全商情処プロ部門 Java 解説　平成 25 年 2 月サンプル問題

２級【７】問１（銭湯の問題）

解答　（１）nen < 12

処理条件の「6 歳以上 12 歳未満」という処理が問題になっていることが分かれば比較的簡単に解ける。プログラムが（１）を通過する時点で変数 nen の値は 6 以上となっているので、「nen >= 6」という条件は記述しない。

System.in

System クラスの in フィールドのこと。in フィールドは InputStream クラス。標準の入力ストリーム（通信路のこと）で、このストリームはすでに自動的に開いている。

Scanner nyuryoku = new Scanner(System.in);

java.util パッケージのクラス。java.util パッケージをインポートする必要があるので、import java.util.Scanner;という記述がある。System.in を引数として Scanner クラスのコンストラクタを呼び出しインスタンスを生成して、そのインスタンスの参照を nyuryoku に代入している、という命令になる。問題では参照型変数として nyuryoku が使用されているが、Scanner の場合は、sc という名称を使う場合が多い。

なお、Scanner の区切り文字はデフォルトでは空白文字となる。空白文字を含むデータの入力に注意する必要がある。

例）
```
Scanner sc = new Scanner(System.in);
String name = sc.next();
```

キーボードで「yamada tarou」と入力すると、name には「yamada」という文字列の参照が記憶される。もう 1 度 next()メソッドを実行すると、次は「tarou」が記憶される。

nextInt();

データを int としてスキャンする。他に、String としてスキャンする next()メソッドや double としてスキャンする nextDouble()メソッド等がある。

close();

スキャナを閉じる。

２級【７】問２（スポーツテストの問題）

解答　（２）kaisu >=0　[別解] kaisu != -1　　（３）min = kaisu

（２）は while の継続条件である。終了条件ではないので注意が必要である。この条件の間は繰り返す、というものである。処理条件から「回数に-1 が入力されたら」最高と最低を表示し終了することがわかるので、継続条件は「回数が-1 ではない」ということである。よって、処理条件通りに解答すると（２）の答えは「kaisu != -1」となる。実際の解答では「回数には 0 以上の正の数を入力するだろう」と判断しているのか、継続条件は「kaisu >= 0」となっている。
（３）は最小値を更新している。最大と最小を求めるアルゴリズムが解っていれば、比較的容易に解ける問題である。

private

アクセス制御を表す修飾子のひとつ。この修飾子が付くと、同じクラスからは利用できるが他のクラスからは利用できない変数となる。ただし、今回はクラスが 1 つしかないので private にしている意味はない。他に複数クラスがあり、他のクラスからはアクセスされたくない場合に効力を発揮する。

FukkinKaisu(int x, int y)

クラス名 FukkinKaisu と同じであり、これはコンストラクタである。アクセス制御が省略されており、これはデフォルトのアクセス制御と呼ばれる。デフォルトのアクセス制御だと、同じクラスや同じパッケージからは利用できるが、他のパッケージからは利用できない。ただし、private 同様、今回はクラスが 1 つしかなくパッケージの考え方もないため、デフォルトのアクセス制御でも public でも特に大差はない。public にする場合だと、以下のような記述となる。

```
public FukkinKaisu(int x, int y)
```

FukkinKaisu fk = new FukkinKaisu(0, 99);

0 と 99 を引数として渡して、コンストラクタを呼び出し、FukkinKaisu クラスのインスタンスを生成している。0 と 99 は、コンストラクタの x と y に相当し、それがコンストラクタ内部で、max と min に代入されることとなる。

２級【７】問３（部品の問題）

解答　（４）inCode == -1　　（５）p = p + 1　［別解］p++

（４）はｎに強制的に 10 を代入するための条件である。ｎはデータ件数を表しているので、10 が代入されるということは、（４）の１つ上の命令の while を抜けることとなる。つまり、ｎに 10 を代入することにより while から抜ける、というためのものである。while を抜けるための条件は、処理条件の 2 より、「部品データを 10 件入力」するか「部品コードに-1 が入力」されるかのいずれかである。よって、この（４）には部品コードに-1 が入力された、ということを記述することとなる。ちなみに、while を抜けさえすればよいので、（４）の１つ下の n = 10 は、n = 11 でも n = 100 でも同じ結果となる。
（５）は配列に記憶されているデータを順に出力するためのものである。配列の添字としてｐを使用しているので、ｐを 1 ずつ増加させる必要がある。インクリメント演算子を使用して、p++と記述しても正解である。

arrayIn()メソッドは、while の中に if があり、その中に if があり、その中に while がある、という構造となっており、それぞれのブロックを四角で囲う等の工夫をして自分自身の理解を助ける必要がある。

print()と println()

print()は文字を出力し、println()は文字を出力しその後改行する。

２級【７】問１（暗号の問題）

解答　（１）j = 0　　（２）kazu = j + angoKey

（１）の次の行で moji[j]を使用しており、j の初期化が必要なことが分かる。次の 3 行の while が線形探索だということがわかれば、配列の先頭から探索するということがわかるので、j には 0 を代入すればよいということがわかる。

（２）の次の行の if で kazu が使用されており、この（２）で kazu に値を代入する必要があることがわかる。kazu は出力（System.out.print）で使用されているので、暗号文の出力のための添字である。暗号文は、平文に暗号キーだけずらしたものとなるので、線形探索の結果わかった平文の添字 j に暗号キーの angoKey を足したものが kazu となる。なお、次の行の if(kazu >= 58)というのは、angoKey を足した結果配列 moji を超えてしまっては困るので、超えたら最初に戻って暗号を作るために 58 を引いている。

nextLine()

次の 1 行を読み込む。今回では、空白文字を含む文字列を入力したいのでこのメソッドを使用している。ただし、以下のようなプログラムの場合は注意が必要である。

例）
```
Scanner sc = new Scanner(System.in);
String name1 = sc.next();
String name2 = sc.nextLine();
```

最初にキーボードから「yamada」と入力しエンターキーを押した時点で、name1 には文字列「yamada」の参照が記憶される。しかし、「yamada」入力後にエンターキーを押しているので、次の 1 行を読み込む nextLine()では「yamada」入力後からエンターキーまでを読み込んでしまい、name2 には自動的に「""」が入力されてしまい、入力待ちが終了する。つまり、最初の入力は「yamada（空文字）[改行]」であり、sc.next()実行後にカーソルが「yamada」と「(空文字）[改行]」の間に移動し、sc.nextLine()実行時に「(空文字）[改行]」を読み込む。よって name2 には「""」が記憶される。

length()

String クラスのメソッド。文字列の長さを返す。

例）
```
String mojiretu = "hello";
```
mojiretu.length()だと、文字数の「5」が返される。

charAt(i)

String クラスのメソッド。指定された位置にある 1 文字を返す。引数の数値は 0 から文字列の長さ（length()）マイナス 1 の範囲となる。

例）
```
String mojiretu = "hello";
```
mojiretu には hello という文字が配列のように記憶されるイメージである。

mojiretu の先頭から 0 番目は「h」、1 番目は「e」…4 番目は「0」である。

２級【７】問２（探索の問題）

解答　（３）syohinCode[p] != tansakuCd
（４）p == 10　[別解] p > 9　　（５）suryo * syohinTanka[p]

（３）の次の行の while には p = p + 1 しかなく、この形は線形探索であることがわかる。線形探索では、探したいデータと違う間は配列上を進んでいき探索を繰り返すので、この継続条件は「探したいデータと違う」という「syohinCode[p] != tansakuCd」が答えとなる。
探索した商品コードが見つからない、ということは配列 SyohinCode を探したが見つからずに端まで行ってしまった、ということである。よって（４）には、添字が端まで行ってしまったことを表す「p == 10」が当てはまる。添字の最大値である 9 を超えた、と考えたら「p > 9」と記述しても正解である。
（５）は kingaku の計算であり、線形探索で見つかった添字 p を使用することが分かれば比較的簡単に解くことができる。

エスケープシーケンス

通常の文字では表せない特別な文字をエスケープシーケンスという。Java では「¥」を使用する。例えば、改行は「¥n」でありタブは「¥t」である。エスケープシーケンスを表す文字として「¥」を使用しているので、本来の単なる「¥」を使用するためには「¥¥」と表示する。

２級【７】（修学旅行の問題）

解答　（１）kiboKodo　　（２）kiboNinzuKei[n] + kiboNinzu
**　　（３）syoriNinzu　　（４）12　　（５）jyogen[n]**

（１）は while の形から線形探索であることがわかるかどうかがポイントである。それがわかれば簡単に解ける。線形探索を行う理由は、探したデータを元に別の対応するデータを出力したり、どこかに集計したりするからである。
（２）と（３）では、集計を行っている。それは処理条件１からも判断することができる。線形探索を行い、その後集計を行っている、ということがわかれば、この（２）と（３）も比較的簡単に解ける。
（４）は for の継続条件である。For の中の出力（System.out.printf）や実行結果から、配列の中身を全て出力していることが分かる。よって添字を 0～11 まで進める必要があるので、継続条件は n < 12 となる。
（５）は「if の条件を満たす場合にどのような処理が行われているか」を考えると解きやすい。条件を満たす場合",*”を出力しており、処理条件３から「人数上限より希望人数計が多い場合」ということがわかる。

private static int[] …

static が付くとクラス変数となり、インスタンスを生成しなくても利用することができる。オブジェクトごとに存在するのではなくクラスに１つだけ存在する。しかし今回の場合はインスタンスを生成しており、また Ryokou クラスを外部から使用していないので特に static の意味はない。

throws IOException

プログラムで例外が発生した場合、それを 1 つ前の呼び出し元に投げるという命令。IOException は入出力例外の発生を表す。今回はファイルを読んだり書いたりする問題なので、ファイル操作の時に「ファイルが存在しない」等の例外が発生する可能性がある。その際、throws で例外を回避して、例外処理を 1 つ前の呼び出し元に任せるという処理を行っている。Input()メソッドで例外が発生した場合、thorows によって呼び出し元の main()メソッドに返される。main()メソッドでは、さらに throws を行い例外を回避している。本来はどこかで例外を受け取る処理が必要だが、例外を受け取る try～catch が 1 級範囲のためか、ここでは throws で例外をかわす処理となっている。

in = new Scanner(new FileReader("tua.txt"));

FileReader クラスのインスタンスを作成し、それを引数として Scanner クラスのインスタンスを作成している。これによりファイルを読み込むことができる。Scanner in;とまとめて、Scanner in = new Scanner(new FileReader("tua.txt"));と記述することも可能。また、FileReader ではなく、ファイル指定のための File クラスを指定しても同じ処理が実現でき

る。さらに、Scanner ではなく BufferedReader を使用してもファイルの読み込みができる。この問題では Scanner を使用しているが、他にも様々な手法があることを知っておく必要がある。

while(in.hasNext())

hasNext()はスキャナの中に文字が残っていれば true を返す。Scanner クラスのインスタンスである in に文字が残っている間は繰り返す、という命令になる。なお、この問題では読み込むデータは全て int 型なので、hasNext()ではなく int 型が残っているかどうかを調べる hasNextInt()を使用してもよい。

ot = new BufferedWriter(new FileWriter(""))

FileWriter が直接文字を 1 文字書きこむものであり、BufferedWriter が文字をためてまとめて効率よく書き込むものである。ファイルに出力するには、このように FileWriter と bufferedWriter を組み合わせたものが一般的。1 文字ずつ書き込むのであれば FileWriter のみでも動作は可能だがプログラムがわかりにくくなる。BufferedWriter ot;と組み合わせて、BufferedWriter ot = new BufferedWriter(new FileWriter("ibento.csv"));と書くこともできる。

ot.newLine();

改行文字を書き込む。改行文字は"¥n"とは限らずシステムによって定義されたものとなる。OS に応じて適切な改行文字を書き込むことができる。

ot.flush();

バッファにたまっている書き込むべき内容を、明示的にファイルに書き込む。Flush()がなければ実際に HDD 等の物理デバイスにいつ書き込まれるかはわからないので、強制的に書き込ませるために使用。なお、BufferedWriter クラスの close()メソッドでは、必ずフラッシュしてから閉じるので、今回の場合は明示的に flush()をしなくても close()の時点で自動的にフラッシュされることとなる。