計算機プログラミングI 2005　久野クラス　＃4

久野　靖*

2005.10.28

はじめに

注意! お知らせしてあるように、来週 11/4 は休講で、次回は 11/11 になります。

皆様の感想を拝見していると、繰り返しが難しいという人と楽勝だという人と分かれているようです。もちろん慣れないと難しいだろうと思いますけど、繰り返しが使えないとどうにもならないので頑張ってください。前回は演習問題の解説が長めでしたが、今回はもうすこし簡単にして本題のオブジェクトの話を多くしたいと思います。

ところで、**クラスの名前は「英字で始まり英字と数字とアンダースコア (下線、「_」) だけ使うように**してください。マイナス記号を使ってハマっている人がいるそうですので (当然「x-y」という名前は名前ではなく「x マイナス y」になってしまいます。そしてクラス名のとこには演算は書けません)。

1　繰り返しの練習問題

演習 2a

これは結局、何個データがあったか数えられればできる。数えるには、たとえば変数 count を用意して最初に 0 にしておき、データを 1 つ加えるごとに count は 1 増やすようにすれば、常に「今何個のデータを足した状態か」を追跡できる。ループが終わったらこれで割れば平均が出せる。

```
import java.io.*;

public class r3ex2a {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    double sum = 0.0;
    int count = 0;
    while(true) {
      System.out.print("x> ");
      double x = (new Double(in.readLine())).doubleValue();
      if(x == 0.0) { break; }
      sum = sum + x; count = count + 1;
    }
    System.out.println("sum = " + sum + "; ave = " + (sum/count));
  }
}
```

ループの中で割り算している人がいたが、count はループの中で $1, 2, ...$ と増えていくわけだから

$$\frac{(x_1 + x_2 + x_3 + \ldots + x_N)}{N}$$

*筑波大学大学院経営システム科学専攻

が計算される代わりに

$$\frac{x_1}{1} + \frac{x_2}{2} + \frac{x_3}{3} + \ldots + \frac{x_N}{N}$$

が計算されてしまう。そんなものは平均ではないですよね。

1 つの考え方として、「ループの中身を周回する回数ぶん展開して考える」というものがある。つまり、上のプログラムで 3 回回るのだったら次のようになっていると考えるわけだ。

```
double sum = 0.0;
int count = 0;
// ループ 1 回目
  System.out.print("x> ");
  double x = (new Double(in.readLine())).doubleValue();
  sum = sum + x; count = count + 1;
// ループ 2 回目
  System.out.print("x> ");
  double x = (new Double(in.readLine())).doubleValue();
  sum = sum + x; count = count + 1;
// ループ 3 回目
  System.out.print("x> ");
  double x = (new Double(in.readLine())).doubleValue();
  sum = sum + x; count = count + 1;
System.out.println("sum = " + sum + "; ave = " + (sum/count));
```

こうなっていると「最後に合計を 3 で割っている」ことがよく分かるでしょう? 1 つの考え方として覚えておくと役立つかも知れません。

あと、「`int count = 0;`」をループの中に書いて動かずに呆然としている人がいたが、ループの中に書いたら毎回 0 にクリアされてしまうので無意味。で、変数宣言を「`{...}`」の中に書くと外に出たときにはなくなっているのでコンパイルエラーになる。この場合、コンパイラがちゃんとエラーを教えてくれるから正しく外側に書く助けになるでしょ?

演習 2b-d

演習 2b〜d はまとめて示そう。まず、最大値を求めるには前回解説したように「現在の最大」を覚えておいてそれより大きい値が現われたらそちらを新たな最大として覚え直す。それが何番目かを記録するには、上でやったのと同様に回数を数えていて、最大を覚える時にその回数を覚えればよい。

```
import java.io.*;

public class r3ex2b {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    double max = Double.NEGATIVE_INFINITY;
    int count = 0, maxpos = 1;
    while(true) {
      System.out.print("x> ");
      double x = (new Double(in.readLine())).doubleValue();
      if(x == 0.0) { break; }
      ++count;
      if(x > max) { max = x; maxpos = count; }
    }
    System.out.println("max = " + max + "; pos: " + maxpos);
  }
}
```

なお、上の場合は「最大値が複数現われた場合は最初のものが何番目か」を表示していたが、これを「最後のもの」にするには「`if(x >= max) ...`」と変更すればよい (おわかりかな)。

しかしこの `max` の初期値は? やってみた人は分かると思うが、たとえば `max` の初期値を 0 にすると、マイナスの数ばかりのデータの時に正しい答えが出ない (0 はマイナスの数より大きいから)。そこで、「`Double` 型で表し得るもっとも小さい値」を入れておけば、何が来てもそれ以上ではあるから、正しく答えが出るわけである。なお、この値は本当に「$-\infty$」というわけではなく、「ものすごく絶対値の大きいマイナスの値なのでこれ以上は表せない」ということを示す単なる目印の値である。これに類似した値として「$+\infty$」、「NaN」(非数) がある。たとえば 0 で割ったり負の数の平方根を取ったりして NaN と遭遇した人は既にいると思いますがどうですか。

そんな答えは知らない、という人 (大抵はそうですよね) はどうしたらよかったか? 方法としては次の 2 つがあると思う。

1. ループに入る前に 1 つデータを読み、`max` にはその値を入れる。
2. 「最初のデータを処理していない」という旗 (フラグ) を持っておき、ループの中で最初にデータを読んだ時は比較せずに常に `max` に最初の値を入れる。

どちらでもまったく正解。

2 配列の練習問題

演習 3a-c

3 つの小問とも似たようなパターンなので、1 つにまとめて示す。

```
import java.io.*;

public class r3ex4 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    // 配列にデータを読み込む
    int[] a = new int[100];
    int count = 0;
    while(count+1 < a.length) {
      System.out.print(count + "> ");
      int v = (new Integer(in.readLine())).intValue();
      if(v == 0) { break; }
      a[count] = v;
      ++count;
    }
```

ここまでは例題と全然変わらないでよい。この先で最大と最少を求める。

```
    // 最大、最少を求める。
    int max = a[0];
    int min = a[0];
    for(int i = 1; i < count; ++i) {
      if(a[i] > max) max = a[i];
      if(a[i] < min) min = a[i];
    }
    System.out.println("max: " + max + ", min: "+min);
```

見ての通り、まず 0 番目を `max`、`min` 双方に入れておき、残りの各要素と照らしあわせて `max`、`min` を更新する。配列だともうデータはそこにあるので、さっきのように初期値をいくつにするか悩まないで済む。では次に合計と平均。

```
    // 合計、平均を求める。
    int total = 0;
    for(int i = 0; i < count; ++i) total += a[i];
    double average = (double)total / (double)count;
    System.out.println("total: " + total + ", average: " + average);
```

合計は変数 total に「足し込んで」行けばできる。さて、平均はそれを個数で割るのだけど、整数のままで割り算すると切捨て除算になることに注意! こういう時はキャストで実数に変換してから除算すればよい。さて、最後に平均より大きいものを出力。

```
    // 平均より大きいものを出力。
    for(int i = 0; i < count; ++i) {
      if(a[i] > average) {
        System.out.println("larger than average: " + a[i]);
      }
    }
  }
}
```

ここでも「きちんとするなら」a[i] を double にキャストしてから average と比較するのだけど、そうしてもあんまり見やすくないので、ここでは直接比較した (この場合も実際には int が double に変換されてから比較されている)。

演習 4

これは Java そのものは疑似コードの通り作ればよい。

```
import java.io.*;

public class r3ex4 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    // 配列にデータを入力する
    int[] a = new int[100];
    int count = 0;
    while(count < a.length) {
      System.out.print(count + "> ");
      int v = (new Integer(in.readLine())).intValue();
      if(v == 0) { break; }
      a[count] = v;
      ++count;
    }
    // 昇順に並べ替える
    for(int i = 0; i < count; ++i) {
      for(int k = 1; k < count; ++k) {
        if(a[k-1] > a[k]) { int x = a[k-1]; a[k-1] = a[k]; a[k] = x; }
      }
    }
    // 結果を出力する
    for(int i = 0; i < count; ++i) { System.out.println(a[i]); }
  }
}
```

それで、なぜ並べ替えられるのか? まず、ループの中の if 文のところを見てみよう。これは図 1 のように、配列を先頭からスキャン (走査) していきながら、ある箱と次の箱の間で「大小順が逆」だったら、「その 2 つの箱の内容を入れ換える」働きをする。

内側のループではそれが終わったら、今度は次の 2 つの箱の間で同じ動作をする…ということは、「大きい値」はつぎつぎに配列の後ろ側に向かってずれて行くことになる。ただし途中で自分より大きい値にぶつかったら、そこでは交換はされないので、その大きい値がその後右の方にずれて行くことになる。

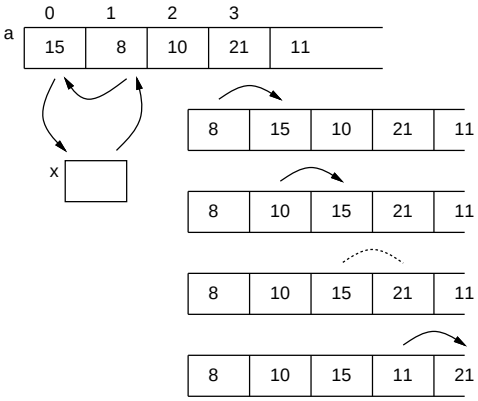

図 1: バブルソートの原理 (1)

ではこれを外側のループで何回も繰り返すことの意味は何だろう? それは 2 のように、1 回目の繰り返しで「最大の値」は必ず右端に来る。ということは、2 回目の繰り返しで「2 番手の値」がその左隣に落ち着くことは確実である。というようにして、count 回やれば最後の値まであるべき位置に落ち着く。

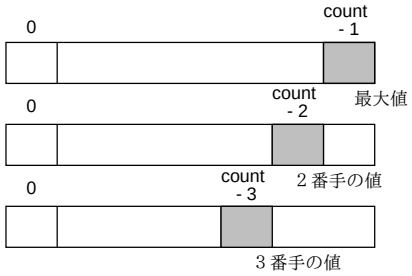

図 2: バブルソートの原理 (2)

実は内側のループは毎回「k < count - i」のところまでやれば十分である。というのは、そこから先は既に大きい奴が落ち着いているので、交換が起きることはないからである。

さらに考えると、以上のは「最も運が悪くても」大丈夫なようにしたわけであり、実際にはこんなに回数を回さなくても並び終わっているはずですね。そこで、「旗」を用意しておき、1 回のスキャンの前に旗を立て、交換があったら旗を下ろすようにする。スキャンし終わって旗が立ったままなら「交換はなかった」つまり全部並んでいたことになる。

```
boolean done = false;
while(!done) {
  done = true;
  for(int i = 1; i < count; ++i) {
    if(a[i-1] > a[i]) {
      int x = a[i-1]; a[i-1] = a[i]; a[i] = x; done = false;
    }
  }
}
```

この方が速いかというと、残念ながらだいたいは実行時間は前のとほとんど変らない。つまりそう運よく並び終わるわけではない (ただし、最初からほとんど並んでいるデータの場合は大変有効)。

根本的な問題として、バブルソートでは「隣どうし」のみ交換するので、「大きい」要素が右に移動するのに 1 つずつしか移動できず、交換回数が多くなる。そこで、「隣」の変りに「d 要素向こう」のと比較交換するようにする。間隔 d は最初は目一杯大きくしておき、スキャンするごとに縮めていき、最後は 1 にする。終わったかどうかはこれまで通りフラグで見る。

```
boolean done = false;
int d = count-1;
while(d > 1 || !done) { // 間隔が 1 より大か交換ありなら続行
  done = true;
  for(int i = d; i < count; ++i) {
    if(a[i-d] > a[i]) {
      int x = a[i-d]; a[i-d] = a[i]; a[i] = x; done = false;
    }
  }
  d = (d * 10 + 3) / 13; // 間隔をおよそ 13 分の 10 ずつ縮小
}
```

このアルゴリズムは「コームソート」といい、要素数が多くなった場合にはバブルソートよりもはるかに高速なことが知られている。

演習 5

演習 5 が演習 4 と違うのは、1 つ数値が入力されるつど、既に入っている数値を後ろから見ていって、入力された数値より大きいものを順に後ろに 1 つずつずらし、空いた場所に入力された数値を格納すること (図 3)。この方法は数値を 1 つずつあるべき位置に「挿入」して行くことから「挿入ソート」と呼ばれる。

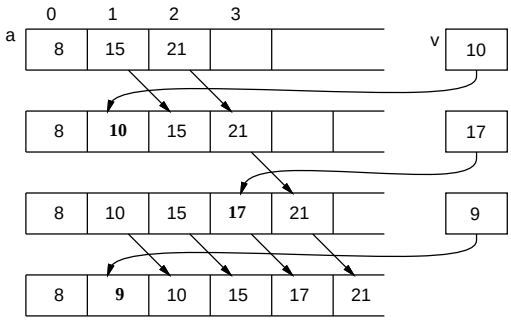

図 3: 挿入ソートの原理

疑似コードは資料にあったので、Java のコードだけ示す。こっちの方がわかりやすかったですか?

```
import java.io.*;

public class r3ex5 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    int[] a = new int[100];
    int count = 0;
    while(count < a.length) {
      System.out.print(count + "> ");
      int v = (new Integer(in.readLine())).intValue();
```

```
      if(v == 0) { break; }
      int i = count;
      while(i > 0 && a[i-1] > v) { a[i] = a[i-1]; i = i - 1; }
      a[i] = v;
      ++count;
    }
    for(int i = 0; i < count; ++i) { System.out.println(a[i]); }
  }
}
```

演習 7 と 8 については紙面がいっぱいなので略。きっとレポートとして回答してくれる人がいるでしょうから…

3 オブジェクトを使う

3.1 オブジェクトの使い方/調べ方

前回までで何回もクラスやオブジェクトとはなんたら、という説明をしたが、具体的でないのでちっとも実にならなかったことと思う。今回はもう 1 度だけクラスとオブジェクトの概念を説明し、それから実際にどんなクラス (つまりオブジェクトの種類) があるかを見てみることにする。

まず、「オブジェクト」とは「さまざまな機能を提供できる『もの』」だということは何回も話した。そして、オブジェクトの種類が「クラス」であることも話した。あるクラスに属する個々のオブジェクトをそのクラスの「インスタンス」と呼ぶ。インスタンスを作るには「new クラス名 (...)」という形の式を使う。

オブジェクトやクラスのさまざまな機能を利用するには「メソッド」を呼ぶ。メソッドは次の 2 種類がある。

クラス名. メソッド名 (...)　←クラスメソッド
式. メソッド名 (...)　←インスタンスメソッド

「式」は何らかのインスタンスを計算するものでなければいけない。クラスメソッドとインスタンスメソッドの違いは、前者がクラスに所属していてインスタンスと関係を持たないのに対し、後者はインスタンスに所属している (たとえば「私」オブジェクトの「名前を返す」メソッドの結果は「あなた」オブジェクトの「名前を返す」メソッドの結果とは違う) ことである。

このクラスの WWW ページから「JDK 1.4 日本語ドキュメント」の横にある「API ドキュメント」のリンクをたどると、標準のクラス群としてどのようなものがあるかを記載したドキュメント (API ドキュメント) が見られる。ドキュメントは「パッケージ」と呼ばれる単位ごとに分かれている。たとえば java.lang.Math(というのは java.lang パッケージに入っている Math というクラスを表す) のドキュメントを見ると、その中に sqrt() というメソッドがある。これが平方根でお世話になった Math.sqrt() なわけである。ちなみに、これは前にも説明したと思うが、クラスメソッドである。クラスメソッド、クラス変数は一覧表のところに「static なんとか」と書いてあるのでそれと分かる (static の意味についてはまた後で説明する)。

同様に、readLine() でキーボードから入力したり"…"という形で書いた文字列は java.lang.String クラスのオブジェクトなので、このクラスのドキュメントを見るとどのようなメソッドがあるかが分かる (「メソッドの概要」と書かれた箇所以降を見て行くこと)。今日は長くなるからやらないが、実は「定数」もクラスに付属しているので、先にやった「Double.NEGATIVE_INIFINITY」なども API ドキュメントから調べることができる。

演習 1 API ドキュメント中の java.lang.Math の「メソッドの概要」の箇所を眺め、三角関数を計算するメソッドを探してみよ。また、java.lang.Double の定数の箇所を眺め、「-∞」の他にどのような定数があるか見てみよ。

演習 2 API ドキュメント中の java.lang.String の「メソッドの概要」の箇所を眺め、文字列に対して次のような操作を行なうメソッドの使い方 (名前とパラメタ) を手元にメモせよ (演習で役に立つ)。

a. i 文字目の文字を取り出す。
b. 文字列と別の文字列 (オブジェクト) が等しいかどうか調べる。
c. 文字列の長さ (文字がいくつ入っているか) を調べる。

d. 文字列の中で指定した文字/文字列が最初に現われる位置を得る。

e. 文字列中のある文字を別の文字に取り換えた文字列を得る。

f. 文字列中の i 文字目から後の部分の文字列を得る。

g. 文字列中の i 文字目から n 文字ぶんの文字列を得る。

h. 文字列中の小 (大) 文字を大 (小) 文字に取り換えた文字列を得る。

なお、「文字」と「文字列」は違うことに注意。「文字」は値の一種であり、型は「`char`」、定数は「`'a'`」などのように「`'`」で囲んで、文字 1 つだけを書く。「文字列」はオブジェクトであり、型/クラスは「`String`」、定数は「`"Abc"`」などのように「`"`」で囲んで、文字は 0 個以上何個でも書ける。

注意: 説明に「正規表現云々…」と書かれたメソッドは JDK 1.4 以降で加わったものなので、少し古い Java 実行系だと使えない。もちろんここでは自由に使っていいが、家で使えなかったりするかも知れないので注意。一般にいつからあるメソッドかは各メソッドの説明本文の末尾の「導入されたバージョン」のところをチェックすれば分かる。何も書いてないのはすごく古くからあるメソッド。

3.2 API ドキュメントの見かたの解説

「いきなり API ドキュメントを見ろと言われても見かたがわからん」という人が多いだろうから、少し解説を書いておく。API ドキュメントはとりあえず、クラス (=オブジェクトの種類) ごとに、そのオブジェクトがどのようなメソッドを持っているかを調べるのに使うことが多いはずである。メソッドにはクラスメソッドとインスタンスメソッドがあり、その有無は API ドキュメントでは「`static`」という指定がついているか否かで区別できる。

```
クラス名.メソッド(...) ← クラスメソッド(static と記されている)
式.メソッド(...)       ← インスタンスメソッド(static がない)
```

インスタンスメソッドを使うことが多いと思うので、その場合についてもう少し説明する。インスタンスメソッドを呼ぶ場合 (2 行目の書き方)、「式」はクラスのインスタンスを表していて、そのクラスが指定したメソッドを持たないといけない。たとえば「`"abcd"`」という式は String のインスタンスを表すので、

```
"abcd".replace('a', '*')
```

は許されるが、「1」という式は int 型であり、これはクラス型ではないので (つまりインスタンスではなく値)、

```
1.replace('a', '*')
```

は許されない。また、たとえインスタンスであっても

```
System.out.replace('a', '*')
```

というのは、許されない。なぜなら、`System.out` は `PrintStream` クラスのインスタンスであって、`PrintStream` クラスは `replace()` というメソッドを用意していないから。たとえば犬に「おすわり!」と言えば分かるけど猫に「おすわり!」といっても分からないのと同様…おわかりかな?

さて、この `replace()` の API ドキュメントを見ると次のように書いてある。

```
String replace(char oldChar, char newChar)
```

これの意味は次の通り。

- まずこのメソッドは `String` クラスのところに書かれていて、`static` と書かれていないので、`String` オブジェクトに対して呼び出すインスタンスメソッド。
- `String` — このメソッドは `String`(文字列) を返す。
- `replace(…)` — もちろん、メソッドの名前。
- `char oldChar` — 最初の引数はかりに `oldChar` と呼ぶことにするが、これは文字型である。
- `char newChar` — 2 番目の引数はかりに `newChar` と呼ぶことにするが、これは文字型である。

つまり oldChar とか newChar という名前は説明の都合上つけてあるわけで、重要ではない (ちょうど自分のプログラムで好きな変数名をつけて構わないのと同じこと)。また、char というのは「文字ですよ」という情報を表しているだけであり、呼び出す時に本当に「char」と書いてはいけない (そうじゃなく、文字の値を渡す)。以上を (説明の文章とつき合わせて) 総合すると、replace() というメソッドは

```
"abcd".replace('a', '*') --- 文字「a」を「*」に置き換える
```

のように使うと判るわけである。まあ最初は慣れなくてしんどいと思うけど、API ドキュメントは調べられないといろいろ困るので気長に練習してください。

3.3 String オブジェクト

さて、お話ばかりではつまらないので、とりあえず上で調べた String クラスを題材にちょっと遊んでみよう。まずは「左右ひっくり返し」の例題を見てみよう。疑似コードを示す。

- 無限に繰り返し:
- 　文字列 str を入力。
- 　str が空文字列なら、ループを抜け出す。
- 　res ← 空文字列。
- 　i を 0 から長さ-1 まで変えながら繰り返し:
- 　　res の先頭に str の i 番目の文字を連結。
- 　ここまで繰り返し。
- 　res を出力。
- ここまで繰り返し。

先にも述べたように、計算機では数は 0 から数えることが多いので、文字列も長さ L の文字列であれば、その中には 0 番目、1 番目、…、$L-1$ 番目の文字が含まれると考える。Java のコードは次の通り。

```
import java.io.*;

public class R4Sample1 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    while(true) {
      System.out.print("String> ");
      String str = in.readLine();
      if(str.equals("")) { break; }
      String res = "";
      for(int i = 0; i < str.length(); ++i) {
        res = str.charAt(i) + res;
      }
      System.out.println(res);
    }
  }
}
```

では説明を。

- 1 回入力したらおしまいではつまらないので、このプログラムは空行 ([RET] のみ) を入力するまで繰り返し処理を行う。
- 「while(true) ... 」で、条件が「真」という定数なので無限に繰り返すことになる。
- 文字列はオブジェクトなので「==」では比較できない。メソッド equals() を使うこと。ここでは空文字列と比較している。空文字列だったら前回学んだ break; を使ってループから抜ける。

- 文字列の長さはメソッド `length()` で調べられる。

では動かしてみよう。

```
% javac R4Sample1.java
% java R4Sample1
String> hello
olleh
String> aho
oha
String>    ← [RET] だけを打った
%
```

なかなか面白いでしょう?

演習 3 上の例題をそのまま打ち込んで動かせ。

演習 4 次の Java プログラムを作れ。`String` クラスのメソッドを活用すること。

a. 入力文字列をすべて大文字にして表示する。(ヒント: `toUpperCase()` を使う)

b. 入力文字列に現われる「`a`」をすべて「`*`」にする。(ヒント: `replace()` を使う)

c. 入力文字列に現われる「`a`、`e`、`i`、`o`、`u`」をすべて「`*`」にする。(ヒント: `replace()` を 5 回使う)

d〜f. 次のような三角形や巡回を表示する。(ヒント: `substring()` を使う)

```
d: String> abcd  e: String> abcd  f: String> abcd
   abcd             abcd             bcda
   abc               bcd             cdab
   ab                 cd             dabc
   a                   d             abcd
```

演習 5 文字列を扱う次のようなプログラムを作れ。

a. 文字列 s を読み込み、その大文字を小文字に、小文字を大文字にそれぞれ入れ替えた文字列をうち出す。

b. 文字列 s を読み込み、その文字列が「回文」(前から読んでも後ろから読んでも同じ) かどうか判定する。回文の定義はまかせる。

c. 文字列 s を読み込み、その文字列の中に部分文字列として回文が含まれているかどうか判定する。さすがに 1 文字や 2 文字はつまらないので、回文の長さは 3 文字以上であるものとする。

d. 文字列 s と文字列 t を読み込み、s の中に t が (連続した) 部分文字列として含まれているかどうか判定する。

e. 文字列 s と文字列 t を読み込み、s の中に t が (とびとびにでも) (その順番で) 含まれているかどうか判定する。

f. 文字列 s と文字列 t を読み込み、s と t がアナグラムの関係にある (s の文字の並び順を交換したものが t になっている) かどうか判定する。

4 コマンド引数

これまで、プログラムにデータを与える時は `BufferedReader` オブジェクトから 1 行ずつ読み込む方法だけを使って来た。しかし、「ちょっとだけ」データを与える時にはプログラムを起動する `java` コマンドの行で一緒に指定する、という方法がある。具体的には、これまでは

```
java クラス名
```

でプログラムを起動しますよ、ということだけ説明してあったが、

```
java クラス名 文字列 文字列 … 文字列
```

のようにデータを書くことでそれがそのまま渡せる。なお、この「文字列」の部分はデータなので「"..."」や「'...'」で囲んでもよいが、中に空白文字などが入っていなければ何も囲まないままでもよい。

さて、この渡したデータを利用するには? それは、プログラムの冒頭に

```
public static void main(String[] args) ...
```

というところがあったが、実はこの「args」というのが (そのすぐ前に書かれているように)「文字列の配列」であり、そこに上のようにして渡したデータが入った状態でプログラムが実行開始する。簡単な例題を示そう。

```
import java.io.*;

public class R4Sample2 {
  public static void main(String[] args) throws Exception {
    for(int i = args.length-1; i >= 0; --i) {
      System.out.println(args[i]);
    }
  }
}
```

このプログラムはもうほとんど説明の余地はないというか、実行例だけ見ていただこう。

```
% java R4Sample2 This is a pen.
pen.
a
is
This
```

演習 6 上の例題をそのまま打ち込んで動かせ。動いたら次のように改造してみよ。

a. 第 2 文型 (SVC) の英文をコマンド引数として渡したらそれを疑問文に変形する (上の例なら「Is this a pen?」にする)。

b. コマンド引数として文字列を 3 つ受け取り、1 番目の文字列中に表れる 2 番目の部分文字列を 3 番目の文字列に置き換える。「Saraba」「ab」「AB」を渡したら「SarABa」が出力されるという感じ。

c. コマンド引数として文字列と整数を受け取り、文字列を整数の回数ぶん繰り返したものを出力する。「ba」「4」を渡したら「babababa」が出力されるという感じ。(ヒント: 受け取った文字列を整数にするにはいつもの「(new Integer(...)).intValue()」。

5 Java の文法

プログラムの正確な書き方は、その言語の「文法」によって定められている。クラスについてはまだ説明していないので、ここではメソッド以下の部分の文法を簡単に示す。なお「[...]」は「あってもなくてもよい」、「|」は「または」、…は「ならんだもの」、,…は「カンマで区切って並んだもの」を表す。

```
メソッド::= 修飾… 型指定 メソッド名 (宣言,…)[throws クラス名 ] { 文… }
修飾 ::= public | static | final
型指定 ::= 型名 | 型名 [] | void
型名 ::= クラス名 | boolean | byte | char
       | int | long | float | double
宣言 ::= 型指定 変数名
文 ::= 宣言 [= 式 ]; | 式;
       | if(式) 文 [else 文 ] | while(式) 文
       | for([ 宣言 =] 式; 式; 式) 文
```

```
        | { 文…} | break; | continue;
式 ::= 式 演算子 式 | 演算子 式 | 式 [式]
        | new クラス名 (式,…) | new 型名 [式]
        | クラス名. メソッド名 (式,…) | 式. メソッド名 (式,…)
        | リテラル | 変数名 | クラス名. 変数名 | 式. 変数名 | ( 式 )
リテラル::= 整定数 [l] | 実定数 [f] | "..." | '...'
```

整数リテラルは「l」がついたのは long、そうでないのは int。実数リテラルは「f」がついたのは float、そうでないのは double。演算子については先に挙げた (「a = b;」の「=」は代入**演算子** であるのに注意。

6 日本語の扱いとファイルの読み書き

6.1 日本語で何が問題になるか

ここまでで、皆様の中には「メッセージに日本語を混ぜても全然問題なく使えている」とか「どうしても日本語が入れられない」とか色々な人がいたと思う。その辺の事情を含めてここで説明しておこう。なお、「情報処理」マターは最低限しか説明しないのでそのつもりで。

まず、日本語の文字コードをファイルに書くとき、大きく分けて次の 4 つが広く使われている。

- iso-2022-jp — いわゆる「JIS コード」であり、メールなどではこれを使うことが標準となっている。
- euc-jp — EUC は「Extended Unix Code」つまり UNIX で漢字を扱うため考案されたもので、UNIX 系システムでは多く使われている。
- SJIS — Microsoft が考案したもので、Windows と Mac で使われている (両者で多少違いがある)。
- UTF-8 — 全世界の文字コードを共通にするという野望の元に作られた文字コード体系 UNICODE をファイルに保存するためのコード。

皆様が使っている Emacs ではどのコードでも利用できる。Emacs を使っている時は、「黒い行」のところに「J」「E」「S」のどれが表示されているかでどのコードのファイルを扱っているかが分かる。また、TextEdit では保存時にダイアログの「エンコーディング」のところで選択できる。

さて、Java の方はどうだろう。Java は「日本向け」ではなく「世界中」で使えるようにするため、内部では UNICODE を採用している。このコード系だと日本語、中国語、韓国語、ヨーロッパ系各言語、その他の文字がすべて使える (無理に詰め込んだために問題もあるのだが)。このため、javac でコンパイルするときに「-encoding 符号化」というオプションを指定することができる。

```
javac -encoding UTF-8 Test.java
javac -encoding JIS Test.java
javac -encoding EUCJIS Test.java
javac -encoding SJIS Test.java
javac -encoding JISAutoDetect Test.java
```

ファイル名はあくまで例。なお、最後の「JISAutoDetect」というのは漢字コード自動判別を使って JIS/EUC/SJIS のどれか判断してくれるという便利なもの。しかしそんなことしなくてもちゃんと漢字が使えていたが…という人もいるはず。その場合はエディタがプログラムをシステムの標準の文字コードで保存してくれ、javac がシステムの標準の文字コードとして読み込んでくれているから)ECC の環境で標準の環境設定のままなら標準のコード系は UTF-8 のはず)。

また家の PC に Java 処理系を入れた場合は SJIS に対して同様に処理してくれるようになっているはず。いずれにせよ、この方法でうまく行かない場合には-encoding を指定すれば大丈夫になるので覚えておくこと。[1]

[1] たまに Emacs で打ち込む時に文字コードが JIS になっていてうまくコンパイルできない人がいる。その場合は「Control-X RET f euc-jp RET」というキー操作をしてから保存すると EUC に変更できる。画面下端の黒い帯の左端が「E」になっていることを確認すること。「J」になっていたら JIS が設定されている。

6.2 プログラム内でのコード変換

上記はコンパイルするプログラムの漢字コードの話だったが、プログラムで「外から読み込む」データやプログラムから「外に書き出す」データはどうしたらいいだろうか。先に、「バイト単位で読む System.in のオブジェクトをもとに、文字単位で読む InputStreamReader オブジェクトを作っている」という話をしたと思うが、ここで漢字コードを「システムの設定により」変換してくれる。

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

しかし、システムの設定でうまく行かない場合もあるので、そのときは InputStreamReader を作るときに次のように自動判別を指定する (自動判別でなく特定のコード系を指定してもいい)。

```
BufferedReader in = new BufferedReader(new InputStreamReader(
                                        System.in, "JISAutoDetect"));
```

また、キーボードから読むのではなく、ファイルからデータを読む場合には System.in の代わりにファイル名を指定して FileInputStream オブジェクトを生成して使うが、この場合も同様、つまり次のようになる。

```
BufferedReader in = new BufferedReader(new InputStreamReader(
                      new FileInputStream("ファイル名"), "JISAutoDetect"));
```

では出力は? これも System.out.println() 等を使えばシステム設定で自動的に変換してくれる。いちいち指定するのは面倒なのでこちらは当面この機能を使うことにするが、説明だけしておくと、漢字コードを指定して出力したい場合は次のようにする。

```
PrintWriter pr = new PrintWriter(new OutputStreamWriter(
                                    System.out, "EUCJIS"), true);
```

ファイルに出力する場合は次の通り。

```
PrintWriter pr = new PrintWriter(new OutputStreamWriter(
                    new FileOutputStream("ファイル名"), "EUCJIS"), true);
```

なお、出力の場合は「自動判別」はあり得ないので出力するコードを特定のものに指定する必要がある。これらの方法で変数 pr に PrintWriter オブジェクトが取得できたら、あとは「pr.println("...");」とか「pr.print("...");」とかを使えばよい。

6.3 例題: ファイルから特定の文字列を探す

ではファイルを読む例題として、ファイル名と文字列を指定して起動すると指定した文字列が現れる行を打ち出す、というものを示す。なお、予めお断りしておくが、漢字 (日本語文字) を探そうとするときはコマンド行引数でうまく漢字が渡らないとダメ (駒場の環境だとうまく行かないかも)。

```
import java.io.*;

public class R4Sample3 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(
                          new FileInputStream(args[0]), "JISAutoDetect"));
    String pat = args[1];
    while(true) {
      String str = in.readLine();
      if(str == null) { break; }
      if(str.indexOf(pat) >= 0) { System.out.println(str); }
    }
  }
}
```

これを動かす場合は、コマンド引数でファイル名と探す文字列を渡す。

```
java R4Sample3  適当なテスト用ファイル  文字列
```

また、環境によっては `System.out.println()` だと UNICODE からシステムの漢字コードへの変換がうまく行われないため文字化けになることもある。そのような場合は先に説明したように `OutputStreamWriter`、`PrintWriter` を経由するように直せばよい。

演習 7 上のプログラムをそのまま読んで動かせ。動いたら次のように改造してみよ。

a. コマンド引数で指定した文字列の直前で改行するようにする。

b. すべての行を左右ひっくり返して出力する。

c. ファイルを上下さかさまに (最初の行が最後になるように) 出力する。ただしファイルの行数は 1000 を越えないとしてよい。

   ヒント: ファイルを全部配列に読み、下から順に出力する。

d. ファイルを行単位で「でたらめ順に」出力する。ただしファイルの行数は 1000 を越えないとしてよい。

   ヒント: `Math.random()` を呼ぶと 0 以上 1 未満の一様乱数が得られる。つまりこれを `n` 倍して切捨てて整数にすると (Java コードで書いてしまうと「`(int)(n*Math.random())`」)、`0`〜`n-1` の一様乱数になる。ファイルを全部配列に読み、これでランダムに選んだ行を出力し、最後の行をその場所に移して埋める。これで行数が 1 減るので、これを繰り返して行く。

e. ファイルを 2 つ指定し、2 つ目のファイルは「探したい文字列を 1 行ずつに分けて」入れておく。1 つ目のファイルの中で、2 つ目のファイルで指定された文字列の「どれかが」現れる行だけを打ち出す。なお、この方法であれば日本語が探せる。

演習 8 前回やった「順に並べ替える」アルゴリズムを用いて、ファイルの内容を順番に並べ替えるプログラムを作れ。10 秒間で何行くらいのファイルなら並べ終わるか計測して報告せよ (ファイルは WWW などから適当なものを選んで保存すればよい)。

ヒント: 2 つの文字列の大小比較は `String` クラスのメソッド `compareTo()` を利用すればよい。たとえば `str1` と `str2` が `String` 型変数だとして、「`str1.compareTo(str2)`」は 2 つの文字列の大/等しい/小に応じて正/0/負の整数を返す。

演習 9 その他、ファイルの読み (書き) 機能を活用する、自分の腕にあった (不当に易しくない) プログラムを考案し、作成せよ。

A　本日の課題 4A

「演習 4」または「演習 5」で動かしたプログラムを含む小レポートを今日中に久野までメールで送ってください。

1. Subject: は「Report 4A」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. 「演習 4」または「演習 5」で動かしたプログラムどれか 1 つのソース。
4. 以下のアンケートの回答。

   Q1. 文字列を扱うプログラムはこれまでやったプログラムと比べてどうでしたか。

   Q2. Java 言語の「オブジェクト」とか「メソッド」についてなんとなくでも分かりましたか。

   Q3. 本日の全体的な感想と今後の要望をお書きください。

B　次回までの課題 4B

次回までの課題は「演習 4」〜「演習 9」の (小) 課題からプログラムを 2 つ以上作り、報告すること。ただし「演習 7」から 1 つ以上選ぶこと。レポートは授業開始時刻の **10 分前** までに、上記と同様に久野までメールで送付してください。

1. Subject: は「Report 4B」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. 選んだプログラム 1 つのソース。
4. その簡単な説明。
5. もう 1 つのプログラムのソース。
6. その簡単な説明。あれば考察等も。
7. 下記のアンケートの回答。

Q1. ファイルの読み (書き) や日本語の扱いについて納得しましたか。

Q2. 今回もまた、疑似コードを書くのと、Java に直すのと、打ち込んで動かすのとで掛かった手間の比率を教えてください。

Q3. 課題に対する感想と今後の要望をお書きください。