

プリント 23 において C 言語で作成した『transact0.bin』を、24 ページの C 言語のよ
うに読み込んで Java で動くものを作成していきます。

その前に、Java クラスの本当の作り方を検討します。

C 言語の構造体との本質的な思想の違いを知るために、次の正しくないクラスを示します。
作成して、実行し検討ください。

```
public class Record2Sti{//staticだけで、フィールドとメソッドを作る
    public static String sho;// 商品コード
    public static short suu; // 数量

    //n番目表示としてオペレータ用で画面表示する
    public static void display(int n){
        System.out.printf("%4d番目レコード%n", n);
        System.out.printf("%t商品コード%s%n", sho);
        System.out.printf("%t数量%d%n", suu);
    }
}
```

次に、このクラスを利用して (商品コード、数量) の 2 件分記憶しようとしている例を示
します。これは、正しく動作しないことを確認するプログラムです

```
public class RecStiTest0 {
    public static void main(String[] arg) throws Exception
    {
        Record2Sti.sho = "A01";
        Record2Sti.suu = 10;

        Record2Sti.sho = "A03"; // "A01"が上書きされる
        Record2Sti.suu = 5;     //10が上書きされる

        Record2Sti.display(1);//1件目表示(失敗)
        Record2Sti.display(2);//2件目表示

        Record2Sti a = new Record2Sti();
        Record2Sti b = new Record2Sti();
        // a.sho = "A01"; /* こう書くとエラーです。*/
        // b.sho = "A03"; /* こう書くとエラーです。*/
    }
}
```

static は、クラス名を付けますが、グローバル変数のように使えます

(Record2Sti.sho = "A01";) つまり、new で Record2St のオブジェクトを生成しても、
記憶域が新たに作られるものでなく、クラスに対して一つしか存在しないのです。よって
の"A03"の代入で、前の記憶内容"A01"が上書きされて無くなってしまいます。

以下に正しいレコードのクラスを示します。(Record2Sti の static を削除した内容です)

```
public class Record2{
    public static String sho;// 商品コード
    public static short suu; // 数量
    //n番目表示としてオペレータ用で画面表示する
    public static void display(int n){
        System.out.printf("%4d番目レコード%n", n);
        System.out.printf("%t商品コード:%s%n", sho);
        System.out.printf("%t数量:%d%n", suu);
    }
}
```

このクラスを2件分を記憶する正しい書き方は次のようになります。

```
public class RecTest0 {
    public static void main(String[] arg){
        Record2 a = new Record2();
        Record2 b = new Record2();

        a.sho = "A01";// 1 件目
        a.suu = 10;

        b.sho = "A03";// 2 件目
        b.suu = 8;

        a.display(1);// 1 件目表示 (オブジェクトaで、実行)
        b.display(2);// 2 件目表示 (オブジェクトbで、実行)
    }
}
```

これで、new で Record2 クラスのオブジェクトを2回生成していますが、static が付かないフィールドにしたので、それぞれ個別に記憶できる記憶域ができたことになります。

display メソッドも static をなくすことで、『クラス名.メソッド呼び出し』でなく、『オブジェクト.メソッド呼び出し』になっています。

ここで、a のオブジェクトに記憶される商品コードと数量がアクセスできます。

static メソッドの呼び出しでは、オブジェクトの指定がないので、static が付かないオブジェクトのフィールドがアクセスできません。(納得するまでソースを読むこと)

さて、C 言語で作成した『transact0.bin』の1レコードを Record2 に読む方法は、次のようになります。なお、オープン済みの FileInputStream クラスオブジェクトが、変数 fr_transact0 に管理されて、Record2 のオブジェクトが変数 rec に管理されている場合のコードです。

```
byte[] a = new byte[4];
fr_transact0.read(a);//4byte読み込み
//文字コードを文字にして連結
rec.sho = "" + (char)a[0] + (char)a[1] + (char)a[2];

a = new byte[2];
fr_transact0.read(a);//2byte読み込み
rec.suu = (short)(0x0fff & a[1]);
rec.suu <<= 8; // 1バイト分左シフト(256倍している)
rec.suu |= 0x0fff & a[0]; //ビットORで結合
```

なお、read メソッドは、戻り値が -1 でファイル読み取りが終了したと判断すればよいでしょう。以上と、C 言語の関数の作り方を参考にし、Record2.java を変更し、Rec2Read.java で、24 ページ C 言語の『rec2_read.c』に相当するプログラムを作成しましょう。

なお、C 言語の『record2_write』関数に相当するメソッドは『write』、『record2_read』関数に相当するメソッドは『read』で作成のこと。(なおファイルエラー処理は、なくてよい)

検討 終わった方は、別メソッドで、FileInputStream クラスの代わりに、

RandomAccessFile クラスで作れるか検討しよう。(Java API の Web 仕様書で確認して)

前ページの解答例を示します。

```

01 import java.io.*;
02
03 public class Record2
04 {
05     // 存在しないデータで、商品コードはこの文字列より小さいコードとする
06     public static String HIGH_VALUE = "Z99";
07
08     public String sho;        // 商品コード
09     public short suu;        // 数量
10
11     //n番目表示としてオペレータ用で画面表示する
12     public void display(int n){
13         System.out.printf("%4d番目レコード\n", n);
14         System.out.printf("%t商品コード:%s\n", sho);
15         System.out.printf("%t数量:%d\n", suu);
16     }
17
18     //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
19     public boolean read(FileInputStream fpr) throws Exception
20     {
21         if (sho != null && sho.equals(Record2.HIGH_VALUE) == true){
22             return false;
23         }
24
25         byte[] a = new byte[6];
26         int n = fpr.read(a); //6byte読み込み
27         if (n == -1){
28             sho = HIGH_VALUE;
29             return false;
30         }
31
32         //文字コードを文字にして連結
33         sho = "" + (char)a[0] + (char)a[1] + (char)a[2];
34
35         suu = (short)(0x0ff & a[5]);
36         suu <<= 8;        // 1バイト分左シフト(256倍している)
37         suu |= 0x0ff & a[4]; //ビットORで結合
38         return true;
39     }
40
41     //レコードをバイナリで書き出す
42     public boolean write(FileOutputStream fpw) throws Exception{
43         if (sho.equals(HIGH_VALUE) == true) {
44             return false; //終端を意味するデータは出力しない。
45         }
46         byte[] a = new byte[6];
47         a[0] = (byte)sho.charAt(0);
48         a[1] = (byte)sho.charAt(1);
49         a[2] = (byte)sho.charAt(2);
50         // a[3] = 0; 書かなくてもが設定されているはず
51         a[4] = (byte)(suu & 0x0ff);
52         a[5] = (byte)(suu >> 8);
53         fpw.write(a); //ファイルへ、まとめて書き込む
54         return true;
55     }
56 }

```

```

01 import java.io.FileInputStream;
02 import java.util.Scanner;
03
04 public class Rec2Read {
05
06     static Scanner stdin = new Scanner(System.in);
07
08     public static void main(String[] arg) throws Exception{
09         Record2 rec = new Record2();
10         String buf;
11         String file = "transact0.bin";
12
13         System.out.print("file name>>");
14         buf = stdin.nextLine();
15         if (buf.length() != 0) {
16             file = buf; // Enter以外の入力ならそれをファイル名とする
17         }
18         FileInputStream fr_transact0 = new FileInputStream(file);
19         //C言語とエラー処理が違うので書かない。
20         //エラー時は、mainの外に投げて(throws)、mainの呼び出しルーチンに任せている
21
22         int n = 1;
23         while (rec.read(fr_transact0) == true) {
24             rec.display(n);
25             n++;
26         }
27
28         fr_transact0.close();
29     }
30 }

```

```

01 import java.io.FileOutputStream;
02 public class Rec2Test1 {
03     public static void main(String[] arg) throws Exception{
04         Record2 rec = new Record2();
05         FileOutputStream fos = new FileOutputStream("transact0.bin");
06         rec.sho="A01";
07         rec.suu = 4;
08         rec.write(fos); //ファイルへ書き込む
09         rec.suu = 5;
10         rec.write(fos); //ファイルへ書き込む
11         rec.suu = 1;
12         rec.write(fos); //ファイルへ書き込む
13         rec.sho="A03";
14         rec.suu = 5;
15         rec.write(fos); //ファイルへ書き込む
16         rec.sho="B02";
17         rec.suu = 3;
18         rec.write(fos); //ファイルへ書き込む
19         rec.suu = 9;
20         rec.write(fos); //ファイルへ書き込む
21         rec.sho="B03";
22         rec.suu = 10;
23         rec.write(fos); //ファイルへ書き込む
24         fos.close();
25     }
26 }

```

商品コード	数量
A01	4
A01	5
A01	1
A03	5
B02	3
B02	9
B03	10

このバイナリファイル
『"transact0.bin"』
を作る

問題 Rec2Test1.java をもっとスマートにできないか検討します。

そのために、Record2 クラスに、次のメソッドを追加しなさい。

そして、Rec2Test1.java のファイルを Rec2Test1A.java の名前でコピーし、そこで、このメソッドを使うように変更しなさい。そして、どう変わったか？どちらがよいか検討しなさい。

```
//商品コードと数量を引数で設定するメソッド
public void init(String shoCode , int n){
    sho = shoCode;
    suu = (short) n;//キャストで、nを上位byteを切り捨てて使う
}
```

次を完成させなさい。 正しく作れたかは Rec2Read.java の実行で分かります。

```
import java.io.FileOutputStream;
public class Rec2Test1A {
    public static void main(String[] arg) throws Exception{
        Record2 rec = new Record2();// 【1】
        FileOutputStream fos = new FileOutputStream("transact0.bin");
        rec.init("A01", 4);
        rec.write(fos);
        rec.init("A01", 5);
        rec.write(fos);
        rec.init("A01", 1);
        rec.write(fos);
        rec.init("A03", 5);
        rec.write(fos);
        rec.init("B02", 3);
        rec.write(fos);
        rec.init("B02", 9);
        rec.write(fos);
        rec.init("B03", 10);
        rec.write(fos);
        fos.close();
    }
}
```

商品コード	数量
A01	4
A01	5
A01	1
A03	5
B02	3
B02	9
B03	10

このバイナリファイル
『"transact0.bin"』
を作る

さて、【1】の、 Record2() の表現は、**コンストラクタ**と呼ばれ、new で生成するときに行うものです。これは、クラスの中で定義することができます。

以下は引数のコンストラクタを定義した例です。(コンストラクタの定義は void などの型定義は無しで、戻り値も指定できません。そして、**クラス名と同じ名前** で作る必要があります)

Record2 クラスのメソッドが書ける位置に次を追加します。

```
public Record2(String sho, int n){ //引数ありコンストラクタ
    init( sho, n );
}
```

これにより、Record2 rec = new Record2();// 【1】の生成はできなくなりますが、Record2 rec = new Record2("A01", 4);// "A01"の商品コードで、数量が4のオブジェクトを生成 という使い方ができるようになります。 (【1】の生成で**どんなエラーか確認**のこと)

Rec2Test1 A.java のファイルを Rec2Test1B.java の名前でコピーし、そこで、このメソッドを使うように変更しなさい。そして、どう変わったか？どちらがよいか検討しなさい。正しく作れたかは Rec2Read.java の実行で分かります。

```
import java.io.FileOutputStream;
public class Rec2Test1B {
    public static void main(String[] arg) throws Exception{
        Record2 rec = new Record2("A01", 4); //コンストラクタによる生成
        // ファイル新規作成し、そのストリームを得る
        FileOutputStream fos = new FileOutputStream("transact0.bin");
        rec.write(fos);
        rec = new Record2("A01", 5);
        rec.write(fos);
        rec = new Record2("A01", 1); //コンストラクタによる生成
        rec.write(fos);
        rec = new Record2("A03", 5); //コンストラクタによる生成
        rec.write(fos);
        rec = new Record2("A02", 3); //コンストラクタによる生成
        rec.write(fos);
        //生成オブジェクトで、メソッド実行
        (new Record2("A02", 9)).write(fos);
        (new Record2("A03", 10)).write(fos);

        fos.close();
    }
}
```

商品コード	数量
A01	4
A01	5
A01	1
A03	5
B02	3
B02	9
B03	10

このバイナリファイル
『"transact0.bin"』
を作る

さて、引数ありコンストラクタだけを作った時点で、new Record2() の引数なしコンストラクタの使用が不可能になります。しかし引数なしコンストラクタを作れば実行できるようになります。例えば次のようなものです。

```
public Record2(){ //引数なしコンストラクタ
    // Record2生成時に設定しておきたいフィールドを設定する記述ですが
    // 指定が必要なければ、ここは書かなくてよい。
    // (フィールドが基本型であれば0に相当する値、参照型ならnullに自動設定される)
}
```

検討問題 1 コンストラクタを作った時点より、次のような配列の初期化も可能になります。

```
Record2[] a = new Record2[]{
    new Record2("A01", 4), //コンストラクタ
    new Record2("A01", 5), //コンストラクタ
};
```

プリント 2 3 ページ 『rec2_test1.c』のコードのように、配列の初期化を利用したプログラムを、Rec2Test1C.java で作成してみましょう。

検討問題 2 Rec2Read.java を Rec2ReadA.java にコピーして、キー入力した値以上の数量になっているレコードだけ表示するプログラムへ変更しましょう。

```
Z:¥Java>java Rec2ReadA
file name>>
入力値以上を数量のレコード表示させます。
入力値>>9
6 番目レコード
  商品コード:A02
  数量:9
7 番目レコード
  商品コード:A03
  数量:10
```