システムソフトウェア特論'17 # 1 プログラミング言語とは

久野　靖 *

2017.7.17

0　本科目の目的/内容/進め方

本科目「システムソフトウェア特論 (Advanced Topics on System Software)」の目的は、プログラミング言語処理系およびそれに関連のあるシステムソフトウェアについて、実例を中心に学ぶことです。その進め方としては、Java 言語を用いて実際にさまざまな実装を構築し、具体的な概念を身に付けて頂くようにします。

毎回、授業に際して例題を配布しますので、実際にそれを動かし、また改良してみて頂きます。実習環境としては西 10 の CED を使用しますが、Java 言語が動けばどのような環境でも実習できますので、自分の PC なども含めて各自でうまく進めてください。

本科目では何らかのプログラミング言語によるプログラミングができることを前提とします。実際に使う言語は Java 言語になりますが、Java 言語固有の部分については授業内で説明するようにします。Java 言語の参考書が必要な場合は「久野禎子, 久野　靖, Java によるプログラミング入門 第 2 版, 共立, 2011」をおすすめします。久野まで連絡いただければ著者割引価格で提供できます。

1　プログラミング言語とは

プログラミング言語 (programming languages) とは何かについて、さまざまな定義が可能だと思いますが、ここでは次のように定めます。

- プログラムを記述するために設計された人工言語

ここでさらにプログラムとは何か、人工言語とは何かが問題になるわけですが、人工言語の方はあとの回で厳密に取り上げることになりますのでそれまで保留しましょう。

プログラムは「コンピュータが実行するべき動作を記述したもの」くらいでいいでしょうか。コンピュータがどんなものかとかコンピュータが動作するとかはだいたい皆様知っているということで。「記述した」というのはつまり「外部化した」「文字などで表現された」という意味になります。外部化され表現されていないものはコンピュータに与えたり他人が読んで検討したりできませんから、これは必要最低限な定義といってよいと思います。

あと「設計された」ですが、設計だけで動かないでいいのかという突っ込みもあるかと思います。これは「設計されただけで実装されていない」プログラミング言語は沢山ありますから、無問題です。

2　様々なプログラミングパラダイム

2.1　プログラミングパラダイムとは

プログラミングパラダイム programming paradigm というのは、少し訳しにくいのですが (なのでカタカナのまま使われることが多い)、プログラミング言語の「枠組み」というか「方式」というふうに考えていただければいいと思います。

*電気通信大学 情報理工学研究科

皆様の中で複数のプログラミング言語を学んだことがあり、ある言語で学んだことは他の言語でも使えるので 2 つ目からは学ぶのが容易である、というふうに思っている方もいると思います。これはつまり、言語の見た目は違っていても、パラダイムは共通なので、1 つ目の言語で身につけたことが 2 つう目の言語でもそのまま通用するから、という風に考えればよいでしょう。

しかし実は、パラダイムの異なる言語だとかなりその「共通」が減るので、学ぶのが大変だったりします。ではなんでそんなに大変なのに違うパラダイムの言語があるの、ということになりますが、それは記述したい題材や対象によって、パラダイムとの相性があるから、ということになるでしょうか (もちろん書く人との側の好みや相性も問題になります)。

では具体的に、どのようなプログラミングパラダイムが存在するのでしょう。網羅的はちょっと大変なので、思い付くままにいくつか挙げる程度にします。

- チューリングマシン
- ランダムアクセスマシン (RAM)、機械語
- 手続き型 (命令型) 言語
- 関数型言語
- 論理型言語

チューリングマシンは計算モデルですがプログラムを書けますので入れています。**RAM**(random access machine) というのはこれも計算モデルですが、普通の CPU の機械語がだいたいこれに相当する動作になります。その後の 3 つはこれから説明していきます。

2.2 論理型言語

論理型言語 (logic programming languages) というのは Prolog やその仲間に相当する言語で、述語論理に基づいています。ここでは Prolog を例にどのような感じかだけ見ていただきます。

Prolog ではさまざな事柄を**述語** (predicate) として表します。「`human(socrates)`」というのは「ソクラテスは人間である」を述語として表記しています。この場合、`human` が述語名、`socrates` は引数となります。引数には変数 (大文字で始まる名前) を指定することもできます (用例はすぐ後で)。

Prolog では述語の集まりを**ホーン節** (Horn clause) と呼ばれる次の 3 種類の形に限定してこれを並べることでプログラムを構成します。

q.
q :- p_1, p_2, ..., p_n.
:- p_1, p_2, ..., p_n.

1 番目の形は**事実** (fact) と呼ばれ、述語 q が「真である」ことを述べています。2 番目の形は**頭部** (head) と**本体** (body) から成る**規則** (rule) で、述語 $p_1 \cdots p_n$ がすべて真であるなら、節 p も真である、ということを述べています。3 番目の形は**目標** (goal) と呼ばれ、$p_1 \cdots p_n$ がすべて真であることを示すという動作を実行開始します。以下では、まず事実と規則を複数与え、そのあと目標を打ち込んでその真偽を確認します。

例を見てみます。この例では、事実は「ソクラテスは人間である」、規則は「X が人間なら、X は過ちを侵す」となっています。このように大文字で始まる X は**変数** (variable) であり、さまざまなものが入ります (ただし同じ名前の変数には同じものが入る必要がある)。

```
% gprolog
GNU Prolog 1.4.4 (64 bits)
| ?- [user]. ←直接プログラム入力
```

```
compiling user for byte code...
human(socrates).          ←事実
fallible(X) :- human(X). ←規則
user compiled...
yes
| ?- fallible(X).    ←目標を与える
X = socrates         ← X として socrated をあてはめると
yes                  ←成立することがわかった
```

最後に「X が fallible であるような X が存在するか」を目標として実行すると、X に socrates をあてはめた場合に成立するという答えが得られます。この例では「3 段論法」(「A である」「A ならば B である」から「B である」を導く) を実現しています。

もう少し意味のある例として、リストの連結を扱います。Prolog ではリストは [...] のように角かっこに囲まれた値の列ですが、[X|T] のように書いた場合は先頭の要素が X、残りが T という意味になります。では例を見てみましょう。

```
% gprolog
GNU Prolog 1.4.4 (64 bits)
| ?- [user]. ←直接プログラム入力
compiling...
app([], X, X).  ← 空リストと X を連結すると X
app([A|X], Y, [A|Z]) :- app(X, Y, Z).
 ↑ X と Y を連結して Z ならば、[A|X] と Y を連結すると [A|Z]
user compiled...
yes
| ?- app([1,2], [3, 4, 5], L).  ← 123 と 45 を連結すると何?
L = [1,2,3,4,5]
yes
| ?- app([1, 2], L, [1, 2, 3, 4]). ← 12 と何を連結すると 1234?
L = [3,4]
yes
| ?- app(L, [3, 4], [1, 2, 3, 4]). ←何と 34 を連結すると 1234?
L = [1,2] ? ;    ← 12 だけどまだあるかも (; でさらに探す)
no               ←やっぱりそれでおわりだった

| ?- app(X, Y, [1, 2, 3]). ← X と Y を連結すると 123 になる X と Y は?
X = []
Y = [1,2,3] ? ;  ← 1 つ目
X = [1]
Y = [2,3] ? ;    ← 2 つ目
X = [1,2]
Y = [3] ? ;      ← 3 つ目
X = [1,2,3]
Y = [] ? ;       ← 4 つ目
no               ←以上でおわり
| ?-
```

このように、Prolog では述語のどの位置を変数、どの位置を値にしてもそれを充足するような変数の値を探そうとするので、プログラムの書き方によってはさまざまな使い方ができます。

2.3 手続き型言語

コンピュータの命令は基本的にメモリからデータを取り出し、演算して、結果をメモリに格納します。そこで、この動作を自然な形で記述するため、メモリ上の領域を「変数」「配列」などに対応させ、「代入」操作によって値を書き換えて行くようなプログラミング言語が作られました。

たとえば次のような記述があったとします。

```
x = x + 1
```

これは等式ではなく、変数 x(実際にはどこかのメモリ番地に対応) の値を取り出し、1 と加算する演算を実行し、結果を x に格納 (代入) する、という内容なわけです。このような演算ど代入 (とその実行の流れを制御する機能) により、プログラムの実行が進んでいくわけです。

最初の**高水準言語** (特定の CPU に依存しない形で記述できるようなプログラミング言語) である Fortran はそのようなものでしたし、現在使われている言語の多くもこの流れを汲んでいます。このような言語を、演算や代入などの命令を順次実行していくという意味で**命令型言語** (imperative language)、または手続きを順番に実行していくという意味で**手続き型言語** (procedural language) と呼びます。

本科目で扱う Java も手続き型言語の仲間ですが、さらにオブジェクト指向の機能が追加されています。この点については Java の話題の中で説明していきます。

2.4 関数型言語

関数型言語 (functional language) とは、関数定義とその適用に基づいて実行が進むようなプログラミング言語です。そこでとくに重要なのが、**副作用** (side effect) の排除ということです。副作用とは、処理を実行したときにその処理結果に加えて、コンピュータ内に状態の変化をもたらすことを言います。

最も典型的な副作用は、手続きの中での (手続きの実行が終わった後も残っている) 変数への値の書き込みです。ですから、手続き型言語の実行は副作用の塊となりがちです。

副作用の何が問題なのでしょうか。副作用があると、手続きを 1 回実行したときともう 1 回実行したときで結果が異なる可能性があります。また、複数の手続きを並行して実行するとき、互いに干渉する可能性があります。これらのことがあると、プログラムを高速に実行する上で制約となることがあります。

これに対し、関数型言語、とくに**純粋関数型言語** (pure functional language) では変数の代入をはじめ一切の副作用がないため、並列実行がしやすく、干渉による誤りが置きにくいという利点が得られます。

イメージが湧かないと思いますので、ここで C 言語を使って純粋関数型のようなスタイルでプログラムを書いてみましょう。`main()` は次のようなものとします。

```
#include <stdio.h>
#include <stdlib.h>
int f(int);
void main(int argc, char *argv[]) {
  printf("%d\n", f(atoi(argv[1])));
}
```

ここで関数 `f()` が実際の計算をしますが、その中は「変数への代入を一切行わない」ように書くことにします (上記の `main()` もそうなっていますね)。変数の初期化は行ってよいことにします。

たとえば、階乗の計算をおこなう場合は次のように書けます。

```
int f(int n) {
  if(n < 1) { return 1; }
  else      { return n * f(n-1); }
}
```

演習 1-1 次の計算を C 言語の「関数型」スタイルで書いてみよ。

a. N を与えて $\sum_{i=1}^{N} i$
b. N を与えて $\sum_{i=1}^{N} i^2$
c. N を与えて $fib(N)$ ただし $fib(0) = fib(1) = 1, fib(n) = fib(n-1) + fib(n-2)$ $(n \geq 2)$。
d. その他の「関数型」スタイルで書くのによいと思う任意の計算。

3 Java 言語の特徴と特性

3.1 Java とオブジェクト指向

手続き型言語はもともと、代入、式、制御構造とそれらを集めて呼び出せるようにした手続き (関数、サブルーチン、メソッドなどとも呼ぶ) が基本要素でしたが、プログラムが大きく複雑になるにつれて限界が明らかになりました。具体的には、複数の手続きが使うデータの定義をグローバル変数にすると、多数のグローバル変数が必要となり、それらのどれかを誰かが壊したためにプログラムが破綻するという問題が頻繁に起こるようになったというのが主なものです。

そこで、複数の手続きとそれらが共通に使うデータをひとまとめにして扱う**モジュール**と呼ばれる機能が作られるようになりました。モジュール内のデータはそこに所属する手続きだけからしかアクセスできないので、「誰かが壊す」ことは避けられます。

そのうち、1 つのモジュールが 1 つの「もの」に対応するデータと機能を持つとプログラムが作りやすいことが知られるようになり、モジュールは「そのモジュールのデータ」を複数つくり出せるようになりました。このような「もの」に対応するデータと手続きの組のことを**オブジェクト** (object)、そのようなデータや手続きを定義する構文のことを**クラス** (class)、これらの機能を提供する言語を**オブジェクト指向言語** (object-oriented language) と呼びます。Java もそのような言語の 1 つです。[1]

クラス方式のオブジェクト指向言語では、クラスから個々のオブジェクトを作り出しますが、これらのオブジェクトのことをそのクラスの**インスタンス** (instance) とも呼びます。これは、クラスそれ自身もオブジェクトとしての機能を持つことから、それと区別するためです。

3.2 Java 言語入門

本科目ではプログラムを書くのに Java 言語を使うのでここで簡単に紹介します。詳しい事項については必要になるつど追加します。

最初の例題を示しましょう。これは 2 つの数値を入力し、その和を出力するというものです。

```
import java.util.*;
public class Sam11 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("s1> "); String s1 = sc.nextLine();
    System.out.print("s2> "); String s2 = sc.nextLine();
```

[1] クラスを持たないオブジェクト指向言語もあります。

```
    double d1 = Double.parseDouble(s1);
    double d2 = Double.parseDouble(s2);
    System.out.println("sum: " + (d1+d2));
  }
}
```

説明は次の通り。

1. import はさまざまなライブラリの取り込みを指示する。ここでは Scanner クラスが java.util の中にあるためこの指定をしている。
2. Java ではすべてのプログラムはクラスという単位で記述する。クラスの構文は「class クラス名 { … }」。クラス名は英字で始まる英数字の並びで、大文字から始める習慣。
3. 実行するプログラムの場合、クラス内にメソッド main() が含まれる必要があり、その先頭から実行が開始される。main() は public static メソッドであり、[2] 返値が void、引数が String[] である必要。
4. Java ではクラスからインスタンスを作るときには「new クラス名 (…)」という構文によります。この構文により、インスタンスが作られた後、そのクラスに定義された**コンストラクタ** (constructor) と呼ばれる特別なメソッドが呼び出されてインスタンスの初期化を行い、その後でインスタンスが返されます。かっこ内に指定するのはコンストラクタに渡すパラメタです。そしてクラス名は型でもあり、あるクラスのインスタンスを格納するにはそのクラス型の変数を使います。Scanner は行単位での入力などを可能にするオブジェクトあり、コンストラクタで入力に使うストリームを渡して作ります。System.in は標準入力から読み込むストリームであり、ここではそれを渡して作っています。
5. System.in には InputStream オブジェクト (クラス InputStream のインスタンス)、System.out と System.error には PrintStream オブジェクトが格納されており、それぞれ標準入力、標準出力、標準エラー出力につながっている。PrintStream オブジェクトはメソッド print()、println()、printf() を持ち、前 2 つは文字列を単に出力する (println() はそのあと改行も出力)。printf() は書式文字列と 0〜N 個の値を受け取り、機能としては C 言語の同名のものと同様である。
6. ここではプロンプトを出力し、次に Scanner オブジェクトのメソッド nextLine() により 1 行ぶん入力し、その文字列を変数に格納することを 2 回おこなう (2 行ぶん入力する)。
7. クラス Double のクラスメソッド parseDouble() は文字列を受け取ってその文字列が表す実数値 (double 型の値) を返す。これらをいずれも double 型の変数 d1、d2 に格納。
8. 本体最後の行で d1 + d2 を計算し、それを文字列と連結し (Java では「＋」は片方が文字列のとき他方も文字列に変換した上で文字列連結をおこなう)、出力する。

これを実行するには、まずこのソースを Sam11.java というファイルに格納する。Java ではソースプログラムのファイル名 (の拡張子を除いた部分) とクラス名とが一致する必要があるので、ファイル名はかならずこの名前にする必要がある。そして、次の 2 つのコマンドを用いる。

- 「javac ソースファイル名」 — ソースファイルをコンパイルし、.class ファイルを生成する。
- 「java クラス名」 — 「クラス名.class」というファイルを探してきてその中の main() から実行を開始する。

[2] public はクラス外から参照され得ることを示す。static はクラスメソッド (オブジェクトを作らなくても呼び出せるメソッド — オブジェクトについては後述) を示す。

実行のようすを示す。

```
% javac Sam11.java
% java Sam11
s1> 1.5
s2> 2.7
sum: 4.2
%
```

演習 1-1 上の例題を打ち込んでそのまま動かせ。動いたら、次のことをやってみよ。

a. 加算の代わりに減算、乗算、除算、剰余をやってみよ。演算子は C 言語と同じである。
b. 上の例は実数だったが、整数でやってみよ。整数型は `int`、文字列から整数への変換は `Integer.parseInt()` である。
c. 整数のかわりに長精度整数でやってみよ。長精度整数型は `long`、文字列から長精度整数への変換は `Long.parseLong()` である。整数では計算できないが長精度整数ではできる例を確認すること。

3.3 値とオブジェクト

Java では (C++もそうですが)「値」と「オブジェクト」を区別しています。そのあたりは次のようになっています。

- 値は四則演算、比較演算などが適用できる。値の型としては `byte`、`char`、`short`、`int`、`long`、`float`、`double`、`boolean` がある。それぞれ 8 ビット整数、16 ビットの文字コード値、16 ビット整数、32 ビット整数、64 ビット整数、32 ビット実数、64 ビット実数、論理値である。
- オブジェクトはクラス名 (大文字で始まる) で表される型を持ち、演算によってできる操作は代入のみである。それ以外の操作はすべてメソッド呼び出し「オブジェクト . メソッド名 (引数,...)」によって行う。
- オブジェクトの値はすべて**参照値** (reference value)、つまりオブジェクトが置かれたメモリ上の番地に対応する値である。オブジェクトの値を代入する場合、次に述べる値の変換の場合を除けば、参照値をそのままコピーし代入する。つまりコピー元とコピー先で同じオブジェクトを共有する。
- 8 種類ある値の型にはそれぞれ、その値を保持するようなオブジェクトのクラス `Byte`、`Character`、`Short`、`Integer`、`Long`、`Float`、`Double`、`Boolean` が対応して存在している。これらのクラスのことを (値を囲むということで) **包囲クラス** (wrapper class) と呼ぶ。そして、値と対応する包囲クラスオブジェクトの間では自動的な変換が行える。

```
Double d3 = 3.141;// double 値から Double オブジェクト
double d4 = d3;   // Double オブジェクトから double 値
```

なお、このような「値とオブジェクトの区別」は今日の言語では少くなっています。というのは、C++や Java が設計された当時は「オブジェクトはメモリに保存するので遅い」「だから速度が必要な値は別扱いしよう」という考えで言語が設計されていたわけですが、その後の研究の進展で「言語仕様上はオブジェクトでも実際に動くコートは値と同等」が実現できるようになり、人間が両方を使い分ける面倒は不要だと分かったためです。

3.4 クラスによるオブジェクト定義とその利用

値とオブジェクトについて分かったところで、次はオブジェクトについてもっと考えてみましょう。Java 言語の設計上の特徴として、「演算子はすべて値にのみ適用し、オブジェクトはすべてメソッド呼び出しによって操作する」という点があります。ただし例外は代入の「=」でして、これは左辺のオブジェクトの値を右辺の変数に代入するという形で統一されています。先に述べたように、実際に代入されるのは「参照値」なので、代入はオブジェクトの操作ではないと考えるのがよいかも知れません

ではオブジェクトを操作するメソッドとは何でしょうか。これは結局「手続き」「C 言語でいう関数」に相当します。その呼び出し方は次のようになります。

クラス名.メソッド名 (パラメタ…)　　…(1)
オブジェクトを表す式.メソッド名 (パラメタ…)　…(2)

Java ではすべての要素 (変数、メソッド) はどれかのクラスの中に入れる必要があるので、その外から参照するときは上記いずれかの書き方を取ります。このうち (1) はクラスの中で「オブジェクトを作らずに」使えるメソッド (static メソッド、クラスメソッド) です。main もそうなっていました。こちらは C 言語の関数に近いです。

もう 1 つの (2) は、クラスからオブジェクト (インスタンス) を作り出し、それに対して呼び出します。これをインスタンスメソッドと呼びます。たとえば次の例を見てください (この例は動かしません)。

```
class Dog {
  String name;
  double speed = 0.0;
  public Dog(String n) { name = n; }
  public String toString() { return "name="+name+" speed="+speed; }
  public String getName() { return name; }
  public double getSpeed() { return speed; }
  public void addSpeed(double d) { speed += d; }
}
```

このコードは Dog というクラスを定義します。このクラスは「犬」を複数扱う (?) アプリケーションで利用するもので、このクラスでは犬それぞれの名前と走っている速さを保持します。これらのデータは変数 name と speed に保持されます。

つまり、この 2 つの変数は「犬の 1 個体ごとに」つまり「犬のオブジェクト (インスタンス)」ごとに 1 セットずつあることになります。このような変数をインスタンス変数と呼びます。

さて、このクラスからインスタンスを作り出すときには、new 演算子を使って次のようにします。

```
Dog d1 = new Dog("pochi");
Dog d2 = new Dog("tama");
```

インスタンスを作ろうとすると、そのクラスに定義されたコンストラクタと呼ばれる特別なメソッドが動いてインスタンスを初期化します。コンストラクタはクラス名と同名なのでそれと分かります。Dog のコンストラクタでは名前の文字列を受け取るので、new では文字列を渡します。そしてコンストラクタでは name をその文字列で初期化します。speed の方は変数定義時に初期値を指定したのでそれがそのまま使われます。

残りのメソッドはすべてインスタンスメソッドで、次の例のように使います。

```
d1.toString() --- pochi の名前と速さの文字列を返す
d2.getName()  --- tama の名前を返す
d1.getSpeed() --- pochi の速さを返す
d2.addSpeed(2.0) --- tama の速さを 2.0 だけ増す
```

つまり、これらのメソッドの中で参照している name、speed はいずれもインスタンス変数なので、メソッド呼び出し時に指定したインスタンス (d1 や d2) に付随している変数がアクセスされる、ということです。これにより、多数の犬を扱うなかで犬の名前と速さの対応が分からなくなったりせずに済むわけです。

なお、この (2) のような呼び出し方のことを「オブジェクトに対して〜して欲しいと呼びかける」という類推から「メッセージ送信記法」と呼びます。

3.5 例題: Toknizer

プログラミング言語のソースファイルは「名前」とか「演算子」などの「かたまり」から成っています。このかたまりのことを「トークン (token)」と呼びます。言語処理系がソースコードを読むときも、このトークン単位で読むことが定石です。

今回は簡単のため、先に出て来た Scanner のメソッド next() によって読み出される 1 かたまりずつをトークンということにしましょう。具体的には、空白や改行で区切られた「白くない並び」がトークンになります。そして、次のような使い方ができるオブジェクトを作ります。

```
Toknizer tok = new Toknizer("t1.txt");  --- ファイル名指定して生成
tok.curTok()  ---  「現在位置」のトークンを文字列として返す
tok.isEOF()   ---  「終端」かどうかを論理値で返す
tok.fwd()     ---  「現在位置」を 1 つ進める
```

なお、終端に来たときは現在のトークンは"$"という文字列だということにします。

では、クラス Toknizer のソースを見てみましょう。

```
class Toknizer {
  Scanner sc;
  String tok;
  boolean eof = false;
  public Toknizer(String fname) throws Exception {
    sc = new Scanner(new FileInputStream(fname)); fwd();
  }
  public boolean isEof() { return eof; }
  public String curTok() { return tok; }
  public void fwd() {
    if(!eof && sc.hasNext()) { tok = sc.next(); }
    else { eof = true; tok = "$"; }
 }
}
```

読み取るための Scanner を保持する sc、現在のトークンを保持する tok、そして終わりかどうかを保持する eof がインスタンス変数です。コンストラクタではソースを読み取るファイル名を指定します。[3]

[3] thorws Exception という指定は、ファイル初期化エラーがあったときにそのエラーを呼び出し側に伝えるための指定ですが、詳しくは後日に説明します。

そして、そのファイルから読み込む FileInputStream を生成し、それを渡して Scanner を生成します。次に、最初のトークンを読むことで初期化を完了しますが、それは後で定義しているメソッド fwd() を呼びます。このように、1 つのインスタンスメソッド (コンストラクタも含まれます) の中から同じインスタンスのインスタンスメソッドを呼ぶときはメッセージ送信記法は不要でメソッド名だけで呼べます。[4]

isEof() は eof の値、curTok() は tok の値をそれぞれ返すだけです。ということは、「肝」は最後のメソッド fwd() ですね。これは、まだ eof でなくて、かつ Scanner から次のトークンが読み取れるなら、そのトークンを tok にいれます。それ以外の場合は終わりですから、eof は true を入れ、tok には"$"を入れます。

では、これを呼び出す側を見てみましょう。1 つのファイルにまとめて入れられるので、実際にはこの部分の末尾に上の Toknizer を入れてコンパイルします。

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

public class Sam12 {
  public static void main(String[] args) throws Exception {
    Toknizer tok = new Toknizer(args[0]);
    Scanner sc = new Scanner(System.in);
    while(true) {
      System.out.print("> ");
      String line = sc.nextLine();
      if(line.equals("p")) {
        System.out.println(tok.curTok());
      } else if(line.equals("e")) {
        System.out.println(tok.isEof());
      } else if(line.equals("f")) {
        tok.fwd(); System.out.println(tok.curTok());
      } else if(line.equals("q")) {
        System.exit(0);
      }
    }
  }
}
```

main は前と同様です。[5] そして、コマンド引数で指定したファイルを渡して Toknizer を作り、変数 tok に入れます。またキーボードから読み込む Scanner も前の例題と同様に作ります。

以後無限ループですが、プロンプトを読み込み、1 行文字列を読み込み、それが何であるかで動作を振り分けています。「文字列が等しいかどうか」は (文字列もオブジェクトなので)「==」は使えず、メソッド equals() を使う必要があります。どんなコマンドを入力したら何をするかは見れば分かりますね。[6]

演習 1-2 この演習問題を打ち込んでそのまま動かせ。適当なファイルを指定して動作を確認せよ。できたら、次のような変更を施してみよ。テスト用の main() も対応して適宜直すこと。

[4] ついでにですが、クラスで定義しているクラスメソッドもメソッド名だけで呼べます。

[5] ただし Toknizer を生成するときにエラーが帰ってくる可能性があるので、ここも throws Exception の指定が必要です。

[6] System.exit() は C ライブラリでいう exit() と同じです。

a. 間違って `fwd()` したときに 1 つ前に戻る `back()` を追加する。
b. 読むだけでなく新たなトークンを (次に読むものとして) 押し込む `push(s)` を追加する。引数はトークン文字列。
c. その時点からファイルを別のファイルに切替える `open(s)` を追加する。引数はファイル名。
d. その他、何か「あったらいいかも」という機能を追加。

4 課題 1A

今回の演習問題から (小問を)1 つ以上選び、プログラムを作成しなさい。作成したプログラムについてレポートを作成し、久野 (`y-kuno@uec.ac.jp`) まで PDF を送付してください。LaTeX の使用を強く希望します。レポートは次の内容を含むこと。期限は次回授業前日一杯。レポートおよびその評点はクラス内で公開します。

- タイトル – 「システムソフトウェア特論 課題# 1」、学籍番号、氏名、提出日付。
- 課題の再掲 — レポートを読む人がどの課題をやったのか分かる程度にやった課題を要約して説明してください。
- 方針 — その課題をどのような方針でやろうと考えたか。
- 成果物 — プログラムとその説明および実行例。
- 考察 — 課題をやってみて分かったこと、気付いたことなど。
- 以下のアンケートの解答。

  Q1. プログラミングは好き/得意ですか、苦手ですか。それはなぜですか。どういうところが特にそう思う?
  Q2. さまざまなプログラミング言語があるということについてどのように考えていますか。
  Q3. リフレクション (課題をやってみて気付いたこと)、感想、要望など。

A APIドキュメント

Java 言語は豊富な標準ライブラリでも知られています。ライブラリのドキュメントも整備されており、以下のサイトで見ることができます。この文書は **API ドキュメント**と呼ばれています。[7]

`http://docs.oracle.com/javase/8/docs/api/`

ライブラリはクラスの集まりであり、パッケージ (`java.util.*`とか `java.lang.*`とか) の単位でグループ化されています。[8]

図 1 は API ドキュメントで `java.lang.String` のところを表示したものです。特定クラスを表示するのは、左下の全クラスの ABC 順の一覧から選ぶか、まず左上のパッケージ一覧でパッケージを選び、次に左下でそのパッケージのクラス一覧から選択します。`String` の例で分かるように、API ドキュメントではクラスごとにどのようなクラスかという説明と、それに続いてそのクラスが持つメソッドやその機能の説明が書かれています。

課題 1-3 次のクラスの API ドキュメントを表示し、どのような機能が含まれているかチェックせよ。続いて、以下のクラスのところを確認し、実際に例題プログラムを手直しし、その機能の動作を確認してみよ。

[7] API は applicaiton programming interface の略。
[8] `java.lang.*`は言語の基本機能を実現するクラス群なので、自動的に `import` されることになっている。

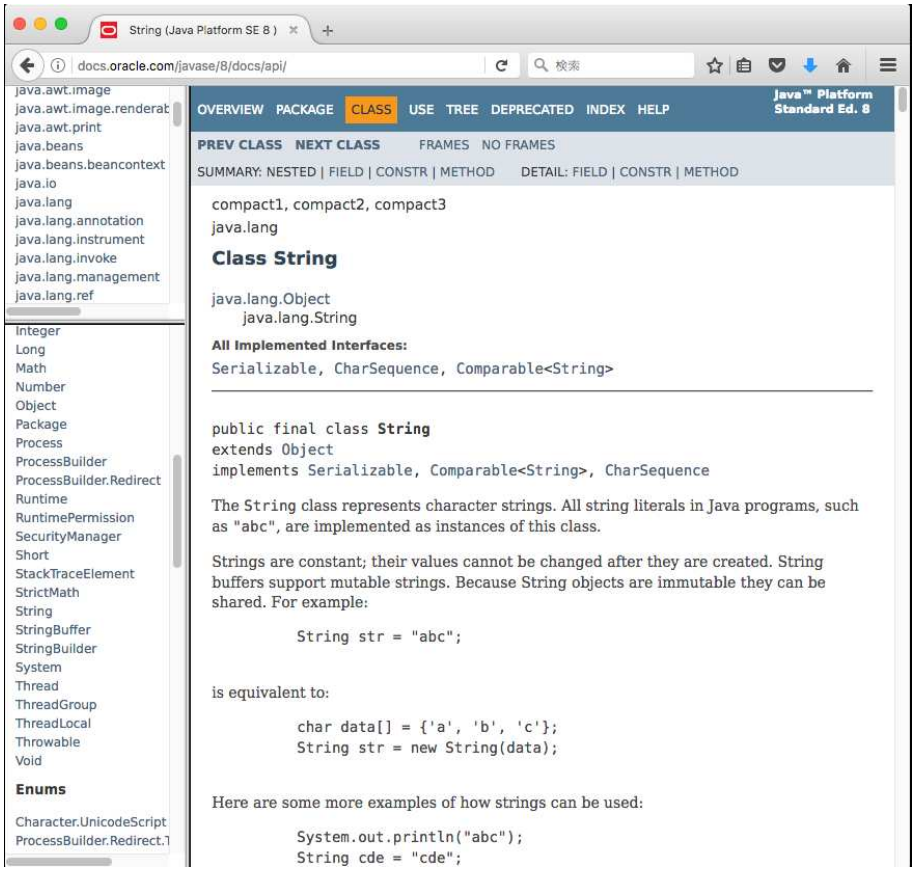

図 1: API ドキュメントの画面

a. `java.lang.String` — 文字列オブジェクト。文字列に対するさまざまな操作がおこなえる。
b. `java.util.Scanner` — 入力読み込みオブジェクト。さまざまな形での入力がおこなえる。
c. `java.lang.Integer`、`java.lang.Double` — 整数や実数の包囲オブジェクト。これらの値に対する加工がおこなえる。
d. `java.io.PrintStream` — `System.out` に格納されている出力オブジェクト。さまざまな出力機能を提供している。
e. その他自分で興味を持ったクラス。