19 オブジェクトベースプログラミン

19–1 プログラムの部品化を進める

部品化／モジュール化再考：

8.3節では、C言語で関数以外のプログラム断片の中にもプログラムの部品／モジュールとして考えられるものがあることを説明した。 すなわち、

関数（または手続き）をモジュールと考えてプログラムを幾つかの機能単位に分ければ、確かにプログラムの各部分の役割分担が明確になる訳であるが、それぞれのモジュールが完全に独立になるかというと、そうでもない。

例えば、

pushdownスタックを
プログラム内に実装する場合

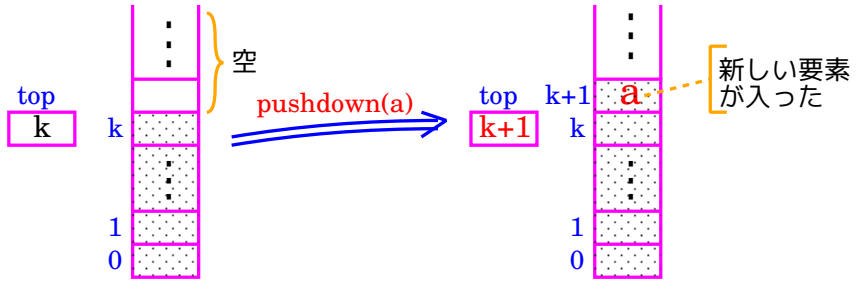
pushdownの関数とpopupの関数はスタック領域を共有することになる。

復習

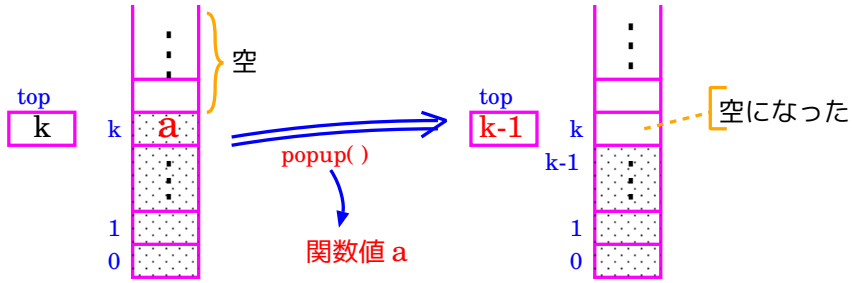
(pushdown)スタック

…pushdownとpopupの操作を備え、データの出し入れが後入れ先出し(last-in-first-out, LIFO)になるデータ記憶領域。

(pushdown操作)



(popup操作)



⇨ 他の部分と完全に独立になる様に、互いに関連したデータ領域，関数／メソッドを1つにまとめる（カプセル化という）べきである。

そうすることによって、これらのまとまりを1つのモジュール／部品として再利用できる様になる。 [pushdown関数だけでは再利用できないが、関連するデータ領域も含めると再利用できるようになる、ということ。]

カプセル化に際して： 次の点を心掛ける。

- 情報隠蔽 ··· 内部のデータは他のモジュールから見えない様にする。
- データの抽象化··· 他のモジュールからの操作要求に応えるために十分な関数／メソッドを用意し、内部のデータを保護する（情報隠蔽）のはもちろん、内部データの実装方法も見えない様にする。

  ⇨ モジュールの独立性／再利用の可能性が高まる。

例19．1（独立なモジュールの例）　例13.4, 例13.5において C言語で pushdown スタックを実装する際は、スタック領域, スタックの top の位置を記憶する領域, スタックを空に初期化する関数, スタックが空かどうかを調べる関数, pushdown関数, popup関数 が1つのモジュールを構成すると考え、これらを1つのファイルの中にまとめた。 すなわち、

```
 1  /*************************************************
 2  /* int型データが最大100個入るスタックを実装したモジュー...
 3  /*-----------------------------------------
 4  /*     外部へのサービスを行うために、次の４つの関数がこの
 5  /*     モジュールの中に用意されている。
 6  /*     (1)スタックを空に初期化する関数 initialize_stack,
 7  /*     (2)スタックが空かどうかを調べる関数 is_empty,
 8  /*     (3)スタックに要素を1つ push-down する関数 pushdown
 9  /*     (4)スタックから要素を1つ pop-up する関数 popup
10  /*************************************************

11  #include <stdlib.h>
12  #define  TRUE   1
```

```
13  #define  FALSE  0

14  typedef  int  Boolean;

15  static int  stack[100];  /* モジュール外からは見えない *
16  static int  top;         /* モジュール外からは見えない *

17  void initialize_stack(void)
18  {
19    top = -1;
20  }

21  Boolean is_empty(void)
22  {
23    if (top < 0)
24      return TRUE;
25    else
26      return FALSE;
27  }
```

```
28  void pushdown(int k)
29  {
30    if (++top >= 100) {
31      printf("stack overflow\n");
32      exit(EXIT_FAILURE);
33    }
34    stack[top] = k;
35  }

36  int popup(void)
37  {
38    if (top < 0) {
39      printf("popup from empty stack\n");
40      exit(EXIT_FAILURE);
41    }
42    return stack[top--];
43  }
```

スタック領域, スタックの**top**の位置を記憶する領域は`static`宣言された外部変数なので、これらの領域はこのファイルの中の関数だけで共有され、外の関数のアクセスから保護されている。

⇨ このモジュールは情報隠蔽が行われており、また、データの抽象化も行われているので、他の部分から独立した (従って、再利用の可能性の高い) モジュールになっている。

同様に、
例題**8.9**で考えたファイル`modules-random.c`(乱数発生),
例題**14.5**で考えたファイル`consumed_time.c`(時間計測)
は、他の部分から独立したモジュールになっている。

[**C**言語を使っても、
ある程度は情報隠蔽、データ抽象化を実現することが出来る！]

19–2 オブジェクトベースプログラミングの考え

前節の考え方に従って、

他の部分と独立になる様に

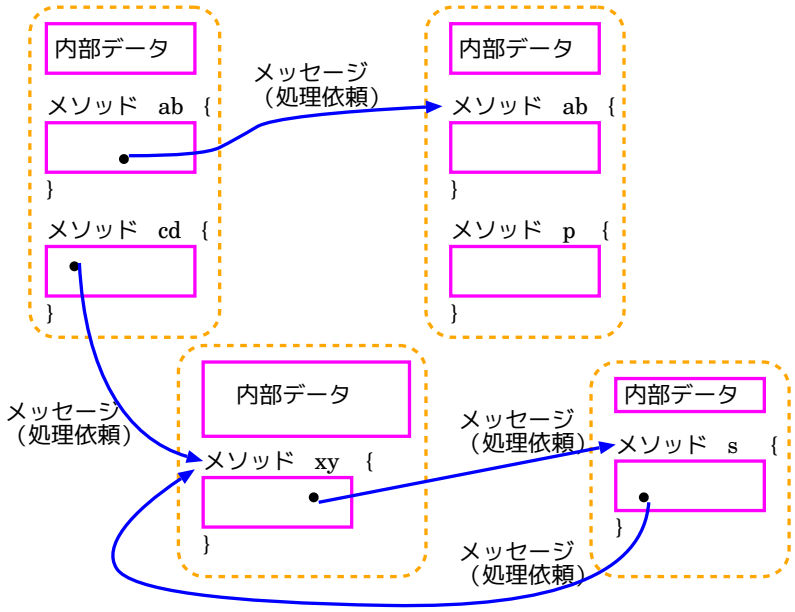
互いに関連したデータ領域，関数／メソッドを1つにまとめたもの

を1つのモジュールと考えれば、

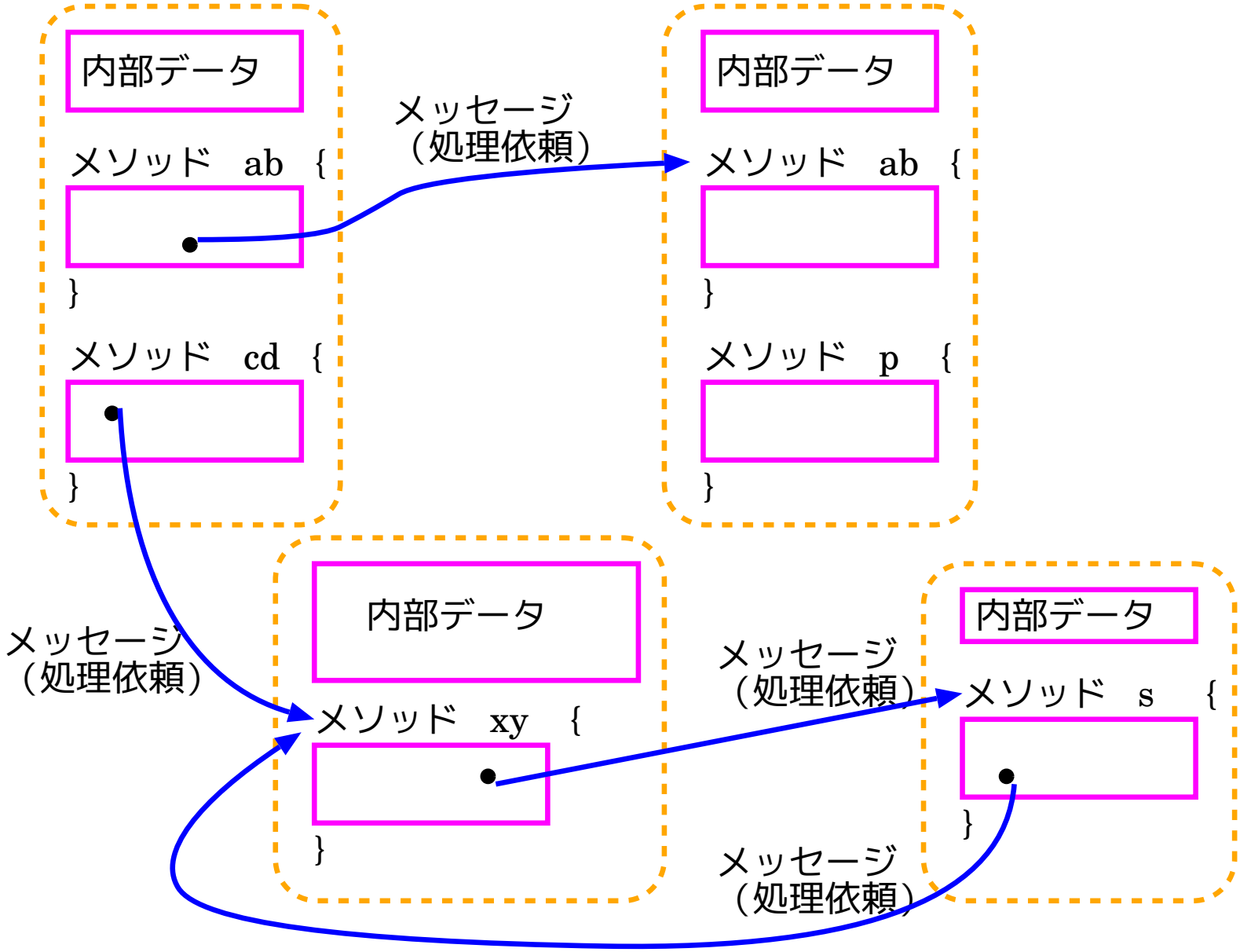
色々なモジュールが連絡（または処理を依頼）し合いながら

全体としての処理を遂行していく、

という図式が浮かび上がってくる。



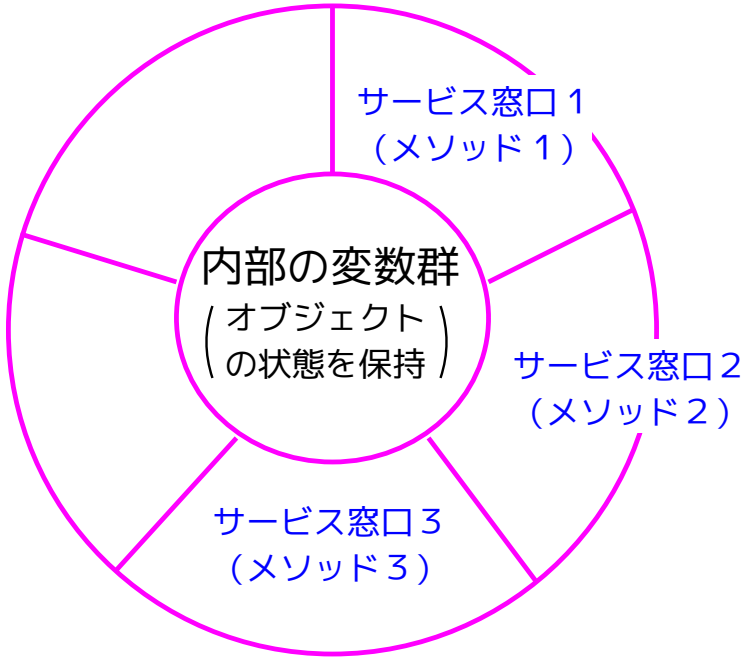
色々なモジュールが連絡（または処理を依頼）し合いながら
全体としての処理を遂行していく、



すなわち、次（オブジェクト）をソフトウェア構成の基本単位と考える。

- 各オブジェクトは、内部に固有の変数群を永続的に持ち、これらの変数領域の状態（値がどうなっているか，内部状態）を参照しながら、どう振舞うかを決める。
- 各オブジェクトの内部状態や内部の実装方式は外部から隠蔽する。その代わり、各オブジェクトには外部にサービスを提供するための充分な窓口（メソッド）を用意し、これらの窓口を通してオブジェクトが外部からの問合せ／処理依頼に応える様にする。

そして、これらのモジュールの間で
問合せ／処理の依頼を繰り返すことにより、
ソフトウェア全体としての処理を進める。



この方式においては、個々のモジュールは

内部のデータをちゃんと自分で保護／管理し

他のモジュールに対するサービス機構（メソッド）も備えている

ので、

外からのサービス依頼に応じて自律的に処理を行う 個体

と考えることができる。

この教科書では 、この様に、

- ◇ カプセル化されたソフトウェア部品（i.e. オブジェクト）を組み合わせてプログラムを構成し、
- ◇ オブジェクトが互いにメッセージを交換し合いながら、全体としての処理を進めてゆく、

といった見方に沿ったプログラム作りのことを、

オブジェクトベースプログラミングと呼ぶ。

⇨ プログラムの動作を

オブジェクト同士の相互作用のシミュレーション

と見做すこともできる。

カプセル化の利点： 情報隠蔽／データのカプセル化を進めオブジェクトの独立性を高めることにより、

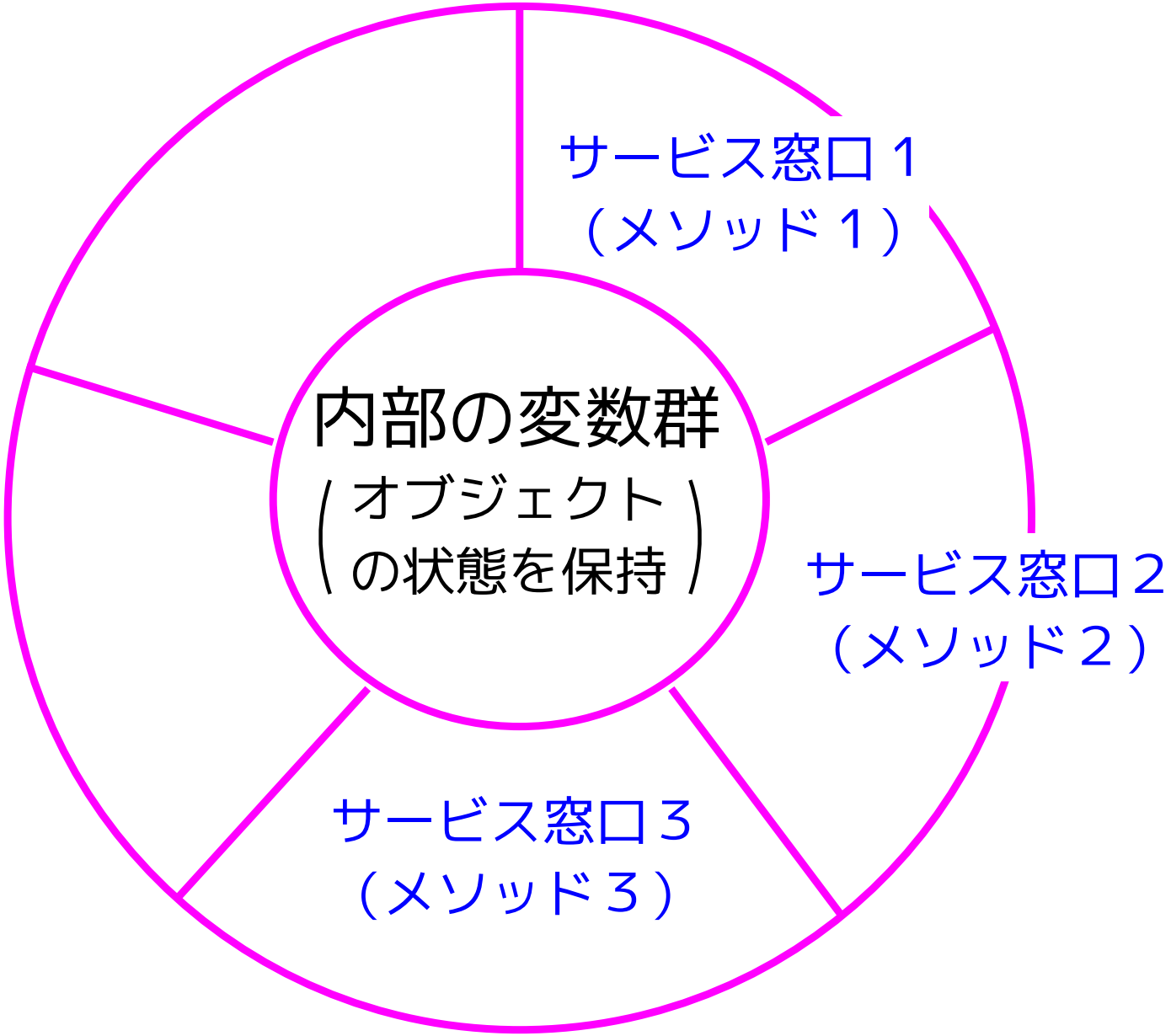
- モジュール性 ··· 他のオブジェクトと切り離して、オブジェクト毎にソースコードを作成・保守することができる。

- 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、他のオブジェクトとは独立に自由にオブジェクトの改良を進めることができる。

- 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。

- 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。

個々のオブジェクトの構造：



19–3 ソフトウェア構成の基本部品となるオブジ

オブジェクトをどんな形で作り出せばよいのか？

例題13.3の様に、
1つのデータ構造(例えばスタック)に関連する記憶領域，関数／メソッド を書き並べて、「これらの集まりがオブジェクトですよ」と宣言する方式も考えられるだろう。

しかし、これでは、例えばスタックが2つ必要な場合、
中身がほとんど同じオブジェクトの定義を、
スタック毎に別々に書き下ろさなければならない。

⇨ オブジェクトを生成するための型枠(テンプレート)を考えればよい。

クラスとインスタンス：

Javaでは、クラスがこの「型枠(テンプレート)」の役割を果たす。

クラス定義の中にオブジェクトを新たに生成するための方法(コンストラクタと呼ぶ）を用意して、生成するオブジェクトの内部変数の値を色々と初期設定することが出来るようになっている。

具体的には、 コンストラクタの呼出し名はクラス名と同じ（そういう約束)であり、new演算子を用いて

new クラス名 (引数の並び)

と書くことによってオブジェクトが生成され、そこへの参照が値として返ってくる。

生成されたオブジェクトを、型枠であるクラスのインスタンスという。

例題13.3で示したCプログラムに相当するものを具合的にJavaでどう書くのか？

例題19．2（スタックを使って文字列を反転する）　例題**13.3**では、

(1) int型データが最大**100**個入るスタックを表す汎用のモジュール stack-int100.c を作成し、

(2) 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Cプログラム stack-reverse-word.c を作成

した。これに対し、ここでは stack-int100.c に相当するオブジェクトをインスタンスとして生成するクラスを定義し、これを利用して例題**13.3**で示されたCプログラムと(ほぼ)同等の処理を行うプログラムを**Java**で構成してみよ。

（考え方）　17.3節の「個々のソースファイルの構成」と題した段落で、

> クラス定義としては、次の2種類がある。
>
> ◇ ソフトウェア部品の定義，
>
> ◇ ソフトウェア部品を利用して目的とする処理を記述したもの

と述べたが、これまで「◇ ソフトウェア部品の定義」を行ったクラス定義は現れて来なかった。

しかし、ここでは、

> stack-int100.cに相当するオブジェクトをインスタンスとして生成する際の型枠として振舞うクラスを定義する

ことが求められている。

この種のクラスを定義する際に注意すべきは、.....

注意点１：

> オブジェクトの中心に位置するのは配列や変数の領域であって、
> メソッドはそれらの領域を扱うための付随物でしかない

実際、Javaにおけるクラス定義はC言語の構造体定義の拡張

- インスタンスは
  C言語の構造体に相当するデータ領域に、関連操作を付加したもの
  という構成
- 個々のメソッドは単独では何の意味もない。

注意点２：

データ領域が何を表すかを反映してクラスの名前を付けるべき

補足（クラス名の付け方、「クラス」，「インスタンス」という用語）：

クラス名が○○○である場合 、
　　クラス定義から作られる
　　どの例（instance）も実際に○○○を表すオブジェクト
になっていると、プログラムが読み易くなる。

「○○○のクラス」と言った場合 、
　　「○○○を表すオブジェクトを生成する際の型枠」
という意味だけでなく、
　　「○○○を表すオブジェクトの集合（class）」
という意味も含まれている。

(プログラミング)

StackOf100Ints... (1)のstack-int100.cに相当するオブジェクトを生成するためのクラス(型枠)

⇨ ソースファイル名は StackOf100Ints.java

ReverseWordMain... (2)のstack-reverse-word.cに相当する処理を行うmain()メソッドを含むクラス

⇨ ソースファイル名は ReverseWordMain.java

補足(main()メソッドを含むクラスの名前):

実行の起点となる場所が明確になれば有難い。

⇨ mainメソッドを含むクラスの名前に Main という文字列を含ませることが多い。

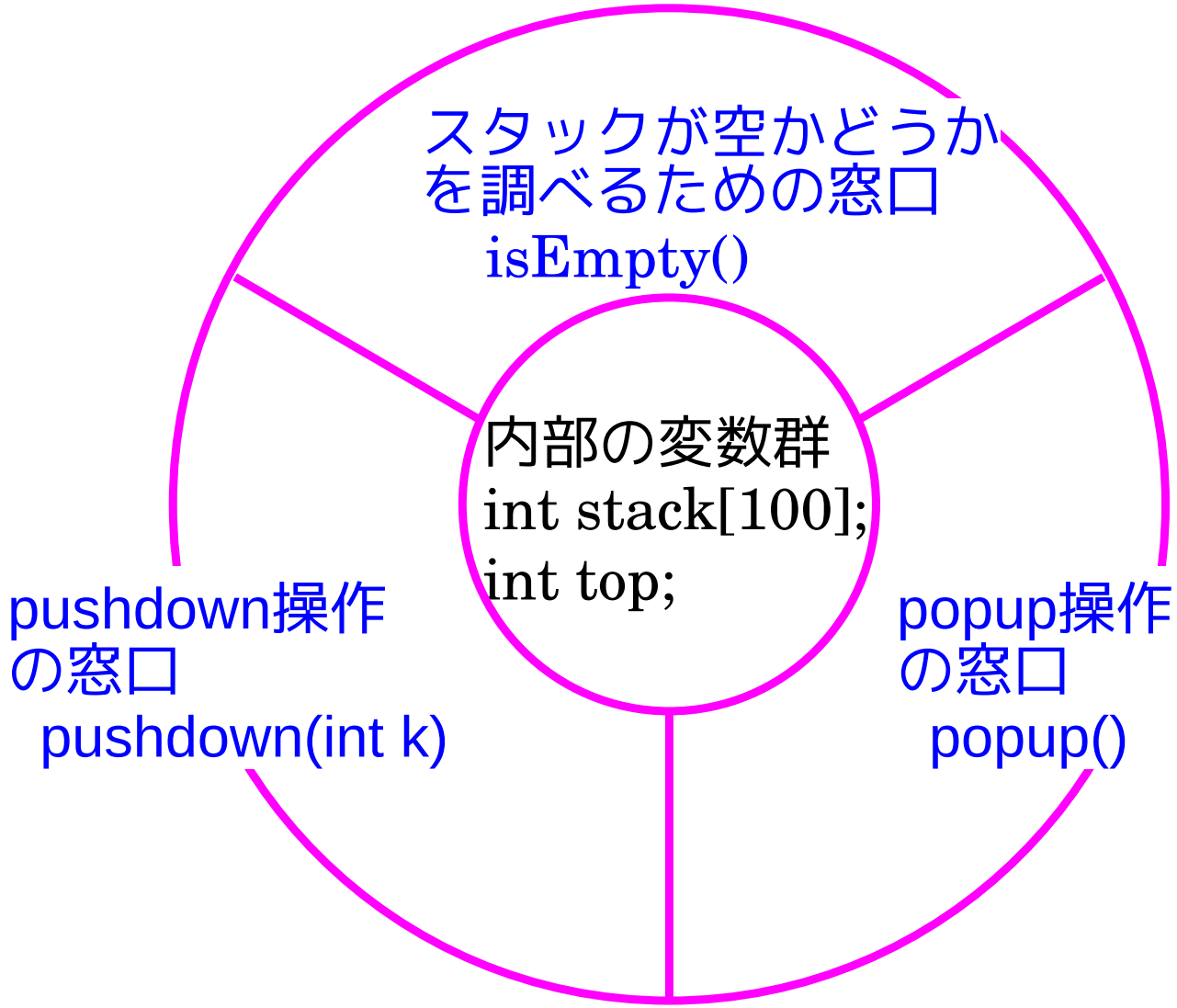
---

stack-int100.cに相当するオブジェクトを生成するための型枠

```
[motoki@x205a]$  cat -n StackOf100Ints.java
     1 /* ------------------------------------------------------
     2  * int型データが最大100個入るスタックのクラス
     3  * ------------------------------------------------------
     4  *    (以下のクラス定義をオブジェクトの型枠(テンプレート)...
     5  *    (用いることにより、                                )
     6  *    (「int型データが最大100個入るスタック」を表すオブジ...
     7  *    (必要に応じて何個でも生成し、利用できる様になる。)
     8  */
     9 public class StackOf100Ints {
    10     private int[] stack = new int[100]; //"private"は..
    11     private int top;                    //で、「外部から.
    12        ↑
```

C言語の static宣言に相当

⇨ 次の様なオブジェクトを随時生成できる様になる。



---

stack-int100.cでは

```
void initialize_stack(void)
{
  top = -1;
}
```

← オブジェクトの初期設定

⇨ Javaでは、コンストラクタに含めるのが妥当

↑

- クラス定義に基づいてインスタンス生成を行う動作主。

または、

- クラス定義内の、
  インスタンスの初期設定をどう行うかを記述した部分。

↓

```
13      //コンストラクタ(オブジェクト生成を申込む窓口,
14      //                    オブジェクトをどう初期設定するか...
15      public StackOf100Ints() {
16          top = -1;
17      }
```

↑ クラス名と同じ名前

```
18
19      //スタックが空かどうかを調べる操作
20      public boolean isEmpty() {
21          if (top < 0)
22              return true;
23          else
24              return false;
25      }
26
27      //pushdown操作
28      public void pushdown(int k) {
29          if (++top >= 100) {
30              System.out.println("stack overflow");
31              System.exit(-1);←C言語の exit(-1) に相当
32          }
33          stack[top] = k;
34      }
```

```
    35
    36      //popup操作
    37      public int popup() {
    38          if (top < 0) {
    39              System.out.println("popup from empty stack'
    40              System.exit(-1);
    41          }
    42          return stack[top--];
    43      }
    44 }
[motoki@x205a]$
```

---

stack-reverse-word.cに相当するJavaプログラム

.....ソフトウェア部品を利用して目的とする処理を行う

```
[motoki@x205a]$  cat -n ReverseWordMain.java
     1 /* ------------------------------------------------------
     2  * StackOf100Intsインスタンスを生成して利用する例
     3  * ------------------------------------------------------
     4  *    文字列を1個読み込み、
     5  *    それをスタックモジュールを用いて反転した後出力する。
     6  */
     7
     8 import java.util.Scanner;
     9                    インスタンス生成を全然想定していないクラス
    10 public class ReverseWordMain {
    11     public static void main(String[] args) {
```

```
12          Scanner inputScanner = new Scanner(System.in);
13
14          System.out.print("Input a string:  ");
15          String word = inputScanner.next();
16          char[] s = new char[word.length()];
17          for (int i=0; i<word.length(); i++)
                                        //String-->char型配列
18              s[i] = word.charAt(i);
```

stack-reverse-word.c内の次の行に対応

```
printf("Input a string:  ");
scanf("%s", s);
```

JavaではString型オブジェクトとchar型配列を区別

⇨ (1) 読み込んだ文字列をString型のオブジェクトとして構成

(2) 読み込んだ文字列の長さを容量とするchar型配列sを確保

(3) 文字列をs上に構成

クラス名がデータ型名となる

↓

```
19
20          StackOf100Ints stack = new StackOf100Ints();
21                                        //スタックオブジェクトを生成
22          for (int i=0; i<word.length(); ++i)
23              stack.pushdown(s[i]);
24          for (int i=0; !stack.isEmpty(); ++i)
25              s[i] = (char)stack.popup();
```

↑

キャスト演算

stack-reverse-word.c内の次の行に対応（文字列反転の作業）

```
    for (i=0; s[i]!='\0'; ++i)
      pushdown(s[i]);
    for (i=0; !is_empty(); ++i)
      s[i] = popup();
```

Stringクラスのコンストラクタを呼び出して、配列sと同じ文字列を保持するString型インスタンスを生成

↓

```
    26          String reversedWord = new String(s);
                                          //char型配列-->String
    27          System.out.println("Reversed string: " + revers
    28      }
    29 }
[motoki@x205a]$
```

コンパイル・実行の様子

```
[motoki@x205a]$  javac StackOf100Ints.java
[motoki@x205a]$  javac ReverseWordMain.java
[motoki@x205a]$  java ReverseWordMain
Input a string:   abc123
Reversed string: 321cba
[motoki@x205a]$
```

クラス定義に関するJava文法の概略を次に示す。

クラス定義の構成：

```
アクセス修飾子  class  クラス名 ..... {
        フィールド宣言;
        .....
        フィールド宣言;

        コンストラクタの宣言;
        .....
        コンストラクタの宣言;

        メソッド宣言;
        .....
        メソッド宣言;
}
```

この中に

クラスから生成されるオブジェクト(i.e.インスタンス)の設計図が記述されることになる。

各々の宣言の役割は次の通り。

- フィールドの宣言 ··· インスタンスが保有する変数領域、あるいは
  クラス自身が保有する変数領域についての記述。

- コンストラクタの宣言 ··· クラス内の設計図に基いてインスタンスを
  生成し、インスタンス内の変数(i.e. フィールド領域)をどう初
  期設定するのかについての記述。

- メソッドの宣言 ··· インスタンスが保有するメソッド、あるいは
  クラス自身が保有するメソッドについての記述。

構文的には、クラス定義はC言語の構造体定義の拡張

なし { コンストラクタ宣言、
メソッド宣言、
クラス自身が保有する変数領域を記述したフィールド宣言

⇨ C言語の構造体定義に相当

クラス定義の中のフィールド，メソッドの内、どれがインスタンスのものなのか?:

- static宣言された フィールド，メソッド はクラス全体に共通のもので、それぞれ クラスフィールド，クラスメソッド と呼ばれる。

  クラスフィールドは、 クラスに固有のデータ（e.g. それまでに生成したインスタンスの個数）を記憶するためのものである。

  クラスフィールドへのアクセスは

  クラス名 . クラスフィールドの名前 ,

  クラスメソッドの呼び出しは

  クラス名 . クラスメソッド名 (引数の並び)

  と書く。

- static宣言 されていない フィールド，メソッド はインスタンスのもので、それぞれインスタンスフィールド，インスタンスメソッド と呼ばれる。

  インスタンスフィールドは、 個々のインスタンスに固有のデータ を記憶するためのもので、インスタンス毎に1個用意されます。

  インスタンスフィールドへのアクセスは

  インスタンスを参照する変数の名前 . インスタンスフィールドの名前

  インスタンスメソッドの呼び出しは

  インスタンスを参照する変数の名前

  . インスタンスメソッド名 (引数の並び)

  と書く。

例19．3（static修飾子を多用すると...；好ましくないクラス定義の例

例題13.3... (1)int型データが最大100個入るスタックを表す
汎用のモジュール stack-int100.c を作成し、
(2) 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
Cプログラム stack-reverse-word.c を作成

例題19.2... 例題13.3とほぼ同等の処理を行うプログラムをJavaで

しかし、
stack-int100.c 自体が型枠でなくオブジェクト

⇨ 次のJavaプログラムの方が例題19.2で示したものよりも
例題13.3のCプログラムに近い

```
[motoki@x205a]$ cat -n StackOf100Ints_static.java
     1 /* ----------------------------------------
     2  * int型データが最大100個入るスタックのクラス
     3  * ----------------------------------------
     4  *   (staticな構成要素だけから成るので、クラス自体が1...
     5  *   (それゆえ、このクラス定義を用いても、
     6  *   (独立なスタックインスタンスを複数生成することは出...
     7  */
     8 public class StackOf100Ints_static {
     9     private static int[] stack = new int[100]; //privat
    10     private static int top;                    //外部か
    11
    12     //スタックを空に初期設定する操作
    13     public static void initializeStack() {
    14         top = -1;
    15     }
    16
```

```
17      //スタックが空かどうかを調べる操作
18      public static boolean isEmpty() {
19          if (top < 0)
20              return true;
21          else
22              return false;
23      }
24
25      //pushdown操作
26      public static void pushdown(int k) {
27          if (++top >= 100) {
28              System.out.println("stack overflow");
29              System.exit(-1);
30          }
31          stack[top] = k;
32      }
33
```

```
34      //popup操作
35      public static int popup() {
36          if (top < 0) {
37              System.out.println("popup from empty stack'
38              System.exit(-1);
39          }
40          return stack[top--];
41      }
42  }
[motoki@x205a]$ cat -n ReverseWordMain_usingStaticStack.java
 1  /* ----------------------------------------------------
 2   * StackOf100Ints_staticクラス自体をモジュールとして利用

 3   * ----------------------------------------------------
 4   *    文字列を1個読み込み、
 5   *    それをスタックモジュールを用いて反転した後出力する。
 6   */
```

```
 7
 8 import java.util.Scanner;
 9
10 public class ReverseWordMain_usingStaticStack {
11     public static void main(String[] args) {
12         Scanner inputScanner = new Scanner(System.in);
13
14         System.out.print("Input a string:  ");
15         String word = inputScanner.next();
16         char[] s = new char[word.length()];
17         for (int i=0; i<word.length(); i++)
                                        //String-->char型配列
18             s[i] = word.charAt(i);
19
20         StackOf100Ints_static.initializeStack();
                                        //スタックを空に初期設定
21         for (int i=0; i<word.length(); ++i)
```

```
22              StackOf100Ints_static.pushdown(s[i]);
23          for (int i=0; !StackOf100Ints_static.isEmpty();
24              s[i] = (char)StackOf100Ints_static.popup();
25          String reversedWord = new String(s);
                                        //char型配列-->String
26          System.out.println("Reversed string: " + revers
27      }
28 }
[motoki@x205a]$  javac StackOf100Ints_static.java
[motoki@x205a]$  javac ReverseWordMain_usingStaticStack.java
[motoki@x205a]$  java ReverseWordMain_usingStaticStack
Input a string:   abc123
Reversed string: 321cba
[motoki@x205a]$
```

注目点

上記のクラス StackOf100Ints_static については 、
どの構成要素にもstatic修飾子

⇨ クラスの共通要素として(インスタンスを生成せずとも)最初から存在し使える状態にある。

⇨ クラス自体が1つのオブジェクトとして機能し、

```
StackOf100Ints_static.initializeStack(),
StackOf100Ints_static.isEmpty(),
StackOf100Ints_static.pushdown(引数),
StackOf100Ints_static.popup()
```

といった書き方もできる。

考察(クラスの別の利用方法)

構文上はStackOf100Ints_staticはクラス

⇨ ●コンストラクタを呼び出してインスタンスを生成することも可能

●インスタンスから各種staticメソッドを利用することも可能

⇨ 次の様なJavaプログラムも可能

```
[motoki@x205a]$ cat -n ReverseWordMain_usingStaticStack2.java

     1 /* ------------------------------------------------------
     2  * StackOf100Ints_staticクラス自体をモジュールとして利用

     3  * ------------------------------------------------------
     4  *    文字列を1個読み込み、
     5  *    それをスタックモジュールを用いて反転した後出力する。
```

```
 6  */
 7
 8 import java.util.Scanner;
 9
10 public class ReverseWordMain_usingStaticStack2 {
11     public static void main(String[] args) {
12         Scanner inputScanner = new Scanner(System.in);
13
14         System.out.print("Input a string:  ");
15         String word = inputScanner.next();
16         char[] s = new char[word.length()];
17         for (int i=0; i<word.length(); i++)
                                        //String-->char型配列
18             s[i] = word.charAt(i);
19
```

```
20          StackOf100Ints_static
                        stack = new StackOf100Ints_static();
21          stack.initializeStack();
22          for (int i=0; i<word.length(); ++i)
23              stack.pushdown(s[i]);
24          for (int i=0; !stack.isEmpty(); ++i)
25              s[i] = (char)stack.popup();
26          String reversedWord = new String(s);
                                    //char型配列-->String
27          System.out.println("Reversed string: " + revers
28      }
29 }
[motoki@x205a]$ javac ReverseWordMain_usingStaticStack2.java
[motoki@x205a]$ java ReverseWordMain_usingStaticStack2
Input a string:   abc123
Reversed string: 321cba
[motoki@x205a]$
```

注目点

以上の様にStackOf100Ints_staticは「stack-int100.cに相当するオブジェクトをインスタンスとして生成するクラス」

⇨ 例題**19.2**で 要求された条件を満たす様にも思える。

しかし、

StackOf100Ints_staticクラスで複数のインスタンスを生成しても、

- それらはStackOf100Ints_staticクラス自体の表すオブジェクトへの橋渡しをするだけのもの
- 本質的には全て同一のスタックオブジェクトになってしまう。

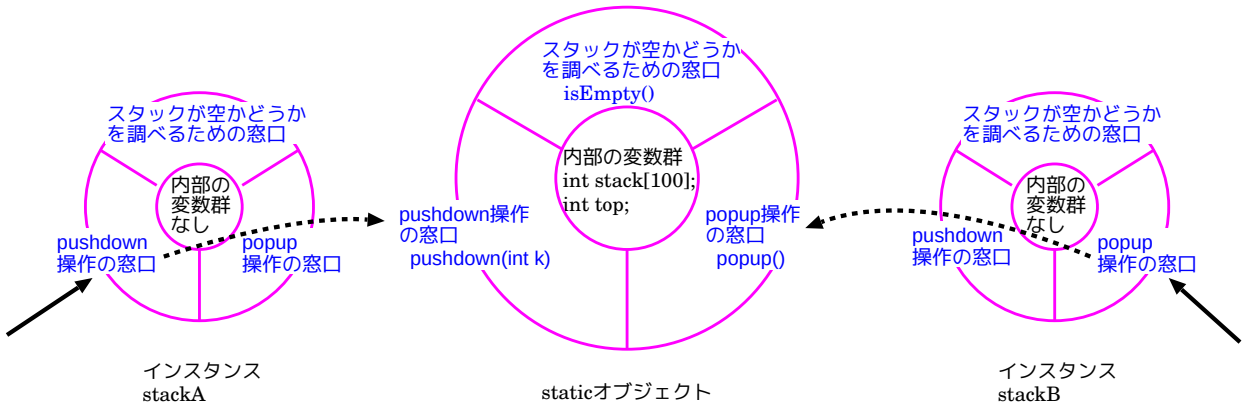
⇧

次の**Java**プログラムの実行結果からも分かる。

```
[motoki@x205a]$  cat -n UseStaticStackMain.java
     1 // StackOf100Ints_staticクラスでは
     2 // 独立なスタックインスタンスを複数生成することが出来ない..
     3 public class UseStaticStackMain {
     4     public static void main(String[] args) {
     5         StackOf100Ints_static
                       stackA = new StackOf100Ints_static();
     6         StackOf100Ints_static
                       stackB = new StackOf100Ints_static();
     7         stackA.initializeStack();
     8         stackB.initializeStack();
```

```
 9          stackA.pushdown(1);
10          System.out.println("stackA.pushdown(1);");
11          stackA.pushdown(333);
12          System.out.println("stackA.pushdown(333);");
13          System.out.println(
                "==> stackB.popup()=" + stackB.popup());
14          System.out.println(
                "    stackB.popup()=" + stackB.popup());
15      }
16  }
```

```
[motoki@x205a]$  javac UseStaticStackMain.java
[motoki@x205a]$  java UseStaticStackMain
stackA.pushdown(1);
stackA.pushdown(333);
==> stackB.popup()=333
    stackB.popup()=1
[motoki@x205a]$
```



クラスを定義し、それを基にインスタンスを生成して利用する典型例

例題19．4（長方形を表すオブジェクトのクラス）　クラス定義の典型的な例として、

長方形の幅と高さのデータを保持し、関連するメソッドとして

- (自オブジェクトを説明する)標準的な文字列表現を返すメソッドと
- 面積を返すメソッド

を備えたオブジェクト

のクラス `Rectangle` を定義せよ。そして、 このクラスを利用して

①標準入力から得られたデータを基に
　`Rectangle`インスタンスを3つ生成し

②その中で最大面積の`Rectangle`インスタンスの情報を出力する

Javaプログラムを作成せよ。

（考え方）

汎用性のあるものが望ましい

⇨ ◇個々のRectangleインスタンスには固有のid番号を保持させる

◇id番号の一意性を保証するために最新のid番号を常に保持する

領域をRectangleのクラスフィールドとして用意

十分に情報隠蔽も考慮すべき

⇨ ◇個々のフィールドは外部から直接アクセスできない様にし、

◇外部からの正当な利用ができる様に参照や値変更のメソッドを用意

（プログラミング）

MaxAmong3RectanglesMain.java... Rectangleクラスを利用して指定された作業を行う

Rectangle.java...Rectangleクラスを定義

```
[motoki@x205a]$  cat -n MaxAmong3RectanglesMain.java
     1 /* 次の作業を順に行うJavaプログラム
     2 /* (1)標準入力から得られたデータを基にRectangleイン... */
     3 /* (2)面積最大のインスタンスを見つけてその情報を出力
     4
     5 import java.util.*;
     6
     7 class MaxAmong3RectanglesMain {
     8     public static void main(String[] args) {
     9         Rectangle[] rectangle = new Rectangle[3];
    10         int    indexOfMaxArea;
    11         double width, height, area, maxArea;
    12         Scanner inputScanner = new Scanner(System.in);
    13
```

```
14          //標準入力からのデータを入力、それを基に長方...
15          for (int i=0; i<3; i++) {
16              System.out.print("長方形の幅と高さを入力: ")
17              width  = inputScanner.nextDouble();
18              height = inputScanner.nextDouble();
19              rectangle[i] = new Rectangle(width, height)
20          }
21
22          //生成された長方形インスタンスを出力
23          System.out.println();
24          System.out.println("標準入力からのデータを" +
                                   "基に生成されたインスタンス");
25          for (int i=0; i<3; i++) {
26              System.out.println(rectangle[i]);
27          }
28
```

```
    29          //面積最大の長方形インスタンスを見つけて出力
    30          indexOfMaxArea = 0;
    31          maxArea        = rectangle[0].getArea();
    32          for (int i=1; i<3; i++) {
    33              area = rectangle[i].getArea();
    34              if (area > maxArea) {
    35                  indexOfMaxArea = i;
    36                  maxArea        = area;
    37              }
    38          }
    39          System.out.println("==>面積最大のものは" +
                                       rectangle[indexOfMaxArea]);
    40      }
    41 }
[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...
 5     protected double width;    //長方形の幅
 6     protected double height;   //長方形の高さ
 7
 8     private static int numberOfRectangles = 0;
                          //生成した長方形インスタンスの個数
 9
10     //コンストラクタ
11     public Rectangle() {
12         this(1.0, 1.0);
13     }
14
15     public Rectangle(double width, double height) {
16         this.id     = ++numberOfRectangles;
17         this.width  = width;
18         this.height = height;
```

```
19      }
20
21      //長方形インスタンスの標準的な文字列表現を定める
22      @Override
23      public String toString() {
24          return "rectangle(id=" + id
                    + ", width=" + width
25                  + ", height=" + height + ")";
26      }
27
28      //ゲッターメソッド
29      public int getId() {
30          return id;
31      }
32
33      public double getWidth() {
34          return width;
```

```
35      }
36
37      public double getHeight() {
38          return height;
39      }
40
41      public static int getNumberOfRectangles() {
42          return numberOfRectangles;
43      }
44
45      //セッターメソッド
46      public void setWidth(double width) {
47          this.width = width;
48      }
49
50      public void setHeight(double height) {
51          this.height = height;
```

```
    52      }
    53
    54      //長方形インスタンスの面積を計算して返す
    55      public double getArea() {
    56          return width*height;
    57      }
    58 }
[motoki@x205a]$  javac MaxAmong3RectanglesMain.java
[motoki@x205a]$  java MaxAmong3RectanglesMain
長方形の幅と高さを入力:  3  17
長方形の幅と高さを入力:  10 10
長方形の幅と高さを入力:  15  5

標準入力からのデータを基に生成されたインスタンス
rectangle(id=1, width=3.0, height=17.0)
rectangle(id=2, width=10.0, height=10.0)
rectangle(id=3, width=15.0, height=5.0)
```

```
==>面積最大のものはrectangle(id=2, width=10.0, height=10.0)
[motoki@x205a]$
```

19–4 文法のまとめ クラス定義の形式

オブジェクト指向プログラミングにおいて、

クラス . . . ソフトウェアの構成部品であるオブジェクトを
（必要に応じて何個でも）作り出すための「工場」に相当

インスタンス . . . クラスの中に保存された設計図に基づいて
作り出されたソフトウェア部品

クラス定義の形式(通常):

```
アクセス修飾子  class  クラス名 ..... {
        フィールド宣言;
        .....
        フィールド宣言;

        コンストラクタの宣言;
        .....
        コンストラクタの宣言;

        メソッド宣言;
        .....
        メソッド宣言;
}
```

ここで、

- アクセス修飾子 の部分は クラスへのアクセス許可の範囲を指示
  - ◇ public... アクセス権限を全然行わない場合
  - ◇ 何も書かない... 利用を同一のパッケージに限定したい場合
    (i.e. 関連したクラスを集めたもの)

```
アクセス修飾子  class  クラス名 ..... {
        フィールド宣言;
        ......
        フィールド宣言;

        コンストラクタの宣言;
        ......
        コンストラクタの宣言;

        メソッド宣言;
        ......
        メソッド宣言;
}
```

- クラス全体を管理するための宣言の前には　static という修飾子
  ⇨　残りのフィールド、メソッドの部分を構成要素として
  インスタンスが作り出される。

```
アクセス修飾子  class  クラス名 ..... {
    フィールド宣言;
    .....
    メソッド宣言;
    .....
}
```

- 生成された各々のインスタンスの中では 、
  staticの付かないフィールドに対応した変数領域が確保され、ここにインスタンスの内部状態が保持されることになる。これらをインスタンスフィールド，非staticフィールド，またはインスタンス変数と呼ぶ。

  一方、 クラス全体を管理するために 、
  staticの付いたフィールド毎に対応する変数領域が1個ずつ確保され、ここにクラスに共通の情報が保持される。これらをクラスフィールド，staticフィールド，あるいはクラス変数と呼ぶ。

  また、staticの付かないメソッドをインスタンスメソッドと呼び、staticの付くメソッドをクラスメソッドと呼ぶ。

インスタンスメソッドから

. . . クラス変数やクラスメソッドへのアクセスは自由

クラスメソッドから

. . . インスタンス変数やインスタンスメソッドにアクセスするには
どのインスタンスに所属のものかを明示

フィールド宣言，メソッド宣言の形式：

- 各々のフィールド宣言は C言語の変数宣言の様なもので、例えばクラスフィールドの場合は次の様に記述される。

  アクセス修飾子 `static` データ型 フィールド名の列;

  フィールド宣言の場所に初期値を書くこともできる。これもC言語における変数の初期値指定の仕方と同様で、例えば、

```
double width = 1.0;
```

- 各々のメソッド宣言は C言語の関数定義の様なもので、例えばクラスメソッドの場合は次の様な形で記述される。

  アクセス修飾子 `static` 戻り値の型 メソッド名(引数列) {

```
  ............
}
```

- フィールド名やメソッド名の前に付ける[アクセス修飾子]としては次の4つが可能である。

アクセス修飾子	利用可能な範囲
`private`	クラス内
（修飾子なし）	同一パッケージ内
`protected`	同一パッケージ内，およびサブクラス
`public`	（無制限）

- メソッド名が同じでも、引数の個数や引数の型のリストが違えば異なるメソッドとして識別される。(メソッドの多重定義という。)
  ⇨ メソッド定義の頭部の中で
  メソッド名(第1引数の型，第2引数の型，...)
  という部分は定義したメソッドの適用範囲を示すもので、これをメソッドのシグニチャという。

- toString()というインスタンスメソッドを用意しておけば 、
  文字列を指定する場所にインスタンス(への参照)を書く(e.g. System.out.println(r)) だけで、自動的にtoString()メソッドが呼びだされ標準的な文字列表現に変換された上で処理される。

コンストラクタ宣言の形式：

- コンストラクタ はインスタンスを生成·初期化する時に呼び出されるもので、次の様な形で記述される。

```
アクセス修飾子 クラス名(引数列) {
    ............
}
```

宣言の仕方は、メソッドの場合と同様であるが、次の3点で異なる。

◇ staticは付けない。　（冗長な修飾子は付けない。）

◇ 戻り値の型は指定しない。

◇ メソッド名に相当する所にはクラス名を書く。

- クラス定義の中にコンストラクタの記述が全然無い場合 、
次の様な引数なしのコンストラクタ (デフォルトコンストラクタという) がコンパイラによって自動的に生成され追加される。

```
クラス名() {
    super();
}
```

- メソッドの場合と同様に、呼出し名が同じでも、引数の個数や引数の型のリストが違えば異なるコンストラクタとして識別される。(コンストラクタの多重定義という。)

インスタンスの生成・初期設定：

{K.Arnold 他「プログラミング言語 Java 第 4 版」2.5.1 節 }

次の順に行われる。

① オブジェクトの生成。

② データ型毎に決められた デフォルトの初期値 が内部の各々のインスタンス変数に割り当てられる。

③ フィールド宣言の部分等に初期値指定 の記述があれば、その設定が為される。

④ コンストラクタ が呼び出され必要な設定が為される。

(注意 ：これとは対照的に、
局所変数については、指定外の初期設定が行われることはない。)

クラス型変数の宣言： `int` や `double` といった基本データ型と同様に、個々のクラスはデータ型として扱われ、

クラス名 `v ;`

という変数宣言によって、`v` は クラス名 という名前のクラスのインスタンス（への参照）を保持できる様になる。

19–5 何をクラスとして定義すべきか？

一般的には 、問題文中の「名詞」に注目し、

- 汎用性のあるオブジェクトに繋がりそうなもの、あるいは

- 特殊だけれどもオブジェクト化するとプログラム全体が
  互いに独立な部分に綺麗に分割されそうなもの

を見つけ出し、
それらをソフトウェア部品として扱うためのクラスを定義する。

例題19．5（複素多項式の計算）　非負整数データ $\mathbf{n}$ と複素数データ $\mathbf{C_n}, \mathbf{C_{n-1}}, \ldots, \mathbf{C_0}$ を読み込み、これらの定数値の下で複素多項式

$$\mathbf{C_n z^n + C_{n-1} z^{n-1} + C_{n-2} z^{n-2} + \cdots + C_1 z + C_0}$$

の値が

$$\mathbf{z = e^{i\pi k/5} = \cos\frac{\pi k}{5} + i\ \sin\frac{\pi k}{5}} \qquad \mathbf{(k=0,1,2,3,\ \ldots,\ 9)}$$

のそれぞれの値に対してどの様に変化するかを、表の形に見易く出力するJavaプログラムを作成せよ。

（考え方）　...例題11.7で作成したCプログラムのJava版を作れ、

この問題の場合も、

　　どういうクラスを用意するかを決めるために

　　問題文中に現れる「名詞」に注目する、

ということが有効

この問題の場合は、

「複素数」$\longrightarrow$ クラス CompexNumber

「複素多項式」$\longrightarrow$ クラス CompexPolynomial

(プログラミング)

CalcCompexPolynomialMain.java ... 他クラスを利用して指定された作業を行う

CompexNumber.java ... 複素数のクラスを定義

CompexPolynomial.java ... 複素多項式のクラスを定義

```
[motoki@x205a]$ cat -n CalcComplexPolynomialMain.java
     1 /* 非負整数データ n と複素数データ C_n, C_n-1, ..., C_0 を
     2 /* これらの定数値の下で複素多項式
     3 /*          C_n*z^n + C_n-1*z^n-1 + ... + C_1*z + C_0
     4 /* の値が
     5 /*          z = exp(i π k/5)   (k=0,1,2, ..., 9)
     6 /* のそれぞれの値に対してどの様に変化するかを、表の形に見..
     7 /* Javaプログラム
     8
     9 import java.util.*;
    10
```

```
11 class CalcComplexPolynomialMain {
12     private static final int MAX_DEGREE=100;
13
14     public static void main(String[] args) {
15         int degree;
16         ComplexNumber[] coeff =
                           new ComplexNumber[MAX_DEGREE+1];
17         Scanner inputScanner = new Scanner(System.in);
18
```

```
19          //標準入力からのデータを入力して
                                    複素多項式オブジェクトを構成
20          System.out.print("複素多項式の次数(100以下):");
21          degree = inputScanner.nextInt();
22          for (int i=degree; i>=0; --i) {
23              System.out.printf("  %d次係数の実部と虚部:",
24              coeff[i] = new ComplexNumber(
                                inputScanner.nextDouble(),
25                              inputScanner.nextDouble());
26          }
27          ComplexPolynomial polynomial =
                        new ComplexPolynomial(degree, coeff);
28
29          //構成された多項式オブジェクトを出力(確認のため)
30          System.out.printf("%n構成された多項式：%n");
31          System.out.println(polynomial);
32
```

```
33          //多項式の値を計算して出力
34          System.out.printf("%n" +
35              " k  %16sz%15s  c[d]*z^d+c[d-1]*z^(d-1)+ ..
36              "--  ------------------------------" +
37              "  ------------------------------------
38          for (int k=0; k<10; ++k) {
39              ComplexNumber z = new
                  ComplexNumber(Math.cos(Math.PI*k/5.0),
40                                Math.sin(Math.PI*k/5.0));
41              ComplexNumber result =
                                  polynomial.calculate(z);
42              System.out.printf("%2d  (%13.6e)+(%13.6e)i
43                                k, z.getRealPart(),
                                  z.getImaginaryPart(),
44                                result.getRealPart(),
                                  result.getImaginaryPart());
45          }
```

```
    46     }
    47 }
[motoki@x205a]$  cat -n ComplexNumber.java
     1 /* 複素数を表すオブジェクトのクラス */
     2
     3 public class ComplexNumber {
     4     private final double re;  //実部
     5     private final double im;  //虚部
     6
     7     //コンストラクタ
     8     public ComplexNumber(double re, double im) {
     9         this.re = re;
    10         this.im = im;
    11     }
    12
    13     //複素数インスタンスの標準的な文字列表現を定める
    14     @Override
```

```
15      public String toString() {
16          return "("+ re + ")+(" + im + ")i";
17      }
18
19      //ゲッターメソッド
20      public double getRealPart() {
21          return re;
22      }
23
24      public double getImaginaryPart() {
25          return im;
26      }
27
28      //四則演算
29      public ComplexNumber addBy(ComplexNumber c) {
30          return new ComplexNumber(re+c.re, im+c.im);
31      }
```

```
32
33      public ComplexNumber subtractBy(ComplexNumber c) {
34          return new ComplexNumber(re-c.re, im-c.im);
35      }
36
37      public ComplexNumber multiplyBy(ComplexNumber c) {
38          return new ComplexNumber(re*c.re - im*c.im,
39                                   re*c.im + im*c.re);
40      }
41
42      public ComplexNumber divideBy(ComplexNumber c) {
43          double tmp = c.re*c.re + c.im*c.im;
44          return new ComplexNumber(
                                 (re*c.re + im*c.im) / tmp,
45                               (im*c.re - re*c.im) / tmp);
46      }
47  }
```

```
[motoki@x205a]$  cat -n ComplexPolynomial.java
     1 /* 複素多項式を表すオブジェクトのクラス */
     2
     3 public class ComplexPolynomial {
     4     private int degree;                  //次数
     5     private ComplexNumber[] coeff;  //係数
     6
     7     //コンストラクタ
     8     public ComplexPolynomial(int degree,
                                    ComplexNumber[] coeff) {
     9         this.degree = degree;
    10         this.coeff  = new ComplexNumber[degree+1];
    11         System.arraycopy(coeff, 0, this.coeff, 0,
                                                  degree+1);
    12     }
    13
```

```
14      //複素多項式インスタンスの標準的な文字列表現を定める
15      @Override
16      public String toString() {
17          String result =
                    "complex polynomial of degree " + degree
18                  + " and coefficients\n  c[" + degree
19                  + "] = " + coeff[degree];
20          for (int i=degree-1; i>=0; i--)
21              result += ",\n  c[" + i + "] = "
                            + coeff[i];
22          return result;
23      }
24
```

```
25      //変数値を与えて複素多項式の値を計算
26      public ComplexNumber calculate(ComplexNumber z) {
27          ComplexNumber result = coeff[degree];
28          for (int i=degree-1; i>=0; i--)
29              result = (result.multiplyBy(z))
                          .addBy(coeff[i]);
30          return result;
31      }
32 }
[motoki@x205a]$  javac CalcComplexPolynomialMain.java
[motoki@x205a]$  java CalcComplexPolynomialMain
複素多項式の次数(100以下)： 3
  3次係数の実部と虚部： 1.0 -2.0
  2次係数の実部と虚部： -3.0 4.0
  1次係数の実部と虚部： 5.0 -6.0
  0次係数の実部と虚部： -7.0 8.0
```

構成された多項式：

```
complex polynomial of degree 3 and coefficients
  c[3] = (1.0)+(-2.0)i,
  c[2] = (-3.0)+(4.0)i,
  c[1] = (5.0)+(-6.0)i,
  c[0] = (-7.0)+(8.0)i
```

k	z	c[d]*z^d+c[d-1]*z^(d-1)+
0	(1.000000e+00)+(0.000000e+00)i	(-4.000000e+00)+(4.000
1	(8.090170e-01)+(5.877853e-01)i	(-2.566385e+00)+(6.036
2	(3.090170e-01)+(9.510565e-01)i	(-1.657253e+00)+(6.932
3	(-3.090170e-01)+(9.510565e-01)i	(1.572893e+00)+(1.093
4	(-8.090170e-01)+(5.877853e-01)i	(-2.430068e+00)+(2.021
5	(-1.000000e+00)+(1.224647e-16)i	(-1.600000e+01)+(2.000
6	(-8.090170e-01)+(-5.877853e-01)i	(-2.089617e+01)+(6.728
7	(-3.090170e-01)+(-9.510565e-01)i	(-1.219093e+01)+(-9.308

```
 8  ( 3.090170e-01)+(-9.510565e-01)i  (-6.016509e+00)+( 2.1237
 9  ( 8.090170e-01)+(-5.877853e-01)i  (-5.815581e+00)+( 3.9631
[motoki@x205a]$
```

例題19．6（自習 push-downスタック）　例題13.3ではC言語を用いてint型データが最大100個入るpush-downスタックを表す汎用モジュールstack-int100.c を実装し、例題19.2では ..... をJavaで実装した。　これに対して、ここでは、
◇ 容量をインスタンス生成時に指定 でき、また
◇ 任意のオブジェクト (への参照情報) を要素として格納できる
様に、push-downスタックオブジェクトのクラスをJavaで実装せよ。

（考え方）

- どのオブジェクトもObjectインスタンスの一種

⇨ Objectインスタンスをスタックの要素として格納する様にすれば、
「任意のオブジェクトを要素として格納できる」

- 配列を使って実装

Javaの場合◇スタックの初期容量の指定もインスタンス生成時で良い
◇java.util.Arrays.copyOf というメソッドを利用すれば
容量不足による強制終了を避けることも出来る。

(プログラミング)

- 汎用のソフトウェア部品のクラスの単体での動作テストは「JUnit」で行うのが良いとされているが、
- ここではmainメソッドを用意してを行う

```
[motoki@x205a]$  cat -n StackOfAnyObjects.java
     1 /* Objectインスタンスを格納するpushdownスタック... */
     2
     3 import java.util.*;
     4
     5 public class StackOfAnyObjects {
     6     private static final int
                         DEFAULT_INITIAL_CAPACITY = 100;
     7     private static final int
                         DEFAULT_CAPACITY_INCREMENT = 100;
     8
     9     private Object[] stack;
```

```
10      private int indexOfTopEle;
11
12      //コンストラクタ
13      public StackOfAnyObjects() {
14          this(DEFAULT_INITIAL_CAPACITY);
15      }
16
17      public StackOfAnyObjects(int initialCapacity) {
18          stack = new Object[initialCapacity];
19          indexOfTopEle = -1;
20      }
21
22      //pushdownスタックオブジェクトの
                                    標準的な文字列表現を定める
23      @Override
```

```
24      public String toString() {
25          return "pushdownStack (type=Object, capacity="
26                  + stack.length + ", currentNumOfEele="
27                  + (indexOfTopEle+1) + ")";
28      }
29
30      //スタックの詳細情報を得る
31      public String getDetailedConfig() {
32          String result = "stack contains "
                    + (indexOfTopEle+1) + " elements: {";
33          for (int i=0; i<indexOfTopEle; ++i)
34              result += "\n    " + stack[i] + ",";
35          if (indexOfTopEle >= 0)
36              result += "\n    " + stack[indexOfTopEle];
37          result += " }";
38          return result;
39      }
```

```
40
41      //スタックが空かどうか調べる
42      public boolean isEmpty() {
43          return indexOfTopEle == -1;
44      }
45
46      //pushdown操作
47      public void pushdown(Object element) {
48          if (indexOfTopEle+1 == stack.length) {
49              stack = Arrays.copyOf(stack, stack.length
50                      + DEFAULT_CAPACITY_INCREMENT);
51              System.out.printf(
                  "###Stack capacity is increased###%n" +
52                  "#<New> %s%n", this);
53          }
54          stack[++indexOfTopEle] = element;
55      }
```

```
56
57      //popup操作
58      public Object popup() {
59          if (indexOfTopEle < 0)
60              throw new EmptyStackException();
61          Object element = stack[indexOfTopEle];
62          stack[indexOfTopEle--] = null;
                                  //取り出した要素への参照を解除
63          return element;
64      }
65
66      //スタックに格納された要素の個数
67      public int getNumOfEle() {
68          return indexOfTopEle+1;
69      }
70
71      //スタックのtop要素をのぞき見
```

```
72      public Object peepTop() {
73          if (isEmpty())
74              return null;
75          else
76              return stack[indexOfTopEle];
77      }
78
79      //スタックの指定要素をのぞき見
80      public Object peepEleOfIndex(int index) {
81          return stack[index];
82      }
83
84      //--------------------単体での動作テスト用----------
85      public static void main(String[] args) {
86          StackOfAnyObjects stack1 =
                new StackOfAnyObjects(2);
87          System.out.println(
```

```
                "stack1: " + stack1.getDetailedConfig());
88          System.out.println(
                "|実行: stack1.pushdown(\"a\");");
89          stack1.pushdown("a");
90          System.out.println(
                "|実行: stack1.pushdown(new Integer(2));");
91          stack1.pushdown(new Integer(2));
92          System.out.println(
                "|実行: stack1.pushdown(new int[] {3, 4, 5}
93          stack1.pushdown(new int[] {3, 4, 5});
94          System.out.println(
                "|実行: stack1.pushdown(new StackOfAnyObjec
95          stack1.pushdown(new StackOfAnyObjects());
96          System.out.println(
                "==>stack1: " + stack1.getDetailedConfig()
97          System.out.println(
                "|実行: StackOfAnyObjects stack2 = "
```

```
 98                 + "(StackOfAnyObjects) stack1.popup();");
 99             StackOfAnyObjects stack2 =
                     (StackOfAnyObjects) stack1.popup();
100             System.out.println(
                    "==>((int[])stack1.peepTop())[0])="
101                 + ((int[])stack1.peepTop())[0]);
102             System.out.println(
                    "|実行: stack2.pushdown(stack1);");
103             stack2.pushdown(stack1);
104             System.out.println(
                    "==>stack1: " + stack1.getDetailedConfig()
105             System.out.println(
                    "==>stack2: " + stack2.getDetailedConfig()
106         }
107 }
[motoki@x205a]$  javac StackOfAnyObjects.java
[motoki@x205a]$  java StackOfAnyObjects
```

```
stack1: stack contains 0 elements: { }
|実行: stack1.pushdown("a");
|実行: stack1.pushdown(new Integer(2));
|実行: stack1.pushdown(new int[] {3, 4, 5});
###Stack capacity is increased###
#<New> pushdownStack (type=Object, capacity=102, currentNumOfE
|実行: stack1.pushdown(new StackOfAnyObjects());
==>stack1: stack contains 4 elements: {
    a,
    2,
    [I@11b9fb1,
    pushdownStack (type=Object, capacity=100, currentNumOfEele
|実行: StackOfAnyObjects stack2 = (StackOfAnyObjects) stack1.p
==>((int[])stack1.peepTop())[0])=3
|実行: stack2.pushdown(stack1);
==>stack1: stack contains 3 elements: {
    a,
```

```
    2,
    [I@11b9fb1 }
==>stack2: stack contains 1 elements: {
    pushdownStack (type=Object, capacity=102, currentNumOfEele
[motoki@x205a]$
```

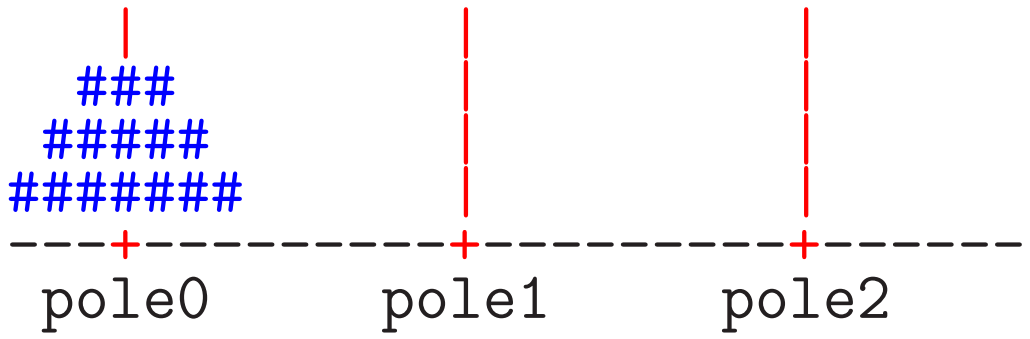
例題19．7（自習 ハノイの塔問題）　3本の棒と大きさの互いに異なる n 枚の円盤があり、円盤が全て下から大きい順に1つの棒に挿し込まれている。この時、

積み重ねられた円盤の大きさは常に下の方が大きい

という状態を保ったまま円盤を1枚ずつ1本の棒から別の棒に移すという操作を繰り返し、最終的に全ての円盤を指定した棒に積み上げるための手順を見つけ出す問題をハノイの塔問題という。正整数 n を読み込み、円盤が n 枚ある時の

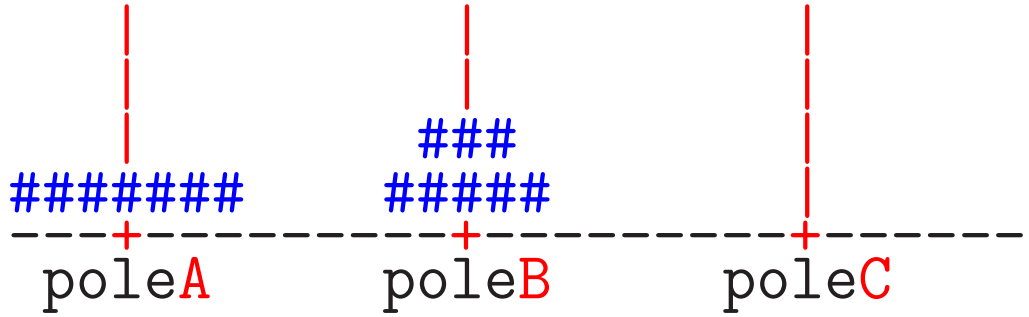
- Hanoiの塔の問題の解となる円盤移動の列 と
- 各々の時点での円盤の配置状況

を起こる順に混ぜて表示していくJavaプログラムを作成せよ。

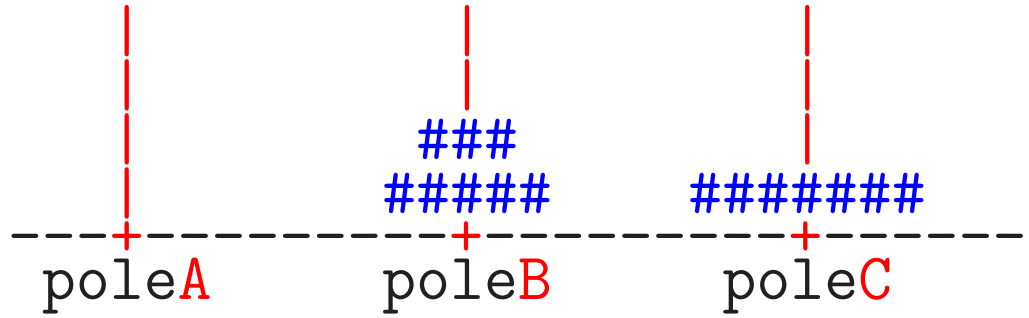


（考え方）　移動手順を再帰的に記述できる。

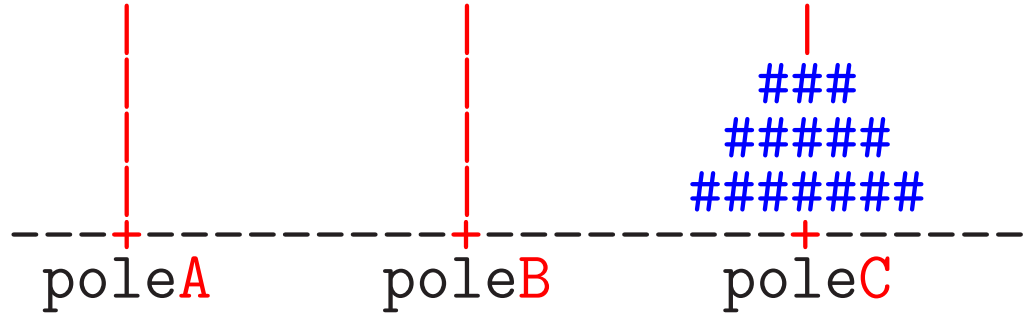
(1) 棒Aに挿し込まれている n 枚の円盤の内、上の（n−1）枚を棒Bに移し、



(2) 棒Aの残りの1枚の円盤を棒Cに移し、



(3) 棒Bに移した（n−1）枚の円盤を棒Cに移す



という決まりきった手順で、棒Aの n 枚の円盤をそっくり棒Cに移せる

問題の要求していることを行うため 、
次の3種類のオブジェクトを考え、円盤移動の手順の進行に沿って各々の時点での円盤の配置状況をはっきりとオブジェクトの形で追跡・保持

- 円盤オブジェクト ··· 属性値として円盤の半径。
- 塔オブジェクト ··· push-downスタックの一種と考える。
  pushdownやpopupする要素は円盤オブジェクト
- 円盤の配置状況を表すオブジェクト ···
  ◇構成要素として3個の塔オブジェクトをもつ。
  ◇toStringメソッドで円盤の配置状況を文字列として返す様に

(プログラミング)

◇塔オブジェクト ...例題19.4で示したStackOfAnyObjectsクラスのインスタンス

◇円盤オブジェクトのクラス ...円盤の配置状況を表すオブジェクトのクラスの中で局所的に定義

◇円盤が n 個あった時、半径が 1～n の円盤が1個ずつあるとし、例えば円盤が3枚の場合の初期状態を次の様に表示する

```
   |          |          |
  ###         |          |
 #####        |          |
#######       |          |
---+----------+----------+------
 pole0      pole1      pole2
```

```
[motoki@x205a]$  cat -n TowerOfHanoiMain.java
     1 /* 正整数 n を読み込み、円盤がn枚ある時の          */
     2 /*      |Hanoiの塔の問題の解となる円盤移動の列   */
     3 /*      |と 各々の時点での円盤の配置状況         */
     4 /* を起こる順に混ぜて表示していくJavaプログラム */
     5
     6 import java.util.Scanner;
     7
     8 class TowerOfHanoiMain {
     9     private static TowerOfHanoiConfig config;
    10
    11     public static void main(String[] args) {
    12         Scanner inputScanner = new Scanner(System.in);
    13
    14         //標準入力から円盤の枚数データを入力
    15         System.out.print("円盤の枚数: ");
    16         int numOfDisks = inputScanner.nextInt();
    17
    18         //円盤の初期配置状況を表すオブジェクトを生成
```

```
19          config = new TowerOfHanoiConfig(numOfDisks);
20          System.out.println(config);
21
22          //予備の棒pole1を利用して全ての円盤を
                                              pole0からpole2に移す
23          int pole0=0, pole1=1, pole2=2, indentSize=0;
24          generateMoveSequence(numOfDisks, pole0, pole2,
                                      pole1, indentSize);
25      }
26
27      //numOfDisk枚の円盤を予備の棒viaを利用して
28      //棒fromから棒toに移動するための手順を示す
29      private static void generateMoveSequence(
                          int numOfDisk, int from, int to,
30                        int via, int indentSize) {
31          if (numOfDisk == 1) {
32              for (int i=0; i<indentSize; ++i)
33                  System.out.print(">");
34              System.out.printf(
```

```
                    "円盤1枚を pole%d から pole%d に移す。%n",
35                  from, to);
36                config.moveDisk(from, to);  //円盤移動
37                System.out.println(config);
                                        //円盤の配置状況を表示
38            } else {
39                generateMoveSequence(numOfDisk-1, from,
                                    via, to, indentSize+1);
40                generateMoveSequence(1, from, to, via,
                                    indentSize);
41                generateMoveSequence(numOfDisk-1, via,
                                    to, from, indentSize+1);
42            }
43        }
44 }
[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 {
 6          private final int radius;
 7
 8          //コンストラクタ
 9          public Disk(int radius) {
10              this.radius = radius;
11          }
12
13          //Diskインスタンスの標準的な文字列表現を定める
14          @Override
15          public String toString() {
16              return "disk of radius " + radius;
17          }
18
19          //ゲッターメソッド
20          public int getRadius() {
21              return radius;
22          }
```

```
23      }
24      //-----------------------------------------------------
25
26      private StackOfAnyObjects[] pole;
27      private final int numOfDisks;
28      private final int poleHeight;
29      private final int maxRadiusOfDisk;
30
31      //コンストラクタ
32      public TowerOfHanoiConfig(int numOfDisks) {
33          this.numOfDisks = numOfDisks;
34          this.poleHeight = numOfDisks + 1;
35          if (numOfDisks < 2)
36              this.maxRadiusOfDisk = 2;
37          else
38              this.maxRadiusOfDisk = numOfDisks;
39
40          pole = new StackOfAnyObjects[3];
41          for (int i=0; i<3; ++i)
```

```
42              pole[i] = new StackOfAnyObjects(
                                                numOfDisks);
43          for (int i=maxRadiusOfDisk; i>0; --i)
44              pole[0].pushdown(new Disk(i));
45      }
46
47      //途中の円盤の配置状況を表すインスタンスの
                                        標準的な文字列表現を定める
48      @Override
49      public String toString() {
50          String result = "";
51          for (int h=poleHeight-1; h>=0; --h) {
52              for (int i=0; i<3; ++i) {
53                  if (h >= pole[i].getNumOfEle()) {
54                      for (int k=0; k<maxRadiusOfDisk;
                                                    ++k)
55                          result += " ";
56                      result += "|";
57                      for (int k=0; k<maxRadiusOfDisk;
```

```
                                                    ++k)
58                          result += " ";
59                      } else {
60                          int radius = ((Disk)pole[i].
                               peepEleOfIndex(h)).getRadius();
61                          for (int k=0; k<maxRadiusOfDisk
                                                    -radius; ++k)
62                              result += " ";
63                          for (int k=0; k<2*radius+1; ++k)
64                              result += "#";
65                          for (int k=0; k<maxRadiusOfDisk
                                                    -radius; ++k)
66                              result += " ";
67                      }
68                      result += "   ";
69                  }
70                  result += "\n";
71              }
72
```

```
73          for (int i=0; i<3; ++i) {
74              for (int k=0; k<maxRadiusOfDisk; ++k)
75                  result += "-";
76              result += "+";
77              for (int k=0; k<maxRadiusOfDisk; ++k)
78                  result += "-";
79              result += "---";
80          }
81          result += "\n";
82
83          for (int i=0; i<3; ++i) {
84              for (int k=0; k<maxRadiusOfDisk-2; ++k)
85                  result += " ";
86              result += "pole" + i;
87              for (int k=0; k<maxRadiusOfDisk-2; ++k)
88                  result += " ";
89              result += "   ";
90          }
91
```

```
 92          return result;
 93      }
 94
 95      //円盤の移動操作
 96      public void moveDisk(int from, int to) {
 97          if (pole[from].isEmpty()) {
 98              System.out.printf(
                                "ERROR: 空のpole[%d]から" +
                                "円盤を取り出そうとした%n" +
 99                              "==>強制終了%n",
100                             from);
101             System.exit(-1);
102         }
103         if (!pole[to].isEmpty()
              && ((Disk)pole[from].peepTop()).getRadius()
104             >=((Disk)pole[to].peepTop()).getRadius()){
105             System.out.printf("ERROR: 禁止された"+
                                "移動を行なおうとした%n" +
106                             " 移動元のpole[%d]のtop"+
```

```
                                    "にある円盤の半径=%d%n" +
  107                               " 移動先のpole[%d]のtop"+
                                    "にある円盤の半径=%d%n" +
  108                               "==>強制終了%n",
  109                               from, ((Disk)pole[from].
                                          peepTop()).getRadius(),
  110                               to, ((Disk)pole[to].
                                          peepTop()).getRadius());
  111                 System.exit(-1);
  112             }
  113             pole[to].pushdown(pole[from].popup());
  114         }
  115 }
[motoki@x205a]$  javac TowerOfHanoiMain.java
[motoki@x205a]$  java TowerOfHanoiMain
円盤の枚数: 3
   |          |          |
  ###         |          |
 #####        |          |
```

```
#######       |           |
---+---------+---------+------
 pole0      pole1      pole2
>>円盤1枚を pole0 から pole2 に移す。
   |           |           |
   |           |           |
 #####         |           |
#######        |          ###
---+---------+---------+------
 pole0      pole1      pole2
>円盤1枚を pole0 から pole1 に移す。
   |           |           |
   |           |           |
   |           |           |
#######     #####        ###
---+---------+---------+------
 pole0      pole1      pole2
>>円盤1枚を pole2 から pole1 に移す。
   |           |           |
```

```
   |          |          |
   |          |          |
   |         ###         |
#######     #####        |
---+---------+---------+------
 pole0      pole1      pole2
```

円盤1枚を pole0 から pole2 に移す。

```
   |          |          |
   |          |          |
   |          |          |
   |         ###         |
   |        #####     #######
---+---------+---------+------
 pole0      pole1      pole2
```

>>円盤1枚を pole1 から pole0 に移す。

```
   |          |          |
   |          |          |
   |          |          |
   |          |          |
  ###       #####     #######
---+---------+---------+------
 pole0      pole1      pole2
```

```
>円盤1枚を pole1 から pole2 に移す。
   |          |          |
   |          |          |
   |          |        #####
   |          |       #######
  ###         |       #######
---+----------+----------+------
 pole0      pole1      pole2
>>円盤1枚を pole0 から pole2 に移す。
   |          |          |
   |          |          |
   |          |         ###
   |          |        #####
   |          |       #######
---+----------+----------+------
 pole0      pole1      pole2
[motoki@x205a]$
```

19–6 動的データ構造を扱う例

内容変更

例題19．8（Napier数eの1000桁計算）　ネピア数(Napier number)

$$e=\lim_{x\to\infty}(1+\frac{1}{x})^x=\sum_{i=0}^{\infty}\frac{1}{i!}=2.718281828\cdots\cdots$$

の高精度計算(小数点以下1000桁まで)を、例題8.1と同様に

$$e \approx \sum_{i=0}^{450}\frac{1}{i!}$$

$$= (((\cdots((1\cdot\frac{1}{450}+1)\frac{1}{449}+1)\cdots)\frac{1}{3}+1)\frac{1}{2}+1)\frac{1}{1}+1$$

という計算順序で行うJavaプログラムを作成せよ。但し、例題8.1と全く同様に10進小数を配列で表して計算を進めることもできるが、ここでは、1桁分の数を記憶するオブジェクトを線形リスト状に繋げて、それらの間でメッセージ伝達を繰り返すことによって計算を進めていくことにせよ。

（考え方）　どういうクラスを用意するか

- 小数点以下1000桁の10進小数を保持・管理するオブジェクトがあれば、mainメソッド側ではこのオブジェクト内の実装に気を使うこともなく、保持した数値への加算や除算の依頼だけを行えば良い。

⇨ この種のオブジェクトをインスタンスとするクラス
NumberWith1000DecimalPlaces を用意

- 1桁分の数を記憶するオブジェクトのクラスDigit
  ... NumberWith1000DecimalPlacesクラスの中で局所的に定義

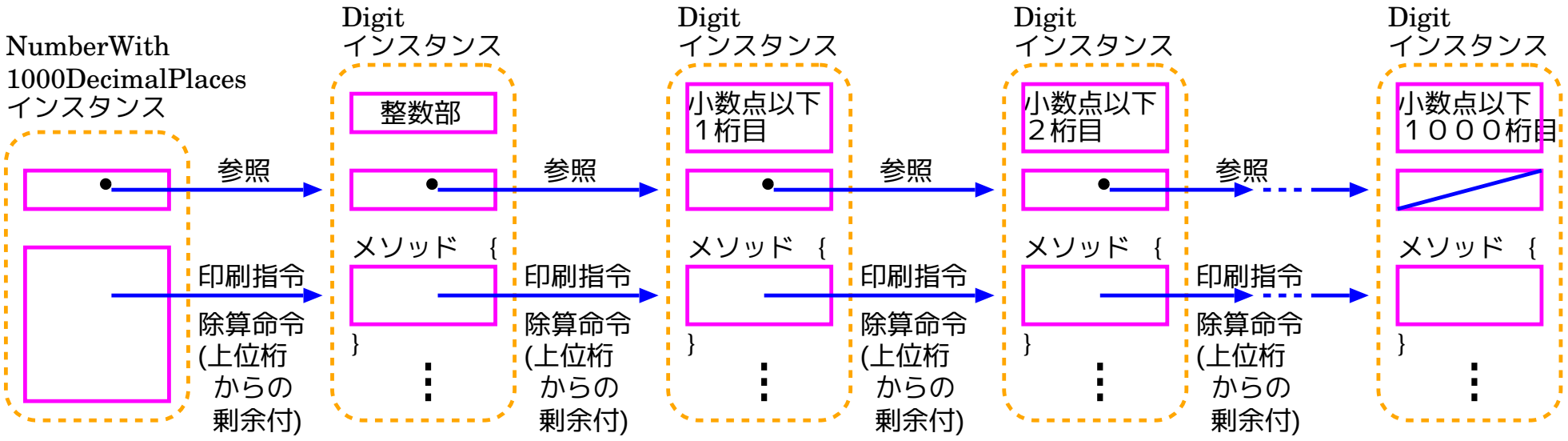
オブジェクト間の処理の連携に関しては

手計算で除算を行う状況を思い描いてもらいたい。

除算は上位の桁から下位の桁に向かって進めるので、それぞれの桁を表すオブジェクトからその1つ下の桁を表すオブジェクトに除算指令が次々と伝達する様にすれば良い。

メッセージを伝達するためには、その伝達先のオブジェクトへの参照情報も必要になる。

従って、オブジェクト間の作業依頼の関係を次の図の様に考える。



(プログラミング)

```
[motoki@x205a]$ cat -n CalcNapierNumberMain.java
     1 /* Napier数eを小数点以下1000桁まで計算して出力する... */
     2
     3 public class CalcNapierNumberMain {
     4     public static void main(String args[]) {
     5         NumberWith1000DecimalPlaces
     6             e = new NumberWith1000DecimalPlaces();
                                              // e <-- 0
     7
     8         e.add(1);                      // e <-- e+1
     9         for (int k=450; k>0; k--) {
    10             e.divideBy(k);             // e <-- e/k
    11             e.add(1);                  // e <-- e+1
    12         }
    13
    14         System.out.print("e = ");
```

```
15            e.print("    ");
16        }
17    }
[motoki@x205a]$  cat -n NumberWith1000DecimalPlaces.java
 1  /* 小数点以下1000桁までの数を表すオブジェクトのクラス */
 2
 3  public class NumberWith1000DecimalPlaces {
 4      //小数部の1桁分(もしくは整数部)の値を表す
                                          オブジェクトのクラス
 5      //(NumberWith1000DecimalPlacesインスタンスでは
 6      //このDigitオブジェクトを線形リスト状に繋げて
                                  「小数点以下1000桁までの数」を表す)
 7      private static class Digit {
 8          private int    value;           //小数部1桁分の値
 9          private Digit nextLowerDigit; //すぐ下の桁の
10                                        //オブジェクトへの参照
11          //コンストラクタ
```

```
12          public Digit(Digit nextLowerDigit) {
13              value = 0;
14              this.nextLowerDigit = nextLowerDigit;
15          }
16
17          //Digitインスタンスの標準的な文字列表現を定める
18          @Override
19          public String toString() {
20              return "a node that is an element of a link
21                      + " and that represents some decimal
22          }
23
24          //Digitインスタンスの表す桁に引数numを加算
25          public void add(int num) {
26              value += num;
27          }
28
```

```
29          //上位桁からの余りremainderFromUpperDigitが
30          //あることを仮定し、この桁以下のオブジェクト群の
                                        表す数値をdivisorで割る
31          public void divideBy(int divisor,
                                 int remainderFromUpperDigit) {
32              value += 10 * remainderFromUpperDigit;
33              int remainderToLowerDigit
                                            = value % divisor;
34              value /= divisor;
35              if (nextLowerDigit != null)
36                  nextLowerDigit.divideBy(divisor,
                                        remainderToLowerDigit);
37          }
38
39          //この桁が現在の小数部の出力行中でcolumn桁目に
40          //当たることを仮定し、この桁以下のオブジェクト群
41          //の保持する数値(0~ 9)を順に出力する。 但し、
```

```
42          //引数fillerAtHeadOfLinesは2行目以下の左端に
                                        詰める文字列を表す。
43          public void print(int column,
                              String fillerAtHeadOfLines) {
44              switch (column) {
45              case 10: case 20: case 30:
                                          case 40: case 50:
46                  System.out.print(value + " ");
47                  column++;
48                  break;
49              case 60:
50                  System.out.println(value);
51                  System.out.print(fillerAtHeadOfLines);
52                  column = 1;
53                  break;
54              default:
55                  System.out.print(value);
```

```
56                  column++;
57                  break;
58              }
59
60              if (nextLowerDigit != null)
61                  nextLowerDigit.print(column,
                                         fillerAtHeadOfLines);
62              else
63                  System.out.println();
64          }
65      }
66      //-----------------------------------------------
67
68      private Digit topDigit;  //整数部
69
70      //コンストラクタ
71      public NumberWith1000DecimalPlaces() {
```

```
72          topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
75      }
76
77      //NumberWith1000DecimalPlacesインスタンスの標準的な

78      //文字列表現を定める
79      @Override
80      public String toString() {
81          return "linearly linked list representation of'
82                  + " a number with 1000 decimal places";
83      }
84
85      //保持する「小数点以下1000桁までの数」に
                                                整数numを加算
86      public void add(int num) {
```

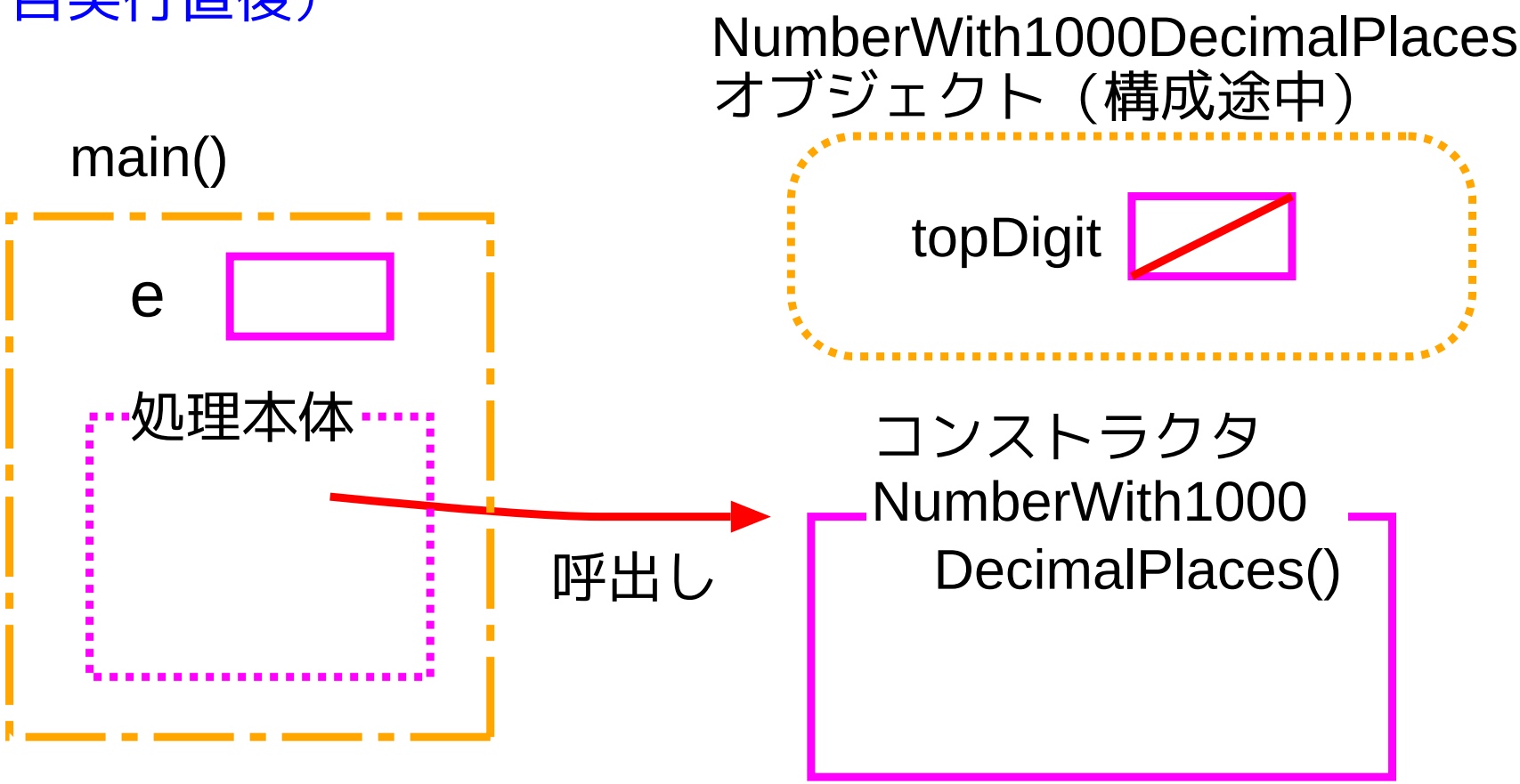
```
87          topDigit.add(num);
88      }
89
90      //保持する「小数点以下1000桁までの数」を
                                              divisorで除算
91      public void divideBy(int divisor) {
92          topDigit.divideBy(divisor, 0);
93      }
94
95      //保持する「小数点以下1000桁までの数」を出力。但し、
96      //現在の出力行で既に何文字か出力されていると考え、
97      //それと同じ長さの文字列fillerAtHeadOfLinesを
98      //2行目以降の左端に埋めることにする。
99      public void print(String fillerAtHeadOfLines) {
100         System.out.print(topDigit.value + ".");
101         int extendedLength
                              = (topDigit.value+".").length();
```

```
   102          for (int i=0; i<extendedLength; i++)
   103              fillerAtHeadOfLines += " ";
   104          topDigit.nextLowerDigit.print(1,
                                              fillerAtHeadOfLines);
   105      }
   106 }
[motoki@x205a]$  javac CalcNapierNumberMain.java
[motoki@x205a]$  java CalcNapierNumberMain
e = 2.7182818284 5904523536 0287471352 6624977572 4709369995 9
      6277240766 3035354759 4571382178 5251664274 2746639193 2
      8174135966 2904357290 0334295260 5956307381 3232862794 3
      8298807531 9525101901 1573834187 9307021540 8914993488 4
      7614606680 8226480016 8477411853 7423454424 3710753907 7
      5517027618 3860626133 1384583000 7520449338 2656029760 6
      7093287091 2744374704 7230696977 2093101416 9283681902 5
      4637721112 5238978442 5056953696 7707854499 6996794686 4
      9316368892 3009879312 7736178215 4249992295 7635148220 8
```

```
       6680331825 2886939849 6465105820 9392398294 8879332036 2
       3012381970 6841614039 7019837679 3206832823 7646480429 5
       7825098194 5581530175 6717361332 0698112509 9618188159 3
       5988885193 4580727386 6738589422 8792284998 9208680582 5
       4841984443 6346324496 8487560233 6248270419 7862320900 2
       3043699418 4914631409 3431738143 6405462531 5209618369 0
       7683964243 7814059271 4563549061 3031072085 1038375051 0
       1718986106 8739696552 1267154688 9570350354
[motoki@x205a]$
```

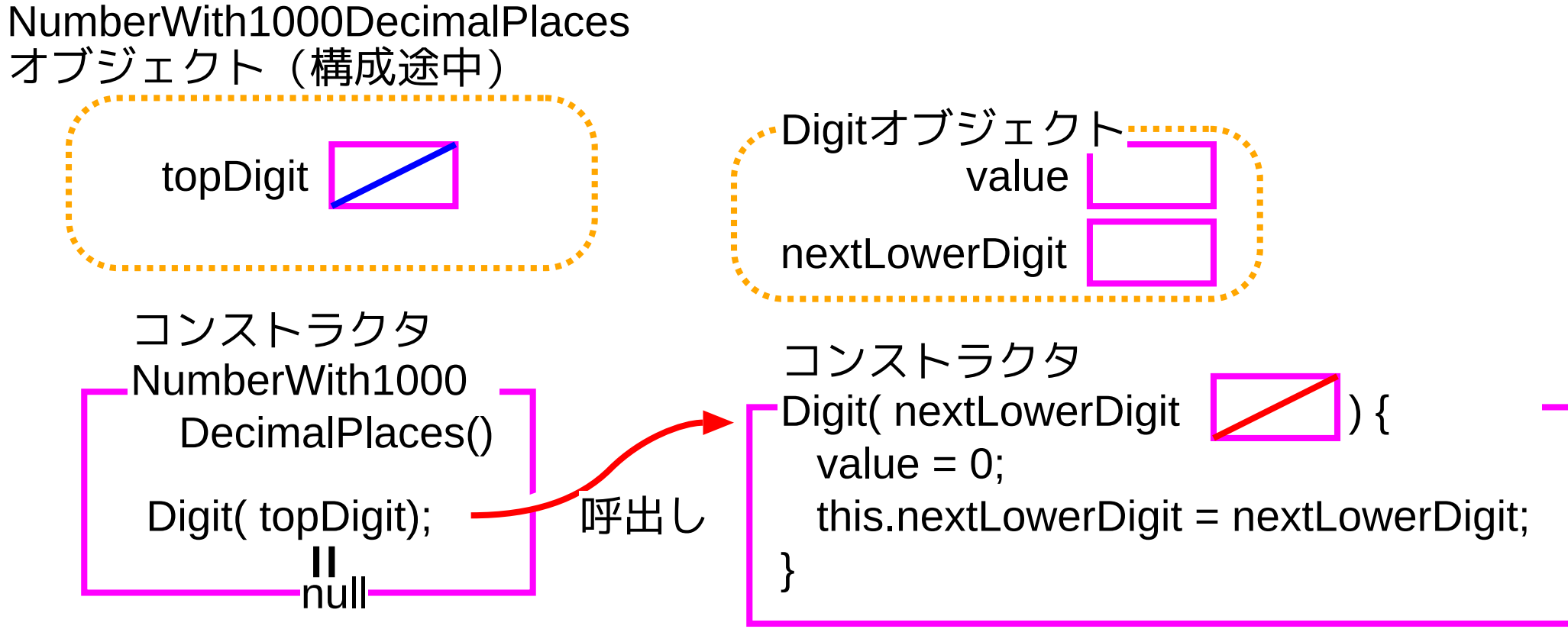
```
71      public NumberWith1000DecimalPlaces() {
72          topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
75      }
```

NumberWith1000DecimalPlaces.javaの 72～74行目の実行の様子：
(72行目実行直後)



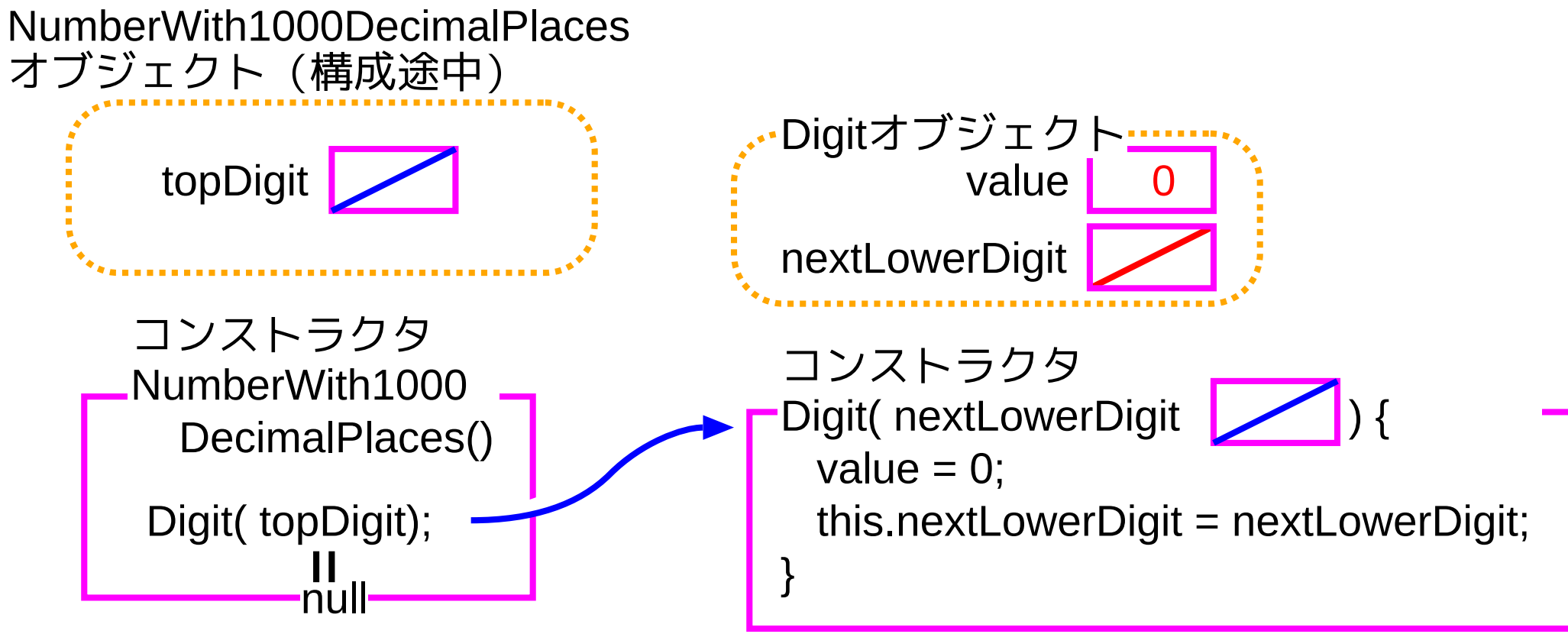
```
71      public NumberWith1000DecimalPlaces() {
72          Digit  topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
75      }
```

(74行目，コンストラクタ呼び出し，i=1000)



```
 8          private int    value;              //小数部1桁分の値
 9          private Digit nextLowerDigit; //すぐ下の桁のオブ..

12          Digit(Digit nextLowerDigit) {  //コンストラクタ
13              value = 0;
14              this.nextLowerDigit = nextLowerDigit;
```

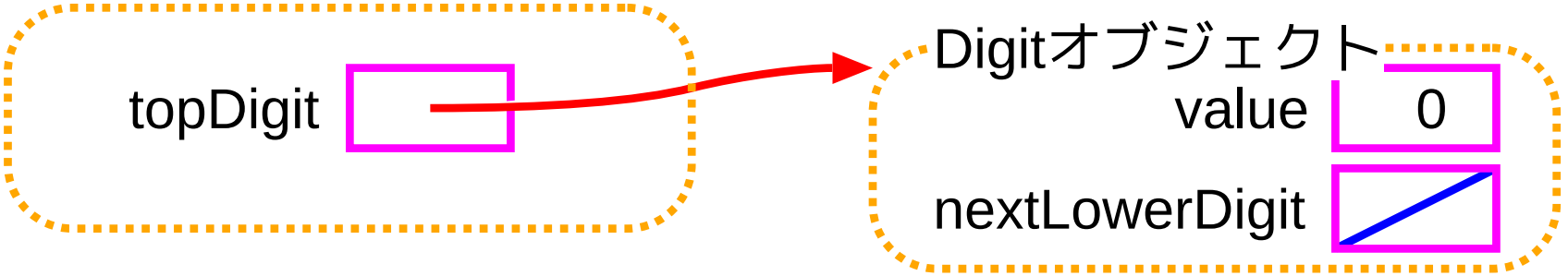
（コンストラクタDigit()内，14行目実行直後）



```
71      public NumberWith1000DecimalPlaces() {
72          Digit  topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
```

(74行目実行直後，i=1000)

NumberWith1000DecimalPlaces
オブジェクト（構成途中）

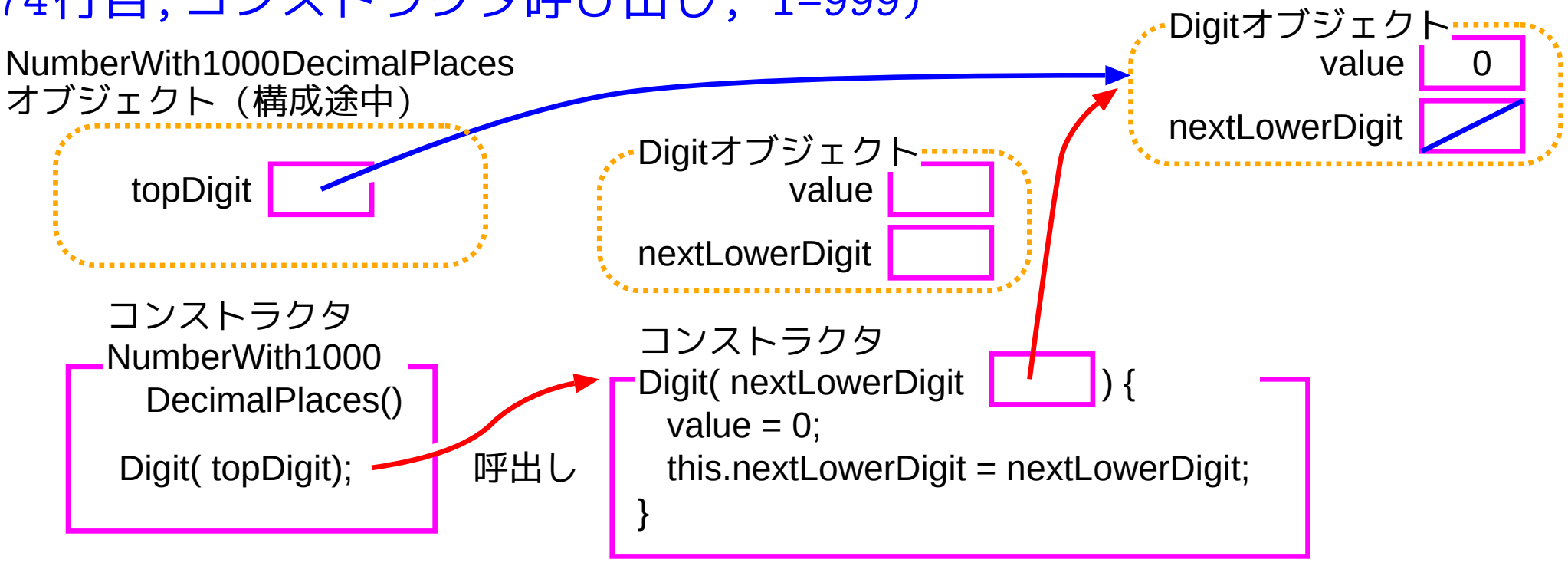


コンストラクタ
NumberWith1000
DecimalPlaces()

Digit(topDigit);

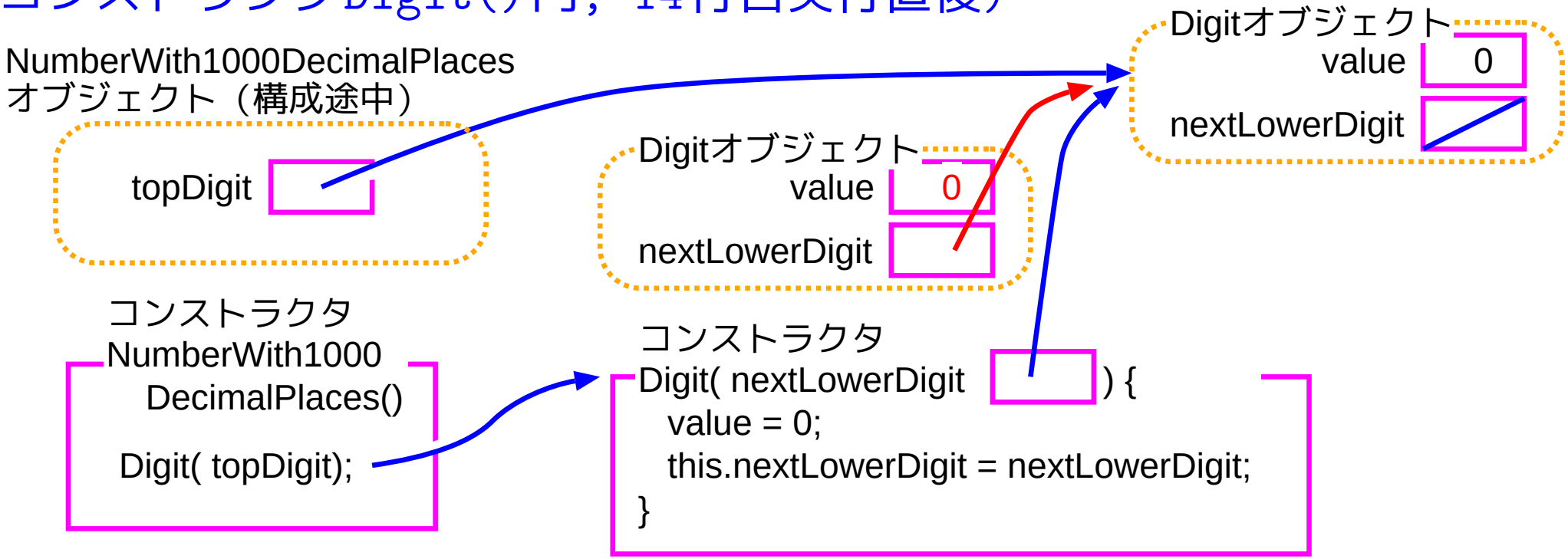
```
71      public NumberWith1000DecimalPlaces() {
72          Digit  topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
```

(74行目，コンストラクタ呼び出し，i=999)



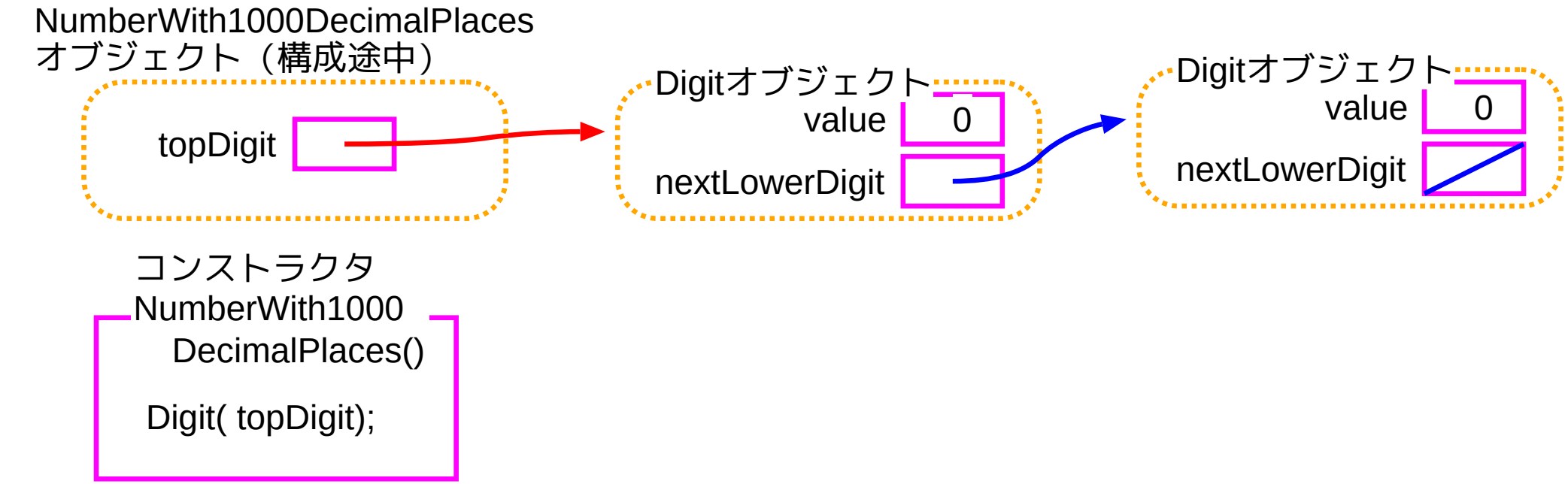
```
 8          private int    value;              //小数部1桁分の値
 9          private Digit nextLowerDigit; //すぐ下の桁のオブ..

12          Digit(Digit nextLowerDigit) {  //コンストラクタ
13              value = 0;
14              this.nextLowerDigit = nextLowerDigit;
```

（コンストラクタDigit()内，14行目実行直後）



```
71      public NumberWith1000DecimalPlaces() {
72          Digit  topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
```

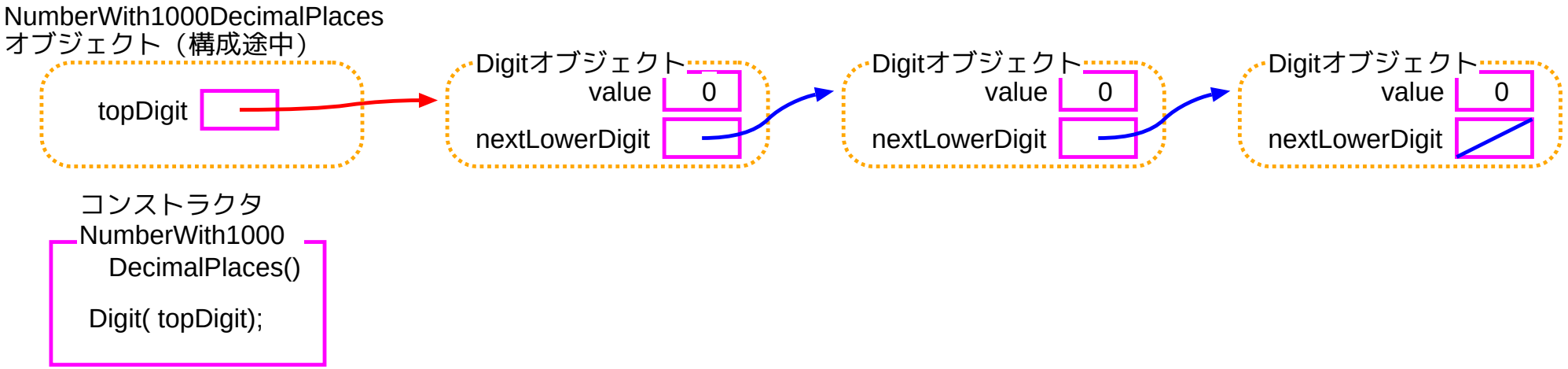
(74行目実行直後，i=999)



.....

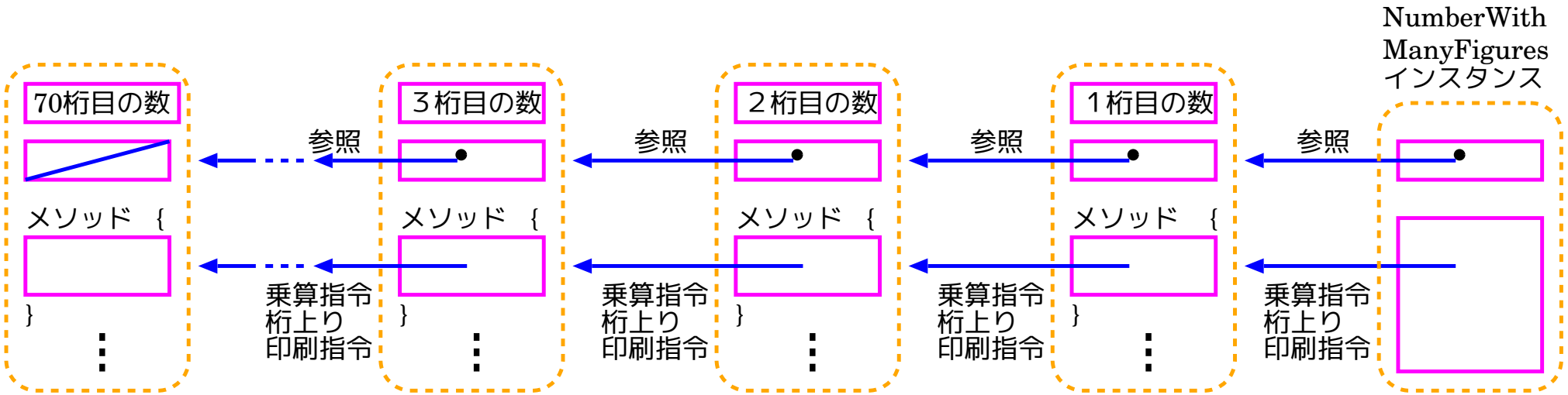
```
71      public NumberWith1000DecimalPlaces() {
72          Digit  topDigit = null;
73          for (int i=1000; i>=0; i--)
74              topDigit = new Digit(topDigit);
```

(74行目実行直後，i=998)



. . . . . .

□演習19．11（53!の高精度計算）　53! を正確に計算して出力するJavaプログラムを作成せよ。但し、ここでは例19.2に倣って、1桁分の数を記憶するオブジェクトを多数作り、それらの協調で計算を進めていくことにする。



例題19．9（自習 2分木の中間順走査）　例題12.4では、

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

② それを2分木上に追加登録する (但し、

(左の子とその子孫の識別子) $\leq$ (親の識別子)

(親の識別子) $<$ (右の子とその子孫の識別子)

という関係を損なわない様に追加登録する)、

という作業を繰り返し、最後に2分木を中間順に走査し登録されたものを順に出力するCプログラムを作成した。これに対して、ここでは(ほぼ)同等の処理を行うプログラムをJavaで構成してみよ。

（考え方）　どういうクラスを用意するか

- 2分木の節点上にStringオブジェクトとintデータの組を基本要素として保持し、常に

  （左の子とその子孫のStringオブジェクト）

  ≤　（親のStringオブジェクト）

  （親のStringオブジェクト）

  ＜　（右の子とその子孫のStringオブジェクト）

  という関係を満たす様に管理するオブジェクトがあれば、...

  ⇨　この様な、2分木を管理するオブジェクトのクラス

  BinaryTreeOfStringInt を用意

- 2分木の要素を表すオブジェクトのクラスNode

  ...　2分木のクラスの中で局所的に定義

(プログラミング)

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

```
[motoki@x205a]$  cat -n SortIdIntPairsByBinaryTreeMain.java
     1 /* (1)識別子と整数データの組を読み込んでは
     2 /* (2)その組を2分木の葉先に追加(但し、2分木を中間順に */
     3 /*      走査した時、識別子に関して辞書順になる様に追加)
     4 /* という作業を繰り返し、最後に2分木を中間順に走査し保持
     5 /* されたものを順に出力するJavaプログラム
     6
     7 import java.util.Scanner;
     8
     9 class SortIdIntPairsByBinaryTreeMain {
    10     public static void main(String[] args) {
```

```
11          Scanner inputScanner = new Scanner(System.in);
12          BinaryTreeOfStringInt tree =
                                  new BinaryTreeOfStringInt();
13
14          //データ入力と2分木への挿入
15          while (inputScanner.hasNext()) {
16              String newString = inputScanner.next();
17              if (inputScanner.hasNextInt()) {
18                  int newInt = inputScanner.nextInt();
19                  tree.insert(newString, newInt);
20              } else {
21                  System.out.printf("ERROR: invalid data
22                                          "  ==>inut is aborted
23                  break;
24              }
25          }
26
27          //2分木を中間順に走査し保持されたものを順に出力
28          tree.inorderTraverseAndShowContents();
```

```
    29     }
    30 }
[motoki@x205a]$  cat -n BinaryTreeOfStringInt.java
     1 /* Stringオブジェクトとintデータの組を基本要素として */
     2 /* 2分木で(中間順走査した時Stringの辞書順になる様に) */
     3 /* 保持するオブジェクトのクラス                       */
     4
     5 public class BinaryTreeOfStringInt {
     6     //2分木を構成する基本要素を表すオブジェクトのクラス
     7     private static class Node {
     8         private String stringData;
     9         private int    intData;
    10         private Node   leftChild, rightChild;
    11
    12         //コンストラクタ
    13         public Node(String stringData, int intData) {
    14             this(stringData, intData, null, null);
    15         }
    16
```

```
17          public Node(String stringData, int intData,
18                      Node leftChild, Node rightChild) {
19              this.stringData = stringData;
20              this.intData    = intData;
21              this.leftChild  = leftChild;
22              this.rightChild = rightChild;
23          }
24
25          //Nodeインスタンスの標準的な文字列表現を定める
26          @Override
27          public String toString() {
28              return "binary node of String-int pair ("
                        + stringData
29                      + ", " + intData + ")";
30          }
31
32          //このNode以下を中間順に走査した時に
33          //stringDataの辞書順を保てる様に、
                                        葉先に新要素を追加
```

```
34          public void insertLexicoOrder(
                            String newString, int newInt) {
35              if (newString.compareTo(stringData)<=0) {
36                  if (leftChild == null)
37                      leftChild = new Node(newString,
                                                 newInt);
38                  else
39                      leftChild.insertLexicoOrder(
                                        newString, newInt);
40              } else {
41                  if (rightChild == null)
42                      rightChild = new Node(newString,
                                                 newInt);
43                  else
44                      rightChild.insertLexicoOrder(
                                        newString, newInt);
45              }
46          }
47
```

```
48          //このNode以下を中間順に走査して
                                        順に蓄えられた内容を表示
49          public int inorderTraverseAndShowContents(
                                            int serialNumber) {
50              if (leftChild != null)
51                  serialNumber = leftChild.
52                      inorderTraverseAndShowContents(
                                                serialNumber);
53              System.out.printf("%4d %11d  %s%n",
54                                  serialNumber++,
                                    intData, stringData);
55              if (rightChild != null)
56                  serialNumber = rightChild.
57                      inorderTraverseAndShowContents(
                                                serialNumber);
58              return serialNumber;
59          }
60      }
61      //-----------------------------------------------
```

```
62
63      private Node root;
64
65      //コンストラクタ
66      public BinaryTreeOfStringInt() {
67          root = null;
68      }
69
70      //BinaryTreeOfStringIntインスタンスの標準的な
                                                文字列表現を定める
71      @Override
72      public String toString() {
73          return "binary tree of String-int pairs\n"
74                  + "   that are seen to be sorted "
75                  +          "by lexicographic order of St
76                  + "        when the tree is inorderly tr
77      }
78
```

```
79      //2分木を中間順に走査した時に
80      //stringDataの辞書順を保てる様に、葉先に新要素を追加
81      public void insert(String newString, int newInt) {
82          if (root == null)
83              root = new Node(newString, newInt);
84          else
85              root.insertLexicoOrder(newString, newInt);
86      }
87
88      //2分木を中間順に走査して順に蓄えられた内容を表示
89      public void inorderTraverseAndShowContents() {
90          System.out.printf("        intData    stringData
91                            "----  ----------  ----------
92          if (root == null)
93              System.out.println("  (empty)");
94          else
95              root.inorderTraverseAndShowContents(1);
96      }
97  }
```

```
[motoki@x205a]$ javac SortIdIntPairsByBinaryTreeMain.java
[motoki@x205a]$ cat dynamic-llist.data
asdfghjk 55555
qwertyui 222222
abc      123
ppp_qqq  2345
zxcv     987
kkk      456
efghi    77
[motoki@x205a]$ java SortIdIntPairsByBinaryTreeMain
                                        < dynamic-llist.data
       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]$
```
