

プリント 33 ページの検討問題の解答例を示します。

これは、Record2 の入力を繰り返し、商品コードの最小値(辞書並び)を表示するプログラム Rec2InputMinimum.java です。次の実行結果例のように、入力の繰り返し終了は、商品コード入力で Record2.HIGH\_VALUE の入力を行った時点です。

```
Z:¥Java>java Rec2InputMinimum
1 件目入力です
  商品コード>>D05
  数量>>50
2 件目入力です
  商品コード>>B10
  数量>>90
3 件目入力です
  商品コード>>B05
  数量>>75
4 件目入力です
  商品コード>>C01
  数量>>45
5 件目入力です
  商品コード>>Z99
  数量>>0
入力で、最小(辞書並び)の商品コードのレコードは次の通りです。
  4 番目レコード
    商品コード:B05
    数量:75
```

この場合、商品コードである文字列の大小比較が必要になります。それには、文字列のメソッド compareTo を利用すればよいでしょう。

(Web ログオンで『<http://manabu.quu.cc/up/jv/eJ1040.htm#String>』を参考)

Rec2InputMinimum.java は、次のようなコードになるでしょう。(解答例)

```
01 public class Rec2InputMinimum
02 {
03     public static void main(String[] arg)
04     {
05         java.util.Scanner stdin = new java.util.Scanner(System.in);
06         Record2 recMin = new Record2(Record2.HIGH_VALUE, 0); //最小レコード憶用
07         Record2 rec;
08         int n = 0;          // 入力データ個数用
09         int iMin = 0;
10         do
11         {
12             n++;
13             System.out.println(n + "件目入力です");
14             rec = new Record2(stdin); //キー入力データでコンストラクタを実行
15             if (rec.sho.compareTo(recMin.sho) < 0)
16             {
17                 recMin = rec;
18                 iMin = n;
19             }
20         } while (rec.sho.equals(Record2.HIGH_VALUE) == false);
21         System.out.println("入力で、最小(辞書並び)の商品コードのレコードは次の通りです。");
22         recMin.display(iMin);
23     }
24 }
```

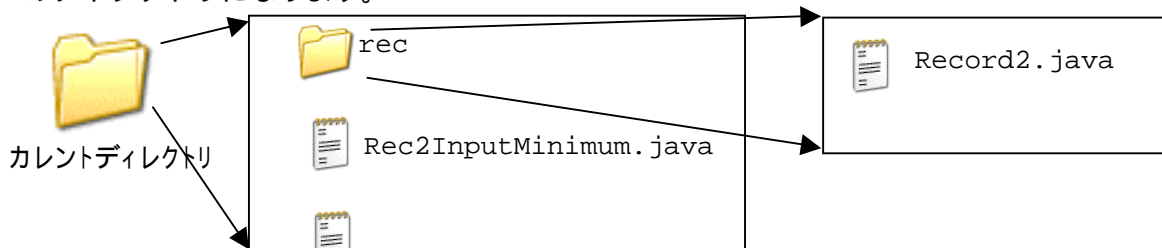
### パッケージ (package) の使用

パッケージとは、クラスが属する入れ物です。Scanner クラスは java.util のパッケージに入っています。FileInputStream クラスは java.io のパッケージに入っています。例えば、Scanner クラスの変数 stdin を宣言する場合は、java.util.Scanner stdin と宣言できます。または、ファイル先頭に import java.util.Scanner を書くことにより、パッケージを省略したクラス名だけで変数宣言ができるようになります。

クラスのソースファイル先頭に `package パッケージ名;` を書くことで、そのクラスは、`パッケージ名` のパッケージに属するようになります。

さて、ここで Record2 の rec のパッケージに入れてみます。

Rec2InputMinimum.java があるディレクトリに rec の名前のフォルダを作成します。その中に、Record2.java のファイルだけ移動します。パッケージは、実質的にその名前のディレクトリになります。



そして Record2.java の先頭行に `package rec;` と書きます。

そして、Rec2InputMinimum.java の先頭に、`import rec.Record2;` を書きます。

カレントディレクトリで、Rec2InputMinimum.java をコンパイルすると、

rec の中に、Record2.class ファイルが生成されて実行できるでしょう。

なお、過去の Record2 を利用するソースファイルも、カレントディレクトリに過去のパッケージ前の Record2.class が残っていれば、コンパイルや実行が可能です。それは rec 内の Record2 ではなく、カレントディレクトリの Record2.class で動いていることとなります。そして `import rec.Record2;` が指定される Rec2InputMinimum だけが、rec 内の Record2 で動作していることとなります。

### カプセル化 (それは、フィールドを隠すことです)

さてオブジェクト指向で、再利用するクラスを作る場合、『カプセル化すべき』と言われていきます。これは、『外部からは、公開されたメソッドを利用することでしか、内部フィールドのデータを操作できないようにすること』です。

それは、Record2 のフィールド (static フィールドは除く) の sho と suu の public を、private に変更することで実現できます。これを施すことにより、誤ってオブジェクトのデータを変更するようなコードが書き難くなります。当然に、前述のプログラム Rec2InputMinimum リスト 15 行と 20 行で、コンパイルエラーになります。エラーを確認ください。そして、Record2.java に次メソッドを追加して、これを利用するように Rec2InputMinimum を変更し、実行できることを確認ください。

```
// 商品コードの大小比較 自身より、引数の方が大きければ負
public int compareTo(Record2 rec){
    return sho.compareTo(rec.sho);
}
```

なお、インターネットで、**カプセル化**の利点を調べてみましょう。

カプセル化を進めることによりオブジェクト内部の仕様変更が外部に影響しなくなります。それによって、クラスの再利用が容易になります。

なお、カプセル化によって隠されたフィールドを操作するメソッドの追加が必要です。前述のように、Record2 クラスの sho と suu が private になると、別のクラスでは、内容を見ることも、変更することもできません。

よって、クラスの将来的な利用方法に応じて、これらフィールドの値取得や変更メソッドを用意します。

その場合の操作メソッドのネーミングは、情報の取得に `getxxx`、情報の設定に、`setxxx`(設定用引数) とするのが一般的です。(xxx は対象のフィールド名です) 数量の suu のフィールド用であれば、次のメソッドを追加すればよいでしょう。

以下を Record2.java に追加ください。

```
//数量の情報取得メソッド
public short getSuu(){
    return suu;
}
//数量の情報設定メソッド
public void setSuu(int suuData){
    suu = (short)suuData;
}
```

また、このような数値フィールドであれば、別の Record2 の数量情報を加算するメソッドを用意すると便利な場合があります。例えば次のようなものです。

```
//数量の情報を加算するメソッド
public void addSuu(Record2 rec) {
    suu += rec.suu;
}
```

しかし、可能性のある操作をどんどん追加していくと、ソースが大きくなり、逆にクラスの目的が読み取り難くなることもあります。ここでは必要最低限ということで、これは、追加しないことにします。

a の数量に b の数量を加算するに、次のコードで置き換え可以从です。

```
a.setSuu( a.getSuu() + b.getSuu() );
```

なお、sho の商品番号は中を見なくても比較ができ、しかも商品コードは一度決まってしまうと、再び変更することは稀なので、`getSho` や `setSho` を追加しないことにしてみます。

### プリント 36 ページ 検討問題の解答例

以下に、`BufferedWriter` を引数にする `write` メソッドの解答例を示します。

```
01 //レコードをテキストで書き出す
02 public boolean write(BufferedWriter bw) throws Exception{
03     if (sho.equals(HIGH_VALUE) == true){
04         return false; // 終端を意味するデータが出力しない。
05     }
06     bw.write(sho);
07     bw.newLine();
08     bw.write(""+suu);
09     bw.newLine();
10     return true;
11 }
```

(Record2 のメソッドです)

前述のメソッドを使った Rec2Test1CText.java の解答例を示します。

```

01 import rec.Record2;
02 import java.io.FileOutputStream;
03 import java.io.OutputStreamWriter; //文字出力ストリーム
04 import java.io.BufferedWriter; //行出力ストリーム
05 public class Rec2Test1CText{
06     public static void main(String[] arg) throws Exception {
07         Record2[] a = new Record2[]{
08             new Record2("A01", 4), //コンストラクタ
09             new Record2("A01", 5), //コンストラクタ
10             new Record2("A01", 1), //コンストラクタ
11             new Record2("A03", 5), //コンストラクタ
12             new Record2("B02", 3), //コンストラクタ
13             new Record2("B02", 9), //コンストラクタ
14             new Record2("B03", 10), //コンストラクタ
15         };
16         FileOutputStream fos = new FileOutputStream("transact0.txt"); //ファイル生成
17         OutputStreamWriter osw = new OutputStreamWriter(fos, "MS932"); //文字ストリームを生成
18         BufferedWriter bw = new BufferedWriter(osw); //文字ストリームから行ストリームを生成
19         for (int i = 0; i < a.length; i++){
20             a[i].display(i + 1); //表示
21             a[i].write(bw); //ファイル書き込み
22         }
23         bw.close();
24         osw.close();
25         fos.close();
26     }
27 }

```

同様に、BufferedReader を引数にした read メソッドと利用例の Rec2ReadText.java を示します。

```

01 //レコード読み込み 終端や、失敗時に、shoフィールドへHIGH_VALUEをセット
02 public boolean read(BufferedReader br) throws Exception{
03     if (sho != null && sho.equals(Record2.HIGH_VALUE) == true) {
04         return false;
05     }
06     sho = br.readLine();
07     String s_suu = br.readLine();
08     if (sho == null || s_suu == null) {
09         sho = HIGH_VALUE;
10         return false;
11     }
12     suu = Short.parseShort(s_suu);
13     return true;
14 }

```

(Rrcord2のメソッドです)

```

01 import rec.Record2;
02 import java.io.*;
03 public class Rec2ReadText {
04     public static void main(String[] arg) throws Exception{
05         FileInputStream fis = new FileInputStream(arg[0]); //mainの引数をファイル名として使う
06         InputStreamReader isr = new InputStreamReader(fis); //byteから文字ストリーム生成
07         BufferedReader br = new BufferedReader(isr); //文字から文字列ストリーム生成
08         Record2 rec = new Record2();
09         int n = 1;
10         while (rec.read(br) == true){
11             rec.display(n);
12             n++;
13         }
14         br.close();
15         isr.close();
16         fis.close();
17     }
18 }

```

Z:¥Java>java Rec2ReadText transact0.txt で実行させます