全商情処プロ部門 Java 解説　平成 25 年 2 月サンプル問題

１級【７】（映画館の問題）

解答　（１）ageNum[age] += num　（２）index = age
（３）n < m　（４）work = list.get(n)　（５）i < list.size()

問題を解く前に、クラスやメソッドを正確に把握する必要がある。具体的にはクラスやメソッドを四角で囲って視覚的に理解する、メソッドを呼び出している個所をマーカーで線を引く、等である。この問題は、映画１つ１つに Movie インスタンスを生成し、それを ArrayList で管理している。また、Movie インスタンスは配列 ageNum を持っている。わかりやすく言うと、配列 ageNum を持っている沢山の Movie インスタンスを、ArrayList という配列のようなものの中で管理している、ということである。ArrayList は「オブジェクトを配列として扱うことができる」と考えればよい。

（１）は、メソッド名(calc)が表している通り、計算するためのインスタンスメソッドである。Movie インスタンスが個々に持っている配列 ageNum に入場者数を足す必要がある。呼び出し元を見てみると、list.get(code).calc(…);となっている。get(code)は ArrayList クラスのメソッドで、code 番目の要素を返すものである。list.get(code)により、ArrayList の中にある code 番目の Movie インスタンスが呼び出されることになる。

（２）は、下の else やその下の CHARGE[index]から、index を求める必要があることが分かる。処理条件２に「添字は年齢区分と対応している」とあることから解答を導くことができる。

（３）を含む for の二重ループは隣接交換法による並べ替えである。外側の for では m を減算しており、また実際に交換している個所では n 番目と n+1 番目が使用されているので、これらから隣接交換法だと判断できる。その判断ができれば、（３）には n < m となることがわかる。

（４）では、ArrayList クラスの set(index, E)メソッドが使用されている。これは E の内容を index 番目に置き換える、というものである。（４）で n 番目を work に避難させ、次の行で n+1 番目を n 番目に入れて、次の行で避難させたものを n+1 番目に入れる、という 3 つのステップで入れ替えを実現している。(４)の下 2 行からの流れを考えると、n 番目を work に避難させるには、list.set(work, list.get(n))でもよさそうだが、これではいけない。参照型変数 work は、Movie work;で宣言されただけであり、参照は入っていない。参照が入っていない場所を使用することができないので、list.set(work, list.get(n))では何番目に避難させてよいのかわからず正確に動作しない。Work = list.get(n)とすることにより、リストの n 番目を指し示す参照を work に記憶させることにより、移動させたこととして考える。

（５）では映画の数だけ出力することとなる。映画の数というのは Movie インスタンスの数と同じ意

味である。Movie インスタンスは ArrayList で管理されているので、ArrayList の中にある参照の個数を調べればよい。ArrayList のメソッドとして size()メソッドがあるということを知っておく必要がある。

ArrayList<Movie> list = new ArrayList<Movie>();

ArrayList クラスはサイズ変更可能な配列を作成できる。<>で Movie を指定しているのは、ArrayList で使用する型を指定するジェネリクスという機能である。変わった書き方となるが「この ArrayList では Movie を配列として扱う」とあらかじめ宣言した、ということである。もちろん、<>の中には他のクラスを指定することが可能である。

while((line = br.readLine()) != null)

まず、line = br.readLine()を実行することにより、ファイルから 1 行読み込み、その内容を line に記憶している。そしてそれが null と一致しなければ繰り返す、ということとなる。null と一致しない、ということはデータがある、ということである。この while の中の書き方は、「データが存在する間は繰り返す」という処理の書き方として標準的な記述方法である。

なお、readLine()は終端を¥n や¥r で認識する。読み込んだ movie.csv は作品名がカンマで区切られているのではなく、作品名 1 つが 1 レコードとして改行で区切られていることになる。もし作品名がカンマで区切られているのであれば、readLine()は終端を認識できず全ての作品を 1 つのデータとして読み込んでしまうこととなる。

Movie movie = new Movie(line);　　list.add(movie);

ファイルから読み込んだ作品名を line に記憶し、それを引数として Movie クラスのインスタンスを生成している。そして生成したインスタンスの参照を add()メソッドによりリストに加えている。この 2 行をまとめると、list.add(new Movie(line));となる。

String[] str = line2.split(",");

split()メソッドは指定された文字で分割するという String クラスのメソッド。line2 で読み込んだ内容を","で分割し、それを配列 str に格納することとなる。

例)

```
line2 = "yamada,tanaka,katou";
String[] str = line2.split(",");
```

str[0]には「yamada」、str[1]には「tanaka」、str[2]には「katou」が格納される。

Int code = Integer.parseInt(str[0]);等

Integer クラスの parseInt()メソッドを直接呼び出している。これは parseInt()メソッドが static なメソッドなので、Integer クラスのインスタンスを生成しなくてもこうして呼び出せる。parseInt()メソッドは文字列を整数に変換する。str[0]の参照が指し示す文字列を整数に変換して、参照ではなく実際の値（基本データ型としての int 型）として code に

代入しているというものである。

try ～ catch(･･･)

try で囲われたブロックで例外が発生した場合は、catch の処理を行うというものである。問題を解く上では特に意識せず読み飛ばしても良い。しかし実際は、例外がどこで発生してどこで例外を catch するかを十分考える必要がある。

１級【７】（データベースの問題）

解答　（１）rank[m] = 1　（２）p = m + 1
（３）total[syasyu][week] + kin　（４）total[0][n]　（５）kubun == 1

クラスが 4 つあるのでそれぞれの役割を考える必要がある。最初の Customer クラスは抽象クラスであり、今回はスーパークラスとなる。General クラスと Special クラスは Customer クラスを継承したサブクラスである。この構造を踏まえた上で問題を解くと比較的理解し易い。

（１）は Customer クラスのコンストラクタ内部での処理である。for の中には順位を表す配列 rank の要素数だけ繰り返すようになっている。そのことから配列 rank に対する処理であることが推測できる。順位付けのアルゴリズムではまず最初に順位を「1」で初期化する、ということがわかっていれば簡単に解ける。

（２）は juni()メソッドの二重ループで何をしているのか把握するのが重要となる。その名の通り juni()メソッドでは順位付けをおこなっている。それは二重ループの中で配列 rank に 1 ずつ加算していることからもわかる。（２）の次の行に p が出ているが、初期化がされていないので（２）で行う必要がある。隣り合うもの同士を比較して順位を計算するので（２）は p = m + 1 となる。

（３）は二次元配列 total に集計するべきものが理解できれば簡単に解ける。（３）の 1 行上にある calc(jikan)というのは General クラスか Special クラスの calc(int jikan)メソッドが呼び出されることとなる。（４）の下 3 行にある Customer クラスの calc(int jikan)メソッドは abstract の付いた抽象メソッドなので呼び出されない。この抽象メソッドをサブクラスがオーバーライドしたものが呼び出される。正確には General クラスか Special クラスのインスタンスから呼び出される。

（４）は"%,7d"の"d"に対応した変数を指定する。"d"とは 10 進数の整数のことである。また直後に"円"と表示していることから金額であることがわかる。処理条件２の最後に「利用金額計と順位を」表示すると書いてあるので利用金額計を表す total[0][n]を表示すればよい。なお実行結果の（月）や(火)は実際に出力されるわけではない。実行結果をわかり易くするために示していると思われる。

（５）は General クラスのインスタンスの集計を行うのか、Special クラスのインスタンスの集計を行うのかを分岐している。つまり一般会員か特別会員かの判断をする命令を書けばよい。

public abstract class Customer

抽象クラスを表す。抽象クラスの特徴は、①インスタンスが生成できない、②抽象メソッドを指定することによりサブクラスでのオーバーライドを強制することができる、というものである。抽象クラスがあるということは、必ずどこかのクラスが extends で継承しているということがわかる。

protected

異なるパッケージのクラスからは利用できないが、異なるパッケージでもサブクラスからは利用できる、というアクセス制御。今回はパッケージがどうなっているのかわからない問題なので影響はない。

public abstract int calc(int jikan);

抽象メソッド。抽象クラスに抽象メソッドを書くことによって、必ずこのメソッドをサブクラスではオーバーライドしなくてはいけない。つまり、このような抽象メソッドが出ているということは、問題のプログラムでどこかで必ずオーバーライドしているということである。抽象メソッドやオーバーライドを使うということは、どこかでポリモーフィズムを実現していることが多いが、この問題ではポリモーフィズムは実現されていない。

Class.forName("ドライバ名");

紛らわしいが、これはClassという名前のクラスである。そのClassクラスのstaticメソッドであるforName()メソッドを呼び出してJDBCドライバの名前を指定する。JDBCとは、Javaとデータベースを接続するためのインタフェースであり、Javaでデータベースを扱う際必要となる。データベースの種類ごとに指定するドライバ名が異なる。

例）MySQLの場合

```
Class.forName("com.mysql.jdbc.Driver")
```

Connection con = …
Statement stmt = …
ResultSet rs = …

Connectionで特定のデータベースと接続し、ConnectionからSQL文を送るためのStatementインスタンスを生成し、StatementでSQL文を実行しResultSetで結果を受け取る、というものである。データベースに接続しSQL文を実行する際のお決まりの3つの命令なので覚えてしまうのが早い。この 3 行が問題として問われることは少ないと思われる。なお、ResultSetインスタンスを生成する際にSQL文とまとめると次のようになり、String sqlStrが不要となる。

```
ResltSet rs = stmt.executeQuery("SELECT * FROM … 利用番号 ASC");
```

rs.getInt("曜日コード");

rsにはSQL文を実行した表が丸ごと返される。その表からgetInt()メソッドで指定した列名を抜き出す処理となる。getInt()メソッドはint型を返す。他に、getDouble()メソッドやgetString()メソッドがある。

１級【７】（体育祭の問題）

解答　（１）11 - rank　　（２）lowIndex = 0
（３）contest[index].getCode() < code
（４）scoreCalc(nen, kumi, rank)　（５）p < contest.length

この問題も最初に abstract を使用した抽象クラス、抽象メソッドが存在する。ということは必ず抽象クラスを継承（extends）している個所があり、また必ず抽象メソッドをオーバーライドしている個所があることがわかる。今回はさらにポリモーフィズムが実現されている少し難しい問題である。main()メソッドは while の中に For が三重になっており四重のループ構造となっているので、その点も注意が必要である。解く際に自分で解き易く線を引いたり色分けをしたりすること。

（１）は 2 箇所あり（１）を含む scoreCalc()メソッドも 2 つあることから、このメソッドはオーバーライドされていることがわかる。また、（１）で計算した score を二次元配列 scoreTotal に足していることから score で得点を計算することがわかる。順位が 1 位なら得点は 10 点、2 位なら 9 点…となっており、順位＋得点は必ず「11」となる。11-順位で得点が計算でき、逆に 11-得点で順位が計算できる。

（２）の下の 2 行をみると low や high が出てきており、その足した結果を 2 で割っているので、変数名や処理から二分探索であることがわかる。LowIndex に値が設定されていないので、（２）で設定する必要がある。この個所は二分探索の問題でもよく出題される。特に HighIndex に値を設定するとき length を使用しているので記述方法やスペルを間違わないよう注意すること。

（３）も二分探索でよく出題される問題である。こういう場合は条件を満たした場合、lowIndex = index + 1 になる場合はどういう場合か、ということを考えるとよい。この場合、下の方を表す lowIndex が上の方に移動して探索範囲を狭めることとなる。つまり今探した個所より、さらに上の方を探すこととなる。よって、探したい code が大きい場合であり、contest[index].getCode() < code となる。contest[index].getCode()は、配列 contest に格納されている AttractionContest か TrackContest のインスタンスの参照を呼び出して、そのインスタンスの getCode()メソッドを呼び出している。

contest[]にはインスタンスの参照が記憶されている。（４）は contest[index].と書かれている時点で、配列 contest に記憶されているインスタンスのメソッドを呼び出していることがわかる。また、int a = contest[index].という風に戻り値を返していない形となっているので、戻り値のない void 型のメソッドを呼び出していることがわかる。つまり（４）には、printScoreTotal()メソッドか scoreCalc()メソッドしか答えとしてあり得ない。ここでは scoreCalc()メソッドを呼び出し、キーボードから入力された値を引数にして必要な計算を行っている。ここはポリモーフィズムを実現している個所となる。

（５）の for の中では、各競技の得点を出力し合計を計算している。各競技を全て出力するので、p は配列 contest の要素数の間は繰り返すこととなる。よって答えは p ＜ contest.length である。

public int[][] getScoreTotal(){ return scoreTotal; }

二次元配列 scoreTotal をそのまま戻り値として渡すメソッドである。このメソッドでは scoreTotal の参照を渡すので呼び出し元の main()メソッドに参照を教えることとなり、main()メソッドから書き換えが可能になってしまう。そうなるとアクセス制御の意味がなくなる。対策として、配列を戻すのではなく配列から取り出した値を戻す、という方法が挙げられる。

getScoreTotal()メソッドを以下のように変更して、int 型の値を戻すようにする。

```
public int getScoreTotal(int m, int n){
        return scoreTotal[m][n];
}
```

また、（５）の２行下の命令文は、配列をそのまま全部もらってその中から１つだけ取り出して処理している、というあまり見かけない処理方法となっているので、以下のような処理に変更することが望ましい。

```
gokei += contest[p].getScoreTotal(m, n);
```

contest[i] = new AttractionContest(…);

Contest 型の参照を格納する配列 contest には、当然 Contest 型の参照が記憶できるが、それだけでなく Contest 型を継承した AttractionContest 型と TrackContest 型も記憶できる。この命令では、Contest 型配列の contest に AttractionContest インスタンスを生成して記憶している。左右のクラスの方が違うことが大きなポイントとなる。左辺がスーパークラスであり、右辺がサブクラスとなっている。この型変換のことをアップキャストという。こうすることにより、AttractionContest 型と TrackContest 型という異なる型のインスタンスをまとめて１つの配列で管理することができるようになる。また、同じ名称のメソッドがあるので、contest[i].scoreCalc(…)と呼び出すことにより、自動的に AttractionContest 型と TrackContest 型の scoreCalc(…)を呼び出してくれる。これらポリモーフィズムを実現するためには、今回の様なメソッドのオーバーライドとアップキャストが必要になる。