計算機プログラミングI 2005　久野クラス　＃5

久野　靖*

2005.11.11

はじめに

ファイルの扱いは面倒でしたか? まあ、このクラスでは以後ファイルの入出力はほとんど使わないと思いますので、使う必要ができた時に自発的に勉強してみてください。練習問題解説でも例は用意しました。

前回までで文字入力ベースのプログラムはだいぶ書けるようになったと思うので、今回からいよいよグラフィクスの入ったものに進むことにしましょう。グラフィクスものの題材の場合、アプレットにしておくと他の人にも簡単に見てもらえるので、とりあえずアプレットを中心に進めます。動かし方がこれまでとだいぶ変わるので、その練習が必要ですから、早めに実習に入りたいと思います。

1　String クラスの練習問題

演習 4a. 入力文字列をすべて大文字に

今回は Java コードをいきなり見れば十分ですね。

```
import java.io.*;

public class r4ex4a {
  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 = str.toUpperCase(); // ☆以下ではこの
      System.out.println(res);        // ☆2行だけ変更していく
    }
  }
}
```

要するに、入力した文字列 str に対してメソッド toUpperCase() を呼ぶには「str.toUpperCase()」で、その結果 str の小文字をすべて大文字に変換した文字列が返されるので、それを res に入れて打出す。なお

```
System.out.println(str.toUpperCase());
```

のように 1 行にしてしまって変数 res は使わない、というのでもよい。ときに

```
str.toUpperCase();
```

*筑波大学大学院経営システム科学専攻

とだけ書いてうまく行かないで悩んでいる人がいたようだが、あくまでも `toUpperCase()` は「小文字を大文字に変えた新しい文字列を返す」のであって、その結果を変数に入れない限り元の `str` は元のまんまであることに注意。

授業で説明した例題にあった for ループは文字列の内容を 1 文字ずつ処理するために必要だったのだが、この例のように「使える」メソッドがあるならそれを使ってしまえばループで処理しなくてすみ、楽ちんである。

演習 4b. 入力文字列に現われる「a」をすべて「*」に

以下では上のリスティングで☆のついた行の差し換えだけ示します。

```
String res = str.replace('a', '*');
System.out.println(res);
```

これは上とほとんど同様だが、`toUpperCase()` ではメソッドに引数がなかったのに対し、replace() では「どの文字を」「どの文字に」という 2 つの引数が必要になる。なお、Java では「`"..."`」は String(文字列) を表し、「`'?'`」は 1 文字を表すという区別があるのに注意。なお、先の例題と同様いきなり

```
System.out.println(str.replace('a', '*'));
```

としても構わない (美観や趣味の問題)。

演習 4c. 「a、e、i、o、u」をすべて「*」に

これはヒントにあるように、replace() を 5 回使えばできる。

```
String res = str.replace('a', '*');
res = res.replace('e', '*');
res = res.replace('i', '*');
res = res.replace('o', '*');
res = res.replace('u', '*');
System.out.println(res);
```

まず「a」は先の通り処理したから、次に「res に対して」今度は「e」の処理を同様に行い、その結果を…新しい変数に入れてもいいのだが、同じ変数でもいいので `res` に入れ直してしまう。これを続ければいいわけだ。

ところで、毎回 `res` に入れ直さなくても次のようにしてもよい。

```
String res = str.replace('a', '*').replace('e', '*').
  replace('i', '*').replace('o', '*').replace('u', '*');
```

なぜこれでいいかというと、`replace()` が返すのは String オブジェクトだから、それに対しまた次の `replace()` を呼んでも構わないから。この方がかっこいいかどうか?

しかし、5 文字くらいだったらこれでも (また先の方法でも) いいけど、20 文字くらいになったらもう書くのがイヤになるでしょう? そこで次のようにしてもよい。

```
String res = str;
String sub = "aeiou";
for(int i = 0; i < sub.length(); ++i) {
  res = res.replace(sub.charAt(i), '*');
}
```

つまり、置き換える文字をまとめて文字列として用意し、その各文字を引数として `replace()` を適用する、というのを for 文で繰り返すわけである。まあ今回はここまでしなくてもよかったけど。なお、正規表現関係のメソッドを使えばもっと簡単にできます (興味あったら調べてみてください)。

演習 4d. 三角形その 1

これは substring() を使って文字列の先頭文字から N 文字目までを取り出す、というのを N を変えながら繰り返せばよい。

```
for(int i = 0; i < str.length(); ++i) {
  System.out.println(str.substring(0, str.length()-i));
}
```

つまり、str.length() が4であれば、for 文の中で「str.substring(0, 4)」、「str.substring(0, 3)」、「str.substring(0, 2)」、「str.substring(0, 1)」を順に出力するわけである。

演習 4e. 三角形その 2

これは後ろの方から取り出せばいいのだが、その前に適当な数の空白を用意しないと右側がそろってくれない。その空白もループ 1 周につき 1 文字ずつ増えるのだから、もう 1 つ変数を用意して、それに空白を 1 文字ずつ連結して伸ばして行く。

```
String space = "";
for(int i = 0; i < str.length(); ++i) {
  System.out.println(space + str.substring(i));
  space = space + " ";
}
```

なお、substring() でパラメタ (引数) の数が 1 つのものは、その文字から末尾までを取り出すことになっている。しかし、せっかくさまざまなメソッドがあるのに「1 文字ずつ空白をくっつける」という細かい操作をやるのは悔しいですね。たとえば、space という変数にうんと長い空白文字列が入れてあったとすると、N 文字の空白は「space.substring(0, N)」と書けば済むでしょう? その方向で「別解」を考えてみよう。

ここで問題は、どんな入力を持って来てもそれより長いような空白文字列を予め用意することができない、ということ。しかし、使う前に「十分長く」してしまうことは簡単ですね。というわけで、まずメソッドの先頭部分で

```
String space = "                         "; // とりあえずの長さ
```

とした上で、☆の個所を次のようにすればよい。

```
while(space.length() < str.length()) {  // 長さが足りなければ
  space = space + space;                 // 長くする!
}
for(int i = 0; i < str.length(); ++i) {
  System.out.println(space.substring(0, i) + str.substring(i));
}
```

演習 4f. 巡回表示

これはじつは簡単で、単に「2 文字目から最後まで」と「先頭」とをくっつければ 1 文字ずれるので、それを何回も単に繰り返す。

```
String res = str;
for(int i = 0; i < str.length(); ++i) {
  res = res.substring(1) + res.charAt(0);
  System.out.println(res);
}
```

演習 5a. 大文字小文字の入れ換え

「一気に」全部大文字や小文字にするのなら toUpperCase() や toLowerCase() を使えばいいが、入れ換えるとなると面倒である。ヒントとしては、toUpperCase() で「変化した」文字があればそれは元が小文字だったのでその変化した値を取ればいい、それ以外の文字は大文字にせよ英字ではない文字にせよ toLowerCase() の結果を取ればいいはず。

```
String res = "", up = str.toUpperCase(), low = str.toLowerCase();
for(int i = 0; i < str.length(); ++i)
  if(str.charAt(i) != up.charAt(i))  res += up.charAt(i);
  else                               res += low.charAt(i);
System.out.println(res);
```

charAt() で取り出したものは文字 (値) だから「!=」で比較してよいことに注意。なおこの例では for や if の中にそれぞれ文が 1 つずつしか入らないので「{...}」で囲まないで短く書いてみた。変数の宣言も「変数名 = 式」というのをいくつも並べていいのでそうしてみた。

演習 5b. 回文のテスト

これは前回例題の「左右ひっくり返し」をもとにすれば簡単、つまりひっくり返した後で元の文字列と同じかどうか見る。

```
String res = "";
for(int i = 0; i < str.length(); ++i) res = str.charAt(i) + res;
System.out.println(res.equals(str));
```

これも for 文のところは短く書いてみた。あと res と str が等しいかどうか equals() で調べた結果は論理値 (true か false か) だから、それをそのまま書き出している。

演習 5c. 回文が埋まっているかのテスト

こちらは「与えられた文字列の長さ 3 以上のすべての部分文字列について」それが回文かどうか前問と同じ方法で調べてみればいい。

```
String mesg = "does not includes a palindrome.";
for(int len = 3; len <= str.length(); ++len) {
  for(int pos = 0; pos <= str.length()-len; ++pos) {
    String sub = str.substring(pos, pos+len), res = "";
    for(int i = 0; i < sub.length(); ++i) res = sub.charAt(i) + res;
    if(res.equals(sub)) mesg = "inludes palindrome.";
  }
}
System.out.println(mesg);
```

ループが 3 重くらいになるとごちゃごちゃになりますかね (笑)。

演習 5d. 部分文字列が含まれているかのテスト

ここからは文字列を 2 つ読み込む必要があるがその部分は略すことにして変数 s と t に入っているものとして計算部分を書く。実は前問ができていれば簡単で、s の部分文字列で長さが文字列 t と同じものをすべて取り出して等しいかどうか調べるだけ。

```
String mesg = "NO.";
for(int pos = 0; pos <= s.length()-t.length(); ++pos) {
  if(s.substring(pos, pos+t.length()).equals(t)) mesg = "YES.";
}
System.out.println(mesg);
```

ところで t の方が長い文字列だったらどうなるでしょう? そのときは、条件を満たすような pos が存在しないので for 文の本体は 1 回も実行されず、そのまま NO と出力される。正しいでしょう?

演習 5e. 部分文字列が分散して含まれているかのテスト

こちらは連続していないので equals() は使えず、順番に見ていくことにする。考え方としては、s のどこを見ているかを指し示す変数 sp と t のどこを見ているかを指し示す変数 tp を用意し、双方の見ている位置が一致したら双方を先に進め、一致しなかったら sp だけ先に進める。そして tp の最後まで来たらぶじ見つかったことになる。

```
int sp = 0, tp = 0;
while(sp < s.length() && tp < t.length())
  if(s.charAt(sp) == t.charAt(tp)) {
    ++sp; ++tp;
  } else {
    ++sp;
  }
System.out.println(tp == t.length());
```

「いつ」ループを終るかだが、s と t ともにまだ比較すべき文字列が残っていれば続ける、そうでなければ終るということ。このループが必ず終るかというと、if のどちらの枝へ来ても sp は増えていくからいつかは条件が成り立たなくなるので大丈夫。

演習 5f. アナグラムの判定

これは先のとはまったく違って、s から t の各文字を 1 文字ずつ取り除いていって最後に s が空っぽになれば OK。

```
for(int i = 0; i < t.length(); ++i) {
  int pos = s.indexOf(t.charAt(i));
  if(pos < 0) break;
  s = s.substring(0, pos) + s.substring(pos+1);
}
System.out.println(s.equals(""));
```

ただし、本当は s と t の長さが等しいことを最初にチェックしておくべきだがそれはさぼっている。

演習 6a. 疑問文への変形

この問題はいろいろに解釈できるが、題意としてはあんまり凝ったことは要求していないつもりで、SVC の文なので (1)be 動詞を先頭に移す、(2) 先頭を大文字、2 語目は小文字に、(3) 文末の「.」を「?」に、くらいの変形を想定していた。これをやるコードは次の通り。

```
import java.io.*;

public class r4ex6a {
  public static void main(String[] args) throws Exception {
    String subj = args[0].toLowerCase(); // 主語は小文字
```

```
    String verb = args[1].substring(0, 1).toUpperCase() +
                  args[1].substring(1);  // 動詞は先頭大文字
    args[args.length-1] = args[args.length-1].replace('.', '?');
                          // 最後の語の「.」を「?」に取り換え
    System.out.print(verb + " " + subj); // 主語と動詞出力
    for(int i = 2; i < args.length; ++i) {
      System.out.print(" " + args[i]);   // 残りの語を出力
    }
    System.out.println("");              // 最後に行かえを出力
  }
}
```

文を 1 行に出力するのが結構面倒でしょう?

演習 6b. 文字列の置き換え

indexOf() と substring() を使えば簡単。

```
import java.io.*;

public class r4ex6b {
  public static void main(String[] args) throws Exception {
    int pos = args[0].indexOf(args[1]);
    if(pos >= 0) { // あった→「前の部分+置き換え部分+残り」
      System.out.println(args[0].substring(0, pos) + args[2] +
                         args[0].substring(pos+args[1].length()));
    } else {       // なかった→元のまま出力
      System.out.println(args[0]);
    }
  }
}
```

ところで、ほかに replaceFirst()、replaceAll() というメソッドも見つけた人がいると思う (JDK 1.4 以降〜比較的新しい機能)。このうち前者を使うと次のように簡単になる。

```
import java.io.*;

public class r4ex6b1 {
  public static void main(String[] args) throws Exception {
    System.out.println(args[0].replaceFirst(args[1], args[2]));
  }
}
```

ただし、こちらは第 1 引数を「正規表現」(パターン) として扱うので、パターンを表すものを置き換えると (たとえば「.」を置き換えると) 思った通りにならない。これに対処するには特殊文字の前に「\」を前置するなどの対処が必要。

演習 5c. 文字列を指定回数だけ連結

これは 2 番目のコマンド引数を数値に変換できれば簡単。

```
import java.io.*;
```

```
public class r4ex6c {
  public static void main(String[] args) throws Exception {
    int count = (new Integer(args[1])).intValue();
    String res = "";
    for(int i = 0; i < count; ++i) res += args[0];
    System.out.println(res);
  }
}
```

2　ファイル関係の問題

詳しく説明していると実習の時間がないと思うので、ごく簡単な説明と回答例だけ示します。まずパターンのところで2行に分けるには、indexOf() であてはまる位置を探して、そのような位置があれば substring() で2つに分けて出力。

```
import java.io.*;

public class r4ex7a {
  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(); // 1行ずつ読む
      if(str == null) { break; }
      int pos = str.indexOf(pat);
      if(pos >= 0) {             // パターンがあてはまったら
        System.out.println(str.substring(0, pos)); // 2行に
        System.out.println(str.substring(pos));    // 分解
      } else {                   // そうでなければ
        System.out.println(str); // 元のまま出力
      }
    }
  }
}
```

左右ひっくり返しは、1行読んでは、前回の例題同様にして左右をひっくり返した文字列を作り、それを出力。

```
import java.io.*;

public class r4ex7b {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(args[0]), "JISAutoDetect"));
    while(true) {
      String str = in.readLine(); // 1行読む
      if(str == null) { break; }
      String res = "";            // 前回やったと同様にひっくり返す
      for(int i = 0; i < str.length(); ++i) res = str.charAt(i) + res;
      System.out.println(res);    // 出力
    }
```

```
  }
}
```

上下ひっくり返しは、配列の例題でやったようにデータをまず配列に読み込んでしまい、最後に逆順に出力。

```
import java.io.*;

public class r4ex7c {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(args[0]), "JISAutoDetect"));
    String[] lines = new String[1000]; // 配列を用意
    int count = 0;
    while(count < lines.length) {
      lines[count] = in.readLine();    // 1 行読む
      if(lines[count] == null) break;  //「おしまい」なら終わる
      ++count;                         // 行数を 1 ふやす
    }
    for(int i = count-1; i >= 0; --i) { // 最後から順に
      System.out.println(lines[i]);     // 出力
    }
  }
}
```

でたらめ順で出力する場合、読み込むところまでは上と同じ。その後、1 行ランダムに選んで出力し、最後にある行をそこに移して埋める。

```
import java.io.*;

public class r4ex7d {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(args[0]), "JISAutoDetect"));
    String[] lines = new String[1000];
    int count = 0;
    while(count < lines.length) {
      lines[count] = in.readLine();
      if(lines[count] == null) break;
      ++count;
    }                   // ここまで先の例題と同じ
    while(count > 0) {  // 行が残っている間
      int j = (int)(count * Math.random()); // ランダムに選ぶ
      System.out.println(lines[j]);         // 出力
      lines[j] = lines[count-1]; --count;   // そこを埋める
    }
  }
}
```

ファイルに書かれた文字列 (複数) を探す問題は、ファイルを 2 つ読むのが面倒なのと、チェックするときも文字列の数だけ繰り返し調べるということ。

```
import java.io.*;
```

```
public class r4ex7e {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(args[0]), "JISAutoDetect"));
    BufferedReader wd = new BufferedReader(new InputStreamReader(
                        new FileInputStream(args[1]), "JISAutoDetect"));
    String[] pat = new String[1000]; // 探す文字列を入れる配列
    int count = 0;
    while(count < pat.length) {
      pat[count] = wd.readLine();
      if(pat[count] == null) { break; }
      ++count;                      // 全部読む→これまでと同様
    }
    while(true) {
      String line = in.readLine(); // 探されるファイルを 1 行読む
      if(line == null) { break; }
      boolean found = false;       // 文字列があったか示す旗
      for(int i = 0; i < count; ++i)
        if(line.indexOf(pat[i]) >= 0) { found = true; break; }
      if(found) System.out.println(line); // あったら行出力
    }
  }
}
```

演習 7 は問題の説明が分かりにくかったかも知れない。文字列の大小関係は compareTo() で調べられるので、それを利用してあとは前にやったのと同じアルゴリズムで並べられますよ、ということのつもり。

```
import java.io.*;

public class r4ex8 {
  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(args[0]), "JISAutoDetect"));
    String[] lines = new String[1000];
    int count = 0;
    while(count < lines.length) {
      lines[count] = in.readLine();
      if(lines[count] == null) { break; }
      ++count;
    }                     // 読むところまで同じ
    boolean done = false; // バブルソート。これもほぼ同じ
    while(!done) {
      done = true;
      for(int i = 1; i < count; ++i) {
        if(lines[i-1].compareTo(lines[i]) > 0) { // ここだけ違う
          String x = lines[i-1]; lines[i-1] = lines[i]; lines[i] = x;
          done = false;
        }
      }
```

```
    }
    for(int i = 0; i < count; ++i) System.out.println(lines[i]);
  }
}
```

3 アプレット

さて、ここまでに作ってきた Java プログラム — 「java」コマンドで動かす奴 — は「スタンドアロンアプリケーション」と呼ばれる種類のものである。これとは別に、見たことはある人が多いと思うが、Web ブラウザの画面内で動くような Java プログラムもあり、これを「アプレット」と呼んでいる。

アプレットも (スタンドアロン) アプリケーションも Java プログラムであることに変りはないが、おもに外側の部分の書き方がちがっている。具体的にはアプレットでは次のような形になる。

```
import java.awt.*;
import javax.swing.*;

public class 名前 extends JApplet {
  型 変数 = 初期値;
  型 変数 = 初期値;
  ...

  public void paint(Graphics g) {
    文…
  }
  その他必要なメソッド定義…
}
```

もう少し細かく説明しよう。

- まず、これまでと使うライブラリが違うので、import 文で awt パッケージと swing パッケージを必ず指定する (他に必要なものがあれば import 文はいくつでも追加してよい)。
- 次に、アプレット用に作るクラスはクラス名の後に「`extends JApplet`」という指定が必要 (その具体的な意味は長くなるのでもっと後で説明する)。
- アプレットに限らないが、クラスではメソッドを複数定義して使うことができ、それらのメソッドが共通に使う変数をまず書く。
- アプレットではさまざまなことができるが、最低限「何かを画面に描く」ことはしないと何も表示されないので意味がない。そのため、画面に描くためのメソッドを必ず用意する。これが `paint()` である。
- `paint()` は、ブラウザがアプレットに画面を描かせようと思った時何回でも繰り返し呼び出される。だから、この中であまりたくさん仕事をすると画面表示が遅くなってうまくない。呼び出されるとき、`paint()` には引数として `Graphics` クラスのインスタンスが渡される。このオブジェクトは「画面に描くためのペン」であると思って欲しい。このオブジェクトのさまざまなメソッドを呼び出すことで、画面にさまざまな描画を行うことができる。
- アプレットでは `paint()` 以外にもいくつかメソッドを定義することがあるが、話が長くなるのでこれも後回しにする。

では、最初のアプレットの例題を見てみよう。

```
import java.awt.*;
import javax.swing.*;

public class R5Sample1 extends JApplet {
```

```
    Font fn = new Font("Helvetica", Font.BOLD, 24);
    public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D)g;
      g2.setFont(fn);
      g2.setPaint(new Color(100, 0, 255));
      g2.drawString("Hello, World", 30, 30);
    }
  }
```

やってることは次の通り。

- まず、表示に使うフォント (字形) を大きなものにするため、Font オブジェクトを作成して変数 fn に入れる。なお、「"Helvetica"」というのは英文フォントなので、日本語を表示させたければ「"Serif"」とかがよいかも。これはアプレットの実行開始時に最初に行う。あとは全部 paint() が呼ばれたときの動作。
- paint() は先に説明したように、ブラウザが「画面を表示したいなあ」と思った時に、あくまでも「ブラウザが」呼び出してくる。いつ、何回呼び出されるかはブラウザまかせ。これまでの main() のように「ここから実行する」というわけではないので注意。だから、ここでは絵を書くのに必要な動作だけを行うこと。
- paint() にはパラメタとして画面に描くためのペンが渡されて来る。歴史的事情により、そのペンの型は Graphics オブジェクトになっているが、実際にはより高機能な Graphics2D オブジェクトが渡されるので、Graphics2D 型の変数 g2 にキャスト (強制型変換) して格納し、以後こちらを使う。
- ペンをさっき用意したフォントを使うように設定。
- ペンの「絵の具」として描画に使う色を設定。
- そして最後に、画面上の位置を指定して文字列を表示する。なお、画面上の位置や大きさは「ピクセル」(計算機の画面は細かい点の集まりであり、その 1 つの点をピクセルと呼ぶ) 単位で指定する。

さて、アプレットを表示するためには、アプレットを埋め込んだ表示用の HTML ファイルが必要である。アプレットしか入っていない簡単なテスト用の HTML ファイルを示しておこう。

```
<html>
<head><title>sample</title></head>
<body>
<h1>sample</h1>
<applet code="R5Sample1.class" width=300 height=200></applet>
</body>
</html>
```

HTML ファイルの名前は好きに決めてよいが、どうせ課題 5A を提出するときに report5a.html というファイルを作らなければならないので、それに合わせて report5a.html というファイル名にしておく。また、HTML ファイルでアプレット領域の大きさを指定しているが (ここでは 300x200 ピクセルとしている)、大きさは適宜変更してよい (むやみに大きくするのはキモイですよと警告しておく)。ともかく HTML ファイルをブラウザで開くと図 1 のように文字が表示される。

なお、Windows の Internet Explorer で内蔵の Java 処理系を使うと Java のバージョンが古くて上記アプレットがうまく表示できないことがある。その場合は Sun から J2SDK 1.4.2 をダウンロードしてインストールし、なおかつそのとき「IE でも Sun の処理系を使う」ように設定してください。

さて、以下では課題用の HTML ファイルを置く場所を各自のホームディレクトリ直下の「cp1」という名前のディレクトリに統一させて頂き、なおかつそこに置いたファイルを誰にでも見えるように保護設定して頂くことにする。このディレクトリに置いたファイルは適宜コピーさせて頂いてクラスのページで公開する予定である。久野がレポートを採点でき、なおかつ互いに他の人の作品を見て切磋琢磨できるためにはこのようにする必要があるので、ご了承頂きたい。以下そのための指示。

演習 1 以下の手順に従って、上の例題アプレットを含んだページを作ってブラウザで表示してみよ。

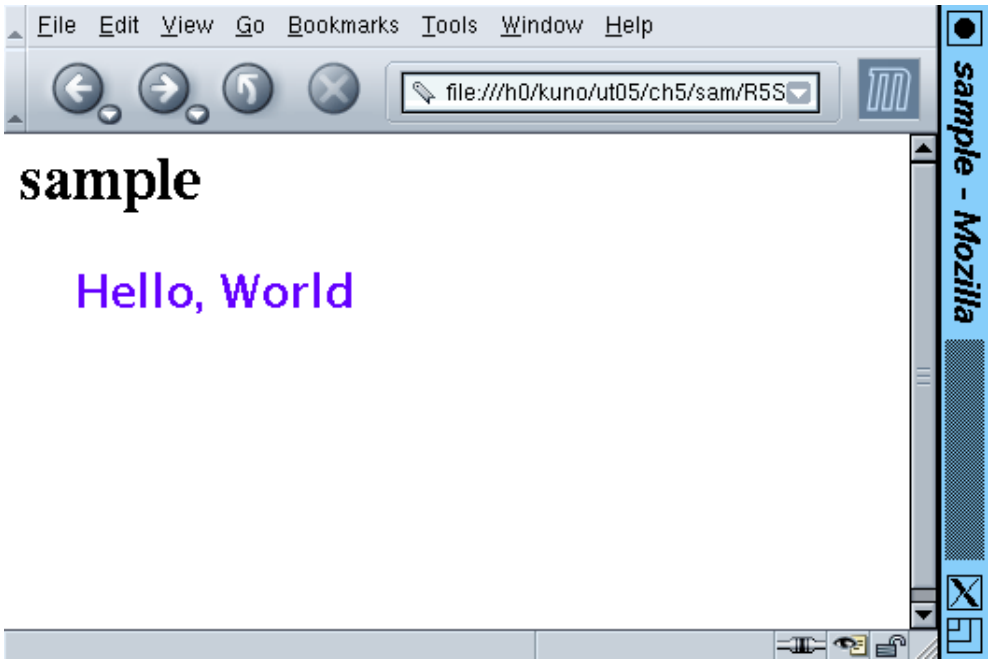

図 1: 簡単なアプレット

1. まず、ホームディレクトリの保護設定を「誰でもたどれる」に変更し、ホームディレクトリ直下に「cp1」というディレクトリを作り、その内容を「誰でも見える」にしてからそこへ行く。

   ```
   cd
   chmod a+x .   ←「.」(ピリオド) の保護設定を変更
   mkdir cp1     ←新しいディレクトリを作る
   chmod a+rx cp1  ←保護設定を変更
   cd cp1        ←そこへ移動
   ```

2. エディタでこのディレクトリの下にファイル `R5Sample1.java` を作成する。(別のディレクトリに作っても無意味だから注意!)
3. 「`javac R5Sample1.java`」でコンパイルする。
4. エディタでこのディレクトリの下にファイル `report5a.html` を作成する。(別のディレクトリに作っても無意味だから注意!)
5. ファイルを作成したりコンパイルしたあとは必ず、「`chmod  a+r  *`」を実行して、全部のファイルを他人に見えるように設定する」**重要!これやらないと他人に見えない!**
6. 完成したら Mozilla 等のブラウザの「ファイルを開く」でこのページを開くか、ブラウザの代わりにアプレットをテストするコマンド appletviewer を使って、「`appletviewer report5a.html`」のように HTML ファイルを指定して起動。**自分で OK になったら、必ず隣の人のブラウザで「レポート提出チェック」経由で「5a」の「f」のリンクを開いて確認してもらうこと! 保護設定の間違いは「自分には読めるが他人には読めない」という状態なので、自分ではチェックできない。**

注意! Java コードを手直しした場合には多くのブラウザではブラウザの「再読み込み」ボタンでは新しいプログラムを読み直してくれない。その場合、「Java コンソール」を開いてそこで「キャッシュをクリア」を実行してから再読み込みすること。

演習 2 次のことを行え。

a. `java.awt.Graphics` のドキュメントを見て、`drawString()` の使い方をチェックせよ。次に上の例題を修正して、文字の表示位置を変更してみよ。

b. `java.awt.Color` のドキュメントを見て、色の作り方をチェックせよ (ヒント: `new Color(...)` の使い方は「コンストラクタ」の項を見る)。次に上の例題を修正して、文字の色を変更してみよ。

c. 例題では文字列を 1 回しか描いていなかったが、同じ文字列を「場所を変えながら、さらに色も変えながら複数回描く」ようにしてみよ。なるべくならループを使うこと。

d. java.awt.Graphics のドキュメントを見ると、円、矩形、線分、多角形などさまざまなものを表示するメソッドがあることが分かる。どれか 1 つ好きなものを選び、上の例題に追加してその図形も描くようにしてみよ。文字とは色を変えること。

4 Graphics2D の機能を使う

Graphics クラスのメソッドを調べてみて、あんまり大した機能がないなと思った人はいないだろうか? 実際そうなので、このあたりの機能は 1995 年ころに Java が普及し始めた時とほとんど同じである。しかしこのままではもっと「高度な」絵を作るのに不便なので、Java2 と呼ばれるバージョンからはこれを拡張した Graphics2D クラスが作られた。

実は既にさっきの例題ではこちらの機能も使えるように準備はしてあったのだが、とりあえず簡単な Graphics の機能から先に説明した。以後は Graphics2D の API ドキュメントも併せて見ていただきたい。ただし機能が込み入っていて、他の関連クラスも一緒に見ないと分からないので大変だと思う。ここではとりあえず「おすすめの」機能を使った例題を見て頂こう。import が 1 つ増えているので注意。

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class R5Sample2 extends JApplet {
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    GeneralPath p1 = new GeneralPath();
    for(int i = -15; i <= 15; ++i) {
      float x = i * 0.1f;
      float y = x*x*x - x;
      if(i == -15) p1.moveTo(x*100f + 150f, -y*100f + 100f);
      else         p1.lineTo(x*100f + 150f, -y*100f + 100f);
    }
    g2.setPaint(new Color(100, 0, 255));
    g2.setStroke(new BasicStroke(3f));
    g2.draw(p1);
  }
}
```

説明は次の通り。

- GeneralPath というのは、「見えない線が引けるペン」のようなもの。これまでのペンと違い、このペンは「ペンをあげてどこまで移動」(moveTo())、「ペンを降ろしてどこまで線/曲線を引く」(lineTo()、curveTo())、「最初の点まで線を引いて戻る」(closePath()) という機能を用意している。
- 上の例では、関数 $y = x^3 - x$ のグラフを x が-1.5〜1.5 の範囲で 0.1 きざみで描いている。ただし、アプレット領域の大きさ 300x200、原点が右上隅、yg が大きくなる方向が下向きなので、中央が (150,100) になり、x の範囲が-1.5〜1.5、y が大きくなる方向が上向きになるように適宜変換している。内側の if 文は、一番最初だけは moveTo()、あとは lineTo() を使うようにするためのもの。
- 上記の方法で見えない線を引いたあと、Graphics2D オブジェクトに対してできることとして「この線にそってなぞれ — draw()」、「この線の内側を塗りつぶせ — fill()」がある。ここでは draw() を使っている。
- なぞる時でも塗る時でも、その時の色や模様を setPaint() で設定しておける。設定できるものとしては、これまで使って来たような単色 (Color オブジェクト) でもいいし、だんだん色調が変わって行くような模様 (GradientPaint オブジェクト) でもいい。

- なぞる線のようすは setStroke() で設定しておける。とりあえず一番簡単な「一定の太さの線」は BasicStroke オブジェクトで指定できる。ここでは「太さ 3 の線」としている。

上の例がどういう絵になっているかは、画面で見てもらうのがいいと思う。

一応これでグラフは描けるのだが、座標変換の計算を自分でする代わりに Graphic2D 側でやってもらうようにもでき、その方が分かりやすいかも知れない。この機能は次のようなものである。

- なぞったり塗ったりする前に translate()、rotate()、scale() というメソッドを呼んでおくことで、描かれる図形の位置を並行移動したり、任意角度ぶん回転したり、任意倍率で拡大/縮小することができる。

この機能を使ったバージョンは次の通り。

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class R5Sample3 extends JApplet {
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    GeneralPath p1 = new GeneralPath();
    for(int i = -15; i <= 15; ++i) {
      float x = i * 0.1f;
      float y = x*x*x - x;
      if(i == -15) p1.moveTo(x, y); else p1.lineTo(x, y);
    }
    g2.setPaint(new Color(100, 0, 255));
    g2.setStroke(new BasicStroke(0.03f));
    g2.translate(150f, 100f);
    g2.scale(100f, -100f);
    g2.draw(p1);
  }
}
```

すなわち、今度はグラフは素直に「そのまま」パスにして、パスを描く前に原点移動と拡大 (なおかつ Y 軸方向の反転) を施している。なお、拡大すると線の幅も太くなるので、線の幅はさっきの 1/100 にしている。

演習 3 上の例題の好きな方をそのまま打ち込んで動かせ。動いたら次のように改良してみよ。

a. 2 次曲線、サイン曲線など別のグラフを描かせてみよ。
b. グラフを傾けてみよ。
c. X 軸と Y 軸も表示されるようにしてみよ。
d. 目盛りがないと淋しいので、目盛りをつけてみよ。どんな風なものをつけるかは各自にまかせるが、おおよその座標値が読み取れるようなものであること。
e. 複数の関数のグラフが同時に表示されるようにせよ。
f. 関数のグラフではなく、うずまき型とか楕円 (斜めに傾いているとなおよい) とかリサージュ曲線とか任意の 2 次元曲線を描く。

5 メソッド init() による初期設定

上で「paint() は描くことだけ行うように」と述べたが、描く絵の内容によっては描くための計算が結構必要だったりするかも知れない。そのような計算を paint() の中で行うと、画面の描き直しのたびに同じ計算をするので無駄だし、ブラウザから見て「描く作業に時間が掛かりすぎる」状態なのでよくない。また、描画の中で乱数を使っている場合、描

き直すと (乱数を再度生成してそれに基づき描くため) 形が変わってしまうという問題もある。このような問題のある例を見ていただこう。

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class R5Sample4 extends JApplet {
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    GeneralPath p1 = new GeneralPath();
    float x = 150f, y = 100f, len = 30f; p1.moveTo(150f, 100f);
    for(int i = 0; i < 100; ++i) {
      float theta = (float)(2*Math.PI*Math.random());
      x += len*Math.cos(theta); y += len*Math.sin(theta);
      p1.lineTo(x, y);
    }
    g2.setPaint(new Color(100, 0, 255));
    g2.setStroke(new BasicStroke(3f));
    g2.draw(p1);
  }
}
```

このプログラムは長さ 30 の線分が 100 個連なった折れ線を描くが、線分のつながった部分での折れ角は乱数によって決定している。そして、その乱数計算が `paint()` の中にあるため、ブラウザ画面を (一部でも) 隠して再度見えるようにしただけで線の形が変わってしまう。

このような問題を避けるために、アプレットでは初期設定用の `init()` というメソッドも用意することができる。ブラウザはアプレットを起動するとき最初に 1 回だけ、`init()` を読んでくれる。だから、ここで計算などの初期設定を済ませておき、`paint()` はそれに基づいて画面を表示する、というのが正しい分担なわけである。

でも、`init()` で計算した結果をどうやって `paint()` に渡すのか? それは、両方のメソッドの「外側に」ある変数に入れておけばいいわけである。上の例を手直しして、折れ線の計算を `init()` で行うように直したものを次に示す。

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class R5Sample5 extends JApplet {
  GeneralPath p1 = new GeneralPath();
  public void init() {
    float x = 150f, y = 100f, len = 30f; p1.moveTo(150f, 100f);
    for(int i = 0; i < 100; ++i) {
      float theta = (float)(2*Math.PI*Math.random());
      x += len*Math.cos(theta); y += len*Math.sin(theta);
      p1.lineTo(x, y);
    }
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.setPaint(new Color(100, 0, 255));
    g2.setStroke(new BasicStroke(3f));
    g2.draw(p1);
```

```
    }
  }
```

このように `init()` で折れ線の計算は終わっているので、`paint()` の側ではただそれを画面に出すだけでよい。

演習 4 `init()` で適当な変数にデータを初期設定しておくという考えに基づいて、乱数を使って次のような絵を描いてみよ。(**注意!** 画面を閉じてまた開いた時などに絵が別のものに変わってしまわないこと。)

a. 画面上にさまざまな色の矩形 (他の図形でもよい) が 20 個ほどちらばっている。

b. ランダムにうねうねとくねった曲線 (ヒント: `curveTo()` を使うとよい。使い方がやや難しいがそこはチャレンジ)。

c. でたらめに単語を並べた文章。

なお、たとえば 0〜255 の整数の一様乱数を作りたければ「`(int)(256*Math.random())`」という式を使えばよい。

演習 5 自分のオリジナルな美しい/かっこいい絵を描くアプレットを作れ。ただし自分の腕前から見て不当に優しくないこと。

A　本日の課題 5A

「演習 2」で作成したアプレット (どれか 1 つ) を格納した WWW ページのための HTML ファイルを、自分の `cp1` ディレクトリの下に `report5a.html` という名前で作成すること。ファイル名はすべて小文字を使うように。(演習が正しくできていればもうできているよね?)。久野クラスのページ経由で**自分以外の人も**見える状態であることを必ず確認すること。

注意! chmod コマンドを実行させるのを忘れていると他人から読めません。読めない場合は採点が「▲」印になります。そのときは chmod コマンドを実行して保護設定を直し、直した旨連絡してください。

また、そのプログラムのコードはいつも通り、本日中に久野までメールで送付してください。具体的な内容は次の通り。

1. Subject: は「Report 5A」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. 下記のアンケートの回答。

   Q1. お絵描きをするアプレットはこれまでのプログラムと比べてどうでしたか。

   Q2. API ドキュメントでオブジェクトのメソッドを調べるのに慣れましたか。

   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 5B

演習 5 のアプレットを作成してください。美しさ/かっこよさの判断は各自にまかせます。演習 4 までのものをやった結果、それが美しい/かっこいいと思うならそれを提出することもできます。

アプレットそのものはそれを表示するページの HTML ファイルを自分の `cp1` ディレクトリの下に `report5b.html` という名前で作成すること。久野クラスのページ経由で**自分以外の人も**見える状態であることを必ず確認すること。

上記とは別に、プログラムのコードはいつも通り、久野までメールで送付してください。具体的な内容は次の通り。

1. Subject: は「Report 5B」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. 下記のアンケートの回答。

   Q1. 計算機で絵を描くことは、手で絵を描くことと比べてどのように違うと考えますか。

   Q2. 上記の違いを活かして「美しい」絵を描くにはどのような工夫が必要だと考えますか。

   Q3. 感想と今後の要望をお書きください。

授業が始まってもレポートを打っている人がいますが、期限を過ぎた後はいつ出しても点数は同じですからゆっくり提出してください。

警告! まだ「動く絵」(アニメーション) を作ろうとしないこと。**paint()** の中で山のように描画や時間待ちを繰り返してアニメーションふうに見せるようにはできるかも知れませんが、システムによくない影響がありますし環境によっては動きません。アニメーションは後日ちゃんとやりますので、今回は止まった絵で勝負してください。この警告を無視した場合は点数を割り引く可能性があります。