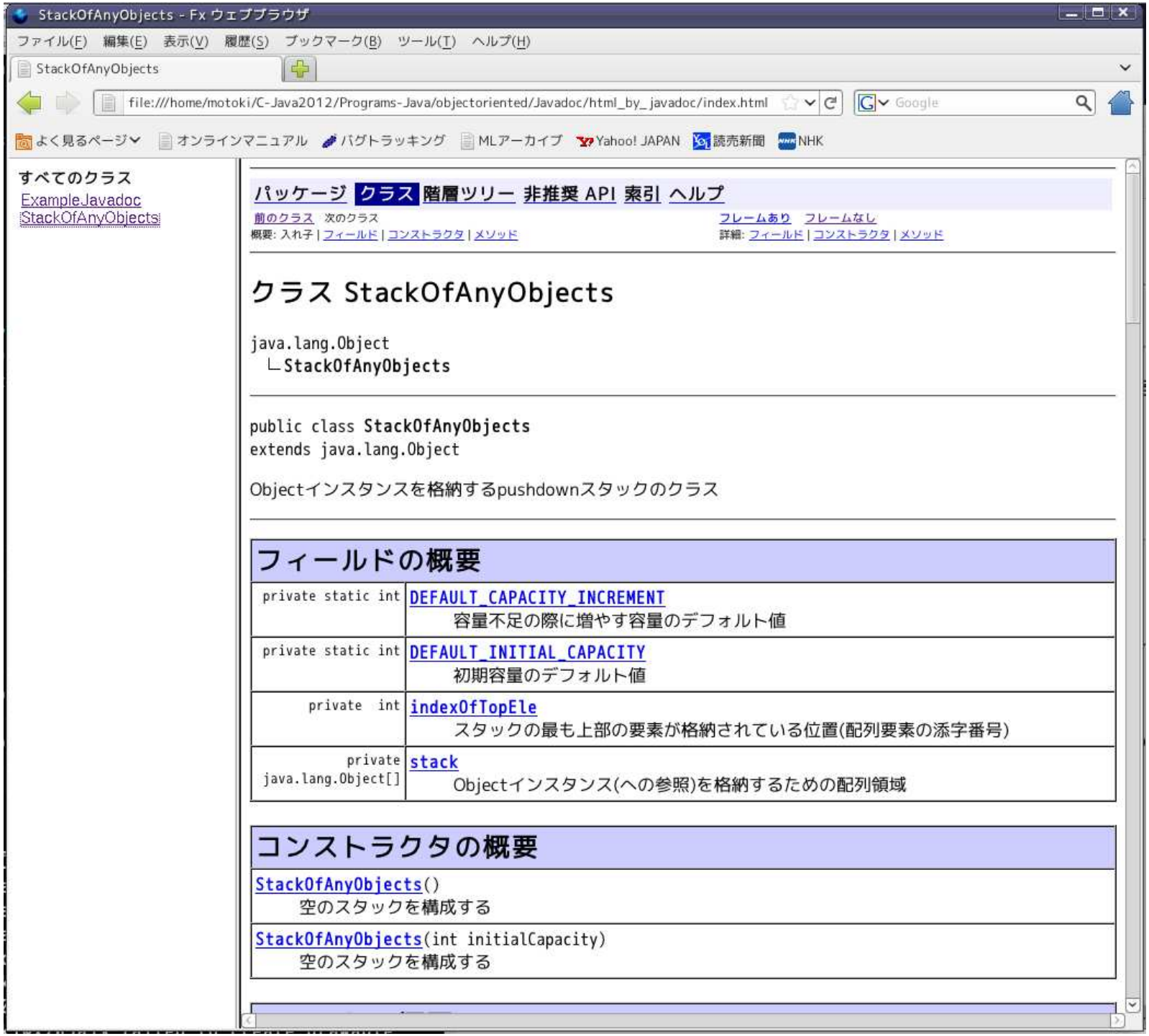
```
103     /**
104      * スタックのtop要素をのぞき見
105      * @return スタックのtop要素(への参照)
106      */
107     public Object peepTop() {
108         if (isEmpty())
109             return null;
110         else
111             return stack[indexOfTopEle];
112     }
113
114     /**
115      * スタックの指定要素をのぞき見
116      * @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
Google
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 ＿＿＿ 整数 という形のデータを読み込んでは、
（＿＿＿：1個以上の空白）

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パッケージ内に用意された

```
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="  + wid
24                  + ", height=" + height + ", color=" + co
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

22–7 抽象クラス

例題14.4では 、
C言語の下でソフトウェア部品の汎用化を進めるために、
3つの整列化モジュール

btree-heapsort.c, bubblesort.c, llistsort.c

の外部仕様(i.e.外向けに提供する外部関数の名前や使い方)の統一

一方Javaでは 、ソフトウェア部品の汎用化を進めるために
「抽象クラス」というものを利用できる。

抽象メソッド...クラス定義の中で本体部の記述がなく、
次の形で宣言されている(インスタンス)メソッド

アクセス修飾子 abstract 他の修飾子, あれば
戻り値の型 メソッド名(引数列);

抽象クラス...抽象メソッドを含み
‘‘abstract’’ という修飾子付きで宣言されたクラス。
インスタンスを生成できない。

抽象クラスを使う利点：

- 統一感のあるプログラムの作成に繋がる。
- メソッドのオーバーライドのやり忘れをコンパイルの時点で防げる。
- 意図しないインスタンス生成をコンパイルの時点で防げる。
- 互いに類似した複数のクラスがある場合、
  - ◇ これらのクラスに共通する部分の詳細だけを記述し、
  - ◇ 個別対応が必要なメソッドについては、抽象メソッドにした

  抽象クラスを用意すれば、
  - ◇ 同じコードをあちこちの場所に書く必要が無くなり、
    コードの冗長さが無くなる。
  - ◇ 各サブクラスにはサブクラス特有の処理だけを書けばよく、
    コードの見通しが良くなる。
  - ◇ 別の類似クラスが新たに必要になった場合も、
    そのクラスを簡単に定義できる様になる。

例22．6（2次元座標上の図形オブジェクトに共通の枠組みを定める抽象ス

2次元座標上の円や長方形、三角形といった図形オブジェクトを多数扱う場合、これらに共通の枠組みとして抽象クラスを考え、個々の種類のオブジェクトのクラスをこの抽象クラスのサブクラスとして定義することにすれば、図形オブジェクト全体を統一的に扱うことができる様になる。

具体例として、

Shape2D ... 図形オブジェクト全体の抽象スーパークラス,
Circle2D ... 円オブジェクトのサブクラス,
Rectangle2D ... 長方形オブジェクトのサブクラス,
Triangle2D ... 三角形オブジェクトのサブクラス

の定義例を次に示す。

```
[motoki@x205a]$  cat -n Shape2D.java
     1 /**
     2  * 頂点等の座標情報を保持する2次元図形オブジェクト
     3  * に共通の枠組みを定める抽象スーパークラス
     4  */
```

```
 5 abstract class Shape2D {
 6     protected final int id;  //図形インスタンスに付ける...
 7     private static int numberOfShapes = 0;
                                //生成した図形インスタンスの個数
 8
 9     /** 2次元座標上に図形オブジェクトの共通部を構成する */
10     public Shape2D() {
11         id = ++numberOfShapes;
12     }
13
14     /** 2次元座標上の図形インスタンスの標準的な文字列... */
15     @Override
16     public abstract String toString();メソッド名の統一
17
18     /** 2次元座標上の図形インスタンスの面積... */
19     public abstract double getArea(); メソッド名の統一
20 }
```

```
[motoki@x205a]$ cat -n Circle2D.java
     1 /**
     2  * 中心の座標と半径の情報を保持する2次元円オブジェクト...
     3  */
     4 class Circle2D extends Shape2D {
     5     private double x;         //円の中心のx座標
     6     private double y;         //円の中心のy座標
     7     private double radius;    //円の半径
     8
     9     /** 2次元座標上に円オブジェクトを構成する */
    10     public Circle2D(double x, double y, double radius)
    11         this.x  = x;
    12         this.y  = y;
    13         this.radius = radius;
    14     }
    15
    16     /** 2次元座標上の円インスタンスの標準的な文字列... */
```

```
17      @Override
18      public String toString() {
19          return "circle[id=" + id + "] of center (" + x
20                  + ") and radius " + radius;
21      }
22
23      /** 2次元座標上の円インスタンスの面積を計算して返す */
24      @Override
25      public double getArea() {
26          return Math.PI*radius*radius;
27      }
28
29      //--------------------単体での動作テスト用------------
30      public static void main(String[] args) {
31          Circle2D fig = new Circle2D(1.0, 0.0, 2.0);
32          System.out.printf("fig = %s%n" +
33                            "  ==> fig.getArea() = %g%n",
```

```
    34                                  fig, fig.getArea());
    35     }
    36 }
[motoki@x205a]$  javac Circle2D.java
[motoki@x205a]$  java Circle2D
fig = circle[id=1] of center (1.0,0.0) and radius 2.0
  ==> fig.getArea() = 12.5664
[motoki@x205a]$  cat -n Rectangle2D.java
     1 /**
     2  * 頂点の座標情報を保持する2次元長方形オブジェクトの...
     3  */
     4 class Rectangle2D extends Shape2D {
     5     private double x0, y0;
                                  //長方形の1つの頂点のx座標,y座標
     6     private double x1, y1;
                                //(x0,y0)と対角の位置にある頂点のy座標
     7
```

```
 8      /** 2次元座標上に長方形オブジェクトを構成する */
 9      public Rectangle2D(double x0, double y0,
                           double x1, double y1) {
10          this.x0 = x0;
11          this.y0 = y0;
12          this.x1 = x1;
13          this.y1 = y1;
14      }
15
16      /** 2次元座標上の長方形インスタンスの標準的な文字列... *
17      @Override
18      public String toString() {
19          return "rectangle[id=" + id + "] of vertices ("
20                  + x0 + "," + y0 + "), ("
21                  + x1 + "," + y0 + "), ("
22                  + x1 + "," + y1 + "), ("
23                  + x0 + "," + y1 + ")";
```

```
24      }
25
26      /** 2次元座標上の長方形インスタンスの面積を計算... */
27      @Override
28      public double getArea() {
29          return Math.abs((x1-x0)*(y1-y0));
30      }
31
32      //--------------------単体での動作テスト用----------
33      public static void main(String[] args) {
34          Rectangle2D fig = new Rectangle2D(0.0, 0.0,
                                                  1.0, 2.0);
35          System.out.printf("fig = %s%n" +
36                            "   ==> fig.getArea() = %g%n",
37                            fig, fig.getArea());
38      }
39  }
```

```
[motoki@x205a]$  javac Rectangle2D.java
[motoki@x205a]$  java Rectangle2D
fig = rectangle[id=1] of vertices (0.0,0.0), (1.0,0.0), (1.0,2
  ==> fig.getArea() = 2.00000
[motoki@x205a]$  cat -n Triangle2D.java
     1 /**
     2  * 頂点の座標情報を保持する2次元三角形オブジェクトの...
     3  */
     4 class Triangle2D extends Shape2D {
     5     private double x0, y0;
                              //三角形の1つの頂点のx座標,y座標
     6     private double x1, y1;
                             //三角形の2つ目の頂点のx座標,y座標
     7     private double x2, y2;
                             //三角形の3つ目の頂点のx座標,y座標
     8
     9     /** 2次元座標上に三角形オブジェクトを構成する */
```

```
10      public Triangle2D(double x0, double y0,
11        double x1, double y1, double x2, double y2) {
12          this.x0 = x0;
13          this.y0 = y0;
14          this.x1 = x1;
15          this.y1 = y1;
16          this.x2 = x2;
17          this.y2 = y2;
18      }
19
20      /** 2次元座標上の三角形インスタンスの標準的な文字列... *
21      @Override
22      public String toString() {
23          return "triangle[id=" + id + "] of vertices ("
24                  + x0 + "," + y0 + "), ("
25                  + x1 + "," + y1 + "), ("
26                  + x2 + "," + y2 + ")";
```

```
27      }
28
29      /** 2次元座標上の三角形インスタンスの面積を計算... */
30      @Override
31      public double getArea() { //ヘロンの公式
32          double sideLeng1 =
                Math.sqrt((x0-x1)*(x0-x1)+(y0-y1)*(y0-y1));
33          double sideLeng2 =
                Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
34          double sideLeng3 =
                Math.sqrt((x2-x0)*(x2-x0)+(y2-y0)*(y2-y0));
35          double s = (sideLeng1+sideLeng2+sideLeng3)/2;
36          return Math.sqrt(s*(s-sideLeng1)
                             *(s-sideLeng2)*(s-sideLeng3));
37      }
38
39      //--------------------単体での動作テスト用----------
```

```
    40      public static void main(String[] args) {
    41          Triangle2D fig = new Triangle2D(0.0, 0.0,
                                                2.0, 0.0, 1.0, 1.0);
    42          System.out.printf("fig = %s%n" +
    43                            "   ==> fig.getArea() = %g%n",
    44                            fig, fig.getArea());
    45      }
    46 }
[motoki@x205a]$  javac Triangle2D.java
[motoki@x205a]$  java Triangle2D
fig = triangle[id=1] of vertices (0.0,0.0), (2.0,0.0), (1.0,1.
  ==> fig.getArea() = 1.00000
[motoki@x205a]$
```

例題22．7（整列化モジュールに共通の枠組みを定める抽象スーパークラ
例題14.4(C言語) で行った、

> 3つの整列化モジュール btree-heapsort.c, bubblesort.c, llistsort.c の外部仕様 (i.e. 外向けに提供する外部関数の名前や使い方) の統一

に相当することをJavaで行ってみよ。

（考え方）　例22.6に倣って、

int配列内の要素を昇順に並べ替える機能を備えた整列化モジュールに共通の枠組みとして抽象クラスを考え、
個々の整列化手法ごとに、その手法で整列化するモジュールのクラスを抽象クラスのサブクラスとして定義すればよい。

その際 、
同一のサブクラスからインスタンスを複数生成しても意味がない
⇨ サブクラスごとにインスタンスを1個だけ生成し、それを使い回す

3つの整列化手法については　Cプログラム

btree-heapsort.c(例題13.2),

bubblesort.c(例題14.4),

llistsort.c(例題14.4)

に記述されている通り。

加筆

線形リスト(例題4のllistsort.cに相当するもの)の実装に関しては 、

- intデータを基本要素として
  線形リストで(小さい順に)保持するオブジェクトがあれば、...
  ⇨ この様な、線形リストを管理するオブジェクトのクラス
  LinkedListOfInt を用意
- 線形リストの要素を表すオブジェクトのクラスNode
  ... 線形リストのクラスの中で局所的に定義

- C言語の場合...要素はデータを入れる領域
  Javaの場合...要素はオブジェクトであり実行主体として動作できる
  ⇨ 個々の要素(Node)をオブジェクトとして捉え、
  これらのオブジェクト間の協調によって
  新規要素の挿入や保持要素の表示作業を行う

（プログラミング）　ここで関連するクラスとして、

SortModuleForIntArray ... 整列化モジュール全体の抽象スーパークラス,

HeapsortIntArray ... heapsortモジュールのサブクラス,

BubblesortIntArray ... bubblesortモジュールのサブクラス,

LListsortIntArray ... 連結リストへの挿入に基づく整列化モジュールのサブクラス,

LinkedListOfInt ... intデータを各節点に保持する連結リストオブジェクトのクラス

```
[motoki@x205a]$ cat -n SortModuleForIntArray.java
     1 /**
     2  * int配列内の要素を昇順に並べ替える機能を備えた整列化
     3  * モジュールに共通の枠組みを定める抽象スーパークラス
     4  */
     5 public abstract class SortModuleForIntArray {
     6     /**
     7      * コンストラクタの代わりに外部に
                                    整列化モジュールを提供する窓口。
     8      * サブクラスを定義する時に、
                            このメソッドは隠蔽されなければならない。
     9      *       (staticなので「オーバーライド」とは言わない)
    10      */                  クラスメソッドなのでabstract宣言不可
    11     public static SortModuleForIntArray getInstance() {
    12         return null;
    13     }
    14
```

```
    15      /** 整列化モジュールの説明(主に手法)を答える */
    16      @Override
    17      public abstract String toString();
    18
    19      /** 引数で与えられた配列内の要素を昇順に並べ替える */
    20      public abstract void sort(int[] a);
    21  }
[motoki@x205a]$  cat -n HeapsortIntArray.java
     1  /**
     2   * int配列内の要素をheapsort手法で昇順に並べ替える機能
     3   * を備えた整列化モジュールを作り出すためのクラス
     4   */
```

```
 5 public class HeapsortIntArray
                                  extends SortModuleForIntArray {
 6     //クラス内部でインスタンスを1個だけ生成
 7     //  (コンストラクタはprivate宣言してあるので、  )
 8     //  (生成されるインスタンスはこの1個だけになり、)
 9     //  (これが使い回されることになる。            )
10     private static final HeapsortIntArray
                          INSTANCE = new HeapsortIntArray();
11
12     //コンストラクタ(外部からインスタンス生成不可)
13     private HeapsortIntArray() {
14         super();
15     }
16
17     /** コンストラクタの代わりに外部に整列化モジュール
                                              を提供する窓口 */
18     public static HeapsortIntArray getInstance() {
19         return INSTANCE;
20     }
```

```
21
22      /** 整列化モジュールの説明(主に手法)を答える */
23      @Override
24      public String toString() {
25          return "Heapsort module";
26      }
27
28      /** 引数で与えられた配列内の要素をheapsort手法で... */
29      @Override
30      public void sort(int[] a) {
31          //下からheapを構築してゆく
32          for (int k=a.length/2-1; k>=0; --k)
33              heapify(a, a.length, k, a[k]);
34
35          //大きい順にheapから取り出してゆく
36          for (int k=a.length-1; k>=1; --k) {
37              int tmp = a[k];
38              a[k]    = a[0];
39              heapify(a, k, 0, tmp);
```

```
40          }
41      }
42
43      /*-----------------<privateメソッド>--------------------
44       * 番号 hole の節点より下の部分がheapの条件を満たす...
45       * 新要素を加えてhole以下の部分がheapの条件を満たす...
46       * (詳細): 番号 hole の節点のデータ記憶域は空で、そ...
47       *     という値がどの節点にも記録されていない、また、hol
48       *     部分がheapの条件を満たしている、という状況を...
49       *     この様な状況の時に、holeより下にあるデータを上...
50       *     する操作を繰り返し行い、適当な時点で空の節点に...
51       *     newElement を割り当てることにより、hole以下...
52       *     heapの条件を満たす様にする。
53       *
54       * @param  a           int型配列
55       * @param  treeSize    2分木と見做す部分配列(a[0],a[1],
56       * @param  hole        a[0]~ a[treeSize-1]の表す2分木..
57       * @param  newElement  2分木の節点に振り分けていない...
58       */
```

```
59      private void heapify(int[] a, int treeSize,
                             int hole, int newElement) {
60          int  siftupCand;       //siftup candidate
61
62          while ((siftupCand = hole*2+1) < treeSize) {
63              if (siftupCand+1<treeSize                  //右.
64                  && a[siftupCand]<a[siftupCand+1]) //右.
65                  ++siftupCand;                          //大.
66                                                  //右の子が
67              if ( newElement >= a[siftupCand])
68                  break;          //newElementをholeの場...
69
70              a[hole] = a[siftupCand];      //sift up
71              hole    = siftupCand;
72          }
73          a[hole] = newElement;
74      }
75
76      //--------------------単体での動作テスト用----------
```

```
    77      public static void main(String[] args) {
    78          int[] a = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
    79          getInstance().sort(a);
    80          System.out.println("after sorting ("
                                        + getInstance() + "):");
    81          System.out.print("  a = {");
    82          for (int i=0; i<a.length-1; ++i)
    83              System.out.print(a[i] + ", ");
    84          System.out.println(a[a.length-1] + "}");
    85      }
    86 }
[motoki@x205a]$  javac HeapsortIntArray.java
[motoki@x205a]$  java HeapsortIntArray
after sorting (Heapsort module):
  a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[motoki@x205a]$  cat -n BubblesortIntArray.java
     1 /**
     2  * int配列内の要素をbubblesort手法で昇順に並べ替える機能
     3  * を備えた整列化モジュールを作り出すためのクラス
```

```
 4  */
 5 public class BubblesortIntArray
                                  extends SortModuleForIntArray {
 6     //クラス内部でインスタンスを1個だけ生成
 7     //  (コンストラクタはprivate宣言してあるので、  )
 8     //  (生成されるインスタンスはこの1個だけになり、)
 9     //  (これが使い回されることになる。            )
10     private static final BubblesortIntArray
                        INSTANCE = new BubblesortIntArray();
11
12     //コンストラクタ(外部からインスタンス生成不可)
13     private BubblesortIntArray() {
14         super();
15     }
16
17     /** コンストラクタの代わりに外部に整列化モジュールを...
18     public static BubblesortIntArray getInstance() {
19         return INSTANCE;
20     }
```

```
21
22      /** 整列化モジュールの説明(主に手法)を答える */
23      @Override
24      public String toString() {
25          return "Bubblesort module";
26      }
27
28      /** 引数で与えられた配列内の要素をbubblesort手法で... *
29      @Override
30      public void sort(int[] a) {
31          for (int i=0; i<a.length-1; ++i) {
32              for (int j=a.length-1; j>i; --j) {
33                  if (a[j-1] > a[j]) {
34                      int temp = a[j-1];  //a[j-1] と a[j]
35                      a[j-1]   = a[j];    //の大小を調べ...
36                      a[j]     = temp;    //逆順なら交換
37                  }
38              }
```

```
    39          }
    40      }
    41
    42      //--------------------単体での動作テスト用------------
    43      public static void main(String[] args) {
    44          int[] a = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
    45          getInstance().sort(a);
    46          System.out.println("after sorting ("
                                        + getInstance() + "):");
    47          System.out.print("  a = {");
    48          for (int i=0; i<a.length-1; ++i)
    49              System.out.print(a[i] + ", ");
    50          System.out.println(a[a.length-1] + "}");
    51      }
    52 }
[motoki@x205a]$  java BubblesortIntArray
after sorting (Bubblesort module):
  a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[motoki@x205a]$  cat -n LListsortIntArray.java
```

```
 1 /**
 2  * (1)int配列内の要素を次々と連結リストの大小順を保つ
                                        位置に挿入していき、
 3  * それが終わったら、(2)連結リストに保持されたものを
                                        順にint配列内に移す、
 4  * という手法で昇順に並べ替える機能を備えた
 5  * 整列化モジュールを作り出すためのクラス
 6  */
 7 public class LListsortIntArray
                                extends SortModuleForIntArray {
 8     //クラス内部でインスタンスを1個だけ生成
 9     //  (コンストラクタはprivate宣言してあるので、  )
10     //  (生成されるインスタンスはこの1個だけになり、)
11     //  (これが使い回されることになる。            )
12     private static final LListsortIntArray
                          INSTANCE = new LListsortIntArray();
13
14     //コンストラクタ(外部からインスタンス生成不可)
15     private LListsortIntArray() {
```

```
16          super();
17      }
18
19      /** コンストラクタの代わりに外部に整列化モジュールを...
20      public static LListsortIntArray getInstance() {
21          return INSTANCE;
22      }
23
24      /** 整列化モジュールの説明(主に手法)を答える */
25      @Override
26      public String toString() {
27          return "sort module that is based on insertion
28      }
29
30      /** 連結リストへの挿入に基づく手法で、
31       * 引数で与えられた配列内の要素を昇順に並べ替える */
32      @Override
33      public void sort(int[] a) {
34          LinkedListOfInt list = new LinkedListOfInt();
```

```
35
36          //配列内の要素を連結リストに挿入(大小順を保つ)
37          for (int i=0; i<a.length; ++i)
38              list.insert(a[i]);
39
40          //連結リストに保持されたものを順にint配列内に移...
41          list.getOutContents(a);
42      }
43
44      //--------------------単体での動作テスト用------------
45      public static void main(String[] args) {
46          int[] a = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
47          getInstance().sort(a);
48          System.out.println("after sorting ("
                                    + getInstance() + "):");
49          System.out.print("  a = {");
50          for (int i=0; i<a.length-1; ++i)
51              System.out.print(a[i] + ", ");
52          System.out.println(a[a.length-1] + "}");
```

```
    53     }
    54 }
[motoki@x205a]$  cat -n LinkedListOfInt.java
     1 /**
     2  * intデータ群を小さい順に連結リストで保持する
                                                オブジェクトのクラス
     3  */
     4 public class LinkedListOfInt {
     5     //連結リストを構成する基本要素を表すオブジェクト
                                                    のクラス
     6     private static class Node {  入れ子クラス
     7         private int    intData;
     8         private Node   next;
     9
    10         //コンストラクタ
    11         public Node(int intData, Node next) {
    12             this.intData    = intData;
    13             this.next       = next;
    14         }
```

```
15
16          //Nodeインスタンスの標準的な文字列表現を定める
17          @Override
18          public String toString() {
19              return "node of int data " + intData;
20          }
21
22          //このNode以降の「小さい順」を保てる場所に新要...
23          public void insertAscendingOrder(int newInt) {
24              if (next==null || newInt<=next.intData)
25                  next = new Node(newInt, next);
26              else
27                  next.insertAscendingOrder(newInt);
28          }
29
30          //このNode以降に蓄えられた内容を配列aのindex...
31          public void getOutContents(int[] a, int index)
32              a[index] = intData;
33              if (next != null)
```

```
34                  next.getOutContents(a, index+1);
35              }
36          }
37          //-----------------------------------------
38
39          private Node head;
40
41          /** intデータを小さい順に保持する連結リストとして空の...
42          public LinkedListOfInt() {
43              head = null;
44          }
45
46          /** LinkedListOfIntインスタンスの標準的な文字列表現...
47          @Override
48          public String toString() {
49              return "linked list of int data\n"
50                      + "   that are sorted by ascending orde
51          }
52
```

```
    53      /** データの小さい順を保てる場所に(引数の)新要素を挿...
    54      public void insert(int newInt) {
    55          if (head==null || newInt<=head.intData)
    56              head = new Node(newInt, head);
    57          else
    58              head.insertAscendingOrder(newInt);
    59      }
    60
    61      /** 連結リストに蓄えられた内容を記録された順に(引数の).
    62      public void getOutContents(int[] a) {
    63          if (head != null)
    64              head.getOutContents(a, 0);
    65      }
    66 }
[motoki@x205a]$  javac LListsortIntArray.java
[motoki@x205a]$  java LListsortIntArray
after sorting (sort module that is based on insertion in a lir
  a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[motoki@x205a]$
---
```

22–8 Javaの扱う型とラッパークラス，ボクシン

Javaの扱うデータ型の分類： Javaにおけるデータ型は次のように分類できる。

- **基本データ型**
  - 数値型
    - 整数型
      - byte ・・・8ビット符号付き2の補数表現の整数
      - short・・・16ビット符号付き2の補数表現の整数
      - int ・・・32ビット符号付き2の補数表現の整数
      - long ・・・64ビット符号付き2の補数表現の整数
    - 浮動小数点数型
      - float ・・・32ビットIEEE規格754実数
      - double・・・64ビットIEEE規格754実数
  - ブール型 boolean・・・ true か false を表す。
  - 文字型 char・・・ 16ビットUnicode **UTF-16** で文字を表す。
- **参照型**
  - **クラス型** ・・・クラス定義に基いて生成されるインスタンスを表す。
  - **インタフェース型**・・・外部へのサービスのための指定されたメソッドを実装したオブジェクトであることを表す。
  - **配列型** ・・・同じデータ型の記憶領域の並びを表す。

ラッパークラス:

基本データ型のデータをオブジェクトとして扱いたい

⇨ 基本データ型データをカプセル化したオブジェクトのクラス
が用意されている。 ラッパークラス

具体的には、

基本データ型		ラッパークラス	基本データ型		ラッパークラス
byte	⟶	Byte	float	⟶	Float
short	⟶	Short	double	⟶	Double
int	⟶	Integer	boolean	⟶	Boolean
long	⟶	Long	char	⟶	Character

例えば

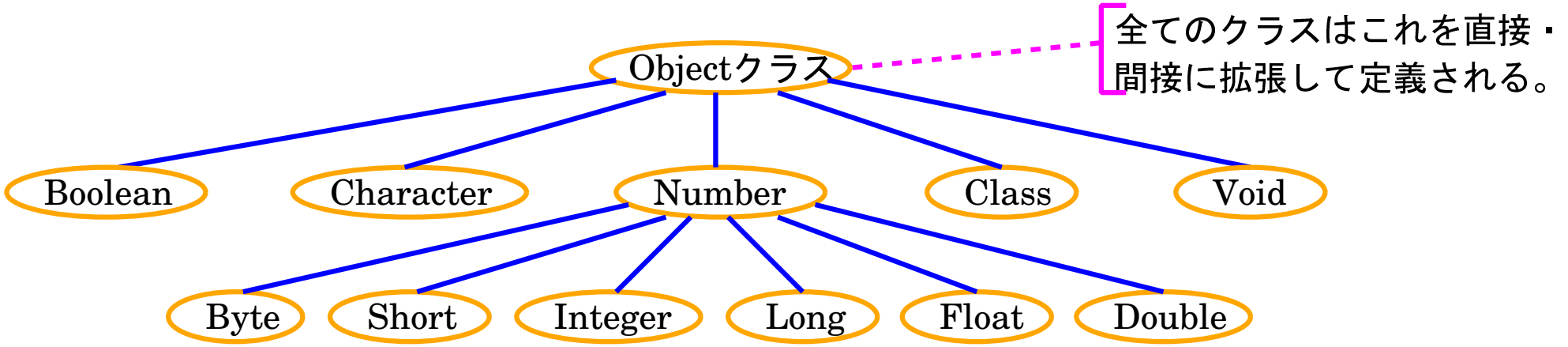
```
new Integer( int型の式 )
```

という風に書けばintデータをカプセル化したオブジェクトが出来る。

クラス階層は ⟶次の通り

クラス階層は $\longrightarrow$ 次の通り

（数値型全体を統合的に扱うために Number という抽象クラス）



補足：

Void...メソッドの戻り値がないことを表すvoidに対応。

Class...クラスについての情報を保持しているオブジェクトのクラス。

（ついでに書いたが、これはラッパークラスではない。）

ラッパークラスに備わっているフィールドとメソッド：

Voidクラスを除く各ラッパークラスに、次の定数、コンストラクタ、メ

ソッドがほぼ共通に備わっている。

定数(クラスフィールド)

メソッド名 ··· 説明
public static final 対応する基本データ型 MIN_VALUE (Boolean以外) ··· 対応する基本データ型で表現可能な最小値。
public static final 対応する基本データ型 MAX_VALUE (Boolean以外) ··· 対応する基本データ型で表現可能な最大値。
public static final int SIZE (Boolean以外) ··· 対応する基本データ型で値を表現するのに使われるビット数。
public static final Class<ラッパークラス> TYPE ··· ラッパークラスに対応する基本データ型を表しているClassオブジェクトへの参照。

コンストラクタ

メソッド名 ··· 説明
ラッパークラス（対応する基本データ型 a）   ··· 基本データ a を基にラッパークラスのインスタンスを生成。
ラッパークラス（String s）   ··· 文字列 s をラッパークラスに対応する基本データを表すと見做して、その表すデータを基にラッパークラスのインスタンスを生成。

クラスメソッド

メソッド名 ··· 説明
public static ラッパークラス valueOf(対応する基本データ型 a)
··· 基本データaを基にラッパークラスのインスタンスを生成し、そこへの参照を返す。
public static ラッパークラス valueOf(String s) (Character以外)
··· 文字列sをラッパークラスに対応する基本データを表すと見做して、その表すデータを基にラッパークラスのインスタンスを生成し、そこへの参照を返す。
public static 対応する基本データ型 parseラッパークラス(String s) (Character以外)
··· 文字列sをラッパークラスに対応する基本データを表すと見做して、その表す基本データを返す。
public static String toString(対応する基本データ型 a)
··· 引数で指定された値aの対応する基本データ型としての文字列表現を返す。

インスタンスメソッド

メソッド名 … 説明
public int compareTo(ラッパークラス *other*) … *other* と比較して、*other* より小の時は負、同じ時は0、*other* より大の時は正の値を返す。
public 対応する基本データ型 対応する基本データ型Value() … メソッド実行の依頼を受けたラッパーオブジェクトの保持している基本データ値を返す。
public String toString() … オブジェクトの文字列表現を返す。(Objectクラスのメソッドをオーバーライド)
public boolean equals(Object *obj*) … *obj* と同じ型で同じ値を保持しているかどうかの判定結果を返す。(Objectクラスのメソッドをオーバーライド)
public int hashCode() … ハッシュテーブルで使用されているハッシュコードを返す。(Objectクラスのメソッドをオーバーライド)

自動ボクシング，自動アンボクシング:

ボクシング（変換） . . . 基本データ型 $\longrightarrow$ ラッパークラス型

アンボクシング（変換） . . . ラッパークラス型 $\longrightarrow$ 基本データ型

JDK1.5(J2SE5.0,2004)以降では 、

必要に応じて自動的にボクシング変換やアンボクシング変換が行われる。

例えば

```
Integer x = 100;
int a = x;
x = x + 20;
```

というコードが与えられた場合、コンパイラは次のコードの省略形が与えられたものと見做してコンパイル作業を行う。

```
Integer x = new Integer(100);
int a = x.intValue();
x = new Integer(x.intValue() + 20);
```

例22. 8 (J.Bloch(2008),p.23; 自動ボクシングで多数のオブジェクトが

次の様なコードを平気で書く様になるかもしれない。

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

```
 1 /**
 2  * 自動ボクシングにより多数のオブジェクトが生成され処理が
 3  * 遅くなる可能性があることを例示するためのJavaプログ...
 4  */
 5 public class RemarkOnSuperfluousBoxingMain {
 6     public static void main(String[] args) {
 7         Long sum = 0L;
 8         for (long i=0; i<=Integer.MAX_VALUE; ++i) {
 9             sum += i;
10         }
11         System.out.println("0+1+2+...+Integer.MAX_VALUE
12     }
13 }
```

```
[motoki@x205a]$  javac RemarkOnSuperfluousBoxingMain.java
[motoki@x205a]$  time java RemarkOnSuperfluousBoxingMain
0+1+2+...+Integer.MAX_VALUE = 2305843008139952128


real 0m15.472s
user 0m15.385s
sys 0m0.177s
[motoki@x205a]$
```

しかし、このプログラムの場合 、
自動ボクシング変換により 全部で約 2^{31} 個のLongインスタンスが生成されることになり、実行時間は約15.4秒となる。
一方 、
7行目で宣言している型をLong型 から long型 に変更するだけで、
9行目でのオブジェクト生成が無くなり、その結果、次の会話例で示される様にプログラムの実行時間は約5.3秒に短縮される。

```
[motoki@x205a]$  cat -n RemarkOnSuperfluousBoxingMain2.java
```

```
 1 /**
 2  * 自動ボクシングにより多数のオブジェクトが生成され処理
 3  * が遅くなる可能性があることを例示するためのJavaプロ...
 4  */
 5 public class RemarkOnSuperfluousBoxingMain2 {
 6     public static void main(String[] args) {
 7         long sum = 0L;
 8         for (long i=0; i<=Integer.MAX_VALUE; ++i) {
 9             sum += i;
10         }
11         System.out.println("0+1+2+...+Integer.MAX_VALUE
12     }
13 }
```

```
[motoki@x205a]$  javac RemarkOnSuperfluousBoxingMain2.java
[motoki@x205a]$  time java RemarkOnSuperfluousBoxingMain2
0+1+2+...+Integer.MAX_VALUE = 2305843008139952128
```

```
real 0m5.294s
user 0m5.278s
sys 0m0.013s
[motoki@x205a]$
```

22–9 多態性

一般のプログラミング言語で 、
1つの変数やメソッド修正(または関数呼出し)が実行時の状況により色々に振舞える能力を多態性(多相性, ポリモーフィズム)と呼ぶ。

多態変数 . . . 実行の状況に応じて色々なデータ型の値を持てる変数
多態引数 . . . 〃 引数
(純)多態関数 . . . 多態引数を持つ関数

関数実体(1つ)の内部で、実引数のデータ型に応じて適切な処理内容が選択される。

多重定義 . . . 1つの関数名や演算記号に対して
引数のデータ型毎に別々の関数本体, 演算処理が定義される

⇨ 多態関数を呼び出す側では
___ 多態引数のデータ型に応じて呼出す関数を切り替える必要が無い。

例22．9（C言語における演算記号の多重定義）

+演算子
- 整数 の加算を行う時の処理
- 実数 の加算を行う時の処理
- (Java) 文字列を繋げる働き

```
System.out.println("e = " + e);
```

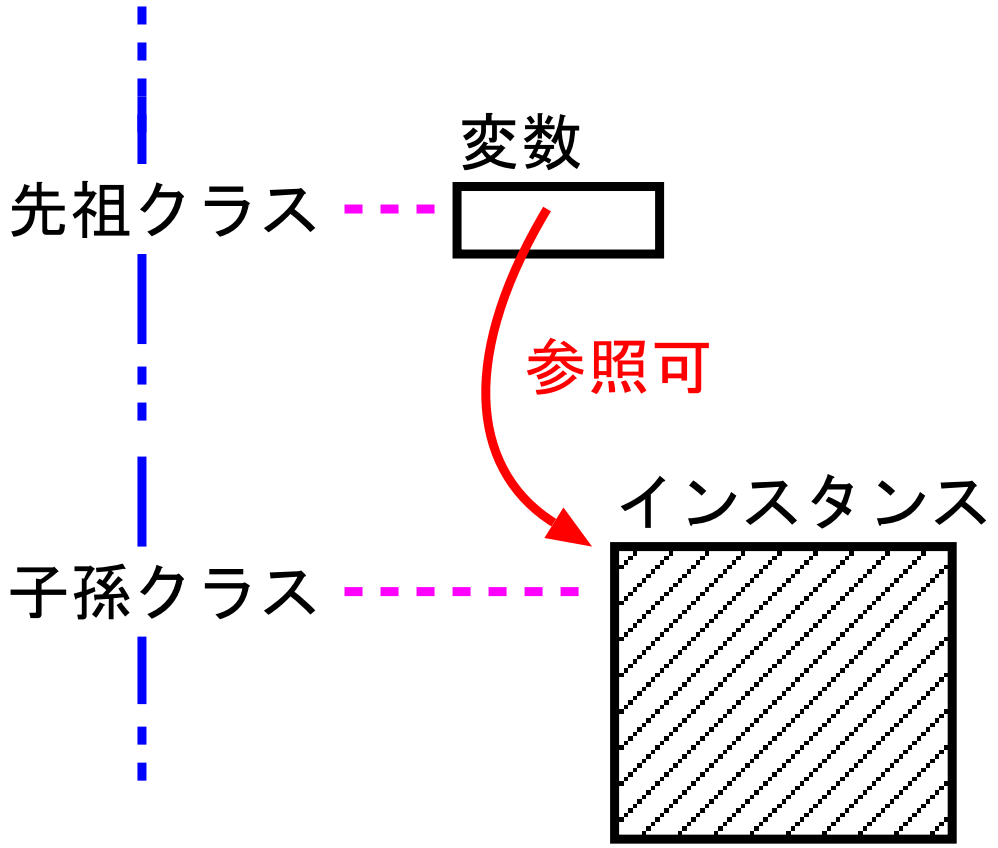
非オブジェクト指向言語における
代表的な多重定義・多態性の例

例22. 10 (Java, メソッドの多重定義)

```
[motoki@x205a]$  cat -n ExampleOfOverloadMain.java
     1 /**
     2  * メソッドの多重定義を例示するためのJavaプログラム
     3  */
     4 public class ExampleOfOverloadMain {
     5     public static void main(String args[]) {
     6         display(33);
     7         display(33.0);
     8     }
     9
    10     static void display(int num){   多重定義
    11         System.out.println("int :" + num);
    12     }
    13
    14     static void display(double num){   多重定義
    15         System.out.println("double :" + num);
    16     }
    17 }
[motoki@x205a]$  javac ExampleOfOverloadMain.java
[motoki@x205a]$  java ExampleOfOverloadMain
int :33
double :33.0
[motoki@x205a]$
```

スーパークラスとサブクラスの関係を親子関係と見たクラス階層で 、

- 祖先に位置するクラスの型を持つ変数は 、その子孫に位置するクラスのインスタンス(への参照)を値として持つことが出来る。



⇨ (少なくとも潜在的には) どれも多態変数になる。

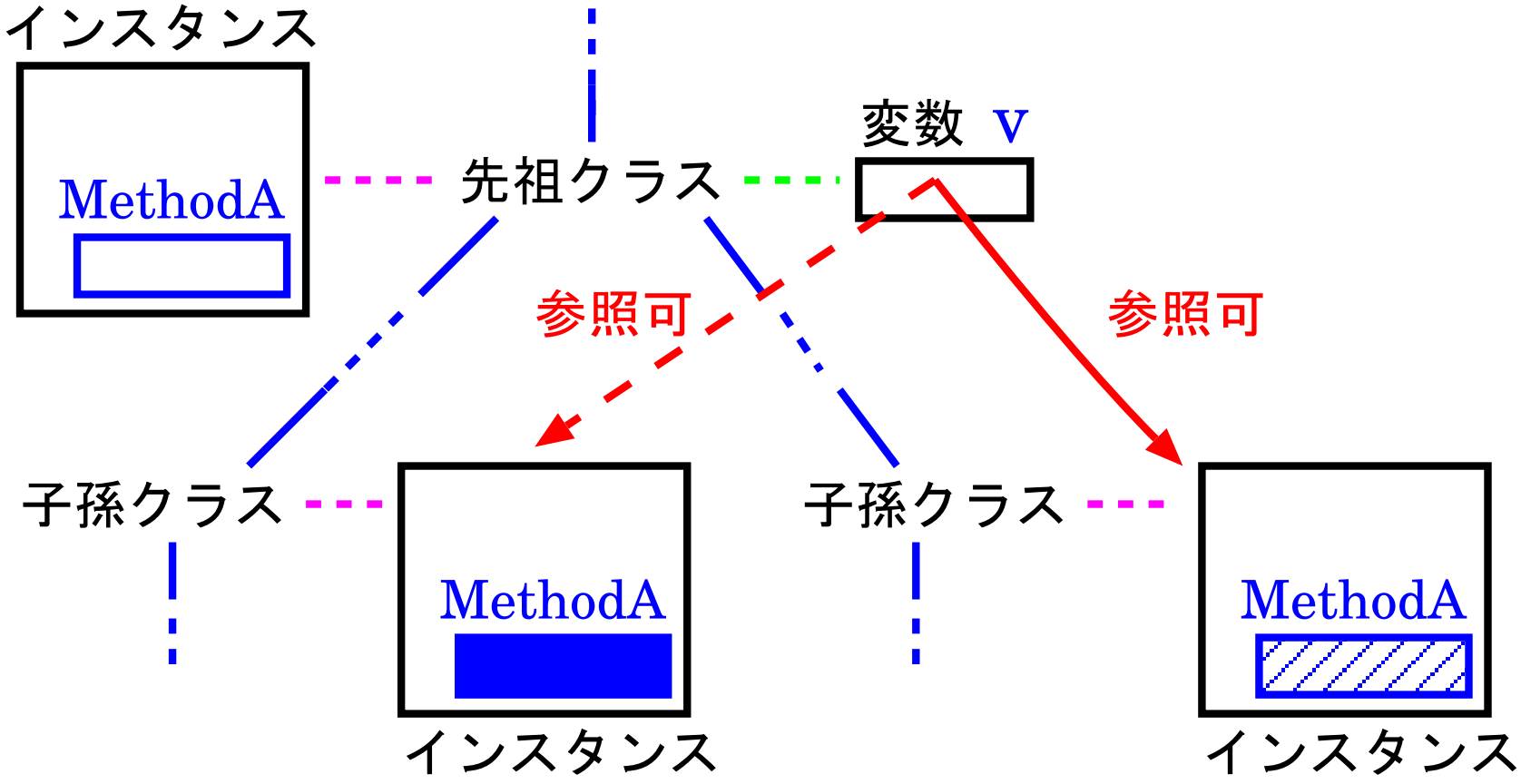
- クラス階層の葉節点以外の位置にクラス Ancestor があり、そのクラス内にインスタンスメソッド MethodA が備わっているとする。いま、

```
Ancestor v ;
```

という変数 v が子孫クラスのインスタンスを参照する時、

プログラム内で **v.MethodA(** 引数列 **)** と書くことによって 、v の参照するインスタンス自身が保持するインスタンスメソッド MethodA が呼び出される。

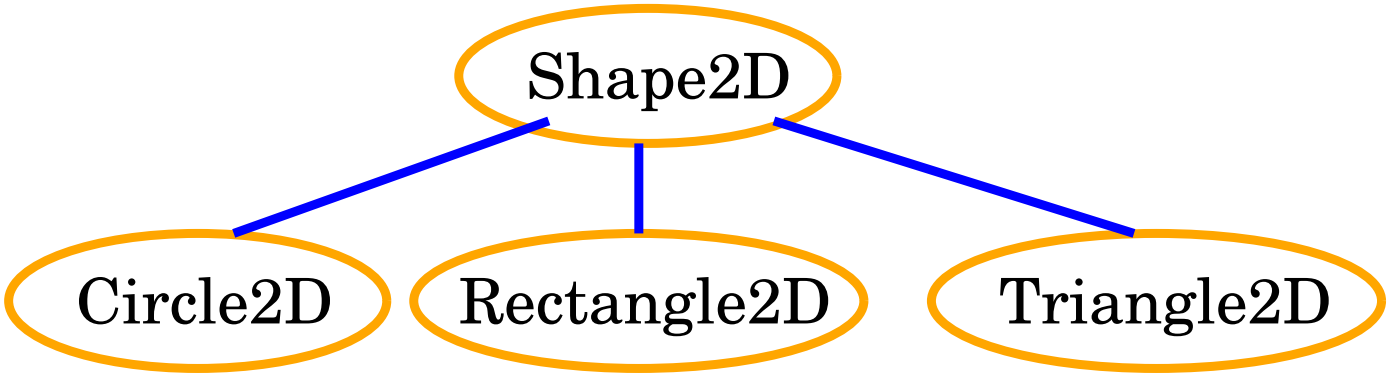
... (MethodAがオーバーライドされていた場合は、Ancestor内で定義されたMethodAではなくオーバーライドされた結果のMethodAが呼び出される。)



⇨ 子孫クラスの各々が独自にオーバーライドしたインスタンスメソッドMethodAを持つ場合は、字面上は同じv.MethodA(引数列)
でも、vがどの子孫クラスのインスタンスになっているかに応じてMethodAは色々な振る舞い(多態性)を見せてくれる。

例22．11（2次元座標上の図形オブジェクト群の統一的な扱い，メソッド

例22.6で考えた4つのクラス Shape2D, Circle2D, Rectangle2D, Triangle2D の間には



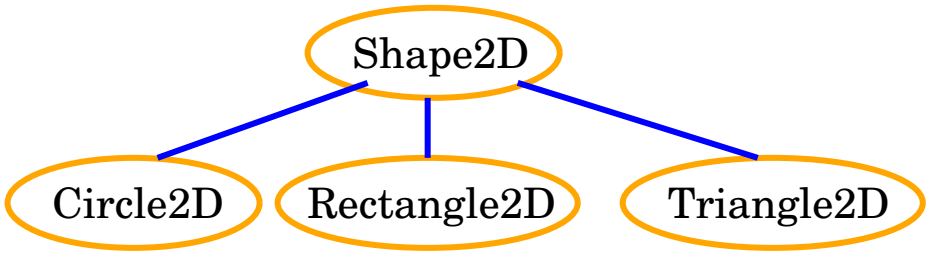
というクラス継承関係があり、

```
Shape2D fig;
```

と宣言されていれば、変数 fig は Circle2D のインスタンス(への参照)を値として持つこともできるし、Rectangle2D や Triangle2D のインスタンス(への参照)を値として持つこともできる。

⇨ 変数 fig は多態変数

さらに、



- スーパー(抽象)クラスである Shape2D の中で宣言されている getArea() というインスタンスメソッドは各々のサブクラス内でオーバーライドされているので、

⇨ fig.getArea()

と書くだけで、

fig が Circle インスタンスであるかどうか、
Rectangle インスタンスであるかどうか、
Triangle インスタンスであるかどうか

に応じて、自動的に適切な処理を選択して計算結果を返してくれる。

同様に、toString() も多態的に振舞う

多態性によって、
⇨ メソッドを使う側への負担は少なくなる。

(以上のことを実際に確かめるJavaプログラム)

```
[motoki@x205a]$  cat -n TestShape2DMain.java
     1 /**
     2  * 次のクラス内で宣言／定義されたインスタンスメソッド toStri
     3  * getArea() が多態的に振舞うことを確認するためのJava...
     4  *  ・Shape2D ... 2次元図形に共通の枠組みを定める抽象
                                                    スーパークラス
     5  *  ・Circle2D ... 2次元円オブジェクトのクラス,
     6  *  ・Rectangle2D ... 2次元長方形オブジェクトのクラス,
     7  *  ・Triangle2D ... 2次元三角形オブジェクトのクラス
     8  */
     9 class TestShape2DMain {
    10     public static void main(String args[]) {
    11         Shape2D fig;
    12
    13         fig = new Circle2D(1.0, 0.0, 2.0);
    14         System.out.printf("fig = %s%n" +
    15                           "   ==> fig.getArea() = %g%n",
    16                           fig, fig.getArea());
    17
```

```
18            fig = new Rectangle2D(0.0, 0.0, 1.0, 2.0);
19            System.out.printf("fig = %s%n" +
20                              "   ==> fig.getArea() = %g%n",
21                              fig, fig.getArea());
22
23            fig = new Triangle2D(0.0, 0.0, 2.0, 0.0, 1.0, 1
24            System.out.printf("fig = %s%n" +
25                              "   ==> fig.getArea() = %g%n",
26                              fig, fig.getArea());
27        }
28 }
[motoki@x205a]$  javac TestShape2DMain.java
[motoki@x205a]$  java TestShape2DMain
fig = circle[id=1] of center (1.0,0.0) and radius 2.0
   ==> fig.getArea() = 12.5664
fig = rectangle[id=2] of vertices (0.0,0.0), (1.0,0.0), (1.0,2
   ==> fig.getArea() = 2.00000
fig = triangle[id=3] of vertices (0.0,0.0), (2.0,0.0), (1.0,1.
   ==> fig.getArea() = 1.00000
[motoki@x205a]$
```

補足：

プログラム 11行目 を　Object fig;　と書き換えた場合、figは多態変数になる。

しかし、このfigを多態的に振舞わせようとしても適用できるインスタンスメソッドは元々Objectに備わっていた名前のものだけ

```
[motoki@x205a]$  cat TestShape2DMain2.java
class TestShape2DMain2 {
    public static void main(String args[]) {
        Object fig = new Circle2D(1.0, 0.0, 2.0);
        System.out.printf("fig = %s%n", fig);
    }
}
[motoki@x205a]$  javac TestShape2DMain2.java
[motoki@x205a]$  java TestShape2DMain2
fig = circle[id=1] of center (1.0,0.0) and radius 2.0
[motoki@x205a]$  cat TestShape2DMain3.java
class TestShape2DMain3 {
```

```
    public static void main(String args[]) {
        Object fig = new Circle2D(1.0, 0.0, 2.0);
        System.out.printf("  ==> fig.getArea() = %g%n",
                          fig.getArea());
    }
}
[motoki@x205a]$  javac TestShape2DMain3.java
TestShape2DMain3.java:5: シンボルを見つけられません。
シンボル: メソッド getArea()
場所    : java.lang.Object の クラス
                          fig.getArea());
                              ^
エラー 1 個
[motoki@x205a]$
```

例題22．12（整列化モジュールの動作テストを担当するモジュール）
例題22.7で定義した3つのクラス HeapsortIntArray，BubblesortIntArray，LListsortIntArray は共通の抽象スーパークラスをもっている。従って、多態変数／多態性の考えを用いることにより、これらのインスタンス(整列化モジュール)を統合的に扱うことが出来る。これを体験するために、この種の整列化モジュールの提供する「int配列内の要素を昇順に並べ替える機能」が正しく動作するかどうかを例題13.2に倣ってテストする機能、すなわち

①0~ 999の間のランダムな整数を要素とする
大きさ100の配列を生成し、
②それに対して与えられた整列化モジュールを適用して
並べ替え作業を行い、
③その結果を出力する、

という風にテストする機能を備えたモジュールのクラスを定義してみよ。

（考え方）

共通の抽象スーパークラスが SortModuleForIntArray

⇨ SortModuleForIntArray型の変数を引数に持ち、
引数で与えられた整列化モジュールに対して
所定の動作テストを施すインスタンスメソッド
を備えたモジュールのクラスを定義すれば良い。

その際、

- 擬似乱数の生成に関しては
  ライブラリ内の java.util.Random というクラスを利用できる。

（プログラミング）

```
[motoki@x205a]$ cat -n TesterForSortModuleIntArray.java
     1 import java.util.Scanner;
     2 import java.util.Random;
     3
     4 /**
     5  * SortModuleForIntArrayモジュールの提供する
     6  * 「int配列内の要素を昇順に並べ替える機能」が
                                              正しく動作するかどうか
     7  * をテストする機能を備えたモジュールを作り出すためのクラ...
     8  */
     9 public class TesterForSortModuleIntArray {
    10     private static final int SIZE  =100;
    11     private static final int WIDTH = 10;
    12
    13     private Scanner inputScanner;
    14
```

```
15      /** 整列化モジュールの動作テストを行うモジュールを構.. *
16      public TesterForSortModuleIntArray() {
17          this.inputScanner = new Scanner(System.in);
18      }
19
20      /** 指定された所から得た擬似乱数シードを用いて
21       * 整列化モジュールの動作テストを行うモジュールを構成..
22      public TesterForSortModuleIntArray(
                                        Scanner inputScanner) {
23          this.inputScanner = inputScanner;
24      }
25
26      /** オブジェクトの説明を答える */
27      @Override
28      public String toString() {
29          return "Tester for module that is to sort int c
30      }
31
```

```
32      /** SIZE個のランダムなデータから成る配列に対して
33       * 引数で与えられた整列化モジュールを実行してみる */
34      public void runOnRandomData(抽象クラス    多態変数
                        SortModuleForIntArray sortModule) {
35          int[] a = new int[SIZE];
36
37          //擬似乱数の設定
38          System.out.print("擬似乱数の初期シード(long値): "
39          long seed = inputScanner.nextLong();
40          Random randomGenerator = new Random(seed);
41
42          //配列aの各々の要素に0~ 999の乱数値を設定
43          for (int i=0; i<a.length; ++i)
44              a[i] = randomGenerator.nextInt(1000);
45
46          //整列化前の配列の内容を表示
47          System.out.printf("%nbefore sorting:%n");
48          prettyPrint(a);
49
```

```
50          //整列化
51          sortModule.sort(a);
52
53          //整列化後の配列の内容を表示
54          System.out.println("after sorting("
                                        + sortModule + "):");
55          prettyPrint(a);
56      }
57
58      /*-----------------<privateメソッド>-------------------
59       * 引数で与えられた配列の要素を順に全て出力(1行にWIDTH
60      private void prettyPrint(int[] a) {
61          int NumOfEleInLine=0;
62
63          for (int i=0; i<a.length; ++i) {
64              System.out.printf("%7d", a[i]);
65              ++NumOfEleInLine;
66              if (NumOfEleInLine >= WIDTH) {
67                  System.out.println();
```

```
     68                     NumOfEleInLine = 0;
     69                 }
     70             }
     71             if (NumOfEleInLine > 0)
     72                 System.out.println();
     73         }
     74 }
[motoki@x205a]$  cat -n TestSortModulesIntArrayMain.java
      1 /**
      2  * int配列内の要素を昇順に並べ替える機能を備えた整列化モ...
      3  *  ・HeapsortIntArrayオブジェクト,
      4  *  ・BubblesortIntArrayオブジェクト,
      5  *  ・LListsortIntArrayオブジェクト
      6  * の3つを考え、これらが正しく整列化動作をするかどうかを
      7  *      整列化モジュールをテストする機能を備えた
      8  *      TesterForSortModuleIntArrayオブジェクト
      9  * を用いてテストするJavaプログラム
     10  */
     11 public class TestSortModulesIntArrayMain {
```

```
12      public static void main(String[] args) {
13          TesterForSortModuleIntArray
14              tester = new TesterForSortModuleIntArray();
15                                  色々な部品の組み合わせを試す
16          //HeapsortIntArrayオブジェクトの動作テスト
17          tester.runOnRandomData(HeapsortIntArray.getInst
18          System.out.println("---");
19
20          //BubblesortIntArrayオブジェクトの動作テスト
21          tester.runOnRandomData(BubblesortIntArray.getIn
22          System.out.println("---");
23
24          //LListsortIntArrayオブジェクトの動作テスト
25          tester.runOnRandomData(LListsortIntArray.getIns
26      }
27  }
[motoki@x205a]$  javac TestSortModulesIntArrayMain.java
[motoki@x205a]$  java TestSortModulesIntArrayMain
擬似乱数の初期シード(long値):  333
```

```
before sorting:
    386    503    585    138    315    941    443    328    98
    494    602    849    712    734    139    870    765    99
    718    359    663    733    234    282    666    745    36
    874    744    227    650    795    316    249     48    46
    166    364    415     13    926    798    309    953    21
    645    724    853    544    714    613    327      5    82
    827    217    653    460    894    803    330    169    44
    223    954     37    975    545    277    690    135    45
    170    408    946    437    535     93    918    660    90
    291    705    165    242    910    344    880    679    97
after sorting(Heapsort module):
      5     13     37     48     82     93    135    138    13
    166    169    170    210    217    223    227    234    23
    245    249    277    282    291    309    315    316    32
    330    344    359    361    364    386    408    415    42
    437    443    447    456    460    465    494    503    51
    544    544    545    585    602    613    618    645    65
    660    663    666    679    690    705    712    714    71
```

```
    733     734     734     744     745     765     795     798     80
    827     849     853     870     874     880     894     901     90
    918     926     941     946     953     954     975     978     98
---
擬似乱数の初期シード(long値):  333

before sorting:
    386     503     585     138     315     941     443     328     98
    494     602     849     712     734     139     870     765     99
    718     359     663     733     234     282     666     745     36
    874     744     227     650     795     316     249      48     46
    166     364     415      13     926     798     309     953     21
    645     724     853     544     714     613     327       5     82
    827     217     653     460     894     803     330     169     44
    223     954      37     975     545     277     690     135     45
    170     408     946     437     535      93     918     660     90
    291     705     165     242     910     344     880     679     97
after sorting(Bubblesort module):
      5      13      37      48      82      93     135     138     13
```

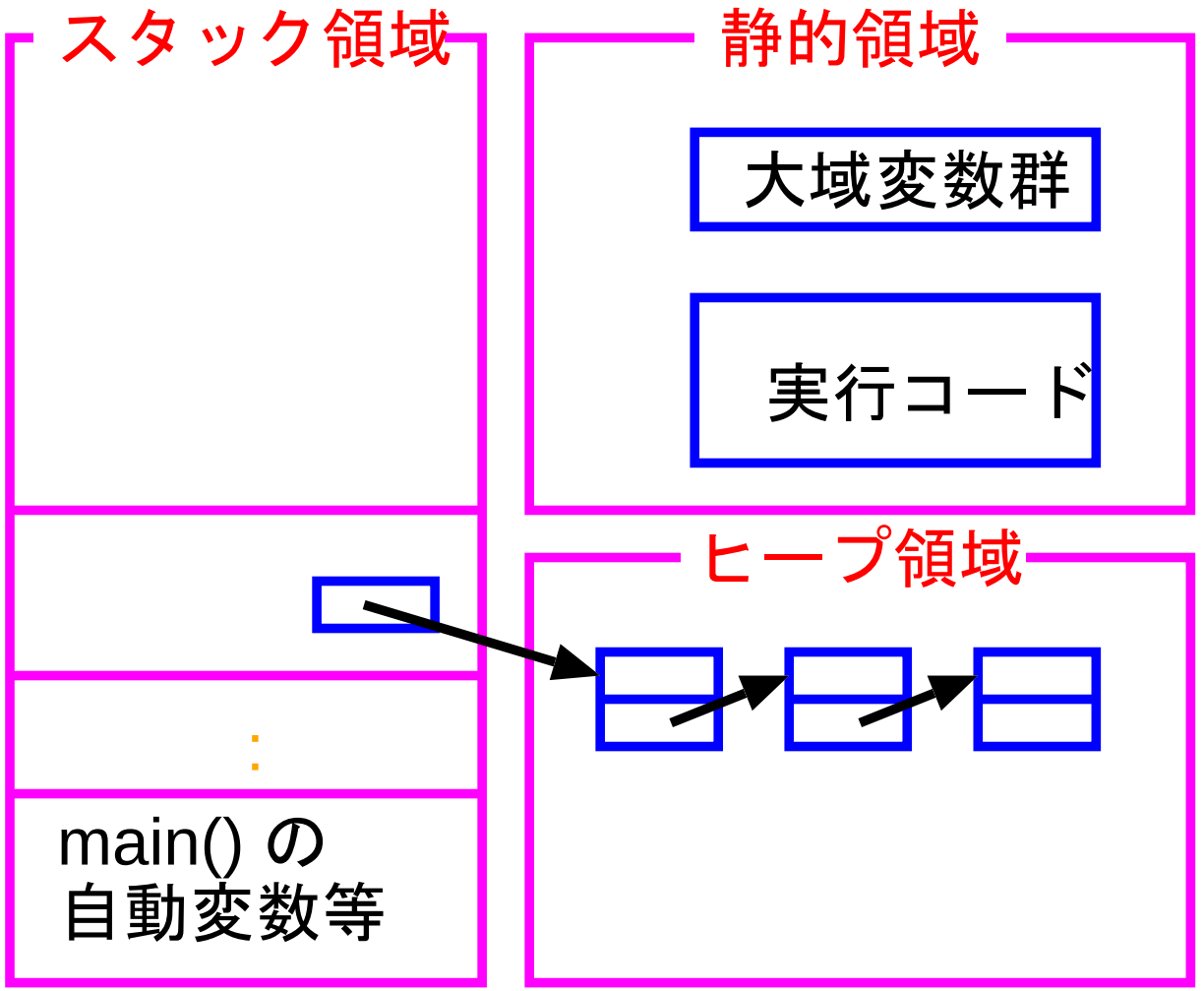
```
    166     169     170     210     217     223     227     234     23
    245     249     277     282     291     309     315     316     32
    330     344     359     361     364     386     408     415     42
    437     443     447     456     460     465     494     503     51
    544     544     545     585     602     613     618     645     65
    660     663     666     679     690     705     712     714     71
    733     734     734     744     745     765     795     798     80
    827     849     853     870     874     880     894     901     90
    918     926     941     946     953     954     975     978     98
---
擬似乱数の初期シード(long値):  333

before sorting:
    386     503     585     138     315     941     443     328     98
    494     602     849     712     734     139     870     765     99
    718     359     663     733     234     282     666     745     36
    874     744     227     650     795     316     249      48     46
    166     364     415      13     926     798     309     953     21
    645     724     853     544     714     613     327       5     82
```

```
    827    217    653    460    894    803    330    169     44
    223    954     37    975    545    277    690    135     45
    170    408    946    437    535     93    918    660     90
    291    705    165    242    910    344    880    679     97
after sorting(sort module that is based on insertion in a link
      5     13     37     48     82     93    135    138     13
    166    169    170    210    217    223    227    234     23
    245    249    277    282    291    309    315    316     32
    330    344    359    361    364    386    408    415     42
    437    443    447    456    460    465    494    503     51
    544    544    545    585    602    613    618    645     65
    660    663    666    679    690    705    712    714     71
    733    734    734    744    745    765    795    798     80
    827    849    853    870    874    880    894    901     90
    918    926    941    946    953    954    975    978     98
[motoki@x205a]$
```

22–10 Javaにおけるメモリ領域の使い方

一般に、プログラム実行時にはメモリ領域は 静的領域、スタック領域、ヒープ領域 の3つに分けて管理される。

- 静的領域 ··· 実行コード、大域変数等を格納。プログラム実行から終了まで内部配置は固定。
- スタック領域 ··· 関数呼び出しの柔軟な実現のために利用。
- ヒープ領域 ··· 動的なデータ記憶領域を確保したい時に利用。



オブジェクト指向プログラムを実行する際のメモリの使い方、

基本的な枠組 $\Longrightarrow$ 上記の通り。
細部 $\Longrightarrow$ 従来と異なる点もある。例えば、

- 例えば、Javaでは、
  クラス情報は必要になった時点でロードされる方式。
  $\Longrightarrow$クラス情報を格納する領域はもは静的ではない。
  $\Longrightarrow$メソッドエリアと呼ばれている。

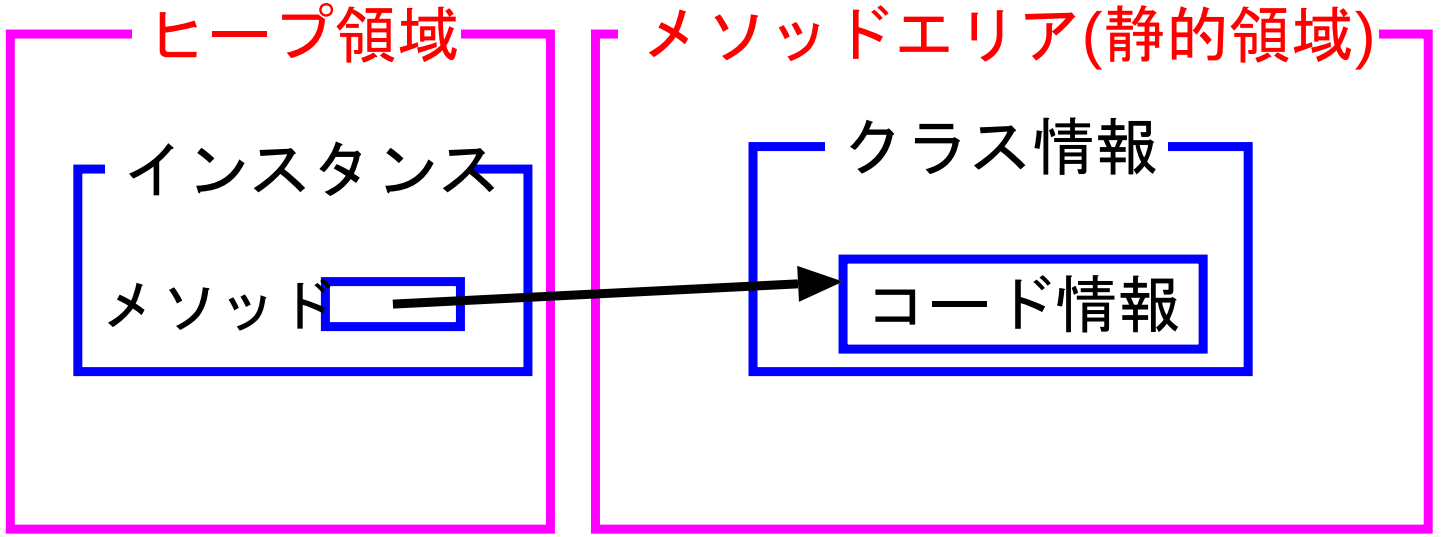
- オブジェクト指向の場合は、
  インスタンスオブジェクトは全て動的に生成される。
  $\Longrightarrow$ヒープ領域は頻繁に利用される。

特にJavaの場合、

インスタンスオブジェクト関連のメモリ領域の使い方は...

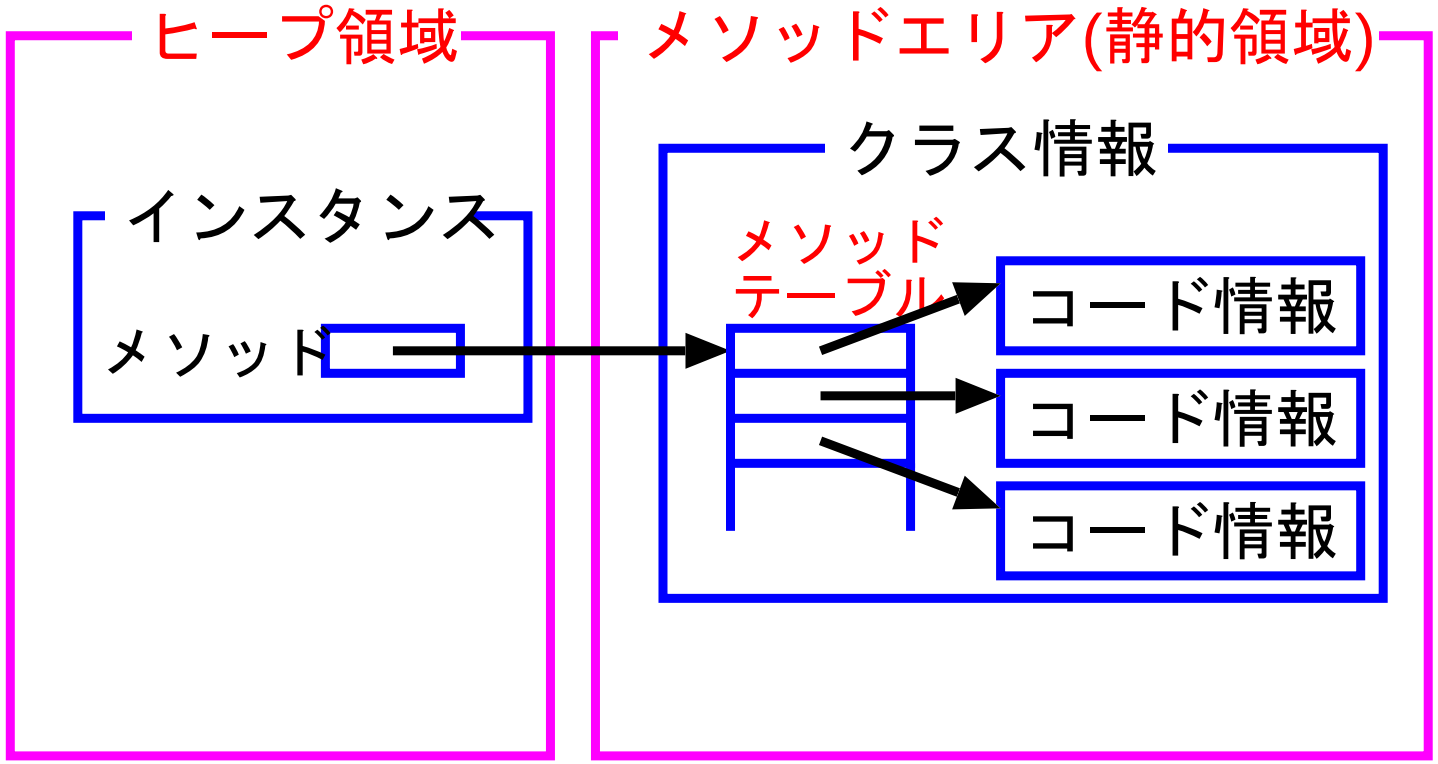
- new演算子が実行されると、
  インスタンスのための領域がヒープ領域内に確保され、
  確保された領域へのポインタがnew演算の結果の式の値となる。

- インスタンスメソッドのコード情報は、
  クラス情報の一部としてメソッドエリア(静的領域)内に配置され、
  そこへの参照情報(ポインタ)が個々のインスタンス領域の中に置かれる。



更に詳しく言うと、

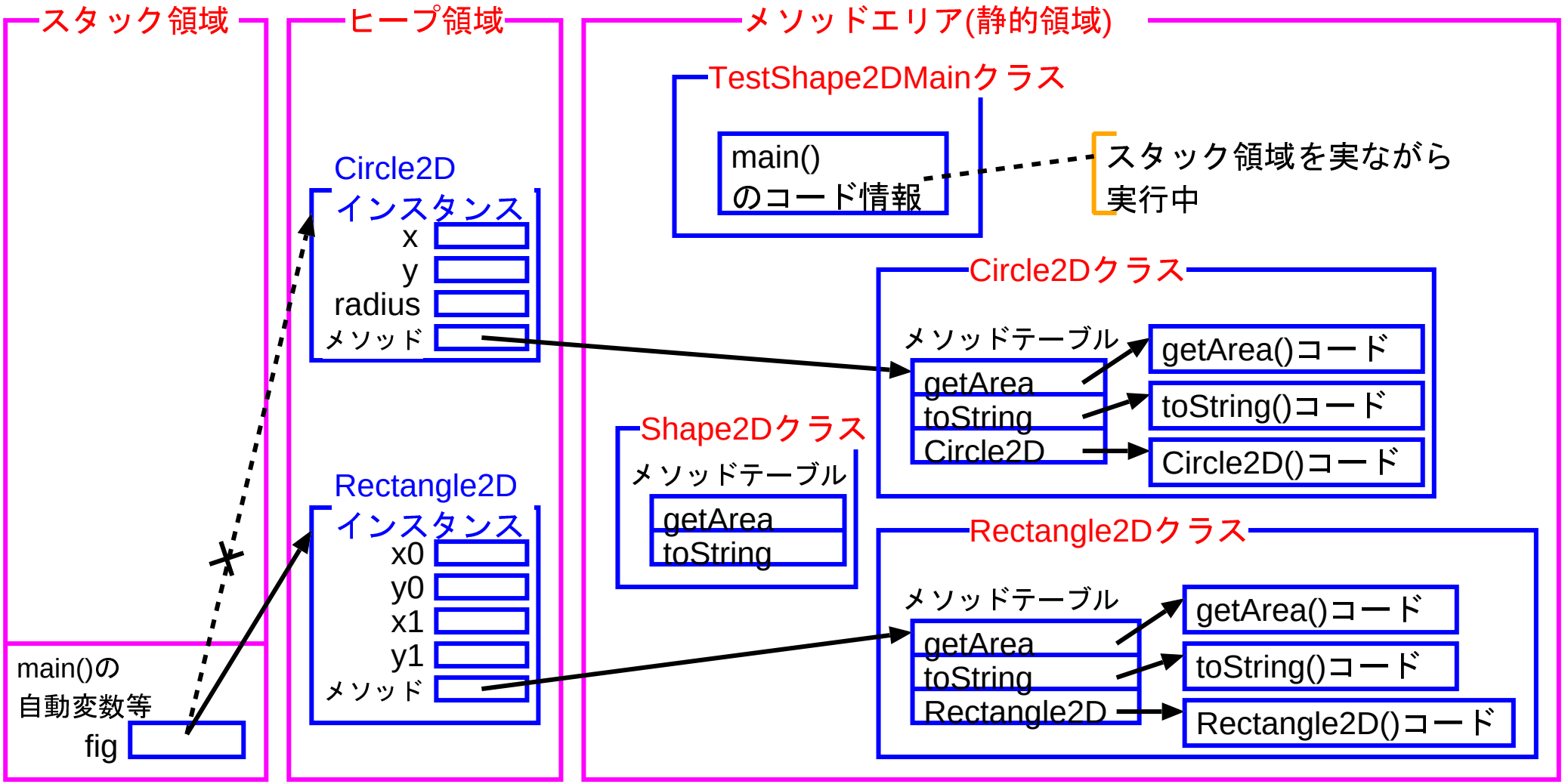
多態性の実装を容易に行うため、
クラス情報の領域内には、メソッドテーブルが配置され、個々のインスタンスの中にはそのメソッドテーブルへの参照情報だけが置かれることもある。



- ヒープ領域内では
  自動的にガベージコレクションが行われる。

例22．13（Java，メソッドの多態性）

例22.11のプログラムの18行目を実行直には、メモリ内部の状態は次の様になっているものと考えられる。



22–11 単一継承とインタフェース

Javaでは、既定義クラスを拡張定義する際、拡張の基になるクラスを1つだけ指定する、という単一継承の原則が採用されている。

その理由：

> 複数のスーパークラス間に共通のメソッド等が存在する場合、
> どのスーパークラスのメソッド等をサブクラスに継承させるか、
> といった問題も発生して面倒なことになりかねない。

単一継承を採用

⇨ Objectクラスを頂点としたクラス階層（木構造）

⇨ 多態変数／多態性の考えを用いることにより、
クラス階層の下で類似したクラスを統合的に扱うことが出来る。

しかし 、Object以外の共通の先祖クラスを持たないクラス同士でも、本質的に同じ役割を果たすメソッドを共通に持っていて、それらのメソッドを通してこれらのクラスを統合的に扱いたい場合もある。

こういった場合 、$\longrightarrow$「インタフェース」を利用することが出来る。

インタフェース

... ソフトウェアモジュール間の相互作用の修正仕方 (i.e. 外部仕様) を規定した「契約」

- 多人数／多チームでの円滑なソフトウェア開発を促進。
- インタフェースはその機能 (メソッドの外部仕様) をもつクラス群の範囲をはっきりと規定する。

⇨ 逆に統合的に扱いたいクラス群があった場合、
それらのもつべき機能 (メソッドの外部仕様) をインタフェースとして定義すれば、

◇ そのインタフェースが元々統合的に扱いたかったクラス群の範囲を定めることになり、更には

◇ 「インタフェース型」の多態変数を用いる事により目的のクラス群を統合的に扱える様になる。

インタフェースの定義、利用 ··· 構文的には抽象クラスと類似

インタフェース定義の形式(通常):

public または「なし」　　既存インタフェースの拡張

```
アクセス修飾子 interface インタフェース名 ..... {
    型名 フィールド名 = 式; public static final を暗黙に仮定
        .....
    型名 フィールド名 = 式;

    戻り値の型 メソッド( 引数列 ); public abstract を暗黙に仮定
                                    → static 不可
        .....
    戻り値の型 メソッド( 引数列 );
}
```

- 定数フィールド, 抽象メソッド以外のメンバーとしては 、
  入れ子クラスや入れ子インタフェースも可。(詳細省略。)

インタフェースを実装したクラスの定義：

- 修飾子 class クラス名 implements インタフェース
  , ..., インタフェース {

  フィールドの宣言、メソッドの定義、など

  }

あるいは、（既定義クラスを拡張する場合は）

修飾子 class クラス名 extends スーパークラス
implements インタフェース, ..., インタフェース {

フィールドの宣言、メソッドの定義、など

}

- 実装を宣言したインタフェース内に明記されている全ての抽象メソッドの実装を行わなければならない。

インタフェース型変数の宣言：

クラスと同様に、個々のインタフェースはデータ型として扱われ、

インタフェース　v;

という変数宣言によって、v は

インタフェース を実装したクラスのインスタンス

(への参照) を保持できる様になる。

⇨ インタフェース型の変数は、

◇ そのインタフェースを実装したインスタンスを統合的に扱うことを可能にし、

◇ そのインタフェース内に記述されているメソッドに関する多態的な振る舞いを可能にする。

例22．14（Comparatorインタフェース） 例題22.4では、Comparator<IdIntPair>というインタフェースを実装した無名クラスを構成し、そのインスタンスを生成した。 $\longrightarrow$ プログラム中の次の部分

```
58                      new  Comparator<IdIntPair> () {
59                          public int compare(IdIntPair d1, Idl
60                              return d1.stringData.compareTo(c
61                          }
62                      }
```

この部分は、次の記述の省略形と見ることもできる。

```
    class Tmp implements  Comparator<IdIntPair>  {
        public int compare(IdIntPair d1, IdIntPair d2) {
            return d1.stringData.compareTo(d2.stringData);
        }
    }
    ..................  new Tmp() ...
---
```

例題22．15（時間計測モジュール，整列化モジュールの動作速度を計測

例題14.3では、時間計測のためのモジュール consumed_time.c をC言語で実装した。これに相当するオブジェクトのクラスをJavaで実装せよ。更に、

例題22.7で考えたスーパー抽象クラス SortModuleForIntArray.java のサブクラスのインスタンス (整列化モジュール) に対して、「int配列内の要素を昇順に並べ替える手順」の動作速度を例題 14.4 の clock-sort-5-10-etc.c に倣って計測する機能、すなわち

> 要素数が 5, 10, 25, 50, 100, 200 の場合に対して
> ①問題例のランダムな設定,
> ②整列化プログラムの実行
> を各々 800000回, 400000回, 160000回, 80000回, 40000回, 20000回 繰り返して1回あたりの計算時間を求め、その結果を出力する、

という機能を備えたモジュールのクラスを定義してみよ。

(考え方)

時間計測モジュールに関して 、⟶ 次を利用可

- ThreadMXBeanインタフェース

  ManagementFactory.getThreadMXBean()··· インタフェースを
  実装しているインスタンス(への参照)を取得

  インスタンス.getCurrentThreadCpuTime() ··· 現在のスレッドが
  それまでに消費したCPU時間(ナノ秒単位, long型)

- RuntimeMXBeanインタフェース

  ManagementFactory.getRuntimeMXBean()修正··· インタフェース
  を実装しているインスタンスを(への参照)を取得

  インスタンス.getUpTime() ··· Java仮想マシンの
  それまでの稼働時間(ミリ秒単位,long型)

- Systemクラス ··· java.langパッケージ内

  System.currentTimeMillis() ··· 1970年1月1日0時0分からの
  経過時間(ミリ秒単位,long型;計測値の粒度はOSによる)

これらのインタフェース／クラスに備わったメソッドを用いて計測開始時点と計測終了時点との時間差を求めるオブジェクトを生成出来る様にすれば良いだけ

ただ、C言語の場合の consumed_time.c と同じ様にしようとすると計測結果を入れる 構造体に相当するものの「クラス」が必要 になる。

⇨ ここでは新たなクラスを定義するのを避け、代わりに、

◇ 計測終了時点を知らせる合図（メソッド呼出し）があると、
①その時点のCPU時間／Java仮想マシン稼働時間／カレンダー時刻を調べ、（インスタンス） 内部に保存してある時間量との差を求めて内部の変数に記録
②先程の計測終了の 合図時点 に調べたCPU時間等の結果を計測開始時点のデータとしてインスタンス 内部の変数に記録 する。そして、

◇ 要求（メソッド呼出し）に応じて、内部の変数に記録された
「 前々回のマーク時点から前回のマーク時点の間の時間 」を返す

整列化モジュールの動作速度を計測するモジュールに関しては 、
例題22.12と同様に考えて、

> SortModuleForIntArray型の変数を引数に持ち、
> 引数で与えられた整列化モジュールに対して
> 所定の方法で動作速度の計測を行うインスタンスメソッド

を備えたモジュールのクラスを定義すれば良い。その際、

◇ 動作速度の計測を行う手順 については、
⟶ C言語で同様の処理を行っている
clock-sort-5-10-etc.c（例題14.4 ） を参考に

◇ 擬似乱数の生成 に関しては、
⟶例題22.12 と同様に
ライブラリ内の java.util.Random というクラスを利用

（プログラミング）　ここで関連するクラスとして、

StopWatch ··· 時間計測モジュールのクラス，

TimerForSortModuleIntArray ··· 整列化モジュールの動作速度を計測するモジュールのクラス

そして、

TimeSortModulesIntArrayMain.java ··· 例題22.7で作成したクラス(HeapsortIntArray, BubblesortIntArray, LListsortIntArray)から生成される3つの整列化モジュール について、TimerForSortModuleIntArrayインスタンス を用いて動作速度を計測

```
[motoki@x205a]$  cat -n StopWatch.java
     1 import java.lang.management.ThreadMXBean;
     2 import java.lang.management.RuntimeMXBean;
     3 import java.lang.management.ManagementFactory;
     4
```

```
 5 /**
 6  * ストップウォッチ風に計算時間を測るためのオブジェクトの...
 7  */
 8 public class StopWatch {
 9     private ThreadMXBean threadMXBean;
10     private RuntimeMXBean runtimeMXBean;
11
12     /* 開始時間を保持*/
13     private long startCpuTime;    //ナノ秒,スレッドのCPU
14     private long startUpTime;     //ミリ秒,Java仮想マシ...
15     private long startRealTime;   //ミリ秒,カレンダー上の.
16
17     /* 最新の区間の時間計測結果を保持*/
18     private double lastIntervalCpuTime;    //秒,スレッド..
19     private double lastIntervalUpTime;     //秒,Java仮想
20     private double lastIntervalRealTime;   //秒,カレンダ.
21
```

```
22      /** 計算時間計測のためのオブジェクトを生成 */
23      public StopWatch() {
24          threadMXBean  =
                    ManagementFactory.getThreadMXBean();
25          runtimeMXBean =
                   ManagementFactory.getRuntimeMXBean();
26
27          startCpuTime  = -1;  //エラー検査のため
28          startUpTime   = -1;
29          startRealTime = -1;
30      }
31
32      /** 計算時間計測オブジェクトの標準的な文字列表現... */
33      @Override
34      public String toString() {
35          return "module for measuring consumed time";
36      }
37
38      /** 時間計測開始 */
```

```
39      public void start() {
40          startCpuTime  = threadMXBean.
                                getCurrentThreadCpuTime();
41          startUpTime   = runtimeMXBean.getUptime();
42          startRealTime = System.currentTimeMillis();
43      }
44
45      /** 前回のマーク時点からの経過時間を計算 */
46      public void calculateLastIntervalTime() {
47          long currentCpuTime  =
                    threadMXBean.getCurrentThreadCpuTime();
48          long currentUpTime   =
                               runtimeMXBean.getUptime();
49          long currentRealTime =
                              System.currentTimeMillis();
50
51          lastIntervalCpuTime  =
                    (currentCpuTime - startCpuTime) * 1e-9;
52          lastIntervalUpTime   =
```

```
                (currentUpTime  - startUpTime)  * 1e-3;
53          lastIntervalRealTime =
                (currentRealTime- startRealTime)* 1e-3;
54
55          startCpuTime  = currentCpuTime;  //次の区間の
56          startUpTime   = currentUpTime;   //時間計測
57          startRealTime = currentRealTime; //のため
58      }
59
60      /** 最新の区間のCPU時間を返す */
61      public double getLastIntervalCpuTime() {
62          if (startCpuTime == -1)
63              System.out.println("(Warning) StopWatch moc
64          return lastIntervalCpuTime;
65      }
66
67      /** 最新の区間にJava仮想マシンが稼働した時間を返す */
68      public double getLastIntervalUpTime() {
69          if (startUpTime == -1)
```

```
70              System.out.println("(Warning) StopWatch moc
71          return lastIntervalUpTime;
72      }
73
74      /** 最新の区間のカレンダー時間を返す */
75      public double getLastIntervalRealTime() {
76          if (startRealTime == -1)
77              System.out.println("(Warning) StopWatch moc
78          return lastIntervalRealTime;
79      }
80  }
```

```
[motoki@x205a]$ cat -n TimerForSortModuleIntArray.java
     1 import java.util.Scanner;
     2 import java.util.Random;
     3
     4 /**
     5  * SortModuleForIntArrayモジュールの提供する
     6  * 「int配列内の要素を昇順に並べ替える機能」の動作速度
     7  * を計測する機能を備えたモジュールを作り出すためのクラス
     8  */
     9 public class TimerForSortModuleIntArray {
```

例題22.12で示したTesterForSortModuleIntArray.java,
例題14.4で示したclock-sort-5-10-etc.c, それから
上で示した「(考え方)」
を参考に自分で考えてみて下さい。(実習レポート課題8)

```
[motoki@x205a]$ cat -n TimeSortModulesIntArrayMain.java
     1 /**
     2  * int配列内の要素を昇順に並べ替える機能を備えた
                                          整列化モジュールとして
     3  *  ・HeapsortIntArrayオブジェクト,
     4  *  ・BubblesortIntArrayオブジェクト,
     5  *  ・LListsortIntArrayオブジェクト
     6  * の3つを考え、これらの動作速度を
     7  *       整列化モジュールの動作速度を計測する機能を備えた
     8  *       TimerForSortModuleIntArrayオブジェクト
     9  * を用いて調べるJavaプログラム
    10  */
    11 public class TimeSortModulesIntArrayMain {
```

例題22.12で示したTestSortModulesIntArrayMain.java
を参考に自分で考えてみて下さい。（実習レポート課題8）

```
[motoki@x205a]$ javac TimeSortModulesIntArrayMain.java
[motoki@x205a]$ java TimeSortModulesIntArrayMain
```

```
Clocking the average execution time of the module
that sorts 5, 10, 25, 50, 100, or 200 elements.
  (*** Heapsort module ***)
Input a random seed (0 - 92233720368547758O7):  333

        ** time for sort **    **time for initialize**
size    cpu_time   real_time    cpu_time   real_time
         (m sec)    (m sec)     (m sec)    (m sec)
----    --------- ---------    --------- ---------
   5      0.00006    0.00005      0.00014    0.00016
  10      0.00028    0.00028      0.00025    0.00024
  25      0.00100    0.00101      0.00056    0.00058
  50      0.00250    0.00243      0.00113    0.00116
 100      0.00550    0.00560      0.00250    0.00230
 200      0.01250    0.01260      0.00450    0.00475
---
Clocking the average execution time of the module
that sorts 5, 10, 25, 50, 100, or 200 elements.
  (*** Bubblesort module ***)
```

```
Input a random seed (0 - 92233720368547758O7):  333

        ** time for sort **    **time for initialize**
size    cpu_time   real_time    cpu_time   real_time
         (m sec)    (m sec)     (m sec)    (m sec)
----    --------  ---------    ---------  ---------
   5      0.00006    0.00004      0.00014    0.00015
  10      0.00023    0.00024      0.00025    0.00024
  25      0.00125    0.00126      0.00056    0.00059
  50      0.00400    0.00406      0.00125    0.00118
 100      0.01400    0.01390      0.00225    0.00235
 200      0.05300    0.05290      0.00450    0.00465
---
Clocking the average execution time of the module
that sorts 5, 10, 25, 50, 100, or 200 elements.
  (*** sort module that is based on insertion in a linked list
Input a random seed (0 - 92233720368547758O7):  333

        ** time for sort **    **time for initialize**
```

size	cpu_time (m sec)	real_time (m sec)	cpu_time (m sec)	real_time (m sec)
5	0.00011	0.00014	0.00011	0.00011
10	0.00028	0.00027	0.00020	0.00022
25	0.00075	0.00083	0.00056	0.00053
50	0.00225	0.00219	0.00100	0.00106
100	0.00675	0.00658	0.00225	0.00228
200	0.02250	0.02280	0.00450	0.00425

```
[motoki@x205a]$
```

例題22．16（インタフェースを用いて整列化モジュールを統合的に扱う

例題22.7では 共通の 抽象スーパークラス を設定して3つの整列化モジュールのクラス HeapsortIntArray, BubblesortIntArray, LListsortIntArray を定義し、

例題22.12 修正では これらを統合的に扱う例題として、この種の整列化モジュールの提供する「int配列内の要素を昇順に並べ替える機能」が正しく動作するかどうかを

①0~ 999の間のランダムな整数を要素とする
大きさ100の配列を生成し、
②それに対して与えられた整列化モジュールを適用して
並べ替え作業を行い、
③その結果を出力する、

という風にテストする機能を備えたモジュールのクラスを定義した。

ここでは、抽象スーパークラスではなくインタフェースを用いて整列化モジュールを統合的に扱える様にして、これらと同等のことを行え。

（考え方）

例題22.7で考えた抽象スーパークラス SortModuleForIntArray はstati
メソッド getInstance() と2つの抽象メソッド toString(), sort()
をメンバーにもつ。

整列化モジュールのクラス定義 に関しては、

⟶ この中の2つの抽象メソッド toString(), sort() を
メンバーにしたインターフェースを定義し、
このインタフェースを実装する形で定義

整列化モジュールの動作テストを行うモジュールのクラス定義

⟶ 本質的には例題22.12 修正で作成した
TesterForSortModuleIntArray.java 中で使われている
抽象スーパークラスの型 を
インターフェースの型 に変更するだけ

（プログラミング）　ここで関連するクラスとして、

ISortModuleForIntArray ··· 整列化モジュールを統合的に扱うための修正インタフェース，

HeapsortIntArray2 ··· heapsortモジュールのクラス，

BubblesortIntArray2 ··· bubblesortモジュールのクラス，

LListsortIntArray2 ··· 連結リストへの挿入に基づく整列化モジュールのクラス，

TesterForSortModuleIntArray2 ··· 動作テストモジュールのクラス

そして、

TestSortModulesIntArrayMain2.java ··· 定義した HeapsortIntArray2, BubblesortIntArray2, LListsortIn から生成される 3つの整列化モジュール について、TesterForSortModuleIntArray2インスタンスを用いて動作テストを行うJavaプログラム

```
[motoki@x205a]$  cat -n ISortModuleForIntArray.java
     1 /**
     2  * int配列内の要素を昇順に並べ替える機能を備えた
     3  * 整列化モジュールの備えるべきインタフェース
     4  */
     5 public interface ISortModuleForIntArray {
     6     /** 整列化モジュールの説明(主に手法)を答える */
     7     String toString();
     8
     9     /** 引数で与えられた配列内の要素を昇順に並べ替える */
    10     void sort(int[] a);
    11 }
[motoki@x205a]$  cat -n BubblesortIntArray2.java
     1 /**
     2  * int配列内の要素をbubblesort手法で昇順に並べ替える
     3  * 機能を備えた整列化モジュールを作り出すためのクラス
     4  */
     5 public class BubblesortIntArray2
                          implements ISortModuleForIntArray {
```

```
 6      //クラス内部でインスタンスを1個だけ生成
 7      //  (コンストラクタはprivate宣言してあるので、  )
 8      //  (生成されるインスタンスはこの1個だけになり、)
 9      //  (これが使い回されることになる。            )
10      private static final BubblesortIntArray2 INSTANCE
                                = new BubblesortIntArray2();
11
12      //コンストラクタ(外部からインスタンス生成不可)
13      private BubblesortIntArray2() {
14          super();
15      }
16
17      /** コンストラクタの代わりに外部に整列化モジュールを...
18      public static BubblesortIntArray2 getInstance() {
19          return INSTANCE;
20      }
21
22      /** 整列化モジュールの説明(主に手法)を答える */
23      @Override
```

```
24      public String toString() {
25          return "Bubblesort module";
26      }
27
28      /** 引数で与えられた配列内の要素をbubblesort手法で... *
29      @Override
30      public void sort(int[] a) {
31          for (int i=0; i<a.length-1; ++i) {
32              for (int j=a.length-1; j>i; --j) {
33                  if (a[j-1] > a[j]) {
34                      int temp = a[j-1];  //a[j-1] と a[j
35                      a[j-1]   = a[j];    //の大小を調べ...
36                      a[j]     = temp;    //逆順なら交換
37                  }
38              }
39          }
40      }
41
```

```
    42      //--------------------単体での動作テスト用------------
    43      public static void main(String[] args) {
    44          int[] a = {9, 8, 6, 7, 5, 3, 1, 2, 4, 0};
    45          getInstance().sort(a);
    46          System.out.println("after sorting (" + getInsta
    47          System.out.print("  a = {");
    48          for (int i=0; i<a.length-1; ++i)
    49              System.out.print(a[i] + ", ");
    50          System.out.println(a[a.length-1] + "}");
    51      }
    52 }
[motoki@x205a]$  cat -n HeapsortIntArray2.java
```

BubblesortIntArray2.javaの場合と同様に、
HeapsortIntArray.javaを少し手直しするだけ。
("extends" ではなく "implements"。)

```
[motoki@x205a]$  cat -n LListsortIntArray2.java
```

BubblesortIntArray2.javaの場合と同様に、
LListsortIntArray.javaを少し手直しするだけ。
("extends" ではなく "implements"。)

```
[motoki@x205a]$  cat -n TesterForSortModuleIntArray2.java
     1 import java.util.Scanner;
     2 import java.util.Random;
     3
     4 /**
     5  * インターフェースISortModuleForIntArrayを実装したモ...
     6  * 「int配列内の要素を昇順に並べ替える機能」が正しく動作...
     7  * をテストする機能を備えたモジュールを作り出すためのクラ...
     8  */
     9 public class TesterForSortModuleIntArray2 {
    10     private static final int SIZE  =100;
    11     private static final int WIDTH = 10;
    12
    13     private Scanner inputScanner;
    14
```

```
15      /** ISortModuleForIntArrayモジュールの動作テストを
16       * 行うモジュールを構成する */
17      public TesterForSortModuleIntArray2()
18          this.inputScanner = new Scanner(System.in);
19
20
21      /** 指定された所から得た擬似乱数シードを用いて
22       * ISortModuleForIntArrayモジュールの動作テストを
23       * 行うモジュールを構成する */
24      public TesterForSortModuleIntArray2(
                                        Scanner inputScanner)
25          this.inputScanner = inputScanner;
26
27
28      /** オブジェクトの説明を答える */
29      @Override
30      public String toString() {
31          return "Tester for module that is to sort int c
32      }
```

```
33
34      /** SIZE個のランダムなデータから成る配列に対して
35       * 引数で与えられた整列化モジュールを実行してみる */
36      public void runOnRandomData(
                        ISortModuleForIntArray sortModule) {
37          int[] a = new int[SIZE];
38
39          //擬似乱数の設定
40          System.out.print("擬似乱数の初期シード(long値): "
41          long seed = inputScanner.nextLong();
42          Random randomGenerator = new Random(seed);
43
44          //配列aの各々の要素に0~ 999の乱数値を設定
45          for (int i=0; i<a.length; ++i)
46              a[i] = randomGenerator.nextInt(1000);
47
48          //整列化前の配列の内容を表示
49          System.out.printf("%nbefore sorting:%n");
50          prettyPrint(a);
```

```
51
52          //整列化
53          sortModule.sort(a);
54
55          //整列化後の配列の内容を表示
56          System.out.println("after sorting(" + sortModul
67          prettyPrint(a);
58      }
59
60      /*-----------------<privateメソッド>-----------------
61       * 引数で与えられた配列の要素を順に全て出力(1行にWIDTH
62      private void prettyPrint(int[] a) {
63          int NumOfEleInLine=0;
64
65          for (int i=0; i<a.length; ++i) {
66              System.out.printf("%7d", a[i]);
67              ++NumOfEleInLine;
68              if (NumOfEleInLine >= WIDTH) {
69                  System.out.println();
```

```
70                      NumOfEleInLine = 0;
71                  }
72              }
73              if (NumOfEleInLine > 0)
74                  System.out.println();
75          }
76 }
```

[motoki@x205a]$ cat -n TestSortModulesIntArrayMain2.java

```
 1 /**
 2  * インターフェースISortModuleForIntArrayを実装し
 3  * int配列内の要素を昇順に並べ替える機能を備えた整列化モ...
 4  *  ・HeapsortIntArray2オブジェクト,
 5  *  ・BubblesortIntArray2オブジェクト,
 6  *  ・LListsortIntArray2オブジェクト
 7  * の3つを考え、これらが正しく整列化動作をするかどうかを
 8  *      整列化モジュールをテストする機能を備えた
 9  *      TesterForSortModuleIntArray2オブジェクト
10  * を用いてテストするJavaプログラム
11  */
```

```
12 public class TestSortModulesIntArrayMain2 {
13     public static void main(String[] args) {
14         TesterForSortModuleIntArray2
15             tester = new TesterForSortModuleIntArray2()
16
17         //HeapsortIntArrayオブジェクトの動作テスト
18         tester.runOnRandomData(HeapsortIntArray2.
                                              getInstance());
19         System.out.println("---");
20
21         //BubblesortIntArrayオブジェクトの動作テスト
22         tester.runOnRandomData(BubblesortIntArray2.
                                              getInstance());
23         System.out.println("---");
24
25         //LListsortIntArrayオブジェクトの動作テスト
26         tester.runOnRandomData(LListsortIntArray2.
                                              getInstance());
27     }
```

```
    28 }
[motoki@x205a]$  javac TestSortModulesIntArrayMain2.java
[motoki@x205a]$  java TestSortModulesIntArrayMain2
```

実行の様子は省略

22–12 ほぼ自習 ジェネリック型

例題19.6や例22.1で定義したStackOfAnyObjectsクラスに見られる様に、 構成要素をObject型に設定 して様々な種類のデータを扱える汎用データ構造を構成することがある。

しかし、この様にすると汎用性と引き換えに次の様な不満点も発生する。

- コンパイル時のエラーチェックが甘くなる。
  (本来とは違う型のデータを汎用データ構造側に渡しても
  コンパイルエラーとならない。)

- 汎用データ構造を利用するプログラムが多少煩雑になる。
  (汎用データ構造を利用するプログラムの中で 常に要素
  データの型を認識し、汎用データ構造からデータを取り
  出す際は適切な型へのキャストを行う必要がある。)

例22．17（Object型を使った汎用スタックの不満点）　例22.1で定義されているStackOfAnyObjectsクラスを利用する単純なjavaプログラム...

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

```
 1 /*
 2  * StackOfAnyObjectsクラスでは、注意深い使用が必要で、
 3  * 誤用してもコンパイル時に見落とされ実行時に初めてエラー
 4  * となることもある、ということを例示
 5  */
 6 public class AbuseOfStackOfAnyObjectsMain {
 7     public static void main(String args[]) {
 8         StackOfAnyObjects stack = new StackOfAnyObjects
 9
10         stack.pushdown("123");                    ←無チェック
11         //  ...(しばらく後に)...      要キャスト
12         Integer someInteger = (Integer) stack.popup();
13         System.out.println("someInteger = "
                                          + someInteger);
14     }
15 }
```

```
[motoki@x205a]$  javac AbuseOfStackOfAnyObjectsMain.java
[motoki@x205a]$  java AbuseOfStackOfAnyObjectsMain
Exception in thread "main" java.lang.ClassCastException:
        java.lang.String cannot be cast to java.lang.Integer
        at AbuseOfStackOfAnyObjectsMain.
                main(AbuseOfStackOfAnyObjectsMain.java:12)
[motoki@x205a]$
```

注目するのは次の2点

- コンパイルは通るが実行時エラーとなる：

- プログラム 12行目 の様に、適宜キャスト演算を施す必要がある：

試しにプログラム 10~ 12行目 を

```
10                  stack.pushdown(new Integer(123));
11                  //  ...(しばらく後に)...
12                  Integer someInteger = stack.popup();
```

という風に変えてコンパイルし直すと 、次の様にエラーが検出

```
[motoki@x205a]$ javac AbuseOfStackOfAnyObjectsMain2.java
AbuseOfStackOfAnyObjectsMain2.java:12: 互換性のない型
検出値   : java.lang.Object
期待値   : java.lang.Integer
        Integer someInteger = stack.popup();
                                    ^
エラー 1 個
```

⇨ 上記の不満点を解消するために、 JDK1.5以降では

ジェネリッククラス... クラスに型パラメータを設け、インスタンス内部で想定する基本要素の型を型パラメータで指定できる

ジェネリック型... ジェネリッククラスから生成されるオブジェクトの型

ジェネリッククラス定義の形式：通常次の様な形式

修飾子 class クラス名 < 型パラメータ名 , ..., 型パラメータ名 > .

{

フィールドの宣言、メソッドの定義、など

}

ここで、

- Java文法上は、各々の 型パラメータ名 に任意の識別子を用いることができる。しかし、慣習上、型パラメータ名 は単一の英大文字、特に...
  - E ··· 要素(element)の型を表す場合
    （「コレクションフレームワーク」でよく使われている。）
  - K ··· キー(key)の型を表す場合
  - N ··· 数(number)の型を表す場合
  - V ··· 値(value)の型を表す場合
  - T ··· 一般的な型(type)を表す場合
  - S,U,V,... ··· 第2,第3,第4,...の型を表す場合

- 各々の 型パラメータ名 を実在する特定の型（型引数という）で置き換えて得られる、

  クラス名 < 型引数 , ..., 型引数 >

  という形のものがジェネリック型。

- 型パラメータに対応付ける型引数を限定する書き方 もある。例えば、Comparableインタフェースを実装した型引数に限定したい場合は

  型パラメータ名 の部分を

  ```
  E extends Comparable<E>
  ```
  や
  ```
  E extends Comparable<? super E>
  ```

- クラス定義の本体部では、具体的な型名を書けるほとんどの場所に型パラメータを書くことができる。(例外もある→次の項)

- 1つのジェネリッククラスの定義によって導入されるクラスは1個だけ
  ⇨ クラス定義の本体部の書き方に次の様な制約
  ◇ staticフィールドの型に型パラメータを使用不可。

  ◇ staticメソッド内やstatic初期化子で型パラメータを使用不可。

  ◇ 型パラメータで指定された型のオブジェクトを直接生成できない。
  例えば、 `new E[size]` という書き方は許されず 、代わりに
  `(E[]) new Object[size]`
  といった書き方をする。

- 型パラメータや型引数の情報は、オブジェクトが正しく使われているかどうかをチェックするためにコンパイラによって使用される。しかし、個々のインスタンスは自分自身の属するジェネリック型の情報を内部に持たない。

例22．18（ジェネリック版スタック）例22.1で示したStackOfAnyObjectクラスをジェネリック化してStackGenericというクラスを定義

```
[motoki@x205a]$  cat -n StackGeneric.java
     1 import java.util.*;
     2
     3 /**
     4  *  ジェネリック版 pushdownスタックのクラス
     5  * @author 元木達也
     6  * @version 0.0
     7  */
     8 public class  StackGeneric<E>  {
     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      /** Eインスタンス(への参照)を格納するための配列領域 */
16      private  E [] 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  StackGeneric () {
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       @SuppressWarnings("unchecked")
33      public  StackGeneric (int initialCapacity) {
```

```
34                  //配列stackにはpushdown()メソッドによってEクラスの
35                  //インスタンスのみが格納されるので、次のキャストは安全。
36          stack =  (E[])  new Object[initialCapacity];
37          indexOfTopEle = -1;
38      }
39
40      /**
41       * スタックオブジェクトの標準的な文字列表現を求める
42       * @return 標準的な文字列表現
43       */
44      @Override
45      public String toString() {
46          return "pushdownStack ( generic_type , capacity
47                  + ", currentNumOfEle="
                            + (indexOfTopEle+1) + ")";
48      }
49
```

```
50      /**
51       * スタックに格納されている要素の情報を得る
52       * @return スタックの内容を表す文字列
53       */
54      public String getDetailedConfig() {
55          String result = "stack contains "
                    + (indexOfTopEle+1) + " elements: {";
56          for (int i=0; i<indexOfTopEle; ++i)
57              result += "\n    " + stack[i] + ",";
58          if (indexOfTopEle >= 0)
59              result += "\n    " + stack[indexOfTopEle];
60          result += " }";
61          return result;
62      }
63
64      /**
65       * スタックが空かどうかを調べる
66       * @return スタックが空かどうか
67       */
```

```
68      public boolean isEmpty() {
69          return indexOfTopEle == -1;
70      }
71
72      /**
73       * 新しい E 要素をスタックにpush-downする
74       * @param element スタックにpush-downする新要素
75       */
76      public void pushdown( E  element) {
77          if (indexOfTopEle+1 == stack.length) {
78              stack = Arrays.copyOf(stack, stack.length
79                                    + DEFAULT_CAPACITY_INCREMENT);
80              System.out.printf("###Stack capacity is inc
81                                        "#<New> %s%n", this);
82          }
83          stack[++indexOfTopEle] = element;
84      }
85
86      /**
```

```
 87      * スタックから最も上部の要素を取り出す
 88      * @return スタックの最も上部の要素
 89      */
 90     public  E  popup() {
 91         if (indexOfTopEle < 0)
 92             throw new EmptyStackException();
 93          E  element = stack[indexOfTopEle];
 94         stack[indexOfTopEle--] = null;
 79                                  //取り出した要素への参照を解除
 95         return element;
 96     }
 97
 98     /**
 99      * スタックに格納された要素の個数を調べる
100      * @return スタックに格納された要素の個数
101      */
102     public int getNumOfEle() {
103         return indexOfTopEle+1;
104     }
```

```
105
106     /**
107      * スタックのtop要素をのぞき見
108      * @return スタックのtop要素(への参照)
109      */
110     public  E  peepTop() {
111         if (isEmpty())
112             return null;
113         else
114             return stack[indexOfTopEle];
115     }
116
117     /**
118      * スタックの指定要素をのぞき見
119      * @param index のぞき見したいスタック要素の番号
120      * @return 番号indexのスタック要素(への参照)
121      */
122     public  E  peepEleOfIndex(int index) {
123         return stack[index];
```

```
   124     }
   125 }
[motoki@x205a]$  cat -n TestStackGenericMain.java
     1 /**
     2  * StackGenericクラスの動作を確認するためのJavaプログ...
     3  */
     4 public class TestStackGenericMain {
     5     public static void main(String args[]) {
     6         StackGeneric<String> stack1 =
                                    new StackGeneric<String>();
     7         stack1.pushdown("a");
     8         stack1.pushdown("bcd");
     9         stack1.pushdown("efg");
    10         stack1.pushdown("hij");
    11         System.out.println(stack1.popup() + ", " +
    12                            stack1.popup() + ", " +
    13                            stack1.popup());
    14
    15         StackGeneric<Integer> stack2 =
```

```
                                        new StackGeneric<Integer>(2);
    16          stack2.pushdown(new Integer(1));
    17          stack2.pushdown(new Integer(2));
    18          stack2.pushdown(new Integer(3));
    19          System.out.println(stack2.popup() + ", " +
    20                             stack2.popup() + ", " +
    21                             stack2.popup());
    22      }
    23 }
[motoki@x205a]$  javac TestStackGenericMain.java
[motoki@x205a]$  java TestStackGenericMain
hij, efg, bcd
###Stack capacity is increased###
#<New> pushdownStack (generic_type, capacity=102, currentNumOf
3, 2, 1
[motoki@x205a]$
```

ここで、例22.1で示したStackOfAnyObjects.javaから変更した箇所を下線で表している。

- **32行目 ··· “SuppressWarnings("unchecked")” というアノテーションを挿入。これによって次のコンストラクタをコンパイル時に「unchecked」という種類の警告が出るのを抑制する。**

補足： この行をコメントアウトしてコンパイルすると、次の様に警告が出る。

```
|[motokix205a]$  javac StackGeneric.java
|注:StackGeneric.java の操作は、未チェックまたは安全ではありません。
|注:詳細については、-Xlint:unchecked オプションを指定して
                                        再コンパイルしてください。
```

更に、この警告文に従って再コンパイルすると、...

```
|[motokix205a]$  javac -Xlint:unchecked StackGeneric.java
|StackGeneric.java:36: 警告:[unchecked] 無検査キャストです
|検出値 : java.lang.Object[]
|期待値 : E[]
|        stack = (E[]) new Object[initialCapacity];
|                      ^
|警告 1 個
```

22–13 ほぼ自習 jarファイル，クラスパスの指

C言語では
.oファイル群を1つのライブラリファイル(.aファイル)に纏めることができた。⟶Javaでも同様のことが可能

jarファイル： Javaでは、
複数のクラスファイルやリソース(e.g.gifファイル)を単一のアーカイブファイル(jarファイルという)に纏めておくことができる。

- jarファイルの拡張子は .jar 。
- jarファイルはクラスファイル群の公開・配付に利用されている。実際、単に「ライブラリ」と言えばjarファイルのことを指す。
- jarファイルはZIP形式で内容を保持する。
- jarファイルを作成したり操作したりするために、JDKの中にjarコマンドが用意されている。(コマンドの書式はtarコマンドに類似。)

jarファイルの作成： jarコマンドを次の様な形式で使う。

jar cvf [jarファイル名] [jarファイルに含めたいファイルのリスト]

ここで、

- jarコマンドのキーの意味は次の通り。
  - c ··· create(作成)。
  - v ··· verbose(詳細報告)。省略可。
  - f ··· file(標準出力でなくファイルに出力を送る)。

- [jarファイルに含めたいファイルのリスト] は
  jarファイルに含めたいファイルの名前を
  空白で区切って並べた文字列
  を表す。この中で、
  - * ··· 全ファイルの意味
  - ディレクトリ ··· ディレクトリ以下の内容が再帰的に追加

---

jarファイルの内容表示： jarコマンドを次の様な形式で使う。

```
jar tvf [jarファイル名]
```

ここで、

- jarコマンドのキーの意味は次の通り。
  - t ··· table(一覧表)。
  - v ··· verbose(詳細報告)。省略可。
  - f ··· file(内容表示するjarファイルがコマンドラインで指定されていることを表す)。

- [jarファイル名] には内容表示したいjarファイルの(パスと)名前を指定する。

---

jarファイルの内容抽出： jarコマンドを次の様な形式で使う。

jar xvf [jarファイル名] [jarファイルから抽出したいファイルのリスト, 省略可]

ここで、

- jarコマンドのキーの意味は次の通り。
  - x ··· extract(抽出)。
  - v ··· verbose(詳細報告)。省略可。
  - f ··· file(抽出元のjarファイルがコマンドラインで指定されていることを表す)。

- [jarファイル名] には抽出元のjarファイルの(パスと)名前を指定する。

- [jarファイルから抽出したいファイルのリスト, 省略可] は
  jarファイルから抽出したいファイルの名前を
  空白で区切って並べた文字列
  を表す。
  省略した場合 ⟶jarファイル中の全ファイル

例22．19（自分専用のライブラリを用意）

例題22.7 で定義した HeapsortIntArrayクラス,
BubblesortIntArrayクラス,
LListsortIntArrayクラス,
例題22.15 で定義した StopWatchクラス,
例22.18で定義した StackGenericクラス
は汎用性があり、将来再利用する可能性も大いにある。

⇨ ◇ これらに関連するクラスの.classファイルを自分専用のライブラリとして ../mylib というディレクトリの中にまとめて置いておく。

◇ 整列化関連の.classファイルは全てsortIntArray.jar という名前のjarファイルにまとめて保管する。

```
[motoki@x205a]$  ls *.class
ls: *.class にアクセスできません: そのようなファイルやディレクトリはありません
[motoki@x205a]$  javac StopWatch.java
[motoki@x205a]$  javac *sortIntArray.java
[motoki@x205a]$  javac StackGeneric.java
[motoki@x205a]$  ls *.class
BubblesortIntArray.class  LinkedListOfInt$Node.class   StackGe
HeapsortIntArray.class    LinkedListOfInt.class        StopWat
LListsortIntArray.class   SortModuleForIntArray.class
[motoki@x205a]$  jar cvf sortIntArray.jar
                                        BubblesortIntArray.class \
    HeapsortIntArray.class LListsortIntArray.class
                                        LinkedListOfInt\$Node.class \
    LinkedListOfInt.class SortModuleForIntArray.class
マニフェストが追加されました。
BubblesortIntArray.class を追加中です。(入 = 1441) (出 = 840)(41%
```

縮されました)
HeapsortIntArray.class を追加中です。(入 = 1612) (出 = 958)(40%
縮されました)
LListsortIntArray.class を追加中です。(入 = 1511) (出 = 891)(41%
縮されました)
LinkedListOfInt$Node.class を追加中です。(入 = 1049) (出 = 612)(4
縮されました)
LinkedListOfInt.class を追加中です。(入 = 851) (出 = 540)(36% 収
縮されました)
SortModuleForIntArray.class を追加中です。(入 = 363) (出 = 252)(3
縮されました)
[motoki@x205a]$ jar tvf sortIntArray.jar
     0 Sat Jan 21 18:20:36 JST 2012 META-INF/
    71 Sat Jan 21 18:20:36 JST 2012 META-INF/MANIFEST.MF
  1441 Sat Jan 21 17:25:18 JST 2012 BubblesortIntArray.class
  1612 Sat Jan 21 17:25:18 JST 2012 HeapsortIntArray.class
  1511 Sat Jan 21 17:25:18 JST 2012 LListsortIntArray.class

```
  1049 Sat Jan 21 17:25:18 JST 2012 LinkedListOfInt$Node.class
   851 Sat Jan 21 17:25:18 JST 2012 LinkedListOfInt.class
   363 Sat Jan 21 17:25:18 JST 2012 SortModuleForIntArray.clas
[motoki@x205a]$  mkdir ../mylib
[motoki@x205a]$  mv sortIntArray.jar ../mylib
[motoki@x205a]$  cp StackGeneric.class ../mylib
[motoki@x205a]$  cp StopWatch.class ../mylib
[motoki@x205a]$  ls ../mylib
StackGeneric.class  StopWatch.class  sortIntArray.jar
[motoki@x205a]$
```

クラスパスの指定： 標準パッケージやimport宣言されたパッケージの他に別クラスへの参照がある場合、 コンパイルや実行の際に 、
デフォルトでは、⟶カレントディレクトリ内から
必要なクラス定義や.classファイルが探される。

カレントディレクトリ以外の場所を探してもらいたい場合は、
⟶次のいずれかの方法 （両方指定⟶(方法1)が優先）

(方法1) コマンドのオプション指定 ：
javacコマンドやjavaコマンドを実行する際に
-classpath クラスパスの指定
または
-cp クラスパスの指定
という形のオプションを指定する。ここで、クラスパスの指定 の部分は探してもらいたいディレクトリやjarファイルのパス指定をコロン(:)で区切って並べた文字列で表す。（途中の空白は許されない。）

(方法2) 環境変数CLASSPATHの設定 ：

（方法2）環境変数CLASSPATHの設定 ：

bash を使っている場合はコマンドライン上で例えば

CLASSPATH= クラスパスの指定; export CLASSPATH

とし、 tcsh を使っている場合はコマンドライン上で

setenv CLASSPATH クラスパスの指定

とする。ここでも、クラスパスの指定 の部分はパスの明示されたディレクトリやjarファイルをコロン(:)で区切って並べた文字列である。

例22．20（クラスパスを指定してライブラリ内のStackGenericを利用）

先の例22.19で自分専用のライブラリの要素として登録した
../../mylib/StackGeneric.classを利用して、
例22.18で定義したTestStackGenericMain.java をコンパイル・実行

```
[motoki@x205a]$  ls
TestStackGenericMain.java          TimerForSortModuleIntArray.j
TimeSortModulesIntArrayMain.java
[motoki@x205a]$  ls ../../mylib
StackGeneric.class  StopWatch.class  sortIntArray.jar
[motoki@x205a]$  javac -cp ../../mylib TestStackGenericMain.ja

[motoki@x205a]$  java -cp .:../../mylib TestStackGenericMain
hij, efg, bcd
###Stack capacity is increased###
#<New> pushdownStack (generic_type, capacity=102, currentNumOf
3, 2, 1
[motoki@x205a]$
```

ここで、

- 最後の java コマンド で、
  クラスパスにカレントディレクトリ (.) を追加しないと、...

```
[motoki@x205a]$ java -cp ../../mylib TestStackGenericMain
Exception in thread "main" java.lang.NoClassDefFoundError: T
Caused by: java.lang.ClassNotFoundException: TestStackGeneri
at java.net.URLClassLoader$1.run(URLClassLoader.java:217)
at java.security.AccessController.doPrivileged(Native Method
at java.net.URLClassLoader.findClass(URLClassLoader.java:205
at java.lang.ClassLoader.loadClass(ClassLoader.java:321)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:
at java.lang.ClassLoader.loadClass(ClassLoader.java:266)
Could not find the main class: TestStackGenericMain. Program
[motoki@x205a]$
```

例22．21（クラスパスを指定してライブラリ内の各種ファイルを利用）

先の例22.19で構成した自分専用のライブラリ
(../../mylib内の.classファイルとjarファイル)を利用して、例題22.15 で定義した TimeSortModulesIntArrayMain.java (とそれに関連しているjavaソースファイル)をコンパイル・実行

```
[motoki@x205a]$  ls
TestStackGenericMain.class  TimeSortModulesIntArrayMain.java
TestStackGenericMain.java   TimerForSortModuleIntArray.java
[motoki@x205a]$  ls ../../mylib
StackGeneric.class  StopWatch.class  sortIntArray.jar
[motoki@x205a]$
     javac -cp .:../../mylib:../../mylib/sortIntArray.jar \
                              TimeSortModulesIntArrayMain.java
[motoki@x205a]$  ls
TestStackGenericMain.class          TimerForSortModuleIntArray$
```

```
TestStackGenericMain.java          TimerForSortModuleIntArray.
TimeSortModulesIntArrayMain.class  TimerForSortModuleIntArray.
TimeSortModulesIntArrayMain.java
[motoki@x205a]$
      java -cp .:../../mylib:../../mylib/sortIntArray.jar \
                                    TimeSortModulesIntArrayMain
Clocking the average execution time of the module
that sorts 5, 10, 25, 50, 100, or 200 elements.
  (*** Heapsort module ***)
Input a random seed (0 - 92233720368547758O7):  333

        ** time for sort **    **time for initialize**
size    cpu_time   real_time    cpu_time   real_time
         (m sec)    (m sec)     (m sec)    (m sec)
----    ---------  ---------   ---------  ---------
   5      0.00008    0.00006     0.00014    0.00015
  10      0.00023    0.00026     0.00025    0.00024
```

```
  25        0.00100     0.00098        0.00056     0.00058
  50        0.00250     0.00244        0.00113     0.00115
 100        0.00550     0.00573        0.00250     0.00233
 200        0.01300     0.01295        0.00450     0.00470
            (途中省略)
 200        0.02300     0.02265        0.00400     0.00425
[motoki@x205a]$
```

22–14 ほぼ自習 パッケージ管理

- プログラムが大規模になり
- 関連する.classファイルやjarファイルが
  あちこちのディレクトリに分散配置される様になると、...

⇨ 大量のソースコードの管理も難しくなる。

- ◇ クラスパスの指定も大変。
- ◇ 目的のクラスを探し出す手間が増える。
- ◇ クラス名の衝突の可能性も出てくる。

⇨ 「パッケージ」と呼ばれる仕組み

- ◇ クラス間のアクセス制御を行うためでもある

パッケージ管理の機構：

機能面，目的面で互いに関連したクラスやインタフェースを1つのグループ（パッケージ）にまとめて管理することができる。

- 階層構造を持つことができる。
  但し、名前を区切るための記号としてピリオド（.）

- ソースプログラムの先頭にpackage文
  ....定義したクラス（やインタフェース）の所属パッケージを指定
  package文のないプログラムの場合 $\longrightarrow$ 無名パッケージに所属

  補足（無名パッケージの用途）：
  小規模，一時的なアプリケーション、開発の初期段階

- コンパイルや実行の際には 、 ...

パッケージ管理の機構：

機能面，目的面で互いに関連したクラスやインタフェースを1つのグループ（パッケージ）にまとめて管理することができる。

- 階層構造を持つことができる。
  但し、名前を区切るための記号としてピリオド（.）
- ソースプログラムの先頭にpackage文
  ....定義したクラス（やインタフェース）の所属パッケージを指定
- コンパイルや実行の際には 、
  - ◇ パッケージ名中のピリオド（.）で区切られた名前の各々はディレクトリ名として解釈される。
  - ◇ 環境変数CLASSPATHで指定されたクラスパスを起点に、パッケージ名の表すディレクトリ階層をたどって必要な.classファイルが探されることになる。

  ⇨ パッケージ名の階層構造に合致した形で、実際のファイルシステム上に.classファイル群を階層的に構成しておく必要

標準パッケージ：

多くの標準パッケージが備わっている。
（全てjavaパッケージのサブパッケージ） 例えば、

- java.lang ··· Object, String, Math, Thread, Class 等の基本的なクラスを含むコアパッケージで、この中で定義されたクラスは（import宣言無しでも）自由に使える。
- java.io ··· 入出力, ファイル操作に関連したクラスから成る。
- java.util ··· 一般的なユーティリティのためのクラスから成る。
- java.awt ··· GUIを記述するためのクラスから成る。
  （Abstract Window Toolkit）
- java.applet ··· アプレットを記述するためのクラスから成る。
- java.beans ··· JavaBeansコンポーネントアーキテクチャにおける、独立したソフトウェアコンポーネントを記述するためのクラスから成る。

- java.lang.instrument ··· 仮想マシン上で動作しているアプリケーションを計測できるエージェント定義するためのクラスから成る。
- java.lang.management ··· 仮想マシンとその上のOSを監視・管理するためのクラスから成る。
- java.math ··· 任意精度の算術計算等のためのクラスから成る。
- java.net ··· ソケットやURL等のネットワークの基盤を扱うためのクラスから成る。
- java.nio ··· 通常より複雑だが高性能な入出力(New I/O)のためのクラスから成る。
- java.nio.charset ··· 文字セットとそのエンコーディングを定義しているクラスから成る。
- java.rmi ··· Remote Method Invocation。他ホスト上の他仮想マシンからのメソッド呼び出しも可能なオブジェクトを生成す

るためのクラスから成る。

- java.security ··· セキュリティに関連したクラスから成る。
- java.sql ··· 関係データベースを使用するためのJDBC(Java Database Connectivity)パッケージ。
- java.text ··· 数字や日付の書式・解析、文字列のソート、キーによるメッセージ検索等のためのクラスから成る。
- java.util.concurrent ··· 効率的なマルチスレッドのアプリケーションを書くためのユーティリティから成る。
- java.util.jar ··· jarファイルを読み書きするためのクラスから成る。
- java.util.logging ··· コード内からロギングするためのフレームワークを提供する。
- java.util.prefs ··· アプリケーションの実行状況やユーザとシステムの設定を管理するための仕組みを提供する。

- java.util.zip ··· ZIPファイルを読み書きするためのクラスから成る。
- java.util.regex ··· 正規表現を扱うためのクラスから成る。

更に、標準拡張と呼ばれている次の様なパッケージもある。

- javax.accessibility ··· 障害者が使用可能なGUIを開発するためのフレームワークを提供する。
- javax.naming ··· ディレクトリとネーミングサービスを扱うためのクラス, サブパッケージから成る。
- javax.sound ··· ディジタルサウンドを扱うためのサブパッケージから成る。
- javax.swing ··· GUIコンポーネントを扱うためのクラスから成る。(awtはネイティブのGUIに依存しているが、こちらは全てのシステム上で出来るだけ同じ様に見え振舞う様に書かれている。)

所属パッケージの指定：

クラスやインタフェースの所属するパッケージを指定したい場合は、ソースプログラムの先頭に次の形式のpackage文を書く。

```
package パッケージ名 ;
```

ここで、

- パッケージ名 の付け方 に関しては、次の様な指針が一般的
  - ◇ ピリオド(.)で区切られた名前の部分には英小文字だけを使う。
  - ◇ 会社や組織の場合、そのインターネットドメイン名の構成要素を逆順に並べた文字列(e.g. jp.ac.niigata_u.ie.ce.)でパッケージ名を開始する。その後に、地域やプロジェクトの名前も挿入して、名前の衝突を未然に防ぐ。

> 補足：
> 世界中の様々な人達の作ったプログラムが公開される可能性
> $\Longrightarrow$ 面倒な手間無しでこれらを安全に利用したい

所属パッケージの指定：

クラスやインタフェースの所属するパッケージを指定したい場合は、
ソースプログラムの先頭に次の形式のpackage文を書く。

package　パッケージ名　;

ここで、

- パッケージ名 の付け方 に関しては、次の様な指針が一般的

. . . . . .

- 所属パッケージの指定されたクラスについては、

  パッケージ名 . クラス名

  という名前（完全限定名という）で一意にクラスを特定できる

⇨ 環境変数 CLASSPATHが適切に設定されている場合 、

◇ 完全限定名を使えば、別パッケージからでもクラスを参照可

◇ どのディレクトリにいても、

java　クラスの完全限定名

というコマンド入力で指定クラスのmainメソッドを起動可

例22．22（パッケージへの登録）

例題22.7 で定義した HeapsortIntArrayクラス,
Bubble sortIntArrayクラス,
LListsortIntArrayクラス,
例題22.15 で定義した StopWatchクラス,
例22.18で定義した StackGenericクラス
は汎用性があり、将来再利用する可能性も大いにある。

⇨ ◇ 関連するクラスを mypackage というパッケージに登録して、
◇ 色々な場所からこのパッケージの中のクラスを利用できる
様にしたい。

⇨ 次のことを行えば良い。

. . . . . .

例22.22(パッケージへの登録)

例題22.7 で定義した HeapsortIntArrayクラス,

.....

例22.18で定義した StackGenericクラス

は汎用性があり、将来再利用する可能性も大いにある。

⇨ .....

⇨ 次のことを行えば良い。

(1) 登録したいクラスを定義したソースプログラム群をディレクトリ

~/C-Java2012/Programs-Java/mypackage

の中にコピー

CLASSPATHで~/C-Java2012/Programs-Java が指定されていると仮定

(2) 各々のソースプログラムの先頭に次の行を挿入。

```
package mypackage;
```

(3) コマンドライン上で

```
CLASSPATH=~/C-Java2012/Programs-Java; export CLASSPATH
```

(4) コンパイル。

補足： .classファイルが出来た後は、ソースファイルは別の場所で管理しても良い。

実際に以上の (1),(2) を行った状況下で 、
簡単な確認作業と (3),(4) の作業を行っている様子

```
[motoki@x205a]$  pwd
/home/motoki/C-Java2012/Programs-Java/mypackage
[motoki@x205a]$  ls
BubblesortIntArray.java  LinkedListOfInt.java          StopWatch
HeapsortIntArray.java    SortModuleForIntArray.java
LListsortIntArray.java   StackGeneric.java
[motoki@x205a]$  cat BubblesortIntArray.java
package mypackage;
```

以下、例題 22.7 の BubblesortIntArray.java と同じ

```
[motoki@x205a]$  cat HeapsortIntArray.java
package mypackage;
```

以下、例題 22.7 の HeapsortIntArray.java と同じ

```
[motoki@x205a]$  cat LListsortIntArray.java
```

```
package mypackage;
```

以下、例題22.7 の LListsortIntArray.java と同じ

```
[motoki@x205a]$  cat LinkedListOfInt.java
package mypackage;
```

以下、例題22.7 の LinkedListOfInt.java と同じ

```
[motoki@x205a]$  cat SortModuleForIntArray.java
package mypackage;
```

以下、例題22.7 の SortModuleForIntArray.java と同じ

```
[motoki@x205a]$  cat StackGeneric.java
package mypackage;
```

以下、例題22.18 の StackGeneric.java と同じ

```
[motoki@x205a]$  cat StopWatch.java
package mypackage;
```

以下、例題22.15 の StopWatch.java と同じ

```
[motoki@x205a]$
     CLASSPATH=~/C-Java2012/Programs-Java; export CLASSPATH
[motoki@x205a]$  javac *.java
[motoki@x205a]$  ls
BubblesortIntArray.class     LinkedListOfInt.java
BubblesortIntArray.java      SortModuleForIntArray.class
HeapsortIntArray.class       SortModuleForIntArray.java
HeapsortIntArray.java        StackGeneric.class
LListsortIntArray.class      StackGeneric.java
LListsortIntArray.java       StopWatch.class
LinkedListOfInt$Node.class   StopWatch.java
LinkedListOfInt.class
[motoki@x205a]$
```

- 環境変数 CLASSPATH が適切に設定されていないと、...

```
[motoki@x205a]$  javac BubblesortIntArray.java
BubblesortIntArray.java:7: シンボルを見つけられません。
シンボル: クラス SortModuleForIntArray
public class BubblesortIntArray extends SortModuleForIntArra
                                        ^
BubblesortIntArray.java:25: メソッドはスーパータイプのメソッド
をオーバーライドまたは実装しません
    @Override
    ^
BubblesortIntArray.java:31: メソッドはスーパータイプのメソッド
をオーバーライドまたは実装しません
    @Override
    ^
エラー 3 個
[motoki@x205a]$
```

別パッケージ内のクラスの利用：

ソースプログラムの最初の方に次の形式のimport文を置いておくと、クラスの完全限定名の代わりに単純なクラス名でクラスを参照できる様になる。

import パッケージ名 . クラス名 ; または import パッケージ名

ここで、

- 状況に応じて使い分けるのが良い。
  - ◇ クラス名も示す 書き方は、どのクラスを使うのかが明確になる。
  - ◇ ワイルドカードを使う 書き方は、パッケージ内の多数のクラスを利用したい時に簡潔にimport宣言できる。

- 同一パッケージ内のクラスを利用する場合 はimport宣言は不要である。
  （java.lang標準パッケージと同様に自動的にimportされる。）
  しかし、コンパイル前に環境変数CLASSPATHの設定を忘れずに行っておく必要がある。

例22．23（パッケージ内のクラスの利用）

プログラムの中から先の例22.22で構成したmypackageパッケージ内のStackGeneric というクラスを利用できる様にするためには、
プログラムの前の方に

```
import mypackage.StackGeneric;
```

という行を挿入すれば良い。
これを例22.18で定義したTestStackGenericMain.java に対して行なった状況下で 、
簡単な確認作業とその改変版をコンパイル・実行している様子

```
[motoki@x205a]$  pwd
/home/motoki/C-Java2012/Programs-Java/objectoriented/example_p
[motoki@x205a]$  ls
TestStackGenericMain.java          TimerForSortModuleIntArray.j
TimeSortModulesIntArrayMain.java
```

```
[motoki@x205a]$  cat TestStackGenericMain.java
import mypackage.StackGeneric;
```

以下、例題22.18 の TestStackGenericMain.java と同じ

```
[motoki@x205a]$
   CLASSPATH=~/C-Java2012/Programs-Java:.; export CLASSPATH
[motoki@x205a]$  javac TestStackGenericMain.java
[motoki@x205a]$  ls
TestStackGenericMain.class  TimeSortModulesIntArrayMain.java
TestStackGenericMain.java   TimerForSortModuleIntArray.java
[motoki@x205a]$  java TestStackGenericMain
hij, efg, bcd
###Stack capacity is increased###
#<New> pushdownStack (generic_type, capacity=102, currentNumOf
3, 2, 1
[motoki@x205a]$
```

22–15 ほぼ自習 アクセス制御とカプセル化

情報隠蔽・カプセル化を進めるためには適切なアクセス制御が必要

⇨ アクセス修飾子の働きを次にまとめておく。

クラス、インタフェースに対するアクセス制御：

クラス定義等に付ける アクセス修飾子	効果
private	（指定不可）
（なし）	同一パッケージからのみ利用可
protected	（指定不可）
public	他パッケージからも利用可

クラスのメンバー(フィールド, メソッド, コンストラクタ, 入れ子クラス, 等)に対するアクセス制御:

<table>
<tr><td rowspan="2">メンバーに付ける
アクセス修飾子</td><td colspan="2">効果</td></tr>
<tr><td>クラスがpublicの場合</td><td>クラスが
publicでない場合</td></tr>
<tr><td>private</td><td colspan="2">クラス内からのみアクセス可</td></tr>
<tr><td>(なし)</td><td colspan="2" rowspan="3">同一パッケージからのみアクセス可</td></tr>
<tr><td>protected</td><td>同一パッケージ、
及びサブクラス
からのみアクセス可</td></tr>
<tr><td>public</td><td>他パッケージからもアクセス可</td></tr>
</table>

インタフェースのメンバー(フィールド, メソッド, 等)に対するアクセス制御:

メンバーに付けるアクセス修飾子	効果	
	インタフェースがpublicの場合	インタフェースがpublicでない場合
private	(指定不可)	
(なし)	(public宣言されたものとして扱われる)	
protected	(指定不可)	
public	他パッケージからもアクセス可	同一パッケージからのみアクセス可

次に、
アクセス修飾子を利用して情報隠蔽・カプセル化を進めている例を、これまでに示したプログラムの中から幾つか抜き出し再確認

例22．24（アクセス制御; 内部作業用クラス）

例題19.5... TowerOfHanoiConfigクラスのメンバーとして
privateでstaticな入れ子クラス Disk を用意

例題19.6... NumberWith1000DecimalPlacesクラスのメンバーとして
privateでstaticな入れ子クラス Digit を用意

例題19.7... BinaryTreeOfStringIntクラスのメンバーとして
privateでstaticな入れ子クラス Node を用意

いずれの入れ子クラスも 、外側のクラスの中で使うことしか想定していないので、private宣言し外部からの利用は出来なくしている。

```
[motoki@x205a]$  cat -n TowerOfHanoiConfig.java
     1 /* Hanoiの塔の問題における、途中の円盤の配置状況を表す...
     2
     3 public class TowerOfHanoiConfig {
     4     //棒に挿す円盤を表すオブジェクトのクラス
     5     private static class Disk {
                 ......
    23     }
             ......
   115 }
---
```

例22．25（アクセス制御; 内部作業用フィールド, メソッド）

例題19.6(と例22.1)... StackOfAnyObjectsクラスのメンバーとして
privateなインスタンスフィールド DEFAULT_INITIAL_CAPACITY, DEFAULT_CAPACITY_INCREMENT, stack, indexOfTopEle を用意

例題22.7... HeapsortIntArrayクラスのメンバーとして
privateなインスタンスメソッド heapify() を用意

これらは、いずれもインスタンス外から自由に利用させる必要がないものなので、private宣言して情報隠蔽

```
[motoki@x205a]$  cat -n StackOfAnyObjects.java
     1 /* Objectインスタンスを格納するpushdownスタックオブ... *
     2
     3 import java.util.*;
     4
     5 public class StackOfAnyObjects {
     6     private static final int DEFAULT_INITIAL_CAPACITY =
     7     private static final int DEFAULT_CAPACITY_INCREMENT
     8
     9     private Object[] stack;
    10     private int indexOfTopEle;
           ......
___ 107 }
```

例22．26（アクセス制御; アクセッサを用いたフィールドの管理）

例題19.4... Rectangleクラスのメンバーとして

protected,finalなインスタンスフィールド id と

protectedなインスタンスフィールド width, height を用意

これらのフィールドのアクセス修飾子をpublicや「なし」にすると 、

外部から中身の閲覧だけでなく自由に書き換えができてしまい問題

⇨ アクセス修飾子をprotectedとしてこれらのフィールドへの

直接のアクセスを自クラスとサブクラス内に限定し、

代わりに必要に応じてゲッターメソッドやセッターメソッドを用意

例えばidについては 、

インスタンス生成以降値を変更することはない

$\Longrightarrow$ ゲッターメソッド(getId())だけを用意, 更にfinal宣言も

ここでpublic宣言されたアクセッサについても 、

将来の状況に応じてアクセス修飾子を変更して

情報隠蔽の度合いを調節可

```
[motoki@x205a]$  cat -n Rectangle.java
     1 /* 長方形を表すオブジェクトのクラス */
     2
     3 public class Rectangle {
     4     protected final int id;    //長方形インスタンスに付け.
     5     protected double width;    //長方形の幅
     6     protected double height;   //長方形の高さ
             ......
    28     //ゲッターメソッド
    29     public int getId() {
    30         return id;
    31     }
             ......
    58 }
```

例22．27（アクセス制御;インスタンス生成を抑制）

例題22.7 ... HeapsortIntArrayクラスのコンストラクタを
private宣言し外部からのインスタンス生成を出来なくしている。
(複数のインスタンスを生成してもメモリの無駄にしかならないため)

外部からのインスタンス生成を抑制する代わりに 、
クラス内部でインスタンス1個を生成し保持した上で、
そこへの参照値を外部に対して教えるpublic，staticなメソッド
getInstance() を用意

```
[motoki@x205a]$  cat -n HeapsortIntArray.java
     1 /**
     2  * int配列内の要素をheapsort手法で昇順に並べ替える機能
     3  * を備えた整列化モジュールを作り出すためのクラス
     4  */
     5 public class HeapsortIntArray extends SortModuleForIntA
     6     //クラス内部でインスタンスを1個だけ生成
     7     //  （コンストラクタはprivate宣言してあるので、  ）
```

```
 8      //  （生成されるインスタンスはこの1個だけになり、）
 9      //  （これが使い回されることになる。　　　　　　　）
10      private static final HeapsortIntArray INSTANCE
                                        = new HeapsortIntArray();
11
12      //コンストラクタ(外部からインスタンス生成不可)
13      private HeapsortIntArray() {
14          super();
15      }
16
17      /** コンストラクタの代わりに外部に整列化モジュールを...
18      public static HeapsortIntArray getInstance() {
19          return INSTANCE;
20      }
          ......
86  }
```

22–16 ほぼ自習 オブジェクト指向のまとめ、利

オブジェクト指向の特徴と利点を以下に列挙する。

オブジェクト指向の基本的な特徴：

- クラスを定義し、そのクラスのインスタンス（ソフトウエア部品）を必要なだけ生成して利用する。
  ⇨ （利点0）コードの簡素化 ··· 類似コードをあちこちに書かなくて済むので。

オブジェクト指向の3大要素：

- カプセル化 ··· 情報隠蔽を進めてオブジェクト（ソフトウェア部品）の独立性を高める。
  ⇨ （利点1.1）モジュール性 ··· 他のオブジェクトと切り離して、オブジェクト毎にソースコードを作成・保守することができる。（19.2節の記述）
  （利点1.2）情報隠蔽の恩恵 ··· 内部の実装方式がちゃんと隠蔽さ

れていて隠蔽しているはずの事柄に依存したコードが他のオブジェクト中に現れないことが保証されるなら、オブジェクト内部の実装方式を自由に変更することができる。(19.2節の記述)

アクセッサメソッドを用いて情報隠蔽を行なっている場合
情報の保持方法の変更はアクセッサメソッドの処理内容を変更するだけで済む。

(利点1.3) コードの再利用 ··· 一般的な処理を行うオブジェクトの場合、他からの独立性を高めることにより、コード再利用の可能性が高まる。(19.2節の記述)

(利点1.4) ソフトウェア全体の保守の容易さ ··· 1つのオブジェクトで異常が発生した場合でも、そのオブジェクトを代替オブジェクトに差し替えたり、場合によってはそのオブジェクトを全体から切り離して残りの部分を運用したり(fail soft)、ということを行い易い。(19.2節の記述)

- 継承 ··· 既存のクラスの内容を引き継いで新たな別のクラスを定義できる。

  ⇨ （利点2.1）効率的なプログラミング ··· 簡単に既存クラスを拡張できる。また、類似クラスができそうな場合は、それらの共通部分を親クラスとして構成することにより、類似コードをあちこちに書かなくて済む。

  （利点2.2）間違いの可能性の減少 ··· 各種機能を整理して配置し、類似コードが多数に場所に分散するのを極力避けることが出来るので、コードの修正忘れも少なくなる。

- 多態性 ··· 同じメソッドに対してオブジェクトごとに異なる振る舞い。

  ⇨ （利点3）コードの簡素化 ··· 同種のインスタンスを統合的に扱えるので。

注意： ちゃんと書けばこういう利点の恩恵に与れる、という話である。以上の利点を充分に引き出せてないプログラムはJavaで書いてあっても非オブジェクト指向と言える。

例22．28（手続き指向プログラムvs.オブジェクト指向プログラム）

同じ問題に対して

- 手続き的に構成されたプログラムと
- オブジェクト指向の考え方で構成されたプログラム

を対比することによって、オブジェクト指向の特徴と利点の認識を深めて下さい。

扱った問題： 2種類の書式

"Book,[本の書名],[著者名],[価格],[在庫数]"

と

"DVD,[DVDのタイトル],[価格],[在庫数]"

に従った文字列データを要素とする配列を引数として受け取り、中に書かれた在庫データを分析した上で

本の総冊数 = [調査結果]

DVDの総数 = [調査結果]

10冊以上在庫がある本のタイトル数 = [調査結果]

総金額 = [調査結果]

という風に出力するメソッドを作る。

手続き指向プログラム：

```
[motoki@x205a]$  cat StockTakingNonOOP.java
/**
 * 在庫状況を把握するためのクラス(手続き指向版)
 */
public class StockTakingNonOOP {
    /** 在庫状況の概要を出力する */
    public static void printOutline(String[] lines) {
        int countOfBooks = 0,                    //本の総冊数
            countOfDVDs = 0,                     //DVDの総数
            countOfTeemingBookTitles = 0,
                                  //10冊以上在庫がある本のタイトル数
            amount = 0;                          //総金額

        for (String line : lines) {
            if (line.startsWith("Book")) {
                String[] data = line.split(",");
                int count = Integer.parseInt(data[4]);
                countOfBooks += count;
```

```
            if (count >= 10) {
                ++countOfTeemingBookTitles;
            }
            int price = Integer.parseInt(data[3]);
            amount += price * count;
        } else {
            String[] data = line.split(",");
            int count = Integer.parseInt(data[3]);
            countOfDVDs += count;
            int price = Integer.parseInt(data[2]);
            amount += price * count;
        }
    }

    System.out.println("本の総冊数 = " + countOfBooks);
    System.out.println("DVDの総数  = " + countOfDVDs);
    System.out.println("10冊以上在庫がある本のタイトル数 =
                        + countOfTeemingBookTitles);
    System.out.println("総金額     = " + amount + "円");
```

```
    }

    //---------------------単体での動作テスト用-----------------
    public static void main(String[] args) {
        String[] stockData = {
            "Book,ゼロから学ぶ!最新Javaプログラミング,日経...,25
            "DVD,サウンド・オブ・ミュージック,1490,10",
            "Book,プログラミング言語Java第4版,K.Arnold他,441(
            "Book,Javaチュートリアル第4版,S.Zakhour他,5040,5"
        printOutline(stockData);
    }
}
[motoki@x205a]$  javac StockTakingNonOOP.java
[motoki@x205a]$  java StockTakingNonOOP
本の総冊数 = 27
DVDの総数  = 10
10冊以上在庫がある本のタイトル数 = 1
総金額      = 99320円
[motoki@x205a]$
```

オブジェクト指向プログラム：

```
[motoki@x205a]$  cat StockTakingOOP.java
/**
 * 在庫状況を把握するためのクラス(オブジェクト指向版)
 */
public class StockTakingOOP {
    /** 在庫状況の概要を出力する */
    public static void printOutline(String[] lines) {
        int countOfBooks = 0,                    //本の総冊数
            countOfDVDs = 0,                     //DVDの総数
            countOfTeemingBookTitles = 0,
                                  //10冊以上在庫がある本のタイトル数
            amount = 0;                          //総金額

        for (String line : lines) {
            Item item = null;
            if (line.startsWith("Book")) {
                item = new Book(line);
                countOfBooks += item.getCount();
```

多態変数

```
                if (((Book)item).isTeeming()) {
                    ++countOfTeemingBookTitles;
                }
            } else {
                item = new DVD(line);
                countOfDVDs += item.getCount();
            }
            amount += item.getAmount();
        }

        System.out.println("本の総冊数 = " + countOfBooks);
        System.out.println("DVDの総数  = " + countOfDVDs);
        System.out.println("10冊以上在庫がある本のタイトル数 =
                           + countOfTeemingBookTitles);
        System.out.println("総金額     = " + amount + "円");
    }

    //--------------------単体での動作テスト用-----------------
    public static void main(String[] args) {
```

```
        String[] stockData = {
            "Book,ゼロから学ぶ!最新Javaプログラミング,日経...,25
            "DVD,サウンド・オブ・ミュージック,1490,10",
            "Book,プログラミング言語Java第4版,K.Arnold他,4410
            "Book,Javaチュートリアル第4版,S.Zakhour他,5040,5"
        printOutline(stockData);
    }
}

//----------------------補助的なクラス定義-----------------------
/** 商品のクラス */
class Item {
    private int count = 0;
    private int price = 0;

    public int getCount() {
        return count;
    }
```

```
    public int getPrice() {
        return price;
    }

    protected void setCount(int count) {
        this.count = count;
    }

    protected void setPrice(int price) {
        this.price = price;
    }

    public int getAmount() {
        return price * count;
    }
}

/** 本のクラス */
class Book extends Item {
```

```
    private static final int THRESHOLD_FOR_TEEMING_BOOK = 10

    public Book(String bookInfo) {
        String[] data = bookInfo.split(",");
        setPrice(Integer.parseInt(data[3]));
        setCount(Integer.parseInt(data[4]));
    }

    public boolean isTeeming() {
        return (getCount() >= THRESHOLD_FOR_TEEMING_BOOK);
    }
}

/** DVDのクラス */
class DVD extends Item {
    private static final int THRESHOLD_FOR_TEEMING_BOOK = 10

    public DVD(String dvdInfo) {
        String[] data = dvdInfo.split(",");
```

```
            setPrice(Integer.parseInt(data[2]));
            setCount(Integer.parseInt(data[3]));
        }
}
[motoki@x205a]$  javac StockTakingOOP.java
[motoki@x205a]$  java StockTakingOOP
本の総冊数 = 27
DVDの総数  = 10
10冊以上在庫がある本のタイトル数 = 1
総金額      = 99320円
[motoki@x205a]$
```

補足： オブジェクト指向にするとコード量も増え、かえって複雑になった様に見えないこともない。しかし、

- 抽象化を進め、
- 類似部分を抽出し親クラスとしてまとめる、

等してオブジェクト指向化することによって、プログラムが幾つかの独立な部品に分けられ、商品の種類が増える等の時にも関連する部品だけに修正範囲を留めることができる様になっている。

オブジェクト指向設計の原則:

{日経ソフトウェア編「ゼロから...」第3部1章p.181}

- 単一責務の原則(SRP, Single Responsibility Principle) ···
  分割された個々のプログラムに複数の目的・機能を負わせるべきでない、という指針。(その方が、将来の部分的な変更もやり易い。)

- 開放閉鎖の原則(OCP, Open-Closed Principle) ···
  構築するクラス群は、
  機能拡張可能(open)で、
  拡張の際には既存コードには手を加えなくて済む(closed)、
  様なものが良い、という指針。

補足（良いプログラムを構築するための考え方と実践方法）:

{日経ソフトウェア編「Javaツール完全理解」第2部2章}

- ソフトウェアの価値の3条件
  - ◇ シンプル ··· プログラムの理解や修正が容易になる。
  - ◇ コミュニケーション可 ··· プログラムの書き手と読み手がソースコードを通じて十分にコミュニケーションできる。
  - ◇ 柔軟性 ··· 変更に対する柔軟性がある。（初期開発費用より修正費用の方が大きいので、これも重要。）
- プログラミングの原則
  - ◇ YAGNI (You Aren't Going to Need It.) ···
    今必要なことだけをやる。
  - ◇ DRY (Don't Repeat Yourself.) ···
    コードの重複を避け、必要があればできるだけ再利用する。
  - ◇ PIE (Program Intently and Expressively.)···
    意図が明確に伝わる様にコードを書く。
- 代表的なベストプラクティス ..... （次ページ）

補足(良いプログラムを構築するための考え方と実践方法,続き):

- ソフトウェアの価値の3条件 ..... (前ページ)
- プログラミングの原則 ..... (前ページ)
- 代表的なベストプラクティス
  - ◇ リファクタリング ··· 外部に対する振舞いを変えずに、ソースコードの内部構造を整理し簡素化すること。
  - ◇ テストファースト ··· 実装者の視点で実装コードを書く前に、利用者の視点でテストコードを書く。これによって余分な複雑さを排除できることを期待する。
  - ◇ ドメイン駆動設計 (Domain-Driven Design, DDD) ··· 問題領域(domain)をモデル化し、それを中心に据えてソフトウェアを設計する。(モデル自体もドメイン知識を使って反復的に深化させていく。)

22–17 ほぼ自習 ソフトウェアの部品化と再利用

プログラミング言語の進化 プログラミング言語に備わっているべき事柄は、...

- アルゴリズムを容易にコード化できるための表現能力
- ソフトウェアの寿命が伸びた
  ⟹ 保守も大事
  ⟹ 出来上がったプログラムの理解や修正の容易さも重要
- ソフトウェアには高い品質が必要
  ⟹ プログラムの中に余計な複雑さや単純な間違いが入り込みにくいということも大切
- ソフトウェアの生産性を上げたい
  ⟹ 実績のあるプログラムを再利用する仕組み

⇨ プログラミング言語がどの様に進化してきたのかを、
--- 次の様にまとめることができる。

	表現能力	保守性	品質保証	再利用性	導入された機構/考え方
機械語					
アセンブリ言語	△				
高級言語	○			△	• サブルーチン ⟹ 表現能力, 再利用性向上
構造化	○	△	△	△	• 3つの基本構造 ⟹ 保守性向上 • サブルーチンの独立性を高める ⟹ 再利用性多少向上
オブジェクト指向	○	○	○	○	• クラス(関連する変数とメソッドをまとめる仕掛け) ⟹ 保守性向上, 再利用性向上 • 例外機構 ⟹ 品質向上 • ガベージコレクション ⟹ 品質向上 • 型チェックの強化 ⟹ 品質向上 • 多態性 ⟹ 再利用性向上 • 継承 ⟹ 再利用性促進

時間 ↓

ソフトウェア再利用技術発展の流れ

オブジェクト指向より前の構造化言語では、

$\Longrightarrow$ 再利用できるソフトウェアと言えばサブルーチンだけ
コード変換，入出力処理，数値計算，. . . の汎用ライブラリ程度

オブジェクト指向の考えが導入されると、

(関連性の強いサブルーチンや大域変数を1つのクラスとしてまとめて粒度の大きいソフトウェア部品を作り出す仕組みがある)

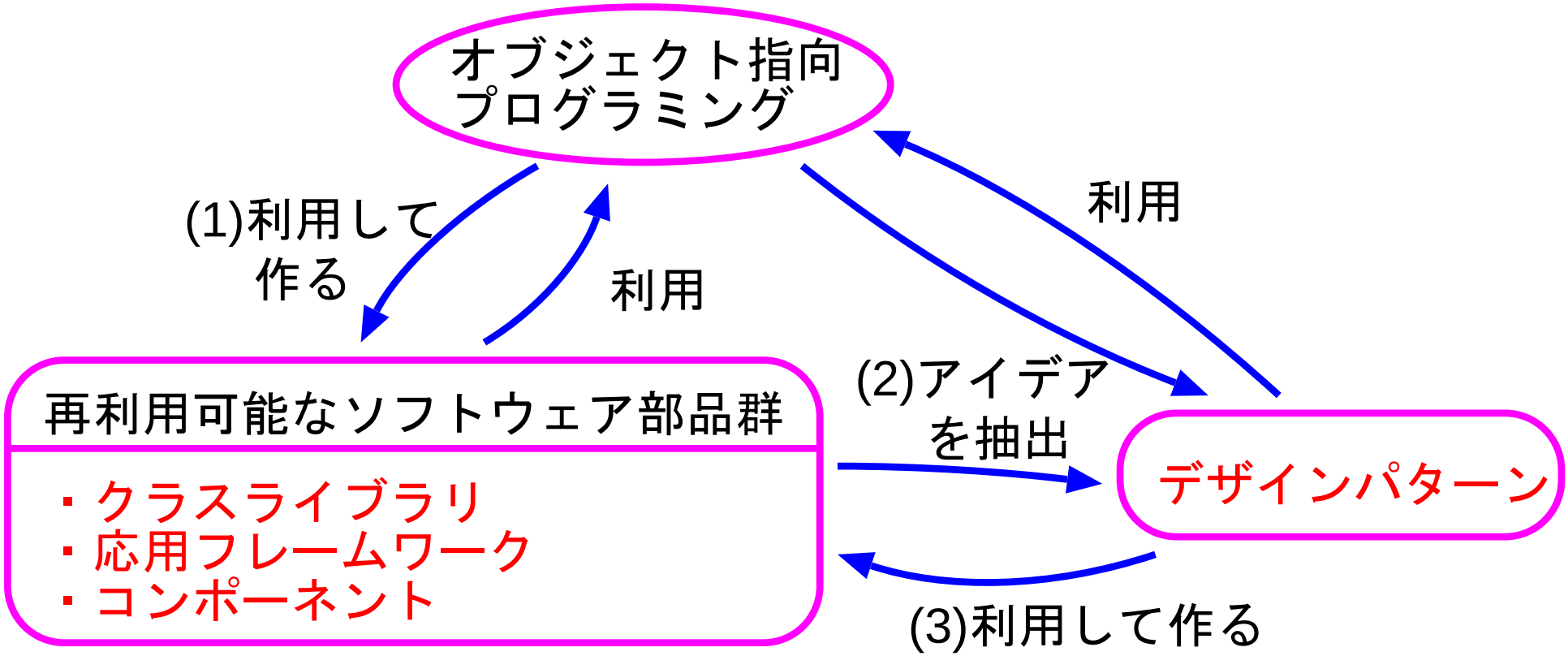
$\Longrightarrow$ ソフトウェア再利用の可能性は大きく広がる。

$\Longrightarrow$ ソフトウェア部品として既に存在しているソースコードや実行形式モジュールを使い回すのが当り前。

再利用可能なソフトウェア部品としては、現在、

- クラスライブラリ、
- (応用) フレームワーク、
- コンポーネント、
- デザインパターン··· ソフトウェア設計のアイデアを後で利用できるように文書化したもの

と呼ばれるものがある。



クラスライブラリ　　・・・汎用的な機能をもつクラスを多数蓄積したもの

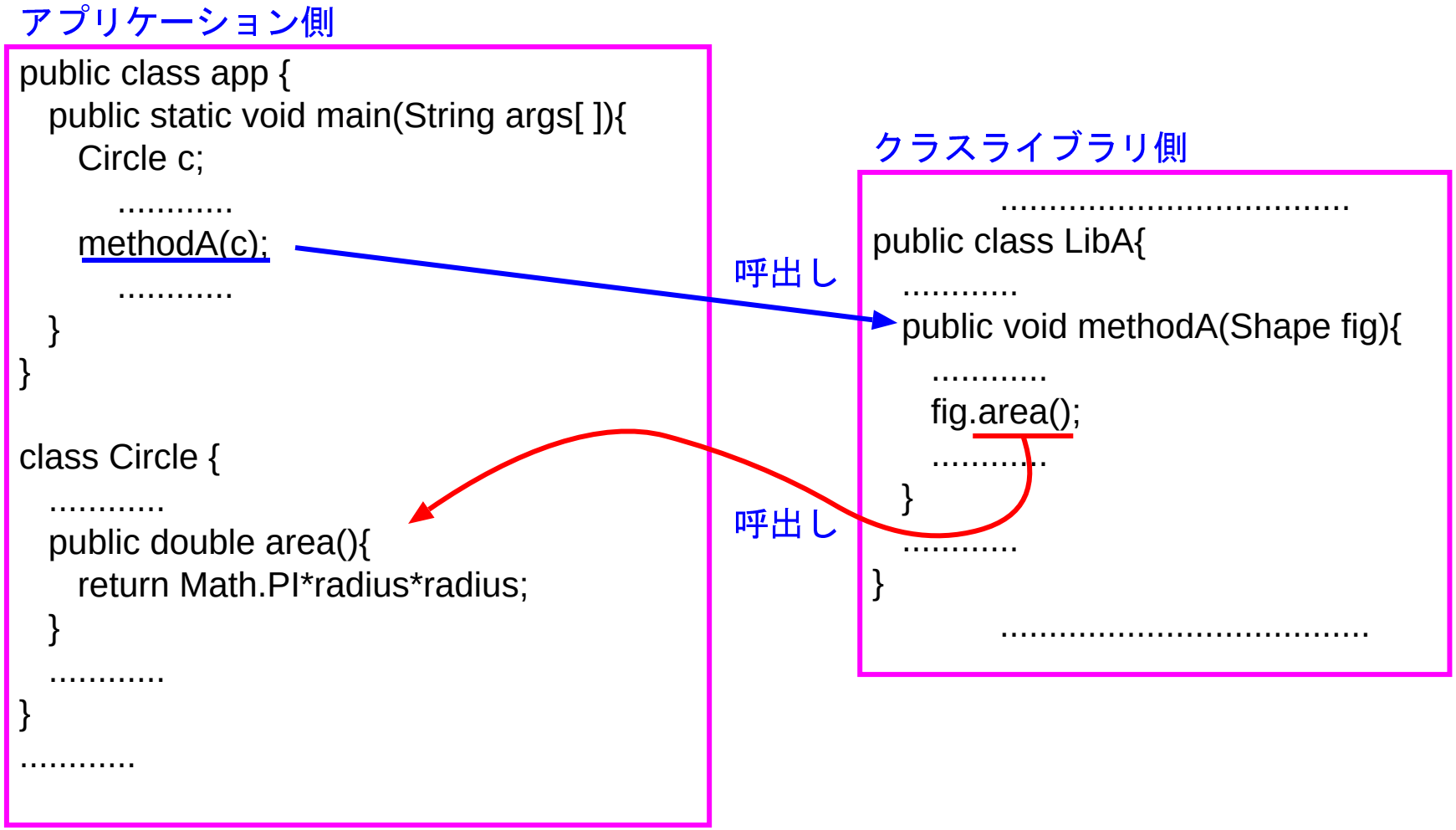
オブジェクト指向の前と比べて

- ソフトウェアが格段に豊富になった。
- ライブラリの利用の仕方も広がった。

クラスライブラリの場合は次の様な3つの利用の仕方が可能

- 用意されたクラスのインスタンスを作成して
  付属のインスタンスメソッド等を利用。
  （クラスの利用，従来のライブラリ関数呼び出しに相当。）

- ライブラリのコードからアプリケーション固有の処理を呼び出す。
  （多態性の利用。）



- ライブラリ内のクラスを拡張・補正して、新しいクラスを作成。（継承の利用。）

特に、現在のJava(J2SE7.0; JDK1.7, Java SE Development Kit 1.7)には、GUI, 入出力, ネットワーキング, ... 等のために、合わせて4000にものぼるクラスライブラリが整備されている。

言語仕様は最小限に抑えて
必要な機能はクラスライブラリとして提供される
$\Longrightarrow$ 言語仕様の互換性を保ちながら
クラスライブラリの拡張によってバージョンアップを行うことが可能。

$\Rightarrow$ この豊富なライブラリを使いこなすことが、
Java習熟への1つの道です。

(応用) フレームワーク

特定の業務分野について、色々な利用者に共通な部分は完成させておいて、個別の要求のある部分だけを追加するだけでそれぞれの利用者に合った応用プログラムを作成できる様にした、いわば「半完成品」を応用フレームワークあるいは単にフレームワークという。

> クラスライブラリも応用フレームワークも再利用可能なソフトウェア部品群という点では同じ。ただ、目的と再利用部品の使われ方が違う。

例22．29（応用フレームワーク, Javaアプレット）

Javaアプレットは応用フレームワークの代表例。

> 実際、Appletクラスのmainメソッドはアプレットの全体的な動作を規定するものとして予め定義されており、我々はmain の中から呼び出される init(), start(), paint(), ... 等の細部を書くだけで、動的なWebページを手軽に作ることができる。

コンポーネント　…(クラスライブラリ，応用フレームワークと全然違う)

平澤 (2004) によれば、

- クラスよりも粒度が大きく、
- (ソースコード形式でなく) バイナリ形式で提供される、

そして、

- ソフトウェア部品の定義情報も提供される、
- 機能的に独立性が高く内部の詳細を知らなくても利用できる、

というものを一般にコンポーネントと呼ぶ。(広く浸透していない。)

利用の仕方も特徴的で、

ソースコードを書くのではなく、
視覚的なツールを用いて直接関連する部品を配置・設定・接続する

ことによって、短時間で応用ソフトウェアを組み立てる。

具体例：

マイクロソフト社VisualBasic(1990年代前半~)で導入 $\longrightarrow$ActiveX

Java環境ではJavaBeansと呼ばれる仕組み

Beansと呼ばれるコンポーネントを

BDK 等の（ビルダ）ツールで接続してソフトウェアを組み立てる。

デザインパターン

..... オブジェクト指向に基づいて
再利用性や柔軟性の高いソフトウェアを開発しようとする際に、
様々な場面で適用される「お決まりの設計指針」

有意義で適用範囲の広いデザインパターンが見つかれば、
①全てのソフトウェア開発者がその恩恵を受けることができ、また
②それが開発者間の共通認識として定着すれば開発者間のコミュニケーションも容易になる。

この様な状況はアルゴリズムやデータ構造の場合と同じ。

具体的なデザインパターンとしては、
E.Gamma, R.Helm, R.Johnson, J.Vlissides という4人の技術者達(GoF the Gang of Four)が発表した次の23種類(GoFのデザインパターンと呼ばれている)が有名で、これらはJavaのクラスライブラリの作成にも大いに利用されている。

GoFによる分類	パターン名	再利用を妨げる要因のどれに効果が期待されるか?						
		クラス名を固定したインスタンスの生成	特定の処理内容への依存	プラットフォームに依存したapplicattion programinterfaceの使用	特定のアルゴリズムへの依存	クラス同士の密接な依存関係	継承によるデメリット	クラス数の急激な増加
生成に関するパターン	Abstract Factory	○		○		○		
	Builder				○			
	Factory Method	○						
	Prototype	○						
	Singleton							
構造に関するパターン	Adapter							
	Bridge			○		○	○	○
	Composite						○	
	Decorator						○	○
	Facade					○		
	Flyweight							
	Proxy							
振舞いに関するパターン	Chain of Responsibility		○			○	○	
	Command		○			○		
	Interpreter							
	Iterator				○			
	Mediator					○		
	Memento							
	Observer					○	○	
	State							
	Strategy				○		○	
	Template Method				○			
	Visitor				○			

GoFの著作においては、

これらのデザインパターンは次の様な項目に分けて説明されている。

- パターン名 ···
- 目的 ··· そのデザインパターンがどの様な設計課題に対処するか、何をもたらすか、原理と意図、等を簡潔に。
- （別名 ···　　）
- 動機 ··· 設計上の問題点、及び、そのパターン内のクラスやオブジェクトの構造がどの様にその問題を解決するか、のシナリオ。
- 適用場面 ··· このデザインパターンを適用できる状況。
- 構造 ··· クラス間の関係、要求のシーケンス、...を図で。
- 構成要素 ··· 使われるクラス、オブジェクトと、各々の役割。
- 協調関係 ··· 各構成要素がどの様に協調して役割を果たすか。
- 結果 ··· そのパターンが要求に対してどの様に効果を発揮するか。
- 実装 ··· 実装方法や注意点。
- サンプルコード ··· そのデザインパターンを使って実装した例。
- 利用例 ··· そのデザインパターンが実際のシステムで利用された例。
- 関連するパターン ··· 別のデザインパターンとの関係。

例22．30（Iteratorパターンの活用）

例えば、配列 arr[] の個々の要素に対してgetName()の実行依頼を出し、戻って来た文字列を全て表示するのに

```
for (int i=0; i<arr.length; i++)
  System.out.println(arr[i].getName());
```

これは、
　　オブジェクトの集合体を表すのに配列を用いるということに依存したコード

⇨ **Iteratorパターンの指針に従えば、上のコードは**

```
Iterator it = 集合体オブジェクト.iterator();
while (it.hasMoreElements()) {
  クラス名 obj = it.nextElement();
  System.out.println(obj.getName());
}
```

という風に、集合体の実装方法に依存しないコードに置き換わる。

補足： J2SE5.0以降では

Iteratorを用いずに拡張for文を用いて

```
for (クラス名 element :  arr)
    System.out.println(element.getName());
```

と書けるが、この拡張for文も内部ではIteratorの仕組みを利用

例22．31（Singletonパターンの活用）

インスタンス生成を1度だけに限定したいクラスもある。

⇨ **Singletonパターン**

これはこの講義ノートの中でも既に

例題22.7 の HeapsortIntArray.java,
BubblesortIntArray.java,
LListsortIntArray.java,

例題22.12 の TesterForSortModuleIntArray.java

を構築する際に利用している。

例22．32（Strategyパターンの活用）

処理の大枠は固定するが、
その中で使うアルゴリズム(strategy)は色々と切り替えて使いたい、
という場合もある。

⇨ **Strategyパターン**

例題22.12で考えた TesterForSortModuleIntArray クラス
... インスタンスには個別の整列化モジュールは持たせなかった

Strategyパターンに従って
個別の整列化モジュールとその名前を
内部のインスタンス変数にもたせる様に変形すると...

```
[motoki@x205a]$  cat TesterForFixedSortModuleIntArray.java
import java.util.Scanner;
import java.util.Random;

/**
```

```
 * 内部に保持するSortModuleForIntArrayモジュールの
 * 「int配列内の要素を昇順に並べ替える機能」が正しく動作するか
 * どうかをテストする機能を備えたモジュールを作り出すためのクラス
 */
public class TesterForFixedSortModuleIntArray {
    private static final int SIZE  =100;
    private static final int WIDTH = 10;

    private final SortModuleForIntArray sortModule;

    //コンストラクタ
    public TesterForFixedSortModuleIntArray(
                          SortModuleForIntArray sortModule) {
        this.sortModule = sortModule;
    }

    /** オブジェクトの説明を答える */
```

```
    @Override
    public String toString() {
        return "Tester for " + sortModule;
    }
```

以下の3行を追加

```
    /** SIZE個のランダムなデータから成る配列に対して
     * 内部で保持する整列化モジュールを実行してみる */
    public void runOnRandomData() {
```

以下、例題22.12 の TesterForSortModuleIntArray.java と同じ

```
[motoki@x205a]$  cat TestFixedSortModulesIntArrayMain.java
/**
 * ・内部に保持する[HeapsortIntArrayオブジェクト]の整列化動作を...
 *    する機能を備えたTesterForFixedSortModuleIntArrayオブジェ...,
 * ・内部に保持する[BubblesortIntArrayオブジェクト]の整列化動作...
 *    する機能を備えたTesterForFixedSortModuleIntArrayオブジェ...,
 * ・内部に保持する[LListsortIntArrayオブジェクト]の整列化動作...
 *    する機能を備えたTesterForFixedSortModuleIntArrayオブジェ...
```

```
 * を生成し、これらを用いて内部に保持されている3つの整列化モジュー..
 * int配列内の要素を正しく昇順に並べ替えるかどうかをテストするJava
 */
public class TestFixedSortModulesIntArrayMain {
    public static void main(String[] args) {
        //HeapsortIntArrayオブジェクトの動作テスト
        TesterForFixedSortModuleIntArray testerForHeapsort
            = new TesterForFixedSortModuleIntArray(
                                HeapsortIntArray.getInstance());
        testerForHeapsort.runOnRandomData();
        System.out.println("---");

        //BubblesortIntArrayオブジェクトの動作テスト
        TesterForFixedSortModuleIntArray testerForBubblesort
            = new TesterForFixedSortModuleIntArray(
                              BubblesortIntArray.getInstance());
        testerForBubblesort.runOnRandomData();
```

```
        System.out.println("---");

        //LListsortIntArrayオブジェクトの動作テスト
        TesterForFixedSortModuleIntArray testerForLListsort
            = new TesterForFixedSortModuleIntArray(
                                LListsortIntArray.getInstance());
        testerForLListsort.runOnRandomData();
    }
}
[motoki@x205a]$  javac TestFixedSortModulesIntArrayMain.java
[motoki@x205a]$  java TestFixedSortModulesIntArrayMain
```

実行の様子は省略
