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

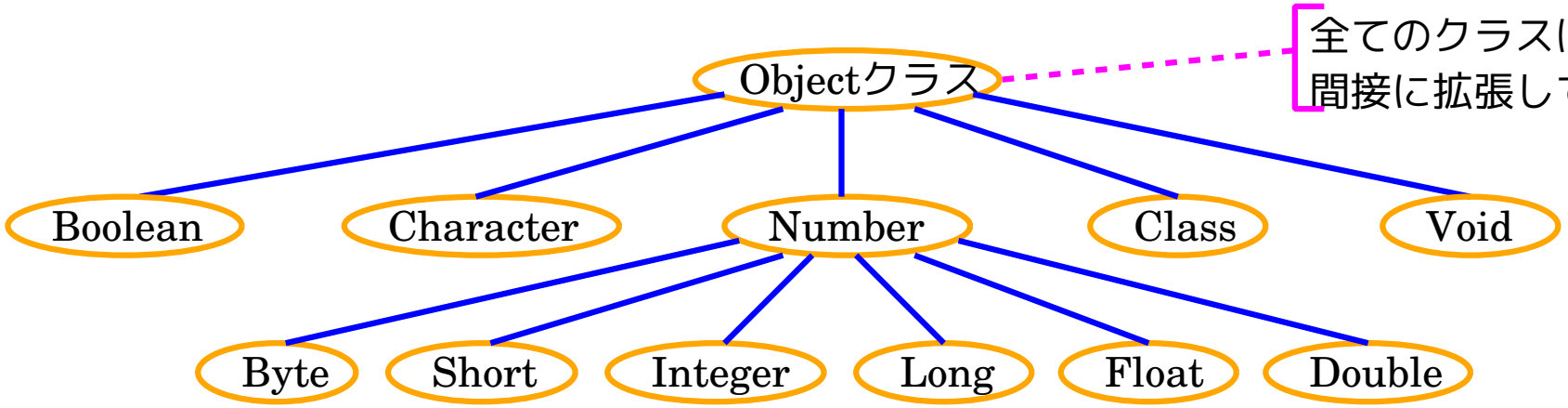
```
new Integer( int型の式 )
```

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 ⟶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);
```

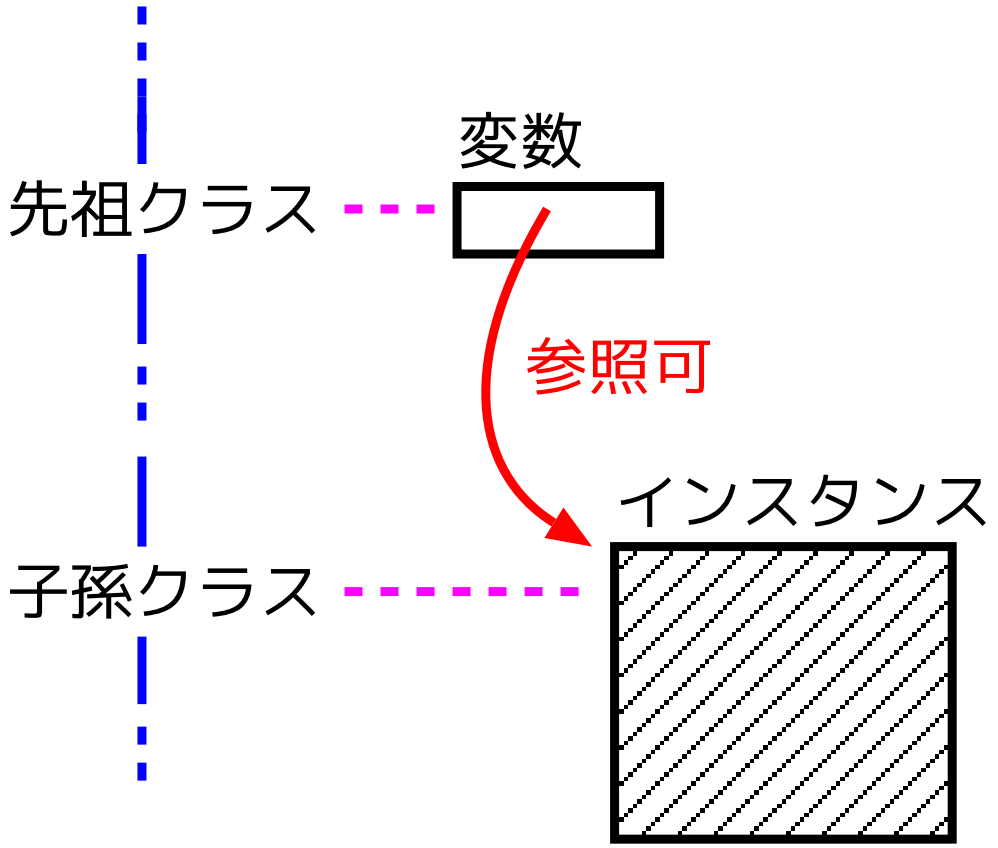
非オブジェクト指向言語における
代表的な多重定義・多態性の例

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、

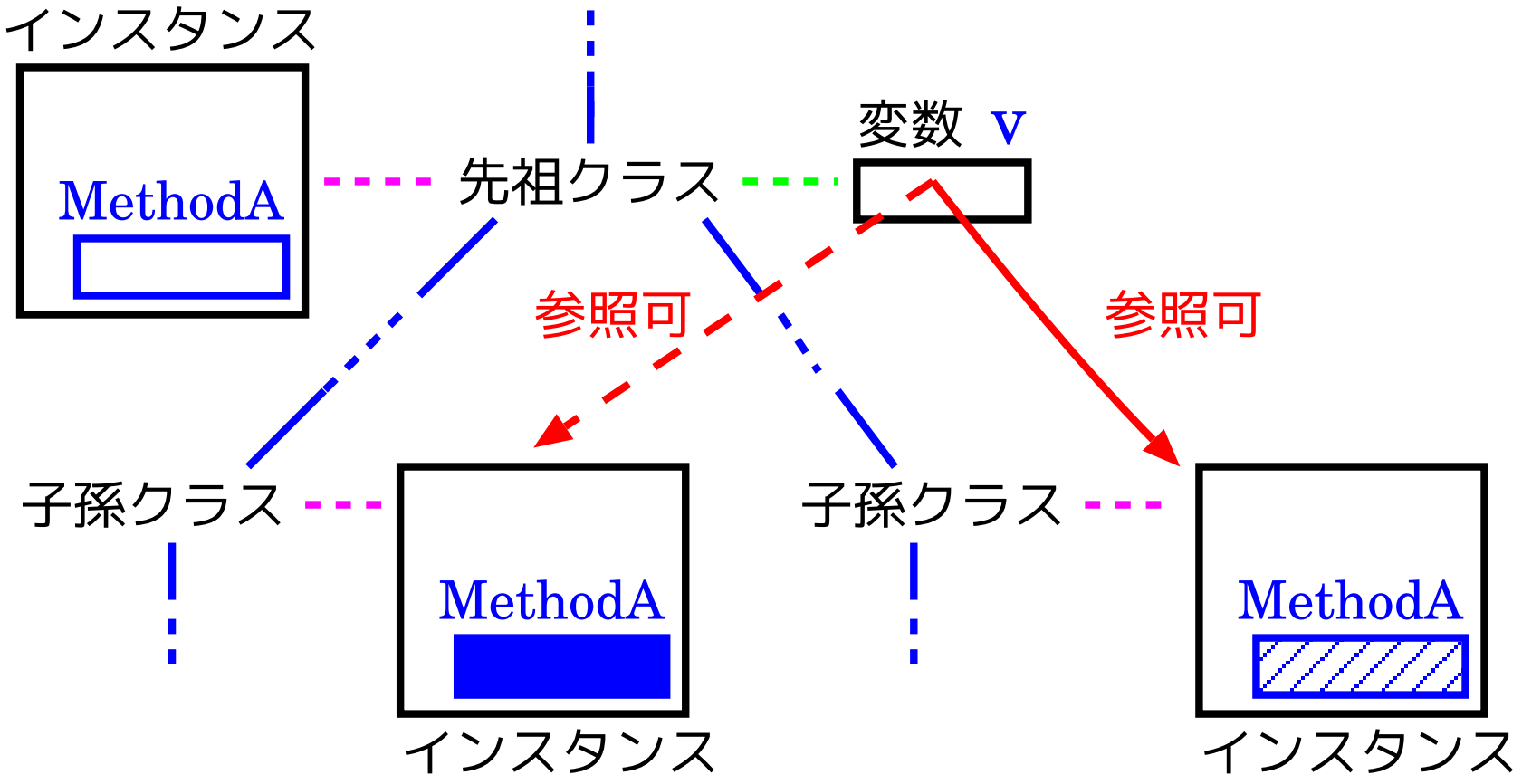
```
Ancestor v ;
```

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。

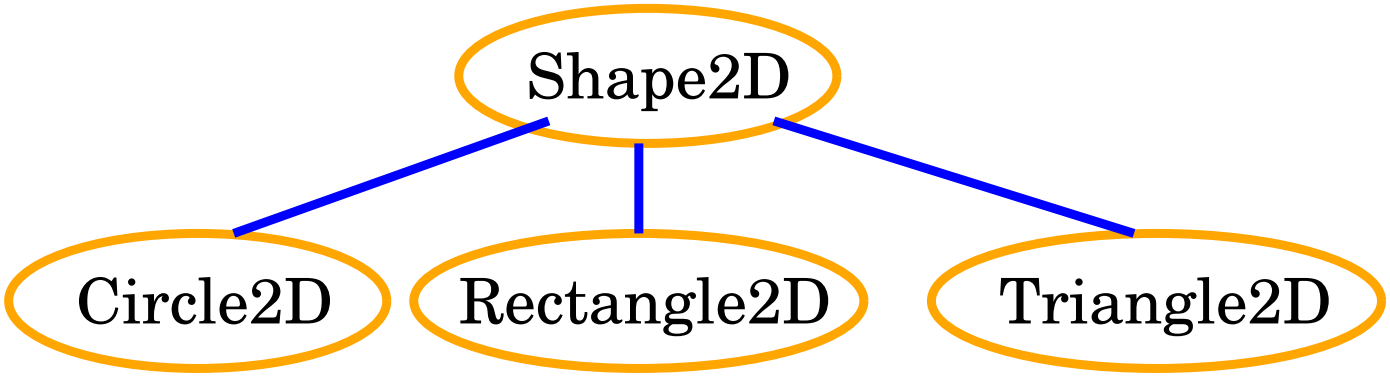
… (MethodAがオーバーライドされていた場合は、
Ancestor内で定義されたMethodA
ではなく
オーバーライドされた結果のMethodA
が呼び出される。)



⇨ 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



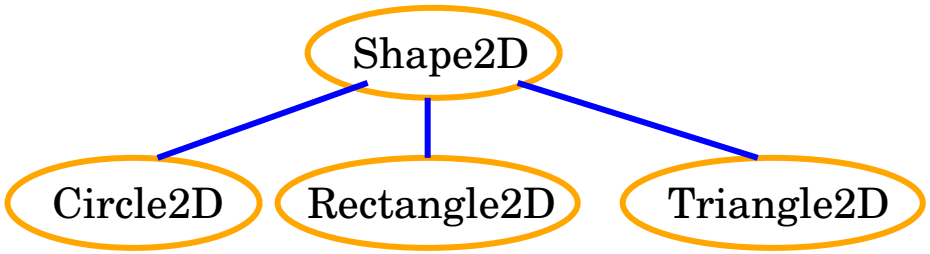
というクラス継承関係があり、

```
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
```

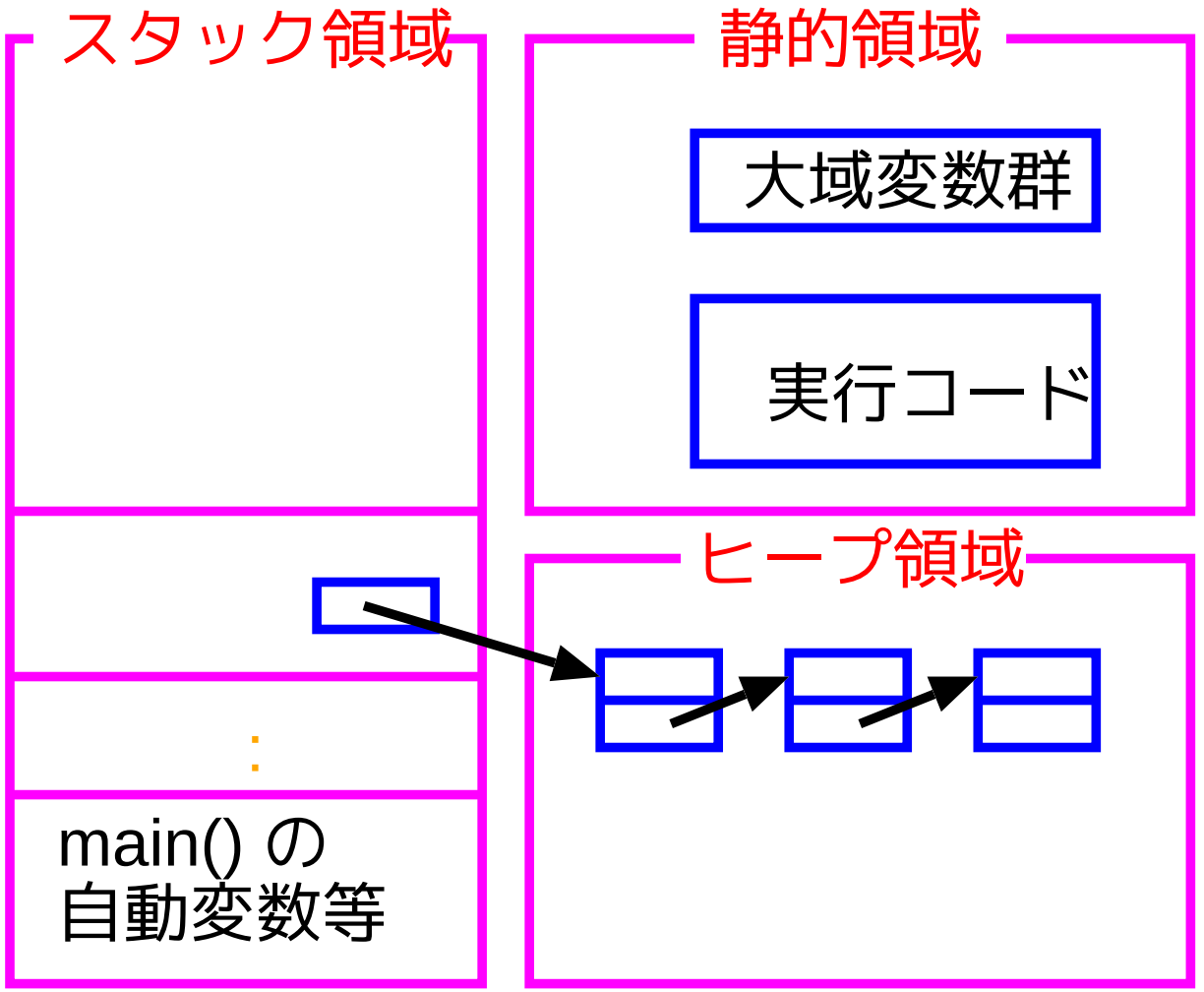
```
    166     169     170     210     217     223     227     234     23
    245     249     277     282     291     309     315     316     32
    330     344     359     361     364     386     408     415     42
    437     443     447     456     460     465     494     503     51
    544     544     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。

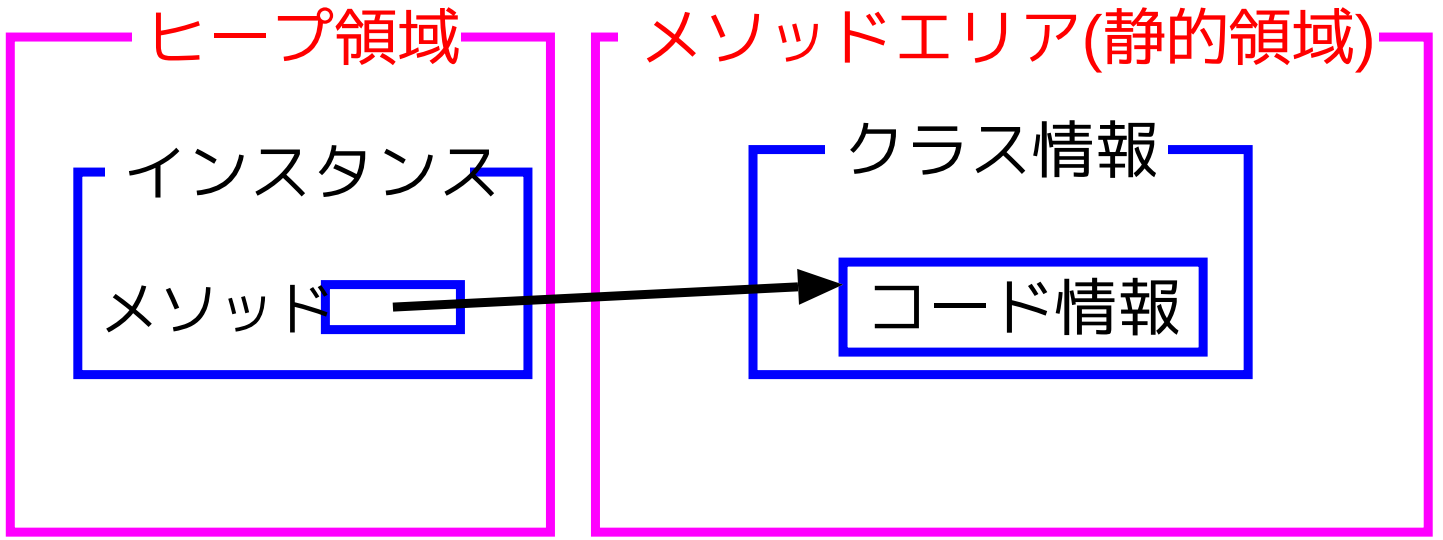
- オブジェクト指向の場合は、
  インスタンスオブジェクトは全て動的に生成される。
  $\Longrightarrow$ ヒープ領域は頻繁に利用される。

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...

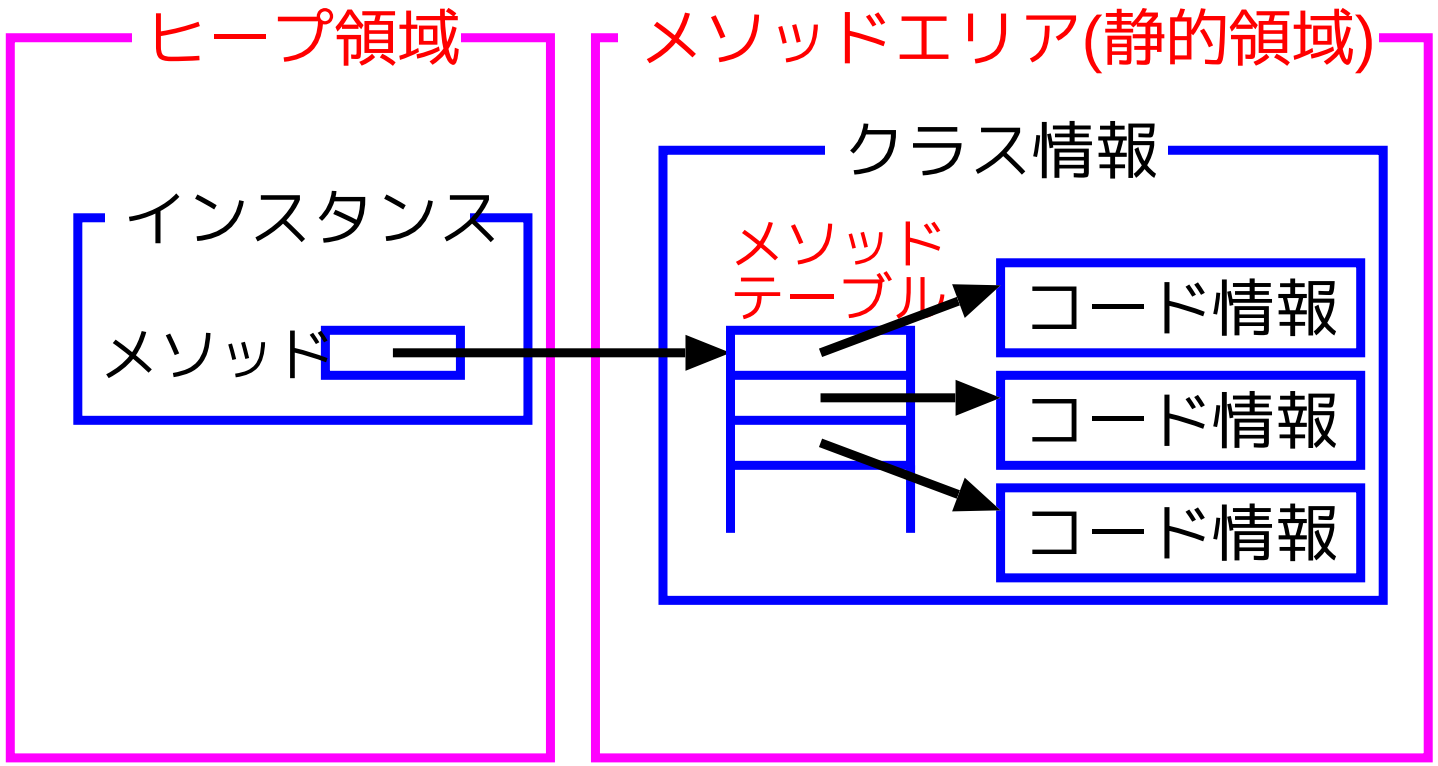
- new演算子が実行されると、
  インスタンスのための領域がヒープ領域内に確保され、
  確保された領域へのポインタがnew演算の結果の式の値となる。

- 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、

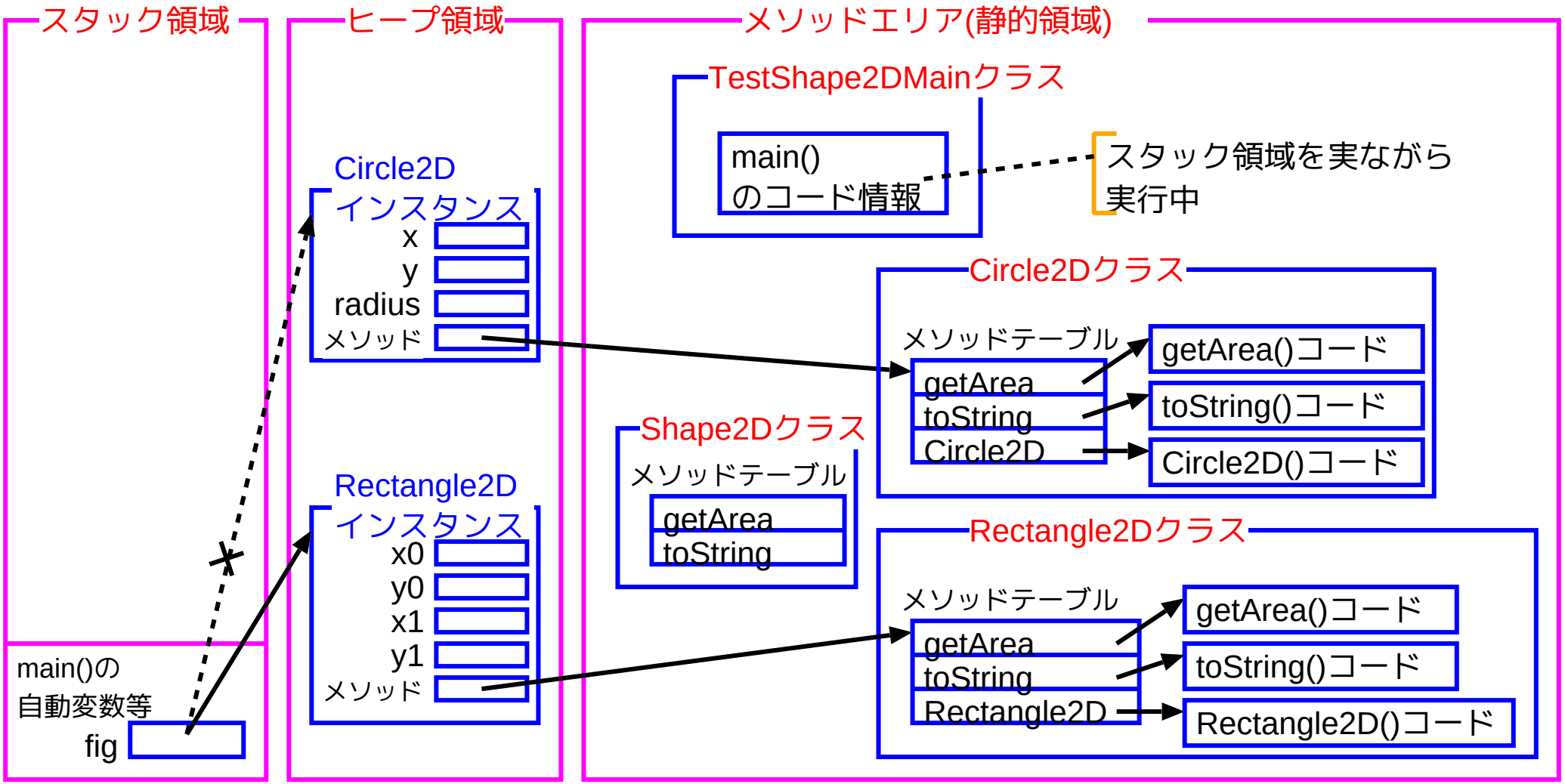
多態性の実装を容易に行うため、
クラス情報の領域内には、メソッドテーブルが配置され、個々のインスタンスの中にはそのメソッドテーブルへの参照情報だけが置かれることもある。



- 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 、 $\longrightarrow$ 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
