（2008-05-20 版）

Java によるプログラミング入門 (9)

1　Lesson 7 入力と例外の処理（自習用）

この章では Java を用いて端末からの入力を処理する課題に取り組みます.

> **問題意識**　友達に電子メールを送りたくて,「アドレスを書いて」と尋ね，快くメモ用紙に書いてもらったとします．そのアドレスが妥当なものかどうか，どのようなチェックが可能でしょうか？

端末への**出力**については System.out.println() の利用を学びましたが，端末からの**入力**の処理がこの章で初めて取り扱われるのには以下のような理由があります[1]

- Java では入力の処理は複数の API を組み合わせて使わなければならず，オブジェクトの取り扱いが避けられません.
- 入力の処理はさまざまな例外的な事象を想定し，対応しなければなりません．そのためには Java の例外処理の機構を用いる必要があります.
- 入力データは妥当性をチェックし，適切に対応する必要があり，条件判定やループなども用いてプログラミングする必要があります.

[1] 他の言語に比べ Java は以下のようなことをしっかり考えないといけないような設計になっている，という点もあります.

なお，今日ではユーザとの対話処理は端末からの文字入力よりウィンドウを使ったGUI[2] が主流です．しかしながら，Java での端末からの入力処理は「ストリームの処理」としてファイル入力処理に共通しており，有用です[3].

[2] グラフィカルユーザインターフェイス

[3] さらにソケットを使ったネットワークプログラミングとも共通しています.

1.1 タスク 5

3 タンクの問題について，給水管と各タンクの持つパラメータを端末から入力できるようにプログラムしなさい．例えば入力は以下のようなフォーマットに従うものを考えます.

```
1.0
10.0，1.0，0.5
5.0，3.0，0.3
4.0，1.0，0.6
```

ここで 1 行目は給水管の流量，2 ～ 4 行はそれぞれ 1 番目 ～ 3　番目のタンクについて，底面積，初期水位，排水係数を与えるものとします．なお，これらの数値は ‘,’ (コンマ）で区切るものとします.

1.2 テキストファイルとストリーム

端末からの文字入力は Java ではデータが文字単位で順次流れ込んでくる**「ストリーム (stream)」**として扱われます．ファイルからの入力でも同じように文字で構成されたファイル（**テキストファイル**）の文字が先頭から順次流れ込んでくるストリームとして扱われます．このような文字ストリーム [4] の処理における基礎知識として

- 文字ストリームを構成するデータは “1” や “a” などの文字のほかに「空白」,「タブ記号」,「改行」[5] などの目に見えない記号から構成されます.
- 文字ストリームには終了 (**End of File, EOF**) が生じます.
- 「改行」記号で挟まれた文字列を**「行 (line)」**と呼びます.
- 行の中には数値などが書かれますが, 単語や数値など一かたまりの文字表現を**「トークン (token)」**と呼びます．トークンは「空白」や「カンマ」などを区切りとして切り出しますが，区切りに使われる記号を**「デリミタ (delimiter)」**と呼びます.
- “1200” や “1.23E10” など文字列として表記された数値はプログラムの中でこれを数値として解釈する必要があります．このような字句の解釈を**「パース (parse)」**と呼びます.
- 数値として解釈しようとした文字列が例えば “ABC012” など数値として不適切な場合はパースに失敗します．外部からの入力ではこのような失敗に配慮しなければなりません.

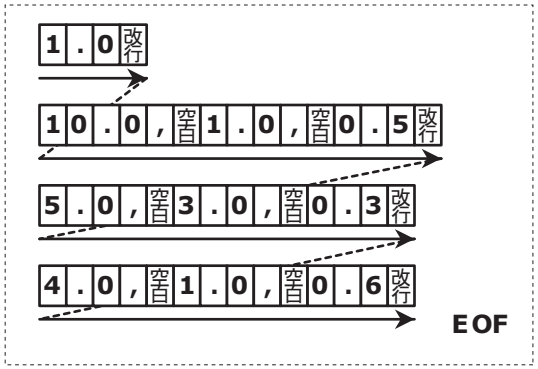

図 1　テキストファイル

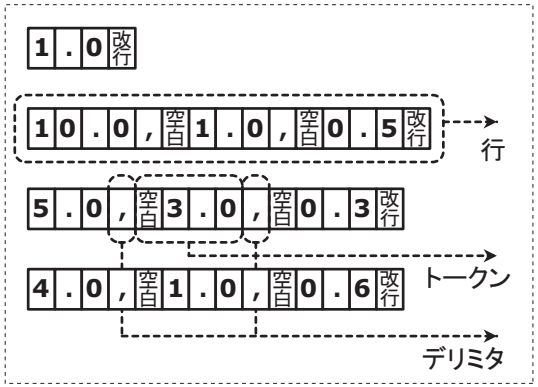

図 2　行，トークン，デリミタ

[4] ストリームには文字として処理する文字ストリームと 1 バイトを単位としてデータとして処理するデータストリームがあります．ここでは前者のみ取り扱います.

[5] 改行の記号はシステムによって異なりますが, Java の文字ストリーム処理ではこの差異を API で吸収してくれます.

1.3　データとその検査

外部からプログラムで必要とする数値が入力された場合には，パースにより数値としては解釈できても，さらにそれが適切な値であるかどうかを検査しなければなりません．例えばタンクの例では

- 給水量は 0 以上（非負）の数値でなければなりません．
- タンクの底面積は正の数値でなければなりません．（0 は許されません．0 の場合，タンクの水位の計算で 0 による割り算が生じます）．
- タンクの初期水位は非負．
- タンクの流出係数は非負．

などが検査すべき事項です．もちろん，極端に大きな数値や極端に 0 に近い数値も誤りである可能性がありますが，ここでは予め妥当な範囲を規定していないので検査しません．

> **外部から入力されるデータは信用せず徹底的に検査すべし．**インターネットに公開されるサーバではうっかりミスだけでなく**悪意のある入力**への対応も求められます．

1.4 例外とその処理

Java では例外を処理する特別の機構があります.

- あるメソッド呼び出しで通常の処理とは異なる扱い[6] を必要とする場合，そのメソッドは**「例外というクラスのオブジェクト[7]」**を呼び出し元に**「投げ(throw)」**，処理を中断します.
- 例外を投げる可能性のあるメソッドを呼び出す側はそのメソッド呼び出しを**「try ブロック」**で囲まなければなりません．また例外が生じた場合の処置を**「catch ブロック」**で捕捉しなければなりません．catch ブロックは例外のクラスに応じて複数書くことができます.
- 例外が生じた場合には処理を中断することが多いのですが，通常処理であるか，例外処理であるかにかかわらず必ず行いたい後処理は**「finally ブロック」**として書くことができます．catch ブロックの中で例えばメソッド呼び出しから復帰する return を行う場合でも finally ブロックは必ず実行されます.

[6] 例えばストリームから文字を読もうとして読み取りそのものに失敗するなど

[7] Exception クラスやそのサブクラスなどのオブジェクト

try ～ catch ～ finally の構文は以下のようになります：

```
try {
  例外を発生する可能性のあるメソッド呼び出し;
}
catch (Exception e) {
  例外発生時の処理；
}
finally { // 省略可
  通常処理，例外処理によらず try ブロック終了時に行う処理
}
```

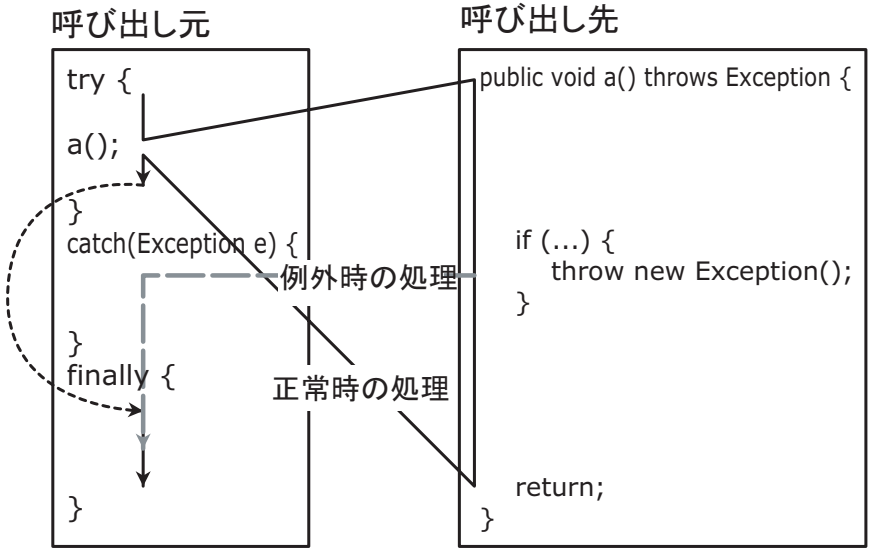

図 3　try/catch/finally 構文の動作

1.5 端末からの入力の処理

1.5.1 処理の概要

端末からの入力を処理するためには以下のような操作が必要です：

- 入力ストリーム System.in から文字ストリームから入力する InputStream-Reader のオブジェクト, バッファ付ストリーム入力のために BufferedReader のオブジェクトを順次生成します.
- 入力を促す文字列**「プロンプト (prompt)」**を System.out.println() などを使って端末に表示します.
- バッファ付ストリームリーダから１行読み取った文字列を得ます．その際，読み取れなかった場合の例外処理が要求されます.
- １行の文字列からトークンの切り出しのために StringTokenizer クラスのオブジェクトを生成します.
- StringTokenizer から順次トークンを得ます.
- トークンをパースして必要な型のデータを得ます．その際，パースに失敗した際の例外処理が要求されます.
- データの値の妥当性を検査します.
- 検査結果などを端末に表示し，誤りがある場合は適切な処理を促します.

1.5.2 ストリームの取り扱い

端末からの入力（標準入力）ストリームは System.in です．これから文字ストリーム入力 InputStreamReader のオブジェクトを生成し，さらにバッファ付ストリーム入力 BufferedReader のオブジェクトを生成するには以下のようにします[8].

これらのクラスを使うには `import java.io.*` を指定します.

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

標準入力に代えてファイルから入力する場合は InputStreamReader の代わりに FileReader を使います．読み出しを終えたら close() メソッドでファイルを閉じます．例えば parameters.csv というファイルからの入力は以下のようになります．ただしファイルが存在しない場合などに例外が生じるので try ブロックが必要です.

```
FileReader fr;
BufferedReader br;
try {
    fr = new FileReader("parameters.csv");
    // fr から読み出す処理をここに.
    fr.close();
} catch (IOException e) {
    System.out.println("IOException");
    System.exit(1);
}
br = new BufferedReader(fr);
```

[8] このように段階的になっているのは機能分担を行うためです．InputStreamReader はシステムに依存する文字コードの処理を, BufferedReader は入力をある長さで溜め込んで (buffering して）入力を効果的に行うとともに行単位の入力を行う機能を提供します.

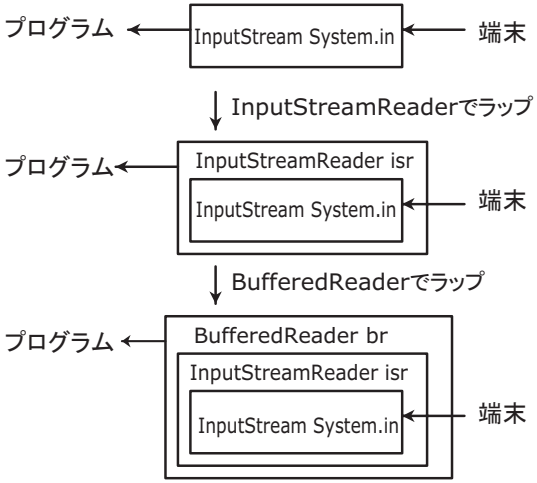

図 4　ストリームの扱い

同じように System.out の代わりにファイルに出力する場合は FileWriter と PrintWriter を使います．PrintWriter では書き出した内容をすぐにファイルには書き込まずにある量がたまるまで貯めておきます（バッファリング）．終了時にはバッファに残っている内容をファイルに書き出す flush() を実行してから close()

```
FileWriter fw;
PrintWriter pw;
try {
    fw = new FileWriter("parameters.csv");
    pw = new PrintWriter(pw);
    // pw を使う処理をここに．
    pw.flush();
    pw.close();
} catch (IOException e) {
    System.out.println("IOException");
    System.exit(1);
}
```

1.5.3 バッファ付ストリームリーダからの１行の読み出し

行単位でのデータの読み出しには BufferedReader クラスの readLine() メソッドを用います．このメソッドは例外 (IOException) を throw する [9] ので try ブロックが必要です．また，入力終了時には null が返るのでその処理も必要です．以下ではいずれの場合もメッセージを出力して System.exit() で実行を終了します [10]．

```
try {
    String line = br.readLine();
    if (line == null) {
        System.out.println("No input");
        System.exit(1);
    }
} catch (IOException e) {
    System.out.println("IOException: " + e);
    System.exit(1);
}
```

[9] API のメソッドがどのような例外を throw するかは API のオンラインドキュメントを見れば分かります．

[10] 通常，正常な終了では 0 をそうでない場合はそれ以外の値を引数で与えるのが慣習です．

1.5.4 トークンの切り出しと数値への変換

トークン切り出しは `import java.util.*`を指定し StringTokenizer を使います.

```
StringTokenizer st = new StringTokenizer(line,",");
if (st.hasMoreTokens()) {
    String token = st.nextToken();
} else {
    // トークンが得られなかったときの処理
}
```

トークンをパースして実数型の数値を得るには以下のようにします．Double.parseDouble() は NumberFormatException を throw するので try ブロックが必要です．整数型への変換には Integer.parseInt() を用います.

```
// String token にトークンがあり，double value に結果を入れる
try {
    value = Double.parseDouble(token);
} catch (NumberFormatException e) {
    System.out.println("Unable to parse (" + token + ") as double.");
    System.exit(1);
}
```

1.6 タスクの実装例

実装の方針

- タンクのパラメータを保持するクラスとして TankParameters を作る．このクラスにはフィールドとして
  - タンクの底面積
  - タンクの初期水位
  - 排水係数
  - これらの値がセットされたかどうか

  を持たせます．また，メソッドとして以下の３つを用意します．
  - タンクの各パラメータに値を代入するメソッド [11]．
  - タンクのパラメータ値の妥当性をチェックするメソッド．
  - タンクのパラメータを文字列表現にするメソッド [12]．
- タンクのパラメータを読み込むクラスとしては ParameterReader を作る．このクラスには
  - 標準入力から生成されたバッファ付ストリームをフィールドとして持たせ，
  - 給水管の流量を読み取る readFlowRate()，
  - １つのタンクのパラメータを読み取る readTankParameters()

  というメソッドを用意します．

[11] set— というメソッド名を慣習として使い，セッターと呼ばれます．

[12] toString() という名前のメソッドを用意すれば，System.out.println() の引数としてこのクラスのオブジェクトが与えられたときこのメソッドが呼び出されて出力の表現に使われます．

1.7 TankParameters.java

```
 1: public class TankParameters {
 2:   private double tankArea = 0;
 3:   private double initialLevel = 0;
 4:   private double drainageCoefficient = 0;
 5:   private boolean tankAreaDefined = false;
 6:   private boolean initialLevelDefined = false;
 7:   private boolean drainageCoefficientDefined = false;
 8:   private String message;
 9:
10:   public boolean isParametersValid() {
11:       message = "";
12:       if (tankAreaDefined == false) {
13:           message = message + "Tank Area is not set; ";
14:           return false;
15:       }
16:       if (initialLevelDefined == false) {
17:           message = message + "Initial Level is not set; ";
18:           return false;
19:       }
20:       if (drainageCoefficientDefined == false) {
21:           message = message + "Drainage Coefficient is not set; ";
22:           return false;
23:       }
24:       if (tankArea <=0) {
25:           message = message + "TankArea is non positive; ";
26:           return false;
27:       }
```

```
28:         if (initialLevel <0) {
29:             message = message + "InitialLevel is negative; ";
30:             return false;
31:         }
32:         if (drainageCoefficient <0) {
33:             message = message + "Drainage Coefficient is negative; ";
34:             return false;
35:         }
36:         return true;
37:     }
38:     public void setTankArea(double tankArea) {
39:         tankAreaDefined = true;
40:         this.tankArea = tankArea;
41:     }
42:     public void setInitialLevel(double initialLevel) {
43:         initialLevelDefined = true;
44:         this.initialLevel = initialLevel;
45:     }
46:     public void setDrainageCoefficient(double drainageCoefficient) {
47:         drainageCoefficientDefined = true;
48:         this.drainageCoefficient = drainageCoefficient;
49:     }
50:     public String putMessage() {
51:         return message;
52:     }
53:     public String toString() {
54:         return "Tank Area = " + tankArea + " Init. Level = " +
55:                 initialLevel + " Drainage Coef. = " + drainageCoefficient;
56:     }
57: }
```

1.8 ParameterReader.java

```
 1: import java.io.*;
 2: import java.util.*;
 3:
 4: public class ParameterReader {
 5:   InputStreamReader isr;
 6:   BufferedReader br;
 7:
 8:   public ParameterReader() {
 9:       isr = new InputStreamReader(System.in);
10:       br = new BufferedReader(isr);
11:   }
12:   private void putMessage(String message) {
13:       System.out.println(message);
14:   }
15:   public TankParameters readTankParameters(String tankName) {
16:       TankParameters tankParameters  = new TankParameters();
17:       while (true) {
18:           putMessage("Input params for "+tankName+": tank area, init. level, drainage coef.>");
19:           String line="";
20:           try {
21:               line = br.readLine();
22:           } catch (IOException e) {
23:               System.out.println("IOException: " + e);
24:               System.exit(0);
25:           }
```

```
26:             if (line == null) {
27:                 putMessage("No input");
28:                 System.exit(0);
29:             }
30:             StringTokenizer st = new StringTokenizer(line,",");
31:             if (st.hasMoreTokens()) {
32:                 String token = st.nextToken();
33:                 try {
34:                     double tankArea = Double.parseDouble(token);
35:                     tankParameters.setTankArea(tankArea);
36:                 } catch (NumberFormatException e) {
37:                     System.out.println("Unable to parse "+ token + " as double");
38:                 }
39:             }
40:             if (st.hasMoreTokens()) {
41:                 String token = st.nextToken();
42:                 try {
43:                     double initialLevel = Double.parseDouble(token);
44:                     tankParameters.setInitialLevel(initialLevel);
45:                 } catch (NumberFormatException e) {
46:                     System.out.println("Unable to parse "+ token + " as double");
47:                 }
48:             }
49:             if (st.hasMoreTokens()) {
50:                 String token = st.nextToken();
51:                 try {
52:                     double drainageCoefficient = Double.parseDouble(token);
53:                     tankParameters.setDrainageCoefficient(drainageCoefficient);
```

```
54:                 } catch (NumberFormatException e) {
55:                     System.out.println("Unable to parse "+ token + " as double");
56:                 }
57:             }
58:             if (tankParameters.isParametersValid()) {
59:                 return tankParameters;
60:             } else {
61:                 putMessage(tankParameters.putMessage());
62:             }
63:         }
64:     }
65:
66:     public double readFlowRate() {
67:         while(true) {
68:             putMessage("Input flow rate>");
69:             String line="";
70:             try {
71:                 line = br.readLine();
72:             } catch (IOException e) {
73:                 System.out.println("IOException: " + e);
74:                 System.exit(0);
75:             }
76:             if (line == null) {
77:                 putMessage("No input");
78:                 System.exit(0);
79:             }
80:             StringTokenizer st = new StringTokenizer(line,",");
```

```
81:             if (st.hasMoreTokens()) {
82:                 String token = st.nextToken();
83:                 try {
84:                     double flowRate = Double.parseDouble(token);
85:                     if (flowRate > 0) {
86:                         return flowRate;
87:                     } else {
88:                         putMessage("Flow Rate " + token + " should be positive number");
89:                     }
90:                 } catch (NumberFormatException e) {
91:                     System.out.println("Unable to parse "+ token + " as double");
92:                 }
93:             }
94:         }
95:     }
96:
97:     public static void main(String args[]) {
98:         final int NUMBER_OF_TANKS = 3;
99:         ParameterReader parameterReader = new ParameterReader();
100:        double flowRate = parameterReader.readFlowRate();
101:        System.out.println("Flow Rate = " + flowRate);
102:        for (int i=0;i<NUMBER_OF_TANKS; i++) {
103:            String tankName = "Tank"+i;
104:            TankParameters tankParameters = parameterReader.readTankParameters(tankName);
105:            System.out.println("  Paramr of" + tankName + " = " + tankParameters);
106:        }
107:    }
108:}
```