1　実験目的・課題

ファイル入出力、正規表現について学ぶ

- テストを行った結果表を読み込み集計する
- 個人の名前と教科ごとの平均点を表示するプログラムを作成する
- 成績表は Tab 区切りファイルで保存されているとする
- 正規表現を使い入力エラーに対応する
- 教科数を増やした場合に対応する
- 成績クラスを継承した新しいクラスを作成する
- 継承したクラスにおいてメソッドのオーバーライドをする

以上の 7 つのことを行う。入力の成績表は表 1,2,3 に示す。

Number	Name	Score1	Score2	Score3	Score4
1	Michael	75	86	89	31
2	Emma	74	63	70	48
3	Hannah	73	45	82	31
4	Emily	40	30	49	48
5	Daniel	46	59	47	70
6	Oliver	59	26	61	44
7	Mia	74	64	62	72
8	James	86	61	51	78
9	Charlotte	85	50	66	61
10	Luna	52	72	57	56
11	Mason	60	80	50	49

表 1　4 教科成績表エラーなし

Number	Name	Score1	Score2	Score3	Score4
1	Michael	75	86	89	31
2	Emma	74	63	70	S8
3	Hannah	73	45	82	31
4	Emily	40	30	40	
5	Daniel	46	59	47	70
6	Oliver	59	2.6	61	44
7	Mia	74	64	62	
8	James	86	61	51	78
9	Charlotte	85	50	66	611
10	Luna	52	72	T7	56
11	Mason	60	80	50	49

表 2　4 教科成績表エラーあり

Number	Name	Score1	Score2	Score3	Score4	Score5	
1	Michael	75	86	89	31	39	
2	Emma	74	63	70	S8	88	
3	Hannah	73	45	82	31	26	26
4	Emily	40	30	48	40		
5	Daniel	46	59	47	70	72	
6	Oliver	59	26	61	44	65	
7	Mia	74	64	62			
8	James	86	61	51	78	60	
9	Charlotte	85	50	66	611	79	
10	Luna	52	72	T7	56	85	
11	Mason	60	80	50	49	58	

表 3　5 教科成績表エラーあり

2　基礎知識

2.1　ファイル入力

2.1.1　Java の標準入出力

キーボードからなどの標準入力は、InputStream というバイト列を保持する型で受け取る。普通、このインスタンスを独自生成することはなく、Java の標準 API である System クラスのメンバ変数 in を使用する。このとき直接文字列で受け取るのではなくバイト列で受け取るのは文字エンコードによる制約を受けないためである。

このバイトストリームを文字ストリームへ変換するのが InputStreamReader である。バイトを読み込み指定された charset を使い文字にデコードすることが出来る。charset とは、16 ビット Unicode コード単位

のシーケンスとバイト・シーケンス間のマッピングをすることが出来るものである。InputStreamReader の read() メソッド (int read(void)) を呼び出すたびにバイト入力ストリームから 1 バイト読み込まれ文字へ変換される。以下のプログラムは、標準入力を 1 文字読み込み出力するものである。

ソースコード 1　1 文字読み込み

```
1  import java.io.*;
2
3  public class main{
4    public static void main(String args[]){
5      InputStream is = System.in;
6      InputStreamReader reader = new InputStreamReader(is);
7      try{
8        int n = reader.read();
9        System.out.println(n);
10     }catch(IOException err){
11       System.out.println(err);
12     }
13   }
14 }
```

このプログラムの入力に全角の’ あ’ や半角の”ab”を与えると、それぞれ”12354”や”97”が出力された。また、9 行目を System.out.println((char)n); のようにか書き換え同じ入力を与えたところ、’ あ’,’a’ が出力された。つまり、read() メソッドは入力を 1 文字読み取りそれの文字コードを返していることがわかる。

この文字型入力ストリームをバッファリングすることによってテキストを効率よく読み込むことをできるようにするものが BufferedReader クラスである。以下のプログラムは 1 行読み込み、それを出力することが出来る。

ソースコード 2　1 行読み込み

```
1  BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
2  try{
3    String line = reader.readLine();
4    System.out.println(line);
5  }catch(IOException err){
6    System.out.println(err);
7  }
```

出力では、OutputStream クラスの出力ストリームがバイトストリームを受け付け、特定の相手に送る。PrintStream クラスはそのプラットフォームのデフォルト文字セットを使用してバイトに変換し、出力ストリームへ出力するクラスである。System.out.println() の println() メソッドはこのクラスが提供している。バイトではなく文字を書き込むことが必要な状況では PrintWriter を使用する。例えば、以下のように使用する。

ソースコード 3　1 行出力

```
1  public static void main(String args[]){
2    OutputStream os = System.out;
```

```
3     PrintStream writer = new PrintStream(os);
4     writer.println("sample output");
5   }
```

2.1.2 ファイルからの入力

Java では、java.io.File クラスによってファイルやディレクトリを 1 つのオブジェクトとして扱うことができる。ファイル入力でも標準入力の時と同じように FileInputStream,FileReader によってファイルから入力バイトを取得することが出来る。oracle の java doc によると raw バイトのストリームを読み込むときに FileInputStream が、文字ストリームを読み込むときには FileReader が推奨されている。

標準入出力で BufferedReader(InputStreamReader(InputStream)) のようにしたようにファイル入力では BufferedReader(FileReader(File)) とすることで標準入力と同様に書くことが出来る。以下に例を示す。

ソースコード 4　ファイル入力

```
1   try{
2     File file = new File("sample.txt");
3     FileReader fr = new FileReader(file);
4     BufferedReader br = new BufferedReader(fr);
5     System.out.println(br.readLine());
6   }catch(FileNotFoundException err){
7     System.out.println(err);
8   }catch(IOException err){
9     System.out.println(err);
10  }
11 }
```

BufferedReader(InputStreamReader(FileInputStream(File))) としても同様である。

2.2 例外処理

ゼロ除算や配列外参照などを行うと Java では例外として処理を中断させる。例外処理とは、プログラム上でそのような事態が発生したときにどう対処するかを記述するものである。

Java の例外は、Throwable クラスを継承したオブジェクトをスローすることで発生する。例外としてありうるものは

- Exception
  - ex. IOException,ClassNotFound
- RuntimeException
  - ex. IndexOutOfIndex
- Error
-
  - ex. VirtualMachineError

の 3 つである。

実際にスローされる例外クラスは全てが Throwable クラスを直接継承しているのではなく、図 1 のように

RuntimeException は Exception を継承するようになっている。

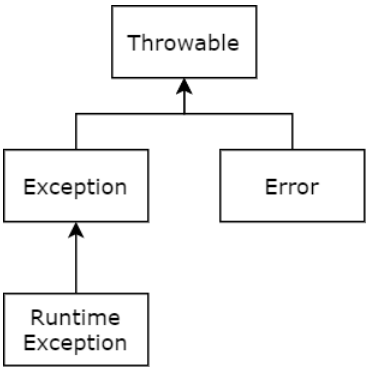

図 1　例外クラスの継承

Exception がスローされるコードはコンパイル時に例外処理の実装が強制され、RuntimeException は強制されない。Error は処理の継続が難しい致命的な場合であるため、例外処理を記述することはできない。

例外処理の構文は以下のようになる。

ソースコード 5　例外処理の構文

```
try{
  例外が起こりうる処理
}catch(例外の型　変数名){
  例外が起きた時の処理
}finary{
  最後に実行される処理
}
```

起こりうる例外が一つとは限らないので catch は複数書くこともできる。また、finary は書いても書かなくてもよい。また、スローされた例外をキャッチせずにそのままスローすることもでき、ソースコード 4 はソースコード 6 のように書ける。

ソースコード 6　例外処理を行わずにスローする

```
import java.io.*;

public class main{
  public static void main(String args[]) throws IOException{
    try{
      File file = new File("sample.txt");
      FileReader fr = new FileReader(file);
      BufferedReader br = new BufferedReader(fr);
      System.out.println(br.readLine());
    }catch(FileNotFoundException err){
      System.out.println(err);
    }
  }
```

```
14  }
```

2.3 正規表現

正規表現は文字列として指定し、Pattern クラスのインスタンスにコンパイルする必要がある。その結果のパターンを使用して任意の文字シーケンスを正規表現とマッチできる Matcher オブジェクトを生成できる。通常、以下のように使用する。

ソースコード 7　正規標準の呼び出し方

```
1  Pattern p = Pattern.compile("a*b");
2  Matcher m = p.matcher("aaaab");
3  boolean b = m.matches();
```

また次の文はソースコード 7 と等価である。

```
1  boolean b = Pattern.matches("a*b","aaaab");
```

ただし、マッチを繰り返す場合、コンパイル済みパターンを再利用したほうが効率が良い。

以下に正規表現で使うことのできる構文を示す。

文字 x	x
任意の文字	.
バックスラッシュ文字	\\
タブ文字	\t
改行文字	\n
a,b または c	[abc]
a,b,c 以外の文字	[^abc]
a-z の文字	[a-z]
a-z あるいは A-Z	[a-zA-Z]
b,c を除く a-z	[a-z&&[^bc]]
小文字の英字 (a-z)	\p{Lower}
大文字の英字 (A-Z)	\p{Upper}
英字 (a-zA-Z)	\p{Alpha}
空白またはタブ	\p{Blank}

表 4　正規表現の文字クラス

最長一致	
X、1 または 0 回	X?
X、0 回以上	X*
X、1 回以上	X+
X、n 回	X{n}
X、n 回以上	X{n,}
X、n 回以上、m 回以下	X{n,m}
最短一致	
X、1 または 0 回	X??
X、0 回以上	X*?
X、1 回以上	X+?
X、n 回	X{n}?
X、n 回以上	X{n,}?
X、n 回以上、m 回以下	X{n,m}?
論理演算子	
X の直後に Y	XY
X または Y	X |Y
X、前方参照を行う正規表現グループ	(X)

表 5　最長 (最短) 一致数量子、論理演算子

最長一致と最短一致とは、文字列”abbbc”に対して”ab+”のマッチを考えた時、最長一致では”abbb”がマッチするのに対し、最短一致 (”ab+?”) では”ab”がマッチする。

前方参照を行う正規表現グループは () によって番号を付けることができる。例えば ((A)(B(C))) は次の 4 つのグループに分けられる。

0. ((A)(B(C)))
1. (A)
2. (B(C))
3. (C)

グループ 0 は常に全体を指す。前方参照を行う正規表現グループが分類されたあと、入力シーケンスの各部分シーケンスがこれらのグループをマッチされ、マッチするたびに部分シーケンスが保存される。正規表現グループの部分シーケンスは、前方参照として表現内であとで使用することが出来る。また、マッチ終了後に各部分を取り出すことが出来る。

3　実装方法

3.1　入力文字列の解析

まず、2.1.2 ソースコード 4 に基づいて入力ファイルを読み込む。入力ファイルのパスはコンストラクタで与えるものとした。ただし入力の一番上の行はカラム名であるため、最初の 1 行は読み飛ばす処理を入れ

る。また、複数行の入力があるため入力がなくなるまで while 文で読み続ける。全ての入力を受け取った後に readLine() を行うと返り値は null であるため、null でない間処理を繰り返し続ける。このとき発生しうる例外は、文字エンコードがサポートされていない、ファイルが存在しない、入力に失敗する、の 3 つであるため、それぞれ catch しエラーメッセージを表示するようにする。

ソースコード 8　複数行の入力

```
1  File file = new File(this.path);
2  BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(file
       ), "UTF-8"));
3  br.readLine();
4  String line;
5  while ((line = br.readLine()) != null) {
6    seiseki tmp = this.parseSeiseki(line);
7    if(tmp!=null)this.data.add(tmp);
8  }
```

次に、読み込んだ 1 行が成績クラスの条件を満たしているか、入力が不正なデータではないか正規表現でチェックする。入力行は、Number,Name,Score1,2,3,4 の 6 つが Tab 区切りで与えられるはずである。5 教科の場合は Score5 が追加され 7 つが Tab 区切りで与えられる。これを正規表現で表すと以下のようになる。

ソースコード 9　入力行の構文解析に使用する正規表現

```
1  "([0-9]+) \\t ([A-Za-z]+) \\t ([0-9]+) \\t ([0-9]+) \\t ([0-9]+) \\t ([0-9]+)"
2  "([0-9]+) \\t ([A-Za-z]+) \\t ([0-9]+) \\t ([0-9]+) \\t ([0-9]+) \\t ([0-9]+) \\
       t ([0-9]+)"
```

この正規表現のパターンの切り替えは、コンストラクタで何教科あるかを与えられるようにした。このパターンにマッチしなければ入力行は正しくなく、null を返す。マッチした場合、正規表現グループの 1 に Number、2 に Name、3 から 6 あるいは 7 までに Score が格納されている。これをそれぞれ Matcher クラスの String group(int) メソッドで取り出し成績クラスのコンストラクタに渡し、それを返す。この操作を行っているのが parseSeiseki である。

このように入力にエラーがないとして取り出されたものを ArrayList に add していくことで最終的に全てのデータを読み込むことが出来る。

3.2　教科ごとの平均点の算出

教科数を N、正しい成績が入力された人数を M、j 人目の i 科目目を $A_{j,i}$ をとして、i 教科目の平均点は $n_i = \frac{1}{M}\sum_{j=0}^{M-1} A_{j,i}$ で計算することが出来る。

3.3　成績クラスの継承

成績クラスを継承した 5 教科用成績クラスをソースコード 10 に示す。

ソースコード 10　成績クラスを継承したクラス

```
1  public class seiseki5 extends seiseki{
```

```
2     public seiseki5(int num,String name,List<Integer> score){
3       super(num,name,score);
4     }
5     @Override
6     public double get_average(){
7       double average = 0;
8       for(int i=0;i<5;++i)average+=this.Scores.get(i);
9       return average/5.0;
10    }
11 }
```

コンストラクタは super() として継承の元となったスーパークラスのコンストラクタを呼び出すことができる。個人の平均点を求めるメソッドを 4 教科のものから 5 教科のものにオーバーライドしている。メソッドは「メソッド名 + 引数リスト」(この組み合わせをシグニチャという) で一意に特定し、オーバーロードはシグニチャが同じになり、メソッド名が同じでもシグニチャが異なるものはオーバーロードという。

4 結果と考察

4.1 4 教科エラーなし

4 教科エラーなし入力を読み込んだ時の出力結果を図 2 に示す。

1	Michael	75	86	89	31	70.3
2	Emma	74	63	70	48	63.8
3	Hannah	73	45	82	31	57.8
4	Emily	40	30	49	48	41.8
5	Daniel	46	59	47	70	55.5
6	Oliver	59	26	61	44	47.5
7	Mia	74	64	62	72	68.0
8	James	86	61	51	78	69.0
9	Charlotte	85	50	66	61	65.5
10	Luna	52	72	57	56	59.3
11	Mason	60	80	50	49	59.8
		65.8	57.8	62.2	53.5	

図 2　4 教科エラーなし入力の結果

表 1 と比べるとデータが正しく読み取れていることがわかる。また、Michael と Score1 について平均点を Python で計算したものを以下に示す。この結果と比べると平均点が正しく計算出来ていることがわかる。

```
1  >>> def average(list):return sum(list)/len(list)
2  ...
3  >>> Michael=[75,86,89,31]
4  >>> Score1=[75,74,73,40,46,59,74,86,85,52,60]
```

```
>>> average(Michael)
70.25
>>> average(Score1)
65.81818181818181
```

Michael の平均点を図 2 と比べると小数点以下 2 桁目を四捨五入して表示していると考えられる。これは、少数の出力に Java の String の format メソッドを使い、String.format(”%5.1f”,average); としているためである。%f は少数の出力を表し、5 は 5 文字になるように先頭を空白で埋め、.1 は小数点以下 1 桁目まで出力するということである。

4.2 4 教科エラーあり

図 3 に 4 教科エラーあり入力に対する出力を示す。

```
 1     Michael   75   86   89   31  70.3
 3      Hannah   73   45   82   31  57.8
 5      Daniel   46   59   47   70  55.5
 8       James   86   61   51   78  69.0
 9   Charlotte   85   50   66  611 203.0
11       Mason   60   80   50   49  59.8
                70.8 63.5 64.2145.0
```

図 3　4 教科エラーあり入力の結果

これも計算して確かめると正しい答えが出力されていることがわかる。Charlotte の Score4 が 611 点と他と比べて非常に大きい数になっているため Score3 と Score4 の平均点の間に空白が挟まらずに見にくくなってしまっている。これを解決するためには format で幅が 5 文字ではなく 6 文字になるようにすればよい。

このテストの最大得点がわからないため 611 点はエラーではないと判断したがもしテストの点数が 100 点より大きいものをエラーとするなら、”[0-9]+”というパターンで得点を表現するのではなく、”[0-9]{1,2}|100”とすることで、0 から 9 までの数が 1 桁または 2 桁、あるいは 100 のときのみに正規表現にマッチするため、期待する動作ができると考えられる。以上の 2 つの修正をした結果を図 4 に示す。

```
 1     Michael    75    86    89    31  70.3
 3      Hannah    73    45    82    31  57.8
 5      Daniel    46    59    47    70  55.5
 8       James    86    61    51    78  69.0
11       Mason    60    80    50    49  59.8
                 68.0  66.2  63.8  51.8
```

図 4　4 教科エラーあり入力の結果 2

4.3　5 教科エラーあり

5 教科エラーあり入力、成績クラスを継承したものを使用したプログラムの実行結果を図 5 に示す。このときプログラム中の seiseki という文字列を全て seiseki5 というふうに置き換えなければならない。しかし、これを眼で見て一つ一つ行うのはミスをする可能性が大きく、プログラムが大きければ時間がかかってしまう。ここで NetBeans の置換という機能を使えば 1 つのファイルの中の特定の文字列を一斉に別の文字列に置き換えることが出来るため、これを使用した。

```
 1     Michael    75    86    89    31    39  64.0
 5      Daniel    46    59    47    70    72  58.8
 6      Oliver    59    26    61    44    65  51.0
 8       James    86    61    51    78    60  67.2
11       Mason    60    80    50    49    58  59.4
                 65.2  62.4  59.6  54.4  58.8
```

図 5　5 教科エラーあり入力の結果

4 教科から 5 教科への切り替えのためにソースコード 9 のように 2 つのパターンを切り替えなければならない。入力が 4 教科か 5 教科しかないならばこれでもいいが 6,7,... と増えて言った場合パターンを書き直すあるいはコメントアウトなどするのは面倒である。ここで、2 つの正規表現に注目してみると、”\\t ([0-9]{1,2}|100)” が繰り返される回数しか違いがないことがわかる。よって、教科数を N として、ソースコード 11 のようにすることで N の数字を変えるだけで教科数の変更に対応できるようになる。

ソースコード 11　N 教科の正規表現

```
1  String regex = "([0-9]+) \\t ([a-zA-Z]+)";
2  for(int i=0;i<N;++i)
3    regex += "\\t ([0-9]{1,2}|100)";
4  Pattern ptn = Pattern.compile(regex);
```

参考文献

[1] Java SE API & ドキュメント
https://www.oracle.com/jp/java/technologies/javase/documentation/api-jsp.html

[2] IT 専科 Java 入門
http://www.itsenka.com/contents/development/java/