単位： 出席状況と実習課題の達成度によって評価します。

プログラミング作業の流れ： 適当なエディタ（たとえば TeraPad や Meadow など）で適当なソースを書き、それをコマンドプロンプト上でコンパイルして、最後にコマンドプロンプト上で実行します。具体的には次のようにします。

1. エディタで Hello.java を作成

```
class Hello {
   public static void main(String[] args) {
      System.out.println("こんにちは");
   }
}
```

2. コマンドプロンプト上で javac Hello.java
3. コマンドプロンプト上で java Hello

何でわざわざ Java で書くの？的プログラム。またはプログラミングの基本（その 1）： まずは四則演算の仕方を身に付けましょう。はじめに足し算から。足し算 5 + 2 の結果を出力するプログラムは、次のように書きます。

```
class Addition {
   public static void main(String[] args) {
      int x = 5;
      int y = 2;
      System.out.println(x + y);
   }
}
```

同じ内容を C 言語で書くと次のようになります。

```
#include <stdio.h>
int main() {
   int x = 5;
   int y = 2;
   printf("%d\n", x + y);
   return 0;
}
```

この程度の単純なプログラムならば、どの言語を使ってもよく、どの言語でもだいたい似た感じでプログラミングできます。言い換えれば、言語に依存しないプログラミングの基礎とも言えます。上記のプログラムをすこし改良しましょう。

```
/* ふたつの整数を読み込んで、その和を返す。 */
import java.util.Scanner;
class Addition2 {
   public static void main(String[] args) {
      System.out.println("整数をふたつ入力せよ。");
      Scanner yomu = new Scanner(System.in);
      int x = yomu.nextInt();
      int y = yomu.nextInt();
      System.out.println("和は" + (x + y));
   }
}
```

同じ内容を C 言語で書くと次のようになります。

```
#include <stdio.h>
int main() {
   int x, y;
   printf("整数をふたつ入力せよ。\n");
   scanf("%d %d", &x, &y);
   printf("和は %d\n", x + y);
   return 0;
}
```

つぎに割り算をしてみます。

```
/* 割り算いろいろ */
class Division {
   public static void main(String[] args) {
      int x = 5;
      int y = 2;
      int shou1 = x / y;
      double shou2 = x / y;
      double shou3 = (x + 0.) / y;
      double shou4 = (double)x / (double)y;
      System.out.println(shou1 + "," + shou2);
      System.out.println(shou3 + "," + shou4);
   }
}
```

割り算には注意が必要です。割り算に限らず、整数同士の演算は整数のまま扱われ、結果も整数です。したがって 5/2 の結果（すなわち 2）を浮動小数点数の shou2 に代入しても値は 2.0 となってしまいます。ところが shou3 のように片方だけでも浮動小数点数ならば、演算に先立って他方も浮動小数点数に格上げされて、浮動小数点数の演算として扱われます。なので shou3 の値は 2.5 です。あるいは shou4 のようにあらかじめ明示的に型を変換しておくことも出来ます。型の変換をキャストといいます。C 言語でも事情はまったく同じで、割り算をするには注意が要りました。

数学の関数もすこし用意されています。たとえば $\sqrt{2}$ や $\sin(\pi/7)$ の値を出力するには次のようにします。

```
/* 2 の平方根と sin(π/7) の値を出力する。*/
class MathExample {
   public static void main(String[] args) {
      double x = Math.sqrt(2);
      double y = Math.sin(Math.PI / 7);
      System.out.println("2 の平方根は" + x);
      System.out.println("sin(π/7) は" + y);
   }
}
```

同じ内容を C 言語で書くと次のようになります。

```
#include <stdio.h>
#include <math.h>
int main() {
   double x = sqrt(2);
   double y = sin(M_PI / 7);
   printf("2 の平方根は %f\n", x);
   printf("sin(π/7) は %f\n", y);
   return 0;
}
```

付録：

型	値の例	格納できる値の範囲
byte	−10, 3	8 ビット符号付き整数
short	−10, 3	16 ビット符号付き整数
int	−10, 3	32 ビット符号付き整数
long	−10, 3	64 ビット符号付き整数
float	−0.5, 1.25	32 ビット符号付き浮動小数点数
double	−0.5, 1.25	64 ビット符号付き浮動小数点数

Math クラスのメソッド	意味
Math.pow(x, y)	x^y
Math.log(x)	$\log x$
Math.abs(x)	$\|x\|$
Math.E	2.718281828459045

プログラミング II 実習課題（1 回目）　　学籍番号の下 2 桁 ☐☐ 名前＿＿＿＿＿＿＿＿

学籍番号の下 2 桁 ☐☐ 名前＿＿＿＿＿＿＿＿

1 次のプログラムをコンパイルしたらエラーが出た。コンパイルが通るように修正せよ。

```
/* 掛け算 5 × 2 の結果を出力する。*/
class Kakezan {
   public static void main(String[] args) {
      x = 5 * 2;
      System.out.println("5 × 2 = " + x);
   }
}
```

2 次のように記述して 7/2 の計算をしたところ値が 3.0 と出力された。プログラムを修正して正しく 3.5 と出力させよ。

```
/* 割り算 7 ÷ 2 の結果を出力する */
class Warizan {
   public static void main(String[] args) {
      int a = 7;
      int b = 2;
      double kotae = a / b;
      System.out.println(kotae);
   }
}
```

3 実習の感想やプログラミングの質問、あるいは講義と実習に対する要望などを記せ。

4 次のように記述して 0.1 を 10 回足した。

```
class Tashizan {
   public static void main(String[] args) {
      double sum = 0;
      for (int i = 1; i <= 10; i++) {
         sum = sum + 0.1;
      }
      System.out.println("答えは " + sum);
   }
}
```

(1) 出力された値は 0.9999999999999999 であった。なぜ 1.0 とならないのか（ヒント：0.1 を 2 進法で表すと?)。

(2) 正しく 1.0 と計算させるにはプログラムをどのように修正すればよいか。

Q & A： 前回の実習で出された質問に回答します。

- `System.out.println("答えは" + 10)` の + はどういう意味ですか。

– 演算子 + は文字列の連結または数値の加算を行います。質問の + は文字列の連結を意味しています。より詳しく言うと、「文字列 + 数値」や「数値 + 文字列」の演算では、数値が文字列に変換された上で連結が行われるという規則があります。したがって、まず整数値 10 が文字列 "10" に変換されて、次にふたつの文字列 "答えは" と "10" が連結されます。以下のふたつの例を比べてみましょう。

```
System.out.println("1 たす 2 は" + (1 + 2));
System.out.println("1 たす 2 は" + 1 + 2);
```

前者の出力は「1 たす 2 は 3」ですが、後者の出力は「1 たす 2 は 12」となります。演算は括弧で囲まれた部分を優先しますが、それ以外は左から評価していきます。

- クラス名にハイフン - は使えないのですか。

– 使えません。アンダースコア _ で我慢して下さい。

- println で出力するとき改行文字 \n は使えないのですか。

– 使えます。たとえば

```
System.out.println("七");
System.out.println("隈");
```

と書くのは

```
System.out.println("七\n 隈");
```

と書くのと同じことです。

- Scanner yomu = new Scanner(System.in) の yomu って何ですか。

– 変数です。より詳しく言うと、Scanner クラスのインスタンスを new で生成し、それを Scanner 型の変数 yomu に代入しています。このあたりの考え方と Java 特有の用語については、のちの講義で説明がありますので、いまは分からなくて構いません。変数名 yomu は、読む、のつもりで書きました。変数には勝手な名前がつけられます。

.....

Linux 上で作業したい： 総合情報処理センターの Linux で Java のプログラミングをする場合は、コマンド java のかわりにコマンド gij を使って下さい。作業の流れは以下のようになります。

1. `cd ~/windows/java/`
   注意： Linux の `~/windows/` と Windows の `H:¥` は同じ場所を見ています。
2. `emacs Hello.java &`
   注意： 文字コードはデフォルトで UTF-8 になります。
3. `javac Hello.java`
   注意： もし Hello.java の文字コードが UTF-8 以外のもの、たとえば Shift JIS ならば、オプションを付けて
   `javac -encoding Shift_JIS Hello.java`
   とコンパイルします。
4. `gij Hello`

蛇足： 文字コードの変換にはコマンド nkf を使います。たとえば「`nkf --overwrite -s Hello.java`」とすれば Shift JIS になります。あるいは Hello.java を emacs で開いているときに emacs 上で「`C-x RETURN f sjis-dos`」としてもオーケー。

.....

Java のインストール方法： 自宅の Windows 機で Java のプログラミングをするには、ウェブ上から Java の開発環境を入手してインストールする必要があります。手順を説明したファイルが

`www.rsp.fukuoka-u.ac.jp/~shuji/math/2010/how2install.pdf`

にあります。ファイルの閲覧にはユーザー名とパスワードが必要です。ユーザー名は math2009 でパスワードは abracadabra です。

何でわざわざ Java で書くの？的プログラム。またはプログラミングの基本（その 2）： 今回は条件分岐について実習します。特に if 文の使い方をマスターしましょう。これもまた C 言語とほとんど同じ書き方をします。

はじめに、入力した自然数が偶数か奇数かを判定するプログラムを考えます。剰余演算子 % が便利に使えます（剰余演算子は、割り算を行った結果の余りを求める演算子で、たとえば 7 % 3 の値は 1 になります）。

```
/* 自然数を読み込んで、その偶奇を判定する。*/
import java.util.Scanner;
class EvenOdd {
   public static void main(String[] args) {
      System.out.println("自然数を入力せよ。");
      Scanner yomu = new Scanner(System.in);
      int n = yomu.nextInt();
      if (n % 2 == 0) {
         System.out.println(n + "は偶数です。");
      }
      else {
         System.out.println(n + "は奇数です。");
      }
   }
}
```

次に、自然数 n に対してその階乗 $n!$ の値を返すプログラムを考えます。以下の 2 点に注目しましょう。

- このプログラムにはメソッドが 2 つ書いてあります。

– C 言語で「関数」と呼んでいたものを Java では「メソッド」と呼びます。このプログラムには main と fact の 2 つのメソッドがあります。

- 再帰的プログラミング

– 階乗を計算するメソッド fact は if 文を用いて再帰的・自己言及的に定義してあります。階乗 $a_n = n!$ が漸化式 $a_n = na_{n-1},\ a_0 = 1$ をみたすことを利用しています。

```
/* 自然数 n を読み込んで、階乗 n! の値を返す。*/
import java.util.Scanner;
class Factorial {
  /* main メソッド */
  public static void main(String[] args) {
    System.out.println("自然数を入力せよ。");
    Scanner yomu = new Scanner(System.in);
    int n = yomu.nextInt();
    System.out.println(n + "の階乗は" + fact(n));
  }
  /* 階乗を計算するメソッド */
  public static long fact(int n) {
    if (n == 0) {
      return 1;
    }
    else {
      return n * fact(n-1);
    }
  }
}
```

付録： if 文の条件分岐で書く条件式では、次に示す関係演算子を使って 2 つの値を比較することが多く、その結果は true か false のどちらか一方になります。

演算子	意味	演算子	意味
==	等号（左辺と右辺が等しい）	!=	$\neq$
>	$>$	<	$<$
>=	$\geq$	<=	$\leq$

プログラミング II 実習課題（2 回目）　　ペア番号 [　　]　学籍番号の下 2 桁 [　][　][　][　]

1 整数値をひとつ読み込んでその絶対値を返すプログラムを次のように書いた。空欄を埋めよ。

```
import java.util.Scanner;
class Absolute {
  public static void main(String[] args) {
    Scanner nyuryoku = new Scanner(System.in);
    System.out.print("整数値＝");
    int n = [　　　　].nextInt();
    int abs;
    [　　　　　　　　　　　　　　　　]
    System.out.println("その絶対値は" + abs);
  }
}
```

2 キーボードから入力した自然数 n に対して和 $1+\cdots+n$ の値を返すプログラムを次のように書いた。空欄を埋めよ。ただし再帰的プログラミングの考え方を用いること。

```
import java.util.Scanner;
class Sum {
  /* main メソッド */
  public static void main(String[] args) {
    System.out.println("自然数を入力せよ。");
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    System.out.println("1 から" + n + "までの
和は" + sum(n));
  }
  /* 和を計算するメソッド */
  [　　　　　　　　　　　　　　　　]
}
```

3 自然数 n, k に対して n 個から k 個を選ぶ組み合わせの数 ${}_nC_k$ を表示するプログラムを書け。ただしクラス名を `Combination` とする。また ${}_nC_k$ の値を求めるためのメソッド名を `binom` とし、その定義は再帰的に行うこと。
ヒント：組み合わせの数 ${}_nC_k$ は k の値に応じて次をみたす。

$$ {}_nC_k = \begin{cases} 0 & (k > n) \\ 1 & (k = 0) \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & (\text{それ以外}) \end{cases} $$

この関係は「n 個から k 個を選ぶ」方法は、n 個のなかの最初の 1 個に注目すると「それを選び、残りの $n-1$ 個から $k-1$ 個を選ぶ」か「それを選ばす、残りの $n-1$ 個から k 個を選ぶ」しかないことを意味している。

4 プログラミングに関する質問を記せ。

何でわざわざ Java で書くの？的プログラム。またはプログラミングの基本（その 3）： 今回は「処理の繰り返し」と「配列」について実習します。これもまた C 言語とほとんど同じ書き方をします。

- 処理の繰り返し
  コンピュータは一定の仕事を繰り返して実行し続けるのが得意であり、そのような仕事に高速計算の威力を発揮します。繰り返しの基本構造には 2 つの種類があります。ひとつは、繰り返しの実行回数を定めておくものであり、これには `for` 文を用います。もうひとつは、繰り返しのたびに条件判断を行って続行や打ち切りを決定するものであり、これには `while` 文を用います。

– `for` 文
  階乗の値を出力するプログラムです。前回の実習では階乗を計算するのに `fact` というメソッドを定義しましたが、今回は `fact` を変数として定義しています。

```
/* 自然数 n を読み込んで、階乗 n! の値を返す。*/
import java.util.Scanner;
class Factorial2 {
  public static void main(String[] args) {
    System.out.println("自然数を入力せよ。");
    Scanner yomu = new Scanner(System.in);
    int n = yomu.nextInt();
    long fact = 1;
    for (int i = 2; i <= n; i++) {
      fact = i * fact;
    }
    System.out.println(n + "の階乗は" + fact);
  }
}
```

  `i++` は変数 `i` の値を 1 増やすことを意味しています。あるいは `++i` と書いても構いません。どちらもオペランドの値をインクリメントすることを意味しています。

– `while` 文
  100 以下の自然数値を当てさせるゲームです。正解するまで入力を要求します。いつ正解するのか分からないので `for` 文ではなく `while` 文を使って処理を繰り返しています。

```
/* 数当てゲーム。*/
import java.util.Random;
import java.util.Scanner;
class Kazuate {
  public static void main(String[] args) {
    Random ransuu = new Random();
    /* 100 以下の勝手な自然数を atari に代入 */
    int atari = 1 + ransuu.nextInt(100);
    System.out.println("数当てゲームです。100 以
下の自然数を当てて下さい。");
    System.out.println("いくつ？");
    Scanner yomu = new Scanner(System.in);
    int n = yomu.nextInt();
    while (n != atari) {
      if (n > atari) {
        System.out.println("もっと小さい数。");
        n = yomu.nextInt();
      }
      else {
        System.out.println("もっと大きな数。");
        n = yomu.nextInt();
      }
    }
    System.out.println("当たり。");
  }
}
```

- 配列

データ処理では、複数個のデータをまとめて取り扱う必要があります。たとえば 5 人分のテストの点数を扱うことを考えます。

```
/* 5 人分のテストの平均点を求める。*/
class MeanValue {
  public static void main(String[] args) {
    int[] scores = new int[5];
    scores[0] = 50;
    scores[1] = 55;
    scores[2] = 70;
    scores[3] = 65;
    scores[4] = 80;
    int sum = 0;
    for (int i = 0; i < scores.length; i++) {
      sum = sum + scores[i];
    }
    float mean = (float)sum / scores.length;
    System.out.println("平均点は" + mean);
  }
}
```

このプログラムでは、配列の宣言をして `new` 演算子により要素を確保し、各要素に値を代入するまでを別々の命令文で実行しました。これらの処理は次のように 1 行で済ませることもできます。

```
int[] scores = {50, 55, 70, 65, 80};
```

この処理を配列の初期化といいます。また、配列を参照している変数に `.length` をつけると、その配列に含まれる要素の数が取得できます。次に、このプログラムをすこし書き換えて、点数を

```
$ java MeanValue2 50 55 70 65 80
```

のようにコマンドの引数として入力できるようにしてみましょう。

```
/* コマンドライン引数に与えられたデータの平均値を求め
る。ただしデータは空白で区切られた整数たち。*/
class MeanValue2 {
  public static void main(String[] args) {
    int n = args.length;
    int sum = 0;
    for (int i = 0; i < n; i++) {
      sum = sum + Integer.parseInt(args[i]);
    }
    System.out.println("データは" + n + "個。");
    float mean = (float)sum / n;
    System.out.println("平均値は" + mean);
  }
}
```

`main` メソッドの引数は `String` の配列なので、たとえば引数で渡された `50` は数値の `50` ではなく、`5` という文字と `0` という文字が連結した文字列として扱われます。この文字列を `int` 型に変換するためには `Integer` クラスの `parseInt` メソッドを使います。

蛇足： 上記 2 番目のプログラムではコマンドの引数としてデータを入力しましたが、ふつうはデータを別のファイルに記録しておきそのファイルを読み込むようにします。1 行「`50 55 70 65 80`」とだけ書いたファイル `data.txt` を用意します。Linux ならば

```
$ java MeanValue2 ‘cat data.txt‘
```

とすると「`java MeanValue2 50 55 70 65 80`」と入力したのと同じ意味になります。逆引用符 ‘ は、その中に書かれたコマンドを実行し、その結果をその位置に書き込む働きがあります。引用符 ’ とは意味がまったく違いますので注意しましょう。一方、同じことを Windows のコマンドプロンプトでやろうと思うと少し面倒ですが、たとえば次のようにするとできます。

```
$ for /f "tokens=*" %i in (data.txt) do java MeanValue2 %i
```

プログラミング II 実習課題（3 回目・その 1）　　ペア番号 [　　]　学籍番号の下 2 桁 [　][　][　][　]

1 ある教室で 5 点満点の試験を行ったところ、受験生たちの点数は以下のようになった。

3 1 3 4 4 1 1 4 5 3 3 2 4 3 5 2 1 4 1 4 0 5 4 0

これに基づき右のような分布図を出力するプログラムが書きたい。ソースコードの空欄を埋めよ。

```
0:**
1:*****
2:**
3:*****
4:*******
5:***
```

```
class Distribution {
    public static void main(String[] args) {
        int[] scores =  {3, 1, 3, 4, 4, 1, 1, 4, 5, 3, 3, 2, 4, 3, 5, 2, 1, 4, 1, 4, 0, 5, 4, 0};
        int[] heads = [　　];  /* 各点数の人数を記録するために要素数 6 の配列を確保 */
        for (int i = 0; i < [　　]; i++)
            heads[i] = 0;  /* どの点数も 0 人で初期化 */
        /* 各点数の人数をカウントする。例えばもし 3 点の人がいたら heads[3] の値をひとつ増やせばいい */
        for (int i = 0; i < [　　]; i++)
            heads[scores[[　　]]]++;
        /* 分布図を出力 */
        for (int i = 0; i < [　　]; i++) {
            System.out.print(i + ":");
            for (int j = 0; j < [　　]; j++)
                System.out.print("*");
            System.out.println("");
        }
    }
}
```

2 自然数 n に対して「もし n が偶数なら 2 で割り、そうでなければ 3 倍して 1 を足す」という操作を繰り返すと、有限回で 1 に到達するであろうことが予想されている（コラッツ予想。数学における未解決問題のひとつ）。たとえば 3 から始めると $3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$ となり 7 ステップで 1 に到達する。キーボードから入力した自然数 n に対して、それが 1 になるまでのステップ数を表示するためのプログラムを次のように書いた。空欄を埋めよ。

```
import java.util.Scanner;
class Collatz {
    public static void main(String[] args) {
        System.out.print("自然数＝");
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        System.out.println(howmanysteps(n));
    }
    public static int howmanysteps(int n) {
        [ ここに書くべき内容は右の余白部分に記せ。 ]
    }
    public static int collatz(int n) {
        if ([　　]) return n/2;
        else return 3*n + 1;
    }
}
```

3 プログラミングに関する質問を記せ。

プログラミング II 実習課題（3 回目・その 2・**提出は任意**）　　ペア番号 [ ]　　学籍番号の下 2 桁 [ ][ ] [ ][ ]

自然数の組 m, n に対してその最大公約数を $\gcd(m,n)$ と書く。このとき方程式 $am + bn = \gcd(m,n)$ をみたす整数 a, b が必ず存在することが知られている。そのような整数解 a, b を具体的にひとつ求めるためには「ユークリッドの互除法」が有効である。ユークリッドの互除法のアルゴリズムのひとつに次のものがある。

開始　ベクトル $\boldsymbol{u} = (u_0, u_1, u_2)$, $\boldsymbol{v} = (v_0, v_1, v_2)$ を用意する。ただし $\boldsymbol{u}$ の初期値を $(u_0, u_1, u_2) = (1, 0, m)$ とし、$\boldsymbol{v}$ の初期値を $(v_0, v_1, v_2) = (0, 1, n)$ とする。

ループ　u_2 と v_2 がどちらも 0 でない限りつぎの操作を続ける：　もし $u_2 > v_2$ ならば $\boldsymbol{u} - \boldsymbol{v}$ を改めて $\boldsymbol{u}$ とおき、そうでなければ $\boldsymbol{v} - \boldsymbol{u}$ を改めて $\boldsymbol{v}$ とおく。

終了　もし $u_2 \neq 0$ ならば $u_2 = \gcd(m,n)$ でしかも $u_0 m + u_1 n = u_2$ が成り立ち、そうでなければ $v_2 = \gcd(m,n)$ でしかも $v_0 m + v_1 n = v_2$ が成り立つ。

自然数 m, n の入力に対して、方程式 $am + bn = \gcd(m,n)$ の整数解 a, b を一組出力するプログラム `Diophantos.java` が書きたい。ただし整数 m, n はコマンドライン引数として入力するものとし、整数解 a, b は下記の実行結果のような形式でコンソール画面に表示させたい。

```
$ java Diophantos 1976 1217
(93)*1976+(-151)*1217 = 1 = gcd(1976,1217)
```

ソースコードの空欄を埋めよ。

```
public class Diophantos {
    public static void main(String[] args) {
        int m = Integer.parseInt(args[0]);
        int n = Integer.parseInt(args[1]);
        [                                    ]
        System.out.println("("+x[0]+")*" +m+ "+("+x[1]+")*" +n+ " = " +x[2]+ " = gcd("+m+","+n+")");
    }
    public static int[] euclid(int m, int n) {
        [                                    ]
        if (u[2]!=0) {
            return u;
        }
        else {
            return v;
        }
    }
}
```

提出の締切は一週間後（10 月 13 日）の 13:00 とする。

Q & A： 前回までに出された質問に回答します。

- 実行コマンド java のコマンドライン引数としてクラス名のほかに整数値を読み込む必要があるプログラムの場合、main メソッドの引数の型を String[] ではなく最初から int[] と宣言しておけば良いのではありませんか。

– なるほどその通りですが、実はそれはできません。Java では main メソッドの引数は String 型の配列と決められています。

- 次のプログラムの出力が 9 となりません。なぜですか。

```
class ExclusiveOr {
  public static void main(String[] args) {
    byte x = 3;
    System.out.println(x^2);
  }
}
```

– それは Java では ^ が冪乗を意味しないからです。演算子 ^ は排他的論理和演算子といい、式 x^y は x と y の各ビットごとに排他的論理和をとった値を返します。その論理演算はビット単位で次の真理値表にしたがいます。

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

したがって byte 型の整数 3 すなわち 00000011 と、整数 2 すなわち 00000010 に対して、各ビットごとに排他的論理和をとれば 00000001 ですので、式 3^2 の値は 1 となります。

.....

先週までの 3 回の実習で、プログラミングの基本を学びました。今週から少しずつ Java 固有のプログラミング作法を身に付けていきましょう。まずはメソッドからです。

Java によるプログラミング（その 1・メソッド）： メソッドというのは C 言語でいう関数のことでした。まず、先々週の実習で紹介した階乗のプログラムを再掲します。このプログラムにはメソッドが 2 つ定義されており、片方のメソッド (main) がもう片方のメソッド (fact) を呼び出しています。

```
/* 自然数 n を読み込んで、階乗 n! の値を返す。*/
import java.util.Scanner;
class Factorial {
  /* main メソッド */
  public static void main(String[] args) {
    System.out.println("自然数を入力せよ。");
    Scanner yomu = new Scanner(System.in);
    int n = yomu.nextInt();
    System.out.println(n + "の階乗は" + fact(n));
  }
  /* 階乗を計算するメソッド */
  public static long fact(int n) {
    if (n == 0) {
      return 1;
    }
    else {
      return n * fact(n-1);
    }
  }
}
```

これが「メソッドの呼び出し」の分かりやすい例です。この例ではひとつのクラス内でメソッドを呼び出していますが、Java ではさらに他のクラスに属するメソッドをも呼び出すことができます。

```
/* 階乗 n!の値を出力する（メソッド呼び出しの例）。*/
import java.util.Scanner;
class MethodCallExample {
  public static void main(String[] args) {
    Factorial3 f = new Factorial3();
    Scanner scan = new Scanner(System.in);
    System.out.print("階乗 n!を計算します。n=");
    f.n = scan.nextInt();
    f.printfact();
  }
}
class Factorial3 {
  int n;
  long fact() {
    long x = 1;
    for (int i = 2; i <= n; i++) {
      x = i * x;
    }
    return x;
  }
  void printfact() {
    System.out.println(n + "の階乗は" + fact());
  }
}
```

次のことに注目します。

- このプログラムにはクラスが 2 つ定義されています。

– このプログラムは 2 つのクラスから構成されているという点で、いままで見てきたプログラムと異なります。

– ファイル名には main メソッドが含まれるほうのクラス名を用います。つまりこの場合は MethodCallExample.java です。

– クラスファイル *.class はクラスごとに作られます。いまソースファイルは 1 つですが、そこから 2 つのクラスファイル MethodCallExample.class と Factorial3.class が生成されます。コマンド java で実行するのは MethodCallExample のほうです。

– このように複数のクラスの宣言をひとつのファイルの中に書くことができる一方で、これらクラスはそれぞれ別々のファイルに分けて書くこともできます。Java で大きい規模のプログラムを作成するときには「ひとつのファイルにひとつのクラス」を宣言するのが一般的です。そうすればクラスをファイル単位で管理できるというメリットがあります。この実習では、ひとつのファイルで全体が見渡せるように、複数のクラスの宣言をひとつのソースファイルにまとめてしまうことが多くなりますが、複数に分けた例を紹介することもあります。どちらも同じように機能します。

- new 演算子

– クラスは、たこ焼きを焼くためのカタのようなものであって、たこ焼きの実体は、別に生成する必要があります。実体の生成を行うのが new 演算子です。new 演算子によって生成された実体を「インスタンス」と呼びます。また、インスタンス内に格納されている変数を「インスタンス変数」といいます。

- ドットの使い方

– f.n

これは「インスタンスを代入した変数＋ドット＋インスタンス変数の名前」という形をしています。これによってインスタンス変数が参照できます。ドットを日本語の「の」に置き換えて考えると分かりやすいでしょう。

– f.printfact()

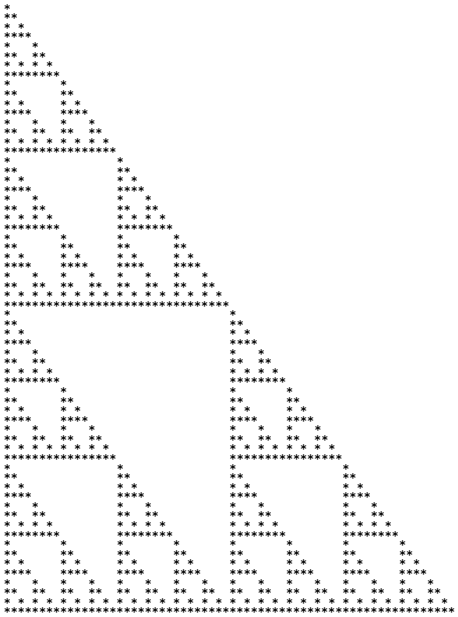
これは「インスタンスを代入した変数＋ドット＋メソッド名」という形をしています。これによってインスタンスにメソッドを実行させることができます。これも「メソッドの呼び出し」といいます。

プログラミング II 実習課題（4 回目）　　ペア番号 [　　]　学籍番号の下 2 桁 [　][　][　][　]

1 自然数 n に対してサイズが $(n+1)\times(n+1)$ の正方行列 $A=(a_{ij})_{i,j=0,\ldots,n}$ を作る。ただし i 行 j 列成分 a_{ij} を

$$a_{ij}=\begin{cases}0 & (\text{二項係数 } {}_iC_j \text{ が偶数のとき})\\ 1 & (\text{二項係数 } {}_iC_j \text{ が奇数のとき})\end{cases}$$

と定める。行列 A を「見る」ために、0 を空白に対応させ、1 を星印に対応させよう。この対応のもとで行列 A を表示すると、たとえば $n=63$ のときは次のようになる。



自然数 n の入力に対して、このような図形を描くプログラムが書きたい。空欄を埋めよ。

```
class Sierpinski {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    draw(n+1);
  }
  public static void draw(int m) {
    Combination f = new Combination();
    for (int i=0; i<m; i++) {
      [                                ]
    }
  }
}
```

ただしクラス Combination は次のものとする（第 2 回目の実習課題を参照のこと）。

```
class Combination {
  public static int binom(int n, int k) {
    if (k>n) return 0;
    else if (k==0) return 1;
    else return binom(n-1,k-1)+binom(n-1,k);
  }
}
```

2 前問のプログラム Sierpinski は、入力する自然数の値をすこし大きくしただけで実行時間が爆発してしまう。これはクラス Combination において二項係数が再帰的に定義されているためである。そこで今回は二項係数を配列として定義し直してみよう。二項係数 ${}_nC_k$ の表

n \ k	0	1	2	3	4	5	6
0	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
2	1	2	1	0	0	0	0
3	1	3	3	1	0	0	0
4	1	4	6	4	1	0	0
5	1	5	10	10	5	1	0
6	1	6	15	20	15	6	1

を利用すれば、たとえば次のように書ける。

```
class Combination2 {
  int n;
  public int[][] binom() {
    int size = n+1;
    int[][] a = new int[size][size];
    for (int i=0; i<size; i++) {
      a[i][0] = 1; /* 0 列目は 1 */
      a[i][i] = 1; /* 対角線は 1 */
      for (int j=i+1; j<size; j++)
        a[i][j] = 0; /* 上三角行列は 0 */
      for (int j=1; j<i; j++)
        a[i][j] = a[i-1][j-1]+a[i-1][j];
    }
    return a;
  }
}
```

このクラス Combination2 を利用して、前問の図形を描くプログラムを書け（裏面に記すこと）。ただしクラス名を Sierpinski2 とする。

3 プログラミングに関する質問を記せ。

Q & A： 前回までの実習で出された質問に回答します。

- switch 文において、条件判定のための制御式は、その値が整数でなければならないのですか。

– そうです。より正確に言うと整数型でなければなりません。具体的には char, byte, short, int, Character, Byte, Short, Integer, enum のいずれかである必要があります。enum は列挙する (enumerate) という意味です。というわけで enum を利用して次のように書けば String 型で switch したような気になれます。このプログラムの出力は「BLUE は青」です。

```
/* 列挙型を利用した switch 文の例。*/
class EnumExample {
    enum Color {BLUE, RED};
    public static void main(String[] args) {
        translate(Color.BLUE);
    }
    public static void translate(Color x) {
        switch (x) {
        case BLUE:
            System.out.println(x + "は青");
            break;
        case RED:
            System.out.println(x + "は赤");
            break;
        default:
        }
    }
}
```

.....

Java によるプログラミング（メソッドの続き）： 今日はコンストラクタについて実習します。まずは先週の復習をしましょう。

```
/* インスタンス・インスタンス変数・メソッド呼び出し */
class MethodCallExample3 {
   public static void main(String[] args) {
      Position p = new Position();
      p.x = 1;
      p.y = 2;
      p.printPosition();
   }
}
class Position {
   int x;
   int y;
   void printPosition() {
      System.out.println("(" + x + "," + y + ")");
   }
}
```

このプログラムでは、まず Position クラスのインスタンスを生成して、次に 2 つのインスタンス変数に値を代入しています。どうせなら、インスタンスを生成するときに、インスタンス変数の初期化も込めて

```
Position p = new Position(1, 2)
```

と書けたら便利です。ところが Position はクラス名なので引数がとれません。なのでこのように書くことはできません。でもやはり Position(1, 2) と書きたい。これを実現する仕組みがコンストラクタです。コンストラクタには次の特徴があります。

- クラス名と同じ名前をもった特殊なメソッドです。
- 引数をとることができます。
- 戻り値はありません。型も指定できません。(ので正しくは、コンストラクタはメソッドではありません。また、メソッド呼び出しのようにコンストラクタを呼び出すこともできません。)

先の例にコンストラクタを追加すると次のようになります。

```
/* コンストラクタの例。*/
class MethodCallExample4 {
   public static void main(String[] args) {
      Position2 p = new Position2(1, 2);
      p.printPosition();
   }
}
class Position2 {
   int x;
   int y;
   Position2(int x1, int y1) {
      x = x1;
      y = y1;
   }
   void printPosition() {
      System.out.println("(" + x + "," + y + ")");
   }
}
```

これで目的が達成できました。さて、次にもうひとつ新しいことを覚えましょう。上のプログラムではコンストラクタの仮引数に x1, y1 という名前を付けました。でも、この名前を単に x, y と書くことがもし許されるのなら、そのほうが嬉しい。なぜなら

- 仮引数の名前を何にするのかいちいち悩まなくて済むし、
- どの変数に値を代入するための引数なのかが分かりやすくなる

から。次のプログラムを見てください。this が新しい言葉です。

```
class MethodCallExample5 {
   public static void main(String[] args) {
      Position3 p = new Position3(1, 2);
      p.printPosition();
   }
}
class Position3 {
   int x;
   int y;
   Position3(int x, int y) {
      this.x = x;
      this.y = y;
   }
   void printPosition() {
      System.out.println("(" + x + "," + y + ")");
   }
}
```

この特別な変数 this が指し示しているものは、インスタンスです。もっと詳しく言うと、メソッドとコンストラクタは、自分を起動したインスタンスを変数 this で参照することができます。

```
/* this の例。*/
class MethodCallExample6 {
   public static void main(String[] args) {
      ThisExample ex = new ThisExample();
      ex.printValues();
   }
}
class ThisExample {
   int x = 10;
   void printValues() {
      int x = 0;
      System.out.println(x);
      System.out.println(this.x);
   }
}
```

プログラミング II 実習課題（5 回目）　　ペア番号 [　]　学籍番号の下 2 桁 [　][　][　][　]

1 以下の問に答えよ。

(1) 平面上の点の位置情報、および位置情報の出力機能をもつクラス Point を次のように宣言した。空欄にコンストラクタを追加せよ。

```
class Point {
  double x, y;
  [                    ]
  public void print() {
    System.out.println("("+ x +","+ y +")");
  }
}
```

(2) Point クラスを利用したアプリケーションを次のように作った。これはコマンドライン引数で指定した個数ぶんの座標をランダムに生成し、その結果をコンソール画面に出力するものである。空欄を埋めよ。

```
import java.util.Random;
class PointApp {
  public static void main(String[] args) {
    int num = Integer.parseInt(args[0]);
    printPoints(num);
  }
  /* m 個の点をランダムに作って座標を表示 */
  public static void printPoints(int m) {
    Point[] q = makePoints(m);
    for (int i=0; i<m; i++)
      [                    ]
  }
  /* n 個の点をランダムに作る */
  public static Point[] makePoints(int n) {
    Random rand = new Random();
    Point[] p = [                    ]
    for (int i=0; i<n; i++) {
  /* nextDouble() は 0 以上 1 以下の実数を生成 */
      double x = 20*rand.nextDouble()-10;
      double y = 20*rand.nextDouble()-10;
      p[i] = [                    ]
    }
    [                    ]
  }
}
```

2 平面上に n 個の点 $(x_i, y_i)_{i=0,\ldots,n-1}$ が与えられたとき、それらの点をすべて通るような $n-1$ 次の多項式

$$y = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_{n-1}(x - x_0)(x - x_1)\cdots(x - x_{n-2})$$

が存在する。係数 $a_0, \ldots, a_{n-1}$ を求めるアルゴリズムのひとつに「ニュートン補間」がある。以下の問に答えよ。

(1) ニュートン補間とはどのようなアルゴリズムか調べよ。

(2) 補間多項式の $x = t$ での補間値を求めるためのクラスを次のように書いた。

```
class Interpolation {
   Point[] p;
   public double newton(double t) {
      int n = p.length;
      double[] a = new double[n];
      double[] x = new double[n];
      double[] y = new double[n];
      for (int i=0; i<n; i++) {
         x[i] = p[i].x;
         y[i] = p[i].y;
      }
      for (int i=0; i<n; i++) {
         for (int j=i-1; j>=0; j--)
            y[j] = (y[j+1]-y[j])/(x[i]-x[j]);
         a[i]=y[0];
      }
      double f = a[n-1];
      for (int i=n-2; i>=0; i--)
         f = a[i] + f*(t-x[i]);
      return f;
   }
}
```

ランダムに生成した n 個の点に対して、それらを補間する $n-1$ 次多項式を $f(x)$ としたとき、補間値 $f(1), f(2), \ldots, f(n)$ を出力するプログラムを書け（裏面に）。ただしクラス名は InterpolationApp とする。

3 プログラミングに関する質問を記せ。

Q & A： 前回までの実習で出された質問に回答します。

- Factorial3 クラス（4 回目の実習資料参照）を使って $n!$ を計算したところ $n \geq 21$ で値がおかしくなりました。コンピュータでは計算できないということですか。

– Factorial3 クラスでは階乗の値を long 型で宣言していました。long 型が扱えるのは 64 ビット符号付き整数、すなわち -2^{63} 以上 $2^{63}-1$ 以下の整数です。21! はそれを超えるため値がおかしくなります。桁数に注目すると $2^{63}-1$ と 20! はともに 19 桁、21! は 20 桁の整数です。さて、だからといって「コンピュータは 21! の値が計算できない」とは言えません。たとえば Java では次のような任意精度の演算が用意されています。

```
import java.math.BigDecimal;
class BigDecimalExample {
  public static void main(String[] args) {
    Factorial4 f = new Factorial4();
    f.n = Integer.parseInt(args[0]);
    f.printfact();
  }
}
class Factorial4 {
  int n;
  BigDecimal fact() {
    BigDecimal x = BigDecimal.valueOf(1);
    for (int i = 2; i <= n; i++) {
      x = x.multiply(BigDecimal.valueOf(i));
    }
    return x;
  }
  void printfact() {
    System.out.println(n +"の階乗は"+ fact());
  }
}
```

- 次のプログラムをコンパイルしたら「<identifier> がありません」というエラーが出ました。このプログラムのどこが間違っているのですか。

```
class MethodCallExample7 {
   public static void main(String[] args) {
      IdentifierError x = new IdentifierError();
      x.width = 2; x.height = 3;
      x.printArea();
   }
}
class IdentifierError {
   int width, height, area;
   area = width * height;
   void printArea() {
      System.out.println(area);
   }
}
```

– identifier とは識別子のことです。メソッドの外にプログラムを書いた場合（や public を pubric と書き間違えた場合など）に、このエラーが出ます。下から 5 行目がエラーの原因です。クラス内では変数の宣言とメソッドの宣言だけができ、通常のプログラムはすべてメソッドの内部に書きます。

.....

Java によるプログラミング（その 2・継承）： 先週までに学んだのは、メソッド、メソッド呼び出し、インスタンス、インスタンス変数、コンストラクタです。このように新しい言葉が次々に出てくるので（今後も出てきます）、よく整理しながら進んで下さい。オブジェクト指向によってプログラムを作るとき、重要になる概念が 3 つあります。それは「継承」と「ポリモーフィズム」と「カプセル化」です。今日は継承について実習します。

これから新しく作成しようとするクラスに、今までに作ったクラスと共通点が多い場合、共通する部分を再利用できれば効率的です。オブジェクト指向の言語では既存のクラスの機能を再利用しそれを拡張することで新しいクラスが作成できます。この仕組みのことを「継承」といいます。例を見ましょう。また二項係数です。ただし二項係数をこのようにプログラムするのはまったく馬鹿馬鹿しいことなので、してはいけません。

```
/* ふたつの自然数を読み込んで、組み合わせの数を返す。*/
class MethodCallExample8 {
   public static void main(String[] args) {
      Combination3 c = new Combination3();
      c.n = Integer.parseInt(args[0]);
      c.k = Integer.parseInt(args[1]);
      c.printcombinat();
   }
}
class Combination3 extends Factorial3 {
   int n, k;
   long combinat() {
      super.n = this.n;
      long x = super.fact();
      super.n = this.k;
      long y = super.fact();
      super.n = this.n - this.k;
      long z = super.fact();
      return x/y/z;
   }
   void printcombinat() {
      System.out.println(combinat() + "通り");
   }
}
```

あるクラスを継承したクラスを宣言するときには extends キーワードを使います。例えば「class B extends A」と書いた場合

- クラス A はクラス B のスーパークラス（親クラス）である
- クラス B はクラス A のサブクラス（子クラス）である
- クラス B はクラス A を継承したクラスである
- クラス B はクラス A から派生したクラスである

などと言います。いろいろな表現をしますがどれも同じことを意味しています。サブクラスはスーパークラスの変数とメソッドを引き継ぎます。サブクラスからスーパークラスのメソッドや変数を呼び出すときには super キーワードを使います。

もし、スーパークラスが持つメソッドと同じ名前のメソッドをサブクラスで宣言した場合は、スーパークラスのメソッドの内容がサブクラスのメソッドの内容によって上書きされます。これを「オーバーライド」といいます。上書き (overwrite) ではなくオーバーライド (override) と表現するので注意しましょう。英語の override は、…に優先する、とか、くつがえす、という意味です。

```
/* 継承およびオーバーライドの例。 */
class OverrideExample {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    a.printX(); a.printY(); b.printX(); b.printY();
  }
}
class A {
  void printX() {System.out.println("AX");}
  void printY() {System.out.println("AY");}
}
class B extends A {
  void printX() {System.out.println("BX");}
}
```

プログラミング II 実習課題（6 回目）　　ペア番号 [　]　学籍番号の下 2 桁 [　][　]

1 友人が靴屋を経営することになり、その顧客情報を管理するためのプログラム作成を依頼された。そこで手始めに、顧客の名前の情報と、それを出力する機能をもつ CustormerCard クラスを作った。以下、ソースコードの空欄を埋めよ。

```
class CustomerCard {
    String name;
    [                                                ] /* コンストラクタ */
    void printInfo() {
        System.out.println("氏名：" + name);
    }
}
```

このクラスにさらに靴のサイズの情報を追加したい。ところが、友人は近く帽子屋の経営も始めるという。将来的には靴屋と帽子屋の 2 店舗をあわせて顧客管理する必要があるだろう。このクラスに靴のサイズ、その他の靴についての情報（購入履歴など)、帽子のサイズ、その他の帽子についての情報、などをすべて盛り込むのはよくない。なぜなら、もしこのようにすると、対応する店舗の種類が増えるごとにインスタンス変数がひたすら増えていくことになり、しかも、その多くはほかの店舗には必要ない情報となってしまうからである。「顧客の名前」は靴屋にも帽子屋にも必要な内容だから CustormerCard クラスをスーパークラスとし、そのサブクラスとして、靴屋専用の情報を持った ShoeCustomerCard クラスを作ることにする。

```
class ShoeCustomerCard [                                ] {
    double shoeSize;
    /* コンストラクタ */
    ShoeCustomerCard(String name, double shoeSize) {
        super(name); /* ← 説明しませんでしたが、これがスーパークラスのコントラクタの呼び出しかたです。*/
        [                                ]
    }
    /* 顧客情報を出力するメソッド。スーパークラスの printInfo メソッドをオーバーライドする。*/
    [                                ] {
        [                                ] /* まずスーパークラスの printInfo メソッドを実行 */
        System.out.println("靴のサイズ：" + shoeSize);
    }
}
```

最後に、実際に顧客情報を登録および閲覧するために CustomerManagement クラスを作った。右欄がその実行結果である。

```
class CustomerManagement {
    public static void main(String[] args) {
        ShoeCustomerCard[] cards = new ShoeCustomerCard[10];
        cards[0] = [                                ]
        cards[1] = [                                ]
        for (int i=0; i<cards.length; i++) {
            if (cards[i]==null) break;
            System.out.println(i + "番目の顧客カードの情報");
            [                                ]
            System.out.println("----------------");
        }
    }
}
```

```
$ java CustomerManagement
0 番目の顧客カードの情報
氏名：松浦望
靴のサイズ：25.5
----------------
1 番目の顧客カードの情報
氏名：保坂亮介
靴のサイズ：28.0
----------------
$
```

2 プログラミングに関する質問を記せ。

Java によるプログラミング（その 3・ポリモーフィズム）： 前回は継承について実習しました。いま Person クラスをスーパークラスとし、そのサブクラスとして Student クラスと Teacher クラスが宣言されているとします。

```
class Person {…}
class Student extends Person {…}
class Teacher extends Person {…}
```

このとき、たとえば Student クラスのインスタンスを生成し、それを何か適当な名前の変数に代入することができました。

```
Student s = new Student();
```

実はこれを次のように書くこともできます。

```
Person s = new Student();
```

変数の型 (Person) と、代入するインスタンスの型 (Student) が一致していません。どうしてこのような書き方が許されるのでしょうか。Student クラスは Person クラスから派生したものなので Person クラスの持っている機能をすべて持っています。だから Student クラスのインスタンスを Person 型の変数に代入することができるのです。これについてはあとで具体例を見ましょう。逆の書き方は許されません。実際

```
Student s = new Person();
```

はコンパイルエラーになります（この理由を考えてみましょう。いまメソッド method が Student クラスだけに備わっており Person クラスには無いとします。変数 s は Student 型なので s.method() と書けば、本来ならば method メソッドが呼び出せるはずです。ところが実際には、変数 s に Person 型のインスタンスが入ってしまっているため s.method() は実行不可能となってしまいます。）

さて、このように、スーパークラスを型とする変数に、サブクラスのインスタンスが代入できることのメリットは何でしょうか。メリットをふたつ挙げます。

- 配列

– 配列で複数のインスタンスを管理するときに役立ちます。たとえば Student クラスのインスタンスと Teacher クラスのインスタンスがそれぞれ複数あるときに、それらを全部まとめて Person 型のインスタンスに代入してしまうことができます。たとえば

  ```
  Person[] persons = new Person[3];
  persons[0] = new Person();
  persons[1] = new Student();
  persons[2] = new Teacher();
  ```

  のように書くことができます。

- ポリモーフィズム

– ポリモーフィズム (polymorphism) とは、多態性や多相性などと訳される言葉です。あるいは生物学においては多型性と訳され、たとえばイミダス（という現代用語辞典）で「一塩基変異多型」(single nucleotide polymorphisms) を引くと次のような説明があります。

> ヒトの遺伝子は基本的に誰でも同じであるが、実際には個人により微妙に異なっている。この微妙な差を多型といい、数百個の塩基からなる遺伝子中で塩基一つだけが異なっていることから一塩基変異多型（SNPs）とよばれる。

  ポリモーフィズムは、プログラミングの世界では「多様な動作をする」という意味で用いられます。具体的には「同じ型の変数に代入されたインスタンスに対して同じ名前のメソッドを呼び出しているのに、実際に参照されているインスタンスの種類によって異なる動作をすること」を指します。

ポリモーフィズムの仕組みを理解するために Person クラスと Student クラスと Teacher クラスのそれぞれに selfintroduction（自己紹介）という名前のメソッドを追加してみましょう。同じ名前のメソッドですがそれぞれ処理する内容が異なります。また Teacher クラスにだけ duty（職務）メソッドを追加しておきます。

```
class Person {
    void selfintroduction() {
        System.out.println("人間です。");
    }
}
class Student extends Person {
    /* スーパークラスのメソッドをオーバーライド */
    void selfintroduction() {
        System.out.println("学生です。");
    }
}
class Teacher extends Person {
    /* スーパークラスのメソッドをオーバーライド */
    void selfintroduction() {
        System.out.println("教員です。");
    }
    /* Teacher クラス固有のメソッド */
    void duty() {
        System.out.println("授業をします。");
    }
}
```

次のクラスは、これらのクラスを利用するためのクラスです。

```
class PolymorphismExample {
    public static void main(String[] args) {
        Person[] persons = new Person[3];
        persons[0] = new Person();
        persons[1] = new Student();
        persons[2] = new Teacher();
        for (int i = 0; i < persons.length; i++) {
            persons[i].selfintroduction();
        }
    }
}
```

この PolymorphismExample クラスの実行結果は

```
人間です。
学生です。
教員です。
```

となり、それぞれのインスタンスが持つ selfintroduction メソッドが適切に実行されたことが確認できます。変数 persons[0], persons[1], persons[2] の型はどれも Person ですが、それぞれのメソッドを呼び出したとき、変数の型がすべて同じなのにもかかわらず、変数が参照しているインスタンスのメソッドがきちんと実行されました。これがポリモーフィズムです。もう一度まとめると、ポリモーフィズムとは「スーパークラスの型に入ったインスタンスに対してメソッドを呼び出したときに、そのインスタンスで宣言されたメソッドが実行され、インスタンスの種類によって異なる動作をすること」です。

Teacher クラスには duty メソッドを宣言していました。これを実行するには次のように書けばよいですね。

```
Teacher t = new Teacher(); t.duty();
```

しかし上のプログラムのようにスーパークラスの型に入れる場合

```
Person t = new Teacher(); t.duty();
```

としてもコンパイルエラーになり、実行できません。なぜなら Person クラスには duty メソッドがないからです。このような場合には変数 t をキャストし ((Teacher)t).duty() と書くことで duty メソッドが実行できます。なお Teacher クラスでないものを Teacher クラスにキャストすると実行時にエラーが出ます。コンパイル時にはエラーが出ないので注意しましょう。

プログラミング II 実習課題（7 回目）　　ペア番号 [　　]　学籍番号の下 2 桁 [　][　][　][　]

1 先週の実習では、靴屋の経営を始めた友人のために、顧客管理のプログラムとして下記の CustomerCard クラスおよび ShoeCustomerCard クラスを作った。この友人が帽子屋の経営も始めたので、靴屋と帽子屋をあわせて顧客管理したい。

```
class CustomerCard {
   String name;
   CustomerCard(String name) {
      this.name = name;
   }
   void printInfo() {
      System.out.println("氏名：" + name);
   }
}
```

```
class ShoeCustomerCard extends CustomerCard {
   double shoeSize;
   ShoeCustomerCard(String name, double shoeSize) {
      super(name);
      this.shoeSize = shoeSize;
   }
   void printInfo() {
      super.printInfo();
      System.out.println("靴のサイズ：" + shoeSize);
   }
}
```

(1)　靴屋のときと同様に CustomerCard クラスを継承して帽子屋専用の HatCustomerCard クラスを作った。空欄を埋めよ。

```
class HatCustomerCard [　　] CustomerCard {
   double hatSize;
   HatCustomerCard(String name, double hatSize) {
      [　　] /* スーパークラスのコンストラクタを呼び出す */
      [　　] /* インスタンスの hatSize を初期化 */
   }
   void printInfo() {
      [　　] /* スーパークラスの printInfo メソッドを呼び出す */
      System.out.println("帽子のサイズ：" + hatSize);
   }
}
```

(2)　実際に顧客情報を登録および閲覧するためのクラス CustomerManagement2 を作れ。ただし右欄に示したような実行結果が得られるようにせよ。また、プログラム作成にはポリモーフィズムの考え方を用いること。

```
$ java CustomerManagement2
0 番目の顧客カードの情報
氏名：松浦望
靴のサイズ：25.5
----------------
1 番目の顧客カードの情報
氏名：保坂亮介
靴のサイズ：28.0
----------------
2 番目の顧客カードの情報
氏名：白石修二
帽子のサイズ：57.0
----------------
$
```

2 プログラミングに関する質問を記せ。

Java によるプログラミング（その 4・カプセル化）： 今日は、オブジェクト指向によるプログラミングで大切になる 3 つの概念（継承・ポリモーフィズム・カプセル化）のうち、最後のカプセル化について実習します。

Java には、ほかのクラスからあるクラスのフィールドやメソッドへアクセスできなくしたり、それらの属性を定めたりするキーワードがあります。クラスやフィールドやメソッドの宣言で用いるこうしたキーワードのことを「修飾子」といいます。

修飾子	意味
public	ほかのどのクラスからもアクセス可能
protected	サブクラスまたは同じパッケージ内のクラスからのみアクセス可能
なし	同じパッケージ内のクラスからのみアクセス可能
private	同じクラス内からのみアクセス可能
final	クラスの宣言につけるとサブクラスを作れないクラスになり、メソッドの宣言につけるとサブクラスでオーバーライドできなくなり、フィールドの宣言につけると値を変更できなくなる
static	フィールドの宣言につけるとクラス変数になり、メソッドの宣言につけるとクラスメソッドになる

このほかにも abstract, syncronized, transient, native, volatile, strictfp などの修飾子があり（、このうちのいくつかについては後日の実習で説明し）ます。

- アクセス修飾子（public, protected, private）

クラスやフィールド、メソッドへのアクセスを制御するための修飾子を「アクセス修飾子」といいます。

```
class Car {
    private int speed;
    public void speedUp() {
        if (speed < 100) speed++;
    }
}
```

この Car クラスのフィールドには、インスタンス変数 speed があります。変数 speed には修飾子 private が付いているので Car クラスの外からは変数 speed を直接参照することはできません。ただし、同じクラス内で宣言された speedUp メソッドからは変数 speed にアクセスできます。

たとえば、他のクラスから、次のように変数 speed に値を代入しようとするとコンパイルエラーになります。

```
Car car = new Car();
car.speed = 50;
```

もし speed に修飾子 private が付いていなかったらどうでしょうか。外部から直接 speed の値を変更できるため

```
car.speed = 10000;
```

のような自動車としてあり得ない値を設定されてしまう恐れがあります。このように修飾子 private を使って外部から直接アクセスできないようにすることを「隠蔽」といいます。

クラスの中だけで使用するメソッドにも、外部から呼び出せないように修飾子 private をつけておくのが安全です。このように、フィールドやメソッドを適切に隠蔽することを「カプセル化」といいます。カプセル化のメリットは、外部からのアクセスをコントロールできること、クラスの利用者が内部処理の複雑さを意識せずに済むこと、クラスの独立性が高まりクラス内部の仕様変更などが外部に影響しにくくなること、などが挙げられます。

クラスに付けられる修飾子は public のみです。また、複数のクラスの宣言をひとつのファイルの中に記述する場合、public 修飾子をつけられるクラスはひとつだけです。

- static 修飾子

– クラス変数
変数には 2 つの種類があります。インスタンス変数とクラス変数です。今まで実習で取り扱ってきた変数はすべてインスタンス変数です。クラス変数は、主にそのクラスに関する情報や、そのクラスから生成されたインスタンス全部にかかわる情報、あるいはインスタンスの間で共有したい値を扱うために使われます。変数を宣言するとき先頭に修飾子 static を付ければクラス変数になります。クラス変数は、クラスにひとつだけ存在し、インスタンスを生成しなくても使用できる（ので、いきなり「クラス名+ドット+変数名」で参照可能）、などの特徴があります。

```
class Point2 {
  static int counter = 0;
  int x, y;
  Point2(int x, int y) {
    this.x = x;
    this.y = y;
    counter++;
  }
}
```

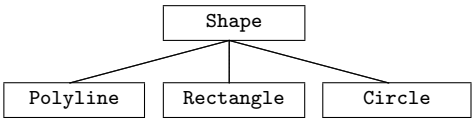
```
class StaticVariableExample {
  public static void main(String[] args) {
    Point2 p[] = new Point2[5];
    for (int i = 0; i < p.length; i++) {
      p[i] = new Point2(i,i);
    }
    System.out.print("生成したインスタンスは");
    System.out.println(Point2.counter + "個。");
  }
}
```

– クラスメソッド
メソッドにもやはり 2 種類あり、修飾子 static が付いていればクラスメソッド、付いていなければインスタンスメソッドです。単に引数を使って計算した結果を返すだけ（インスタンス変数は必要ない）、といったメソッドを宣言する場合などにクラスメソッドを用います。クラスメソッドは、クラス変数と同じように、インスタンスを生成しなくてもいきなり「クラス名+ドット+メソッド名」で呼び出すことができます。たとえば $\sqrt{2}$ の値は Math.sqrt(2) として得られますが、このように書けるのは java.lang.Math クラスに sqrt というクラスメソッドが定義されているからです。なお、クラスメソッドの中からインスタンス変数を参照したり、インスタンスメソッドを呼び出したりすることはできません。

```
class Calc {
  static double getDiskArea(double r) {
    return Math.PI*r*r;
  }
}
```

```
import java.util.Scanner;
class StaticMethodExample {
  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    System.out.print("円の面積を求めます。半径=");
    double r = scan.nextDouble();
    System.out.print("半径" + r + "の円の面積は");
    System.out.println(Calc.getDiskArea(r));
  }
}
```

Java によるプログラミング（その 5・抽象クラスとインタフェース）： 折線と四角形と円が描画できるお絵描きソフトを作ることを考えてみます。図形の情報と、その図形を画面に描画する機能はそれぞれに対応したクラスで宣言します。折線のクラスは Polyline とし、四角形のクラスは Rectangle とし、円のクラスは Circle としましょう。これらの図形クラスには共通する属性（線の太さや色など）があるので、それらを Shape（形）というスーパークラスにまとめます。




```
class Shape {}
class Polyline extends Shape {}
class Rectangle extends Shape {}
class Circle extends Shape {}
```

それぞれのクラスのインスタンスは、次のようにすべて Shape 型の配列に入れて管理できます。

```
Shape[] s = new Shape[3];
s[0] = new Polyline();
s[1] = new Rectangle();
s[2] = new Circle();
```

また、もし各クラスに図形を描画する機能として draw メソッドが宣言されていれば、次のようにポリモーフィズムの仕組みを使って、それぞれの図形を描画することができます。

```
for(int i = 0; i < 3; i++) {
    s[i].draw();
}
```

さて、今後、次のようにして Shape クラスのインスタンスを生成することはあるでしょうか。

```
Shape t = new Shape();
```

Shape クラスは図形クラスに共通する属性をまとめるために作ったのであって、そもそも何か具体的な形を表すものではありません。したがって Shape クラスのインスタンスを生成することはなさそうです。というか、むしろ生成してはいけません。

ならば始めから、インスタンスを生成できないクラスとして Shape クラスを宣言しておけば、誰かが間違ってインスタンスを生成してしまうような事故も防げて、都合がよいでしょう。インスタンスを生成できないクラスのことを「抽象クラス」といいます。修飾子 abstract をつけてクラスの宣言をすれば抽象クラスになります。

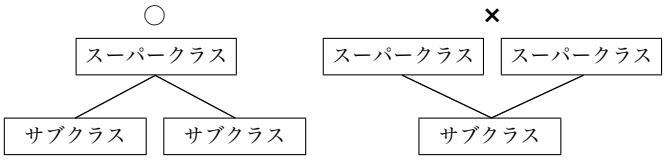
また、メソッドに修飾子 abstract をつけると「抽象メソッド」になります。抽象メソッドは、メソッドの宣言だけがあり、中身がない空っぽのメソッドです。たとえば Shape クラスの中で draw メソッドを宣言するときは、抽象メソッドにします。

```
abstract class Shape {
    abstract void draw();
}
```

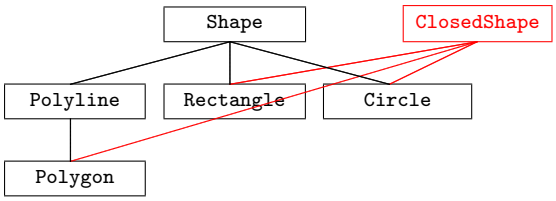
なお、抽象メソッドを含むクラスは必ず抽象クラスにする必要があります。抽象クラスを継承したサブクラスは、抽象メソッドをオーバーライドしない限り抽象クラスのままになります。したがって、スーパークラス Shape を抽象クラスにすることは、サブクラスで確実に draw メソッドをオーバーライドさせるための手段にもなります。

.....

継承には「スーパークラスはひとつしか持てない」という制約がありました。つまり Java は多重継承のできない言語です。



いま、クラス Polyline からさらにサブクラス Polygon（多角形）が派生しているとします（クラス階層をまとめたものが、下図の黒で書かれた部分です）。これらの図形のうち、閉じていて、面積が考えられるのは Rectangle, Circle, Polygon の 3 つです。この 3 つのクラスに共通する機能として、面積を求める getArea メソッドを宣言することを考えた場合、もし getArea メソッドを持つクラス ClosedShape を親クラスとして持つことができれば（下図の赤で書かれた部分です）、ポリモーフィズムの仕組みが使えて便利です。しかし、これでは多重継承になってしまうため Java では許されません。



こうした場合にも対応するため、継承関係にないクラスの間でポリモーフィズムを実現するための手段が提供されています。それが「インタフェース」です。インタフェースは、クラスが持つべきメソッドを記したルールブックのようなものです。インタフェースを使うには次の手順を踏みます。

1. getArea メソッドを持たなくてはならない、というルールを作る（インタフェースの宣言）

```
interface HasGetAreaMethod {
    double getArea(); /* メソッドは宣言のみで中身は空 */
}
```

2. Rectangle, Circle, Polygon のそれぞれが「ルールを守ります」と宣言する（implements 宣言）

3. Rectangle, Circle, Polygon のそれぞれに getArea メソッドを追加する（インタフェースの実装）

```
class Circle extends Shape implements HasGetAreaMethod {
    public double getArea() {
        内容
    }
}
```

なお、インタフェースで宣言したメソッドは、修飾子を何もつけなくても、暗黙のうちに修飾子 public abstract がついているものとして扱われます。したがって、クラスで実装する際には、必ず修飾子 public をつける必要があります。

インタフェースを使うことで、特定のクラスを継承しなくても「このクラスはこのようなメソッドを持っている」ということを約束することができます。これによって、継承関係にないクラスの間でポリモーフィズムの仕組みが使えます。すなわち、インタフェースを型とした変数にそのインタフェースを実装したクラスのインスタンスを代入することができます。

```
HasGetAreaMethod[] closedShape = new HasGetAreaMethod[3];
closedShape[0] = new Rectangle();
closedShape[1] = new Circle();
closedShape[2] = new Polygon();
```

なお、インタフェース自体のインスタンスを作ることはできません。また、インタフェースで宣言した変数は暗黙のうちに public static final になるため、後で変更できない変数、つまり定数になります。

プログラミング II 実習課題（8 回目）　　ペア番号 [　　]　学籍番号の下 2 桁 [　][　][　][　]

1 次のプログラムはどちらもコンパイルエラーになる。適切に修正せよ。

```
class Point {
    int x, y;
    static void addX(int dx) {this.x += dx;}
}
```

```
class Hello {
    public static void main(String[] args) {printHello();}
    void printHello() {System.out.println("Hello");}
}
```

2 次の各問いに答えよ。

(1)　クラス A, B, C が次のように宣言されている。

```
class A {  }
class B extends A {  }
class C {  }
```

以下の代入文のうち誤りがあるものに×印を付せ。

1. `A a = new A();`　2. `A a = new B();`　3. `A a = new C();`　4. `B b = new A();`　5. `B b = new B();`　6. `B b = new C();`

(2)　インタフェース I とクラス A, B, C が次のように宣言されている。

```
interface I {}
abstract class A {  }
class B extends A {  }
class C implements I {  }
```

以下の代入文のうち誤りがあるものに×印を付せ。

1. `A a = new A();`　2. `B b = new B();`　3. `C c = new C();`　4. `I i = new I();`　5. `A b = new B();`　6. `B a = new A();`
7. `I b = new B();`　8. `I c = new C();`

(3)　空欄を埋めてプログラムを完成させよ。

```
interface I {void printMessage(String message);}
class A [                  ] I {
   [                  ] void printMessage([                  ]) {System.out.println(s);}
}
```

3 試験の成績で名簿をソートするために次のクラスを作った。

```
public class Grade implements Comparable {
    public int num;  public String name;  public Integer score;
    public Grade(int num, String name, Integer score) {
        this.num = num;  this.name = name;  this.score = score;
    }
    public void print() {System.out.println("学籍番号:" +num+ "," + "名前:" +name+ "," + "点数:" +score);}
    public int compareTo(Object obj) {return score.compareTo(((Grade)obj).score);}
}
```

このクラスを利用して、実際に数学の点数で名簿をソートするため、次のクラスを作った。

```
import java.util.Arrays;
public class GradeMath {
    public static void main(String args[]) {
        Grade[] math = new Grade[5];
        math[0] = new Grade(1, "佐藤", 80);
        math[1] = new Grade(2, "鈴木", 70);
        math[2] = new Grade(3, "高橋", 50);
        math[3] = new Grade(4, "田中", 60);
        math[4] = new Grade(5, "渡辺", 90);
        Arrays.sort(math);
        for (int i=0; i<math.length; i++) math[i].print();
    }
}
```

「インタフェース Comparable」や「クラス Object」および「Integer クラスの compareTo メソッド」や「Arrays クラスの sort メソッド」などについて調べ、このプログラムでなぜ名簿がソートできるのか、できるだけ詳しく説明せよ。特に compareTo メソッドと sort メソッドの関係に注意すること。

4 プログラミングに関する質問を記せ。

Java によるプログラミング（その 6・例外処理）： プログラム中のスペルの間違いや文法の誤りは、コンパイルの際にエラーとして知らされます。あるいは、コンパイルは問題なくできたものの、実行時にエラーが起こることもあります。前者をコンパイルエラーといい後者をランタイムエラーといいます。次の例はどちらもランタイムエラーを引き起こします。

```
double a = 1;
double b = 0;
double c = a/b;
```

```
int[] a = new int[2];
a[0] = 50;
a[1] = 80;
a[2] = 60;
```

プログラムの実行中に発生するこのようなトラブルや、それを表すものを「例外」といい、例外が起こることを「例外が発生する」とか「例外が投げられる」とか「例外がスロー (throw) される」と表現します。

重要な処理を行うプログラムの場合、なにか問題が起こったからといってプログラムが止まってしまっては困ります。そのため、例外がスローされたときにその例外をキャッチし処理を続行するための仕組みが用意されています。これを「例外処理」といいます。例外処理には次のような try ～ catch 文を使います。

```
try {
    「try ブロック」
    本来実行したい処理を記述する。
    実行途中で例外が投げられる可能性がある。
}
catch(例外の型 変数名) {
    「catch ブロック」
    例外が投げられたときの処理を記述する。
}
finally {
    「finally ブロック」
    例外発生の有無に拘わらず最後に行う処理を記述する。
    省略可能。
}
```

先ほどの 2 種類の例外（ゼロによる割り算、および配列の範囲を超えたインデックスの指定）に対処する例を示します。

```
public class ExceptionExample {
  public static void main(String[] args) {
    int[] score = new int[5];
    int b = (int)(Math.random()*10);
    System.out.println("b=" + b);
    try {
      double c = (double)1/b;
      System.out.println("1/b=" + c);
      score[b] = 1;
      System.out.println("例外は発生しませんでした。");
    }
    catch(ArithmeticException e) {
      System.out.println(e);
      return;
    }
    catch(ArrayIndexOutOfBoundsException e) {
      System.out.println(e);
      return;
    }
    finally {
      System.out.println("finally ブロックの処理です。");
    }
    System.out.println("プログラムを終了します。");
  }
}
```

何回か実行し、その実行結果を観察してください。この例のように、もし catch ブロックの中に return 文がある場合は、そこでメソッドの処理が終わります（それがもし main メソッドの場合はプログラムが終わります）。しかし finally ブロックがあるので、メソッドの処理を終える前に、常に finally ブロックの処理が実行されます。return 文によってメソッドを途中で終えているため try ～ catch 文の外側にある命令は必ずしも実行されないことに注意しましょう。

一般に、返却型を void で宣言したメソッドは値を返さないため return 文は必須ではありませんが、もしメソッドの途中でプログラムの流れを強制的にメソッド呼出し元に戻したい場合は「return;」と、返却値を与えない return 文を実行します。

Java によるプログラミング（その 7・入出力）： プログラムの多くは、プログラム内部で数値や文字列などのデータを扱うだけでなく、データを外部に送り出したり、外部からデータを受け取ったりします。オブジェクト指向言語である Java は、そういったデータの入出力にもクラスを使います。データの流れをクラス InputStream および OutputStream のインスタンスとして扱い、それらに対して読み書きを行うためにクラス InputStreamReader および OutputStreamWriter を用います。

データには大きくわけて文字列データとバイナリデータの 2 種類があり、また、データの入力元・出力先にもファイルやネットワークなどいろいろのものがありますが、以下ではファイルに格納された文字列データを読み出すプログラムについて説明します。この場合、データはクラス InputStream から派生したクラス FileInputStream のオブジェクトとして扱います。Java ではオブジェクト指向の特徴を生かし、入力データ全般について成り立つ性質をクラス InputStream に持たせ、それを継承する形でいろいろな種類の入力データに対応するよう、多くのクラスが用意されています。

ファイルからデータを読み込むには、たとえば以下のような手順を踏みます（いままで見てきたプログラムと同様、実現方法は一通りではありません）。

1. FileInputStream のインスタンスを生成。データが保存されたファイルのファイル名（ファイルパス）を引数にする。

   ```
   InputStream is = new FileInputStream("foo.txt");
   ```

2. InputStreamReader のインスタンスを生成。ステップ 1. で作成したストリームオブジェクト is を引数にする。

   ```
   InputStreamReader isr = new InputStreamReader(is,"SJIS");
   ```

3. BufferedReader のインスタンスを生成。ステップ 2. で作成したストリームオブジェクト isr を引数にする。

   ```
   BufferedReader br = new BufferedReader(isr);
   ```

   英語でバッファ (buffer) とは緩衝装置、つまり 2 つのものの間にはさまって衝撃を和らげる仕組みのことを意味しますが、コンピュータの世界では、何か処理をする前に一時的にデータを蓄えておく“容器”のようなものを指します。たとえばデータをファイルに書き込む場合、書き込み命令があるたびにファイルにアクセスするよりも、ある程度書き込む文字がたまってからまとめて書き込んだほうが効率的に処理できます。同じことはファイルの読み込みにも言え、クラス BufferedReader は文字列を受け取る際のバッファ機能を提供します。

4. ステップ 3. で作成したオブジェクト br の readLine メソッドを使って、文字列データを String 型で受け取る。

   ```
   String line = br.readLine();
   ```

   このメソッド readLine は文字列を 1 行ずつしか読み込まないので、データが複数行にわたる場合は while 文などを用いて、読み込みの処理を繰り返す必要があります。

5. 最後に、ステップ 3. で作成したオブジェクト br の close メソッドでストリームを閉じる。

   ```
   br.close();
   ```

ネットワーク通信（最終日に実習予定です）においても、データのやり取りをここに説明したのと同じような方法で行います。

プログラミング II 実習課題（9 回目）　　ペア番号 [　]　学籍番号の下 2 桁 [　][　][　][　]

1 次のプログラムは、文字列データの記述されたファイル test.txt から 1 行ずつ文字列を読み込み、読み込んだ内容を標準出力に出力する例である。空欄を埋めよ。

```
import java.io.*;
public class FileReadExample {
    public static void main(String[] args) {
        [　　　　　　] {
            InputStream file = [　　　　　　　　　　　　　　　　　　　　]
            InputStreamReader reader = [　　　　　　　　　　　　　　　　　　　　]
            BufferedReader buffreader = [　　　　　　　　　　　　　　　　　　　　]
            String s;
            while((s = [　　　　　　　　　　　　　　　　　　　　]) != null) {
                [　　　　　　　　　　　　　　　　　　　　]
            }
            [　　　　　　].close();
        }
        catch(IOException e) {
            System.out.println(e);
        }
    }
}
```

2 次のプログラムは、文字列データの記述されたファイル test.txt を読み込んで、行の先頭に行番号をつけたものを test2.txt というファイルに出力するプログラムの例である。空欄を埋めよ。

```
import java.io.*;
public class FileWriteExample {
    public static void main(String[] args) {
        try {
            // ファイル入力用のストリームを構築
            FileReader fr = new FileReader("test.txt");
            BufferedReader br = new BufferedReader(fr);
            // ファイル出力用のストリームを構築
            FileWriter fw = new FileWriter("test2.txt");
            BufferedWriter bw = new BufferedWriter(fw);
            String s;
            int lineNum = 1;
            while([　　　　　　　　　　　　　　　　　　　　　　　　　　　　]) {
                bw.write([　　　　　　] + ":" + [　　　　　　] + "¥r¥n");
                [　　　　　　　　　　]
            }
            br.close();
            bw.close();
        }
        catch(IOException e) {
            System.out.println(e);
        }
    }
}
```

Java によるプログラミング（その 8・スレッド）： これまでに作成したプログラムはすべて、プログラムコードに書かれた命令を上から順番にひとつずつ実行して処理を進めていくタイプのものでした。これに対して、複数の命令を同時に実行しながら処理を進めていくタイプのプログラムもあります。前者をシングルスレッドのプログラムといい、後者をマルチスレッドのプログラムといいます。スレッド (thread) とは「糸」とか「議論などの筋道、つながり、脈絡」という意味の言葉です。この言葉は日常的にも使われ、メーリングリストやウェブ上の掲示板では「同一主題についての一連のやりとり」のことをスレッドと呼んでいます。

シングルスレッドのプログラムの場合、前の処理が終わらないと次の処理が始められないため、時間のかかる処理が実行されている間はただ待つしかありません。そのようなプログラムをマルチスレッド化すれば、時間のかかっている処理の進捗状況を画面に表示したり、並行してほかの処理ができたりするので、プログラムの使い勝手や利便性が大きく向上します。マルチスレッドのプログラムを作成するには 2 通りの方法があり、ひとつは「Thread クラスを継承すること」で、もうひとつは「Runnable インタフェースを実装すること」です。ここでは前者について説明します。

- Thread クラスの継承

具体例を見ましょう。簡単なマルチスレッドプログラムです。

```
class MultiThreadExample {
    public static void main(String[] args) {
        MyThread thrd = new MyThread();
        thrd.start();
        for(int i=0; i<10; i++)
            System.out.println("MultiThreadExample
の main メソッド" + i);
    }
}
```

3 行目で MyThread オブジェクトを生成し 4 行目でそのオブジェクトが持っている start メソッドを呼び出しています。これによって新しいスレッドが始まります。この新しいスレッドで実行したい内容はクラス MyThread の run メソッドに書いておきます。

```
class MyThread extends Thread {
    public void run() {
        for(int i=0; i<10; i++)
            System.out.println("MyThread の run メ
ソッド" + i);
    }
}
```

プログラムの実行結果は「クラス MultiThreadExample の main スレッドによる出力」と「クラス MyThread のスレッドによる出力」が入り乱れた状態になります。各スレッドがどのようなタイミングで処理を進めるかはプログラムを実行してみないと分かりません。実行結果は毎回異なります。

新しいスレッドにおいて run メソッドの内容を実行するとき run メソッドを直接呼び出してはいけないことに注意しましょう。呼び出すべきものは start メソッドです。run メソッドは start メソッドを経由して間接的に呼び出されます。

- スレッドの制御

マルチスレッドのプログラムでは複数の処理が同時に進行します。各スレッドが独立して処理を進めるので、どのタイミングで run メソッドに記述した内容の処理が終わるのか、事前に知ることはできません。しかし場合によっては「画像ファイルのダウンロードが終わってからその画像を画面に表示する」というように、あるスレッドの処理が終わってからほかのスレッドの処理を始めたいことがあります。これを実現するのが join メソッドです。スレッド A のなかでスレッド B の join メソッドを呼び出すと、スレッド A はスレッド B の処理が終わるまで待機します。join メソッドは InterruptedException 型の例外を投げることがあるので try ～ catch 文の中に入れておきます。このプログラムは先の MultiThreadExample と違い、まず MyThread オブジェクトの run メソッドの内容がすべて実行され、それがすべて終わってから main スレッドの残りの処理が行われます。

```
class MultiThreadExample2 {
    public static void main(String[] args) {
        MyThread thrd = new MyThread();
        thrd.start();
        try {
            thrd.join();
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        for(int i=0; i<10; i++)
            System.out.println("MultiThreadExample2
の main メソッド" + i);
    }
}
```

- スレッドの同期

マルチスレッドは同時に複数の処理が行える便利な機能ですが、扱いには注意を要します。特に、ひとつの変数に異なるスレッドが同時にアクセスするような場合は処理の結果に不整合が生じる恐れがあります。例を見ましょう。100 人が銀行に預金します。ただし各人が 1 円の預金を 1 万回繰り返します。

```
class MultiThreadExample3 {
    public static void main(String[] args) {
        Customer[] c = new Customer[100];
        /* Customer オブジェクトを 100 個つくりそれぞ
れのスレッドを開始する */
        for(int i=0; i<100; i++) {
            c[i] = new Customer();
            c[i].start();
        }
        /* すべてのスレッドが終了するのを待つ */
        for(int i=0; i<100; i++) {
            try {
                c[i].join();
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        System.out.println(Bank.money);
    }
}
```

```
class Customer extends Thread {
    public void run() {
        for(int i=0; i<10000; i++)
            Bank.addYen();
    }
}
```

```
class Bank {
    static int money = 0;
    static void addYen() {
        money++;
    }
}
```

最終的に money の値は 100 万になるはずですが、このプログラムの実行結果はそうなりません。複数のスレッドが一斉に money の値を参照したりインクリメントしたりするために生じた不整合です。この問題を解決するにはクラス Bank のメソッド addYen にロック機能を持たせればよく、これはメソッド addYen に修飾子 synchronized を付けることで実現できます。

Java によるプログラミング（その 9・GUI）： これまでに作成したプログラムは CUI (Character-based User Interface) アプリケーションでした。今回は GUI (Graphical User Interface) アプリケーションを作ります。GUI のためのクラスライブラリとして Swing ライブラリと AWT (Abstract Window Toolkit) ライブラリがありますが、後者は Swing が登場する以前からある古いライブラリで、実行する OS によって外観が異なるなどの欠点があるため、近年では Swing が主流になっています。

GUI アプリケーションのウィンドウを作成するには、まずフレームを生成し、そのあとで、フレームの上にボタンなどの細かな部品を配置していきます。Swing ライブラリにはフレームを作るためのクラスとして `JFrame` があります。`JFrame` クラスでフレームを作るには次の手順を踏みます。

1. `JFrame` のインスタンスを生成。

   ```
   JFrame frame = new JFrame();
   ```

2. ウィンドウの右上に表示される「×ボタン」が押されたときの動作を `JFrame` クラスの `setDefaultCloseOperation` メソッドで指定。

   ```
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   ```

   このように `JFrame.EXIT_ON_CLOSE` という定数を渡すと、アプリケーションが終了します。

3. `JFrame` クラスの `setSize` メソッドでフレームのサイズを指定。

   ```
   frame.setSize(300,200);
   ```

   とすると横 300 ピクセル、縦 200 ピクセルになります。

4. `JFrame` クラスの `setVisible` メソッドでフレームを表示。

   ```
   frame.setVisible(true);
   ```

次に、フレームの上にボタンやチェックボックスなどの部品を配置します。これらの部品をコンポーネント (component) といいます。

ところで `JFrame` クラスから作成したフレームは 1 枚の板のように見えますが、実際にはペインと呼ばれる層がいくつか重なって出来ています。ペイン (pane) とは英語で「碁盤の目の一区画」や「羽目板・パネル」という意味合いです。ペインには Glass Pane や Content Pane や Layered Pane や Root Pane などがあり、それぞれ別の機能を持っています。ボタンなどのコンポーネントは Content Pane に追加していきます。

そこで `JFrame` クラスのオブジェクトから Content Pane を取り出し、取り出したペインに対してコンポーネントを追加します。Content Pane を取得するには `JFrame` クラスの `getContentPane` メソッドを使います。

```
import javax.swing.*;
public class FrameExample extends JFrame {
  public static void main(String[] args) {
    new FrameExample();
  }
  FrameExample() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().add(new JButton("ボタン"));
    setSize(300,200);
    setVisible(true);
  }
}
```

これを実行すると、ボタンがひとつだけ配置されたフレームが表示されます。この `main` メソッドの中では `new FrameExample()` としてインスタンスを生成しているだけですが、`FrameExample` クラスのコンストラクタで必要な処理が行われているので、実行するときちんとフレームが表示されます。ただしボタンを配置しただけなのでクリックしても何も起こりません。

ウィンドウ内のボタンが押された、キーボードのキーが押された、などアプリケーションに対して何らかの操作が行われたときには、プログラム内に「操作が行われた」という通知が送られます。この通知のことをイベントといいます。通知のあったことを「イベントが発生した」と表現します。プログラム実行時に発生する例外を例外オブジェクトで表したのと同じように、イベントはイベントオブジェクトで表されます。また、イベントが発生したときの処理をイベント処理といいます。イベント処理をプログラムに組み込むことによって、ユーザの操作に応じて動作するアプリケーションが作成できます。

イベント処理を行うプログラムを作成するには、「リスナの作成」と「リスナの登録」というふたつのステップが必要です。

1. リスナを作成する
   イベント発生の通知を受け取ることができるクラスを作成します。`ActionListener` インタフェースを実装する必要があります。イベントが発生したときには、このインタフェースで宣言されている `actionPerformed` メソッドが呼び出されます。インタフェースの実装ではこのメソッドをオーバーライドし、インベントに応じた処理を記述します。

2. ソースにリスナを登録する
   イベントが発生したときに、その通知をだれに送るのかをソース（イベントが発生したオブジェクトのこと。たとえばボタンなど）に登録します。これをリスナ登録といいます。ステップ 1. で作成したオブジェクトをリスナ登録しておけば、イベントが発生したときにそのオブジェクトに通知が送られます。

```
import javax.swing.*;
import java.awt.event.*;
public class ButtonExample extends JFrame implements
ActionListener {
  public static void main(String[] args) {
    new ButtonExample();
  }
  ButtonExample() {
    JButton b = new JButton("ボタン");
    /* ボタンに対して自分自身をリスナ登録する */
    b.addActionListener(this);
    getContentPane().add(b);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300,200);
    setVisible(true);
  }
  public void actionPerformed(ActionEvent e) {
    System.out.println("ボタンが押されました。");
  }
}
```

最後に、直線や円などの各種の図形を画面に描画してみます。Swing で図形を表示する場合、まずコンポーネントの上に図形を描画し、次のそのコンポーネントを上述の方法でフレームにのせる、という手順を踏みます。コンポーネントが画面に表示されるときには `paintComponent` メソッドが実行されるので、このメソッドをオーバーライドして独自の描画命令を記述することにより、好きな図形をコンポーネントの上に描画することができます。

```
import javax.swing.*;
import java.awt.*;
public class GraphicsExample extends JFrame {
  public static void main(String[] args) {
    new GraphicsExample();
  }
  GraphicsExample() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().add(new PanelExample());
    setSize(300,300);
    setVisible(true);
  }
}
class PanelExample extends JPanel {
  public void paintComponent (Graphics g) {
    for (int i = 0; i < 360; i += 30) {
      int x = 150 + (int)(100*Math.cos(i*Math.PI/180));
      int y = 150 + (int)(100*Math.sin(i*Math.PI/180));
      g.fillOval(x-10, y-10, 20, 20);
      g.drawLine(150,150,x,y);
    }
  }
}
```

プログラミング II 実習課題（11 回目）　　ペア番号 [　　]　学籍番号の下 2 桁 [　][　][　][　]

テキストファイルの内容を表示する GUI アプリケーション OpenFile が作りたい。ただし、テキストファイルのファイル名は実行時にコマンドライン引数として与えるようにし（例えば java OpenFile test.txt のように）、ウィンドウ内のボタンを押したときにテキストファイルの内容が表示されるようにしたい。空欄を埋めよ。

```
import [                              ]
import java.awt.*;
import java.awt.event.*;
import [                              ]
public class OpenFile [                                                    ] {
    public static void main(String[] args) {
        new OpenFile([                ]);
    }
    String fileName;
    JTextArea txtArea = new JTextArea(); /* テキストエリアを作成 */
    OpenFile(String name) {
        fileName = name; /* テキストファイルのファイル名を代入 */
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JScrollPane scrl = new JScrollPane(txtArea); /* テキストエリアにスクロールバーを作成 */
        JButton button = new JButton("ファイル" + name + "の内容を表示"); /* ボタンを作成 */
        [                                    ] /* ContentPane にスクロールバーを配置 */
        getContentPane().add(BorderLayout.SOUTH, button);  /* ContentPane にボタンを配置 */
        [                                    ] /* リスナ登録 */
        setSize(300,200);
        [                                    ] /* フレームを表示 */
    }
    public void actionPerformed(ActionEvent ev) {
        try {
            [                                                              ]
        }
        catch(IOException ex) {
            System.out.println(ex);
        }
    }
}
```

ヒント①　try ブロックに記述する内容は、先々週の実習課題を参考にせよ。

ヒント②　テキストエリアに文字列を追加するには JTextArea オブジェクトの append メソッドを使う。

Java によるプログラミング（その 10・ネットワーク通信）： 今回はネットワークを介して情報のやりとりを行うプログラムを作成します。ネットワーク経由でのデータの送受信も、ファイルの入出力と同じようにストリームオブジェクトを介して行います。そのため、いったんネットワーク接続が確立した後は、これまでに学んだデータ入出力の方法と同じようにしてデータの送受信が行えます。

ふたつのプログラムがネットワーク接続によって通信を行う場合、一方をサーバーと呼び、他方をクライアントと呼びます。接続を待ち受け接続要求があればそれに応答する側のプログラムがサーバーで、サーバーに接続要求を出す側のプログラムがクライアントです。クライアントがサーバーに接続要求を出すときには、サーバーの IP アドレスとポート番号を指定しなくてはなりません。IP アドレスは、ネットワークに接続しているコンピュータに割り振られたネットワーク上の所在地情報のことで、現在のところ 32 ビットの数値が一般的に使われています。通常はこれを 8 ビットずつドットで区切り 133.100.211.227 のように 10 進法で表します。ポート番号は、プログラムどうしが通信でデータをやりとりするときに使用する出入り口（すなわちポート）につけられた番号のことで 0 から 65535 までの数字で表されます。ただしこのうち 0 から 1023 までは予約ポートと呼ばれ、使い道が決まっている（たとえば 23 番は SSH のために使われ 80 番は HTTP のために使われる）ので、自分で作成したプログラムでは 1024 以上の値を指定します。

ネットワーク通信を行うときには「ソケット」という仕組みを使います。ソケットは「穴、受け口」の意味で、日常用語としては電球を差し込むときの受け口となる器具（右図）を指します。コンピュータの世界では、ソケットに IP アドレスとポート番号のペアを設定しておき、ソケットに対してデータを読み書きすることでプログラム間での通信（ソケット通信）を行います。ソケット通信を行うために必要となるオブジェクトが Socket と ServerSocket です。まずは簡単なネットワーク通信の例を作ってみましょう。サーバーがクライアントに文字列を送信し、クライアントは受け取った文字列をコンソール画面に出力するプログラムです。ここでは簡単のため、サーバーとクライアントを 1 台のコンピュータの中で動作させることにします。

- サーバーのプログラム

サーバーのプログラムでは、次の手順でネットワークの接続と文字列の送信を行います。

1. ServerSocket オブジェクトを生成。コンストラクタの引数として、このプログラムで接続要求を受け付けるポート番号を指定する（指定した番号が適切でない場合は例外が投げられる）。
2. クライアントからの接続要求を accept メソッドで待つ。
3. Socket オブジェクトを使って、文字列を出力するためのストリームオブジェクトである PrintWriter オブジェクトを生成。
4. PrintWriter オブジェクトを介して文字列の送信を行う。
5. close メソッドでストリームを閉じる。

- クライアントのプログラム

クライアントはサーバーに対して接続要求を送り、接続が確立したらサーバーからの文字列を受信します。このようなプログラムは次の手順で作成します。

1. Socket オブジェクトを生成。コンストラクタの引数としてサーバーの IP アドレスとポート番号を指定する（サーバーへの接続が失敗した場合は例外が投げられる）。
2. Socket オブジェクトを使って、文字列を受け取るためのストリームオブジェクトである BufferedReader オブジェクトを生成。
3. BufferedReader オブジェクトを介してサーバーからの文字列を受信する。
4. close メソッドでストリームを閉じる。

これらを踏まえて作成したサーバーおよびクライアントのプログラムはそれぞれ次のようになります。

```
/* サーバーのプログラム */
import java.io.*;
import java.net.*;
public class SimpleServer {
  public static void main(String[] args) {
    try {
      /* 50000 番ポートでソケットを作成 */
      ServerSocket servsock = new ServerSocket(50000);
      while (true) {
        /* クライアントからの接続要求を待つ */
        Socket sock = servsock.accept();
        /* ストリームを作成 */
        PrintWriter pw
          = new PrintWriter(sock.getOutputStream());
        /* ストリームに文字列を出力 */
        pw.println("私はサーバーです。");
        /* ストリームを閉じる（ソケットが開放される）*/
        pw.close();
      }
    }
    catch (BindException be) {
      System.out.println(be);
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
  }
}
```

```
/* クライアントのプログラム */
import java.io.*;
import java.net.*;
public class SimpleClient {
  public static void main(String[] args) {
    try {
      /* サーバーの IP アドレスとポート番号を指定してソケットを作成。127.0.0.1 はそのコンピュータ自身を表す特別な IP アドレス。ほかのコンピュータに接続する場合は、接続したいコンピュータの IP アドレスを指定する。 */
      Socket sock = new Socket("127.0.0.1", 50000);
      /* ストリームを作成 */
      InputStreamReader isr
        = new InputStreamReader(sock.getInputStream());
      BufferedReader br = new BufferedReader(isr);
      /* ストリームから文字列を受け取る */
      String s = br.readLine();
      System.out.println(s);
      /* ストリームを閉じる */
      br.close();
    }
    catch (ConnectException ce) {
      System.out.println(ce);
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
  }
}
```

プログラムを実行するには、まずサーバーのプログラムを実行し、それからクライアントのプログラムを実行します。クライアントはサーバーからの文字列を受け取り、それをコンソール画面に出力して終了します。サーバーは動作し続けて、別のクライアントからの接続要求を待つ状態になります。

このプログラムでは複数のクライアントからの接続要求を同時に受け付けることを想定していないため、あるクライアントがサーバーに接続している間、ほかのクライアントはサーバーに接続することはできません。これだと、たとえばサイズの大きいデータを送受信する場合や、チャットサーバーのようにクライアントとの接続が長時間にわたって継続される場合などに、サービスの質が低下してしまい不都合です。この問題を解決するためにはスレッドを利用します。

プログラミング II 実習課題（12 回目）　　提出不要

次のふたつのプログラムは、サーバーとクライアントの間でネットワーク通信を行い、サーバが配信する文字列をクライアントのウィンドウに表示する GUI アプリケーションを作成したものである。

```
/* サーバーのプログラム */
import java.io.*;
import java.net.*;
import javax.swing.*;
public class ServerExample extends JFrame {
  public static void main(String[] args) {
    new ServerExample();
  }
  String[] messages = {"あ","い","う","え","お"};
  JTextArea txtArea = new JTextArea();
  ServerExample() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JScrollPane scrl = new JScrollPane(txtArea);
    getContentPane().add(scrl);
    setSize(300,200);
    setVisible(true);
    try {
      ServerSocket servsock = new ServerSocket(60000);
      while (true) {
        Socket sock = servsock.accept();
        PrintWriter pw
          = new PrintWriter(sock.getOutputStream());
        String message = getMessage();
        pw.println(message);
        pw.close();
        /* クライアントに送ったメッセージをテキストエリア
に出力 */
        txtArea.append(message + "\r\n");
      }
    }
    catch (BindException be) {
      System.out.println(be);
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
  }
  String getMessage() {
    /* 配列 messages の要素をランダムに選択 */
    int n = (int)(Math.random()*messages.length);
    return messages[n];
  }
}
```

```
/* クライアントのプログラム */
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
public class ClientExample extends JFrame implements
ActionListener {
  public static void main(String[] args) {
    new ClientExample();
  }
  JTextArea txtArea = new JTextArea();
  ClientExample() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JScrollPane scrl = new JScrollPane(txtArea);
    JButton button = new JButton("メッセージ取得");
    getContentPane().add(scrl);
    getContentPane().add(BorderLayout.SOUTH, button);
    button.addActionListener(this);
    setSize(300,200);
    setVisible(true);
  }
  /* ボタンが押されたときの処理 */
  public void actionPerformed(ActionEvent ae) {
    try {
      Socket sock = new Socket("127.0.0.1", 60000);
      InputStreamReader isr
      = new InputStreamReader(sock.getInputStream());
      BufferedReader br = new BufferedReader(isr);
      String message = br.readLine();
      /* サーバーから受け取ったメッセージをテキストエリア
に出力 */
      txtArea.append(message + "\r\n");
      br.close();
    }
    catch (ConnectException ce) {
      System.out.println(ce);
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
  }
}
```

以下の各課題を実行せよ。

1. 上記のプログラムを実行し、動作を確かめよ。
2. 友人同士でサーバー役とクライアント役を決め、ネットワーク経由でデータを送受信せよ。なお、サーバ役の人が自分のコンピュータに割り振られた IP アドレスを知るには、コマンドプロンプトで「`ipconfig /all`」と入力すればよい（これがもし UNIX 系の OS ならば「`ifconfig`」とする）。
3. サーバー `133.100.211.160` の `12345` 番ポートに接続し、メッセージを受け取れ。ただし、クライアントのプログラムにおいて `InputStreamReader` のインスタンスを生成する際、コンストラクタの第 2 引数として「`"EUC_JP"`」を指定すること。
4. 上記のサーバーのプログラムをマルチスレッド化せよ。ただし既にクラス `JFrame` を継承しているため、クラス `Thread` を重ねて継承することはできない。このような場合はインタフェース `Runnable` を実装することによりマルチスレッド化する。インタフェース `Runnable` の実装方法についてはウェブで調べよ。
5. プログラミングの基本事項（条件分岐、処理の繰り返し、配列）、およびプログラミング II で学んだこと（継承、ポリモーフィズム、インタフェース、クラス変数とクラスメソッド、スレッド等）について復習せよ。